**Purdue**

# Application Failure Prediction

*Professor:* Saurabh Bagchi *Candidate:* Cheng Chen

**Abstract**

In modern days, application failure is a grave issue and needs to be addressed. The adverse effect of application failure is somewhat mitigated if we are able to accurately infer when application executions (referred to as "job") would fail beforehand. We build a Multi-Layer Perceptron (MLP) model to achieve this goal leveraging training data collected from Purdue ITaP's central computing cluster.

Codes are available at `https://github.com/ChengChen2020/failure-prediction`.

## 1 INTRODUCTION

With the growing scale of super-computing systems, scientists are now able to solve challenging computing problems in a matter of seconds which would take hundreds of years on a personal computer. However, with increasing scale (and complexity thereof) grows the probability of application failure, either due to hardware or software errors. Such application failures not only delay scientific progress, but also leads to a tremendous amount of wasted resources, both in terms of time and energy consumption. If we are able to predict when an application would fail due to a system error or due to software bugs, preventive mechanisms such as checkpointing can be initiated to save intermediate results, thereby, reducing the amount of wasted computation.

## 2 BACKGROUND

**Balanced Accuracy**. Balanced accuracy is defined as the average of positive class accuracy and the negative class accuracy. To achieve high balanced accuracy, we need to properly handle (extremely) unbalanced data.

## 3 ALGORITHMS

For *Model Complete*, we can use all the features available in the dataset to train the model. We build a MLP model using PyTorch. Figure 1 shows detail configurations of our proposed model.

We use Cross-Entropy loss with weights to handle imbalance of the data, and Adam optimizer with learning rate `5e-4`. We use StandardScaler from `sklearn` to standardize the raw data.

## 4 EXPERIMENTS

### 4.1 Data

The training data contains 20,000 jobs, which has about 8% failure data (this is referred to as the "positive class").

For each job, we have data about the resources the job uses (features) and whether the job succeeded or failed (label). The resources for which we have data are:

- Memory: Aggregate memory utilization for a job.
- Network: Aggregate network utilization for a job (in logarithmic scale).
- Local IO: Aggregate local disk IO for a job (in logarithmic scale).
- Network File System (NFS): Aggregate remote filesystem (NFS) IO for a job (in logarithmic scale).

Sample entries are shown in Figure 2.

```
----------------------------------------------------------------
        Layer (type)               Output Shape         Param #
================================================================
            Linear-1               [-1, 1, 128]             640
              ReLU-2               [-1, 1, 128]               0
            Linear-3               [-1, 1, 128]          16,512
              ReLU-4               [-1, 1, 128]               0
           Dropout-5               [-1, 1, 128]               0
            Linear-6                 [-1, 1, 2]             258
================================================================
Total params: 17,410
Trainable params: 17,410
Non-trainable params: 0
----------------------------------------------------------------
Input size (MB): 0.00
Forward/backward pass size (MB): 0.00
Params size (MB): 0.07
Estimated Total Size (MB): 0.07
----------------------------------------------------------------
```

Figure 1: Detail configurations of our proposed model. We have one input layer, one hidden layer and one output layer. Hidden dimension is set to be 128. Dropout probability is set to be 0.5 for the hidden layer.

| | job_id | memory_GB | network_log10_MBps | local_IO_log10_MBps | NFS_IO_log10_MBps | failed |
|---|---|---|---|---|---|---|
| 0 | jobID1634295 | 44.3904 | -1.0262 | 0.8033 | -3.0000 | 0 |
| 1 | jobID2033452 | 31.5839 | -1.4608 | -0.6080 | -2.9967 | 0 |
| 2 | jobID2068800 | 154.4610 | -0.5508 | -0.3637 | -3.0000 | 0 |
| 3 | jobID78826 | 71.8570 | 3.5701 | -0.1281 | 2.3040 | 0 |
| 4 | jobID2935014 | 30.0370 | 3.3094 | -0.9463 | -3.0000 | 0 |
| 5 | jobID349906 | 37.4598 | -1.4783 | 0.7389 | -3.0000 | 0 |
| 6 | jobID494334 | 7.4572 | 3.0176 | -0.5827 | -3.0000 | 0 |
| 7 | jobID1173855 | 9.7618 | -1.9703 | 0.2197 | -3.0000 | 0 |
| 8 | jobID1172335 | 28.9417 | 3.4735 | -0.9132 | -3.0000 | 0 |
| 9 | jobID1152712 | 8.2252 | 0.6708 | -0.1757 | -1.2731 | 0 |

Figure 2: Sample entries from the training dataset.

### 4.2 Experiments and Results

We extract 5% training data as validation data and train for 300 epochs. We vary the hyperparameters and network structures to find the best combination.

The highest balanced accuracy on training data is 80.838%. The best balanced accuracy we achieve on private leaderboard is 78.802%.

## 5 CONCLUSIONS AND FUTURE WORK

The balanced accuracy we achieved is not very high. We suspect it is due to lack of insight and preprocessing for the dataset. For further improvements, we should be able to use certain intuition or prior knowledge to clean the dataset, possibly generating new features. Furthermore, we can use other ways (undersampling, oversampling and generating synthetic data) to handle this imbalanced dataset.

## 6 ACKNOWLEDGEMENTS

Thanks for Prof. Saurabh Bagchi's time and consideration.