Fault Impact Analysis: Towards Service-Oriented Network Operation & Maintenance

Sandro Hunziker
Zindi Username: sandrohu

E-mail: sandrohunziker99@gmail.com

1. Data Preprocessing and Features

The first step in my work process was handling the missing values. For this this I filled the NaN values with the mean values of the corresponding CSV File. This approach is quiet naive and could be improved by filling the missing values with the hourly means.
Concerning the feature selection, if the fault occurred at time t, I once used the access success rate, ressource utilition rate, TA, bler, cqi, mcs, data rate at time $t-1$. Meaning the values one hour before the fault occurred. These features were Z-score normalized.

$$\hat{x}_{t-1} = \frac{x_{t-1} - \mu}{\sigma}$$

$\hat{x}_{t-1}$	Normalized value
x_{t-1}	Original value
μ	Mean of all observed values
σ	Standard deviation of all observed values

As a next set of features, I calculated the difference between the normalized features at time $t-1$ and $t-2$, so the new features are calculated as follows.

$$\Delta\hat{x} = \hat{x}_{t-1} - \hat{x}_{t-2}$$

As a last feature I encoded the hour before the fault occurred with a One-Hot-Encoder.
As the target variable I calculated the difference between the data rate

$$\Delta y = y_t - y_{t-1}$$

A negative Δy corresponded to a 1 everything else was a 0.

2. Training and Testing Data

I classified the data points into two categories. First I used all data points, even those were no fault occurred (*general data*). The reason for this was, that the model could learn general relationships and how the system behaves if there is no fault. For category, I used the data points where a fault occurred (*fine tune data*). These data points represent the actual goal of this competition.
For training I used once *general data* and oversampled the *fine tune data* eight times to 'show'

the model in relation more data points where a fault occurred.
For the testing, so the data were the code was tested and the leader board created, I made an error and calculated the difference between the features differently. I calculated

$$\Delta\hat{x} = \hat{x}_t - \hat{x}_{t-1}$$

instead of

$$\Delta\hat{x} = \hat{x}_{t-1} - \hat{x}_{t-2}$$

I think this mistake reduced the performance of my model quiet a bit.

3. Model

As a model I used Deep Neural Network (DNN) implemented in PyTorch. The DNN was built out of four layers were the first layer took the features described above as an input. Except for

	Inputs	Outputs	Activation Function
Layer 1	40	512	Relu
Layer 2	512	1024	Relu
Layer 3	1024	256	Relu
Layer 4	256	1	Sigmoid

Table 1. Architecture of the DNN

the last layer I used a dropout rate of 0.5 to reduce the tendency to overfitting. The learning rate started at 0.0001 and decayed with a factor of 0.96 after each epoch. The batch size was 64 and I trained the model for 50 epochs. The Adam optimizer was used to upgrade the weights of the model.
As a loss function I used a differentiable F1-score. This score was calculated as follows.

$$tp = \hat{y} * y$$

$$fp = (1 - \hat{y}) * y$$

$$fn = \hat{y} * (1 - y)$$

$$recall = tp/(tp + fn)$$

$$precision = tp/(tp + fp)$$

$$F1 = 2 * (precision * recall)/(precision + recall)$$

tp	True positive
fn	False negative
fp	False positive
$\hat{y}$	True value
y	Predicted value

So the goal was to minimize $1 - F1$.

4. Soft- and Hardware

The code was written in Python and my environment is attached on Github. The data preprocessing and feature extraction was performed on my local CPU and the DNN was trained on Google Colab on a GPU.

5. Conclusion

Regarding the naive handling of missing values and the mistake I made in creating the testing data set, I think the model performed still quiet good. By applying a more advanced data preprocessing and fixing the mistake I made with the feature extraction, the model could be improved further.