

# Machine Learning Engineer Nano-Degree

## Capstone Project

Payam Mohajeri  
19 January 2020

---

### Domain Background

Virtualisation of application environments and containerisation of applications is a trending topic these days. Different solutions are provided to virtualise an operating system or containerise an application; and implement a cluster of nodes for managing or maintaining applications. One of the common topics within these clustering solutions is about distributing the load based on the available infrastructure resources, for example processing power and memory. For distributing the load, a component needs to keep monitoring the environment and provide the information to a scheduler to maintain the environment and organise future application deployments. The scheduler has an important role for improving and enhancing the environment or applications stability. One of the key metrics that scheduler has to consider is CPU utilisation. This project is going to look into providing a model for predicting the CPU utilisation.

---

### Problem Statement

Improving the stability and performance of distributed applications across the globe is a challenging topic for enterprises. An application environment can get unstable when there is a heavy load on specific applications or nodes. Users often face high response time and low performance in case of any environment instability and this can have a huge impact on the branding or business of targeted enterprises and organisations. Today's competitive market requires organisations to provide stable and well performing application environment to their customers, but providing such stability based on a limited visibility on utilisation of infrastructure resources is very challenging. On this project I'm focusing on improving the visibility of system metrics by providing a model for predicting the CPU utilisation based on the previous infrastructure monitoring records. Besides improving the stability, saving energy and costs are also other use cases on such model.

---

### Datasets and Inputs

Following provided dataset from Burn CPU Burn competition on Kaggle will be used:  
<https://www.kaggle.com/c/model-t4/data>

The goal is to predict the load on the CPUs in a cluster of servers based on the applications behaviour running on these servers. There are two CPUs per each server and in total there are seven servers in a cluster.

“The dataset consists of a set of variables that were measured over about a one month period. Measurements were taken in one minute intervals and on each server. Measurements are usually the average or sum over that one minute interval. For instance the number of packets received, the average number of IO operations, etc.”

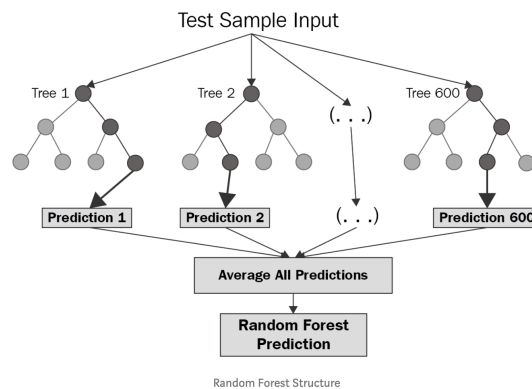
This dataset is from a real cluster that is used to control train traffic in a geographical area spanning several cities. These fields are available on provided CSV files:

- sample time: the date and time the data was sampled.
- m\_id: the ID of the server the data was sampled at.
- appxxxx: data about specific application.
- pagexxx: data on memory usage of the server.
- syst\_xxx: data on page fault rate, number of processes, etc.

- state\_xxx: data on the state the system is in.
- io\_xxx: data about general IO usage, (file IO, direct IO).
- tcp\_xxx: data on incoming and outgoing TCP traffic.
- lxxx, ewxxx: data on incoming and outgoing network traffic.
- cpu\_01\_busy: the variable we are trying to predict.

## Solution Statement

The solution is to collect the system related metrics like the usage of memory, I/O, network traffic and train a model which can link the recorded data to the CPU utilisation and can provide the estimation in case the system behaviour is similar. This trained model for our prediction needs to be based on a supervised learning. Different regression algorithm can be used to train the model. As a first idea I would like to try on Random Forests and Extra Trees methods since the data set has so many different fields and it might be better to try ensemble learning methods.



Random Forest Regressor:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html>

Extra Trees Regressor:

<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html>

## Benchmark Model

Based on the information provided on <https://www.kaggle.com/c/model-t4/leaderboard> the benchmark model is a Two Variable Random Forest model which have been evaluated and scored as 11.18079. For our case here, I think Two Variable model is a very basic model and I'm looking to get better results since there are more than 2 variables to be considered according to our datasets.

## Evaluation Metrics

Based on the information provided on <https://www.kaggle.com/c/model-t4/data> the evaluation metric is the Root Mean Square Error:

$$S = \sqrt{\frac{1}{n} \sum_{i=1}^n (\hat{l}_i - l_i)^2}$$

\*  $\hat{l}$  is the predicted load and  $l$  the actual load.

“For every line in the test set, submission files should contain two columns: Id and Prediction. Prediction is a real number in the range of 0 to 100 predicting how busy CPU 1 was in the given machine on the given date.”

---

## Analysis

To apply the basic analysis we first would need to download and then explore the data. The train has 178780 and each entry has 89 attributes including the sample time and different related system metrics.

```
In [3]: train.head()
```

```
Out[3]:
```

	sample_time	m_id	syst_direct_ipo_rate	syst_buffered_ipo_rate	syst_page_fault_rate	syst_page_read_ipo_rate	syst_process_count	syst_other_states	page_pa
0	2010-11-24 00:01:00	a	80.48	1261.97	15.55	2.10	271	12	
1	2010-11-24 00:01:00	b	73.80	624.38	6.43	0.45	317	10	
2	2010-11-24 00:01:00	c	40.57	466.18	6.40	0.45	258	22	
3	2010-11-24 00:01:00	d	68.75	495.10	6.50	0.45	224	10	
4	2010-11-24 00:01:00	e	47.45	436.65	6.42	0.45	253	10	

5 rows × 89 columns

```
In [4]: train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 178780 entries, 0 to 178779
Data columns (total 89 columns):
sample_time          178780 non-null object
m_id                 178780 non-null object
syst_direct_ipo_rate 178780 non-null float64
syst_buffered_ipo_rate 178780 non-null float64
syst_page_fault_rate 178780 non-null float64
syst_page_read_ipo_rate 178780 non-null float64
syst_process_count    178780 non-null int64
syst_other_states     178780 non-null int64
page_page_write_ipo_rate 178780 non-null float64
page_global_valid_fault_rate 178780 non-null float64
page_free_list_size   178780 non-null int64
page_modified_list_size 178780 non-null int64
io_mailbox_write_rate 178780 non-null float64
io_split_transfer_rate 178780 non-null float64
io_file_open_rate     178780 non-null float64
io_logical_name_trans 178780 non-null float64
io_page_reads         178780 non-null float64
io_page_writes        178780 non-null float64
page_free_list_faults 178780 non-null float64
page_modified_list_faults 178780 non-null float64
page_demand_zero_faults 178780 non-null float64
state_compute         178780 non-null int64
state_mwait           178780 non-null int64
```

---

## Implementation

For implementation I've used Python programming language, panda and scikit libraries; as well as the RandomForest and ExtraTrees algorithms. At the beginning after loading the train and test data, entries had to be normalised on date, time and on the ID of the server the data was sampled at:

```

23 def splitSampleTime(data):
24
25     sub = pd.DataFrame(data.sample_time.str.split(' ').tolist(), columns="date time".split())
26     date = pd.DataFrame(sub.date.str.split('-').tolist(), columns="year month day".split())
27     time = pd.DataFrame(sub.time.str.split(':').tolist(), columns="hour minute second".split())
28
29     data['year'] = date['year']
30     data['month'] = date['month']
31     data['day'] = date['day']
32
33     data['hour'] = time['hour']
34     data['minute'] = time['minute']
35
36     return data
37
38 def mapID(data):
39
40     coded = {'a': 1, 'b': 2, 'c': 3, 'd': 4, 'e': 5, 'f': 6, 'g': 7}
41     data['m_id'] = data['m_id'].astype(str).map(coded)
42
43     return data

```

Then the following classification algorithms could be called:

```

2 import pandas as pd
3 from sklearn.ensemble import RandomForestRegressor
4 from sklearn.ensemble import ExtraTreesRegressor
5
6 def createRandomForest():
7     clf = RandomForestRegressor(n_estimators=50)
8     return clf
9
10 def createExtraTree():
11     clf = ExtraTreesRegressor(n_estimators=50)
12     return clf
13
14 def classify(clf, train, features, target, test, filePath):
15
16     clf.fit(train[features], train[target])
17
18     with open(filePath, "w") as outfile:
19         outfile.write("Id,Prediction\n")
20         for index, value in enumerate(list(clf.predict(test[features]))):
21             outfile.write("%s,%s\n" % (test['Id'].loc[index], value))
22

```

And here is the main function:

```

45 train=pd.read_csv("./input/train.csv")
46 test=pd.read_csv("./input/test.csv")
47
48 train = splitSampleTime(train)
49 train = mapID(train)
50
51 test = splitSampleTime(test)
52 test = mapID(test)
53
54 target = 'cpu_01_busy'
55 features = [col for col in train.columns if col not in ['cpu_01_busy', 'sample_time']]
56
57 clf = createExtraTree()
58 classify(clf, train, features, target, test, "./output/extratree.csv")
59
60 clf = createRandomForest()
61 classify(clf, train, features, target, test, "./output/randomforest.csv")

```

---

## Results

Following table shows the results of above Random Forest and Extra Tree regressors and also compares the outcome with the Two Variable Random Forest model which is used as a benchmark model:

Method	Evaluation Score
Extra Tree	3.39807
Random Forest	3.78130
Two Variable Random Forest	11.18079
All Zero	27.40211

---

## Conclusion

Both Extra Tree and Random Forest have been performed better than the benchmark model. Extra Tree is performing slightly better than Random Forest on the problem specifics and data set here. It seems that the train and test data are quite clean and we don't have much noise there, that's why the Random Forest method appears to be biased on local optimum while Extra Randomised Trees is trying to jump out of local optimum and trying to reach to global optimum.