Definition

Project Overview

The idea and dataset of this project is collected from a Kaggle competition named Burn CPU Burn which is a part of machine learning workshop given in InTraffic. The goal of this project is to predict the load on CPU. The CPUs are running in a cluster of server and hosting various applications and processes. According to the competition page, there are two CPUs in each server and seven servers in the cluster. We need to predict the load of the second CPU. The dataset is a collection of variable collected over a period of a month and the measurements are taken in one month intervals and on each server. This is usually the average or sum over that one month interval. Total Timeline Tracking Tool has been used to collect data.

Problem Statement

This dataset is a large collection of information regarding the load of CPU. Based on the previous behavior, we will predict the future load of some of those CPUs. As the load is continuous numeric value, the problem is actually to design a regression model which can predict future CPU load information.

Metrics

Based on the characteristics of the problem, Root Mean Squared Error (RMSE) will be used as metrics. The RMSE score is given by the following formula.

$$\mathrm{RMSE} = \sqrt{\frac{1}{n}\sum_{j=1}^{n}(y_j - \hat{y}_j)^2}$$

Equation 1 : Root Mean Squared Error

The lower the error the better the model performance is. We will compare our prediction using the leaderboard of the related contest from here.

Analysis

Data Exploration

There are *88* features and *1* target column in the dataset. Out of *88* features there is *1* timestamp column, *1* categorical column and the rest of those are numerical columns. As the target data is provided, we can describe the problem as supervised learning problem. A summary of the data set is given below.

Feature Name	Data Type	Description
sample_time	Timestamp	The date and time the data was sampled.
m_id	Categorical	The ID of the server the data was sampled at.
appxxxx	Numerical	Data about specific application.
pagexxx	Numerical	Data on memory usage of the server.
syst_xxx	Numerical	Data on page fault rate, number of processes, etc
state_xxx	Numerical	Data on the state the system is in.
io_xxx	Numerical	Data about general IO usage
tcp_xxx	Numerical	Data on incoming and outgoing TCP traffic.
Llxxx, ewxxx	Numerical	Data on incoming and outgoing network traffic.
cpu_01_busy	Numerical	The variable we are trying to predict.

Table 1: Feature Description

There are total *178780* samples in the dataset. We are using python *pandas* library to analyze this dataset. A quick overview can be generated using *describe* method. A sample statistic about the dataset can be viewed from here. *m_id* column is categorical in this dataset. In relevant section it will be encoded to numerical value. From the dataset, this is clear, that the features are not normalized. However, as we will use tree based method, we have decided not to normalize the dataset to keep its features intact.

We have also done a correlation measure of all features with target data (*cpu_01_busy*). The correlation matrix can be seen from here. In the correlation matrix, we can see, there are some features for which the correlation is *NaN* because of all zero values in those features. In data preprocessing section, we will remove those features. Figure *1* shows the correlation of each feature with *cpu_01_busy*.

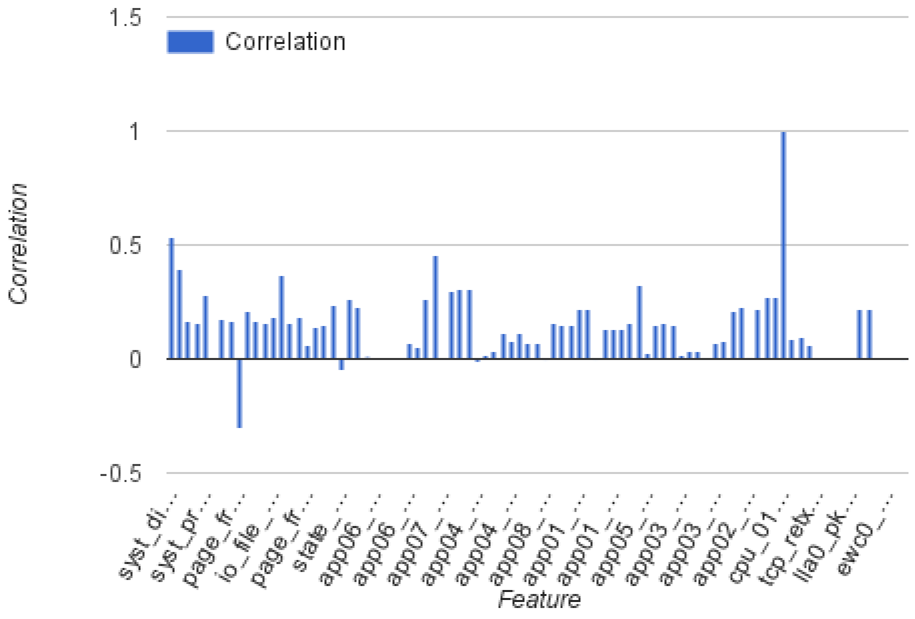

Figure 1: Feature correlation with cpu_01_busy

Exploratory Visualization

Scatter matrix plot is an important aspect of data visualization to determine relationship among various features. We have used OpenRefine to discover various relationships in this dataset. A screenshot of that plot is given below.

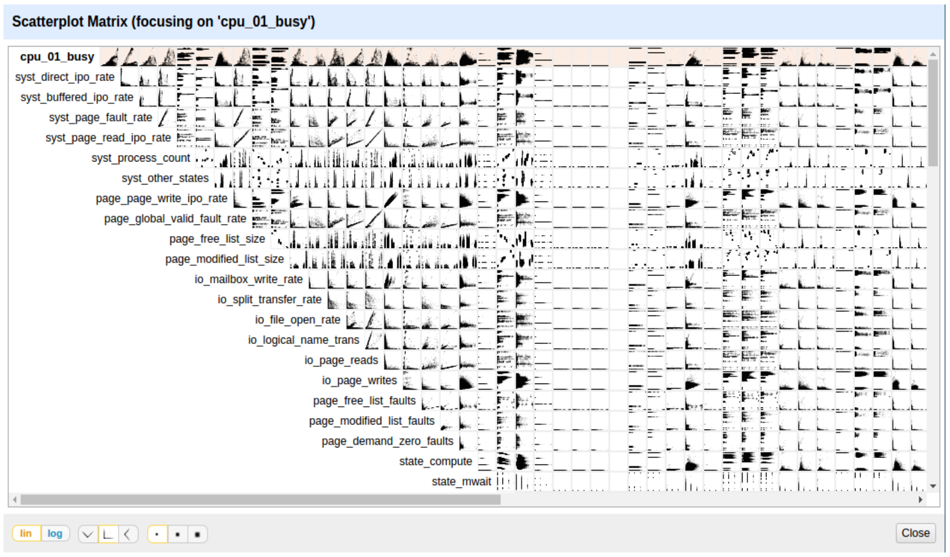

Figure 2: Scatter matrix plot of features

As we can see, the *sample_time* feature is not directly usable for our purpose. That's why we have extracted *day*, *hour* and *day of week* data from *sample_time* feature. The relationship of mean of *cpu_01_busy* with *hour*, *day* and *day of week* can be visualized using the following figures.

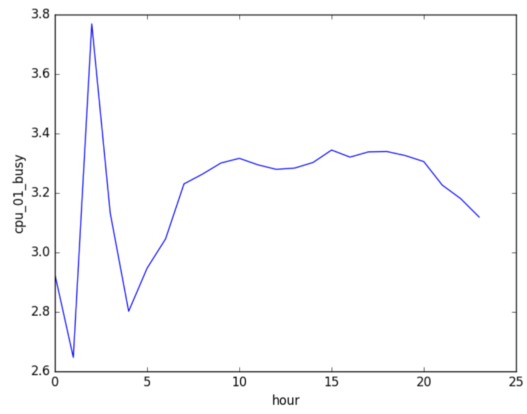

Figure 3: Mean cpu_01_busy for each hour

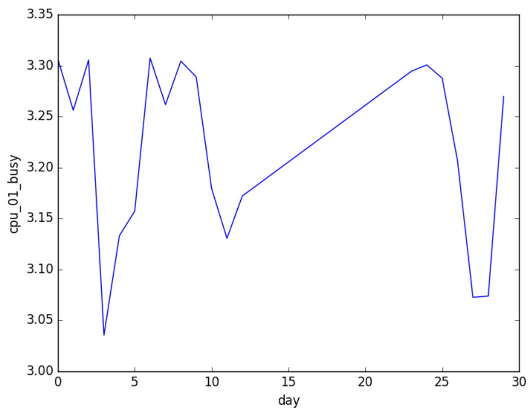

Figure 4: Mean cpu_01_busy for each day of a month

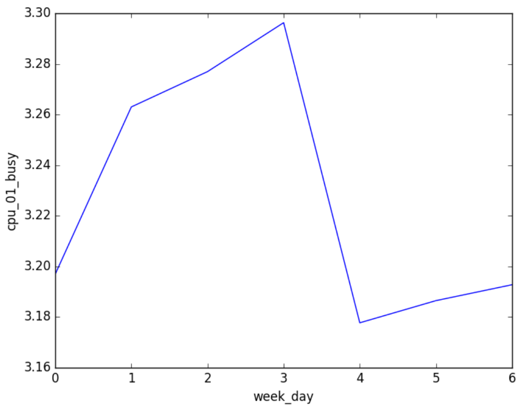

Figure 5: Mean cpu_01_busy for each day of week

From these figures, we can see, *hour*, *day* and *day of week* data have a good correlation with our target feature. We will use these in building our final model (to predict *cpu_01_busy*).

Algorithms and Techniques

As the problem is a supervised regression, we will use multiple algorithms to generate the desired model for prediction. We have decided to use three tree based algorithms for this purpose. Grid search and cross validation strategy will be used to find the best parameter set for all of the algorithms. Now we will provide a brief description of each algorithm along with the related parameters.

Decision Tree

Decision tree is a tree like structure which uses its branches to illustrate every possible outcome for a decision or sequence of decisions. It can be used for both classification and regression problem. Decision tree goes through every sample of dataset and makes simple rules to predict outcome for previously unseen data. This is easy to understand and interpret but calculation becomes harder if the unseen data has more variation than training set.

Parameters

1. max_depth : The maximum depth of the tree

Random Forest Regressor

This is a collection of decision tree for regression purpose using the various sub sample of data set and uses the average output of each decision tree to tackle the problem of over fitting and improve accuracy. Sample may or may not be drawn with replacement based on parameter setting.

Parameters

1. n_estimators : The number of trees in the forest.
2. max_depth : The maximum depth of the tree.
3. n_jobs : The number of jobs to run in parallel.

Extra Tree Regressor

This is a collection of randomized decision trees and works on various sub samples of the dataset along with the averaging to improve accuracy and control over fitting.

Parameters

1. n_estimators : The number of trees in the forest.
2. max_depth : The maximum depth of the tree.
3. n_jobs : The number of jobs to run in parallel.

Benchmark

We will compare our prediction using the leaderboard of the related contest from here. Currently the top score (RMSE) is *3.27691*. We will compare our model against this score and also submit the prediction to relevant competition to compare our model against public and private dataset.

Methodology

Data Preprocessing

There is no missing data in the data set, that's why, no preprocessing was necessary to fill the empty values. However the is a column named *sample_time*. We have extracted *day, hour* and *day of week* information from that feature before removing it from the data set. Again, we have calculated the correlation of each feature with the target variable and found some feature as *NaN*. That means some features have constant values and these features are also removed for the dataset.

Implementation

There are multiple steps in our implementation. At first we have extracted *day*, *hour* and *day of week* from *sample_time* and then after removing it we have encoded the categorical values to numerical value using the *LabelEncoder* module of python sklearn. The next step is to calculated correlation of each feature with *cpu_01_busy*. If any correlation is zero, we have removed the related feature from the dataset.

The next step is to implement a grid search using *n_estimators* and *max_depth* to find the best performing parameter. In case of Decision Tree we have only used *max_depth* feature. The grid search part is implemented using *GridSearchCV* package of python scikit learn library. For grid search, the values of *n_estimators* are *[200, 250, and 300]* and *max_depth* are [40, 50 and 60].

Before using grid search the training dataset has been divided using 80-20 rule. 80% data has been used for training and 20% data has been used for validation purpose. We have used a 5 fold cross validation to generalize model accuracy.

n_estimators	max_depth	Extra Tree	Random Forest	Decision Tree
200	40	2.572	2.709	3.832
250	40	2.571	2.707	
300	40	2.57	2.71	
200	50	2.572	2.714	3.847
250	50	2.573	2.71	
300	50	2.572	2.708	
200	60	2.577	2.709	3.839
250	60	2.573	2.709	
300	60	2.568	2.706	

Table 2 : Cross Validation Result

Based on Table 2 we can see *n_estimators* = 300 and *max_depth* = 60 are the best parameters so far. We could have tried more parameter variance but it took around *6* hours to complete the grid search process. That's why we have stopped the experiment in that stage.

Using these parameters we have generated three separate models using Decision Tree, Random Forest and Extra Tree. Those models have been used to predict the testing dataset provided by Kaggle. After that we have submitted those predictions to the corresponding competition. The result of the submission is given below.

	Pubic Score	**Private Score**
Extra Tree	3.2744	3.26851
Random Forest	3.86558	3.77884
Decision Tree	6.99619	7.16176

Table 3 : Test data prediction result in Kaggle

Refinement

In initial stage, we have used [*10* and *20*] as *the max_depth* for both Random Forest and Extra Tree. However, the result was not that significant for our purpose. To achieve better accuracy, we have depth of tree in both models. Additional depth has increased the model building time significantly (Around 3 hours each), however, this is an important investment of time to increase the model accuracy. As we know, the public score is generated using a small portion (15%) of test data and private score is generated based on the reaming samples. In Table 3, we can see the Extra Tree has scored *3.26851* while the #1 score is *3.27691*. We can see, our solution has outperformed the best available solution.

Results

Model Evaluation and Validation

Figure 6 shows the performance of each model using public, private and average training score. We can see the training and testing score difference for extra tree is less than random forest and decision tree. This is the indication that, extra tree is having less variance other than the two models.

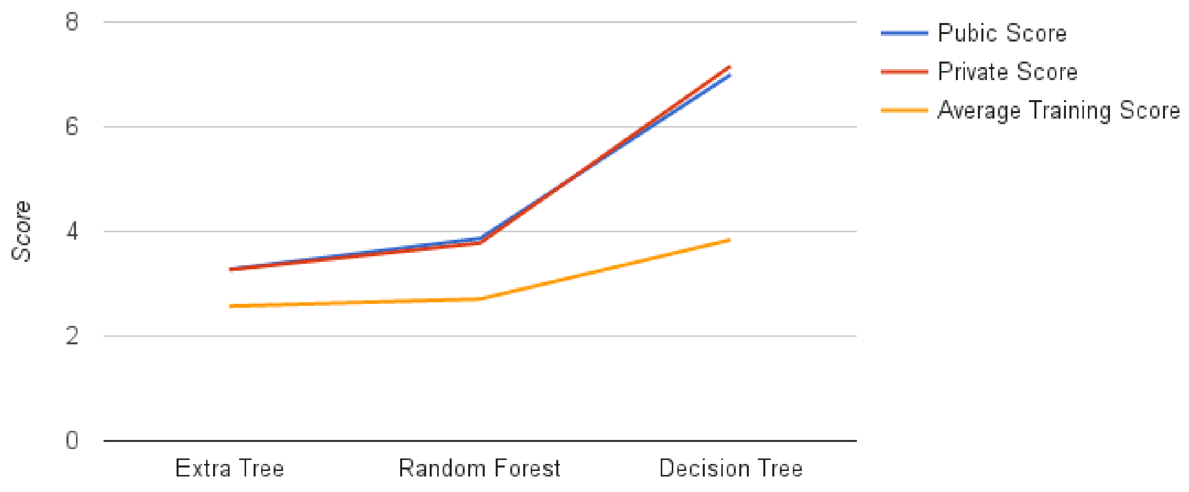

Figure 6: Public, private and average training score of each algorithms

Figure 6 also shows the prediction accuracy of each model using best set of parameters. We can see the effectiveness of Extra Tree model from this figure.

Justification

From the competition leaderboard, we can see, the best performing model has scored 3.27691. From Table 3, we can see that our best model has outperformed the #1 model of the leader board. Comparing these results, we can say that we have found a better performing model to address the problem.

Conclusion

Free-Form Visualization

A figure (Figure 6) has been drawn showing the performance of algorithms in this project. As discussed earlier, we have used RMSE metrics to evaluate these performances. This figure provides a visualization of the best model produced by this project.

Reflection

We have started the project with some statistical analysis to determine some unimportant feature so that we can remove those to make model creation process easier. In this step we have used correlation analysis to detect some features to remove. The next step is to extract some important features from existing features. This step has created *hour*, *day* and *day of week* feature along with the numerical representation of a categorical feature. In the next step we have divided the training set using 80-20 rule to use 5 fold cross validation using grid search.

In this project we have used three tree based algorithms to find the best performing model. The figure provided in previous section shows the best performing model. We have submitted our prediction to main Kaggle contest to evaluate the performance of test dataset and it has showed significant improvement compared to the best model of the competition.

Improvement

The best way to improve the accuracy of the proposed model is to identify new features from the existing dataset. Though we haven't explicitly evaluate inter feature interaction, it would be great to evaluate two and three way interaction among the features. Besides, we have made limited choice in selecting grid search parameters to reduce model generation time. We may try increasing new parameter to tweak to improve model accuracy. Besides these algorithms, we could try neural network to discover more complex relationship and parameter to improve model accuracy drastically. As the parameter selection process of neural network is trickier, we have kept it in future scope.