MACHINE LEARNING ENGINEER NANODEGREE

Predicting machine failure

Capstone Project

Marcus Holmgren
July 2021, Bijjie Lijhtie, South Lapland, Sweden

I. Definition

Exploring the accuracy of a simple neural network model on AI4I 2020 Predictive Maintenance Dataset.

Project Overview

Using machine learning to be able to predict machine failure can be beneficial in scheduling maintenance intervals, lead to better total cost of ownership for equipment parts and increase security when machine parts operate reliably. This projects aims to explore the AI4I 2020 Predictive Maintenance Dataset[1], a synthetic dataset that reflects real predictive maintenance data encountered in industry, and build a simple neural network model that is trained on AWS SageMaker.
The dataset originally contains 14 feature columns of which I decided to use 6 for my model training.

Problem Statement

I have decided to limit this experiment and not include the five individual failure modes and expand the model to a multi classification problem. Both as a time restraint and not to add extra complexity to model building and training. My main goal is to try to use explore if it is possible to predict machine failure from the available features from the AI4I 2020 Predictive Maintenance Dataset.
The model that I have chosen to build will be a simple neural network model built in PyTorch. Why I choose PyTorch is because there are some excellent tutorials available on AWS SageMaker example[2] GitHub repository. This model will then be deployed and be used to test and make a few predictions about machine failure.

Metrics

Since the problem that my model is trying to solve is a classification problem. I will use classification metrics from the scikit-learn[3] library to calculate accuracy, precision and recall. The accuracy metric is useful since it tells how many predictions the model was successful to correctly categories as machine failure or not. This metrics should be high if we want our model to be precise. Since I don't expect the model to achieve 100% accuracy two common metrics to measure the power of the model are precision and recall. Precision answer the question of all predicted machine failures, how many are truly relevant. Recall answer the question of all the machine failures, how many are found by the model[4]. Perfect precision and recall would mean there would not be any type I (incorrect rejection) or type II (incorrect retaining) errors[5]. Depending

[1] http://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

[2] https://github.com/aws/amazon-sagemaker-examples

[3] https://scikit-learn.org/stable/modules/model_evaluation.html#classification-metrics

[4] https://www.oreilly.com/library/view/evaluating-machine-learning/9781492048756/

[5] https://towardsdatascience.com/model-evaluation-i-precision-and-recall-166ddb257c7b

on what we favour in the outcome from the model we might accept one being higher than the other.

The data set that I have selected to train my model on is highly imbalanced but I will use oversampling with SMOTE to ensure that there is an even balance of machine failure categories in the training data. I therefor intend to explore and see what results I get from the following three metrics

Accuracy is calculated as the weighted mean of true predictions

$$Accuracy = \frac{T_p + T_n}{T_p + f_n + f_p + T_n}$$

Precision gives the fraction of examples of true positive values that belong to the positive predictions.

$$Precision = \frac{T_p}{T_p + f_p}$$

Recall summarise how well true positive values was predicted and is the same calculation as sensitivity.

$$Recall = \frac{T_p}{T_p + f_n}$$

II. Analysis

Because it is difficult to obtain real predictive maintencen datasets I have chosen to work with the synthetic dataset produced by Stephan Matzka, School of Engineering - Technology and Life, Hochschule für Technik und Wirtschaft Berlin, and published at UCI Machine Learning Repository[6].

Data Exploration

The dataset consists of 10 000 data points stored as rows with a total of 14 features in columns.

There are two columns of identifiers and I did not use them in this study:
- **UID**: numeric range from 1 to 10000
- **Product ID**: unique identifier

The six feature columns I have decided to use to train my model are:
- **Type**: consisting of a letter L, M, or H for low (50% of all products), medium (30%) and high (20%) as product quality variants and a variant-specific serial number
- **Air temperature [K]**: generated using a random walk process later normalized to a standard deviation of 2 K around 300 K
- **Process temperature [K]**: generated using a random walk process normalized to a standard deviation of 1 K, added to the air temperature plus 10 K.
- **Rotational speed [rpm]**: calculated from a power of 2860 W, overlaid with a normally distributed noise
- **Torque [Nm]**: torque values are normally distributed around 40 Nm with a Ïf = 10 Nm and no negative values.

[6] http://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset

- **Tool wear [min]**: The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process. and a

The target feature in this data set is the **Machine failure** column. It is a failure indicator from the five independent failure mode columns. They are explained on the UCI Machine Learning Repository page:

- **TWF**: tool wear failure
- **HDF**: heat dissipation failure
- **PWF**: power failure
- **OSF**: overstrain failure
- **RNF**: random failure

Because this is a synthetic dataset it is rather complete and did not contain any null or missing data points.
Looking at the data points with a pairwise relationships plot reveals that there are mainly unimodal data points.

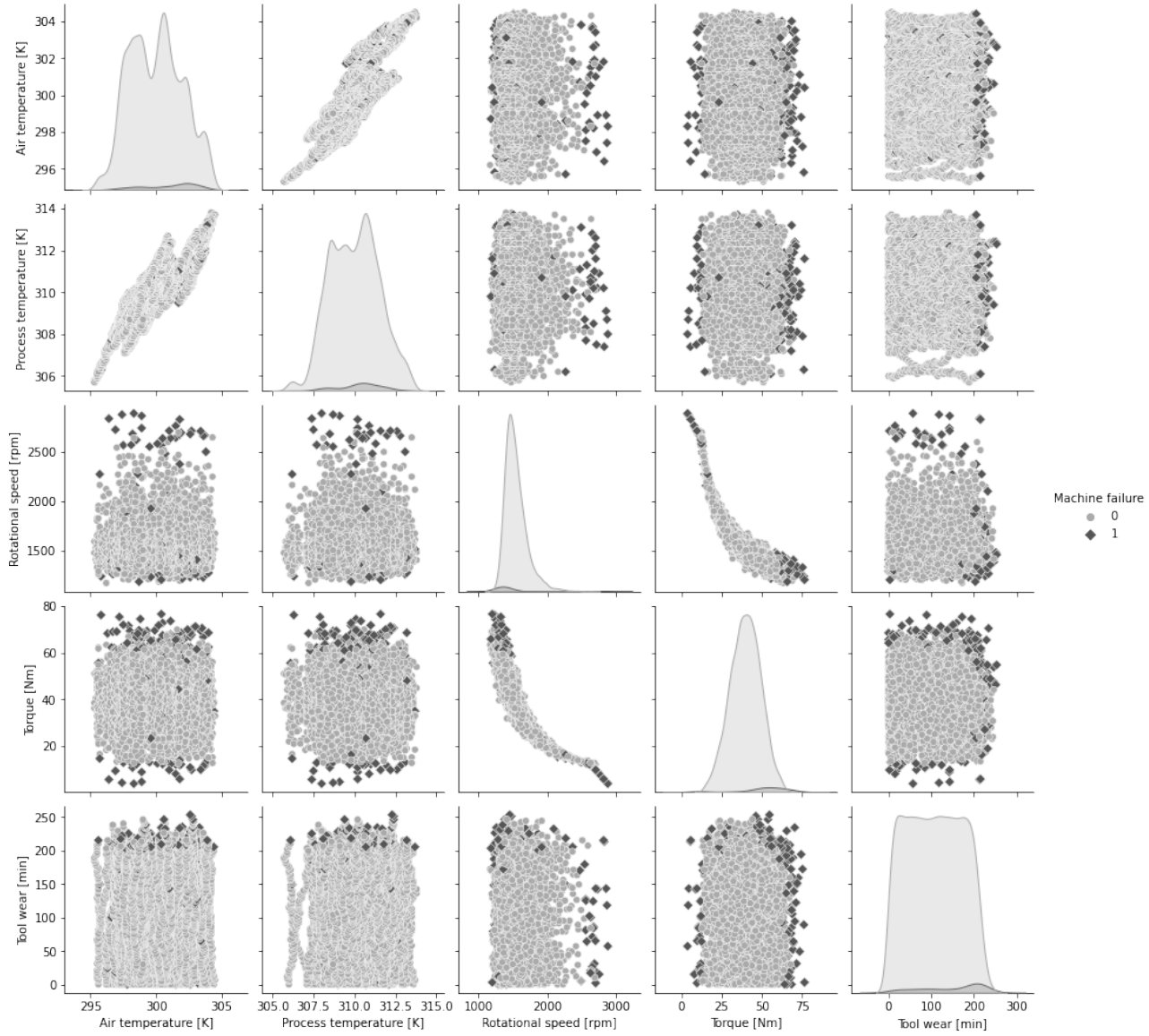

Figure 1. Pairwise relationships

96.6% of the dataset entries are not machine failure and only 3.4% is machine failure. This makes the dataset highly imbalanced and the will be important when generating training and evaluation set to have fair distribution of failures.

In the pair wise plots there is mainly distinct clusters of machine failures plotted in darker grey.

Exploratory Visualisation

This section explore the ranges of variables in boxplots[7].

The two temperature features are recorded in Kelvin degrees.

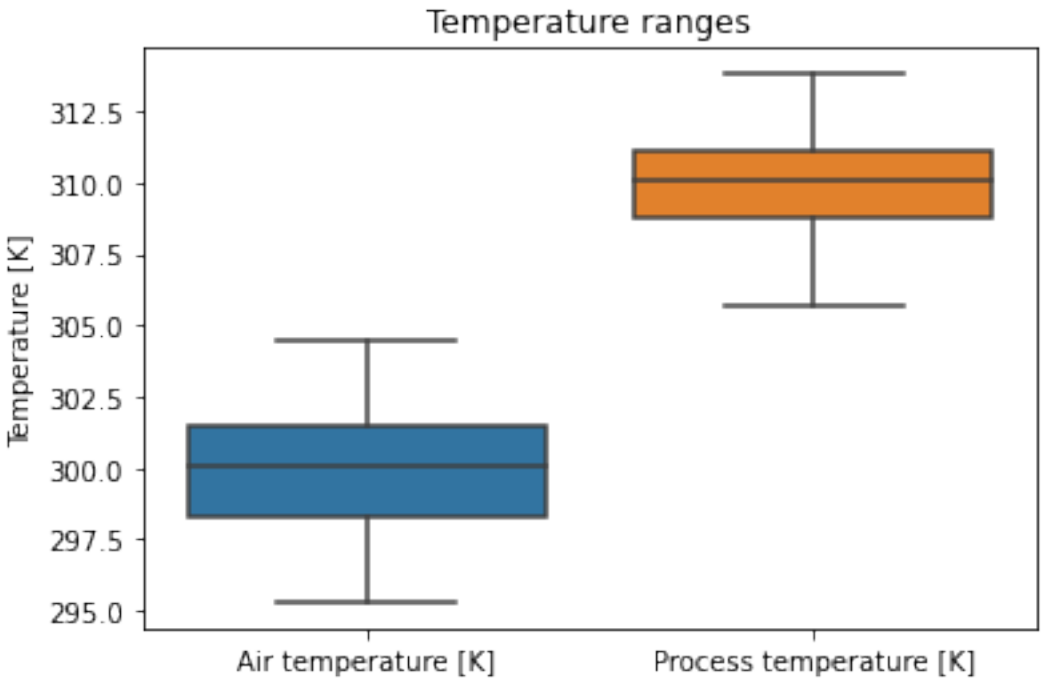

Figure 2. Temperature ranges box plot

Rotational speed have a few products that have speeds above 1800 rpm. They are indicated as the dots above the Q3 in the box plot.

---

[7] Understanding Boxplots, https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51

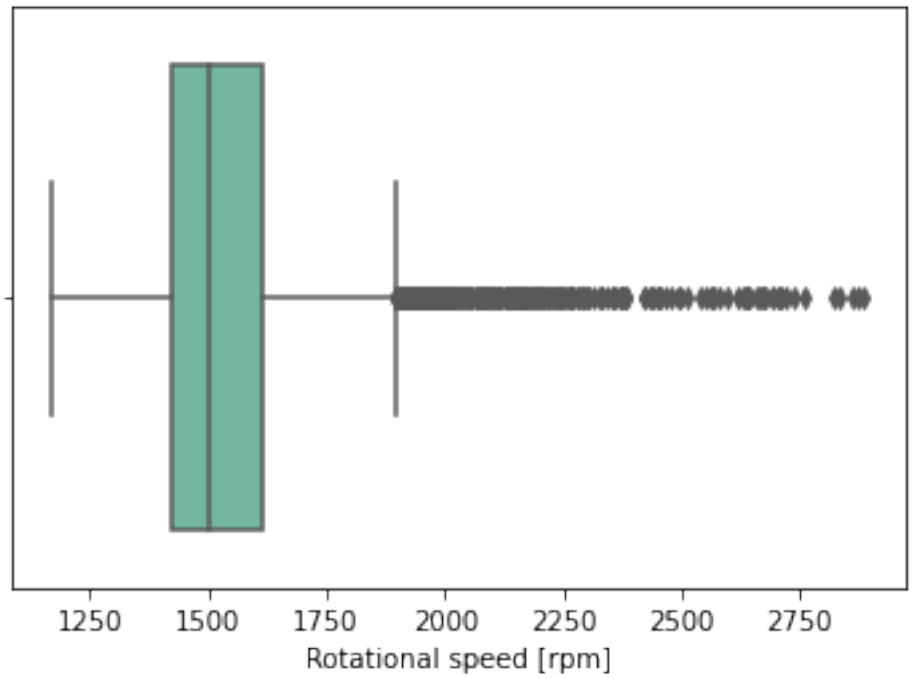

Figure 3. Rotational speed box plot

Torque in newton-meter, Nm. Also shows outliers in the low and high ranges of newton-meter

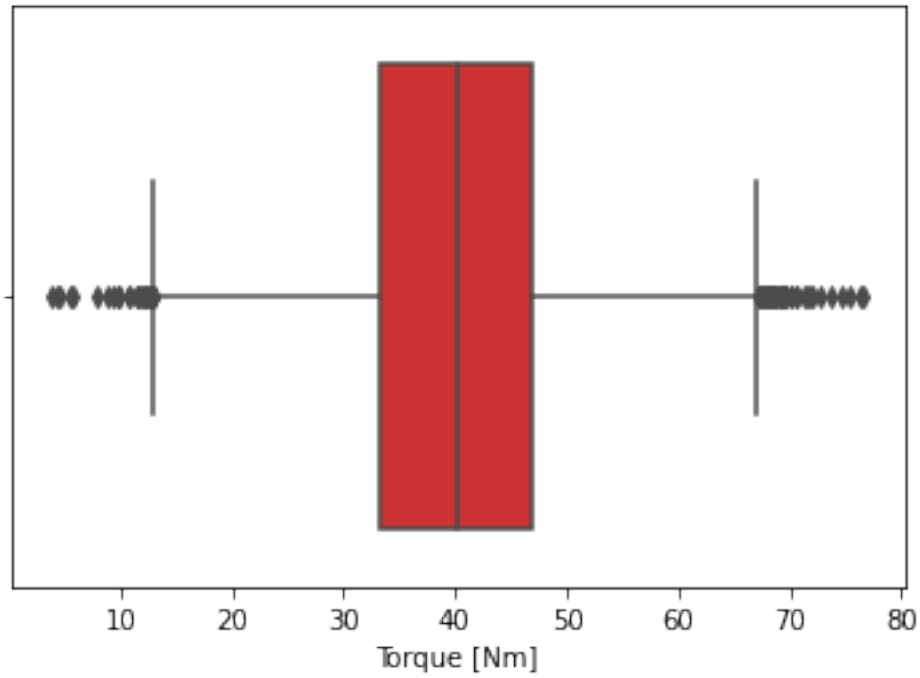

Figure 4. Torque box plot

Tool wear in The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process

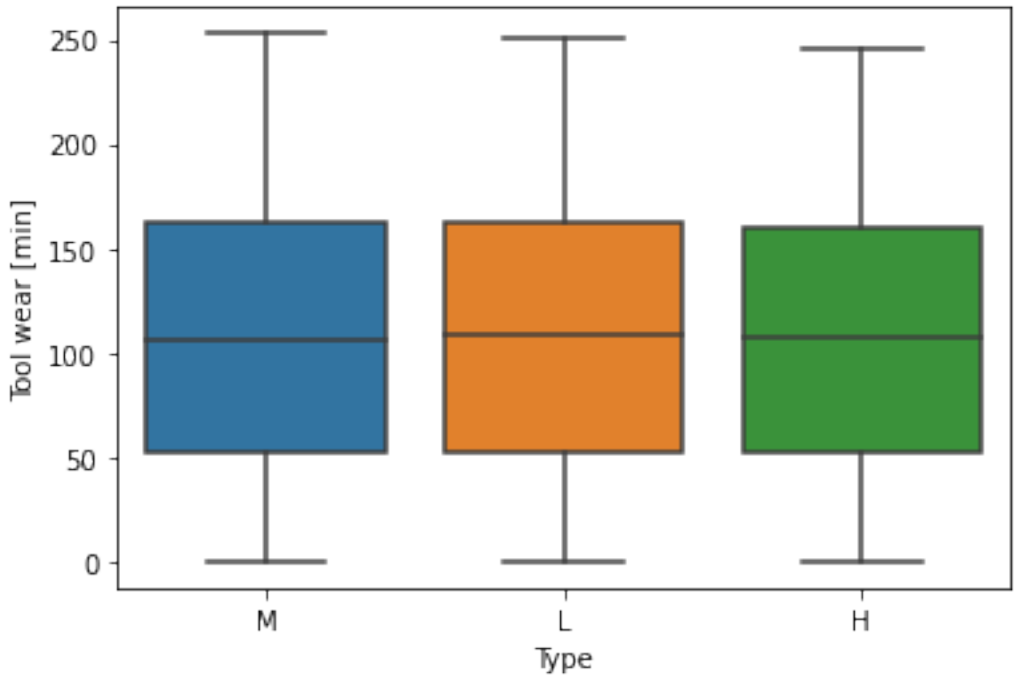

Figure 5. Type box plot

Algorithms and Techniques

From the data exploration and the pairwise relationship plot it looks like the data have linear relations and that there are clusters of distinct machine failures for several of the features.

For the baseline model I wanted to use a model that is easy to interpret, are known to perform well and that I don't have to write any custom training scripts. The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. I could repurpose the example from Breast Cancer Prediction[8]

For my own model I wanted to get a chance to learn more PyTorch so I decided to repurpose the example from SageMaker script mode[9].

The neural network model I have decided to use is a three layer fully connected model. The input layers have an activation function of tanh that have the benefit of centering output around zero. The only hidden layer uses a sigmoid activation function that centers the output around 0.5.

Since the available features are rather few I don't intend to use any regularisation technique, like dropout. To ensure that the deep neural network don't overfit the data I will first try fewer number of training epochs. If the training results show that accuracy is lower than the base model I will add a dropout layer and increase the training epochs.

The strength and benefit of neural network models are that they don't require so much feature engineering to perform rather well. Contrary to other machine learning models that would be easier to interpret if highly correlated features where removed with some technique like Principal Component Analysis, PCA.

The neural network summary

---

[8] https://sagemaker-examples.readthedocs.io/en/latest/introduction_to_applying_machine_learning/breast_cancer_prediction/Breast%20Cancer%20Prediction.html

[9] https://github.com/aws/amazon-sagemaker-examples/tree/master/sagemaker-script-mode

```
NeuralNet(
  (fc1): Linear(in_features=8, out_features=16, bias=True)
  (fc2): Linear(in_features=16, out_features=4, bias=True)
  (fc3): Linear(in_features=4, out_features=1, bias=True)
)
```

Benchmark

For evaluation of my deep neural network model I decided to use an algorithm that provides a robust result and that can easily be interpreted. The Amazon SageMaker linear learner algorithm provides a solution for both classification and regression problems. Since I'm using this algorithm with a predictor type of binary classification the model returns a score and predicted label. The predicted label is 0 or 1, and score is a single floating point number that indicates how strongly the algorithm believes that the label should be 1.

Linear Learner Algorithm, https://docs.aws.amazon.com/sagemaker/latest/dg/linear-learner.html

The evaluation metrics I used from the trained linear learner algorithm was scikit-learn metrics functions. The threshold for the prediction was set to above 0.5 for machine failure. For my first attempt with this baseline model the score was a rather poorly low accuracy of 4%. The precision and accuracy was also 4%.

The final baseline training used the dataset that was both oversampled and min-max scalar. The linear model reached an accuracy of 15% that I don't find impressive. The precision is 0.19 and the recall is 0.24

The confusion matrix is normalised against the predictions.

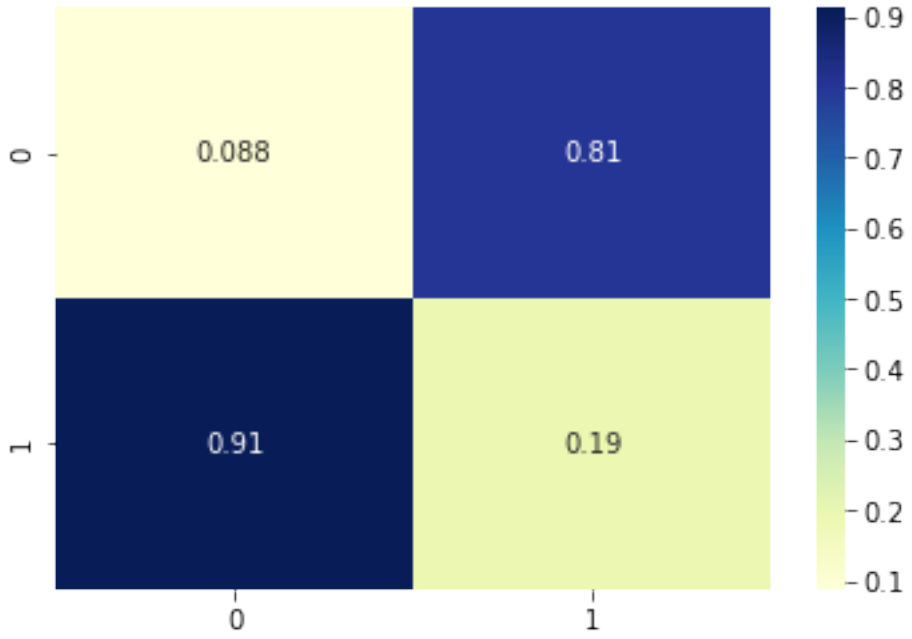

Figure 6. Linear learner baseline confusion matrix

III. Methodology

(approx. 3-5 pages)

Data Preprocessing

In my first training of baseline model with the AWS Linear Learner algorithm the result gave a rather poor accuracy of only 4%. The original dataset is highly imbalance with only less than 4% of the records being of machine failure. To compensate for this I decided to use oversampling to even the balance between the binary machine failure mode. Because there are only 339 true machine failure in the dataset I felt that it is better to oversample the failure rates than to under sample the false values and get a smaller dataset for training. After leveraging the imbalance-learn package and the Synthetic Minority Over-sampling Technique, SMOTE. Both the machine failure binary categories are balanced.

Machine failure	FALSE	TRUE
Original dataset	9661	339
SMOTE - oversampling	9661	9661

https://imbalanced-learn.org/stable/install.html

The Type feature is a categorical columns with three distinct values. This feature is transformed with one-hot encoding from the Pandas library.

The scikit-learn MinMaxScalar was used to normalise the five columns that have records that have entries that consist of big numbers that could give unintended influence on a feature.

Implementation

The implementation of the machine learning algorithms was done on AWS SageMaker with the help of the existing examples of the linear learner and PyTorch using script mode. This help me with the cold start problem of knowing how to and where to start. After mimicking the examples and starting to issue training jobs I found the Cloudwatch logs helpful in giving me directions on the adjustments that I needed to do to get to a successful training job.

One adjustment that I had to do with the Type features that was one-hot encoded in the feature engineering notebook. I had to change them from integers to floats, because PyTorch expects features to be floating point numbers.

The errors during training that I encountered was mainly caused by my lack of using PyTorch. But reading through the output and log files it was possible to identify the part of the script files that needed adjustment.

Refinement

The only thing I needed to adjust in the PyTorch model was the input layers neuron count to match the 8 features I decided to use in the train and test dataset. Then I set the the hidden layers input to be double the amount of neutrons from the input layer. This seems to work good enough for this experiment.

For the metric I found in the Seaborn plotting library the heat map that I could combine with Scikit-learn confusion matrix to make the figure 6 & 7 plots that showed the prediction power of the two models.

IV. Results

(approx. 2-3 pages)

The custom neural network model accuracy is a lot higher than the linear learner baseline.

Model Evaluation and Validation

When I got a model that trained and had higher accuracy than the baseline model I was satisfied of the result for this experiment.

Justification

The linear learner baseline model only reach accuracy of 0.15. And the PyTorch model reach accuracy of 0.79. I don't know if this is enough to be a useable model? But I think it proves that this problem can be model as a machine learning problem and can be explored to find a relevant model.

V. Conclusion

Free-Form Visualization

This is the confusion matrix of the PyTorch model that was able to outperform my baseline model. The matrix and confusion metrics seems to show that this model could be useful.

The performance metrics:
- Accuracy 0.79
- Precision 0.77
- Recall: 0.81

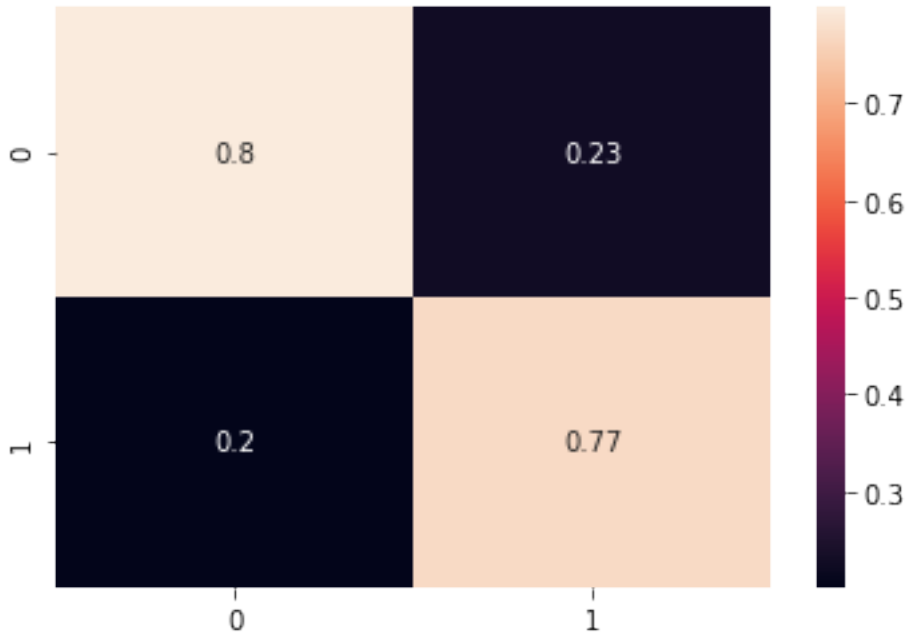

Figure 7. PyTorch neural network confusion matrix

Reflection

The most difficult aspect of training a machine learning model is finding training data. In this experiment I have used a synthetic dataset. Other than that the main thing that is difficult is that I have done this experiment in my spare time. So I have not been able to do dedicate enough time to all details and have to try and find a good enough effort, this is hard to do.

After oversampling the dataset to adjust for the imbalance the linear learner algorithm achieved an accuracy of 50%[10]. I wonder if this was because of overfitting and that the oversampling of the few failure cases got the model to memorise them? Or if with more careful selection of training and test dataset the model can improve its accuracy? I don't have any answer to this and that would require more investigation and experimentation with the linear leaner algorithm. But since I decided to use it as a baseline model I accept the current low accuracy of 0.15.

Improvement

The rather simple PyTorch model performs relative well given the data set. The thing that could improve the models accuracy is of course gathering more data. But I'm curious if being more strategic when generating the training and testing data, to ensure that there is a good distribution of the recorded machine failures, could improve the models accuracy also. In the second training with the linear learner model.

Since there are only eight features in the data I would also investigate some random forest model before deciding to start investigating some more advance deep neural network model.

Instead of oversampling to even the balance of the target classification and use accuracy as quality metrics. A different approach to investigate is trying a model without oversampling and using an other quality metric. Like the AUC ROC that blends together true positives and false positives into a single summary value.

---

[10] https://github.com/marcusholmgren/symmetrical-maintenance-broccoli/blob/94ba3dde0d7fde2af6fc814033f7b542cb15f4e1/linear-learner-baseline.ipynb