EP2420 Project 1 - Advanced Project

André Silva

November 21, 2020

Project Overview

In this project, we train machine learning models, namely *Linear Regression*, *Random Forest Regression*, *Neural Network Regression*, and *Random Forest Classification*, that map infrastructure measurements of a networked service to service-level metrics.

We conduct several experiments to measure the performance of these models conditional to the utilization of pre-processing techniques such as *L2 Normalization*, *Restriction to Interval*, *Standardization*, *Outlier Removal*, and *Discretization*.

With the results from the different experiments, we find that pre-processing and data manipulation enables models to perform more accurately when predicting unseen data.

Background

We will be analyzing the data set *VoD flash-crowd*, using the low-level infrastructure measurements to predict the video frame rate observed by clients.

The dataset was recorded on a test-bed running a *Video-on-Demand (VoD)* service, with a load generator following a *flash-crowd* pattern where the load level peaks at certain flash events.

More information on the infrastructure and specificities of the dataset can be found in [1].

Task I

In this task we aim at providing an overview of the dataset, describing some relevant statistics on a sample of features and the target metric, as well as explain what said features mean.

The dataset *VoD flash-crowd* provides us with 36633 samples of 1670 features, and 9 different types of targets.

In Table 1, the statistics for the chosen features, as well as the target *DispFrames*, which represents the video frame rate, are displayed.

Feature name	Mean	Std Dev	Maximum	Minimum	25th percentile	90th percentile
3_cpu16_.sys	3.55	1.95	1.50×10^1	0.00	0.00	0.00
23_RxPacktes	3.15×10^2	4.12×10^2	2.37×10^3	0.00	3.70×10^1	4.70×10^1
3_frmpg.s	-1.15×10^1	2.07×10^4	1.08×10^5	-7.49×10^4	-5.31×10^4	-4.55×10^4
0_cpu18_.idle	9.97×10^1	5.98×10^{-1}	1.00×10^2	9.50×10^1	9.70×10^1	9.80×10^1
2_cpu17_.usr	6.03×10^1	1.82×10^1	9.20×10^1	0.00	8.00	1.34×10^1
4_cpu15_.iowait	4.65×10^{-3}	7.21×10^{-2}	3.03	0.00	0.00	0.00
4_cpu5_.sys	2.90×10^1	1.08×10^1	7.40×10^1	0.00	6.06	9.18
3_cswch.s	7.30×10^4	1.79×10^4	1.07×10^5	7.82×10^3	2.19×10^4	2.65×10^4
38_TxBytes	2.79×10^6	3.18×10^6	1.96×10^7	0.00	3.47×10^5	4.63×10^5
2_dev8.0_avgrq.sz	2.21×10^1	5.78×10^1	1.02×10^3	0.00	0.00	0.00
DispFrames	2.20×10^1	4.32	2.50×10^1	0.00	3.00	8.00

Table 1: Statistics for chosen features and target

The following list gives a short description of these features. This information was retrieved from linux manual pages for the command `sar` [2].

- *3_cpu16.sys* - Percentage of CPU utilization that occurred while executing at the system level (kernel).
- *23_RxPackets* - Total number of packets received per second.

- *3_frmpg.s* - Number of memory pages freed by the system per second.
- *0_cpu18_.idle* - Percentage of time that the CPU or CPUs were idle and the system did not have an outstanding disk I/O request.
- *2_cpu17_.usr* - Percentage of CPU utilization that occurred while executing at the user level (application).
- *4_cpu15_.iowait* - Percentage of time that the CPU was idle during which the system had an outstanding disk I/O request.
- *4_cpu5_.sys* - Percentage of CPU utilization that occurred while executing at the system level (kernel).
- *3_cswch.s* - Total number of context switches per second.
- *38_TxBytes* - Total number of bytes transmitted per second.
- *2_dev8.0_avgrq.sz* - The average size (in sectors) of the requests that were issued to the device.

Task II

2.1

Table 2 provides the calculated *Normalized Mean Absolute Error* (NMAE) for each of the regressors. The parameters utilized for the random forest regression and the neural network regression are the default parameters of, respectively, `RandomForestRegressor` [4] and `MLPRegressor` [6], unless specified in the table.

Regressor	NMAE	Training time (in seconds)
`LinearRegression`	0.318	5.09
`RandomForestRegressor(n_estimators=10)`	0.087	1.24×10^2
`MLPRegressor(max_iter=1000, act='logistic', hidden_layers=(10,10))`	0.144	5.50×10^1

Table 2: *Normalized Mean Absolute Error* for each regressor tested

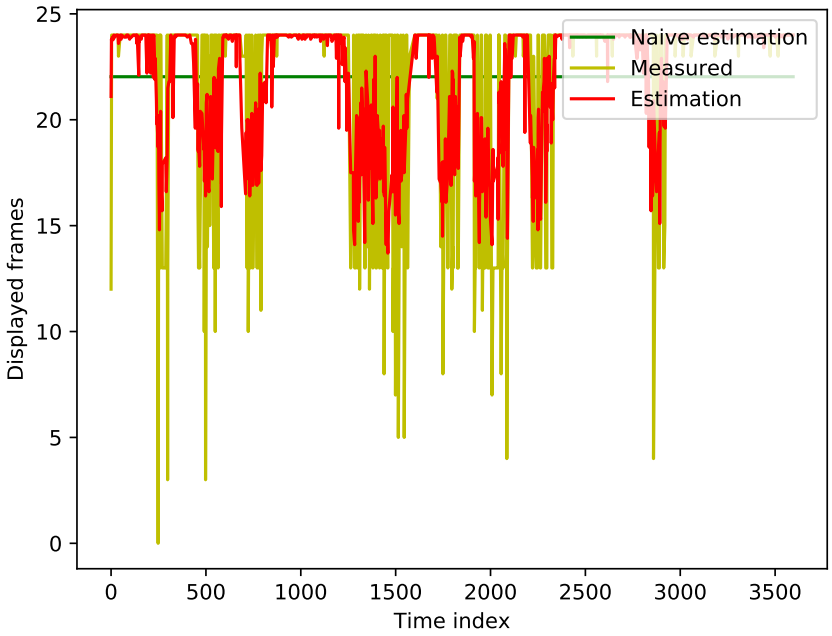

Figure 1: Time series plot for *VoD flashcrowd* using *RandomForestRegressor*

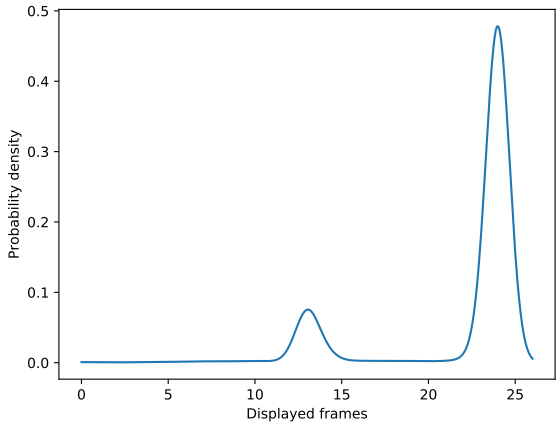

Figure 2: Density plot of the target values in the test set

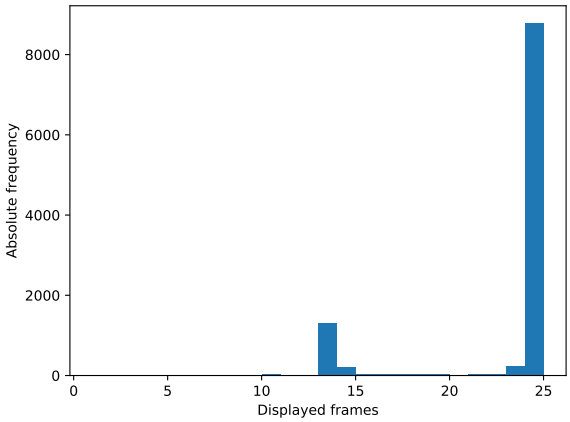

Figure 3: Histogram of the target values in the test set

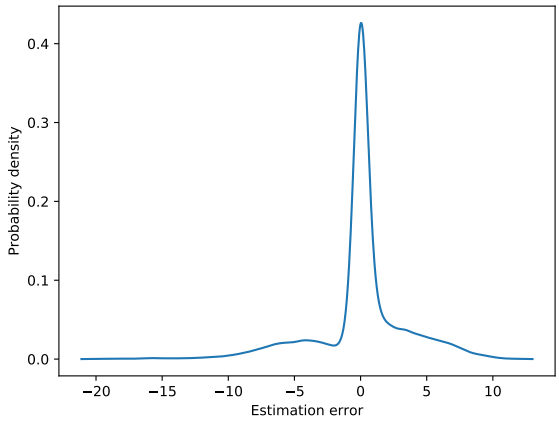

Figure 4: Density plot of the estimation errors $y_i - \hat{y}_i$ in the test set using *RandomForestRegressor*

By looking at the results in Table 2, we can see that, of the three regression techniques utilized, *Linear Regression* provided the worst accuracy, while being the least expensive in terms of training time. *Neural Network Regression* was less expensive than *Random Forest Regression*, which was the most expensive but also the one which provided the best results.

The bad results of *Linear Regression* can be partially explained by the impact outliers have on this type of models, as we used a regular least squares regression and the lack of pre-processing. The same can be said about *Neural Network Regression*, where we also note that the hyper-parameters utilized play a significant role and could, if further investigated, allow this model to outperform *RandomForestRegression*. Random Forests are, on the other hand, known to perform out-of-the-box, meaning hyper-parameters and pre-processing of data usually don't result in major improvements when compared to the default and unprocessed data models.

In Figure 1 we can see the time series plot of the first 3600 seconds. In red we plot the predicted predicted target values, in yellow the measured target values and in green the mean value. We can see that the model can accurately predict when the target value drops from 24, but fails to predict most values lower than 15. By looking at Figures 2 and 3, we notice that observed target values fluctuate between 24 and 13, which could indicate that the system has a control system that triggers a lower frame rate when load reaches a certain level. This is also what we observe in the measured curve in Figure 1.

2.2

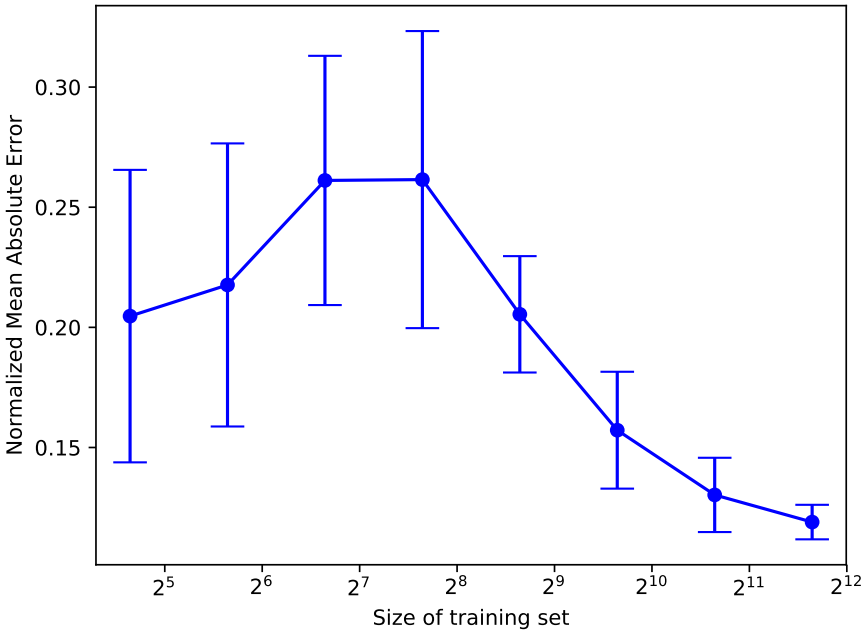

Figure 5: Measured *Normalized Mean Absolute Error* for a given training set size using *Lasso Regression* [5]. Error bars represent 2σ over 50 measurements. Data points represent means.

Intuition tells us that the more information we have access to, the better conclusion we can extract from them.

In Figure 5, we can find the results of our experiment to prove this assumption. The blue dots represent means over 50 measurements for each training set size, while errors bars represent 2σ. By looking at it, we notice that the curve begins by unexpectedly increasing, followed by an expected decrease.

We can also notice that the variance of the measurements starts off very high, and shows a decreasing tendency as we increase the size of the training set. This partly explains why the curve doesn't follow our initial expectations.

Another reason that can explain this is the fact that outliers in small training sets have a higher influence than in large training sets, and the probability of including outliers in randomly picked training sets decreases as we decrease its size.

Overall, we can conclude that the larger a training set is the better a model can predict the system, as the training set becomes more general and so more information can be inferred from it.

Task III

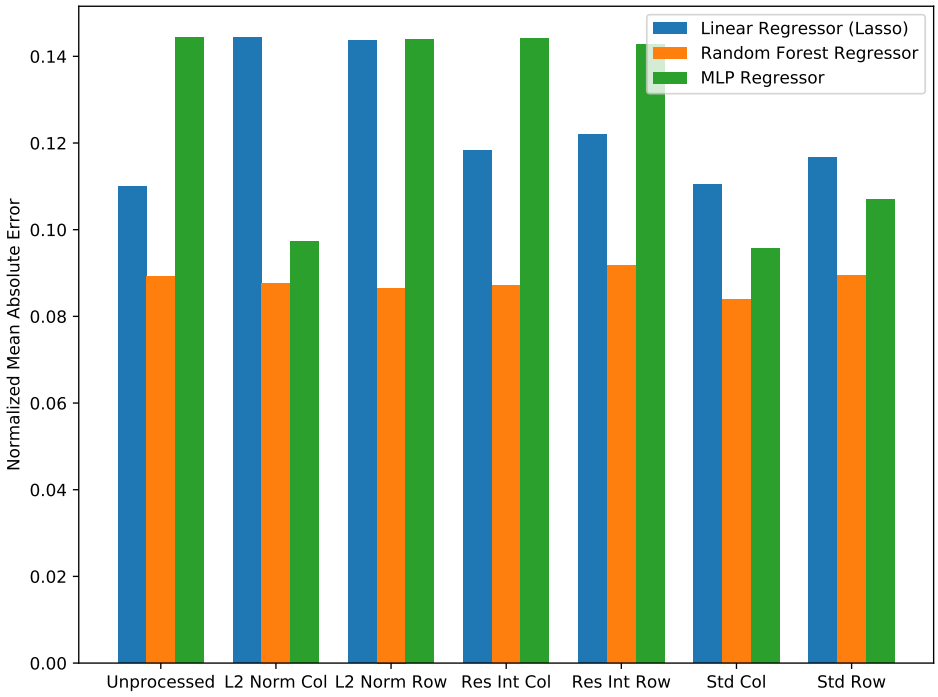

Figure 6: Measured *Normalized Mean Absolute Error* for a given pre-processed dataset.

In Figure 6, we present the results of our experiment with the objective of evaluating the impact of pre-processing techniques on the considered machine learning models' accuracy.

We can see that pre-processing the data impacts some models more than others. In the case of *Random Forest Regression* [4], the gains or losses of accuracy are marginal for all pre-processing methods. This happens by definition, since tree partition algorithms are not affected by the scaling or centralization of datapoints.

MLP Regressor [6] seems to be the model which suffers the biggest impact from data pre-processing, showing either no difference or a considerable gain.

Lasso [5] consistently shows worse performance on pre-processed data when compared to the original one.

By analyzing Figure 6 again, we can see that standardizing along each column is the method that results in the best results for all 3 regression techniques.

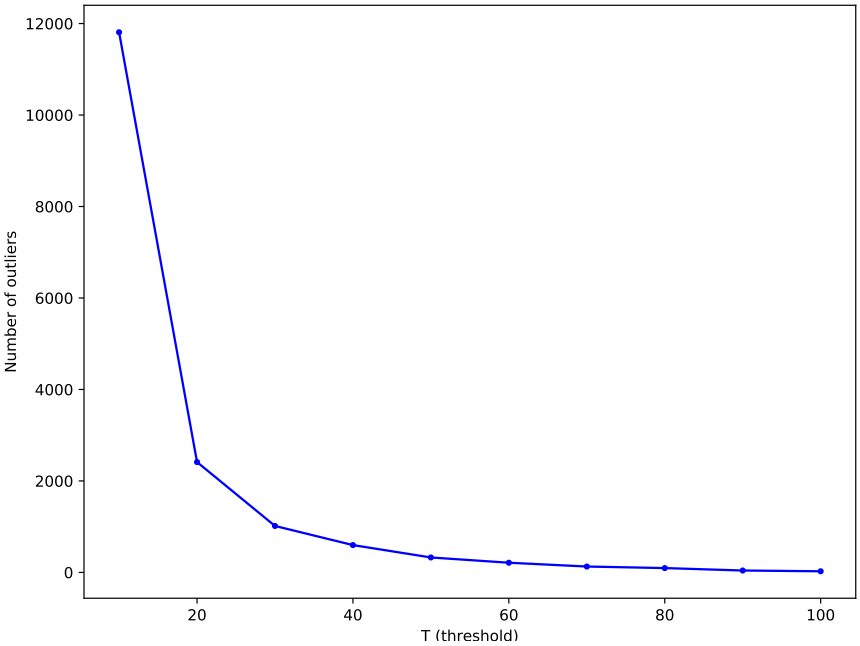

Figure 7: Number of outlier points for a given threshold on the standardized design matrix along each column.

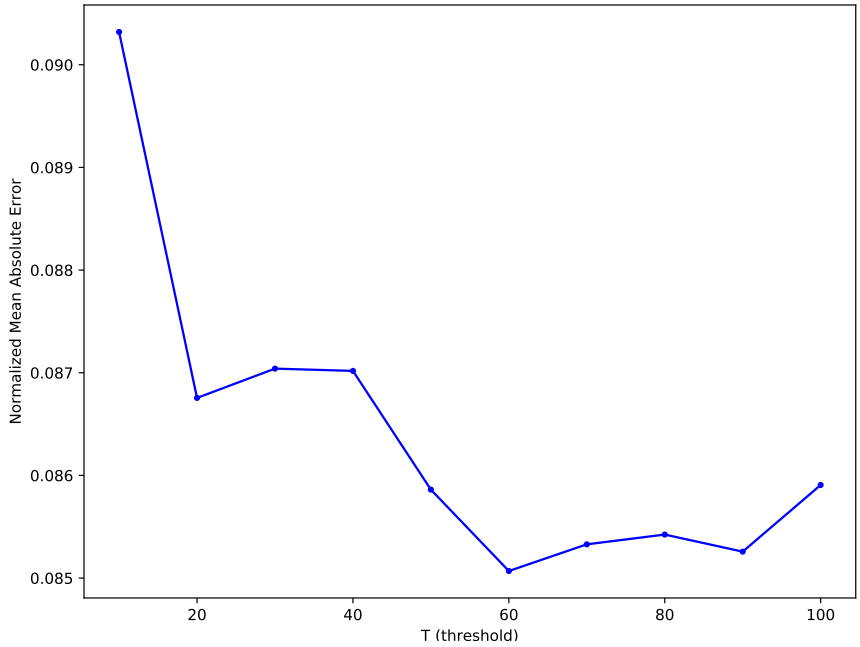

Figure 8: *Normalized Mean Absolute Error* for a given threshold T (outlier removal) using *Random Forest Regression* [4].

In Figures 7 and 8 we plot the results of our experiment with the purpose of measuring the impact of outlier removal on the accuracy of our model. Figure 7 plots the number of outliers removed for a given threshold (deviation from mean). Figure 8 plots the NMAE for a given threshold using *Random Forest Regression*.

Figure 7 shows, as expected, a function that decreases while T increases, eventually converging to 0. This happens because we have standardized the data along each column, making each feature space look like a Gaussian distribution with 0 mean and unit variance.

As we filter rows based on absolute values, we will find less and less examples of outliers as T increases, by the definition of a Gaussian distribution.

Figure 8 shows us that the *NMAE* starts off by decreasing, as we increase the threshold for outlier removal, suggesting low threshold values may lead to the removal of important parts of the dataset. As we incorporate more datapoints in our dataset, we can better predict outcomes. This goes in hand with the results of Task 2.1.

As we reach a very high treshold we notice a slight increase, which might represent the impact more extreme outliers have on the outcome of the tested model, *Random Forest Regression* [4]. This approximation to a parabola was the expected result for this experiment.

Task IV

In this task we apply discretization into bins to the target value, video frame rate, with the intuit of predicting the distribution of said metric and later, in Task V, its percentiles.

Since we are analysing *VoD flash-crowd*, we are going to convert our target values into an histogram of 31 labels, with midpoints 0, 1, 2, ..., 30.

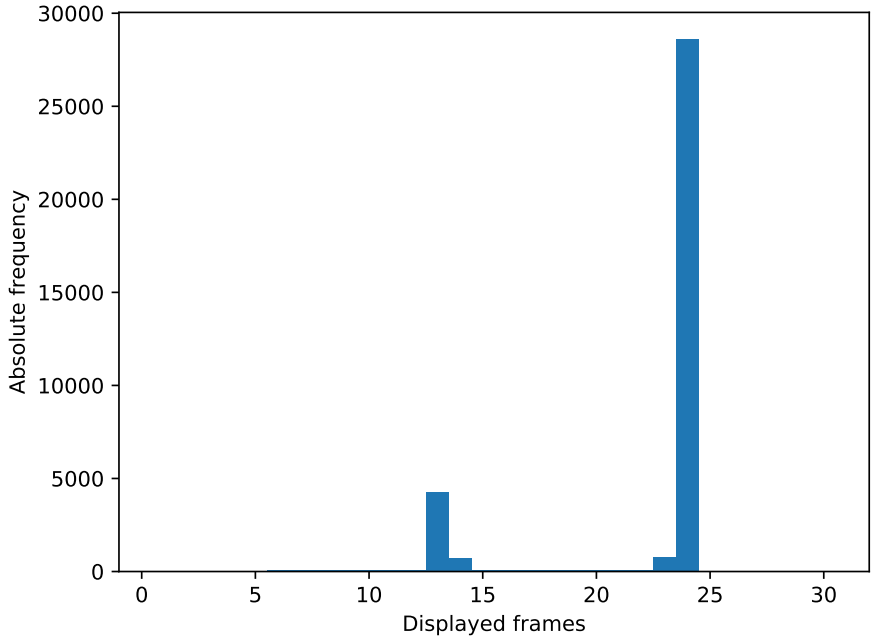

Figure 9: Histogram of the video frame rate on the interval $y \in [-0.5, 30.5]$, with a bin size of 1

We then take each bin of the histogram as a class and fit a *RandomForestClassifier* [3] to the data, which has been target of standardization by column and outlier removal. The hyper-parameters used are the default ones, apart from the number of estimators, which was 10. We train and test the model on a 70-30 split, as was the case for previous tasks.

The calculated NMAE for this model was approximatly 7.3%. In Task II, we had been able to train a model which got 8.7% NMAE, which was then marginally improved in Task III.

We can conclude by these results that the discretization of our target values allows us to better predict them. The loss of information as a result of this process is negligible, given the nature of the target values.

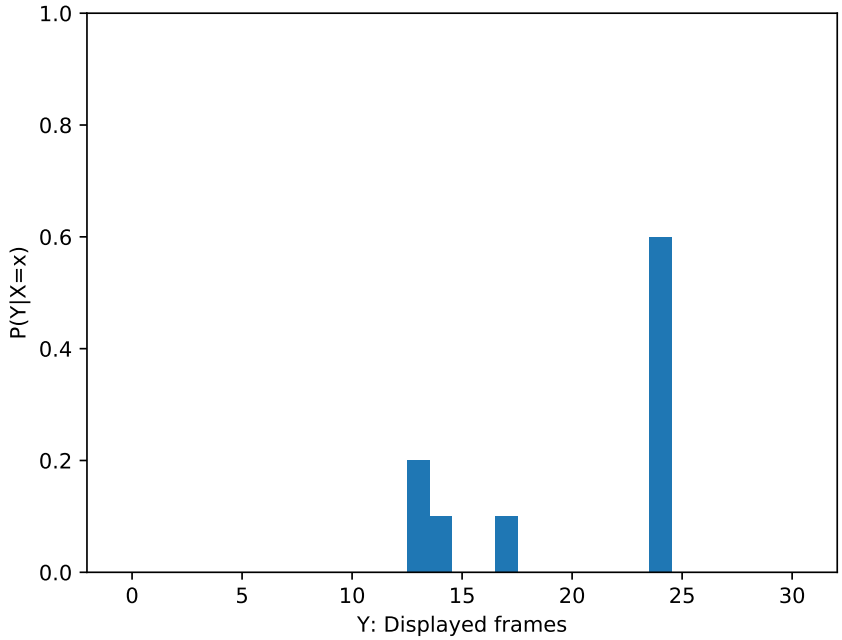

Figure 10: Predicted histogram for a randomly selected sample of the test set. The measured target value was 24.0

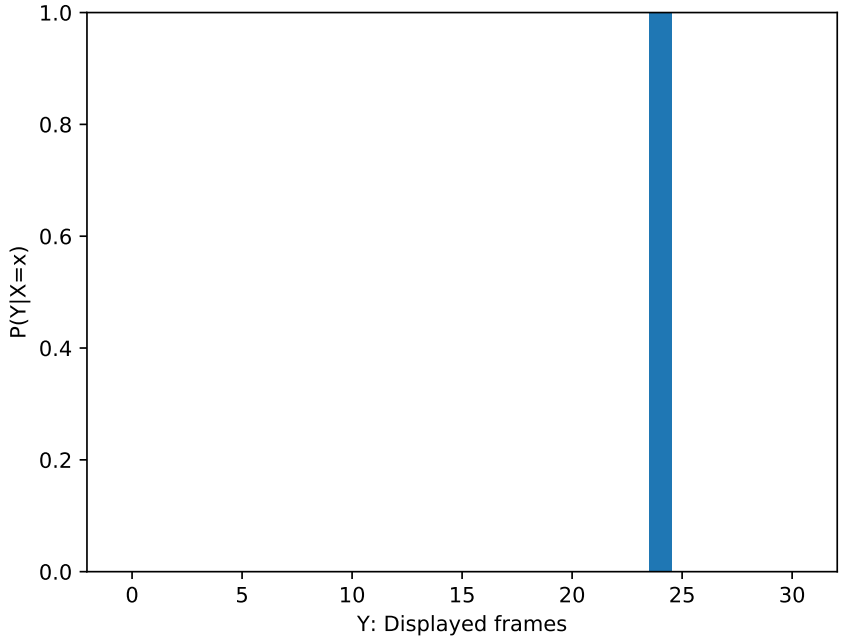

Figure 11: Predicted histogram for a randomly selected sample of the test set. The measured target value was 24.0

In Figures 10 and 11, we can see the predicted histograms for two randomly selected samples of the test set, both with a measured value of 24.0.

In the first case, $P(Y|X)$ is distributed between four classes, 13, 14, 17 and 24, whereas in the second case it is concentrated entirely on class 24. We can extract from this that, when the system is in a state equivalent to the first sample, it is, predictably, more unstable relative to the second, and that we can expect some disturbances in the video frame rate registered by the client.

Task V

In this task, we take the histogram prediction build in Task IV and use it to predict percentiles of the target metric.

To predict the percentile values, we need to convert our bins into cumulative bins, so that every bin now represents $P(Y \leq y|X = x)$. We can then, given a certain percentile P_i, say the predicted percentile is the value corresponding to the first cumulative bin with a value greater or equal to P_i.

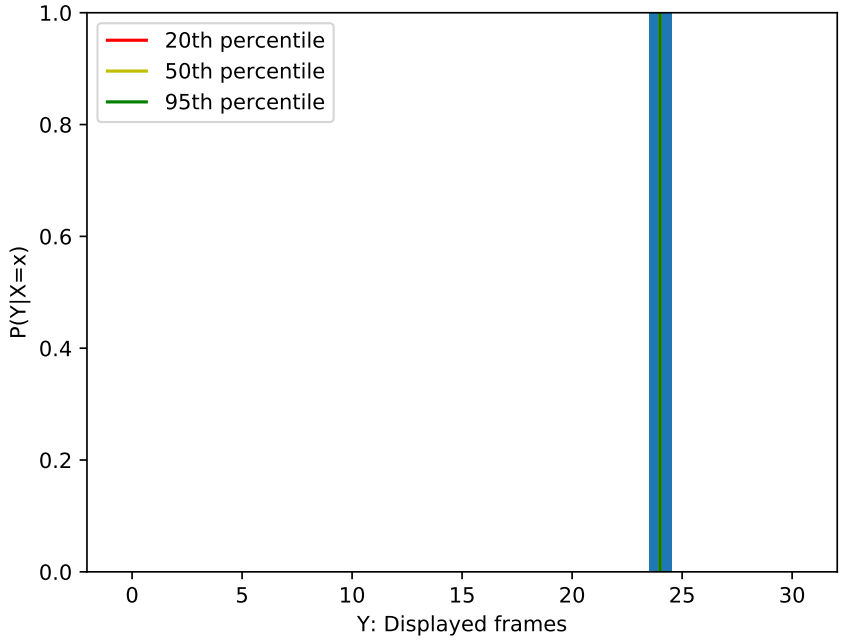

Figure 12: Predicted histogram with percentiles for a randomly selected sample of the test set. The measured target value was 24.0

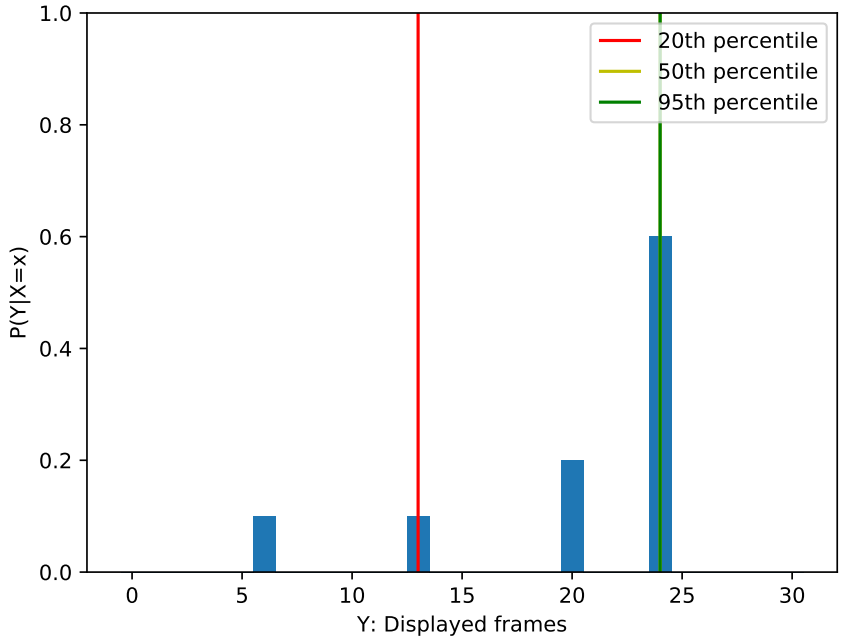

Figure 13: Predicted histogram with percentiles for a randomly selected sample of the test set. The measured target value was 23.0

Figures 12 and 13 show the predicted histograms for two randomly selected samples of the test set. In red we can see the predicted 20th percentile, yellow the 50th and green the 95th.

Since the histogram is distributed along a small number of bins, we can see that some percentiles fall in the same bin. Figure 12 represents an extreme case, where the probability mass function is concentrated in a single bin.

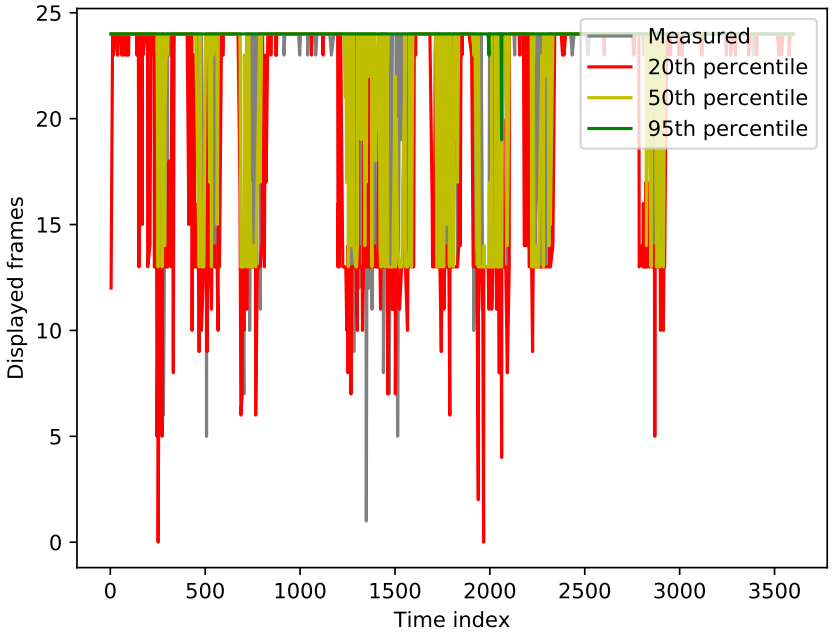

Figure 14: Time series plot for *VoD flashcrowd* including predicted percentiles.

Figure 14 shows us the time series plot of our dataset for the first 3600 seconds. In gray we can see the measured values, in red the 20th percentile, yellow the 50th and green the 95th.

The percentile curves follow our expectations. The 20th percentile curve reaches the lowest levels of the measured metric, while the 95th percentile almost never drops from the value 24. The 50th percentile curve fluctuates mostly between 13, 14, and 24, which are the most common occurrences of the target metric, as shown in Figure 9.

Percentile	Estimation of percentile
20th percentile	0.703
50th percentile	0.895
90th percentile	0.999

Table 3: Percentiles and the correspondent estimations.

In Table 3, we can see the estimations for each percentile using the provided formula.

We can see that none of the estimations are very accurate, since we were expecting values close to, respectively, 0.2, 0.5 and 0.9.

Taking into account the conclusions regarding Figures 12 and 13, where we saw that percentiles often fell in the same bins (as can also be confirmed by Figure 14), we can conclude that our discretization process and the model's histogram prediction limits our estimations, as a sample that falls into a certain percentile can also, very likely, fall into a different percentile at the same time.

Discussion

In Task I, we investigated relevant statistics of our dataset, a sample of features and the target value. We also discovered what the sampled features mean in the context of the networked system.

In Task II, we aimed at building three different models for predicting the target value and found that our *RandomForestRegressor* behaved the best. We also studied the impact of training set size on the accuracy of a Lasso Regressor, reaching the conclusion that the more data we have for training the better we can predict the behavior of the system.

In Task III, we studied the impact of data pre-processing and outler removal on the performance of our models. We found that standardizing along each column was the method that provided the best results, improving accuracy on Linear Regression and Neural Network Regression, while Random Forest Regression was not impacted by any pre-processing technique. We also found that outlier removal enabled our *RandomForestRegressor* to better predict, as long as we choose a reasonable threshold.

In Task IV, we predicted the distribution of our target metric using histrograms. We accomplished this by, first, discretizing our target metric into bins and then building a *RandomForestClassifier*. The accuracy of this model represented an improvement when compared to the models of previous tasks. We then selected two samples of the test set to demonstrate the predicted histogram.

In Task V, we predicted percentiles of our target metric using the model built in the previous task. We concluded that, due to the discretization and nature of the dataset, predicted percentiles often fall in the same bin (or class). We confirmed these results when our estimation of the percentiles produced inaccurate results.

The main difficulties encountered during this project were related to the size of the dataset, which resulted in significant computational overhead during some experiments.

Further work can include the incorporation of more machine learning techniques, such as dimensionality reduction and more deep model selection, with the objective of improving accuracies and obtain a better understanding of the system

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)*. 2019, pp. 1–9. DOI: 10.23919/CNSM46954.2019.9012741.

[2] *sar command manual page*. https://linux.die.net/man/1/sar. accessed: 2020-10-31.

[3] *sklearn.ensemble.randomforestclassifier*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html#sklearn.ensemble.RandomForestClassifier. accessed: 2020-11-13.

[4] *sklearn.ensemble.randomforestregressor*. https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.randomforestregressor.html#sklearn.ensemble.randomforestregressor. accessed: 2020-10-31.

[5] *sklearn.linear_model.Lasso*. https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.Lasso.html. accessed: 2020-11-07.

[6] *sklearn.neural_network.MLPRegressor*. https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPRegressor.html#sklearn.neural_network.MLPRegressor. accessed: 2020-10-31.