



Reservoir Computing in R: a Tutorial for Using reservoirnet to Predict Complex Time-Series

ISSN 2824-7795

- Thomas Ferté  Inserm Bordeaux Population Health Research Center UMR 1219, Inria BSO, team SISTM, F-33000 Bordeaux, France, Inserm, Inria
Bordeaux Hospital University Center, Pôle de santé publique, Service d'information médicale, F-33000 Bordeaux, France, CHU de Bordeaux
- Kalidou Ba  Inserm Bordeaux Population Health Research Center UMR 1219, Inria BSO, team SISTM, F-33000 Bordeaux, France, Inserm, Inria
Dan Dutarte  Inria BSO, Inria
Pierrick Legrand Inria BSO, F-33000 Bordeaux, France, Inria
IMB, Institut de Mathématiques de Bordeaux, UMR CNRS 5251, IMB
- Vianney Jouhet  Inserm Bordeaux Population Health Research Center UMR 1219, team AHeAD, F-33000 Bordeaux, Inserm
Bordeaux Hospital University Center, Pôle de santé publique, Service d'information médicale, F-33000 Bordeaux, France, CHU de Bordeaux
- Romain Griffier  Inserm Bordeaux Population Health Research Center UMR 1219, team AHeAD, F-33000 Bordeaux, Inserm
Bordeaux Hospital University Center, Pôle de santé publique, Service d'information médicale, F-33000 Bordeaux, France, CHU de Bordeaux
- Rodolphe Thiébaut  Inserm Bordeaux Population Health Research Center UMR 1219, Inria BSO, team SISTM, F-33000 Bordeaux, France, Inserm, Inria
Bordeaux Hospital University Center, Pôle de santé publique, Service d'information médicale, F-33000 Bordeaux, France, CHU de Bordeaux
- Xavier Hinault  Inria BSO, F-33000 Bordeaux, France, Inria
Univ. Bordeaux, CNRS, IMN, UMR 5293, Bordeaux, France, CNRS
LaBRI, Univ. Bordeaux, Bordeaux INP, CNRS UMR 5800., LaBRI
- Boris Hejblum  ¹ Inserm Bordeaux Population Health Research Center UMR 1219, Inria BSO, team SISTM, F-33000 Bordeaux, France, Inserm, Inria

Date published: 2024-12-10 Last modified: 2024-12-10

Abstract

Reservoir Computing (RC) is a machine learning method based on neural networks that efficiently process information generated by dynamical systems. It has been successful in solving various tasks including time series forecasting, language processing or voice processing. RC is implemented in Python and Julia but not in R. This article introduces `reservoirnet`, an R package providing access to the Python API `ReservoirPy`, allowing R users to harness the power of reservoir computing. This article provides an introduction to the fundamentals of RC and showcases its real-world applicability through three distinct sections. First, we cover the foundational concepts of RC, setting the stage for understanding its capabilities. Next, we delve into the practical usage of `reservoirnet` through two illustrative examples. These examples demonstrate how it can be applied to real-world problems, specifically, regression of COVID-19 hospitalizations and classification of Japanese vowels. Finally, we present a comprehensive analysis of a real-world application of `reservoirnet`, where it was used to forecast COVID-19 hospitalizations at Bordeaux University Hospital using public data and electronic health records.

¹Corresponding author:

Keywords: Reservoir Computing, Covid-19, Electronic Health Records, Time series

1 **Contents**

2	1 Introduction	3
3	2 RC presentation	4
4	3 Usage workflow	5
5	3.1 Installation	5
6	3.2 Package workflow overview	7
7	3.3 Basic regression use-case	7
8	3.3.1 Covid-19 data	7
9	3.3.2 First reservoir	8
10	3.3.3 Forecast	11
11	3.4 Classification	15
12	3.4.1 The Japanese vowel dataset	15
13	3.4.2 Classification (sequence-to-vector model)	16
14	3.4.3 Transduction (sequence-to-sequence model)	19
15	4 Avanced case-study: Covid-19 hospitalizations forecast	22
16	4.1 Introduction	22
17	4.2 Methods	23
18	4.2.1 Data	23
19	4.2.2 Evaluation framework	23
20	4.2.3 Models	24
21	4.2.4 Hyperparameter optimisation using random search	24
22	4.3 Results	25
23	4.3.1 Hyperparameter selection	25
24	4.3.2 Forecast performance	25
25	4.3.3 Number of model to aggregate	29
26	4.3.4 Input feature importance	29
27	4.4 Discussion	34
28	5 Discussion and conclusion	34
29	References	35
30	Session information	38

31 1 Introduction

32 Reservoir Computing (RC) is a prominent machine learning method, proposed by Jaeger (2001), Maass,
33 Natschläger, and Markram (2002) and Lukoševičius and Jaeger (2009) that has gained significant
34 attention in recent years for its ability to efficiently process information generated by dynamical
35 systems. This innovative approach leverages the dynamics of a high-dimensional “reservoir” (we
36 define it below) to perform complex computations and solve various tasks based on the response
37 of this dynamical system to input signals. RC has demonstrated its efficacy in tackling various
38 challenges, encompassing pattern classification and time series forecasting in applications ranging
39 from electrocardiogram analysis to bird calls (Trouvain and Hinaut 2021), language processing
40 (Hinaut and Dominey 2013), power plants, internet traffic, stock prices, and beyond (Lukoševičius
41 and Jaeger 2009; Tanaka et al. 2019).

42 Originally, the RC paradigm was implemented in artificial firing-rate neurons (“Echo State Networks”,
43 Jaeger (2001)) and spiking neurons (“Liquid State Machine”, Maass, Natschläger, and Markram (2002))
44 as a recurrent neural network (RNN) where the internal recurrent connections, denoted as the
45 reservoir, are randomly generated and only the output layer (named “read-out”) is trained. The
46 reservoir projects temporal input signals onto a high-dimensional feature space, facilitating the
47 learning of non-linear and temporal interactions. Thus, this recurrent layer contains high-dimensional
48 non-linear recombination of the inputs and past states: it is a “reservoir of computations” from
49 which useful information can be linearly extracted (or “read-out”) to provide the desired outputs.
50 This offers the advantage of decreasing the computing time compared to conventional RNNs while
51 consistently maintaining performance (Vlachas et al. 2020). Besides, this RC paradigm fostered
52 increasing interest thanks to its ability to be implemented on classical computers, as the hidden
53 recurrent layer can be kept untrained. A wide range of physical media can be also used to replace
54 it and Tanaka et al. (2019) recently reviewed this prolific field: from FPGA hardware (Penkovsky,
55 Larger, and Brunner 2018), to spin waves using magnetic properties (Nakane, Tanaka, and Hirose
56 2018), skrymions (Prychynenko et al. 2018) or optical implementations (Rafayelyan et al. 2020).
57 This provides interesting and potentially more efficient alternative to traditional machine learning
58 computing.

59 RC leverages various hyperparameters to introduce prior knowledge about the relationship between
60 input variables and output targets. But because the connections within the reservoir are randomly
61 initialized, the same set of hyperparameters may exhibit diverse behaviors across different instances
62 of the reservoir connections. This unpredictability makes it challenging to anticipate the performance
63 of a particular hyperparameter setting, as identical settings may produce varying outcomes when
64 applied to distinct instances of the reservoir. Moreover, selecting the most suitable hyperparameters
65 often requires researchers to experiment with multiple combinations on a training dataset and
66 evaluate their performance on a separate test set². Although this approach can be resource-intensive
67 and time-consuming, it is a compromise that is acceptable considering the rapid simulation capabilities
68 offered by RC. Furthermore, there is a current absence of implementation in R, rendering the method
69 challenging for users unfamiliar with Python (Trouvain and Hinaut 2022) or Julia (Martinuzzi et al.
70 2022).

71 Here, we offer comprehensive guidance to assist new users in maximizing the benefits of RC. Initially,
72 a broad introduction to reservoir computing is presented in Section 2, followed in Section 3 by a
73 tutorial on its application using `reservoirnet`, an R package built upon the `ReservoirPy` Python
74 module (Trouvain, Rougier, and Hinaut 2022; Trouvain and Hinaut 2022; Trouvain et al. 2020).
75 Section 3 then introduces the workflow usage on `reservoirnet` for RC with two basic use-cases,

²In this article, we employ the term “train set” to refer to the combined dataset consisting of both the training and validation sets, which are cycled through in a cross-validation manner.

76 and finally, in Section 4 we investigate the various challenges associated with an advanced case-
 77 study leveraging RC for forecasting COVID-19 hospitalizations. This case-study exploration includes
 78 detailed guidance on the modeling strategy, the selection of hyperparameters, and the implementation
 79 process.

80 2 RC presentation

81 RC is a machine learning paradigm which is most often implemented as Echo State Networks (ESNs),
 82 i.e. the firing-rate neuron version (Jaeger 2001). An ESN is described by three matrices of connectivity:
 83 an input layer W_{in} , a recurrent layer W and an output layer W_{out} . At each time step, the input vector
 84 u_t is projected into the reservoir which is also combined with reservoir past state $x(t - 1)$ through
 85 the recurrent connections. The output $y(t)$ is linearly read-out from the reservoir. Input W_{in} and
 86 recurrent W matrices are kept random; only the output matrix W_{out} is trained in an offline or online
 87 method. Often a ridge regression (i.e. a regularized linear regression) is used to obtain the desired
 88 outputs $y(t)$ from the reservoir states $x(t)$. Figure 1 depicts the architecture. For simplicity, we will
 89 use the term “reservoir computing” for “Echo State Network” in the remainder of the paper.

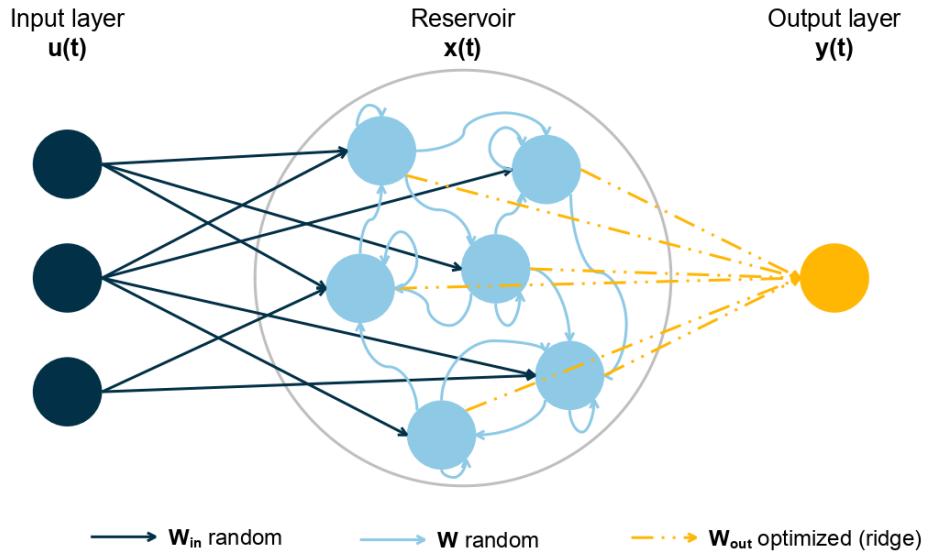


Figure 1: Reservoir computing is composed of an input layer, a reservoir and an output layer. Connection between input layer and reservoir and inside reservoir are random. Only the output layer is optimized based on a ridge regression. Adapted from Trouvain et al. (2020)

90 The input layer $u(t)$ is an M -dimensional vector, where M is the number of input time series, which
 91 corresponds to the values of the input time series at time t where $t = 1, \dots, T$. The reservoir layer $x(t)$
 92 is an N_{res} -dimensional vector where N_{res} is the number of nodes in the reservoir. The value $x(t)$ is
 93 defined as follow:

$$x(t + 1) = (1 - \alpha)x(t) + \alpha \tanh(Wx(t) + W_{in}u(t + 1)) \quad (1)$$

94 The leaking rate $\alpha \in [0, 1]$ defines the update rate of the nodes. The closer α is to 1, the more the
 95 reservoir is sensitive to new inputs $u(t)$. Therefore, the reservoir state at time $t + 1$ denoted $x(t + 1)$
 96 depends on the reservoir state at the previous time $x(t)$ and the new inputs $u(t + 1)$. Both W_{in} and W
 97 are random matrices of size $N_{res} \times M$ and $N_{res} \times N_{res}$ respectively.

98 W_{in} is a matrix (usually sparse) generated using a Bernoulli (bimodal) distribution where each value
 99 can be either $-I_{scale}(m)$ or $I_{scale}(m)$ with an equal probability where $m = 1, \dots, M$ corresponds to a
 100 given feature in the input layer. The input scaling, denoted I_{scale} , is a hyperparameter coefficient
 101 common to all features from the input layer or specific to each feature m . In that case, the more
 102 important the feature is, the greater should be its input scaling. W is a matrix (usually sparse) where
 103 values are generated from a Gaussian distribution $\mathcal{N}(0, 1)$. Then, the W matrix is scaled according to
 104 the defined spectral radius, a hyperparameter defining the highest eigen value of W .

105 The final layer is a linear regression with ridge penalization where the explanatory features are the
 106 reservoir state and the variable to be explained is the outcome to predict such that:

$$W_{out} = YX^T(XX^T + \lambda I)^{-1}$$

107 Where $x(t)$ and $y(t)$ are accumulated in X and Y respectively such that:

$$X = \begin{bmatrix} x(1) \\ x(2) \\ \vdots \\ x(T) \end{bmatrix} \text{ and } Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(T) \end{bmatrix}$$

108 The parameter λ is the ridge penalization which aims to prevent overfitting. Additionally, one can also
 109 connect the input layer to the output layer to the reservoir nodes. In that case, X is the accumulation
 110 of both such that :

$$X = \begin{bmatrix} x(1), u(1) \\ x(2), u(2) \\ \vdots \\ x(T), u(T) \end{bmatrix} \text{ and } Y = \begin{bmatrix} y(1) \\ y(2) \\ \vdots \\ y(T) \end{bmatrix}$$

111 Overall, there are four main hyperparameters to be chosen by the user: i) the leaking rate which
 112 defines the memory of the RC, ii) the input scaling which defines the relative importance of the
 113 features, iii) the spectral radius which defines the connections of the neurons inside the reservoir
 114 which in turn defines the degree of non-linear combination of features, and iv) the ridge penalization
 115 which controls the degree of overfitting. The choice of hyperparameters often requires the user to
 116 evaluate the performance of different combinations of hyperparameters on a validation set before
 117 selecting the optimal combination to forecast on the test set.

118 3 Usage workflow

119 In this section, we will cover the basics of `reservoirnet` use including installation, classification and
 120 regression. A more in depth description is provided in Section 4 with the covid-19 forecast use case.

121 3.1 Installation

122 `reservoirnet` is an R package making the Python module `ReservoirPy` easily callable from R using
 123 `reticulate` R package Ushey, Allaire, and Tang (2024). It is available on CRAN (see <https://cran.r-project.org/package=reservoirnet>) and can be installed using:

```
# Install reservoirnet package from CRAN
install.packages("reservoirnet")
```

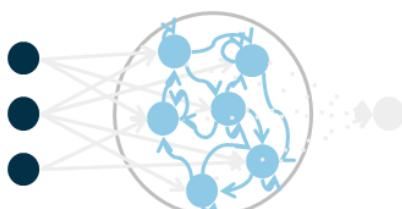
125 Reservoir Computing (RC) is well suited to both regression and classification tasks. We will introduce
 126 a simple example for both task.

Input layer :X



Instantiate reservoir :

```
reservoir <- createNode(nodeType = "Reservoir")
```



Instantiate output layer :

```
readout <- createNode(nodeType = "Ridge")
```



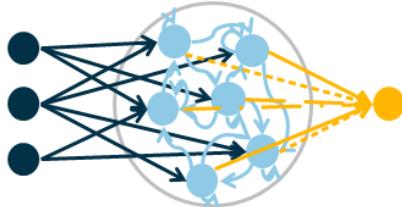
Build model :

```
model <- reservoir %>>% readout
```



Fit model :

```
fit <- reservoirR_fit(node = model,
  X = X,
  Y = Y)
```



Forecast :

```
predict_seq(node = fit$fit, X = X)
```

Figure 2: Workflow of reservoirnet.

127 **3.2 Package workflow overview**

128 The workflow of `reservoirnet` is described in Figure 2. A reservoir model is created by the association
129 of an input layer (a matrix), a reservoir, and an output layer. Both the reservoir and the output layer
130 are created using the function `reservoirnet::createNode()` by specifying the node type (i.e., either
131 `Reservoir` or `Ridge`).

132 This function accepts several arguments to specify the hyperparameters of the reservoir and will be
133 detailed in future sections. After the reservoir and output layer are created, they can be connected
134 using the `%>>%` operator, a specific pipe operator dedicated to `reservoirnet`. The model can then be
135 fitted using `reservoirR_fit()` and used to make predictions on a new dataset using `predict_seq()`.

136 **3.3 Basic regression use-case**

137 **3.3.1 Covid-19 data**

138 In this first use-case, we will introduce the fundamental usage of the `reservoirnet` package. This
139 demonstration will be conducted using the COVID-19 dataset that is included within the package.
140 These data encompass hospitalization, positive RT-PCR (Reverse Transcription Polymerase Chain
141 Reaction) results, and overall RT-PCR data sourced from Santé Publique France, which are publicly
142 available on `data.gouv.fr` (for further details, refer to `help(dfCovid)`). Our primary objective is to
143 predict the number of hospitalized patients 14 days into the future. To accomplish this, we will
144 initially train our model on data preceding the date of January 1, 2022, and subsequently apply it to
145 forecast values using the subsequent dataset.

146 We can proceed by loading useful packages - namely `ggplot2` Wickham, Navarro, and Pedersen
147 (2018) and `dplyr` Wickham et al. (2023), data and define the task:

```
# Load usefull packages
library(dplyr)
library(ggplot2)
library(patchwork)
library(reservoirnet)

# load dfCovid data from the reservoirnet package which contains Covid data
data("dfCovid")
# Set the forecast horizon to 14 days
dist_forecast = 14
# Set the train-test split to 2022-01-01
traintest_date = as.Date("2022-01-01")
```

148 Due to the substantial fluctuations observed in both RT-PCR metrics, our initial step involves applying
149 a moving average computation over the most recent 7-day periods for these features. Additionally,
150 we augment the dataset by introducing an `outcome` column and an `outcomeDate` column, which
151 will serve as valuable inputs for model training. Moreover, we calculate the `outcome_deriv` as the
152 difference between the `outcome` and the number of hospitalized patients (`hosp`), representing the
153 variation in hospitalization in relation to the current count of hospitalized individuals. The resulting
154 smoothed data is visualized in Figure 3.

```
dfOutcome <- dfCovid %>%
  # outcome at 14 days
  mutate(outcome = lead(x = hosp, n = dist_forecast),
  # Create a new column 'outcome' which contains the number of hospitalizations ('hosp')
  # shifted forward by 'dist_forecast' days (14 days). This represents the outcome we want
```

```

    outcomeDate = date + dist_forecast,
    # Create a new column 'outcomeDate' which is the current date plus the forecast period (11 days)

    outcome_deriv = outcome - hosp) %>%
    # Create a new column 'outcome_deriv' which is the difference between the predicted outcome and actual outcome.
    # This represents the change in hospitalizations over the forecast period.

    # rolling average for tested and positive_pcr
    mutate_at(.vars = c("Positive", "Tested"),
              .funs = function(x) slider::slide_dbl(.x = x,
                                                    .before = 6,
                                                    .f = mean))
    # Apply a rolling mean (7-day average) to the 'Positive' and 'Tested' columns.
    # The 'slider::slide_dbl' function is used to calculate the mean over a window of 7 days.
    # This smooths out daily fluctuations and provides a better trend indicator.

```

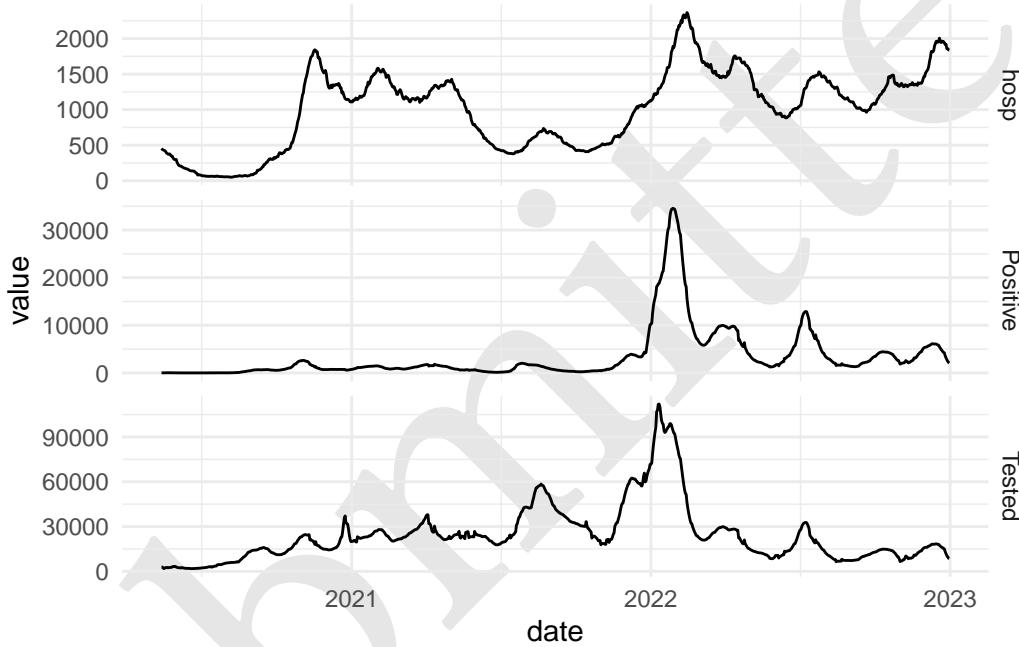


Figure 3: Hospitalizations, number of positive RT-PCR and number of RT-PCR of Bordeaux University Hospital.

155 3.3.2 First reservoir

156 Setting a reservoir is done with the `createNode()` function. The important hyperparameters are the
157 following :

- 158 • Number of nodes (`units`) : it corresponds to the number of nodes inside the reservoir. Usually,
159 the more the better, but more nodes increases the computation time.
- 160 • Leaking rate (`lr`) : the leaking rate corresponds to the balance between the new inputs and the
161 previous state. A leaking rate of 1 only consider information from new inputs.
- 162 • Spectral radius (`sr`): the spectral radius is the maximum absolute eigenvalue of the reservoir
163 connectivity matrix. A small spectral radius induces stable dynamics inside the reservoir, a
164 high spectral radius induces a chaotic regime inside the reservoir.

- 165 • Input scaling (`input_scaling`): the input scaling is a gain applied to the input features of the
 166 reservoir.
 167 • Warmup (`warmup`) : it corresponds to the number of time step during which the data are
 168 propagating into the reservoir but not used to fit the output layer. This hyperparameter is set
 169 in the `reservoirR_fit()` function.

170 In addition, we can set the seed (`seed`). Because the reservoir connections are set at random, setting
 171 the seed is a good approach to ensure reproducibility.

172 For this part of the tutorial, we will set the hyperparameter at a given value. Hyperparameter
 173 optimization will be detailed at Section 4.

```
# Create a reservoir computing node using the 'createNode' function from the reservoirnet package.

# Arguments:
# - nodeType = "Reservoir": Specify the type of node to be a reservoir.
# - seed = 1: Set the seed for reproducibility, ensuring consistent results when the model is run.
# - units = 500: Set the number of reservoir units (neurons) to 500.
# - lr = 0.7: Set the leakage rate (lr) of the reservoir, which controls how quickly the reservoir forgets past inputs.
# - sr = 1: Set the spectral radius (sr) of the reservoir, which influences the stability and memory properties.
# - input_scaling = 1: Set the input scaling factor, which scales the input signal before it is fed into the reservoir.

reservoir <- reservoirnet::createNode(nodeType = "Reservoir",
                                       seed = 1,
                                       units = 500,
                                       lr = 0.7,
                                       sr = 1,
                                       input_scaling = 1)
```

174 Then we can feed the data to the reservoir and see the activation state of the reservoir $x(t)$. To do so,
 175 we first prepare the data and transform it to a matrix.

```
## select explanatory features of the train set and transform it to an array
X <- dfOutcome %>%
  filter(outcomeDate < traintest_date) %>%
  select(hosp, Positive, Tested) %>%
  as.matrix()
```

176 Then we run the `predict_seq()` function. It takes as input a node (i.e a reservoir or a reservoir
 177 associated with an output layer) and the feature matrix.

```
# Generate the state of the reservoir using the 'predict_seq' function from the reservoirnet package.

# Arguments:
# - node = reservoir: The reservoir computing node created earlier.
# - X = X: The input data matrix containing the features 'hosp', 'Positive', and 'Tested'.
# The function computes the state of the reservoir for each time step in the input sequence,
# effectively transforming the input data into the reservoir's high-dimensional state space.

reservoir_state <- predict_seq(node = reservoir, X = X)
```

178 Now we can visualize node activation using the `plot()` function presented at Figure 4 .

```
# Plot the reservoir state activation over time
plot(reservoir_state)
```

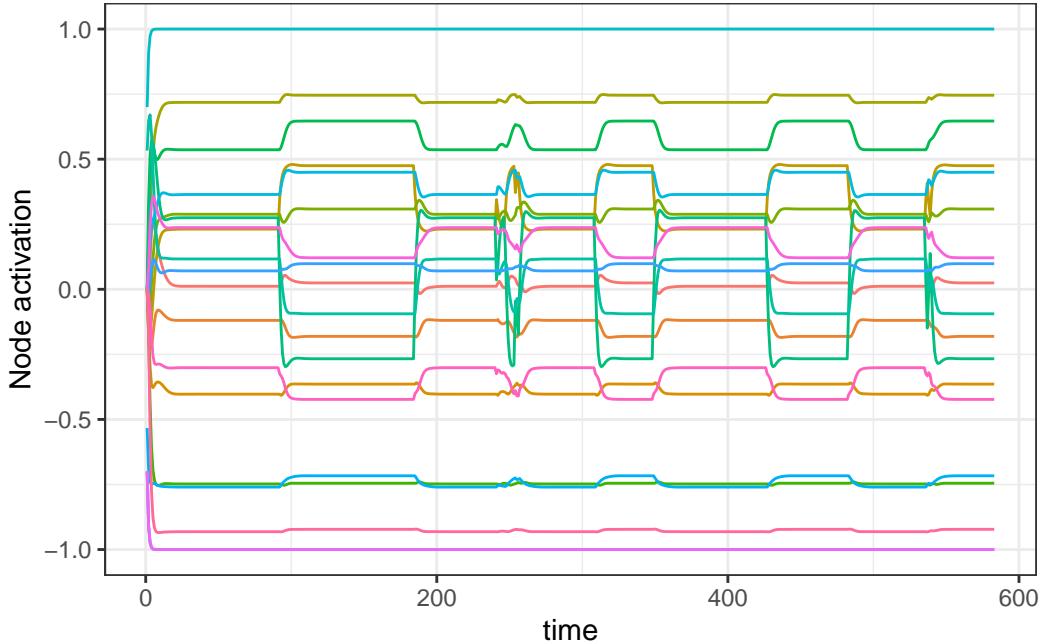


Figure 4: 20 random nodes activation over time.

179 Numerous nodes within the system exhibit a consistent equilibrium state. The challenge arises when
 180 the output layer attempts to extract knowledge from these nodes, as they do not convey meaningful
 181 information. This issue can be attributed to the disparate scales of the features. To address this
 182 concern, a practical approach involves normalizing the features by dividing each of them by their
 183 respective maximum values, thereby scaling them within the range of -1 to 1 by dividing by the
 184 maximum of the absolute value. Of note, here the features will be scaled between 0 and 1 because all
 185 features are positive.

```
# Standardise features by dividing by the maximum value can improve performance
# After standardisation, all features are on a similar scale which helps RC
stand_max <- function(x) return(x/max(abs(x)))
# scaled features
Xstand <- dfOutcome %>%
  filter(date < traintest_date) %>%
  select(hosp, Positive, Tested) %>%
  mutate_all(.funs = stand_max) %>%
  as.matrix() %>%
  as.array()
```

186 We then feed them to the reservoir and plot the node activation again. Compared to Figure 4, the
 187 obtained node activation at Figure 5 shows interesting trend outputs as no node seems saturated.

```
# feed the scaled features to the reservoir
reservoir_state_stand <- predict_seq(node = reservoir,
                                       X = Xstand,
                                       reset = TRUE)

# plot the output
plot(reservoir_state_stand)
```

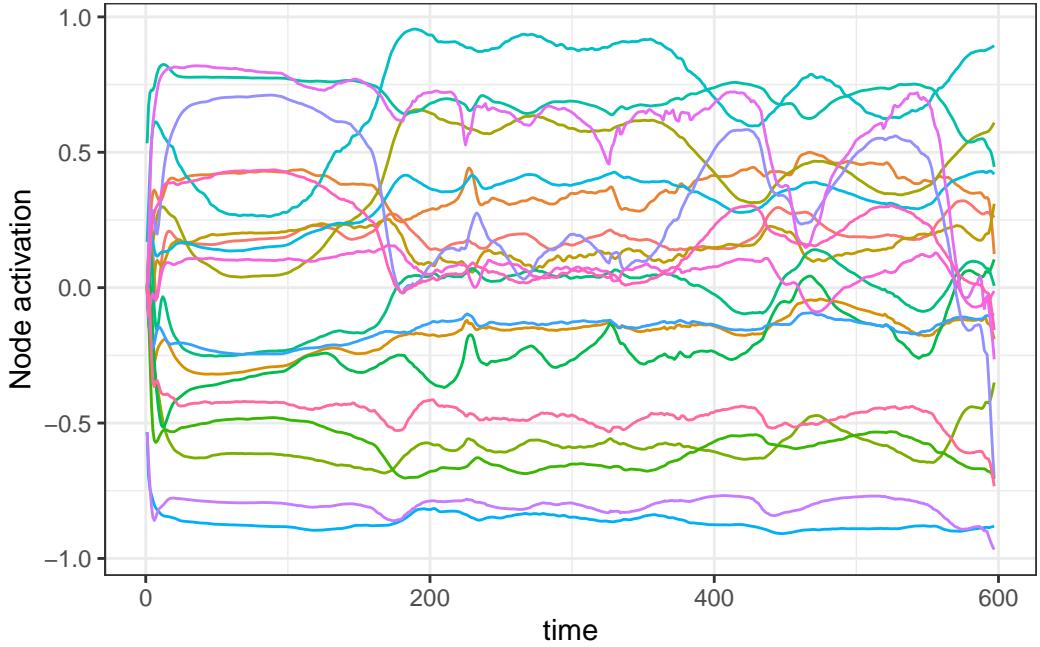


Figure 5: 20 random node activation over time. Scaled features.

188 3.3.3 Forecast

189 In order to train the reservoir, we should train the last layer which linearly combines the neuron's
 190 output.

191 3.3.3.1 Set the ESN

192 Initially, we establish the output layer with the `createNode()` function, incorporating a ridge penalty
 193 set at `1e3`. It's important to note that this hyperparameter can be subject to optimization, a topic
 194 that will be explored in Section 4. This parameter plays a pivotal role in fine-tuning the model's
 195 conformity to the data. When set excessively high, the risk of underfitting arises, whereas setting
 196 it too low can lead to overfitting. We connect the output layer to the reservoir, with the `link()`
 197 function, making the model ready to be trained.

```
readout <- reservoirnet::createNode(nodeType = "Ridge",
                                     ridge = 1e3)
# Create a readout node using ridge regression with the 'createNode' function from the reservoirnet
# Arguments:
# - nodeType = "Ridge": Specify the type of node to be a ridge regression readout.
# - ridge = 1e3: Set the regularization parameter (ridge) for the ridge regression to 1000.
# Ridge regression is used to prevent overfitting by adding a penalty on the size of the coefficients.

model <- reservoirnet::link(reservoir, readout)
# Link the reservoir and readout nodes to form a complete reservoir computing model.
# The 'link' function connects the high-dimensional state generated by the reservoir to the readout
# allowing the model to learn the mapping from the reservoir states to the target outputs.
```

198 3.3.3.2 Set the data

199 First we separate the train set on which we will learn the ridge coefficients and the test set on which

200 we will make the forecast. We define the train set to be all the data before 2022-01-01 and the test
201 data to be all the data to have forecast both on train and test sets.

```
# Perform some data management to isolate train and test sets
# train set
dftrain <- dfOutcome %>% filter(outcomeDate <= traintest_date)
yTrain <- dftrain %>% select(outcome)
yTrain_variation <- dftrain %>% select(outcome_deriv)
xTrain <- dftrain %>% select(hosp, Positive, Tested)
# test set
xTest <- dfOutcome %>% select(hosp, Positive, Tested)
```

202 We standardize with the same formula as seen before. We learn the standardization on the training
203 set and apply it on the test set. Then we convert the dataframe to matrix.

```
# copy train and test sets
xTrainstand <- xTrain
xTeststand <- xTest
# standardise based on training set values
ls_fct_stand <- apply(xTrain,
                       MARGIN = 2,
                       FUN = function(x) function(feature) return(feature/(max(x))))
lapply(X = names(ls_fct_stand),
       FUN = function(x){
         xTrainstand[,x] <- ls_fct_stand[[x]](feature = xTrain[,x])
         xTeststand[,x] <- ls_fct_stand[[x]](feature = xTest[,x])
         return()
       })
# convert to array
lsdf <- lapply(list(yTrain = yTrain,
                     yTrain_variation = yTrain_variation,
                     xTrain = xTrainstand,
                     xTest = xTeststand),
               function(x) as.matrix(x))
```

204 3.3.3.3 Train the model and predict

205 We then feed the reservoir with the train set using the `reservoirR_fit()` function. To do so, we set
206 a `warmup` of 30 days during which the data are propagating into the reservoir but not used to fit the
207 output layer.

```
### train the reservoir ridge output
fit <- reservoirnet::reservoirR_fit(node = model,
                                      X = lsdf$xTrain,
                                      Y = lsdf$yTrain,
                                      warmup = 30,
                                      reset = TRUE)
```

208 Now that the ridge layer is trained, we can forecast using the `predict_seq()` function. We set the
209 parameter `reset` to `TRUE` in order to clean the reservoir from the data used by the training set.

```
# Forecast with the trained reservoir on the test data
vec_pred <- reservoirnet::predict_seq(node = fit$fit,
```

```

X = lsdf$xTest,
reset = TRUE)

# Make figure to represent forecast on the train and test sets.

dfOutcome %>%
  mutate(pred = vec_pred) %>%
  na.omit() %>%
  ggplot(mapping = aes(x = outcomeDate)) +
  geom_line(mapping = aes(y = outcome,
                           color = "observed")) +
  geom_line(mapping = aes(y = pred,
                           color = "forecast")) +
  annotate("rect",
           xmin = traintest_date,
           xmax = max(dfOutcome$outcomeDate, na.rm = T),
           ymin = 0,
           ymax = max(dfOutcome$outcome, na.rm = T)*1.1,
           alpha = .2) +
  annotate("text", label = "Test set", x = as.Date("2022-08-01"), y = 2200, size = 7) +
  annotate("text", label = "Train set", x = as.Date("2021-03-01"), y = 2200, size = 7) +
  scale_color_manual(values = c("#3772ff", "#080708")) +
  theme_minimal() +
  labs(color = "", x = "Date", y = "Hospitalizations")

```

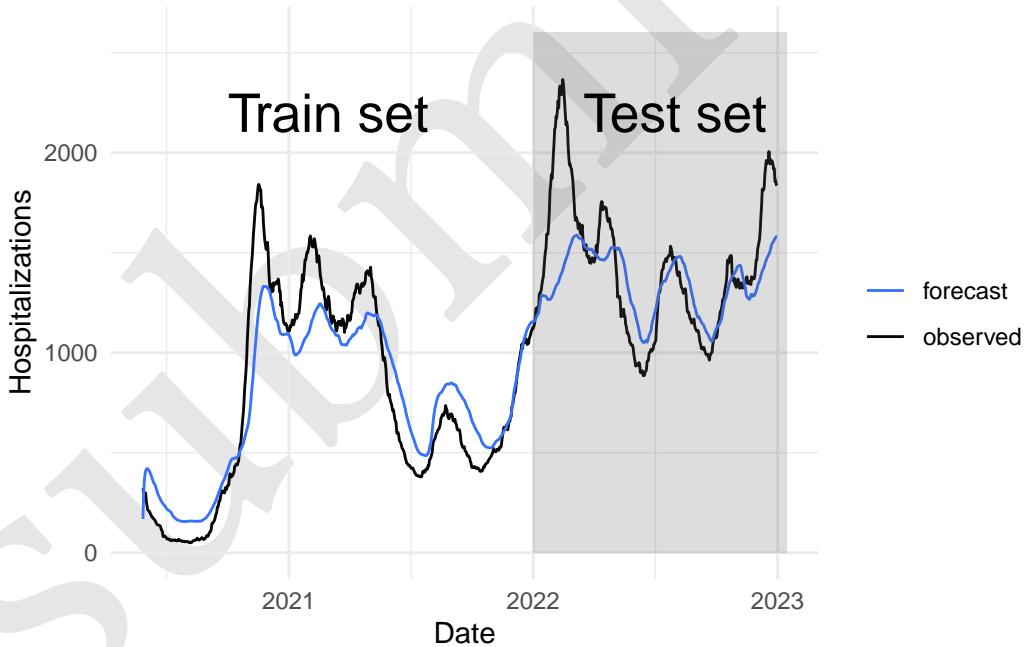


Figure 6: Forecast

We observe that the model forecast at Figure 6 is not fully accurate, both on the test set and the train set. In that case, one option could be to reduce ridge penalization to fit more closely the data, the optimization of ridge hyperparameter will be discussed at Section 4. Another possibility is to ease the learning of the algorithm by forecasting the variation of the hospitalization instead of

214 the number of hospitalized patients. For that step, we will learn on the `outcome_deriv` contained
 215 in `yTrain_variation` data which is defined outcome as `outcome_deriv = outcome - hosp`. As
 216 depicted at Figure 7, this strategy improved the model forecast.

```

## Fit reservoir on outcome variation instead of raw outcome
fit2 <- reservoirnet::reservoirR_fit(node = model,
                                      X = lsdf$xTrain,
                                      Y = lsdf$yTrain_variation,
                                      warmup = 30,
                                      reset = TRUE)

## Get the forecast on the test set
vec_pred2_variation <- reservoirnet::predict_seq(node = fit2$fit,
                                                   X = lsdf$xTest,
                                                   reset = TRUE)

## Transform the outcome variation forecast into hospitalization forecast
vec_pred2 <- vec_pred2_variation + xTest$hosp

## Plot the results
dfOutcome %>%
  mutate(Raw = vec_pred,
        Variation = vec_pred2) %>%
  tidyr::pivot_longer(cols = c(Raw, Variation),
                      names_to = "Outcome_type",
                      values_to = "Forecast") %>%
  na.omit() %>%
  ggplot(mapping = aes(x = outcomeDate)) +
  geom_line(mapping = aes(y = outcome,
                           color = "observed")) +
  geom_line(mapping = aes(y = Forecast,
                           color = "Forecast")) +
  annotate("rect",
           xmin = traintest_date,
           xmax = max(dfOutcome$outcomeDate, na.rm = T),
           ymin = 0,
           ymax = max(dfOutcome$outcome, na.rm = T)*1.1,
           alpha = .2) +
  annotate("text", label = "Test set", x = as.Date("2022-08-01"), y = 2200, size = 5) +
  annotate("text", label = "Train set", x = as.Date("2021-03-01"), y = 2200, size = 5) +
  facet_wrap(Outcome_type ~ .,
             labeller = label_bquote(cols = "Outcome" : .(Outcome_type))) +
  scale_color_manual(values = c("#3772ff", "#080708")) +
  theme_minimal() +
  theme(legend.position = "bottom") +
  labs(color = "", x = "Date", y = "Hospitalizations")

```

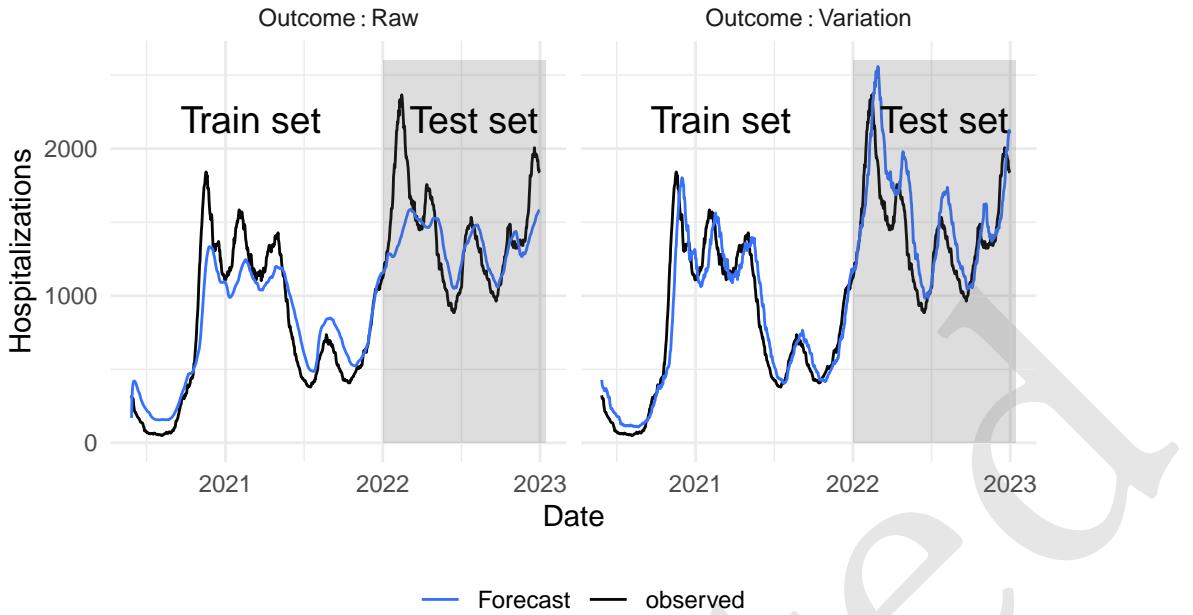


Figure 7: Covid-19 hospitalizations forecast. The model is either trained to forecast the number of hospitalizations (denoted Raw) or the variation of the hospitalizations compared to current level of hospitalisation (denoted Variation)

217 3.4 Classification

218 3.4.1 The Japanese vowel dataset

219 This example is largely inspired from the [classification tutorial of reservoirpy](#). To illustrate the
 220 classification task, we will use the Japanese vowel dataset (Kudo, Toyama, and Shimbo (1999)). The
 221 data can be loaded from `reservoirnet` as follow :

```
# Get the Japanese vowels dataset using the 'generate_data' function from the reservoirnet package
# The dataset contains preprocessed features and labels for classification.
# Then we isolate train and test sets
japanese_vowels <- reservoirnet::generate_data(dataset = "japanese_vowels")[[1]]
X_train <- japanese_vowels$X_train
Y_train <- japanese_vowels$Y_train
X_test <- japanese_vowels$X_test
Y_test <- japanese_vowels$Y_test
```

222 The dataset comprises 640 vocalizations of the Japanese vowel æ, contributed by nine distinct
 223 speakers. Each vocalization represents a time series spanning between 7 and 29 time steps, encoded
 224 as a 12-dimensional vector denoting the Linear Prediction Coefficients (LPC). A visual representation
 225 of six distinct utterances from the test set, originating from three different speakers, is depicted in
 226 Figure 8.

227 The primary objective involves the attribution of each utterance to its respective speaker, this is
 228 denoted as classification or sequence-to-vector encoding. The secondary objective involves the
 229 attribution of each time step of each utterance to its speaker, this is denoted as transduction or
 230 sequence-to-sequence encoding.

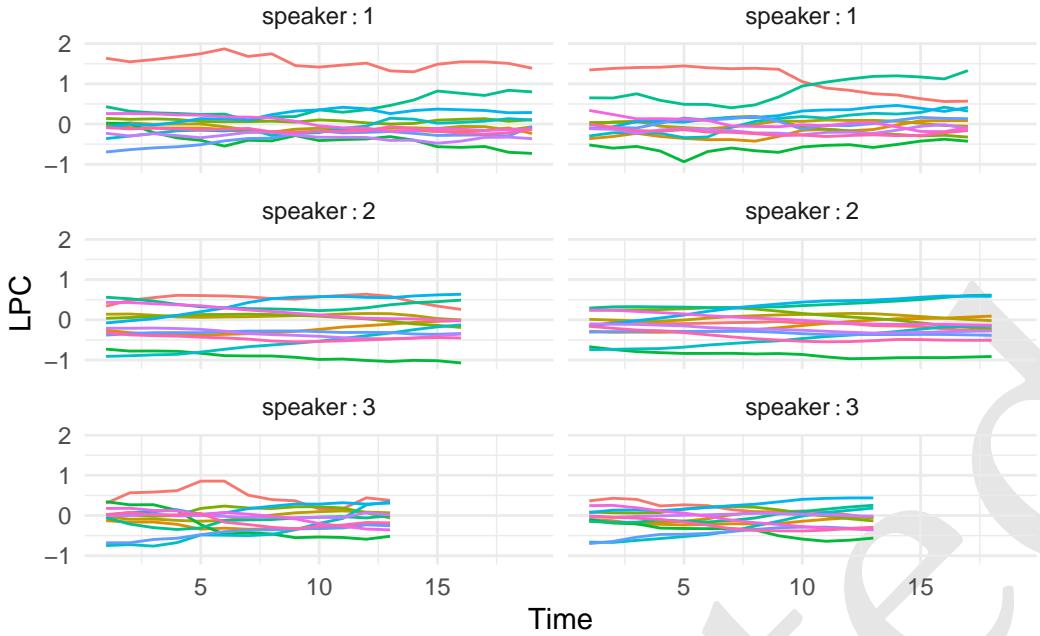


Figure 8: Vowel dataset, sample with 3 speakers and 2 utterance each.

231 3.4.2 Classification (sequence-to-vector model)

232 The first approach is the sequence-to-vector encoding. For this task we aim to predict the speaker of
 233 the whole utterance (i.e the label is assigned to the whole sequence). We first start by creating the
 234 reservoir and the output layer using `createNode()` function.

```
reservoir <- reservoirnet::createNode("Reservoir", units = 500,
                                         lr=0.1, sr=0.9,
                                         seed = 1)
# Create a reservoir computing node with 500 units using the 'createNode' function from the reservoirnet package.
# Arguments:
# - units = 500: Set the number of reservoir units (neurons) to 500.
# - lr = 0.1: Set the leakage rate (lr) of the reservoir to 0.1, controlling how quickly the reservoir forgets past information.
# - sr = 0.9: Set the spectral radius (sr) of the reservoir to 0.9, influencing the stability and dynamics of the reservoir.
# - seed = 1: Set the seed for reproducibility, ensuring consistent results when the model is run multiple times.
readout <- reservoirnet::createNode("Ridge", ridge=1e-6)
# Create a readout node using ridge regression with the 'createNode' function from the reservoirnet package.
# Arguments:
# - ridge = 1e-6: Set the regularization parameter (ridge) for the ridge regression to 1e-6.
# Ridge regression is used to prevent overfitting by adding a penalty on the size of the coefficients.
```

235 To perform this task, we need to modify the training and testing processes. Leveraging the inherent
 236 inertia of the reservoir, information from preceding time steps is preserved, effectively endowing the
 237 RC with a form of memory. Consequently, the final state vector encapsulates insights gathered from
 238 all antecedent states. In the context of the sequence-to-vector encoding task, only this ultimate state
 239 is employed. This process is executed as follows:

```
states_train = list()
k <- 1
for (x in X_train) {
```

```

# Loop over each training sample in X_train.
states <- reservoirnet::predict_seq(node = reservoir, X = x,
                                     reset=TRUE)
# Generate the reservoir states for the current training sample using the 'predict_seq' function
states_train[[k]] <- t(as.matrix(states[nrow(states),]))
# Extract the final reservoir state for the current sample and store it in 'states_train'.
k <- k+1
}

```

240 Then we can train the readout based on this last state vector. In that case, Y_train contains a single
 241 label for each utterance.

```
# Fit the reservoir using the last state vector (each observation is the whole vowel sequence)
res <- reservoirnet::reservoirR_fit(readout,X = states_train, Y = Y_train)
```

242 The prediction is also modified using only the final state :

```

# The operation is repeated for the test set :
# - the last state vector is extracted for each utterance
# - the model is used to forecast the utterance from this last state vector
Y_pred <- list()
k <- 1
for (x in X_test) {
  states <- reservoirnet::predict_seq(node = reservoir, X = x,
                                       reset=TRUE)
  y <- reservoirnet::predict_seq(node = readout,
                                 X = as.array(states[nrow(states),]))
  Y_pred[[k]] <- y
  k <- k+1
}

```

243 Figure 9 shows the prediction for the 6 utterances depicted at Figure 8 where the model correctly
 244 identifies the speaker.

```

# A figure represents the performance on the test set
dfplotseqtovec <- lapply(vec_sample,
                          FUN = function(i){
    speaker <- which(Y_test[[i]][1,] == 1)
    Y_pred[[i]] %>%
      as.data.frame() %>%
      tidyverse::pivot_longer(cols = everything(),
                             names_to = "pred_speaker",
                             values_to = "prediction") %>%
      mutate(pred_speaker = gsub(x = pred_speaker,
                                 pattern = "V", ""))
    mutate(speaker = speaker, .before = 1,
          utterance = i,
          target = speaker == pred_speaker) %>%
    return()
  }) %>%
bind_rows()

ggplot(dfplotseqtovec,
```

```

mapping = aes(x = pred_speaker,
              y = prediction,
              fill = target)) +
geom_bar(stat = "identity") +
facet_wrap(utterance ~ speaker,
           labeller = label_bquote(cols = "speaker" : .(speaker)),
           ncol = 2) +
scale_fill_manual(values = c("#BDBDBD", "#A3CEF1")) +
theme_minimal() +
theme(legend.position = "none") +
labs(y = 'Score',
     x = "Speaker")

```

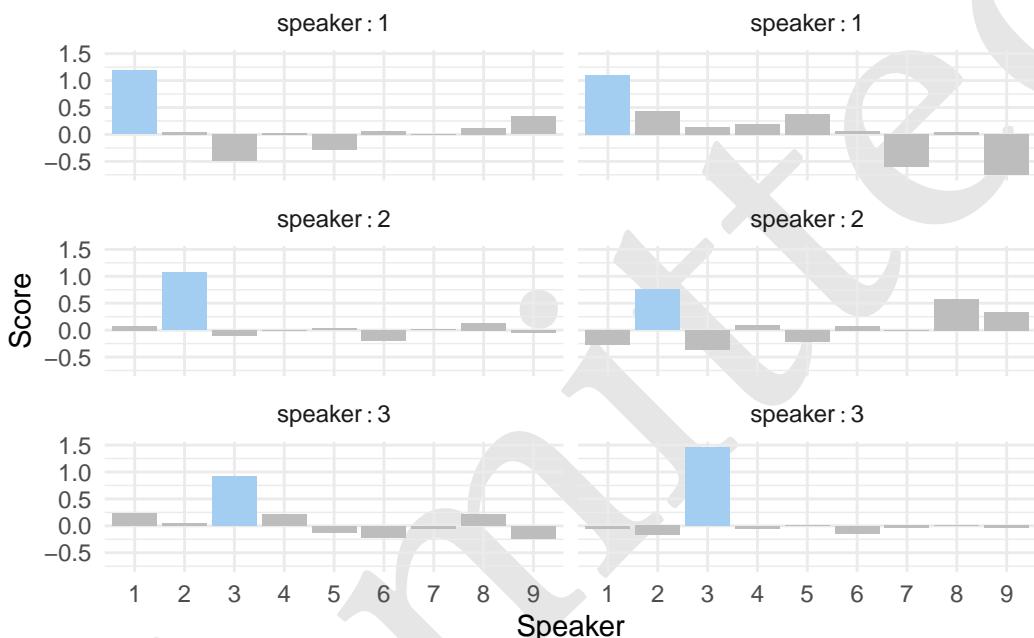


Figure 9: Prediction in a sequence-to-sequence approach 6 samples with 3 speakers and 2 utterance each. The speaker to predict is depicted in blue. For each of the 6 utterance, the model correctly identifies the speaker.

²⁴⁵ Then, we can also compute the overall accuracy :

```

# The overall accuracy is evaluated
accuracy <- function(pred, truth) mean(pred == truth)

Y_pred_class <- sapply(Y_pred,
                        FUN = function(x) apply(as.matrix(x), 1, which.max))
Y_test_class <- sapply(Y_test,
                        FUN = function(x) apply(as.matrix(x), 1, which.max))

score <- accuracy(pred = Y_test_class, truth = Y_pred_class)

print(paste0("Accuracy: ", round(score * 100, 3), "%"))

```

```
246 [1] "Accuracy: 92.703%"
```

247 3.4.3 Transduction (sequence-to-sequence model)

248 For this task, the goal is to predict the speaker for each time step of each utterance. The first
249 step is to get the data where the label is repeated for each time step. This is easily done with the
250 `repeat_targets` argument as follow :

```
# For this new task where we want to forecast for each time step (instead of each utterance)  
# we start by getting the data in the appropriate format  
# Then we split the train and test data  
japanese_vowels <- reservoirnet::generate_data(  
  dataset = "japanese_vowels",  
  repeat_targets=TRUE)$japanese_vowels  
X_train <- japanese_vowels$X_train  
Y_train <- japanese_vowels$Y_train  
X_test <- japanese_vowels$X_test  
Y_test <- japanese_vowels$Y_test
```

251 Then we can train a simple Echo State Network to solve this task. For this example we will connect
252 both the input layer and the reservoir layer to the readout layer which is performed by the `%>>%`
253 operator :

```
# Create an input, a reservoir and an output layers  
source <- createNode("Input")  
readout <- createNode("Ridge", ridge=1e-6)  
reservoir <- createNode("Reservoir", units = 500, lr=0.1, sr=0.9, seed = 1)  
# Connect the input layer to the reservoir and connect both the input layer and the reservoir to the readout  
model <- list(source %>>% reservoir, source) %>>% readout
```

254 We can then fit the model and predict the labels for the test data. The `reset` parameter is set to TRUE
255 to remove information from the reservoir from the training process.

```
# Fit the RC model  
model_fit <- reservoirnet::reservoirR_fit(node = model,  
                                         X = X_train,  
                                         Y = Y_train,  
                                         warmup = 2)  
  
# Predict with the fitted model  
Y_pred <- reservoirnet::predict_seq(node = model_fit$fit,  
                                      X = X_test,  
                                      reset = TRUE)
```

256 From the `Y_pred` and `Y_test` we represent at Figure 10 the predictions for the same patients as in
257 Figure 8.

```
# Make a graph with a label for each time of each utterance  
dfplotseqtoseq <- lapply(vec_sample,  
  FUN = function(i){  
    speaker <- which(Y_test[[i]][1,] == 1)  
    Y_pred[[i]] %>%  
      as.data.frame() %>%  
      tibble::rowid_to_column(var = "Time") %>%  
      tidyverse::pivot_longer(cols = -Time,
```

```
        names_to = "pred_speaker",
        values_to = "prediction") %>%
  mutate(pred_speaker = gsub(x = pred_speaker,
                             pattern = "V", ""),
         speaker = speaker,
         uterrance = i,
         .before = 1) %>%
  return()
}) %>%
bind_rows()

ggplot(dfplotseqtoseq, mapping = aes(x = Time,
                                       y = pred_speaker,
                                       fill = prediction)) +
  geom_tile() +
  facet_wrap(uterrance ~ speaker,
             labeller = label_bquote(cols = "speaker" : .(speaker)),
             ncol = 2) +
  scale_fill_gradient2(low = "#8ECAE6", high = "#FB8500", mid = "#023047",
                       midpoint = 0) +
  theme_minimal() +
  labs(y = 'Predicted speaker',
       fill = "Prediction score")
```

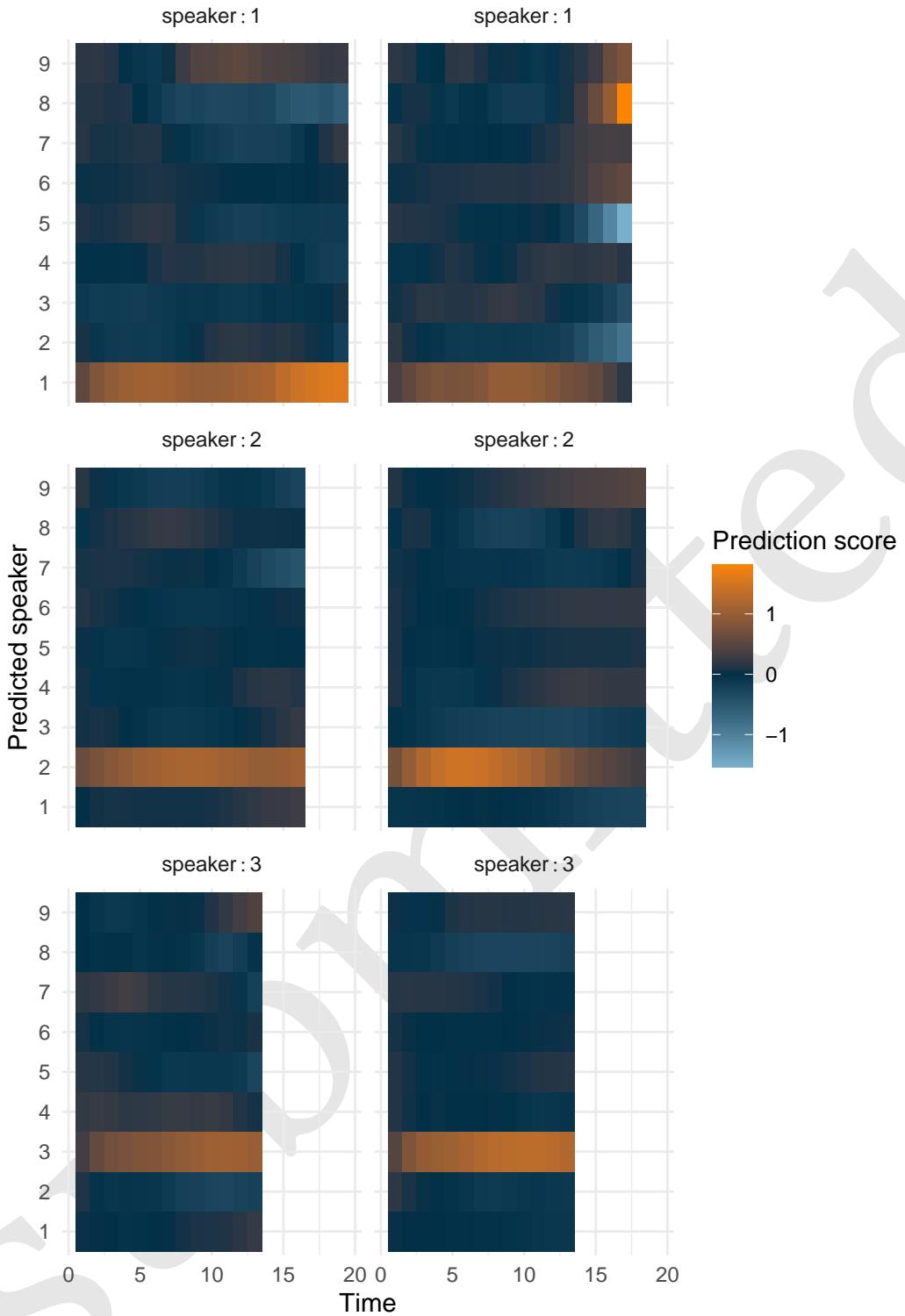


Figure 10: Prediction in a sequence-to-sequence approach 6 samples with 3 speakers and 2 utterance each. The higher the score of the speaker, the lighter the color.

258 For those 6 utterances, the model correctly identify the speaker for most of the time steps. We can
 259 then evaluate the overall accuracy of the model :

```
# Compute the accuracy
```

```

Y_pred_class <- sapply(Y_pred, FUN = function(x) apply(as.matrix(x),
1,
which.max))
Y_test_class <- sapply(Y_test, FUN = function(x) apply(as.matrix(x),
1,
which.max))
score <- accuracy(array(unlist(Y_pred_class)), array(unlist(Y_test_class)))

print(paste0("Accuracy: ", round(score * 100,3) ,"%"))

260 [1] "Accuracy: 92.456%"

```

261 4 Avanced case-study: Covid-19 hospitalizations forecast

262 4.1 Introduction

263 Since late 2020, millions of cases of SARS-CoV-2 infection have been documented across the globe
 264 (World Health Organisation 2020; COVID-19 Cumulative Infection Collaborators 2022; Carrat et al.
 265 2021). This ongoing pandemic has exerted significant strain on healthcare systems, resulting in a surge
 266 in hospitalizations. This surge, in turn, necessitated modifications to the healthcare infrastructure and
 267 gave rise to population-wide lockdown measures aimed at preventing the saturation of healthcare
 268 facilities (Simões et al. 2021; Hübner et al. 2020; Kim et al. 2020). The capacity to predict the
 269 trajectory of the epidemic on a regional scale is of paramount importance for effective healthcare
 270 system management.

271 Numerous COVID-19 forecasting algorithms have been proposed using different methods (e.g en-
 272 semble, deep learning, compartmental), yet none has proven entirely satisfactory (Cramer et al. 2022;
 273 Rahimi, Chen, and Gandomi 2021). In France, short-term forecasts with different methods have
 274 been evaluated with similar results (Paireau et al. 2022; Carvalho et al. 2021; Mohimont et al. 2021;
 275 Pottier 2021). In this context a machine learning algorithm based on linear regression with elastic-net
 276 penalization, leveraging both Electronic Health Records (EHRs) and public data, was implemented at
 277 Bordeaux University Hospital (Ferté et al. 2022). This model, which aimed at forecasting the number
 278 of hospitalized patients at 14 days, showed good performance but struggled to accurately anticipate
 279 dynamic shifts of the epidemic.

280 RC has been used in the context of covid-19 epidemic forecast (Kmet and Kmetova 2019; Liu et
 281 al. 2023; Ray, Chakraborty, and Ghosh 2021; Zhang et al. 2023; Ghosh et al. 2021). Among them,
 282 Ghosh et al. (2021), Liu et al. (2023) and Ray, Chakraborty, and Ghosh (2021) used it to forecast
 283 epidemic, Zhang et al. (2023) performed sentiment analysis and Kmet and Kmetova (2019) used
 284 it to solve optimal control related to vaccine. The evaluation of RC for epidemic forecast showed
 285 promising results in all approaches, being competitive with Long-Short Term Memory (LSTM) and
 286 Feed-Forward Neural Network (FFNN) in Ray, Chakraborty, and Ghosh (2021). However, the test
 287 period was short for Ghosh et al. (2021} (21 and 14 days) and Ray, Chakraborty, and Ghosh (2021)
 288 (86 days) making it difficult to evaluate the behavior of the methods during epidemic dynamic shift.
 289 This was not the case for Liu et al. (2023) (6 months) but they implemented daily ahead forecast
 290 which would be difficult to use to manage a hospital. Finally, all three implementations used only
 291 one time series as input whereas it has been shown that using different data sources could improve
 292 forecast Ferté et al. (2022). Therefore, it is still difficult to assess the usefulness of RC over a large
 293 period and using many time series as inputs.

294 RC can be viewed as an extension of penalized linear regression, where inputs undergo processing by a
 295 reservoir, introducing the capacity for memory and non-linear combinations. Given the effectiveness

296 of penalized linear regression in COVID-19 forecasting, as highlighted in Ferté et al. (2022), and the
297 promising results exhibited by RC in epidemic forecasting, as demonstrated in studies such as Ghosh
298 et al. (2021), Liu et al. (2023), and Ray, Chakraborty, and Ghosh (2021), we have opted to employ RC
299 for the prediction of hospitalizations at 14 days at the University Hospital of Bordeaux.

300 The primary aim of this study is to assess the performance of RC in this forecasting task. Secondary
301 objectives include (i) comparing the performance of RC with that of elastic-net penalized regression
302 (identified as the optimal model in Ferté et al. (2022)) and (ii) evaluating variations in RC performance
303 based on different architectural choices, such as the connection between the input layer and the
304 output layer, and the use of one input scaling per feature versus a common input scaling.

305 **4.2 Methods**

306 **4.2.1 Data**

307 The study utilized aggregated data spanning from May 16, 2020, to January 17, 2022, regarding
308 the COVID-19 epidemic in France, drawing from various sources to enhance forecasting accuracy.
309 These sources encompassed epidemiological statistics from Santé Publique France, weather data
310 from the National Oceanic and Atmospheric Administration (NOAA), both providing department-
311 level data (Smith, Lott, and Vose 2011; Etalab 2020) and Electronic Health Record (EHR) data from
312 the Bordeaux Hospital providing hospital-level data. All data were daily updated. Santé Publique
313 France data included information on hospitalizations, RT-PCR tests, positive RT-PCR results, variant
314 prevalence, and vaccination data, categorized by age groups. NOAA data contributed temperature,
315 wind speed, humidity, and dew point data, allowing for the computation of the COVID-19 Climate
316 Transmissibility Predict Index (Roumagnac et al. 2021). EHRs data included hospitalizations, ICU
317 admissions, ambulance service records, and emergency unit notes, with relevant COVID-19-related
318 concepts extracted from the notes. Data are discussed more in depth in Ferté et al. (2022).

319 First derivative over the last 7 days were computed to enrich model information. To take into account
320 measurement error and daily noise variation, data were smoothed using a local polynomial regression
321 with a span of 21 days. As previously described, input features were scaled between -1 and 1 by
322 dividing the observed value by the maximum of the absolute value of the given input feature.

323 All data are publicly available. Weather data can be obtained from Smith, Lott, and Vose (2011) using
324 R package `worldmet` (Carslaw 2023). [Vaccine data](#) can be downloaded from Etalab (2020). EHRs data
325 can be downloaded on dryad (Ferté et al. 2023). For privacy issues, publicly available EHRs data
326 below 10 patients were obfuscated to 0. For convenience, all data were downloaded, merged and
327 provided as replication material.

328 **4.2.2 Evaluation framework**

329 The task was to forecast 14 days ahead the number of hospitalized patients. As seen at Section 3.3,
330 we will train the model to predict the variation of hospitalization, denoted as \widehat{hosp} , defined as
331 $outcome_{t+14} = \widehat{hosp}_{t+14} - \widehat{hosp}_t$. Metrics computation and visualizations will be performed on the
332 predicted number of hospitalizations denoted as $\widehat{hosp}_{t+14} = outcome_{t+14} + \widehat{hosp}_t$.

333 The dataset was separated into two periods. First period from May 16, 2020 to March 1, 2021 served
334 to identify relevant hyperparameters. Remaining data was used to evaluate the model performance.

335 The performance of the model was evaluated according to several metrics:

- 336 • the mean absolute error : $MAE = mean(|\widehat{hosp}_{t+14} - hosp_{t+14}|)$.
- 337 • the median relative error : $MRE = median(|\frac{\widehat{hosp}_{t+14} - hosp_{t+14}}{hosp_{t+14}}|)$.

- the mean absolute error to baseline : $MAEB = \text{mean}(|\widehat{\text{hospt}_{t+14}} - \text{hospt}_{t+14}| - |\text{hospt}_t - \text{hospt}_{t+14}|)$.
- the median relative error to baseline : $MREB = \text{median}\left(\left|\frac{\widehat{\text{hospt}_{t+14}} - \text{hospt}_{t+14}}{\text{hospt}_t - \text{hospt}_{t+14}}\right|\right)$

Median was chosen over mean for *MRE* and *MREB* because those metrics tend to have extremely high values when the denominator is close to 0 (i.e when the number of hospitalized patients is close to 0 or the number of patients hospitalized at 14 days is close to the current number of hospitalized patients respectively). *MAEB* and *MREB* compare model performance to a baseline model which predicts the current number of hospitalized patients at 14 days. Those metrics help to determine the information added by the model and is a good baseline as covid-19 forecast model do not always outperform this basic forecast (Cramer et al. (2022)).

Because the outcome is obfuscated below 10 hospitalizations for privacy reason, we set both the outcome and the forecast to 10 when the observed value was 0 or the forecasted value was below 10 when evaluating the model performance.

4.2.3 Models

We compared RC to elastic-net penalized regression (denoted as *Enet*). Furthermore we evaluated RC based on several architectures. First we compared RC with a single input scaling common to all features and a RC with on specific input scaling per feature. Second we compared RC where the input layer is connected to the output layer in addition to the connection between reservoir and output layer. Therefore, five models were evaluated :

- Elastic-net penalized regression denoted *Enet*
- RC with a single input scaling and no connection between input and ouput layers denoted *Common IS R %»% O*
- RC with a single input scaling and connection between input and ouput layers denoted *Common IS I+R %»% O*
- RC with multiple input scaling and no connection between input and ouput layers denoted *Multiple IS R %»% O*
- RC with multiple input scaling and connection between input and ouput layers denoted *Multiple IS I+R %»% O*

Because of the randomness of the reservoir, we took the median forecast of 10 reservoir on the train set to evaluate the performance of a given hyperparameter set. On the test set we aggregated the forecast of 40 reservoirs, each of them having one of the 40 best hyperparameter sets found on the train set. In addition, because covid-19 hospitalization is a non-stationary process, models were re-trained everyday using all previous days. To ease computation burden, only one day over two was used to find hyperparameters on the training set.

4.2.4 Hyperparameter optimisation using random search

RC relies mainly on 4 hyperparameters including the leaking rate (i.e “memory” parameter), spectral radius (i.e “chaoticity” parameter), input scaling (i.e “feature gain” parameter) and ridge (i.e penalization parameter). Input scaling can be either, common to all features or specific to each feature which increases the number of hyperparameter by the number of features.

Following the notation from *glmnet* package (Friedman, Hastie, and Tibshirani 2010), elastic-net penalized linear regression relies on two hyperparameters, lambda (i.e the penalization parameter) and alpha (i.e the compromise between lasso and ridge penalty)

Hyperparameter were selected in the training set (i.e before March 1, 2021) using a wrapper approach and a random search sampler using 2000 samples for each model. The sampling distribution were

381 defined as follow :

- 382 • (RC) ridge and (Enet) lambda : log-uniform law defined between 1e-10 and 1e5
- 383 • (RC) input scaling and spectral radius : log-uniform law defined between 1e-5 and 1e5
- 384 • (RC) leaking rate : log-uniform law defined between 1e-3 and 1
- 385 • (Enet) alpha : uniform defined between 0 and 1

386 We provided large search space for all hyper-parameters. Search space was slightly reduced for
387 leaking rate based on previous results and because a leaking rate of 1e-3 already imply that new
388 inputs make the reservoir change really slowly which is not inline with the dynamic of covid-19 but
389 would be appropriate for an application where the phenomena to forecast has a slow dynamic.

390 Finally, we provided an additional Enet model similar to the one in Ferté et al. (2022) where alpha
391 was set to 0.5 and lambda was re-evaluated everyday in the test set based on previous data using the
392 cross-validation procedure provided by `glmnet`.

393 4.3 Results

394 The goal of this task is to predict 14 days ahead the hospitalization. Figure 11 shows both the training
395 set (i.e before 2021-03-01) and the test set where the blue curve correspond to the input features (first
396 derivatives are not shown) and the orange curves correspond to the outcome the model is trained
397 on (i.e the hospitalization variation) and the hospitalizations at 14 days on which the performance
398 metrics are computed. The figures outline that the relation between the input features and the
399 outcome evolve over time and that the time series is not stationary. For instance IPTCC (*Index*
400 *PREDICT de Transmissivité Climatique de la COVID-19*) seems correlated to the outcome except that
401 it completely miss the summer 2021 increase.

402 4.3.1 Hyperparameter selection

403 Figure 12 shows the hyperparameter optimisation using random search for the different RC architec-
404 tures. We observe that model with multiple input scaling achieved better performance on the train
405 set compared to model with single input scaling which is expected as they can adapt more closely to
406 the data thanks to specific input scaling for each feature.

407 As expected, we observe that the optimal leaking rate is above 1e-2 for all RC which is coherent with
408 the short term dynamic of covid-19 epidemic. Trend for other hyperparameter are less clear even
409 though best hyperparameter sets were close for RC with common input scaling and for RC with
410 multiple input scaling.

411 Figure 13 shows the hyperparameter search for RC with multiple input scaling and connected input
412 layer. We observe that the random search tend to favor high importance given to derivative of
413 positive RT-PCR (including the elderly) and the derivative of IPTCC. The rest of the feature do not
414 show clear pattern.

415 4.3.2 Forecast performance

416 Table 1 shows the performance on the test set. Best model according to all metrics was RC with
417 common input scaling and connection between input and output layers. Having one input scaling per
418 feature did not improve the model which might be due to low generalisability of the hyperparameter
419 of the training set to the test set due to non-stationarity. Additionally, connecting input layer to
420 output layer improved the model forecast. All RC models performed better than the elastic-net
421 model.

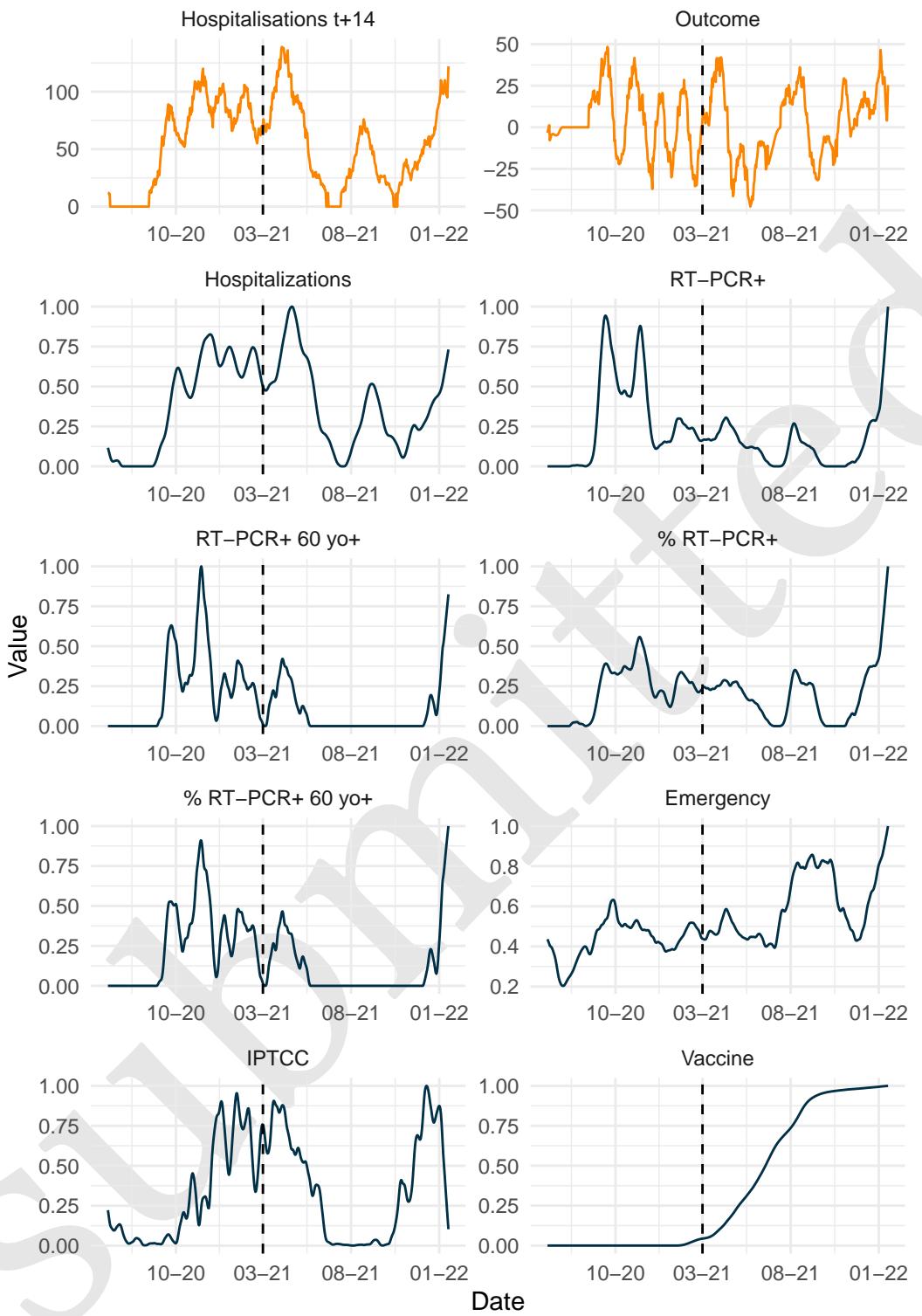


Figure 11: Covid-19 epidemic at BUH. Outcome of interest is presented in orange. Model is trained to forecast Outcome curve which corresponds to the difference between Hospitalisations at 14 days and current hospitalisations. Other features are scaled (divide by the maximum of the feature) represented in darkblue.

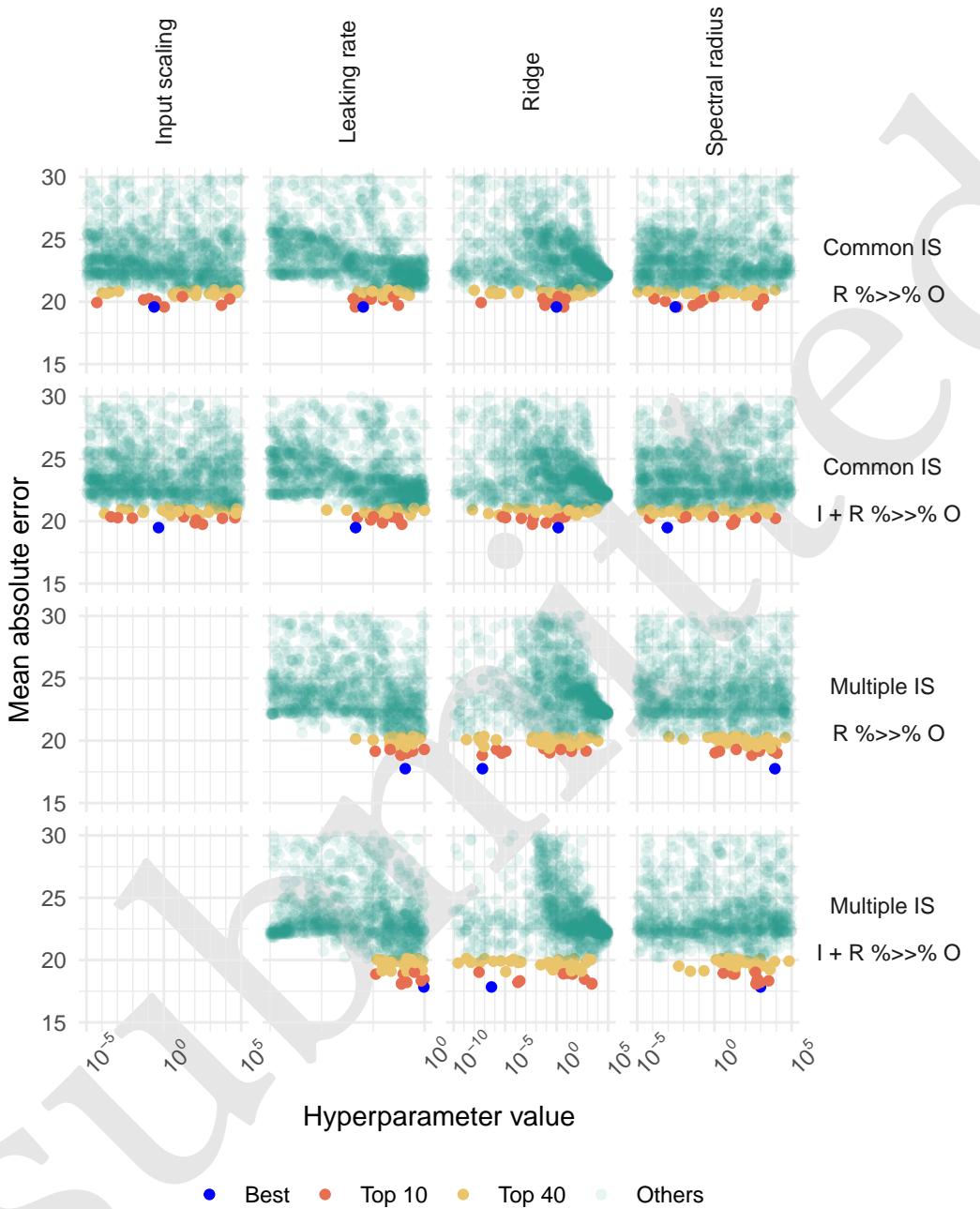


Figure 12: Hyperparameter evaluation on training set by random search. Hp sets with MAE above 30 were removed for clarity of visualisation.

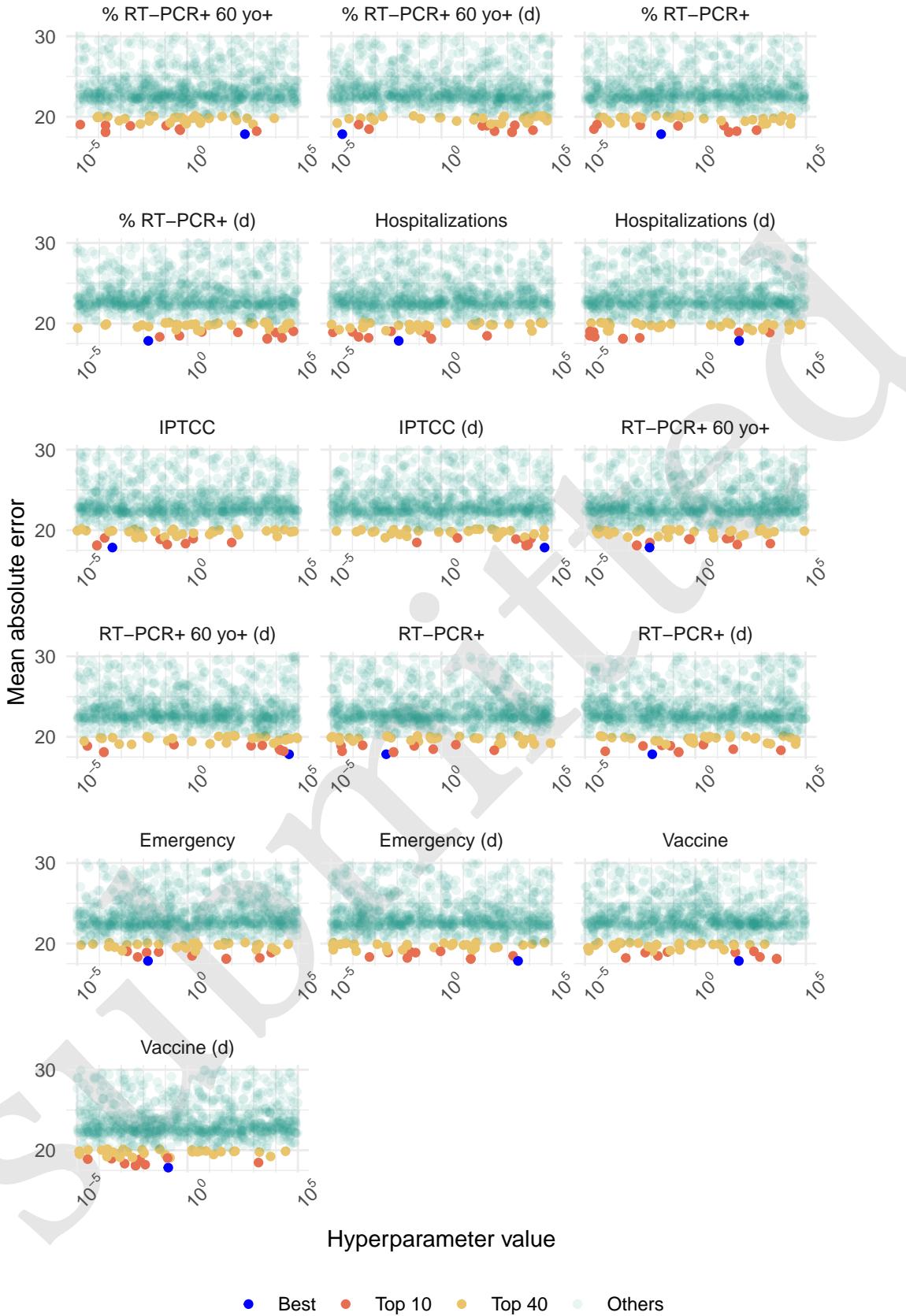


Figure 13: Hyperparameter evaluation on training set by random search of the model with multiple input scaling and no connection between input layer and output layer. Hp sets with MAE above 30 were removed for clarity of visualisation.

Table 1: Model performance with several reservoir configuration. For each setting, 40 reservoirs are computed and the forecast is the median of the 40 forecasts. Results show the performance metrics : MAE = Mean Absolute Error, MRE = Median Relative Error, MAEB = Mean Absolute Error to Baseline, MREB = Median Relative Error to Baseline.

Model Performance				
Model	MAE	MRE	MAEB	MREB
Common IS R %»% O	15.23	0.26	-3.50	0.85
Common IS I + R %»% O	14.84	0.26	-3.89	0.83
Multiple IS R %»% O	15.38	0.28	-3.35	0.82
Multiple IS I + R %»% O	15.25	0.28	-3.49	0.83
Elastic-net	16.40	0.29	-2.34	0.93

422 Figure 14 shows the forecast of the different models. We note that models struggle to accurately
423 forecast slope shifts. For instance, summer 2021 initial increase is partially predicted by all models
424 but its decrease is not well predicted. Winter 2021 increase is anticipated by all models but they tend
425 to overestimate it because of the rise of vaccine effect.

426 4.3.3 Number of model to aggregate

427 Figure 15 show the individual forecast for the 40 best sets of hyperparameters of each RC architecture.
428 Due to the internal random connection of the reservoir, we observe forecast stochasticity and relying
429 on only one forecast is unreliable. We explored the number of model needed at Figure 16 which
430 shows that after 10 models, forecast is stable and even 5 models for the simpler model with common
431 input scaling which rely on less hyperparamters.

432 4.3.4 Input feature importance

433 We compared the coefficients of the output layer estimated for the input layer and the reservoir
434 nodes. Additionally, we compared the coefficient given to the input layer by the output layer in the
435 reservoir and the coefficient estimated by the elastic-net model.

436 Figure 17 illustrates the ranking of input layer compared to all connections to the output layer,
437 including the 500 reservoir nodes and the 16 features of the input layer (excluding bias). The figure
438 shows that the model with common input scaling tends to assign less weight to input layer compared
439 to the model with multiple input scaling. This suggests that the reservoir with common input scaling
440 provides more information than the reservoir with multiple input scaling, which aligns with its better
441 performance, as shown in Table Table 1.

442 Furthermore, Figure 18 compares the coefficients assigned to input features by the elastic-net model
443 and the RC models. While the coefficients are generally consistent across RC models, there are
444 some notable differences with elastic-net. Specifically, certain features deemed important by the
445 elastic-net model, such as the derivative of RT-PCR, and the derivative of Vaccine, are less important
446 for the reservoir computing model. This may indicate that these features predictive ability is better
447 conveyed by their relationship with other features, which is captured by the reservoir computing
448 model but might not be by the elastic-net model. Conversely, emergency, IPTCC, proportion of
449 positive RT-PCR, and hospitalizations are more important for the reservoir computing model than
450 for the elastic-net model.

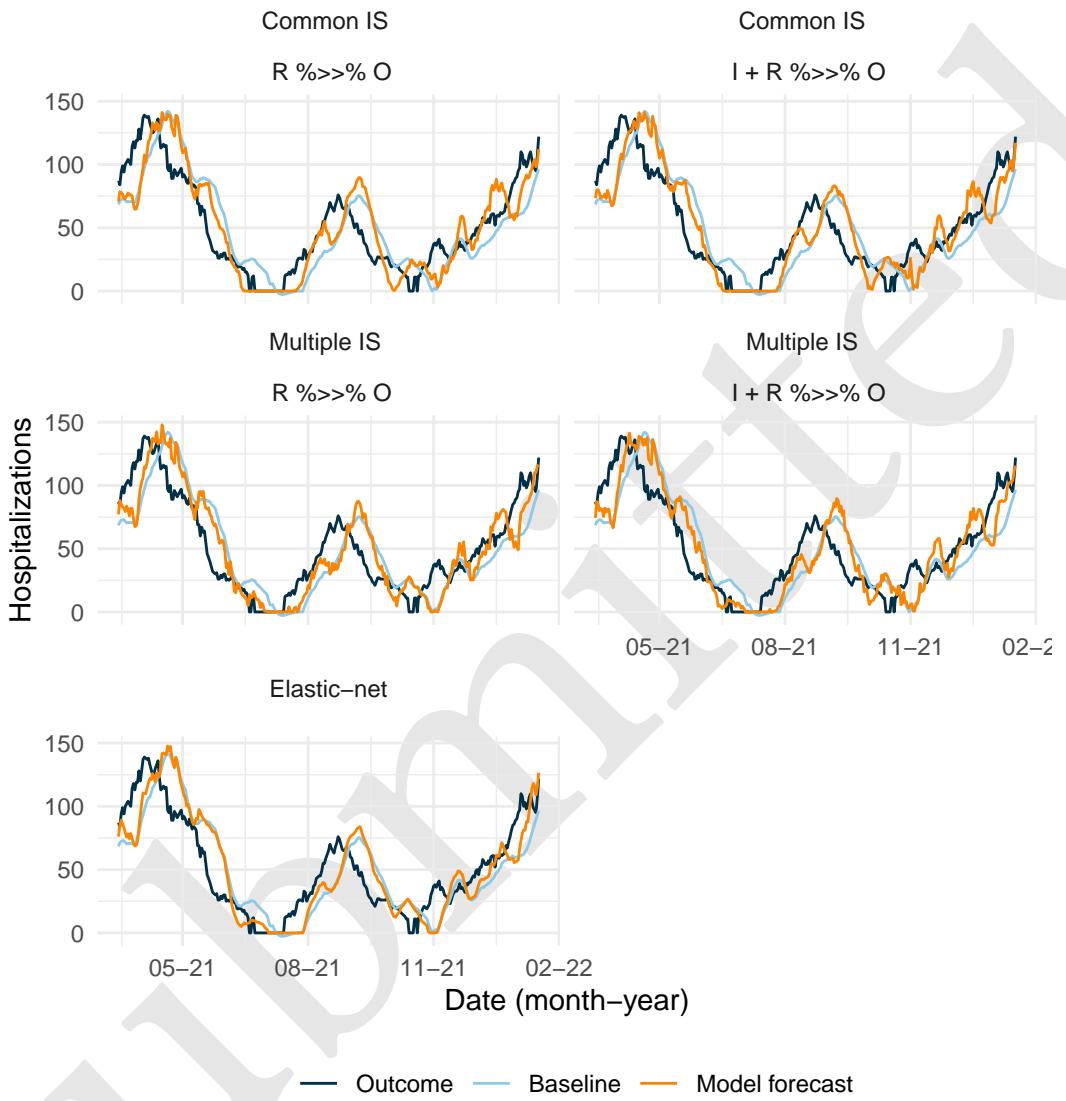


Figure 14: Reservoir computing forecast depending on the setting with and without monthly update. Red line is the median forecast of 40 reservoirs. Grey lines are individual forecast of each of the 40 reservoirs.

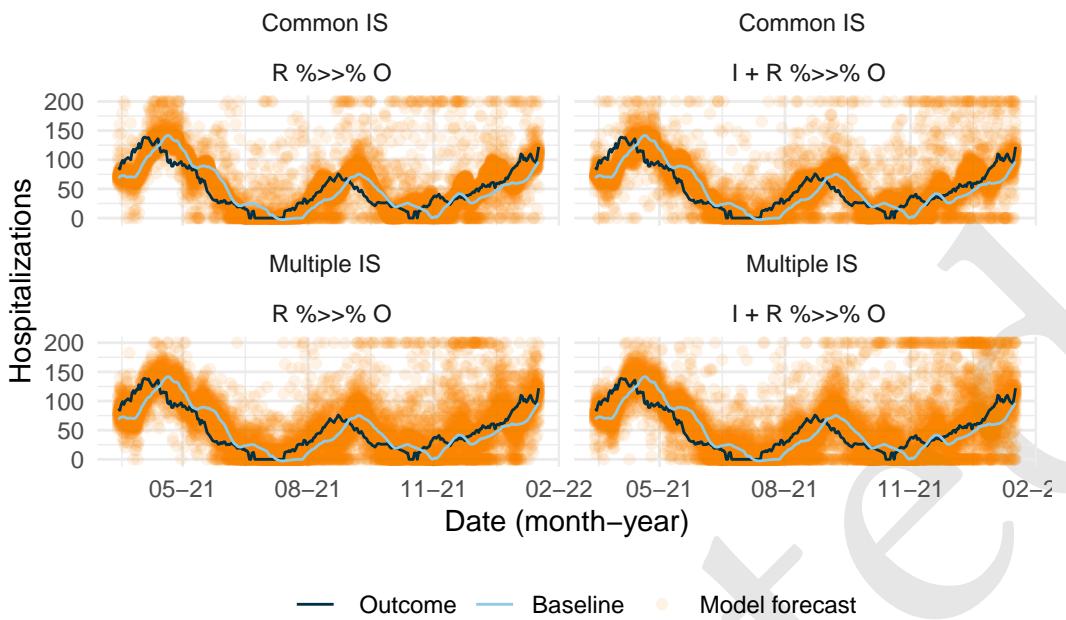


Figure 15: Individual forecast the 40 best hyperparameter sets for the different RC configuration. Forecast value above 200 were set to 200 for clarity.

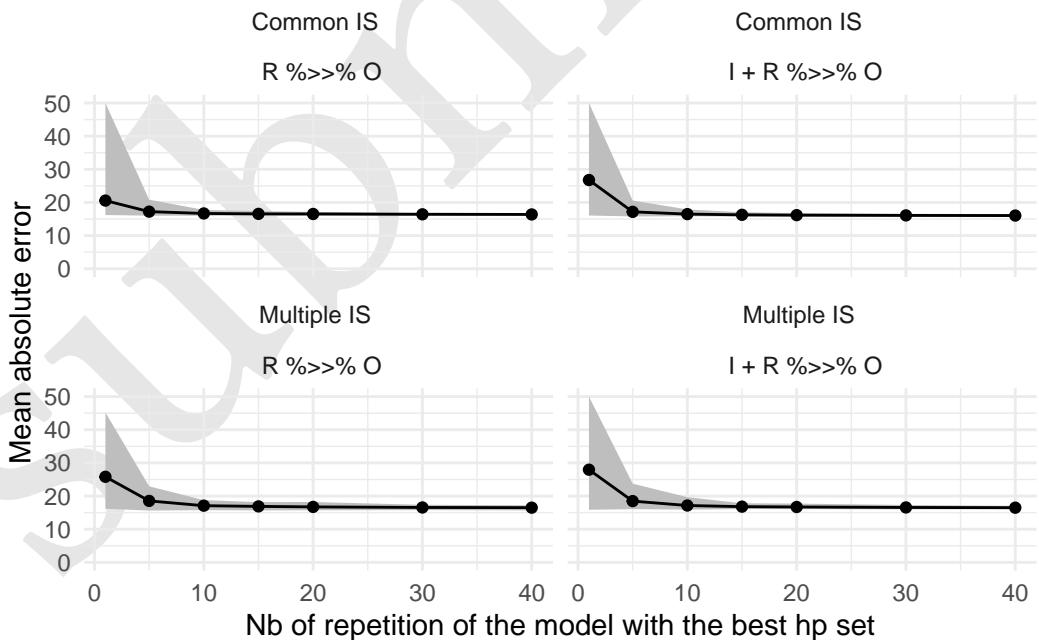


Figure 16: Mean absolute error depending on the number of aggregated reservoir.

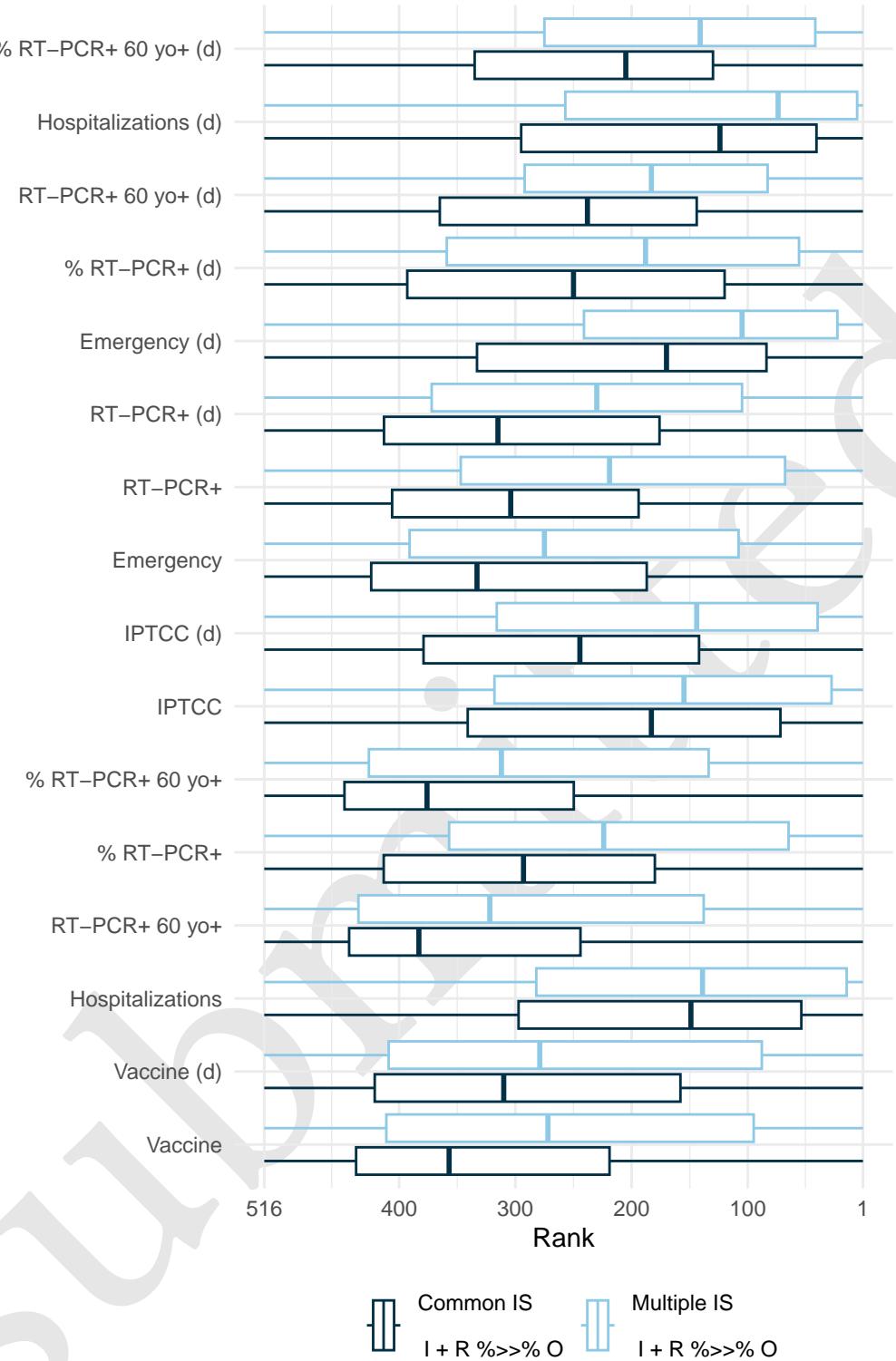


Figure 17: Mean feature importance of the 40 best hyperparameter sets by model, focus on the connection between the input and output layers. Models with direct connection between input and output layer are included. The rank is obtained by comparing the feature input layer and all other connection coefficients (both input and reservoir corresponding coefficients) attributed by the output layer at each date for each hyperparameter set. The higher the output layer's coefficient for the input layer, the closer its rank will be to 1 and the more important the feature is.

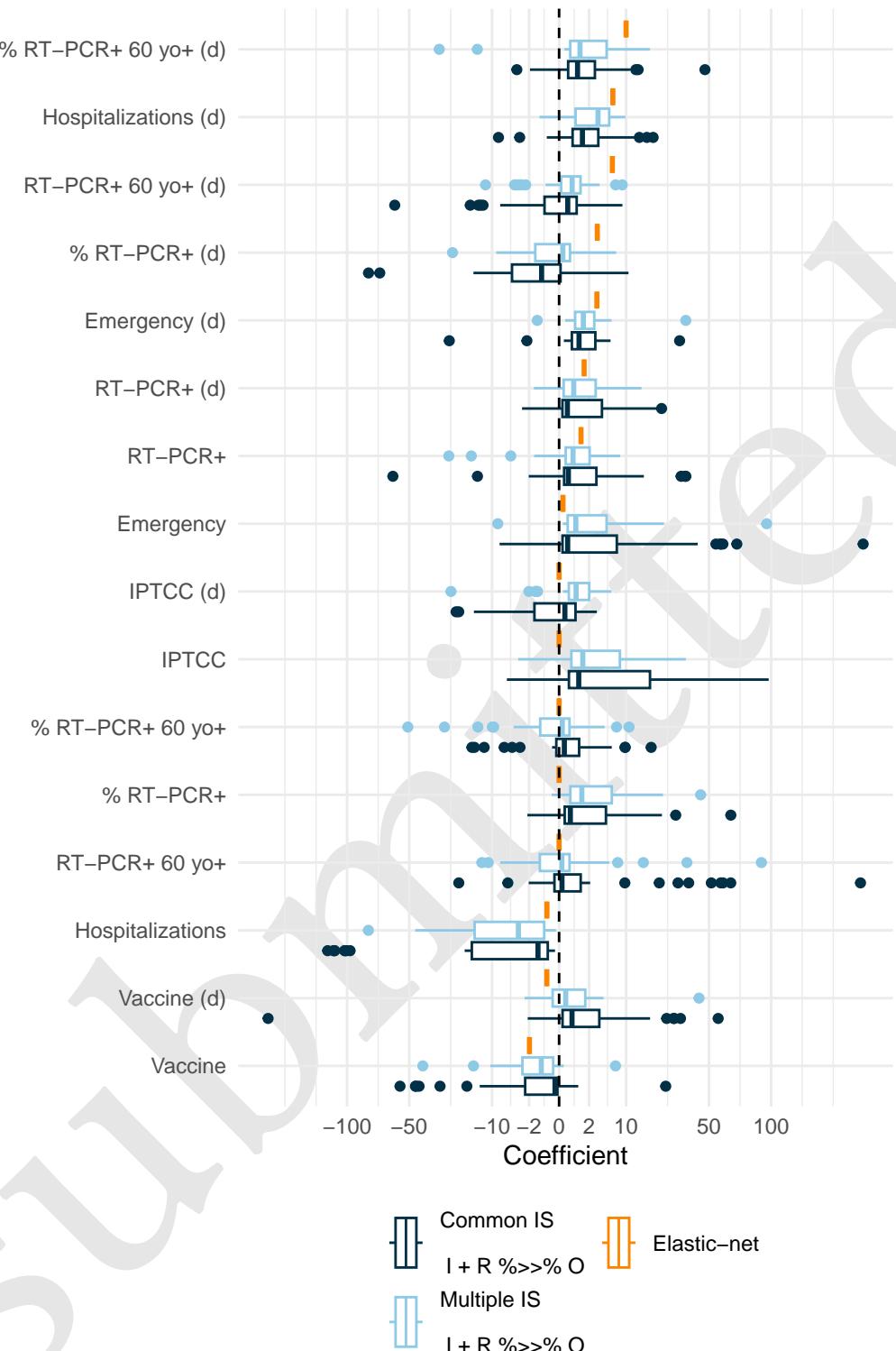


Figure 18: Mean feature coefficient of the 40 best hyperparameter sets by model and the elastic-net model. Only models with direct connection between input and output layer are included. The coefficients were calculated as the average value across all dates for each feature, model and hyperparameter set.

451 **4.4 Discussion**

452 In this specific application, we have demonstrated that RC exhibits commendable performance in
453 comparison to Elastic-net, which serves as the reference model. Furthermore, we highlight the
454 inherent challenges in forecasting within this context, primarily stemming from the non-stationarity
455 of the time series.

456 All computations in this study were conducted using the `reservoirnet` package, and the entire
457 codebase is accessible on Zenodo (Ferté et al. 2024). This R package demonstrates its efficacy in
458 implementing various reservoir architectures, including connection between the input layer and the
459 output layer, as well as the utilization of several input scaling, all within the context of a real-world
460 use case.

461 Given the substantial number of hyperparameters involved, we acknowledge that random search
462 may not be the most efficient optimization algorithm. We have retained this approach for the sake
463 of simplicity in this tutorial paper; however, meta-heuristic approaches, particularly those utilizing
464 evolutionary algorithms, may prove more efficient, especially when employing multiple input scaling
465 (Bala et al. 2018).

466 This study represents a novel contribution to epidemic forecasting utilizing RC. Notably, previous
467 literature predominantly focused on simpler problems characterized by fewer input features or
468 shorter evaluation periods (Liu et al. 2023; Ray, Chakraborty, and Ghosh 2021; Ghosh et al. 2021).
469 Our findings underscore the potential of this approach for future epidemics, suggesting its potential
470 to surpass more traditional epidemiological tools while maintaining a lightweight model structure
471 compared to other RNNs.

472 It is worth noting that all models, including the Ferté et al. (2022) models, encounter challenges
473 in accurately anticipating slope shifts, indicating the need for further investigation. Specifically,
474 additional work is warranted to extend the application of RC to high-dimensional settings, building
475 upon the insights gained from models based on a more extensive set of features.

476 **5 Discussion and conclusion**

477 In this paper, we introduce the R package `reservoirnet`, which serves as a versatile tool for imple-
478 menting reservoir computing based on `ReservoirPy`'s Python library. It offers flexibility in defining
479 the reservoir architecture, including options for specifying connections between the input layer and
480 the output layer, as well as variations in input scaling as demonstrated on a real-world use case.

481 We provided a comprehensive overview of the basic usage of the `reservoirnet` package through
482 illustrative examples in regression and classification tasks. This introductory section serves as
483 a foundation for R users, offering step-by-step guidance on constructing and training reservoir
484 computing models using the package. By demonstrating the application of RC in both regression
485 and classification scenarios, we aim to equip users with the essential knowledge and skills needed to
486 harness the capabilities of reservoir computing for diverse tasks.

487 Drawing on the robust foundation of the `ReservoirPy` structure, a well-maintained Python library,
488 this package inherits its reliability and longevity. We have focused on providing access to the
489 fundamental features, building upon the strong base provided by `ReservoirPy`. Therefore, this initial
490 version of `reservoirnet` must evolve in tandem with the growing understanding and adoption of
491 RC within the R community.

492 References

- 493 Bala, Abubakar, Idris Ismail, Rosdiazli Ibrahim, and Sadiq M. Sait. 2018. “Applications of Meta-
494 heuristics in Reservoir Computing Techniques: A Review.” *IEEE Access* 6: 58012–29. <https://doi.org/10.1109/ACCESS.2018.2873770>.
- 495 Carrat, Fabrice, Julie Figoni, Joseph Henny, Jean-Claude Desenclos, Sofiane Kab, Xavier de Lamballerie,
496 and Marie Zins. 2021. “Evidence of Early Circulation of SARS-CoV-2 in France: Findings from
497 the Population-Based ‘CONSTANCES’ Cohort.” *European Journal of Epidemiology*, February, 1–4.
498 <https://doi.org/10.1007/s10654-020-00716-2>.
- 499 Carslaw, David. 2023. “Worldmet: Import Surface Meteorological Data from NOAA Integrated
500 Surface Database (ISD).” <https://cran.r-project.org/web/packages/worldmet/index.html>.
- 501 Carvalho, Kathleen, João Paulo Vicente, Mihajlo Jakovljevic, and João Paulo Ramos Teixeira. 2021.
502 “Analysis and Forecasting Incidence, Intensive Care Unit Admissions, and Projected Mortality
503 Attributable to COVID-19 in Portugal, the UK, Germany, Italy, and France: Predictions for 4
504 Weeks Ahead.” *Bioengineering* 8 (6): 84. <https://doi.org/10.3390/bioengineering8060084>.
- 505 COVID-19 Cumulative Infection Collaborators. 2022. “Estimating Global, Regional, and National
506 Daily and Cumulative Infections with SARS-CoV-2 Through Nov 14, 2021: A Statistical Analysis.”
507 *Lancet*. [https://doi.org/10.1016/S0140-6736\(22\)00484-6](https://doi.org/10.1016/S0140-6736(22)00484-6).
- 508 Cramer, Estee Y., Evan L. Ray, Velma K. Lopez, Johannes Bracher, Andrea Brennen, and et al.
509 2022. “Evaluation of Individual and Ensemble Probabilistic Forecasts of COVID-19 Mortality
510 in the United States.” *Proceedings of the National Academy of Sciences* 119 (15): e2113561119.
511 <https://doi.org/10.1073/pnas.2113561119>.
- 512 Etalab. 2020. “Les Données Relatives Au COVID-19 En France - Data.gouv.fr.” <https://www.data.gouv.fr/fr/pages/donnees-coronavirus/>.
- 513 Ferté, Thomas, Kalidou Ba, Dan Dutartre, Pierrick Legrand, Vianney Jouhet, Romain Griffier, Rodolphe
514 Thiébaut, Xavier Hinaut, and Boris P. Hejblum. 2024. “Thomasferte/Jss_reservoirnet: First
515 Release.” Zenodo. <https://doi.org/10.5281/ZENODO.11281341>.
- 516 Ferté, Thomas, Vianney Jouhet, Romain Griffier, Boris P. Hejblum, Rodolphe Thiébaut, and Bordeaux
517 University Hospital Covid-19 Crisis Task Force. 2022. “The Benefit of Augmenting Open Data
518 with Clinical Data-Warehouse EHR for Forecasting SARS-CoV-2 Hospitalizations in Bordeaux
519 Area, France.” *JAMIA Open* 5 (4): ooac086. <https://doi.org/10.1093/jamiaopen/ooac086>.
- 520 Ferté, Thomas, Vianney Jouhet, Romain Griffier, Boris Hejblum, Rodolphe Thiébaut, and Bordeaux
521 University Hospital Covid-19 Crisis Task Force. 2023. “The Benefit of Augmenting Open Data
522 with Clinical Data-Warehouse EHR for Forecasting SARS-CoV-2 Hospitalizations in Bordeaux
523 Area, France.” Dryad. <https://doi.org/10.5061/DRYAD.HHMGQNKX>.
- 524 Friedman, Jerome, Trevor Hastie, and Rob Tibshirani. 2010. “Regularization Paths for Generalized
525 Linear Models via Coordinate Descent.” *Journal of Statistical Software* 33 (1): 1–22.
- 526 Ghosh, Subrata, Abhishek Senapati, Arindam Mishra, Joydev Chattopadhyay, Syamal K. Dana,
527 Chittaranjan Hens, and Dibakar Ghosh. 2021. “Reservoir Computing on Epidemic Spreading:
528 A Case Study on COVID-19 Cases.” *Physical Review E* 104 (1): 014308. <https://doi.org/10.1103/PhysRevE.104.014308>.
- 529 Hinaut, Xavier, and Peter Ford Dominey. 2013. “Real-Time Parallel Processing of Grammatical
530 Structure in the Fronto-Striatal System: A Recurrent Network Simulation Study Using Reservoir
531 Computing.” *PLOS ONE* 8 (2): e52946. <https://doi.org/10.1371/journal.pone.0052946>.
- 532 Hübner, Martin, Tobias Zingg, David Martin, Philippe Eckert, and Nicolas Demartines. 2020. “Surgery
533 for Non-Covid-19 Patients During the Pandemic.” *PLoS ONE* 15 (10): e0241331. <https://doi.org/10.1371/journal.pone.0241331>.
- 534 Jaeger, Herbert. 2001. “The “Echo State” Approach to Analysing and Training Recurrent Neu-
535 ral Networks-with an Erratum Note.” Bonn, Germany: German National Research Center for
536 Information Technology GMD Technical Report 148 (January).
- 537
- 538
- 539
- 540

- 541 Kim, Gina, Mengru Wang, Hanh Pan, Giana H. Davidson, Alison C. Roxby, Jen Neukirch, Danna Lei,
542 Elicia Hawken-Dennis, Louise Simpson, and Thuan D. Ong. 2020. “A Health System Response to
543 COVID-19 in Long-Term Care and Post-Acute Care: A Three-Phase Approach.” *Journal of the*
544 *American Geriatrics Society* 68 (6): 1155–61. <https://doi.org/10.1111/jgs.16513>.
- 545 Kmet, Tibor, and Maria Kmetova. 2019. “Bézier Curve Parametrisation and Echo State Network
546 Methods for Solving Optimal Control Problems of SIR Model.” *Biosystems* 186 (December): 104029.
547 <https://doi.org/10.1016/j.biosystems.2019.104029>.
- 548 Kudo, Mineichi, Jun Toyama, and Masaru Shimbo. 1999. “Multidimensional Curve Classification
549 Using Passing-Through Regions.” *Pattern Recognition Letters* 20 (11): 1103–11. [https://doi.org/10.1016/S0167-8655\(99\)00077-X](https://doi.org/10.1016/S0167-8655(99)00077-X).
- 550 Liu, Bocheng, Yiyuan Xie, Weichen Liu, Xiao Jiang, Yichen Ye, Tingting Song, Junxiong Chai, Manying
551 Feng, and Haodong Yuan. 2023. “Nanophotonic Reservoir Computing for COVID-19 Pandemic
552 Forecasting.” *Nonlinear Dynamics* 111 (7): 6895–6914. <https://doi.org/10.1007/s11071-022-08190-z>.
- 553 Lukoševičius, Mantas, and Herbert Jaeger. 2009. “Reservoir Computing Approaches to Recurrent
554 Neural Network Training.” *Computer Science Review* 3 (3): 127–49. <https://doi.org/10.1016/j.cosrev.2009.03.005>.
- 555 Maass, Wolfgang, Thomas Natschläger, and Henry Markram. 2002. “Real-Time Computing Without
556 Stable States: A New Framework for Neural Computation Based on Perturbations.” *Neural
557 Computation* 14 (11): 2531–60. <https://doi.org/10.1162/089976602760407955>.
- 558 Martinuzzi, Francesco, Chris Rackauckas, Anas Abdelrehim, Miguel D. Mahecha, and Karin Mora.
559 2022. “ReservoirComputing.jl: An Efficient and Modular Library for Reservoir Computing
560 Models.” *Journal of Machine Learning Research* 23 (288): 1–8. <http://jmlr.org/papers/v23/22-0611.html>.
- 561 Mohimont, Lucas, Amine Chemchem, François Alin, Michaël Krajecki, and Luiz Angelo Steffenel.
562 2021. “Convolutional Neural Networks and Temporal CNNs for COVID-19 Forecasting in France.”
563 *Applied Intelligence*, April. <https://doi.org/10.1007/s10489-021-02359-6>.
- 564 Nakane, Ryosho, Gouhei Tanaka, and Akira Hirose. 2018. “Reservoir Computing With Spin Waves
565 Excited in a Garnet Film.” *IEEE Access PP* (January): 1–1. <https://doi.org/10.1109/ACCESS.2018.2794584>.
- 566 Paireau, Juliette, Alessio Andronico, Nathanaël Hozé, Maylis Layan, Pascal Crépey, Alix Roumagnac,
567 Marc Lavielle, Pierre-Yves Boëlle, and Simon Cauchemez. 2022. “An Ensemble Model Based
568 on Early Predictors to Forecast COVID-19 Health Care Demand in France.” *Proceedings of the
569 National Academy of Sciences* 119 (18): e2103302119. <https://doi.org/10.1073/pnas.2103302119>.
- 570 Penkovsky, Bogdan, Laurent Larger, and Daniel Brunner. 2018. “Efficient Design of Hardware-
571 Enabled Reservoir Computing in FPGAs.” *Journal of Applied Physics* 124 (16): 162101. <https://doi.org/10.1063/1.5039826>.
- 572 Pottier, Loïc. 2021. “Forecast of the Covid19 Epidemic in France.” *medRxiv*. <https://doi.org/10.1101/2021.04.13.21255418>.
- 573 Prychynenko, Diana, Matthias Sitte, Kai Litzius, Benjamin Krüger, George Bourianoff, Mathias Kläui,
574 Jairo Sinova, and Karin Everschor-Sitte. 2018. “Magnetic Skyrmion as a Nonlinear Resistive
575 Element: A Potential Building Block for Reservoir Computing.” *Physical Review Applied* 9 (1):
576 014034. <https://doi.org/10.1103/PhysRevApplied.9.014034>.
- 577 Rafayelyan, Mushegh, Jonathan Dong, Yongqi Tan, Florent Krzakala, and Sylvain Gigan. 2020. “Large-
578 Scale Optical Reservoir Computing for Spatiotemporal Chaotic Systems Prediction.” *Physical
579 Review X* 10 (4): 041037. <https://doi.org/10.1103/PhysRevX.10.041037>.
- 580 Rahimi, Iman, Fang Chen, and Amir H. Gandomi. 2021. “A Review on COVID-19 Forecasting Models.”
581 *Neural Computing & Applications*, February, 1–11. <https://doi.org/10.1007/s00521-020-05626-8>.
- 582 Ray, Arnob, Tanujit Chakraborty, and Dibakar Ghosh. 2021. “Optimized Ensemble Deep Learning
583 Framework for Scalable Forecasting of Dynamics Containing Extreme Events.” *Chaos (Woodbury,*
584 *N.Y.)* 31 (11): 111105. <https://doi.org/10.1063/5.0074213>.

- 591 Roumagnac, Alix, Eurico de Carvalho Filho, Raphaël Bertrand, Anne-Kim Banchereau, and Guillaume
 592 Lahache. 2021. “Étude de l'influence Potentielle de l'humidité Et de La Température Dans La
 593 Propagation de La Pandémie COVID-19.” *Médecine de Catastrophe - Urgences Collectives*, Douleur
 594 et situations d'exceptionPandémie COVID-19, 5 (1): 87–102. <https://doi.org/10.1016/j.pxur.2021.01.002>.
- 595 Simões, Jorge, João Paulo Moreira Magalhães, André Biscaia, António da Luz Pereira, Gonçalo
 596 Figueiredo Augusto, and Inês Fronteira. 2021. “Organisation of the State, Model of Health System
 597 and COVID-19 Health Outcomes in Six European Countries, During the First Months of the
 598 COVID-19 Epidemic in 2020.” *The International Journal of Health Planning and Management*, June,
 600 10.1002/hpm.3271. <https://doi.org/10.1002/hpm.3271>.
- 601 Smith, Adam, Neal Lott, and Russ Vose. 2011. “The Integrated Surface Database: Recent Developments
 602 and Partnerships.” *Bulletin of the American Meteorological Society* 92 (6): 704–8. <https://doi.org/10.1175/2011BAMS3015.1>.
- 603 Tanaka, Gouhei, Toshiyuki Yamane, Jean Benoit Héroux, Ryosho Nakane, Naoki Kanazawa, Seiji
 604 Takeda, Hidefumi Numata, Daiju Nakano, and Akira Hirose. 2019. “Recent Advances in Physical
 605 Reservoir Computing: A Review.” *Neural Networks* 115 (July): 100–123. <https://doi.org/10.1016/j.neunet.2019.03.005>.
- 606 Trouvain, Nathan, and Xavier Hinaut. 2021. “Canary Song Decoder: Transduction and Implicit
 607 Segmentation with ESNs and LTSMs.” In *Artificial Neural Networks and Machine Learning – ICANN*
 608 2021, edited by Igor Farkaš, Paolo Masulli, Sebastian Otte, and Stefan Wermter, 71–82. Lecture
 609 Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-030-86383-8_6.
- 610 ———. 2022. “ReservoirPy: A Simple and Flexible Reservoir Computing Tool in Python.” <https://inria.hal.science/hal-03699931>.
- 611 Trouvain, Nathan, Luca Pedrelli, Thanh Trung Dinh, and Xavier Hinaut. 2020. “ReservoirPy:
 612 An Efficient and User-Friendly Library to Design Echo State Networks.” In *Artificial Neural
 613 Networks and Machine Learning – ICANN 2020*, 494–505. Springer International Publishing.
 614 <https://inria.hal.science/hal-02595026>.
- 615 Trouvain, Nathan, Nicolas Rougier, and Xavier Hinaut. 2022. “Create Efficient and Complex Reservoir
 616 Computing Architectures with ReservoirPy.” In *From Animals to Animats 16*, edited by Lola
 617 Cañamero, Philippe Gaussier, Myra Wilson, Sofiane Boucenna, and Nicolas Cuperlier, 91–102.
 618 Lecture Notes in Computer Science. Cham: Springer International Publishing. https://doi.org/10.1007/978-3-031-16770-6_8.
- 619 Ushey, Kevin, JJ Allaire, and Yuan Tang. 2024. *Reticulate: Interface to 'Python'*. <https://rstudio.github.io/reticulate/>.
- 620 Vlachas, P. R., J. Pathak, B. R. Hunt, T. P. Sapsis, M. Girvan, E. Ott, and P. Koumoutsakos. 2020.
 621 “Backpropagation Algorithms and Reservoir Computing in Recurrent Neural Networks for the
 622 Forecasting of Complex Spatiotemporal Dynamics.” *Neural Networks* 126 (June): 191–217. <https://doi.org/10.1016/j.neunet.2020.02.016>.
- 623 Wickham, Hadley, Romain François, Lionel Henry, Kirill Müller, and Davis Vaughan. 2023. “dplyr: A
 624 Grammar of Data Manipulation.” <https://CRAN.R-project.org/package=dplyr>.
- 625 Wickham, Hadley, Danielle Navarro, and Thomas Lin Pedersen. 2018. *ggplot2: Elegant Graphics for
 626 Data Analysis (3e)*. 3rd ed. Springer-Verlag New York. <https://ggplot2-book.org/>.
- 627 World Health Organisation. 2020. “WHO Coronavirus (COVID-19) Dashboard.” <https://covid19.who.int>.
- 628 Zhang, Qihuang, Grace Y. Yi, Li-Pang Chen, and Wenqing He. 2023. “Sentiment Analysis and
 629 Causal Learning of COVID-19 Tweets Prior to the Rollout of Vaccines.” *PLoS One* 18 (2): e0277878.
 630 <https://doi.org/10.1371/journal.pone.0277878>.

639 Session information

```
sessionInfo()

640 R version 4.4.1 (2024-06-14)
641 Platform: x86_64-pc-linux-gnu
642 Running under: Ubuntu 24.04.1 LTS
643
644 Matrix products: default
645 BLAS:    /usr/lib/x86_64-linux-gnublas/libblas.so.3.12.0
646 LAPACK:  /usr/lib/x86_64-linux-gnulapack/liblapack.so.3.12.0
647
648 locale:
649 [1] LC_CTYPE=C.UTF-8          LC_NUMERIC=C           LC_TIME=C.UTF-8
650 [4] LC_COLLATE=C.UTF-8       LC_MONETARY=C.UTF-8   LC_MESSAGES=C.UTF-8
651 [7] LC_PAPER=C.UTF-8        LC_NAME=C             LC_ADDRESS=C
652 [10] LC_TELEPHONE=C        LC_MEASUREMENT=C.UTF-8 LC_IDENTIFICATION=C
653
654 time zone: Etc/UTC
655 tzcode source: system (glibc)
656
657 attached base packages:
658 [1] stats      graphics   grDevices datasets  utils      methods   base
659
660 other attached packages:
661 [1] reservoirnet_0.2.0 patchwork_1.3.0   ggplot2_3.5.1     dplyr_1.1.4
662
663 loaded via a namespace (and not attached):
664 [1] gt_0.11.1        utf8_1.2.4       generics_0.1.3    tidyverse_1.3.1
665 [5] renv_1.0.11      xml2_1.3.6       rstatix_0.7.2     lattice_0.20-45
666 [9] stringi_1.8.4    digest_0.6.37    magrittr_2.0.3    evaluate_1.0.1
667 [13] grid_4.4.1      timechange_0.3.0 fastmap_1.2.0    rprojroot_2.0.4
668 [17] Matrix_1.7-1    jsonlite_1.8.9   slider_0.3.2     backports_1.5.0
669 [21] brio_1.1.5      Formula_1.2-5    purrr_1.0.2      fansi_1.0.6
670 [25] scales_1.3.0    abind_1.4-8      cli_3.6.3       rlang_1.1.4
671 [29] munsell_0.5.1   withr_3.0.2      yaml_2.3.10     tools_4.4.1
672 [33] ggsignif_0.6.4  colorspace_2.1-1  ggpunr_0.6.0     here_1.0.1
673 [37] broom_1.0.7     reticulate_1.40.0 png_0.1-8      vctrs_0.6.5
674 [41] R6_2.5.1        lifecycle_1.0.4   lubridate_1.9.3  snakecase_0.11.1
675 [45] stringr_1.5.1    car_3.1-3       janitor_2.2.0    warp_0.2.1
676 [49] pkgconfig_2.0.3  pillar_1.9.0     gtable_0.3.6     Rcpp_1.0.13-1
677 [53] glue_1.8.0       xfun_0.49       tibble_3.2.1     tidyselect_1.2.1
678 [57] knitr_1.49      farver_2.1.2    htmltools_0.5.8.1 labeling_0.4.3
679 [61] rmarkdown_2.29    carData_3.0-5   testthat_3.2.1.1 compiler_4.4.1
```