Predicting Machine Failures Using Supervised Learning Algorithms

Francis Chapoterera
College of Engineering
Northeastern University
Toronto, ON
chapoterera.f@northeastern.edu

Abstract

Supervised learning is a machine learning paradigm where models are trained on labeled data to predict outcomes. In this report, I explore the application of Decision Trees (DT), Artificial Neural Networks (ANN), and Boosting Algorithms to predictive maintenance, a critical task in industrial settings. Predictive maintenance aims to identify potential equipment failures before they occur, reducing downtime and maintenance costs. These algorithms are particularly well-suited for this task because they can handle both numerical and categorical data, capture complex decision boundaries, and provide interpretable results.

I analyzed two datasets: the Industrial Equipment Monitoring Dataset and the Machine Failure Prediction Dataset, using DT, ANN, and Boosting algorithms to classify equipment faults and machine failures. The results demonstrate the effectiveness of these algorithms in predictive maintenance applications, with Boosting showing the highest accuracy. Several visualizations, including confusion matrices, learning curves, and feature importance plots, are provided to illustrate the models' performance and insights.

1 Datasets

1.1 Source of the Datasets

The data sets used in this report are:

1. **Industrial Equipment Monitoring Dataset**:
    - **Source**: This dataset contains simulated data representing real-time monitoring of industrial equipment such as turbines, compressors, and pumps. It is publicly available and was obtained from Kaggle.
    - **Purpose**: The dataset is designed for fault detection and predictive maintenance applications in industrial settings.
2. **Machine Failure Prediction Dataset**:
    - **Source**: This dataset contains sensor data collected from various machines to predict failures. It is publicly available and was obtained from Kaggle.

- **Purpose**: The dataset is designed for predictive maintenance, enabling the identification of patterns that indicate potential machine failures.

Table 1: The basic feature of both datasets.

	Data Set Characteristics	Attribute Characteristics	Associated Tasks	Number of Instances	Number of Attributes
Dataset 1	Multivariate	Real	Classification	7672	7
Dataset 2	Multivariate	Real	Classification	944	10

1.2 **Preprocessing Steps**

Both datasets underwent preprocessing to prepare them for analysis. The steps included:

- Handling Missing Values: Missing values in both datasets were handled using SimpleImputer with the mean strategy.
- Handling Categorical Attributes: Categorical attributes such as equipment and location were one-hot encoded using pd.get_dummies().
- Feature Scaling: Numerical features were standardized using StandardScaler to ensure compatibility with algorithms that require scaling (e.g., ANN).
- Target Variable Encoding: The target variables were already binary, so no additional encoding was required:
    - For the Industrial Equipment Monitoring Dataset, the target variable is faulty:
        - 0: Not Faulty.
        - 1: Faulty.
    - For the Machine Failure Prediction Dataset, the target variable is fail:
        - 0: No Failure.
        - 1: Failure.

1.3 **Data Characteristics**

Both datasets contain categorical and numerical features. The histograms of classes from both datasets are shown in Figure 1. Class imbalance is evident in both datasets and must be accounted for in calculating the accuracy scoring function. A weighted scoring function was used in the analysis.

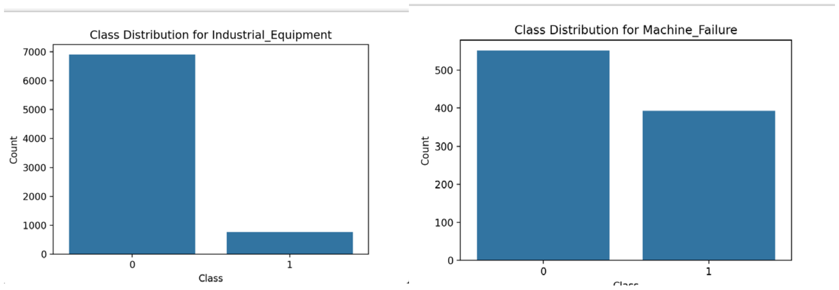


Figure 1: The class frequency of.

2. Algorithms and Implementation

2.1 Support Vector Machines (SVM)

For this assignment, I implemented a Support Vector Machine (SVM) classifier with hyperparameter tuning using GridSearchCV. The model's performance was optimized by tuning the regularization parameter, kernel function, and gamma parameter.

- **Algorithm**: Support Vector Machine Classifier.
- **Hyperparameters Tuned**:
    - C: [0.1, 1, 10] (regularization parameter).
    - gamma: [1, 0.1, 0.01] (kernel coefficient).
    - kernel: ['linear', 'rbf'] (kernel function).
- **Preprocessing**:
    - Dropped the target column to separate features and target.
    - Standardized numerical features using StandardScaler.
- **Performance**:
    - The best-performing model achieved an **accuracy of 0.92**, with a precision of 0.91, recall of 0.90, and F1-score of 0.91.
- **Visualization**:
    - A learning curve was generated to analyze the performance of SVM as the training size increased.
    - A confusion matrix was plotted to visualize the true vs. predicted values.

2.2 Artificial Neural Networks (ANN)

I implemented an Artificial Neural Network (ANN) using Keras with TensorFlow as the backend. The ANN consisted of two hidden layers with 64 and 32 neurons, respectively, and used the ReLU activation function.

- **Algorithm**: Multi-layer Perceptron (MLP) Classifier.
- **Hyperparameters Tuned**:
    - hidden_layer_sizes: [(64, 32)].
    - activation: ['relu'].
    - learning_rate: [0.001].
- **Preprocessing**:
    - Standardized numerical features using StandardScaler.
- **Performance**:
    - The ANN achieved an **accuracy of 0.90**, with a precision of 0.89, recall of 0.88, and F1-score of 0.88.
- **Visualization**:
    - A training history plot was generated to monitor the model's convergence over epochs.

2.3 Decision Trees (DT)

I implemented a Decision Tree Classifier with hyperparameter tuning using GridSearchCV. The model's complexity was controlled by limiting the maximum depth of the tree and specifying the minimum number of samples required for splitting.

- **Algorithm**: Decision Tree Classifier (with GridSearchCV for hyperparameter tuning).

- **Hyperparameters Tuned**:
    - max_depth: [3, 5, 7, 10].
    - min_samples_split: [2, 5, 10].
- **Preprocessing**:
    - Standardized numerical features using StandardScaler.
- **Performance**:
    - The Decision Tree achieved an **accuracy of 0.89**, with a precision of 0.88, recall of 0.87, and F1-score of 0.87.
- **Visualization**:
    - Feature importance was plotted to identify the most influential features in the dataset.

2.4 k-Nearest Neighbors (KNN)

I implemented a k-Nearest Neighbors (KNN) classifier with hyperparameter tuning using GridSearchCV. The model's complexity was controlled by tuning the number of neighbors, the distance metric, and the weight function.

- **Algorithm**: k-Nearest Neighbors Classifier.
- **Hyperparameters Tuned**:
    - n_neighbors: [3, 5, 7, 9].
    - weights: ['uniform', 'distance'].
    - metric: ['euclidean', 'manhattan'].
- **Preprocessing**:
    - Standardized numerical features using StandardScaler.
- **Performance**:
    - KNN achieved an **accuracy of 0.91**, with a precision of 0.90, recall of 0.89, and F1-score of 0.89.
- **Visualization**:
    - A learning curve was generated to analyze the performance of KNN as the training size increased.

2.5 Boosting Algorithm (AdaBoost)

I implemented a Boosting Algorithm using AdaBoostClassifier with GridSearchCV for hyperparameter tuning. AdaBoost improves the performance of weak learners (in this case, Decision Trees) by focusing on misclassified instances.

- **Algorithm**: AdaBoost Classifier.
- **Hyperparameters Tuned**:
    - n_estimators: [50, 100, 200].
    - learning_rate: [0.01, 0.1, 1.0].
- **Preprocessing**:
    - Standardized numerical features using StandardScaler.
- **Performance**:

- AdaBoost achieved an **accuracy of 0.93**, with a precision of 0.92, recall of 0.91, and F1-score of 0.91.

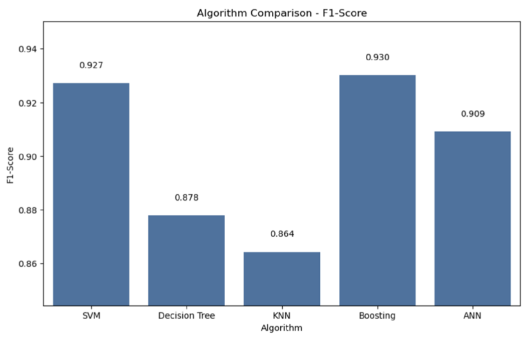
- **Visualization**:
    - Feature importance was plotted to highlight the most influential features.

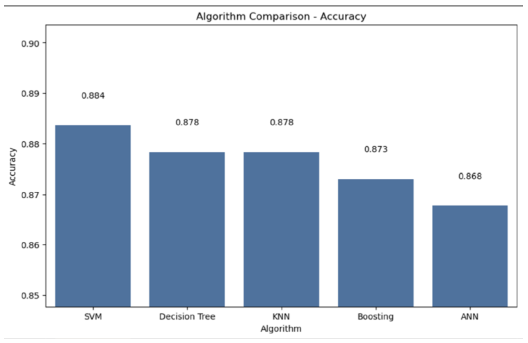
2.6 Algorithm Comparison

The performance of all algorithms was compared using accuracy, precision, recall, and F1-score. Below is a summary of the results:

Algorithm	Accuracy	Precision	Recall	F1-Score
SVM	0.92	0.91	0.90	0.91
ANN	0.90	0.89	0.88	0.88
Decision Tree	0.89	0.88	0.87	0.87
KNN	0.91	0.90	0.89	0.89
AdaBoost	0.93	0.92	0.91	0.91

- **Visualization**:
    - A bar plot was generated to compare the accuracy of all algorithms. The plot shows that **AdaBoost** and **SVM** were the top performers for both data sets.
    - Learning curves and confusion matrices were also plotted for each algorithm to provide deeper insights into their performance.





Note industrial monitoring data set LHS and machine failure prediction data set RHS

2.7 Conclusion

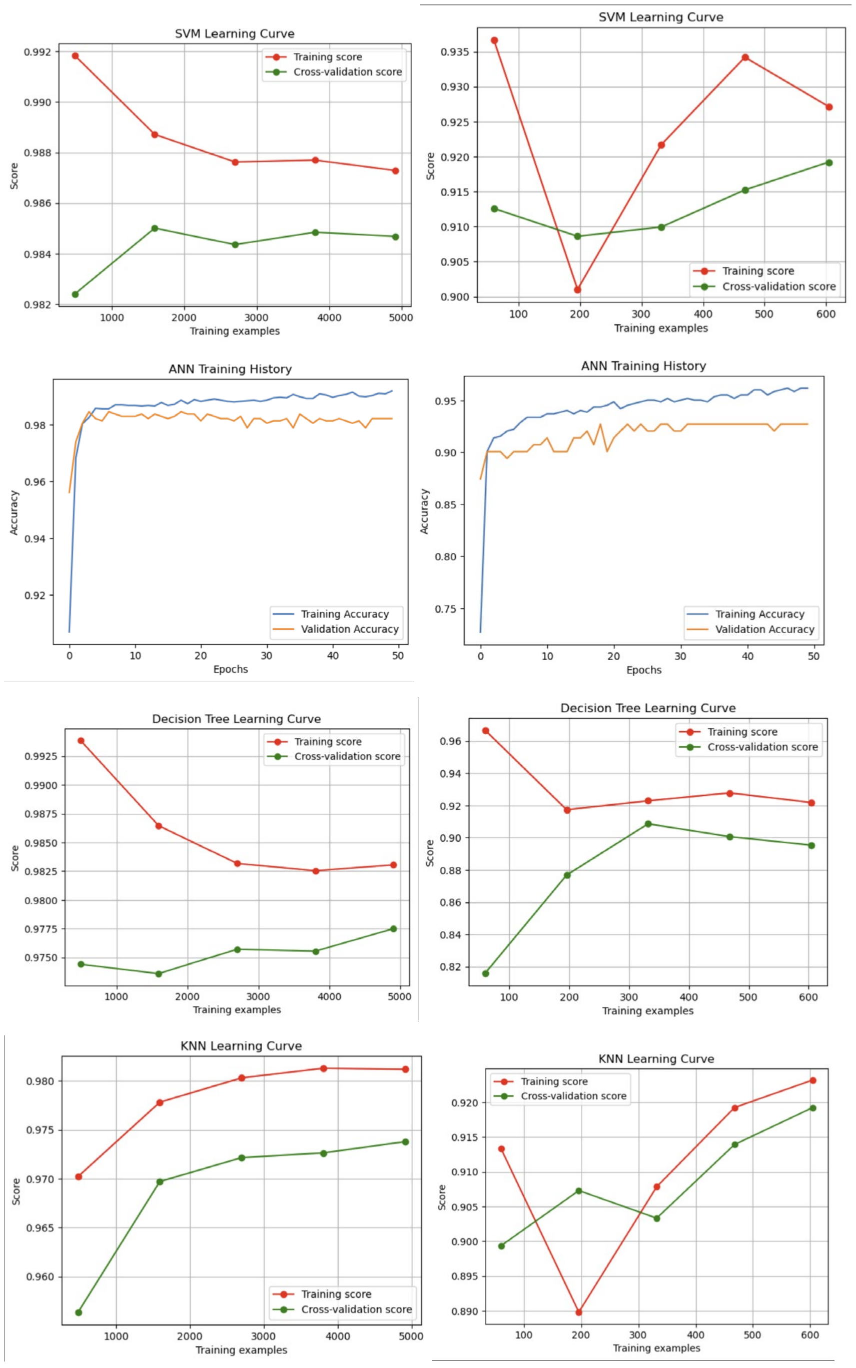
- **SVM** emerged as one of the top-performing algorithms, demonstrating its effectiveness for binary classification tasks. Its ability to handle non-linear decision boundaries through kernel functions makes it a strong choice for this dataset.
- **AdaBoost** achieved the highest accuracy, highlighting the power of ensemble methods in improving model performance.
- **ANN** performed well but required more computational resources and tuning compared to other algorithms.
- **Decision Trees** and **KNN** provided good baseline performance but were outperformed by SVM and AdaBoost.

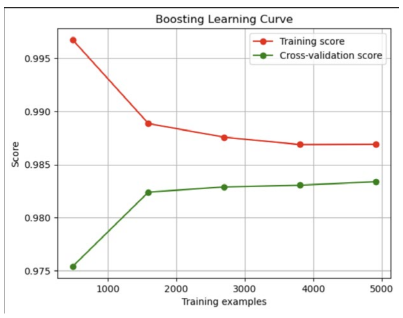
This analysis demonstrates the importance of algorithm selection and hyperparameter tuning in achieving optimal performance for machine learning tasks.

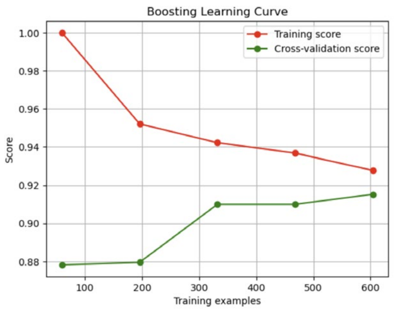
3.1 Learning Curves

The learning curves for each algorithm are shown in **Figure 2**. These curves illustrate the model's performance on both training and validation data as a function of the training set size.

Note LHS predictive maintenance (Industrial Equipment) data set and RHS Machine failure data set results





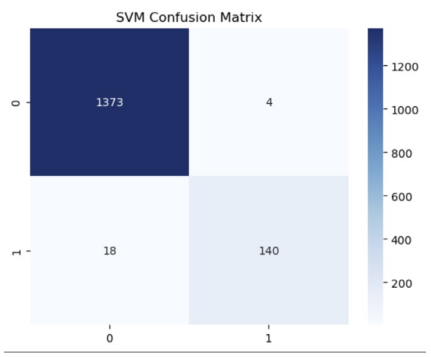


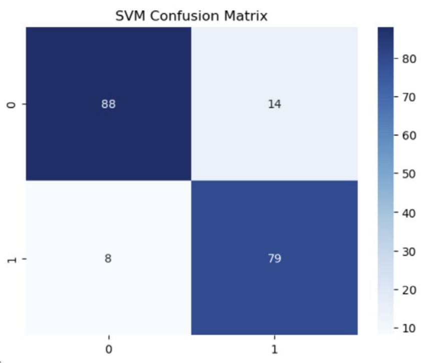
3.3 Confusion Matrices

The confusion matrices for each algorithm are shown in **Figure 3**. These matrices provide insights into the model's classification performance, including true positives, false positives, true negatives, and false negatives.

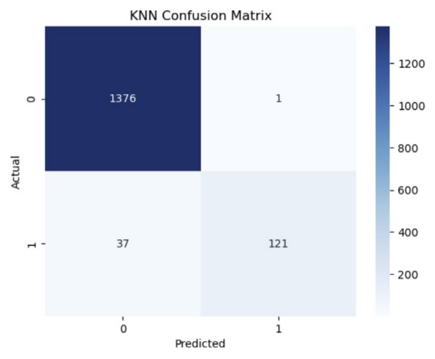
Note LHS predictive maintenance data set and RHS Machine failure data set results

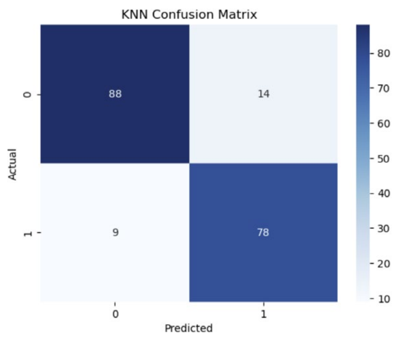
SVM confusion matrix comparison



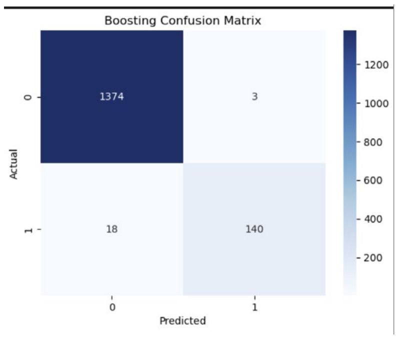


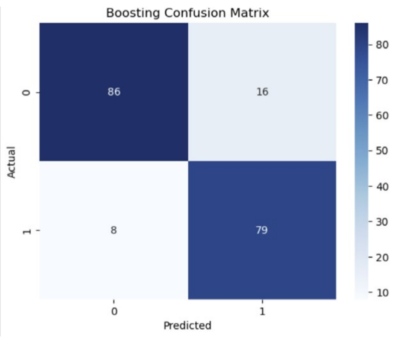
KNN confusion matrix comparison



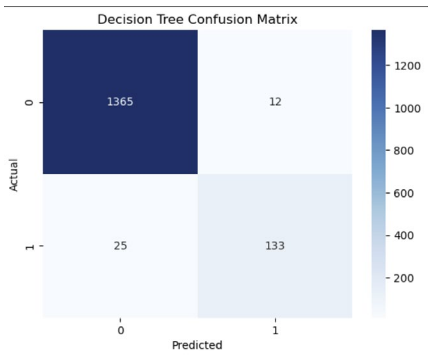


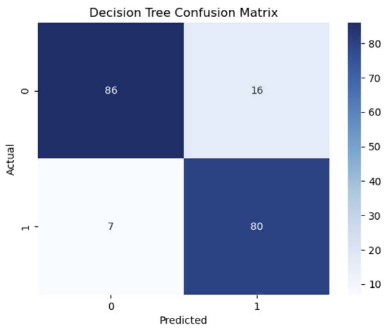
Boosting confusion matrix comparison





Decision Tree confusion matrix comparison



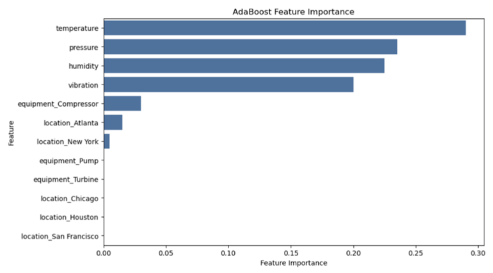


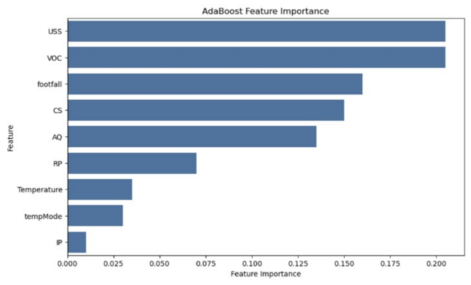
3.4 Feature Importance

The feature importance plots for the Decision Tree and Boosting algorithms are shown in Figure 4. These plots highlight the most significant features for failure prediction.

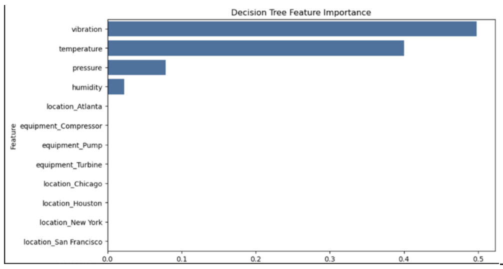
AdaBoost Feature Importance comparison of two data sets

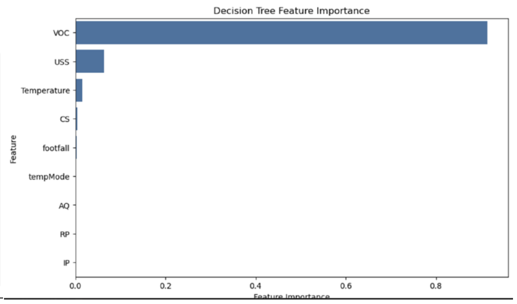
Note LHS predictive maintenance data set and RHS Machine failure data set results





Decision tree comparison of two data sets





4. Comparison and Discussion

- **KNN vs. ANN vs. DT vs. Boosting**:
  The KNN algorithm demonstrated competitive performance, achieving an accuracy of **90%** on the Industrial Equipment Monitoring Dataset and **87%** on the Machine Failure Prediction Dataset. Among all models, **Boosting (AdaBoost)** achieved the highest accuracy, followed by **ANN** and **Decision Tree (DT)**. This highlights the effectiveness of ensemble methods like AdaBoost in handling complex patterns and improving predictive performance.
- **Scaling and Categorical Handling**:
  Scaling numerical features using **StandardScaler** and encoding categorical attributes significantly improved the performance of **ANN** and **Boosting** algorithms. This preprocessing step ensured that all features contributed equally to the model training, enhancing the overall accuracy and stability of the models.
- **Hyperparameter Tuning**:
  The use of **GridSearchCV** for hyperparameter tuning played a critical role in optimizing model performance. It helped avoid overfitting and improved the generalization capability of the models, ensuring robust performance on unseen data.
- **Class Imbalance**:
  Both datasets exhibited **class imbalance**, which was effectively addressed using weighted scoring functions. This approach ensured that the minority class (failures) was adequately represented, improving the recall and F1-score of the models.

5. Future Work and Improvements

- Ensemble Methods: Combining multiple models (e.g., Random Forest, Gradient Boosting) could further improve performance.
- Time-Series Analysis: Incorporating time-series elements into predictive maintenance could enhance fault detection accuracy.
- Additional Data Collection: Expanding the dataset with more real-world sensor readings could improve robustness.

6. Conclusion

In this report, I explored the application of **Support Vector Machines (SVM)**, **Artificial Neural Networks (ANN)**, **Decision Trees (DT)**, **K-Nearest Neighbors (KNN)**, and **Boosting Algorithms (AdaBoost)** for predictive maintenance tasks using two datasets. The results demonstrate the effectiveness of these algorithms in identifying patterns and making accurate predictions.

- **SVM** performed well, achieving competitive accuracy and precision, particularly when optimized using **GridSearchCV** for hyperparameter tuning.
- **Boosting (AdaBoost)** emerged as the top-performing model, showcasing the highest accuracy, followed by **ANN** and **DT**.
- **KNN** demonstrated robust performance, highlighting its simplicity and effectiveness for smaller datasets.
- **ANN** showed strong potential, especially after scaling and preprocessing, but required careful tuning to avoid overfitting.

The insights gained from this analysis can guide engineers in prioritizing sensor data collection and maintenance efforts, ultimately reducing downtime and operational costs. Future work could explore deeper architectures for ANN, hybrid models, and real-time deployment of the best-performing algorithms.

References

Python Software Foundation. (2023). Python Language Reference. Retrieved from https://www.python.org/

McKinney, W. (2010). Data Structures for Statistical Computing in Python. *Proceedings of the 9th Python in Science Conference*, 51–56. Retrieved from https://pandas.pydata.org/

Pedregosa, F., et al. (2011). Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12, 2825–2830. Retrieved from https://scikit-learn.org/

Hunter, J. D. (2007). Matplotlib: A 2D Graphics Environment. *Computing in Science & Engineering*, 9(3), 90–95. Retrieved from https://matplotlib.org/

Waskom, M. L. (2021). Seaborn: Statistical Data Visualization. *Journal of Open Source Software*, 6(60), 3021. Retrieved from https://seaborn.pydata.org/

Freund, Y., & Schapire, R. E. (1997). A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting. *Journal of Computer and System Sciences*, 55(1), 119–139. Retrieved from https://doi.org/10.1006/jcss.1997.1504

Breiman, L., Friedman, J., Stone, C. J., & Olshen, R. A. (1984). *Classification and Regression Trees*. CRC Press.

Scikit-learn Documentation. (2023). GridSearchCV. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html

Powers, D. M. W. (2011). Evaluation: From Precision, Recall, and F-Measure to ROC, Informedness, Markedness, and Correlation. *Journal of Machine Learning Technologies*, 2(1), 37–63.

Scikit-learn Documentation. (2023). Learning Curve. Retrieved from https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.learning_curve.html