EP2420: Estimating Service Metrics from Device Measurements

Ruihan Zhang

December 5, 2022

Project Overview

This project focuses on using machine-learning methods for predicting service-level metrics in a real-world network system, which is either a Video-on-demand system (VOD) or a Key-Value store (KV) dataset.

Our main goal is generally a regression problem - how to map the infrastructure measurements X to predictions of service-level metrics Y. We will look in detail at 4 tasks, the different methods of preprocessing and various models, and how each of these variations will impact the result in terms of prediction accuracy.

Background

NMAE

Normalized Mean Absolute Error(NMAE) is our major criterion for the evaluation of a particular model. It is defined as the following:

$$NMAE = \frac{1}{\bar{y}}(\frac{1}{m}\sum_{j=1}^{m}|y_j - \hat{y}_j|)$$

,whereby $\hat{y}_j$ is the our model prediciton for the measured service metric y_j, and $\bar{y}$ is the mean value of y_j in the test set. Sum up all the absolute value of all the measured errors and devide by the number of samples tested we have the *Mean Absolute Error*(MAE). And we normalize such value by deviding the MAE by the average of the observations, so that we can compare between different datasets.

Machine Learning

Machine learning(ML) is an active field of data science. They are targeting complex problems which usually involves dealing with large amount of data. By traditional means, algorithms are carefully designed by engineers to deal with specific tasks. But machine learning adopts the idea of "learning". By feeding more and more training data into the model, the comptuter seems to understand the data better, and gives more accurate predicitons.

The models we're going to leverage in this project are all supervised learning methods, where each data point contains features and an associated label that we want to map into.

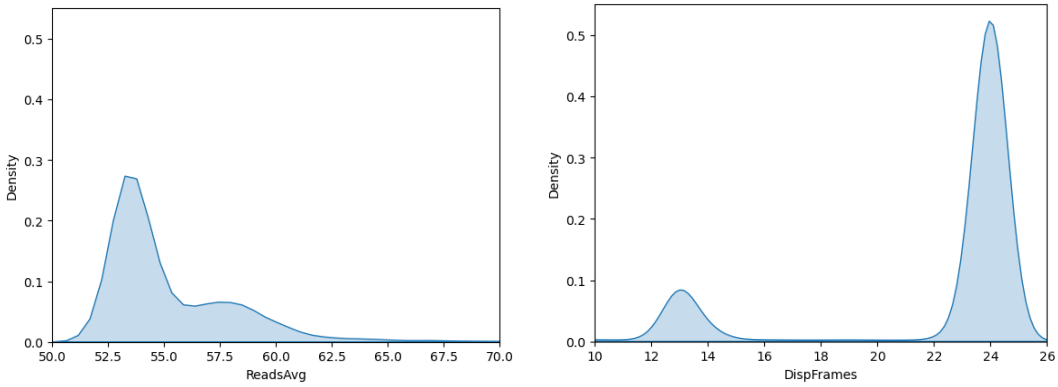
Neural Networks are probably the most popular machine learning methods nowadays. We're going to study with then simplest architecture of them - a fully connected feedforward artificial neural network, or a multilayer perceptron in the traditional ML sense.

Random Forest is another powerful tool for us to use. It's an ensemble method built on the classic ML algorithm - Decition Tree.

Task I: Data Description

In this first task of the project, utilizing NumPy and Pandas modules in python, I played around with the two given datasets - "KV" and "VoD" in my python environment. And have found the following charateristics of the dataset:

1. The KV dataset consists of 9500 rows of samples and 1725 columns of features as input X, while the VoD dataset has 17500 samples of 1672 features.
2. The target feature in KV is "ReadsAvg", with mean value of 55.2 and standard deviation of 3.11. The target feature in VoD is "DispFrames", with mean value of 22.0 and standard deviation of 4.32.
3. The density plot of these target features is shown in Figure 1.



(a) in KV service (b) in VoD service

Figure 1: Desity plot of the target features in KV and VoD datasets: The "ReadsAvg" which measures the total input of the network is mostly at its baseline. And at its peak time, it goes up to 6 times larger than that, but only for a short time. The "DispFrames" concentrates at two points: 13 fps and 24 fps, as a result of the system's normal working state and the throttling state.

Then we tried to pre-process the data using the three methods below on the columns first and then on the sample rows, which produces produces six design matrices X_1, X_2, ..., X_6 as numpy arrays. How these different methods of preprocessing would impact our modeling will be further discussed in Task III.

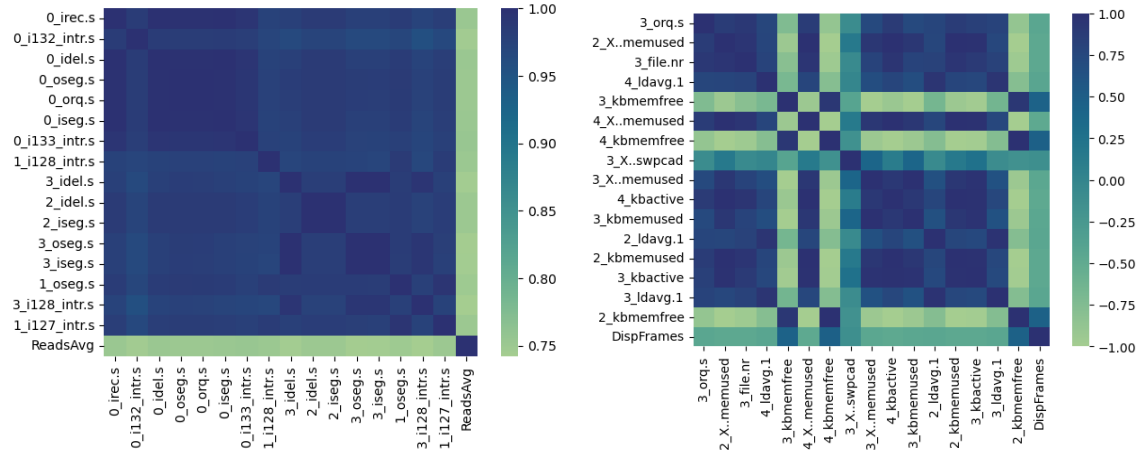
1. l^2 Normalization: linearly scale the values of each feature column (or sample row) so that its l2 -norm becomes 1. *(sklearn.preprocessing.normalize)*
2. Restriction to interval: linearly map the values of each feature column (or sample row) so that all they lie within the interval $[0, 1]$. *(sklearn.preprocessing.minmax_scale)*
3. Standardization: linearly map the values of each feature column (or sample row) so that they have 0 mean and a variance of 1. *(sklearn.preprocessing.scale)*

Finally, we use an ExtraTreeRegressor to fit our observed data trace, by doing so the model generates an importance metrics for each input feature. We will select only the top 16 features for future training. This is because dataset has too many features as its columns, but not all of them is correlated to our target feature, so we want to reduce the "noise" as much as possible, and focus on the most useful data.

Figure 2 are the heatmap plots for visualizing the correlation among the selected target features. We can see that in KV service 2(a), we have a relatively even heatmap, which means every feature we show on

the graph have a high correlation with not only our target feature but also the rest of the features. So they contain basically the same kind of message.

But in VoD 2(b), some features like the "2_kbmemfree" and "3_kbmemfree""4_kbmemfree" are highly correlated with each other but not with the other features, which is telling us that maybe two or more groups of factors are contributing to our target feature at the same time.



(a) in KV service

(b) in VoD service

Figure 2: Desity plot of the target features in KV and VoD datasets. The cells show the correlation between two features or between a feature and the target Y.

Task II: Estimating Service Metrics from Device Statistics

In this part, we're going to roughly train different types of models and get our first result of estimating the service metrics from device statistics.

Model Training

Three models are trained and tested: Linear regression, random forest, and neural network. And a naive method for comparing results is used, which is just taking the mean y values of the training set and use it to predict all values in the test set.

Our dataset is split into two parts: the training set for computing the model and the test set for evaluating the accuracy of the model. 30% of the observation data are randomly chosen to form a test set, and the remaining 70 will be our training set. And only 16 selected features (produced in the previous task) are used for training and testing.

Some of the parameters that I chose for random forest are shown in Table 1, and for neural networks are shown in Table 2.

Dataset	n_estimators	criterion	max_depth	min_samples_split
KV	100	squared error	any	2
VoD	100	squared error	any	2

Table 1: Random Forest parameters

Dataset	type	hidden_layer_sizes	max_iter	learning_rate_init	solver
KV	Regressor	(10,10,10,10)	5000	0.0001	adam
VoD	Classifier	(16,16,16)	5000	0.0001	adam

Table 2: Neural Network parameters

The *Normalized Mean Absolute Error (NMAE)* of the three models on two dataset is given in the Table 3. As we can see from the results, the random forest method works best in both datasets, out performing neural network and linear regression.

And in KV, the neural network method is giving worse prediction than the linear model. That's probably because we didn't properly preprocess our dataset, so feature values could vary largely in magnitude. Moreover, the hyperparameters for training the model are not optimized to the best performance. Even though we have trained for thousands of epochs, they are still outperformed by random forests.

However, all of them are at least better than the naive method, which means they provide us with more information than jus the average level of the measured metrics.

Dataset	naive method	linear regression	random forest	neural network
KV	0.0417	0.0217	0.0201	0.0260
VoD	0.146	0.121	0.0626	0.0838

Table 3: Model accuracy comparison

Predicting

Let's now see our Random Forest regression result in detail:

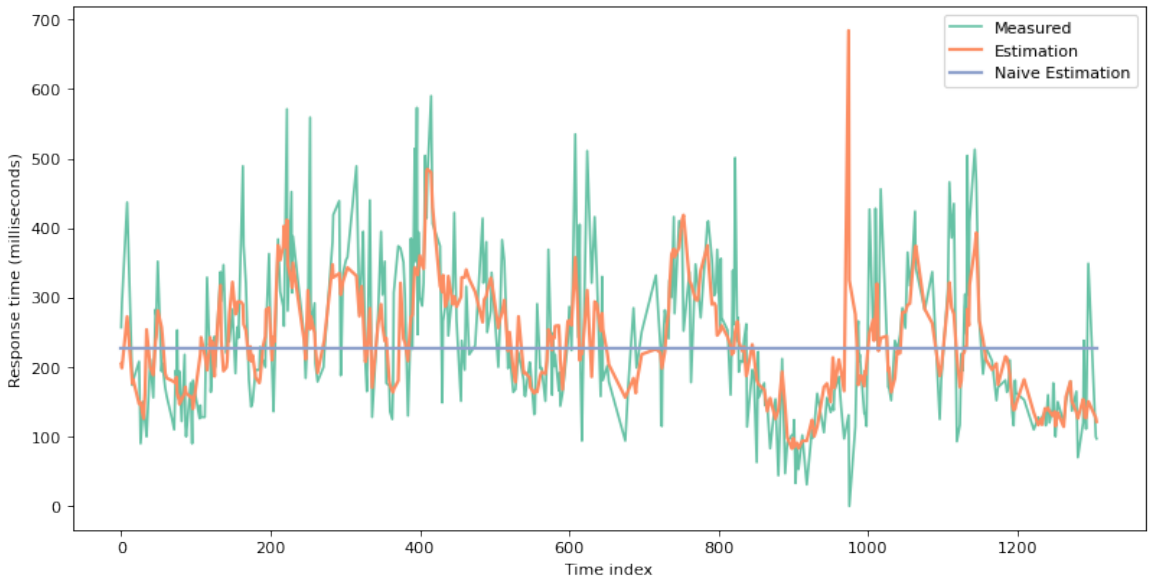
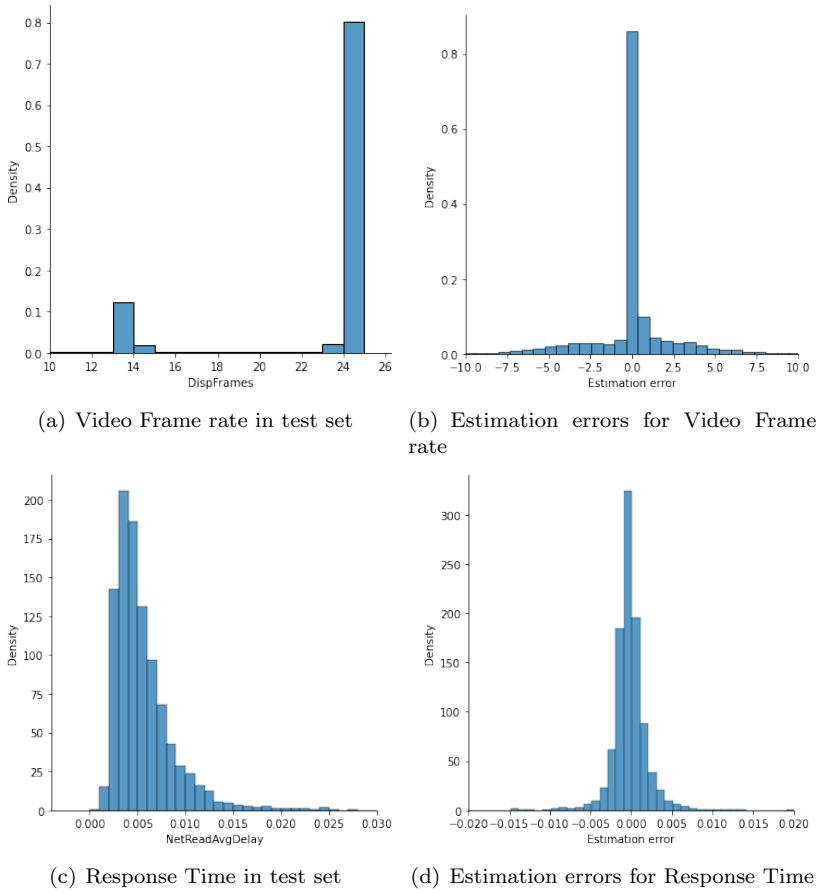


Figure 3: Time series plot with measurements and predictions (VOD service)

Figure 3 is a clip of the predictions on the "NetReadOperation" feature in the VoD dataset using random forest regression. We can see that the orange line closely follows the up and downs of the original measurement. So, it is an acceptable result that captures the basic tendency of how the curve changes.



(a) Video Frame rate in test set

(b) Estimation errors for Video Frame rate

(c) Response Time in test set

(d) Estimation errors for Response Time

Figure 4: Density plots for VoD service

Figure 4 shows the distribution of our target fe ature and our estimation error. We can see a centered distribution of the errors around zero, which means the prediction is not biased towards one side or the other.

Computation overhead

Training the random forest model cost us about 2 seconds running time, while linear regression only 0.8s, neural network around 10s. So, there is some difference in computational overhead of the three models. But thanks to the high efficency of Scikit-learn module that we took advantage of, no models will require more than 60s seconds of running time. However, Scikit-learn can only create the simplest form of fully-connected neural network, also known as MLP. We might want to switch to Keras in future studies.

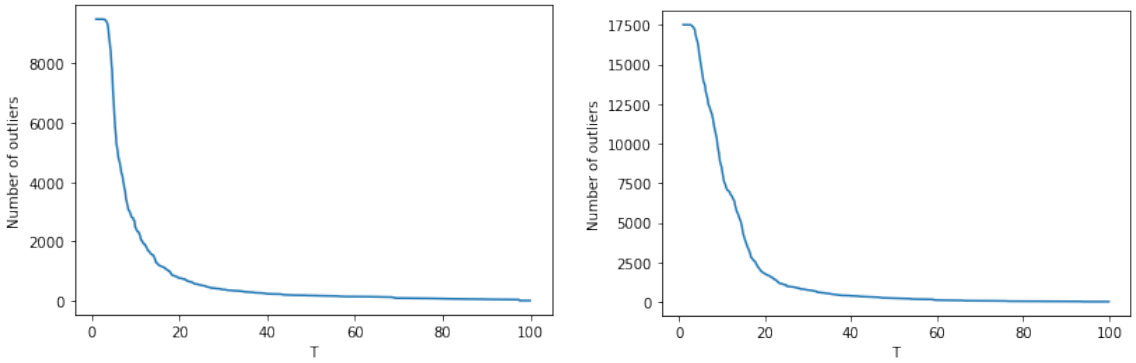
Task III: Studying the Impact of Data Pre-processing and Outlier Removal on the Prediction Accuracy

We first studied how different pre-processing method will impact the performance of our model. As shown in Table 4, performing min-max scale or standard scale on each feature column are the best. And sometimes if we had chosen the wrong method(For example, row-wise l2 normalization), we would make it even worse. We're going to choose the column-wise standardization as our default preprocessing strategy in future studies.

Method	Linear Regression	Random Forest	Neural Network	mean error
original	0.120	0.0572	1.21	0.4650
row-wise normalization	0.115	0.0850	0.127	0.109
column-wise normalization	1.06	0.244	0.208	0.505
row-wise min-max scale	0.124	0.084	0.126	0.111
column-wise min-max scale	0.123	0.0699	0.098	0.0971
row-wise standardization	0.116	0.102	0.118	0.112
column-wise standardization	0.119	0.076	0.097	0.0976

Table 4: NMAE result of different preprocessing methods

Figure 5 shows the number of outliers above threshold parameter T. Whether a sample observation is removed or not is decided by if that row contains any element whose absolute value is above the set threshold. Starting from $T = 0$, all samples are removed, which is not what we want. As T increases, less and less samples are regarded as out-liers. We might want to set T arount 60 to remove a certain number of outliers but not too much or too less.



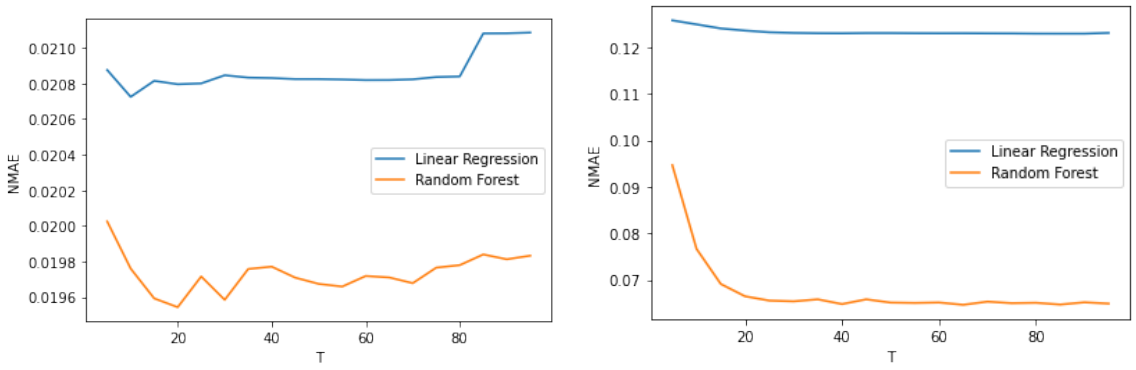
(a) Outliers in KV dataset

(b) Outliers in VoD dataset

Figure 5: number of outliers in function of T

So, how would the outlier removal hyperparameter T affect the performance of the trained model? Let's find this out. Figure 6 shows the impact of outlier removal in our two datasets. The

In KV 6(a) the random forest model, we can see a drop in NMAE in the left. This is due to removing too much training samples including the normal ones will degenerate our model. And in the right there's a slight rise in NMAE, we can see that around $T = 55$ maybe the best performance. Unfortunately, in VoD 6(b), removing outliers isn't going to significantly improve the accuracy. What T we choose should be according to the charateristics of the dataset.



(a) in KV dataset

(b) in VoD dataset

Figure 6: Model performance(NMAE) in function of T

Task IV - Predicting the Distribution of Target Variables using Histograms

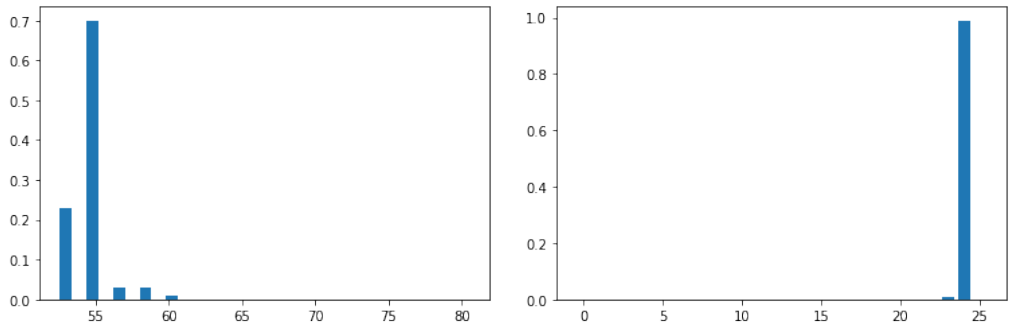
In this task, we are taking a new perspective on our problem. By discretizing the target space Y we're transforming the regression task into a classification task, we might lose some accuracy since our output is now a discrete set ranther than continous numbers. But we're getting a distribution (a histogram on Y) of the possible predicitons.

Comparing the results in Table 5 with Table 3 we can see that, for KV the prediciton accuracy is slightly worse than the random forest regressor, since the discretization caused some information lost. But in VoD, the result is way much better than the regressor, because the target feature in VoD are integers, which makes a perfect transition into labeled classes.

Dataset	RandomForestClassifier
KV	0.0206
VoD	0.0384

Table 5: Model accuracy using histogram prediciton

Figure 7 beblow illustrates the histograms predicted from two X samples from the test set.



(a) Prediction in KV test set, measured y=53.86

(b) Prediction in VoD test set, measure y=24

Figure 7: Sample histogram predicitons. The height of the bars shows the predicting probability for that class, which sums up to 1.

We could draw the conclusion that choosing the suitable model with respect to our target feature and

also the proper method of preprocessing is very crucial in imporving the prediciton performance.

References

[1] F. S. Samani, H. Zhang, and R. Stadler, "Efficient learning on high-dimensional operational data," in 2019 15th International Conference on Network and Service Management (CNSM), IEEE, 2019.

[2] I. Goodfellow, Y. Bengio, and A. Courville, Deep learning. MIT press, 2016.

Some kind suggestions

1. The final order of preprocessing, feature selection, outlier removal and train-test split is not specified clearly, which raises many questions when conducting each part individually. For example in Task III, it confuses me a lot to put outlier removal before or after the train-test split.

2. The design of two datasets at once, not only causes trouble in many replications of codes, and complex variable names but also makes it harder in presenting the result. Our focus is not on comparing the difference between datasets in this project. So I suppose there's little significance in showing similar results of the two datasets in parallel. Why not focus on one dataset indead? We would have more time to dive deeper rather than spending a lot of time organizing our codes.