EP2420 *Project 1 - Advanced Project*

Joseph Attieh

November 26, 2021

Project Overview

With the help of the newest advances in machine learning, researchers were able to build models that can predict system performance metrics from a set of measurements collected from the system. In this project, we are presented with data traces collected from a Video-on-Demand service deployed in an experimental setup at KTH [1]. We are asked to train regression models that use these infrastructure measurements to predict the service-level metric, which in our case is the Video Frame Rate. The rest of the report is structured in three main parts: The first part introduces the reader to the machine learning concepts used in this project. The second part corresponds to the data description. The rest of the parts match the tasks in the project description.

Background

This section explains the concepts behind the machine learning methods used in this paper. The concepts tackled here are the following: Linear Regression, Tree-based models, Neural Networks, and Hyper-parameter search.

Linear Regression model

Linear regression [2] is a statistical approach that models the relationships between a target dependent variable and a set of independent variables by trying to fit a linear equation to some observed data (i.e., training data). Linear regression learns the relationship between the feature variables $X \in R^n$ and the predicted value $Y \in R$ by fitting a hypothesis from the following hypothesis space: $H^{(n)}_{linear} = \{h^{(w)} : \mathbb{R} \rightarrow \mathbb{R} : \mathrm{h^{(w)}(x) = w^T x}$ with $\mathrm{w} \in \mathbb{R}^{\mathrm{n}}\}$.

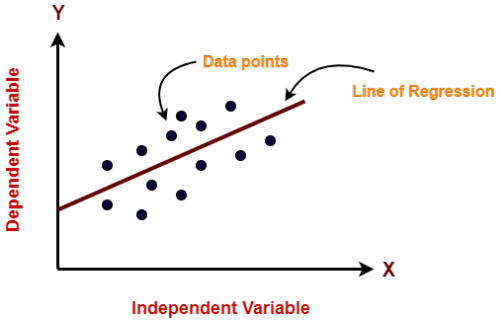

Figure 1: Example of linear regression.

Fig 1 presents an example of linear regression in which the dependent variable is the target variable and the independent variable is the feature. The line of regression represents the model which tries to approximate the relationship between both variables by fitting over the data points.

Tree-based machine learning methods

Tree-based methods are models that partition the feature space into a set of regions/rectangles, then proceed by fitting a simple model in each one of these regions [3]. This set of methods is known as tree-based methods since the set of boundaries/rules used to segment the space can be summarized through a tree structure. To make a prediction for a given observation, the algorithm returns the mean of the training data in the region/rectangle in which the observation belongs to in space[3].

We can mention two tree-based methods:

1. **Regression Trees**: Regression trees are tree-based methods. They usually are built by using a fast divide and conquer greedy algorithm known as the Recursive Partitioning algorithm [3]. This algorithm builds a tree by recursively splitting the training sample into smaller subsets. At each node the best split test is chosen according to some local criterion (greedy), in a way that selecting a subset of the data in a certain step might lead to better descendant subsets. The algorithm keeps on partitioning until a termination criteria is satisfied (i.e., threshold on the goodness measure or on the size of subset). The inner nodes of the tree represent decision nodes (boundaries of the regions in space) while the leaf nodes represent the target values (i.e., the value we want to predict). Subsequently, generating predictions for new (unlabelled) data points is done by navigating the decision tree, comparing the new data point's features with the inner-tree node's decision criteria, in order to reach the leaf node. Some measures of goodness of the split are the entropy and the GINI index. The algorithm is showed below:

Algorithm 1 Recursive Partitioning Algorithm [3]

Input Set of training samples and target values $\{(x_i, y_i)\}_{i=1,\dots,|X|}$
Output Regression tree
if termination criterion **then**
  Create a leaf node and assign it a constant value as target
  Return leaf node
else
  Find the best splitting target value y_S using statistical measure
  Create Node t with the set of y_S
  $Left\_branch(t) = RecursivePartitioningALgorithm(\{(x_i, y_i) : x_i \rightarrow y_S\})$
  $Right\_branch(t) = RecursivePartitioningALgorithm(\{(x_i, y_i) : x_i \not\rightarrow y_S\})$
  Return Node t
end if

2. **Random Forest Regressors**: Random forest is an ensemble regressor that operates by constructing multiple regression trees at training time and generating the prediction by outputting the mean of the values generated by all the regression trees. It is built by randomly selecting a subspace of features at each inner decision node to grow branches of a tree. Then, the bagging method is used to generate training data subsets for building individual trees accordingly. Finally, a prediction is made by traversing all the trees. We should note that the trees run in parallel with no interaction amongst them[3].

Neural networks (NN)

Neural networks are supervised machine learning techniques that are designed to learn by multi-connection of layers of neurons. Each neuron in a certain layer forms a weighted sum of the inputs from previous layers connected to it with a threshold value and produces a non-linear function of this sum as its output value. Neural networks are nonlinear models, which allows them to model real world complex relationships. Also, the number of layers, the number of neurons per layer, the activation functions, and the activation thresholds

of each neuron are all parameters that need to be properly tuned as part of learning and decision making processes.

Hyper-parameter Search

Classifiers/Regressors have some parameters that need to be set prior to the training stage. These parameters are called hyper-parameters, and are usually set through a process called hyper-parameter optimization. The most commonly strategies for hyper-paramter optimization are Grid search [4] and Random search [5]. Grid Search finds the best set of hyper-parameters by trying every combination of hyperparameter values. For example, searching p different parameter values for n parameters would require p^n trials of cross-validation (very inefficient). By contrast, Random Search selects random combinations of parameters to train the model. The python library *sklearn* provides with the *RandomizedSearchCV* tool to perform a random search search. Practically, Random search is more effective than the Grid Search [5].

Data Description

This project is based on the dataset "VoD-flashcrowd (JNSM) - Part 1". This dataset can be described as follows:

- Samples: The dataset contains 18316 samples. Every sample corresponds to an instance of the state of the testbet (i.e., every row has a set of features where the features represent the state of the system).
- Features: After dropping the first two columns (index and timestamp), the number of features in the dataset are 1670. These features are a set of measurements collected from the testbet at KTH. These features are either cluster-based features extracted from the kernel of the Linux operating system that runs on the server executing the applications or port-based features extracted from the OpenFlow switches [1]. Due to the huge number of features, we will not be describing every one of them.
- Target: Since we are dealing with a VoD service, the service level metric that we want to predict is the Display Frame Rate (frames/sec), i.e., 'DispFrames'. More information of this target metric (i.e., time serie plot and histogram) can be found in Task 2.

Task I - Data Exploration and Pre-processing

This section contains the answers to the questions under the first task in the project description.

1. **Feature selection**: As mentioned previously, we are dealing with 1670 features. Since we are dealing with a lot of features, we will be applying a feature selection technique to reduce the size of the feature space, to allow faster, less memory consumption, and (maybe) more accurate computations (by disregarding nondescriptive dimensions). We choose to apply a (supervised) tree-based feature selection method to reduce the dimensionality of the feature space. In our case, we will be using a Random Forest Regression model. As explained in earlier sections, the random forest model is an ensemble of regression trees. Such a model can determine the importance of every feature by relying on the underlying regression trees; in fact, regression trees usually determine the importance of the feature based on the decrease of the impurity of this feature. These measures can be averaged over the trees to find the importance/ranking of the features.
   To do that, we choose to use the *SelectFromModel* from *sklearn* which allow us to get the top 16 features of a *RandomForestRegressor*. No specific hyperparameter tuning was applied to the Random Forest Regressor and default values were kept.
   The top 16 features are the following: 2_*kbcommit*, 2_$file.nr$, 2_*ldavg*.1, 3_*cpu*20_.*idle*, 3_*kbcommit*, 3_*ldavg*.1, 3_*orq.s*, 4_*kbcommit*, 4_*X..commit*, 4_$file.nr$, 4_*plist.sz*, 4_*ldavg*.1, 5_*sum_intr.s*, 5_*irec.s*, 36_$TxPacktes$, 36_$TxBytes$.1. The statistics of these 16 features are displayed in the notebook.

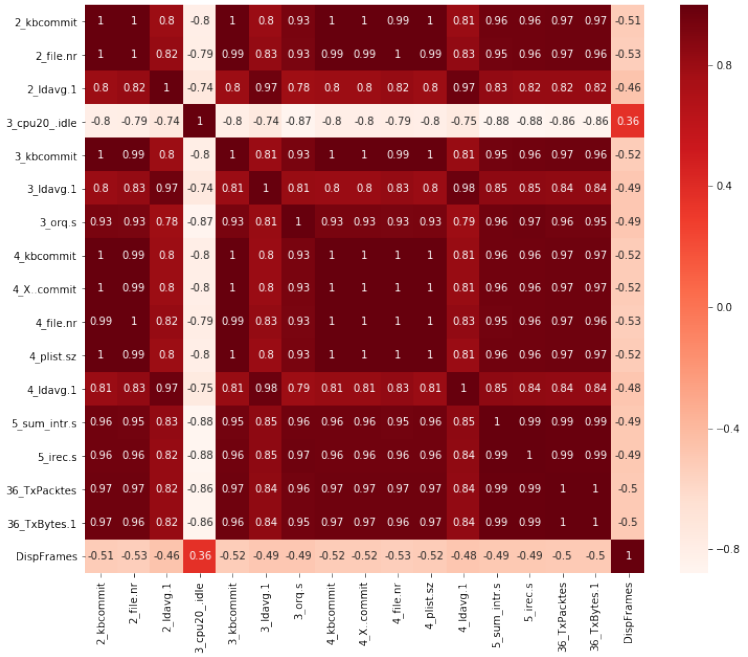

Figure 2: Correlation matrix among the 16 features and between the features and target.

The correlation among features and between features and target can be seen in Fig 2.
After looking at the correlation matrix, we can observe the following:

- 15 out of 16 features are strongly correlated with each other, but at the same time are weakly correlated with the target variable.
- 1 feature (3_*cpu*20_.*idle*) is weakly correlated with all other 15 features, yet is strongly correlated to the target variable.

We can highlight that Random Forest Regression models perform feature selection without taking into account the correlation between the features; in other words, these models might favor certain correlated features and attribute them the same/similar importance.

2. **Feature preprocessing**: Multiple preprocessing methods are used in this project. We should note that we have the option to either apply the preprocessing by row or column. We denote by the matrix M the raw data matrix in which every row correspond to a certain sample and every column to a certain feature. We denote by M_p the processed version of that matrix, by i the row index and j the column index.

   (a) L_p Normalization: This normalization technique adjusts the values of the column/row to have an L_p-norm to be 1. In this project, we choose to perform an L_2 normalization. For instance, if we apply the following per column, every element in column j would be scaled as follows: $M^p_{ij} = \frac{M_{ij}}{\sqrt{\sum_{i=1}^{m}(M^2_{ij})}}$

   (b) Restriction to Interval: This processing technique linearly maps the values of each column/row so that values lie within the interval [0,1]. This is also known as Min-Max scaling. For instance, if we apply the following per column, every element in column j would be scaled as follows: $M^p_{ij} = \frac{M_{ij} - min(M_{:,j})}{max(M_{:,j}) - min(M_{:,j})}$

(c) Standardization: This processing technique linearly maps the values of each column/row to have a mean equal to 0 and a variance equal to 1. For instance, if we apply the following per column, every element in column j would be scaled as follows: $M^p_{ij} = \frac{M_{ij} - \mu_{:,j}}{\delta_{:,j}}$

We apply these pre-processing methods once per row and once per column. This will result in 6 design matrices. These matrices will be used in the third task as requested in the project description.

Task II - Estimating Service Metrics from Device Statistics

In this section, we build our model by selecting the raw unprocessed design matrix while only keeping the top 16 features that we got from the feature selection method.

1. **Training and testing sets**: We first split the dataset (i.e., the data in the design matrix M) into two sets, one used at the training stage of the model and one at the testing stage (data from the testing set is not used in any way at the training stage). This is very crucial to evaluate the models that we are going to train, as we need to check whether those models have learned a valid hypothesis that can generalize well. We chose to perform the split between the training set and testing set only **once**.
   Since the split happens once, we decide to create the train and test set by randomly drawing from the original matrix, in a way that 70% of the data is in the training set and 30% of it is in the testing set.

2. **Evaluation metric**: To evaluate the models, we were advised to compute the estimation error in terms of the *Normalized Mean Absolute Error*. This is computed as follow:

$$NMAE(y, \hat{y}) = \frac{1}{\overline{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y_i}|) \text{ in which y is the target value and } \hat{y} \text{ is the predicted value} \quad (1)$$

   This function has been implemented in the report. We could have also divided the result obtained from the *sklearn.metrics.mean_absolute_error* by the average of the target value.

3. **Models**: In this section, we will discuss the machine learning models used to perform the prediction.

   (a) Baseline: As a baseline, we rely on a naive method which predicts the target value by computing the mean of the target value over the samples in the training set. The average value returned is 22.071.

   (b) Linear regression: This model assumes that the relationship between the features and the target variable is linear. We can use Linear Regression (i.e., Polynomial regression with a maximum degree r equal to 1) to try to predict the target value.
   For the implementation, we rely on the *sklearn.linear_model.LinearRegression* implementation provided by sklearn.

   (c) Random Forest Regressor: Before training the random forest regressor, we need to search for the optimal hyperparameters for the model. Since our goal is to minimize the NMAE on the testing set, we optimize the parameters to minimize the NMAE on the training set. We proceed by first performing a Random Search on a wider range for the parameters of the Random Forest Regressor. Once we got the initial optimal parameters, we narrow down the region/range of the parameters to revolve around the optimal parameters and we perform a second random search. The values obtained by this second random search are the values that we will be using to set up the Random Forest Regressor. We should note that after re-running the notebook, we might get a different split between training and testing sets, thus a different set of hyperparameters for the model. Fig 3 presents the hyperparameters that we chose to optimize for the Random Forest Regressor as well as the results of the first random search and the results of the second random search.

(d) Neural Network: Before training the neural network, we need to search for the optimal parameters that are suitable with our training data. We should note that the network that we are building has hidden layers with a sigmoid activation function and one last layer containing one neuron (corresponding to the target value) with a linear activation function (to be able to predict a range of continuous values).
We perform the following hyperparameter tuning: First, we search for a learning rate that minimizes the NMAE. After we have done that, we search for the number of epochs needed to achieve optimal results. Then, we perform a search for the optimal batch size. Finally, we perform a GridSearch to set the number of hidden layers and the number of neurons in each layer. We decided to set the number of neurons per layer as a constant value (this network can represent smaller networks since the weights of some neurons might converge and become zero, resulting in smaller networks). The table displayed in Fig 4 presents the hyperparameters that we chose to optimize for the Neural Network as well as their values.

Hyerparameter	Description	Random Search 1	Random Search 2
n_estimators	Number of trees in the random forest	400	200
max_features	Number of features to consider at every set/node split	sqrt	sqrt
max_depth	Maximum number of levels in a tree. If None, nodes are expanded until all leves are pure or until leaves contains less than min_samples_split	40	50
min_samples_split	Minimum number of samples required to split a node	2	2
min_samples_leaf	Minimum number of samples required at each leaf node	1	1
bootstrap	Method of selecting samples for training each tree	False	False

Figure 3: Description [6] and values of the hyperparameters of the Random Forest Regressor.

Hyerparameter	Description	Optimal value
learning_rate	Size of the step taken towards at every iteration towards achieving the minimum of the loss function during the optimization of weights	0.1
epochs	Number of complete iteration through the whole dataset	20
batch_size	The number of samples to be propagates through the network during backpropagation	1
layers	Number of layers in the architecture	2
neurons	Number of neurons per layer in the architecture	10

Figure 4: Description and values of the hyperparameters of the Neural Network.

4. **Results**:

(a) After training the models on the training set, we compute the estimation error of every model (i.e., NMAE) on the testing set. The results are displayed in the table below:

Table 1: NMAE, training and testing time of different models trained on the raw data

Models	NMAE	Training time (seconds)	Testing time (seconds)
Baseline	0.143	-	-
Linear Regression	0.129	0.0156	0.0808
Random Forest Regressor	0.0655	20.72	0.281
Neural Network	0.0878	357.664	0.103

We should note that more accurate results could have been retrieved in the case we performed k-fold cross validation, or we repeated the experiments a higher number of times (Monte-Carlo).

(b) Time series plot: Fig 5 shows a time series plot of the measurement's target, the estimation of the Random Forest Regressor and the baseline.

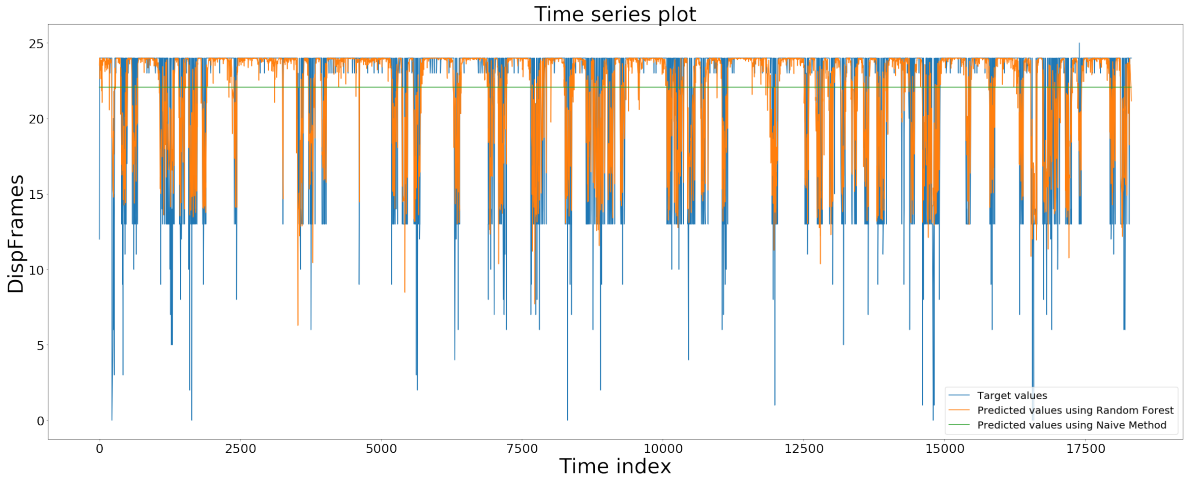

Figure 5: Time series of the target values and the estimations of both Random Forest and baseline.

(c) Fig 6 presents a density plot and a time series plot for the target values on the whole set.

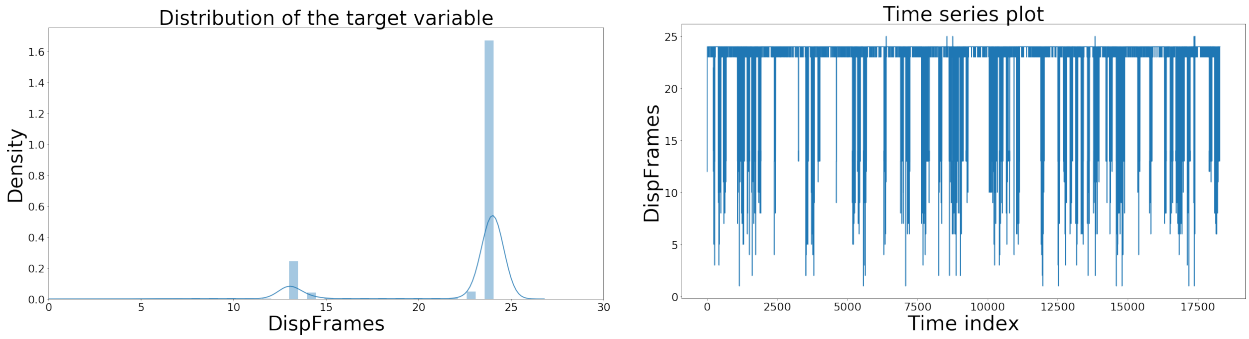

Figure 6: (a) Density plot and (b) time series plot of the target value

(d) Fig 7 shows the density and histogram plots of the estimation errors of the baseline, linear regression, random forest regression and neural network on the test set. We should note that the error plotted is the absolute value of the difference, i.e., $|y - \hat{y}|$ as it gives a better estimate of how far the predicted values are from the target values.

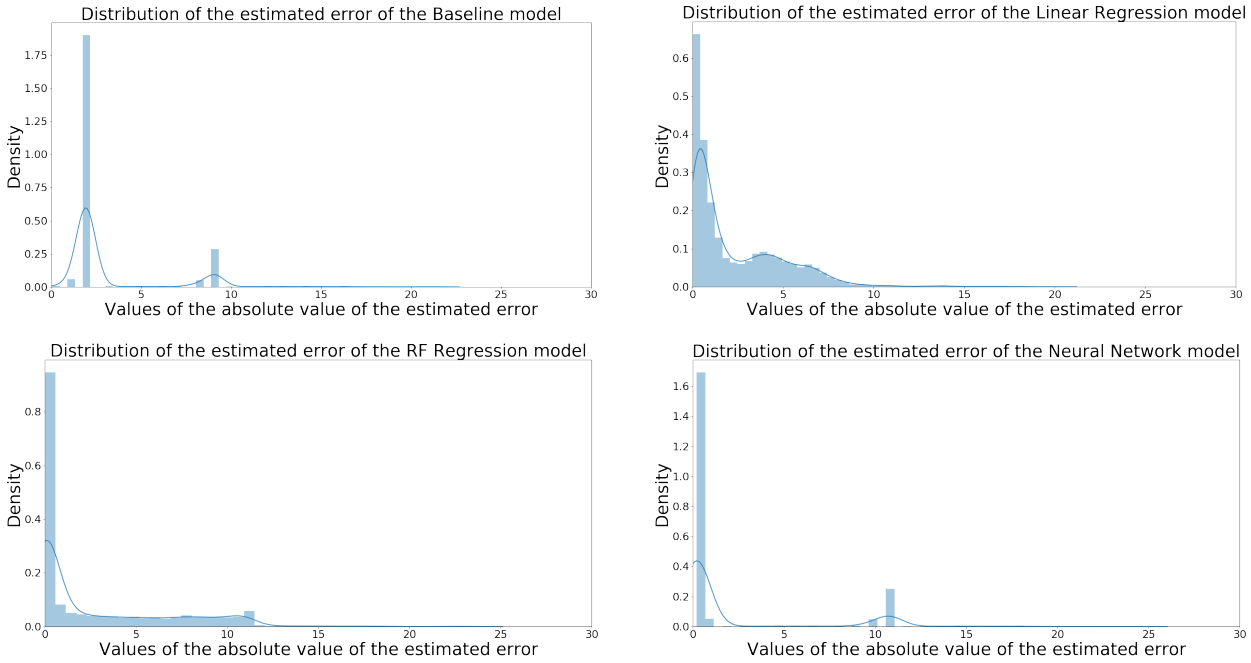

Figure 7: Density and histogram plots of the estimation error of the Baseline, Linear Regression, Random Forest Regressor and Neural Network models

5. Looking at Table 3, we can clearly observe the difference in the evaluation metric between the four methods used. The baseline performs the worst among the four methods, which is due to the fact that it is quite naive and does not rely in any way on the features/measurements. We can also notice that both the Random Forest Regressor and the Neural Network perform much better than the Linear Regression model in terms of accuracy. This is also clear when looking at the distribution of the error of each of these models; the estimation error for the Random Forest Regression model and the neural network have a bigger peak at 0, while the distribution of the error in the Linear Regression model has a lower peak at 0, and a worst distribution that extends in a hat shape. We can also see that the distribution of the error in the neural network model has two main peaks (one of them is at 0). This difference in performance can be explained as follows:

   (a) Both random forest model and neural network models are much more complex than the Linear Regression model, as they are able to approximate non-linear functions.
   In fact, random Forest Regression models can approximate non-linear functions by splitting over the features/variables ranges while the neural networks can perform this through a series of non-linear operations that are performed by navigating through the layers of the network. On the contrary, Linear Regression models can only produce functions with a linear relationship to the chosen set of features. Therefore, it makes sense for the first two models to perform better and generalize better whenever trained properly.

(b) The Random Forest Regression model performed better than the neural network model. This can be explained by the fact that it is from the family of ensemble methods (i.e., it aggregates multiple regressors). Furthermore, hyperparameter tuning is much easier with Random Forest models compared with the Neural Networks (very expensive in terms of time).

However, at the same time, the Linear Regression model took much less time to fit/train over the training data compared to the other models. This is due to the complexity of the Forest algorithm which builds multiple regression trees and the neural network model which updates the weights of the neurons based on a number of epochs, batch_size, learning_rate and architecture which highly affects the performance (time consuming compared to the linear regression model which only needs to build one model).

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

1. **Pre-processing** : In the first task, we applied different preprocessing methods and we generated 7 design matrices. We use every preprocessed matrix and we train the three different regression models that we have. The models trained have the same hyper-parameters set before. We present two tables: the first table displays the results with the same top 16 features from Task 1 while the second table displays the results after applying feature selection on each design matrix on its own.

Table 2: NMAE of different models trained on different design matrices (feature selection on original dataset)

	L2 norm by column	L2 norm by row	Min-max norm by column	Min-max norm by row	Standardization by column	Standardization by row
Linear regression	0.108	0.122	0.108	0.121	0.108	0.122
Random Forest Regression	0.066	0.089	0.066	0.087	0.065	0.103
Neural network	0.0856	0.0845	0.0852	0.0925	0.0809	0.0850

Table 3: NMAE of different models trained on different design matrices (feature selection on each dataset)

	L2 norm by column	L2 norm by row	Min-max norm by column	Min-max norm by row	Standardization by column	Standardization by row
Linear regression	0.108	0.112	0.108	0.106	0.108	0.106
Random Forest Regression	0.066	0.08	0.066	0.084	0.065	0.082
Neural network	0.0923	0.0816	0.0815	0.0847	0.0820	0.0817

The Tables above follow a similar trend. Therefore, analysis will be carried on the second table as it considers the best 16 features for every processed matrix (fair comparison). After examining the table, we can notice the following:

- The Linear regression model performs better when the data is processed (original NMAE 0.129 vs on average 0.108). However, the processing techniques used did not have a major influence on the performance. Furthermore, processing techniques applied on the same axis maintained similar NMAE.
- The Random Forest Regression model maintained the same performance (no improvements in NMAE compared to unprocessed matrix) when the processing technique was applied by column. However, the performance degraded whenever the model was used with a processing technique that was applied per row. This might be explained by the fact that these model rely on computing a purity value to perform the split, which is mostly affected by the feature distribution (i.e., the distribution is mostly changed when processing is applied per row, which affect the algorithm).
- The Neural Network model maintained similar performance on all the processing methods used. This can be explained by the fact that neural networks have weights that can scale up or down the values of the features, which render the preprocessing method not drastically important in our case.

If we were to choose the best preprocessing method that works with all regressors based on both these tables, we choose the Standardization per column.

2. **Oulier Removal**:

   (a) Outliers are defined as samples of which one component has an absolute value larger than a given threshold T. We compute and plot the number of outliers in the dataset with respect to the given threshold on the values of the features. We perform this study on the column standardized design matrix.
   Fig 8 presents the results of the study; the plot clearly shows that the higher the threshold T, the less samples have features with an absolute value under this threshold, and the smaller the number of outliers is.

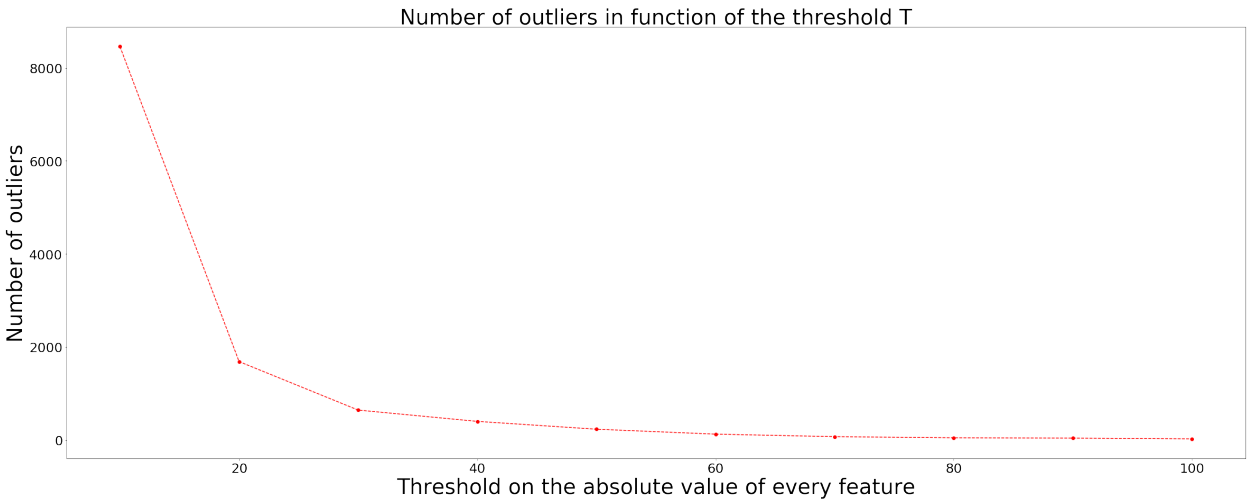

Figure 8: Number of outliers in the dataset in function of the threshold T

(b) In this section, we choose to investigate the error of the random forest regressor in function of T on the data set. Therefore, we choose the design matrix with standardized columns and we perform outlier detection by filtering out the samples which are considered outliers. Then, we perform feature selection as we described in the first task (using a Random Forest Regressor). Finally, we split the set into training and testing sets and we train the model. Results are showed in the table below:

Table 4: NMAE values of the Random Forest Regressor on the test set

T	10	20	30	40	50	60	70	80	90	100
NMAE of RF	0.078	0.069	0.07	0.064	0.073	0.077	0.068	0.068	0.069	0.066

As we can observe from the table, the threshold that yields the lowest NMAE is T=40 with an NMAE equal to 0.064. That would imply that filtering the number of outliers based on a threshold equal to 40 would improve the performance of the Random Forest Regressor. Looking at the value of the table, the error keeps on fluctuating until T>70 where it stabilizes; this is reasonable as the number of outliers detected is very small when the threshold is high (see Fig8). We can also notice that the error does not fluctuate a lot even after removing outliers (i.e., the improvement is not drastic); this is due to the fact that random forest regressors are robust against outliers (the outliers end up by in their own nodes as they have higher values than other feature samples).

Task IV - Predicting the Distribution of Target Variables using Histograms

In this section, we predict the target variable by training a classifier that estimates the distribution of the target variable given a certain sample then computing the expectation from this distribution. To do so, we perform the following :

- First, the data is preprocessed according to the method that works best with our dataset (i.e., standardizing per column). Then, we perform outlier removal to keep 99% of the samples. This is achievable by setting the threshold to be T=53. Afterward, we perform supervised tree-based feature selection to keep the top 16 features. Finally, we perform a 70/30 split to get the training/testing sets.

- Every target continuous value of the DispFrames is mapped to a certain bin from a set of bins ranging from [0.5 30.5] with a bin size of 1. Afterward, we attribute to each sample a new label that corresponds to the center of the bin of this sample's target value. For instance, a sample with the DispFrames equal to 24 belongs to the [23.5 24.5[bin and is attributed the label "24".

- To predict the distribution of $P(Y|X)$, we train a classifier that can learn this conditional probability by learning a hypothesis that maps the sample's features to the new centers of bins. The project description advised us to experiment with the *RandomForestClassifier* provided by *sklearn* since it can compute the probability of the target labels given the sample features using the function *predict_proba*. According to sklearn's documentation[6], the predicted class probabilities are computed as the mean predicted class probabilities of the trees in the forest, where the class probability of a single tree is the fraction of samples of the same class in a leaf. Before training the model, we run a Random Search to perform hyperparameter tuning and search for the parameters of the Random Forest Classifier. These are the same parameters shown in Fig 3. The optimal values are the following: *n_estimators: 100, min_samples_split: 2, min_samples_leaf: 1, max_features: sqrt, max_depth: 40 and bootstrap: False.*

- We train the RF Classifier on the training set. For every sample in the testing set, we compute the probability distribution/density of the target value given that sample and we approximate the value of DispFrames by computing the expectation as follows $y_{expectation} = \sum_{label} Prob_{label} \times label$. We

compute the NMAE value on these results and we obtain an NMAE of **0.0682**.
Fig 9 shows the distribution of the estimation errors of this model:

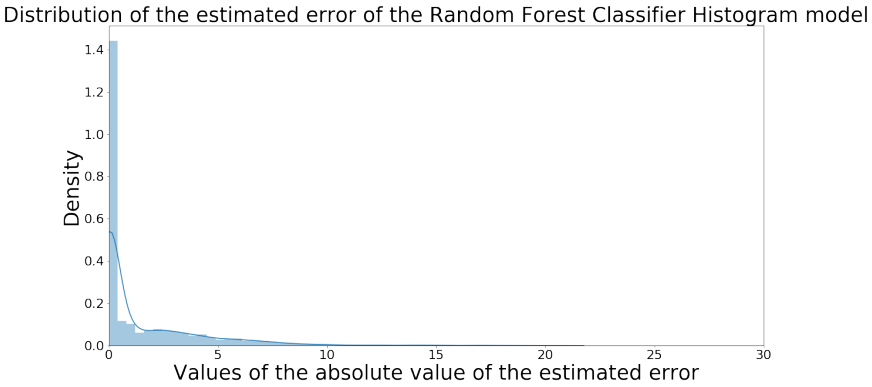

Figure 9: Distribution of the estimated error $|y - \hat{y}|$ where $\hat{y}$ is the expectation computed using the histogram

- For illustration purposes, we chose two random samples from the test set and we predict the distribution of the target variable. We visualize the expected value using a red line. These can be visualized below:

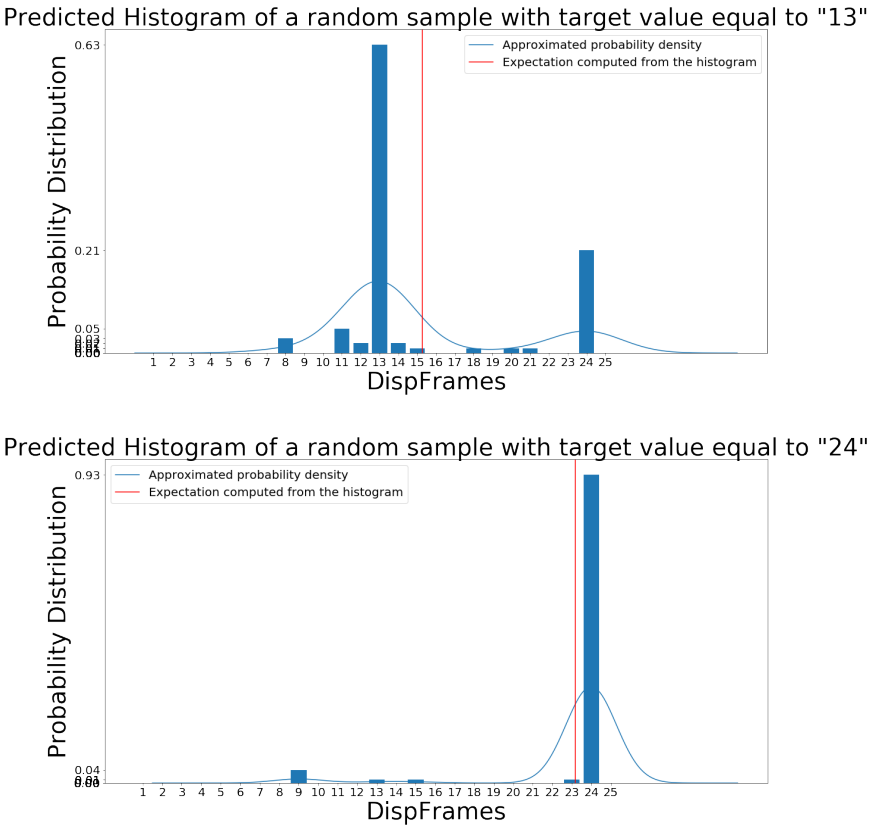

Figure 10: Distribution of the target variable for two random samples in the test set.

Based on the previous results, we can observe the following:

- The NMAE computed using the method proposed (0.0682) is very close to the NMAE of the random forest regressor computed in both Tasks II (0.0655) and III (0.065). This confirms that the work achieved in this report is consistent and correct.
- Furthermore, we can infer that using a random forest classifier and computing the target variable as the expected value of the probability distribution is equivalent to predicting using a random forest regressor. This can also be seen by looking at the distribution of the error of both models, which are very similar in terms of shape.
- Looking at the probability distribution of the randomly drawn samples, we can observe that the distributions have peaks around 13 and around 24. This can be explained by the fact that more than 96% of the target values lie around these two points (which affects the probability distribution). Furthermore, the highest peak of each sample (i.e., the highest conditional probability) matches with the sample's target value.

Optional Task V - Predicting Percentiles of Target Metrics

The goal of this task is to estimate the 20th, 50th and 95th percentile values of the target Y (DispFrames) using the estimator from Task IV. Let's describe how to compute the percentiles values from a given histogram.
We will define the same terminology defined in [7].

- q: q is defined as the percentage value of the metric that we want to measure (i.e., $q \in [0, 1]$) .
- a: a is defined as the value of the target variable at a quantile value equal to q.
- p(y): p(y) is defined as the probability density function that is approximated by the histogram computed from Task IV.

As we all know from the basics of probability, we can write q as follow: $q = \int_{y=-\infty}^{a} p(y)dy$. This integral is no other than the cumulative distribution function (cdf) of the target metric y, formulated as $q = P(a)$. Our task is to estimate the value of a given a certain quantile q and having a certain distribution $p(y)$ represented by a histogram, for every target value in the sample set. To do so, we do the following:

1. First, we estimate the cumulative distribution function from the histogram. This is done by going through the bins iteratively and computing the cumulative value of the probability. We denote by $P_k(y|x)$ the value of the cumulative value at the bin k.
2. Now that we have the values of the cdf computed for each bin, we need to find the bin in which the cdf is the closest to q. This is no other than finding the number k (bin number) in which $P_k(y|x) < q \leq P_{k+1}(y|x)$. Let b_k and b_{k+1} be the boundaries of the bin k.
3. Assuming that the cdf is linear within the same bin (i.e., we can model the cdf as a linear function written in the form $y = ax + b$), we can estimate the value of a from the histogram produced by x and at the quantile value q as follows: $a_m(x, q) = b_k + (q - P(y|x)_k) \times \frac{b_{k+1} - b_k}{P(y|x)_{k+1} - P(y|x)_k}$.

To get some insights and verify the approach, we apply this computation on one instance of the data set. Fig 11 shows the three quantile values computed as well as the histogram of the distribution inferred from the estimator.

We then perform the same computation on the first hour of data in the training set. Fig 12 presents a time series plot displaying the target value, 20th quantile, 50th quantile and 95th quantile values.

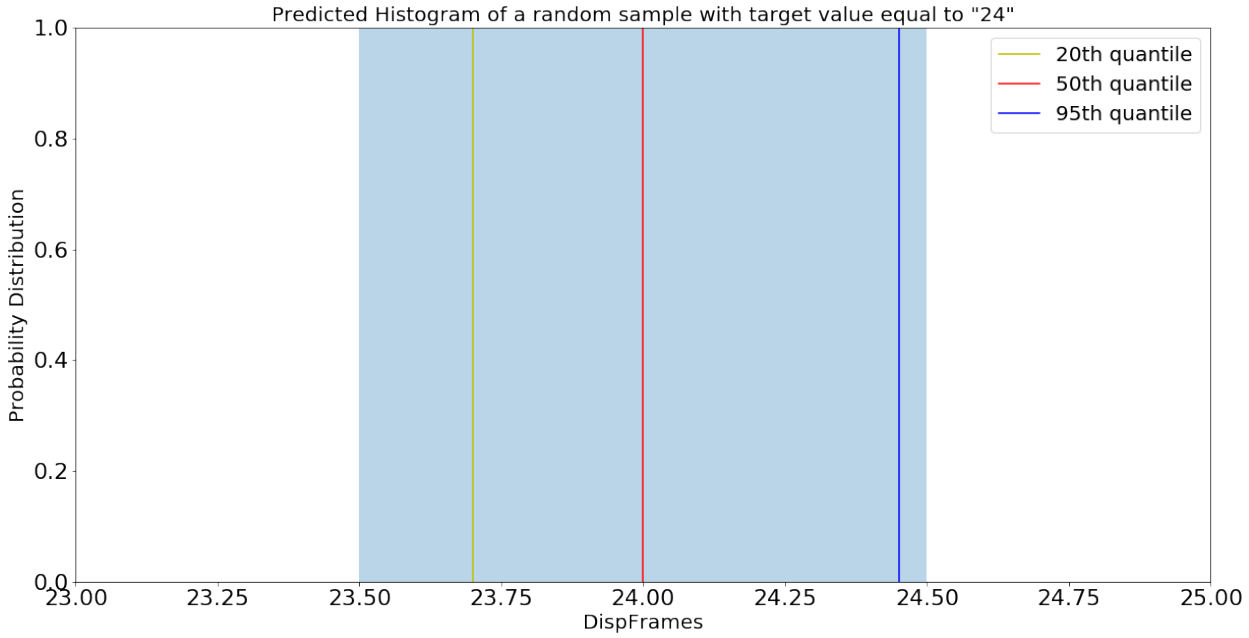

Figure 11: Histogram highlighting the probability distribution of a random sample as well as 20th, 50th and 95th quantiles values

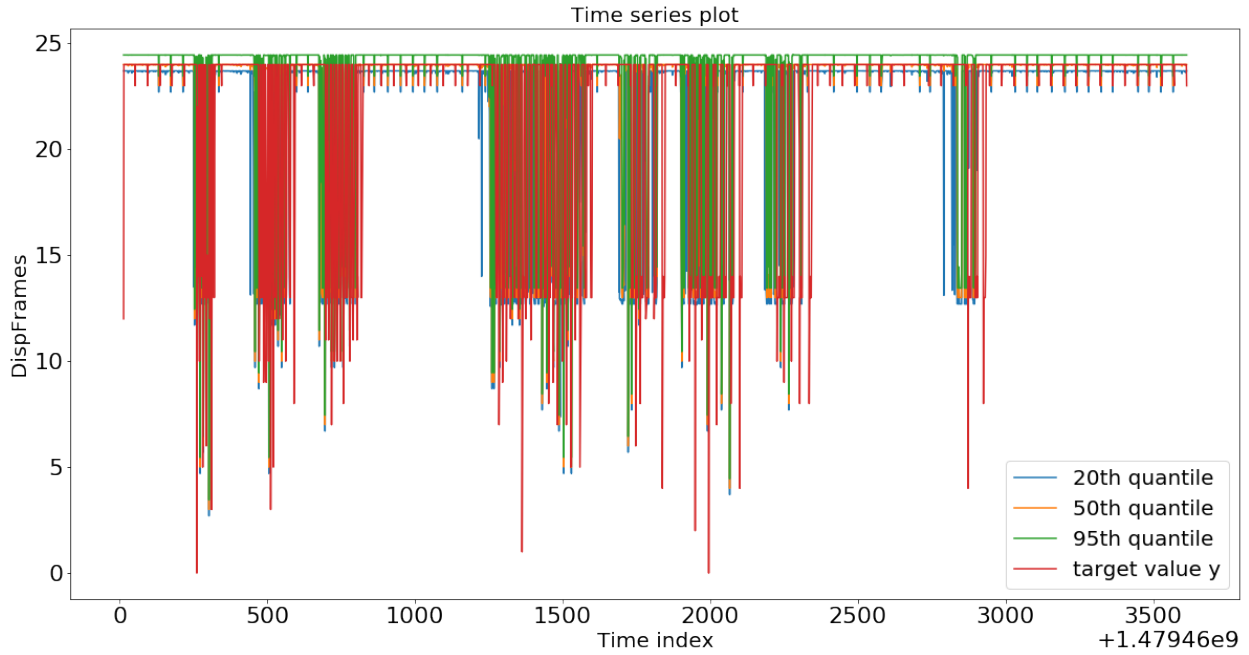

Figure 12: Time series plot of the 20th quantile, 50th quantile, 95th quantile and the target value

As we can see from both plots, it seems that the target value almost always coincides with the 50th percentile value. Furthermore, we can observe that the 20th percentile value is always below the target value while the 95th is always above it.

We proceed by evaluating the accuracy of the predicted percentile values using the Clivento-Cantelli Theorem introduced in the question sheet. This theorem proposes to estimate the goodness of the approximations by trying to compute the value of the percentages using the approximated values of a by evaluating the formula $\frac{1}{n}\sum_{t=1..n}\{y^{(t)} \leq a_m(x^{(t)}, q)\}$.
This formula counts the number of samples that have a target value that is smaller than the computed quantile for that sample, and divides it by the number of samples in the set. We compute this quantity on both the first hour set and the testing set. The results are displayed in the table below:

Table 5: Estimated percentage values from the quantiles generated using the approach presented

Models	Estimated q on first hour	Estimated q on Testing Set
q=0.2	0.130	0.0716
q=0.5	0.678	0.328
q=0.95	0.907	0.999

As we can see from the table, we were not exactly able to accurately predict the quantile percentage values. We can see that using the histogram to compute these values is not very accurate, as it underestimates the 0.2 and 0.5 quantiles, and overestimates the 0.95 quantiles on the test set. This can be attributed to the fact that the Display frame rate is takes integer values that are mostly either 24 or 13. This can also be attributed the fact that we have assumed a linear cumulative distribution function within the same bin. Reducing the bin size will make the interval of this assumption smaller and will result in more accurate approximations.

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)*, pp. 1–9, 2019.

[2] A. Jung, "A gentle introduction to supervised machine learning," *CoRR*, vol. abs/1805.05052, 2018.

[3] L. F. R. A. Torgo, "Inductive learning of tree-based regression models," 1999.

[4] H. Larochelle, D. Erhan, A. Courville, J. Bergstra, and Y. Bengio, "An empirical evaluation of deep architectures on problems with many factors of variation," in *Proceedings of the 24th international conference on Machine learning*, pp. 473–480, ACM, 2007.

[5] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. Feb, pp. 281–305, 2012.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[7] F. S. Samani, R. Stadler, C. Flinta, and A. Johnsson, "Conditional density estimation of service metrics for networked services," *IEEE Transactions on Network and Service Management*, vol. 18, no. 2, pp. 2350–2364, 2021.