EP2420 – Project 1
Estimating Service Metrics from Device Measurements

Léo Lebuhotel

November 24, 2023

Project Overview

The objective of this project is to investigate the service quality of a video-on-demand service using a service metric Y on the client side and device statistics X on the server side. The goal is to use machine-learning techniques to solve a regression problem. We have access to 3600 observations of 12 different features. We will train a model $\boldsymbol{M} : \boldsymbol{X} \rightarrow \widehat{\boldsymbol{Y}}$ approximating Y the best.

In the first task, we will explore the data, by looking at its statistics, and we will visualize it with different plots. We need to get familiar with the problem. Then, in the second task, we will use Linear Regression to estimate the service metrics from the device statistics and question this choice and we will study the influence of the training set's size on the estimation error of different linear regression models.

In the third and fourth tasks, we are going to reduce the number of useful features using different methods and we will reduce the dimensionality of the feature space by using Principal Component Analysis.

Finally, we are going to train a different kind of model : Neural Networks. We will train a basic neural network and then we will tune its parameters to obtain the best precision. We will compare the results of the linear model and the neural network model.

Background

We will be using linear regression. When data is assumed to be linearly distributed, we use linear regression to model the relationship between a scalar response and a one or multi-dimensional variable. In our case, our variable is 12 – dimensional. We can write :

$\hat{Y} = \theta_0 + \sum_{i=1}^{d} \theta_i x_i$ where $\theta = \{\theta_0, \cdots, \theta_d\}$ is the parameters vector and d the dimension of the variable.

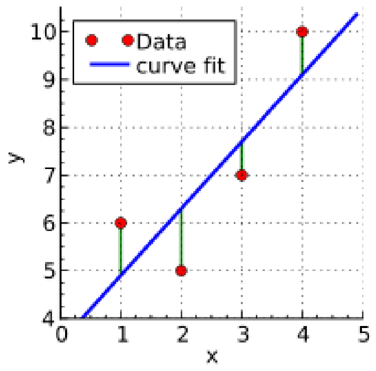


Figure 1 : Observations in red, and linear regression curve in blue. [1]

To find the ideal parameters, we will minimize the error between the model estimations and the observations $J(\theta) = \sum_{i=1}^{n} (\hat{Y}_i - Y_i)^2$. This function is convex and is minimized when its gradient is zero. We will use sk-learn's Linear Regression Model to obtain the ideal parameters.

Then, we will use neural networks. Neural networks consist of interconnected nodes organized into layers. Each connection between nodes is associated with parameters : weights and biases.

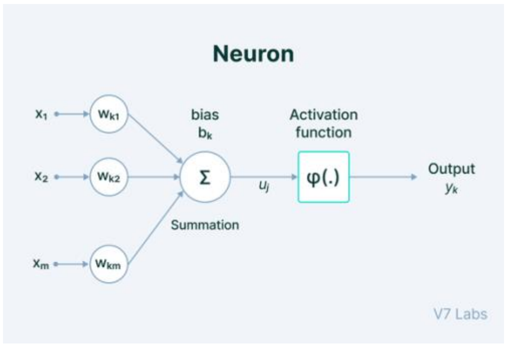


Figure 2 : Basic structure of a neural network. [2]

Each neuron, or perceptron, uses weights to sum its inputs, and then add a bias. An activation function is used to introduce non-linearity in the model and handle complexities that reflect the real-world. At the end, the output y_k of the neuron k is $y_k = \phi\,(\sum_{i=1}^{n} w_{ki}x_i + b_k)$ where n is the number of features of a sample, w is the weight vector of the neuron, and b the bias added to the perceptron. The weights determine the strength of the connections between nodes, while biases introduce an additional constant term.

Neural networks learn from data by adjusting weights and biases during training. The network's architecture, defined by the number of layers and number of units per layer, processes input through forward and backward passes. In the forward pass, data propagates through layers, producing predictions. The backward pass adjusts parameters to minimize the error via optimization algorithms, commonly stochastic gradient descent.

Parameters include weight and bias and change through the iterations, but other parameters are fixed for a training, and we need to tune them : the learning rate governs the step size during weight updates, affecting convergence speed ; regularization techniques, like L1 and L2, prevent overfitting by adding penalty terms to discourage excessive feature importance and we need to tune these penalty terms; the number of layers and the number of neurons per layer; the number of iterations needed for convergence. We can tune these by hand, by changing their value each time that we train the network. For that, we use a validation set, that is used during training not to adjust the parameters, but to visualize the performance of the network during training and avoid overfitting. We can change the hyperparameters and choose the value that performs better on the validation set. And only at the end, after hyper-parameter tuning, we can use the test set, that must remain unseen.

But there are techniques that allow us to tune the hyperparameters, not relying entirely on our intuition, but on mathematics schemes. We first need to list all the hyperparameters and all the values that we should try for them. We now have our hyperparameter space : each feature is a dimension of that space. Then we select a strategy to evaluate which combination of the hyperparameters is the best.

Grid Search exhaustively evaluates all possible combinations within the specified search space, Random Search randomly samples combinations within the search space, Bayesian Optimization uses probabilistic models to guide the search based on observed outcomes and Hyperband is a combination of random search and successive halving algorithms for efficient resource allocation. [3]

Data description

We use in this project a trace of 3600 observations, collected once every second from the system over the course of an hour. The Service metric Y corresponds to the Video Frame Rate (frame/s) on the client side.

Feature	Description
runq-sz	Run queue length
%%memused	Percentage of used memory
proc/s	Rate of process creation
cswch/s	Rate of context switching
all %%usr	Percentage of CPU utilization
ldavg-1	Load average for the last minute
totsck	Number of used sockets
pgfree/s	Rate of freeing pages
plist-sz	Number of tasks in the task list
file-nr	Number of file handles
idel/s	Number of IP datagrams delivered to IP user
tps	Rate of I/O requests to physical devices

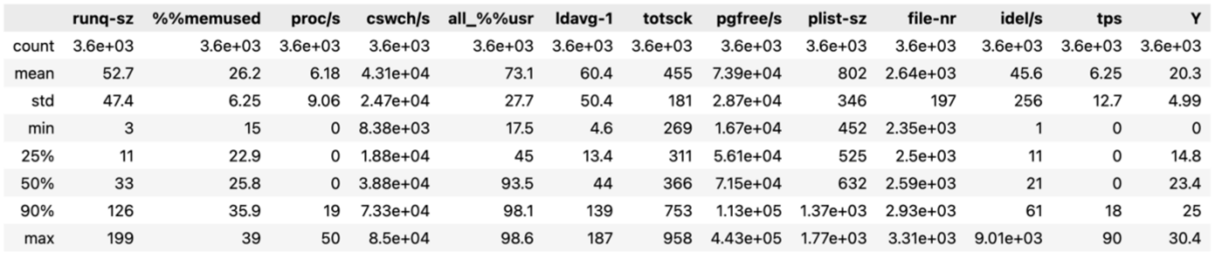
Table 1 : Device Statistics X on the server side. Rates are given in units per second.

There are no duplicates or missing values in the Device Statistics. We can preprocess the data by normalizing it (for each feature, subtract the mean of all the samples and divide by the standard deviation).

We will visualize the data during Task 1.

Task 1

We will use machine-learning techniques to train a model $\boldsymbol{M} : \boldsymbol{X} \rightarrow \widehat{\boldsymbol{Y}}$ approximating Y the best. In the first task, we will explore the data, by looking at its statistics, and we will visualize it with different plots. We need to get familiar with the problem.



	runq-sz	%%memused	proc/s	cswch/s	all_%%usr	ldavg-1	totsck	pgfree/s	plist-sz	file-nr	idel/s	tps	Y
count	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03	3.6e+03
mean	52.7	26.2	6.18	4.31e+04	73.1	60.4	455	7.39e+04	802	2.64e+03	45.6	6.25	20.3
std	47.4	6.25	9.06	2.47e+04	27.7	50.4	181	2.87e+04	346	197	256	12.7	4.99
min	3	15	0	8.38e+03	17.5	4.6	269	1.67e+04	452	2.35e+03	1	0	0
25%	11	22.9	0	1.88e+04	45	13.4	311	5.61e+04	525	2.5e+03	11	0	14.8
50%	33	25.8	0	3.88e+04	93.5	44	366	7.15e+04	632	2.59e+03	21	0	23.4
90%	126	35.9	19	7.33e+04	98.1	139	753	1.13e+05	1.37e+03	2.93e+03	61	18	25
max	199	39	50	8.5e+04	98.6	187	958	4.43e+05	1.77e+03	3.31e+03	9.01e+03	90	30.4

First, we load the data from the .csv files using the *pandas* library. We use the function describe() on the datafile to obtain the table 1.

Table 2 : Basic statistics for each feature X and the target Y

Table 2 shows that there are not outliers, since all the data is in an interval of $[\mu - 3\sigma; \mu + 3\sigma]$ for each feature. The service metrics Y

Then, we use impose a condition to certain columns to filter the data, and we can count that there are 449 observations with CPU Utilization lower than 50% and memory utilization lower than 25%. It is low since there are in total 3600 observations, the server is thus not overloaded. There are also in average 321 used sockets for observations with less than 50 000 context switches per seconds. We can conclude from these numbers that our system is not under heavy stress in terms of resources and the load average for the last minute “ldavg-1”, with a mean of 60.4%, confirms that. The other features also confirm that.

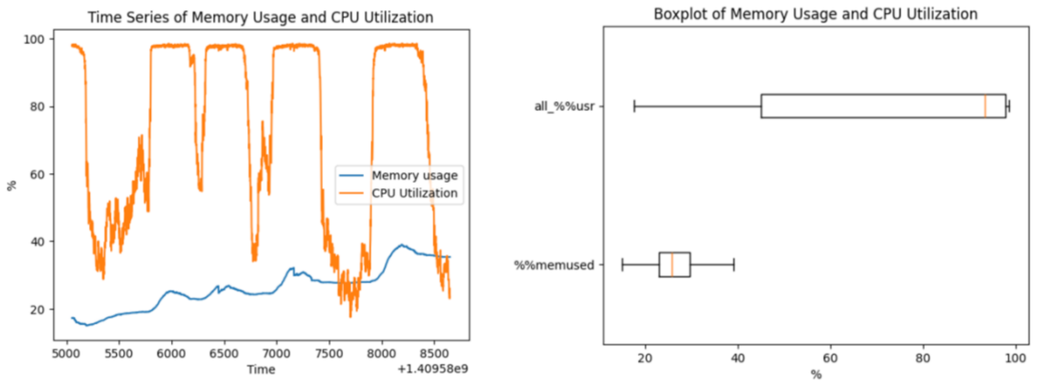


Figure 3 : Memory Usage and CPU Utilization comparison. Time series of the percentage of utilization (a) and boxplot of the percentage of utilization (b)

Figure 3a shows that the CPU is periodically almost 100% used, and periodically around 50%, and figure 3b is interesting to show that the CPU is far more demanded than the memory. That explains the huge variance on the CPU utilization. The memory usage is getting more and more used over time but is still very low (less than 40%). The demand on the resources is variating over time, and the CPU is involved in load balancing, because the load balancers may distribute incoming requests unevenly. The CPU is periodically processing high demanding tasks, while the memory utilization is efficient, and the system has sufficient memory available for its workload. It is possible that the system is incorrectly dimensioned because there are a lot of periods where the CPU Utilization is 100% and that should not be the case.

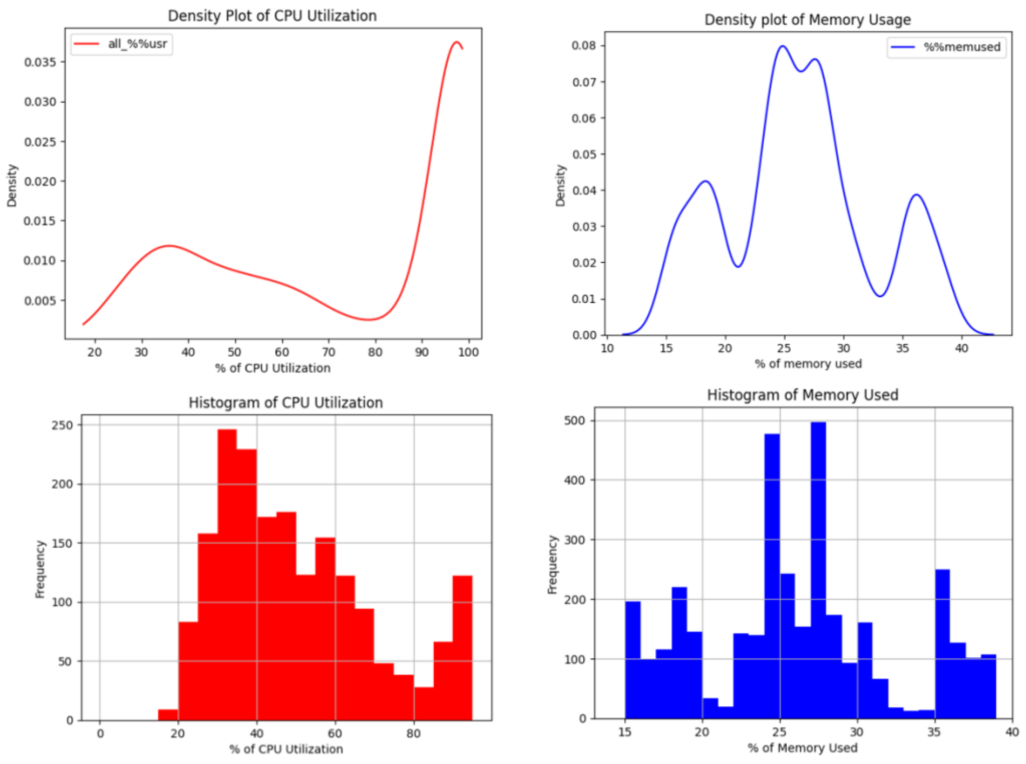


Figure 4 : Density plot and histogram of CPU Utilization (left column) and of Memory Usage (right column).

The histograms and density plots are telling us the same thing. Figure 4 shows us that there is a high utilization of CPU in most observations and that there is a low memory utilization, centralized around 30%. It is confirming what we saw previously. The CPU is actively processing tasks as video encoding, content delivery or request handling. This utilization is predictable, because a VoD service has often periods of high utilization, demanding high resources. Furthermore, the memory utilization is efficient, and the system has sufficient memory available for its workload. But the CPU utilization reaches 100% often, and the system is overloaded, resulting in a lower video frame rate. It is a problem ! The histogram of CPU utilization doesn't reflect the distribution on the density plot because the width of the bins is pretty high (5%) and does not allow to capture the distribution correctly.

Quiz 1

When we are using linear regression, we need the data to be linearly distributed. If it is not the case, we are going to have a low accuracy and a high error. Else, the prediction error is minimized, thus the accuracy is maximized.

Task 2

We now want to train and test a linear regression model, to estimate the video frame rate of the VoD service using the device statistics of the server. We split the data into a train and a test set, and we train the model on the train set using sk-learn! We then estimate the predictions on the test set and compare them to the measured service metrics. Then, we are going to study the influence of the training set's size on the estimation error by training linear regression models on train sets with different sizes and see what happens on the Normalized Mean Absolute Error (NMAE)

θ_0	θ_1	θ_2	θ_3	θ_4	θ_5	θ_6	θ_7	θ_8	θ_9	θ_{10}	θ_{11}	θ_{12}
-8.85 e-4	-2.87 e-2	8.36 e-2	7.09 e-3	1.94 e-4	1.25 e-1	-1.28 e-2	1.74 e-2	-1.32 e-5	-1.22 e-2	2.69 e-2	7.30 e-5	-6.92 e-4

Table 3 : Coefficients of the linear regression model

We obtain these coefficients. Let's see if the model is correct by checking at the accuracy. We compute the *Normalized Mean Absolute Error* $NMAE = \frac{1}{\bar{y}}(\frac{1}{m}\sum_{i=1}^{m}|y_i - \hat{y}_i|)$, whereby $\hat{y}_i$ is the model estimation of the measured metric y_i using the linear regression model, and $\bar{y}$ is the average of the observations on the test set. We obtain a NMAE of 8.92% for the linear model, and 23.1% for the naïve model, that corresponds to the average of the observations on the train set. Thus, the linear model is more accurate than the naïve model.

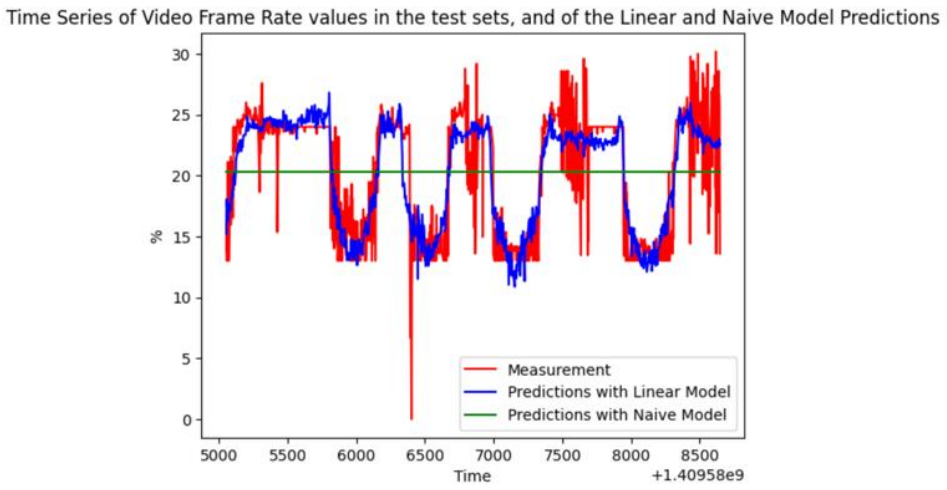


Figure 5 : Time series of the test set observations (red), test linear predictions (blue), test naïve predictions (green)

The previous NMAE for the naïve model should not be underestimated. This value looks low (23.1%), because the estimations are bounded, but when we look at figure 5, we clearly understand that the linear model is more accurate because it models the behavior of the system over time, unlike the naïve model.

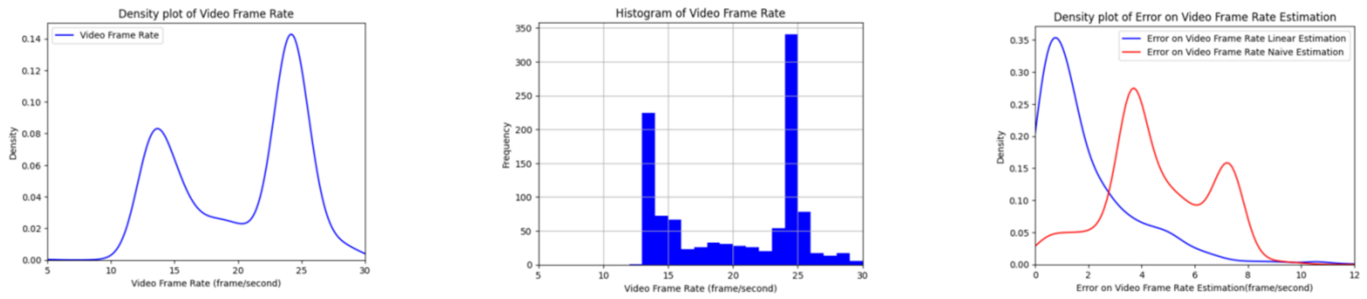


Figure 6 : Density plot of Video Frame Rate (a), Histogram of Video Frame Rate (b), Density plots of Error on Video Frame Rate Estimation (c)

Figure 6b shows us that the error on the video frame rate estimation using the linear model is lower than with the naïve model, and it confirms everything we saw before. It is more accurate than the naïve model, and we should prefer using linear regression, because it performs good, as we saw on figure 5.

We can also acknowledge that there are some observations where the video frame rate is lower, around 15 frames/second, reflecting a high utilization of resources, and a malfunctioning of the system that needs to be fixed.

Now, we are going to analyze the influence of the training set's size on the estimation error.

We have 3600 observations and create a test set T of 1000 observations randomly selected. We will create from the 2600 remaining observations six different train sets $S_1, \cdots, S_6$ by selecting uniformly at random 50, 100, 200, 400, 800 and 1600 observations. We train a linear model using each of these train sets and compute the NMAE on the test set for each model.

Train set	S_1	S_2	S_3	S_4	S_5	S_6
Size	50	100	200	400	800	1600
NMAE	0.125	0.102	9.29 e-2	9.05 e-2	8.95 e-2	8.84 e-2

Table 4 : NMAE for each train set

We can see from Table 4 that the bigger the size of the train set is, the lower the error is. We only trained a model for each training size once. To confirm the tendency, we need to perform the above 50 times, to limit the influence of randomness on the results. For each train set size, we will randomly select the number of observations and train a linear model with this set. Then, we compute the NMAE, which gives us 50 NMAE values for each train set size. We will be able to conclude by looking at the mean and the standard deviation for each size. The best way to visualize the results is to plot them.

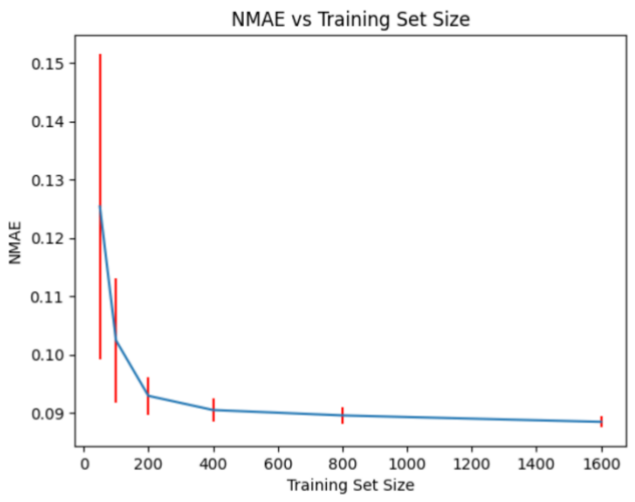


Figure 7 : Normalized Mean Absolute Error (NMAE) on the test set vs size of the training set.

Figure 7 confirms us that the size of the training set has a high influence on the error on the test set. Not only the error is higher when the size of the training set is low, but the standard deviation is higher. It is logical, since we are using a little split of the data, that does not represent the inherent distribution correctly. That means that we can have splits that represent the data correctly, resulting in a lower error, but other splits that do not represent the data correctly, and the error increases. Since the number of combinations of these splits is high, it results in a high standard deviation.

Furthermore, the distribution of video frame rate (frame/second) is not distributed the same in each of the 50-sized splits, and it is not distributed the same way as in the test set. Thus, the data is not represented correctly, and the error is high.

When the size increases, the variance gets lower, and the training set represents the true distribution more accurately. It informs us that it is very important to have a train set the bigger possible to have correct results.

Quiz 2

The computational complexity of the optimal method is $O(2^n)$ since we are computing the NMAE on the test set for each of the feature subsets, and we know that a feature subset of size n has 2^n subsets. As the number of features increases, the computational complexity grows exponentially. We can select the subset of features that minimizes the NMAE.

The computational complexity for the heuristic method is $O(n)$ since we are using a univariate feature selection technique. We compute the Pearson correlation for each feature with the target y, and we only have n computations. We can select the features that have the highest Pearson correlation with the target.

Task 3

Optimal method

The objective of this task is to effectively reduce the number of features needed to accurately estimate the video frame rate Y. We are using an Optimal method, and a heuristic method. We will here focus on the optimal method. We have here $n = 12$ features, and we can create $2^{12} = 4096$ subsets of these features. For each of these subsets, except the empty subset (because we can't train a model without feature !), we will train the model on the corresponding features, and then compute the NMAE on the test set.

The minimum NMAE is 0.0853 and is obtained for the following subset of features : [*'idel/s', 'file-nr', 'plist-sz', 'pgfree/s', 'all_%%usr', 'cswch/s', '%%memused'*]. This information changes every time we run the code because the train/test split relies on randomness. The computing time of the optimal method is 9.2 seconds in my environment.

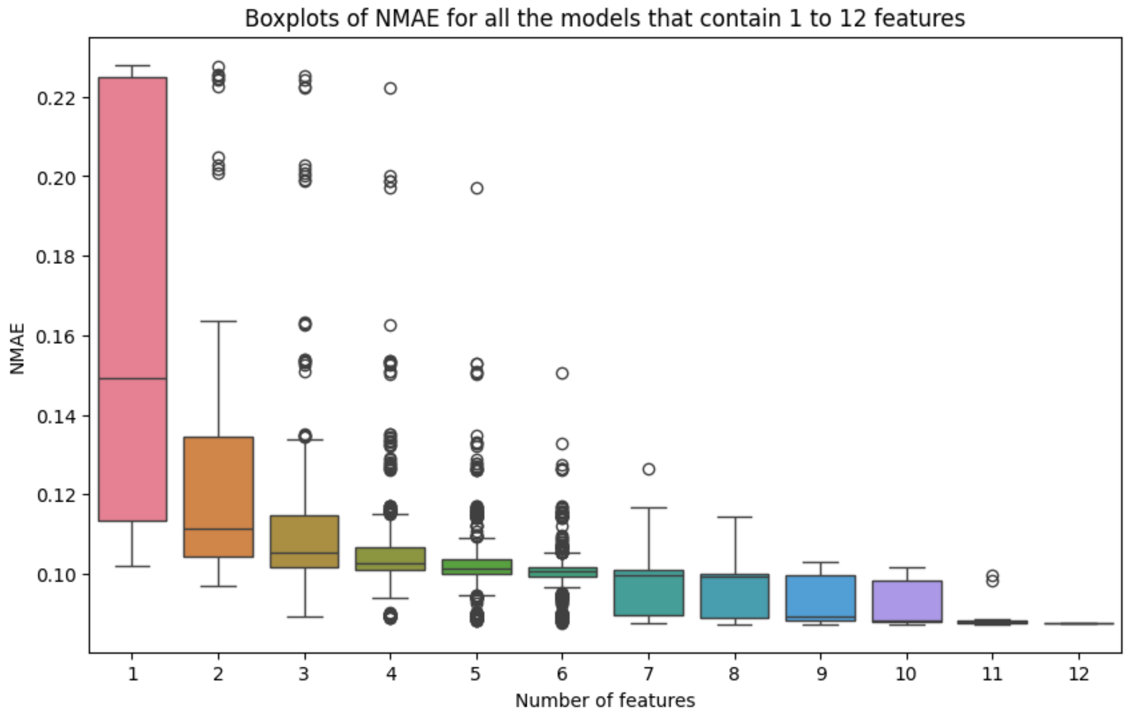


Figure 8 : Boxplots of NMAE for all the models that contain 1 to 12 features.

From this boxplot, we can see that the more features we consider, the lower is the error in average. It is logical, because we have a 12 features problem, and considering only 2 or 3 that are not relevant, the error will be high. But we see that the minimum error is acceptable for models with 3 or 4 features, and that is very interesting. It is logical that that the average is not very good, because we consider all combinations of 3 or 4 features, and some of these combinations don't give any information on the problem. But that informs us that there exist combinations where the error is acceptable, and not so far from the 12 features problem.

We could look at all 3 features model and take the combination of features that minimizes the error. We could also look at the 7 and over features model, but if the number of features is too close to 12, then the feature reduction problem loses interest. And that is logical that we got the subset of features [*'idel/s', 'file-nr', 'plist-sz', 'pgfree/s', 'all_%%usr', 'cswch/s', '%%memused'*], because 7 models feature have the lowest minimum error (for this run of code, this can change due to the randomness in the train/test splits).

Heuristic method

We are going to reduce the number of useful features using a Heuristic method, that consists in computing the Pearson Correlation of each feature with the Y value on the training set. We can then classify the features according to the values of this correlation, and we will compute NMAE for the best feature, then the two best features, and so on…

We take each feature of X and compute the Pearson correlation of the feature with the target value, on the training set. For feature j and target y, this correlation coefficient is computed as :

$$r_{x_{:,j},y} = \frac{1}{m}\sum_{i=1}^{m}(x_{i,j} - \bar{x_{:,j}})(y_i - \bar{y})/(\sigma_{x_{:,j}} * \sigma_y)$$

Whereby $\overline{x_{:,j}}$ and $\bar{y}$ are sample means and m is the size of the training set ; $\sigma_{x:,j}$ is the standard deviation of the $x_{:,j}$ values, namely $\sigma_{x_{:,j}} = \sqrt{(\frac{1}{m}\sum_{i=1}^{m}(x_{i,j} - \bar{x_{:,j}})^2)}$, and likewise for σ_y.

The correlation values are between -1 and 1. To compare each feature, we take the square of the correlation value, and we sort the new values. The highest squared correlation corresponds to the top feature, the second highest squared correlation to the second top feature, etc.

Feature	runq-sz	plist-sz	totsck	cswch/s	ldavg-1	file-nr	all_%%usr	%%memused	proc/s	idel/s	tps	pgfree/s
Squared correlation	0.686	0.669	0.662	0.598	0.575	0.513	0.3359	0.101	2.22 e-2	1.09 e-2	1.14 e-3	2.45 e-4

Table 5 : Squared correlations of each feature, sorted in a descending way.

From Table 1 we can conclude that the feature "runq-sz" is the top feature. We will then compute 12 different linear models. For the first model, we will consider only the top feature, for the second model we will consider the two top features, etc. We train each model on the train set, and then we do predictions using the test set. We compare these predictions to the real labels, and we compute the Normalized Mean Absolute Error (NMAE). We have then 12 different NMAE values, one for each number of top features used. We can plot the NMAE against the number of top features considered.

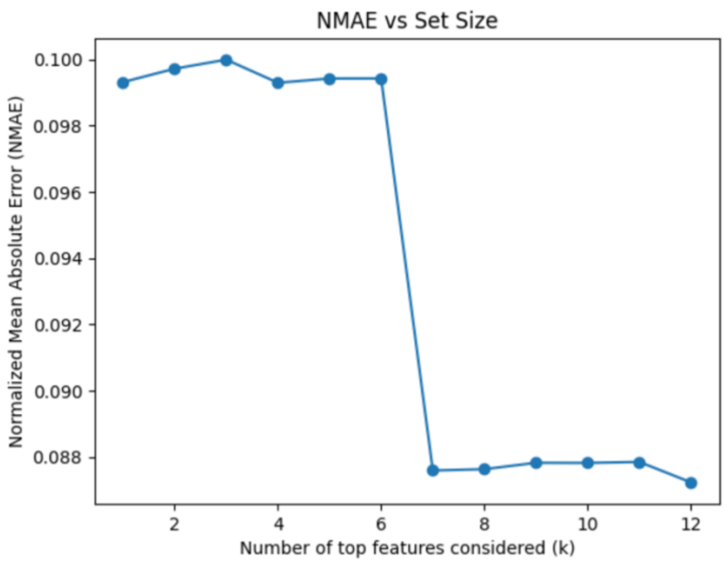


Figure 9 : Normalized Mean Absolute Error vs Number of top features considered.

On Figure 9, we can see that the NMAE is almost constant when we consider six or less features, and then there is a notable negative step when we consider seven or more features. From this figure, we can conclude that it is better to consider one feature only rather than six, because we obtain the same error for less computations. The best number of top features to consider is seven, and it is useless to consider more because the error will be roughly the same. The Heuristic method is fast, and it takes only 0.2 s to compute the Pearson Correlations between all pairs of a feature column and the target column.

We will now compare the Optimal and the Heuristic methods.

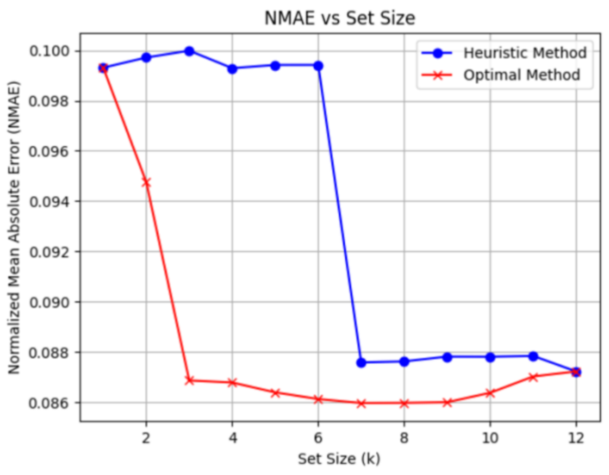


Figure 10 : NMAE vs Number k of Top features considered (for the Heuristic method), Size k of the subsets of features (for the Optimal method)

Figure 10 shows that the Optimal method can perform better with a low number of features of considered. Its global performance is better than the Heuristic method, and it achieves a very low minimum error for subsets of size three and more. The Heuristic method needs to consider seven features to achieve close (and still not as good) performances. When we consider at all the twelve features and at only one feature, both methods have the same error. It is logical because the subsets of features used are the same.

The Optimal method aim to find the best subset of features based on minimizing the prediction error and the Heuristic method involves making decisions based on intuitive judgments without exhaustive search. But the Heuristic method is still very useful and because it requires less computational resources, even if it might not guarantee the identification of the most optimal subset of features. The Optimal method considers all possible subsets, trains a model, and predicts target values for each of these subsets, it is a more empiric method. We saw that the computational time for the Optimal method as 9.2s, whereas it is only 0.2 s for the Heuristic method, that is based on Correlation computation.

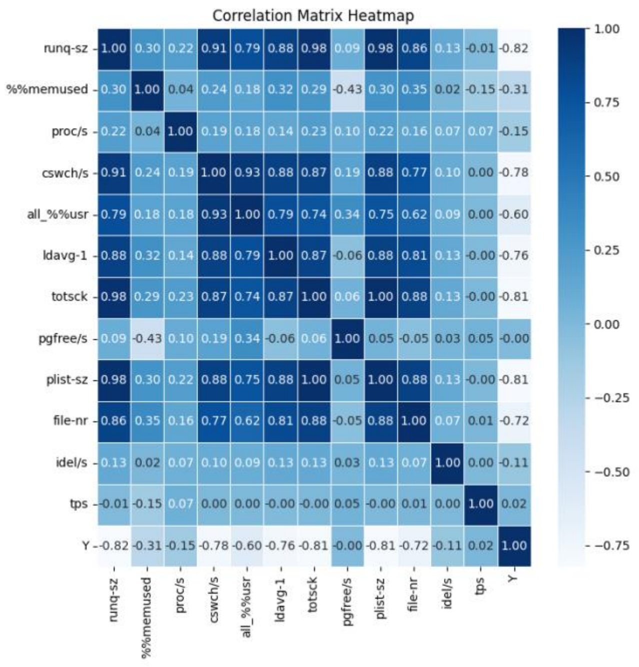


Figure 11 : Correlation Matrix Heatmap

From Figure 11 we can have very useful and interesting information. The Heuristic Method is based on the Correlation between each feature and the target Value Y. If two features have a high correlation (near -1 or 1), we can consider that they carry the same information, thus we can get rid of one.

By looking at the Correlation Matrix Heatmap, we can see that there is a low correlation between feature “runq-sz” and features “”%%memused”, “proc/s”, “pgfree/s”, “idel/s”, “tps”. The correlation between feature “runq-sz” and the other features is very high, and they are useless.

The correlation between all the selected features is very low, and we can see that there are in total six “independent” features. But if we look at the correlation between these selected features and the Video Frame Rate, we can see that the correlation is very high (-0.82) for “runq-sz”, then lower (-0.31) for “%%memused” and still lower (-0.15) for “procs/s”.

The other selected features have very low correlation with the target value. That is why the Optimal method’s error reaches a minimum for 3 features, and then stays constant, because the added features don’t add any information.

Quiz 3

The PCA method is an unsupervised method and don’t need information about the class labels. Its goal is to maximize the variance of the distribution of the data along principal components. Furthermore, it provides a linear transformation, that means that it is very easy to compute the eigenvectors and the eigenvalues of the covariance matrix. That also means that it may not capture non-linear relationships. Anyway, we are using linear models here, and we have seen in previous tasks that the prediction error is low, meaning that they are linear patterns in the data.

PCA method performs Dimensionality Reduction, while the Optimal and Heuristic methods perform Feature Selection. That means that we get rid of certain features. The PCA still considers all the features but projects the 12-dimensional (in our case) data points onto a lower dimensional space. Since it is a linear method, the convergence is assured. It is very easy to implement and does not require a lot of computational resources, compared to the Optimal method, that may in practice outperform the PCA method.

Task 4

In this task, we will perform Principal Component Analysis (PCA). It is a traditional method for linear dimensionality reduction using Singular Value Decomposition of the data to project it to a lower-dimensional space. The principal axis of the lowest dimensional space (1-d) is the eigenvector of the correlation matrix of the data, associated with the highest eigenvalues. The two principal axis of the 2-d space are the first two eigenvectors of the correlation matrix, associated with the two highest eigenvalues, etc.

We perform PCA on the training set and produce 12 different linear mappings that reduce the dimensionality of the feature space from 12 to k = 1,2,…,12. For each k, we map the original training set to a training set in the reduced space. This reduced space has been calculated by using the training set. We use these mappings to map the original test set to test sets in the reduced space. For each k, we train a linear regression model on the mapped test set, and we compute the NMAE.

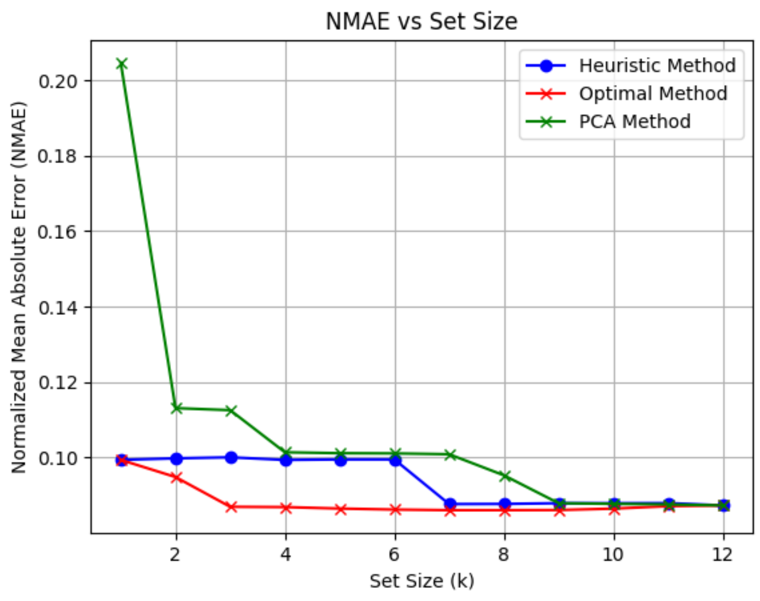


Figure 12 : NMAE vs Number k of Top features considered (for the Heuristic method), Size k of the subsets of features (for the Optimal method), Dimensionality k of the reduced space (for the PCA method)

From Figure 12, we can see that the Optimal method has the best results in terms of NMAE. The error is lower, and the methods needs a very low k value to converge to its minimum. It is the most effective at selecting the most relevant features and thus results in the most accurate predictions.

The Heuristic method has also good results, and converges to the same minimum, but needs a higher value of k.

Finally, the PCA method has a very high error compared to the two other methods when k is lower than 3, and it needs a much higher value of k to converge to a minimum error (k = 9). That means that we can reduce the dimensionality only from 12-d to 9-d if we want to have a good accuracy. Since the principal components are linear combinations of the original features and are chosen to capture the maximum variance in the data, this method might not always give the most accurate predictions because it assumes that the directions with the largest variances are the most informative, which might not be the case.

In conclusion, the Optimal method is the more accurate method since it achieves a lower error, but it takes a lot of time to compute, almost 100 times more (9.2s) than the Heuristic method or the PCA method (0.1 s for the PCA method and 0.2s for the Heuristic method). Since the complexity of the Optimal method grows exponentially with the number of features, we will rather use this method on low dimensional feature spaces. We will also use this method in the scenarios where we need a very high accuracy and to capture complex relationships in the data (and we have the resources to do it).

The Heuristic method is less computationally expensive than the Optimal method and is very efficient of datasets with a moderate number of features. When we need a quick and interpretable analysis, this is the best method.

The PCA method is very efficient even with many features and considers all the features. It is not a feature selection method, unlike the previous methods. If the underlying data distribution is linear, PCA method is very efficient. We can also use it to visualize the data more clearly, since we can project it on a lower 2-d or 3-d space.

We have studied the influence of the number of features k needed to obtain a good accuracy, for each method, and we compared the three methods and spotlighted their strengths and their weaknesses.

Quiz 4

There are different strategies to do hyperparameter search : Grid Search, Random Search, Bayesian Optimization, Hyperband…

Random Search is a hyperparameter optimization strategy where hyperparameter values are randomly chosen from a predefined search space. Unlike Grid Search, which systematically evaluates all combinations, Random Search explores the hyperparameter space by randomly selecting configurations.

This strategy is more fitted in scenarios with a large hyperparameter search space because it efficiently explores diverse combinations of hyperparameters without a high computational cost. Each randomly sampled configuration is trained and evaluated independently, making Random Search a flexible and resource-efficient strategy for finding optimal hyperparameter values.

Task 5

We are now going to use a different model : a Neural Network Regressor, which is a nonlinear function. The performances of the linear model were good, and it seems that the data is linearly distributed. Let's see what happens when we introduce non-linearity in our problem.

We split the data into a train, validation, and test set. The validation set is useful for hyperparameter tuning during training and can also prevent the Neural Network from overfitting. The train set and validation set represent both 40% of the data, and the test set represents 20%. We will first configure a Neural Network and train a model without tuning the hyperparameters very deeply. We are going to modify the number of layers and the number of neurons per layer, by hand, and it relies mostly on intuition.

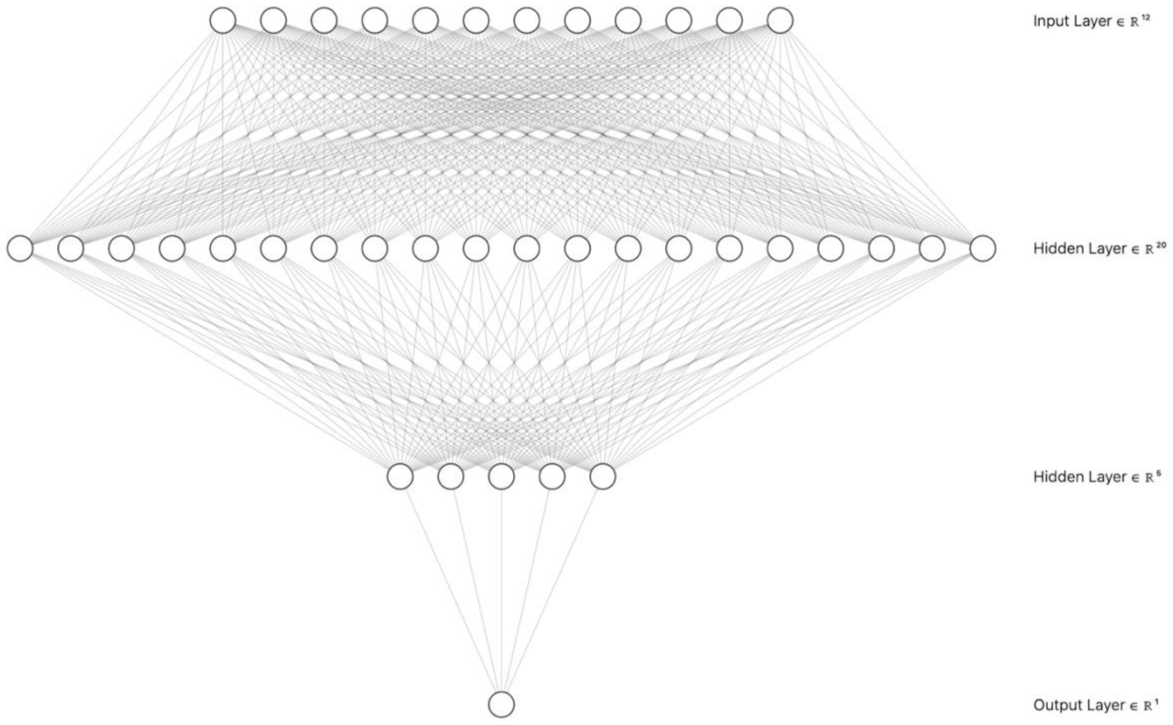


Figure 13 : Overview of the first structure.

There are 2 hidden layers with respectively 20 and 5 neurons. The learning rate is 0.01 and the learning algorithm is SGD. There is L1 regularization on each layer with lambda = 1e-5. The activation function of each layer is 'ReLu' except for the last layer where it is 'Linear'. I changed the structure by hand, and this provided a good accuracy. I used the 'Keras' library to create the neural network, train it and obtain the predictions.

This model has a test NMAE of 11.2% . This is not better than the previous linear model with 8.91% of NMAE but it is already good, and it is still better than the Naïve model, with a NMAE of 22.1%.

We will now tune the model to have a lower error. We are going to use the 'Keras' library and especially the package 'Keras_tuner'. We create a function 'build_model(hp)' that allows us to write the same code as before, but now we can specify the number of layers, the number of neurons per layer, and all the hyperparameters as variables. We then create a 'tuner' that will use this variable model and conduct hyperparameter search with the specified strategy. I did three tunings with three different strategies : Random Search, Bayesian Optimization and Hyperband. I kept the best model, based on estimation error.

We still need to tune the tuner… by hand ! I had to specify the minimum and maximum values for each hyperparameter, the step value. So, I ran my code multiple times by changing the number of layers (from [1,3] to [1,6]), etc. We don't want a too dense hyperparameter space because the computing time would be too long. Or, in the case of Random Search, we would maybe focus on too bad combinations of hyperparameters, and train multiple models with 7 layers when only 4 or less could be enough !

After multiple runs of the code, I sticked to these borders for the hyperparameters :

- **Number of layers** : from 1 to 5.
- **Number of neurons per layer** : from 4 to 96 with a step of 4.
- **Normalization strategy** : L1 or L2, with a lambda value from 1e-5 to 1e-1 (and a logarithmic scale).
- **Learning rate :** 0.1, 0.01 or 0.001.
- **Optimizer** : SGD if the learning rate is 0.1, Adam in the other cases.

Note that I didn't use a variable for the number of epochs needed for convergence, because I introduced in my code an early stopping condition that allows to stop the neural network training if the loss does not decrease significantly after a few epochs. It allows faster computing.

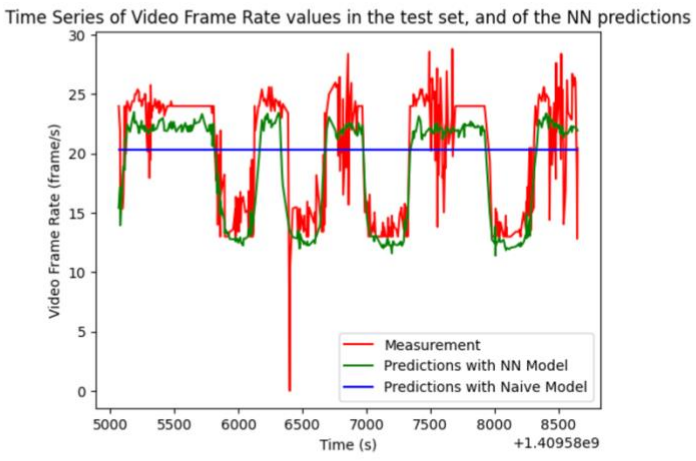
I then did three hyperparameters tuning with the three strategies and I obtained these results.

	Random Search	**Bayesian Optimization**	**Hyperband**
NMAE on the Test Set (%)	8.06	8.17	7.56
Elapsed Time	5m20s	6m 11s	2m29s

Table 6 : Results of the different hyperparameter search strategies.

From Table 6 we can conclude that the Hyperband strategy is the best for our case. It has the best NMAE on the Test Set and requires less time than the other methods. We will keep the best parameters found with Hyperband and use them for our model. We can see that every strategy finds hyperparameters combinations that are better than both the Naïve model (22.1%) and the Linear Regression Model (8.91%). Neural Networks with hyperparameters tuning are indeed better than Linear models for regression tasks.

We will now compare the predictions of the Naïve model, the first Neural Network model and the tuned Neural Network model, on a time series plot.



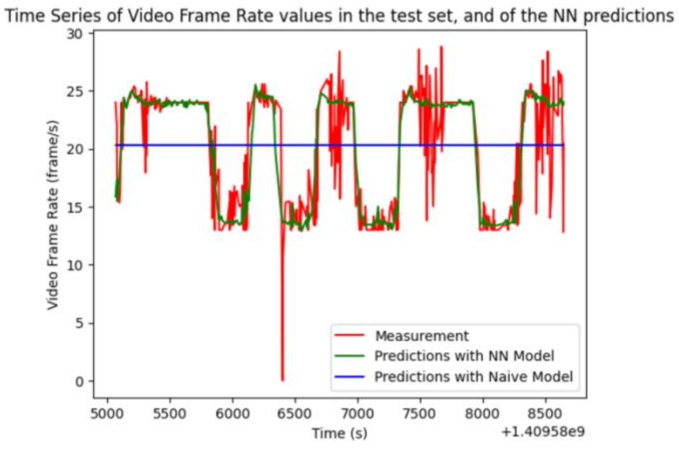


Figure 14 : Time Series Plot of Video Frame Rate measurements and predictions for the Naïve Model, and the First Neural Network Model (left) and the tuned Neural Network model (right).

From Figure 14, we can see that the Naïve Model is not preferable at all. The NMAE is "low", but it has no temporal flexibility and does not represent the data variations over time.

We can also see that the first NN model was pretty good and a low positive offset (that we can add via a bias term on the last layer) would make it even more accurate. It represents the variation of the Video Frame Rate over time correctly but adds a little bit of noise. The data is already noisy because we randomly selected samples for the Test set.

The tuned NN model is more accurate and sticks to the data correctly. There is very few noise and it does not consider the outliers created by the train/validation/test set split. This model is very satisfying.

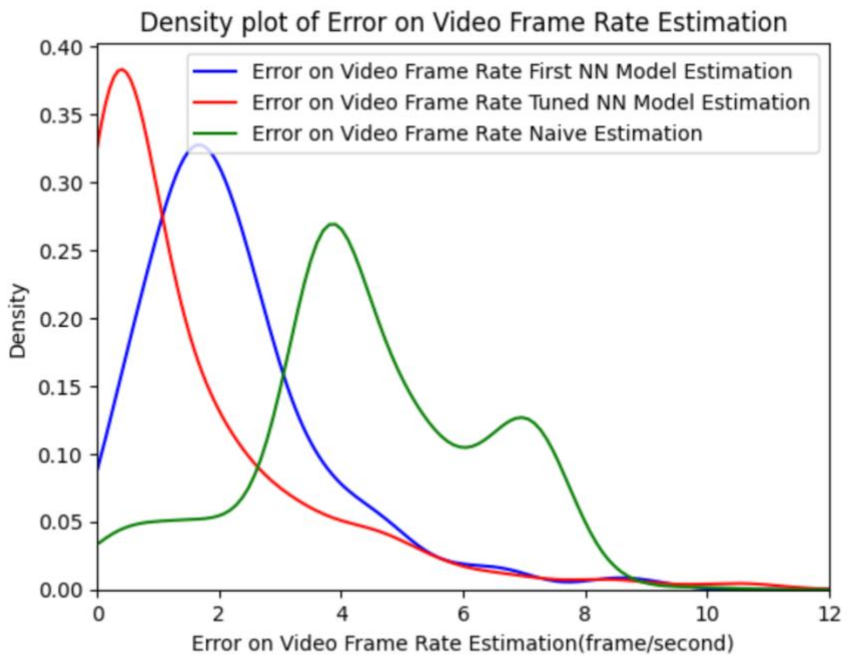


Figure 15 : Density plot of Error on Video Frame Rate estimation for the tuned NN model (red), the first NN model (blue) and the Naïve model (green).

From Figure 15, we can clearly see that the tuning of the hyperparameters of the Neural Network model was very efficient, because it has an average error on the prediction of 0.75 frame/s, when the average Video Frame Rate measurement value is 20.3 frame/s. That represents an error of 3.7%. The first Neural Network model has also a small error, but we will prefer the Linear model since it has a lower error and is much easier to implement. The Naïve method is just here to provide information on what a "bad" value of error is.

Hyperparameter tuning is fundamental when using models with a large number of hyperparameters. Neural Networks are very efficient but need to be tuned in order to be better than the other models. We saw that the Linear model has a very low error on the Test Set, and the first NN model didn't do better. Only after tuning the Neural Network was able to outperform the Linear model. The key strength of Neural Networks is that we can use them when there are linear patterns but also non-linear patterns in the data. It is important to get familiar with them because they can have a good accuracy in a lot of different situations, whereas we should stick to linear distributed data when using a Linear regression model.

Discussion

During this project, we have seen a lot of Data science routines that will be very useful for the future. We have prepared and visualized the data during Task I, and then we tried to predict Video Frame Rate values by using a linear model in Task II.

We then managed to understand the influence of the training set size on the Test error (Task II.2).

We implemented Feature Selection (Optimal and Heuristic method) and Dimensionality Reduction methods to understand how we can apprehend complex data more easily (Task III and IV).

Finally, we implemented Neural Networks in Task V, in order to outperform the Linear model and we understood that a fine hyperparameter tuning is very important to have very good performances.

Bibliography

[1] https://commons.wikimedia.org/wiki/File:Linear_least_squares_example2.svg

[2] https://www.v7labs.com/blog/neural-network-architectures-guide

[3] https://medium.com/criteo-engineering/hyper-parameter-optimization-algorithms-2fe447525903