**DWDM Submission Template**

|  |  |
| --- | --- |
| All Roll Numbers of your Team: | 18BCD7041, 18BCD7048 |
| Student names in your team : | GOKUL R, BELLAMKONDA SAI VINAY |
| Contribution by each team member in percentage to complete this project: | 45-55 |
| Slot (L1/L2/L4): | L2 |
| **Title of the Project:** | APS Failure at Scania Trucks |
| **Objective of the Project (What exactly the project is about?)** | To predict whether a truck has APS failure or not and to minimize the cost of failures. |

|  |  |  |
| --- | --- | --- |
| **Dataset Link** | **Number of rows and Columns** | **About columns** |
| **https://archive.ics.uci.edu/ml/datasets/APS+Failure+at+Scania+Trucks** | **60000 \* 171** | **Number of Categorical columns: 1**  **Number of Integer/Float Columns: 170**  **Number of Pure String Columns: 1** |
|  |  | **(Avg.) Unique Values in each Column:**  **17133.78** |

|  |  |  |
| --- | --- | --- |
| **Challenges identified in the project** | **How did you address that challenge?** | **References** |
| 1) **Missing Values** | Forward and backward fill imputation, imputation by mean,median,mode | https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html |
| 2) **Class Imbalance** | SMOTE - Oversampling | https://github.com/scikit-learn-contrib/imbalanced-learn |
| 3) **Data Scalability** | Dimensional Reduction, Normalization | <https://www.geeksforgeeks.org/data-reduction-in-data-mining/>  https://www.geeksforgeeks.org/data-normalization-in-data-mining/ |
| **4) Skewness** | Skew function - Pandas library | https://pythontic.com/pandas/dataframe-computations/skew |
|  |  |  |

|  |  |  |
| --- | --- | --- |
| **Without Pre-processing- Different Algorithms** | **Performance(Accuracy/other confusion matrix measures)** | **Which model worked well on the test data and WHY?** |
| **Algorithm 1- (Logistic Regression)** | Train Accuracy – 0.9848  Train Confusion Matrix:  [[512 3]  [ 6 70]]  ROC AUC: 0.9576  Mean Absolute Error: 0.0152  Test Accuracy – 0.9636  Confusion Matrix:  [[133 2]  [ 4 26]]  ROC AUC: 0.9259  Mean Absolute Error: 0.0363  Total cost: 1550 | Logistic Regression performed “well” enough on the training and test data, when compared to other models.  Training accuracy was 0.9848, while the test accuracy was 0.9636. Case of Overfitting.  However, this was the model that produced the least cost, with 1.5k.  However, This cost is NOT the best and accurate cost we can find, because we have not applied any preprocessing on the dataframe. |
| **Algorithm 2-(K-Nearest Neighbours)** | Train Accuracy – 0.94  Train Confusion Matrix:  [[509 6]  [ 26 50]]  ROC AUC: 0.8231  Test Accuracy – 0.85  Confusion Matrix:  [[128 7]  [ 17 13]]  ROC AUC: 0.6907  Total cost: 9060 |
| **Algorithm 3-(Random Forests)** | Train Accuracy – 0.952  Confusion Matrix:  [[514 1]  [ 27 49]]  ROC AUC: 0.8214  Test Accuracy – 0.9212  Confusion Matrix:  [[134 1]  [ 12 18]]  ROC AUC: 0.796  Total cost: 6010 |
| **Algorithm 4-(SVM)** | Train Accuracy – 0.8714  Confusion Matrix:  [[515 0]  [ 76 0]]  ROC AUC: 0.5000  Test Accuracy – 0.8182    Confusion Matrix:  [[135 0]  [ 30 0]]  ROC AUC: 0.5000 |
| **Algorithm 5-(Gaussian Naives Bayes)** | Train Accuracy – 0.9086  Confusion Matrix:  [[474 41]  [ 13 63]]  ROC AUC: 0.8747  Test Accuracy –0.9030    Confusion Matrix:  [[124 11]  [ 5 25]]  ROC AUC: 0.8759  Total cost: 2610 |
| **Algorithm 6-(Decision tree)** | Train Accuracy – 1.0000  Confusion Matrix:  [[515 0]  [ 0 76]]  ROC AUC: 1.0000  Test Accuracy –0.8909    Confusion Matrix:  [[128 7]  [ 11 19]]  ROC AUC: 0.7907  Total cost: 6050 |

|  |  |  |
| --- | --- | --- |
| **Which Pre-processing technique you applied?** | **Why did you apply that pre-processing Technique?** | **References** |
| Forward and Backward fill **- Imputation**  Imputation by mean,median,mode **- Imputation** | To deal with missing values in the dataframe, will propagate last valid observation forward and backward.  To deal with missing values in the dataframe by taking the mean,median and mode. | <https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.fillna.html>  https://medium.com/@ODSC/data-imputation-beyond-mean-median-and-mode-6c798f3212e3 |
| Outlier Detection **- Skewness** | To deal with "outliers",  Since they are strongly associated with skewness, and outliers tend to remain outliers in the residuals, making residuals skewed. | https://medium.com/@atanudan/kurtosis-skew-function-in-pandas-aa63d72e20de#:~:text=skewness()%20function%20in%20pandas,present%20in%20the%20DataFrame%20object. |
| SMOTE **- OverSampling** | To deal with high class imbalance, by randomly increasing minority class examples by replicating them | https://imbalanced-learn.readthedocs.io/en/stable/generated/imblearn.over\_sampling.SMOTE.html |
| Min-Max Scaler **- Normalization**  Standard Scaler **- Normalization** | To change enormous numeric values to a common scale. | https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.MinMaxScaler.html |

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Pre-processing technique name?** | **Data Mining Algorithms**  **you applied?** | **Performance(Accuracy/other confusion matrix measures)**  **(Before pre-processing) (After Pre-processing)** | | **Which model worked well on the test data and WHY?** |
| **Forward fill imputation** | **Logistic Regression;**  **KNN;**  **Random Forests;**  **SVM;**  **Naives Bayes;**  **Decision tree;** | **Logistic Regression**  Training  Accuracy - 0.9848  Test  Accuracy - 0.9636  Test Mean Absolute Error: 0.0363  **KNN**  Train  Accuracy score: 0.94  Confusion Matrix:  [[509 6]  [ 26 50]]  ROC AUC: 0.8231  Test  Accuracy score  0.85  Confusion Matrix:  [[128 7]  [ 17 13]]  ROC AUC: 0.6907  **Random Forest**  Train  Accuracy score:  0.952  Confusion Matrix:  [[514 1]  [ 27 49]]  ROC AUC: 0.8214  Test  Accuracy score: 0.9212  Confusion Matrix:  [[134 1]  [ 12 18]]  ROC AUC: 0.796  **SVM**  Train  accuracy score: 0.871  Confusion Matrix:  [[515 0]  [ 76 0]]  ROC AUC: 0.5000  Test  Accuracy score: 0.8182    Confusion Matrix:  [[135 0]  [ 30 0]]  ROC AUC: 0.5000  **Naives Bayes**  Train  Accuracy score: 0.9086  Confusion Matrix:  [[474 41]  [ 13 63]]  ROC AUC: 0.8747  Test  Accuracy score: 0.9030    Confusion Matrix:  [[124 11]  [ 5 25]]  ROC AUC: 0.8759  **Decision Tree**  Train  Accuracy score: 1.0000  Confusion Matrix:  [[515 0]  [ 0 76]]  ROC AUC: 1.0000  Test  Accuracy score: 0.8909    Confusion Matrix:  [[128 7]  [ 11 19]]  ROC AUC: 0.7907 | **Logistic Regression**  Training  Accuracy - 0.9889  Test  Accuracy - 0.9854  Test Mean Absolute Error: 0.0138  **KNN**  Train  Accuracy score: 0.991  Confusion Matrix:  [[58873 115]  [ 425 574]]  ROC AUC: 0.7863  Test  Accuracy score: 0.9834  Confusion Matrix:  [[15562 51]  [ 214 161]]  ROC AUC: 0.7130  **Random Forest**  Train  accuracy score: 0.9859  Confusion Matrix:  [[58970 18]  [ 828 171]]  ROC AUC: 0.5854  Test  accuracy score: 0.9806  Confusion Matrix:  [[15612 1]  [ 309 66]]  ROC AUC: 0.5880  **SVM**  Train  Accuracy score: 0.983  Confusion Matrix:  [[58986 2]  [ 971 28]]  ROC AUC: 0.5140  Test  accuracy score: 0.9838  Confusion Matrix:  [[58986 2]  [ 971 28]]  ROC AUC: 0.5140  Best threshold: 0.0232  Min cost: 6880.00  Train Cost: 39550  Test Cost: 17720  **Naives Bayes**  Train  accuracy score: 0.9663  Confusion Matrix:  [[57096 1892]  [ 129 870]]  ROC AUC: 0.9194  Test  accuracy score: 0.9668  Confusion Matrix:  [[15120 493]  [ 38 337]]  ROC AUC: 0.9335  **Decision Tree**  Train  accuracy score: 1.0000  Confusion Matrix:  [[58988 0]  [ 0 999]]  ROC AUC: 1.0000  Test  accuracy score: 0.9877  Confusion Matrix:  [[15537 76]  [ 121 254]]  ROC AUC: 0.8362 | The XGBoost model performed well (Total cost) with the minimized median, after imputation with mode, oversampling and normalization.  . |
| **Backward fill imputation** | **Logistic Regression;**  **KNN;**  **Random Forests;**  **SVM;**  **Naives Bayes;** | **Logistic Regression**  Training  Accuracy - 0.9848  Test  Accuracy - 0.9636  Test Mean Absolute Error: 0.0363  **Random Forest**  Train  Accuracy score:  0.952  Confusion Matrix:  [[514 1]  [ 27 49]]  ROC AUC: 0.8214  Test  Accuracy score: 0.9212  Confusion Matrix:  [[134 1]  [ 12 18]]  ROC AUC: 0.796  **SVM**  Train  accuracy score: 0.871  Confusion Matrix:  [[515 0]  [ 76 0]]  ROC AUC: 0.5000  Test  Accuracy score: 0.8182    Confusion Matrix:  [[135 0]  [ 30 0]]  ROC AUC: 0.5000  **Naives Bayes**  Train  Accuracy score: 0.9086  Confusion Matrix:  [[474 41]  [ 13 63]]  ROC AUC: 0.8747  Test  Accuracy score: 0.9030    Confusion Matrix:  [[124 11]  [ 5 25]]  ROC AUC: 0.8759 | **Logistic Regression**  Training  Accuracy - 0.9890  Test  Accuracy - 0.9865  Test Mean Absolute Error: 0.0135  **Random Forest**  Train  Accuracy score:  0.9860  Confusion Matrix:  [[58976 20]  [ 822 178]]  ROC AUC: 0.5888  Test  Accuracy score: 0.9807  Confusion Matrix:  [[15619 1]  [ 307 68]]  ROC AUC: 0.5906  **SVM**  Train  accuracy score: 0.983  Confusion Matrix:  [[58995 1]  [ 972 28]]  ROC AUC: 0.5140  Test  Accuracy score: 0.9838    Confusion Matrix:  [[58995 1]  [ 972 28]]  ROC AUC: 0.5140  **Naives Bayes**  Train  Accuracy score: 0.9666  Confusion Matrix:  [[57122 1874]  [ 127 873]]  ROC AUC: 0.9206  Test  Accuracy score: 0.9677    Confusion Matrix:  [[15141 479]  [ 38 337]]  ROC AUC: 0.9340 |
| **Forward fill imputation + OverSampling** | **Logistic Regression;**  **Random Forests;**  **Naives Bayes;**  **Bagging;**  **Ada boost;** | **Logistic Regression**  Training  Accuracy - 0.9848  Test  Accuracy - 0.9636  Test Mean Absolute Error: 0.0363  **Random Forest**  Train  Accuracy score:  0.952  Confusion Matrix:  [[514 1]  [ 27 49]]  ROC AUC: 0.8214  Test  Accuracy score: 0.9212  Confusion Matrix:  [[134 1]  [ 12 18]]  ROC AUC: 0.796  **Naives Bayes**  Train  Accuracy score: 0.9086  Confusion Matrix:  [[474 41]  [ 13 63]]  ROC AUC: 0.8747  Test  Accuracy score: 0.9030    Confusion Matrix:  [[124 11]  [ 5 25]]  ROC AUC: 0.8759  **Bagging**  Train  Accuracy score: 0.9663  Confusion Matrix:  [[78 0]  [ 3 8]]  ROC AUC: 0.8636  **Ada Boost**  Validation  accuracy score:  1.0000  Confusion Matrix:  [[78 0]  [ 0 11]]  ROC AUC: 1.0000 | **Logistic Regression**  Training  Accuracy - 0.7194  Test  Accuracy - 0.07429  Test Mean Absolute Error: 0.2314  **Random Forest**  Train  Accuracy score:  0.9463  Confusion Matrix:  [[55296 3692]  [ 2639 56349]]  ROC AUC: 0.9463  Test  Accuracy score: 0.9585  Confusion Matrix:  [[14669 944]  [ 351 15262]]  ROC AUC: 0.9585  **Naives Bayes**  Train  Accuracy score: 0.9181  Confusion Matrix:  [[56997 1991]  [ 7673 51315]]  ROC AUC: 0.9181  Test  Accuracy score: 0.9319  Confusion Matrix:  [[15090 523]  [ 1602 14011]]  ROC AUC: 0.9319  **Bagging**  Train  Accuracy score: 0.9438  Confusion Matrix:  [[18789 1234]  [ 1040 19368]]  ROC AUC: 0.9437  **Ada Boost**  Validation  accuracy score:  0.9923  Confusion Matrix:  [[3610 27]  [ 28 3471]]  ROC AUC: 0.9923 |
| **Forward fill imputation + OverSampling + Normalization** | **Logistic Regression;**  **Random Forests;**  **Naives Bayes;** | **Logistic Regression**  Training  Accuracy - 0.9848  Test  Accuracy - 0.9636  Test Mean Absolute Error: 0.0363  **Random Forest**  Train  Accuracy score:  0.952  Confusion Matrix:  [[514 1]  [ 27 49]]  ROC AUC: 0.8214  Test  Accuracy score: 0.9212  Confusion Matrix:  [[134 1]  [ 12 18]]  ROC AUC: 0.796  **Naives Bayes**  Train  Accuracy score: 0.9086  Confusion Matrix:  [[474 41]  [ 13 63]]  ROC AUC: 0.8747  Test  Accuracy score: 0.9030    Confusion Matrix:  [[124 11]  [ 5 25]]  ROC AUC: 0.8759 | **Logistic Regression**  Training  Accuracy - 0.9568  Test  Accuracy - 0.9628  Test Mean Absolute Error: 0.0363  **Random Forest**  Train  Accuracy score:  0.9472  Confusion Matrix:  [[55331 3657]  [ 2567 56421]]  ROC AUC: 0.9472  Test  Accuracy score: 0.9598  Confusion Matrix:  [[14681 932]  [ 322 15291]]  ROC AUC: 0.9598  **Naives Bayes**  Train  Accuracy score: 0.9228  Confusion Matrix:  [[56805 2183]  [ 6923 52065]]  ROC AUC: 0.9228  Test  Accuracy score: 0.9345    Confusion Matrix:  [[15015 598]  [ 1446 14167]]  ROC AUC: 0.9345 |
| **Imputation by Mean+Standard Scaling** | **Logistic Regression;**  **SVM;**  **Random Forest;**  **XG Boost** | **Logistic Regression**  Train cost: 0  Test cost:1550  **SVM**  Train cost: 38000  Test cost:15000  **Random Forest**  Train cost: 13510  Test cost:6010 | **Logistic Regression**  Train cost: 32150  Test cost:15060  **SVM**  Train cost: 39550  Test cost:17720  **Random Forest**  Train cost: 44390  Test cost:13250  **XG Boost**  Train cost: 18600  Test cost:12380 |
| **Imputation by Mode+Standard Scaling** | **Logistic Regression;**  **SVM;**  **Random Forest;**  **XG Boost** | **Logistic Regression**  Train cost: 0  Test cost:1550  **SVM**  Train cost: 38000  Test cost:15000  **Random Forest**  Train cost: 13510  Test cost:6010 | **Logistic Regression**  Train cost: 32440  Test cost:15800    **SVM**  Train cost: 42660  Test cost:18290  **Random Forest**  Train cost: 44110  Test cost:13770  **XG Boost**  Train cost: 16810 CV cost: 5260  Test cost:10180 |
| **Imputation by Mean+OverSampling** | **Random Forest;**  **XG Boost** | **Random Forest**  Train cost: 13510  Test cost:6010 | **Random Forest**  Train Cost: 11420  Test Cost: 15320  **XG Boost**  Train Cost: 15850  Test Cost: 10170 |  |
| **Imputation by Median +**  **OverSampling** | **Random Forest;**  **XG Boost** | **Random Forest**  Train cost: 13510  Test cost:6010 | **Random Forest**  Train Cost: 6960  Test Cost: 13360  **XG Boost**  Train Cost: 5320  Test Cost: 18650 |  |
| **Imputation by Mode**  **+OverSampling** | **Random Forest;**  **XG Boost** | **Random Forest**  Train cost: 13510  Test cost:6010 | **Random Forest**  Train Cost: 11140  Test Cost: 10240  **XG Boost**  Train Cost: 7820  Test Cost: 20650 |  |

|  |
| --- |
| **Summary:** |
| **Number of Pre-processing Techniques applied with their names: 9**  **Forward fill imputation, Backward fill imputation, Normalization, OverSampling, Column reduction (based on NA percentage), Imputation by mean,median and mode, Skewness**  **Number of Data Mining Algorithms applied with their names: 10**  **Logistic Regression; KNN; Random Forests; SVM; Naives Bayes; Decision tree;Bagging; Ada Boost, XGBoost; Grid Search** |
| **Which algorithm showed highest performance after “All” pre-processing techniques and WHY?:**  **In terms of the cost, the best algorithm we found for this data is XGBoost algorithm with a minimum cost of 10170.**  **XGBoost library uses the approach where new models are created that predict the residuals or errors of prior models, and then added together to make the final prediction, hence why its preferred and is efficient.** |
| **Conclusion-Write in your own words:**  **Working with a massive dataset with completely anonymized column names is quite a difficult task in terms of understanding the data w.r.t the target variable. We faced high time complexity issues when trying to fit the model even after preprocessing.**  **Out of all models, XGBoost algorithm minimized the cost much lesser than the rest of the algorithms.** |