Grid search

Before the process of picking and training models initiates, a topic has to be clarified. The machine learning field has plenty of existing models, each of which has its respective hyperparameters. Due to the size of the data, some models will require more time to be trained than others, which will result in a very slow testing and trying process. In order to simplify the process *Grid Search* will be used.

- The traditional way of performing hyperparameter optimization has been **grid search**, or a parameter sweep, which is simply an exhaustive searching through a manually specified subset of the hyperparameter space of a learning algorithm.(267/Hands-on MAL)

```
from sklearn.ensemble import GradientBoostingClassifier

gbrt = GradientBoostingClassifier()
from sklearn.model_selection import GridSearchCV
param_grid=[{'max_depth':[2,3,4,5,6,7],'n_estimators':[20,30,40,50,60,70,80,90,100],'learning_rate':[.1,.2,.3,.4,.5,.6,.
grid_search = GridSearchCV(gbrt, param_grid, cv=5, scoring='accuracy', return_train_score=True, n_jobs=-1)
grid_search.fit(X_train,y_train)
```

```
GridSearchCV(cv=5, estimator=GradientBoostingClassifier(), n_jobs=-1,
             param_grid=[{'learning_rate': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7,
                                            0.8, 0.9, 1],
                          'max_depth': [2, 3, 4, 5, 6, 7],
                          'n_estimators': [20, 30, 40, 50, 60, 70, 80, 90,
                                           100]}],
             return_train_score=True, scoring='accuracy')
```

```
grid_search.best_params_
```

```
{'learning_rate': 0.2, 'max_depth': 7, 'n_estimators': 70}
```

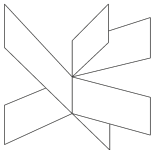
(Example of a GridSearch finding the best parameters for a GradientBoostingClassifier)

Linear Models

Referring to the plot of the balanced dataset, now the problem domain begins to show itself. It can be seen that the points labeled as alarms (Blue/Abnormal) are grouping up together, but are still hardly separable from the majority of points labeled as normal system behaviour (Orange/Normal). This tells us that *linear machine learning models* might not be able to distinguish accurately between both.

- **Linear Models** are a class of models that are widely used in practice and have been studied extensively in the last few decades. Linear models make a prediction using a *linear function* of the input features. (47/Introduction to MAL)

To prove our concerns, the dataset will be fit to two of the most popular Linear Models used today - *Logistic Regression* and *Support Vector Classifier*.



- **Logistic Regression (Logit Regression)** is commonly used to estimate the probability that an instance belongs to a particular class. If the estimated probability is greater than 50%, then the model predicts that the instance belongs to that class. (140/Hands-on MAL)
- **Support Vector Classifier (SVM)** - Given a set of training examples, each marked as belonging to one or the other of two categories, an SVM training algorithm builds a model that assigns new examples to one category or the other, making it a non-probabilistic binary linear classifier. (153/Hands-on MAL)

Logistic Regression

```
from sklearn.linear_model import LogisticRegression

logreg=LogisticRegression(C=1000).fit(X_tr_scaled,y_train)

print("Accuracy of the training set: {:.2f}".format(logreg.score(X_tr_scaled,y_train)))
print("Accuracy of the testing set: {:.2f}".format(logreg.score(X_te_scaled,y_test)))
```

```
Accuracy of the training set: 0.72
Accuracy of the testing set: 0.70
```

Support Vector Machine

```
from sklearn.svm import SVC

svm = SVC(kernel='rbf', C=1, gamma=.1).fit(X_train, y_train)

print("Accuracy on the training set: {:.2f}".format(mlp.score(X_tr_scaled,y_train)))
print("Accuracy on the testing set: {:.2f}".format(svm.score(X_test,y_test)))
```

```
Accuracy on the training set: 0.83
Accuracy on the testing set: 0.73
```

70% accuracy accounts for ⅔ correctly labeled rows. On the first glance the algorithms work pretty good with the provided data, but there are still 30% of rows which are misclassified.

Nearest neighbor model

So far only linear models have been tested on the dataset, the outcome of which wasn't satisfactory, telling us that it is hard to put a line on a low dimensional graph

between the two system states. There exists another nonlinear model, which requires little to no parameter tuning and is fast and simple - *the k-nearest neighbors* model.

- The **k-NN algorithm** is arguably the simplest machine learning algorithm. Building the model consists only of storing the training dataset. To make a prediction for a new data point, the algorithm finds the closest data points in the training dataset - its “nearest neighbors”. (37/Introduction to MAL)

K-nearest neighbor

```
from sklearn.neighbors import KNeighborsClassifier
clf = KNeighborsClassifier(n_neighbors=4)

clf.fit(X_train, y_train)

print("Train set accuracy: {:.2f}".format(clf.score(X_train, y_train)))
print("Test set accuracy: {:.2f}".format(clf.score(X_test, y_test)))
```

```
Train set accuracy: 0.83
Test set accuracy: 0.73
```

The results of the k-NN algorithm are underwhelming and resemble the results from the SVM algorithm. The reason for that can be the grouping of the data points and lack of class clusters, resulting in the poor performance of this algorithm. So far it is established that there can’t be a clean line between the two classes and that the classes are clustering together, making it hard for nearest path algorithms to distinguish between both. Another approach has to be undertaken.

Decision Tree

To reach a conclusion whether linear and k-nearest neighbor models are sufficient for our dataset, a comparison to another type of Machine Learning algorithm has to be made. A fitting model for the case is the *Decision Tree* model.

- **Decision trees** are widely used models for classification and regression tasks. Essentially, they learn a hierarchy of if/else questions, leading to a decision. (72/Introduction to MAL)

Fitting such a model does not require any complex adjustment of parameters. The only parameter that needs to be tweaked is the depth of the tree, meaning how many questions should the decision tree ask before it reaches a conclusion. After a couple of iterations of the algorithm, the best depth established for the current dataset was 7.

Decision tree

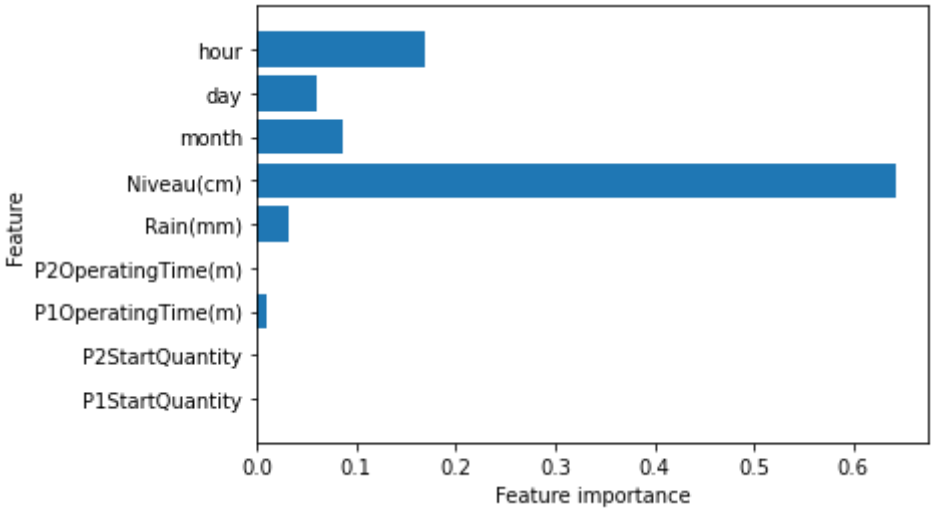
```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=7,random_state=42)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(tree.score(X_test,y_test)))
```

```
Accuracy on training set: 0.80
Accuracy on test set: 0.77
```

Now it can be seen that the accuracy is 77%, classifying ¾ of the labels correctly. This works much better than the linear models with little to no parameter adjustment.

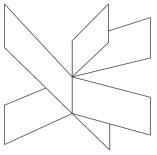
Feature Importance

Another helpful function of the Decision tree model is its feature importance, meaning which features does the algorithm look the most at, when it makes a prediction.



(Visualization of feature importance of the trained decision tree model, higher means more important)

In order to elaborate further on the diagram above, the use of *Binary Classification* models needs to be discussed first.



Defining the scope

In order for a machine learning model to be chosen, the problem has to be concisely phrased - "*The goal of predicting breakdowns is to accurately assign an event text, given a set of values*". The phrase resembles the description of one of the two main fields of machine learning algorithms - *Classification*.

- **Classification** is a process related to categorization, the process in which ideas and objects are recognized, differentiated and understood. (85/Hands-on MAL)

The classification approach falls into the category of *supervised machine learning* where a human (Herning Vand employee) has to provide a set of previously labeled attributes (pump information), which the model can use to train itself to predict new input, based on the training data.

- **Supervised learning** is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. (27/Introduction to MAL)

In our case the use of classification falls into the *binary classification* category.

- **Binary classification** is the task of classifying the elements of a set into two groups on the basis of a classification rule. (27/Introduction to MAL)

So far only binary classification models have been used for the purpose of swift testing and analysis. Another type of classification known in the Machine Learning field is called *multi-label classification*.

- In machine learning, **multi-label classification** and the strongly related problem of **multi-output classification** are variants of the classification problem where multiple labels may be assigned to each instance. (27/Introduction to MAL)

Up to now the features have been classified as an alarm/ not alarm. This is a clear and fast approach for our case, but the process of alarm labeling conducted by the Herning Vand employees goes into more detail. Let's look at all the labels from all the alarm data provided by the company.

(List of alarm labels, 64 in total)

For further clarification, the attributes have to be paid a second visit.

- Operating time
- Water quantity
- Rain quantity
- Level of water

A clear relation can be drawn from some of the alarm labels to the attributes. For example all labels starting with ”Niveau” relate to the level of water in the pump.

Unfortunately that case does not apply to every single label. Some alarms are triggered by hardware malfunctioning or pressure, which would be hard to estimate solely by using pump contents, rain and operating time. In order for all of the labels to be properly classified, more features have to be included in the current dataset (f.eks voltage, current, etc).

There is another point that needs to be made. As previously mentioned, Herning Vand collects pump information every hour unlike the alarms, which occur at a certain time (minutes, seconds) between the hours of data collection. To properly adjust the labels, if the alarm happened before the 30 minute mark (XX:30) of the hour, it was rounded to the current hour (XX:00), and respectively if it occurred above the 30 minute mark (XX:00) it was rounded towards the next hour (XX+1:00).

The rounding of the alarm times resulted in specific rows holding multiple labels of alarms.

```
ARNN01PS1 Offline
Alarm Batterifejl G401
Alarm Driftfejl P1
Alarm Driftfejl P2
Alarm fejl på modul bus
Alarm høj niveau
Alarm højhøj niveau
Alarm modem kommunikation
Alarm netfejl
Alarm overløb
Alarm termofejl P1
Alarm termofejl P2
Alarm trip fejl P1
Alarm trip fejl P2
Alarm tørløb
Alarm:Alarmkontrol er blokeret
Alarm:Batterifejl
Alarm:Høj niveau vippe
Alarm:Overløb
Alarm:Styrestrømsfejl
Drift fejl P1 (HFI / Sikring)
Drift fejl P2 (HFI / Sikring)
Flow:Alarmkontrol er blokeret H
Flow:Alarmkontrol er blokeret HH
Flow:Alarmkontrol er blokeret L
Flow:Alarmkontrol er blokeret LL
HAMDQYPS2 Offline
HAMDYBPS1 Offline
Mixer Cmd:Bloker Alarm SCADA
Niveau:Alarmkontrol er blokeret H
Niveau:Alarmkontrol er blokeret HH
Niveau:Alarmkontrol er blokeret L
Niveau:Alarmkontrol er blokeret LL
Niveau:Høj Alarmgrænse overskredet
Niveau:Høj Høj Alarmgrænse overskredet
Niveau:Lav Alarmgrænse overskredet
Niveau:Lav Lav Alarmgrænse overskredet
Niveau:Ledningsbrud
Normal
Omrører:Termofejl
PST Alarm blokering
PST Tilbagestop aktiv
Pumpe 1 Cmd:Bloker Alarm SCADA
Pumpe 1:Driftfejl
Pumpe 1:Ikke i Auto fra SCADA
Pumpe 1:Omskifter ikke i Auto
Pumpe 1:Sikkerhedsafbryder
Pumpe 1:Termofejl
Pumpe 1:Trip. Afventer automatisk reset
Pumpe 2 Cmd:Bloker Alarm SCADA
Pumpe 2:Driftfejl
Pumpe 2:Ikke i Auto fra SCADA
Pumpe 2:Omskifter ikke i Auto
Pumpe 2:Sikkerhedsafbryder
Pumpe 2:Termofejl
Pumpe 2:Trip. Afventer automatisk reset
RPA Rapport kontrol af pumper
Rapport uge check
Tryk:Alarmkontrol er blokeret H
Tryk:Alarmkontrol er blokeret HH
Tryk:Alarmkontrol er blokeret L
Tryk:Alarmkontrol er blokeret LL
Tryk:Ledningsbrud
Understation offline
64
```

2020-03-30	07:00:00	Pumpe 2:Omskifter ikke i Auto, Pumpe 1 Cmd:Bloker Alarm SCADA
2020-03-30	08:00:00	Niveau:Høj Alarmgrænse overskredet, Pumpe 2:Trip. Afventer automatisk reset
2020-03-30	09:00:00	Pumpe 2:Trip. Afventer automatisk reset, Niveau:Lav Lav Alarmgrænse overskredet, Niveau:Lav Alarmgrænse overskredet

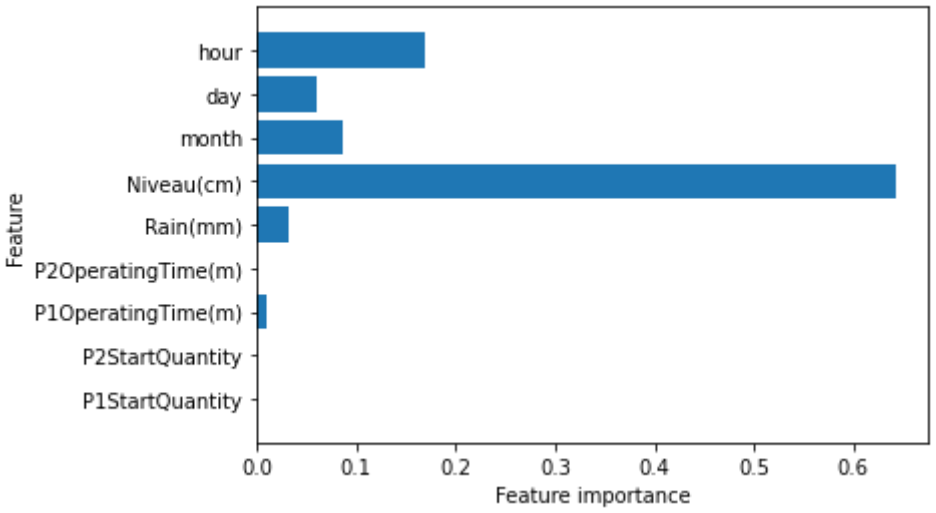
(Example of multiple labels of alarms in the dataset)

The problem of accurately estimating alarms does not end here. Some alarm labels have priority attached to them, which is another complex label that cannot be attained just by the little amounts of data features (pump attributes) provided by the company.

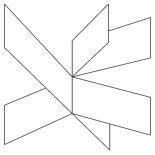
So far we discussed that:

- Most of the alarms have little to no relation to the provided data.
- There are cases in which rows hold multiple alarm labels.
- Some alarm labels have a priority label as well.

To visually represent the data insufficiency and the poor relation of the attributes to the alarm labels, the feature importance of the decision tree model table needs to be revisited.



Looking at the table it



can be deduced that the decision tree model values the level of water in the pipe, followed by the time of the measurement.
It makes sense for the Niveau to be of big importance, because the malfunctioning of the pump systems can lead to fluctuations in the level of water in the pipes.
Time is also an important feature, because the pumps follow a cycle of activation every 23rd hours of the day.

Little can be said for the rest of the attributes. Rain and Operating Time are valued to some extent but it doesn't look like the model bases the majority of its prediction on them.

This observation tells that it is not possible to train a multi-label classification algorithm with the current dataset.

(Example of the cycle of activation of the pumps)

	P1StartQuantity	P1OperatingTime(m)	P2StartQuantity	P2OperatingTime(m)	Rain(mm)	Niveau(cm)	date	time
0	0.0	0.0	0.0	0.0	0,0	98,9	2017-01-01	00:00:00
1	0.0	0.0	0.0	0.0	0,2	101,3	2017-01-01	01:00:00
2	0.0	0.0	0.0	0.0	1,2	97,7	2017-01-01	02:00:00
3	0.0	0.0	0.0	0.0	0,8	96,4	2017-01-01	03:00:00
4	0.0	0.0	0.0	0.0	1,6	97,3	2017-01-01	04:00:00
5	0.0	0.0	0.0	0.0	1,0	97,6	2017-01-01	05:00:00
6	0.0	0.0	0.0	0.0	0,8	97,9	2017-01-01	06:00:00
7	0.0	0.0	0.0	0.0	0,2	98,3	2017-01-01	07:00:00
8	0.0	0.0	0.0	0.0	0,0	98,0	2017-01-01	08:00:00
9	0.0	0.0	0.0	0.0	0,0	97,4	2017-01-01	09:00:00
10	0.0	0.0	0.0	0.0	0,0	97,0	2017-01-01	10:00:00
11	0.0	0.0	0.0	0.0	0,0	97,8	2017-01-01	11:00:00
12	0.0	0.0	0.0	0.0	0,0	97,9	2017-01-01	12:00:00
13	0.0	0.0	0.0	0.0	0,0	98,1	2017-01-01	13:00:00
14	0.0	0.0	0.0	0.0	0,0	97,2	2017-01-01	14:00:00
15	0.0	0.0	0.0	0.0	0,0	97,4	2017-01-01	15:00:00
16	0.0	0.0	0.0	0.0	0,0	98,0	2017-01-01	16:00:00
17	0.0	0.0	0.0	0.0	0,0	96,7	2017-01-01	17:00:00
18	0.0	0.0	0.0	0.0	0,0	96,8	2017-01-01	18:00:00
19	0.0	0.0	0.0	0.0	0,0	97,3	2017-01-01	19:00:00
20	0.0	0.0	0.0	0.0	0,0	98,2	2017-01-01	20:00:00
21	0.0	0.0	0.0	0.0	0,0	98,2	2017-01-01	21:00:00
22	0.0	0.0	0.0	0.0	0,0	96,0	2017-01-01	22:00:00
23	77.0	183.0	78.0	143.0	0,0	96,9	2017-01-01	23:00:00

Now that it is made clear that it would be difficult to train a multi-label model, let's get back to the best binary classification model tested so far - the decision tree model.

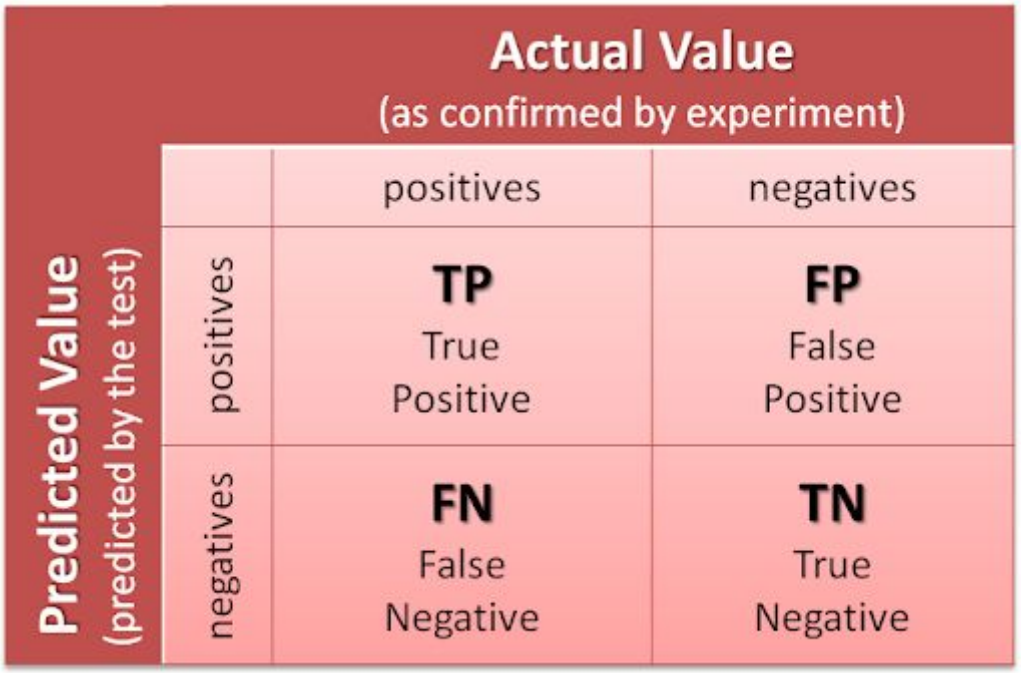
Decision tree

```
from sklearn.tree import DecisionTreeClassifier

tree = DecisionTreeClassifier(max_depth=7,random_state=42)
tree.fit(X_train, y_train)

print("Accuracy on training set: {:.2f}".format(tree.score(X_train, y_train)))
print("Accuracy on test set: {:.2f}".format(tree.score(X_test,y_test)))
```

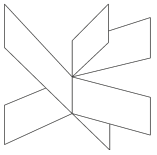
```
Accuracy on training set: 0.80
Accuracy on test set: 0.77
```

To gain more insight into the estimates of the model, let's show a confusion matrix.

- A much better way to evaluate the performance of a classifier is to look at the **confusion matrix**. The general idea is to count the number of times instances of class A are classified as class B. (90/Hands-on MAL)



The confusion matrix will give us an accurate representation of how many rows were correctly/incorrectly labeled by the model.



```
y_pred = tree.predict(X_test)
from sklearn.metrics import confusion_matrix
confusion_matrix(y_test, y_pred)
```

```
array([[737, 197],
       [206, 729]], dtype=int64)
```

After getting the confusion matrix of the decision tree model, it can be seen that the model failed to label 197 “alarms” as actual alarms, and wrongly labeled 206 “not alarms” as alarms. This is normal, given the low variance of the provided dataset and the poor relation of the attributes to the actual alarms.