Scarlatina
– a time series analysis

Tudományos Diákköri Dolgozat

Máté Kormos
Applied Economics BA
III. grade
Faculty of Economics
March 2016

Supervisor: Benedek András Rózemberczki
Abstract

In this paper, some methods are taken into account to model the time series data of scarlet fever cases (also known as, by its Latin name, scarlatina) in Hungary, relying on weekly data from 23rd March 1998 to 5th October 2015. The purpose of this paper is to select the most appropriate method for this task. The methods involved are machine learning techniques (boosting trees and support vector machine regression) and 'traditional' ones (nonlinear least squares, nonlinear Poisson, nonlinear negative binomial regression). First, discussed are the methods to be followed by the description of the exact models applied to the given data. Last, the comparison of the models is carried out on an independent dataset only to let the nonlinear Poisson regression emerge as the best one.

JEL Classification: I10, I18, I19

Keywords: Scarlet fever, Scarlatina, Count data, Time-series, Machine learning, Poisson regression, Negative binomial regression, Nonlinear least squares, Boosting trees, Support vector machine
Acknowledgements

The author would like to express thanks to the supervisor, Benedek András Rózemberczki, for providing free access to the scarlatina data and for his help and guidance on writing this paper.
Contents

List of Figures

List of Tables

1 Introduction 1

2 Data 2

3 Methods 4

3.1 Boosting trees . 4

3.1.1 Single tree . 5

3.1.2 Boosting trees . 7

3.2 Support Vector Machine . 7

3.3 Nonlinear least squares . 9

3.4 Poisson regression . 10

3.5 Negative binomial . 13

4 Models 16

4.1 Sample partitioning . 16

4.2 Variables . 16

4.3 Selection and tuning . 19

4.3.1 Cross-validation . 19

4.3.2 Loss measure . 19

4.4 Boosting trees . 20

4.4.1 Predictor selection . 20

4.4.2 Tuning the number of terminal nodes . 22

4.4.3 Number of weak learners . 23

4.4.4 Final ensemble . 23

4.5 Support vector machine . 23
List of Figures

2.1 Partitioning of the sample ........................................... 3

3.1 A made-up binary regression tree, based on a binary and a continuous
variable. The outcome variable \( y_i \) is the number of the newly infected
on week \( i \). The number of terminal nodes is three: \( R_m, m = 1, 2, 3 \) .... 5

4.1 Histograms of Scarlatina and that of its logarithm ..................... 17
4.2 Time series of Scarlatina and that of its logarithm ..................... 18
4.3 Plotmatrix of the variables ............................................ 18
4.4 Predictor selection ..................................................... 21
4.5 Splits selection ....................................................... 22
4.6 SVM grid search ....................................................... 25
4.7 Training prediction of the ML models ................................ 31
4.8 Training prediction of the traditional models .......................... 32
4.9 Estimated distribution with maximum likelihood ....................... 32

5.1 Test prediction with boosting trees and SMV ........................ 35
5.2 Test prediction with traditional models ............................... 36
List of Tables

2.1 Descriptive statistics of the data sets .................................. 3
4.1 Boosting trees models of three different predictor group .......... 21
B.1 Regression parameters – traditional models ....................... II
B.2 Test results ........................................................................ III
Chapter 1: Introduction

Scarlet fever (or, by its latin name, *scarlatina*) is a bacterial infection, mainly affecting children, that can be treated with antibiotics. In this paper, selecting the most appropriate model for predicting time series data of scarlet fever cases in Hungary from 23rd March 1998 to 5th October 2015 is of interest.

In doing so, the author compares different regression approaches, involving machine learning and more traditional techniques. The former consists of boosting trees and support vector machine, whilst the latter is formed of nonlinear least squares, nonlinear Poisson and nonlinear negative binomial regression. Once the methods are detailed and presented, their application to the scarlatina time series data is addressed. To select the most suitable model, testing of the models on an independent dataset is completed.

The structure of the paper is as follows: Chapter 2 provides insight into the data, Chapter 3 presents the theory and the derivation of the methods, whilst Chapter 4 details their application to scarlatina time series only to be compared, using the test set, in Chapter 5.

---

Chapter 2: Data

The scarlatina data used in this paper are time series ranging from 23rd March 1998 to 5th October 2015, and with weekly observations are making up a sample of \( n = 916 \). The source of the data is the National Center for Epidemiology\(^1\), more precisely the data are published in the online journal of the Center: Epinfo. The unit in which the data are measured is the reported scarlet fever cases (i.e. infected people) per week.

The treatment of the sample is as follows. The whole sample is partitioned into two sets: the training and the test set. 100 observations are left to form the latter one, from 11st November 2013 to the closing date. Figure 2.1 illustrates the time series and the partitioning, in addition to Table 2.1 which contains some descriptive statistics of the sets.

As the figure and the table suggest, the training set is more volatile than the test set. This may result in the problem of overfitting in case the models, constructed based on the training set, fit too hard, grasping too much noise. However, this is the topic of Chapter 5.

\(^1\)National Center for Epidemiology website last access: 22nd February 2016.
Figure 2.1: Partitioning of the sample

![Partitioning of the sample](image)

Table 2.1: Descriptive statistics of the data sets

<table>
<thead>
<tr>
<th></th>
<th>Whole set</th>
<th>Training set</th>
<th>Test set</th>
</tr>
</thead>
<tbody>
<tr>
<td>Mean</td>
<td>69.3624</td>
<td>72.6887</td>
<td>42.2200</td>
</tr>
<tr>
<td>Std.</td>
<td>52.3760</td>
<td>53.7108</td>
<td>27.7480</td>
</tr>
<tr>
<td>Median</td>
<td>60</td>
<td>62</td>
<td>38.5000</td>
</tr>
<tr>
<td>Max.</td>
<td>304</td>
<td>304</td>
<td>112</td>
</tr>
<tr>
<td>Min.</td>
<td>2</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>Iqr.</td>
<td>64.5000</td>
<td>67</td>
<td>46.5000</td>
</tr>
</tbody>
</table>

Notes: Presented are some basic statistics measures of the scarlatina time series concerning the whole sample and the training and the test set. The abbreviations marked with dot: Std. – standard deviation, Max. – maximum value, Min. – minimum value, Iqr. – interquartile range.
Chapter 3: Methods

In this chapter, given is a general review of some machine learning (henceforth ML) and regression techniques that can be of assistance examining the spread of scarlet fever. Firstly, the ML approaches are presented in Section 3.1 (boosting trees) and 3.2 (support vector machine) to be followed by the nonlinear squares (Section 3.3 henceforth NLS), the Poisson (Section 3.3) and the negative binomial (Section 3.5) regression. The ML methods and the specific areas of their applications outlined in this paper are largely based on the comprehensive work of Hastie et al. (2009), whilst the other methods rely on Cameron and Trivedi (2005), Cameron and Trivedi (1998), Greene (2003) and Wooldridge (2010).

The notations in this chapter are as follows: \( y \) is the vector of the independent variable, \( X_{n \times J} \) is the design matrix, with \( i = 1, 2, \ldots, n \) observations and \( j = 1, 2, \ldots, J \) predictors. Consequently, sought is the conditional expected value of \( Y_i \) given the input vector \( X_i \), as a function of \( X_i \):

\[
E(Y_i \mid X_i) = f(X_i).
\]  
(3.1)

The methods above differ in means they construct and derive the function \( f(X_i) \) so this chapter is dedicated to this question.

3.1 Boosting trees

Being one of the most efficient machine learning (ML) techniques, boosting trees is widespread due to its flexibility, which also characterizes most of its ML counterparts [Hastie et al. 2009]. As opposed to parametric approaches, such as the NLS or nonlinear count data models, ML methods are famous for not establishing prerequisites (for e.g. distribution knowledge) for modelling, facilitating a more flexible application relying on the very structure of the data rather than prefixed parameter-
Boosting trees is a so-called ensemble model, combining more weak learners. The weak learners in this case are, as the name suggests, regression trees, first introduced by Morgan and Sonquist (1963) offering a proper way to handle interaction effects between variables, and later improved by Breiman et al. (1984) and Quinlan (1993). As Hastie et al. (2009) puts it, boosting is to sequentially apply weak learning algorithm to repeatedly modified versions of the data, thereby producing a sequence of weak learners. In the following, the author first discusses the single weak learner tree and after that described is the boosting.

3.1.1 Single tree
Regression trees are easy to interpret and visualize. The graph on Figure 3.1 illustrates the structure of a binary tree. As it can be seen, the tree is based on series of splitting: at every split point the tree splits into two nodes (from here comes the name binary\footnote{For clarification: both terms 	extit{single} and 	extit{binary} are used in this chapter, however, they are not to be confused. 	extit{Single} tree means one regression tree, while 	extit{binary} stands for the splitting method, i.e. partitioning into two parts.}) until the process terminates, leading to the final tree of decision nodes (at which it splits) and terminal nodes (at which it does not; marked with $R_m$).

Figure 3.1: A made-up binary regression tree, based on a binary and a continuous variable. The outcome variable $y_i$ is the number of the newly infected on week $i$. The number of terminal nodes is three: $R_m$, $m = 1, 2, 3$. 

Even though the concept is clear, the following questions have to be addressed to fit a binary tree model:
1. What should serve as the basis of split decisions?

2. How deep should the tree be (the number of terminal nodes, $M$)?

3. Having finalized the terminal nodes, what value should the dependent variable have ($E(Y_i \mid R_m)$)?

Formally, to answer Question 1, the whole input space ($X$) is partitioned into two half-plane defined by variable $X_j$ and split point $s$:

$$R_1(j, s) = \{ X \mid X_j \leq s \}$$

$$R_2(j, s) = \{ X \mid X_j > s \},$$

that is, into two nodes fitted by the optimization problem using least-squares loss:

$$\min_{j,s} \left\{ \min_{c_1} \left[ \sum_{x_i \in R_1(j,s)} (y_i - c_1)^2 \right] + \min_{c_2} \left[ \sum_{x_i \in R_2(j,s)} (y_i - c_2)^2 \right] \right\},$$

where $c_1$ and $c_2$ are the answers to Question 3: $E(Y_i \mid R_m) = c_m$. Given $j$ and $s$, the inner minimization is solved by

$$\hat{c}_1 = \text{average} \left( y_i \mid x_i \in R_1(j, s) \right)$$

$$\hat{c}_2 = \text{average} \left( y_i \mid x_i \in R_2(j, s) \right),$$

thus the average of the dependent variable in the space defined by nodes. In the words of Hastie et al. (2009), for each splitting variable $j$ the determination of the split point $s$ is computationally cheap and by scanning through each input variable $X_j$ the solution $(j^*, s^*)$ to (3.4) can be found, gaining two nodes. This procedure is to be repeated until the final depth of the tree is achieved.

Answered are Question 1 and 3 above, leaving the tree depth yet to be defined. The number of terminal nodes is a tuning parameter to be set (with cross-validation for example) while one regression tree is fitted. For ease of notation, let

$$T(X, \Xi) = \sum_{m=1}^{M} c_m I(X \in R_m)$$

denote the fitted single regression tree, with $X$ marking the input space, $I$ being the identity function signing of which node $x_i$ is part and $\Xi = \{ c_m, R_m \}_{m=1}^{M}$, so that

$$E(Y_i \mid x_i) = T(x_i, \Xi)$$

(3.8)
3.1.2 Boosting trees

As it is mentioned in the introduction of Section 3.1, boosting trees consists of repeatedly applying weak learners to the data. This happens in a forward stage-wise additive fashion (Hastie et al., 2009), so the expression (3.7) is modified as

\[ T_s(X, \Xi_s) \]

such that \( s = 1, 2, \ldots, S \), which previously stood for the split point, hereafter means the stage at which the single tree is estimated. Using this notation and the loss function \( L \), the way of fitting is

\[ \hat{\Xi}_s = \arg \min_{\Xi_s} \left\{ \sum_{i=1}^{n} L[y_i, \eta f_{s-1}(x_i) + T_s(x_i, \Xi_s)] \right\} \],

(3.10)

where \( f_{s-1} \) is the aggregated prediction of the previously fitted trees:

\[ f_{s-1}(x_i) = \sum_{q=1}^{s-1} T_q(x_i, \Xi_q) \]

(3.11)

\( \eta \) is the learning rate, with values \( \eta \in [0, 1] \), which is a tuning parameter: the closer \( \eta \) is to 1, the harder one fits the training data, potentially risking overfitting. Repeating (3.10) \( S \) times results in the boosted ensemble,

\[
E(Y_i \mid x_i) = \sum_{s=1}^{S} T_s(x_i, \Xi_s) \\
= \sum_{s=1}^{S} \sum_{m=1}^{M} c^s_m I(x_i \in R^s_m),
\]

(3.12)

(3.13)

\( R^s_m \) marking the \( m \)th terminal node of the \( s \)th tree and \( c^s_m \) correspondingly (Hastie et al., 2009).

3.2 Support Vector Machine

Support vector machine (henceforth SVM) of Vapnik (1996) augmented with kernel functions provides an appropriate tool for taking advantage of the specific data structure. With kernel functions at its centre, SVM is built on the similarity between the data points and the way it operates is discussed in this section.
The way of fitting a kernel-based SVM is described in the following. First, consider the regression function

\[ f(x) = \sum_{k=0}^{K} \omega_k h_k(x) \tag{3.14} \]

taking inputs \( h_k(x) \), that is, basis expansion functions enlarging the feature space. Then

\[ H(\omega) = \sum_{i=1}^{n} V_\zeta(y_i - f(x_i)) + \frac{\Lambda}{2} \| \omega \|^2, \tag{3.15} \]

is minimized with respect to \( \omega \), where \( V_\zeta(.) \) is the \( \zeta \)-insensitive loss

\[ V_\zeta(r_i) = \begin{cases} 0 & \text{if } |r_i| < \zeta, \\ |r_i| - \zeta & \text{otherwise}, \end{cases} \tag{3.16} \]

with \( r_i = y_i - f(x_i) \) measuring residuals only outside of the \( \zeta \)-range. Hence \( \zeta \) is a tuning parameter as well as the penalty (\( \Lambda \)), suggested by Thodoros Evgeniou et al. (1999), to be imposed on the estimated parameters. Solving (3.15) (see Hastie et al. (2009)) results in

\[
\hat{f}(x) = \sum_{i=1}^{n} \hat{\kappa}_i K(x, x_i) \\
= \begin{bmatrix} K(x, x_1) & K(x, x_2) & K(x, x_3) & \ldots & K(x, x_n) \end{bmatrix} \begin{bmatrix} \kappa_1 \\ \kappa_2 \\ \kappa_3 \\ \vdots \\ \kappa_n \end{bmatrix}, \tag{3.17} \]

such that the prediction of \( y \) is

\[
\hat{y} = K\hat{\kappa}, \tag{3.19} \]

where \( K \) is a ‘modified’ version of the Gram matrix, such that the inner product elements of the Gram is replaced with the kernel distance measure\(^2\) of the observation.

\(^{2}\)Note that the inner product is also a similarity measure (linear kernel) however, the kernel used in (3.20) is a more general form.
pairs:

\[
K = \begin{bmatrix}
K(x_1, x_1) & K(x_1, x_2) & K(x_1, x_3) & \ldots & K(x_1, x_n) \\
K(x_2, x_1) & K(x_2, x_2) & K(x_2, x_3) & \ldots & K(x_2, x_n) \\
K(x_3, x_1) & K(x_3, x_2) & K(x_3, x_3) & \ldots & K(x_3, x_n) \\
\vdots & \vdots & \vdots & \ddots & \vdots \\
K(x_n, x_1) & K(x_n, x_2) & K(x_n, x_3) & \ldots & K(x_n, x_n)
\end{bmatrix}.
\] (3.20)

The Gaussian kernel function is used henceforward, with the inputs \(x_i\) and \(x_i'\), of the form

\[
K(x_i, x_i') = \exp\left(\frac{-\|x_i - x_i'\|^2}{2\sigma^2}\right),
\] (3.21)

where the squared Euclidean distance between \(x_i\) and \(x_i'\) is

\[
\|x_i - x_i'\|^2 = (x_{i,1} - x_{i',1})^2 + (x_{i,2} - x_{i',2})^2 + \ldots + (x_{i,N} - x_{i',N})^2 \] (3.22)

and the kernel scale parameter \(\sigma\) is a tuning parameter of the SVM regression.

### 3.3 Nonlinear least squares

Let \(f_{NLS}(X_i)\), nonlinear in \(X_i\), be the function in (3.1). Then one intends to minimize the squared error loss

\[
\hat{f}_{NLS}(x_i) = \arg\min_{f(x_i)} \sum_{i=1}^{n} (y_i - f(x_i))^2.
\] (3.23)

However, before such an optimization could be carried out, it is necessary that the structure of \(f_{NLS}(.)\) – the way the function handles its input \(x\) – be determined. That is, the parameters of the function need to be prefixed. Therefore Equation (3.23) converts to

\[
\hat{\Theta} = \arg\min_{\Theta} \sum_{i=1}^{n} (y_i - f_{NLS}(x_i, \Theta))^2
\] (3.24)

with \(\Theta\) marking the parameters.
Deriving Equation (3.24) with respect to the elements of parameter set $\Theta$, as the first-order condition, results in the score vector:

$$
\frac{\partial \hat{\Theta}}{\partial \Theta_1} = -2 \sum_{i=1}^{n} (y_i - f(x_i, \Theta)) \frac{\partial f(x_i, \Theta)}{\partial \Theta_1} = 0 \tag{3.25}
$$

$$
\frac{\partial \hat{\Theta}}{\partial \Theta_2} = -2 \sum_{i=1}^{n} (y_i - f(x_i, \Theta)) \frac{\partial f(x_i, \Theta)}{\partial \Theta_2} = 0 \tag{3.26}
$$

$$
\vdots \tag{3.27}
$$

$$
\frac{\partial \hat{\Theta}}{\partial \Theta_l} = -2 \sum_{i=1}^{n} (y_i - f(x_i, \Theta)) \frac{\partial f(x_i, \Theta)}{\partial \Theta_l} = 0, \tag{3.28}
$$

where $l$ is the number of parameters in $\Theta$, the solution to which is found with iteration process [Seber and Wild (2003), Cameron and Trivedi (2005)] using the Levenberg-Marquardt algorithm [Levenberg (1944), Marquardt (1963)]. If the second-order conditions\(^3\) are also met, the optimal solution is obtained.

### 3.4 Poisson regression

Whereas the NLS fits the model minimizing the squared error loss, the approaches in this and the next section (Section 3.5) acquire the solution with assumptions of distribution of the dependent variable. Nevertheless, these models also seek the conditional expected value of $y$ (see (3.1)) and prefixing the function structure is also an essential step. In this section, one of the distribution-based methods is presented, namely the Poisson regression approach [Greene (2003), Wooldridge (2010)].

Consider the expected value of variable $y$,

$$
E(Y) = \mu. \tag{3.29}
$$

Hence $y$ is drawn from the Poisson distribution with probability mass function

$$
P(Y = y_i) = \frac{\lambda^{y_i}}{y_i!} e^{-\lambda} \tag{3.30}
$$

and the mean of this distribution is

$$
E(Y) = \lambda, \tag{3.31}
$$

\(^3\)positive definite Hessian
it is easily seen that

\[
P(Y = y_i) = \frac{\lambda^y}{y!} e^{-\lambda} = \frac{E(Y|^y)}{y!} e^{-E(Y)} = \mu^y e^{-\mu}.
\]

(3.32)

Why is the transformation in Equation (3.32) important? Because being given the expected value ($\mu = \lambda$), one can perfectly describe the Poisson distribution and the probability mass function $P(Y = y_i)$. But even so, the case is exactly the reverse here: one has the observed values and distribution of $y$ and is in search of the expected value $E(Y)$ in the form $E(Y | X)$. The question arises: is it feasible to estimate the conditional expected value $E(Y | X)$ given the observed values $(x, y)$? The answer is yes and the technique with which it is manageable is called maximum likelihood estimation of Sir R. A. Fisher (summarized by Aldrich (1997))

The core idea of this approach is to maximize the likelihood of the observed sample distribution with respect to the function $f_P(X)$, for which

\[E(Y | X) = e^{f_P(X, \Theta)}\]

(3.33)

with $\Theta$ marking prefixed parameters. The exponential function is chosen to fend off negative values of $E(.)$. Formally, consider first the conditional probability of one sample element $y_i$ (for unconditional probability refer to Equation (3.32)):

\[
P(Y = y_i | x_i, \Theta) = \frac{E(Y | x_i)^y}{y!} e^{-E(Y|x_i)} = \frac{e^{f_P(x_i, \Theta)^y}}{y!} e^{-e^{f_P(x_i, \Theta)}}.
\]

(3.34)

(3.35)

Accordingly, the probability of the whole sample $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ is given by

\[
P(y_1, y_2, \ldots, y_n | x_1, x_2, \ldots x_n, \Theta) = \prod_{i=1}^{n} \frac{e^{f_P(x_i, \Theta)^y}}{y!} e^{-e^{f_P(x_i, \Theta)}} = \prod_{i=1}^{n} \frac{e^{y_i f_P(x_i, \Theta)}}{y!} e^{-e^{f_P(x_i, \Theta)}}
\]

(3.36)

(3.37)
Then the likelihood function $L$ is the expression

$$L(\Theta \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) = \prod_{i=1}^{n} \frac{e^{y_i f_P(x_i, \Theta)}}{y_i!} e^{-e^{f_P(x_i, \Theta)}},$$  \(3.38\)

which is to be maximized with respect to $f_P(.), \text{ that is, to } \Theta$:

$$\max L(\Theta \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) = \max_{\Theta} \prod_{i=1}^{n} \frac{e^{y_i f_P(x_i, \Theta)}}{y_i!} e^{-e^{f_P(x_i, \Theta)}}. \quad (3.39)$$

The expression (3.39) is multiplicative, however, it can be linearized by taking the natural logarithm

$$\ell(\Theta \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) = \log L(\Theta \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n), \quad (3.40)$$

which is a positive monotonic transformation not altering the solution to the problem, leading to the log-likelihood function $\ell(.)$ to be minimized

$$\max \ell(\Theta \mid .) = \max_{\Theta} \sum_{i=1}^{n} \left[ \log \left( e^{y_i f_P(x_i, \Theta)} \right) + \log \left( e^{-e^{f_P(x_i, \Theta)}} \right) - \log \left( y_i! \right) \right]. \quad (3.41)$$

Using the properties of the logarithm and because the $\log(y_i!) \text{ term does not include } \Theta$, Equation (3.41) can be rephrased as

$$\max \ell(\Theta \mid .) = \max_{\Theta} \sum_{i=1}^{n} \left[ y_i f_P(x_i, \Theta) - e^{f_P(x_i, \Theta)} \right], \quad (3.42)$$

only for the partial derivatives to be taken with respect to the element of the parameter set $\Theta$:

$$\frac{\partial \ell(\Theta \mid .)}{\partial \Theta_1} = 0$$

$$\frac{\partial \ell(\Theta \mid .)}{\partial \Theta_2} = 0$$

$$\vdots$$

$$\frac{\partial \ell(\Theta \mid .)}{\partial \Theta_l} = 0 \quad (3.43)$$

and gaining the solutions in a similar fashion to that of NLS if the second-order conditions\(^4\) are met.

\(^4\)negative definite Hessian
3.5 Negative binomial

In this section, the derivation of the negative binomial estimator, which is much the same as that of the Poisson approach (Section 3.4), is outlined. To start with, a negative binomial distribution is assumed for $y$ to be drawn from which, and then by inserting the conditional expected value $E(Y \mid X)$ into the probability mass function, the fitting procedure is, again, the maximum likelihood.

Initially, consider the unconditional mean of $y$

$$E(Y) = \mu \quad (3.44)$$

and the probability mass function of the negative binomial distribution

$$P(Y = y_i) = \left(\frac{y_i + \rho - 1}{y_i}\right) p^y (1 - p)^{\rho - 1} \quad (3.45)$$

from which it is drawn from. Then

$$\mu = \frac{pp\rho}{1 - p}, \quad (3.46)$$

or equivalently

$$p = \frac{\mu}{\mu + \rho} \quad (3.47)$$

$$= \frac{E(Y)}{E(Y) + \rho}. \quad (3.48)$$

Once it is conditioned on $X$ and $\Theta$ as

$$E(Y \mid X) = e^{f_{Nb}(x, \Theta)}, \quad (3.49)$$

with $\Theta$ standing for the prefixed parameter structure and using $\exp(.)$ to avoid negative values, the probability mass function (3.45) of a single sample value $y_i$ given $x_i, \Theta$ and $\rho$ is

$$P(Y = y_i \mid x_i, \Theta, \rho) = \left(\frac{y_i + \rho + 1}{y_i}\right) \left(\frac{E(Y \mid x_i)}{E(Y \mid x_i) + \rho}\right)^y \left(\frac{\rho}{E(Y \mid x_i) + \rho}\right)^{\rho} \quad (3.50)$$

$$= \left(\frac{y_i + \rho + 1}{y_i}\right) \left(\frac{e^{f_{Nb}(x_i, \Theta)}}{e^{f_{Nb}(x_i, \Theta)} + \rho}\right)^y \left(\frac{\rho}{e^{f_{Nb}(x_i, \Theta)} + \rho}\right)^{\rho}. \quad (3.51)$$

Note that as opposed to the case of the Poisson regression, there is an unknown parameter ($\rho$) of the mass function, hence $\rho$ is to estimated alongside with $\Theta$. Therefore, the probability of the whole sample $(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$ and thus the likelihood is:

$$P(y_1, y_2, \ldots, y_n \mid x_1, x_2, \ldots, x_n, \Theta, \rho) = \mathcal{L}(\Theta, \rho \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) \quad (3.52)$$
Substituting the combination \((y_i + \rho + 1) \mathcal{C}_i = (y_i + \rho + 1)\) in (3.51) with Gamma functions results is

\[
\mathcal{L}(\Theta, \rho \mid \cdot) = \prod_{i=1}^{n} \frac{\Gamma(y_i + \rho)}{\Gamma(\rho) \Gamma(y_i + 1)} \left( \frac{e^{f_{Nk}(x_i, \Theta)}}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right)^{y_i} \left( \frac{\rho}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right)^{\rho} .
\] (3.53)

As is the case in Section 3.4, the purpose is maximizing \(\mathcal{L}\)

\[
\max_{\Theta, \rho} \mathcal{L}(\Theta, \rho \mid \cdot) = \max_{\Theta, \rho} \prod_{i=1}^{n} \frac{\Gamma(y_i + \rho)}{\Gamma(\rho) \Gamma(y_i + 1)} \left( \frac{e^{f_{Nk}(x_i, \Theta)}}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right)^{y_i} \left( \frac{\rho}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right)^{\rho} .
\] (3.54)

Again, one linearizes \(\mathcal{L}\) by taking the natural logarithm

\[
\ell(\Theta, \rho \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) = \log \mathcal{L}(\Theta, \rho \mid y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n),
\] (3.55)

and then obtains

\[
\max_{\Theta, \rho} \ell(\Theta, \rho \mid \cdot) = \max_{\Theta, \rho} \sum_{i=1}^{n} \left[ \log(\Gamma(y_i + \rho)) - \log(\Gamma(\rho)) - \log(\Gamma(y_i + 1)) + \ldots \\
\ldots + y_i \log \left( \frac{e^{f_{Nk}(x_i, \Theta)}}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right) + \ldots \\
\ldots + \rho \log \left( \frac{\rho}{e^{f_{Nk}(x_i, \Theta)} + \rho} \right) \right] (3.56)
\]

\[
= \max_{\Theta, \rho} \sum_{i=1}^{n} \left[ \log(\Gamma(y_i + \rho)) - \log(\Gamma(\rho)) - \log(\Gamma(y_i + 1)) + \ldots \\
\ldots + y_i \log \left( e^{f_{Nk}(x_i, \Theta)} \right) - y_i \log \left( e^{f_{Nk}(x_i, \Theta)} + \rho \right) + \ldots \\
\ldots + \rho \log(\rho) - \rho \log \left( e^{f_{Nk}(x_i, \Theta)} + \rho \right) \right]. (3.57)
\]

\(^5\Gamma(a) = (a - 1)!\)
Dropping the term not including the parameters with respect to which the maximum likelihood is sought, results in

\[
\max_{\Theta, \rho} \ell(\Theta, \rho) = \max_{\Theta, \rho} \sum_{i=1}^{n} \left[ \log(\Gamma(y_i + \rho)) - \log(\Gamma(\rho)) + \ldots \right. \\
\left. \ldots + y_i \log(e^{f_{N\Theta}(x_i, \Theta)}) - y_i \log(e^{f_{N\Theta}(x_i, \Theta) + \rho}) + \ldots \right. \\
\left. \ldots + \rho \log(\rho) - \rho \log(e^{f_{N\Theta}(x_i, \Theta) + \rho}) \right]. \\
(3.58)
\]

Finally, the partial derivatives are taken as the first-order conditions of maxima:

\[
\frac{\partial \ell(\Theta, \rho)}{\partial \Theta_1} = 0 \\
\frac{\partial \ell(\Theta, \rho)}{\partial \Theta_2} = 0 \\
\vdots \\
\frac{\partial \ell(\Theta, \rho)}{\partial \Theta_l} = 0 \\
\frac{\partial \ell(\Theta, \rho)}{\partial \rho} = 0 \\
(3.59)
\]

and if the second-order ones\(^6\) are met, a maxima is found. However, in the case of the NLS, the Poisson and the negative binomial approach, it is important to be aware of the fact that there might be multiple minima/maxima, so the solutions are sensitive to the initial guesses and are to be treated with reservations.

\(^6\)negative definite Hessian
Chapter 4: Models

In this chapter, the process of fitting the models in Chapter 3 to the scarlatina time series data is detailed. Consequently, the author first presents the boosting trees (Section 3.1) and the support vector regression machine (Section 3.2), and afterwards the traditional models, namely the NLS (Section 3.3), Poisson (Section 3.4), negative binomial (Section 3.5), are considered.

4.1 Sample partitioning

As it is mentioned in Chapter 2 the data consists of \( n = 916 \) observations, which is partitioned into two parts in order to test the models on an independent dataset. The first 816 observations form the training set, leaving \( 916 - 816 = 100 \) datapoints for the testing. In this chapter, every reference to the data, variables, etc. is strictly related to the training set, which is used for fitting the models. The testing is addressed in Chapter 5.

4.2 Variables

Before the modelling, it is needed to declare the variables used in the process. First of all, the phenomenon to be predicted is the number of scarlet fever cases per week, so

\[
y_i = \text{Scarlatina}_i,
\]

with \( i \) marking the \( i \)th week. Even though the ML techniques might not establish as rigid assumptions of normality as other models do, taken is the logarithm of the scarlatina cases, ending up with the 'indirect' dependent variable

\[
\log(y_i) = \log(\text{Scarlatina}_i),
\]

\[\text{in Section ?? and ?? the logarithm is used hence all the error measures are in } \log(\text{Scarlatina})\]
being closer to the bell curve. The reason for this transformation is that however much the ML techniques might be insensitive to normality, they are sensible to the structure of the data. Consequently, noise in the data can distort the results, however, by taking the logarithm the data and the noise is compressed, avoiding this phenomenon. Shown are the histograms of the two versions in Figure 4.1 and the time series plots in Figure 4.2.

**Figure 4.1:** Histograms of Scarlatina and that of its logarithm

![Histogram of Scarlatina and its logarithm](image)

The group of the predictor variables is partly obvious, however, some predictors are of interests during selection. Since the data are time series, the time variable $t$ is a clear choice, ranging from 0 to 816 marking the weeks, embodying the pass of time, thus the potential underlying effects of medical development and other untraced transactions. It is a common method for time series analysis to use the lagged values of the dependent variable, on the other hand it is less clear how many lagged terms should be put in use. Month dummies grasping seasonality may also be a reasonable choice.

The final set of predictors is a result of model selection, detailed in 4.3 nonetheless, the initial set is defined as

$$X = [t, \log(\text{Scarlatina}_{-1}), \log(\text{Scarlatina}_{-2}), \text{Mdummy}], \quad (4.1)$$
where Scarlatina\(_{-1,2}\) represents the first and the second lagged values and Mdummy is the month dummy variables. Figure 4.3 shows the relationship between the dependent and the predictor variables.

Figure 4.3: Plotmatrix of the variables

![Plotmatrix of the variables](image)
4.3 Selection and tuning

As is true of the most ML techniques, boosting trees and SVM fitting requires fine tuning of some parameters, in addition, the set of predictors is also a source of uncertainty. Hence variable selection and tuning is an important part of the training. In this section, the procedure of carrying out such steps is examined.

4.3.1 Cross-validation

A prevalent way of selection and tuning is $k$-fold cross-validation, within the confines of which one randomly omits observations from the training sample – $1/k$th of the whole sample size at a time, repeated $k$ times – making it feasible to predict the generalization error, and thereby to draw conclusions as of the tuning and variable structure of the given model [Picard and Cook (1984)]. Despite Hastie et al. (2009)’s warning of applying cross-validation to time series data, 10-fold cross-validation, lacking of better alternative, is applied to the ML techniques in the models.

4.3.2 Loss measure

Having acquired the cross-validation approach, the only question remaining unanswered is that of the actual error measure. Prior to describing it, it is substantial to emphasize that the loss measure in this subsection is not utilized in the fitting algorithm of the models, the only purpose it serves is to assess the models as against each other.

An off-the-shelf loss assessment function is that of Huber (1964), which unifies the squared error loss for small residuals and absolute loss for the bigger ones, offsetting the vulnerability of squared error loss to big residuals. The Huber loss ($L_H$) is as follows:

$$L_H(r_i) = \begin{cases} \frac{1}{2}r_i^2 & \text{if } |r_i| \leq \delta, \\ \delta|r_i| - \frac{1}{2}\delta^2 & \text{otherwise}, \end{cases} \quad (4.2)$$

where $r_i = y_i - \hat{y}_i$, and $\delta$ is a parameter to set. High value of $\delta$ pushes up the limit under which the squared error occurs, thus tends to be less robust enlarging bigger residuals, whilst small $\delta$ values the loss is absolute in more cases. However, properly allocating $\delta$ is hard and the author’s attempts in doing so proved to be insufficient, so

---

2because randomly aborting the continuity of the time series may obscure the natural process

3For that used for boosting trees refer to Section 4.4 and see Equation (3.16).
instead of relying on the robustness of Huber, the attention is focused on the more traditional, yet robust, mean percentage error (MAPE):

\[ MAPE = 100 \frac{1}{n} \sum_{i=1}^{N} \frac{|y_i - \hat{y}_i|}{y_i}, \] (4.3)

which is used for comparison from now on. Note, however, that for fine tuning weighted MAPE\(^4\) is applied, because of software requisites, thus the results do differ from the ones in Chapter 5 where no weights are assigned.

4.4 Boosting trees

During the boosting trees fitting the objectives are

1. to define the final set of the predictors

2. to fine tune the parameters (the value of \( M \) in \( R_M \) of the weak learners)

3. to declare the number of weak trees (in the range from 1 to 100\(^5\)).

These steps might be performed simultaneously based on at least three-dimensional space of models, nevertheless, this would be computationally demanding, so the chosen procedure is the step-by-step fashion.

4.4.1 Predictor selection

Firstly, predictor selection is addressed with 100 weak learners and medium losses of the different number of splits (= 1, 2, 3, ..., 7, leading to \( m = 2, 3, 4, ..., 8 \) terminal nodes) in the weak trees, resulting in three models:

\(^4\) with each observation weighted with \( w_i = \frac{1}{n} \)

\(^5\) As it seen in Figure 4.4 it is useless to check a broader range.
Table 4.1: Boosting trees models of three different predictor group.

<table>
<thead>
<tr>
<th></th>
<th>$X$</th>
<th>MAPE = $\text{median}_m(.)$</th>
</tr>
</thead>
<tbody>
<tr>
<td>Full</td>
<td>$t, \log(y_1), \log(y_2), \text{Mdummy}$</td>
<td>$\text{MAPE}(T_{\text{full}}(X, \Xi(m)))$</td>
</tr>
<tr>
<td>Restricted 1</td>
<td>$t, \log(y_1)$</td>
<td>$\text{MAPE}(T_{\text{res1}}(X, \Xi(m)))$</td>
</tr>
<tr>
<td>Restricted 2</td>
<td>$t$</td>
<td>$\text{MAPE}(T_{\text{res2}}(X, \Xi(m)))$</td>
</tr>
</tbody>
</table>

**Notes:** $\text{median}_m(.)$ stands for taking the median over the different terminal nodes number; for eg. $\text{median}_m(\text{MAPE}(T_{\text{full}}(X, \Xi(m))))$ results in a vector of 100x1 with each $j$th row representing the median MAPE given the model containing $j$ piece of weak learners. Note that here $T$ represent a whole ensemble model, not a single tree. The index $\text{full}$, $\text{res1}$, $\text{res2}$ stands for the Full, the Restricted 1 and the Restricted 2 ensembles, respectively.

Figure 4.4 illustrates the weak tree-wise cumulative MAPE of the above models. One can see that whilst the 'Restricted 2' model containing only the time $t$ variable seems to underfit the data, and the 'Full', on the contrary, is more likely to overfit, 'Restricted 1' appears to be a reasonable choice. Hence the models 'Restricted 2' and 'Full' are dropped in favour of the 'Restricted 1', which is thought of as the boosting trees model from now on.

Figure 4.4: Predictor selection
4.4.2 Tuning the number of terminal nodes

As is discussed in Subsection 3.1.2, boosting trees is a committee of weak trees, each of which is characterized by \( \Xi_s = \{ c_{s,m}, R_{s,m} \}_{m=1}^{M} \) (with \( s \) being the \( s \)th weak tree). The number of terminal nodes \( M \) is the same in each additional tree and is of interest in this section.

Figure 4.5 shows two plots about the MAPE loss given different number of splits\(^6\) from 1 to 7. The upper plot shows the cumulative loss as a function of number of weak trees, however, it is not very informative, since the errors vary with the number of learners, unlike the one at the bottom, which plots the median and mean MAPE error - aggregated over the number of trees\(^7\) to block out that volatility. A conclusion can be drawn as follows: the stump tree (splits= 1, \( M = 2 \)) is slightly outperformed only by the 6-split-tree\(^8\) judging by the median error; but even so it is unwise to prefer the latter to the stump, because the median error is only slightly better and the 6-split-tree may as well overfit. Accordingly, the best boosting tree model is 'Full' based on stumps.

---

\(^6\)Terminal nodes number \( M \) is derived by adding 1 to the number of splits

\(^7\)median\(_S\)(MAPE(\( T_{full}(X, \Xi(S)) \)))

\(^8\)Median MAPE of the stump= 2.3360 \cdot 10^{-6}; median MAPE of the 6-split-tree= 2.0750 \cdot 10^{-6}
4.4.3 Number of weak learners

Selecting the correct number of weak learners cumulated in the ensemble is critical to avert overfitting. In doing so, the author turns to an arbitrary solution: examined are the marginal changes in the cumulative MAPE errors as more and more weak trees are included in the model such that when the improvement is negligible a cut is made at that amount of weak trees. Formally, one seeks the solution to the problem

\[ S^* = \arg \min_s \text{MAPE}(f_s(X)) - \text{MAPE}(f_{s-1}(X)) > \text{constant}, \]  

\[ (4.4) \]

using the notation of Equation (3.11). For constant arbitrarily chosen is the value $10^{-7}$, leading to and optimal weak tree number of 19, which is, inferring from Figure 4.1 and 4.5 seems reasonable, because at $s = 19$ the loss breaks down and stays relatively steady afterwards.

4.4.4 Final ensemble

To summarize, the final tree ensemble is the one with

- predictors: $t, \log(y-1)$;
- weak trees consisting of two terminal nodes (stumps) ($M^* = 2$);
- number of weak trees included: 19 ($S^* = 19$).

Note, however, that the learning rate $\eta$ (see Equation (3.10)) is chosen to be constant 0.1, shrinking the weight of cumulative predictions to 10%, not being subject to fine tuning.

4.5 Support vector machine

The general description of support vector machine is found in Section 3.2, here it is reviewed how the method is applied to the concrete data. Similarly to the boosting trees, the problem of declaring the following requisites rise:

1. the group of predictors, $X$
2. the loss property, $\zeta$
3. the imposed penalty, $\Lambda$
4. the kernel property, $\sigma$.

It would be attractive for the set of predictors to remain the same as it is in the 'Restricted 1' tree ensemble (see Subsection 4.4.1 or Table 4.1), for tractability considerations; however, this is not the case. Since the SVM prediction relies on the distance between the observed and the query points, the usage of the $t$ variable, which linearly increases with the passing of time, essentially results in a wrong forecast for a distant-in-time query point. Consequently, variable $t$ is omitted, and the second-order lagged term is included, such that

$$X = [\log(y_{-1}), \log(y_{-2})].$$

The loss property (see Equation (3.16)), namely half the width of the $\zeta$-insensitive range, takes up a default value\(^9\) provided by MATLAB, and is not subject to tuning for resource considerations.

Therefore the focus of tuning is on the penalty term and the kernel scale (for details of those see Equation (3.15) and (3.16) respectively). The implementation of the tuning takes place in a two-dimensional model space with one dimension being the penalty and the other being the kernel scale. Then one scans through all the combinations of the coordinates examining the occurring MAPE loss of the models defined by the coordinates – a so-called grid search procedure.

The reciprocal\(^10\) values $(\frac{1}{\Lambda})$ of the first coordinate, penalty, ranges from 0.01 to 10 with uniform step size of 0.5, with small values imposing bigger penalty on harder fitting and the other way around. The kernel scale parameter $\sigma$ is also in $[0.01, 10]$ with 0.5 uniform step size; here, small values of $\sigma$ ’enlarge’ the distance between two observations in the kernel and result in harder edges, whilst big $\sigma$ values favour softer squashing of the distance. Subsequently, one ends up with a model grid of size 20x20. Figure 4.6 shows the MAPE loss as the function the grid. It is clearly seen that the MAPE is less sensitive to the value of $\sigma$ – except small values of $\sigma$ – than to that of the inverse penalty $(\frac{1}{\Lambda})$. The figure suggests that the biggest generalization error starts decreasing towards $(\frac{1}{\Lambda}, \sigma) = (0, 10)$ coordinate only to reach its

\(^9\) $iqr(\log(Scarlatina)) / 1.349$, with $iqr$ marking the interquartile range

\(^{10}\) Operation with the reciprocal values is justified because one gives the penalty is such a form as the input of MATLAB.
minima at the coordinate pair \((\frac{1}{\Lambda}^*, \sigma^*) = (0.4351, 6.0100)\), where with small inverse penalty hard-fitting is avoided, and the 'medium' value of \(\sigma\) provides 'medium' edges.

Figure 4.6: SVM grid search

4.5.1 Final SVM

To conclude, as the result of the tuning, the best SVM is defined by

- predictors: \(\log(y_{-1}), \log(y_{-2})\);
- half the width of \(\zeta\)-insensitive loss: \(\zeta^* = \frac{iqr(\log(y))}{1.3490}\);
- inverse penalty term: \(\frac{1}{\Lambda}^* = 0.4351\);
- kernel scale: \(\sigma^* = 6.0100\).

4.6 Traditional models

In this section, the fitting of the traditional methods (see Section 3.3, 3.4, 3.5) to the scarlatina data is reported. As it is seen and emphasized throughout this paper, the aforementioned ML techniques are more flexible and, because of this, they require
fine tuning, which is done with cross-validation. The successive methods are more rigid, necessitating less guidance on their fitting. Thus the subsequent steps are called for in the fitting process of all the 'traditional' models:

1. defining the set of predictors, $X$
2. determining the function $f(X_i)$ in (3.1).

For consistent comparison with the boosting trees model, the set of predictors remains unchanged, that is,

$$X = [t, y_{-1}]$$

or, depending on the model

$$X = [t, \log(y_{-1})].$$

Yet there is one substantial difference, specifically that in this section not the natural logarithm of the dependent variable ($\log(y)$) is used as the dependent variable but directly the number of scarlet fever cases. The reason for this choice is that the pattern of $y$ is easier to model with functions used in this chapter than the pattern of $\log(y)$. In Chapter 5, where the testing is addressed, it is seen that these differences can be neutralized by the exponential function.

However, the function, which characterizes the relationship between the conditional expected value $E(Y \mid X)$ and the input space, is yet of interest. Again, for comparability, similar functions are chosen for the NLS, the Poisson and the negative binomial approach, and are detailed in Subsections 4.6.1, 4.6.2 and 4.6.3, respectively.

## 4.6.1 Nonlinear least squares

In this subsection, the problem of fitting NLS (see Section 3.3) to the scarlatina data is dealt with. If

$$E(Y \mid x_i, \Theta) = f_{NLS}(x_i, \Theta), \quad (4.5)$$

then with $\Theta = [\beta_{NLS}, \phi_{NLS}]$, $f_{NLS}(x_i, \beta_{NLS}, \phi_{NLS})$ is selected to satisfy

$$f_{NLS}(x_i, \beta_{NLS}, \phi_{NLS}) = \beta_{0}^{NLS} + \beta_{1}^{NLS} \cdot \sin \left( \frac{t_i}{\beta_{2}^{NLS} - \beta_{3}^{NLS}} \right) + \ldots$$

$$+ \beta_{4}^{NLS} \cdot \sin \left( \frac{t_i}{\beta_{5}^{NLS} - \beta_{6}^{NLS}} \right) + \phi_{NLS} \cdot y_{i-1}, \quad (4.6)$$
with $y_i$ marking the number of scarlet fever cases on the $i$th week, Scarlatina. The reason the sine function is chosen for is that, how Figure 2.1 illustrates, the scarlet fever cases follow a seasonal pattern, for the modelling of which sine is suitable. Accordingly, one sinusoidal term models low-frequency data and the other term is responsible for the high-frequency. $\beta_0^{NLS}$ is for vertical adjustment, whilst $\beta_1^{NLS}$ and $\beta_4^{NLS}$ are for the amplitude, $\beta_2^{NLS}$ and $\beta_5^{NLS}$ for the frequency, $\beta_3^{NLS}$ and $\beta_6^{NLS}$ for the horizontal adjustment.

Then the optimization in Equation (3.23) is

$$\hat{f}_{NLS}(x_i, \beta_{NLS}^{NLS}, \phi_{NLS}^{NLS}) = \arg \min_{f_{NLS}(\cdot)} \sum_{i=1}^{n} \left( y_i - f_{NLS}(x_i, \beta_{NLS}^{NLS}, \phi_{NLS}^{NLS}) \right)^2$$

that is to say, given $X$

$$[\hat{\beta}^{NLS}, \hat{\phi}^{NLS}] = \min_{\beta_{NLS}^{NLS}, \phi_{NLS}^{NLS}} \sum_{i=1}^{n} \left[ y_i - \left( \beta_0^{NLS} + \beta_1^{NLS} \cdot \sin \left( \frac{t_i}{\beta_2^{NLS}} - \beta_3^{NLS} \right) \right) + \ldots + \beta_4^{NLS} \cdot \sin \left( \frac{t_i}{\beta_5^{NLS}} - \beta_6^{NLS} \right) + \phi_{NLS}^{NLS} \cdot y_{i-1} \right]^2.$$

Finally, the estimated elements of the parameter vector $\beta^{NLS}$ is obtained by solving Fisher’s score

$$\frac{\partial[\hat{\beta}^{NLS}, \hat{\phi}^{NLS}]}{\partial \beta_{1}^{NLS}} = -2 \sum_{i=1}^{n} (y_i - f(\cdot)) \frac{\partial f(\cdot)}{\partial \beta_{1}^{NLS}} = 0$$

$$\frac{\partial[\hat{\beta}^{NLS}, \hat{\phi}^{NLS}]}{\partial \beta_{2}^{NLS}} = -2 \sum_{i=1}^{n} (y_i - f(\cdot)) \frac{\partial f(\cdot)}{\partial \beta_{2}^{NLS}} = 0$$

$$\vdots$$

$$\frac{\partial[\hat{\beta}^{NLS}, \hat{\phi}^{NLS}]}{\partial \beta_{6}^{NLS}} = -2 \sum_{i=1}^{n} (y_i - f(\cdot)) \frac{\partial f(\cdot)}{\partial \beta_{6}^{NLS}} = 0$$

$$\frac{\partial[\hat{\beta}^{NLS}, \hat{\phi}^{NLS}]}{\partial \phi^{NLS}} = -2 \sum_{i=1}^{n} (y_i - f(\cdot)) \frac{\partial f(\cdot)}{\partial \phi^{NLS}} = 0$$

with respect to each element in $\beta^{NLS}$ and $\phi^{NLS}$. To solve the maximum likelihood optimization problem, the quasi-newton BFGS-algorithm ([Brödren (1970)]) is used. The results is found in Table B.1.
4.6.2 Poisson

In this subsection, detailed is the process of applying Poisson regression (see Section 3.4) to the scarlet fever data. First of all, let us consider Equation (3.33) such that

$$E(Y | x_i, \Theta) = e^{f_P(x_i, \Theta)}$$ (4.10)

so that with $\Theta = [\beta^P, \phi^P]$, where $P$ stands for Poisson, the log-likelihood to maximize is

$$\max_{\Theta} \ell(\Theta | y_1, y_2, \ldots, x_n) = \max_{\beta^P, \phi^P} \ell(\beta^P, \phi^P | y_1, y_2, \ldots, x_n).$$ (4.11)

Here, the same function structure is obtained as in Equation (4.6):

$$f_P(x_i, \beta^P, \phi^P) = \beta^P_0 + \beta^P_1 \cdot \sin \left( \frac{t_i}{\beta^P_2} - \beta^P_3 \right) + \ldots$$

$$\ldots + \beta^P_4 \cdot \sin \left( \frac{t_i}{\beta^P_5} - \beta^P_6 \right) + \phi^P \cdot \log(y_{i-1}),$$ (4.12)

with the only difference being the application of $\log(y_{i-1})$ instead of $y_{i-1}$, which is justified by the $\exp(.)$ in Equation (4.10). The log-likelihood maximization in Equation (3.42) is

$$\max_{\beta^P, \phi^P} \ell(\Theta | .) = \max_{\beta^P, \phi^P} \sum_{i=1}^{n} \left\{ y_i \cdot \left[ \beta^P_0 + \beta^P_1 \cdot \sin \left( \frac{t_i}{\beta^P_2} - \beta^P_3 \right) + \ldots \right. \right.$$}

$$\left. \ldots + \beta^P_4 \cdot \sin \left( \frac{t_i}{\beta^P_5} - \beta^P_6 \right) + \phi^P \cdot \log(y_{i-1}) \right] + \ldots$$

$$\ldots - e^{\beta^P_0 + \beta^P_1 \cdot \sin \left( \frac{t_i}{\beta^P_2} - \beta^P_3 \right) \ldots}$$

$$\ldots \cdot e^{\beta^P_4 \cdot \sin \left( \frac{t_i}{\beta^P_5} - \beta^P_6 \right) + \phi^P \cdot \log(y_{i-1})} \right\}.$$ (4.13)
Then the score vector is:

$$\frac{\partial \ell(\beta, \phi | \cdot)}{\partial \beta_1} = 0$$

$$\frac{\partial \ell(\beta, \phi | \cdot)}{\partial \beta_2} = 0$$

$$\vdots$$

$$\frac{\partial \ell(\beta, \phi | \cdot)}{\partial \beta_6} = 0$$

$$\frac{\partial \ell(\beta, \phi | \cdot)}{\partial \phi} = 0,$$

(4.14)

the BFGS-solution to which yields the estimated parameters $\hat{\beta}^P$ and $\hat{\phi}^P$, found in Table B.1.

4.6.3 Negative binomial regression

In this subsection, the problem of applying the negative binomial regression (See Section 3.5) to the time series data is outlined. To begin, consider the conditional expected value,

$$E(Y | x_i) = e^{f_{\text{Nb}}(x_i, \Theta)},$$

(4.15)

then let $\Theta$ be equal to $[\beta^{Nb}, \phi^{Nb}]$, where $Nb$ is for negative binomial, to express the log-likelihood to be maximized

$$\max_{\Theta, \rho} \ell(\Theta, \rho | y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n) =$$

$$\max_{\beta^{Nb}, \phi^{Nb}, \rho} \ell(\beta^{Nb}, \phi^{Nb}, \rho | y_1, y_2, \ldots, y_n, x_1, x_2, \ldots, x_n).$$

(4.16)

The structure of $f_{Nb}(\cdot)$, again, is the same as the one seen in case of the Poisson:

$$f_{Nb}(x_i, \beta^{Nb}, \phi^{Nb}) = \beta_0^{Nb} + \beta_1^{Nb} \cdot \sin \left( \frac{t_i}{\beta_2^{Nb}} - \beta_3^{Nb} \right) + \ldots$$

$$\ldots + \beta_4^{Nb} \cdot \sin \left( \frac{t_i}{\beta_5^{Nb}} - \beta_6^{Nb} \right) + \phi^{Nb} \cdot \log(y_{i-1}),$$

(4.17)
such that the log-likelihood, substituting $f_{Nb}(.)$ into Equation (3.58), is

$$
\max_{\beta_{Nb}, \phi_{Nb}, \rho} \ell(\beta_{Nb}, \phi_{Nb}, \rho | .) = \max_{\beta_{Nb}, \phi_{Nb}, \rho} \sum_{i=1}^{n} \left\{ \log(\Gamma(y_i + \rho)) - \log(\Gamma(\rho)) + \ldots + y_i \cdot \log \left[ e^{\beta_{Nb}^0 y_i + \beta_{Nb}^1 \sin\left( \frac{\pi}{2} - \beta_{Nb}^3 \right) + \beta_{Nb}^4 \sin\left( \frac{\pi}{6} - \beta_{Nb}^6 \right) + \phi_{Nb} \log(y_i - 1) + \rho \right] ight\}.
$$

(4.18)

However, one has to omit the $\log(\Gamma(y_i + \rho))$ term in order to maximize with the score equation system,

$$
\frac{\partial \ell(\beta_{Nb}, \phi_{Nb}, \rho | .)}{\partial \beta_{Nb}^1} = 0
$$

$$
\frac{\partial \ell(\beta_{Nb}, \phi_{Nb}, \rho | .)}{\partial \beta_{Nb}^2} = 0
$$

$$
\vdots
$$

$$
\frac{\partial \ell(\beta_{Nb}, \phi_{Nb}, \rho | .)}{\partial \beta_{Nb}^6} = 0
$$

$$
\frac{\partial \ell(\beta_{Nb}, \phi_{Nb}, \rho | .)}{\partial \phi_{Nb}} = 0
$$

$$
\frac{\partial \ell(\beta_{Nb}, \phi_{Nb}, \rho | .)}{\partial \rho} = 0,
$$

(4.19)

because the $\Gamma(y_i + \rho)$ expression is prohibitively large to be computed. Table B.1 contains the parameters estimated with BFGS-algorithm, $\beta_{Nb}$, $\phi_{Nb}$ and $\rho$, obtained with the $\Gamma(.)$ function omitted.
4.7 Discussion

Having finalized the parameters and predictors of the machine learning (boosting trees and support vector regression vehicles) and the traditional models (NLS, Poisson and negative binomial regression), one then fits the models to the whole training set without cross-validation.

Figure 4.7: Training prediction of the ML models

Figure 4.7 shows the results of the machine learning fitting, and 4.8 shows that of the traditional approaches. The tree ensemble very heavily underfits with cutting down high extremes, whilst the SVM performs a bit better, but yet underfits. As far as the traditional models are concerned, the NLS and the Poisson regression seems suitable – possibly singing overfitting –, whereas the negative binomial suggests some underfitting. For qualitative details see Table B.2, where the training set errors are shown in parenthesis.

Figure 4.9 shows how well the Poisson and the negative binomial maximum likelihood fitting estimates the observed distribution of scarlatina cases. It is clearly seen that the Poisson performs better, maybe because the enforced dropping of the \( \Gamma(.) \) term from the negative binomial regression (see Subsection 4.6.3).
Figure 4.8: Training prediction of the traditional models

Figure 4.9: Estimated distribution with maximum likelihood
To conclude, as is seen in Chapter 5, the final step to assess the performance of all the models in Chapter 3 is to test them on an independent data set, which may well reflect the surmises as of over-, and underfitting.
Chapter 5: Test

This chapter is dedicated to testing the models in Chapter 4 using an independent data set consisting of 100 observations from 11st November 2013 to 5th October 2015. In doing so, the models are used for predicting the scarlet fever cases in the range considered, and the performance of them is measured with different types of losses. Namely, there are six loss functions put in to use:

\[ ME \text{ (mean error)} = \frac{1}{n} \sum_{i=1}^{n} y_i - \hat{y}_i \quad (5.1) \]

\[ MPE \text{ (mean percentage error)} = 100 \frac{1}{n} \sum_{i=1}^{n} \frac{y_i - \hat{y}_i}{y_i} \quad (5.2) \]

\[ MAE \text{ (mean absolute error)} = \frac{1}{n} \sum_{i=1}^{n} |y_i - \hat{y}_i| \quad (5.3) \]

\[ MAPE \text{ (mean absolute percentage error)} = 100 \frac{1}{n} \sum_{i=1}^{n} \left| \frac{y_i - \hat{y}_i}{y_i} \right| \quad (5.4) \]

\[ MSE \text{ (mean squared error)} = \frac{1}{n} \sum_{i=1}^{n} (y_i - \hat{y}_i)^2 \quad (5.5) \]

\[ RMSE \text{ (root mean squared error)} = \sqrt{MSE} \quad (5.6) \]

where \( \hat{y}_i \) marks the fitted values, and since the testing is accomplished on the test set, \( n \) here denotes the number of observations in the test set, that is, \( n = 100 \).

Table B.2 summerizes the calculated values of the above errors in the test set, as well as in the training set. Judging by the test error losses, boosting trees emerges as the winner among the machine learning techniques, whilst, all models taken into

\footnote{Note that, as opposed to the case of tuning, no weights are assigned here to the observations.}
account, the Poisson regression outperforms each of them.

Figure 5.1 and 5.2 demonstrates the goodness of fit of the machine learning and the traditional models, respectively. SVM seems relatively steady around the mean, not predicting outliers, though insufficiently following the pattern of the series. On the other hand, the boosting trees systematically remains under the level of the actual scarlet fever cases, but yet manages the predict the underlying process in a more satisfying fashion. However, it is obvious that none the machine learning techniques bears a comparison with the traditional models in forecasting the illness.

Figure 5.1: Test prediction with boosting trees and SMV
Figure 5.2: Test prediction with traditional models
Chapter 6: Conclusion

In this paper, some regression methods were applied to the time series data of scarlet fever cases in Hungary. Machine learning techniques (boosting trees, support vector machine) and traditional models (NLS, Poisson and negative binomial) were addressed in order to determine which one of them proves to be the most excellent at forecasting an independent data set.

Our results in Chapter 5 suggest that amongst the aforementioned models, the Poisson regression produced the most favourable test error measures, underpinned by the visualization in Figure 5.2. It may as well be assumed that its supremacy over the machine learning models is attributed to the fact that the scarlatina cases have a well-predictable sinusodial structure, and as a result it is advisable that one use a prefixed structure – anchoring the model –, rather than applying more flexible, but less anchored models.

The advantage of the Poisson over its prefixed-structure counterparts might be less clear, however, the author has the following thoughts. The reason the Poisson is better than the negative binomial for is that the latter one has one more parameter to be estimated and, in addition, during the optimization process one term containing that parameter were omitted because of computational constraints. Accounting for the superiority to NLS is far less obvious, it might have its roots in the maximum likelihood fitting vs. the squared error fitting.

To conclude, the answer to the research question, that is, which one of the examined models is best for predicting the scarlatina cases is the Poisson regression model.
Epinfo.

Scarlet fever: A group a streptococcal infection.


Appendix A: Notations

Machine learning notations

Boosting trees

\[ T \] – Single weak tree if not stated otherwise
\[ R_m \] – Terminal nodes of a weak tree
\[ M \] – Number of terminal nodes in a weak tree
\[ c_m \] – Prediction value of the \( m \)th terminal nodes
\[ \Xi \] – Parameter of a weak tree
\[ S \] – Number of weak trees in the ensemble

Support vector machine

\[ \omega_k \] – Weight of the \( k \)th basis expansion in SVM
\[ \Lambda \] – Penalty in SVM
\[ \zeta \] – SVM loss function parameter
\[ \kappa_i \] – Weight of the \( i \)th kernel in SVM
\[ K \] – Kernel matrix
\[ \sigma \] – Kernel scale parameter in SVM
Traditional models notations

\( f_{NLS} \) – NLS function
- \( \Theta \) – General parameter notation in all traditional models
- \( l \) – Number of elements in the general parameter set, \( \Theta \)
- \( \mu \) – Expected value of the dependent variable
- \( \lambda \) – Expected value of the Poisson distribution
- \( f_P \) – Poisson function
- \( L \) – Likelihood function
- \( \ell \) – Log-likelihood function
- \( f_{Nb} \) – Negative binomial function
- \( \rho \) – Parameter of the negative binomial distribution
- \( p \) – Parameter of the negative binomial distribution
- \( \Gamma \) – Gamma function

\( \beta^{NLS} \) – Parameter set in the estimated NLS model

\( \phi^{NLS} \) – Autoregressive parameter in the estimated NLS model

\( \beta^P \) – Parameter set in the estimated Poisson model

\( \phi^P \) – Autoregressive parameter in the estimated Poisson model

\( \beta^{Nb} \) – Parameter set in the estimated negative binomial model

\( \phi^{Nb} \) – Autoregressive parameter in the estimated negative binomial model

\( \rho \) – Parameter in the estimated negative binomial model
Appendix B: Tables

Table B.1: Regression parameters – traditional models

<table>
<thead>
<tr>
<th></th>
<th>NLS</th>
<th>Poisson</th>
<th>Negative binomial</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\beta_0$</td>
<td>12.4977</td>
<td>0.8264</td>
<td>1.7847</td>
</tr>
<tr>
<td>$\beta_1$</td>
<td>5.8289</td>
<td>-0.0171</td>
<td>-0.1055</td>
</tr>
<tr>
<td>$\beta_3$</td>
<td>17.9185</td>
<td>78.1917</td>
<td>76.9941</td>
</tr>
<tr>
<td>$\beta_4$</td>
<td>-1.4333</td>
<td>-0.0257</td>
<td>0.0941</td>
</tr>
<tr>
<td>$\beta_5$</td>
<td>5.0208</td>
<td>-3.9460</td>
<td>-3.8145</td>
</tr>
<tr>
<td>$\beta_6$</td>
<td>2.1443</td>
<td>6.3268</td>
<td>4.3506</td>
</tr>
<tr>
<td>$\phi$</td>
<td>0.8261</td>
<td>0.8166</td>
<td>0.5888</td>
</tr>
<tr>
<td>$\rho$</td>
<td></td>
<td></td>
<td>0.1814</td>
</tr>
</tbody>
</table>

Notes: the table contains the estimated values of the $\beta$ parameters in the NLS, Poisson and negative binomial regression models. In addition, the estimated $\rho$ parameter of the negative binomial density is attached. Values are rounded to four decimals. All the parameters are significant at $\alpha = 5\%$, except the ones responsible for vertical adjustment: $\beta_3$ and $\beta_6$. 
### Table B.2: Test results

<table>
<thead>
<tr>
<th>Model</th>
<th>ME</th>
<th>MPE</th>
<th>MAE</th>
<th>MAPE</th>
<th>MSE</th>
<th>RMSE</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>(37.6250)</td>
<td>(32.9251)</td>
<td>(38.8323)</td>
<td>(47.2790)</td>
<td>(3140.2806)</td>
<td>(56.0382)</td>
</tr>
<tr>
<td></td>
<td>(12.6122)</td>
<td>(-61.2456)</td>
<td>(35.1573)</td>
<td>(95.9988)</td>
<td>(2470.1579)</td>
<td>(49.7007)</td>
</tr>
<tr>
<td>NLS</td>
<td>-10.1790</td>
<td>-94.2050</td>
<td>17.0164</td>
<td>103.5430</td>
<td>435.7759</td>
<td>20.8752</td>
</tr>
<tr>
<td></td>
<td>(0.3382)</td>
<td>(-24.7250)</td>
<td>(18.2844)</td>
<td>(42.2980)</td>
<td>(761.0403)</td>
<td>(27.5870)</td>
</tr>
<tr>
<td>Poisson</td>
<td>-4.6314</td>
<td>-40.0928</td>
<td>13.9424</td>
<td>55.7575</td>
<td>369.4489</td>
<td>19.2211</td>
</tr>
<tr>
<td></td>
<td>(-0.0076)</td>
<td>(-21.8011)</td>
<td>(18.1916)</td>
<td>(39.0455)</td>
<td>(761.2274)</td>
<td>(27.5903)</td>
</tr>
<tr>
<td>Neg. binom.</td>
<td>-9.1817</td>
<td>-77.0574</td>
<td>15.5569</td>
<td>86.0984</td>
<td>402.9021</td>
<td>20.0724</td>
</tr>
<tr>
<td></td>
<td>(2.9979)</td>
<td>(-34.2770)</td>
<td>(22.1729)</td>
<td>(53.2065)</td>
<td>(1050.5967)</td>
<td>(32.4129)</td>
</tr>
</tbody>
</table>

**Best**
- Poisson
- B. trees

**Notes:** The table contains the calculated test error measures for the models. In parenthesis, shown are the same measures calculated for the training set. All values are rounded to four decimals.
Appendix C: Program codes

C.1 Preparation

1 Rawdata=readtable('Fertozo.xls');
2 % Training sample:
3 cutindex=816;
4
5 %New variables - whole sample
6 y.whole=Rawdata.Skarlat;
7 ylog.whole=log(Rawdata.Skarlat);
8 t.whole=[1:length(ylog.whole)]';
9 mdummy.whole=dummyvar(month(Rawdata.Timing));
10 Timingconv.whole=datetime(Rawdata.Timing,'InputFormat','yyyy. ...

   MM. dd.');
11 yloglag.whole(:,:,1)=lagmatrix(ylog.whole,[1:52]')
12 ylag.whole=lagmatrix(y.whole,[1:52]')
13 %New variables - training sample
14 y.train=y.whole(1:cutindex);
15 ylog.train=ylog.whole(1:cutindex);
16 t.train=t.whole(1:cutindex);
17 mdummy.train=mdummy.whole(1:cutindex,:);
18 Timingconv.train=Timingconv.whole(1:cutindex);
19 yloglag.train=yloglag.whole(1:cutindex,:)
20 ylag.train=ylag.whole(1:cutindex,:)
21 %New variables - test sample
22 y.test=y.whole(cutindex+1:end,:);
23 ylog.test=ylog.whole(cutindex+1:end,:);
24 t.test=t.whole(cutindex+1:end,:);
25 mdummy.test=mdummy.whole(cutindex+1:end,:);
26 Timingconv.test=Timingconv.whole(cutindex+1:end,:);
27 yloglag.test=yloglag.whole(cutindex+1:end,:)
28 ylag.test=ylag.whole(cutindex+1:end,:);
%Load modified whole sample data to new table
Baranyhimlo=Rawdata.Baranyhimlo;
HIV=Rawdata.HIV;
Hepatitis=Rawdata.Hepatitis;
Mono=Rawdata.Mono;

Prepdata=table(Timingconv.whole,t.whole,y.whole,...
    ylog.whole,yloglag.whole,...
    HIV,Hepatitis,Mono,Baranyhimlo,mdummy.whole);

clear HIV Hepatitis Mono Baranyhimlo

%Plot the partitioning
fig=figure;
plot(Timingconv.whole,y.whole,'r')
hold on
plot(Timingconv.train,y.train)
xlabel('Time','Interpreter','LaTeX','Fontsize',12)
ylabel('Scarlatina cases','Interpreter','LaTeX','Fontsize',12)
title('\textbf{Partitioning of the ... sample}','Interpreter','LaTeX',...
    'Fontsize',13)
l=legend('Test set','Training set');
l.FontSize=12;
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig,'figpart.pdf','-dpdf','-r300');

%Descriptives
wholedesc=descriptives(y.whole);
traindesc=descriptives(y.train);
testdesc=descriptives(y.test);
Desc=table(wholedesc,traindesc,testdesc);
clear wholedesc traindesc testdesc

%Plot histogram of y and ylog
fig=figure;
subplot(1,2,1)
histogram(y.train)
title('\textbf{Histogram of Scarlatina}','Interpreter','LaTeX',...
    'Fontsize',13)
xlabel('Scarlatina','Interpreter','LaTeX','Fontsize',12)
ylabel('Frequency','Interpreter','LaTeX','Fontsize',12)
subplot(1,2,2)
hist(ylog.train)
title('Histogram of $\log$(Scarlatina)','...'
'Interpreter','LaTeX','Fontsize',13)
xlabel('$\log$(Scarlatina)','Interpreter','LaTeX','Fontsize',12)
ylabel('Frequency','Interpreter','LaTeX','Fontsize',12)
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'fighistsc.pdf', '-dpdf', '-r300' );

% Plot time series
fig=figure;
subplot(2,1,1)
plot(Timingconv.train,y.train)
datetick('x','keeplimits')
xlabel('Time','Interpreter','LaTeX','Fontsize',12)
ylabel('Scarlatina','Interpreter','LaTeX','Fontsize',12)
title('Time series plot of ...Scarlatina)','Interpreter','LaTeX',...'
'Fontsize',13)
subplot(2,1,2)
plot(Timingconv.train,ylog.train,'r')
datetick('x','keeplimits')
xlabel('Time','Interpreter','LaTeX','Fontsize',12)
ylabel('$\log$(Scarlatina)','Interpreter','LaTeX','Fontsize',12)
title('Time series plot of $\log$(Scarlatina)','...'
'Interpreter','LaTeX','Fontsize',13)
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figtssc.pdf', '-dpdf', '-r300' );

%Input variables for modelling – whole sample
x.whole=[t.whole, yloglag.whole(:,1), yloglag.whole(:,52) ...mdummy.whole];
xres1.whole=[t.whole yloglag.whole(:,1)];
xres2.whole=t.whole;

%Input variables for modelling – training sample
x.train=x.whole(1:cutindex,:);
xres1.train=xres1.whole(1:cutindex,:);
xres2.train=xres2.whole(1:cutindex,:);
%Input variables for modelling – test sample
x.test=x.whole(cutindex+1:end,:);

xres1.test=xres1.whole(cutindex+1:end,:);

xres2.test=xres2.whole(cutindex+1:end,:);

%Plotmatrix, relationships
fig=figure;
[f ax]=plotmatrix(x.train(:,1:3),ylog.train,'r*');
title('	extbf{\log}(Scarlatina) as a function of input ... variables}','...
'Interpreter','LaTeX','Fontsize',13)
set(get(gca,'YLabel'),'String','$\log$(Scarlatina)','...
'Interpreter','LaTeX')
set(gca,'FontSize',12)
set(get(ax(1),'XLabel'),'String','Time','Interpreter','LaTeX')
set(ax(1),'FontSize',12)
set(get(ax(2),'XLabel'),'String','$\log($Scarlatina$_{-1})$'...
'Interpreter','LaTeX')
set(ax(2),'FontSize',12)
set(get(ax(3),'XLabel'),'String','$\log($Scarlatina$_{-2})$'...
'Interpreter','LaTeX')
set(ax(3),'FontSize',12)

fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figrel.pdf', '-dpdf', '-r300' );
clear f ax

C.2 Models

C.2.1 Machine learning

Boosting trees

%% Fitting boosted regression trees with varying weak trees
tic;
k=1;
for splits=1:7
rng(1);
ttemplate=templateTree('MaxNumSplits',splits);
Tensemble=fitensemble(x.train,ylog.train,'LSBoost',...
'LearnRate',0.1,'CrossVal','on','kfold',10);
rng(1);
Tensembleres1=fitensemble(xres1.train,ylog.train,'LSBoost',...%
100,ttemplate,...
'LearnRate',0.1,'CrossVal','on','kfold',10);
rng(1);
Tensembleres2=fitensemble(xres2.train,ylog.train,'LSBoost'...%
100,ttemplate,...
'LearnRate',0.1,'CrossVal','on','kfold',10);
klossfull(:,k)=kfoldLoss(Tensemble,'mode','cumulative',...%
'lossfun',@mapehandle);
yloghatfull(:,k)=kfoldPredict(Tensemble);
klossres1(:,k)=kfoldLoss(Tensembleres1,'mode','cumulative',...%
'lossfun',@mapehandle);
yloghatres1(:,k)=kfoldPredict(Tensembleres1);
klossres2(:,k)=kfoldLoss(Tensembleres2,'mode','cumulative',...%
'lossfun',@mapehandle);
yloghatres2(:,k)=kfoldPredict(Tensembleres2);
k=k+1;
end
% clear k splits
toc;

%Losses into structure variable
Tloss.full=klossfull;
Tloss.res1=klossres1;
Tloss.res2=klossres2;
Yloghat.Tree.CVfull=yloghatfull;
Yloghat.Tree.CVres1=yloghatres1;
Yloghat.Tree.CVres2=yloghatres2;
% clear klossfull klossres1 klossres2 yloghatfull yloghatres1 ...
yloghatres2

% Tree ensemble selection

% Selection of the restricted ensembles

%medians accross the 7 splits opportunity
Tloss.medianSplits.full=median(Tloss.full,2,'omitnan');
Tloss.medianSplits.res1=median(Tloss.res1,2,'omitnan');
Tloss.medianSplits.res2=median(Tloss.res2,2,'omitnan');

%plot the medians
fig=figure;
plot(Tloss.medianSplits.full,'*')
hold on
plot(Tloss.medianSplits.res1,'r*')
plot(Tloss.medianSplits.res2,'g*')
xlabel('Number of weak ... learners','Interpreter','LaTeX','Fontsize',12)
ylabel('Cumulative MAPE ... loss','Interpreter','LaTeX','Fontsize',12)
l=legend('Full','Restricted1','Restricted2');
l.FontSize=12;
l.Interpreter='LaTeX';
title('\textbf{Predictor selection}','Interpreter','LaTeX',... 'FontSize',13)
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figpred.pdf', '-dpdf', '-r300');

% Selection of the weak learner splits of the selected ... restricted model

%plot
fig=figure;
subplot(2,1,1)
plot(Tloss.res1(:,1),'d')
hold on
plot(Tloss.res1(:,2),'rd')
plot(Tloss.res1(:,3),'gd')
plot(Tloss.res1(:,4),'md')
plot(Tloss.res1(:,5),'kd')
plot(Tloss.res1(:,6),'cd')
plot(Tloss.res1(:,7),'bd')
xlabel('Number of weak ... learners','Interpreter','LaTeX','Fontsize',12)
ylabel('Cumulative MAPE ... loss','Interpreter','LaTeX','Fontsize',12)
l=legend('S1','S2','S3','S4','S5','S6','S7');
l.Orientation='horizontal';
l.FontSize=12;
l.Interpreter='LaTeX';
title('\textbf{Splits selection}','Interpreter','LaTeX',... 'FontSize',13)
subplot(2,1,2)
plot(median(Tloss.res1),'rs','LineWidth',2)
hold on
plot(mean(Tloss.res1),'gs','LineWidth',2)
hold off
xlabel('Number of splits','Interpreter','LaTeX')
ylabel('MAPE loss','Interpreter','LaTeX')
l=legend('Median','Mean');
l.FontSize=12;
l.Interpreter='LaTeX';
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figsplits.pdf', '-dpdf', '-r300' );

optsplits=1;

% Selection of the number of weak learners
clear optnumber
i=1;
while Tloss.res1(i,1)−Tloss.res1(i+1,1)>0.000001
    optnumber=i;
    i=i+1;
    if i>102
        break
    end
end
clear i

%% Best tree ensemble: Tensembleres1

template=templateTree('MaxNumSplits',optsplits);
Tensemblefinal=fitensemble(xres1.train,ylog.train,'LSBoost',...
    optnumber,ttemplate,'LearnRate',0.1);

%Prediction with the best ensemble without CV
Yloghat.Tree.final=resubPredict(Tensemblefinal);

%Loss by best ensemble without CV
Yloss.train.Tree=resubLoss(Tensemblefinal,'lossfun',@mapehandle);
Support vector machine

```matlab
%% SVM with varying kernel scale and constraint
xSVM.whole=[yloglag.whole(:,1:2)];
xSVM.train=xSVM.whole(1:cutindex,:);
xSVM.test=xSVM.whole(cutindex+1:end,:);

tic;

a=1;
for sigma=0.01:0.5:10
    b=1;
    for i=0.01:0.5:10
        rng(1);
        SVMRBF=fitrsvm(xSVM.train,ylog.train,'KernelFunction','gaussian',
                    'CrossVal','on',
                    'kfold',10,'BoxConstraint',
                    (iqr(ylog.train)/1.349)*i,'KernelScale',2*sigma^2);
        ivalue(b)=i;
        constraint(b)=(iqr(ylog.train)/1.349)*i; % 1/lambda
        pred=kfoldPredict(SVMRBF);
        SVMfit(:,(20*(a-1)+b))=pred;
        SVMLoss(b,a)=kfoldLoss(SVMRBF,'mode','average','lossfun',
                                @mapehandle);
        b=b+1;
    end
    sigmavalue(a)=sigma;
    a=a+1;
end
toc

%% SVM plot and grid search
fig=figure;
surface(constraint,sigmavalue,SVMLoss)
xlabel('$\frac{1}{\Lambda}$','Interpreter','LaTeX','Fontsize',12);
ylabel('$\sigma$','Interpreter','LaTeX','Fontsize',12);
ylabel('MAPE loss','Interpreter','LaTeX');
title(['\textbf{MAPE loss of SVM as a function of kernel scale'}]);
```

XI
'and the inverse penalty term')}, 'Interpreter', 'LaTeX', ...
'Fontsize', 13)

fig.PaperUnits = 'normalized';
fig.PaperPosition = [0.01 0.01 0.99 0.4];
print(fig, 'figgrid.pdf', '-dpdf', '-r300');

 [~, SVMmincoordS] = min(min(SVMLoss, [], 2)); % minimizer constraint coordinate

 [~, SVMmincoordC] = min(min(SVMLoss)); % minimizer sigma coordinate
optConstraint = constraint(SVMmincoordC);
optSigma = sigmavalue(SVMmincoordS);
clear SVMmincoordC SVMmincoordS

%% SVM training with the selected kernel scale and constraint
% Training best SVM
SVMRBFfinal = fitrsvm(xSVM.train, ylog.train, 'KernelFunction', ...
'gaussian', 'CrossVal', 'off', ...
'BoxConstraint', optConstraint, ...
'KernelScale', 2 * optSigma^2);

% Prediction with best SVM without CV
Yloghat.SVM.final = resubPredict(SVMRBFfinal);

% 'Impute' the first observation lost by autoregression
Yloghat.SVM.final = [0; 0; Yloghat.SVM.final];

% Loss by best constraint SVM without CV
Yloss.train.SVM = resubLoss(SVMRBFfinal, 'lossfun', @mapehandle);

%% Plot tree and SVM predictions
Yloghat.SVM.final(1:2) = NaN;
fig = figure;
plot(Timingconv.train, y.train)
hold on
plot(Timingconv.train, exp(Yloghat.Tree.final), 'r')
plot(Timingconv.train, exp(Yloghat.SVM.final), 'g')
datetick('x', 'keeplimits')
title('\textbf{Prediction with boosting trees and SVM}', ...
'Interpreter', 'LaTeX', 'FontSize', 13)
1.legend('Actual', 'BT', 'SVM');
1.FontSize = 12;
1.Interpreter = 'LaTeX';
ylabel('Scarlatina', 'Interpreter', 'LaTeX', 'FontSize', 12)
xlabel('Time', 'Interpreter', 'LaTeX', 'FontSize', 12)
fig.PaperUnits = 'normalized';

XII
C.2.2 Traditional models

NLS

```matlab
%% NLS

data = [t.train, ylag.train(:,1)];

% NLS function
NLSfun = @(bnls, x) ... 
    bnls(1) + bnls(2) * sin(x(:,1)./bnls(3) - bnls(4)) + ... 
    bnls(5) * sin(x(:,1)./bnls(6) - bnls(7)) + bnls(8) * x(:,2)

% Initial parameter guess
NLSnull = [60 30 65 10, ... % beta: 1, 2, 3, 4
            10 5 3 0.2];

% Options
opts = statset('Display','iter','TolFun',1e-30);

% Fitting
NLS = fitnlm(data, y.train, NLSfun, NLSnull, 'options', opts)

fig = figure;
plot(Timingconv.train, y.train)
hold on
plot(Timingconv.train, NLS.Fitted, 'r')
datetick('x','keeplimits')
title('	extbf{NLS fit}','Interpreter','LaTeX','Fontsize',13)
xlabel('	extbf{Time}','Interpreter','LaTeX','Fontsize',12)
ylabel('	extbf{Scarlatina}','Interpreter','LaTeX','Fontsize',12)
l=legend('Actual','Fitted');
l.FontSize=12;
l.Interpreter='LaTeX';
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figNLS.pdf', '-dpdf', '-r300');
clear data l fig
```
%% Poisson

%Function to minimize
yloglag.train(1,1)=0;
data=[y.train yloglag.train(:,1) t.train];
pftomin=@(data,bp) -sum(data(:,1).*(bp(1)+... 
    bp(2)*sin(data(:,3)/bp(3)-bp(4))+... 
    bp(5)*sin(data(:,3)./bp(6)-bp(7))+... 
    bp(8)*data(:,2))+... 
    -exp(bp(1)+bp(2)*sin(data(:,3)/bp(3)-bp(4))+... 
    bp(5)*sin(data(:,3)./bp(6)-bp(7))+... 
    bp(8)*data(:,2)));

pftominy=@(bp) pftomin(data,bp);

%Starting values
startp=[0.614, 0.0537, -20.51, 78.000, -0.0672, -3.857, 4.152, 0.591];

%Fminunc
opt=optimoptions('fminunc','FunValCheck','off');
[bphat, pfvalhat, poutput, pgrad, phessian]=fminunc(pftominy,startp)

%Fitted
Yfit.poi=exp(bphat(1)+bphat(2)*sin(t.train/bphat(3)-bphat(4))... 
    +bphat(5)*sin(t.train./bphat(6)-bphat(7))... 
    +bphat(8)*yloglag.train(:,1));
Yfit.poi(1,1)=NaN;

%Figure
plot(Timingconv.train,y.train)
hold on
plot(Timingconv.train,Yfit.poi,'g')
legend('Actual','Fitted')
title('Poisson')
figure
histogram(y.train)
hold on
histogram(Yfit.poi)
title('Poisson')
% Neg binom

% Function to minimize
yloglag.train(1,1)=0;
data=[y.train yloglag.train(:,1) t.train];
nbtomin=@(data,bnb) -sum(-log(gamma(bnb(9)))+...
data(:,1).*(bnb(1)+...
bnb(2)*sin(data(:,3)./bnb(3)−bnb(4))...
+bnb(5)*sin(data(:,3)./bnb(6)−bnb(7))...nbn(8)*data(:,2))...−data(:,1).*log(exp(bnb(1)+bnb(2)*sin(data(:,3)./bnb(3)−bnb(4)))...+bnb(5)*sin(data(:,3)./bnb(6)−bnb(7))...nbn(8)*data(:,2)+bnb(9))...+bnb(9)*log(bnb(9))−bnb(9)*log(exp(...
bnb(1)+bnb(2)*sin(data(:,3)./bnb(3)−bnb(4))...+bnb(5)*sin(data(:,3)./bnb(6)−bnb(7))...nbn(8)*data(:,2)+bnb(9)));
nbtominy=@(bnb)nbtomin(data,bnb);

% Options
opt=optimoptions('fminunc','FunValCheck','off','Display','iter');

% Starting values
rng(1);
startnb=[0.614, 0.0537,−20.51,78.000,−0.0672,−3.857,4.152,0.591, ...0.6];

% Fminunc
[bnbhat, nbfvalhat,nboutput]=fminunc(nbtominy,startnb,opt)

% Fitted
Yfit.negbinom=exp(bnbhat(1)+...bnbhat(2)*sin(t.train./bnbh(3)−bnbh(4))...+bnbhat(5)*sin(t.train./bnbh(6)−bnbh(7))...+bnbhat(8).*yloglag.train(:,1));
Yfit.negbinom(1,1)=NaN;
C.3 Assessment

C.3.1 Assessment function

```matlab
function out=assess(Y,Yfit,W)
Yfit(isnan(Yfit))=0;

%ONLY for W=ones(length(Y),1)*1/length(Y) and POSITIVE data!

%ME
out.ME=sum(Y-Yfit)/length(Y);

%MPE
out.MPE=100*sum((Y-Yfit)./Y)/length(Y);

%MAE
out.MAE=sum(abs(Y-Yfit))/length(Y);

%MAPE
out.MAPE=100*sum(abs((Y-Yfit)./Y))/length(Y);

%MSE
```

XVI
C.3.2 Assessing models

```matlab
% Test predictions
xtestML=xres1.test;
xtest=[t.test,ylag.test(:,1)];

%ML
Yhat test.Tree=exp(predict(Tensemblefinal,xtestML));
Yhat test.SVM=exp(predict(SVMRBFfinal,xSVM.test));

%Traditional
Yhat test.NLS=predict(NLS,xtest);
Yhat test.Poi=exp(bphat(1)+bphat(2)*sin(t.test./bphat(3)-bphat(4))...
    +bphat(5)*sin(t.test./bphat(6)-bphat(7))...%
    +bphat(8).*yloglag.test(:,1));
Yhat test.Nbin=exp(bnbhat(1)+...
    bnbhat(2)*sin(t.test./bnbhat(3)-bnbhat(4))...
    +bnbhat(5)*sin(t.test./bnbhat(6)-bnbhat(7))...%
    +bnbhat(8).*yloglag.test(:,1));

%Plot test fitting
%ML
fig=figure;
hold on
plot(Timingconv.test,y.test)
plot(Timingconv.test,Yhat test.Tree,'r')
plot(Timingconv.test,Yhat test.SVM,'g')
datetick('x','keeplimits')
title('Test prediction with boosting trees and SVM',...
    'Interpreter','LaTeX','Fontsize',13)
l=legend('Actual','BT','SVM');
l.FontSize=12;
l.Interpreter='LaTeX';
ylabel('Scarlatina','Interpreter','LaTeX','FontSize',12)
xlabel('Time','Interpreter','LaTeX','FontSize',12)
```

XVII
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figtestml.pdf', '-dpdf', '-r300');
clear l fig
fig=figure;
%Traditional
plot(Timingconv.test,y.test)
hold on
plot(Timingconv.test,Yhattest.NLS,'-r')
plot(Timingconv.test,Yhattest.Poi,'-g')
plot(Timingconv.test,Yhattest.Nbin,'-m')
datetick('x','keeplimits')
title('\textbf{Test prediction with traditional models}',...
    'Interpreter','LaTeX','Fontsize',13)
xlabel('Time','Interpreter','LaTeX','Fontsize',12)
ylabel('Scarlatina','Interpreter','LaTeX','Fontsize',12)
l=legend('Actual','NLS','Poisson','Neg.binom');
l.FontSize=12;
l.Interpreter='LaTeX';
fig.PaperUnits='normalized';
fig.PaperPosition=[0.01 0.01 0.99 0.4];
print(fig, 'figtradtest.pdf', '-dpdf', '-r300');

%%% Assessment

%In-sample
%Tree ensemble
insample.Tree=assess(y.train,exp(Yloghat.Tree.final),...
    ones(length(ylog.train),1)./length(ylog.train));

%SVM
insample.SVM=assess(y.train,exp(Yloghat.SVM.final)...,
    ones(length(ylog.train),1)./length(ylog.train));

%NLS
insample.NLS=assess(y.train,NLS.Fitted,...
    ones(length(ylog.train),1)./length(ylog.train));

%Poisson
insample.Poi=assess(y.train,Yfit.poi,...
    ones(length(ylog.train),1)./length(ylog.train));
%Negative binomial

insample.Nbin=assess(y.train,Yfit.negbinom,...
    ones(length(ylog.train),1)./length(ylog.train));

%Out-of-sample

%Tree ensemble

outsample.Tree=assess(y.test,Yhattest.Tree,...
    ones(length(ylog.test),1)./length(ylog.test));

%SVM

outsample.SVM=assess(y.test,Yhattest.SVM,...
    ones(length(ylog.test),1)./length(ylog.test));

%NLS

outsample.NLS=assess(y.test,Yhattest.NLS,...
    ones(length(ylog.test),1)./length(ylog.test));

%Poisson

outsample.Poi=assess(y.test,Yhattest.Poi,...
    ones(length(ylog.test),1)./length(ylog.test));

%Negative binomial

outsample.Nbin=assess(y.test,Yhattest.Nbin,...
    ones(length(ylog.test),1)./length(ylog.test));