Introduction to DS

Term project:
«Minimizing Repair Costs for Scania Trucks by APS Failure predictions»

Prof. Mikhail Belyaev,
Prof. Maxim Panov

Skoltech

Kirill Shcherbakov

MSc 1-year Space Engineering Systems student

≈ $347500

for trucks which need to be serviced

1. Unnecessary checks done by a mechanic - $10

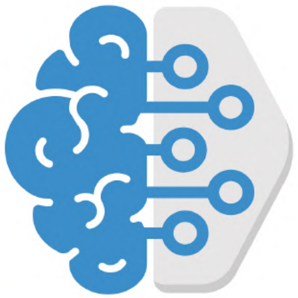
2. Missing a faulty truck, which may cause a breakdown in the future - $500

Lifecycle of trucks without ML model

A truck is working	Sensors reading	A truck is working	Breaking Phase	Breaking of a truck

Lifecycle of trucks with ML model

APS Failure prediction by our ML model



A truck is working

Sensors reading

Service maintenance

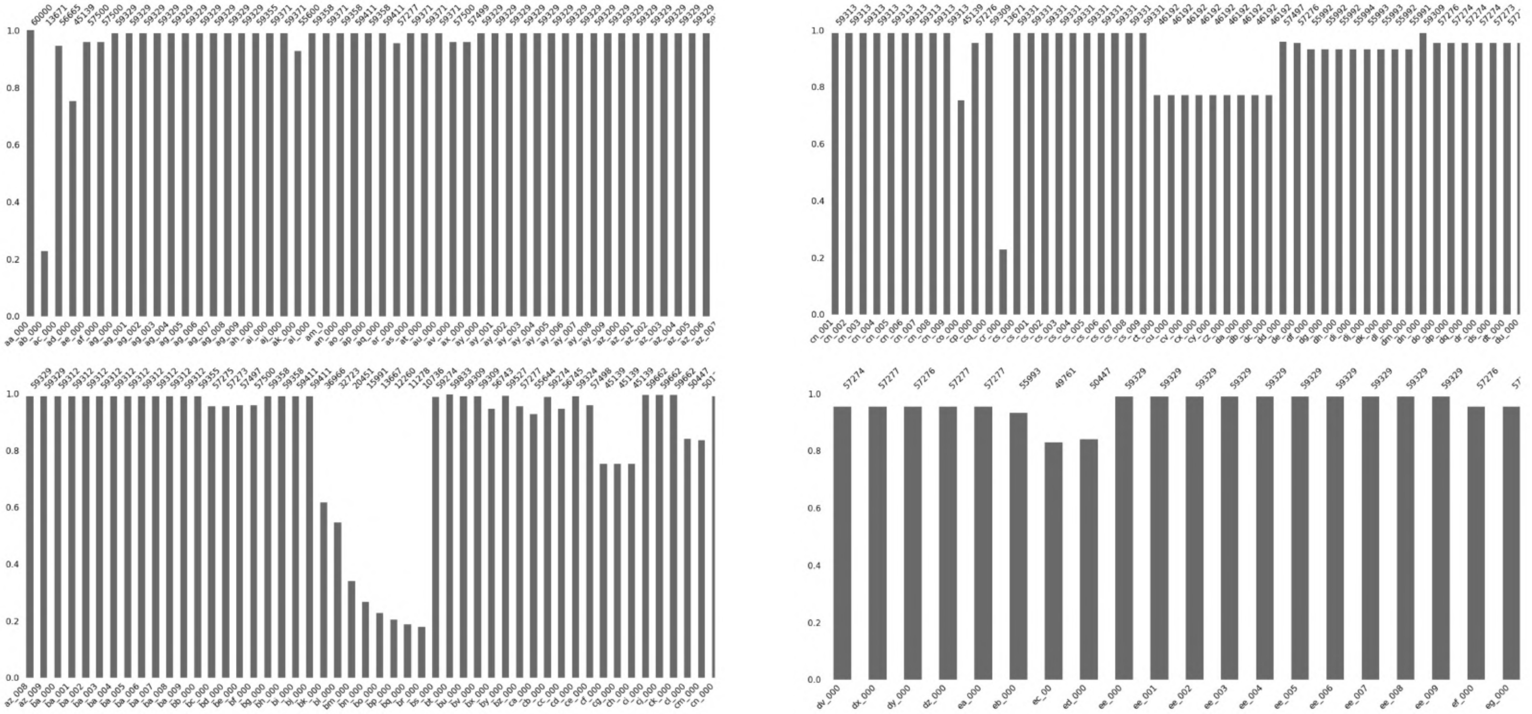
A truck is working

A truck is working

Reduce costs

Dataset preparation

Barchart of not missed values in each features



We decided to drop features which have missing values of more than 60%

Using median imputation

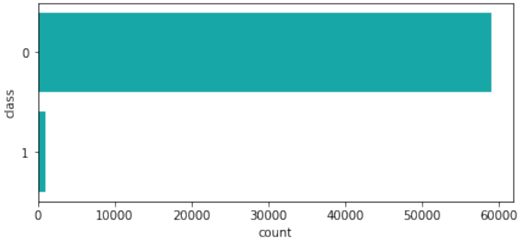
Source:

https://www.researchgate.net/publication/309195602_Prediction_of_Failures_in_the_Air_Pressure_System_of_Scania_Trucks_Using_a_Random_Forest_and_Feature_Engineering

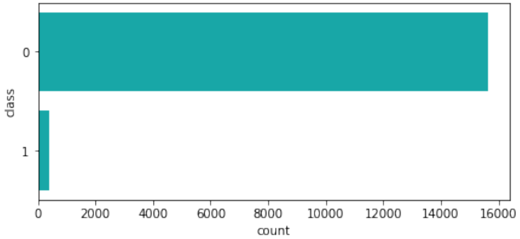
Exploratory analysis of the data

Distribution of class labels

Train dataset



Test dataset



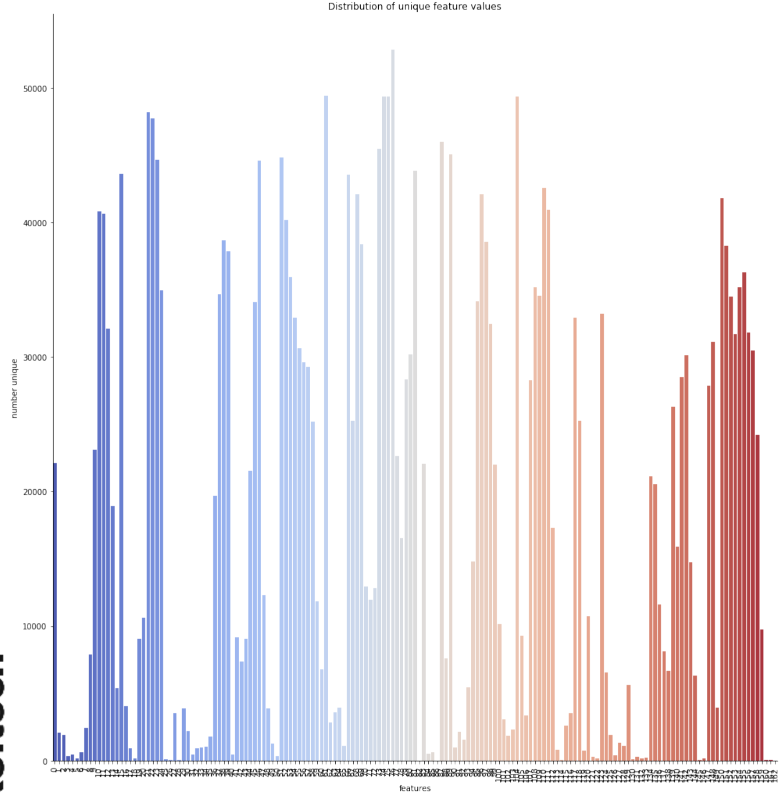
Using downsampling and upsampling to balance our dataset

Code of downsampling and upsampling

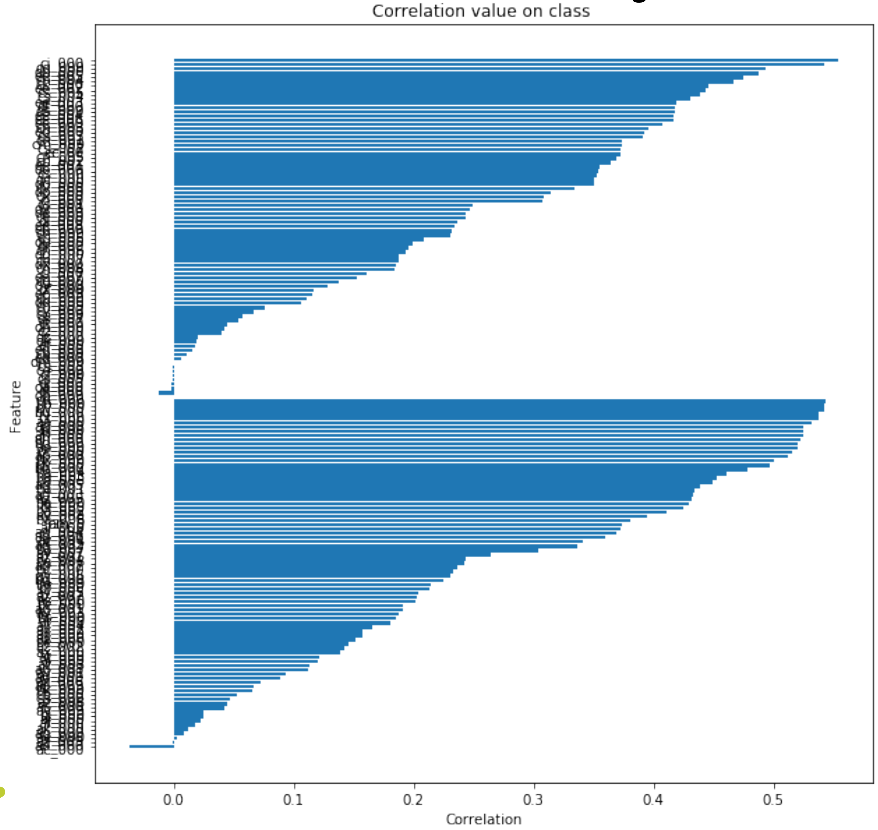
```
In [133]: train_impMedian['class'] = y_train

In [135]: #Undersampling the negative class

          train_neg_sampled = train_impMedian[train_impMedian['class'] == 0].sample(n = 10000,
                                                                                    random_state = 42)
          train_Sampled = train_impMedian[train_impMedian['class'] == 1].append(train_neg_sampled)

In [136]: print("Shape of the train data after under sampling the negative class", train_Sampled.shape[0])

          Shape of the train data after under sampling the negative class 11000

In [138]: y_train_Sampled = train_Sampled['class']
          train_Sampled.drop(['class'],axis = 1, inplace= True)

In [139]: # Upsampling the positive class using Smote Technique
          sm = over_sampling.SMOTE(ratio= 1.0)
          train_Sampled_Smote, y_train_Sampled = sm.fit_sample(train_Sampled,y_train_Sampled)

In [140]: print("Shape of train data after upsampling the positive class by smote", train_Sampled_Smote.sha

          Shape of train data after upsampling the positive class by smote (20000, 162)
```

Exploratory analysis of the data

Number of unique feature values



Correlation features on target

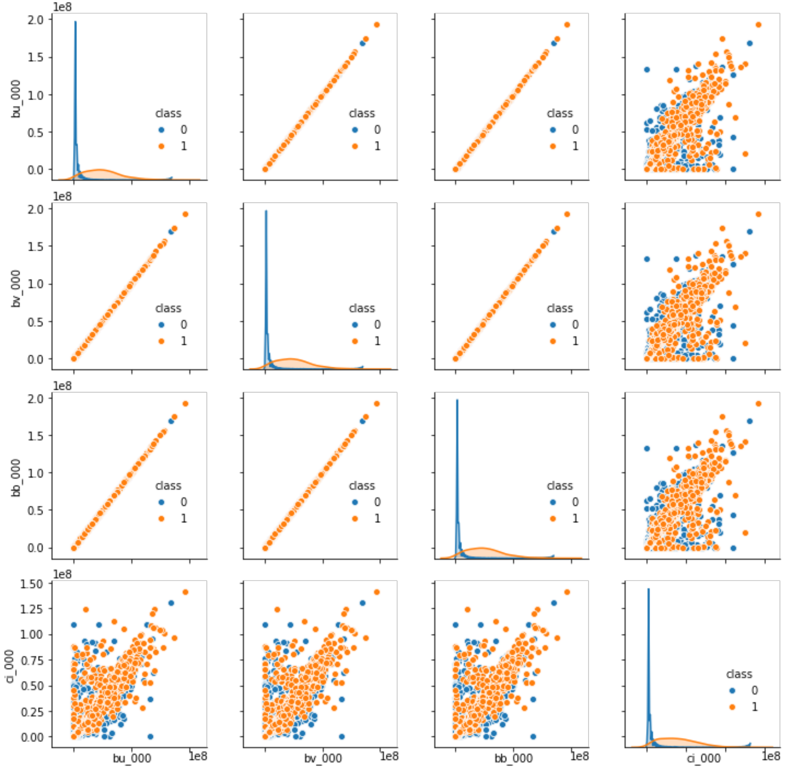


Skoltech

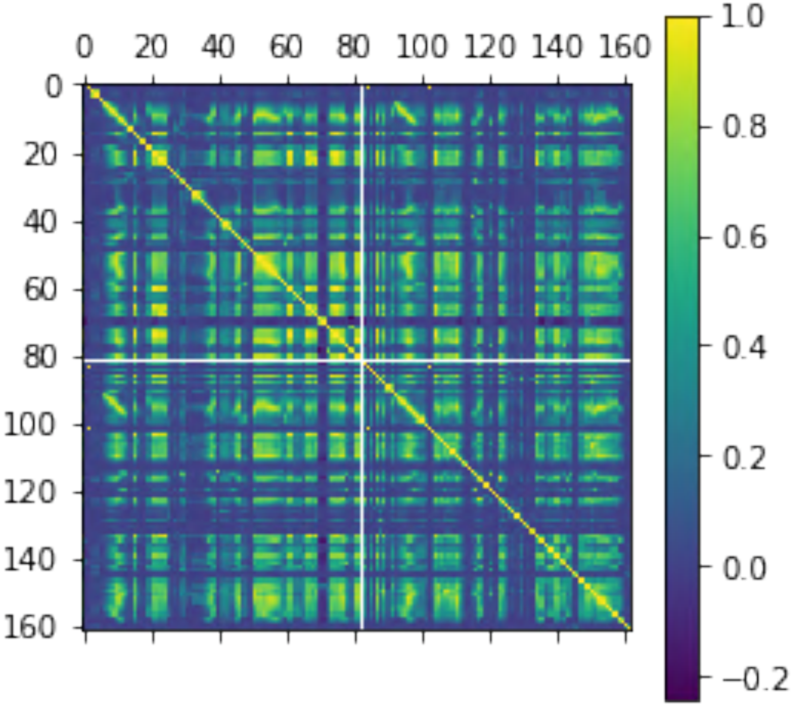
Our hypothesis 1 is the following: features with few unique values and little correlation with the target variable do not contain useful information and do not help classification

Exploratory analysis of the data

Pairplot with 4 most correlated features



Plot of correlation matrix



Skoltech

Our hypothesis 2 is the following: the features in the dataset are quite dependent on each other and their simultaneous presence is redundant.

Building ML models for hypotheses validation

Metrics

The Fβ score, which uses the parameter β to control the balance of recall and precision and is defined as

$$F_\beta = \frac{(1+\beta^2)(Precision \times Recall)}{(\beta^2 \times Precision + Recall)}$$

As β decreases, precision is given greater weight. With β = 1, we have the commonly used F1 score, which balances recall and precision equally and reduces to the simpler equation 2TP/(2TP + FP + FN).

Measure of quality

$$\text{Cost associated with our model} = 500 * FN + 10 * FP$$

Building ML models for hypotheses validation

Comparison ML models performance

Logistic Regression performance

```
#Let's see CV f1 score on LogisticRegression

print("Best parameter (CV f1 logistic regression score=%0.3f):" % model.best_score_)

Best parameter (CV f1 logistic regression score=0.767):

==========================================================
 Results from Grid Search LinearRegression best model
==========================================================
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15625
           1       0.48      0.89      0.62       375

    accuracy                           0.97     16000
   macro avg       0.74      0.93      0.81     16000
weighted avg       0.99      0.97      0.98     16000

==========================================================
 Results from Grid Search LinearRegression best model
==========================================================
Confusion Matrix:
 [[15264   361]
 [   41   334]]
==========================================================
Type 1 error (False Positive) =  361
==========================================================
Total cost =  24110
==========================================================
```

```
# Defining the LR model and performing the hyper parameter tuning using gridsearch
clf_lr = LogisticRegression(n_jobs= -1,random_state=42)
params = {'C' : [
                10**-4,10**-3,10**-2,10**-1,1,10**1,10**2,10**3],
          'penalty': ['l1', 'l2'],
         }

cv = KFold(n_splits=5)
model = GridSearchCV(clf_lr,cv =cv,n_jobs= -1,param_grid=params,scoring='f1',verbose= 10)
model.fit(train_lr_std,y_train_Sampled)
print("Best estimator is", model.best_params_)

Fitting 5 folds for each of 16 candidates, totalling 80 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
[Parallel(n_jobs=-1)]: Done   2 tasks      | elapsed:    2.5s
[Parallel(n_jobs=-1)]: Done   9 tasks      | elapsed:    3.1s
[Parallel(n_jobs=-1)]: Done  16 tasks      | elapsed:    3.8s
[Parallel(n_jobs=-1)]: Done  25 tasks      | elapsed:    4.4s
[Parallel(n_jobs=-1)]: Done  34 tasks      | elapsed:    7.3s
[Parallel(n_jobs=-1)]: Done  45 tasks      | elapsed:   21.5s
[Parallel(n_jobs=-1)]: Done  56 tasks      | elapsed:  1.5min
[Parallel(n_jobs=-1)]: Done  74 out of  80 | elapsed:  4.6min remaining:   22.4s
[Parallel(n_jobs=-1)]: Done  80 out of  80 | elapsed:  5.2min finished
```

Random Forest performance

```
==========================================================
Results from Grid Search RandomForest best model
==========================================================
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15625
           1       0.52      0.94      0.67       375

    accuracy                           0.98     16000
   macro avg       0.76      0.96      0.83     16000
weighted avg       0.99      0.98      0.98     16000

==========================================================
 Results from Grid Search RandomForest best mod
==========================================================
Confusion Matrix:
 [[15296   329]
 [   21   354]]
==========================================================
Type 1 error (False Positive) =  329
Type 2 error (False Negative) =  21
==========================================================
Total cost =  13790
==========================================================

[Parallel(n_jobs=-1)]: Done 1497 tasks      | elapsed: 37.4min
[Parallel(n_jobs=-1)]: Done 1552 tasks      | elapsed: 39.7min
[Parallel(n_jobs=-1)]: Done 1620 out of 1620 | elapsed: 42.6min finished
Best estimator is {'criterion': 'entropy', 'max_depth': 12, 'max_features': 'sqrt', 'n_estimators': 70}
```

```
#Let's see CV f1 score on Random Forest

print("Best parameter (CV f1 random forest score=%0.3f):" % model2.best_score_)

Best parameter (CV f1 random forest score=0.781):
```

```
# model fitting and hyperparamter tunning
clf_rf = RandomForestClassifier()

params1 ={
     'n_estimators':[10,20, 50, 70, 100,150, 200, 300, 500],
     'max_depth':[3, 4, 5, 7, 10, 12],
    'max_features': ['auto', 'sqrt', 'log2'],
    'criterion':['gini', 'entropy']

}
cv = KFold(n_splits=5)
model2=GridSearchCV(clf_rf, param_grid=params1, verbose=10, n_jobs=-1,scoring='f1',c
model2.fit(train_Sampled_Smote, y_train_Sampled)
print("Best estimator is", model2.best_params_)

Fitting 5 folds for each of 324 candidates, totalling 1620 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

XGBoost performance

```
==========================================================
 Results from Grid Search XGBoost best model
==========================================================
              precision    recall  f1-score   support

           0       1.00      0.99      0.99     15625
           1       0.64      0.91      0.75       375

    accuracy                           0.99     16000
   macro avg       0.82      0.95      0.87     16000
weighted avg       0.99      0.99      0.99     16000

==========================================================
 Results from Grid Search XGBoost best model
==========================================================
Confusion Matrix:
 [[15435   190]
 [   35   340]]
==========================================================
Type 1 error (False Positive) =  190
Type 2 error (False Negative) =  35
==========================================================
Total cost =  19400
==========================================================

Best estimator is {'max_depth': 12, 'n_estimators': 1000}
```

```
#Let's see CV f1 score on XGBoost

print("Best parameter (CV f1 XGBoost score=%0.3f):" % model3.best_score_)

Best parameter (CV f1 XGBoost score=0.789):
```

```
# model fitting and hyperparameter tunning using gridsearch
clf_xgb = XGBClassifier()

params2 ={
     'n_estimators':[10, 20, 50, 70, 100, 150, 200, 300, 500, 1000],
     'max_depth':[3, 4, 5, 7, 10, 12],
}
cv = KFold(n_splits=5)
model3=GridSearchCV(clf_xgb , param_grid=params2, verbose=10, n_jobs=-1, scoring='f1'
model3.fit(train_Sampled_Smote, y_train_Sampled)
print("Best estimator is", model3.best_params_)

Fitting 5 folds for each of 60 candidates, totalling 300 fits
[Parallel(n_jobs=-1)]: Using backend LokyBackend with 8 concurrent workers.
```

Best model performance is Random Forest

Using Random Forest further

Skoltech

Building ML models for hypotheses validation

Building Random Forest to validate hypothesis 1

Random Forest with dropping less useful features

```
              precision    recall  f1-score   support

          -1       1.00      0.42      0.59     15625
           1       0.04      0.98      0.08       375

    accuracy                           0.44     16000
   macro avg       0.52      0.70      0.33     16000
weighted avg       0.98      0.44      0.58     16000
```

```
In [36]: from sklearn.metrics import confusion_matrix,f1_score
con_mat =confusion_matrix (y_test, y_pred_rf)
print("-"*117)
print('Confusion Matrix: ', '\n',con_mat)
print("-"*117)
print("Type 1 error (False Positive) = ", con_mat[0][1])
print("Type 2 error (False Negative) = ", con_mat[1][0])
print("-"*117)
print("Total cost = ", con_mat[0][1] * 10 + con_mat[1][0] * 500)
print("-"*117)
```

```
---------------------------------------------------------------------
Confusion Matrix:
 [[6601 9024]
 [   8  367]]
---------------------------------------------------------------------
Type 1 error (False Positive) =  9024
Type 2 error (False Negative) =  8
---------------------------------------------------------------------
Total cost =  94240
---------------------------------------------------------------------
```

Random Forest without dropping less useful features

```
============================================================
 Results from Grid Search RandomForest best model
============================================================
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15625
           1       0.52      0.94      0.67       375

    accuracy                           0.98     16000
   macro avg       0.76      0.96      0.83     16000
weighted avg       0.99      0.98      0.98     16000
```

```
con_mat =confusion_matrix (y_test,y_pred_rf_best)
# Results from GridSearchCV
print("\n============================================================")
print(" Results from Grid Search RandomForest best model" )
print("============================================================")
print('Confusion Matrix: ', '\n',con_mat)
print("============================================================")
print("Type 1 error (False Positive) = ", con_mat[0][1])
print("Type 2 error (False Negative) = ", con_mat[1][0])
print("============================================================")
print("Total cost = ", con_mat[0][1] * 10 + con_mat[1][0] * 500)
print("============================================================")
```

```
============================================================
 Results from Grid Search RandomForest best model
============================================================
Confusion Matrix:
 [[15296   329]
 [   21   354]]
============================================================
Type 1 error (False Positive) =  329
Type 2 error (False Negative) =  21
============================================================
Total cost =  13790
============================================================
```

Conclusion: bad performance, our hypothesis 1 is wrong

Skoltech

Building ML models for hypotheses validation

Building Random Forest to validate hypothesis 2

Random Forest with PCA performance

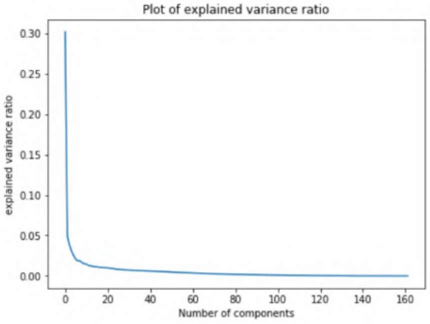
```
=============================================================
 Results from Grid Search RandomForest PCA best model
=============================================================
              precision    recall  f1-score   support

           0       1.00      0.97      0.99     15625
           1       0.47      0.96      0.64       375

    accuracy                           0.97     16000
   macro avg       0.74      0.97      0.81     16000
weighted avg       0.99      0.97      0.98     16000
```

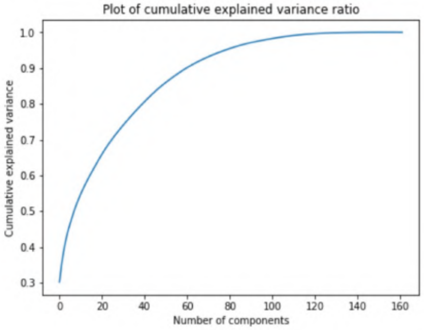
```
con_mat =confusion_matrix (y_test,y_pred_rf_best_pca)
# Results from GridSearchCV
print("\n=============================================================")
print(" Results from Grid Search RandomForest PCA best model" )
print("=============================================================")
print('Confusion Matrix: ', '\n',con_mat)
print("=============================================================")
print("Type 1 error (False Positive) = ", con_mat[0][1])
print("Type 2 error (False Negative) = ", con_mat[1][0])
print("=============================================================")
print("Total cost = ", con_mat[0][1] * 10 + con_mat[1][0] * 500)
print("=============================================================")
```

```
=============================================================
 Results from Grid Search RandomForest PCA best model
=============================================================
Confusion Matrix:
 [[15226   399]
 [   14   361]]
=============================================================
Type 1 error (False Positive) =  399
Type 2 error (False Negative) =  14
=============================================================
Total cost =  10990
=============================================================
```

```
print("With Number of components as 90, the cumulative explained variance ratio is "
      ,train_pca.explained_variance_ratio_[:90].sum())
```

```
With Number of components as 90, the cumulative explained variance ratio is  0.9699427596876562
```





Random Forest without PCA performance

```
=============================================================
 Results from Grid Search RandomForest best model
=============================================================
              precision    recall  f1-score   support

           0       1.00      0.98      0.99     15625
           1       0.52      0.94      0.67       375

    accuracy                           0.98     16000
   macro avg       0.76      0.96      0.83     16000
weighted avg       0.99      0.98      0.98     16000
```

```
con_mat =confusion_matrix (y_test,y_pred_rf_best)
# Results from GridSearchCV
print("\n=============================================================")
print(" Results from Grid Search RandomForest best model" )
print("=============================================================")
print('Confusion Matrix: ', '\n',con_mat)
print("=============================================================")
print("Type 1 error (False Positive) = ", con_mat[0][1])
print("Type 2 error (False Negative) = ", con_mat[1][0])
print("=============================================================")
print("Total cost = ", con_mat[0][1] * 10 + con_mat[1][0] * 500)
print("=============================================================")
```

```
=============================================================
 Results from Grid Search RandomForest best model
=============================================================
Confusion Matrix:
 [[15296   329]
 [   21   354]]
=============================================================
Type 1 error (False Positive) =  329
Type 2 error (False Negative) =  21
=============================================================
Total cost =  13790
=============================================================
```

Building ML models for hypotheses validation

Building Random Forest to validate hypothesis 2

Optimal threshold

Random Forest with PCA with optimal threshold

```
# Finding the best threshold

scores = proverka.predict_proba(data_test_pca)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, scores)

# Algorithm to find the best threshold
min_cost = np.inf
best_threshold = 0.5
costs = []
for threshold in thresholds:
    y_pred_threshold = scores > threshold
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_threshold).ravel()
    cost = 10*fp + 500*fn
    costs.append(cost)
    if cost < min_cost:
        min_cost = cost
        best_threshold = threshold
print("========================================================")
print("Best threshold: {:.4f}".format(best_threshold))
print("========================================================")
print("Min cost: {:.2f}".format(min_cost))
print("========================================================")
y_pred_test_final_pr = proverka.predict_proba(data_test_pca)[:,1] > best_threshold
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test_final_pr).ravel()
print("Total cost = ",10*fp + 500*fn)
print("========================================================")
```

```
========================================================
Best threshold: 0.4281
========================================================
Min cost: 9190.00
========================================================
Total cost =  9190
========================================================
```

Random Forest without PCA with optimal threshold

```
# Finding the best threshold

scores = clf_rf_best.predict_proba(test_impMedian)[:,1]
fpr, tpr, thresholds = roc_curve(y_test, scores)

# Algorithm to find the best threshold
min_cost = np.inf
best_threshold = 0.5
costs = []
for threshold in thresholds:
    y_pred_threshold = scores > threshold
    tn, fp, fn, tp = confusion_matrix(y_test, y_pred_threshold).ravel()
    cost = 10*fp + 500*fn
    costs.append(cost)
    if cost < min_cost:
        min_cost = cost
        best_threshold = threshold

print("========================================================")
print("Best threshold: {:.4f}".format(best_threshold))
print("========================================================")
print("Min cost: {:.2f}".format(min_cost))
print("========================================================")
y_pred_test_final = clf_rf_best.predict_proba(test_impMedian)[:,1] > best_threshold
tn, fp, fn, tp = confusion_matrix(y_test, y_pred_test_final).ravel()
print("Total cost = ",10*fp + 500*fn)
print("========================================================")
```

```
========================================================
Best threshold: 0.2244
========================================================
Min cost: 9050.00
========================================================
Total cost =  9050
========================================================
```

Conclusion: good performance, our hypothesis 2 is right

Skoltech

Obtained results

Interesting findings: finding optimal threshold make great result, many highly correlated with the target features not useful together.

Remarks on ML experiments: non-linear models take large amount of time in comparison with linear, which can be used for changing the performance after manipulations with dataset

The applicability of the model in a real-life scenario: we have a working classifier which is able to tell a fleet operator whether or not a truck needs to be serviced, based solely on sensor readings from the APS, load program to computer in operator center, when the operator receives data from the truck sensor, the program based on our model analyzes them and in the case of unstable operation of this system and the risk of breakage associated with it , makes it possible to take the necessary measures and prevent breakdowns.

Overall conclusions

Main goal reached, obtained performance is good enough.

The cost reduction estimation: as you can see the difference between cost for working without model and with is huge. It means that problem solved and roughly the half of money saved.

Costs without our model:

≈ $347500
for trucks which need to be serviced

1. Unnecessary checks done by a mechanic - $10
2. Missing a faulty truck, which may cause a breakdown in the future - $500

Costs with our model:

≈ $196800
for trucks which need to be serviced

Profit from implementation of our model:

≈ $150700

Thank you for your attention!