Supervised Machine Learning Algorithms for the Analysis of Ship Engine Data

Theodoros Dimitriou[1], Emmanouil Skondras[1], Christos Hitiris[2], Cleopatra Gkola[3], Ioannis S. Papapanagiotou[1], Dimitrios J. Vergados[3], Stavros I. Papapanagiotou[4], Stratos Koumantakis[5], Angelos Michalas[2(✉)], and Dimitrios D. Vergados[1]

[1] Department of Informatics, University of Piraeus, Piraeus, Greece
{theodim,skondras,jpapapanagiotou,vergados}@unipi.gr
[2] Department of Electrical and Computer Engineering, University of Western Macedonia, Kozani, Greece
{c.hitiris,amichalas}@uowm.gr
[3] Department of Informatics, University of Western Macedonia, Kastoria, Greece
{c.gkola,dvergados}@uowm.gr
[4] Internet Business Hellas, Athens, Greece
sip@ibhellas.gr
[5] MAS S.A., Athens, Greece
skoumantakis@maseurope.com

Abstract. Supervised Machine Learning (ML) algorithms are used for making predictions or decisions based on labeled data. In this paper, an overview about existing supervised ML algorithms is performed. In particular, the algorithms that are studied comprehend the Linear Regression (LR), the Ridge Regression (RR), the Decision Tree (DT), as well as Ensemble algorithms. Subsequently, a comparative analysis of the algorithms is performed using a dataset containing data about ship engines. Effective management of ship engines is important for their robust operation, which can then bring significant economic and environmental benefits. Inferences about the condition of engines and predictions about their performance could prove crucial for specifying optimal cruise parameters, early fault detection and timely service planning. The analysis demonstrates the strength and the weaknesses of each algorithm in terms of predicting decay factors of the ship engine by taking into consideration the data included to the aforementioned dataset.

Keywords: Supervised Machine Learning (ML) · Linear Regression (LR) · Ridge Regression (RR) · Decision Tree (DT) · Ensemble algorithms · ship engine data · engine decay prediction

1 Introduction

Supervised machine learning (ML) [1] is a fundamental and powerful approach for solving a wide range of problems in data analysis and prediction, and it forms the basis for many real-world applications of artificial intelligence. Specifically, supervised

L. A. Maglaras and C. Douligeris (Eds.): WiCON 2023, LNICST 527, pp. 97–109, 2024.
https://doi.org/10.1007/978-3-031-58053-6_7

ML algorithms are a class of algorithms used in the field of artificial intelligence and data science for making predictions or decisions based on labelled training data. In supervised learning, the algorithm learns a mapping from input data to output labels by observing and generalizing from a set of example data points where both the input and the corresponding output (target) are known. In general, supervised learning is widely used in various applications, including natural language processing, image recognition, recommendation systems, fraud detection, and healthcare diagnostics.

Some key points about supervised ML algorithms include the data preparation, the classification, and the regression. Specifically, regarding the data preparation, it should be noted that in supervised learning, the dataset is split into two subsets, namely the training data and the validation data. The training data are used to train the model and the validation data are used to evaluate its performance. Accordingly, in classification, the algorithm assigns a category (or label) to input data. Examples include spam email detection (categorizing emails as spam or not) and image classification (identifying objects in images). Finally, regression models predict a continuous numerical value as output. For example, predicting house prices based on features like size, location, and number of bedrooms.

Common metrics used for the evaluation of the performance of supervised learning models include accuracy, precision, recall, F1-score, Mean Squared Error (MSE), and R-squared (for regression) [2, 3]. Also, supervised learning models can suffer from overfitting (fitting the training data too closely) or underfitting (failing to capture the underlying patterns). Techniques like cross-validation and regularization are used to restrict these issues.

Additionally, choosing the right hyperparameters (parameters that are not learned from the data) is essential for optimizing the performance of supervised learning models. Techniques like grid search and random search are often used for hyperparameter tuning. Finally, the scalability of a supervised learning algorithm depends on the size and complexity of the dataset. Some algorithms, like k-nearest neighbours, can be slow for large datasets, while others, like linear models, can scale better.

In this paper, existing supervised ML algorithms are studied. More specifically, the Linear Regression (LR) [4], the Ridge Regression (RR) [5], the Decision Tree (DT) [6] are described, as well as Ensemble [7] algorithms, such as the Bagging Regression (BR) [8], the Random Forest Regression (RFR) [9], the Extra Trees Regression (ETR) [10], the k-Nearest Neighbour (kNN) Regression [11] and the Artificial Neural Network (ANN) [12]. Furthermore, a comparative analysis of the algorithms is performed using the Condition Based Maintenance of Naval Propulsion Plants (CBM) dataset [13] containing data about ship engines. In particular, the CBM dataset contains 11934 records, consisting of the 16 features presented in Table 1.

The main contribution of this work, is that based on the analysis performed, the most effective supervised ML algorithms for the case of shipping data will be selected. The intelligent management of ship engines is an important factor in achieving their optimum efficiency and performance. A key requirement therefore is the use of digital data, models and systems that allow reliable measurement of specific engine performance parameters, their evaluation and the use of the results to make decisions on the operation and maintenance of these engines.

Table 1. The features included to the CBM dataset.

No.	Feature Name	Measurement Unit
1	Lever position (lp)	[1–9]
2	Ship speed (v)	Knots
3	Gas Turbine (GT) shaft torque (GTT))	Kilonewton per meter (kN/m)
4	Revolutions per minute gas turbine shaft (GT rate of revolutions (GTn))	Rounds/Minute (rpm)
5	Gas Generator rate of revolutions (GGn)	Rounds/Minute (rpm)
6	Starboard Propeller Torque (Ts)	Kilonewton (kN)
7	Port Propeller Torque (Tp)	Kilonewton (kN)
8	High Pressure (HP) Turbine exit temperature	Celsius degrees (°C)
9	Air temperature at the gas turbine compressor inlet valve (GT Compressor inlet air temperature)	Celsius degrees (°C)
10	Air temperature at the outlet valve of the gas turbine compressor (GT Compressor outlet air temperature)	Celsius degrees (°C)
11	HP Turbine exit pressure	Bar
12	Air pressure at the gas turbine compressor inlet valve (GT Compressor inlet air pressure)	Bar
13	Air pressure at the gas turbine compressor outlet valve (GT Compressor outlet air pressure)	Bar
14	Gas turbine exhaust gas pressure (GT exhaust gas pressure)	Bar
15	Turbine Injection Control (TIC)	Presentence (%)
16	Fuel flow (mf)	Kilograms per second (Kg/s)

The remainder of the paper is organized as follows: Sect. 2 performs an overview of existing supervised ML algorithms. Subsequently, in Sect. 3 the performance of each algorithm is evaluated. Finally, Sect. 4 concludes our work.

2 Overview of Supervised Machine Learning Algorithms

In this section, existing supervised machine learning algorithms are described including the Linear Regression (LR), the Ridge Regression (RR), the Decision Tree (DT), as well as Ensemble algorithms. Furthermore, the application of these algorithms using the Python sci-kit library [14] is introduced.

2.1 The Linear Regression (LR) Algorithm

The Linear Regression (LR) algorithm is used to predict the value of a variable based on the values of other variables. The variable we want to predict is called dependent variable, while those used to predict other variables are called independent variables.

The algorithm estimates the coefficients of a linear equation, which includes independent variables X that predict the value of the dependent variable Y. LR fits a straight line or surface to minimize the deviations between predicted and actual output values. There are simple linear regression calculations that use the "least squares" method to determine the best possible line that shows the shortest distance from a set of data pairs. Then, the dependent variable Y is estimated from the independent variables X.

LR models are relatively simple and provide an easy-to-interpret mathematical formula that can produce predictions. LR models have become a proven way to predict the future. Since linear regression is a long-established statistical procedure, the properties of linear regression models are well understood and can be trained very quickly.

Code Listing 1 presents an example of the LR algorithm.

- In lines 1 and 2, the necessary libraries are entered. Specifically, the LR model from the scikit-learn and the NumPy libraries is imported.
- In line 3, the data set (dataset X), which includes 2 samples with 2 characteristic values per sample, is specified.
- Line 4 computes the dependent variable Y for each sample of X by performing the inner product of the array X with the vector [2, 4] and adding the constant 5. The inner product yields the output of the values of the array X in the equation:

$$Y = 2 * x_1 + 4 * x_2 + 5 \tag{1}$$

  where x_0 and x_1 are the characteristic values of X.
- In line 5, a Linear Regression object is initially created. Then, via the command model = LinearRegression().fit(X, y), the object is fitted to the given training data (data array X and label values Y).
- In line 6, the prediction is made with the value [8, 9] as input data.
- In line 7, you return the result (array([57.]) which certifies the correct training of the model as for the input data [8, 9] the function $Y = 2*x_1 + 4*x_2 + 5 = 57$.
- Line 8 calculates the prediction coefficient by considering the values of the training data and the given labels. The value of the prediction coefficient ranges between 0 and 1, with 1 shown by line 9 indicating a perfect fit.

Code Listing 1. Usage example of the LR algorithm.

```
1: >> from sklearn.linear_model import LinearRegression
2: >> import numpy as np
3: >> X = np.array([[1, 2], [4, 8]])
4: >> Y = np.dot(X, np.array([2, 4])) + 5  # Y = 2*x1 + 4*x2 + 5
5: >> model = LinearRegression().fit(X, Y)
6: >> model.predict(np.array([[8, 9]]))
7: array([57.])
8: >> model.score(X, Y)
9: 1.0
```

2.2 The Ridge Regression (RR) Algorithm

The Ridge Regression (RR) algorithm aims to avoid overfitting the model by adding a penalty term to the Objective Least Squares (OLS) function. This penalty term is proportional to the square of the magnitude of the coefficients in the regression equation, which

encourages the algorithm to select coefficients with smaller magnitudes. By reducing the size of the coefficients, vertex regression can effectively reduce the complexity of the model and improve its generalization performance to new, unseen data. The RR algorithm can be used in a variety of applications where linear regression is an appropriate modelling technique and where there is a need to control model overfitting.

Code Listing 2 presents an example of the RR algorithm.

- Lines 1 imports the Ridge class from the scikit-learn library.
- Line 2 creates an instance of the Ridge Regression model, with a normalization parameter 'alpha', set to 0.5. The larger the value of alpha, the stronger the normalization effect.
- Line 3 fits the RR model to the data (i.e. learns the relationship between the feature matrix 'X' and the target variable 'y').
- Lines 4 and 5 return the coefficients of the RR function resulting from the training stage.
- In line 6, the constant term of the RR function is returned. In this case is 0.5357142857142847, as shown in line 7.

Code Listing 2. Usage example of the RR algorithm.

```
1: >> from sklearn.linear_model import Ridge
2: >> model = Ridge(alpha=0.5)
3: >> model.fit([1, 2], [2, 3], [3, 4], [4, 5], [2, 4], [5, 6])
4: >> model.coef_
5: array([0.61904762, 0.61904762])
6: >> model.intercept_
7: 0.5357142857142847
```

2.3 The Decision Tree (DT) Algorithm

The Decision Tree (DT) algorithm implements a function that maps a vector of feature values to a single output value, the decision. In order to arrive at its decision, a decision tree performs a sequence of checks, starting from the root and following the appropriate branch, until it reaches a leaf. Each internal node of the tree corresponds to a check on the value of one of the input attributes. The branches from the node are labelled with the possible values of the attribute, and the leaf nodes specify which value should be returned by the function. Input and output values can be discrete or continuous.

Code Listing 3 presents an example of the DT algorithm and the application of cross-validation to evaluate model performance.

- In lines 1 and 2, the decision tree class DecisionTreeRegressor and the cross_val_score model evaluate function and are imported to the program.
- In line 3, the dataset X, which includes 5 samples with 2 characteristic values per sample, is specified.
- Line 4 defines the dependent variable Y for each sample of X.

- In line 5, the DecisionTreeRegressor model is created and in line 6, the model is trained by taking into consideration the dataset X and the dependent variable Y for each sample of X.
- In line 7, 10-fold cross-validation is applied, which evaluates the performance of the decision tree on the data set. The cross_val_score function that is implemented, it takes as parameters the model, the input data array, the target variable vector, and the number of subsets to split the data set into. As output, the function returns a vector of the subset in line 8, which can be used to estimate the overall model return, but also the standard deviation between individual returns.
- Line 9 uses the model to perform a prediction. In this case, the model takes as input the samples that exist to X.
- Line 10 shows the result of the prediction. In this case, we confirm that the model has successfully trained since the result is similar to the content of the dependent variable Y.

Code Listing 3. Usage example of the DT algorithm.

```
1: >> from sklearn.tree import DecisionTreeRegressor
2: >> from sklearn.model_selection import cross_val_score
3: >> X = [[1,0], [2,3], [3,4], [4,3], [5,0]]
4: >> Y = [0, 3, 4, 3, 0]
5: >> model = DecisionTreeRegressor(random_state=0, max_depth=2)
6: >> model.fit(X, Y)
7: >> cross_val_score(model, X, Y, cv=2)
8: array([0.88461538, 0.77777778])
9: >> model.predict(X)
10: array([0., 3., 4., 3., 0.])
```

2.4 The Ensemble Algorithms

The category of the Ensemble algorithms employ a set of prediction algorithms (called "base algorithms" or "base models"). These algorithms combine multiple machine learning techniques to solve a problem, achieving a reduction in bias and variance relative to the individual machine learning models involved [15]. In the following subsections, the Ensemble algorithms are implemented using the indicative dataset $X = [[1, 0], [2, 3], [3, 4], [4, 3], [5, 0]]$, which includes 5 samples with 2 characteristic values per sample. Also, the dependent variable $Y = [0, 3, 4, 3, 0]$ for each sample of X is used. Both X and Y can be considered as an indicative part of the CBM dataset that will be analysed in the next section.

The Bagging Regression (BR) Algorithm. A common implementation of ensemble ML algorithms is called "pooling", where for base models the same algorithm is employed multiple times, but each copy of the algorithm is trained with a different and random subset of the dataset. The two most common subcategories of the pooling implementation are the Pasting and the Bootstrap AGGregating" (Bagging). In the Pasting algorithm, random subsets of the dataset are created. Accordingly, in the Bagging algorithm, the subsets of the dataset are created by reshuffling. Bagging generally

performs better than Pasting, which is why it is preferred. However, if possible, cross-validation can be used to compare the two algorithms [16]. In Bagging sampling, when the number of samples of each classifier equals the total of samples, due to sample reshuffling, on average about 63% of the samples in each classifier are selected, with the remaining 37% not used at all for training, so can be used to verify each classifier. This remaining subset, which is different for each classifier, is called out-of-bag.

Code Listing 4 shows how the Bagging Regression (BR) is used with Support Vector Regression (SVR) as the base estimator. The model is trained on an artificial dataset constructed by the make_regression function. Then, evaluates its performance on a sample input.

- In lines 1 and 2, the BaggingRegressor and the DecisionTreeRegressor classes are imported to the program.
- In line 3, the BaggingRegressor algorithm is initialized with base estimator an instance of the DecisionTreeRegressor, with 5 instances of the base estimators (n_estimators = 5) and seed equal to 5 for the random number generator (random_state = 5). Also, in this line, the algorithm is trained by calling fit, which takes as parameters the input dataset X and the target variable vector Y.
- In line 4, the trained algorithm is called in order to make a prediction by taking as input the new sample [2, 4].
- In line 5, the result of the algorithm's prediction for the sample [2, 4] is presented.

Code Listing 4. Usage example of the BR algorithm.

```
1: >>> from sklearn.ensemble import BaggingRegressor
2: >>> from sklearn.tree import DecisionTreeRegressor
3: >>> model = BaggingRegressor(estimator=DecisionTreeRegressor(), n_estimators=5,
       random_state=5).fit(X, Y)
4: >>> model.predict([[2, 4]])
5: array([3.4])
```

The Random Forest Regression (RFR) Algorithm. A special class of model collection algorithms called Random Forests employ for base models, trees, as well as random sampling with the Bagging algorithm or less commonly with Pasting. However, the algorithm differs in its tree implementation in terms of feature selection of each branch, where the optimum is not sought from all features, but the optimum from a random subset of features.

Code Listing 5 shows how the Random Forest Regression (RFR) is used. The model is trained on an artificial dataset constructed by the make_regression function and then evaluates its performance on a sample input.

- In line 1, the RandomForestRegressor class is imported to the program.
- In line 2, the RandomForestRegressor algorithm is initialized, with a maximum tree depth equal to 2 (max_depth = 2) and a seed of 5 for the random number generator (random_state = 5). Also, in this line, the algorithm is trained by calling fit, which takes as parameters the input dataset X and the target variable vector Y.

- In line 3, the trained algorithm is called in order to make a prediction by taking as input the new sample [2, 4].
- In line 4, the result of the algorithm's prediction for the sample [2, 4] is presented.

Code Listing 5. Usage example of the RFR algorithm.

```
1: >>> from sklearn.ensemble import RandomForestRegressor
2: >>> model = RandomForestRegressor(max_depth=2,
       random_state=5).fit(X, Y)
3: >>> model.predict([[2, 4]])
4: array([3.35])
```

The Extra Trees Regression (ETR) Algorithm. A variant of Random Forests called Extremely Randomized Trees or Extra-Trees, randomly chooses the separation threshold at each distinction of the tree, resulting in much faster training, since finding this threshold is computationally demanding. Random Forests, despite its simplicity, is one of the most powerful ML algorithm available today [16].

Code Listing 6 shows how Extra Trees Regression (ETR) is used through a reference part of the CBM dataset, where the included data are split into training and test data subsets. The model is then trained with the training subset and evaluated with the test subset.

- In lines 1 and 2, the ExtraTreesRegressor class and the train_test_split function are imported to the program.
- Line 3 splits the dataset into training and test sets using the train_test_split function, with a random_state specified to make the split reproducible.
- In line 4, the ExtraTreesRegressor algorithm is initialized, with 10 instances of base estimators (n_estimators = 10) and with a seed of 5 for the random number generator (random_state = 5). Also, in this line, the algorithm is trained by calling fit, which takes as parameters the input dataset X_train and the target variable vector Y_train.
- Line 5 evaluates the performance of the ExtraTreesRegressor algorithm on the test set by calling the model.score method with the X_test and Y_test as inputs.
- Line 6 outputs the result of evaluating the algorithm's performance on the test data.

Code Listing 6. Usage example of the ETR algorithm.

```
1: >>> from sklearn.ensemble import ExtraTreesRegressor
2: >>> from sklearn.model_selection import train_test_split
3: >>> X_train, X_test, Y_train, Y_test = train_test_split(X, Y,  test_size=0.4, random_state=2)
4: >>> model = ExtraTreesRegressor(n_estimators=10, random_state=5).fit(X_train, Y_train)
5: >>> model.score(X_test, Y_test)
6: 0.86375
```

The k-Nearest Neighbor (kNN) Regression Algorithm. The k-Nearest Neighbor (kNN) Regression algorithm prediction result for the value of a feature i depends on the k closest (neighboring) values present in the training data set. Specifically, the distance of the value of feature i from all existing values is calculated and the k closest

values are selected. This calculation is carried out using formula (2), where t expresses the timestamp of receiving the values x1 and x2 for feature i. Formula (3) calculates the target value, where the set Nk(x) contains the indices of the k nearest neighbors of the input value x for feature i, the parameter yi expresses the label specified for feature i [17].

$$d(x_1, x_2) = \left(\sum_{i=0}^{t} (x_{1i} - x_{2i})^2\right)^{\frac{1}{2}} \tag{2}$$

$$f_{kNN}(x) = \frac{1}{k} \sum_{i \in N_k(x)} y_i \tag{3}$$

Code Listing 7 shows the use of the kNN Regression algorithm of the scikit-learn library.

- In line 1, the KNeighborsRegressor class is imported to the program.
- Line 2 creates initializes the kNN algorithm and assigns the value 3 to the n_neighbors variable. Thus, it is specified that the model will consider the 3 nearest neighbors (the three closest values) whenever it is asked to make a prediction for a new input value.
- In line 3, the algorithm is trained using the dataset X and the corresponding outputs specified to the vector Y.
- In line 4, the trained algorithm is called in order to make a prediction by taking as input the new sample [2, 4].
- In line 5, the result of the algorithm's prediction for the sample [2, 4] is presented.

Code Listing 7. Usage example of the kNN Regression algorithm.

```
1: >>> from sklearn.neighbors import KNeighborsRegressor
2: >>> model = KNeighborsRegressor(n_neighbors=3)
3: >>> model.fit(X, Y)
4: >>> model.predict([[2, 4]])
5: array([3.33333333])
```

The Artificial Neural Network (ANN) Algorithm. A multilayer feedforward neural network (multilayer perceptron) was also used to predict engine damage values. The implementation of the Artificial Neural Network (ANN) was done with the Tensorflow library [15], through the Keras programming interface [19]. In contrast to the implementations of algorithms in the scikit-learn library, an ANN algorithm in Tensorflow arises as a composition of individual modules, with the aim of building an appropriate architecture for each application.

The layers of neural nodes inserted into the model to construct the ANN are instances of the tf.keras.layers.Dense class [20], which implements a standard densely connected neural network with the formula: output = activation(dot(input, kernel) + bias), where activation is the per-element activation function passed as the activation argument, dot is the inner product operation, kernel is a matrix of weights generated by layer and bias is a vector of constant terms generated by layer (only valid if use_bias is True).

The use of Dropout levels is a popular deep ANN normalization technique that has proven to be highly successful improving the accuracy of even advanced models by

1–2% [16]. This is a special class of layers that are inserted between normal layers and in each iteration, only during the training phase, they disable random neural nodes of the model, as a result of which they do not participate at all in the current iteration. A node can be disabled in one iteration but active in another, with the probability of being disabled being determined by the dropout rate hyperparameter.

Code Listing 8 presents an example of the RR algorithm.

- In lines 1–3, the Sequential, Dense and Dropout classes are entered into the program respectively.
- In line 4, we initialize and assign to the model variable an object of the tf.keras.Sequential class, in which we will "build" our neural network by adding the levels of the neurons.
- In line 5, we add the first layer of densely connected neurons of type tf.keras.layers.Dense, in which we define the number of neurons (units = 6), the activation function (activation = 'sigmoid') and the number of inputs (input_dim = 14). The last parameter automatically creates a level of type tf.keras.Input and is only needed when creating the first level, if we have not explicitly specified an input level.
- In line 6, we add a level of type Dropout with a probability of turning off neurons, during the training phase, equal to 0.2.
- In line 7, we add another layer of neurons with one neuron for the output of the ANN.

Code Listing 8. Usage example of the ANN algorithm.

```
1: >>> from keras.models import Sequential
2: >>> from keras.layers import Dense
3: >>> from keras.layers import Dropout
4: >>> model = Sequential()
5: >>> model.add(Dense(untis=6, input_dim=14, activation='sigmoid'))
6: >>> model.add(Dropout(rate=0.2))
7: >>> model.add(Dense(untis=1, activation='sigmoid'))
```

3 Performance Analysis

In this section, the supervised ML algorithms are evaluated using the CBM dataset. As mentioned to Section 1, the CBM dataset contains 11934 records and each consists of 16 features. However, for the evaluation of the algorithms, only the 14 features will be considered, since the features 9 (gas turbine compressor inlet air temperature) and 12 (gas turbine compressor inlet air pressure) obtain the same value in the entire records and, thus, they are considered as non-important for our analysis.

Figure 1 presents the average absolute error of gas turbine compressor decay factor accomplished by each ML algorithm. As it is observed, the three most efficient algorithms belong to the ensemble methods and are Extra Trees Regressor, the Random Forest and the Bagging Regressor. This is followed by the decision tree algorithm Decision Tree Regressor and with lower yields, KNN Regressor, Linear Regressor, Ridge Regressor and Artificial Neural Network.

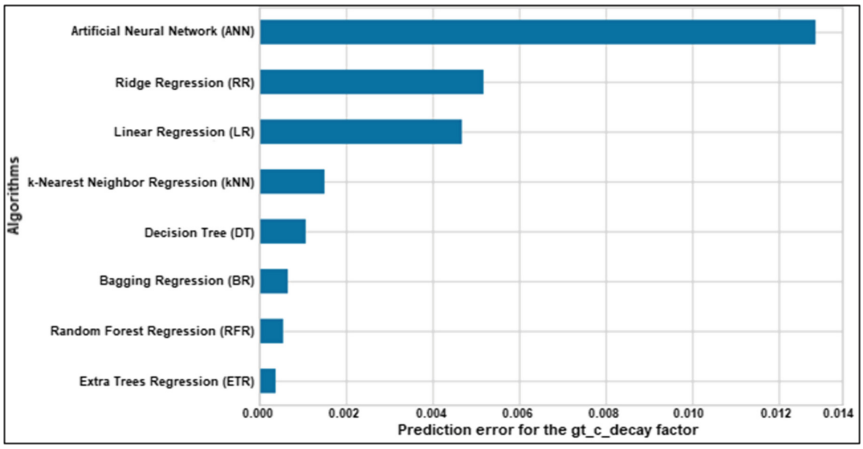

Fig. 1. Average absolute error of gas turbine compressor decay factor based on the CBM data set.

We observe that the algorithms of model collection methods (ensemble) show the best performances. This be justified by considering that the ensemble methods combine multiple machine learning models, achieving a reduction in bias and variance relative to the individual base models they include. In this case, all three ensemble algorithms are based on decision tree estimators, so it is expected that they will have improved performance over the next in line algorithm, that of the single decision tree Decision Tree Regressor. The Decision Tree Regressor and kNN show close performance, outperforming the next two linear regression algorithms Linear Regressor and Ridge Regressor. This is true as the Decision Tree Regressor and kNN algorithms can learn non-linear relationships. ANN accomplishes the worst performance, because the construction and parameterization of the network is particularly complex and in case it is selected as a prediction model, further investigation will be needed for its implementation.

Regarding the prediction of every other feature of the dataset, similar results are shown to the predictions of engine decay (Table 2). The three most efficient algorithms belong to ensemble methods. This is followed by the decision tree algorithm Decision Tree Regressor and, with lower yields, KNN Regressor, linear regression algorithms, and finally the ANN. In this table, we also observe that for the features gt_shaft and gg_rate all prediction algorithms show large errors in predicting their values. Furthermore, concerning the gt_rate feature, we observe that the Linear. Regressor, Ridge Regressor, KNN Regressor and the ANN show large errors in predicting its values, while the rest algorithms succeed sufficient prediction results. For the rest features, sufficient prediction results are succeeded by the most algorithms.

Table 2. Comparative results about the prediction of each characteristic of the CBM dataset.

Name	Linear Regressor	Ridge Regressor	KNN Regressor	Bagging Regressor	Random Forest	Extra Tree Regressor	Decision Tree Regressor	Artificial Neural Network
level_position	0.010	0.011	0.007	0.000	0.000	0.000	0.000	4.150
ship_speed	0.033	0.035	0.022	0.000	0.000	0.000	0.000	13.948
gt_shaft	131.513	178.364	18.994	7.568	6.419	4.369	11.51	27119.9
gt_rate	16.139	19.619	3.368	0.111	0.099	0.063	0.127	2130.34
gg_rate	52.955	85.180	9.334	2.020	1.867	1.458	2.402	8191.31
sp_torque	0.000	0.000	0.338	0.003	0.002	0.002	0.003	225.245
pp_torque	0.000	0.000	0.338	0.003	0.002	0.002	0.004	225,245
hpt_temp	2.280	3.768	2.416	0.581	0.491	0.332	0.872	733.297
gt_c_o_temp	1.514	1.588	0.455	0.263	0.232	0.153	0.388	644.734
hpt_pressure	0.005	0.006	0.000	0.000	0.000	0.000	0.000	1.347
gt_c_o_pressure	0.056	0.063	0.010	0.008	0.007	0.005	9.013	11.264
gt_exhaust_pressure	0.000	0.000	0.000	0.000	0.000	0.000	0.000	0.0299
turbine_inj_control	2.308	2.242	0.894	0.088	0.069	0.046	0.092	32,649
fuel_flow	0.003	0.004	0.002	0.000	0.000	0.000	0.000	0.127

4 Conclusion

In this paper, an overview about existing supervised ML algorithms performed. The Linear Regression (LR), the Ridge Regression (RR), the Decision Tree (DT), as well as Ensemble algorithms were studied. Ensemble algorithms include the Bagging Regression (BR), the Random Forest Regression (RFR), the Extra Trees Regression (ETR), the k-Nearest Neighbor (kNN) Regression and the Artificial Neural Network (ANN). Furthermore, a comparative analysis of these algorithms performed using the CBM dataset. In particular, a dataset containing data about ship engines used, resulting to the evaluation of each algorithm in terms of predicting decay factors of the ship engine. Based on the analysis performed, future work includes the application of the most effective supervised ML algorithms for the analysis of ship engine data that will be produced from sensors installed in specific ships.

Acknowledgements. This work has been partly supported by the University of Piraeus Research Center (UPRC).

References

1. Rawson, A., Brito, M.: A survey of the opportunities and challenges of supervised machine learning in maritime risk analysis. Transp. Rev. J. **43**(1), 108–130 (2023)
2. Jafar Zaidi, S.A., Chatterjee, I., Brahim Belhaouari, S.:COVID-19 tweets classification during lockdown period using machine learning classifiers. In: Applied Computational Intelligence and Soft Computing, Hindawi, pp. 1–8 (2022)

3. Aram, S.A., et al.: Machine learning-based prediction of air quality index and air quality grade: a comparative analysis. Int. J. Environ. Sci. Technol. 1–16 (2023)
4. Kim, T., Sharda, S., Zhou, X., Pendyala, R.M.: A stepwise interpretable machine learning framework using linear regression (LR) and long short-term memory (LSTM): city-wide demand-side prediction of yellow taxi and for-hire vehicle (FHV) service. Transp. Res. Part C: Emerg. Technol. **120** (2020)
5. Suhail, M., Chand, S.: Performance of some new ridge regression estimators. In: 13th International Conference on Mathematics, Actuarial Science, Computer Science and Statistics (MACS), pp. 1–4. IEEE (2019)
6. Costa, V.G., Pedreira, C.E.: Recent advances in decision trees: an updated survey. Art. Intell. Rev. **56**(5), 4765–4800 (2023)
7. Zhang, Y., Liu, J., Shen, W.: A review of ensemble learning algorithms used in remote sensing applications. Appl. Sci. **12**(17), 1–20 (2022)
8. Tekouabou, S.C.K., Cherif, W., Silkan, H.: Improving parking availability prediction in smart cities with IoT and ensemble-based model. J. King Saud Univ.-Comput. Inf. Sci. **34**(3), 687–697 (2022)
9. El Mrabet, Z., Sugunaraj, N., Ranganathan, P., Abhyankar, S.: Random forest regressor-based approach for detecting fault location and duration in power systems. Sensors **22**(2), 1–19 (2022)
10. Wang, K., Zhang, D., Shen, Z., Zhu, W., Ye, H., Li, D.: Novel ship fuel consumption modelling approaches for speed and trim optimisation: using engine data as auxiliary. Ocean Eng. **286**, 1–12 (2023)
11. Viale, L., Daga, A.P., Fasana, A., Garibaldi, L.: Least squares smoothed k-nearest neighbors online prediction of the remaining useful life of a NASA turbofan. Mech. Syst. Signal Process. **190**, 1–15 (2023)
12. Veza, I., et al.: Review of artificial neural networks for gasoline, diesel and homogeneous charge compression ignition engine. Alex. Eng. J.. Eng. J. **61**(11), 8363–8391 (2022)
13. The Condition Based Maintenance of Naval Propulsion Plants (CBM) dataset. https://www.kaggle.com/datasets/elikplim/maintenance-of-naval-propulsion-plants-data-set. Accessed 30 Oct 2023
14. Scikit-learn library. https://scikit-learn.org. Accessed 30 Oct 2023
15. Russell, S., Norvig, P.: Artificial Intelligence: A Modern Approach. Hoboken (2020)
16. Géron, A.: Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems, 2nd edn. O'Reilly Media, Inc. (2019)
17. Poloczek, J., Treiber, N.A., Kramer O.: KNN regression as geo-imputation method for spatio-temporal wind data. In: de la Puerta, J., et al. (eds.) International Joint Conference SOCO'14-CISIS'14-ICEUTE'14. AISC, vol. 299, pp. 185–193. Springer, Cham (2014). https://doi.org/10.1007/978-3-319-07995-0_19
18. The Tensorflow library. https://www.tensorflow.org. Accessed 30 Oct 2023
19. The Keras programming interface. https://keras.io. Accessed 30 Oct 2023
20. The Dense class of the Keras programming interface. https://www.tensorflow.org/api_docs/python/tf/keras/layers/Dense. Accessed 30 Oct 2023