Statistical Learning for Predictive Maintenance

Assignment

Deepak Kovaichelvan - S336570

Date: 29th November 2021

Information categorisation: Open

Contents

1. Introduction

Failure prediction is a major focus area in predictive maintenance in several industries, such as the aerospace industry. Failure prediction helps aircraft manufacturers adopt proactive maintenance practices, aimed at minimising failures and resulting downtime.

This assignment looks into aircraft engine sensor data together with time in number of engine cycles, which have been used to train machine learning models to predict failures on unseen engine data.

2. Task Definition

The dataset is provided in two parts – a train and a test dataset. The train dataset is used to create a machine learning model using supervised learning methods. The generated model is assessed on the test dataset to evaluate the performance (using error or accuracy scores).

The train dataset contains measurements from 4 sensors from 100 engines and simulates run-to-failure operation. Each row of sensor readings is available with the Time-To-Failure and a classification label, which is set to 1 when TTF is within the last 30 stages of engine operation. The dataset also comes with the engine ID and engine cycle count for each of the rows.

The test dataset is similar to the train data, with the main difference being it contains the 4 sensor readings from a randomly selected cycle of operation from 100 different engines. The objective to develop models to predict:

1. Time-To-Failure
2. Classification of which engine will fail in the specified time period

3. Exploratory Data Analysis

Exploratory Data Analysis (EDA) is a critical initial step in machine learning, which involves performing investigations to find patterns, correlations and anomalies in the dataset. The process also helps understand the structure of the underlying data. EDA aids in developing good understanding of the data before diving into the model development [1].

The coding for this assignment was carried out on Python. EDA was performed on the train data, as this has a larger dataset which simulates run-to-failure to help better visualise the data.

3.1. Variables

The train dataset was imported with the name *train_data* in the form of a dataframe (table) using the pandas package. The first 5 rows are shown in Table 1, which summarises the variables available for modelling.

Table 1: First 5 rows of train_data

	id	cycle	s1	s2	s3	s4	ttf	label_bnc
0	1	31	1398.91	554.42	47.23	521.79	112	0
1	2	49	1410.83	553.52	47.67	521.74	98	0
2	3	126	1418.89	552.59	47.88	520.83	69	0
3	4	106	1406.88	552.64	47.65	521.88	82	0
4	5	98	1419.36	553.29	47.46	521.00	91	0

A dependent variable (also called target) is the variable that a machine learning model predicts using the independent variables provided as inputs. It is assumed that the independent variables have a direct impact on the dependent variables. This hypothesis is tested in the EDA.

The independent variables provided are:

1. id – Engine identification number
2. cycle – Time of measurement represented in engine cycles
3. s1, s2, s3 and s4 – Measurements from sensors 1 to 4

The dependent variables are:

1. ttf – Time to failure represented in number of engine cycles
2. label_bnc – Classification label set to 1 when a failure occurs within the last 30 cycles

3.2. Data Summary

Table 2 shows the variables and that there are 20631 rows of data. This is due to the fact that each cycle contains a row for each of the 100 engines, meaning that the average TTF for the engines is 206 cycles.

The table also shows the data types of each of the variables (columns) and confirms there are no missing values, which can hinder the performance of the model. Therefore, there is no need for data cleaning operation to rectify the missing values.

Table 2: Summary of info method applied on train_data

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 20631 entries, 0 to 20630
Data columns (total 8 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   id         20631 non-null  int64
 1   cycle      20631 non-null  int64
 2   s1         20631 non-null  float64
 3   s2         20631 non-null  float64
 4   s3         20631 non-null  float64
 5   s4         20631 non-null  float64
 6   ttf        20631 non-null  int64
 7   label_bnc  20631 non-null  int64
dtypes: float64(4), int64(4)
memory usage: 1.3 MB
```

3.3. EDA - Data Visualisation

The pairplot was created on the *train_data* dataframe to visualise pairwise relationship of variables, the results of which are shown in Figure 1. It displays a grid of axes with each variable forming the x and y axes. Each grid is a scatter plot of a pair of variables. The diagonal plots show the probability distributions of the variables [2].

Some of the observations are:

1. TTF is directly proportional to sensor measurements s2 and s4 and is inversely proportional to s1 and s3. The relationship appears to be non-linear.
2. The failure classification label_bnc follows the opposite trends to TTF with respect to the sensor readings 1 to 4. This is because the classification label reaches 1 when the TTF approaches 0.
3. The diagonal plots confirm the that the variables are normally distributed, this is an underlying assumption in most machine learning models and is beneficial for modelling.

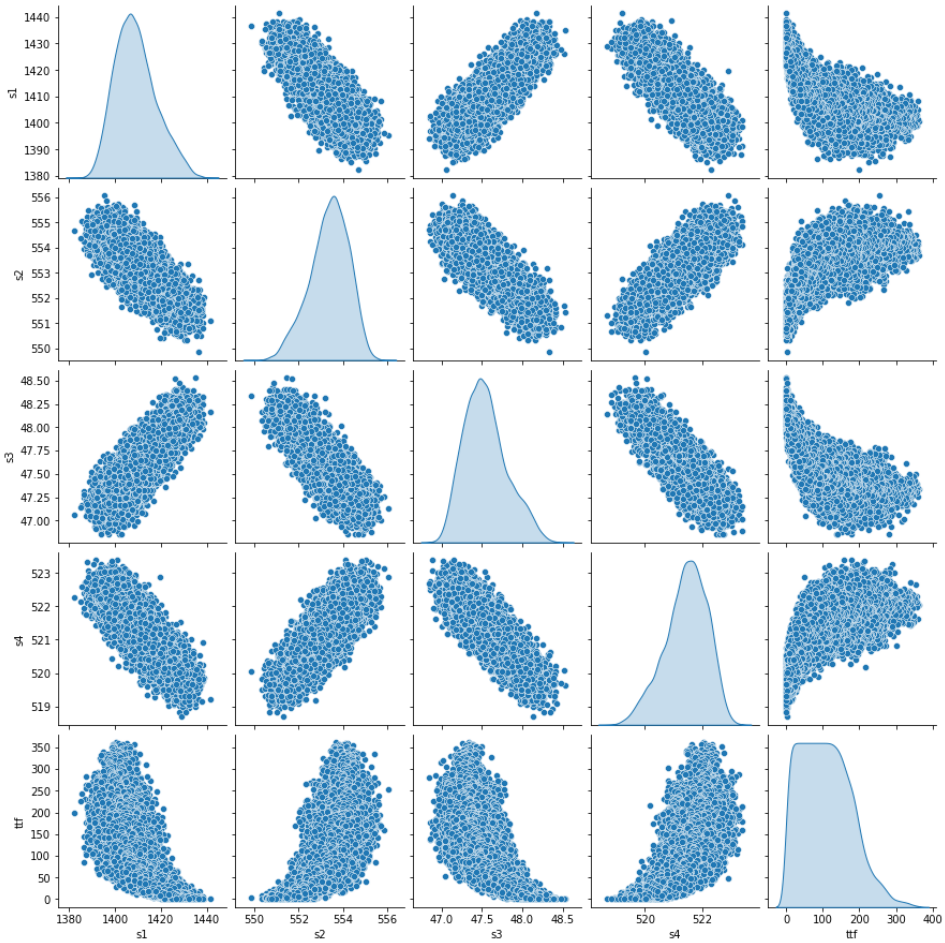


Figure 1: Pairplot of the train_data dataframe variables

3.4. Distribution of Variables

To understand the distribution of the TTF of the engines, a probability distribution has been plotted on the maximum cycle for each engine ID in Figure 2. The plot shows that the distribution is skewed to the left, with the highest distribution around 200 cycles, which is closely corresponds to the mean.

The minimum number of cycles to failure for an engine is 127 cycles and the maximum is 361 cycles with a standard deviation of 46 cycles.

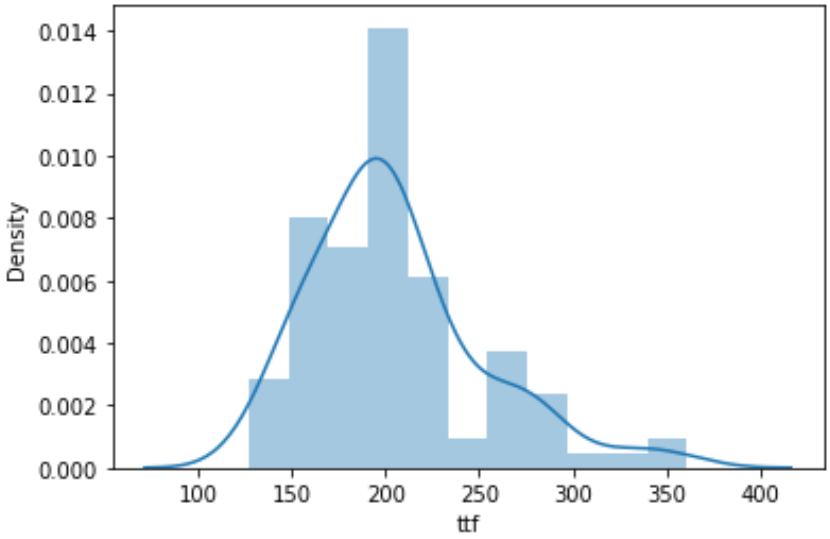


Figure 2: Probability distribution function of TTF

An independent variable without significant variability is unlikely to have an influence on the dependent variable. Therefore, the variability of independent variables is very important. It can be seen from Figure 3 that s1 has a significantly higher standard deviation than the other variables, with a value of 9, and s3 has the least standard deviation of 0.27.

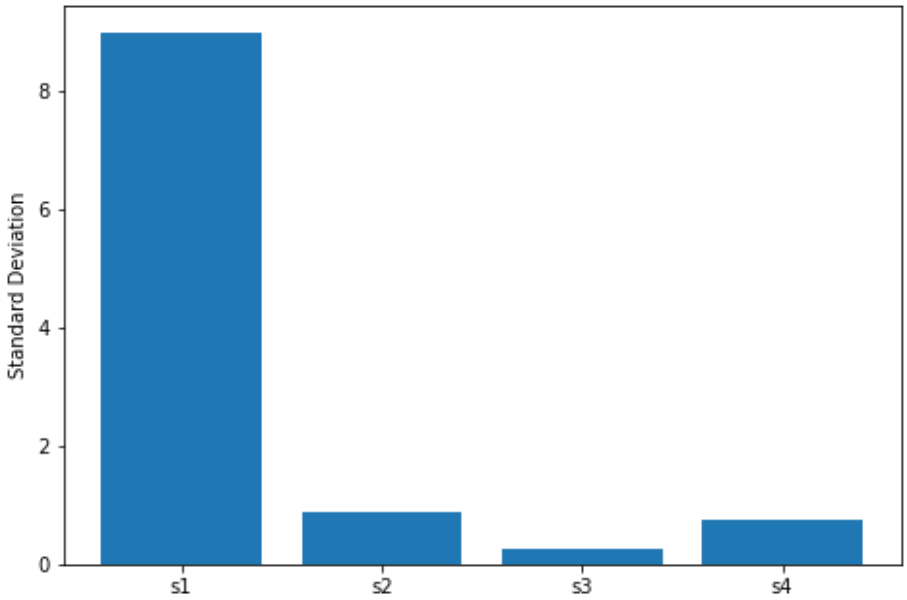


Figure 3: Standard deviation of the sensor readings

3.5. Correlation

While the pairplot is useful to visually observe the direction of correlation between variables, another useful metric is the correlation matrix, which also helps to understand the magnitude. The variables are similarly laid out on a grid of axes, the numbers (-1 to 1) and colours of the grids indicate the magnitude of correlation between the variables. Value of 0 indicates no correlation, values close to 1 and -1 strong positive and negative correlations respectively [3].

The correlation matrix for the train_data is shown in Figure 4, which confirms:

1. There is high correlation between the independent variables and TTF. The independent variables are expected to be good predictors of TTF.

2. There is high correlation between the sensors. This could lead to some uncertainty in the estimates of regression coefficients.

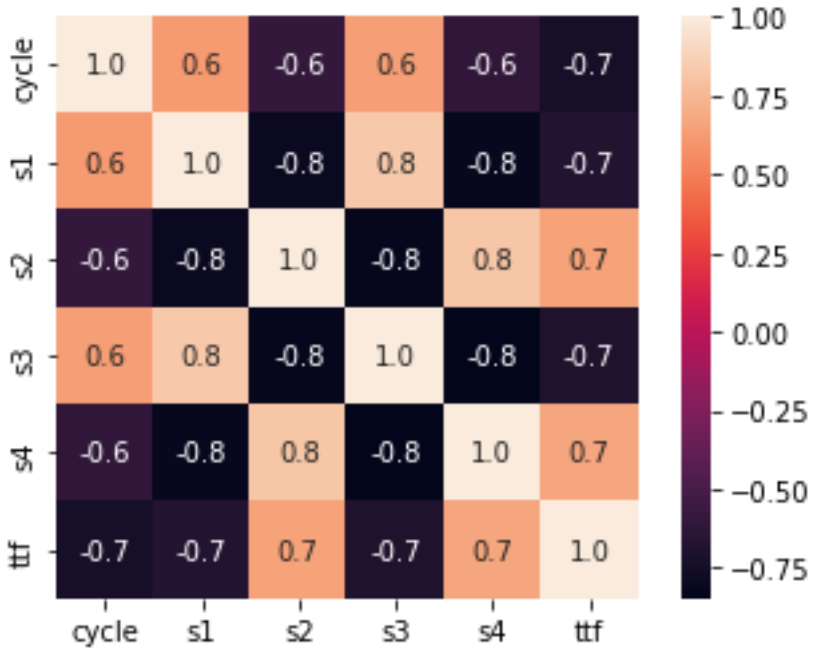


Figure 4: Correlation matrix of the variables

3.6. Class Imbalance

When a dataset has lot more of one classification label than the other, it is said to exhibit class imbalance. This problem's dataset has this property as it has far more zero labels (negatives) than ones (positives), as highlighted in Figure 5.

This impacts the metrics to be used for evaluating the model performance and is discussed further in the classification results section.

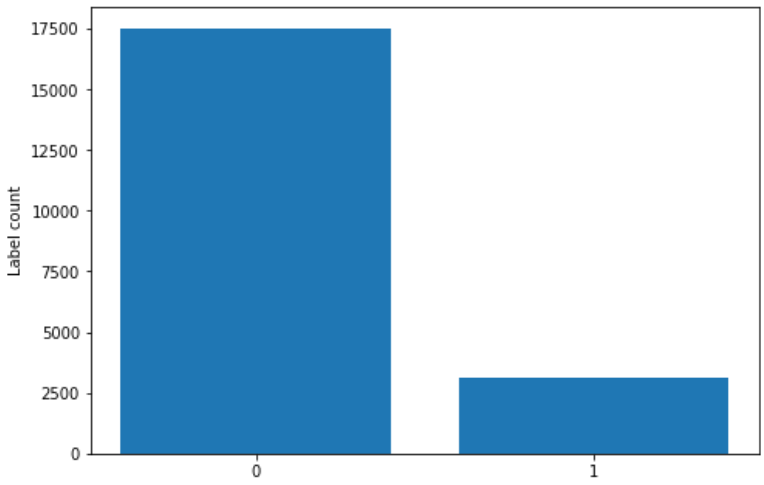


Figure 5: Class imbalance of labels

4. Feature Selection

Although the dataset has a set of independent variables (also called features) as discussed in section 2, which could all be used to train the model, it is important to assess the suitability of the features for the purpose it is intended to serve. A good model should be able to predict on unseen engines to be useful.

Secondly, a closer look at the test and train data reveals the engine IDs used in the two datasets do not mean the same, as the sensor readings for the same ID and cycle count differ as shown in Table 3. Therefore, training on IDs is expected to lead to worse results and have not been used in the models.

Table 3: Comparison of train and test dataset for same ID and cycle count

id	cycle	s1	s2	s3	s4	ttf	label_bnc	dataset
1	31	1398.91	554.42	47.23	521.79	112	0	**Test**
1	31	1396.9	554.13	47.41	521.95	161	0	**Train**

5. Regression

5.1. Regression Models

Since TTF is a quantitative variable, which takes on numerical values, the appropriate technique to use is regression [3]. Machine learning models have been built using scikit-learn, a machine learning library for Python. In this section, following models were built:

- **Linear Regression:** The baseline model used was linear regression, as this is the most basic regression method. This method determines best fit by minimising the sum of squared residuals (vertical distance from the data point to the regression line)

- **Polynomial Regression:** It was seen in the EDA that the relationship between TTF and sensor readings is non-linear, therefore a polynomial regressor was trialled to improve upon the linear regression model.

- **Decision Tree Regression:** Decision Tree is another machine learning algorithm, which can handle non-linearity. This method uses an if-else decision model to generate tree like structure using weighted MSE as the basis of the splits.

- **Random Forest Regression:** Random Forest is an ensemble method that works by combining the decisions from multiple independently trained decision trees (weak models) to arrive at a final decision through voting. Random forests are less prone to overfitting than decision trees and robust to outliers, hence are better with unseen data.

5.2. Methodology

With the dataset already loaded to Python and train and test data split into target and features, the below methodology was repeated for multiple regression methods:

1. Train the model using the *train_data* features and target (TTF).

2. Evaluate quality of fit of the model on *train_data* using the R^2 metric. This is a measure of the proportion of variance that can be explained by the independent variables [4]. Value of 1 means the all of the variance is explained by the model.

3. Use k-fold cross validation on *train_data* to evaluate performance (measured in R^2) on each of the k-fold validation sets and examine the extent to which resulting fits differ to detect overfitting [3]. K-fold provides the opportunity to assess performance on different sets of unseen data. In this case, k of 10 was used to evaluate performance.

4. Predict on *test_data* and evaluate performance using R^2 (percentage terms) and Root Mean Squared Error (absolute terms), which both quantify the ability to predict the value of the response variable.

5.3. Results and Discussion

Table 4: Comparison of results from various regressors

	R^2 (Train)	CV Score	R^2 Test (Test)	RMSE (Test)
Linear Regression	0.64	0.64	0.39	32.36
Polynomial Regression	0.70	0.64	0.54	28.11
Decision Tree Regression	0.70	0.68	0.52	28.86
Random Forest Regression	0.71	0.70	0.56	27.59

Table 4 shows the results from the different regressors trialled with tuned hyperparameters using grid search cross validation.

It can be seen that the regressors capable of handling non-linearity perform better. Therefore, linear regressor has the worst performance as it underfits to the training data, leading to higher training and testing errors, and comes out as the worst performer.

In comparison, polynomial and decision tree regressors achieve better fits compared to linear regression. The main hyper parameter for decision tree is the max depth of the tree, which not limited will fit perfectly to the training data, resulting in overfitting. Similarly, for polynomial regression, degree is the most important hyperparameter.

The best error results are from the random forest regressor, as this method reduces the variance in predictions while managing to achieve similar training variance to decision tree. The model was tuned by performing grid search cross validation on the maximum depth of the trees, number of features per tree and number of trees to take part in voting.

Figure 6 clearly shows that the underlying residuals (vertical distance from the points to the best fit line) are higher in linear regression when compared to the other three models.

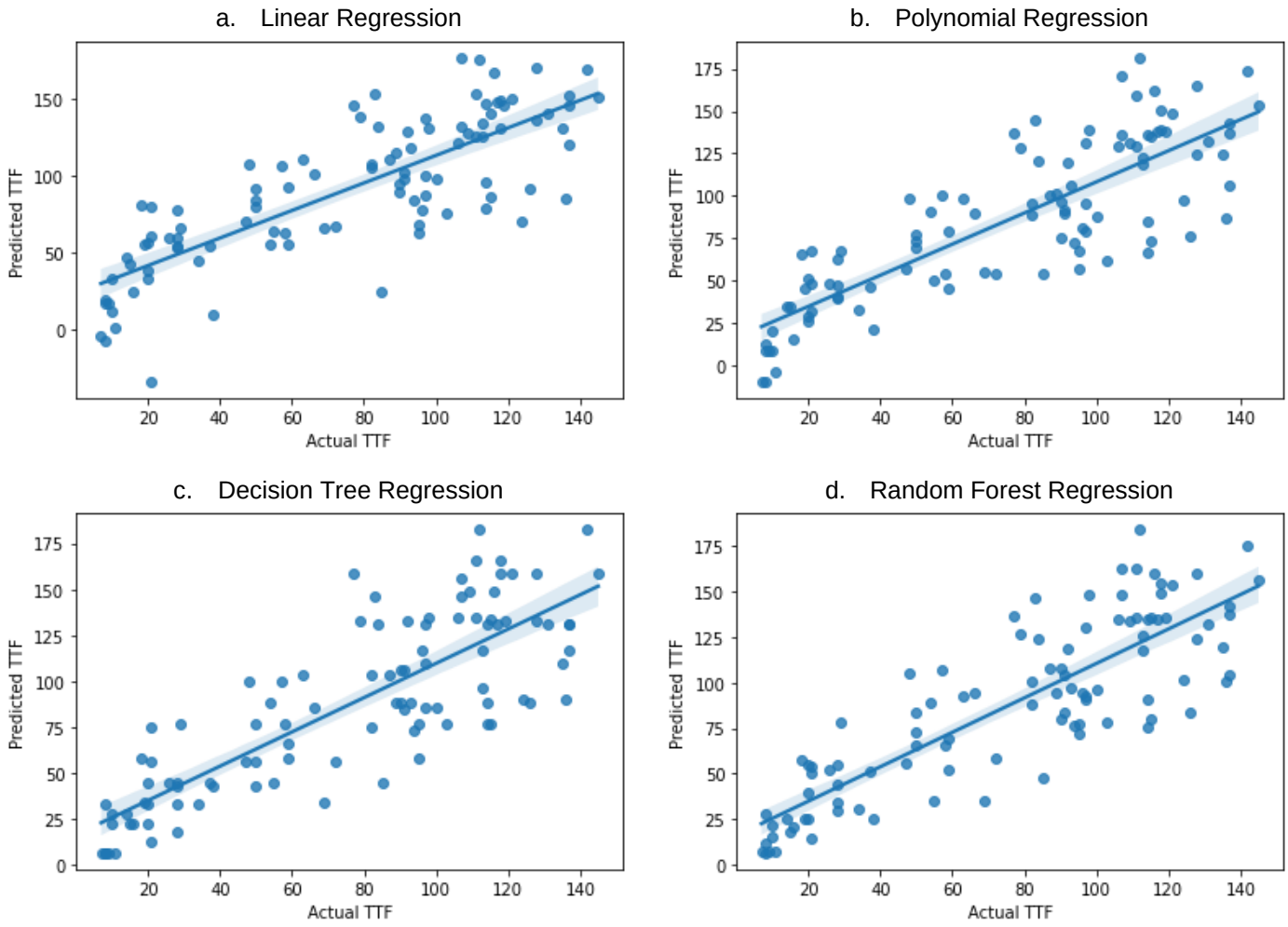


Figure 6: Actual vs Predicted from the regression models

6. Classification

6.1. Classification Models

Since the label is a categorical variable, the appropriate technique to use for the problem is classification. The following classification models were built:

Logistic Regression: Although logistic regression is a classification method, it carries the name regression, as it uses the same equations as linear regression, but passes through a sigmoid function. This enables to predict the probability of discrete category, which is used to determine the labels.

Decision Tree Classifier: Decision Tree classifier is the classification version of the decision tree regressor. It uses entropy as the basis of the splits, in place of weighted MSE.

Random Forest Classifier: Random Forest classifier is the classification version of the random forest regressor, which is discussed in the regression section.

K Nearest Neighbours (KNN): This method is a unique method compared to the other methods used. It does not need the training data until the testing phase. The decision on the test data points is made by majority voting of its k nearest neighbours. Therefore, k is the most important hyper parameter.

6.2. Methodology

1. Train the model using the *train_data* features and target (class label).

2. Evaluate quality of fit of the model on *train_data* using the metrics – accuracy, precision, recall, F1 score and ROC AUC.

3. Use k-fold cross validation on *train_data* to evaluate performance (measured in accuracy) on each of the k-fold validation sets and examine the extent to which resulting fits differ to detect overfitting [3]. In this case, k of 10 folds was used to evaluate performance.

4. Predict class labels on *test_data* and evaluate performance of the models using the same metrics.

6.3. Results and Discussion

The training metrics in Table 5 show the quality of fit of the model to the test data. The high scores of the metrics show that all the models have fit well to the training data to a similar extent. Comparing to the test metrics will show the performance of the model on unseen data.

The test metrics from the various classification methods are summarised in Table 6. The drop in performance on the test data is a result of the overfit to the noise and outliers in the training data. The class imbalance is also expected to have contributed to this drop.

Table 5: Performance of models on the training data

	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	0.95	0.84	0.78	0.81	0.98
Decision Tree Regression	0.95	0.86	0.76	0.81	0.97
Random Forest Regression	0.95	0.88	0.80	0.84	0.99
K Nearest Neighbours	0.95	0.84	0.78	0.81	0.98

Table 6: Performance of models on the test data

	Accuracy	Precision	Recall	F1 Score	ROC AUC
Logistic Regression	0.9	0.89	0.68	0.77	0.96
Decision Tree Regression	0.89	0.88	0.64	0.74	0.94
Random Forest Regression	0.9	0.89	0.68	0.77	0.96
K Nearest Neighbours	0.86	0.87	0.52	0.65	0.92

Accuracy scores of all the four classifiers are quite comparable, with the logistic regression and random forest models performing best and the other two regressors marginally lower. This score represents the proportion of predictions the model got right.

This accuracy score however does not paint a complete picture due to the class imbalance in the target variable, highlighted in Figure 5. For instance, the model can achieve 82% accuracy on the train data just by predicting all as negative class.

It is useful to visualise model performance of predicting the positive and negative classes using confusion matrices in Figure 7, which immediately point out to the differences in performance in predicting the positive classes. KNN performs significantly worse than the other methods due to the higher number of false negatives.

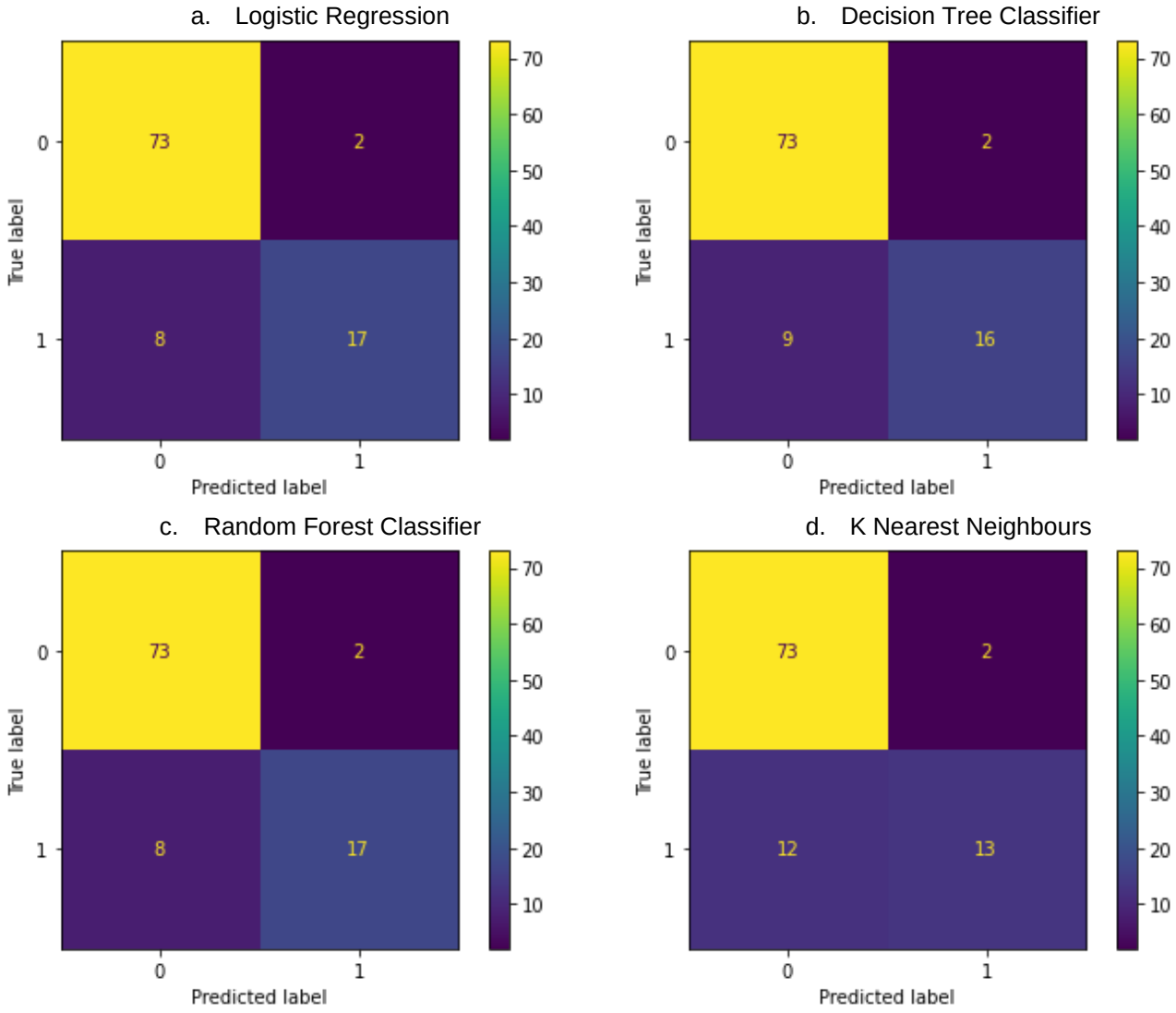


Figure 7: Confusion matrices from the classification methods

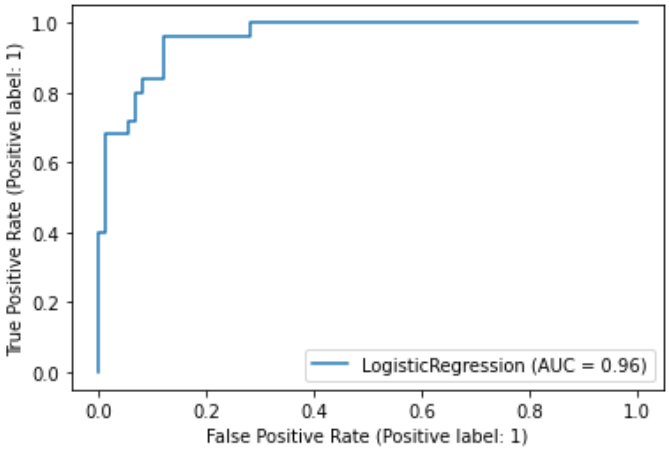
The F_1 scores and ROC AUC show that logistic regression and random forest are the best performing classifiers.

Logistic regression performs well due to the effectiveness of the sigmoid function in handling outliers. Multi-dimensionality can cause overfitting in logistic regression, however due to the limited dimensionality of this problem, the effect is not highly pronounced.

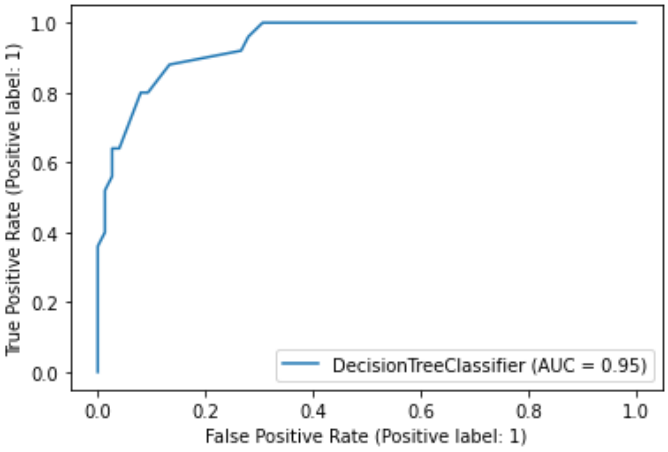
Random forest classifier performs well due to the use of multiple trees to vote to control overfitting. This method is also robust to outliers. In comparison, decision tree performs slightly worse due to higher tendency to overfit compared to random forest.

KNN performs the worst due to its high sensitivity to noise and outliers. The model can be improved by removing severe outliers.

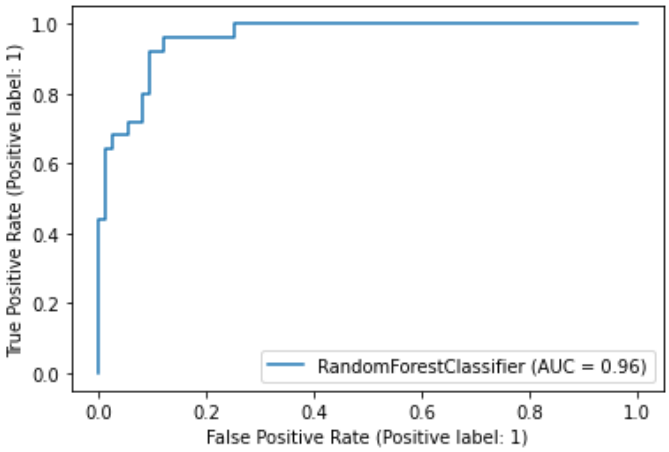
a. Logistic Regression



b. Decision Tree Classifier



c. Random Forest Classifier



d. K Nearest Neighbours

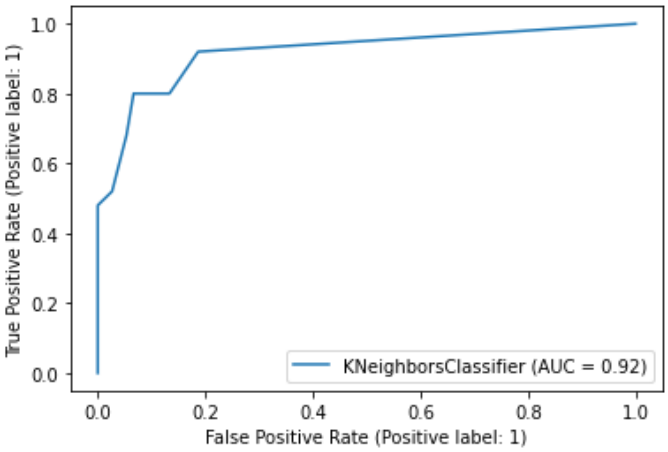


Figure 8: ROC AUC for the classification methods

7. Conclusions and Further Work

Various regression methods were trialled and optimised to predict the TTF. Random forest regression came out as the best performer with a test RMSE of 27.59, highlighting the strengths of ensemble methods.

Random forest and logistic regression came out as the best classifiers with F_1 score of 0.77. The models have better precision than recall, meaning it predicts more false negatives than false positives. This could be improved to enable the company make timely decisions in a safety critical industry.

Both the regression and classification models can be further improved by considering:

1. Performing dimensionality reduction by either studying the feature importance or by carrying out Principal Component Analysis (PCA)

2. Studying outliers and understanding if they could repeat in normal operation. If this is not the case, they can be removed.

8. References

[1] IBM, “What is Exploratory Data Analysis?,” [Online]. Available: https://www.ibm.com/uk-en/cloud/learn/exploratory-data-analysis. [Accessed 19 November 2021].

[2] “seaborn.pairplot,” [Online]. Available: https://seaborn.pydata.org/generated/seaborn.pairplot.html. [Accessed 19 November 2021].

[3] G. James, D. Witten, T. Hastie and R. Tibshirani, An Introduction to Statistical Learning with Applications in R, Springer, 2013.

[4] McGill University, “Goodness of Fit in Linear Regression,” [Online]. Available: http://www.medicine.mcgill.ca/epidemiology/joseph/courses/EPIB-621/fit.pdf. [Accessed 21 November 2021].

Appendices

Appendix A – Definition of Metrics

Precision (P) is the ratio of the number of true positives (TP) to the sum of true positives (TP) and false positive (sum represents the number of samples the model predicts as positive). A system with high precision has a low false positive rate.

$$P = \frac{TP}{TP + FP}$$

Recall (R) is the ratio of the number of true positives (TP) to the sum of true positives (TP) and false negative (sum represents the number of actual positive samples). A system with high recall has a low false negative rate.

$$R = \frac{TP}{TP + FN}$$

F_1 score is the harmonic mean of the precision and recall. A perfect model has a F1 score of 1.

$$F_1 = 2\ X \frac{P\ X\ R}{P + R}$$

Appendix B – Python Code

Python code used to support the results in the report is saved as a Google Colab file in the link below:

https://colab.research.google.com/drive/1JkJezTWJfY1hRG_DIRcV41_uByte7eD_?usp=sharing