Data Mining Assignment

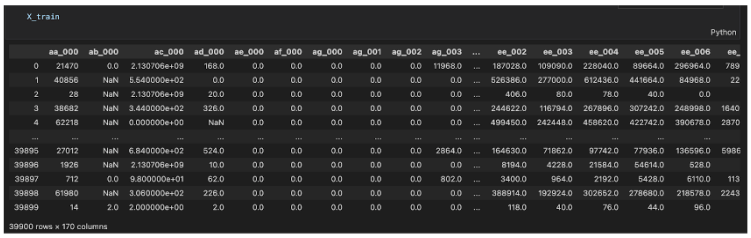
Hyeon Yu (MB3VKJ)

https://www.kaggle.com/yhyeon

1. Introduction

The objective of this assignment is to gain experience with a number of exploratory data analysis techniques, learning, pre-processing, learning and tuning algorithms we have learned about in class to predict whether the APS system has failure and should be repaired, using the ”APS Failure at Scania Trucks”. The assignment is organized on Kaggle to make predictions for the second dataset (X test.csv). The evaluation of the result uses the F3 score. F3 is obtained using TP/(TP + 0.1FP + 0.9FN) putting more weight on the false negative prediction.

2. Dataset



X_train

Python

	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	ag_003	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_
0	21470	0.0	2.130706e+09	168.0	0.0	0.0	0.0	0.0	0.0	11968.0	...	187028.0	109090.0	228040.0	89664.0	296964.0	789
1	40856	NaN	5.640000e+02	0.0	0.0	0.0	0.0	0.0	0.0	0.0	...	526386.0	277000.0	612436.0	441664.0	84968.0	22
2	28	NaN	2.130706e+09	20.0	0.0	0.0	0.0	0.0	0.0	0.0	...	406.0	80.0	78.0	40.0	0.0	
3	38682	NaN	3.440000e+02	326.0	0.0	0.0	0.0	0.0	0.0	0.0	...	244622.0	116794.0	267896.0	307242.0	248998.0	1640
4	62218	NaN	0.000000e+00	NaN	0.0	0.0	0.0	0.0	0.0	0.0	...	499450.0	242448.0	458620.0	422742.0	390678.0	2870
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...
39895	27012	NaN	6.840000e+02	524.0	0.0	0.0	0.0	0.0	0.0	2864.0	...	164630.0	71862.0	97742.0	77936.0	136596.0	5986
39896	1926	NaN	2.130706e+09	10.0	0.0	0.0	0.0	0.0	0.0	0.0	...	8194.0	4228.0	21584.0	54614.0	528.0	
39897	712	0.0	9.800000e+01	62.0	0.0	0.0	0.0	0.0	0.0	802.0	...	3400.0	964.0	2192.0	5428.0	6110.0	113
39898	61980	NaN	3.060000e+02	226.0	0.0	0.0	0.0	0.0	0.0	0.0	...	388914.0	192924.0	302652.0	278680.0	218578.0	2243
39899	14	2.0	2.000000e+00	2.0	0.0	0.0	0.0	0.0	0.0	0.0	...	118.0	40.0	76.0	44.0	96.0	

39900 rows × 170 columns

Figure 1. X_train dataset

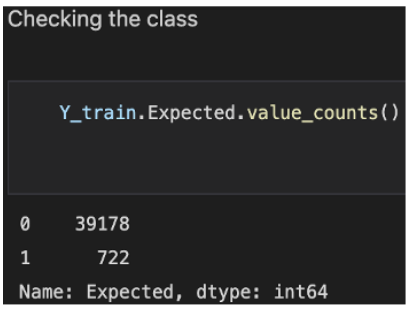


Figure 2. y_train dataset

The dataset consists of data collected from heavy Scania trucks in everyday usage focusing on the Air Pressure System (APS). The APS generates pressurized air that is utilized in various functions in a truck. The dataset’s positive class consists of component failures for a specific component of the APS, while the negative class consists of trucks with failures for components not related to the APS.

The training dataset contains 39,900 instances with 170 attributes. The attribute names of the data have been anonymized for proprietary reasons and consist of single numerical counters and histograms consisting of bins with different conditions. The explanation of the histogram attributes can be further found in the assignment description. The test data has not been checked to prevent data leakage.

2.1 Exploratory Data Analysis

```
X_train.isna().sum()

aa_000        0
ab_000    30779
ac_000     2241
ad_000     9966
ae_000     1660
          ...
ee_007      468
ee_008      468
ee_009      468
ef_000     1813
eg_000     1812
Length: 170, dtype: int64
```

Figure 3. Attributes missing values Counts

	Predictors	NaN
0	br_000	82.012531
1	bq_000	81.120301
2	bp_000	79.493734
3	ab_000	77.140351
4	cr_000	77.140351

Figure 4. Attributes missing values percentages

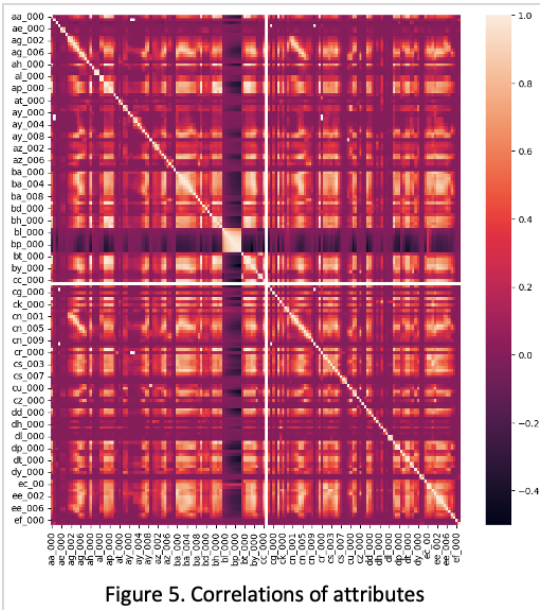


Figure 5. Correlations of attributes

Here are the list of observation made of the performance of EDA

- The training set consists of 170 attributes after removing the id column
- The dataset is highly imbalance (figure 2)
- The missing values are encoded as NaN and there are attributes that contains up to 82% of missing values
- The correlation between the attributes can be seen on figure 5

2.2 Pre-Processing

The training dataset has been split into a training set and a validation set with the validation set having a weight of 25%. 25% seemed to be a good fit with such a limited sample. The X_train dataset has been pre processed in the order of, dropping the “Id” column, removal of attributes with greater than 30% NaN percentage, scaling with StandardScaler(), imputation using mode, and PCA with number of components that can explain the variance of 99%. The validation and test dataset has been fitted to follow pre-processing done on the training dataset and used the same values, attributes of the training dataset to take the data leakage into account.

In addition, the skewness of the dataset has been handled by oversampling using SMOTE. SMOTE creates synthetic samples for the minority class by interpolating between positive instances that lie together in the feature space. Only the training set has been oversampled to take into account that the test set can also be biased and an oversampled validation set will not give a good estimation of the performance of the model.

3. Learning & Tuning Algorithm

Various models have been implemented and tuned using GridSearchCV, when applicable. For each algorithm tried, the confusion matrix, F3 score of the validation set, classification report, and if GridSearchCV have been used, the parameters and the best estimator is shown. The algorithms tried have been discussed in lectures, so the general description is not included.

GridSearchCV has been performed with the cross validation of 5 and the recall scoring has been used. Even though grid search tries all combinations and is exhaustive, it was chosen to find the optimal solution. Cross validation of 5 has been chosen instead of 10 to consider the limited data size and the resource constraint I had. Recall is the proportion of true positives to all positives (same as sensitivity, which measures the proportion of positives that are correctly identified). Recall focuses on the false negative (Type 2 error). Since the dataset was biased toward 0, I decided to use recall as the scoring method to put more weight onto predicting 1s as 1s.

The best performing model was the Random Forest. To try to further improve the performance of the model, different combinations of pre-processing of the dataset have been tried. 3.4.1 will go in detail about the different pre-processing of the dataset used.

3.1 Clustering

3.1.1 K Means

```
K Means
Confusion Matrix
[[9692  103]
 [  75  105]]
Validation F3 Score: 0.5743982494529541
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      9795
           1       0.50      0.58      0.54       180

    accuracy                           0.98      9975
   macro avg       0.75      0.79      0.77      9975
weighted avg       0.98      0.98      0.98      9975
```

Only the K Means algorithm was tested for clustering and tuning was not done since this is a binary clustering and the number of K had to be given.

3.2 Classification

3.2.1 Decision Tree Classifier

```
params = [{'criterion': ['gini', 'entropy'],
           'max_depth': range(1,20,1)}]
```

```
Decision Tree Classifier
Confusion Matrix
[[9580  215]
 [  41  139]]
Validation F3 Score: 0.7041540020263425
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9795
           1       0.39      0.77      0.52       180

    accuracy                           0.97      9975
   macro avg       0.69      0.88      0.75      9975
weighted avg       0.98      0.97      0.98      9975

Best Estimator: DecisionTreeClassifier(max_depth=16)
```

3.2.2 Gaussian Naive Bayes

```
params = {'var_smoothing': [0.0001, 0.001, 0.01, 0.1, 1]}
```

```
Gaussian Naive Bayes
Confusion Matrix
[[9440  355]
 [  19  161]]
Validation F3 Score: 0.75374531835206
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.96      0.98      9795
           1       0.31      0.89      0.46       180

    accuracy                           0.96      9975
   macro avg       0.66      0.93      0.72      9975
weighted avg       0.99      0.96      0.97      9975

Best Estimator: GaussianNB(var_smoothing=0.0001)
```

Only the Gaussian Naive Bayes have been tried. The assumption of Naive Bayes is that all of the attributes are statistically independent, however the histograms attributes will not be statistically independent.

3.2.3 Support Vector Machine

```
params = [{'C': [0.001, 0.01, 0.1],
           'gamma': [0.01, 0.05, 0.1, 0.2, 0.3, 0.4],
           'kernel': ['linear']}]
```

```
SVM
Confusion Matrix
[[9562  232]
 [  26  155]]
Validation F3 Score: 0.7688492063492065
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9794
           1       0.40      0.86      0.55       181

    accuracy                           0.97      9975
   macro avg       0.70      0.92      0.77      9975
weighted avg       0.99      0.97      0.98      9975

Best Estimator: SVC(C=0.1, gamma=0.01, kernel='linear')
```

Since this is a binary classification task and considering overfitting, only linear SVM has been tried.

3.2.4 K Nearest Neighbour

```
params = [{'n_neighbors':range(1,31,1)}]
```

```
KNN
Confusion Matrix
[[9671  123]
 [  38  143]]
Validation F3 Score: 0.7546174142480211
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9794
           1       0.54      0.79      0.64       181

    accuracy                           0.98      9975
   macro avg       0.77      0.89      0.82      9975
weighted avg       0.99      0.98      0.99      9975

Best Estimator: KNeighborsClassifier(n_neighbors=1)
```

The best estimator of the Instance based learning of KNN was surprising to me since the number of neighbors was 1.

3.3 Numerica Prediction

Even though regression algorithms are mostly used for numeric prediction, binary classification can be done by treating the line as a decision boundary.

3.3.1 Logistic Regression

```
params = [{'C': [0.001, 0.01, 0.1, 1, 10],
           'penalty': ["l2"]}]
```

```
Logistic Regression
Confusion Matrix
[[9596  199]
 [  20  160]]
Validation F3 Score: 0.8084891359272359
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9795
           1       0.45      0.89      0.59       180

    accuracy                           0.98      9975
   macro avg       0.72      0.93      0.79      9975
weighted avg       0.99      0.98      0.98      9975

Best Estimator: LogisticRegression(C=10, max_iter=1000000, random_state=42)
```

Surprisingly, the Logistic Regression model performed well on Kaggle. With the performance of the Logistic Regression, It seems like poly or rbf SVM would have performed better than the linear regression.

3.3.2 Linear Regression

GridSearchCV was not used for linear Regression but Linear Regression performed better by fitting the intercept.

```
Linear Regression Fit Intercept
Confusion Matrix
[[9594  200]
 [  41  140]]
Validation F3 Score: 0.7110208227526664
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9794
           1       0.41      0.77      0.54       181

    accuracy                           0.98      9975
   macro avg       0.70      0.88      0.76      9975
weighted avg       0.99      0.98      0.98      9975
```

3.3.3 Perceptron

```
params = [{'penalty': ['l2','l1'],
           'alpha': [0.0001, 0.001, 0.01, 0.1]}]
```

```
Decision Tree Classifier
Confusion Matrix
[[9580  215]
 [  41  139]]
Validation F3 Score: 0.7041540020263425
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9795
           1       0.39      0.77      0.52       180

    accuracy                           0.97      9975
   macro avg       0.69      0.88      0.75      9975
weighted avg       0.98      0.97      0.98      9975

Best Estimator: DecisionTreeClassifier(max_depth=16)
```

By using the Perceptron Learning Rule, I was hoping that the hyperplane would have been moved to a better location to separate the 1s and 0s but it did not.

3.4 Ensemble Learning

During the first version of the assignment, I hypothesized that the Ensemble Learning algorithms would perform greatly better than the other algorithms. Random Forest did give me the best result and further experiments have been done with different pre-processed dataset. However, Ensemble Learning algorithms did not perform as well as I would hope. This might be due to the parameters I have chosen to tune since the algorithms took a really long time to run and I could not try all the combinations I wanted, due to VScode's worker timing out.

3.4.1 Random Forest

```
params = [{'criterion': ['gini', 'entropy'],
           'max_features': ['sqrt'],
           'n_estimators': range(125,201,25),
           'max_depth': [10,20,30]}]
```

```
Random Forest
Confusion Matrix
[[9658  136]
 [  26  155]]
Validation F3 Score: 0.8072916666666667
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.99      0.99      9794
           1       0.53      0.86      0.66       181

    accuracy                           0.98      9975
   macro avg       0.76      0.92      0.82      9975
weighted avg       0.99      0.98      0.99      9975

Best Estimator: RandomForestClassifier(criterion='entropy', max_depth=20,
max_features='sqrt',
                       n_estimators=125, random_state=42)
```

Random Forest performed the best out of all of my models on Kaggle. To achieve further results, different combinations of pre-processing with NaN threshold of 20%, 30%, 40%, and 60%, PCA with 0.95 explained variance and 0.99 Variance, and with and without additional attributes of GaussianMixture. GaussianceMixture of each dataset has been tuned trying spherical, tied, diagonal, and full covariance type to cluster. The result can be seen on the table below with decimals rounded to second decimal points. The confusion matrix, F3 score, classification report, and the best estimator can be seen on the YU_MB3VKJ_Random_Forest.ipynb.

Addition of the cluster as an attribute did not improve the score as much as I wanted. However, I was not able to experiment further with this method due to lack of resources available. Maybe with better tuned or different clustering algorithms would have performed better.

NaN Threshold	PCA	Cluster Attributes	Csv File	F3 Score	Validation Accuracy	Kaggle Private	Kaggle Public
20%	0.95	Not Used	Random_Forest_20_95_MB3VKJ.csv	0.83	0.99	0.82	0.80
20%	0.95	Used	Random_Forest_Cluster_20_95_MB3VKJ.csv	0.83	0.99	0.80	0.82
20%	0.99	Not Used	Random_Forest_20_99_MB3VKJ.csv	0.88	0.98	0.84	0.83
20%	0.99	Used	Random_Forest_Cluster_20_99_MB3VKJ.csv	0.85	0.98	0.83	0.81
30%	0.95	Not Used	Random_Forest_30_95_MB3VKJ.csv	0.80	0.99	0.79	0.80
30%	0.95	Used	Random_Forest_Cluster_30_95_MB3VKJ.csv	0.83	0.98	0.81	0.83
30%	0.99	Not Used	Random_Forest_MB3VKJ.csv	0.81	0.98	0.82	0.84

30%	0.99	Used	Random_Forest_Cluster_30_99_MB3VKJ.csv	0.84	0.99	0.82	0.82
40%	0.95	Not Used	Random_Forest_40_95_MB3VKJ.csv	0.82	0.99	0.80	0.80
40%	0.95	Used	Random_Forest_Cluster_40_95_MB3VKJ.csv	0.84	0.99	0.82	0.82
40%	0.99	Not Used	Random_Forest_40_99_MB3VKJ.csv	0.88	0.99	0.81	0.84
40%	0.99	Used	Random_Forest_Cluster_40_99_MB3VKJ.csv	0.80	0.89	0.81	0.82
60%	0.95	Not Used	Random_Forest_60_95_MB3VKJ.csv	0.81	0.99	0.81	0.81
60%	0.95	Used	Random_Forest_Cluster_60_95_MB3VKJ.csv	0.85	0.98	0.82	0.83
60%	0.99	Not Used	Random_Forest_60_99_MB3VKJ.csv	0.87	0.99	0.82	0.84
60%	0.99	Used	Random_Forest_Cluster_60_99_MB3VKJ.csv	0.83	0.98	0.81	0.83

3.4.2 Gradient Boost

```
params = {'max_depth' : [1, 5, 10, 15],
          'n_estimators':[10, 20, 30, 40, 50]}
```

```
Gradient Boost
Confusion Matrix
[[9629  165]
 [  41  140]]
Validation F3 Score: 0.7238883143743536
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9794
           1       0.46      0.77      0.58       181

    accuracy                           0.98      9975
   macro avg       0.73      0.88      0.78      9975
weighted avg       0.99      0.98      0.98      9975

Best Estimator: GradientBoostingClassifier(max_depth=10, n_estimators=50)
```

Gradient Boost did not perform as well as I thought it would, however this might be due to the limited tuning I have done due to the time it takes for the tuning to be done.

3.4.3 AdaBoost

```
params = {'n_estimators' : range(200,500,50),
          'learning_rate':[0.001, 0.01, 0.1, 1.0]}
```

```
AdaBoost
Confusion Matrix
[[9636  158]
 [  33  148]]
Validation F3 Score: 0.764857881136951
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.98      0.99      9794
           1       0.48      0.82      0.61       181

    accuracy                           0.98      9975
   macro avg       0.74      0.90      0.80      9975
weighted avg       0.99      0.98      0.98      9975

Best Estimator: AdaBoostClassifier(n_estimators=450)
```

AdaBoost also did not perform as well as I thought it would, however this might be due to the limited tuning I have done due to the time it takes for the tuning to be done.

3.4.4 Bagging

```
params = {'n_estimators' : range(5,15,5),
          'max_samples' : [0.05, 0.1, 0.5, 1.0]}
```

```
Bagging LR
Confusion Matrix
[[9506  289]
 [  12  168]]
Validation F3 Score: 0.8088589311506982
Classification Report
              precision    recall  f1-score   support

           0       1.00      0.97      0.98      9795
           1       0.37      0.93      0.53       180

    accuracy                           0.97      9975
   macro avg       0.68      0.95      0.76      9975
weighted avg       0.99      0.97      0.98      9975

Best Estimator: BaggingClassifier(base_estimator=LogisticRegression(C=10, max_iter=1000000,
                                                    random_state=42),
                  max_samples=0.05)
```

Bagging was done using Logistic Regression with the best estimator parameters found in 3.3.1, which gave me a 0.80 private score. Surprisingly Bagging algorithm performed worse than the Logistic Regression algorithm. This might be due to the bootstrapping of the bias dataset.

3.4.5 Stacking

```
Stacking
Confusion Matrix
[[9718   77]
 [  49  131]]
Validation F3 Score: 0.7166301969365427
Classification Report
              precision    recall  f1-score   support

           0       0.99      0.99      0.99      9795
           1       0.63      0.73      0.68       180

    accuracy                           0.99      9975
   macro avg       0.81      0.86      0.83      9975
weighted avg       0.99      0.99      0.99      9975
```

Stacking has been implemented with the base learners as Random Forest Classifier, Decision Tree Classifier, and the SVM with the parameters found during tuning. Logistic Regression was chosen to be the meta learner. This also did not perform as well as I would hope. However, I decided against trying out different combinations of base learners and meta learners and focus on Random Forest instead.

4. Results

F3 Score has been calculated using sklearn.metrics.fbeta_score function in the score_calc function. Scores and accuracies are rounded to the second decimal point. Random Forest algorithm performed the best with the pre-processing mentioned in 2.2, followed by Logistic Regression. The performance of Random Forest was further improved using different pre-processing combination mentioned in 3.4.1.

Algorithm	**Csv File**	**F3 Score**	**Validation Accuracy**	**Kaggle Private**	**Kaggle Public**
K Means	K_means_MB3VKJ.csv	0.57	0.98	0.52	0.60
Decision Tree Classifier	Decision_Tree_Classifier_MB3VKJ.csv	0.70	0.97	0.66	0.70
Gaussian Naive Bayes	Gaussian_ Naive_Bayes_MB3VKJ.csv	0.75	0.96	0.72	0.70
SVM	SVM_MB3VKJ.csv	0.76	0.97	0.77	0.83
KNN	KNN_MB3VKJ.csv	0.71	0.98	0.73	0.70
Logistic Regression	Logistic_Regression_MB3VKJ.csv	0.81	0.98	0.80	0.82
Linear Regression	Linear_Regression_Fit_Intercept_ MB3VKJ.csv	0.71	0.98	0.72	0.73
Perceptron	Perceptron_MB3VKJ.csv	0.70	0.94	0.68	0.68
Random Forest	Random_Forest_MB3VKJ.csv	0.81	0.98	0.82	0.84

Gradient Boost	Gradient_Boost_MB3VKJ.csv	0.72	0.98	0.75	0.72
AdaBoost	AdaBoost_MB3VKJ.csv	0.76	0.98	0.74	0.75
Bagging	Bagging_LR.MB3VKJ.csv	0.81	0.97	0.78	0.82
Stacking	Stacking_MB3VKJ.csv	0.71	0.99	0.67	0.69

5. Conclusion and Experience

Table in 4. Results show the different algorithms tried and the performance of the algorithms using the dataset mentioned in 2.2 Pre-Processing. Random Forest (0.82) and Logistic Regression (0.80) performed the best and I was able to further increase the performance of Random Forest by 2% trying different combinations of pre-processing. I was surprised with the low performance of other Ensemble Learning algorithms and I have a strong belief that better tuning of the algorithms will lead to better scores.

From completion of the assignment, I gained great insight on techniques of exploratory data analysis techniques, how each algorithm performs differently regarding the usage of the same pre-processed dataset, and tuning parameters for different algorithms. I found it very challenging to find the middle ground between the possible parameter combinations and the computational resources available to me.