universität wien

MASTERARBEIT / MASTER'S THESIS

Titel der Masterarbeit / Title of the Master's Thesis

Ultra-short Range Predictions of Wind Speed and Gusts using Support Vector Machines

verfasst von / submitted by

Elisabeth Hartel, BSc

angestrebter akademischer Grad / in partial fulfilment of the requirements for the degree of

Master of Science (MSc)

Wien, 2018 / Vienna, 2018

Studienkennzahl lt. Studienblatt / degree programme code as it appears on the student record sheet:	A 066 910
Studienrichtung lt. Studienblatt / degree programme as it appears on the student record sheet:	Computational Science
Betreut von / Supervisor:	Univ. Prof. Dipl.-Inform. Univ. Dr. Claudia Plant

Contents

Acknowledgement

I would like to thank my thesis supervisor Prof. Dr. Claudia Plant of the Computer Science Faculty at Universität Wien for her input with data mining expertise and direction.

I would like to especially thank Mag. Dr. Irene Schicker and Dipl.-Ing. Petrina Papazek, my supervisors at ZAMG (Zentralanstalt für Meteorologie und Geodynamik), for their continuous and invaluable input, feedback, and support.

Furthermore, I would like to thank my group leader Mag. Alexander Kann and department leader Dr. Yong Wang at ZAMG for the opportunity to research at ZAMG and their input and feedback.

List of Figures

List of Tables

Chapter 1

Introduction

What will the weather be like today? This is a question most people ask themselves on a daily basis. Sometimes looking out of a window suffices, on other days one needs more reliable information.

Why do we need reliable weather forecasts? Besides being prepared for rain and temperature, there are more serious situations like strong storms when weather forecasts are of great importance. For various reasons natural catastrophes have increased in the last decades. These also include storms that can cause substantial damage to infrastructure and health. Damages often occur due to short and intense gusts leading for example to trees and branches falling or roof shingles being removed (Klose 2016).

In wind parks it is already common to predict wind speed and energy output for a better management of inducing wind energy into the power grid. Not as common is gust prediction even though gusts are what cause most damage to wind turbines. For reliable risk management wind parks need good gust predictions.

A group of people for which wind gusts can be perilous are motorcyclists. On exposed roads high gusts can lead a biker to drift off the road or onto the opposite lane which can be incredibly dangerous. Reliable wind gust forecasts can be a big help and warning system.

As wind gusts can be especially dangerous and not as much research is done in reliably forecasting wind gusts as opposed to average wind speed, in this thesis we set a focus on forecasting wind gusts.

We implement a new forecasting method based on machine learning for wind speed and gusts to improve predictions in the nowcasting range which is the time frame of up to six hours. The research is done in cooperation with ZAMG (Zentralanstalt für Meteorologie und Geodynamik), the national meteorological service of Austria.

1.1 Meteorological Background

1.1.1 Definition of Wind and Gusts

Wind is described as a vector with two horizontal and one vertical component (`u`, `v`, `w`). Here, we neglect the vertical component and only consider the horizontal ones. Transforming the horizontal components into polar coordinates, we derive a direction component `dd` (named after the direction that the wind comes from) and a magnitude component `ff` (Klose 2016):

$$\begin{aligned} ff &= \sqrt{u^2 + v^2} \\ dd &= \arctan \frac{u}{v} \end{aligned} \tag{1.1}$$

Wind originates from the force on airmasses exerted by horizontal pressure gradients, resulting in the airmasses moving from high pressure to low pressure. Wind speed and direction

are determined by e.g. the pressure gradient, the roughness of the earth surface, geographical latitude, the vertical and horizontal temperature distribution, and vegetation. Mainly because of the roughness of the earth surface, wind exhibits turbulent and chaotic characteristics.

Wind speed and direction varies when moving vertically through the atmosphere. Here we only consider and predict the surface wind at 10 m height. For this level of the atmosphere the yearly average wind speed in European inland is 2-4 m/s, about 8-15 km/h (Häckel 2016). We generally consider the average over 10 minutes in observations, since wind direction and wind velocity are subject to extensive turbulence. These variations are even more distinct for Austria due to its complex mountainous topography. Here, the main wind phenomena have their origin in local wind systems and not in general circulation.

A wind gust is defined as a short-term difference of the average 10 minute wind speed by at least 5 m/s for a few seconds or minutes. Reasons for the variations in wind direction and speed, called gustiness, are surface roughness, exchange, and turbulence of air currents (Häckel 2016). Through the surface roughness, eddies with horizontal rotational axis form causing the air to oscillate. This exchange of air on a vertical axis causes acceleration and deceleration effects on ground level and leads to gustiness. Exchange can be caused by radiation or humidity and describes the rising of warmer air and compensating sinking of air from higher layers. As higher layers have higher wind speed, the sinking air packets arrive with higher wind speed compared to their surroundings. This accelerates the surface wind leading to short wind gusts until the surface wind adjusts to its previous speed. Because of its instability, the air current itself leads to turbulent movement and consequently to gustiness (Häckel 2016). During heavy rain and thunderstorms these gusts are also called squall limes and they are even stronger when the clouds are higher. As wind gusts are complex in nature because of the superposition of different variables and causes, they are considered intrinsically unpredictable (Sallis, Claster, and Hernández 2011).

1.1.2 Patterns in Wind Observations

Wind exhibits a clear diurnal pattern. On ground level we observe an increase in wind speed during the morning, the maximum in the afternoon, and a subsequent decrease of wind speed in the evening. This diurnal pattern is caused by the exchange of the vertical airmasses due to temperature differences. This also leads to different magnitudes of the diurnal pattern in summer and winter months, with typically the maximum of wind speed in summer months being roughly double than in winter months. It can also exhibit inverse pattern, for example for exposed mountain tops, as in mountains the complex topography and local wind systems come into play. According to Singer and Smith (1953) the relation between wind speed and gustiness is not distinct but a diurnal pattern for gustiness can also be observed.

This leads to the conclusion that meteorological time series, including wind speed and gusts, show clear annual patterns and diurnal influences, of course with variations for different locations.

1.2 Numerical Weather Prediction Models

Since the 1950s forecasts of wind speed and wind gusts have been carried out using numerical weather prediction (NWP) models. NWP models use numerical solutions of the hydrodynamic equations of the atmosphere to predict meteorological parameters (Haltiner 1971). These models are based on a set of momentum equations: the primitive equations. These equations of atmospheric dynamics have three main components: a dynamical core which deals with the basic set of equations of the adiabatic nonviscous flow, the physics equations

representing physical processes like radiation, phase transition, convection, or turbulence, and data assimilation, defining the model's initial state (Monteiro et al. 2009).

A horizontal grid and vertical layers of the atmosphere are used to solve these equations numerically. More grid points and more layers means higher resolution but also more operations for the finite difference methods. Even though the primitive equations have been continuously improved and optimized in the last decades, it is still understood that there are limitations in the predictability of atmospheric flow. Already very small differences in the initialization of an NWP model grow with time and lead to qualitatively different forecasts for a time horizon of a few weeks. Since there is a limit in the accuracy of the initial state, there is always an upper limit to forecasts with NWP models, even with perfect models.

Another challenge for NWP models is the computational requirement (number of cores, amount of time until forecast is available) which restricts its time flexibility. For nowcasting (weather forecasting for up to six hours), NWP output falls short as forecasts are already old.

There are many different kinds of NWP models for different scales. A global model used at ZAMG is the forecast by the Integrated Forecasting System (IFS) from the European Centre for Medium-Range Weather Forecasts (ECMWF). This forecast is run every 12 hours. With the growth of computer resources the spatial and temporal resolution of these weather forecasts have evolved.

For selected regions a finer mesh size can be used. These models are called mesoscale models. They use higher horizontal resolutions for processes with horizontal scales between one and a couple hundred kilometers and a temporal output resolution of one hour or less. This is especially important to reproduce local weather phenomena like thunderstorms or valley wind systems. These models need boundary conditions which can be derived from global models.

In Austria the current operational NWP models are ALARO and AROME, both derived from ALADIN (Aire Limitée Adaption Dynamique Development International), a model from Meteo-France (French national meteorological service). ALARO is the current version of ALADIN. It has a horizontal resolution of 4.8 km, uses 60 layers vertically, runs four times a day, and has a forecast output in one-hour temporal resolution. The forecast horizon is 72 hours and the boundary values are derived from the global IFS model of ECMWF. The AROME (Application of Research to Operations at Mesoscale) model is also derived from ALADIN but has even higher spatial (at least 2.4 km) and temporal resolution (8 times daily) and a shorter forecast horizon of 60 hours. It is also nested into the IFS forecast of ECMWF and has a finer grid for initial conditions that is build from observational data. Compared to ALARO it also includes more interactions between the different classes of hydrometeors, which are the different classes of water or ice particles that are formed in the atmosphere or at the surface of the earth.

The problem both of these NWP models regarding nowcasting with is the computational complexity, which leads to several hours of computation before the forecast is available. If these NWP models are used for the nowcasting range, we need to use a forecast with an older initialization to get nowcasting predictions. Additionally, the NWP forecast has limited horizontal resolution which neglects small-scale phenomena leading to the fact that even a simple persistence model can be better than the NWP model for the nowcasting range (Haiden et al. 2010; Monteiro et al. 2009). In recent years this has lead to the application of statistical and machine learning methods with less computational complexity for various weather forecasts especially for the nowcasting range.

Another operational model at ZAMG is INCA (Integrated Nowcasting through Comprehensive Analysis). It is a model for Austria with spatial resolution of one km and forecasts for every hour for temperature, humidity, wind, and every 15 minutes for cloudiness and precipitation. Its forecast horizon is primarily 12 hours. It improves forecasts of NWP models (ALADIN and ECMWF) by combining them with current observational data and topographical information.

Wind gust forecasts are not a standard output from NWP models but they can be calculated by adding a turbulence component and a convective component to the mean wind speed (Thorarinsdottir and Johnson 2012). Lower resolution NWP models easily miss local wind and gust phenomena. INCA calculates wind gust forecasts based on gust speed observations at surface stations and while relating the mean wind analysis to gust speeds with a gust factor (Haiden et al. 2010).

1.3 Related Work: Machine Learning Methods in Wind and Gust Forecasts

In recent years a variety of machine learning methods have been tested for short-term forecasting. A summary of relevant work for wind speed and gust speed forecasting is given in this section.

1.3.1 Wind Speed Forecasting

Most of the implementations focus on wind speed prediction, with some of those on wind power forecasting for application in wind parks.

Zhou, Shi, and Li (2011) use least squares support vector machines (LS-SVM) to forecast short-term wind speed in hourly frequency. The focus is on parameter tuning for one-step ahead predictions. Pinto et al. (2014) compare the performance of various SVM models to artificial neural network models (ANN) for short-term wind speed forecasting with application in wind energy. Data with five-minute-frequency is used for one-step forecasts and an SVM model achieves best results. Browell, Drew, and Philippopoulos (2017b) use spatio-temporal interdependency to improve very-short-term wind forecasting. They use an autoregressive model (AR) where the output depends linearly on its own previous values and on a stochastic term. Ahmed, Khalid, and Akram (2017) test wind speed time series forecasting for a wind farm in Pakistan with support vector regression (SVR) for hourly wind speed data and compare it to an ANN model. The linear SVR model outperforms the ANN model. Mohandes et al. (2004) use SVMs for mean daily wind speed predictions in Saudi Arabia. The SVMs outperform a multilayer perceptron neural network. Sreelakshmi and Kumar (2008) evaluate different short-term wind speed prediction techniques with SVMs being faster and having better accuracy than neural networks. Kong et al. (2015) preselect a subset of data for wind speed prediction with reduced support vector machines (RSVM) to have a smaller optimization problem while using particle swarm optimization to optimize the parameters. Zhao et al. (2010) apply SVR for wind speed predictions and compare it to back propagation neural networks using wind speed time series from a wind farm in Sweden. They find both methods applicable for prediction of wind speed time series. Wani and Bhat (2017) compare different multiclass SVM algorithms for wind speed prediction. Here, the focus is on predicting classes instead of numerical values. Botha and Walt (2017) forecast hourly wind speed using SVR and they explore the effect of the feature selection by comparing various feature combinations.

In recent years there have been more complex and novel models implemented for wind speed forecasting. Some of them are: Hu, Zhang, Xie, et al. (2014) use a general noise model for ν-SVR for short-term wind speed forecasting. ν-SVR uses the parameter ν to explicitly control the amount of support vectors whereas ϵ-SVR controls the amount of error ϵ allowed for the model. They improve current regression techniques as they do not follow the assumption that error distribution is Gaussian. In Hu, Zhang, Yu, et al. (2016) this idea is developed further, as they find that very-short-term prediction errors do actually obey Gaussian distribution. They developed a framework of ν-SVR which learns tasks of Gaussian noise with heteroscedasticity

(different variables have different finite variance) for predicting short-term wind speed and wind power. Santamaría-Bonfil, Reyes-Ballesteros, and Gershenson (2016) propose a hybrid methodology based on SVMs for hourly wind speed predictions. They use an autoregressive model for feature selection which uses the phase space reconstruction procedure. A genetic algorithm is used to tune the hyperparameters of SVR. Wang et al. (2018) implement a novel multi-step ahead wind speed forecasting multi-kernel learning based regression. They focus on correlation among different prediction steps, though this novel model has a training time of 10-20 minutes and is, therefore, not feasible for our nowcasting horizon. Liu, Niu, et al. (2014) use a hybrid algorithm composed of wavelet transform, genetic algorithm, and SVMs to predict short-term wind speed with 30-min frequency. Kang et al. (2017) use hybrid ensemble empirical mode decomposition and LS-SVMs to produce multi-step-ahead wind speed forecasts in 10 minute intervals with data from a wind farm in Galicia, Spain. Kazor and Hering (2015) explore the role of regimes for short-term wind speed forecasting. They use Gaussian mixture models based on local information for the best forecasts and use a set of regimes to identify predominant wind patterns for a specific geographic region. Schicker et al. (2017) focus on short-range hourly wind speed predictions using an interval-based neural network developed at ZAMG with both observations and NWP data as input. More background on the development of these models is found in Papazek (2018).

Wind Energy Forecasting

There are other methods with emphasis on wind power prediction to predict wind power that will be available for the energy grid. Zeng and Qiao (2011) focus on hourly short-term wind power forecasting with SVMs. They do not predict the wind power directly but predict wind speed first, use this to make a wind power prediction, and find that with a longer prediction horizon the accuracy decreases. Dowell and Pinson (2016) explore five-minute-ahead probabilistic wind power forecasts using sparse vector autoregression while focusing on building a spatio-temporal model. Browell and Gilbert (2017) use a regime-switching autoregressive model for wind power forecasting. They focus on different meteorological regimes to model distinct behaviors separately. This approach won the EEM (European Energy Market) 2017 Wind Power Forecasting Competition. Monteiro et al. (2009) give a summary of wind power forecasting for a more conclusive overview.

1.3.2 Gust Forecasting

Wind gust forecasting is not as explored as wind speed forecasting, therefore the selection is not as extensive. Nevertheless a range of methods have been investigated: Kretzschmar et al. (2004) are the first to predict wind gusts with predicting average hourly wind speed along with hourly wind gust maximum values for a prediction horizon of 24 hours. They use neural network classifiers and select input features based on empirical tests. Spudić, Marić, and Perić (2009) use spatially distributed short-term prediction of wind gusts based on neural networks for use in advance wind turbine control systems. Shanmuganathan and Sallis (2014) explore data mining techniques to generate wind gust models for application in vineyard management decision making. They focus on predicting gust classes and conclude that data mining methods can have significant impact in unravelling location specific patterns using more recent weather conditions. Thorarinsdottir and Johnson (2012) explore probabilistic wind gust forecasting using nonhomogeneous Gaussian regression. Mercer and Dyer (2014) use SVR for daily peak wind gust prediction in the American Midwest. Instead of observations they use output from NWP models as input variables. Sprenger et al. (2017) use the AdaBoost machine learning algorithm to forecast foehn events which are a specific version of strong gusty winds in the alps.

They focus on a categorical (yes/no) prediction of foehn events. Sheridan (2018) also gives a comprehensive overview of current gust forecasting methodologies for further references.

1.4 Objective of this Thesis

The goal of this thesis is to build a new model for predicting wind speed and wind gusts for the nowcasting horizon. The model is based on machine learning, specifically support vector machines and forecasts wind speed and gusts with a 10 minute frequency. Requirements are that the model performs well and runs fast and can be expanded to multiple forecasting stations. The forecast is not a grid forecast but is run for isolated stations (point-forecast). To achieve a well-performing model several computational tests are run to determine optimal hyperparameters for the algorithm and the best data selection. Additionally, the model is compared to several baseline models.

The SVR algorithm is chosen as it has proven excellent performance for regression and time series prediction (Müller et al. 1997) and several studies have shown that SVR outperform neural networks (Khosravi et al. 2018; Botha and Walt 2017). SVR models are computationally faster for the prediction step which is of importance for the nowcasting horizon. Another advantage of SVR is that models are not subject to local minima like neural networks (Smola and Schölkopf 2004). SVR leads to a unique deterministic model while neural networks depend on the initial conditions. Therefore, we only need one model instead of averaging multiple which leads to faster predictions.

1.5 Thesis Overview

Chapter 2 gives an overview over the principles of machine learning and the mathematical background of support vector regression. Chapter 3 takes a closer look into the available data. followed by the forecasting methodology in Chapter 4 which explains computational tests and models build. In Chapter 5 we analyze and evaluate the models and their performance and conclude with a discussion of the results in Chapter 6.

Chapter 2

Machine Learning Principles

Data mining is an application used in many different research areas. The steps usually taken in a data mining process can be summarized as (Plant 2017):

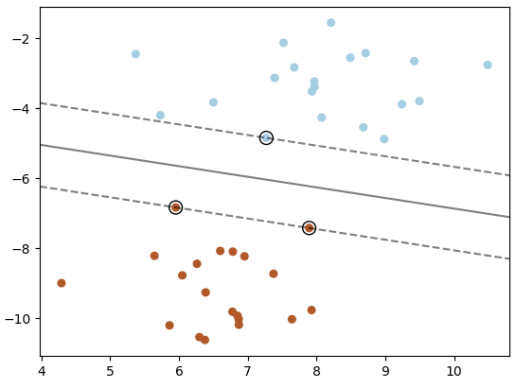
1. Selection of relevant data
2. Preprocessing and cleaning of the data
3. Transformation of data: selection of useful features, dimensionality reduction, feature engineering
4. Data mining: search for patterns in the data
5. Evaluation: statistical validation of the model, evaluation of patterns based on relevant measures

Data mining tasks can be separated into supervised and unsupervised problems. Supervised problems have target values or classification labels which are known for training data while for unsupervised problems these do not exist and relationships between variables have to be learned.

In this study we have training data where the target values are known, it is thus a supervised problem with predictive modeling: the goal is to predict the behavior of new data based on a model. Another important characteristic of the data is the time-dependency. For time series data we want to be careful to conserve the time dependency during the data mining process, ideally using methods which exploit this characteristic. Since we have all observational data with numeric values, we are using regression methods, specifically support vector regression (SVR).

2.1 Principles of Support Vector Regression

Originally, support vector machines (SVMs) were developed to solve classification problems, ergo classifying data into distinct categories. For data that can be classified into different classes, the SVM tries to find a hyperplane that separates these classes from each other. A hyperplane is a subspace with one dimension less than the input space. A simple example of this is shown in Figure 2.1. Here, the sample data set is two dimensional and the hyperplane with the largest margin to both classes (maximum margin separating hyperplane) and equal distance to both is just a line (one-dimensional). The support vectors defining the hyperplane are highlighted. Since it is not always possible to find a separating hyperplane in the original parameter space, the basic idea of SVM is to map data into a high-dimensional feature space where the problem is linearly separable and to solve the problem in this higher dimensional space (this concept is expanded in Section 2.1.2) (Müller et al. 1997).



source: https://scikit-learn.org/stable/_images/sphx_glr_plot_separating_hyperplane_001.png accessed 2018/10/30

Figure 2.1: Maximum margin separating hyperplane for two-class SVM (support vectors are highlighted).

2.1.1 Linear ϵ-SVR

SVMs can be adapted for regression problems where the goal is to find a regression function $y = f(x)$ which can correctly predict the output y for a set of d-dimensional input data $x \in \mathcal{X}$ (input space $\mathcal{X} = \mathbb{R}^d$) with a complexity term ω and threshold b (Müller et al. 1999; Smola and Schölkopf 2004):

$$f(x) = \langle \omega \cdot x \rangle + b \quad \text{with} \quad \omega \in \mathcal{X}, b \in \mathbb{R} \tag{2.1}$$

Primal formulation

Specifically, we use a variant called epsilon-insensitive SVR (ϵ-SVR). It aims at finding a function $f(x)$ which has, on one hand, at most a deviation of ϵ from the actually obtained targets y_i for each training point x_i. On the other hand, it shall be as flat as possible. Epsilon-insensitive means that the function does not consider small residual errors of the magnitude of epsilon (Kramer and Gieseke 2011). Flatness can be ensured by choosing a small ω, which can be determined by minimizing its norm $\|\omega\|^2$. This can be written as a convex optimization problem:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\omega\|^2 \\ &\text{subject to } \begin{cases} y_i - \langle \omega, x_i \rangle - b \leq \epsilon \\ \langle \omega, x_i \rangle + b - y_i \leq \epsilon \end{cases} \end{aligned} \tag{2.2}$$

One assumption here is that this convex optimization problem is feasible, meaning a function $f(x)$ exists with these constraints. This is not necessarily the case as there might be no function $f(x)$ satisfying these constraints for all x_i. Therefore, we introduce slack variables ξ_i and ξ_i^* to deal with the infeasible constraints of the optimization problem. Slack variables in SVR are error terms that permit greater regression errors while penalizing them at the same time. The slack variables permit regression errors up to ξ_i and ξ_i^* while still satisfying the conditions. This is also called the soft margin concept as opposed to the hard epsilon margin

concept in Equation 2.2. This is the standard form of ϵ-SVR, also called the primal formulation (Vapnik 1998) with the sample size l (amount of training points $x_i \in (x_1, ..., x_l)$):

$$\begin{aligned} &\text{minimize } \frac{1}{2}\|\omega\|^2 + C\sum_{i=1}^{l}(\xi_i + \xi_i^*) \\ &\text{subject to } \begin{cases} y_i - \langle \omega, x_i \rangle - b & \leq \epsilon + \xi_i \\ \langle \omega, x_i \rangle + b - y_i & \leq \epsilon + \xi_i^* \\ \xi_i, \xi_i^* & \geq 0 \end{cases} \end{aligned} \tag{2.3}$$

The constant $C > 0$ defines the balance between the flatness of f and the tolerance for errors larger than ϵ. It helps to prevent overfitting as it controls the penalty on observations that lie outside the epsilon margin.

Loss function

The loss function for ϵ-SVR ignores errors that are smaller than ϵ, meaning it treats them as equal to zero. The ϵ-insensitive loss function $|\xi|_\epsilon$ is given by

$$|\xi|_\epsilon := \begin{cases} 0 & \text{if } |\xi| \leq \epsilon \\ |\xi| - \epsilon & \text{otherwise} \end{cases} \tag{2.4}$$

This means that for the soft margin as defined in Equation 2.3, deviations outside of the epsilon margin are penalized linearly.

Dual formulation

Usually, optimization problems are not solved in their standard form (Equation 2.3) but in their Lagrange dual form as this is computationally easier to solve. The optimal solution for the primal and the dual problem is not necessarily equivalent but the dual formulation gives a lower bound for the primal solution (Smola and Schölkopf 2004). This difference between the solutions is also called the duality gap which vanishes at optimality.

It can also be shown that the dual function has a saddle point with respect to the primal and dual variables at the solution. To construct the dual formulation from the primal, we introduce a dual set of variables: α_i, α_i^*. These are the Lagrange multipliers which are used to construct the Lagrangian.[1] The Lagrangian L can be summarized as (α refers to both α_i and α_i^*):

$$L(\alpha) = \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle x_i, x_j \rangle + \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \tag{2.5}$$

Through minimization we obtain the dual formulation of the optimization problem:

$$\begin{aligned} &\text{minimize } L(\alpha) \\ &\text{subject to } \begin{cases} \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases} \end{aligned} \tag{2.6}$$

The vector ω can be written in terms of the data points (derived from the saddle point condition for optimality: $\partial_\omega L = 0$) (Smola and Schölkopf 2004):

$$\omega = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)x_i \tag{2.7}$$

[1] for construction of Lagrangian see (Smola and Schölkopf 2004) or any textbook on Lagrangian mechanics

The approximation function for predicting new values is then described as:

$$f(x) = \sum_{i=1}^{l} (\alpha_i - \alpha_i^*)\langle x_i, x\rangle + b \tag{2.8}$$

To get optimal solutions, we also need the Karush-Kuhn-Tucker (KKT) conditions which state that the product between the dual variables and the constraints has to vanish at the point of the solution (Smola and Schölkopf 2004):

$$\begin{aligned} \alpha_i(\epsilon + \xi_i - y_i + \langle \omega, x_i\rangle + b) &= 0 \\ \alpha_i^*(\epsilon + \xi_i^* + y_i - \langle \omega, x_i\rangle - b) &= 0 \\ (C - \alpha_i)\xi_i &= 0 \\ (C - \alpha_i^*)\xi_i^* &= 0 \end{aligned} \tag{2.9}$$

The KKT conditions ensure that the Lagrange multipliers α_i and α_i^* are often sparse as observations which lie within the epsilon tube have Lagrange multipliers set to zero. The corresponding x_i is called a support vector only if either α_i or α_i^* is nonzero.

2.1.2 Nonlinear SVR

To expand the SVR concept to problems with nonlinear relationships between data x and targets y, we try to find a mapping function Φ for the data such that we can do a linear regression in the high-dimensional feature space $\mathcal{F}$:

$$f(x) = (\omega \cdot \Phi(x)) + b \quad \text{with} \quad \Phi : \mathcal{X} \to \mathcal{F}, \omega \in \mathcal{F} \tag{2.10}$$

Linear regression in the feature space $\mathcal{F}$ corresponds to nonlinear regression in the low dimensional input space $\mathcal{X}$. Here, optimization means finding the flattest function in the feature space as opposed to the input space.

Kernel Trick

Instead of having to calculate the dot product $\omega \cdot \Phi(x)$ in the higher dimensional space, we can calculate it using the kernel trick $k(x_i, x_j) = (\Phi(x_i) \cdot \Phi(x_j))$, which allows us express the dot product implicitly in the low dimensional input space $\mathcal{X}$. Figure 2.2 shows the kernel trick for a two-class support vector classification. It is a good example how this method simplifies the classification by transforming the data into a high-dimensional input space. Here, the kernel function is a simple polynomial.

Functions $k(x_i, x_j)$ that correspond to a dot product in some feature space $\mathcal{F}$ are characterized by Mercer's condition which can be summarized by the following equation (defined on input space $\mathcal{X}$) (Smola and Schölkopf 2004):

$$\int_{\mathcal{X}\times\mathcal{X}} k(x_i, x_j) f(x_i) f(x_j) dx_i dx_j \geq 0 \quad \text{for all} \quad f \in L_2(\mathcal{X}) \tag{2.11}$$

Any symmetric kernel function $k(x_i, x_j)$ that satisfies this condition corresponds to a dot product in some feature space and can be used as a kernel. Common kernels are the linear kernel

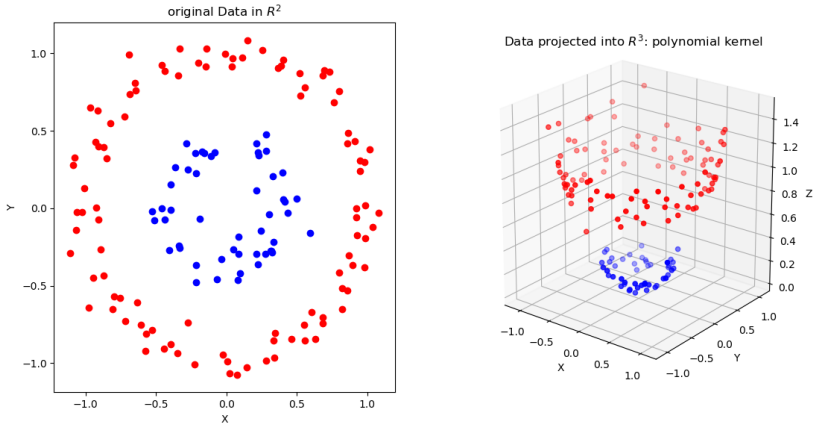
$$k(x_i, x_j) = \langle x_i, x_j\rangle, \tag{2.12}$$

the polynomial kernel (independent term r, degree of the polynomial d, and kernel coefficient γ)

$$k(x_i, x_j) = (\gamma\langle x_i, x_j\rangle + r)^d, \tag{2.13}$$

and the radial basis function (RBF) kernel (kernel coefficient $\gamma > 0$)

$$k(x_i, x_j) = exp(-\gamma\|x_i - x_j\|^2). \tag{2.14}$$



code adapted from `http://www.eric-kim.net/eric-kim-net/posts/1/kernel_trick.html` accessed 2018/11/01

Figure 2.2: Kernel trick for two-class SVM.

Dual formulation

As we are dealing with non-linear SVR, the kernel trick is used to describe the dual function for nonlinear problems:

$$\begin{aligned} &\text{minimize } \frac{1}{2}\sum_{i,j=1}^{l}(\alpha_i - \alpha_i^*)(\alpha_j - \alpha_j^*)\langle\Phi(x_i),\Phi(x_j)\rangle + \epsilon\sum_{i=1}^{l}(\alpha_i + \alpha_i^*) + \sum_{i=1}^{l} y_i(\alpha_i - \alpha_i^*) \\ &\text{subject to } \begin{cases} \sum_{i=1}^{l}(\alpha_i - \alpha_i^*) = 0 \\ \alpha_i, \alpha_i^* \in [0, C] \end{cases} \end{aligned} \tag{2.15}$$

ω can now be expressed as

$$\omega = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)\Phi(x_i) \tag{2.16}$$

Combining Equations 2.10 and 2.16, the non-linear support vector expansion is then described as:

$$f(x) = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)(\Phi(x_i)\cdot\Phi(x)) + b = \sum_{i=1}^{l}(\alpha_i - \alpha_i^*)k(x_i, x) + b \tag{2.17}$$

with the non-complementary Karush-Kuhn-Tucker conditions:

$$\begin{aligned} \alpha_i(\epsilon + \xi_i - y_i + f(x_i)) &= 0 \\ \alpha_i^*(\epsilon + \xi_i^* + y_i - f(x_i)) &= 0 \\ (C - \alpha_i)\xi_i &= 0 \\ (C - \alpha_i^*)\xi_i^* &= 0 \end{aligned} \tag{2.18}$$

2.1.3 Solution of SVR

This dual optimization problem can be expressed as a quadratic programming problem with the solution vectors α_i and α_i^*. There are different solution techniques for quadratic programming problems. The implementation used here is based on LIBSVM (LIBrary for Support Vector Machines) (Chang and Lin 2011) which solves the dual formulation of SVR as shown in Equation 2.15.

The kernel matrix $Q_{i,j} = k(x_i, x_j)$, a positive semidefinite matrix of size $l \times l$, might be too large to store as it is a dense matrix. Therefore, most implementation use a decomposition method for the kernel matrix Q. Decomposition means that not the whole matrix is accessed for each iteration but only a subset of the solution vectors α and, therefore, only some columns of Q are needed for each iteration.

The LIBSVM implementation uses a type of Sequential Minimal Optimization (SMO) which uses only a subset of two elements of α and which has good convergence properties. For a more detailed description of the implementation, see Chang and Lin (2011).

Once an SVR model is trained, the prediction process is quite simple. Following along the approximation function (Equation 2.17), the input vector x along with the support vectors x_i is mapped into the feature space using Φ. Then the kernel functions $k(x_i, x)$ are evaluated by calculating the dot products. Finally, the dot products are added up using the weights $\alpha_i - \alpha_i^*$ and the constant b and we have the prediction output.

Before a model can be trained and predictions are made, we need to look into the properties of the input data.

Chapter 3

Data Analysis

The data used for this thesis is provided by ZAMG and consists of observational data from a subset of the semiautomatic weather stations (TAWES) in Austria. There are about 250 of those stations all over Austria which take various meteorological measurements, most of those with 10-minute frequency. For this thesis only observational data was used as the focus is the nowcasting horizon (forecasting for up to 6 hours). For these short time frames the calculation of the NWP forecasts takes too long to be relevant as they are already old when the calculations are done.

The provided data from the ZAMG TAWES database includes a variety of weather observables like wind speed, wind gust, rain, sunshine, air pressure, relative humidity. In data mining these measurable properties are called features. Not all of them correlate with the values that need to be predicted, so there evidently needs to be some feature elimination. From a meteorological point of view only certain features seem to be useful for prediction of wind speed and wind gusts. Which of those are actually relevant for the forecasting methodology is tested through different computational tests. The features available and tested from the TAWES database are explained in Table 3.1. They are available with a 10-minute frequency and always refer to the last 10 minutes.

To cover different topological and climatological conditions for the methodology 24 different TAWES stations were chosen. The stations chosen are listed in Table 3.2 and an overview of the positions of the stations is shown in Figure 3.1. As one of the aims of this thesis is to have fast running forecasting code, the focus is on getting point forecasts and not grid forecasts. Data analysis is a first step to get familiar with the data and its characteristics.

Figure 3.1: Locations of 24 selected TAWES stations.

feature name	data type	description
statnr	int	station number of TAWES
datum	int	date in format YYYMMDD
stdmin	smallint	HHMM: time of day
tl	smallint	air temperature in 1/10th degree C
dd	smallint	wind direction in degree (0-360; 360=calm)
ddx	smallint	wind direction of wind gust in degree (0-360)
ff	smallint	wind speed in 1/10th m/s (vectorial 10-min mean)
ffx	smallint	wind gust in 1/10th m/s
p	smallint	air pressure in 1/10th hPA
pred	smallint	reduced air pressure in 1/10th hPA
rf	smallint	relative humidity in percent
tp	smallint	dew point temperature in 1/10 degree C
so	smallint	sunshine duration in seconds
rr	smallint	precipitation in 1/10 mm (10-min sum)

Table 3.1: Features available from TAWES database with 10-minute frequency.

statnr	name	state	lat (N)	long (E)	alt. (hm)
11007	Kollerschlag	OOE	483610	135021	714
11035	Wien-Hohe Warte	WIE	481454	162123	198
11070	Krems	NOE	482506	153717	203
11320	Innsbruck-Univ.	TIR	471536	112303	578
11126	Patscherkofel	TIR	471232	112744	2251
11343	Sonnblick	SAL	470315	125727	3109
11346	Rauris	SAL	471325	125933	934
11025	Weitra	NOE	484208	145355	572
11180	Rax/Seilbahnbergstation	NOE	474303	154643	1547
11290	Graz-Universität	STMK	470440	152656	367
11344	Kolm Saigurn	SAL	470410	125905	1626
11101	Bregenz	VBG	472957	94446	424
11108	Gaschurn	VBG	465909	100143	976
11198	Güssing	BGL	470345	161919	215
11170	Lunz am See	NOE	475116	150403	612
11273	Gmünd	KNT	465410	133203	738
11216	Kanzelhöhe	KNT	464038	135407	1520
11036	Schwechat/Flughafen	NOE	480703	163453	183
11380	Reichenau/Rax	NOE	474159	155013	488
11383	Semmering	NOE	473760	154942	988
11384	Hirschenkogel	NOE	473724	155000	1318
11358	Bad Mitterndorf	STMK	473312	135605	814
11246	Fürstenfeld	STMK	470151	160451	271
11012	Kremsmünster	OOE	480318	140752	382

Table 3.2: List of selected TAWES stations (ZAMG 2018).

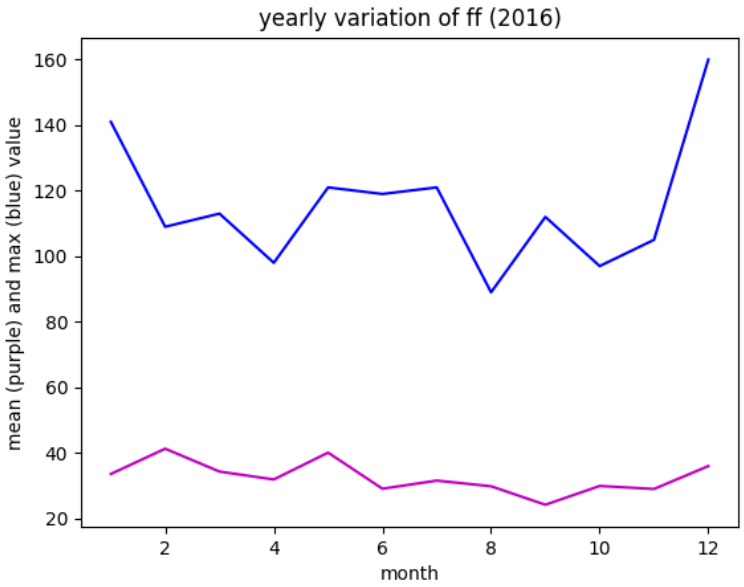


Figure 3.2: Monthly average (purple) and maximum (blue) for wind speed, 2016 (station Wien Hohe Warte 11035).

3.1 Data Characteristics

The data used for training the models is data of the years 2013 to 2017. It is available with 10 minute frequency which gives an abundance of data points: for one year this means there are about 52500 data points per feature. Therefore, it is necessary to perform careful preselections.

To get an overview over the data characteristics, it was analyzed and visualized with various statistical measures. Figures 3.2 and 3.3 give an overview of one year of wind speed and gust data respectively for the Wien Hohe Warte station (station number 11035). We can see that there is some variance for the different months but it is hard to identify a general pattern just from monthly averages and maximums. Other measures like covariance matrices for different features to find important or relevant features were also used but gave no satisfactory results.

In Figure 3.4 data of one year for wind speed and gusts is visualized for station Wien Hohe Warte using a wind rose plot. One can state that in Vienna the predominant wind direction is west. The speed difference for the average 10-minute wind and for wind gusts is also easily visible as the bin boundaries are the same. In Figure 3.5 wind speed and gust speed data for station Gmünd is shown. The bins are set the same as for Figure 3.4, we can deduce that wind and gust speed in Gmünd is overall weaker than at Wien Hohe Warte. We can also observe that major wind directions are south-west and north/north-east. This is caused by the station being in an alpine valley that goes from south-west to north/north-east.

Characteristics of the wind data can be seen when using the autocorrelation function. Autocorrelation explains the factor of randomness within the data. If data is random, then the autocorrelation function is near zero for all time lags. Sinusoidal behavior of the autocorrelation plot links to repeating patterns in the original data.

Figure 3.6 and Figure 3.7 show the autocorrelation function for wind gusts for summer and winter months respectively. For wind speed the plots are very similar. One lag is the interval between two data points (10 minutes). For the summer months there is a clear diurnal pattern observable: 24 hours equal 144 lags which is about the distance from one local maximum to

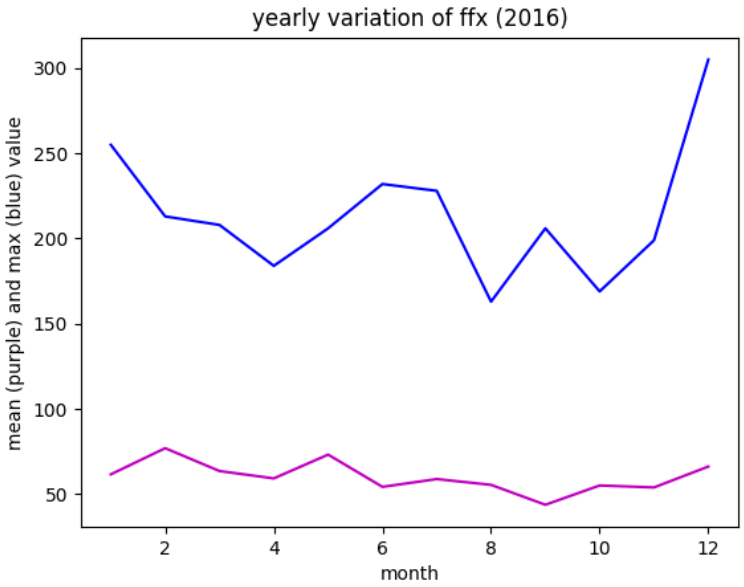
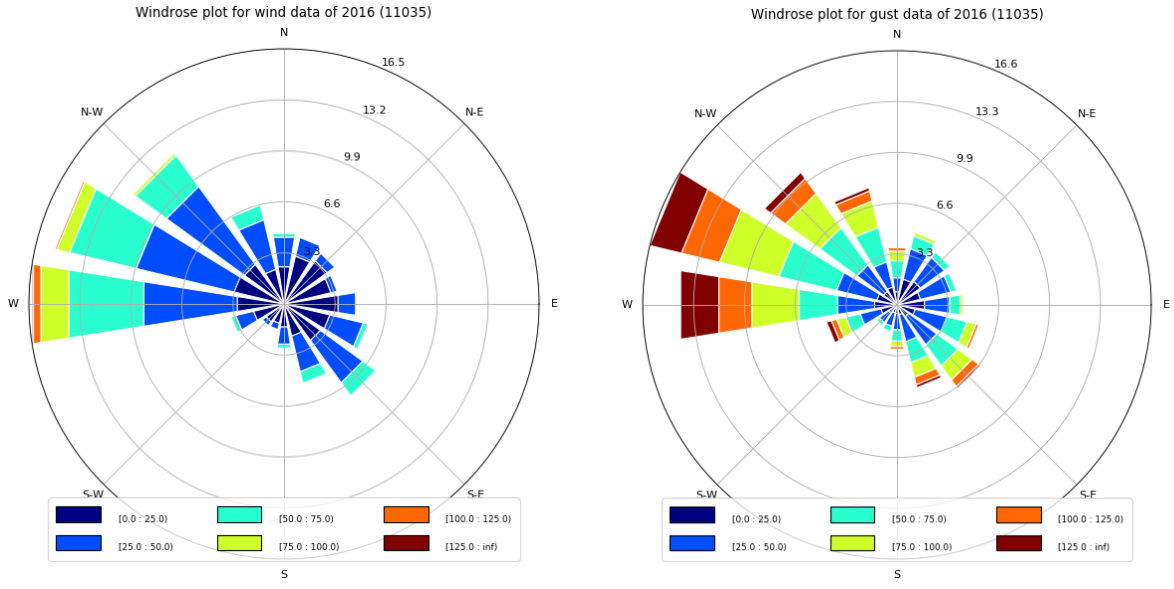


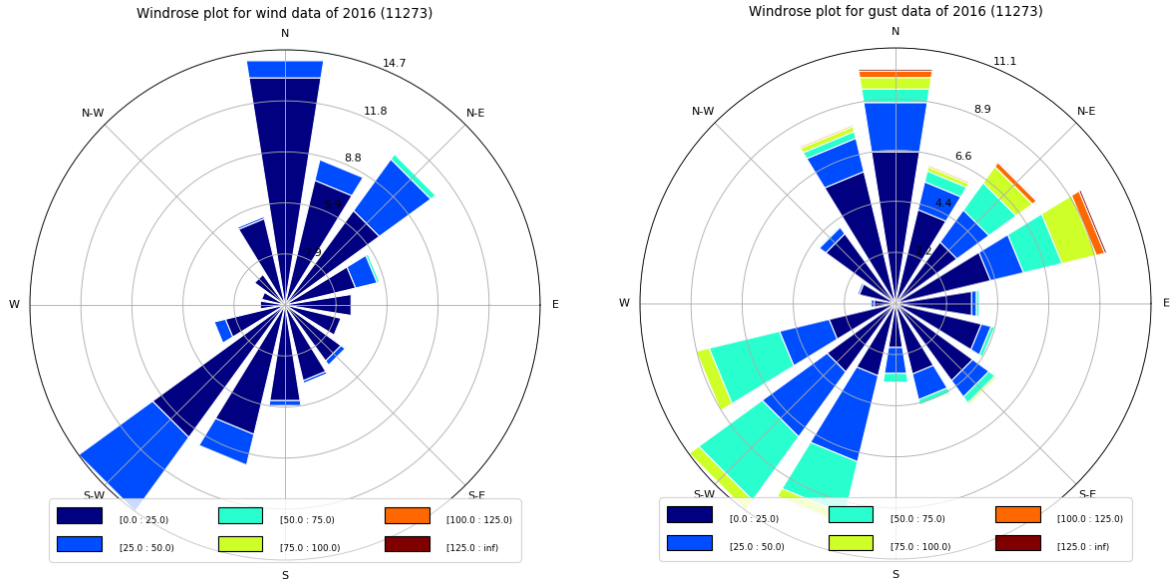
Figure 3.3: Monthly average (purple) and maximum (blue) for wind gusts, 2016 (station Wien Hohe Warte 11035).



(a) Wind speed

(b) Wind gusts

Figure 3.4: Wind rose plot: Wind speed and wind gusts separated into wind directions and bins for velocity (2016) for station Wien Hohe Warte (11035).



(a) Wind speed

(b) Wind gusts

Figure 3.5: Wind rose plot: Wind speed and wind gusts separated into wind directions and bins for velocity (2016) for station Gmünd (11273).

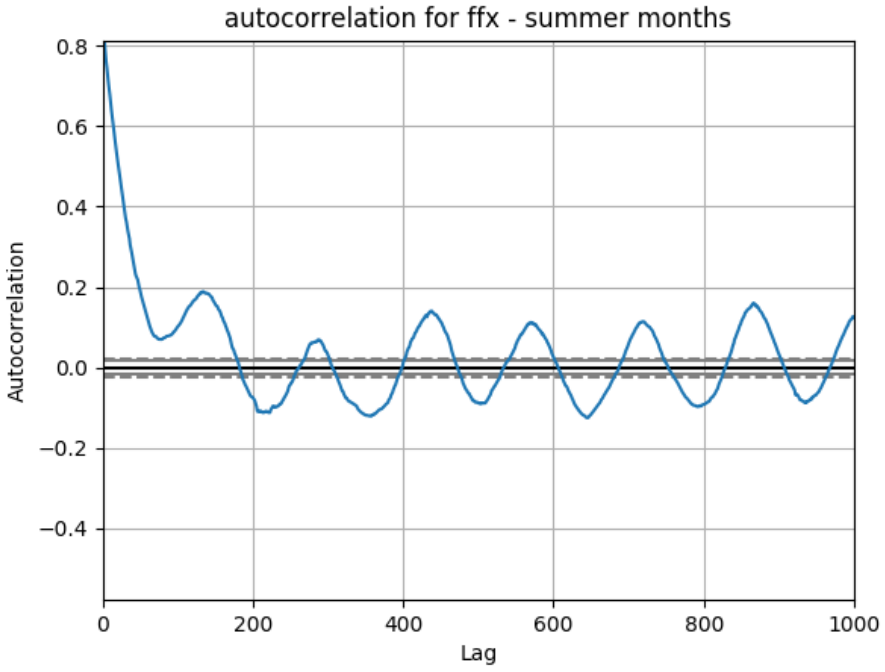


Figure 3.6: Autocorrelation function for wind gusts - summer months (one lag = 10-minute interval).

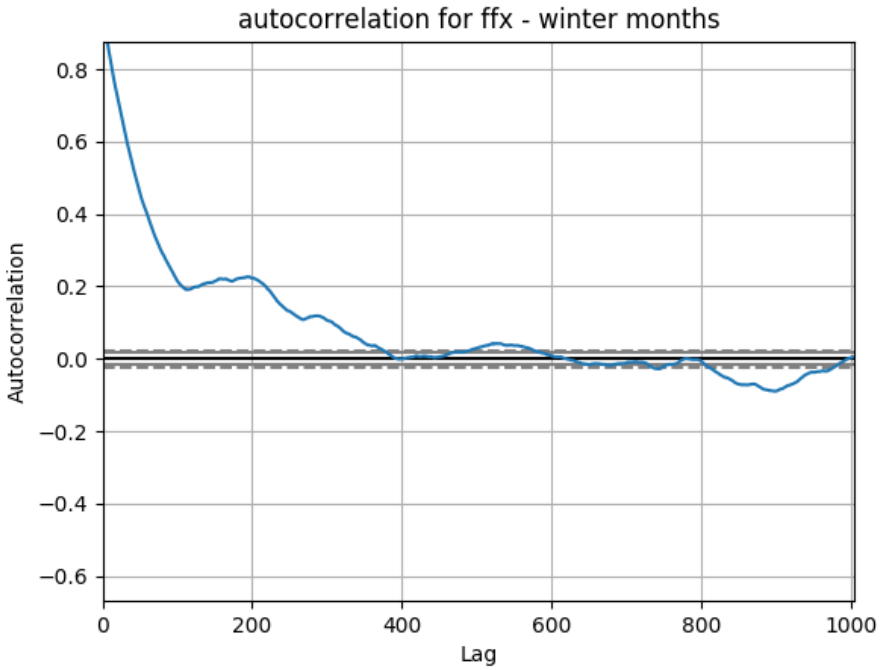


Figure 3.7: Autocorrelation function for wind gusts - winter months (one lag = 10-minute interval).

the next. In contrast to this, the autocorrelation of the winter months shows no clear diurnal pattern.

This leads to the assumption that different models for winter and summer months might make better forecasts since the patterns are so different. To see if this statement holds, the data was split up into different seasons and different models were build to see if separate models make more accurate predictions. This idea has been addressed before, see Dowell and Pinson (2016), Wang et al. (2018), and Friederichs et al. (2009).

Models were build on the dataset split into four seasons: winter (December to February), spring (March to May), summer (June to August), and fall (September to November) and split into two seasons: warm (April to September) and cold (October to March), and for the full dataset.

3.2 Preprocessing of Data

Before we start with training of the SVMs, preprocessing of the data is necessary. Preprocessing includes reformatting data, replacing missing values, and creating new features.

Missing data SVR cannot handle missing data, therefore missing values need to be replaced. In the provided data, missing values are labeled with -999. This can be replaced with NaN (not a number) for easier further evaluation. Generally, the amount of missing data is not significant, as can be seen in Table 3.3. Most stations have less than 0.01% of missing values for every feature. Usually, the amount of missing values is similar in magnitude for every feature. If they vary in the order of one magnitude, the value is taken from the wind and gust feature. Stations where the difference in magnitude is greater than one magnitude are station 11383 where `pred` and `rf` have 5% of missing values and station 11036 where `dd` has greater amount of missing values and `ddx` is actually inexistent. Since these features with relatively

magnitude of missing values	stations
1%	11343, 11320
0.1%	11198, 11384, 11273, 11012
0.01%	11180, 11170, 11290, 11346, 11101
0.001%	11380, 11358, 11344, 11246, 11216, 11108, 11070, 11036, 11025, 11007, 11126, 11035, 11383

Table 3.3: Magnitude of amount of missing values for each station.

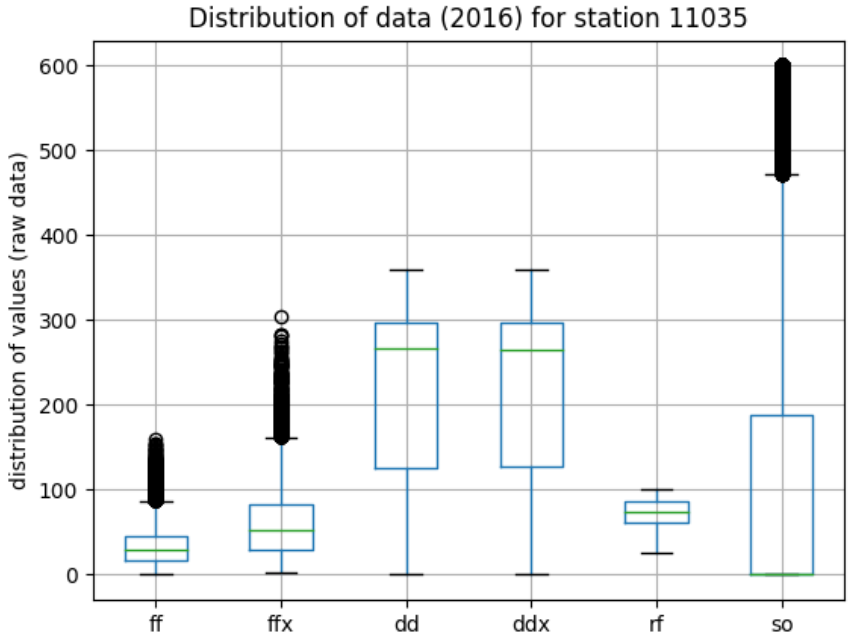


Figure 3.8: Distribution of features of station Wien Hohe Warte (11035). Green line is the median value and the box extends to the Q1 and Q3 quartile (25%-75%) with whiskers for maximum 1.5 IQR and outliers depicted as circles outside the whiskers.

many missing values were not actually used for the models, we can focus on the features with few missing values. For these with few missing data points, we can approximate missing values by interpolation, since there exists an intrinsic relationship between the data points (Zhang et al. 2014).

Normalization Another important part of preprocessing is scaling of the data to avoid features having a larger numeric spread where features with greater ranges might dominate over features with smaller ranges (Hsu, Chang, Lin, et al. 2003). Scaling the data will also avoid numerical difficulties during calculation. For SVM feature scaling can also reduce the computation time.

For the SVR model the RBF kernel assumes that all features behave more or less similar to normally distributed data: centered around zero and with variance in the same order. Figure 3.8 depicts the range of the raw values of some of the features. The data has the same units as described in Table 3.1. Wind speed and gust speed have a majority of data points with lower values as well as a number of outliers. The wind and gust direction have values from 0-360 as they have polar values. Relative humidity `rf` is given in percent and has, thus, values

from 0-100. The distribution of the sunshine duration differs from other features as its median is centered at zero as during the night obviously no sunshine is measured. Additionally, there are many data points which have (almost) full sunshine duration. Full means 600 seconds which is the maximum for 10 minutes. For sunshine duration most values are either zero or between 500 and 600 seconds. As we can see, the data is not normally distributed and a transformation it is generally not feasible.

Instead, we scale the data such that it is as close to normally distributed as possible. In this case the StandardScaler of scikit-learn was used (Pedregosa et al. 2011). Compared to the MinMaxScaler and the RobustScaler it was found to give a slighty better performance. The StandardScaler removes the mean of features and scales the data points to unit variance. In practice only the input features are scaled, the target data does not need to be scaled, therefore no postprocessing of the predictions is necessary.

Time lags The choice of data that is fed into the model is a very important aspect of building the prediction model. More data means longer computation times to build the model and too much additional data can behave like noise and further disturb the prediction (Kramer and Gieseke 2011). For time series, previous values can be of importance too.

Therefore for the predictions not just the current values of the features were used but also time series lags of different features. The effect of previous values were tested for the following features: wind speed, wind gusts, and sunshine duration. The notation for these lagged features is `-i` with `i` denoting the amount of lags (one lag is one 10 minute interval previous). `ffx-6` therefore indicates a lagged gust observation that occured 60 minutes before the current observation. By using grid search and feature evaluation the most relevant lags of these features were selected.

3.2.1 Feature Engineering

Creating new features or merging multiple features is called feature engineering. We can thus build new features from the given data to see if they improve the predictions.

For instance, to identify patterns in the gust behavior, the gust factor `gf` is calculated. It is defined as:

$$\texttt{gf} = \begin{cases} \frac{v_{gust}}{v_{mean}} = \texttt{ffx}/\texttt{ff} & \text{for } v_{mean} > 0 \\ 1 & \text{otherwise} \end{cases} \tag{3.1}$$

The gust factor is a measure especially suitable for the seasonal and diurnal differences. In Figure 3.9 the average gust factor for every month of one year is displayed. It is clear that there are definite seasonal differences. Compared to Figures 3.2 and 3.3, more information on patterns in the data is given. In Figure 3.10 the average for certain times of the day for each month are displayed. This shows an even wider range of differences. During the colder season the diurnal differences are not as strong as during the warmer months where there are clear differences between the day and night hours.

Since there seems to be a connection between sunshine and the intensity of the gusts, another feature `so_sum` is calculated for the accumulated sunshine duration of the previous 60 minutes:

$$\texttt{so\_sum} = \sum_{i=1}^{n=6} \texttt{so-}i \tag{3.2}$$

`so-`i represents the lagged variable with lag i. Other engineered features evaluated are: value `yr` for the day in the annual cycle to represent the yearly variation:

$$\texttt{yr} = cos\left(\frac{\text{dayofyear}}{365} * 360\right), \tag{3.3}$$

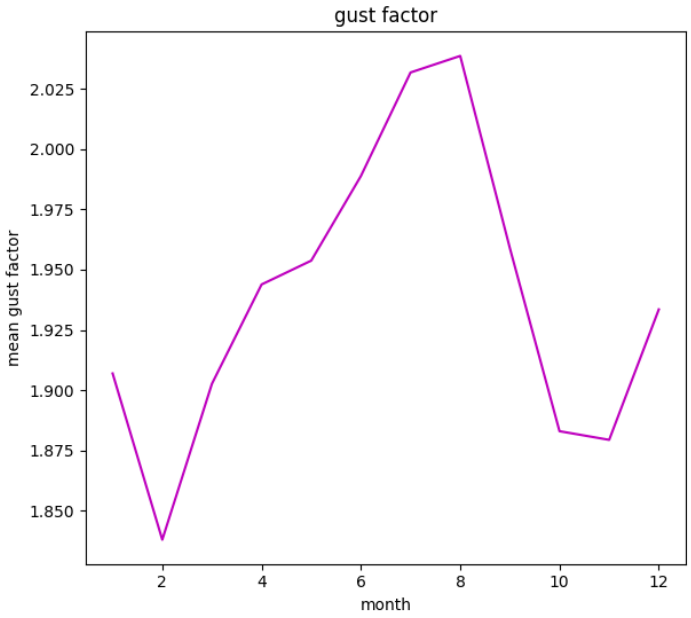


Figure 3.9: Mean gust factor per month (station 11035).

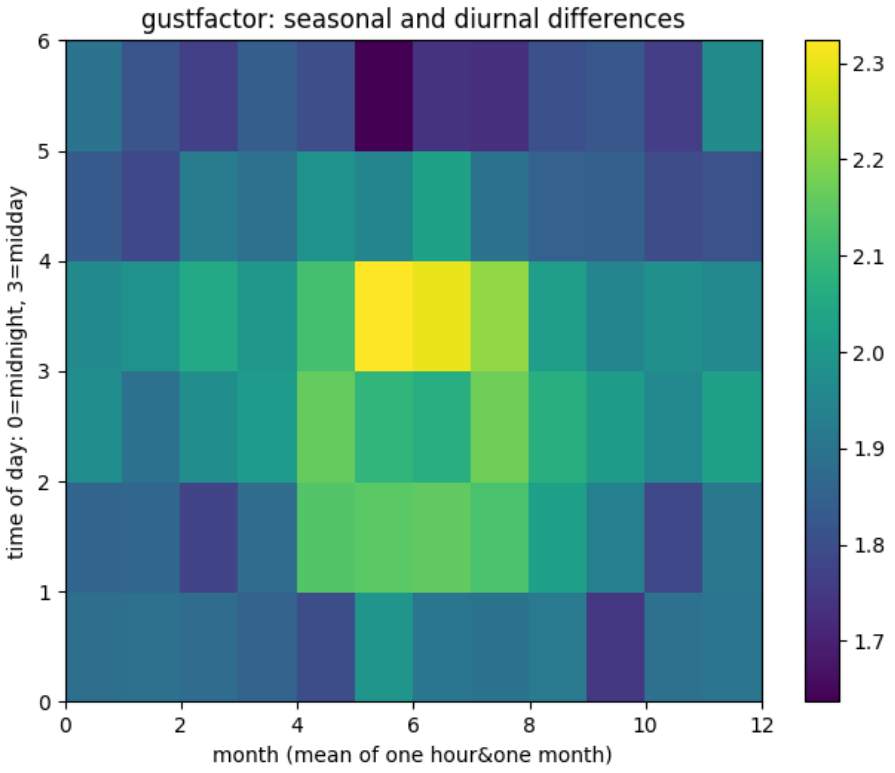


Figure 3.10: Gust factor depending on time of day and month (station 11035).

the u and v vectors for wind speed

$$\mathtt{u} = \mathtt{ff} * sin(\mathtt{dd}) \quad \text{and} \quad \mathtt{v} = \mathtt{ff} * cos(\mathtt{dd}), \tag{3.4}$$

and gusts respectively:

$$\mathtt{ux} = \mathtt{ffx} * sin(\mathtt{ddx}) \quad \text{and} \quad \mathtt{vx} = \mathtt{ffx} * cos(\mathtt{ddx}). \tag{3.5}$$

Chapter 4

Forecasting Methodology

4.1 Software Details and Environment

The implementation of the forecasting method is done in Python (version 2.7) using the scikit-learn machine learning library. As the prediction values are not categorical but numerical the implemented methods are regression methods.

The SVR implementation of scikit-learn is based on LIBSVM, a library for SVM and SVR written in C/C++ (Chang and Lin 2011). The following libraries were used:

sybase_connect	access to ZAMG database
pandas	for data manipulation and analysis
sklearn	scikit-learn: scaler for preprocessing, time series cross-validation, SVR model, MultiOutputRegressor, Linear Regression model, validation scores
numpy	fundamentals for scientific computing
matplotlib	for plotting and figures
joblib	to save models and scalers

Computations were performed on two different x86-64 Linux machines. For data analysis and most of the preliminary tests an eight core machine was used. The main feature selection, grid search, and building of the final models was carried out on a cluster machine with 72 cores.

For the parameter grid search multiple cores were used. For the final model building multiple prediction steps were carried out simultaneously to parallelize predictions. Two different environments for Python packages were used depending on the machine: a nix environment[1] on the eight core machine and an anaconda environment[2] on the 72 core machine. Overall the implementation is dependent on the IT infrastructure of ZAMG and constrained to Python.

4.2 Training Data: Cross-Validation for Time Series

For the computational tests, the training data is split up using 4-fold cross-validation (CV). Since the data consists of time series, it should not be randomized but we want to preserve its time-dependent character. Therefore, the time series cross-validator TimeSeriesSplit from

[1] `https://nixos.org/nix/manual/` accessed 2018/12/10

[2] `https://www.anaconda.com/download/` accessed 2018/12/10

sklearn is used. For n splits the TimeSeriesSplit method splits the data into n+1 consecutive folds. For the n-th cross-validation, it returns the first n folds as training set and the (n+1)-th fold as validation set (here n = 4):

1. Split training data into 5 consecutive folds
2. Train on fold 1, validate on fold 2
3. Train on fold 1+2, validate on fold 3
4. Train on fold 1+2+3, validate on fold 4
5. Train on fold 1+2+3+4, validate on fold 5
6. Calculate average of scores of steps 2-5 to get the CV score

With this method, successive training sets are also supersets of previous training sets. For large amounts of data, the training sets can therefore be quite large which adds to the computational complexity of the model training. To prevent this, it is possible to define a maximum training sample size ensuring that the model effectively only uses a specified amount of data for each training set and not all first n folds.

Different portions of data were used for the training of the SVR model and preliminary tests. For the final models data from the years 2014 to 2016 was used to determine the best hyperparameters and features and to build the models. For the final tests (see Chapter 5), the models were tested on data from January and July 2018.

4.2.1 Validation Scores

Different scores were used for validation and for the final testing. For the feature selection and grid search mostly the coefficient of determination R^2 and the mean squared error (MSE) were used. For final testing the mean absolute error (MAE) and the median absolute error (medAE) were used. Different scores assess different qualities of the model and it is generally good practice to use different scores for training and for testing.

The coefficient R^2 is the square of the correlation coefficient R which tells us how strong the linear relationship between two variables is. It is a measure of how well future samples are likely to be predicted by the model. R^2 over n samples of predictions $\hat{y}$ and observations y is defined as

$$R^2(y, \hat{y}) = 1 - \frac{\sum_{i=0}^{n-1}(y_i - \hat{y_i})^2}{\sum_{i=0}^{n-1}(y_i - \bar{y})^2} \tag{4.1}$$

with the true mean $\bar{y} = \frac{1}{n}\sum_{i=0}^{n-1} y_i$. The perfect score for R^2 is 1.0 and a score of zero means the model is constant (disregards the input features). The score can also be negative as the model can be worse than constant, too. R^2 is not dependent on the scale of the input and it can, therefore, be used to compare models build on different data.

The MSE is defined as

$$\text{MSE}(y, \hat{y}) = \frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y_i})^2 \tag{4.2}$$

It measures the average of the squares of the errors which means a smaller value is desired. Because of the squaring of each term, larger errors are weighted more heavily than smaller ones. Likewise, the root mean squared error (RMSE) is defined as

$$\text{RMSE}(y, \hat{y}) = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y_i})^2} \tag{4.3}$$

It is also sensitive to outliers and smaller values are preferred to larger ones.

For the final tests two different scores were used to not have a biased assessment. The MAE is defined as

$$\text{MAE}(y, \hat{y}) = \frac{1}{n} \sum_{i=0}^{n-1} |y_i - \hat{y}_i| \,. \tag{4.4}$$

which is simply the average distance between prediction and true value. Its ideal value is zero, though this is not realistic in practice. We want to reduce the distance, so a small value is more desirable than a greater one.

In contrast to the MAE which is sensitive to outliers, we also look at the medAE, a measure more robust to outliers. It is defined as

$$\text{medAE}(y, \hat{y}) = \text{median}\left(|y_1 - \hat{y}_1|, \ldots, |y_n - \hat{y}_n|\right) \tag{4.5}$$

The median absolute error is the median of the absolute distances between predictions and true values.

4.3 SVR Configuring: Parameter Tuning via Computational Tests

The different parameters that determine the outcome of an SVR model have varying impact on the model. Parameters that are addressed here are the kernel, the training sample size, and the hyperparameters C, γ, and ϵ. For parameter tuning various computational tests were run to determine the significance of the different parameters and to find the best parameterization for the different models.

4.3.1 Kernel Function for SVR

For SVR various kernel functions can be used, the most common being a linear function, polynomial function, and the radial basis function (RBF) (see Section 2.1.2). For time series prediction it has been found that the RBF function is either superior to other kernels (Ahmed, Khalid, and Akram 2017; Botha and Walt 2017) or that there is no indication that any function is superior if optimal parameters are chosen for each (Mercer and Dyer 2014). Hsu, Chang, Lin, et al. (2003) suggests that the RBF kernel is superior as it is more flexible than the linear kernel but has fewer hyperparameters and fewer numerical difficulties than the polynomial kernel.

To confirm best performance some preliminary tests are done comparing linear and polynomial kernels to the RBF kernel. For the polynomial kernel different degrees (degree $= 1$, 2, and 3) are tested with degree $= 1$ giving the best performance. The linear and polynomial kernel (degree $= 1$) give very similar performance which is not as good but more stable than the RBF kernel. Overall, the RBF kernel has best performance in these preliminary tests. Consequentially, only the RBF kernel is used for further training of the models.

4.3.2 Training Sample Size

The training sample size has more impact on the training than the chosen kernel. Specifically for SVR, this parameter directly influences the computation time for each grid point of the grid search.

For first tests no limit for the training sample size was set. It was then gradually reduced to see the trade-off between runtime and performance score.

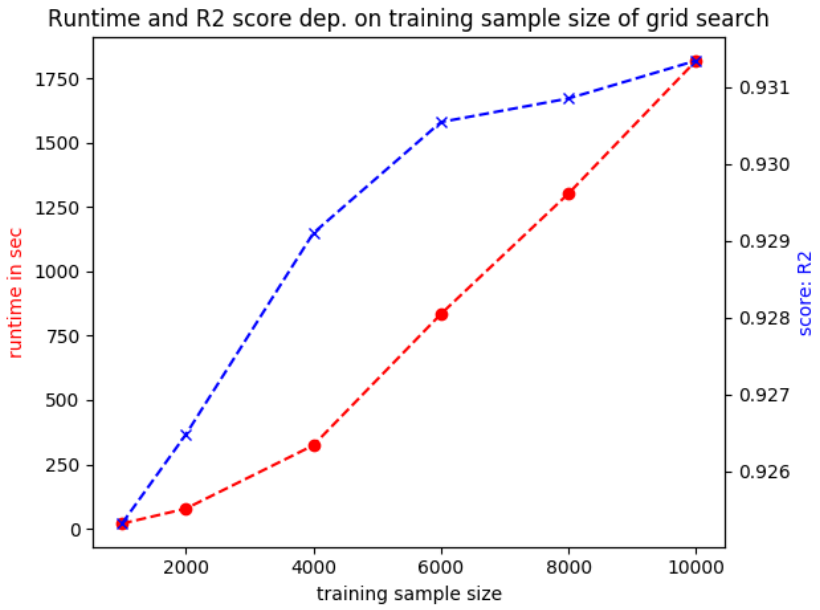


Figure 4.1: Maximum training sample size versus runtime and R^2 score of grid search.

In Figure 4.1 we can see the runtime of an exemplary grid search (red) depending on the chosen maximum training sample size. It is evident that the computation time rises with the training sample size. We can also see the training score (blue) being dependent on the training sample size. With increased training sample size, the model also becomes more accurate and the performance of the training improves. The performance does not increase as rapidly as the computation time, though.

The training sample size was limited to keep runtime reasonable while keeping the training score in mind. The chosen training sample size for the step-wise feature selection and parameter optimization was set to 8000 data points which is about 25 days of data.

For the building of the final models all training data is used, therefore no maximum training sample size is needed.

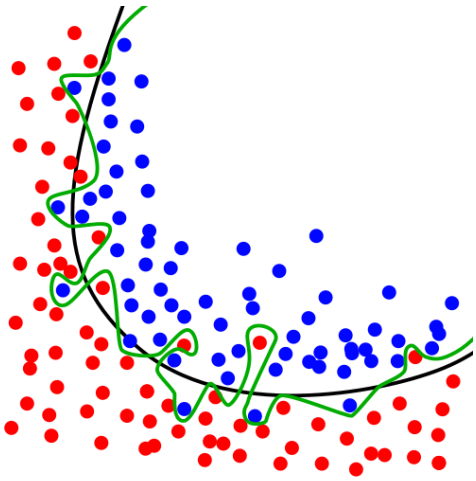
4.3.3 Hyperparameters of SVR

The hyperparameters for SVR are the error term C, the kernel coefficient γ, and ϵ specifying the epsilon-tube (see Chapter 2). These parameters need to be tuned to find the model with the best scoring results and to avoid overfitting.

As introduced in Chapter 2, parameter C determines the balance between model complexity and the tolerance for errors larger than ϵ. If C is set too high, it tries to classify all training examples correctly and is in danger of overfitting. It also prevents overfitting by controlling the penalty on observations that lie outside the epsilon margin. In Figure 4.2 an example for overfitting on a two-class SVM is shown. The green line overfits the training data as it is too dependent on the data and will probably have a higher error on new data compared to the black line.

The kernel coefficient γ determines the influence of one training data point. Low values of γ mean that the selected support vectors have a greater radius of influence and for high values support vectors have a lower radius of influence.

The parameter ϵ describes the width of the epsilon margin which is the tolerance within which regression errors are not penalized. It is generally more stable than C and γ.



source: https://commons.wikimedia.org/wiki/File:Overfitting.svg accessed 2018/12/10

Figure 4.2: Overfitting example for two-class SVM: black is regularized model, green is overfitted.

These hyperparameters are not learned or optimized within the SVR model but need to be specified at initialization. Since the model has high sensitivity to the parameter values it is of importance that a wide range of values is tested.

There are different methods to search over the parameter space for best performance. One of these methods is a parameter grid search which makes one model for every available parameter combination and compares all. The sklearn method GridSearchCV performs an exhaustive grid search over the parameter space while also observing a specific cross-validation method. Another method to search through the parameter space is a genetic algorithm. It uses a fitness function to determine if one solution is superior to another and attempts to improve performance from one generation to the next.

As there is already a native grid search method available in the sklearn package, it is used to tune the hyperparameters for SVR.

First computational tests are done using parameter recommendations from literature (Hsu, Chang, Lin, et al. 2003; Yao et al. 2014; Botha and Walt 2017). Figure 4.3 shows performance for different values of C and γ with ϵ set to 0.1. For high C the computation time explodes. Here, the computation time for $C = 100000$ is up to 500 % longer than for other grid values. The midpoint for the color bar is set to $R^2 = 0.85$ to be able to distinguish the higher values better. The best performance values are between 0.920 and 0.923, all shown in a bright white. It is clear that there is not just one best parameter combination but an area of good performance.

For first computational tests, parameters were first tuned on a coarse grid and then based on those performance scores again on a finer grid. If best scores were on the boundary of the grid, the grid was extended in this direction for the next search. As this can get quite time intensive (going over performance scores to get the correct new boundaries, setting an appropriate step size for the grid, etc), once a method for feature selection was selected, the testing was done on just one grid as there is mostly an area of best performance.

For the different hyperparameters the following ranges of values were tried:

```
C = {0.01 to 1000 000}
gamma = {0.00001 to 100}
epsilon = {0.01 to 100}
```

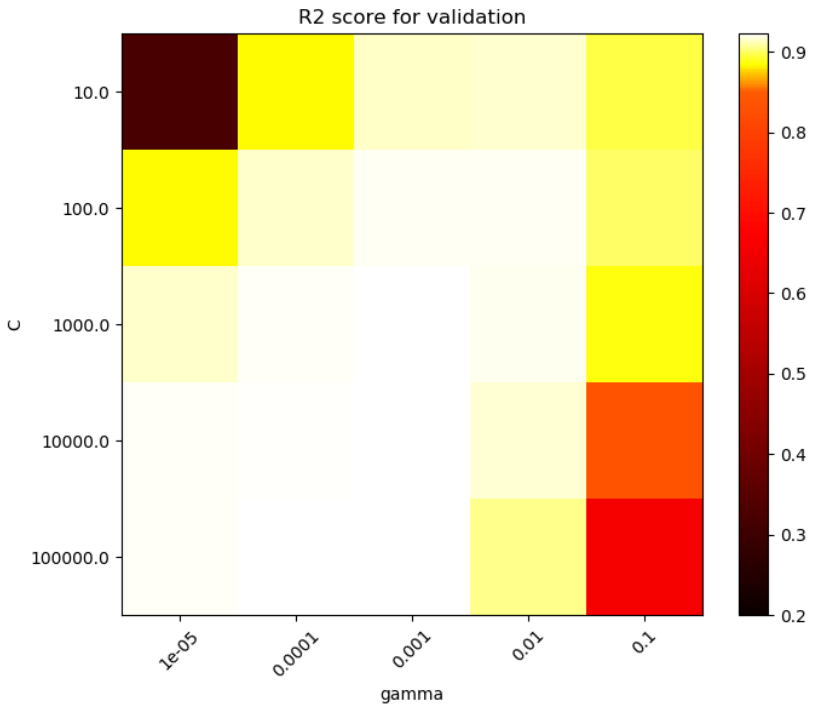


Figure 4.3: R^2 CV scores for exhaustive grid search ($\epsilon = 0.1$).

The grid was cut off on certain edges for different reasons: for high values for C and γ the computation time increased significantly, high values of γ also exhibited worse performance, as did high values of ϵ and and low values of C.

For the final grid search described in Section 4.4.2 the chosen grid is:

```
C = [100, 1000, 10000]
gamma = [0.0001, 0.001, 0.01]
epsilon = [0.01, 0.1, 1]
```

Generally, the best performing hyperparameter combinations are limited to these grid choices, therefore this grid was chosen to limit overall computation time.

4.4 Feature Selection

There are many methods for selecting best features or transformation of features for optimal model performance. Here, we want to use a method with minimal preprocessing as any feature engineering needs to be repeated on new input data for every prediction. It is also important that the time dependency of the input variables is conserved.

As we want to preserve intrinsic knowledge and time dependence, we start with manual testing of the available features. As any machine learning model can only be as good as its input data, it is also of importance to select features which actually add value to the model.

4.4.1 Preliminary Feature Elimination

As listed in Table 3.1, there are a number of features available in 10-minute frequency. Related work generally does not use all features available, but finds best results for different selections. For wind prediction Kong et al. (2015) find best results for only three variables: wind speed, temperature, and air pressure, and Zhou, Shi, and Li (2011) use only wind speed vectors

(with history). Botha and Walt (2017) investigate various combinations of features for wind prediction and find the best combination with wind speed at 60 m, air temperature, and barometric pressure. Mercer and Dyer (2014) use five variables for gust prediction: `u` and `v` wind components, relative humidity, geopotential height, and temperature.

Preliminary tests were carried out to filter nonrelevant variables from Table 3.1. Features that do not have as much impact on wind and gust occurrence as others were tested to see if they can be discarded. Features that were discarded from feature space because they did not improve performance on first models are: temperature `tl`, dew point `tp`, and precipitation `rr`. As we have two pressure variables which carry the same information, only the reduced pressure was kept and pressure `p` was discarded.

Also not included in further feature selection are the engineered features from Section 3.2.1 (besides `so_sum`). In preliminary tests these were not found to bring significant improvement over including the original features they were constructed from. As we want to keep preprocessing to a minimum, it was chosen to not include most of those engineered features into later feature selection.

As the sunshine duration has most significant impact on performance (apart from lagged values of `ff` and `ffx`), feature `so_sum` was kept for further testing. `so_sum` also gives a seasonal indication to the model.

As we can optimize both hyperparameters and feature selection, it was chosen to put more focus on the feature selection. For hyperparameters there is usually not exactly one optimal combination but an area of good performance. For example, the difference between setting $C = 1000$ and $C = 10000$ is neglectable if features are optimized. It seems sufficient to use a coarse grid for parameter search and to optimize the feature selection.

It is not feasible to do all feature selection manually since different features seem relevant for different stations. Consequentially, we need a better system to select best features for each station. As running both parameter grid search and feature selection simultaneously can get computationally expensive, we try to keep a simple model for feature selection to limit overall runtime.

4.4.2 Step-Wise Feature Selection Algorithm

A step-wise feature selection was chosen to select best features from a given feature space. The features included in the feature space are: `ffx`, `ff`, `stdmin`, `so`, `ddx`, `dd`, `rf`, `pred`, and `so_sum`. Preliminary tests were done to check the amount of lags relevant for wind and gust measurements, and sunshine duration.

Wind prediction models are based on the same feature space but with different order, as the order is important for the following step-wise feature selection. As sklearn only supports one output for the SVR model, separate models were build for wind speed and wind gust prediction. The order of the features was selected according to preliminary tests and according to the significance of features.

The step-wise feature selection algorithm is a pipeline that selects features based on the best grid search CV score. A basic pseudocode of the algorithm, including the exact feature space for gust models, is detailed in Algorithm 1.

The R^2 score that determines the CV performance is the mean of 4 CV folds. As the R^2 scores of these folds can vary quite a bit, the standard deviation of the score can be greater than the difference that the score is improved from one selected feature to the next. A version of the algorithm was also tried where this was taken into account and only features that improved performance above a set threshold $sign$ were added to the feature_set.

In Figures 4.4 and 4.5 the best features selected by the step-wise feature selection algorithm are shown for gust and wind speed prediction step one. For gust prediction a history of gust and

Algorithm 1: Step-Wise Feature Selection Algorithm.

Input: training period (startdate, enddate), station number
Output: best CV score, best feature set, and its best hyperparameters

```
data = sybase_connect(startdate, enddate, station number);
interpolate missing values;
target = 'ffx+1';
create lagged features and prediction column;
feature_space = ['ffx', 'ffx-1', 'ffx-2', 'ffx-3', 'ffx-4', 'ffx-5', 'ffx-6', 'ff', 'ff-1', 'ff-2',
  'ff-3', 'ff-4', 'ff-5', 'stdmin', 'so_sum', 'ddx', 'dd', 'rf', 'pred', 'so', 'so-1', 'so-2',
  'so-3', 'so-4'];
set parameter_grid;
best_score = 0;
best_features = [];
best_param = [];
feature_set = feature_space[0];
for i in feature_space do
    X = StandardScaler.fit_transform(data[feature_set]);
    y = data[target];
    cv = TimeSeriesSplit(n_splits = 4);
    gs = GridSearchCV(SVR, parameter_grid, cv).fit(X, y);
    if gs(best_score) better than best_score then
        best_score = gs(best_score);
        best_features = feature_set;
        best_param = gs(best_param);
    else
        discard feature_set[-1];
    end
    feature_set += feature_space[i+1];
end
save best_score, best_features, best_param;
```

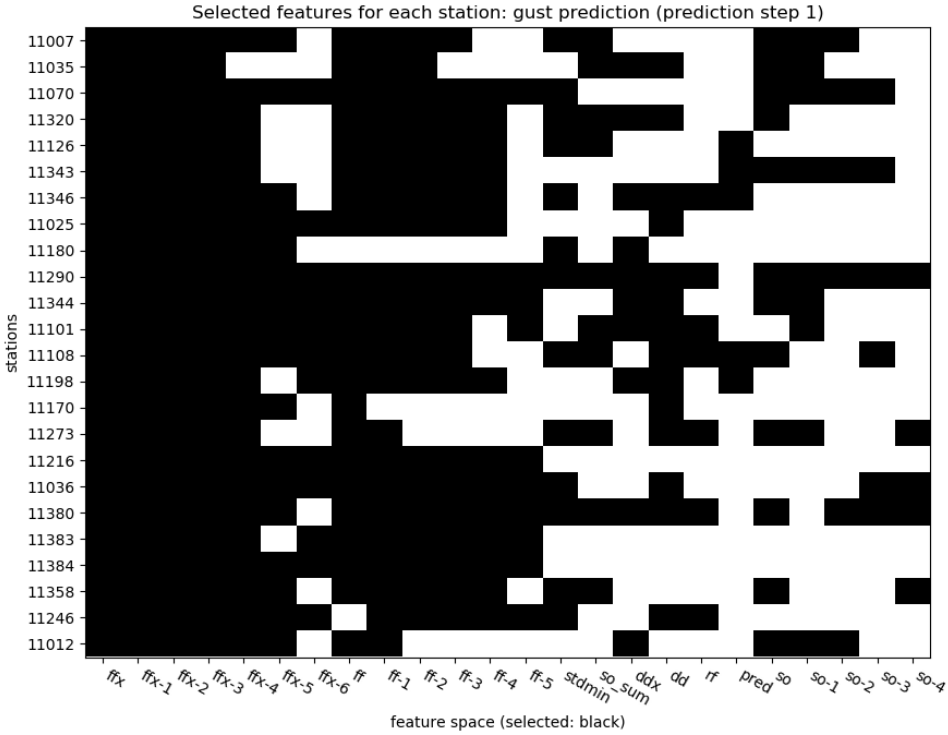


Figure 4.4: Selected features for each station for prediction step one: gust prediction.

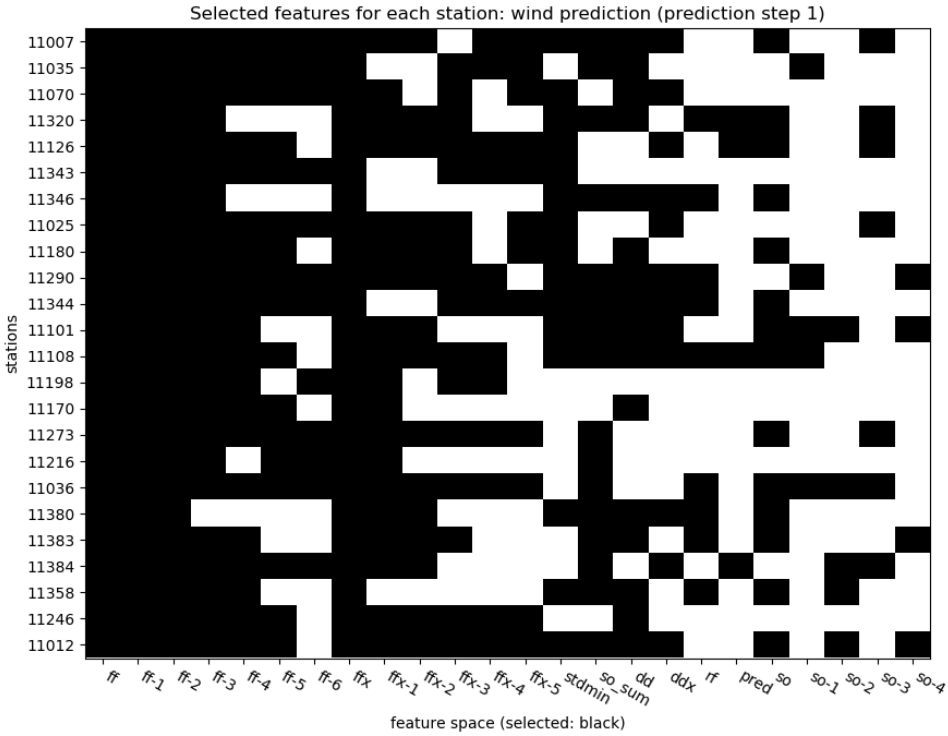


Figure 4.5: Selected features for each station for prediction step one: wind prediction.

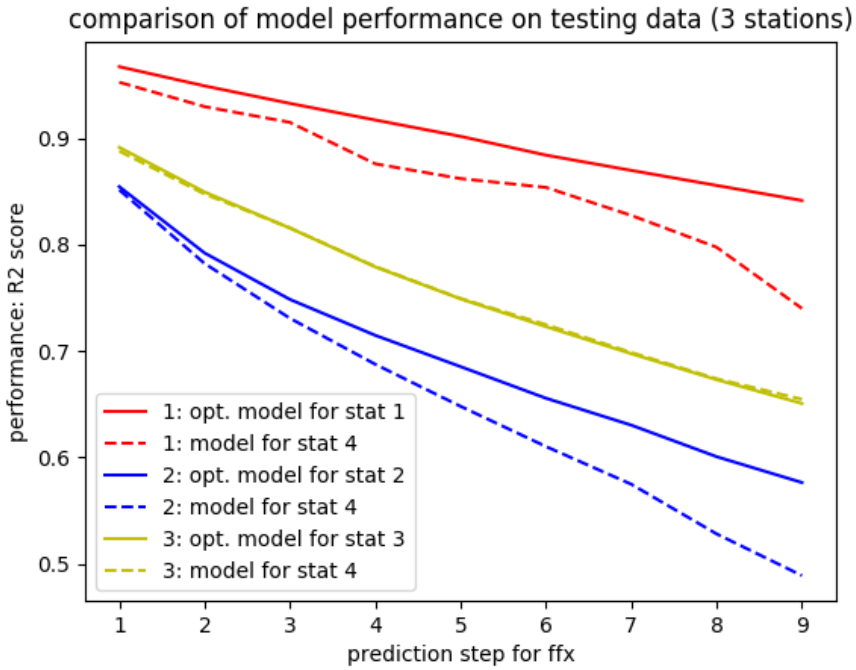


Figure 4.6: Models optimized for individual stations (1-3) versus models optimized for a fourth station (gust prediction).

wind speed values seem significant for most stations. Other features vary greatly for different stations. For wind speed prediction history of gust speed are not as significant as wind speed history. Other features which are valuable for a majority of the stations are: `stdmin`, `so_sum`, `dd`.

To determine that this individual feature selection for each station is necessary, Figure 4.6 compares the performance of predictions of models optimized for three stations on testing data (January 2017) to the performance of predictions based on a model optimized for a fourth station. It can be seen clearly that the R^2 score is better for station one and two continuously for all nine prediction steps. The closer prediction scores for station three can be explained by a more similar feature selection compared to the other stations. Overall, it is evident that an optimized feature selection for each station improves the predictions.

4.5 Additional Methods Investigated

4.5.1 Splitting Models by Seasons

As mentioned in Section 3.1 another method that was investigated separates data and models into different seasons to improve performance. Table 4.1 gives an overview over the training performance of different models for gust prediction. The optimal features for each season were determined through computational tests from a more limited feature space than in Section 4.4.2. Looking at the features it can be observed that sunshine duration has an impact on most seasonal splits but is not chosen for winter and the cold season.

We can also estimate an average R^2 score for all four models for data split into four models: 0.918 and for the data split into two models: 0.919. Comparing these scores to the model score build on the full year dataset (0.9131), the difference is actually less than the standard deviation of these scores.

season	data used	R2 score	RMSE	optimal features
spring	03-05/2013-2016	0.9162	10.80	ffx(bis-3),ff,stdmin,dd,so
summer	06-08/2013-2016	0.8992	9.86	ffx(bis-3),ff,stdmin,dd,so
fall	09-11/2013-2016	0.9258	9.79	ffx(bis-3),ff(bis-2),so
winter	12-02/2013-2016	0.9308	10.91	ffx(bis-3),ff,ddx
warm	04-09/2015-2016	0.9088	10.14	ffx(bis-3),ff,stdmin,so
cold	10-03/2015-2016	0.9290	10.71	ffx(bis-3),ff(bis-2)
year	2015	0.9131	10.47	ffx(bis-3),ff(bis-1),so

Table 4.1: Comparison of R2/RMSE for different seasons: each season with individual grid search for best parameters and features to use (one-step gust prediction).

parameter	description	categories	explanation
windsect	circulation type	0-8	1-8: dominant wind sector octants, 0: if more than one sector dominant
cyclon925	cyclonality at 925hPa	A/C	A: anti cyclonic, C: cyclonic
cyclon500	cyclonality at 500hPa	A/C	A: anti cyclonic, C: cyclonic
moisture	moisture index	M/D	M: moist, D: dry

Table 4.2: Parameters and categories of weather classification (WLKC733 Katalog).

Variations of performance can also be attributed to changes in gust patterns over the seasons. Performance for summer months is not as good as for winter months as gust behavior is more random and not as predictable. Consequently, the results of these preliminary tests do not tell us that there is a performance increase for using individual models for each seasons compared to one model for all seasons.

The SVR method is in itself complex enough that it should be able to differentiate between different pattern which also include different patterns for different seasons. After various computational tests, it was decided to focus on building one advanced model for all seasons instead of multiple models for different seasons as splitting data brings unnecessary complexity.

4.5.2 Splitting Models by Weather Regimes

Instead of building separate models for different seasons one can build different models for different weather regimes. In Browell and Gilbert (2017) regimes are defined by the previous day's weather conditions, specifically the two-dimensional horizontal mean wind vector (u,v) for the preceding 24 hours. The number of regimes is reduced using the k-median algorithm with the number of clusters determined by cross-validation. In Browell, Drew, and Philippopoulos (2017b) regimes are classified by atmospheric classification which are clustered by k-means algorithm to three distinct modes. These regimes are used for a vector-autogressive method to predict wind speed with regime-switching depending on the day's regime. A third method in Browell, Drew, and Philippopoulos (2017a) clusters modes based on the sea-level pressure field, the geopotential height at 500 hPa, and wind speed.

To apply a similar method to data from ZAMG, information on the atmospheric classification models used in Austria is needed. The current Austrian weather regime classification (WLKC733 Katalog) (Krennert 2009) consists of four different parameters detailed in Table 4.2.

These weather types are defined once for each day. In total, there are 72 different classes. These would need to be clustered to be able to work with a reasonable number of regimes. As

all of the parameters are categorical, there is no additional value in using clustering to reduce the amount of regimes.

All the examples listed above used to classify atmospheric schemes are based on numerical data which have the option of easily being clustered by using for example k-means. For the standard Austrian weather regime classification with only categorical data, this is not applicable.

Defining our own weather regimes using the two-dimensional horizontal mean wind vector (based on wind speed and wind direction, as done in Browell and Gilbert (2017)) is an option that was not attempted as this does not actually provide more information for the model as these two features are already included in the building of the model.

4.6 Basic Models

A baseline setup for the forecasting method was chosen to determine how good the different SVR models are performing. Since current methods based on NWP models take several hours of calculations, the predictions are rather old for the nowcasting range. Thus, it is not reasonable to compare these forecasts to our methods. As the forecasts are also with a 10-minute frequency they can only be compared to the available NWP models every full hour. A comparison would need additional interpolation or merging of the predictions.

The implemented baseline methods are a persistence model and a more advanced multiple linear regression model.

4.6.1 Persistence

The persistence model is the most basic time series forecasting model: it takes the value of the current observation $f(x_i)$ and uses it as the prediction value $f(x_{i+1})$:

$$f(x_{i+1}) = f(x_i) \tag{4.6}$$

For multi-step ahead prediction, the current observation value is used for each step:

$$f(x_{i+n}) = f(x_i) \quad \forall n \tag{4.7}$$

4.6.2 Multiple Linear Regression

The second baseline method that was implemented is multiple linear regression (MLR). The linear regression model assumes that the prediction variable y and the features $(x_1, ..., x_p)$ exhibit a linear relationship:

$$y = \beta_0 + \beta_1 x_1 + ... + \beta_p x_p + \epsilon = x^T \beta + \epsilon \tag{4.8}$$

If we have multiple predictor variables x_i, the model is called multiple linear regression.

To optimize the MLR model, a step-wise feature selection algorithm similar to Algorithm 1 was implemented. A pseudo code of the feature selection for each station is detailed in Algorithm 2. This is done on the same data as the training of the SVR (see Section 4.2) with the same CV splits. The data preprocessing is equivalent to Section 3.2 as the data is also normalized using the StandardScaler of sklearn.

The chosen features and lags for the feature space were selected by preliminary tests for MLR and SVR. The feature space for gust prediction is detailed in Algorithm 2 (line 5). For wind prediction the feature space is the same but in different order (see Figure 4.8). The CV score used is also the correlation coefficient R^2.

Algorithm 2: Feature Selection for MLR.

```
Input: training period (startdate, enddate), station number
Output: best feature set
data = sybase_connect(startdate, enddate, station number);
interpolate missing values;
target = 'ffx+1';
create lagged features and prediction column;
feature_space = ['ffx', 'ffx-1', 'ffx-2', 'ffx-3', 'ffx-4', 'ffx-5', 'ff', 'ff-1', 'ff-2', 'ff-3',
  'ff-4', 'ff-5', 'stdmin', 'so_sum', 'so', 'dd', 'ddx', 'rf', 'pred'];
best_score = 0;
best_features = [];
feature_set = feature_space[0] ;
for i in feature_space do
    X = StandardScaler.fit_transform(data[feature_set]);
    y = data[target];
    cv = TimeSeriesSplit(n_splits = 4);
    for n_splits in cv do
        set X_train, y_train, X_test, y_test;
        mlr = LinearRegression.fit(X_train, y_train);
        score = mlr.score(X_test, y_test);
    end
    cv_score = average of scores for n_splits;
    if cv_score better than best_score then
        best_features = feature_set;
    else
        discard feature_set[-1];
    end
    feature_set += feature_space[i+1];
end
save best_features;
```

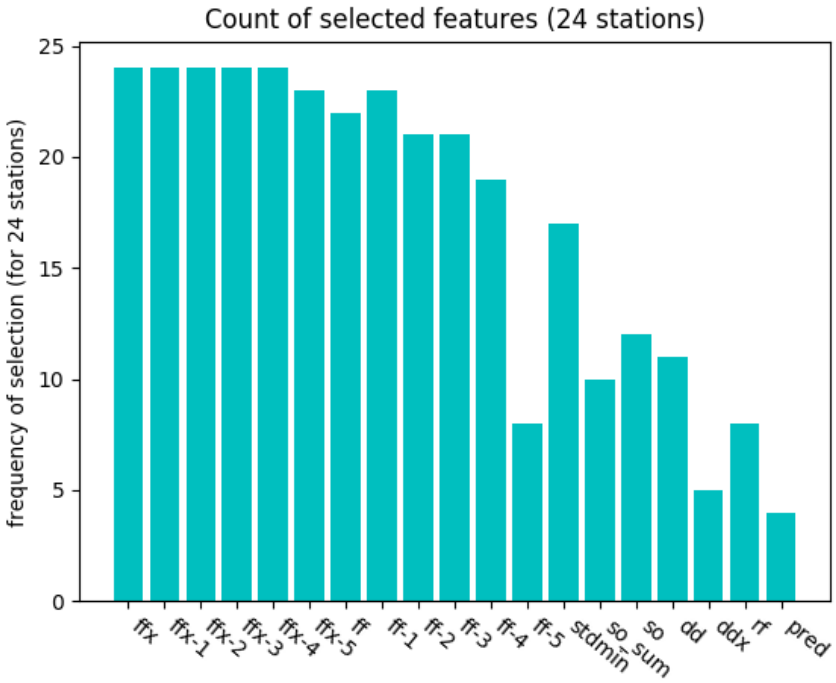


Figure 4.7: Count of features selected by step-wise feature selection for MLR (gust prediction).

The feature selection is done for a one-step MLR, meaning the step-wise feature selection is only done once for the first step and those selected features are then used to train a multi-step model. The count of selected features for all stations is visualized in Figure 4.7 for gust prediction and Figure 4.8 for wind prediction.

Comparing the results it can be seen that the lagged wind speed measurements are more important to gust prediction than the other way around. For gust prediction we can also observe that the time of day (`stdmin`) has a clear impact. Generally, features besides wind and gust measurements vary significantly between different stations which gives clear indication that an individually optimized baseline for each station is of importance.

The optimized MLR model is build by taking the best feature combination from the step-wise feature selection and training the model on the full training data set. This model is saved with the joblib package, as is the scaler, since the new input data for predictions needs to be scaled with the same parameters as the training data.

This optimization is done for both wind speed and gust speed prediction such that there are two models for each station, one for gust prediction and one for wind prediction. The method is the same to build the MLR model, merely the order of the feature space is changed for wind prediction.

4.7 Structure of Multi-Step SVR Model

In building a multi-step model there are some points that need to be considered. Optimizing parameters and features for an one-step ahead prediction does not necessarily mean that we get satisfactory results for a multi-step ahead prediction (Liu and Zio 2017). Most research focuses on single-step methods with not as much research done on tuning parameters for multi-step methods. Since it can get computationally very expensive to tune parameters and run feature selection for each of the 36 prediction steps, we need to find a method to tune the different steps within a reasonable computation time.

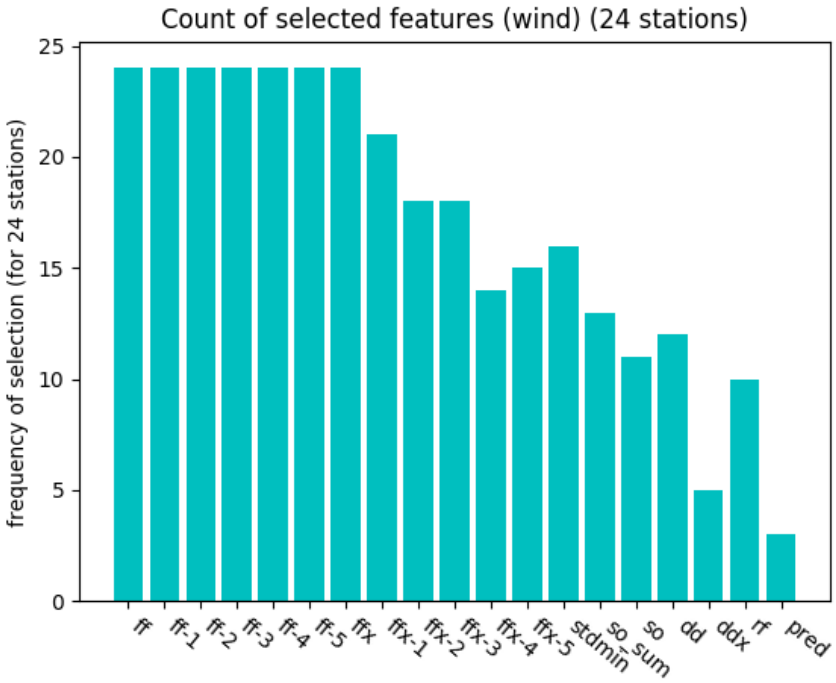


Figure 4.8: Count of features selected by step-wise feature selection for MLR (wind prediction).

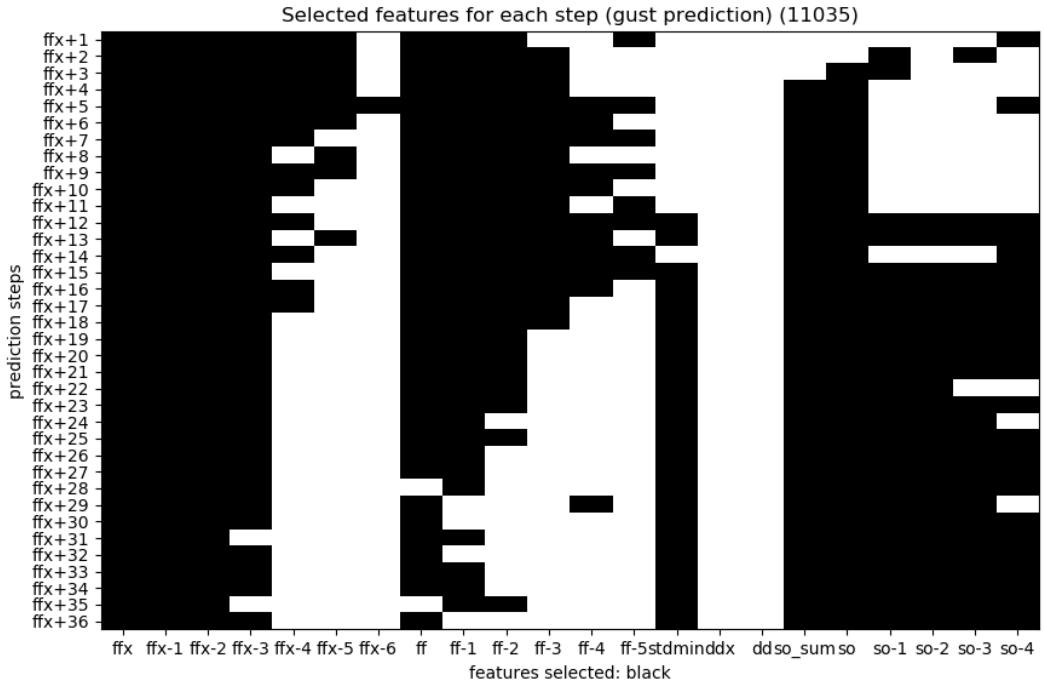


Figure 4.9: Features selected for each prediction step (gust prediction): selected features are shown in black (station Hohe Warte 11035).

In Figure 4.9 the optimal features selected for each of the 36 prediction steps for test station Hohe Warte (11035) for a gust speed forecast are shown. The features are selected using the step-wise feature selection from Section 4.4.2. Here, parameter $sign$ is set to $sign = 0.0001$. It is clear that the optimal features vary depending on the forecasting horizon. In the first steps more lags of `ffx` and `ff` are important compared to later steps. We can also see that the time of day (`stdmin`) seems to make a bigger difference for later hours of predictions. One explanation why there are overall less features chosen for the first two hours could be that here performance is overall better and it is less likely that a new feature will improve the score greater than $sign = 0.0001$.

4.7.1 Multi-Step Strategy

Since the SVR implementation of sklearn does not have native support for multiple outputs, we have several options for building multi-step models. The two most prominent strategies are the iterative and the direct strategy. The iterative strategy (also called recursive) uses the already predicted values to make new predictions. Since it uses approximated values for each new step, we have an accumulation of errors. It is also not possible to predict multiple steps at the same time but predictions need to be done iteratively.

Direct strategy uses a separate model for each prediction step and does not use approximated values. Instead since each prediction is made with a new model there is no statistical dependence between different predictions.

As it is of importance to have fast predictions, we chose to use a direct strategy as this means predictions for different steps can be made at the same time. To speed up building and prediction of the models we use a wrapper called MultiOutputRegressor from sklearn. This wrapper fits one regressor (SVR) for each target step, it is therefore a direct strategy. It uses the same input data and hyperparameters for each regressor. Thus, we are limited to the same feature selection for each step used within each wrapper. The challenge is to optimize the multi-step model as much as possible for the individual steps but keep the number of wrappers used to a low number as optimization for each wrapper takes time.

There do exist some SVR implementations that support multiple outputs natively but as we are constrained by the Python environment at ZAMG, these were not tested.

4.7.2 Architecture of Final Model

For the final model three wrappers (MultiOutputRegressor) with different amounts of steps were used. The architecture of this model is shown in Figure 4.10. Looking back to Figure 4.9, it makes sense to optimize the model with more than one wrapper. As we want to keep the number of wrappers low, the choice was set to three.

For each wrapper a step-wise feature selection algorithm was run selecting best features and hyperparameters for its first step. Then one SVR model was build for each step using the MultiOutputRegressor. The wrappers have overlapping steps which are averaged in the prediction step such that the final output has 36 separate prediction steps in 10 minute intervals.

4.7.3 Additional Model Optimized on Four Wrappers

For comparison, a few test models were build that are optimized on four steps. Instead of running the step-wise feature selection algorithm only for steps 1, 10, and 19, it was additionally run for steps 7 and 13. Then multi-step models were build for optimal features and parameters for step 1 (steps 1-6), 7 (steps 7-12), 13 (steps 13-18), and 19 (steps 19-36). An additional

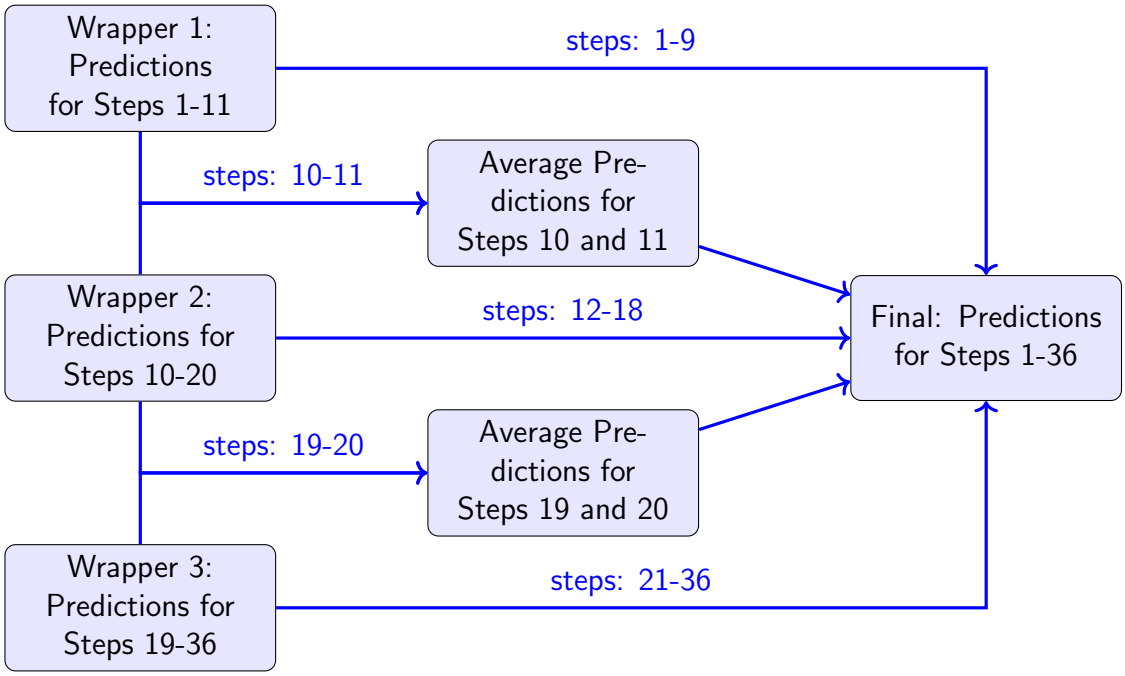


Figure 4.10: Architecture of final prediction model separated into three wrappers.

wrapper in the first half of the forecasting horizon was added instead of the second as the optimal features display greater variety for the first half of prediction steps for test station Hohe Warte (see Figure 4.9). Models were compared for test stations Hohe Warte (11035), Patscherkofel (11126), and Graz-Universität (11290).

In Figures 4.11 and 4.12 these models were compared to the original models optimized on three steps. It is easily visible that a fourth wrapper on the specified steps does not bring improvement and only increases computation time as one additional step-wise feature selection needs to be run.

4.7.4 Additional Manual Adjustments

In Figure 4.13 SVR performance is compared to the implemented baseline models for station Bad Mitterndorf (11358) for the training data (July 2017). It seems that the models built with the second wrapper perform worse than the rest of the models. Figure 4.14 gives an overview over the selected features for each wrapper (black steps). One difference between these wrappers is that for step 1 and 19 features which describe the sunshine duration are selected and for step 10 these are neglected.

This brings up the question if the feature selection chosen by the algorithm can be improved manually. For several stations an in-depth evaluation was carried out to improve performance and feature selection. This was done to try to build an improved SVR (iSVR) by adjusting the feature selection manually.

Stations for which an iSVR model was built are: Bad Mitterndorf (11358), Gmünd (11273), Lunz am See (11170), Güssing (11198), Rauris (11346), Fürstenfeld (11246), Weitra (11025), Rax/Seilbahnbergstation (11180).

For Bad Mitterndorf the original (black) and adjusted features (purple) are shown in Figure 4.14. Additionally to `so_sum` it can be observed that feature `ff` was added for step 10 and 19 and some `so` lags were removed. This was done for some of the stations to restore time dependency of those lagged features.

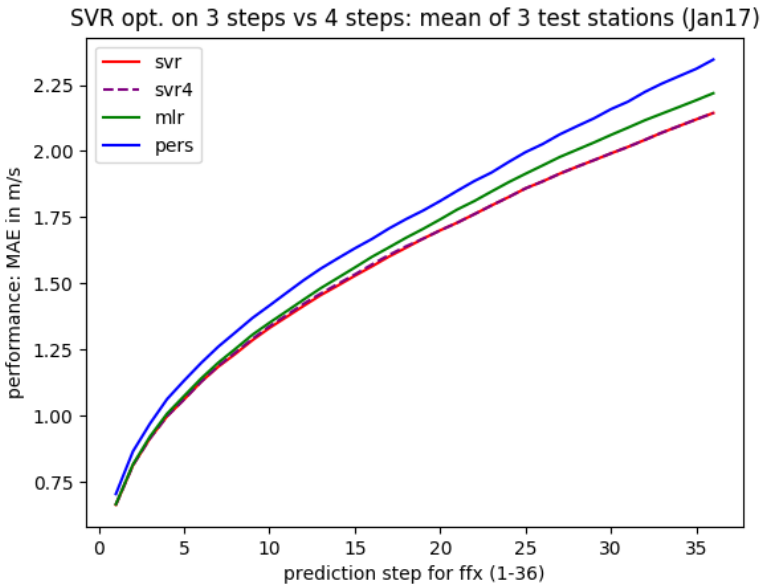


Figure 4.11: Models optimized on three versus four wrappers (gust prediction), data: January 2017, three test stations.

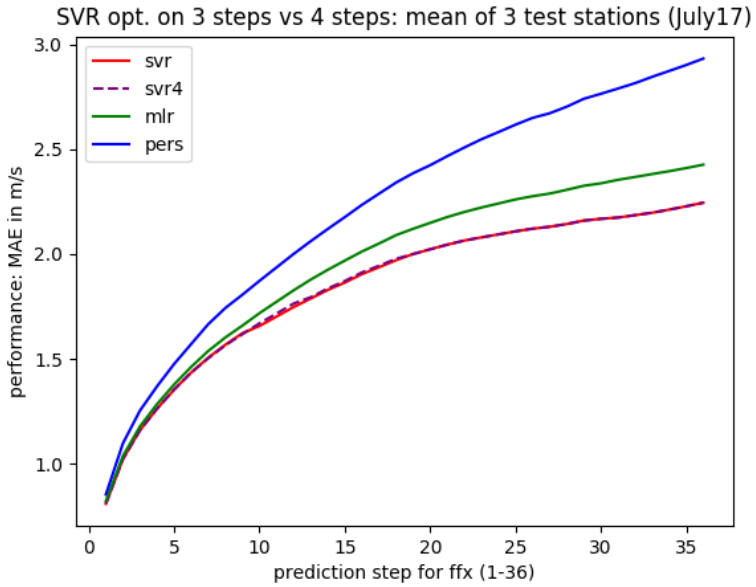


Figure 4.12: Models optimized on three versus four wrappers (gust prediction), data: July 2017, three test stations.

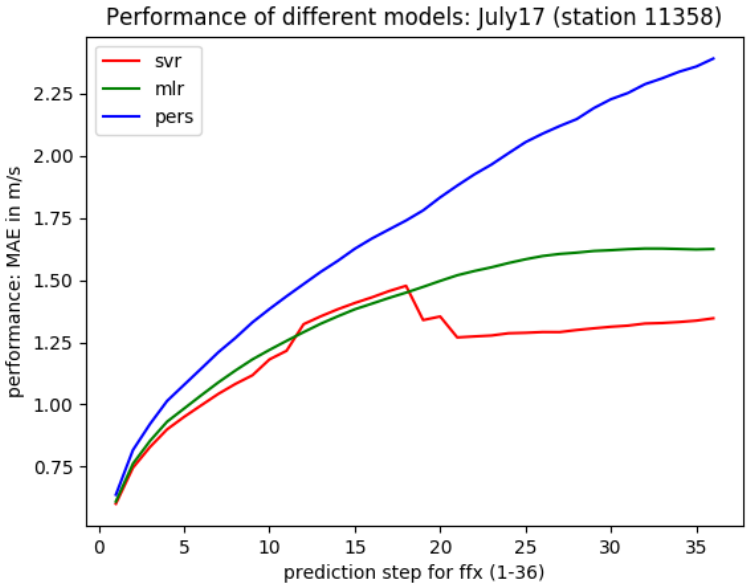


Figure 4.13: Performance of SVR for station Bad Mitterndorf (11358), gust prediction, July 2017.

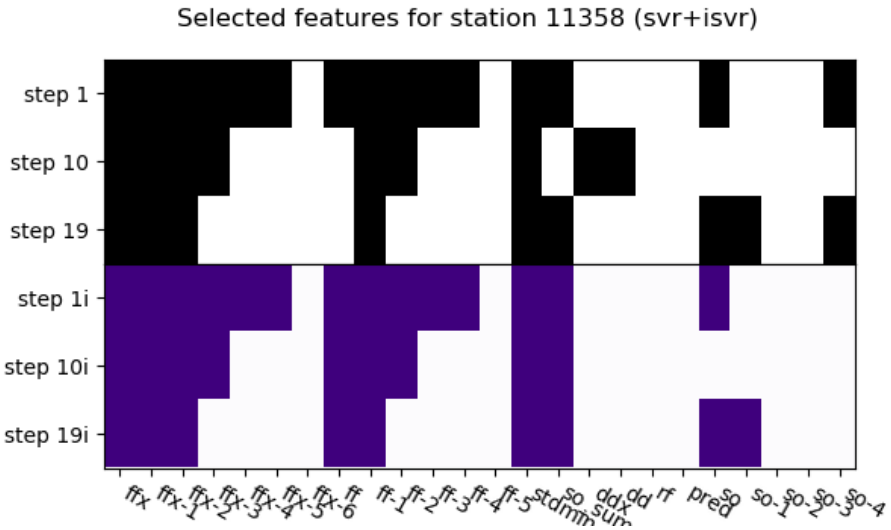


Figure 4.14: Features selected for different steps (gust prediction) station Bad Mitterndorf (11358).

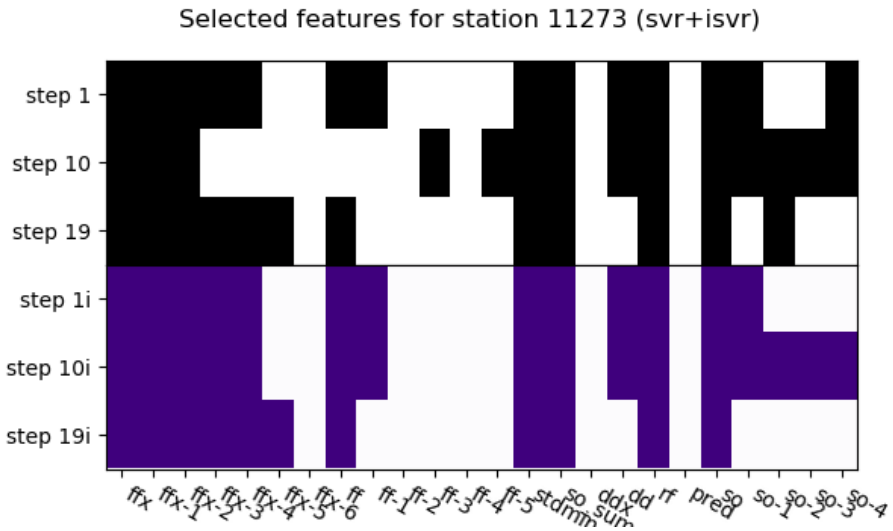


Figure 4.15: Features selected for different steps (gust prediction) station Gmünd (11273).

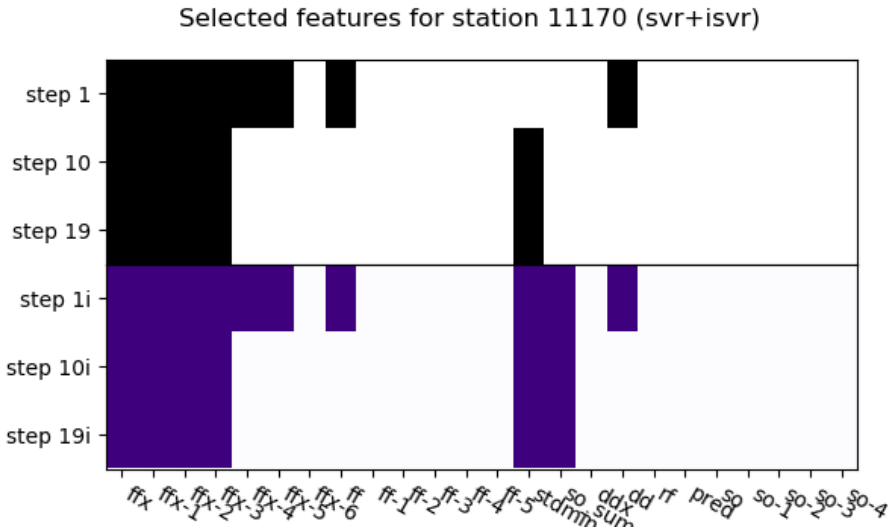


Figure 4.16: Features selected for different steps (gust prediction) station Lunz am See (11170).

For station Gmünd (11273) the original and adjusted features are shown in Figure 4.15. Here, the features selected by the algorithm do not show much consistency for the time dependent values. An iSVR model was build for this station to see if adjustment of these features improves predictions.

Figure 4.16 shows features selected for station Lunz am See (11170). Very few features are selected compared to other stations and none of the models rely on a sunshine duration feature. Here `so_sum` was added to the iSVR model to see if it brings improvement.

4.8 Computational Complexity of the Algorithm

As discussed earlier, the runtime of the algorithm can vary greatly depending on the hyperparameters. It is also of course dependent on the size and composition of the input data set and other parameters (see for example Figure 4.1).

The SVR implementation is at its core a quadratic programming problem. The solver for this quadratic programming problem is a decomposition algorithm which is the most expensive

part of the SVR. It scales between $\mathcal{O}(n_{features} \times n_{samples}^2)$ and $\mathcal{O}(n_{features} \times n_{samples}^3)$ depending on the sparsity of the data set and how many columns of Q are cached (Chang and Lin 2011). As our data set is not sparse but rather dense, the complexity is probably closer to $\mathcal{O}(n_{features} \times n_{samples}^3)$.

Runtimes for one CV fold during grid search range from few seconds to several minutes depending on the choice of hyperparameters. For a couple hundred hyperparameter combinations, this adds up significantly. When building the final models on the whole training data set, computation time increases substantially as $n_{samples}$ is multiplied. On average training of a model for one prediction step takes about half an hour.

Once the models are build, predictions can run quite fast. One prediction for the next six hours takes about 4-6 seconds depending on the station and its model complexity. Delays are most likely to be expected at the database requests for data importing.

4.9 Summary of Workflow

Figure 4.17 summarizes the training and testing phase of the SVR models and the process of predicting new forecasts.

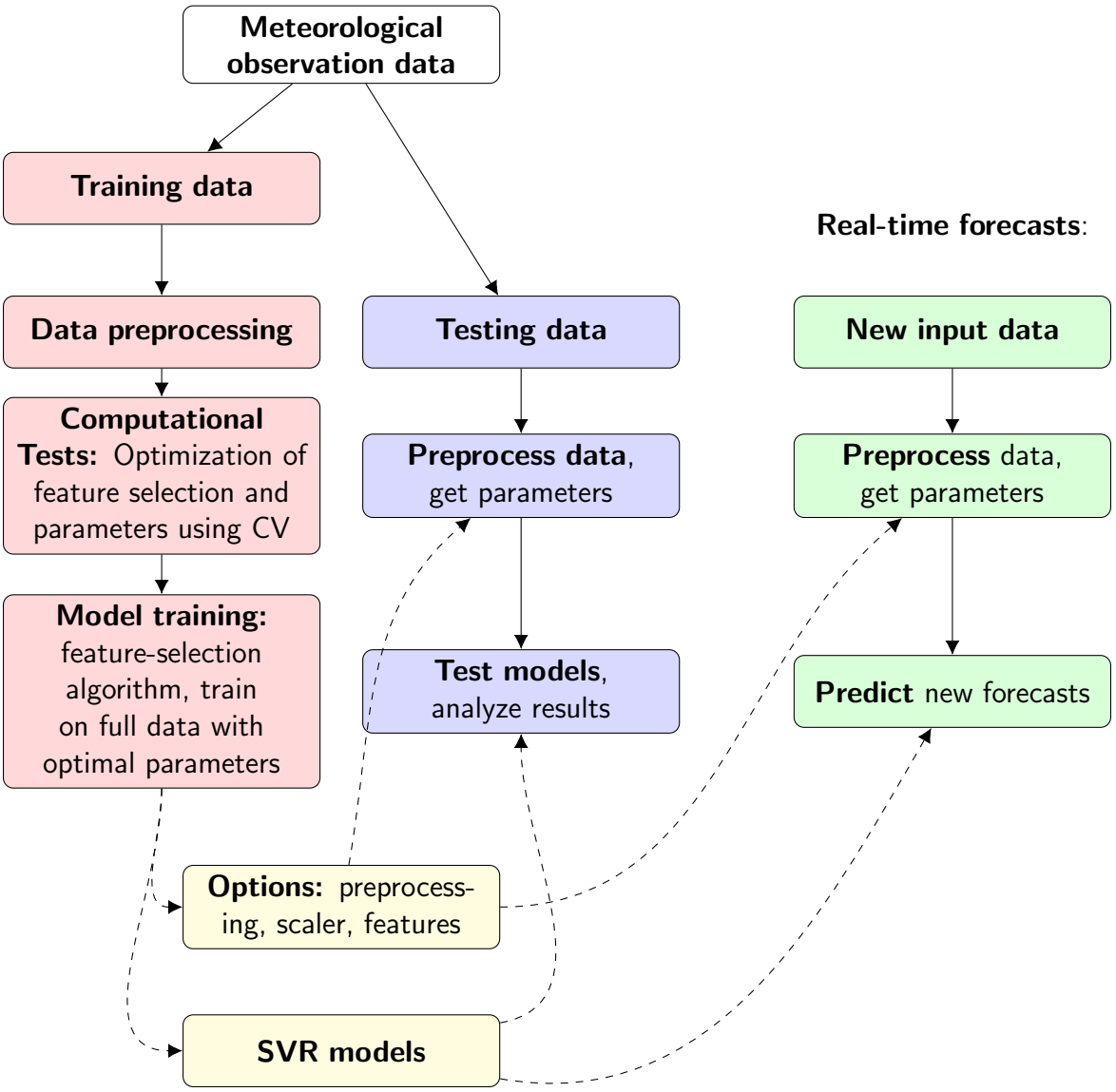


Figure 4.17: Workflow of training and testing phase.

Chapter 5

Results and Evaluation

Two different test months, January 2018 and July 2018, were chosen to evaluate the performance. These months were used to consider different seasons and different gust and wind behavior. Predictions run for each data point are a 36-step ahead prediction with a 10-minute forecast frequency. Thus, six hours in total are forecasted. The MAE of one station is therefore an average over in total 4500 predictions.

We evaluate the average of the predictions for all 24 test stations (see Table 3.2) and the average of the improved models for eight stations (see Section 4.7.4). Further test stations which were evaluated in depth are:

- Wien Hohe Warte (11035), main investigation site at elevation of 198 hm (Wien)
- Bad Mitterndorf (11358), alpine valley station at 814 hm (Steiermark)
- Gmünd (11273), alpine valley station at 738 hm (Kärnten)

These stations were selected as they belong to the subset of eight selected sites (see Section 4.7.4). Furthermore they represent different topological and climatological Austrian regions.

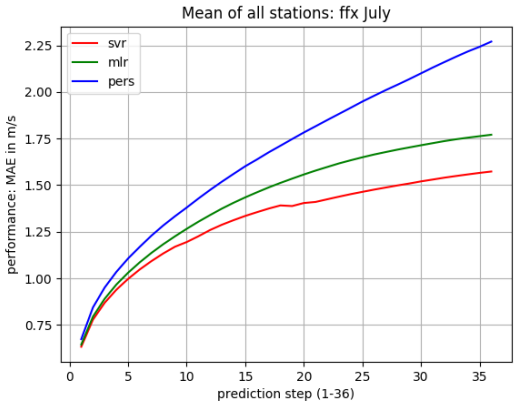
Performance scores used are MAE and medAE (defined in Section 4.2.1). As the MAE scores are average errors of 4500 values, the MAE plot is usually quite smooth. Contrasting this is the medAE score for the persistence model, as the persistence forecast is limited to values of 1/10th m/s. For small medAE scores overall, this means that the medAE plot can have rather large steps. We also investigate the correlation coefficient R^2 which was mainly used for tuning during the training phase. It provides information on the goodness of fit of the models.

5.1 Results for Gust Forecast

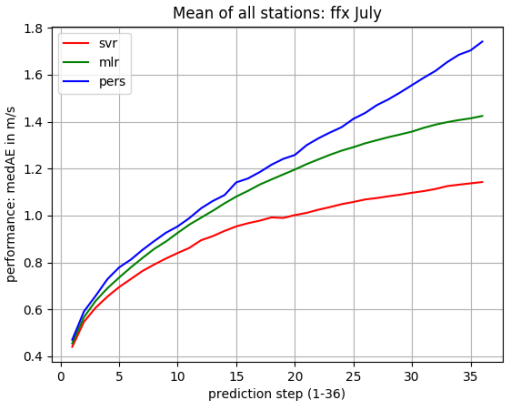
As the focus of this thesis was more on gust forecasts than on wind speed, gust results are described first.

July 2018

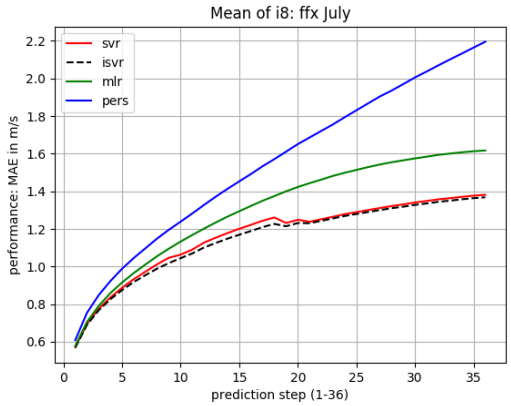
In Figures 5.1a and 5.1b the MAE and medAE for gust forecasts for July 2018 are shown for all stations. Here, and for all following plots the SVR model is shown in red, the MLR model in green, and persistence in blue. All three models perform similarly for the first time step but the SVR model shows better performance for subsequent steps, performing best overall. As expected the persistence performs worst. This pattern is found for almost all stations, except for two mountain stations (11180 Rax/Seilbahnbergstation and 11384 Hirschenkogel) for which performances for SVR and MLR are quite similar.



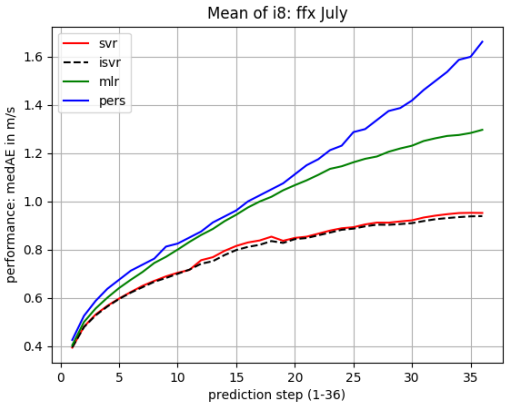
(a) MAE (24 stations)



(b) medAE (24 stations)



(c) MAE (8 stations)



(d) medAE (8 stations)

Figure 5.1: Gust prediction performance, July 2018 (mean of all 24 stations and of eight stations with iSVR).

In Figures 5.1c and 5.1d the eight stations which an iSVR model was build for are compared for July 2018. The iSVR model is shown with a dashed black line. It shows that the iSVR does improve the SVR forecast for these stations, for some steps more than others. The amount of improvement depends on what kind of feature adjustment was made for the iSVR model and how good the SVR model was already performing before tuning was applied. Two of those eight stations are investigated more closely.

In Figure 5.2 performance of the three test stations is shown for July 2018. For station Wien Hohe Warte (11035) (Figures 5.2a and 5.2b) the SVR model performs best. For the first nine steps of the MLR model the performance is similar to SVR but for further-ahead time steps SVR improves in contrast to MLR and persistence.

For station Bad Mitterndorf (11358) (Figures 5.2c and 5.2d) there is a clear improvement for performance of iSVR over SVR. Referring back to Figure 4.14, we can see that here the change in features for wrapper two (time steps 10-18) and three (time steps 19-36) clearly improved predictions, especially for wrapper two. It seems that the feature selection algorithm actually chose wrong features (like wind and gust direction) for wrapper two. The SVR models also perform better than MLR and persistence.

For station Gmünd (11273) (Figures 5.2e and 5.2f) no improvement as significant as at Bad Mitterndorf is seen for iSVR. It seems that just cleaning up features as done here (see Figure 4.15) does not improve the performance as adding the sunshine duration does. Here, SVR does also perform better than the baseline models.

In Figure 5.3 the correlation coefficient R^2 is shown for July 2018 for all 24 stations. As this measure evaluates the goodness of fit, a higher value indicates a better fit. The SVR model performs best while the persistence forecast has negative R^2 values for the second half of the prediction steps.

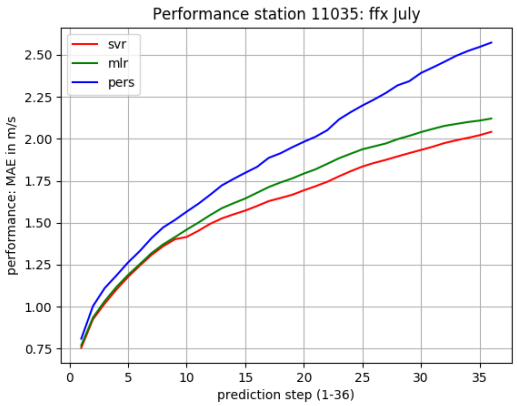
January 2018

In Figures 5.4a and 5.4b the MAE and medAE for gust forecasts for January 2018 is shown for all stations. As for July 2018, the SVR model performs better than the basic models. For the first ten steps, the MAE of the SVR model is similar to the MAE of the MLR model. It does not improve forecasts as much as for the July forecast. We can infer that the SVR model is better at finding patterns in sunshine duration to link to gust behavior than the MLR model and is, thus, better at predicting more turbulent behavior. Comparing this to Figures 5.1a and 5.1b, it can be seen that performance is worse for January than July which points to unusual gust behavior for January. As seen in Figure 3.10, usually summer months exhibit more gust variations than winter months as there is more sunshine, thus, more turbulence and exchange.

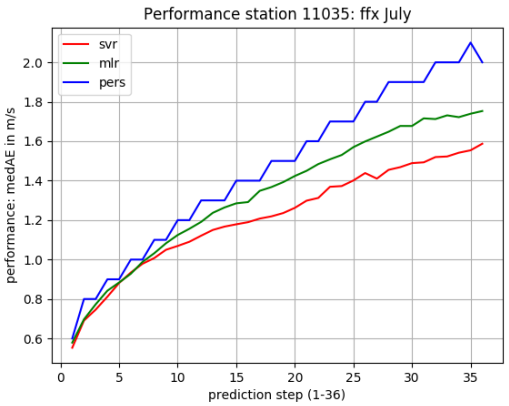
For this test month, one third of the stations show a performance of MLR that is very similar to SVR performance. For the other stations, the SVR performance is better than the baseline models. This also indicates that for this test month the SVR models did not perform as good as for July 2018.

In Figures 5.4c and 5.4d the gust performance for January 2018 is shown for the eight station subset. For January, iSVR apparently does not improve performance. As most of the changes to iSVR were adding the sunshine duration to the features, it is obvious that this increases the performance for summer more than for winter. For the first forecast time steps the performance of SVR is the same as for MLR but for further predictions SVR improves the forecast more than MLR.

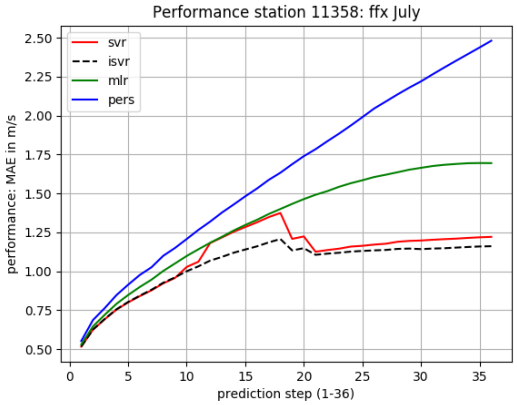
In Figure 5.5 the three test stations are shown. Performance of models for station Wien Hohe Warte (11035) is shown in Figures 5.5a and 5.5b. Here, similar to July, the SVR model is only slightly better than the MLR model for the first nine time steps. Overall, SVR performs better than the baseline models.



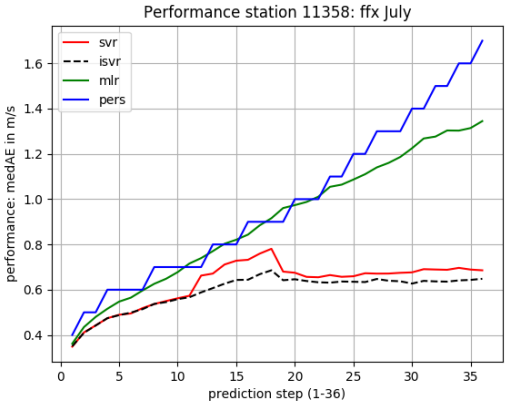
(a) MAE 11035



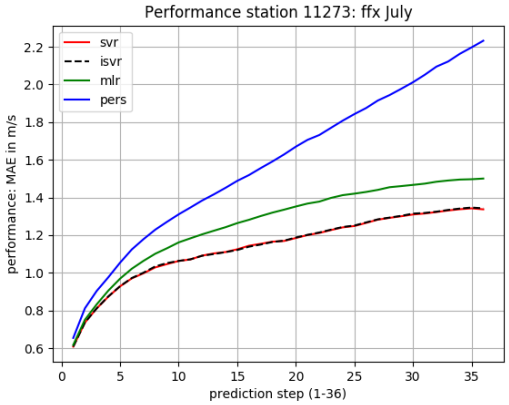
(b) medAE 11035



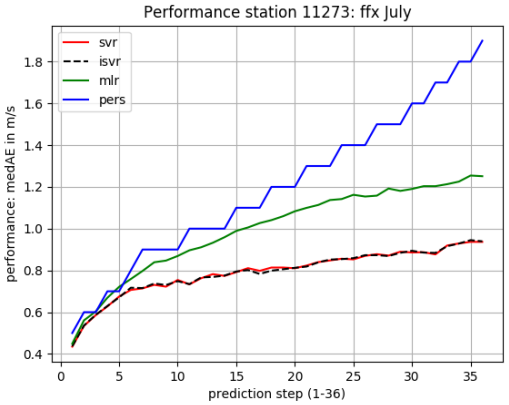
(c) MAE 11358



(d) medAE 11358



(e) MAE 11273



(f) medAE 11273

Figure 5.2: Gust prediction performance July 2018, three stations.

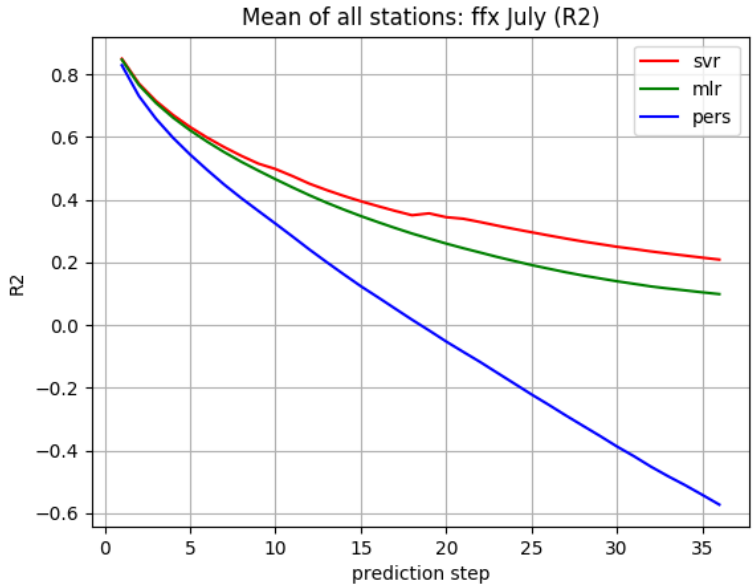
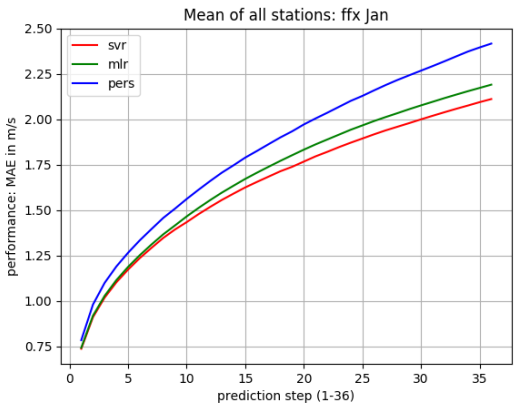
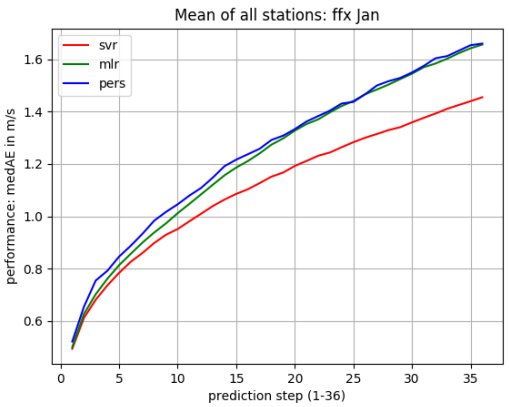


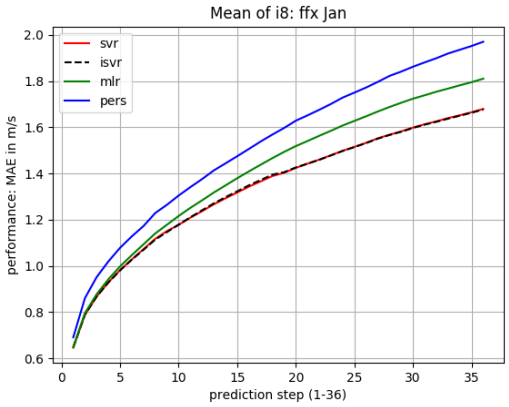
Figure 5.3: Gust prediction: Correlation coefficient R^2 for July 2018 (mean of all stations).



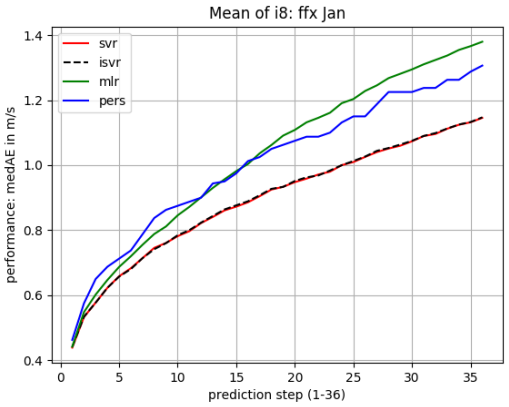
(a) MAE (24 stations)



(b) medAE (24 stations)

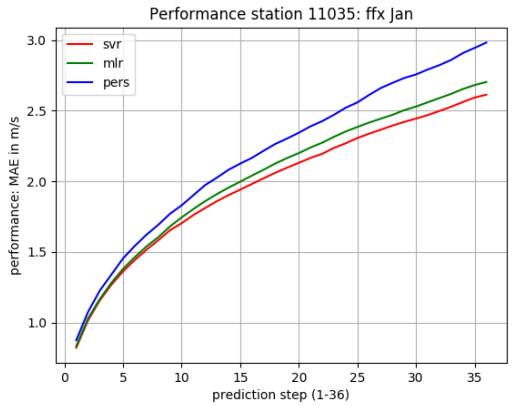


(c) MAE (8 stations)

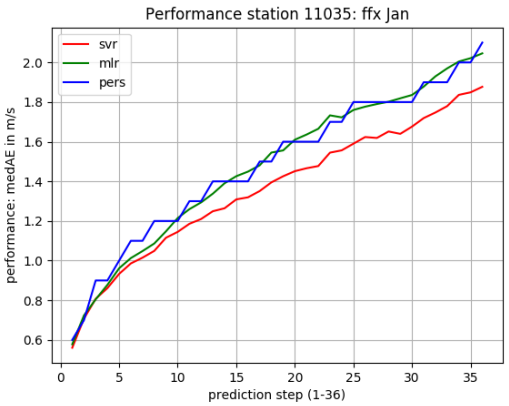


(d) medAE (8 stations)

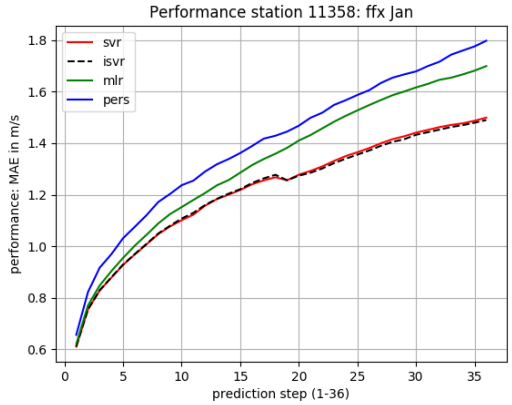
Figure 5.4: Gust prediction performance: January 2018, mean of all 24 stations and eight stations with iSVR.



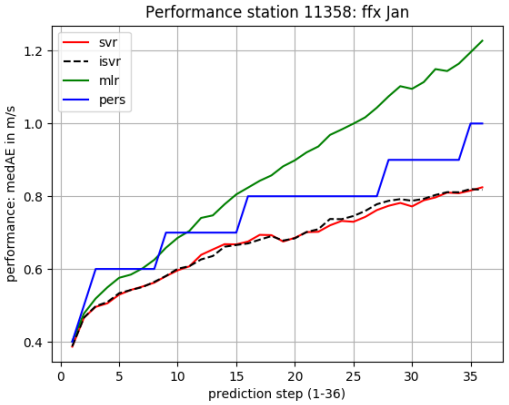
(a) MAE 11035



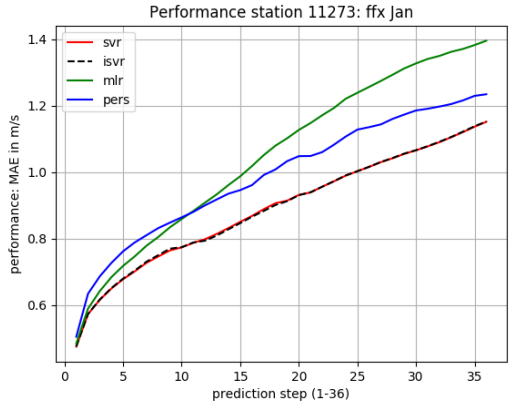
(b) medAE 11035



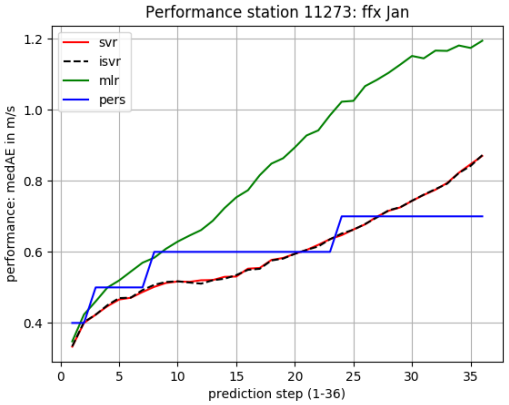
(c) MAE 11358



(d) medAE 11358



(e) MAE 11273



(f) medAE 11273

Figure 5.5: Gust prediction performance: January 2018, three stations.

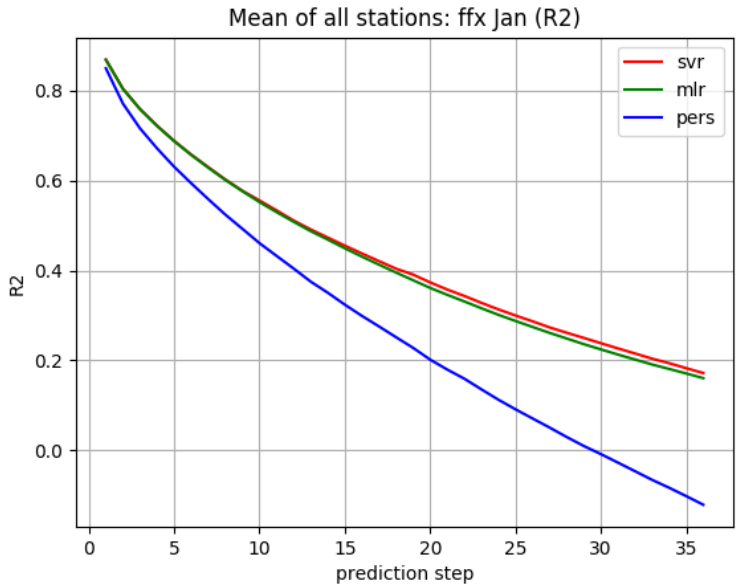


Figure 5.6: Gust prediction: Correlation coefficient R^2 for January 2018 (mean of all stations).

Station Bad Mitterndorf (11358) is shown in Figures 5.5c and 5.5d. Compared to the July test month (Figures 5.2c and 5.2d) the iSVR model does not improve the SVR. Overall, the SVR models perform better than the baseline models.

Performance of station Gmünd (11273) is shown in Figures 5.5e and 5.5f. Here, the iSVR does not bring improvement to the SVR model but the SVR MAE is better than the MAE of the baseline models. The persistence model performs better than MLR for later predictions and the medAE of persistence is even better than SVR for the last ten predictions. As the persistence performance has quite large steps compared to SVR, it is hard to make a direct comparison though.

In Figure 5.6 the correlation coefficient R^2 is shown. The SVR model has best performance, though it is quite close to the MLR model.

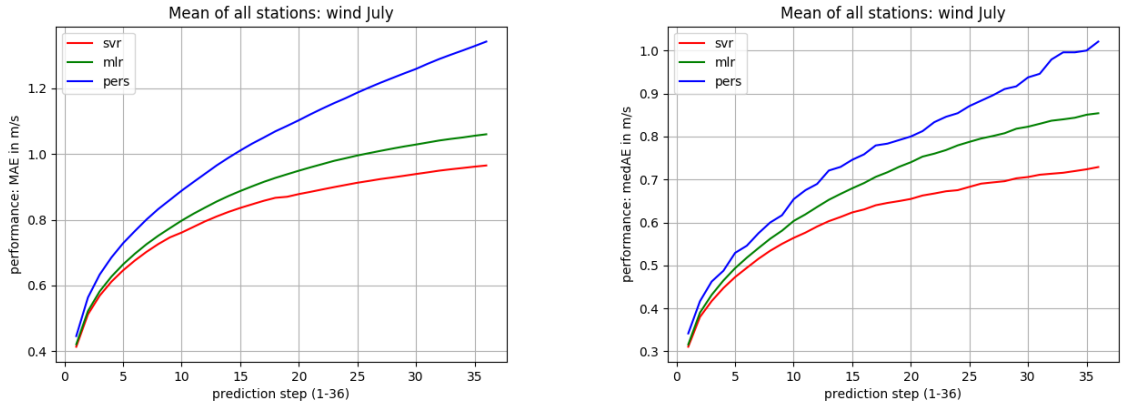
Overall for gust forecasting the SVR model performs well and improves predictions for nowcasting compared to the implemented baseline models. While the implemented models vary in performance depending on the season and the location, overall SVR brings improvement to current forecasts.

5.2 Results for Wind Speed Forecast

July 2018

Figure 5.7 shows mean performance of wind speed prediction for all stations for July 2018. For the first forecast time steps performance of MLR is close to the SVR performance but overall SVR performs better than the baseline models. For the majority of the investigated sites this trend is similar. Only for one mountain top station (Hirschenkogel) the MLR performs better than the SVR model.

Figure 5.8 shows performance for three test stations for July 2018. For station Wien Hohe Warte, shown in Figures 5.8a and 5.8b, the SVR model performs better than the baseline models with similar performance to the MLR model for the first steps. To investigate the impact of a longer training period on the SVR model, one model was build using training data of ten years (2007-2016) for station Wien Hohe Warte (11035). This SVR model is shown



(a) MAE (b) medAE

Figure 5.7: Wind speed prediction: July 2018, mean of all stations.

with a dashed purple line in Figures 5.8a and 5.8b. As this training data set is over three times larger than the previously used, training of a model for one step takes approximately ten hours. This is 20 times longer, corresponding to the complexity discussed in Section 4.8: for a data set three times the size of the previous one, the complexity increases with a factor $3^2 = 9$ to $3^3 = 27$. As we can see here, this longer training period does not improve model performance for the selected month.

Performance of station Bad Mitterndorf is shown in Figures 5.8c and 5.8d. Here, the SVR model also consistently performs better than MLR and persistence model. The small jump in performance at prediction step 19 indicates that models build with wrapper two could probably be improved by adjusting the feature selection. As we can see in Figure 5.9, there is some variation in the selected features for the different wrappers. Removing features `dd` and `rf` from step 1 and 10 might enhance the overall performance.

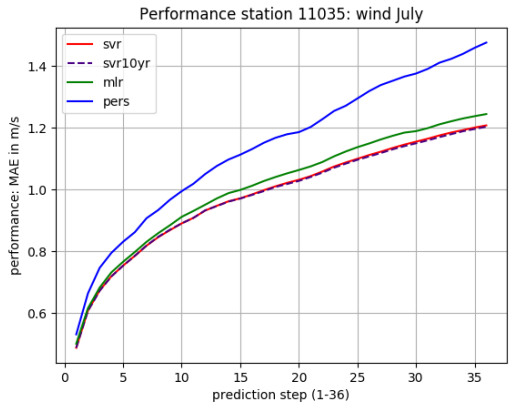
For station Gmünd (Figures 5.8e and 5.8f) performance of SVR is also better than the baseline models. Same as station Bad Mitterndorf, the jump in the SVR performance points to a wrapper one which could be improved in performance. We can see the selected features for each wrapper in Figure 5.10. As we have quite distinct differences between these selections, it is probable that a manual feature adjustment can improve these.

In Figure 5.11 the correlation coefficient R^2 for all stations is shown. Here, we can see that the SVR model seems to have a better fit than MLR and persistence model for wind speed forecasts.

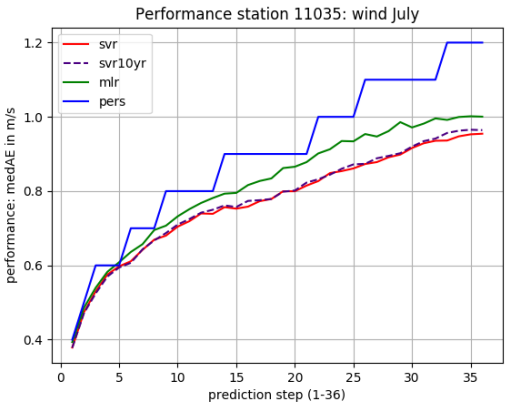
January 2018

For test month January 2018 the mean performance of wind speed models is shown in Figure 5.12. For this test period the mean MAE of the MLR models is better than the MAE of the SVR models. For the mean medAE, the SVR models show better performance.

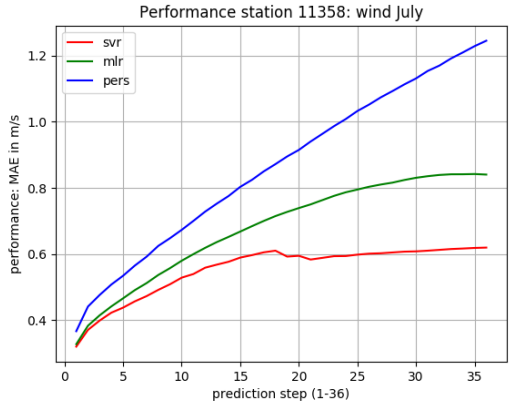
In Figure 5.13 a map with the locations of the test stations is shown. The different colors of markers are indications of the MAE performance of each station. The stations with purple markers indicate an SVR performance that is better than the MLR performance for both MAE and medAE. Stations with blue markers have performance similar to Figure 5.12: SVR performs worse than MLR for MAE score and the other way around for the medAE. Most stations with worse SVR performance than MLR are either in the north-eastern states: Upper Austria, Lower Austria, and Vienna, or they are stations located at higher altitude. This indicates that performance of the model is somehow related to the location of the station for



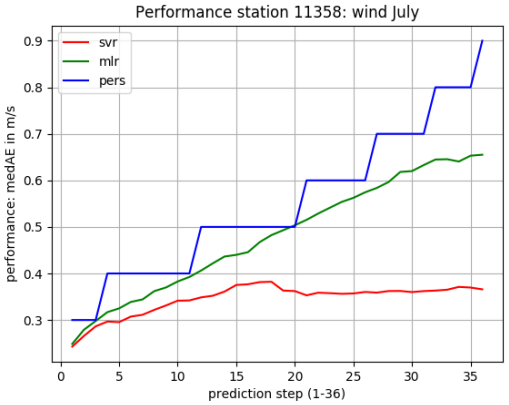
(a) MAE 11035



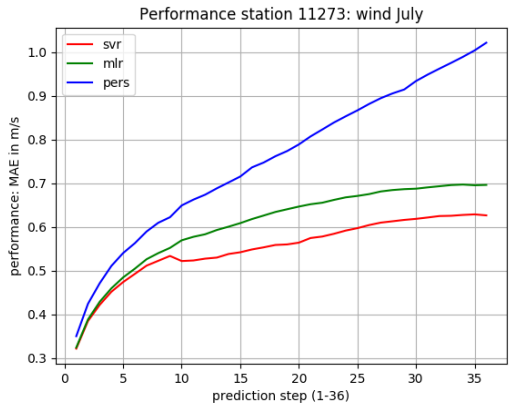
(b) medAE 11035



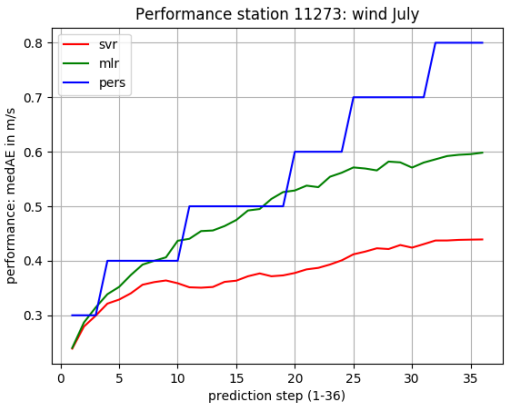
(c) MAE 11358



(d) medAE 11358



(e) MAE 11273



(f) medAE 11273

Figure 5.8: Wind speed prediction performance: July 2018, three stations.

Figure 5.9: Features selected for different steps (wind prediction), station Bad Mitterndorf (11358).

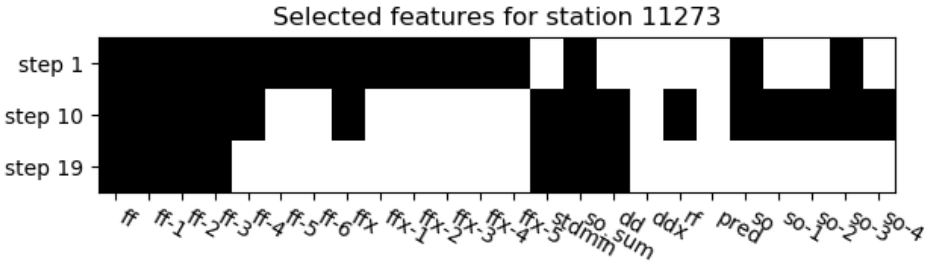


Figure 5.10: Features selected for different steps (wind prediction), station Gmünd (11273).

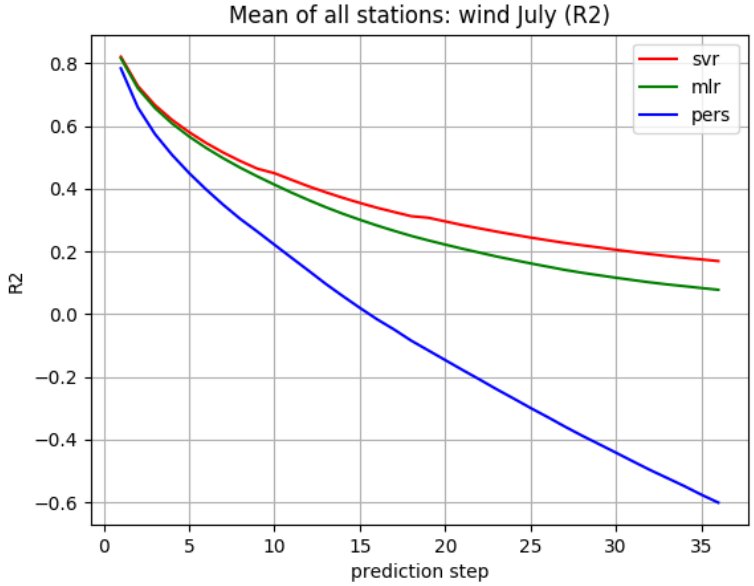
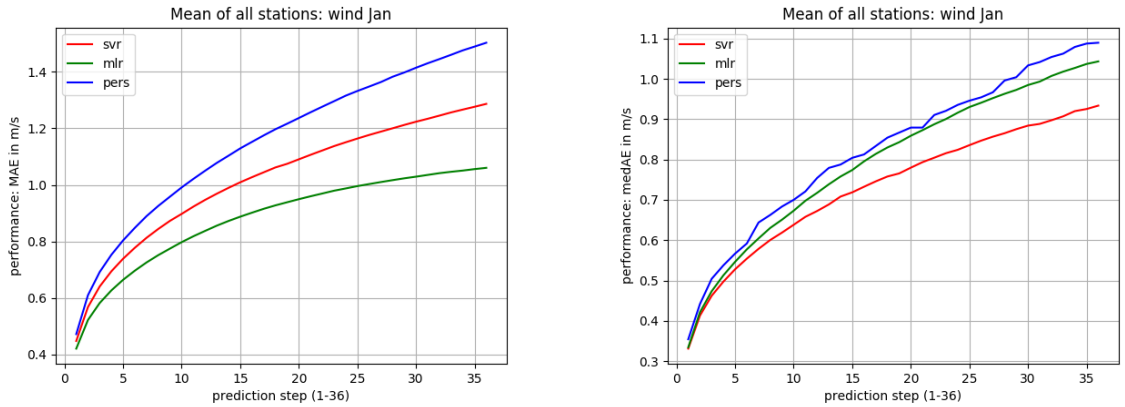


Figure 5.11: Wind speed prediction: Correlation coefficient R^2 for July 2018 (mean of all stations).



(a) MAE (b) medAE

Figure 5.12: Wind speed prediction performance: January 2018, mean of all stations.

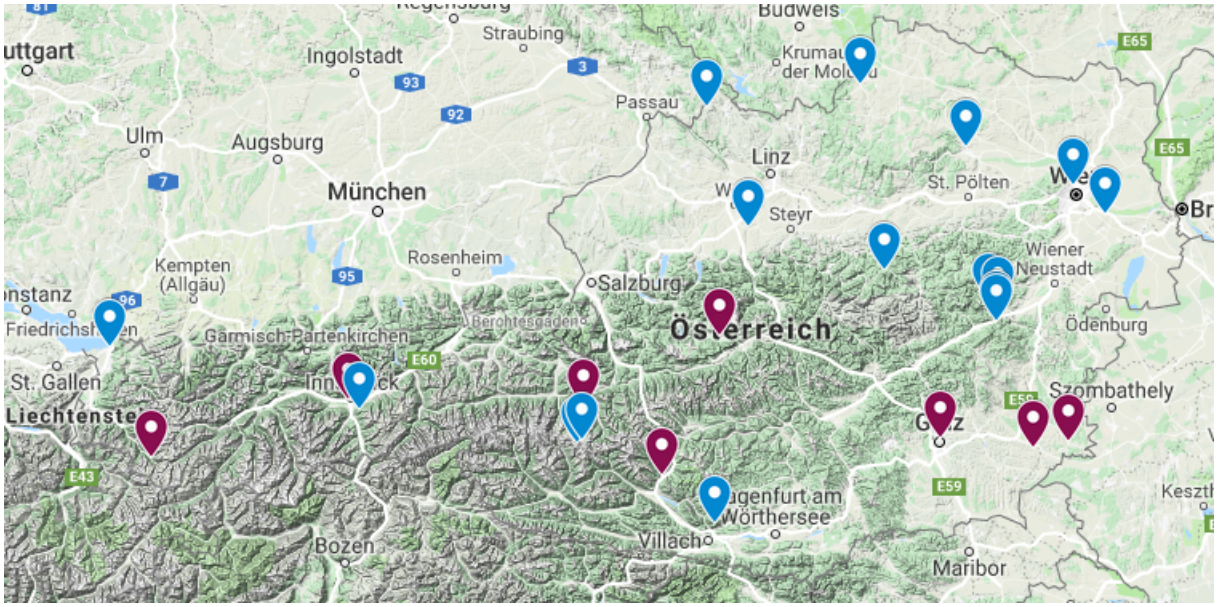


Figure 5.13: Map with stations with performance for January 2018; purple: SVR best performance, blue: MLR best performance.

this test month. As it is region and altitude dependent, it might be a result of unusual wind behavior for these parts of Austria or related to an unusual occurrence of specific weather types.

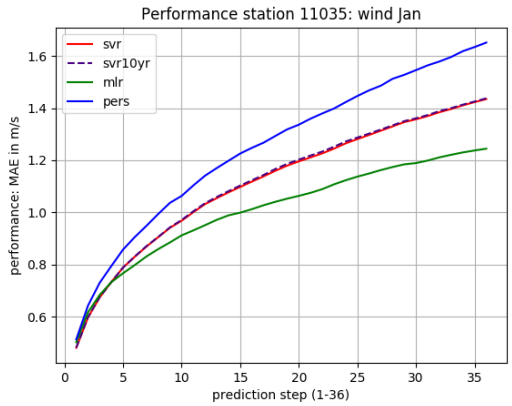
Figure 5.14 shows performance of three test stations for January 2018. Performance for station Wien Hohe Warte is shown in Figure 5.14a and 5.14b. It is one of the stations where the MLR model performs better than the SVR model for MAE but not for medAE. Here, again, the ten-year model does not improve performance either. This suggests that three years of training data are enough variation for the SVR model and using longer training periods might result in overfitting of the model.

Figures 5.14c and 5.14d show performance for station Bad Mitterndorf. Here, the SVR model mostly performs better than the MLR model for MAE and medAE. For the first ten forecast time steps the MLR performs better. The jump in the SVR MAE performance also suggests that an improvement of performance is possible with feature adjustment (see Figure 5.9).

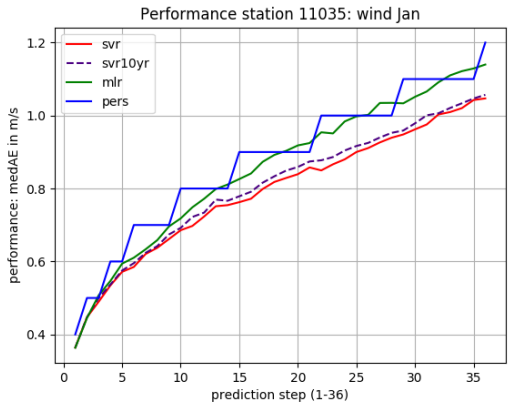
For station Gmünd (11273) performance is shown in Figures 5.14e and 5.14f. The SVR model shows best performance and the persistence model is also better than the MLR model. The medAE value for persistence is also better than SVR for later steps but as the persistence steps are much larger than for SVR these measures are more difficult to directly compare. Here the jump in SVR MAE performance also suggests that a feature adjustment could further improve the model (see Figure 5.10).

The station for which wind speed forecast performance of the SVR was not the best for both test months is Hirschenkogel (11384). It is a mountain top station, therefore more exposed than others and harder to predict. For wind speed prediction, the MLR model is consistently better than the SVR model and for gust prediction both models are very similar in performance.

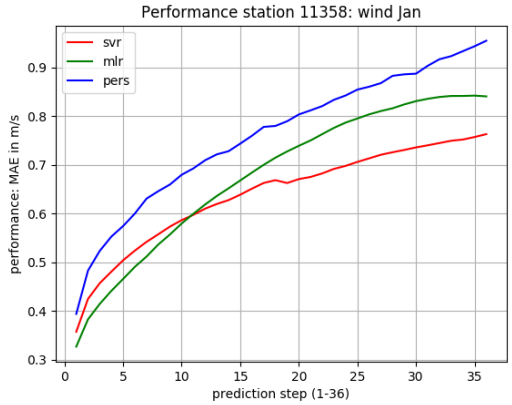
In Figure 5.15 the correlation coefficient R^2 for all stations is shown. For January 2018 performance of the SVR model is similar to the MLR performance. As this score was mostly used for tuning the SVR models, a better performance of SVR here might suggest an overfitting of the models to this score.



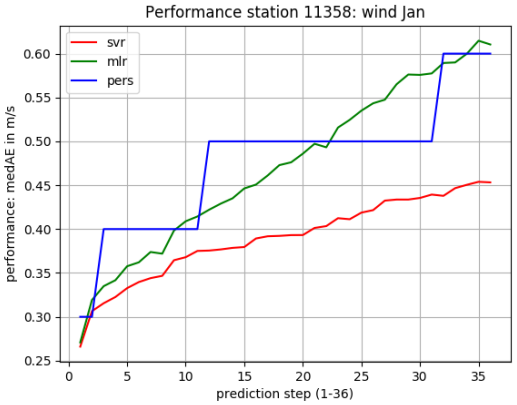
(a) MAE 11035



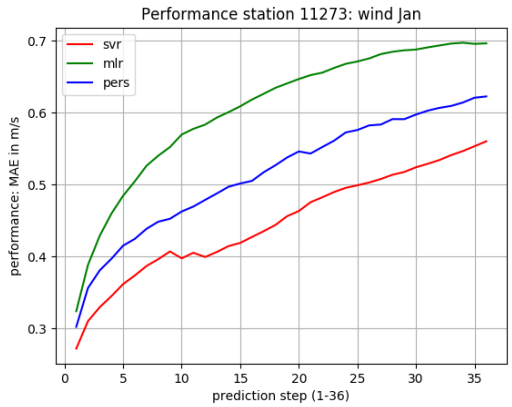
(b) medAE 11035



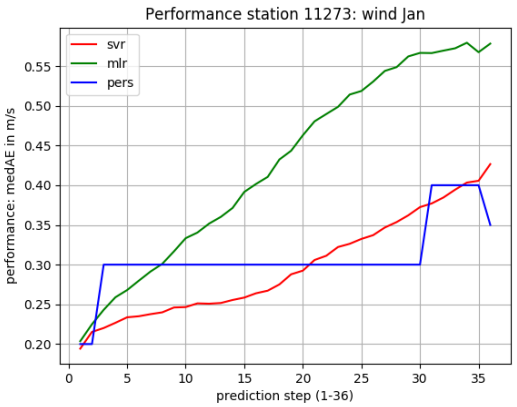
(c) MAE 11358



(d) medAE 11358



(e) MAE 11273



(f) medAE 11273

Figure 5.14: Wind speed prediction: January 2018, three stations.

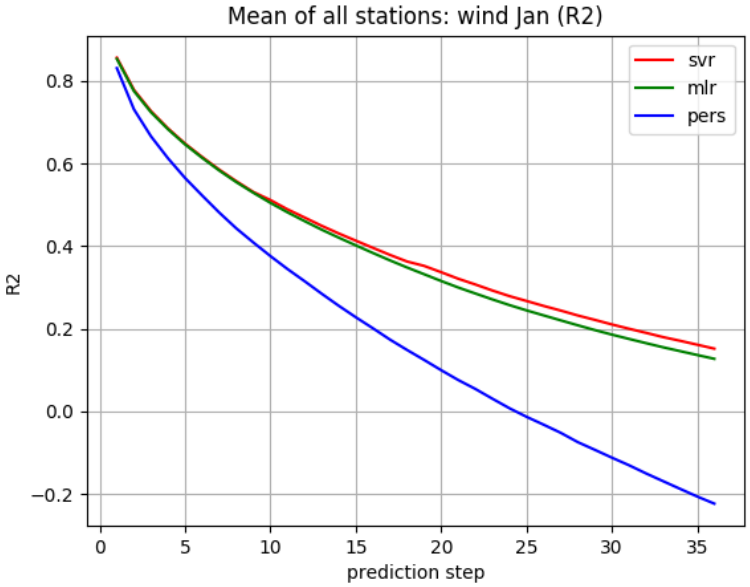


Figure 5.15: Wind prediction: Correlation coefficient R^2 for Jan 2018 (mean of all stations).

Chapter 6

Discussion and Outlook

6.1 Discussion of Results

The aim of this thesis was to build a machine learning model for gust and wind speed forecasting for the nowcasting range (0-6 hours ahead). The implemented SVR method gives good results for both gust and wind predictions and is computationally fast.

Overall, one can conclude that SVR seems to perform better for gust predictions than for wind speed predictions. It should be considered, though, that the main focus of this thesis is gust prediction, thus more time was spent on finding optimizations for gust predictions. If the same amount of time had also been put to explicitly optimizing wind speed models, a better performance might have been achievable, especially for test month January 2018. Specifically for the nowcasting range, we can also conclude that simpler models like MLR sometimes outperform complex models like SVR.

In Figure 6.1 one exemplary time series of prediction values for SVR (red) and MLR (green) are shown together with the actual values (dark blue). Wind speed time series are in continuous lines while gust speed time series are dashed. It is evident that both SVR and MLR can predict time series trends. In this example we can see for gust prediction how SVR is better than MLR at detecting patterns in the data such as the decrease of gust speed. Missing in the predictions is the random gust behavior as this is exactly the part of gusts that is unpredictable.

Another drawback of the SVR model is the high algorithmic complexity and long training times. Compared to SVR, the MLR model needs a fraction of its training time and performs almost as good as SVR. It would be interesting to see if the MLR model could be tuned even further.

Especially for nowcasting with a time horizon of two hours, a statistical method like MLR could be a very good alternative to SVR with a fraction of the training time. Performance is often similar for the first forecast time steps for the SVR and MLR model. For longer time horizons the machine learning model usually performs better than MLR.

To further improve the models, various measures can be taken. We could implement a more advanced method for replacing missing values of features. For these test stations, a simple interpolation proved sufficient as there are only few missing features. With further expansion of the models there might be more gaps in the data and using predictions from rapid update cycle NWP models (faster forecasts than other NWP models) for the training data instead of interpolation might be helpful.

Additionally, it is recommended to perform an in-depth investigation of the feature selection for the models (as done in Section 4.7.4) before using the models in real-time forecasts (called operational forecasts). As it seems like the sunshine duration improves performance of SVR even if it was not selected by the step-wise feature selection, it could be valuable to adjust

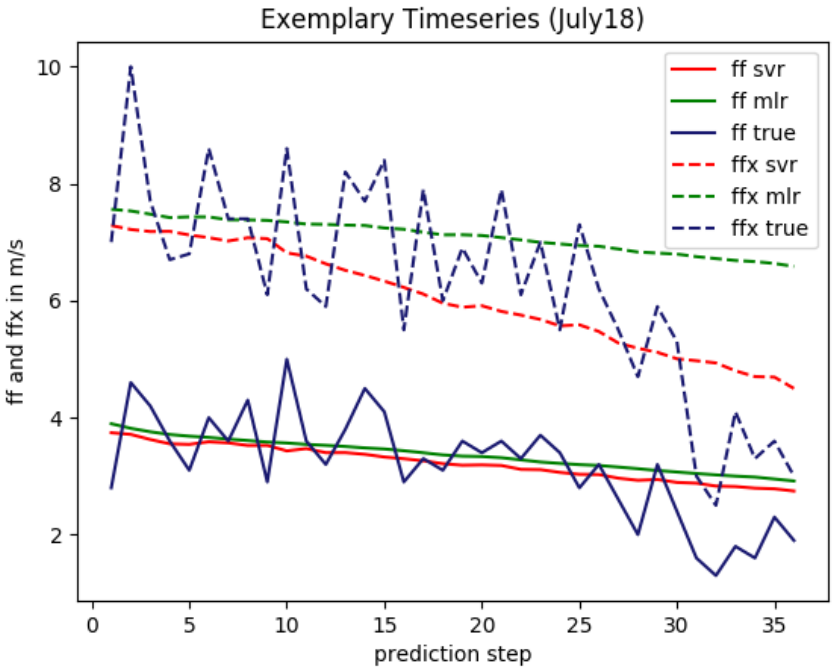


Figure 6.1: Exemplary time series with true values and MLR and SVR predictions for wind speed (continuous lines) and gust speed (dashed lines) (Wien Hohe Warte 11035), July 2018.

parameters of feature selection like number of splits or maximum training sample size. The step-wise feature selection is a helpful basis for determining which features are relevant for the different stations but to further improve models it might be necessary to adjust selected features manually and build models from those adjusted selections.

Another idea to improve predictions is to not only look at predictions for the actual time step but to also look at predictions of the previous two time steps. This would give us three predictions for each of the next 34 prediction steps. Even though input features for each neighboring step are already quite similar, with the random gust behavior there can still be quite a variety in forecasts from one 10-minute step to the next. It would be interesting to see if averaging these three predictions further improves the forecast.

To speed up the training of models, we could also try a GPU implementation for the SVR method.

Generally, before these models are put into any kind of operational use, we need to build a system to re-calibrate the models. Even though it is not clear if more training data would actually improve the models, weather phenomena do change depending on climate change and it can only be an advantage to retrain models on new data.

Overall, we can conclude that SVR is a good method to improve forecasts of wind gusts and wind speed for nowcasting. It performs better in predicting patterns in wind gusts compared to statistical methods and delivers fast results.

6.2 Outlook

Reliable gust predictions are of importance for many real-world applications like wind energy parks, general storm alerts, as well as warnings for drivers of vehicles on exposed roads.

The SVR models build in this thesis are prototypes which might be improved further with the additions and adjustments mentioned previously. To use them in operational forecasts, they can be incorporated into a forecasting model at ZAMG.

As most forecasting models use a spatial grid and not isolated point forecasts, we could also extend our point forecasts to a grid. This could be done for example by interpolating between stations with consideration of the topography of the stations. As these grid forecasts would take more computation time than just a point forecast, it would need to be evaluated if it is worthwhile. Another method to include these models into operational systems would be to use the SVR forecasting output to improve predictions and adjust other grid points accordingly.

Abstract

In this thesis support vector machines (SVMs) are investigated for ultra-short range predictions (next 0-6 hours) of wind speed and wind gusts. Predictions are provided with a ten-minute frequency and should be computationally fast. Numerical forecasting methods commonly used for meteorological forecasting are based on physical principles and therefore require long computation times. Thus, they are not suitable for ultra-short range (called nowcasting) predictions and only meteorological observation data is used here. The aim of this thesis is to evaluate a machine learning approach, support vector regression (SVR), for point forecasts of wind speed and gusts.

Features of the meteorological observation data found to be relevant for prediction of wind speed and gusts are: wind speed, wind gusts, wind direction, gust direction, sunshine duration, time of day, relative humidity, and reduced pressure. For the SVR models a radial basis function is used as kernel function and hyperparameters are optimized individually for each model via grid search. A selection of 24 TAWES sites (semi-automatic weather station) in different Austrian regions with varying topography and climatology is used. A step-wise feature selection algorithm is built to find the best feature selection for each testing site based on the training cross-validation score. The multi-step model is composed of one model for each of the 36 time steps (six hours total). The forecasting horizon is split into three time frames and the feature selection algorithm is run once for each (for the first time step of the time frame). For comparison baseline models are implemented: a persistence approach and a multiple linear regression model (MLR) with optimized feature selection.

The SVR models give good results for both gust and wind predictions and are computationally fast. They perform better in predicting patterns in wind gusts compared to statistical methods. Improvement of forecasts carried out with SVR compared to forecasts done with MLR are greater during the summer test month than during the winter test month. SVR models perform better for gust predictions than for wind speed predictions, which might be due to optimization being focused on gust prediction models. For the nowcasting range, we can conclude that simpler models like MLR sometimes outperform complex models like SVR. The MLR model takes a fraction of the training time and performs almost as good as SVR. For a time horizon of two hours, a statistical method like MLR could be a very good alternative to SVR. For longer time horizons the improvement of forecasts is greater and a SVR model usually performs better than MLR.

Reliable gust prediction is of importance for many real-world applications like wind energy parks, general storm alerts, as well as warnings for drivers of vehicles on exposed roads. Before these prototypes are implemented for any kind of operational use (real-time forecasts), they need to be re-calibrated as weather phenomena do change depending on climate change.

Zusammenfassung

Diese Arbeit untersucht Support Vector Machines (SVMs) für Ultra-Kurzzeit Vorhersagen (nächste 0-6 Stunden) von Windgeschwindigkeit und Windböen. Vorhersagen werden in einer Zehn-Minuten-Frequenz ausgegeben und sollten innerhalb kürzester Zeit verfügbar sein. Numerische Vorhersagemethoden, die üblicherweise für meteorologische Vorhersagen benutzt werden, basieren auf physikalischen Prinzipien und benötigen daher lange Rechenzeiten. Aufgrunddessen sind sie nicht für Ultra-Kurzzeit (genannt Nowcasting) Vorhersagen geeignet. Es werden hier nur meteorologische Beobachtungsdaten verwendet. Das Ziel dieser Arbeit ist es, einen Machine Learning Ansatz, Support Vector Regression (SVR), für Punktprognosen von Wind und Böen zu implementieren.

Relevante Features der meteorologischen Beobachtungsdaten für Wind- und Böenvorhersage sind: Windgeschwindigkeit, Böengeschwindigkeit, Windrichtung, Böenrichtung, Sonnenscheindauer, Tageszeit, relative Feuchte und reduzierter Luftdruck. Für das SVR Modell wird eine radiale Basisfunktion als Kernel verwendet und die Hyperparameter werden für jedes Modell einzeln unter Verwendung von Gridsearch optimiert. Eine Auswahl von 24 TAWES (Teilautomatische Wetterstationen) in verschiedenen Regionen Österreichs mit unterschiedlicher Topographie und Klimatologie wird verwendet. Ein Algorithmus zur schrittweisen Featureauswahl anhand von Kreuzvalidierung wird konstruiert, um die besten Features für jeden Standort zu finden. Das Multi-Step Modell ist aus den Modellen der 36 Einzelzeitschrittvorhersagen zusammengesetzt. Der Vorhersagezeitraum ist in drei Zeitfenster aufgeteilt und der Algorithmus zur Featureauswahl wird einmal für jedes der Zeitfenster ausgeführt (für jeweils den ersten Vorhersagezeitpunkt). Zum Vergleich sind zwei Referenzmodelle implementiert: ein Persistenzmodell und ein multiple lineare Regressionsmodell (MLR) mit optimierter Featureauswahl.

Die entwickelten SVR Modelle liefern gute Ergebnisse für Böen- und Windvorhersage und benötigen nur geringe Rechenressourcen. Muster von Windböen sind besser repräsentiert als durch statistische Methoden und für die Böenvorhersage können bessere Vorhersagen geliefert werden. Resultate für die Windvorhersage liefern ähnliche Ergebnisse für die SVR and MLR Modelle. Dies könnte allerdings daran liegen, dass der Fokus der Arbeit auf Böenvorhersage war und darauf die Optimierung der SVR fokussiert wurde. Besonders für Nowcasting können wir folgern, dass einfachere Modelle wie MLR manchmal bessere Leistung bringen als komplexere Modelle wie SVR. Das MLR Modell verwendet nur einen Bruchteil der Trainingszeit und bringt Ergebnisse, die sehr nah an den SVR Ergebnissen sind. Für einen Vorhersagezeitraum von zwei Stunden ist eine statistische Methode wie MLR eine gute Alternative zu SVR. Für längere Vorhersagezeiträume ist der Unterschied zwischen den Methoden größer und das SVR Modell meist besser als MLR.

Verlässliche Böenvorhersagen sind relevant für viele reale Anwendungen, u.a. Windparks, allgemeine Sturmwarnungen und Gefahrenwarnung für Motorrad- und Autofahrer auf freiliegenden Straßen. Bevor diese Prototypen in jegliche Art von Echtzeitvorhersagen integriert werden können, müssen sie rekalibriert werden, da Wetterphänomene sich mit dem Klimawandel verändern.

Bibliography

Ahmed, S., Khalid, M., and Akram, U. (2017). "A method for short-term wind speed time series forecasting using Support Vector Machine Regression Model". In: *Clean Electrical Power (ICCEP), 2017 6th International Conference on*. IEEE, pp. 190–195.

Botha, N. and Walt, C. M. van der (2017). "Forecasting wind speed using support vector regression and feature selection". In: *Pattern Recognition Association of South Africa and Robotics and Mechatronics (PRASA-RobMech), 2017*. IEEE, pp. 181–186.

Browell, J., Drew, D., and Philippopoulos, K. (2017a). "Improved very short-term spatio-temporal wind forecasting using atmospheric regimes". In: *Wind Energy*.

Browell, J., Drew, D. R., and Philippopoulos, K. (2017b). "Improved very-short-term wind forecasting using atmospheric classification". In: *arXiv preprint arXiv:1705.07158*.

Browell, J. and Gilbert, C. (2017). "Cluster-based regime-switching AR for the EEM 2017 Wind Power Forecasting Competition". In: *European Energy Market (EEM), 2017 14th International Conference on the*. IEEE, pp. 1–6.

Chang, C.-C. and Lin, C.-J. (2011). "LIBSVM: a library for support vector machines". In: *ACM transactions on intelligent systems and technology (TIST)* 2.3, p. 27.

Dowell, J. and Pinson, P. (2016). "Very-short-term probabilistic wind power forecasts by sparse vector autoregression". In: *IEEE Transactions on Smart Grid* 7.2, pp. 763–770.

Friederichs, P. et al. (2009). "A probabilistic analysis of wind gusts using extreme value statistics". In: *Meteorologische Zeitschrift* 18.6, pp. 615–629.

Häckel, H. (2016). *Meteorologie*. 8th ed. Stuttgart: Verlag Eugen Ulmer.

Haiden, T. et al. (2010). "Integrated nowcasting through comprehensive analysis (INCA) system description". In: *ZAMG report* 60.

Haltiner, G. J. (1971). *Numerical weather prediction*. John Wiley and Sons, Inc.

Hsu, C.-W., Chang, C.-C., Lin, C.-J., et al. (2003). *A practical guide to support vector classification*.

Hu, Q., Zhang, S., Xie, Z., et al. (2014). "Noise model based ν-support vector regression with its application to short-term wind speed forecasting". In: *Neural Networks* 57, pp. 1–11.

Hu, Q., Zhang, S., Yu, M., et al. (2016). "Short-term wind speed or power forecasting with heteroscedastic support vector regression". In: *IEEE Transactions on Sustainable Energy* 7.1, pp. 241–249.

Kang, A. et al. (2017). "Short-term wind speed prediction using EEMD-LSSVM model". In: *Advances in Meteorology* 2017.

Kazor, K. and Hering, A. S. (2015). "The role of regimes in short-term wind speed forecasting at multiple wind farms". In: *Stat* 4.1, pp. 271–290.

Khosravi, A. et al. (2018). "Prediction of wind speed and wind direction using artificial neural network, support vector regression and adaptive neuro-fuzzy inference system". In: *Sustainable Energy Technologies and Assessments* 25, pp. 146–160.

Klose, B. (2016). *Meteorologie: Eine interdisziplinäre Einführung in die Physik der Atmosphäre*. 3rd ed. Berlin, Heidelberg: Springer Spektrum.

Kong, X. et al. (2015). "Wind speed prediction using reduced support vector machines with feature selection". In: *Neurocomputing* 169, pp. 449–456.

Kramer, O. and Gieseke, F. (2011). "Short-term wind energy forecasting using support vector regression". In: *Soft Computing Models in Industrial and Environmental Applications, 6th International Conference SOCO 2011*. Springer, pp. 271–280.

Krennert, T. (2009). *Wetterlagenklassifikationen an der ZAMG – von den Anfängen bis zu aktuellen Anwendungen*. URL: https://iwhw.boku.ac.at/Klien_Extremwerte/Temp_/Papers_Methodik/wlk_zamg_tk_090310.pdf (visited on Sept. 21, 2018).

Kretzschmar, R. et al. (2004). "Neural network classifiers for local wind prediction". In: *Journal of Applied Meteorology* 43.5, pp. 727–738.

Liu, D., Niu, D., et al. (2014). "Short-term wind speed forecasting using wavelet transform and support vector machines optimized by genetic algorithm". In: *Renewable Energy* 62, pp. 592–597.

Liu, J. and Zio, E. (2017). "SVM hyperparameters tuning for recursive multi-step-ahead prediction". In: *Neural Computing and Applications* 28.12, pp. 3749–3763.

Mercer, A. and Dyer, J. (2014). "A New Scheme for Daily Peak Wind Gust Prediction Using Machine Learning". In: *Procedia Computer Science* 36, pp. 593–598.

Mohandes, M. A. et al. (2004). "Support vector machines for wind speed prediction". In: *Renewable Energy* 29.6, pp. 939–947.

Monteiro, C. et al. (2009). *Wind power forecasting: State-of-the-art 2009.* Tech. rep. Argonne National Lab.(ANL), Argonne, IL (United States).

Müller, K.-R. et al. (1997). "Predicting time series with support vector machines". In: *Artificial Neural Networks — ICANN'97*. Ed. by W. Gerstner et al. Berlin, Heidelberg: Springer Berlin Heidelberg, pp. 999–1004.

Müller, K.-R. et al. (1999). "Using support vector machines for time series prediction". In: *Advances in kernel methods—support vector learning*, pp. 243–254.

Papazek, P. (2018). "Artificial Neural Network and Data Mining Approaches for Short-range Wind Speed Forecasts". Master's thesis. Universität Wien.

Pedregosa, F. et al. (2011). "Scikit-learn: Machine Learning in Python". In: *Journal of Machine Learning Research* 12, pp. 2825–2830.

Pinto, T. et al. (2014). "Short-term wind speed forecasting using Support Vector Machines". In: *Computational Intelligence in Dynamic and Uncertain Environments (CIDUE), 2014 IEEE Symposium on*. IEEE, pp. 40–46.

Plant, C. (2017). *Data Mining [lecture notes]*.

Sallis, P. J., Claster, W., and Hernández, S. (2011). "A machine-learning algorithm for wind gust prediction". In: *Computers & Geosciences* 37.9, pp. 1337–1344.

Santamaría-Bonfil, G., Reyes-Ballesteros, A., and Gershenson, C. (2016). "Wind speed forecasting for wind farms: A method based on support vector regression". In: *Renewable Energy* 85, pp. 790–809.

Schicker, I. et al. (2017). "Short-range wind speed predictions for complex terrain using an interval-artificial neural network". In: *Energy Procedia* 125. European Geosciences Union General Assembly 2017, EGU Division Energy, Resources & Environment (ERE), pp. 199–206.

Shanmuganathan, S. and Sallis, P. (2014). "Data mining methods to generate severe wind gust models". In: *Atmosphere* 5.1, pp. 60–80.

Sheridan, P. (2018). "Current gust forecasting techniques, developments and challenges". In: *Advances in Science and Research* 15, pp. 159–172.

Singer, I. A. and Smith, M. E. (1953). "Relation of gustiness to other meteorological parameters". In: *Journal of Meteorology* 10.2, pp. 121–126.

Smola, A. J. and Schölkopf, B. (2004). *A tutorial on support vector regression*.

Sprenger, M. et al. (2017). "Nowcasting foehn wind events using the adaboost machine learning algorithm". In: *Weather and Forecasting* 32.3, pp. 1079–1099.

Spudić, V., Marić, M., and Perić, N. (2009). "Neural networks based prediction of wind gusts". In: *European Wind Energy Conference EWEC 2009*.

Sreelakshmi, K. and Kumar, P. R. (2008). "Performance evaluation of short term wind speed prediction techniques". In: *IJCSNS International Journal of Computer Science and Network Security*. Citeseer.

Thorarinsdottir, T. L. and Johnson, M. S. (2012). "Probabilistic wind gust forecasting using nonhomogeneous Gaussian regression". In: *Monthly Weather Review* 140.3, pp. 889–897.

Vapnik, V. (1998). *Statistical learning theory. 1998*. Vol. 3. Wiley, New York.

Wang, Y. et al. (2018). "Correlation aware multi-step ahead wind speed forecasting with heteroscedastic multi-kernel learning". In: *Energy Conversion and Management* 163, pp. 384–406.

Wani, M. A. and Bhat, H. F. (2017). "Multiclass SVM algorithms for wind speed prediction". In: *Renewable Energy Research and Applications (ICRERA), 2017 IEEE 6th International Conference on*. IEEE, pp. 1139–1143.

Yao, B. et al. (2014). "Improved support vector machine regression in multi-step-ahead prediction for rock displacement surrounding a tunnel". In: *Scientia Iranica. Transaction A, Civil Engineering* 21.4, p. 1309.

ZAMG (2018). *Stationsliste_20180701.csv*. German. URL: `https://www.zamg.ac.at/cms/de/klima/messnetze/wetterstationen` (visited on Sept. 27, 2018).

Zeng, J. and Qiao, W. (2011). "Support vector machine-based short-term wind power forecasting". In: *Power Systems Conference and Exposition (PSCE), 2011 IEEE/PES*. IEEE, pp. 1–8.

Zhang, H. et al. (2014). "Support vector regression based on grid-search method for short-term wind power forecasting". In: *Journal of Applied Mathematics* 2014.

Zhao, P. et al. (2010). "Wind speed prediction using support vector regression". In: *Industrial Electronics and Applications (ICIEA), 2010 the 5th IEEE Conference on*. IEEE, pp. 882–886.

Zhou, J., Shi, J., and Li, G. (2011). "Fine tuning support vector machines for short-term wind speed forecasting". In: *Energy Conversion and Management* 52.4, pp. 1990–1998.