A Work Project, presented as part of the requirements for the Award of a Master's degree in Business Analytics from the Nova School of Business and Economics.

HYPERPARAMETER FINE TUNING FOR A TIME SERIES FORECASTING MODEL

MANUEL MARIA DA CUNHA MAGALHÃES

Work project carried out under the supervision of:

Patricia Xufre

17-12-2021

Abstract

This project was conducted in the context of the Project-Based Learning program. The purpose of the program is to provide an experience in a real-life business and data analytics project. During the last 18 months a work collaboration have been carried out between four NOVA SBE Business Analytics master students and Brisa. The main objective of the project was to produce new traffic forecasting models in Python. The individual work carried out by the author of this study, was focused on the hyperparameter fine tuning procedure for the forecasting models. The research for different methodologies resulted in the experimentation of grid search and random search frameworks. As expected, grid search achieved better results but it is a process that requires more computational power and time.

Keywords: Business Analytics, Business and Data Analytics, Hyperparameter Fine Tuning, Grid Search, Random Search

This work used infrastructure and resources funded by Fundação para a Ciência e a Tecnologia (UID/ECO/00124/2013, UID/ECO/00124/2019 and Social Sciences DataLab, Project 22209), POR Lisboa (LISBOA-01-0145-FEDER-007722 and Social Sciences DataLab, Project 22209) and POR Norte (Social Sciences DataLab, Project 22209).

Table of Contents

List of Figures

List of Tables

1. Introduction

The proposed study covers the last 18 months of a project carried out between four NOVA SBE Business Analytics master students and Brisa, a Portuguese private transport infrastructure company. The objective is to identify the different aspects of the project and present academic research related to it. In this specific case, the study was focused on two parts: First, the development of new traffic forecast models, and second, the development of a hyperparameter fine tuning framework for the newly developed traffic forecast models of Brisa. Accordingly, the scope of the research has two main points. On one hand, we will look at the history and development of the time-series forecasting models. On the other, the current state of Hyperparameter Fine Tuning methods, as well as the one specifically developed for the end-product of this project.

Time-series forecasting plays a major role in the today's business world. Most businesses are dependent on forecasts. In the context of Brisa, forecasting the traffic volume is key, as their budget planning process is highly dependent on it.

In the development of such forecasting models, there are hyperparameters that must be set. Hyperparameters are points of choice or configuration that allow a machine learning model to be customized for a specific task or dataset. Typically, a hyperparameter has a known effect on a model in the general sense, but it is not clear how to best set a hyperparameter for a given dataset. Further, many machine learning models have a range of hyperparameters, and they may interact in nonlinear ways. As such, it is often required to search for a set of hyperparameters that result in the best performance of a model on a dataset. This is called hyperparameter optimization, hyperparameter tuning, or hyperparameter search.

While time-series forecasting has been widely researched so far, with pieces such as Holt (1957), Box & Jenkins (1970) and more recently Taylor & Letham (2018), it doesn't exist such

works on the research for long-term traffic forecasting. The topic around hyperparameter optimization dates back to the 1990s, such as with King *et al.* (1995), Michie *et al.* (1994) and Ripley (1992), and it was also established early that different hyperparameter configurations tend to work best for different datasets, Kohavi & John (1995). Nowadays, it is also widely acknowledged that tuned hyperparameters improve over the default setting provided by common machine learning libraries Mantovani *et al.* (2016), Olson *et al.* (2018), Sanders & Giraud-Carrier (2017), Thornton *et al.* (2013). Because of the increased usage of machine learning in companies, hyperparameter optimization is also of substantial commercial interest and plays an ever-larger role there, be it in company internal tools, Golovin *et al.* (2017), either as part of machine learning cloud services, Li & Li (2018)

The final purpose of the study was to develop three new traffic forecasting models for each of the 11 Brisa motorways, one per each vehicle class (light and heavy classes) and one for both, and three forecasting models with the same distinction for total traffic prediction. In the building of these models, a framework had to be developed for the choice of the hyperparameters of each of the models. Therefore, the thesis has the following structure. In chapter 2, an overview of the project context will be given with the introduction of Brisa, its current challenges, and the goals of the project. Chapter 3 will provide the overlook of the current research for both time-series and traffic forecasting and hyperparameter fine-tuning. Chapter 4 will reveal the practical work produced in the project, with the modelling results, the development process and the hyperparameter framework developed for the project. To conclude, in Chapter 5 conclusions will be taken in accordance with the results and development of the project.

2. Context: The company and the challenge of traffic forecasting

2.1. The company

Brisa Auto-Estradas de Portugal, a private transport infrastructure company founded in 1972 and commonly referred to as Brisa, is the largest private operator in the motorway sector in Portugal and one of the largest in Europe. It's motorway network consists of over 1,600 km of road across the Portuguese territory. The company played a major role in the development of Portugal's transport infrastructure. Nowadays, Brisa offers a variety of transportation services, including motorway concessions through Brisa Concessão Rodoviária (BCR), a subsidiary, electronic payment services for tools with ViaVerde, along with new mobility technologies (i.e., e-mobility).

In the present day, Brisa's group has almost 3,000 employees and generated an operating income of EUR 781m in 2019. In total, it holds concessions for 17 motoways, of which 11 are managed by BCR. The subsidiary was responsible for around 90% of the operating income of the group in 2019. (Brisa, 2021)

In accordance with the collaboration between Nova SBE's project-based learning (PBL) and Brisa, the project focused on the 11 motorways under BCR's management.

2.2. Current procedure for traffic forecasting

Having the ability to forecast traffic for all motorways is of high importance for Brisa, as most of their activity revolves around traffic itself. Such traffic forecasts are used for two main purposes, being them:

I. **Decisions for budget allocation**: Allocation of budget into CAPEX and OPEX is affected by the forecasts, as issues regarding the maintenance of the motorways might arise and change the said allocations.

II. **Investment planning**: The company is obliged to widen its motorways with additional lanes, according to the concession contract, if traffic growth is above a defined percentage. Hence, the traffic forecasts are used in order to plan such investments.

Brisa already has a traffic forecast model developed by its own *Traffic Studies and Advanced Analytics* department. The model developed follows a multi-linear regression and uses a few exogenous variables, namely GDP, the Car Ownership Rate, the Fuel Prices and the Industrial Production Index. The traffic forecasting model developed achieves an average deviation between one and two percentage points of the real traffic growth rates.

The current process is as follows: The model is set up in a Microsoft Excel file. Traffic historical data is pulled from a transactional database and aggregated. The exogenous variables data is imported from diverse CSV or Excel files. Having this, different scenarios for the traffic growth for the future eight quarters are modelled and forecasted. The selection of the variables and adjustments performed in the model process are led by the Head of the department. Summing up, the current approach is a multi-linear regression model that includes some independent factors paired with adjustments based on knowledge and experience.

2.3. The project

At the start of the project, the goals were defined in collaboration between Brisa and the PBL team. The main goal was to develop an alternative for the current procedure for traffic growth forecasting. The idea was to create a new regression model in Python for the motorways to be used a few times per year. During the project, there was also the intention of additional exogenous variables in order to identify more variables that might impact the performance of the traffic forecasting models. To conclude, it was made the decision to create a new interactive dashboard for the Head of Brisa's *Traffic Studies and Advanced Analytics* department, Luis

Felipe Marreiro, where he could interact with the forecast settings of the newly developed models. This last topic was developed in another thesis by the project team member Lennart Oser.

Besides having the main goals of the project defined, some “nice-to-have” outcomes of the project were outlined as well. These deliverables were based on a different dataset of the one used for the creation of the new traffic forecasting models. One of the goals was to try to identify fraudulent trips. The second was to identify traffic pattern changes due to the COVID-19 pandemic. The topic of identification of the pattern changes after the pandemic was covered by a project team member, Felix Pagel.

3. Literature Review

3.1. Time series, Forecasting and traffic

Most forecasting problems come from the usage of time-series data, according to Montgomery *et al.* (2005). This sentence makes sense having in mind that forecast is to predict the future and therefore it is time-oriented. Time-series data is defined by Hayes (2021) as a sequence of data points in successive order over a specific period of time. Having the ability to forecast is of crucial importance to the today's decision-making process in the business world, and although forecast is generally not perfect due to the uncertainty of future events, forecasting exercises are developed in order to reduce such errors

For the creation of this project, the focus was given to three specific forecasting methods: exponential smoothing, (S)ARIMA and Facebook Prophet. Exponential smoothing methods were initially described in works such as of Holt (1957), Brown (1959, 1963), Winters (1960) and Pegels (1969), but with no foundation on statistical concepts as later shown by De Gooijer & Hyndman (2006). Another important development for time-series forecasting was brought by Box & Jenkins (1970), with the popularisation of the autoregressive integrated moving average (ARIMA) model. The name of the model covers the three components of the model itself. First the Auto Regression (**AR**): The model utilises the dependent connection between an observation and a specific number of lagged observations. Secondly the Integration (**I**): It differences the raw observations to achieve a stationary time-series. Finally, the Moving Average (**MA**): The model uses the dependency between an observation and the residual errors from a moving average model applied to lag observations. Each one of the components is denoted as a hyperparameter of the model (p, d, q). A classical ARIMA model is restricted to the usage of non-seasonal data, but in reality, some time-series have a seasonal pattern attached to it. In order to overcome this obstacle, there exists the possibility of adding the four seasonal

hyperparameters (P, D, Q)m to the model (Hyndman & Athanasopoulos, 2021, sect. 9.9, para. One), creating a Seasonal ARIMA (SARIMA) model. The before mentioned hyperparameters are then set to non-negative integer values that specify what kind of model is used. The hyperparameters are as follows:

— **p** and seasonal **P**: Indicate the number of the autoregressive terms;

— **d** and seasonal **D**: Indicate the number of differences needed to achieve stationarity;

— **q** and Seasonal **Q**: Indicate the number of lagged forecasting errors to include;

— **m**: Indicates the number of time steps of a single seasonal period.

ARIMA models tend to perform better for long period forecast when compared to exponential smoothing models. More recently, Facebook decided to develop a time-series forecasting model named Prophet, with its increasing popularity being registered by Taylor & Letham (2018). According to Taylor & Letham themselves, Prophet "is flexible enough for a wide range of business time series, yet configurable by non-experts who may have domain knowledge about the data-generating process but little knowledge about time series models and methods" (p. 37). Performs as a decomposable time series model with the model components trends, seasonality and holidays. The advantages of using Prophet, as per the authors, is the higher flexibility when defining seasonality and trends, the usage of non-regular spaced time-series, the speed of fitting and the facilitated interpretation of the parameters.

Recent works compared the usage of SARIMA and Prophet, but such works are few. Most found that generally Prophet underperforms SARIMA models, as Shah (2019), Papastefanopoulos *et al*. (2020), Kumar & Susan (2020). Sometime, Prophet has an advantage over SARIMA models as shown in Almazrouee *et al*. (2020). Summing up, Prophet is a model simple to implement and a good choice if there is the need to predict a vast amount of time-series data but tends to underperform SARIMA models.

Now looking into the forecasting topic in general, over the last years various research on traffic forecast has been made. According to Mahdavian *et al.* (2021, p. 484), traffic forecasting problems can be typically split into three different types according to the time horizon: short-term (5-30min); mid-term (30min to a few hours); and long-term (everything a day and longer). Most of the research focus on short-term traffic prediction, such as Chao Han & Su Song (2003) and Cools *et al.* (2009), but the research on mid-to-long term traffic forecast is rather limited.

3.2. Hyperparameter Fine Tuning

Machine learning models have hyperparameters that need to be tuned. They are "points of choice or configuration that allow a machine learning model to be customized for a specific task or dataset" as stated in a "Machine Learning Mastery" article, written by Brownlee (2020). It is a model configuration argument that needs to be specified by the developer to guide the learning process for a specific dataset.

Traditionally, it is the developer who chooses the hyperparameters for a given problem. However, this demands a significant amount of experience, intuition, and trial and error. Besides, results usually aren't scientifically reproducible and sometimes even suboptimal as per Snoek *et al.* (2012). More recent results indicate that more sophisticated and automated approaches can find better hyperparameter combinations than humans (Coates *et al.*, 2011; Bergstra *et al.*, 2011)

The two most used methods are also the simplest two: grid search and random search. In a grid search, a range of values is pre-determined for a given combination of hyperparameters. Then a grid is created and iterates through every possible combination of all hyperparameter values. However, we must bear in mind that the grid grows exponentially with the number of hyperparameters to tune. Along with a low effective dimensionality, the grid is likely to be suboptimal since it will cover spaces of low importance while under-examining spaces of high

importance. In Random search, a random value is drawn from a pre-defined range of values for each hyperparameter. It can have some advantages in higher-dimensional search spaces. Bergstra and Bengio (2012) demonstrated that random search performs almost as or equally well in higher-dimensional search spaces as grid search, while being much quicker. Later, Li *et al*. (2017) introduced an extension of random search, the Hyperband, which randomly samples a set of hyperparameter configurations. These configurations are then trained for a certain number of iterations and then ranked based on their performance. Afterwards, the best configurations are chosen and trained for an additional number of iterations. This process is repeated until only few configurations remain, which are then trained for the maximum number of iterations possible to find the best configuration.

A major problem of hyperparameter optimization for learning algorithms is that it takes a long time to evaluate a given set of hyperparameters. Having this, in some works, Sequential Model-Based Optimization (SMBO) algorithms have been used when the performance evaluation of the model is expensive (Bergstra *et al.*, 2011; Hutter *et al.*, 2011; Bardenet *et al.*, 2013). SMBO's spend additional computing time calculating the most promising next hyperparameter instantiation, with the objective of diminishing the number of evaluations that the learning algorithm needs. Bayesian optimization (Bergstra *et al.*, 2013; Swersky *et al.*, 2013; Snoek *et al.*, 2012) is one of the most used methods for SMBO, and it focuses on building a probability model describing the performance for given a certain hyperparameter configuration. The model then continuously updates itself with new information gained by sample points that provide information about the performance under a given configuration of the hyperparameter.

Another possible approach is to try evolutionary and swarm algorithms for the search of optimal hyperparameters. Population-based optimization methods work well for optimization tasks over high-dimensional variable spaces, as they are able to evaluate a variety of candidate solutions in parallel. Navarro-Guerrero *et al*. (2016; 2017), Real *et al*. (2017), and Xie *et al*.

(2017) apply evolutionary algorithms to optimize a subset of hyperparameters for neural networks. Recently, several works used reinforcement learning to try to find an appropriate neural network architecture. Zoph *et al*. (2017), trained a recurrent neural network using reinforcement learning to find neural network architectures that could yield a good performance on specific tasks. Baker *et al*. (2017), constructed a Q-learning agent that was trained to find CNN architectures that perform well on various data sets. Nevertheless, many reinforced learning approaches only optimize architectural hyperparameters, while other hyperparameters such as the learning rate and regularization parameters end up being chosen manually.

4. The Models and hyperparameter optimization

4.1. The traffic forecasting models

4.1.1. Development of the algorithms

At the start of the project, an exploratory data analysis was conducted in order to understand the specifications of the datasets provided by Brisa. After the performing such procedure, the focus switched to try to find the best forecasting method for the job at hand. As previously mentioned, three different methodologies were tested: Exponential smoothing, Prophet and SARIMA. The tests began with exponential smoothing models, but the results produced were rather weak. Afterwards, Prophet was tested as well, but since the number of observations per motorway is rather small (maximum number of observations is 128 for A1 motorway) the model didn't produce very promising results. Having this, SARIMA models were then the subject of focus.

One of the main challenges presented when using SARIMA models is to find the ideal set of hyperparameters for the prediction. Usually, this procedure requires a careful analysis and domain proficiency. There are two main methods to tackle this challenge. One is to perform an analysis by decomposing the time-series and manually identify the values for the hyperparameters. Another is to develop a search algorithm to find the best combination possible. The later is the subject of analysis of this thesis and the chosen solution for the project. Three methods were tried to put in place, being them a grid search, a random search and Bayesian methods optimization through the BayessearchCV() module from scikit-learn library in Python. The last method was not possible to apply as the module performs a cross validation which is not desired in this specific context (This topic will be discussed further in subsection 4.2). The chosen solution achieves high performances in terms of accuracy but requires a high amount of computational power and time depending on the range of values that can be set for

the hyperparameters. The initial framework applied prior to the study developed for this work was a grid search algorithm adapted from a "Machine Learning Mastery" article by Brownlee (2018). The algorithm tests all the possible combinations of hyperparameters for a given range of values and calculates the error based on a train a test sets split initially. The only pre-defined value was the seasonal pattern, derived from the exploratory data analysis, which allowed the setting of the hyperparameter m to be manually set to four. The error measure chosen for the model selection was the Root Mean Squared Error (RMSE):

$$RMSE = \sqrt{\sum_{i=1}^{n} \frac{(Predicted_i - Actual_i)^2}{n}} \quad (1)$$

The best three configurations for each motorway and vehicle class combinations (36) were then saved in a CSV and saved into the repository for later utilization in the traffic forecasting process. (Results shown in subsection 4.2)

4.1.2. Choosing the exogenous factors

One of the goals of this project was to try to find new exogenous variables that could correlate with traffic growth and test the behaviour of the new SARIMA models with them. As some variables are already included in the forecasting models that Brisa has, this exercise was split into 2 different approaches. The first one includes the independent variables used by the company. The second approach adds the newly identified variables to the models.

The following steps involved desktop research that allowed to identify 28 new exogenous variables, being them from economic, demographic and tourism-related spectrums. (List of all the variables tested in Table 5)

As the project was restricted due to limited computational power, the exogenous variables and the time-series were tested for correlation so that the five top correlated variables with each motorway could be saved for further use. Afterwards, every possible combination of those five variables would be tested in the models, ending up in the selection of one to 5 variables, based on the model performance.

SARIMA models with exogenous variables (SARIMAX) are models that require some attention regarding the usage of such variables. The number of observations for the exogenous factors must be the same as for the time-series. Additionally, there is the need of having the same number of predicted observations for the exogenous variables as the periods one wants to forecast. In order to counter this issue, an exponential smoothing model was set to forecast the missing values for the exogenous factors.

After carefully looking at the added exogenous variables, one more issue has risen. Some of the new variables are only available on a yearly basis. This led to the creation of two approaches based on the granularity of the data. The first one has data on a yearly basis, and the second on a quarterly basis. For the first approach, quarterly data on the exogenous variables was averaged or summed (according to their granularity), and for the other approach, an interpolation of the quarterly data was made. The figure below shows the four mentioned approaches.

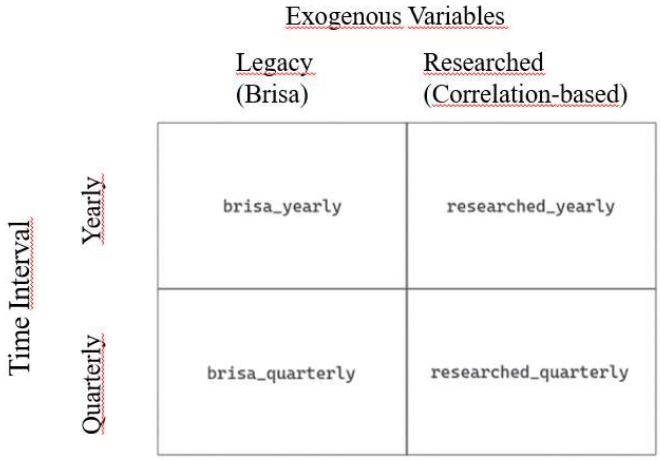

Figure 1: Model approaches

4.1.3. Selection of the models and results

As mentioned before, the error measure used to select the model hyperparameters was the RMSE. This was the same measure applied for the evaluation of the model performance. Such choice is based on the fact that the RMSE allows for interpretation of the results in the units of the time-series (km travelled). Having this, it is expected to observe a high-sensitivity of the model to outliers in the data.

As a second metric for the evaluation of the performance of the models, it was chosen MAPE (Mean Average Percentage Error), which is the sum of the individual absolute errors divided by the sum of the original observations. This measure is only possible to apply since the minimum values observed in the data are always more than zero. This metric is defined by:

$$MAPE = \frac{1}{n}\sum_{i=1}^{n}\left|\frac{Actual_i - Predicted_i}{Actual_i}\right| \quad (2)$$

The MAPE metric allows the comparison of the model performances between the different motorways, as it outputs a percentage.

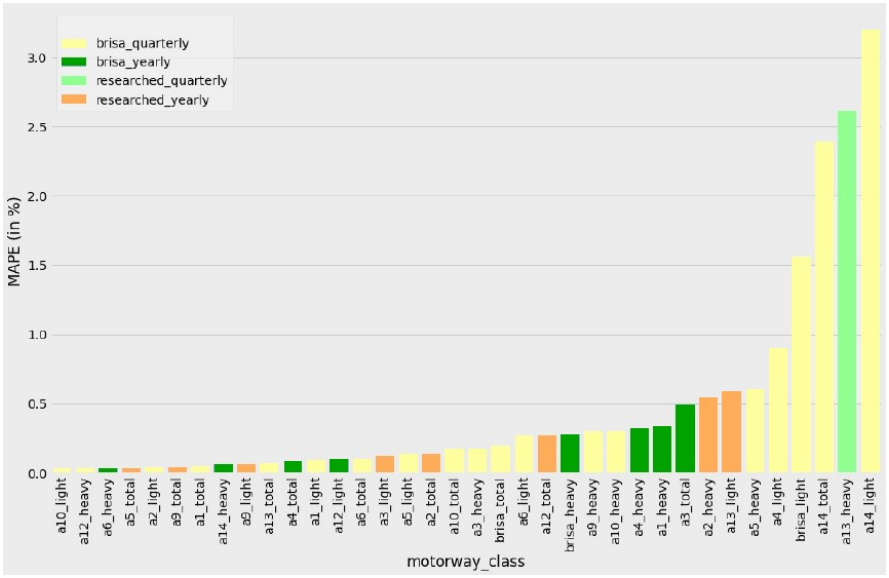

Figure 2: Best models by approach per motorway

Observing the results shown in Figure 2, we can see that most of the models achieved a very high performance. 32 out of the 36 forecasting models had a MAPE below 1%.

Now focusing on the approaches, we can see that the ones that performed best vary quite a bit. 19 out of 36 models achieved better performance by following the brisa_quarterly approach. This shows that the legacy variables used by Brisa (subsection 2.2), are the ones that correlate the most. However, the researched conducted following the goal of finding new variables that could help predict traffic growth proved to be fruitful. On some cases, the usage of new exogenous variables allowed the models to perform better. In 8 out of these 9 models, using aggregated yearly data achieved better results. Most likely, this is due to the fact that most of the new researched variables' data was only available in a yearly basis.

Table 5, in the Appendix, provides a more detailed scope regarding the final models' performance.

4.2. Hyperparameter fine tuning for the models

An optimization procedure involves defining a search space. This can be thought of geometrically as an n-dimensional volume, where each hyperparameter represents a different dimension and the scale of the dimension are the values that the hyperparameter may take on, such as real-valued, integer-valued, or categorical.

- **Search Space**: Volume to be searched where each dimension represents a hyperparameter and each point represents one model configuration.

A point in the search space is a vector with a specific value for each hyperparameter value. The goal of the optimization procedure is to find a vector that results in the best performance of the model after learning, such as maximum accuracy or minimum error.

A range of different optimization algorithms may be used, although two of the simplest and most common methods are random search and grid search.

- **Random Search**. Define a search space as a bounded domain of hyperparameter values and randomly sample points in that domain.
- **Grid Search**. Define a search space as a grid of hyperparameter values and evaluate every position in the grid.

Grid search is great for spot-checking combinations that are known to perform well generally. Random search is great for discovery and getting hyperparameter combinations that you would not have guessed intuitively, although it often requires more time to execute.

There exists a problematic in the implementation of the possible functions provided by scikit-learn. Scikit-learn allows for the experimentation of Grid Search, Random search and Bayesian Search of the hyperparameters through their functions, *GridSearchCV()*, *RandomSearchCV()* and *BayesSearchCV()*, respectively. The problem with the mentioned

functions is that they require Cross-Validation for model validation. In the specific context of this project, as we have a dataset of univariate time-series labelled data, we can't allow for Cross-validation to be performed as we don't want our data reshuffled. Instead, a system of Walk-forward validation is required. Meaning that we should split our data up until a certain "date" and test the results of the models on the remaining data.

4.2.1. The Grid Search framework developed

As mentioned before, the grid search framework implemented in Python was adapted from a "Machine Learning Mastery" article, written by Brownlee (2018).

For the implementation of the model, it was used the SARIMAX model provided by the *statsmodels* library available in Python.

The process starts by defining a function that fits the model with a given configuration and performs a one-step forecast. This function receives the data and a list of the configuration for the hyperparameters of the model, more precisely two tuples and a string. One for the order trend ((p, d, q)), the second for the seasonal order trend ((P, D, Q), m) and a string for the trend hyperparameter (it can assume the values "n" for no trend, "c" for constant trend, "t" for linear trend and "ct" for constant with linear trend).

Afterwards some functions are used for fitting and evaluating of the models repeatedly using a walk-forward validation that splits the dataset into train and test set and evaluates the one-step forecast. A function created for the RMSE metric was used to compare the forecast with the actual observations to calculate an error score.

The next step involved creating a loop to test all different possible model configurations. This is the function that drives the grid searching process. The result is a list of tuples summarizing the model, configurations and result of the error score.

The only thing left was to create a function that defined a list for the model configuration.

A figure of the full written code can be seen in the Appendix. (Figure 4)

4.2.2. The selection of the different search spaces and results

The selection of the starting point for the different search spaces was based in a "Towards Data Science" article written by Graves (2020)

The grid search framework applied in the project, was constructed prior to the study developed in this thesis, hence, the first search space (V1) was used in order to facilitate comparisons between methodologies and because it was the one used in the first trials of the algorithm.

The trend and seasonal hyperparameters of the model can be configured by analysing autocorrelation and partial autocorrelation plots, and this can take some expertise. Nonetheless, an attempt was made by the author. These plots were generated with the support of the stastmodels and matplotlib.pyplot libraries in Python. Some examples of the graphs can be seen in Figure 3 in the Appendix.

It was observed that for most of the cases, there are significant positive spikes in the autocorrelation plots at lags ranging from 1 to 7, which indicates that the parameters p and P might assume a varied range of values, and usually a significant negative spike in the partial autocorrelation plots. This suggests that one is a good starting point for the hyperparameters q and Q. The seasonal component was set to four as previously mentioned in the work. These were considered when constructing some of the search spaces (V2 and V3). Having this, the search spaces described in Table 1 were identified for testing.

For the testing of the different search spaces (V4 reaching to a size of 9216 combinations), higher computational power was required, hence, the author rented a virtual machine with 20 CPU and 64 GB RAM.

Table 1: Search spaces tested

Hyperparameter	Search space V1	Search space V2	Search space V3	Search space V4
p	[0, 1, 2, 3]	[0, 1, 2, 3, 4, 5]	[0, 1, 2, 3, 4, 5, 6, 7]	[0, 1, 2, 3, 4, 5]
d	[0, 1]	[0, 1]	[0, 1]	[0, 1]
q	[0, 1, 2]	[1]	[1]	[0, 1, 2, 3]
P	[0, 1, 2]	[0, 1, 2, 3, 4, 5]	[0, 1, 2, 3, 4, 5, 6, 7]	[0, 1, 2, 3, 4, 5]
D	[0, 1]	[0, 1]	[0, 1]	[0, 1]
Q	[0, 1, 2]	[1]	[1]	[0, 1, 2, 3]
m	[4]	[4]	[4]	[4]
t	[‘n’, ‘c’, ‘t’, ‘ct’]	[‘n’, ‘c’, ‘t’, ‘ct’]	[‘n’, ‘c’, ‘t’, ‘ct’]	[‘n’, ‘c’, ‘t’, ‘ct’]

Below, on Table 2, we can observe the result of the application of the grid search algorithm for the four search spaces.

Table 2: Best RMSE results for each motorway per search space using grid search

motorway	V1	V2	V3	V4
a1_light	53.93	53.93	53.93	29.70
a1_heavy	10.82	10.42	10.42	6.01
a1_total	55.33	72.57	72.57	51.67
a2_light	79.32	52.01	52.01	35.33
a2_heavy	1.48	1.12	1.12	0.93
a2_total	55.36	92.90	92.90	55.36

a3_light	16.11	20.51	20.51	7.11
a3_heavy	1.27	2.20	2.20	1.06
a3_total	13.66	14.02	12.40	4.79
a4_light	14.13	15.40	15.40	12.42
a4_heavy	0.32	0.32	0.32	0.32
a4_total	12.83	14.89	14.89	7.03
a5_light	7.78	10.71	8.02	7.78
a5_heavy	0.31	0.31	0.31	0.21
a5_total	8.90	9.76	9.76	7.57
a6_light	6.27	10.48	10.48	6.27
a6_heavy	0.86	0.62	0.62	0.39
a6_total	9.96	14.87	14.53	7.67
a9_light	1.60	1.60	1.60	1.60
a9_heavy	0.18	0.17	0.17	0.06
a9_total	1.58	1.58	1.58	1.21
a10_light	4.22	1.60	1.60	1.60
a10_heavy	0.27	0.42	0.39	0.21
a10_total	3.67	2.31	2.31	0.43
a12_light	6.83	6.83	6.83	2.68
a12_heavy	1.04	1.16	0.92	0.69
a12_total	8.14	9.08	7.87	5.66
a13_light	5.62	5.62	5.62	5.62
a13_heavy	0.21	0.43	0.43	0.21
a13_total	7.36	10.05	10.05	6.01
a14_light	1.80	2.55	2.55	1.45
a14_heavy	0.26	0.36	0.36	0.12
a14_total	2.63	3.21	2.87	2.21
brisa_light	140.62	208.99	192.56	109.17
brisa_heavy	16.02	14.77	14.77	11.56
brisa_total	150.68	213.75	212.01	112.94

4.2.3. Other methods tested

Since the usage of the RandomSearchCV() function from scikit-learn was not possible, an attempt to apply it was made by the author. As in random search, the procedure is to draw a random value from a pre-defined range of values for each hyperparameter, all the possible solutions shall be incorporated list of configurations created in the grid search framework. Having this, it was decided to add a randomness component to the code when it picks the configurations from the list to test on the model. (Figure 5) The number of draws was set to half the size of the configurations list. Below on Table 3, we can see the results for the different forecasting models:

Table 3: Best RMSE results for each motorway per search space using random search

motorway	**V1**	**V2**	**V3**	**V4**
a1_light	53.93	72.24	53.93	29.70
a1_heavy	10.92	10.42	40.42	6.01
a1_total	73.70	72.57	74.27	53.71
a2_light	97.30	52.01	90.93	51.26
a2_heavy	1.73	1.29	1.24	0.93
a2_total	79.99	102.78	92.90	55.36
a3_light	17.99	20.51	20.99	7.11
a3_heavy	1.43	2.20	2.20	1.19
a3_total	15.95	14.02	18.61	4.79
a4_light	14.37	15.46	15.40	12.42
a4_heavy	0.32	0.40	0.32	0.40
a4_total	12.83	15.28	14.96	13.49
a5_light	7.78	10.77	8.02	8.16
a5_heavy	0.31	0.34	0.31	0.21
a5_total	9.47	9.76	9.97	7.57
a6_light	6.27	13.03	10.48	6.27
a6_heavy	0.88	0.62	0.62	0.39
a6_total	10.25	14.87	14.53	7.67
a9_light	1.76	1.60	1.60	1.93

a9_heavy	0.20	0.18	0.20	0.06
a9_total	1.58	1.58	1.92	1.21
a10_light	4.37	1.60	1.60	1.60
a10_heavy	0.27	0.42	0.39	0.37
a10_total	3.68	3.17	2.31	1.14
a12_light	6.83	6.83	6.83	2.68
a12_heavy	1.13	1.16	0.92	0.69
a12_total	8.14	11.55	8.06	5.86
a13_light	5.62	5.62	5.62	7.30
a13_heavy	0.44	0.56	0.43	0.24
a13_total	7.36	10.52	11.58	6.01
a14_light	1.86	2.83	2.83	1.86
a14_heavy	0.26	0.36	0.36	0.12
a14_total	2.63	3.25	2.87	2.24
brisa_light	140.62	218.58	192.56	137.92
brisa_heavy	16.02	14.77	14.77	13.27
brisa_total	184.05	224.94	213.75	127.97

5. Conclusion

The purpose of the study was to develop Brisa's new traffic forecasting models using Python and develop a hyperparameter fine tuning framework that would allow the models to achieve the best possible performances.

The development of the product for the first goal showed that the study provided results. By identifying new variables, the overall performance of the models improved with the introduction of a new forecasting approach. The application of a SARIMAX model for traffic forecast, instead of Prophet or an exponential smoothing model, allowed a better performance of the algorithm as well. It was also possible to understand the importance of having exogenous variables with the same data granularity as the dataset, as it can impact model performance.

The tackle of the second goal, allowed the achievement of better model performances by the increase of the search space in analysis and the discovery of the new best hyperparameter configurations for each of the 36 forecasting models created (see Table 7). The output of the framework developed is a csv file that shall be uploaded into the Repo of the project team member Lennart Oser for the feeding of the models and the subsequent delivery of end-product with a User Interface to Brisa.

It is of high importance to have as much computational power as possible, as increasing it can only bring better results with the subsequent increase of the search space and an exhaustive search of all the possible hyperparameter configurations for the forecasting models (as per the results achieved, the method that allowed better performance was grid search and the usage of the search space V4, the largest one with 9216 possible combinations). The methodology chosen comes with the fact that the end-product of this project will only be used a few times a year, and hence, code running time is not of high relevance (Nevertheless, Table 8 provides the approximate running time of the code for both methodologies).

The adjustments made in the computational power for the performance of the project, were based on the intuition that Brisa has the availability of using high computational power machines within their *Traffic Studies and Advanced Analytics* department.

Although the desired end of this study was achieved with the increase of the models' performance, it would be interesting to explore the opportunity of implementation of other new methodologies, such as Bayesian optimization methods and differential evolutionary methods, for hyperparameter optimization of univariate time-series forecasting models such as this.

References

Baker, B., Gupta, O., Naik, N. and Raskar, R. (2017). Designing neural network architectures using reinforcement learning, *International Conference on Learning Representations*, (pp. 1–18).

Bardenet, R., Brendel, M., Kégl, B., & Sebag, M. (2013). Collaborative hyperparameter tuning, *International Conference on Machine Learning*, (pp. 199–207).

Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. (2011). Algorithms for hyper-parameter optimization, *Advances in Neural Information Processing Systems 24*, 281-305.

Bergstra, J., & Bengio, Y. (2012). Random search for hyper-parameter optimization, *Journal of Machine Learning Research* 13(1), 281-305.

Bergstra, J., Yamins, D., Cox, D. (2013). Making a science of model search: Hyperparameter optimization in hundreds of dimensions for vision architectures. *International Conference on Machine Learning*, (pp. 115-123).

Box, G. E. P., & Jenkins, G. M. (1970). Time Series Analysis: Forecasting and Control. San Francisco, USA: Holden Day.

Brisa (2021). Brisa - Offical Website.
URL https://www.brisa.pt/en/

Brown, R. G. (1959). *Statistical Forecasting for Inventory Control*. New York, USA: McGraw-Hill.

Brown, R. G. (1963). *Smoothing, forecasting and prediction of discrete time series*. Englewood Cliffs, NJ, USA: Prentice-Hall.

Brownlee, J. (2018). How to Grid Search SARIMA Hyperparameters for Time Series Forecasting.
URL https://machinelearningmastery.com/how-to-grid-search-sarima-model/-hyperparameters-for-time-series-forecasting-in-python/

Brownlee, J. (2020). Hyperparameter Optimization With Random Search and Grid Search.
URL https://machinelearningmastery.com/how-to-grid-search-sarima-model/-hyperparameters-for-time-series-forecasting-in-python/

Coates, A., Ng, A. Y. & Lee, H. (2011). An analysis of single-layer networks in unsupervised feature learning, *International Conference on Artificial Intelligence and Statistics*, (pp. 215-223).

Chao Han, & Su Song (2003). A review of some main models for traffic flow forecasting. In *Proceedings of the 2003 IEEE International Conference on Intelligent Transportation Systems*, (pp. 216‑219). IEEE.
URL http://ieeexplore.ieee.org/document/1251951/

Cools, M., Moons, E., & Wets, G. (2009). Investigating the Variability in Daily Traffic Counts through use of ARIMAX and SARIMAX Models. *Transportation Research Record: Journal of the Transportation Research Board, 2136* (1), 57‑66.
URL http://journals.sagepub.com/doi/10.3141/2136-07

Dasgupta, S., McAllester, D. (2014) (Eds.) *Proceedings of the 30th International Conference on Machine Learning (ICML'13)*. Omnipress.

De Gooijer, J. G., & Hyndman, R. J. (2006). 25 years of time series forecasting. *International Journal of Forecasting*, *22* (3).

Golovin, D., Solnik, B., Moitra, S., Kochanski, G., Karro, J. & Sculley, D. (2017). Google Vizier: A service for black-box optimization. In: Matwin, S., Yu, S., Farooq, F. (Eds.) *Proceedings of the 23rd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD)*. (pp. 1487–1495). ACM Press

Graves, A. (2020). Time Series Forecasting with a SARIMA model.
URL https:// https://towardsdatascience.com/time-series-forecasting-with-a-sarima-model-db051b7ae459

Hayes, A. (2021). What Is a Time Series?
URL https://www.investopedia.com/terms/t/timeseries.asp

Holt, C. C. (1957). Forecasting seasonals and trends by exponentially weighted moving averages. *International Journal of Forecasting, 20* (1), 5–13.

Hutter, F., Hoos, H. & Leyton-Brown, K. (2011). Sequential model-based optimization for general algorithm configuration, *International Conference on Learning and Intelligent Optimization*, (pp. 507–523)

Hutter, F., Hoos, H. & Leyton-Brown, K. (2014). An efficient approach for assessing hyper-parameter importance, *International Conference on Machine Learning*, (pp. 754–762)

Hutter, F., Lücke, J. & Schmidt-Thieme, L. (2015). Beyond manual tuning of hyperparameters, *Künstliche Intelligenz 29* (1), 329-337.

Hyndman, R. J., & Athanasopoulos, G. (2021). *Forecasting: principles and practice*. Melbourne, Australia: OTexts, 3rd edition ed.
URL https://otexts.com/fpp3/intro.html

King, R., Feng, C., Sutherland, A. (1995). Statlog: comparison of classification algorithms on large real-world problems. *Applied Artificial Intelligence an International Journal,* 9(3), 289–333.

Kohavi, R., & John, G. (1995). Automatic Parameter Selection by Minimizing Estimated Error. In Prieditis, A., & Russell, S. (Eds.) *Proceedings of the Twelfth International Conference on Machine Learning*, pp. 304–312. Morgan Kaufmann Publishers.

Kumar, N., & Susan, S. (2020). COVID-19 Pandemic Prediction using Time Series Forecasting Models. In *2020 11th International Conference on Computing, Communication and Networking Technologies (ICCCNT)*, (pp. 1‑7). IEEE.
URL https://ieeexplore.ieee.org/document/9225319/

Li, L., Jamieson, K., DeSalvo, G., Rostamizadeh, A., & Talwalkar, A. (2017). Hyperband: Banditbased configuration evaluation for hyperparameter optimization, *International Conference on Learning Representations*, (pp. 1-15).

Li, F.F. & Li, J. (2018). Cloud AutoML: Making AI accessible to every business,
URL https://www.blog.google/products/google-cloud/cloud-automl-making-ai-accessible-every-business/

Mahdavian, A., Shojaei, A., Salem, M., Laman, H., Yuan, J.-S., & Oloufa, A. (2021). Automated Machine Learning Pipeline for Traffic Count Prediction. *Modelling, 2* (4), 482‑513.

Mantovani, R., Horvath, T., Cerri, R., Vanschoren, J., Carvalho, A. (2016) Hyper-Parameter Tuning of a Decision Tree Induction Algorithm, *5th Brazilian Conference on Intelligent Systems (BRACIS)*, (pp. 37–42). IEEE Computer Society Press

Michie, D., Spiegelhalter, D., Taylor, C., Campbell, J. (1994). (Eds.) *Machine Learning, Neural and Statistical Classification*. Ellis Horwood

Montgomery, D. C., Jennings, C. L., & Kulahci, M. (2005). *Introduction to Time Series Analysis and Forecasting*. Hoboken, NJ, USA: John Wiley & Sons, Inc, 2nd ed.

Navarro-Guerrero, N., (2016). *Neurocomputational mechanisms for adaptive self-preservative robot behaviour*, Dissertation, Universität Hamburg.

Navarro-Guerrero, N., Lowe, R. & Wermter, S. (2017). Improving robot motor learning with negatively valenced reinforcement signals, *Frontiers in Neurorobotics 11*(10), 1–14.

Olson, R., La Cava, W., Mustahsan, Z., Varik, A., Moore, J. (2018). Data-driven advice for applying machine learning to bioinformatics problems, *Proceedings of the Pacific Symposium in Biocomputing*, (pp. 192–203).

Papastefanopoulos, V., Linardatos, P., & Kotsiantis, S. (2020). COVID-19: A Comparison of Time Series Methods to Forecast Percentage of Active Cases per Population. *Applied Sciences*, 10 (11).

Pegels, C. C. (1969). *Exponential Forecasting: Some New Variations. Management Science, 15* (5).

Real, E., Moore, S., Selle, A., Saxena, S., Suematsu, Y. L., Tan, J., Le, Q. V., & Kurakin, A. (2017). Large-scale evolution of image classifiers, *International Conference on Machine Learning*, (pp. 2902–2911).

Ripley, B.D. (1993). Statistical aspects of neural networks. *Networks and chaos—statistical and probabilistic aspects 50*, 40–123.

Sanders, S., Giraud-Carrier, C. (2017) Informing the Use of Hyperparameter Optimization Through Metalearning. In: Gottumukkala, R., Ning, X., Dong, G., Raghavan, V., Aluru, S., Karypis, G., Miele, L., Wu, X. (Eds.) *2017 IEEE International Conference on Big Data (Big Data)*. IEEE Computer Society Press

Shah, V. (2019). *A Comparative Study of Univariate Time-series Methods for Sales Forecasting*. Ph.D. thesis, University of Waterloo, Waterloo, Ontario, Canada.

Snoek, J., Larochelle, H., & Adams, R. P. (2012) Practical Bayesian optimization of machine learning algorithms, *Advances in Neural Information Processing Systems 25*, 2951–2959.

Swersky, K., Snoek, J., & Adams, R. P. (2013). Multi-task bayesian optimization, *Advances in Neural Information Processing Systems 26*, 2004–2012.

Taylor, S. J., & Letham, B. (2018). Forecasting at Scale. *The American Statistician, 72* (1), 37‑45.

Thornton, C., Hutter, F., Hoos, H., Leyton-Brown, K. (2013). Auto-WEKA: combined selection and hyperparameter optimization of classification algorithms. In: Dhillon, I., Koren, Y.,

Ghani, R., Senator, T., Bradley, P., Parekh, R., He, J., Grossman, R., Uthurusamy, R. (Eds.) *The 19th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining (KDD'13)*. (pp. 847–855). ACM Press.

Winters, P. R. (1960). Forecasting Sales by Exponentially Weighted Moving Averages. *Management Science*, 6 (3), 324‑342.

Xie, L. & Yuille, A. (2017). Genetic cnn, *Proceedings IEEE Conference on Computer Vision and Pattern Recognition*, (pp. 1379–1388).

Zoph, B. & Le, Q. V. (2017). Neural architecture search with reinforcement learning, *International Conference on Learning Representations*, (pp. 1–16).

Appendix

Table 4: Exogenous variables tested

Variable name	Description
25-34	Active population aged between 25-34
35-44	Active population aged between 35-44
45-54	Active population aged between 45-54
55-64	Active population aged between 55-64
business_vol_agriculture	Business volume in agriculture sector
business_vol_bankinginsurance	Business volume in Banking & Insurance sector
business_vol_construction	Business volume in construction sector
business_vol_education	Business volume in education sector
business_vol_health	Business volume in health sector
business_vol_hospitality	Business volume in hospitality sector
business_vol_manufacturing	Business volume in manufacturing sector
business_vol_rawmaterials	Business volume in raw materials sector
business_vol_realestate	Business volume in real estate sector
business_vol_retail	Business volume in retail sector
business_vol_total	Business volume total
business_vol_transportation	Business volume in transportation sector
business_vol_utilities	Business volume in utilities sector
companies	Number of registered companies
diesel	Price for diesel per litre
fuel_95	Price for fuel (95) per litre
gdp_abs	Gross domestic product (in EUR)
gdp_pct	Gross domestic product growth
heavy_vehicles	Total heavy registered vehicles
inv_rate_agriculture	Investment rate in agriculture sector
inv_rate_construction	Investment rate in construction sector
inv_rate_education	Investment rate in education sector
inv_rate_health	Investment rate in health sector
inv_rate_hospitality	Investment rate in hospitality sector
inv_rate_manufacturing	Investment rate in manufacturing sector
inv_rate_rawmaterials	Investment rate in raw materials sector
inv_rate_realestate	Investment rate in real estate sector
inv_rate_retail	Investment rate in retail sector

inv_rate_total	Investment rate total
inv_rate_transportation	Investment rate in transportation sector
inv_rate_utilities	Investment rate in utilities sector
ipi_construct	Industrial production index: Construction sector
ipi_manufact	Industrial production index: Manufacturing sector
ipi_total	Industrial production index: Total
licenses_emitted	New driving licenses emitted
pop_female	Female population (absolute)
pop_male	Male population (absolute)
pop_total	Total population
public_inv_industry	Public investment in industry sector
public_inv_transport	Public investment in transport sector
total_energy_consumption	Total energy consumption
total_hh_consumption	Total household energy consumption
total_vehicles	Total registered vehicles
tourism_exports	Travel and tourism related exports
tourism_rev_hotels	Revenues from hotel stays
tourism_rev_overnight total	Revenues from overnight stays
tourism_trips_boat	Tourist travels by residents using boats
tourism_trips_bus	Tourist travels by residents using buses
Tourism_trips_car	Tourist travels by residents using cars
Tourism_trips_plane	Tourist travels by residents using planes
Tourism_trips_total	Tourist travels by residents in total
Tourism_trips_train	Tourist travels by residents using trains

Table 5: Best model results including exogenous variables and approach used

Model	RMSE	MAPE (%)	Exogenous variables used	Approach
a1_light	6.99	0.09	['diesel', 'ipi_total', 'total_vehicles']	brisa_quarterly
a1_heavy	9.18	0.34	['fuel_95', 'diesel', 'ipi_total']	brisa_yearly
a1_total	4.32	0.05	['gdp_abs']	brisa_quarterly
a2_light	1.06	0.04	['gdp_abs', 'fuel_95', 'diesel', 'total_vehicles']	brisa_quarterly
a2_heavy	2.56	0.54	['inv_rate_total', 'inv_rate_hospitality', 'total_energy_consumption', 'inv_rate_construction']	researched_yearly
a2_total	19.14	0.14	['inv_rate_construction', 'inv_rate_hospitality', 'inv_rate_retail']	researched_yearly
a3_light	9.43	0.12	['gdp_abs', 'business_vol hospitality', 'inv_rate_rawmaterials', 'business_vol_retail']	researched_yearly
a3_heavy	0.17	0.17	['gdp_abs', 'fuel_95']	brisa_quarterly
a3_total	39.59	0.49	['diesel', 'ipi_total', 'total_vehicles']	brisa_yearly
a4_light	11.94	0.9	['gdp_abs', 'fuel_95', 'diesel', 'total_vehicles']	brisa_quarterly
a4_heavy	0.69	0.32	['gdp_abs', 'fuel_95', 'ipi_total']	brisa_yearly
a4_total	4.53	0.08	['gdp_abs', 'diesel', 'ipi_total']	brisa_yearly
a5_light	1.5	0.13	['diesel', 'ipi_total']	brisa_quarterly
a5_heavy	0.04	0.6	['fuel 95', 'ipi_total']	brisa_quarterly
a5_total	1.4	0.03	['total_energy_consumption', 'inv_rate_total']	researched_yearly
a6_light	1.88	0.27	['fuel_95', 'diesel', 'ipi_total']	brisa_quarterly
a6_heavy	0.09	0.03	['diesel', 'ipi_total', 'total_vehicles']	brisa_yearly
a6_total	0.77	0.1	['gdp_abs', 'diesel', 'total_vehicles']	brisa_quarterly
a9_light	1.61	0.06	['total_energy_consumption']	researched_yearly
a9_heavy	0.07	0.3	['gdp_abs', 'fuel_95', 'diesel']	brisa_quarterly
a9_total	1.1	0.04	['total_energy_consumption']	researched_yearly
a10_light	0.07	0.03	['gdp_abs', 'fuel_95', 'diesel', 'total_vehicles']	brisa_quarterly
a10_heavy	0.06	0.3	['gdp_abs', 'fuel_95', 'diesel', 'ipi_total']	brisa_quarterly
a10_total	0.47	0.17	['gdp_abs', 'diesel', 'ipi_total']	brisa_quarterly
a12_light	1.95	0.1	['fuel_95', 'diesel']	brisa_yearly
a12_heavy	0.01	0.03	['gdp abs', 'diesel']	brisa_quarterly

a12_total	5.63	0.27	['heavy_vehicles', 'ipi_total', 'total_energy_consumption', 'inv_rate_total']	researched_yearly
a13_light	7.44	0.59	['heavy_vehicles']	researched_yearly
a13_heavy	1.34	2.62	['inv_rate_total', 'inv_rate_retail']	researched_quarterly
a13_total	0.2	0.07	['fuel_95', 'total_vehicles']	brisa_quarterly
a14_light	3.52	3.2	['gdp_abs', 'fuel_95', 'ipi_total', 'total_vehicles']	brisa_quarterly
a14_heavy	0.02	0.06	['fuel_95', 'ipi_total']	brisa_yearly
a14_total	2.84	2.39	['gdp_abs', 'ipi_total']	brisa_quarterly
brisa_light	271.95	1.56	['gdp_abs', 'ipi_total', 'total_vehicles']	brisa_quarterly
brisa_heavy	12.55	0.28	['fuel_95', 'ipi_total']	brisa_yearly
brisa_total	37.18	0.2	['ipi_total', 'total_vehicles']	brisa_quarterly

Table 6: Final configurations per forecasting model

motorway	**Hyperparameter configuration**	**RMSE**
a1_light	[(5, 0, 2), (0, 0, 3, 4), 't']	29.70
a1_heavy	[(0, 0, 0), (3, 1, 3, 4), 'n']	6.01
a1_total	[(0, 0, 3), (1, 0, 0, 4), 'n']	51.67
a2_light	[(3, 0, 3), (4, 0, 0, 4), 'n']	35.33
a2_heavy	[(0, 1, 2), (5, 0, 2, 4), 'ct']	0.93
a2_total	[(2, 0, 2), (2, 0, 2, 4), 't']	55.36
a3_light	[(3, 0, 3), (5, 0, 1, 4), 't']	7.11
a3_heavy	[(0, 0, 0), (1, 1, 3, 4), 't']	1.06
a3_total	[(1, 0, 3), (5, 0, 0, 4), 't']	4.79
a4_light	[(2, 0, 3), (2, 0, 3, 4), 't']	12.42
a4_heavy	[(3, 1, 1), (1, 0, 1, 4), 'ct']	0.32
a4_total	[(0, 0, 1), (3, 0, 0, 4), 'n']	7.03
a5_light	[(3, 0, 2), (1, 1, 2, 4), 'c']	7.78
a5_heavy	[(2, 0, 2), (4, 1, 1, 4), 'ct']	0.21
a5_total	[(0, 0, 2), (5, 1, 0, 4), 'ct']	7.57
a6_light	[(0, 0, 2), (2, 0, 2, 4), 'n']	6.27
a6_heavy	[(1, 1, 2), (4, 1, 2, 4), 'ct']	0.39
a6_total	[(2, 0, 3), (2, 0, 1, 4), 'n']	7.67
a9_light	[(2, 0, 1), (1, 1, 1, 4), 'ct']	1.60
a9_heavy	[(5, 1, 0), (0, 1, 2, 4), 'ct']	0.06

a9_total	[(1, 0, 3), (1, 1, 2, 4), 'ct']	1.21
a10_light	[(1, 0, 1), (5, 1, 1, 4), 'ct']	1.60
a10_heavy	[(1, 0, 2), (3, 1, 0, 4), 'ct']	0.21
a10_total	[(2, 0, 3), (4, 1, 1, 4), 'ct']	0.43
a12_light	[(0, 0, 3), (4, 0, 1, 4), 'n']	2.68
a12_heavy	[(3, 1, 2), (5, 0, 2, 4), 'c']	0.69
a12_total	[(1, 1, 2), (5, 0, 3, 4), 'c']	5.66
a13_light	[(3, 0, 1), (2, 0, 1, 4), 't']	5.62
a13_heavy	[(1, 0, 0), (1, 0, 1, 4), 'ct']	0.21
a13_total	[(0, 0, 3), (5, 0, 1, 4), 'n']	6.01
a14_light	[(2, 1, 0), (4, 0, 3, 4), 'n']	1.45
a14_heavy	[(1, 0, 1), (2, 1, 3, 4), 'ct']	0.12
a14_total	[(2, 1, 2), (2, 1, 3, 4), 'ct']	2.21
brisa_light	[(2, 1, 3), (4, 0, 0, 4), 'c']	109.17
brisa_heavy	[(0, 0, 0), (3, 1, 3, 4), 'ct']	11.56
brisa_total	[(2, 1, 2), (1, 0, 3, 4), 't']	112.94

Table 7: Approximated running times of the code using a 20 CPU and 64 RAM machine

Running times	**V1**	**V2**	**V3**	**V4**
Grid search	1 hour	45 minutes	3 hours	15 hours
Random search	30 minutes	25 minutes	1 hour 30 minutes	6 hours

Figure 3: Examples of Autocorrelation and Partial Autocorrelation plots

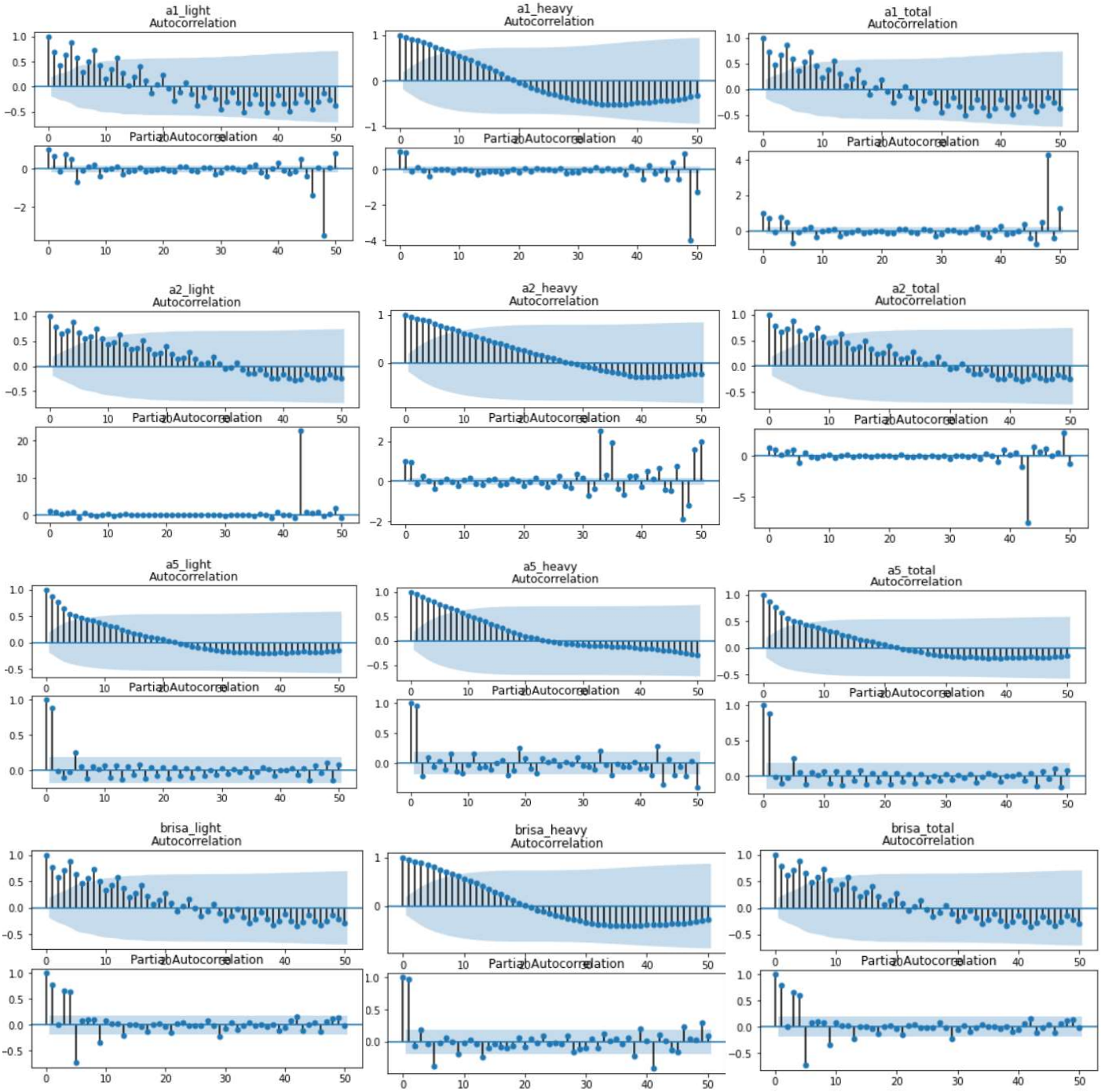

Figure 4: Grid search framework developed

```
# grid search sarima hyperparameters
from math import sqrt
from multiprocessing import cpu_count
from joblib import Parallel
from joblib import delayed
from warnings import catch_warnings
from warnings import filterwarnings
from statsmodels.tsa.statespace.sarimax import SARIMAX
from sklearn.metrics import mean_squared_error

# one-step sarima forecast
def sarima_forecast(history, config):
	order, sorder, trend = config
	# define model
	model = SARIMAX(history, order=order, seasonal_order=sorder, trend=trend,
enforce_stationarity=False, enforce_invertibility=False)
	# fit model
	model_fit = model.fit(disp=False)
	# make one step forecast
	yhat = model_fit.predict(len(history), len(history))
	return yhat[0]

# root mean squared error or rmse
def measure_rmse(actual, predicted):
	return sqrt(mean_squared_error(actual, predicted))

# split a univariate dataset into train/test sets
def train_test_split(data, n_test):
	return data[:-n_test], data[-n_test:]

# walk-forward validation for univariate data
def walk_forward_validation(data, n_test, cfg):
	predictions = list()
	# split dataset
	train, test = train_test_split(data, n_test)
	# seed history with training dataset
	history = [x for x in train]
	# step over each time-step in the test set
	for i in range(len(test)):
		# fit model and make forecast for history
		yhat = sarima_forecast(history, cfg)
		# store forecast in list of predictions
		predictions.append(yhat)
		# add actual observation to history for the next loop
		history.append(test[i])
	# estimate prediction error
	error = measure_rmse(test, predictions)
	return error

# score a model, return None on failure
def score_model(data, n_test, cfg, debug=False):
	result = None
	# convert config to a key
	key = str(cfg)
	# show all warnings and fail on exception if debugging
	if debug:
		result = walk_forward_validation(data, n_test, cfg)
	else:
		# one failure during model validation suggests an unstable config
		try:
			# never show warnings when grid searching, too noisy
			with catch_warnings():
				filterwarnings("ignore")
				result = walk_forward_validation(data, n_test, cfg)
```

```
                except:
                        error = None
        # check for an interesting result
        if result is not None:
                print(' > Model[%s] %.3f' % (key, result))
        return (key, result)

# grid search configs
def grid_search(data, cfg_list, n_test, parallel=True):
        scores = None
        if parallel:
                # execute configs in parallel
                executor = Parallel(n_jobs=cpu_count(), backend='multiprocessing')
                tasks = (delayed(score_model)(data, n_test, cfg) for cfg in cfg_list)
                scores = executor(tasks)
        else:
                scores = [score_model(data, n_test, cfg) for cfg in cfg_list]
        # remove empty results
        scores = [r for r in scores if r[1] != None]
        # sort configs by error, asc
        scores.sort(key=lambda tup: tup[1])
        return scores

# create a set of sarima configs to try
def sarima_configs(seasonal=[0]):
        models = list()
        # define config lists
        p_params = [0, 1, 2]
        d_params = [0, 1]
        q_params = [0, 1, 2]
        t_params = ['n','c','t','ct']
        P_params = [0, 1, 2]
        D_params = [0, 1]
        Q_params = [0, 1, 2]
        m_params = seasonal
        # create config instances
        for p in p_params:
                for d in d_params:
                        for q in q_params:
                                for t in t_params:
                                        for P in P_params:
                                                for D in D_params:
                                                        for Q in Q_params:
                                                                for m in m_params:
                                                                        cfg = [(p,d,q), (P,D,Q,m), t]
                                                                        models.append(cfg)
        return models

if __name__ == '__main__':
        # define dataset
        data = [10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
        print(data)
        # data split
        n_test = 4
        # model configs
        cfg_list = random.sample(sarima_configs(seasonal=[], int(len(sarima_configs=[]))/2))
        # grid search
        scores = grid_search(data, cfg_list, n_test)
        print('done')
        # list top 3 configs
        for cfg, error in scores[:3]:
                print(cfg, error)
```

Figure 5: Random search code change

```
113if __name__ == '__main__':
114        # define dataset
115        data = [10.0, 20.0, 30.0, 40.0, 50.0, 60.0, 70.0, 80.0, 90.0, 100.0]
116        print(data)
117        # data split
118        n_test = 4
119        # model configs
120        cfg_list = random.sample(sarima_configs(seasonal=[]), int(len(sarima_configs=[]))/2))
121        # grid search
122        scores = grid_search(data, cfg_list, n_test)
123        print('done')
124        # list top 3 configs
125        for cfg, error in scores[:3]:
                print(cfg, error)
```