EP2420 *Forecasting Service Metrics*

Joseph Attieh, Vignesh Purushotham Srinivas

December 24, 2021

Project Overview

Forecasting can be defined as the process of making predictions based on past and present data. In this project, we are presented with two datasets, mainly the Key-Value (KV) store based on traces collected from the KTH testbed and the FedCSIS 2020 challenge dataset. We are asked to perform forecasting on the target variables and to explore and compare various forecasting techniques. The rest of the report is structured in four main tasks: Task I and II perform the forecasting using machine learning techniques (Linear Regression and LSTM) while Tasks III and IV perform the forecasting using time series analysis.

Background

Recurrent Neural Network (RNN)

Deep neural networks (DNNs) are powerful machine learning models that can achieve good results on difficult problems such as speech recognition, object recognition, image classification, etc. However, DNNs fail to provide the same performance for sequence-to-sequence problems. Another primary limitation of DNNs is that they are generally applied to problems whose inputs and targets are encoded to fixed dimension-ed vectors. This is a significant limitation because many problems are best represented as sequences whose size is not known a-prior [1].

Recurrent neural networks (RNNs) are feedforward neural networks that elegantly model sequential data by correlating samples in the neighboring position. RNNs can use their internal state (memory) to process inputs of variable lengths, making them suitable for tasks such as handwriting recognition, speech recognition, time series analysis, etc [2]. RNNs are modeled as follows:

For each time step t, the activation $a^{<t>}$ and $y^{<t>}$ are computed as follows [3]:

$$a^{<t>} = g_1\left(W_{aa}a^{<t-1>} + W_{ax}x^{<t>} + b_a\right) \tag{1}$$

$$y^{<t>} = g_2\left(W_{ya}a^{<t>} + b_y\right) \tag{2}$$

where, $W_{ax}, W_{aa}, W_{ya}, b_a, b_y$ are coefficients that are shared temporally and g_1, g_2 activation functions.

Long Short Term Memory (LSTM)

LSTMs are a special kind of Recurrent Neural Networks (RNN). The LSTM contains three gates:

- Forget Gate: this gate is used to determine which information should be forgotten and which information should be retained. This is done by applying a sigmoid function which uses the input and the hidden state. When the forget gate outputs 1, the information is retained, when the output is 0, the information is forgotten.
- Input Gate: this gate is used to determine which values will be updated. A sigmoid function is used to determine which values will be updated and a tanh function is used to create a vector of new candidate values to be added to the state.

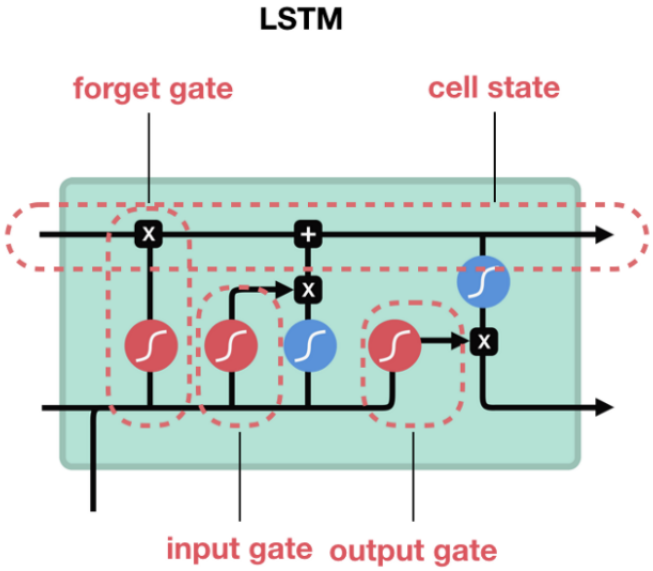


Figure 1: A single LSTM cells. The red blocks represents *sigmoid* activation, and blue blocks represents *tanh* activation.

- Output Gate: this gate Is used to determine the next hidden state based on the output of the previous state and previous hidden state. First the sigmoid function is used to determine which parts of the hidden state will be output. Then this result is multiplied with the tanh function over the cell state.

Time series analysis (trend, seasonality, etc.)

Time series is a sequence of data points collected at successive equally spaced points in time. Examples of time series include service metrics of services collected over a period of time, prices of shares on consecutive days, export totals in successive years, hourly temperature records, etc [4]. Time series analysis is a process of analyzing time series to extract meaningful statistics and characteristics of the data. The primary objectives of such analysis can be classified into the following [4]:

1. **Description**: Time series analysis is employed to understand the general characteristics of the given series. For example, by plotting the time series of the sales, it is possible to infer various seasonal effects and the overall trend. Additionally, such initial analysis can help with identifying outlier events. In general, descriptive time series analysis is applied as the first step to understand the given dataset, decide the pre-processing techniques, derive dominating features that affect the data points, and, to model the data points for further analysis.

2. **Explanation**: Explanatory time series analysis is used to correlate two or more time series.

3. **Prediction**: In predictive time series analysis, the data points of the given time series are modeled using various techniques that allow predicting future values. Predictive time series is actively implemented in sales, economics, and manufacturing domains.

4. **Control**: The control problems with time series analysis are closely related to the prediction problems, where the control decisions are taken based on the predicted value of the time series. Usually, a model is fitted over the time series dataset, future values of series are predicted, and then the input variables are adjusted to optimize the target of the process.

Methods used in Task IV: AR, MA [5, 6]

1. Auto Regressive model(AR): This model is based on the idea that the current value of the serie can be explained as a linar combination of p pas values. Therefore, it performs the forecasting by modelling the next step in the time series as a linear function of the observations at previous time steps. The parameter p is known as the order of the AR term (lag order?), and it specifies the number of lagged observations to consider in the model. The model can be represented by the following formula:

$$\hat{y}_t = c + \phi_1 \times y_{t-1} + \phi_2 \times y_{t-2} + ... + \phi_p \times y_{t-p} + \epsilon_t \tag{3}$$

   In this equation, y_{t-p} is the observation at lag p and ϕ_i are the model parameters. The method is suitable for time series without trend and seasonal components (i.e, where y is stationary). Some models worth mentioning are AR(0) and AR(1). The AR(0) model predicts white noise and models no dependencies. The AR(1) model is as simple as $y_t = \phi_1 \times y_{t-1} + c + \epsilon_t$ and it only considers the previous term in the process.

   The AR model ignore the correlated noise structures, which are usually unobservable, in the time series. In other terms, we can make use of the terms that were not predicted correctly for predicting upcoming observations. Therefore, the MA model was introduced.

2. Moving Average model (MA): This model performs the forecasting by modelling the next step in the time series as a linear function of the residual errors from a mean process at previous time steps. The parameter q is known as the order of the MA term. The model can be represented by the following formula:

$$\hat{y}_t = c + \epsilon_t + \theta_1 \times \epsilon_{t-1} + \theta_2 \times \epsilon_{t-2} + ... + \theta_q \times \epsilon_{t-q} \tag{4}$$

   We should note that the errors ϵ_i are not observable variables (makes the parameter estimations much more difficult than AR); they correspond to the errors from the AR model when fitted on the variable at time i. For instance, ϵ_t is the same variable computed by the following AR model: $\hat{y}_t = c + \phi_1 \times y_{t-1} + \phi_2 \times y_{t-2} + ... + \phi_p \times y_{t-p} + \epsilon_t$.

   It should be noted that the method is suitable for time series without trend and seasonal components, as the MA model is always stationary (observation is a weighted moving average over the past forecast errors). Usually, we can identify whether the time series can be modelled as an MA model by looking at the Auto Correlation Function (ACF), as in general, MA(q) models have nonzero autocorrelations for the first q lags.

3. AutoRegressive Integrated Moving Average (ARIMA): This method models the time series by representing the next step in the sequence as a linear function of the differenced observations and residual errors at previous time steps. ARIMA models are considered as the most general class of models for forecasting a time series. The ARIMA model has the following components:

   - AR (p): Autoregression. A model that uses the dependent relationship between an observation and some number of lagged observations. p is the number of autoregressive terms.
   - I(d): Integrated. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary. Differencing once eliminates a linear trend, while differencing twice eliminates a quadratic trend. d is the number of nonseasonal differences needed for stationarity.
   - MA:(q) Moving Average. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations. q is the number of lagged forecast errors in the prediction equation.

The model can be represented by the following formula:

$$\hat{Y}_t = c + AR(p) + MA(q) \tag{5}$$

$$\hat{Y}_t = c + (\phi_1 \times Y_{t-1} + ... + \phi_p \times Y_{t-p} + \epsilon_t) - (\theta_1 \times \epsilon_{t-1} + ... + \theta_q \times \epsilon_{t-q}) \tag{6}$$

where Y_t corresponds to the differenced version of y_t d times. In other words:

- d=0: $Y_t = y_t$
- d=1: $Y_t = y_t - y_{t-1}$
- d=2: $Y_t = (y_t - y_{t-1}) - (y_{t-1} - y_{t-2}) = y_t - 2 \times y_{t-1} + y_{t-2}$

ARIMA is suitable for time series with trends and without seasonal components.

4. Exponential Smoothing (ES): This method is suitable for forecasting data with no clear trend or seasonal pattern. In its most simplest form, the exponential model taking into account the last p observations is as follows:

$$\hat{y}_t = \alpha \times y_{t-1} + \alpha \times (1-\alpha) \times y_{t-2} + ... + \alpha \times (1-\alpha)^p \times y_{t-p} \tag{7}$$

Another version of this formula can be rewritten in terms of the previous predicted value (however the first predicted value needs to be computed over the past p values for both equations to be equivalent):

$$\hat{y}_t = \alpha y_{t-1} + (1-\alpha) \times \hat{y}_{t-1} \tag{8}$$

How to choose the parameters of the models

There are multiple indicators to choose the parameters of the time series models. After doing some research, we are able to mention the following:

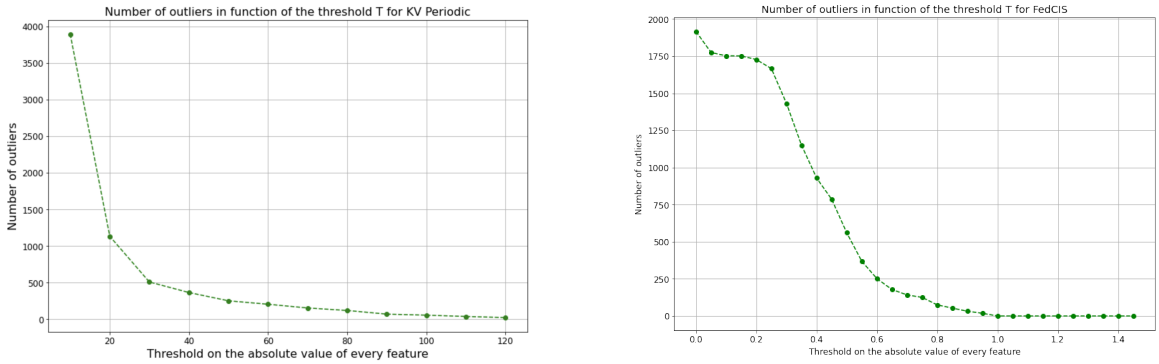
- If a series has a positive Autocorrelation at a high number of lags, we should apply differencing. If the series has an autocorrelation that is small and patternless, then there is no need for differencing. If the autocorrelation of lag 1 is negative, the series might be overdifferenced.
- If the series has a lag-1 autocorrelation that is positive, it apears slightly underdifferenced and exhibits AR signature. If the ACF drops in an exponential trend, a model with p=1 might be a good candidate.
- If the series has a lag-1 autocorrelation that is negative, it apears slightly overdifferenced and exhibits MA signature. This is reflected in a sharp cutoff in the ACF function at a lag equal to q.

Data Description

In this project, we are required to perform forecasting on the following two datasets - "$K-V\_periodic(JNSM)-Part1$" and "$FedCSIS2020challengedataset$". The first dataset is a collection of traces that originate from running the Key-Value store service on the KTH testbed [7]. The FedCSIS trace is a collection of around 2000 samples collected from December 2019 to February 2020 where each sample is an aggregate measurement for over one hour (around 24 samples for each day).

We describe the pre-processing applied to the data as follows:

- **Feature processing**: Firstly, the traces are standardized by column to ensure that values of each column have a mean of 0 and a variance of 1. This is achieved by using the *StandardScaler* module of the *SciKitLearn* package[8]. We chose this pre-processing method since it worked best with the first trace (from project 1).
- **Outlier removal**: We apply the outlier removal algorithm to the pre-processed data traces. The algorithm used in this project is the same one described in the first project. The outlier removal algorithm works by removing samples of the dataset whose absolute value is higher than a given threshold. The Threshold (T) is carefully selected to reject only 1% of the outlier samples from the traces. We compute and plot the number of outliers in the datasets with respect to the given threshold on the values of the features.



(a) Outlier removal in KV trace

(b) Outlier removal in FedCSIS trace

Figure 2: Number of outliers removed as function of Threshold T

Figure 2 shows the number of outliers when varying the threshold T. As the threshold increases the number of samples to eliminate decreases. To eliminate 1% of the samples, $T = 74$ and $T = 3.8$ are chosen respectively for the KV and FedCSIS dataset.

- **Feature selection**: The KV dataset has 1670 features while the FedCSIS dataset only has 9 features. Since we are dealing with a large number of features for the first dataset, we apply a tree-based feature selection technique to reduce the size of the feature space, and hence reducing the training time and resource utilization. We choose to apply a (supervised) tree-based feature selection method to reduce the dimensionality of the feature space. We employ a Random Forest feature selection and select the top 16 features. We use the *ExtraTreeRegressor* [9] from the *SciKitLearn* package and we set the number of estimators to 10.

  Figure 3 shows the correlation among the features. The selected features exhibit at least a 70% relationship to the target variable ($ReadsAvg$). The features themselves possess a high degree of correlation amongst each other.

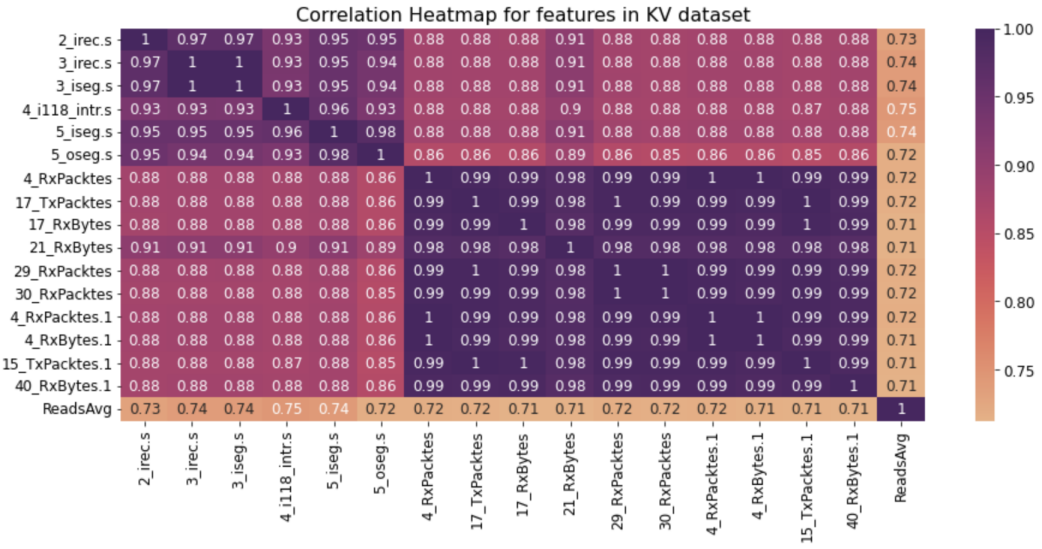


Figure 3: Correlation matrix of the 16 selected features of the KV dataset

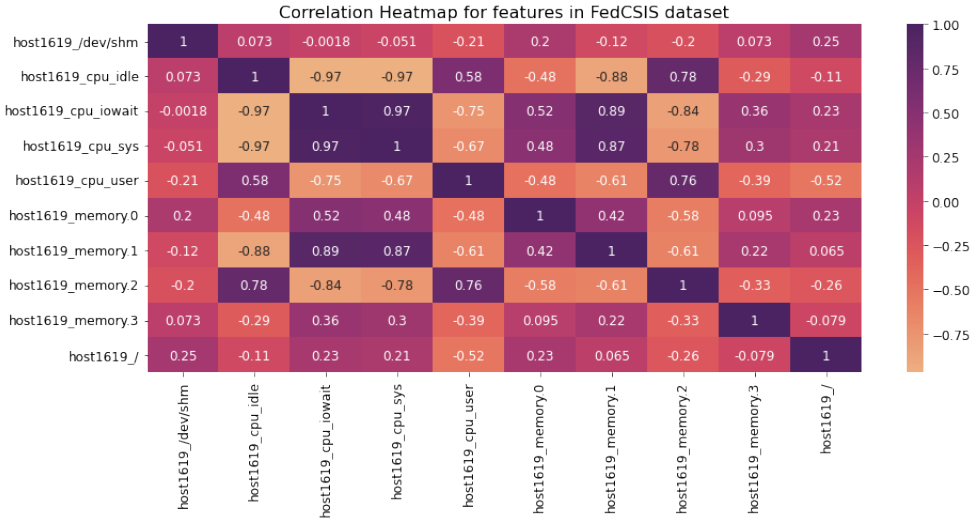
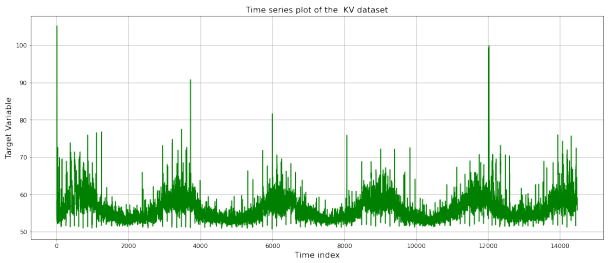


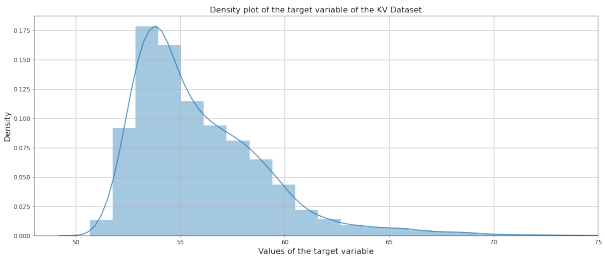
Figure 4: Correlation matrix of the features of the FedCSIS dataset

Figure 4 shows the correlation among features of the FedCSIS dataset. We can notice that the correlation of two features with the output is close to 0, which means that these features are weakly correlated to the output.

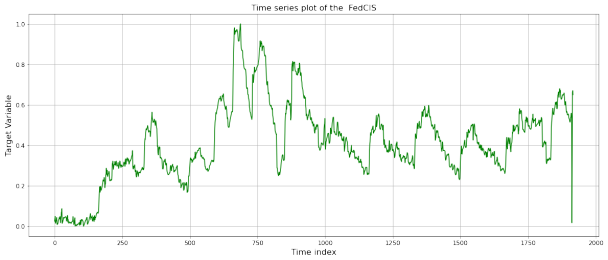
As a last step, we display the time series and density plot of the target variables of both datasets (we will make the font bigger for the next submission and align the plots):



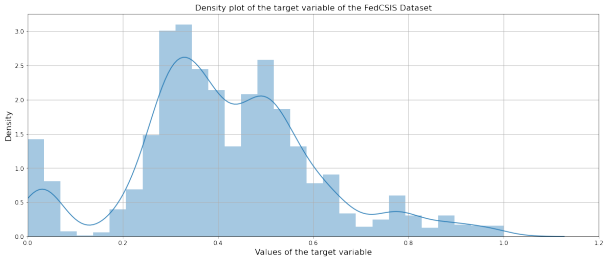
(a) Time series plot of the KV target value



(b) Density of the KV target value



(c) Time series plot of the FedCSIS target value



(d) Density of the FedCSIS target value

Figure 5: Time series and density plots of the target variables

Task 1

Using linear regression for forecasting

In this task, we are required to formulate the forecasting problem as a regression problem. We are required to perform this task only on the KV-periodic dataset. We perform pre-processing, outlier removal then feature selection on the dataset as described in the previous section. Then, we perform a 70/30 split on the dataset to generate a training and testing set. We should mention that we chose not to shuffle the data so that we could use these same datasets for Task II. We transform the transform the time series dataset $\{(x(t), y(t))\}$ to a format that can be used to train a linear regressor: $\{([x_{-l}, ..., x_0], [y_0, ..., y_h])\}$ where l is the lag and h is the time horizon. We conduct the experiments by varying both l and h. Since we are predicting the h target variables independently and due to the time constraints of the experiment, we fix h to 10 and predict the value of the target variable at timestep t as well as all its values after t+h. As for the set of experiments, we train 11 linear regression models by varying lag (0-11) and fixing the time horizon to 10. Basically, l previous values (in the past) will be used to predict h values in the future. For every predicted model, we compute the NMAE of the 11 target variables.

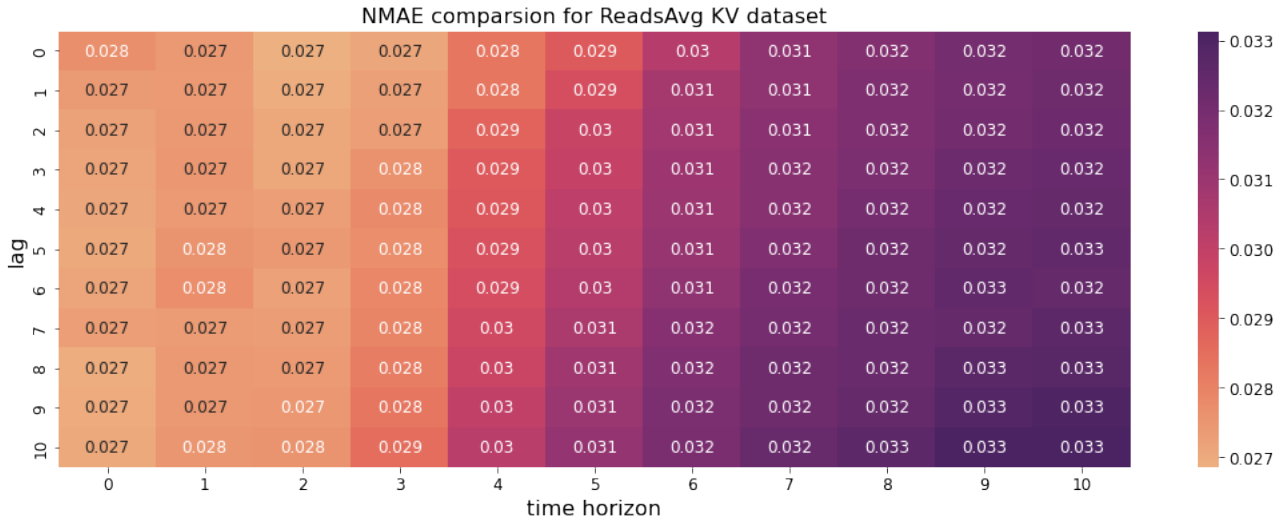


Figure 6: NMAE values of Linear Regression models trained to predict the ReadsAvg metric for a certain value of lag and horizon

Figure 7 displays the different NMAE values for the Linear Regression models trained, where every row represents the time horizon h and every column corresponds to the lag from 0-10. Looking at the table, we can observe the following:

- We can notice that for the same value of time horizon, the NMAEs have approximately the same value regardless of the lag. Since increasing the value of the lag did not result in a decrease in NMAE, we can infer that using more values from the past will not result in better predictions. In other terms, the linear regression model has been able to predict the future target metrics without having to heavily rely on the past samples. This can be explained by the fact that the target sample depends less on input values that are further away in the past and depend more on the input values that are closer (i.e., the target variable depends more strongly on the closest samples, which renders its relationship with input values from the past weaker).
- We can notice that for the same value of lag, the values of the NMAE increase with the increase in the value of the horizon. That would mean that the farther the sample is in the future, the more difficult

it is to predict it. In other words, given the same number of past observations, one can only accurately predict a certain set of values in the future, i.e. the relationship between the samples of the past and future decreases with the increase of the horizon.

We have mentioned that the input data in the far history do not contribute heavily to the forecasting. To validate this claim, we propose to evaluate the effect of previous input samples on the predicted target value. We know that the linear regression model attributes weights to every feature, which corresponds to the strength of relationship/contribution of the feature to the output variable. We inspect the model trained with a lag of 10 to predict a horizon of 10. The more a certain feature of the input variable at a certain lag contributes to the target variable, the more the input variable does. Therefore, we assume that the strength of the relationship between an input variable and a target value is equal to the weight of the feature with the strongest contribution (in terms of absolute value) to the output. We normalized the values per row. The results are displayed below:

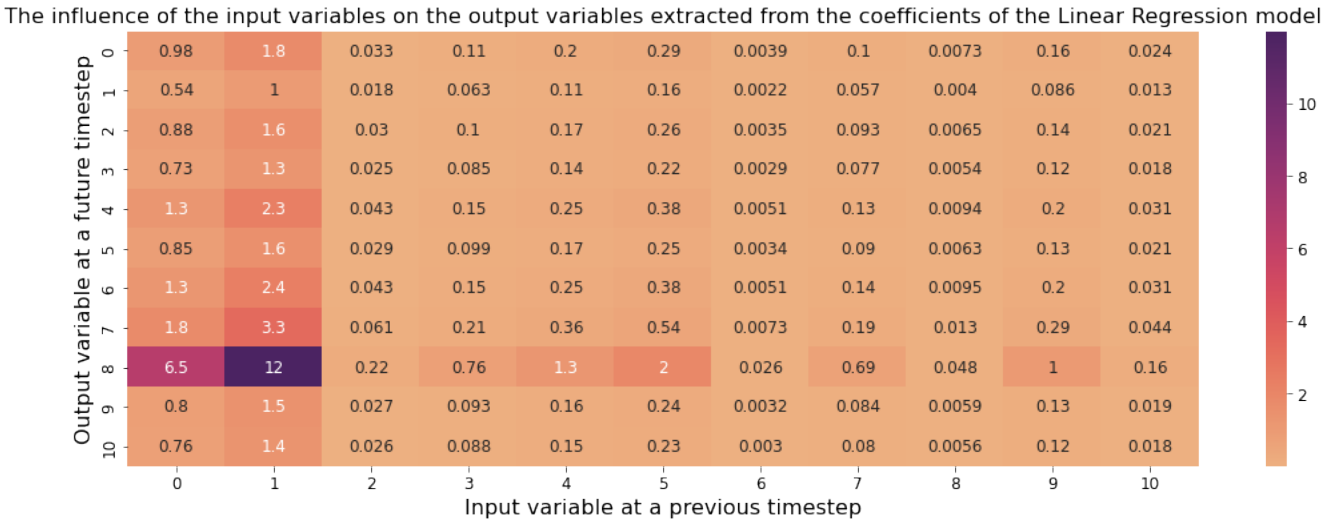


Figure 7: Relationship between input variable and target variables.

As we can see, our claim is valid; the farther the variable is in the past (the right part of the matrix provided), the lower its weight and the less it contributes to the prediction. This validates our claim. We expected such results since the Linear Regression model learns a linear model and does not learn a model that captures long-term relationships.

Task II

This task involves performing forecasting using a specific version of RNN called Long Short-Term Memory (LSTM). Therefore, the same pre-processing and transformations as the first task are applied. We perform a random search to find the best performing value for epochs and the batch size. Then, the depth (number of layers) and the number of neurons in each layer are tuned using the random search technique. We chose to set a DropOut layer of rate 20% after every LSTM layer to prevent over-fitting. This dropout layer drops a neuron activations with a probability of 0.2 at every iteration (i.e., the weights of the dropped neurons are left out of the back-propagation for that iteration). The parameter values that we have found after the hyperparameter tuning on the data with a lag of 10 and a horizon of 10 (the combination that requires the biggest architecture since it has more features) on the specific set of range/search space in the table are showed below:

Hyperparameter	Description	Value	Search Space
Number of epochs	Number of times the learning algorithm loops over the dataset	80	[20, 40, 60, 80, 100]
Size of the batch	Number of samples in the batch for training	20	[10, 20, 30, 40, 50]
Number of layers	Number of LSTM layers in the network	1	[1, 2]
Number of units	Number of LSTM units in the layer	80	[40, 80, 160]

Table 1: Hyperparameters of the LSTM model

We vary the values of lag and horizon as we did in Task I:

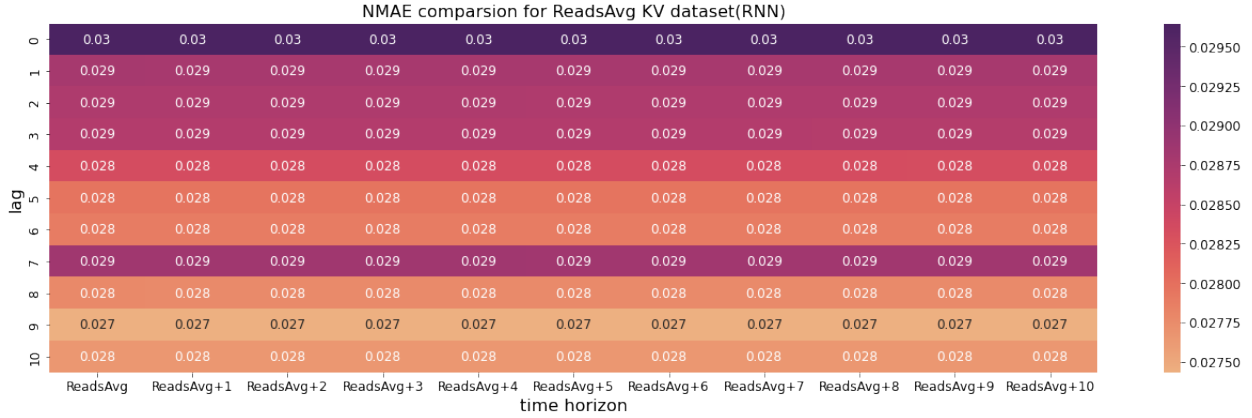


Figure 8: NMAE values of LSTM models trained to predict the ReadsAvg metric for a varying values of lag and horizon

Figure 12 displays the NMAE values for the LSTM models that we trained, where every row represents a certain value of the lag and every column represents a certain value of the horizon. Looking at the table, we can observe the following:

- We can notice that for the same value of horizon, the values of the NMAE decrease when the value of the lag increases. This is due to the fact that the LSTM network can predict future target values more accurately if trained with a larger lag (step). This is quite different from the Linear Regression model, which maintains a similar performance irrespective of the lag value.
- We can notice that for the same value of lag, the NMAEs are similar to each other regardless of the value of the horizon. That means that the model can learn long-term dependencies accurately given the specified number of past inputs. Furthermore, it can maintain the same prediction accuracy for future

target data. This is quite different from the Linear Regression model, in which NMAE was decreasing when increasing the horizon (the linear regression model could not capture long-term dependencies).

- We can notice that all values in the table are very close to each other. That means that using an LSTM model would allow us to use the simplest model, with lag 0 to predict most of the target variables (0,.., 10) without a major loss in the NMAE score.

Optional:

We carry the same analysis by varying both lags and horizons as functions of powers of two (0, 2, 4, 8, ..., 64). We compute the NMAE on the test set. The results can be visualized below:

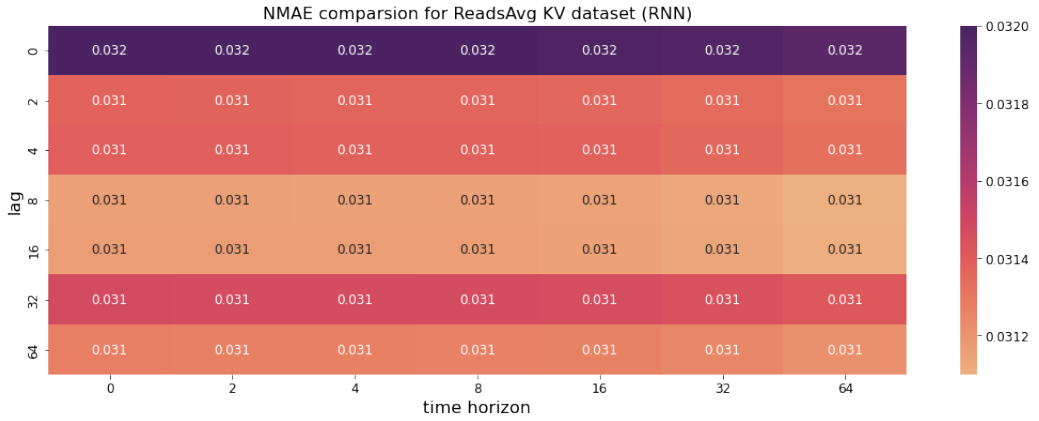


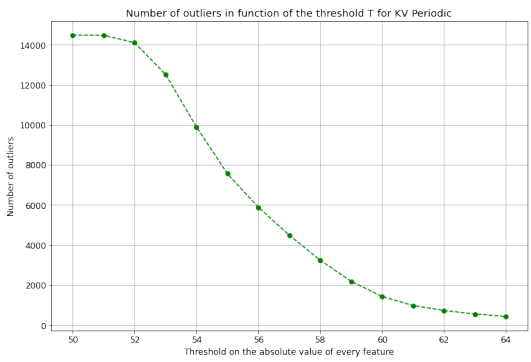
Figure 9: NMAE values of LSTM models trained to predict the ReadsAvg metric for a varying values of lag and horizon as powers of 2 (Optional Task)

We can notice that the NMAE scores have very similar values regardless of the values of lag and horizons specified. In fact, training with a bigger lag did not result in a major improvement in the NMAE score. A possible cause of what we observed is that the LSTM captured the maximum amount of information from the sequences. In other words, having more previous inputs (a bigger lag) might not convey additional information that could help the network perform predictions. This can also be explained by the fact that we increased the number of predicted variables and the number of input variables without adjusting the architecture of the LSTM network. One layer with 80 units might not be enough to capture more accurate dependencies that can be inferred from longer sequences. This can also be attributed to the fact that we trained the network with the same number of epochs for all models. A solution might be to perform hyperparameter tuning again for different values of lag and horizons. The observation can also be explained by the fact that the target variable is periodic, which makes it easier for the LSTM to predict values.

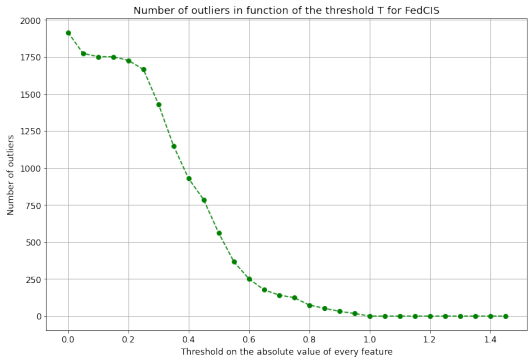
Task III - Time series analysis

In this section, we perform univariate time series analysis on the target variable y. Therefore, we start by performing outlier elimination, then computing the auto-correlation function for every dataset.

1. **Outlier Removal**: We perform outlier removal by considering that the outlier has a feature with an absolute value that is larger than a preset threshold T. We choose the threshold to keep 99% of the data. First, we plot the number of outliers in function of the threshold T for both datasets and we visualize in Fig 13. To keep 99% of the data, we choose T=68 for the KV dataset and T=0.948 for the FedCSIS dataset



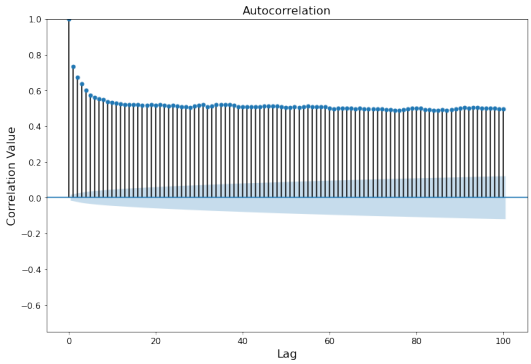
(a) Outlier removal in KV trace



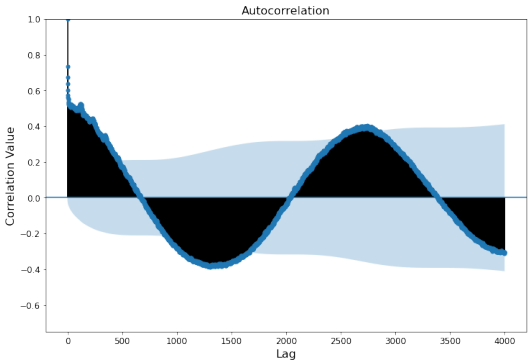
(b) Outlier removal in FedCSIS trace

Figure 10: Number of outliers removed as function of Threshold T

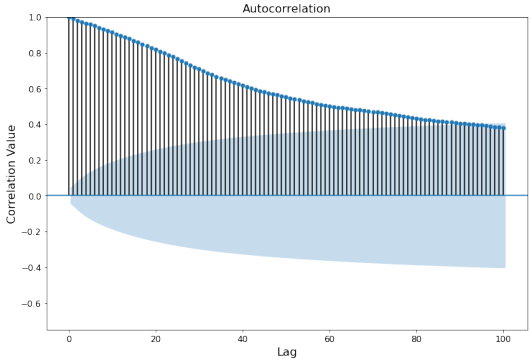
2. **Auto-correlation function (ACF)**: The ACF function computes the correlation of observations in a time series with respect to lag values. We visualize two versions of the ACF plot, the first showing the correlations for l=[0,100] and the other for a bigger lag for each dataset.



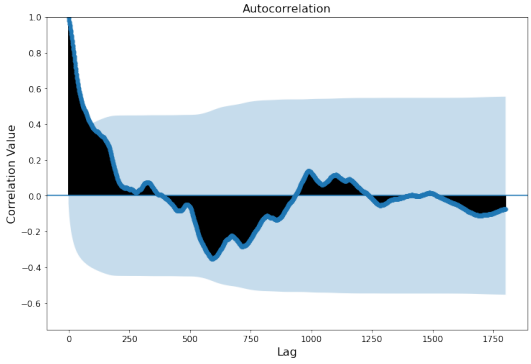
(a) ACF plot of the KV dataset for l =0,..,100



(b) ACF plot of the KV dataset for l =0,..,4000



(c) ACF plot of the FedCSIS dataset for l =0,..,100



(d) ACF plot of the FedCSIS dataset for l =0,..,1800

Figure 11: Autocorrelation functions with different lags

Let's talk more about the ACF plots. This family of plots measures the relationship between a variable's current value and its past values. The horizontal axis of the plot corresponds to the size of the lag between the elements of the time series. For instance, the autocorrelation with lag 3 is the correlation between the time series elements and the corresponding elements that were observed three time periods earlier.
This plot shows whether elements of the time series are positively correlated, negatively correlated, or independent of each other (by computing the Pearson's correlation coefficient $\in [-1, 1]$).

Let's analyze the plots in Fig 11:

(a) KV Dataset:

- Looking at the first ACF plot, we can clearly notice that the autocorrelation value has a peak of 1 at lag equal to 0. This is true as the time series is perfectly correlated to itself. As the lag value increase, the value of the correlation coefficient decreases. We can notice that the first 4 lags have a slightly higher correlation value compared to the subsequent lags of constant correlation (around 0,55). That means that the first 4 past values are mostly correlated to the series and have the biggest predictive power. After a lag of 5, the lagged variables have no major distinct predictive advantage as they all have the same coefficient of correlation. We also notice that the coefficients are positive and outside of the confidence band (i.e., statistically significant observations), which might indicate that a trend is present in the data.
- Looking at the second ACF plot, we can see some type of seasonality that is observed by looking at the cosine-like correlation function. We estimate the period of the time series to be around 2720 time indices.
- To validate our claim, we decompose the series into three components: the trend, the seasonality and the residual. We can visualize these components below.

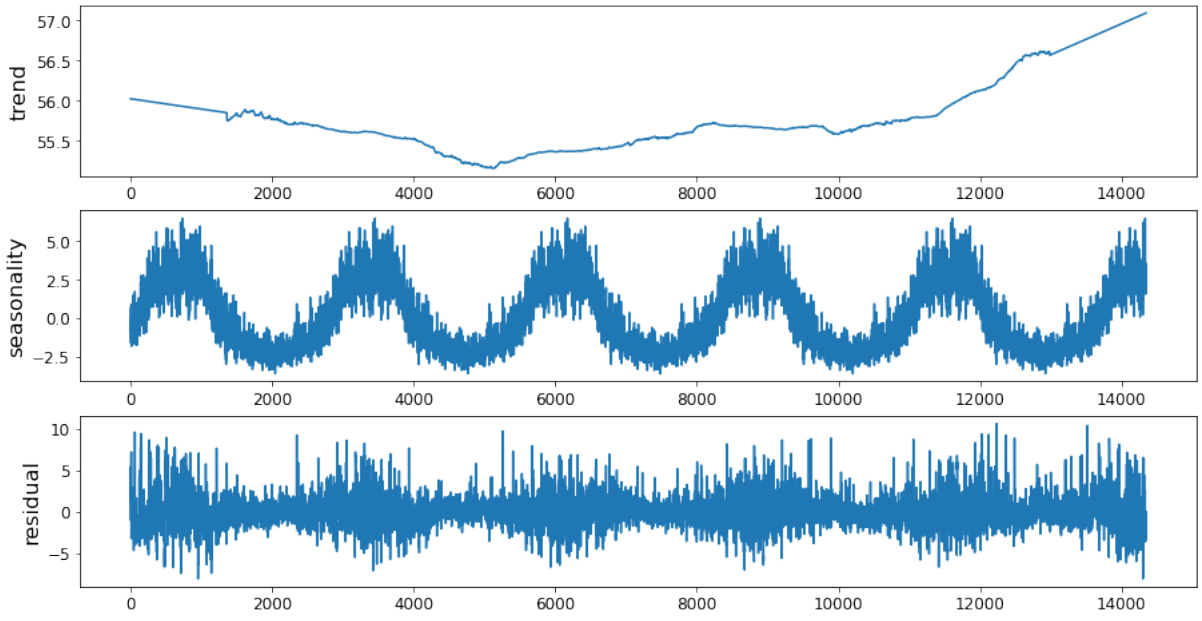


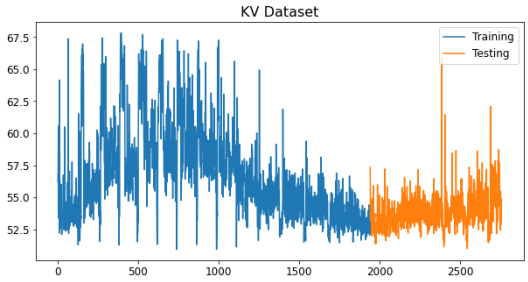
Figure 12: Decomposed time series

(b) FedCSIS Dataset:

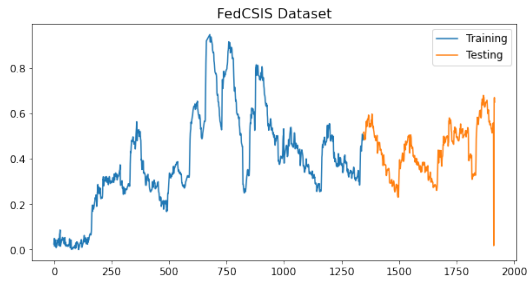
- Looking at the ACF plot, we can see that the autocorrelation decreases with the increase in lag, which means that the farther we go in the past, the less the predictive power of the lagged variable is (since it is less correlated to the current variable at lag zero). We can notice an drop in the values of the correlations that is similar to an exponential drop, which means that the data might be accurately modelled through an AR model with p=1.
- Looking at the second plot, we do not notice a period in the data, which would means that it might have small seasonal components that are not as distinguishable as the ones in the first dataset. Since most of the values of the correlation metric are within the confidence band, we cannot be confident about the significance of the observations here (it is difficult to infer information from the plot).

Task IV - Time series analysis

In this section, we will be trying to fit different models to our datasets. To elimiate the seasonality in the KV-periodic dataset, we truncate our set to the first period, i.e., to the first 2720 time indices. Then, we split the data to training and testing sets by performing a 70/30 split. The datasets are shown below:



(a) Training and testing samples of the KV dataset



(b) Training and testing samples of the FedCSIS dataset

Figure 13: Time series showing the training and testing splits of our datasets

Let's fit the different models mentioned in the project description on each dataset.

1. **KV dataset:**

   (a) Models:

   - **Autoregression (AR) model**: We are required to use an Auto Regression model to forecast the values of the time series, and study the performance of the module after varying the value of p from 1 to 10. We use the AutoReg class of the statsmodels.tsa.ar_model and we fit it to our training dataset to learn the parameters of the model (i.e., the weight of every lagged variable in the prediction). The prediction happens by relying on the previous p values to predict the next value. Since we are interested in evaluating the predictive power of the model, we perform the following: First, we fit the AR(p) model on the training data. The model will learn a constant/intercept value and some weights that correspond to how much a previous sample contributes to the prediction of the next sample (i.e., the next sample predicted by the model is going to be the weighted sum of its previous p samples). Since we are interested in the predictive range of the model, we would like to predict the next h values given this fitted model. Therefore, whenever we are predicting h values at a time, the predictions generated are assigned as the current value of the sample and are used to predict the value of the samples that are consecutive to them, given that they occur within a lag of p away from these samples. In other words, whenever the model performs consecutive predictions at a time, the model

uses the values that it predicted to generate the next predictions (as if we are filling the time series with the predicted values). Performing this experiment once is not useful as we are required to evaluate the goodness of the model on the whole time series.After generating the first 10 values, we add one value from the test set to the training data and we perform a new prediction (i.e., we predict the next 10 values after the values that we added from the test set). The whole process is performed a number of times that is equal to the number of values in the testing set. We will end up with an array of predictions with the size $size\_test \times h$, where every column corresponds to a predicted value at a certain horizon (a lagged version of our test set). To evaluate the model, all we have to do is to compute the NMAE of every column with a certain lagged version of the test set.
We should note that we have experimented with two different methods to generate the predictions. The first method relies on the "predict" function provided by the model. Performing a new prediction using this function requires to refit the model on the set containing the training data and the data taken from the testing set. However, this is not very realistic, as the parameters of the models are being updated every time we add a value from the testing set. The second method is the one that was suggested in class. We first extract the parameters learned by fitting the model on the training set. Afterwards, we write the formulas that are used to perform the prediction of h values given the previous p values. Then, we call this function a number of times that is equal to the size of the test set while at the same time taking values from the test set and adding them to the training set (as if we are shifting and taking p values at a time from the whole time series). This is a better approach since the model's parameters are fixed. It should be mentioned that the results reported in this report are the ones generated from the second approach (we also tested whether our functions worked by comparing its result to the first time we call the predict function). We report only 10 NMAE per model(per set of parameters). The results are displayed in the table below:

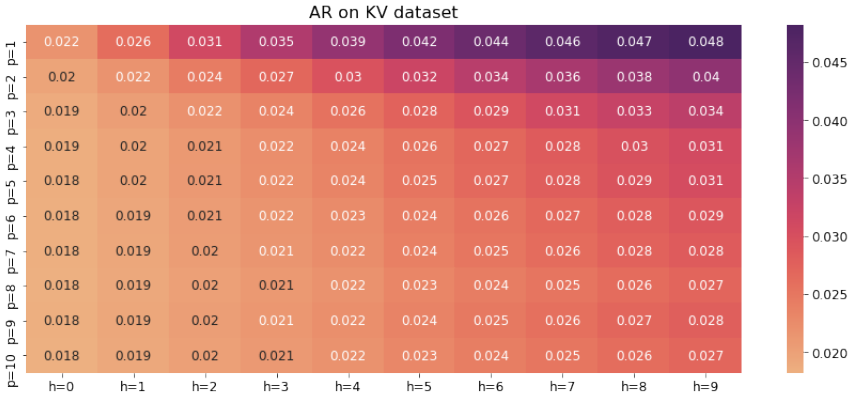


Figure 14: Results of using the AR(p) model to predict the next h values

- **Moving Average (MA) model**: We fit a Moving Average model with a value of q varying from 1 to 10. To do so, we fit an ARMA model with the parameters p and d equal to zero. Then, we extract the parameters of the models and we perform the predictions using our own function as described before. What is quite different about this model is that the new sample depends on a weighted sum of the residuals. For every sample that was not fitted by the model (i.e., a sample that was predicted), the residual of this sample is 0. Therefore, the model will end up predicting a constant value in the case in which the value at a horizon h is bigger than the value of q (i.e, if q=1, then the values after h=2 will be the same since predicting using unfitted values yield a residual of 0). We assumed that the residual is 0 as described in the example in class. The results are showed in Fig 15.

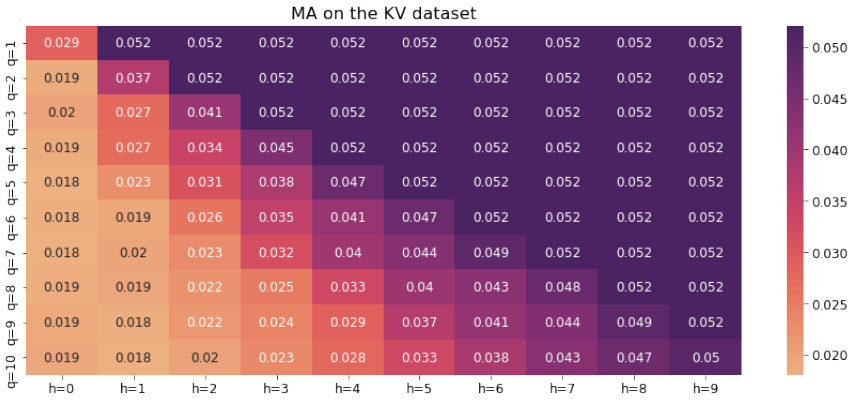
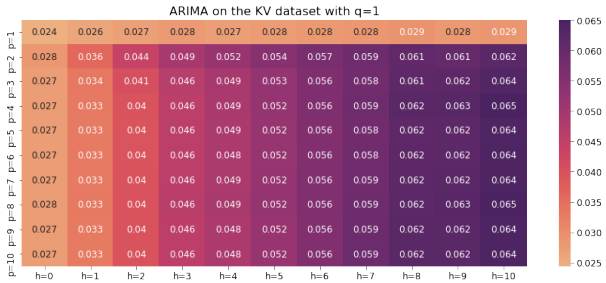
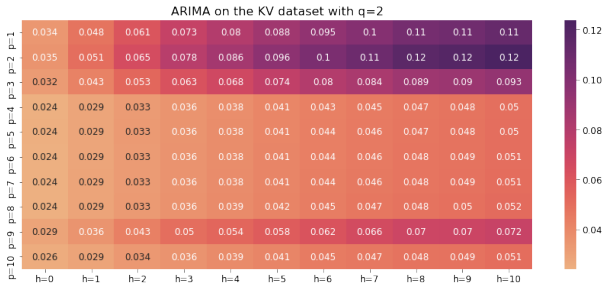


Figure 15: Results of using the MA(q) model to predict the next h values

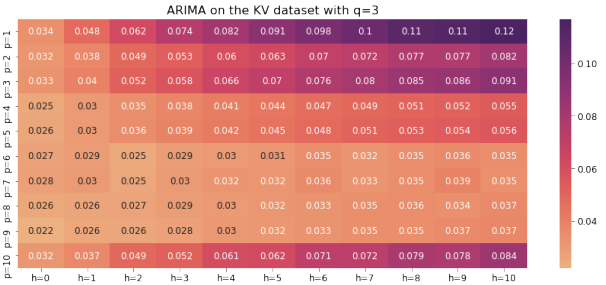
- **Autoregressive Integrated Moving Average (ARIMA) model:** We fit an ARIMA model with the following set of parameters: $d = 1$, $p = 1, ..., 10$ and $q = 1, ..., 5$. As before, we extract the parameters of the models and we perform the prediction using our own function as described before. We report 10 NMAE values per set of parameters showed in Fig 18.



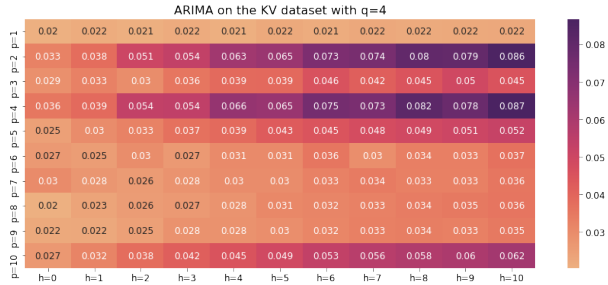
(a) Results for the ARIMA(p,1,1) model



(b) Results for the ARIMA(p, 1, 2) model



(c) Results for the ARIMA(p,1,3) model



(d) Results for the ARIMA(p, 1, 4) model

Figure 16: Results of the ARIMA model on the KV datset

- **Exponential smoothing method:** We fit an exponential smoothing model with a lag value that varies between 0 and 10 and $\alpha = 0.5$. We implemented the two versions of the model that were described in the background. We present the results generated by the first version that depends on the previous p lagged values. As before, we only report 10 NMAE values per model parameters in Fig 18.

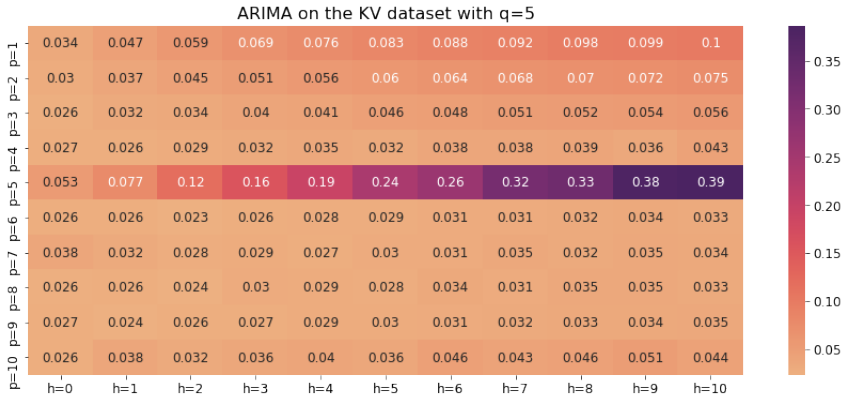


Figure 17: Results for the ARIMA(p,1,5) model on the KV dataset

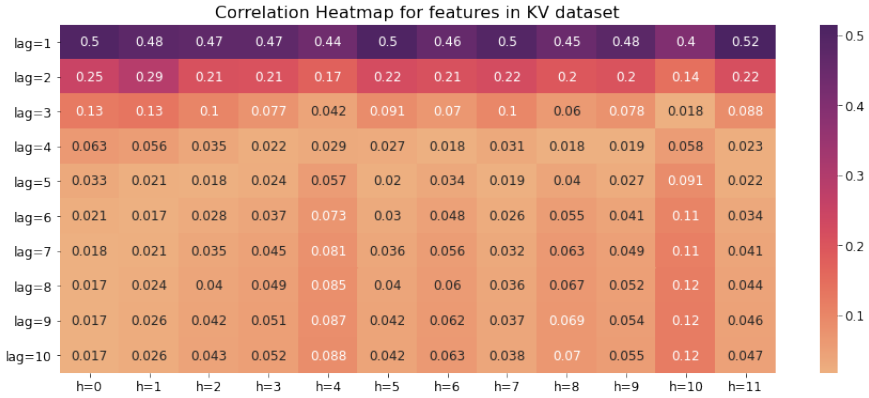


Figure 18: Results of using the exponential model to predict the next h values

(b) **Analysis**: From the results presented, we can observe the following:

- AR model: Let's analyze the results presented in Fig 14.
  - We can notice that for the same value of horizon, the values of the NMAE decrease when the value of the lag p increases. That means that the model performs better (i.e., forecasts more accurately) when trained on more variables from the past. This makes sense since the values up to lag 10 have a very good correlation coefficient with the current target variable (¿0.5). In fact, the stronger the correlation between the output variable and a specific lagged variable, the more weight that autoregression model can put on that variable when modeling and the more accurate the prediction is.
  - We can notice that for the same value of lag p, the NMAE value is increasing when the value of the horizon increases. That means that it is more difficult for the model to predict values that are further away in the future. In the context of the AR model, that would mean that the errors incurred in a previous prediction are being carried to the next prediction (this is a direct consequence of having predictions that are generated from other predictions, as well as a linear model). This is due to the sequential nature of the time series model.
  - We can also notice a very particular behavior: for values of p that are greater than 5, we can notice that the NMAEs are almost the same. This can be explained as follow: even though the model has more lagged variables (i.e., more information about the variable

from the past), these additional lagged values have little influence on the prediction (no major information gain). This is clear as the ACF function shows no major correlation between the target and the lagged samples occurring 5 samples away from the current sample.

We should also mention a nice analogy between the AR model and the Linear Regression model. Both models rely on previous values in some way (one relies on the previous input while the other relies on the previous target).

We can say that the AR model has an acceptable performance on our data. This is mainly attributed to the fact that AR models work best with time series without major trend and seasonal components (we removed seasonality by working over one period).

- MA model: Le's analyze the results in Fig 15.
  - We can see that for the same value of q, the NMAE increase with the increase in the horizon. What is quite interesting in this behavior is that the NMAE keeps on increasing until it reaches a certain horizon of "q" where it takes on a constant value. This behavior can be explained by the nature of the model's prediction function: in fact, when a model is being fit on a training set, it computes the residuals of the data in this training set. This is problematic in the case in which we want to use this model with new data that was not trained on. We took the same assumption that we have seen in the example in class and we set the residual to 0 for the data that is not in the training set. Therefore, a model of order q has q lagged residual variables. For the model to output a constant value, all lagged variables should be 0, i.e., the predicted value should be inferred from data that is completely outside the training set. This is only possible for samples where h¿q, since all the previous q residuals are not in the training set and will be 0.
  - We can notice that for the same value of horizon, the NMAE decreases when q increases. This is quite expected as the model will be able to learn better relationships from the lagged residuals.
  - We can notice that the MA algorithm is not as good as the AR model, as even with maximum value of q, the NMAEs were high for h¿5. We expect these results as we see no major drop in the correlogram within a lag of 10. To decrease the NMAE, one should increase the value of q (so that less samples would satisfy the condition h¿q).
- ARIMA model: This ARIMA system has an integrator component that performs differencing of the raw observations. In other terms, this component is making the data more stationary and is removing some of the trend and seasonality. This will make the data more suitable for both Autoregression and Moving average components of this model. We can visualize the results in Fig 16 and Fig 17.
  - We can note that all models have the integrator with d=1. That means that the time series has been differenced once, which in our case eliminates some trend and makes it more statitionary.
  - Let's visualize the NMAEs for q=1 (one residual error). We can see that the model ARIMA(1,1,1) has a very low NMAE for p=1 for all values of the horizon. That shows that the model was successfully able to predict the values. This can be explained by the fact that the differencing increased the correlation between the target variable and its previous variable, which made the prediction much easier based on one residual component (i.e., the error) and the previous variable. We can notice that for other values of p, the NMAE increases with the horizon. At the same time, the NMAE is constant over the same horizon regardless of the value of p. That would mean that the correlation between target and p-lagged values of p¿2 is not informative, as it mae the NMAE worst (p=2) and added no information gain to the model knowledge (i.e., when p¿2, the NMAEs are constant).
  - Let's visualize the NMAEs for q=2, 3 and 4. We can see that the NMAE is constant for certain values of p: for q=2 , it is constant on p=4,5,6,7,8 and 10; for q=3, it is constant

on p=6, 7, 8 and 9; for q=4, it is constant for p=6,7,8,9. We cannot observe any major reason of this behavior. This behavior is totally dependent on the lagged variables (the value of p) and lagged residuals (the value of q). We can infer that changing the values of p and q changes the properties of the differenced time series, which will affect the model very differently.

  - Let's visualize the NMAEs for q=5. We can see that the NMAEs are very small for all ps except for p=5. This might be explained by the fact that modelling the differenced time series with 5 lagged residual components with one lagged variable is enough to get a good NMAE (we can see the very bad correlation between the 5th lagged variable and the predicted outcome).

  As we can see, the ARIMA model performs well for certain combinations of p and q and very bad on others. However, we can see that using d=1 and p=1 always have the lowest NMAE as it seems that differencing the time series enhanced the correlation between the target variable and its previous variable. With these parameters selected, the value of q does not have a major effect on the NMAE. Degradation in performance can be explained by the fact that differencing the data by its lagged version is removing some of the correlation between the new time series and its lagged versions. In other words, the ACF graph is different, and the coefficients at $lags > 1$ are now smaller.

- Exponential smoothing: This method is usually used for data with no clear trend or seasonal pattern as it computes the forecast as weighted averages that depend on a smoothing parameter α. Setting α to 0 would mean that the forecasts will be equal to the average of the observations while setting it to 1 means that forecasts are equal to the last observation. Therefore, a value of 0.5 is a good compromise. We can see that for a specific value of h, the NMAE decreases when the lag increases; this is normal as more values from the past would induce a more accurate target value. Furthermore, for a specific value of lag, the NMAE value is almost the same regardless of the value of the horizon. That would mean that the model is affected by its own predictions, as any error in the prediction propagates due to the sequential nature of this model. This is our baseline. The algorithm might have underperformed since our series has no clear season (because we removed it).

2. **FedCSIS dataset:** We perform the same experiments as before for this dataset. We report the following results:

   (a) Models:

   - AR model:

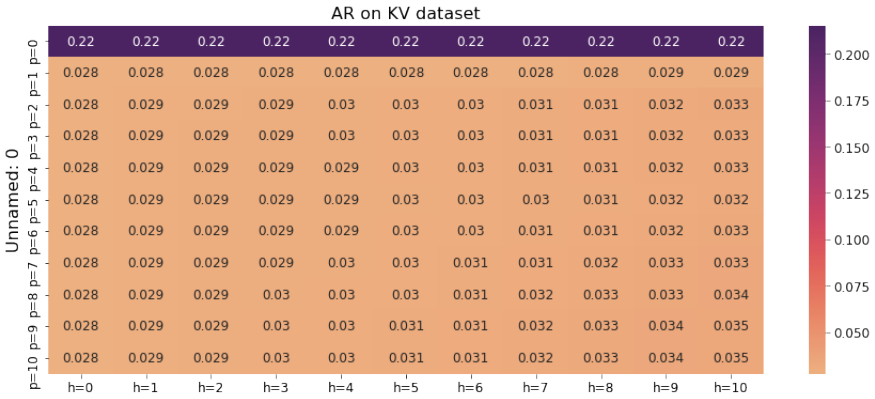


Figure 19: Results of using the AR(p) model on the FedCSIS dataset

- MA model:

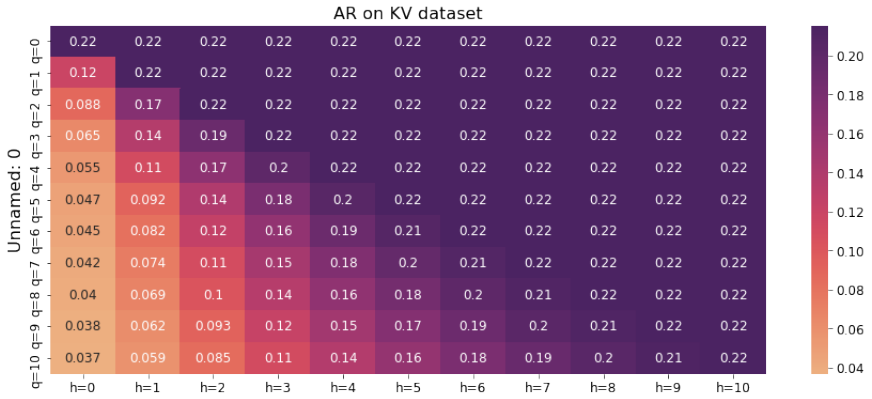
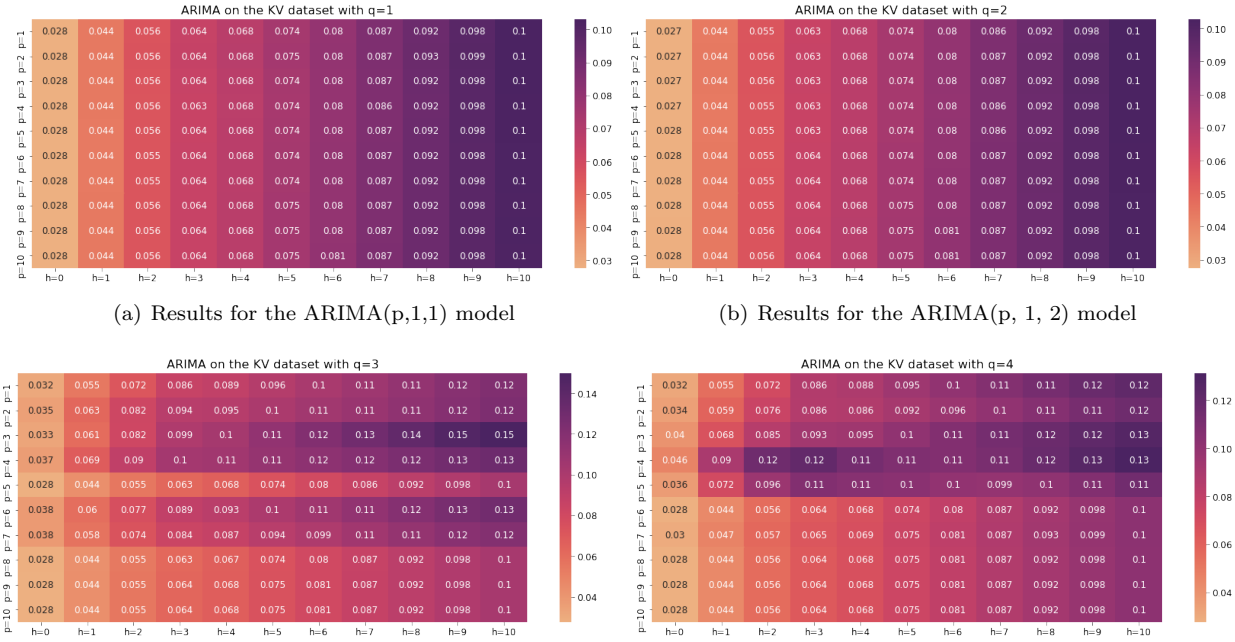


Figure 20: Results of using the MA(q) model on the FedCSIS dataset

- ARIMA model:



(a) Results for the ARIMA(p,1,1) model

(b) Results for the ARIMA(p, 1, 2) model

(c) Results for the ARIMA(p,1,3) model

(d) Results for the ARIMA(p, 1, 4) model

Figure 21: Results of the ARIMA model on the FedCSIS datset

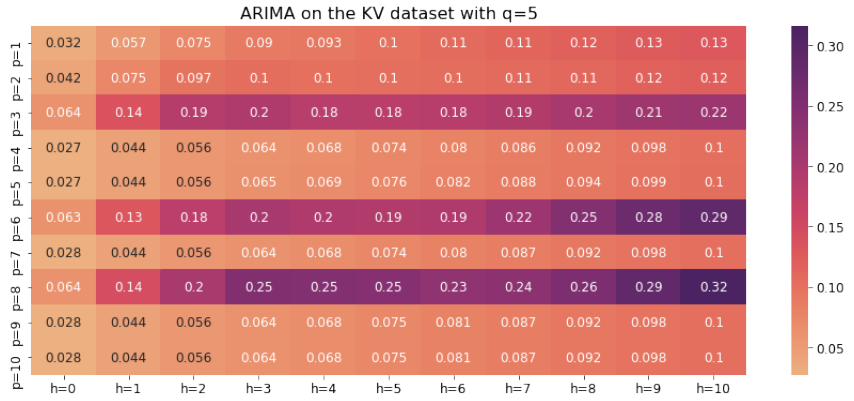


Figure 22: Results for the ARIMA(p,1,5) model for the FedCSIS dataset

- Exponential model:

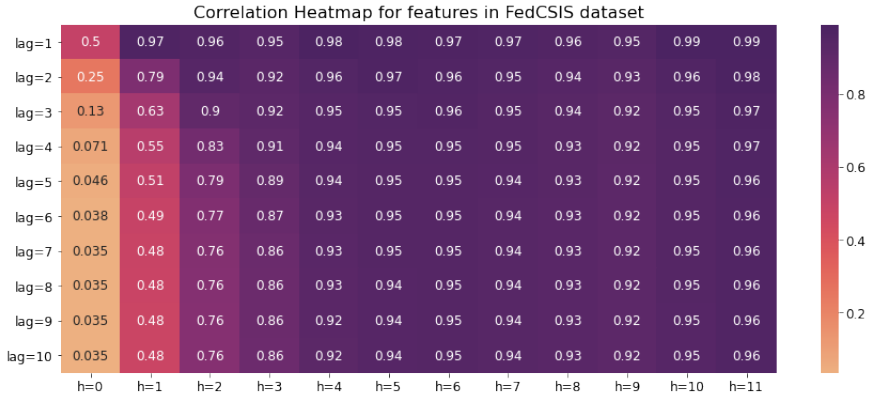


Figure 23: Results of the exponential model on the FedCSIS dataset

(b) Analysis:

Let's analyze the methods applied on this time series:

- AR model: We can see that the NMAE is quite low for all values of p. This can be explained by the fact that the lagged variables are highly correlated with the target variable.
- MA model: We can see that the NMAE behaves similarily to the MA model on the previous dataset. This can be explained by the same reasoning (residuals converging to zero).
- ARIMA model: For this model, we can observe that the model performs best for p=1 with no major improvements regardless of the value of q.
- Exponential model: We can see that the model performs best with a high value of lag and for predicting the value of the first future sample.

We can clearly see that all of the methods perform consistently with the ACF of this time series. This time serie is more challenging than the last one due to the multiple oscillations in the test set (seasonal components) and a small trend. Most of the methods that we have tried are not suitable with this type of series. We can notice that the ARIMA model performs the best. This is due to the integrator

component that removes all traces of trend and seasonality during fitting. The second-best model is the AR model, which makes sense since the correlations in the ACF are very high.

References

[1] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *Proceedings of the 27th International Conference on Neural Information Processing Systems - Volume 2*, ser. NIPS'14. Cambridge, MA, USA: MIT Press, 2014, p. 3104–3112.

[2] Recurrent neural networks(RNNs). [Online]. Available: https://en.wikipedia.org/wiki/Recurrent_neural_network

[3] Recurrent neural networks cheatsheet. [Online]. Available: https://stanford.edu/~shervine/teaching/cs-230/cheatsheet-recurrent-neural-networks

[4] Chris Chatfield and Haipeng Xing, *The Analysis of Time Series: An Introduction with R*. Chapman and Hall/CRC.

[5] Time series: Autoregressive models ar, ma, arma, arima. [Online]. Available: https://people.cs.pitt.edu/~milos/courses/cs3750/lectures/class16.pdf

[6] Lecture notes on forecasting. [Online]. Available: https://people.duke.edu/~rnau/Slides_on_ARIMA_models--Robert_Nau.pdf

[7] 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.

[8] Standardscalar. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

[9] Extratreeregressor. [Online]. Available: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.ExtraTreesRegressor.html