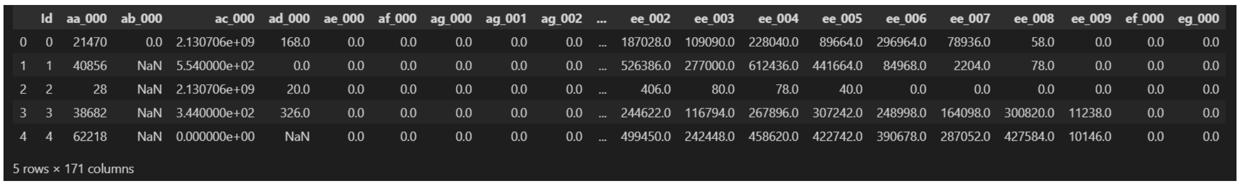
Szigethy Virág
B37EUN

Data mining and machine learning assignment

About the data:

The given data we had to work with was APS Faliure at Scania Trucks, which focuses on the air pressure system of Scania trucks. Component failures for a certain APS system component make up the dataset's positive class. Trucks in the bad class have malfunctions for parts unrelated to the APS. The main goal of the classification is to forecast whether the APS system will fail and require repair as well. After classification, the false negative cases have worse outgoing than the false positives. The attribute names of the data are unknown for us. Both simple numerical counters and histograms made up of bins with various conditions can be found in it. In the data set there are 171 attributes, of which 7 are histogram variables with 39900 samples.

Without any preprocessing, the raw loaded data looked like this:

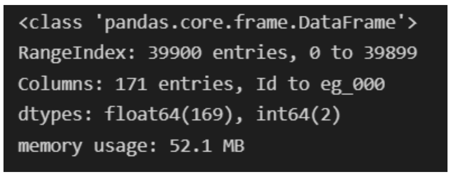


	Id	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee_002	ee_003	ee_004	ee_005	ee_006	ee_007	ee_008	ee_009	ef_000	eg_000
0	0	21470	0.0	2.130706e+09	168.0	0.0	0.0	0.0	0.0	0.0	...	187028.0	109090.0	228040.0	89664.0	296964.0	78936.0	58.0	0.0	0.0	0.0
1	1	40856	NaN	5.540000e+02	0.0	0.0	0.0	0.0	0.0	0.0	...	526386.0	277000.0	612436.0	441664.0	84968.0	2204.0	78.0	0.0	0.0	0.0
2	2	28	NaN	2.130706e+09	20.0	0.0	0.0	0.0	0.0	0.0	...	406.0	80.0	78.0	40.0	0.0	0.0	0.0	0.0	0.0	0.0
3	3	38682	NaN	3.440000e+02	326.0	0.0	0.0	0.0	0.0	0.0	...	244622.0	116794.0	267896.0	307242.0	248998.0	164098.0	300820.0	11238.0	0.0	0.0
4	4	62218	NaN	0.000000e+00	NaN	0.0	0.0	0.0	0.0	0.0	...	499450.0	242448.0	458620.0	422742.0	390678.0	287052.0	427584.0	10146.0	0.0	0.0

5 rows × 171 columns

It can be seen, that since the names of the features were anonymized, it is pretty hard to figure out what they actually mean (at least I was not able to figure it out).

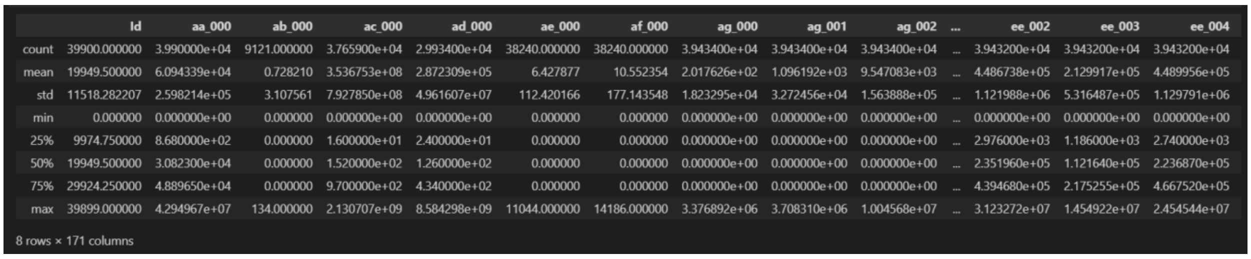
There is a useful function for pandas Dataframes called info which I also applied on the data. The results of it can be seen here:



```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 39900 entries, 0 to 39899
Columns: 171 entries, Id to eg_000
dtypes: float64(169), int64(2)
memory usage: 52.1 MB
```

From this, I think the most useful information was (without looking into the data itself) that the data set does not contain strings, only numerical elements.

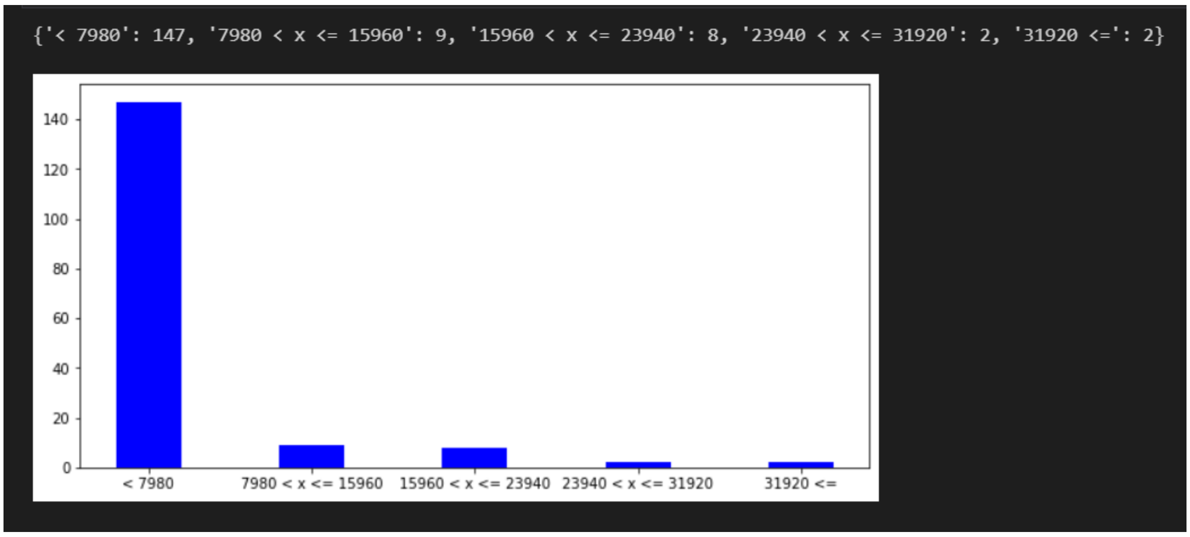
Another useful function in connection with dataframes is describe. It calculates 8 distinct statistical values from each column.



	Id	aa_000	ab_000	ac_000	ad_000	ae_000	af_000	ag_000	ag_001	ag_002	...	ee_002	ee_003	ee_004
count	39900.000000	3.990000e+04	9121.000000	3.765900e+04	2.993400e+04	38240.000000	38240.000000	3.943400e+04	3.943400e+04	3.943400e+04	...	3.943200e+04	3.943200e+04	3.943200e+04
mean	19949.500000	6.094339e+04	0.728210	3.536753e+08	2.872309e+05	6.427877	10.552354	2.017626e+02	1.096192e+03	9.547083e+03	...	4.486738e+05	2.129917e+05	4.489956e+05
std	11518.282207	2.598214e+05	3.107561	7.927850e+08	4.961607e+07	112.420166	177.143548	1.823295e+04	3.272456e+04	1.563888e+05	...	1.121988e+06	5.316487e+05	1.129791e+06
min	0.000000	0.000000e+00	0.000000	0.000000e+00	0.000000e+00	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	...	0.000000e+00	0.000000e+00	0.000000e+00
25%	9974.750000	8.680000e+02	0.000000	1.600000e+01	2.400000e+01	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	...	2.976000e+03	1.186000e+03	2.740000e+03
50%	19949.500000	3.082300e+04	0.000000	1.520000e+02	1.260000e+02	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	...	2.351960e+05	1.121640e+05	2.236870e+05
75%	29924.250000	4.889650e+04	0.000000	9.700000e+02	4.340000e+02	0.000000	0.000000	0.000000e+00	0.000000e+00	0.000000e+00	...	4.394680e+05	2.175255e+05	4.667520e+05
max	39899.000000	4.294967e+07	134.000000	2.130707e+09	8.584298e+09	11044.000000	14186.000000	3.376892e+06	3.708310e+06	1.004568e+07	...	3.123272e+07	1.454922e+07	2.454544e+07

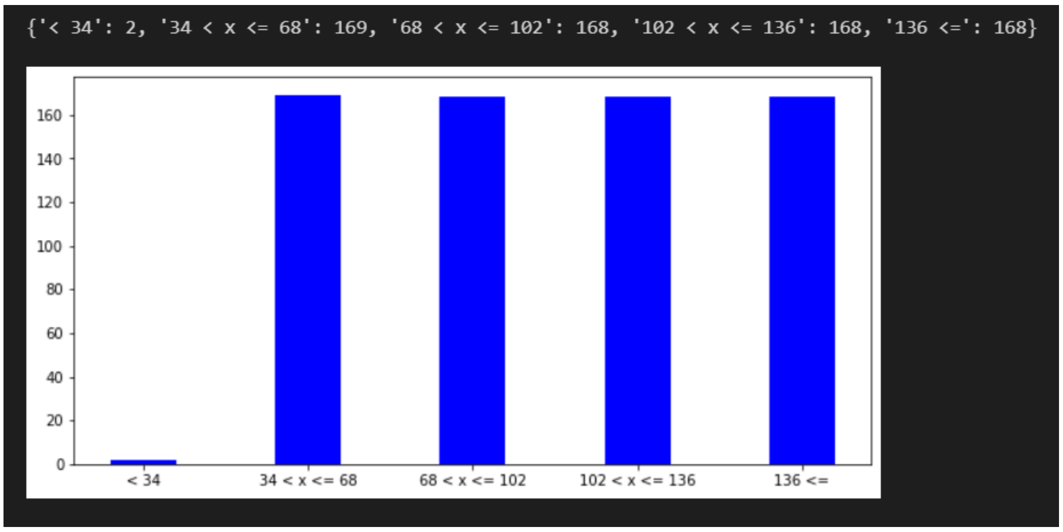
8 rows × 171 columns

Since I found out that there was a pretty large number of Nan-s, I decided that I will make some bar charts about the number of Nan-s in rows and columns. I made 5 equally big sections from them and calculated the results. From the columns, I got the following bar chart with the exact values:



```
{'< 7980': 147, '7980 < x <= 15960': 9, '15960 < x <= 23940': 8, '23940 < x <= 31920': 2, '31920 <=': 2}
```

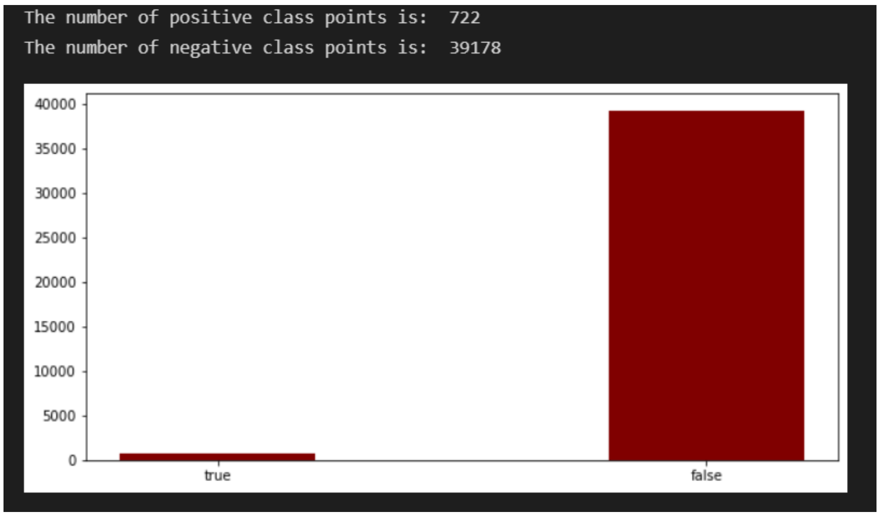
And from the rows, I got this:



```
{'< 34': 2, '34 < x <= 68': 169, '68 < x <= 102': 168, '102 < x <= 136': 168, '136 <=': 168}
```

So unfortunately it turned out that there are loads of samples which contain many Nan values, but only a few features which are extremely ’empty’.

Before trying out any algotithms, I plotted the rate of true and false samples in the given data. It turned out, that there are much more false data saples than true.



```
The number of positive class points is:  722
The number of negative class points is:  39178
```

I also checked if there are any duplicates in the data set, buti t turned out that there are no duplicates.

Preprocessing:

Since a loads of missing values (NaN) appear in the dataset, I had to replace them with something. Before replacing them, I tried the to drop all columns or all rows which contains missing values. By dropping columns, it turned out that only one feature (aa_000) exists in the dataset which has an existing value for every sample. By dropping rows, it turns out that from our 39900 data samples, only 406 contained values for all features. Since I considered those numbers too low, I decided to replace the missing values with something. I tried using mean and median, but based on some algorithms (DecisionTreeClassifier, KNeighborsClassifier and XGBClassifier) it seemed that the best way of replacing missing values is by zero. After that, all my runs were made by the database where missing values were replaced with zeros.

For the trying out algorithms, I made a train and a test set from the data which I had classified labels for. I tried both 70%-30% and 80%-20% splits, but it didn’t seem to make a big difference, so I sticked with 80%-20% split. Since the data set is extremely unbalanced, I used stratification during the train test split in order to get a balanced split.

Feature selection:

Since in the ’About the data’ section it turned out that there are a few features extremely empty (by empty I mean that it has a lot of Nan values), I decided that I try to remove them from my dataset. Because that, I reduced every column from my datafram which had more than 75% of Nan values. By that, I got 165 columns instead of 170. After that, I swapped all Nan-s in the dataframe to zero values. Unfortunately this came to my mind at the last minute, so I did not try it with the algorithm shown below.

Szigethy Virág
B37EUN

Algorithms:

Before I started coding, I went trough the lecture and the seminar PPT-s and PDF-s and made notes about the algorithms I wanted to try out. I was sure that I wanted to try as much classificators as possible, and since clustering was also mentioned in the requirements, I wanted to try KMeans as well. First, I tried out the all the algorithms without any preprocessing and with default parameters, and my idea was to start working and tuning the best performing models after I've seen them without any modification.

1R algorithm:

First I wanted to try a really simple algorithm, and since on lectures we discussed 1R algorithm, I tried that one out. Since 1R algorithm has a pretty high computational need, I used only the first 20000 element of the dataset for training. Unfortunately it was only able to achieve 6.58% accuracy ont he test data.

K Neighbors Classifier:

We already tried this algorithm at the lab sessions, so this was one of the first ones I tried with those data. Without stratification, I got 40.58%, and with stratification I got 49.43%.

Decision Tree Classifier:

This was also a familiar algorithm from the labs, without stratification I was able to achieve 73.61%, and with stratification I got 73.00%. This was my fist 'good' result, so I uploaded the classification made by this classifier on the validation data (without stratification) to kaggle. On kaggle, I scored 68.82% with it .

K-means:

I tried the K-means algorithm as well, since the description of the assignment contained clustering algorithms, but unfortunately the best I could achieve without any preprocessing was a 0.78% accuracy.

SVM:

After that, I tested a model which also did not perform too well without any further modification. It was the Support Vector Classifier. Without stratification I got 1.47% accuracy, and with stratification I got 3.84%.

XGB Classifier:

At the lectures, we did not discuss XGBClassifier in detail (altough there were a few slides about it), and I heard a lot good about it, so I decided to try it as well. Before the results, I want to write a few words from XGBClassifier. It has became one of the most popular algorithms in the past few years, since it is a very efficient classifier and regressor as well. The basic concept of the XGBClassifier is boosting. It has a scikit-learns wrapper interface, so it can be used just like any other models in scikit-learn. This is the model I chose at the end of trying out algorithms, since without stratification, on the test data I achieved 79.30% and with stratification, I got 80.36%. Since at this point it was my best result, I uploaded my results on the validation data to moodle. I scored 75.09% without stratification and 72.76% with stratification.

Random Forest Classifier:

As the next step, I tried out the Random Forest Classifier, which was already familiar from the lessons. Unfortunately It was not very successful. First I got 25.07%, but after stratification I was able to achieve only 23.30%.

Passive Aggressive Classifier:

I tried this one out, because one of my classmates told that he could achieve with it a really good result, but without any tuning, optimization or preprocessing, I only got 36.42% and 13.67%.

SGD Classifier:

SGD Classifier is a linear classifier optimized by SGD. While I tried out SGD Classifier, the first classification result was 0.00% and 71.08% with stratification. I'm still trying to find out why I got such bad results without stratification, but because those results, I tried scaling on the data as preprocessing, because at the Passive Aggressive Classifier, it worked very well. You can read from those result at the 'Further improvements according to algorithms' section.

Ridge Classifier:

Since we learnt about Ridge, I wanted to see how it can perform in a classification task. Without any preprocessing or tuning, I could achieve 55.86% and 49.24 % after stratification.

AdaBoost Classifier:

AdaBoost Classifier also seemed a promising algorithm, since as I heard it is pretty popular. The results I got with this were 75.07% and

Gaussian Naïve Bayes:

Naïve Bayes was a quite big part of the learnt curriculum, so I wanted to try that as well (not only on paper). Without stratification a 60.69% and with it a 74.64% accuracy were achieved.

Tuning:

Bayes search CV:

I tried that method with both SVC and XGBClassifiers as estimators, since in the hyparparameter tuning slide, I read that those are very powerful combinations. Unfortunately I only achieved 52.53% and 32.32%. The lack of great performance might be due to the fact that I only applied the Bayes search CV algorithm on the first 100 samples of the training data which is extremely small part of all the data. It was becouse of the time requirement of the algorithm and my lack of time. With this algorithm, I did not use stratification during train test split.

XGB Classifier:

Since XGB Classifier seemed the best classifier after trying out a few, I decided to make its' performance a little bit better. First of all, I searched on google to get a little more information about the algorithm. By that, I found this website: https://towardsdatascience.com/xgboost-fine-tune-and-optimize-your-model-23d996fab663 Fist of all, I changed the classifiers' parameters from default. I changed max_depth -which sets the maximum depth of the decision tree- from 6 to 11, the n_estimators -which will tell the classifier how many trees are in our ensemble and is equivalint to the number of boosting rounds- from 100 to 120 and the number of subsamples – which could be helpful in setting the complexity of the model- from 1 to 0.8. The first two parameters came from trying different ones out, and the third one came from intuition. I changed that one into a smaller number, because lower subsample could prevent overfitting (but might lead to under-fitting). Since without those parameter changes, I experienced worse accuracy on the unknown validation data, I thought that maybe my model overfitted. From 78.30% I got 82.34% and from 80.36%, I got 79.83%. I also used the algorithm with these parameters for the validation data (without stratification) and by that I got 76.71%.

Cross validation:

For finding the best input parameters for my best performing algorithms (XGB Classifier). I tried a few combinations with 5 fold cross validation. The accuracy I started with was 75.53% with the previously chosen parameters. The best parameters for XGB Classifier turned out as: eta=0.2 (which is the learning rate), gamma=1 (min split loss), max_depth=11, n_estimators=120 and subsample=0.8. With those paramaters, I got a 75.69% accuracy. There are several other parameters of XGB Classifier (such as min_child_weight, max_delta_step, sampling_method, colsample_bytree, lambda, alpha, etc.) but because of the lack of time, I only could try to optimize those 5 parameters. After all, I got pretty similar parameters as the ones I got from trying to optimize on 80-20 train test split.

Further improvements according to algorithms:

Passive Aggressive Classifier + Standard Scaler:

Since my classmate said that he got good results with some preprocessing and this algorithm, I decided to try out something simple about it. Using a Standard Scaler seemed promising, so I tried that out. With this modification, from 36.42%, I could increase the accuracy to 52.92%, and with stratification I achieved an even bigger improvement. From 13.67%, I got 70.64%.

SGD Classifier + Standard Scaler:

Because of the really bad accuracy without any preprocessing and stratification, I tried out SGD classifier with Scandard Scaler as well. From 0.00% I immediately achieved 68.01%, and from 71.08% I got 70.49%.

XGB Classifier + Standard Scaler:

Since XGB Classifier seemed the best choise, I tried it out with the combinations of preprocessing and/or tuning algorithms as well. As the result of XGB Classifier combined with Standard Scaler, I got 70.79% and 75.78% with the already selected parameter values (max_depth=11, n_estimators = 120, subsample = 0.8)

Evaluation:

Cross validation:

Since it turned out that there was a pretty big gap between my estimated accuracy on the test data and the predictions' accuracy on evaluation data (based on kaggle), I decided to try out cross validation to get a more accurate estimate on the precision. First of all, I chose the 5 best performing algorithms so far to be tested by cross validation. It was the Decision Tree Classifier, the XGB Classifier, the SGD Classifier, the AdaBoost Classifier and the Gaussian Naive Bayes. First, I tried out 5 fold cross validation and got the following results: 67.71% for Decision Tree Classifier, 75.53% for XGB Classifier, 52.22% for SGD Classifier, 69.15% for AdaBoost Classifier and 75.83% for Gaussian Naive Bayes. After that, I became curious what would be the difference between 5 and 10 fold cross validation, so I tried out 10 fold cross validation as well. With it, I achieved 69.49%, 76.53%, 58.99%, 69.88% and 75.72%. Since Gaussian Naive Bayes had pretty good performance during cross validation, I uploaded the prediction of it to kaggle. On kaggle, it scored 77.12%, which was my best solution yet.

Ideas for further improvement:

Unfortunately I did not have time to try all the algorithms and techniques I wanted to, but here is a sort list about those which might increase the performance of my model: Hyperparameter tuning with grid search, finding outlier samples and reducing them, using confusion matix or cost matrix, using SVM after PCA, normalization of the data and automatic attribute selection methods (probably correlation). Another useful thing could be the elimintion of those samples which contain too many Nan values.

Summary:

First of all, I used a few functions and made some bar plots to have some idea about the data I will work with. After that, I went trough the lecture slides and the lab codes and collected the algorithms and techniques I wanted to try. I tried out every algorithm without any preprocessing with 80-20 data split, and later on with stratification. Stratification usually led to the increase of the accuracy. I tried out 12 different classifiers with my data, and chose the best performing ones. The very best I tried was XGB Classifier. For the best ones, I examined the performance with Standard scaler as well. This improved the result in almost all cases, so I considered this a very useful technique. By the time, I uploaded a few results to the kaggle competition as well, and I observed a huge gap between the accuracies I got on the test data and the achieved accuracy on the validation data. Because this, I assumed that my model overfitted, and since cross validation could decrease the chance of overfitting, I applied 5 and 10 fold cross validation on the data. It also seemed very useful, since now the gap was not so big. For tuning the models, I used cross validation, and also tried Bayesian cross validation. With the latter, I was not very successful.

Automated ML (extension):

During this work, I also tried an autoML model, which was Extra Trees Classifier. Extra Trees (which stands for extremely randomized trees) is an ensemble supervised machine learning method that uses decision trees and is used by the Train Using AutoML tool. Whith this and without any further modification, my results were 69.39% and 74.65% accuracy.