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, 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[1] 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[2] library to calculate accuracy, precision and recall. They have the formal formulas.

Accuracy is calculated as the weighted mean

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

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

---

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

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

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

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[3].

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 $\ddot{I}f$ = 10 Nm and no negative values.
• **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.

---

[3] http://archive.ics.uci.edu/ml/datasets/AI4I+2020+Predictive+Maintenance+Dataset
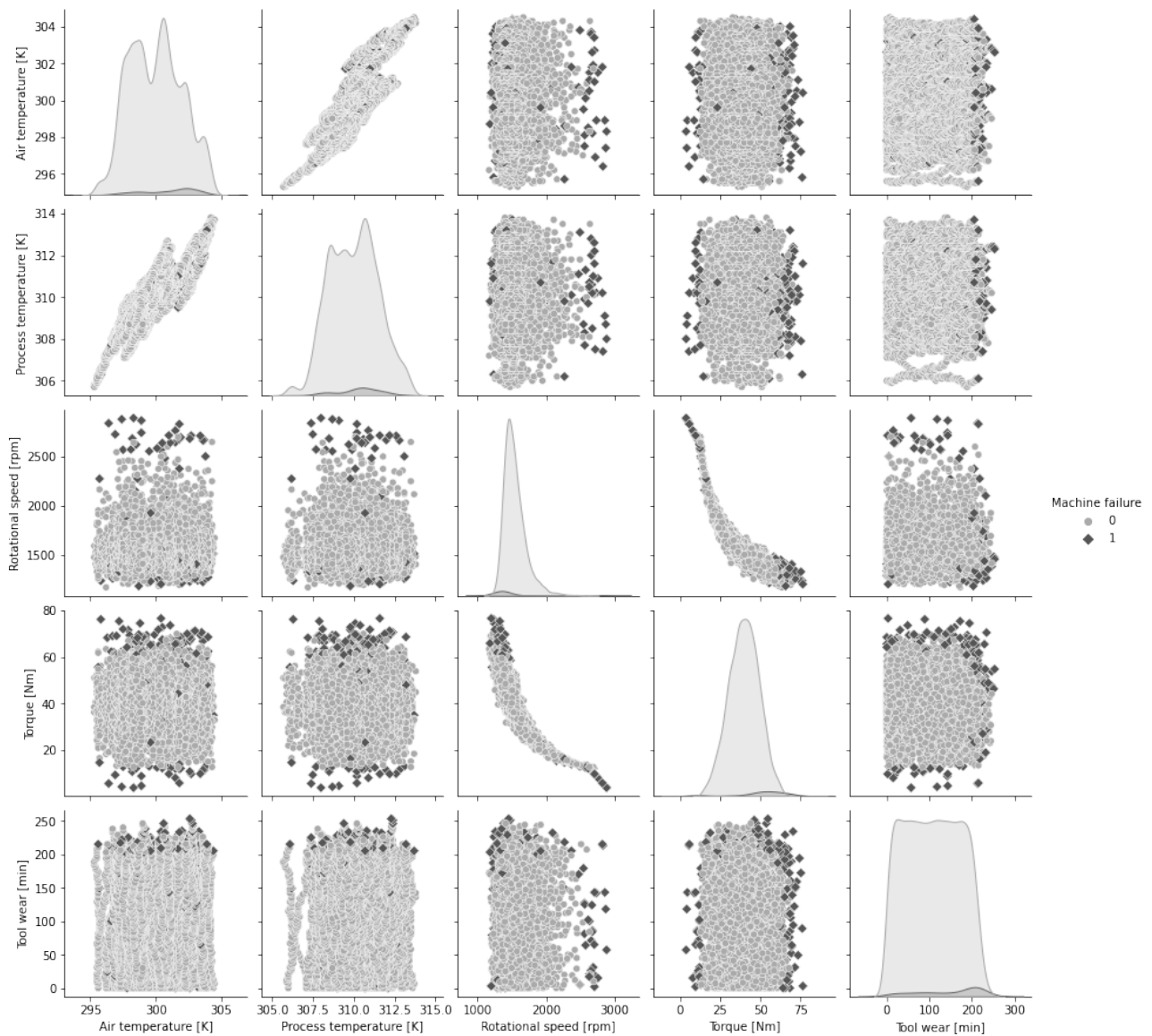
96.6% of the dataset entries are not machine failure and only 3.4% is machine failure. This makes the dataset highly unbalanced and the will be important when generating training and evaluation set to have fair distribution of failures.

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[4].

The two temperature features are recorded in Kelvin degrees.

---

[4] Understanding Boxplots, https://towardsdatascience.com/understanding-boxplots-5e2df7bcbd51
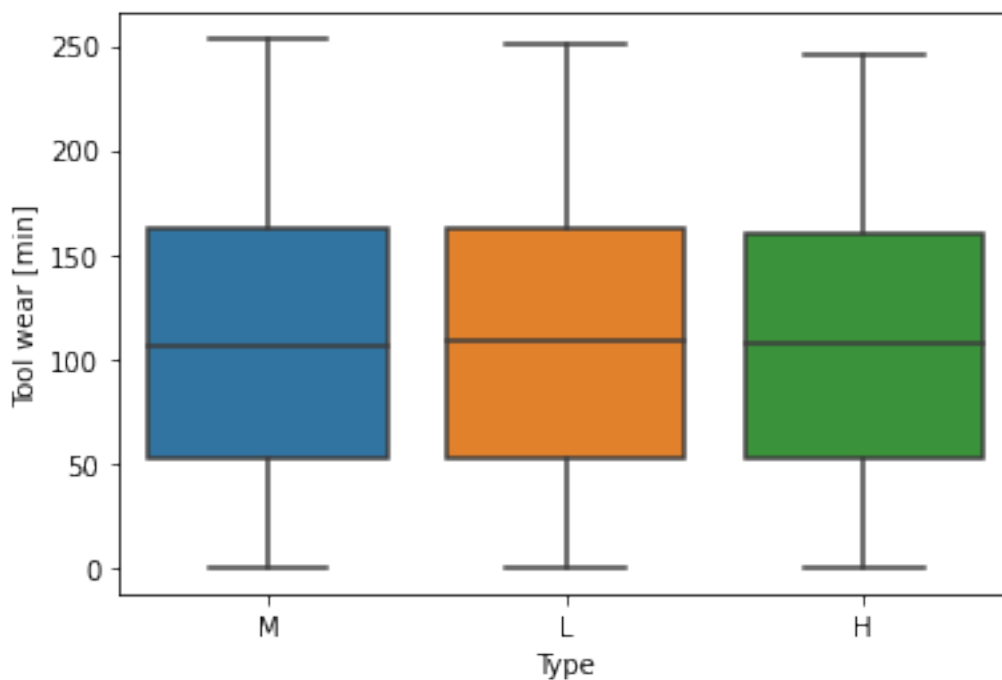
Temperature ranges
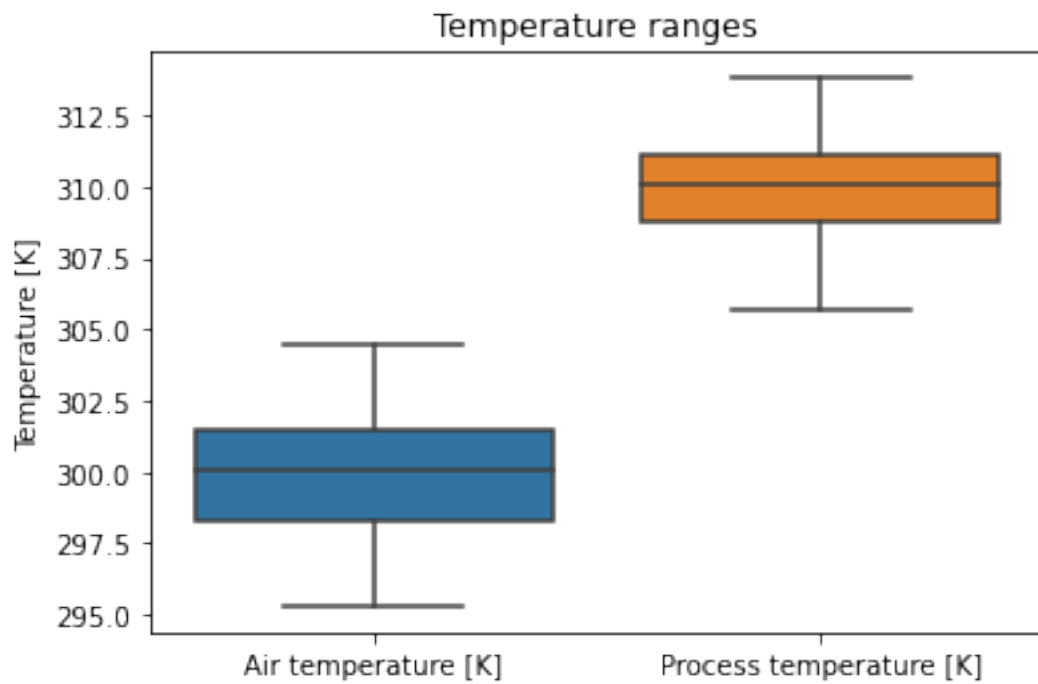
Rotational speed have a few products that have speeds above 1800 rpm. They are indicated as the dots above the Q3 in the box plot.
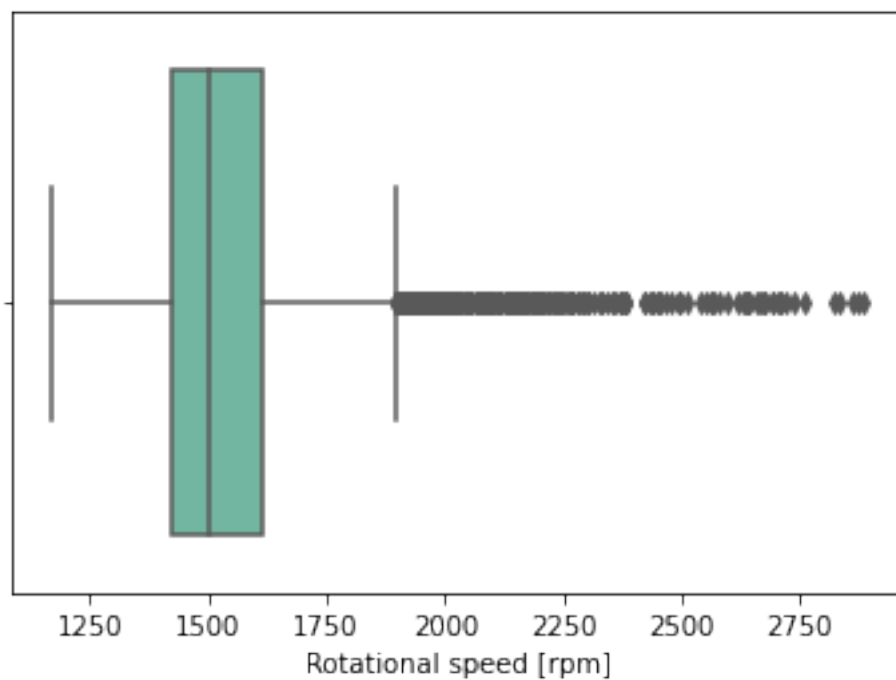
Torque in newton-meter, Nm. Also shows outliers in the low and high ranges of newton-meter
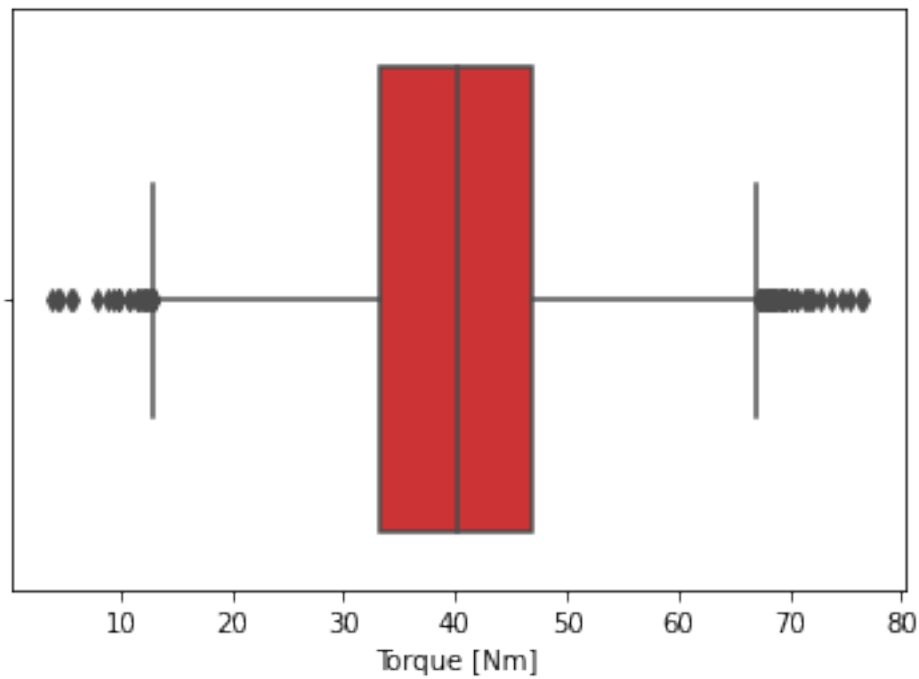
Tool wear in The quality variants H/M/L add 5/3/2 minutes of tool wear to the used tool in the process

---

Algorithms and Techniques

The neural network model I have decided to build is a three layer linear 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.

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%. After oversampling the dataset to adjust for the imbalance the linear learner algorithm achieved an accuracy of 50%[5].

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.
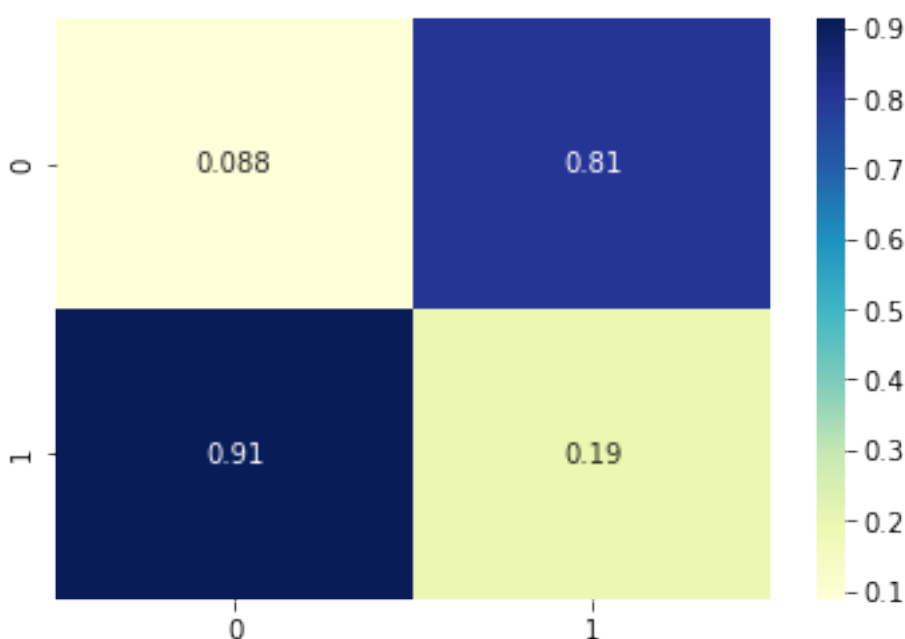
III. Methodology
Leaning on AWS SageMaker for machine learning

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

5 GitHub linear-learner-baseline.ipynb commit: 94ba3dde0d7fde2af6fc814033f7b - accuracy 50%

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.

The scikit-learn MinMaxScalar was used to normalise the five columns that have records that have entries that consist of big numbers.

Implementation

The implementation of the machine learning algorithms was done on AWS SageMake with the help of the existing examples of the linear learner and PyTorch using script mode.

One adjustment that I had to do with the Type features that was one-hot encoded in the feature engineering notebook. Was 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. 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.

IV. Results

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 reached 0.79 accuracy. Precision is 0.77 and recall: 0.81
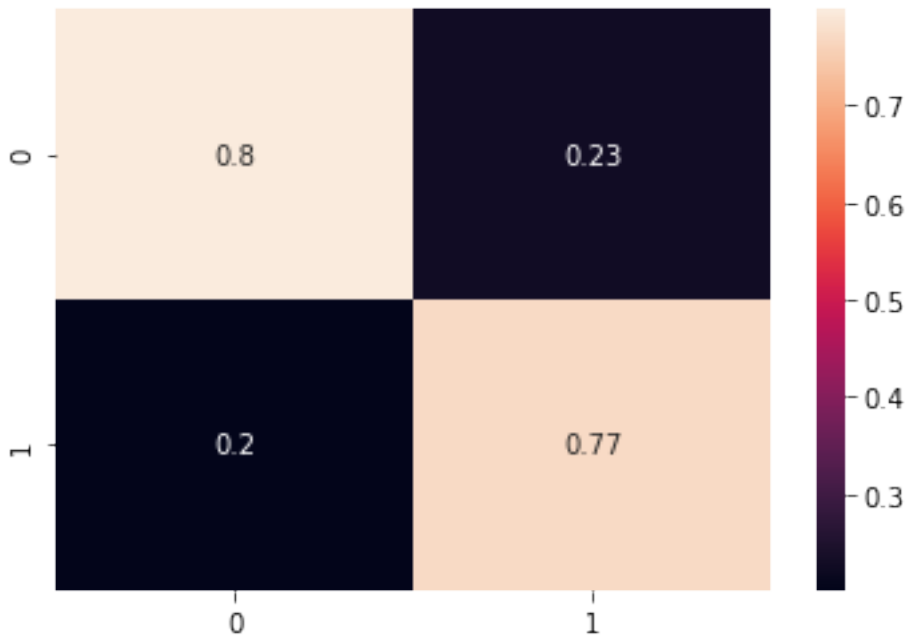
Reflection

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 the work fully rested and that makes it harder to reflect and think about clever approaches to solve problems.

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.

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