Warsaw University of Technology

Faculty of Power and Aeronautical Engineering

Simple Machine Learning Solutions for Condition-Based Maintenance Issues

Predicting the turbine decay state coefficient of a naval propulsion plant using TensorFlow and AWS Cloud Computing

Karol Roliński

January 21, 2021

1. Introduction

The most important recipe for success both in industry and in private life is without a doubt making the right decisions. Otherwise, it could result in costly material losses as well as the health and life of people. In order to facilitate the decision-making process more and more machine learning algorithms are implemented into everyday life.

The importance of preventive maintenance is well known to the industry, but it was not always like this. In the past maintenance was performed only after a breakdown of a component [2]. This was a very expensive approach due to the high costs of machine downtime and the components itself. Nowadays using preventive maintenance is about replacing a component when its life cycle reaches the end. The suitable moment for overhauls can be estimated mainly based on previously gathered experience and the defined average lifespan of a machine. Unfortunately, this method does not guarantee the breakdown-free operation of a system. In addition, replacing components too soon and frequent overhauls can also be quite costly. This is a trade-off between the breakdowns number and the lifetime estimation of the components. It is surely not an easy task, especially when the machine usage varies. This applies e. g. to the shipping industry. Therefore, to reduce costs and efficiently plan the maintenance of ships Condition-Based Maintenance seems to be the best solution. CBM is about monitoring the actual condition of an asset to decide when maintenance needs to be done. In most cases, the decay state of each component cannot be tracked with a sensor. CBM requires an accurate model able to predict the decay state based on other available sensors. Machine learning seems perfect for this task. In this paper it is shown how to create a basic reliable model for condition-based maintenance [3].

2. Model preparation

For the purpose of this paper a dataset from the UCI machine learning repository has been chosen [4]. Owing to the information about experiments contained on the website we obtained the needed data carried out by means of a numeric simulator of a naval vessel. The propulsion system of this frigate was based on a gas turbine (GT). The simulator has been developed and finetuned over the year on similar to the modeled one propulsion systems. This guarantees in theory that the given data is reliable. The featured dataset consists of 11934 instances, 16 features and 2 labels each. The features are as follows:

- Lever position (lp),
- Ship speed (v) [knots],
- Gas Turbine (GT) shaft torque (GTT) [kN m],
- GT rate of revolutions (GTn) [rpm],
- Gas Generator rate of revolutions (GGn) [rpm],
- Starboard Propeller Torque (Ts) [kN],
- Port Propeller Torque (Tp) [kN],
- Hight Pressure (HP) Turbine exit temperature (T48) [°C],
- GT Compressor inlet air temperature (T1) [°C],
- GT Compressor outlet air temperature (T2) [°C],

- HP Turbine exit pressure (P48) [bar],
- GT Compressor inlet air pressure (P1) [bar],
- GT Compressor outlet air pressure (P2) [bar],
- GT exhaust gas pressure (Pexh) [bar],
- Turbine Injecton Control (TIC) [%],
- Fuel flow (mf) [kg/s][4].

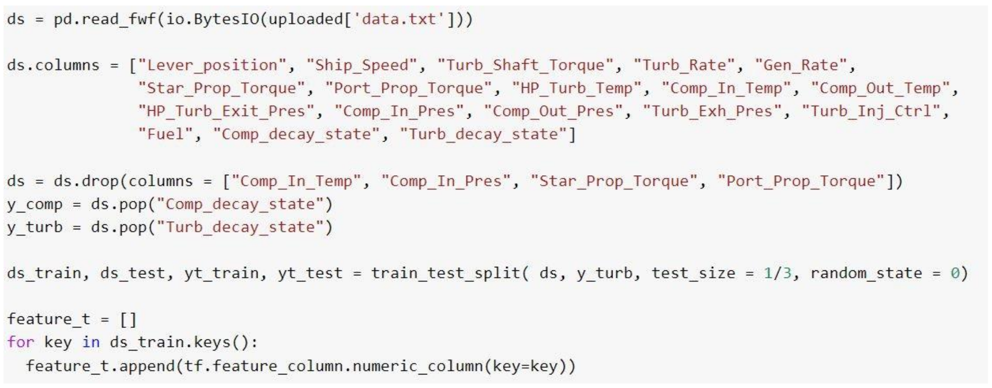
The two labels are as follows:

- GT Compressor decay state coefficient,
- GT Turbine decay state coefficient.

For the compressor decay state discretization, the coefficient has been investigated in the domain [1; 0.95], and the turbine coefficient in the domain [1; 0.975][4].

2.1 Dataset adjustment

The first step in creating a machine learning model is all about the preparation of the dataset. After loading, it is very important to make sure that the modeled process is fully understandable. Next, we eliminate the features that we know that have no influence on the given labels. The choice can be made based on the knowledge of the real-life dependencies or secondarily by observing the initially obtained model. Features with weights close to zero are dropped. Furthermore 3 dataframes were created: one features dataframe (ds) and two separate dataframes for each label (y_comp, y_turb).



```
ds = pd.read_fwf(io.BytesIO(uploaded['data.txt']))

ds.columns = ["Lever_position", "Ship_Speed", "Turb_Shaft_Torque", "Turb_Rate", "Gen_Rate",
              "Star_Prop_Torque", "Port_Prop_Torque", "HP_Turb_Temp", "Comp_In_Temp", "Comp_Out_Temp",
              "HP_Turb_Exit_Pres", "Comp_In_Pres", "Comp_Out_Pres", "Turb_Exh_Pres", "Turb_Inj_Ctrl",
              "Fuel", "Comp_decay_state", "Turb_decay_state"]

ds = ds.drop(columns = ["Comp_In_Temp", "Comp_In_Pres", "Star_Prop_Torque", "Port_Prop_Torque"])
y_comp = ds.pop("Comp_decay_state")
y_turb = ds.pop("Turb_decay_state")

ds_train, ds_test, yt_train, yt_test = train_test_split( ds, y_turb, test_size = 1/3, random_state = 0)

feature_t = []
for key in ds_train.keys():
  feature_t.append(tf.feature_column.numeric_column(key=key))
```

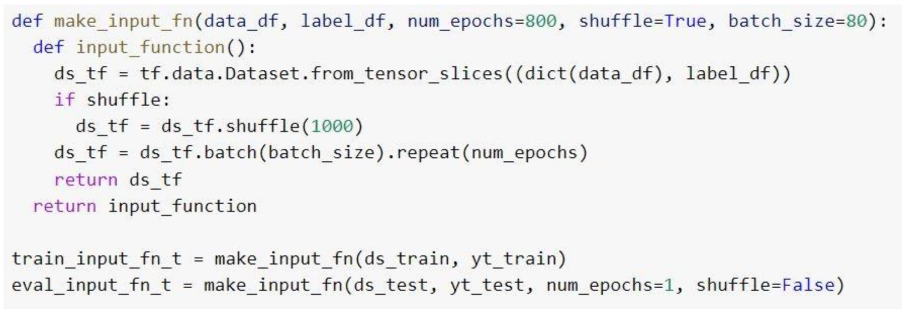
Fig. 1. Preprocessing of the dataset

In order to create a machine learning model, the data must be divided into a training and a test dataset in a 2:1 ratio. The used function shuffles the data to increase the effectiveness of the model.

2.2 TensorFlow estimator

One of the simplest approaches to machine learning are linear regression estimators. They are perfectly suited for predicting numeric values. The estimator tries to find a linear relation between the given features and the label. Creating a linear regressor can be done in many ways e.g. by using libraries specially designed for this. For convenience, the TensorFlow library has been selected. Its very flexible architecture allows us to easily deploy computations across a variety of platforms (CPUs, GPUs, TPUs), from desktops through clusters of servers to mobile and edge devices [5].

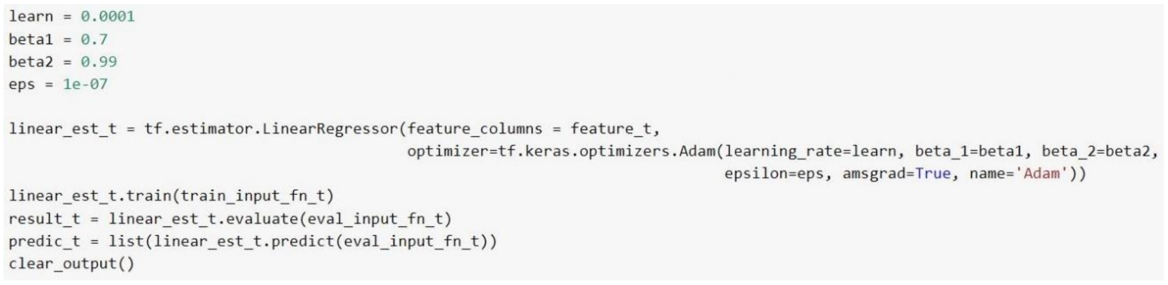
In TensorFlow computations are expressed as stateful dataflow graphs. Therefore, the model we are going to use requires that the data we insert comes in as a tf.data.Dataset object. This means we must create an input function that can convert the owned pandas dataframes into tf.data objects.



```
def make_input_fn(data_df, label_df, num_epochs=800, shuffle=True, batch_size=80):
  def input_function():
    ds_tf = tf.data.Dataset.from_tensor_slices((dict(data_df), label_df))
    if shuffle:
      ds_tf = ds_tf.shuffle(1000)
    ds_tf = ds_tf.batch(batch_size).repeat(num_epochs)
    return ds_tf
  return input_function

train_input_fn_t = make_input_fn(ds_train, yt_train)
eval_input_fn_t = make_input_fn(ds_test, yt_test, num_epochs=1, shuffle=False)
```

Fig. 2. Preparation of the input function

The created estimator features an optimizer that implements the Adam algorithm. It is a stochastic gradient descent method that is based on adaptive estimation of both first-order and second-order moments [5]. As a result of loss checking for various settings the following parameters were chosen: learning rate = 1e-4; beta_1 = 0.7; beta_2 = 0.99 and epsilon = 1e-7.



```
learn = 0.0001
beta1 = 0.7
beta2 = 0.99
eps = 1e-07

linear_est_t = tf.estimator.LinearRegressor(feature_columns = feature_t,
                                            optimizer=tf.keras.optimizers.Adam(learning_rate=learn, beta_1=beta1, beta_2=beta2,
                                                                               epsilon=eps, amsgrad=True, name='Adam'))
linear_est_t.train(train_input_fn_t)
result_t = linear_est_t.evaluate(eval_input_fn_t)
predic_t = list(linear_est_t.predict(eval_input_fn_t))
clear_output()
```

Fig. 3. Defining the linear estimator

After training the model with 2/3 of the available data we successfully created a linear regression model.

2.3 Results

In order to track the effectiveness of the model during its creation, the mean square error for the test set was constantly monitored. After making a few minor adjustments the final version of the model had a loss of 5,67e-5. In Figure 4 is a fragment of the comparison of the prediction (red) and the actual data (green) reported.

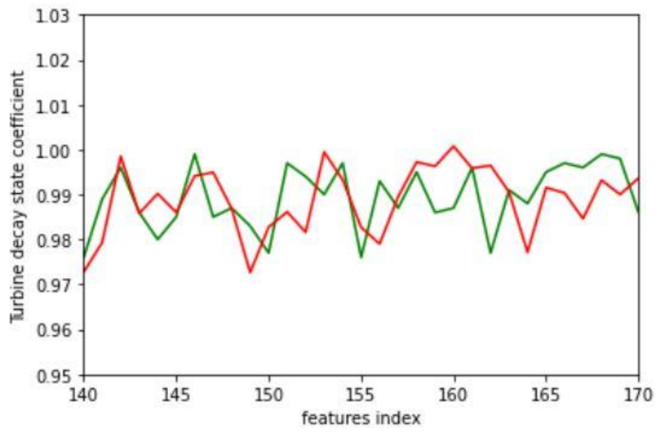


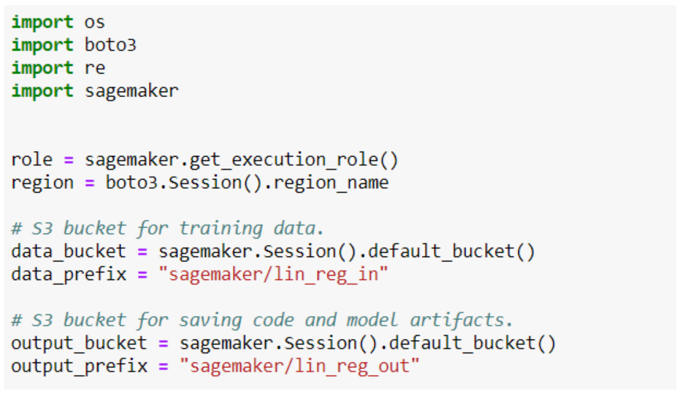
Fig. 4. Comparison of the predicted result with the actual data

3. Cloud Computing

Cloud computing is a service that gathers and processes data without the need to use software installed on your local hardware. The benefits of such an approach include cost savings, mobility, high computing powers and plenty storage space to just name a few. Cloud computing is a perfect solution especially if you need to use high computing power sporadically, because cloud computing can work in a pay-per-use manner [6].

Amazon Web Service offers its users a variety of tools. One of them is specifically dedicated to machine learning. Amazon SageMaker enables users to create, train, and deploy machine learning models in the cloud. It is a perfect tool for the sake of this paper, because it allows to run a Jupyter notebook on a selected instance without the need of manual configuration. The second useful tool provided by AWS is the Amazon Simple Storage Service (S3). It is a storage service that offers high performance data availability and security [6].

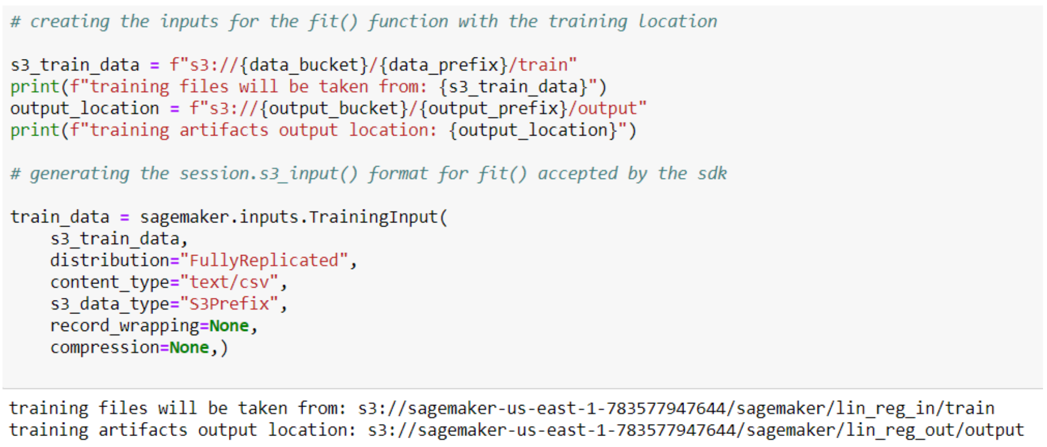
In order to use SageMaker some adjustment to the previously created model have been made. The main difference is that a S3 storage bucket will be used to store all the data that we require for training and evaluation of the model.



```
import os
import boto3
import re
import sagemaker


role = sagemaker.get_execution_role()
region = boto3.Session().region_name

# S3 bucket for training data.
data_bucket = sagemaker.Session().default_bucket()
data_prefix = "sagemaker/lin_reg_in"

# S3 bucket for saving code and model artifacts.
output_bucket = sagemaker.Session().default_bucket()
output_prefix = "sagemaker/lin_reg_out"
```

Fig. 5. Creation of the S3 buckets

The loaded dataset is split into the test and training set and then stored in the S3 bucket. Whenever necessary, data for the model is retrieved from the allocated space.

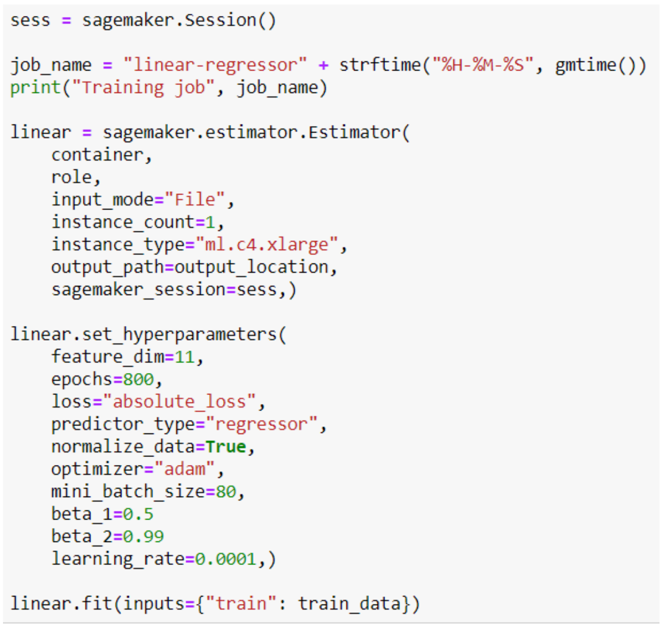


```
# creating the inputs for the fit() function with the training location

s3_train_data = f"s3://{data_bucket}/{data_prefix}/train"
print(f"training files will be taken from: {s3_train_data}")
output_location = f"s3://{output_bucket}/{output_prefix}/output"
print(f"training artifacts output location: {output_location}")

# generating the session.s3_input() format for fit() accepted by the sdk

train_data = sagemaker.inputs.TrainingInput(
    s3_train_data,
    distribution="FullyReplicated",
    content_type="text/csv",
    s3_data_type="S3Prefix",
    record_wrapping=None,
    compression=None,)
```

```
training files will be taken from: s3://sagemaker-us-east-1-783577947644/sagemaker/lin_reg_in/train
training artifacts output location: s3://sagemaker-us-east-1-783577947644/sagemaker/lin_reg_out/output
```

Fig. 6. Preparation of the training function

Previously obtained experience during the creation of the linear estimator locally now allows for faster deployment on the AWS server. SageMaker offers multiple algorithms in a similar way as TensorFlow. In fact, it has this library build within itself. The major difference is that on AWS you must define on which instance you would like the estimator to run. In this case it is the ‘ml.c4.xlarge’, specially optimized for high performance computing.



```
sess = sagemaker.Session()

job_name = "linear-regressor" + strftime("%H-%M-%S", gmtime())
print("Training job", job_name)

linear = sagemaker.estimator.Estimator(
    container,
    role,
    input_mode="File",
    instance_count=1,
    instance_type="ml.c4.xlarge",
    output_path=output_location,
    sagemaker_session=sess,)

linear.set_hyperparameters(
    feature_dim=11,
    epochs=800,
    loss="absolute_loss",
    predictor_type="regressor",
    normalize_data=True,
    optimizer="adam",
    mini_batch_size=80,
    beta_1=0.5
    beta_2=0.99
    learning_rate=0.0001,)

linear.fit(inputs={"train": train_data})
```

Fig. 7. Defining the linear estimator in AWS SageMaker

After evaluating, the model can be used in the same way as the one created previously. The result has a similar accuracy but is totally independent of the capabilities of the workstation the user is currently working on.

4. Conclusion

The above paper clearly shows that machine learning algorithms are a simple tool for improving the decision-making process regarding maintenance. The created model effectively allows to determine the decay state coefficient of the turbine of the simulated naval propulsion plant. Combing machine learning with the wide range of services offered by AWS it creates endless possibilities for the end-user. The fact that Cloud Computing is the fastest growing part of the natwork-based computing can't be denied. It offers a great advantage to customers of all sizes, from simple users or developers to enterprises and all types of organizations.

References

1. Coraddu A., Oneto L., *Machine Learning Approaches for Improving Condition-Based Maintenance of Naval Propulsion Plants*, Journal of Engineering for the Maritime Environment, 2014, DOI: 10.1177/1475090214540874,
2. Kothamasu R., Huang, S. H., *Adaptive mamdani fuzzy model for condition-based maintenance*, Fuzzy Sets and Systems, 158, 2715-2733,
3. Cipollini F., Oneto L., *Condition-Based Maintenance of Naval Propulsion Systems with Supervised Data Analysis*, Preprint submitted to Ocean Engineering May 25, 2018,
4. http://archive.ics.uci.edu/ml/datasets/condition+based+maintenance+of+naval+propulsion+plants
5. https://www.tensorflow.org
6. https://aws.amazon.com/