Supervised forecasting

Ahmed Guecioueur [1,3] and Franz J. Király [† 1,2]

[1] Department of Statistical Science, University College London,
Gower Street, London WC1E 6BT, United Kingdom
[2]The Alan Turing Institute,
The British Library, Kings Cross, London NW1 2DB, United Kingdom
[3]INSEAD, Boulevard de Constance, 77300 Fontainebleau, France

November 19, 2019

TODO: rewrite Abstract & Intro, once we're done with everything.

Abstract

The classical predictive tasks of forecasting and supervised learning have been extensively studied, and benefit from the availability of various machine learning & statistical models. These may be applied to sequential series, with adaptations where necessary. However, in situations when multiple such series are available, neither of those tasks benefit from that additional information. We call this task *supervised forecasting*, and in this dissertation we present a unified theoretical and practical framework for it.

We construct an interface-based framework to theoretically adapt the classical predictive tasks to our new setting. Within this framework, we design various composite prediction strategies that enable us to adopt a wide range of statistical and machine learning models to our task (including Functional Data Analysis and LSTM-based strategies). We also design a kernel-based prediction strategy that is native to the task, without any composition.

We underpin our prediction strategies with a common framework for model comparison, through a statistically-sound workflow that empirically estimates the generalisation error. We use this to perform various experiments on multiple datasets to justify the feasibility of our approach.

Practically, we design & implement the open-source **pysf** package to deal with all stages of supervised forecasting, from data storage and resampling through to composite tunable prediction strategies and an implementation of the evaluation workflow. Written in Python, the package is compatible with popular machine learning libraries such as **scikit-learn** and **keras**.

ahmed.guecioueur@insead.edu
[†]fkiraly@turing.ac.uk

Contents

1. An Introduction to Supervised Forecasting

We consider the situation in which one wishes to forecast the future of a time series, with other independent realisations of the same time series, as well as meta-data features, available for training. For example, predicting the heart rate of a patient, having already observed other patients' heart rates over time; or, predicting operation characteristics of a machine, having already observed other machines over time. This task naturally sits in the intersection of:

(i) *supervised learning*, where independent examples are used to learn a (non-temporal) feature/label relationship, with the task being predicting labels from features, and

(ii) *forecasting*, where the temporo-sequential dependencies within a single time series in isolation are observed and modelled, with the task of predicting the future given the past.

In the situation where the goal is temporal prediction (= forecasting), and multiple time series are available, one may - and in general, should - leverage *both* the temporal association *within* each sample, as well as the past/future relationship observed *across* the independent samples of time series. For this reason, we term this learning task *"supervised forecasting"*.

This manuscript formally introduces and investigates the supervised forecasting task. We make our contributions more precise after an example illustrating the need for this viewpoint.

1.1. Illustrating Examples

As an example, Table 1 shows the heights of 39 boys and 54 girls from the ages of 1 to 18 and the ages at which they were collected[1]. This is a single dataset, consisting of 93 subjects, their genders (a binary boy/girl *series label*) and a time series of height measurements for each of those subjects. The structure of the dataset is, thus, hierarchical (sample, time).

If we wished to forecast one subject's height from the ages of 12 to 18, say, a traditional forecasting model might only consider the height measurements for that one subject up to the age of 12 as inputs - for training as well as forecasting. One would plausibly expect a smarter, "supervised" forecasting procedure to benefit from being trained on the 92 other time series in the dataset, instead of ignoring them. Similarly, one may also expect the same procedure to become more effective by learning how male vs female genders modify the past/future relationship used in the forecast.

Our interest in multiple independent realisations of time series should not be confused with modelling *multivariate* time series; in the latter, multiple time series occur, but these are not statistically independent realisations. However, in applications one frequently finds a combination: multiple independent realisations of multivariate time series. For example, Table 2 shows such a dataset: it consists of average daily temperature and precipitation readings taken at 35 different weather stations in Canada[2]. This dataset may be considered to consist of 35 independent realisations of a bi-variate time series - or, equivalently, 35 independent realisations of pairs of univariate time series, where the task may be to forecast the future of either of the temporal variables, temperature or precipitation.

From an algorithmic perspective it is interesting - and crucial - to note that one may apply both supervised learning and forecasting methods in an attempt to solve the above:

(i) In order to apply a *supervised learning* method, one needs to bring the dataset in question in tabular form. This can be achieved, e.g., by aggregating or tabulating height/temperature/precipitation in yearly/montly bins, such that each time series has an entry for that year/month. The "past" bins, and meta-data variables (e.g., gender) are considered features, the "future" bins are considered labels.

(ii) In order to apply a *forecasting* method, one simply ignores the presence of other time series, and meta-data, and applies the forecasting method to each time series in isolation.

[1] Measurements were taken by Tuddenham and Snyder [43].

[2] Adapted from Ramsay and Silverman [35].

The interesting (and crucial) observation is that *both* of the above are potential solutions to a joint, supervised forecasting task. One may also imagine (or be aware of) methods which solve the task but are *not of the above kind*, i.e., make use other series without tabulating them first. A satisfactory formulation and investigation of supervised forecasting hence needs to encompass all the above possibilities, and also make the relation to supervised learning and forecasting precise - both on the level of tasks and algorithms.

Series index	Series label	Timestamp	Time label
Subject	**Gender**	**Age**	**Height (cm)**
0	boy	1.00	81.3
		1.25	84.2
		1.50	86.4
		...	...
		17.00	193.8
		17.50	194.3
		18.00	195.1
...	...	...	...
92	girl	1.00	76.1
		1.25	78.4
		1.50	82.3
		...	...
		17.00	168.6
		17.50	168.9
		18.00	169.2

Table 1: Subset of the Berkeley "growth" data.

Series index	Timestamp	Time labels	
Weather station	**Day of year**	**Avg. temperature (°C)**	**Avg. precipitation (mm)**
0	1	-3.6	5.2
	2	-3.1	5.8
	3	-3.4	3.9
	...	...	...
	363	-3.2	3.0
	364	-2.8	8.4
	365	-4.2	2.6
...	...	...	...
34	1	-30.7	0.1
	2	-30.6	0.1
	3	-31.4	0.0
	...	...	...
	363	-29.0	0.2
	364	-29.4	0.2
	365	-30.5	0.1

Table 2: Subset of the Canadian weather data.

1.2. Main contributions

Our main scientific contribution is introduction and investigation of the supervised learning task, more precisely: In this manuscript, we attempt to formulate, investigate, and provide solutions for the supervised forecasting task from a formal black-box perspective, i.e., defining the task, strategies that solve it, goodness and evaluation criteria, and

(i) we attempt a precise mathematical formulation of the supervised learning task, including generative setting, nature of a solution, validation and success control - from the black-box perspective.

(ii) we provide meta-algorithms for model visualization, model diagnostics and evaluation, hyper-parameter tuning, model validation, and benchmarking

(iii) we design and formalize an algorithmic sklearn-like interface for supervised forecasting

(iv) we investigate reduction approaches from supervised forecasting to supervised learning and forecasting

(v) we propose a kernel-based algorithm for supervised forecasting which does not arise from reduction, and which is based on a double kernelization (in-between and across samples)

(vi) we implement all the above in a sklearn compatible python toolbox, pysf, which is available on pyPI

(vii) using the pysf toolbox, we conduct a benchmarking study of interface reduction based, and genuine supervised forecasting strategies, on some well-known functional datasets

1.3. Related prior art

The synthetic nature of our contributions touches multiple aspects of existing prior art - from the formal framework specification over meta-algorithms, reduction approaches, concrete supervised forecasting algorithms, and package interface designs. These will be discussed below.

1.3.1. Supervised forecasting from the framework perspective

To the best of our knowledge, a prior formal-mathematical formulation of the supervised forecasting framework does not exist, nor formal descriptions of meta-learning algorithms such as for tuning or evaluation.

The closest formal-statistical setting, from which we adapt a number of important ideas, is that of supervised learning, as for example exposited in the book of Hastie et al. [19]. This includes the set-up of the task as well as formalization of a learning strategy, as well as theory for estimating the *generalisation error* with its common applications in model tuning and model evaluation. Of particular relevance is out-of-sample evaluation in general, and *cross-validation*; Arlot and Celisse [2] provide a survey of the different variants of cross-validation and their properties.

1.3.2. Supervised forecasting methodology - tasks vs algorithms

While a reasonable number of supervised forecasting strategies have already been proposed in multiple branches statistical and machine learning literature, to the best of our knowledge, none of these branches adopt the formal black-box view we propose. More precisely, while most existing branches of literature describe the data type as samples of time series, they usually make additional assumptions that are specific to the model class. They also often do not entirely separate the *"problem"* (the supervised forecasting task) from its potential *"solution"* (the learning algorithm). As a consequence, none of these research areas have a name dedicated to describe the "problem" - which we term supervised forecasting - while all tend to refer to themselves by the name of an algorithm class (e.g., Gaussian processes), or a data model (e.g., functional data analysis), which, furthermore, is usually *unspecific* to the supervised learning task.

1.3.3. Supervised forecasting methodology - statistics and econometrics

The classical statistical approach to modelling discrete, sequentially-correlated observations is to use *stochastic models*. These usually incorporate autoregressive and moving-average components. Box et al. [8] provided a widely-followed procedure for specifying & estimating the ARIMA model, which combines both components is usually used to model a *single* non-stationary time series.

When multiple independent time series are observed, one corresponding to one subject, these are known as *panel data* in econometrics, or *longitudinal data* in biostatistics. Baltagi [3] details a number of

approaches to panel data: at their core, these are linear fixed & random effects models with various extensions to account for serial correlation, heteroscedasticity and other properties like restrictions imposed by economic theory. Diggle et al. [15] show how more traditional statistical techniques, such as generalized linear models and analysis of variance, may be applied to panel data.

The second statistical field in which supervised forecasting is studied is the field of *functional data analysis* (FDA). In FDA, the multiple samples of time series are considered to arise from a two-step generative process: each single time series is assumed to be a temporal samples from continuous functions which are i.i.d. samples. In FDA, these generative continuous functions are called *curves*, and the data is called "functional". Historically, this modelling approach is motivated by Rao [36] recognising that samples of human growth curves were "functional" and could be studied via PCA (principal component analysis) in terms of functional basis expansion coefficients. Ramsay [34] provided a simple conceptual framework to extend this PCA-driven approach of mapping data to functions, to enable the use of other techniques such as canonical correlation analysis. Since then, FDA has grown to encompass a variety of techniques, such as smoothing, dimension reduction, or supervised regression; for example, the functional PCR & PLS models of Reiss and Ogden [39] addresses the latter task, and makes joint use of the former two. Supervised forecasting is a more recent topic: Gooijer and Hyndman [18] relate FDA methods to panel data, and identify the task of multi-step forecasting on panel data (a sub-case of supervised forecasting) as a "fruitful future research area".

[be mention functional regression!]

1.3.4. Supervised forecasting methodology - machine learning

The field of machine learning has produced a vast number of techniques focussed on prediction - most but not all on non-sequential, tabular data, for supervised classification or regression. Hastie et al. [19] detail many such techniques, with some of the most popular method classes being ensemble methods including random forests, kernel methods, and neural networks aka deep learning. While most variants deal with supervised learning or other tasks, we detail below a few known methods for supervised forecasting.

Deep learning techniques may target a variety of tasks depending on the architecture of the underlying *neural network*: *recurrent* neural networks (RNNs) are a variant whose connectivity structure specifically suits sequential data and that can be used for prediction. Williams and Zipser [45] showed how they may be trained. Hochreiter and Schmidhuber [20] re-architected RNNs to deal with the vanishing gradient problem, and their *long short-term memory* (LSTM) architecture has now been widely adopted. LSTMs (and RNNs in general) are trained on samples of sequential data, e.g., in audio or text modelling. The multiple samples naturally correspond to individual series (full or sampled) of a panel dataset.

Kernel-based or Gaussian process based methods include support vector machines and Gaussian process regression (GPR). The kernel trick by Aizerman et al. [1] allows any type of input data including time series as input features [41, 12]. On the other hand, structured output prediction or multi-task learning [29, 7] is concerned prediction of a high-dimensional (and potentially sequentially structured) output. Combining the two, one may theoretically construct kernel-based composite supervised forecasting strategies - though we are not aware of an application case in which this has actually been done.

1.3.5. Reduction-based approaches to supervised forecasting

Reduction is the general term for meta-algorithms converting between machine learning tasks - usually converting a more difficult task to a simpler one, then obtaining a solution to the former from a solution to the latter. For example, Langford et al. [25] have *reduced* the task of quantile regression to that of classification, while Dietterich [14] discusses the use of *sliding windows* as a technique to adapt machine learning algorithms to a sequential setting.

Reduction of supervised forecasting to forecasting is straightforward by forgetting the training examples. While it is intuitive that this is a bad idea in general, we are unaware of an explicit discussion or empirical study; or, of other reduction approaches.

Reduction of supervised forecasting to classical supervised learning is less straightforward and could be achieved in may ways (as we discuss later). The most common way to do so are variants of binning-aggregation of both the input/past and the output/future domain, as done for example in "deep feature synthesis" as used by the **featuretools** package [23] for python, or as done in the fda module of the **mlr** package [5] for R.

1.3.6. Toolboxes and API designs for supervised forecasting

For supervised forecasting, two aspects of toolbox and API design are crucial: data container design to store the samples of time series with meta-data ("panel data"), and modelling pipeline object design to model[3] the algorithmic strategy.

A number of **data containers** for panel data in python exist, including: xarray [21], xpandas [13], and the entity collection in featuretools [23]. We decided to build a custom data container for **pysf**, generalising the state-of-art **pandas** data container [27], due to limitations in all of these (outlined in detail in Section 6), though we may choose to adopt one of the above as a data container in future development.

Regarding **modelling pipeline design**, current state-of-art designs for classical supervised learning include that of: scikit-learn Buitinck et al. [9] which models the prediction use case, and that of mlr [5] which models the full benchmarking workflow. Our design for **pysf** is heavily inspired by these; we consider **pysf** to be an interface-compatible extension of **scikit-learn** for the supervised forecasting task.

In R, there are 2 R packages with a joint design for storage and manipulation of functional data: **mlr** and **fda**. The **mlr** package allows equal length panel data to be stored as part of a data frame and supports multiple panel based tasks, including feature extraction, classical supervised learning, and supervised forecasting - as long as the data is equal length and can be stored in a 2D data frame. The **fda** package, which directly implements key methods from the book of Ramsay and Silverman [35], models data as coefficients to a particular basis expansion. The **fda** package also implements prediction strategies that may be applied directly to the functional data as represented by the basis expansion(s).

While **featuretools**, **xpandas**, **mlr**, and ultimately even **scikit-learn** may be used for method-agnostic supervised forecasting if the data is brought into the right format, it is limited through exactly that, i.e., the necessity to change the data format back and forth - hence carrying out method reduction implicitly and manually (and tediously). To our knowledge, **pysf** is the first attempt of providing an interface design and implementation without such a limitation.

1.4. Outline of the manuscript

We begin our exposition in §2 by defining our problem as the *supervised forecasting* predictive task, relating it to classical predictive tasks, and explaining how we will evaluate the effectiveness of prediction strategies. In §3 we then design a theoretical framework to enable us to build prediction strategies for this new task by using existing prediction models as building blocks. Of course, it is also possible to design native prediction strategies that do not make use of other prediction models, and in **??** we take this approach to design a kernel-based predictor. In §5 we flesh out the approach to model evaluation that we had introduced earlier. In §6 we describe **pysf**, an open-source code implementation of our framework. In §7 we conduct an empirical investigation to validate our approach, before concluding in §8.

Authors' contributions

This manuscript is based on AG's data science MSc thesis, supervised by FK and submitted in August 2018. Revisions were made [describe once done].

[3] Due to an unresolved clash in terminology between the fields of data science and computer science, the word "model" is used in two senses here: "modelling pipeline" refers to a model in the sense of statistical/ML model - as in modelling the data. The second occurrence refers to a model in the sense of class/code/data model: here, the model models the model - or slightly more legibly, the software model mirrors (models) the process of statistically modelling the data - which makes the terminology clash quite bizarre.

FK conceived the original ideas for the supervised forecasting framework, the kernel algorithm in Section **??**, and the high-level package interface. The theoretical framework was worked out as part of AG's thesis work, and jointly in supervision meetings. AG conceived the concrete package interface and implemented the pysf package. AG conducted the benchmarking experiments.

Acknowledgments

We would like to thank Ricardo Silva for pointing out the machine learning literature on reduction. We would like to thank Giovanni Colavizza for discussions on selection of LSTM architectures and hyperparameters.

We would like to thank the Tools, Systems and Practices interest group at the Alan Turing Institute for stimulating conversations, particularly: Raphael Sonabend for his suggestions around the use of task objects as part of evaluation/validation workflows; Frithjof Gressmann for suggestions on the topic of reduction APIs to interface **scikit-learn**; and Markus Löning for pointing out to us the featuretools package.

2. The supervised forecasting task

In this section, we will describe the supervised forecasting task, which we aim to solve as the primary aim of this manuscript. We begin with an intuitive introduction, and then formalise our notions precisely more precisely. Based on the same notation, we then close the section by contrasting our supervised forecasting task with more classical prediction tasks involving time series-like data: supervised learning (on tabular data) and forecasting (on a single time series).

2.1. Introducing the supervised forecasting task

Figure 2.1 is a stylised depiction of the supervised forecasting task.

TODO: intuition in this subsection, then link to the formalisms in the next subsection!

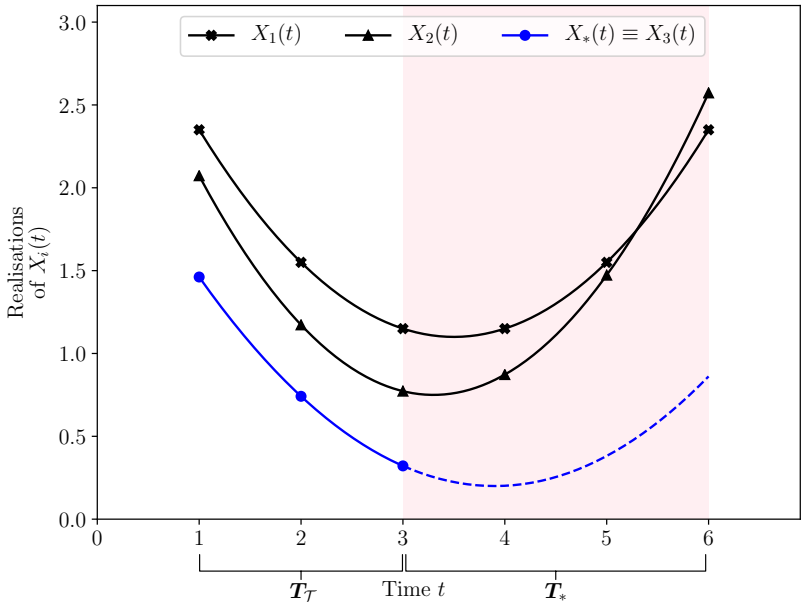

Figure 2.1: Stylized representation of the supervised forecasting setting, in a functional setting. The black dots are the observed training series. Note the black curves may be hypothesized, but are not actually observed. The supervised forecasting task is to forecast, given observation of the blue dots, the behavior along the blue dotted curve. In time-stamped forecasting, the value is queried at certain times, while in functional supervised forecasting, the entire curve is the return object.

2.2. Notational and mathematical conventions

Random variables. To avoid confusion between quantities which are random and non-random, we always explicitly say if a quantity is a random variable. Furthermore, instead of declaring the type of a random variable, say X, by writing it out as a measurable function $X : \Omega \to \mathcal{X}$, we say "$X$ is a random variable taking values in $\mathcal{X}$", or abbreviated "X t.v.in $\mathcal{X}$", suppressing mention of the probability space Ω which we assume to be the same for all random variables appearing. For reading convenience, we will usually denote sets which are the value taking domains of random variables by the same letter in calligraphic script. E.g., X t.v.in $\mathcal{X}$, Y t.v.in $\mathcal{Y}$, and so on.

Sets and tuples. We will use standard set and tuple notation.

Sets of functions. We will denote the set of functions from a set A to a set B by $A \to B$, with bracketing as appropriate. For example, for sets A, B, C we will denote by $A \to [B \to C]$ the set of functions with input in A and output in the set of functions $B \to C$.

2.3. The generative setting: panel data

We first define the type domains in which panel data takes values: sequences and time series.

Definition 2.1. *Let $\mathcal{X}$ be any set (for example, $\mathcal{X} = \mathbb{R}^d$ for some integer d). Let $\mathcal{T} \subseteq \mathbb{R}$.*

(i) We denote by $\text{seq}(\mathcal{X})$ *the set of arbitrary finite length tuples with entries in $\mathcal{X}$, i.e,* $\text{seq}(\mathcal{X}) := \{(x_1, ..., x_m)\ ;\ m \in \mathbb{N}, x_i \in \mathcal{X}\}$. *We call an element of* $\text{seq}(\mathcal{X})$ *a* sequence *(with values) in $\mathcal{X}$. For $\boldsymbol{x} \in \text{seq}(\mathcal{X})$, we denote by $\ell(x)$ the length of $\boldsymbol{x}$, as a tuple.*

(ii.a) We denote by $\Delta(\mathcal{T})$ the ascending sequences in $\mathcal{T}$, i.e., $\boldsymbol{t} \in \Delta(\mathcal{T})$ iff $\boldsymbol{t} \in \text{seq}(\mathcal{T})$ and $\boldsymbol{t}_i \leq \boldsymbol{t}_j$ for all $i \leq j$. We call an element of $\Delta(\mathcal{T})$ a time sequence *(with time points in $\mathcal{T}$), and write $\Delta := \Delta(\mathbb{R})$.*

(ii.b) We denote by $\Delta^\circ(\mathcal{T})$ the strictly ascending sequences in $\mathcal{T}$, i.e., $\boldsymbol{t} \in \Delta^\circ(\mathcal{T})$ iff $\boldsymbol{t} \in \text{seq}(\mathcal{T})$ and $\boldsymbol{t}_i \lneq \boldsymbol{t}_j$ for all $i \leq j$. We will write $\Delta^\circ := \Delta^\circ(\mathbb{R})$, and canonically (and by slight abuse of notation) identify finite sub-sets of $\mathcal{T}$ with Δ°.

(iii.a) We denote $\text{series}(\mathcal{X}, \mathcal{T}) := \{(\boldsymbol{x}, \boldsymbol{t}) \in \text{seq}(\mathcal{X}) \times \Delta(\mathcal{T})\ ;\ \ell(\boldsymbol{t}) = \ell(\boldsymbol{x})\}$ *and call elements of* $\text{series}(\mathcal{X}, \mathcal{T})$ *(discrete)* time series *in $\mathcal{X}$ (at time points in $\mathcal{T}$). We will write abbreviatingly* $\text{series}(\mathcal{X}) := \text{series}(\mathcal{X}, \mathbb{R})$.

(iii.b) In analogy, we write $\text{series}(\mathcal{X}, \mathcal{T}) := \{(\boldsymbol{x}, \boldsymbol{t}) \in \text{seq}(\mathcal{X}) \times \Delta^\circ(\mathcal{T})\ ;\ \ell(\boldsymbol{t}) = \ell(\boldsymbol{x})\}$. *Canonically,* $\text{series}^\circ(\mathcal{X}, \mathcal{T}) \subseteq \text{series}(\mathcal{X}, \mathcal{T})$.

(iii.c) for any $\boldsymbol{s} = (\boldsymbol{x}, \boldsymbol{t}) \in \text{series}(\mathcal{X}, \mathcal{T})$, we write abbreviatingly $\text{vals}(\boldsymbol{s}) := \boldsymbol{x}, \text{time}(\boldsymbol{s}) := \boldsymbol{t}$. *By abuse of notation, we will also consider* $\text{vals}(\boldsymbol{s}), \text{time}(\boldsymbol{s})$ *column vectors.*

Remark 2.2. *Some remarks about our choice of terminology:*

(i) In common usage, elements of $\text{series}(\mathcal{X}, \mathcal{T})$ *and* $\text{seq}(\mathcal{X}, \mathcal{T})$ *are called multivariate (time series or sequences) if elements of $\mathcal{X}$ are tuples of length 2 or longer, otherwise univariate.*

(ii) In common usage, instead of time series, elements of $\text{series}(\mathcal{X}, \mathcal{T})$ *are also sometimes called* indexed series *or* functional data record*, when the index in $\mathcal{T}$ is not interpreted as time - e.g., observations at a certain distance, wavelength, energy, etc.*

(iii) Generally, we will avoid the terminology "series" (in isolation, i.e., without "time"), or "curve", as those terms' usage in literature is non-uniform. For example, series may refer to elements of $\text{series}(\mathcal{X}, \mathcal{T})$*, to elements of* $\text{seq}(\mathcal{X})$*, or to related mathematical structures where the index set is a non-finite domain.*

We would like to stress that our definition of time series is somewhat of a departure from more classical definitions, through the fact that our time series' values are only available at some specific time stamps rather than at an infinity of time points; also, these time stamps are made explicit through the tuple representations. The key reasons for this choice are:

(i) We wish to avoid mixing a hypothesized generative process (of an infinity of data) with what is actually observed in reality (a finite amount of data). Actual time-stamped observations in real world data always take the form of elements of $\text{series}(\mathcal{X})$ where $\mathcal{X}$ is the inhabitant set of some primitive data type (e.g., a real number or a real vector).

(ii) We wish to model knowledge of the time stamps at which the process is observed explicitly. For this, this information needs to be explicitly referrable to, such as through the projection of $\text{series}(\mathcal{X})$ onto Δ_T. In the classical view where the time series is a function, reference to its domain, even when restricted, is notationally cumbersome.

For notational convenience, we introduce further notation to more easily refer to time stamps or values at certain time points:

Notation 2.3. *Let* $\boldsymbol{x} := (\boldsymbol{v}, \boldsymbol{t}) \in \text{series}(\mathcal{X})$ *be a time series.*

(i) For a set $U \subseteq \mathbb{R}$*, we denote by* $\boldsymbol{x} \cap U$ *the time series* $((\boldsymbol{v}_i \,;\, \boldsymbol{v}_i \in U), (\boldsymbol{t}_i \,;\, \boldsymbol{v}_i \in U))$*. That is,* $\boldsymbol{x} \cap U$ *is the time series where the observation times are sub-set to include only those in* U*.*

For the following definitions, further assume that $\boldsymbol{x} \in (\boldsymbol{v}, \boldsymbol{t}) \in \text{series}^{\circ}(\mathcal{X})$*.*

(ii) Let $t := \boldsymbol{t}_i$ *for some* $i \in \mathbb{N}$*. We will denote* $\boldsymbol{x}(t) := \boldsymbol{x}_i$*.*

(iii) More generally, by abuse of notation, we will occasionally identify $\boldsymbol{x}$ *with the function* $\text{time}(\boldsymbol{x}) \to \mathcal{X}, t \mapsto \boldsymbol{x}(t)$*. Note that if* $\boldsymbol{x}$ *is a random variable rather than a constant (such as here),* $\boldsymbol{x}$ *will be a random function with this type.*

We are now ready to formally define the generative setting for panel data.

Definition 2.4. *In the following, we will consider* panel data*, defined as being an i.i.d. sample of the form*

$$(X_1, Z_1), \ldots, (X_N, Z_N) \overset{\text{i.i.d.}}{\sim} (X, Z) \text{ jointly}, \tag{2.1}$$

where X, X_i *t.v.in* $\text{series}(\mathcal{X})$*, and* Z, Z_i *t.v.in* $\mathcal{Z}$*, for some fixed domain sets* $\mathcal{X}, \mathcal{Z}$*.*

In common use cases (but not necessarily assumed here), $\mathcal{X} = \mathbb{R}^k$ and $\mathcal{Z} = \mathbb{R}^n$ for some integers $k, n \in \mathbb{N}$.

Informally, for some sample index i, we intuitively interpret

- the random variable X_i as the i-th time series observed, modelling the i-th (independent) temporal observation process,
- the set-valued random variable $\text{time}(X_i)$ as the time stamps at which the i-th process is observed,
- the random variable Z_i as meta-data for the i-th time series observations. Z_i is not assumed to be observed at a particular time, i.e., not observed at or in conjunction with a time stamp. Z_i is sometimes referred to as a *time-invariant individual effect* in the econometrics literature.

Besides the joint i.i.d. assumption of Eqn. 2.1 we make no further assumptions of independence. As a consequence, the functions X_i are in-principle capable of modelling (observations from) auto-correlated or non-stationary time series.

2.4. Formulating the supervised forecasting task

We now define the supervised forecasting task. Intuitively, we wish to mathematically model the task where an algorithm is asked to make forecasts for the future given the past, after having had the opportunity to train on independent examples, with fully observed past and future. For this, we adopt the usual exposition sequence from supervised learning: (i) specifying the task given the data generative process, (ii) specifying what a "solution" strategy to the task is, and (iii) specifying quantitatively what it means for such a strategy to be "good".
One key distinction which will emerge, to our knowledge so far unrecognized in literature, is whether the algorithmic strategy already knows *for which future time stamps* the prediction is to be made, at the time of training. Practically, the distinction lies in whether the result of fitting the model are the predictions themselves - or, an algorithm which can be queried to produce predictions once given the information of the time points for which predictions are needed. Due to semantic and procedural similarities with supervised learning resp. prediction in functional data analysis, we will call the former *time-stamped supervised forecasting* (or supervised forecasting in the narrow sense), and the latter *functional supervised forecasting*. See Figure 2.1 for a stylized description. Note that algorithms that solve the functional supervised forecasting task can be easily leveraged to solve the time-stamped functional forecasting task, by evaluating at the prediction time stamps.

2.4.1. Setting: supervised forecasting

For both supervised forecasting tasks, we consider the following objects:

- A set of training instances, which is panel data $(X_1, Z_1), \ldots, (X_N, Z_N) \overset{\text{i.i.d.}}{\sim} (X, Z)$ jointly, taking values in series$(\mathcal{X}) \times \mathcal{Z}$, as in Definition 2.4.
- A test instance observation period $\boldsymbol{T} \subseteq \mathbb{R}$ in which observations to forecast from are made. We will assume that $\boldsymbol{T} = (-\infty, \tau]$ for some fixed cut-off time τ.
- A set of forecast time stamps $\boldsymbol{T}_* \subseteq (\tau, \infty)$. We will assume that $\boldsymbol{T}_*$ is a finite set for the time-stamped supervised forecasting task (thus identified with an element of $\text{seq}^\circ(\tau, \infty)$), and an open interval in the functional supervised forecasting task.

Qualitatively, the task will be to train a forecasting algorithm on the training data $(X_1, Z_1), \ldots, (X_N, Z_N)$, such that on further data from the generative process (X, Z), forecasts made for time stamps in $\boldsymbol{T}_*$, based on all observations in $\boldsymbol{T}$, have a low expected loss.

The formal nature of "training the algorithm" and the meaning of "low expected loss" will be detailed in the subsequent paragraphs.

2.4.2. Functionals and learning strategies

In this section, we specify the (type theoretic) types of functions and algorithms involved.

A *fitted* forecasting strategy for supervised forecasting takes the form of a functional

$$\widehat{f} : \underbrace{\text{series}(\mathcal{X})}_{\text{observed portion}} \times \underbrace{\mathcal{Z}}_{\text{features}} \to \underbrace{[\boldsymbol{T}_* \to \mathcal{X}]}_{\text{forecast}}$$

(where the brackets are for illustration and not part of formal type notation).

The output/range of $\widehat{f}$, that is, $\boldsymbol{T}_* \to \mathcal{X}$, has a different real world implementation in the two sub-cases, to which we associate different mathematical representations in alignment with said implementation.
In the *time-stamped case*, $\boldsymbol{T}_*$ is a finite set of time stamps, thus for $\boldsymbol{x} \in \text{series}(\mathcal{X}, \boldsymbol{T}), z \in \mathcal{Z}$ the forecast $\widehat{\boldsymbol{x}} = \widehat{f}(\boldsymbol{x}, z)$ will be a time series $\widehat{\boldsymbol{x}} \in \text{series}^\circ(\mathcal{X}, \boldsymbol{T}_*)$ such that $\text{time}(\widehat{\boldsymbol{x}}) = \boldsymbol{T}_*$. I.e., the output is aligned with

the finitely many forecast time stamps $\boldsymbol{T}_*$, which can easily be stored or represented. Therefore, one may also identify the strategy with a functional

$$\widehat{f}: \underbrace{\text{series}(\mathcal{X})}_{\text{observed portion}} \times \underbrace{\mathcal{Z}}_{\text{features}} \to \underbrace{\text{series}^\circ(\mathcal{X}, \boldsymbol{T}_*)}_{\text{forecast}},$$

where the forecast output is a series in $\text{series}^\circ(\mathcal{X})$, with time stamps in $\boldsymbol{T}_*$.
In the *functional case*, $\boldsymbol{T}_*$ is infinite, therefore there is an (uncountably) infinite progression of values $\widehat{f}(\boldsymbol{x}, z)(t), t \in \boldsymbol{T}_*$, which cannot be stored in a computer. Therefore, the output, an element of $[\boldsymbol{T}_* \to \mathcal{X}]$, must be stored as an algorithm. Hence $\widehat{f}$ is a function which, given a series to forecast from, forecasts an algorithm, which, when given (at the time of forecasting unknown) time stamps, can produce the forecasts.
In a sense, the functional case provides a more complete forecast model of the future, at the cost of such a model in general not being describable by a finite set of values, and in general taking a complex algorithmic form.

2.4.3. The supervised forecasting task

As stated above, in the supervised forecasting task, a learning strategy has access to training data based on which it should estimate a forecasting functional, defined as in the previous Section 2.4.2. The training data is panel data, as described in Section 2.4.1.

Therefore, a deterministic learning strategy takes the form

$$h: \underbrace{\left(\text{series}(\mathcal{X}) \times \mathcal{Z}\right)^N}_{\text{training (panel) data}} \to \underbrace{\text{Type}(\widehat{f})}_{\text{prediction functional}}, \tag{2.2}$$

where, abbreviatingly (and to keep the type readable), we write $\text{Type}(\widehat{f})$ for the formal type of $\widehat{f}$ in Section 2.4.2, that is,

$$\text{Type}(\widehat{f}) = \text{series}(\mathcal{X}) \times \mathcal{Z} \to [\boldsymbol{T}_* \to \mathcal{X}].$$

In this case, the fitted functional is obtained as $f := h((X_1, Z_1), \ldots, (X_N, Z_N))$, which is a $\text{Type}(\widehat{f})$-valued random variable.

Since, more generally, the learning strategy itself may be stochastic, hence introduce randomness which is not covered in the above deterministic learning setting, we will consider the more general (but slightly less intuitive) formalism of a general function-valued random variable f which takes values in $\text{Type}(\widehat{f})$, not necessarily arising from a deterministic learning strategy h as above. We will assume that f may depend on the training data, but that f and the training data are independent from any other random variable encountered, e.g., the test data.

2.5. Performance quantification: generalization error

In order to complete a reasonable specification of the supervised forecasting task, it must be said in which respect a concrete forecasting functional, or a supervised forecasting strategy, is considered "good" or "bad", with respect to a given data generating process (X, Z). This a simple corollary of a definition of a learning task necessarily containing a definition of a goal, thus of what constitutes a solution.

There are two important and distinct ways in which a forecasting algorithm can be good:

(a) the process of *fitting and predicting* possessing a low *expected generalization error* $\varepsilon(t) := \mathbb{E}[L(f(\boldsymbol{x}, Z)(t), X(t))]$, where $\boldsymbol{x} := X(\boldsymbol{T})$, $L: \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a choice of loss function, and t potentially ranges over all time stamps in $\boldsymbol{T}_*$.

(b) a specific *fitted prediction functional* possessing a low *conditional generalization error* $\eta(t) := \mathbb{E}[L(f(\boldsymbol{x}, Z)(t), X(t))|f]$, where $\boldsymbol{x} := X(\boldsymbol{T})$, $L : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a choice of loss function, and t potentially ranges over all time stamps in $\boldsymbol{T}_*$.
Note that ν is a random variable, depending on f, and constant once f is fixed. For a (non-random) realization $\widehat{f}$ of f, i.e., a concrete fitted functional, the realization of $\eta(t)$ is $\mathbb{E}\left[L\left(\widehat{f}(\boldsymbol{x}, Z)(t), X(t)\right)\right]$.

Standard choices for the loss function are the squared loss $L : (x_*, x) \mapsto \|x_* - x\|_2^2$, or the absolute loss $L : (x_*, x) \mapsto \|x_* - x\|_1$.

Remark 2.5. *A few important observations about the above (implicit) definition of η, ν and the associated convention of goodness:*

(i) the quantities ε and η are unknown generative quantities, as they comprise expectations over the generative distribution (X, Z) whose law is unknown. Therefore, they are accessible only via estimates, based on the data. Such estimators will be discussed in the later Section 5.

(ii) ε and η depend on t. In either sub-case of the supervised forecasting task (time-stamped or functional), only a finite number of t will be available in the dataset, therefore if estimates are to be obtained for an infinity of t, these will necessarily be based on assumptions about how the functions ε, η depend on t. This will be briefly discussed in Section 5.

(iii) The cases (a), (b), correspond to different performance guarantees arising. (a) quantifies how well an algorithm will perform when fitted to, and predicting on, similar data. (b) quantifies how well a fitted prediction functional will perform when asked to make predictions on similar data. This difference is also mirrored in the method interface later presented in Section 3.2.3.

2.6. Comparison to closely related learning tasks

We proceed by comparing the above defined supervised forecasting tasks to learning tasks and

2.6.1. Comparison to functional regression

Functional regression with functional response is closely related to functional supervised forecasting. Two common sub-cases of functional regression are functional regression with functional response and scalar covariates, where fitted prediction functionals take the form

$$\widehat{f} : \underbrace{\text{series}(\mathcal{X})}_{\text{covariates}} \to \underbrace{[\boldsymbol{T}_* \to \mathcal{Y}]}_{response},$$

and functional regression with functional response and functional covariates, where fitted prediction functionals take the form

$$\widehat{f} : \underbrace{\mathcal{Z}}_{\text{covariates}} \to \underbrace{[\boldsymbol{T}_* \to \mathcal{Y}]}_{response},$$

in both cases $\boldsymbol{T}_*$ usually being $\mathbb{R}$ or a real interval. The union case of functional responses with functional and scalar covariates is less commonly studied, with prediction functionals

$$\widehat{f} : \underbrace{\text{series}(\mathcal{X})}_{\text{functional covariates}} \times \underbrace{\mathcal{Z}}_{\text{scalar covariates}} \to \underbrace{[\boldsymbol{T}_* \to \mathcal{Y}]}_{response}.$$

In the last, (slightly unusual but) most general case, a functional with type TypeOf$(\widehat{f})$ is estimated from i.i.d. samples $(X_1, Z_1, Y_1), \ldots, (X_N, Z_N, Y_N) \overset{\text{i.i.d.}}{\sim} (X, Z, Y)$, t.v.in series$(\mathcal{X}) \times \mathcal{Z} \times$ series$(\mathcal{Y})$. This is almost the supervised forecasting setting - it were identical if X_i, Y_i were part of the same time series.

Key distinctions to the functional supervised forecasting task are:

(i) in functional regression literature, usually, the task is not clearly separated from the model, which usually takes the form of a (functional) generalized linear model.

(ii) in functional regression literature, it is uncommon to assume that the response series and the covariate series are parts of *the same* time series. It is of course possible to apply functional regression to a supervised forecasting problem, which is an explicit act of reduction, i.e., of supervised forecasting to functional regression. The reverse is not generally or canonically possible, exposing supervised forecasting as the more complex learning problem.

(iii) in functional regression literature, it is usually assumed that the covariates and responses are continuous, i.e., are continuous, non-discrete sub-domains $\mathcal{X}, \mathcal{Y} \subseteq \mathbb{R}^d$ for some d.

Further discussion of specific functional regression methodology may be found in Section[reference section with taxonomy].

2.7. Comparison with the classical forecasting task

As the name implies, the supervised forecasting task is closely related to the (classical) forecasting task. In this, the learner is asked to predict the future values of a series in $\boldsymbol{T}_* \subseteq (\tau, \infty)$, from an observed series $\boldsymbol{x}$ t.v.in series$(\mathcal{X}, \boldsymbol{T})$ with $\boldsymbol{T}_* = (-\infty, \tau]$. See Figure 2.2 for a stylised depiction of the classical forecasting task.

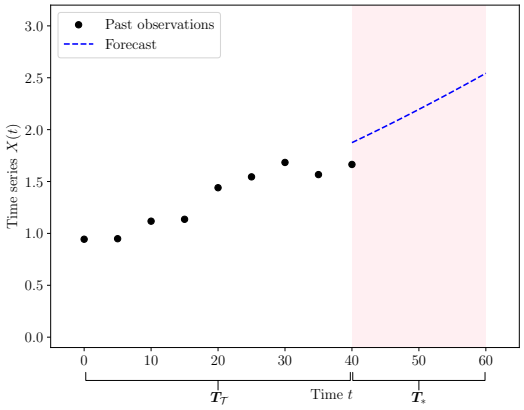

Figure 2.2: Stylised representation of the forecasting setting in **??**.

A key property of forecasting methods, and the setting itself is that a solution, or a sound definition of a solution, is not possible without assuming any kind of regular temporal dependency that replaces the i.i.d. assumption. As such, model fitting over, subject to auto-correlation, such as via common auto-regressive assumptions (such as in the ARIMA class models) are core to the setting.

Formally, forecasting is identical with supervised forecasting without any independent training examples, i.e., the case of $N = 0$ and empty $\mathcal{Z}$ in Section 2.4.1 - the functional as well as the time-stamped sub-cases are commonly considered. As a noteworthy consequence, under this identification, the "fitted" forecasting functional, f, does not depend on any training data. The computational burden of ingesting data, that is, the "fitting", occurs in *evaluation* and *application* of f, rather than its computation from training data.

Especially when comparing this with the case of functional regression in the previous Section 2.6.1 this crucially highlights the existence of (at least) *two different, relevant modes of data ingestion* within the problem: first, the "fitting" of f to the training series; second, the "fitting" of the within-time-series-model on one test series, when substituting the potentially auto-correlated past of the test series into f.

This double ingestion, in particular with the joint view of the two relevant corner cases of functional regression and forecasting, are at the basis our interface based approach discussed in Section 3.

2.8. Comparison to the classical supervised learning task

Sub-cases of time-stamped supervised forecasting are also in close correspondence to the supervised learning task.

In the general supervised learning task, prediction functionals are f taking values in $\underbrace{\mathcal{Z}}_{\text{features}} \to \underbrace{\mathcal{Z}}_{\text{labels}}$ (notation chosen slightly non-standard to avoid a clash later on). These are estimated from N feature-label pairs that are sampled i.i.d. from some data-generating process,

$$(Z_1, Y_1), \ldots, (Z_M, Y_M) \overset{\text{i.i.d.}}{\sim} (Z, Y), \text{ t.v.in } \mathcal{Z} \times \mathcal{Y} \tag{2.3}$$

If $\mathcal{Y}$ can continuously vary as some $\mathcal{Y} \subseteq \mathbb{R}^d$ for some integer d, the task is called supervised regression; if $\mathcal{Y}$ is finite, it is called supervised classification. If $d \geq 2$ the task is called multi-output or multi-target.

The functional f is usually fitted to minimize some expected generalization loss $\mathbb{E}[L(f(Z), Y)]$ in the above setting.

Interestingly, there are multiple distinct ways to identify supervised regression as a sub-case of supervised forecasting. For convenience, we briefly repeat the supervised forecasting setting, without scalar features:

$$X_1, \ldots, X_N \overset{\text{i.i.d.}}{\sim} X, \text{ t.v.in series}(\mathcal{X}, \mathcal{T}), \tag{2.4}$$

and we are seeking a prediction functional $\text{series}(\mathcal{X}, \boldsymbol{T}) \to \text{series}(\mathcal{X}, \boldsymbol{T}_*)$.

In the **first** identification of supervised learning, we assume, restrictively, that all series are observed at exactly the same time points. Thus, $\mathcal{T} = \boldsymbol{T} \uplus \boldsymbol{T}_*$, and it is identical to the time stamps $\text{time}(X_i)$ for any i. From the X_i, we can create data for the above supervised learning scenario by assigning $Z_i := \text{vals}(X_i \cap \boldsymbol{T})$, and $Y_i := \text{vals}(X_i \cap \boldsymbol{T}_*)$, thus $M = N$. On this, we train $\widehat{f} : \mathcal{Z} \to \mathcal{Y}$, and obtain a functional $\text{series}(\mathcal{X}, \boldsymbol{T}) \to \text{series}(\mathcal{X}, \boldsymbol{T}_*)$ by ignoring time stamps (since they are the same for different i).

Pictorially and intuitively, the above strategy corresponds to arranging the values within the series X_i in two tables: those at time stamps before or at τ in the "features" table $(Z_1, \ldots, Z_N)$, those at time stamps after τ in a "targets" table $(Y_1, \ldots, Y_N)$. A supervised learning method is trained on this table, and applied to a new time series by applying it to a row it would correspond to, in that table.

A **second** identification is also possible, with a specific approach to functional (supervised) forecasting. Instead of identifying "before τ" and "after τ" values with features and labels, we can also identify time stamps with features, and values with labels. This strategy is depicted in figure 2.3 is a stylised way.

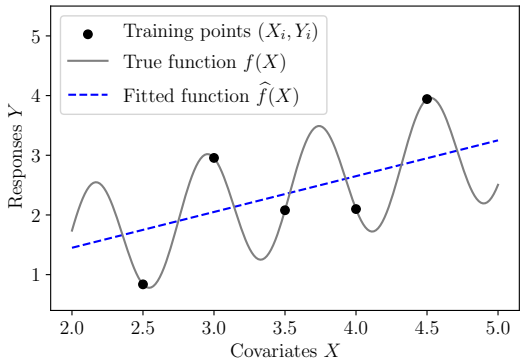

Figure 2.3: Stylised representation of forecasting by supervised learning extrapolation.

Formally, on one instance of test series X_* t.v.in $\text{series}(\mathcal{X}, \boldsymbol{T})$ we set $(Z_1, \ldots, Z_M) := \text{time}(X_*)$, and $(Y_1, \ldots, Y_M) := \text{vals}(X_*)$, and a supervised prediction functional $\widehat{f} : \mathbb{R} \to \mathcal{X}$ is fitted. A prediction for $t \in \boldsymbol{T}_*$ is made as the evaluation $\widehat{f}(t)$. A more sophisticated version of this strategy could be fitted to the pooled

pairs obtained from time(X_i) and vals(X_i).

While the above strategies may or may not be particularly performant (in particular, the first is often much better than the second), they illustrate a key feature of the supervised forecasting task: it can be reduced to a number of related tasks, and it can be reduced in different, not necessarily canonical ways.

Defining this precisely, and showcasing algorithmic relations will be a key part of our interface based approach showcased in Section 3.2.

2.9. Variants not discussed in this manuscript

We would like to point out a few potential variations not discussed here due to scope, but to which extension - in terms of the set-up - is mostly straightforward.

(i) Supervised forecasting, with a sliding observation horizon. This can be treated as a sub-case where, for the same data, different observation horizons are considered.

(ii) Supervised forecasting, with value sets or cut-off times τ that may vary by sample. From the framework perspective, generalization is straightforward, though it incurs a methodological (and notational) overhead.

(iii) Supervised forecasting with multivariate input or output time series where different components may be observed at different times. This introduces complications in methodology and also evaluation.

(iv) Probabilistic supervised forecasting, where the objective of prediction is the conditional distribution of future observation rather than a point prediction value. In this case, the range of the forecasting algorithms needs to be replaced with a corresponding set of distributions, and the point prediction losses need to be replaced by probabilistic losses.

3. Interface-based framework for supervised forecasting

This section explains how we can solve the supervised forecasting task by building prediction strategies out of classical predictors. In Section 2.6, we have defined and discussed predictive tasks at a high level, highlighting the similarities & differences among each. We will now formalise *interfaces* for prediction strategies addressing these tasks, in a simplified language of structured type notation (or class type notation).

The interfaces then enable discussion of reduction and composition strategies, and lend themselves easily to direct implementation in object oriented software design patterns, such as in the pysf package discussed in Section 6 which serves as proof-of-concept and the computational basis for empirical comparisons.

3.1. Interface notation and convention for specifying abstract class interfaces

We introduce some formal notation to allow discussion of data ingestion steps and computation in relation to prediction strategies f and functionals $\widehat{f}$ as discussed in section Section 2.4.2 and later. The notation will follow common notation on definition of (abstract) classes in (typed) object oriented programming. A reader more familiar with mathematical formalism may recognize its equivalence to specification of mutable structured types with function members in a first-order type theory, while a reader more familiar with practical software engineering may opt to entirely ignore that equivalence. In either case, we adopt a number of slightly non-standard conventions:

(i) We will use types and inhabitant sets interchangeably, in line with common mathematical type notation, but in slight departure from formal, or computer scientific type notation. That is, instead of introducing formal types such as `integer` or `positive_real`, we will simply use the mathematical symbols for the inhabitant sets, such as $\mathbb{Z}$ or $\mathbb{R}^+$. This avoids unnecessary parallel notation.

(ii) We adopt the use of field names instead of a type, prefixed by `self`. This indicates that inputs are read from, or results are stored in, the field - similar to common object orientation paradigms such as in python. The type of the field is given with the field. In the showcase example below, the reference to the field `stored_number`, by `self.stored_number` within the methods `store_number` and `compute_square`, instead of reference to the field type $\mathbb{R}$, is an example. This allows us to distinguish inputs and return values by their location in relation to the type - internal (if prefixed by `self`) vs external (if only a type is given).

(iii) In common convention, inhabitants of class types are called objects (of that class), and they usually share implementation in practice. We will not make objects or implementation explicit, leaving it to a concrete package implementation to introduce additional layers of inheritance which we thus avoid to discuss.

For example, consider the following type of a (mutable) class which stores a number and can be queried to return its square:

```
class squarer
  public method store_number      :  ℝ → self.stored_number
  public method compute_square    :  self.stored_number → ℝ⁺
  private variable stored_number  :  ℝ
```

The so-defined structured type, `squarer`, is the type of a mutable class (of the same name) which can be accessed by methods `store_number` and `compute_square`.

In the `squarer` example, the method `store_number` takes, as input a number in $\mathbb{R}$ and outputs a result to the field `stored_number` within the class - nothing is returned to the outside, or the "user", the internal state is mutated. Upon call of the method `method compute_square:`, the field `stored_number` is read out, and from it an external output in $\mathbb{R}^+$, i.e., a positive number is produced.

Note that the class type does specify only type, not semantics of an inhabitant. That is, only input and return types, and where they are stored is specified - what the functions precisely do remains ambiguous from the interface specification in isolation, i.e., typical behaviour, or typical implementation is not specified in the type itself.

In the example, the semantics would include specifying that a typical implementation would store the *same number* that it would receive as the input to `store_number` in `stored_number`, that it would *return that number squared* when `compute_square` is invoked, and that the methods are meant to be *invoked in that sequence*. In this manuscript, explanation of semantics will be presented separately, and all happen in-text, as we think none of the cases are complex enough to justify the use of state-transition diagrams.

3.2. Defining interfaces for prediction strategies

A prediction strategy solves a predictive task, and there can be multiple strategies solving the same predictive task. Thus it is natural to use the "strategy" design pattern, which results in (at most) one class interface per predictive task.

All prediction strategies discussed share a common baseline workflow, hence share a common prediction interface:

1. (user) specification of hyper-parameters that determine behaviour
2. "fitting", i.e., ingestion of data, resulting in a trained model, then
3. making predictions, based on the fitted model

This maps directly onto three distinct interface points:

1. hyper-parameter fields and interface methods, here: setters and getters
2. `fit` methods and a `model` field to store the model
3. `predict` methods to obtain predictions from

The main difference will lie in the type and behaviour of the `fit` and `predict` interface points, mirroring the mathematical particulars of fitting and prediction in the Section §2.

As in common machine learning toolboxes, this would be combined with an inheritance hierarchy for modelling of distinct implementations of the interface, i.e., distinct strategies.

For convenience of the reader, we begin presenting the supervised learning interface which is consolidated through well-known and widely used toolboxes such as [mlr] and [scikit-learn], and then proceed to our suggestions for the more complex learning tasks.

3.2.1. Interface for classical supervised learning

A classical supervised learning predictor is trained on N data points, to predict labels for M test points. Refer to §2.8 for the definitions of $\mathcal{X}, \mathcal{Y}$ and further background.

```
class classical_suplearner:
  method fit:      (𝒳 × 𝒴)^N × parameters      →  model
  method predict:  𝒳^M × model × parameters    →  𝒴^M
  public variable parameters
```

3.2.2. Interface for classical forecasting

A classical forecasting predictor is trained on a single series over some number of time points, to predict future values of the same series a specified number of steps ahead. Refer to **??** for the definition of $\mathcal{X}$ and further background.

```
class classical_forecaster:
  method fit:      𝒳 × parameters          →  model
  method predict:  ℤ₊ × model × parameters  →  𝒳
  public variable parameters
```

3.2.3. Interface for supervised forecasting

A supervised forecasting predictor is trained on N samples of panel data, with common training times ($\boldsymbol{T}_{\mathcal{T}}$ and prediction times $\boldsymbol{T}_{*}$). It then predicts the value of a given test series – whose value at the training times is known – at the common prediction times. Refer to §2.4 for type definitions and further background.

```
class supervised_forecaster:
  method fit:      (𝒵 × Fun(T_𝒯, 𝒳) × Fun(T_*, 𝒳))^N × parameters
                     →  model
  method predict:  𝒵 × Fun(T_𝒯, 𝒳) × model × parameters
                     →  Fun(T_*, 𝒳)
  public variable parameters
```

3.3. A framework for defining predictors as wrappers

It is an important insight that the supervised forecasting task of §2.4 can be implemented by a prediction strategy that is built out of classical predictors. This is the foundation on which we will implement most of the prediction strategies evaluated in this dissertation.

3.3.1. First-order methods for interface transformation

The interface for the *wrapping* – or composition – of prediction strategies can be defined very simply as overloaded first-order methods, each named `wrap` and acting on predictor types. (Alternative, more complex, definitions are possible but unnecessary.)

```
method wrap:  classical_suplearner  →  supervised_forecaster
method wrap:  classical_forecaster  →  supervised_forecaster
```

The simplicity of this approach allows us to define type-invariant `wrap` methods as well, enabling practical machine learning workflow tasks like pipelining and hyperparameter tuning to be brought under the same conceptual umbrella:

```
method wrap:  classical_suplearner   →  classical_suplearner
method wrap:  classical_forecaster   →  classical_forecaster
method wrap:  supervised_forecaster  →  supervised_forecaster
```

3.3.2. Overview of the composite prediction strategy

In general terms, we may formulate a prediction strategy for the supervised forecasting task as a composite prediction strategy, wrapping around some *prediction model*.

The inner prediction model learns a predefined mapping between inputs & outputs (and is backed by its own assumptions and mathematical theory), while the outer wrapper implementation must convert the training & prediction inputs into a format that is suitable for the inner predictor. Both layers may be controlled by hyperparameters, so the composite prediction strategy must distinguish between the inner hyperparameters and the outer ones, while ensuring that they are tuned jointly during the fitting process. Figure 3.1 provides a visual depiction of this framework.

Importantly, the inner prediction model may itself be a prediction strategy that implements one of the 3 tasks we have encountered (classical supervised learning, classical forecasting, or supervised forecasting itself). In this case, implementing the composite prediction strategy is equivalent to implementing the

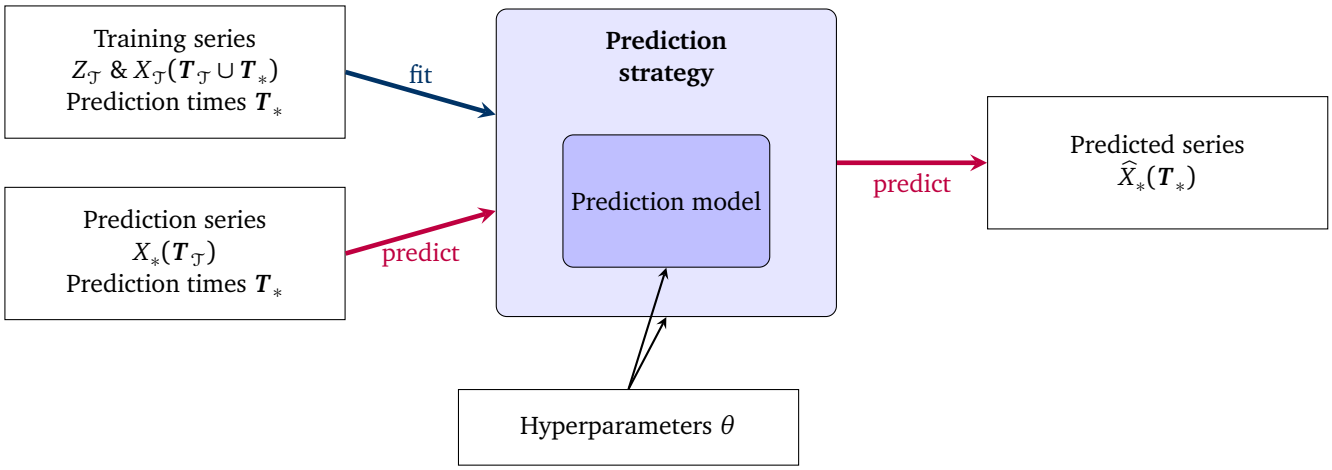

Figure 3.1: Framework for implementing a supervised forecasting prediction strategy as a wrapper around some inner prediction model.

appropriate `wrap` first-order method from §3.3.1. By implementing prediction strategies in conformance with the interfaces of §3.2, we ensure that we are able to use them to build composite prediction strategies. In so doing, we solve the supervised forecasting task using an interface-driven approach.

3.4. Relationship to reduction

Reduction is an approach to solving machine learning problems by decomposing them into smaller subproblems, often necessitating the definition of new loss or regret functions, as in Langford et al. [25]. Our interface-based approach shares the motivation but is applied on the level of the overall prediction strategies rather than the learning algorithms. The use of interfaces limits the extent of coupling between independent prediction strategies. Moreover, our conceptual focus is on the differing tasks, rather than mapping data points and their associated loss or regret functions between two contexts.

3.5. Implementing predictors as wrappers around classical models

We are now in a position to specify how the first-order method `wrap` is implemented for each of the 2 type-transforming cases:

```
method wrap: classical_suplearner → supervised_forecaster
method wrap: classical_forecaster → supervised_forecaster
```

We do so by dealing with each element of a prediction strategy, as defined by the interfaces of §3.2. An implementation of `method wrap` will create & return a new prediction strategy (of the given output type), with an association to an inner prediction strategy (of the given input type), and whose `parameters` and methods `fit` & `predict` are specified explicitly in a way that manipulates the `fit` & `predict` methods of the inner prediction strategy.

See Table 3 for side-by-side definitions of the 4 implementations that we will focus on in this dissertation. (The reader may find it useful to refer to Figures **??** or 5.2 as a visual aid.)

Given the opportunities and constraints of the tasks, other definitions are clearly possible, though these 4 are sufficiently distinct that they serve as useful strategies to evaluate later in this dissertation.

Predictors are either

- *multi-series*, meaning that they take advantage of information available in the set of multiple training series afforded to us by the supervised forecasting setting, or

- *single-series*, meaning they ignore this additional training information, just like predictors in classical settings do.

They also differ in the very nature of the relationships that they learn from the data, as detailed in Table 3:

- The *series* relationship is the least restrictive, since any prediction times may be supplied during the prediction step. Its disadvantage is that it is only suitable for single-series methods.
- The *tabular* relationship is suited specifically to the multi-series setting and may be expected to be more powerful. Its disadvantage is that the prediction timestamps $\boldsymbol{T}_*$ must be available in the training data (i.e. $\boldsymbol{T}_* \subseteq \boldsymbol{T}_i \quad \forall X_i \in X_\mathcal{T}$). It is a special case of a multi-series windowed approach, where the window size covers the entire data width.
- The *windowed* relationship is suitable for either single- or multi-series approaches. It involves passing a pair of contiguous *sliding windows* over the training data, one looking α steps backwards and the other β steps forwards. This typically increases the number of samples available during training. Like many successful classical forecasting methods it is autoregressive, meaning that it predicts covariates from their past selves. Its disadvantages are that it assumes regularly-spaced timestamps, it is only valid for predicting the future (i.e. we require $\text{Min}(\boldsymbol{T}_*) > \text{Min}(\boldsymbol{T}_\mathcal{T})$), and the forecasting error may compound.

Algorithm 1 defines a sliding window forecasting strategy that, when given $X_*(\boldsymbol{T}_\mathcal{T})$ as a covariate, generates a forecast for any timestamp $X_*(t_*)$.

Algorithm 1 Sliding window forecasting strategy
Input: prediction series observations $X_*(\boldsymbol{T}_\mathcal{T})$, a fitted prediction model $\widehat{f}$, prediction times $\boldsymbol{T}_*$, hyperparameters α & β
Output: prediction series forecasts $X_*(\boldsymbol{T}_*)$

define a variable to hold both observations & predictions: $X_*(\boldsymbol{T}_{all}) \equiv X_*(\boldsymbol{T}_\mathcal{T})$
do
    extract the last α (by timestamp) instances of $X_*(\boldsymbol{T}_{all})$
    predict β steps forward in time using the fitted inner prediction model $\widehat{f}$
    append these β predicted values to $X_*(\boldsymbol{T}_{all})$
until all prediction timestamps have been walked over: $\boldsymbol{T}_* \subseteq \boldsymbol{T}_{all}$
extract and return $X_*(\boldsymbol{T}_*)$ from $X_*(\boldsymbol{T}_{all})$

Prediction strategy	First-order `method wrap` being implemented	Additional `parameters`	`fit` to multiple series $X_{\mathcal{T}}$	`predict` for a single series X_*
Single-series	`classical_forecaster` → `supervised_forecaster` or `classical_suplearner` → `supervised_forecaster`	None	Do nothing: the training series $X_{\mathcal{T}}$ are ignored by single-series strategies	1. Fit the inner model to $X_*(\boldsymbol{T}_{\mathcal{T}})$ to learn the relationship $t \to X_*(t) \quad \forall\, t \in \boldsymbol{T}_{\mathcal{T}}$ 2. Predict $X_*(\boldsymbol{T}_*)$ given $\boldsymbol{T}_*$, directly
Single-series Tabular Windowed	`classical_suplearner` → `supervised_forecaster`	• α is the trailing / sliding window size • β is the forward / prediction window size		1. Fit the inner model to $X_*(\boldsymbol{T}_{\mathcal{T}})$ to learn the relationship $X_*(\{t_{j-\alpha-1}, \ldots, t_{j-1}\}) \to X_*(\{t_j, \ldots, t_{j+\beta}\})$, where j is an appropriately-bounded iteration index, incrementing by β, that ranges over $\boldsymbol{T}_{\mathcal{T}}$ 2. Predict $X_*(\boldsymbol{T}_*)$ given $X_*(\boldsymbol{T}_{\mathcal{T}})$, using the sliding window forecasting strategy (Algorithm 1)
Multi-series Tabular		None	Fit the inner model to $X_{\mathcal{T}}(\boldsymbol{T}_{\mathcal{T}})$ to learn the relationship $X_i(\boldsymbol{T}_{\mathcal{T}}) \to X_i(\boldsymbol{T}_*) \quad \forall\, X_i \in X_{\mathcal{T}}$	Predict $X_*(\boldsymbol{T}_*)$ given $X_*(\boldsymbol{T}_{\mathcal{T}})$, directly
Multi-series Tabular Windowed		• α, as above • β, as above • a parameter to control whether to train over – $X_{\mathcal{T}}(\boldsymbol{T}_{\mathcal{T}})$ only, or – $X_{\mathcal{T}}(\boldsymbol{T}_{\mathcal{T}} \cup \boldsymbol{T}_*)$	Fit the inner model to $X_{\mathcal{T}}(\boldsymbol{T}_{\mathcal{T}})$ to learn the relationship $X_i(\{t_{j-\alpha-1}, \ldots, t_{j-1}\}) \to X_i(\{t_j, \ldots, t_{j+\beta}\})$, where • i iterates over all $X_i \in X_{\mathcal{T}}$ • j is an appropriately-bounded iteration index, incrementing by β, that ranges over either $\boldsymbol{T}_{\mathcal{T}}$ or $X_{\mathcal{T}}(\boldsymbol{T}_{\mathcal{T}} \cup \boldsymbol{T}_*)$, according to the hyperparameter setting	Predict $X_*(\boldsymbol{T}_*)$ given $X_*(\boldsymbol{T}_{\mathcal{T}})$, using the sliding window forecasting strategy (Algorithm 1)

Table 3: Side-by-side comparison of the 4 supervised forecasting prediction strategies implemented as type-varying wrappers.

3.6. Baseline predictors for comparison

When we evaluate various composite prediction strategies later on it will prove useful to compare them to *baseline* predictors. These baselines provide naïve, easily-understood predictions that serve as a useful yardstick by which to evaluate the performance of other prediction strategies in the experimental section of this dissertation.

Baseline name	**Description**	**Prediction strategy**	**Inner classical estimator**
Zero predictor	Predict a constant value of 0: $\widehat{X}_*(t_*) = 0 \quad \forall\ t_* \in \mathcal{T}$	Single-series predictor	$f_{classic}(X_{test}) = 0$
Series means predictor	Predict the mean of a series over all training timestamps: $\widehat{X}_*(t_*) = \frac{1}{\#\boldsymbol{T}_\mathcal{T}} \sum_{j=1}^{\#\boldsymbol{T}_\mathcal{T}} X_*(t_j) \quad \forall\ t_* \in \boldsymbol{T}_*$, where t_j is iterated over the training timestamps $\boldsymbol{T}_\mathcal{T}$.	Single-series predictor	$f_{classic}(X_{test})$ $= \text{Mean}(Y_{train})$
Timestamp means predictor	Predict the mean of a timestamp over all training series: $\widehat{X}_*(t_*) = \frac{1}{\#X_\mathcal{T}} \sum_{i=1}^{\#X_\mathcal{T}} X_i(t_*) \quad \forall\ t_* \in \boldsymbol{T}_*$, where i is an iterator over the training series $X_\mathcal{T}$.	Multi-series tabular predictor	$f_{classic}(X_{test})$ $= \text{Mean}(Y_{train})$
Series linear interpolator	For each timestamp $t_* \in \boldsymbol{T}_*$, predict either: • $\widehat{X}_*(t_*) = X_*\big(\text{Min}(\boldsymbol{T}_\mathcal{T})\big) \quad \forall\ t_* < \text{Min}(\boldsymbol{T}_\mathcal{T})$ • $\widehat{X}_*(t_*) = X_*\big(\text{Max}(\boldsymbol{T}_\mathcal{T})\big) \quad \forall\ t_* > \text{Max}(\boldsymbol{T}_\mathcal{T})$ • A linear interpolation between the training neighbours of $t_* \quad \forall\ \text{Min}(\boldsymbol{T}_\mathcal{T}) \leq t_* \leq \text{Max}(\boldsymbol{T}_\mathcal{T})$ depending on t_*'s relationship to the training timestamps $\boldsymbol{T}_\mathcal{T}$.	Single-series predictor	Custom implementation

Table 4: Our 4 baseline prediction strategies.

4. Supervised forecasting in the literature

In this section we will review existing supervised forecasting techniques in the literature and show how they fit within our conceptual framework. Some of these techniques are native to the supervised forecasting task, while most implicitly make use of reduction. Considering these diverse techniques within a single framework emphasises the fact that reduction is a key concept in solving the supervised forecasting task.

We will also make clear what inputs these models are trained on. Recall that we wish to predict a sequence of time labels, and in order to make that prediction one may train a model on some combination of (i) the sequence of time labels, (ii) the timestamps for those time labels, and (iii) the series labels associated with each individual time series.

From the literature:

- Dietterich [14] describes sliding window methods for the task of "sequential supervised learning". This task is very similar to supervised forecasting, with the main difference being that supervised forecasting includes series labels and time indices in the panel data set.
- **?**] solve a particular probabilistic version of the supervised forecasting task, where the test series is considered to be conditionally exchangeable with the training series. They do so by specifying a Bayesian linear model, and their model makes use of series labels ("characteristics" of the subjects) as well as sequences of time labels, in order to predict conditional distributions.
- **?**] solve the supervised forecasting task through a reduction to supervised learning: time series are split into two portions, and a Gaussian Process-based regression (equivalent to kernel ridge regression) model is trained on the past time series (as inputs) and then used to predict the values of future time series. This model makes use of sequences of time labels as well as their associated timestamps.
- **?**] reviews functional regression, which is an offshoot of the field of functional data analysis that Ramsay and Silverman [35] originated. Functional regression is a reduction to supervised learning, where the inputs may include scalars and functions, and the outputs are functions. It is also clearly a case of supervised forecasting, since the future portion of the time series that we wish to forecast can be considered a function. In principle, such models may make use of the time labels only, or the time labels (considered a discretely-observed "function") and series labels (considered "scalars").
- **?**] proposed a diffusion-based model of new product adoption. In practice, using this model for forecasting a new product's sales is frequent in industry practice; it is a native supervised forecasting task to estimate some of the parameters from one or more existing time series of similar products and another parameter from preliminary data for the new product. This model acts on sequences of time labels.
- In classical econometrics, panel data is typically assumed to have an underlying structure that is some variant of the Error Correction Model, and is typically estimated using fixed or random effects. **?**] explains that such models may also be used for forecasting when an econometric panel data model is jointly fit to all series, including the test series for which we aim to make forecasts, and then step-ahead forecasts can be generated for any of the series. When the estimation procedure is some form of regression, this supervised forecasting task is therefore a reduction to regression. Note, however, that classical econometric panel data models cannot be fit to the entire set of time periods that are available in the training set, as they are limited to times that are common to the training and test series. These econometric models act on sequences of time labels.
- **?**] survey the "nowcasting" task, which produces forecasts for one time series from a collection of others, typically at different frequencies. One common general approach is to fit a state space model (perhaps with dimensionality reduction) to the existing series to estimate some common latent variable, and link the latent variable to a model of the series to be forecast. Such an approach could be extended to be a supervised forecasting task. These models act on sequences of time labels.

•

•

TODO: other models that could be easily turned into supervised forecasting

- LSTM models are often used for time series forecasting, but a literature review did not find any instances where an LSTM model was trained on multiple time series, even though there does not seem to be any mechanical impediment to doing so. Training an LSTM on a set of multiple time series and then applying the trained model to produce forecasts for an additional time series would be an instance of supervised forecasting. This model acts on sequences of time labels.
- TODO: in the above, for LSTM, talk about batch sizes? actually I think that doesn't make sense
- TODO: but see the following for some justification of an LSTM trained on many test series: `https://stats.stackexchange.com/questions/412276/how-to-handle-many-times-series-si`
- TODO: factor models
- TODO: panel regression
- TODO: some preprocessing to convert multiple ARIMA values into the same series, based on the following conversation: `https://stats.stackexchange.com/questions/23036/estimating-same-model-over-multiple-time-series` and maybe mention at the same time that VAR-like methods do not apply here because each series is a different instance of the same underlying process, rather than a different variable
- TODO: there may be a link to hierarchical forecasting if we modify that somewhat: maybe a single-level hierarchy?

5. Generalisation error estimation for strategy evaluation

In **??** we defined the generalisation error in 2 ways and stated that we would estimate it in order to evaluate and compare the performances of our prediction strategies. In this section, we will derive valid estimators for the generalisation error and will show that they have good properties (unbiasedness & consistency). We will then extend the estimation procedure from a single training-test split to multiple ones, using the procedure of cross validation.

5.1. Preliminaries

Before proceeding with our derivations, we will explicitly set up the assumptions and notation that we will use. We will also state some preliminary results for later.

We first encountered the loss function $L : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ in §2.4. Based on its arguments, we must distinguish between *fixed* and *random* loss values: if a loss function L is evaluated against known, fixed observations then it must have a fixed, deterministic value; similarly, if one or both of its arguments are random then the result of applying L must be a generative random variable.

Notation 5.1. *Throughout this section, let*

- $L\big(\widehat{X}_i(t), X_i(t)\big)$ *denote the fixed, known loss value calculated from observations* $X_i(t)$ *and known predicted values* $\widehat{X}_i(t)$ *for a series index* i*, and*
- $L\big(\widehat{X}(t), X(t)\big)$ *denote the generative, unobservable loss value theoretically calculated from the generative process* $X(t)$ *and the corresponding output of the prediction strategy* $\widehat{X}(t)$*,*

for some convex loss function L *and a given time index* t*.*

We do not specify a particular form of loss function L in this section since our results are valid in a general setting. For specific implementations, refer to Appendix C.

Assumption 5.2. *Throughout this section, we will assume a fixed set of* N *series samples*

$$\{X_1, X_2, \ldots, X_N\} \tag{5.1}$$

that are each fully observed at a fixed set of K *timestamps*

$$\{t_1, t_2, \ldots, t_K\}. \tag{5.2}$$

We are now able to define a fixed loss matrix $\boldsymbol{M}$ based on our observations, and the underlying random loss matrix $\boldsymbol{L}$.

Notation 5.3. *The* $N \times K$ *data matrix* $\boldsymbol{M}$ *of fixed loss values contains the observed loss value for series* i *and time point* t_j *at row* i *column* j*:*

$$\boldsymbol{M}_{ij} = L\big(\widehat{X}_i(t_j), X_i(t_j)\big) \tag{5.3}$$

$$\Leftrightarrow \boldsymbol{M} = \begin{bmatrix} L\big(\widehat{X}_1(t_1), X_1(t_1)\big) & \ldots & L\big(\widehat{X}_1(t_K), X_1(t_K)\big) \\ \vdots & \ddots & \vdots \\ L\big(\widehat{X}_N(t_1), X_N(t_1)\big) & \ldots & L\big(\widehat{X}_N(t_K), X_N(t_K)\big) \end{bmatrix} \tag{5.4}$$

Notation 5.4. *The* $K \times 1$ *column vector* $\boldsymbol{L}$ *of generative random variables contains the loss random variable associated with time point* t_j *at row* j*:*

$$\boldsymbol{L}_j = L\big(\widehat{X}(t_j), X(t_j)\big) \tag{5.5}$$

$$\Leftrightarrow \boldsymbol{L} = \begin{bmatrix} L\big(\widehat{X}(t_1), X(t_1)\big) \\ \vdots \\ L\big(\widehat{X}(t_K), X(t_K)\big) \end{bmatrix} \tag{5.6}$$

$\boldsymbol{M}$ is also related to $\boldsymbol{L}$ via the sample covariance matrix. We will make use of the following lemma in a later derivation but it is useful to state now.

Lemma 5.5. *The $K \times K$ sample covariance matrix Σ is an unbiased estimator for $\mathrm{Var}[\boldsymbol{L}]$, the covariance between the generative loss random values associated with the K time points. It can be calculated as*

$$\Sigma = \underbrace{\frac{1}{N-1}}_{unbiased} \sum_{i=1}^{N} \big(\boldsymbol{M}_i - \overline{\boldsymbol{M}}\big)\big(\boldsymbol{M}_i - \overline{\boldsymbol{M}}\big)^T \tag{5.7}$$

where

- *$\boldsymbol{M}_i$ is the $1 \times K$ vector containing the ith series of observations of the loss random variables; i.e. the ith row of $\boldsymbol{M}$,*
- *$\overline{\boldsymbol{M}}$ is the $1 \times K$ vector containing the sample mean of the loss observations at each of the K time points,*
- *the pre-factor $\frac{1}{N-1}$ results in an unbiased estimator of the true "population" covariance,*
- *$\boldsymbol{I}_N$ denotes the $N \times N$ Identity matrix, and $\mathbb{1}_N$ denotes the $N \times 1$ vector of ones.*

Proof. See Johnson and Wichern [22] pp. 138-139 for derivations of Σ and pp. 121-123 for a proof that this definition of the sample variance is an unbiased estimator of the population variance. □

Notation 5.6. *Throughout this dissertation, we denote*

- *convergence in distribution by $\xrightarrow{D}$, and*
- *convergence in probability by $\xrightarrow{P}$.*

The following collections of well-known results will be useful in our derivations.

Lemma 5.7. *For a sequence $Y_1, Y_2, \ldots$ and random variable Y,*

1. *if $Y_n \xrightarrow{P} Y$ and h is a continuous function, then $h(Y_n) \xrightarrow{P} h(Y)$, and*
2. *if $Y_n \xrightarrow{P} Y$ then $Y_n \xrightarrow{D} Y$.*

Proof. See Casella and Berger [10] pp. 233, 236 & 262 for sketches of proofs. □

Theorem 1. *Let $Y_1, \ldots, Y_n$ be a random sample from a population Y with mean μ, variance $\sigma^2 < \infty$ and kurtosis γ. Define the sample mean $\overline{Y} = \frac{1}{n}\sum_{i=1}^{n} Y_i$ and the sample variance $s^2 = \frac{1}{n-1}\sum_{i=1}^{n}(Y_i - \overline{Y})^2$. Then*

1. $\mathbb{E}[\overline{Y}] = \mu$
2. $\mathrm{Var}[\overline{Y}] = \sigma^2/n$
3. $\mathbb{E}[s^2] = \sigma^2$
4. *(Central Limit Theorem)*

$$\sqrt{n}(\overline{Y} - \mu) \xrightarrow{D} \mathcal{N}(0, \sigma^2),$$

with the additional assumption of $Y_1, \ldots, Y_n$ being i.i.d.

5. *(Central Limit Theorem for the sample variance)*

$$\sqrt{n}(s^2-\sigma^2) \xrightarrow{D} \mathcal{N}\big(0\,,\ \underbrace{(\gamma+2)\sigma^4}_{\mathrm{Var}[(Y-\mathbb{E}[Y])^2]}\ \big),$$

with the additional assumption of $\mathbb{E}[Y^4] < \infty$. *This also holds if the sample variance* s^2 *is defined as biased (i.e. with prefactor* $1/n$*) and refers to the kurtosis defined as* $\gamma = \sigma^{-4}\ \mathbb{E}\big[\big(Y-\mathbb{E}[Y]\big)^4\big] - 3 = \sigma^{-4}\ \mathrm{Var}\big[\big(Y-\mathbb{E}[Y]\big)^2\big] - 2$.

Proof. For Parts 1-3, see Casella and Berger [10] pp. 212-214. For Part 4, see Billingsley [4] pp. 357-361. For Part 5, see Omey and Van Gulck [33] Theorems 1 & 2. □

5.2. Estimators for a single training-test split

Now that we have the preliminaries in place, Figure 5.1 shows the path that this subsection's exposition will take: we will derive estimators for the timestamp-specific generalisation error $\varepsilon(t)$ and the overall generalisation error ε. We will also derive estimators of the variances of the estimators themselves – these are needed to quantify and distinguish the predictors' performance. Throughout, we will show that the estimators are *unbiased*. At the end, we will use the Central Limit Theorem and other properties to show that the estimators are *consistent*.

$\varepsilon(t)$	$\rightarrow$	ε	Generalisation error
$\downarrow$		$\downarrow$	
$\widehat{\varepsilon}(t)$		$\widehat{\varepsilon}$	Estimators of the generalisation error
$\downarrow$		$\downarrow$	
$v(t)$		v	Variances of the estimators
$\downarrow$		$\downarrow$	
$\widehat{v}(t)$		$\widehat{v}$	Estimators of the variances of the estimators

Figure 5.1: Generalisation error quantities & related estimators. Arrows indicate the direction in which the terms are derived from one another.

5.2.1. Estimators of the generalisation error

Lemma 5.8. *Consider the sample mean of the N observed loss values at timestamp t,*

$$\widehat{\varepsilon}(t) = \frac{1}{N}\sum_{i=1}^{N} L\big(\widehat{X}_i(t), X_i(t)\big). \tag{5.8}$$

This is an unbiased estimator of $\varepsilon(t)$, the expected prediction error at time t.

Proof. By Theorem 1 Part 1, $\mathbb{E}[\widehat{\varepsilon}(t)] = \varepsilon(t)$. □

Lemma 5.9. *Consider the sample mean of the loss observations over all N series and K timestamps,*

$$\widehat{\varepsilon} = \frac{1}{NK}\sum_{i=1}^{N}\sum_{j=1}^{K} L\big(\widehat{X}_i(t_j), X_i(t_j)\big). \tag{5.9}$$

This is an unbiased estimator of ε, the expected prediction error over all K time points.

Proof. By Theorem 1 Part 1, $\mathbb{E}[\widehat{\varepsilon}] = \varepsilon$. □

Remark 5.10. *Our definition of $\widehat{\varepsilon}$ is equivalent to an unweighted mean of the estimators $\{\widehat{\varepsilon}(t_j)\}_{j=1}^{K}$,*

$$\widehat{\varepsilon} = \frac{1}{K}\sum_{j=1}^{K} \widehat{\varepsilon}(t_j), \tag{5.10}$$

while the more general case of a weighted mean,

$$\widehat{\varepsilon}_W = \sum_{j=1}^{K} w_j \widehat{\varepsilon}(t_j) \quad \textit{subject to} \quad \sum_{j=1}^{K} w_j = 1, \tag{5.11}$$

combined with additional assumptions, might allow us to find an estimator with lower variance: $\mathrm{Var}[\widehat{\varepsilon}_W] < \mathrm{Var}[\widehat{\varepsilon}]$. *For example, if we were to (unrealistically) assume the random variables $\{\widehat{\varepsilon}(t_j)\}_{j=1}^{K}$ were independent with known variance then the estimator with minimum variance would be the well-known solution $w_j = \frac{1/\mathrm{Var}[\widehat{\varepsilon}(t_j)]}{\sum_{k=1}^{K} 1/\mathrm{Var}[\widehat{\varepsilon}(t_k)]}$, as per Meier [28].*

5.2.2. Estimators of the variances of the estimators

The estimators that we have just derived ($\widehat{\varepsilon}$ & $\widehat{\varepsilon}(t)$) have variances (v & $v(t)$) that we must now also derive estimators for ($\widehat{v}$ & $\widehat{v}(t)$).

Lemma 5.11. *An unbiased estimator for the variance $v(t) = \text{Var}[\widehat{\varepsilon}(t)]$ is $1/N$ the unbiased sample variance:*

$$\widehat{v}(t) = \frac{1}{N \underbrace{(N-1)}_{\text{unbiased}}} \sum_{i=1}^{N} \Big(L\big(\widehat{X}_i(t), X_i(t)\big) - \widehat{\varepsilon}(t) \Big)^2 \tag{5.12}$$

Proof. We will apply Theorem 1. To show that it is a valid estimator, recall that $\widehat{\varepsilon}(t)$ is a sample mean, with variance

$$v(t) = \text{Var}[\widehat{\varepsilon}(t)] \qquad \text{by definition} \tag{5.13}$$

$$= \frac{\text{Var}\big[L\big(\widehat{X}(t), X(t)\big)\big]}{N} \qquad \text{by Theorem 1 Part 2} \tag{5.14}$$

Plugging in an estimator for $\text{Var}\big[L\big(\widehat{X}(t), X(t)\big)\big]$ into the above – in this case the unbiased sample variance of the observed loss values $\frac{1}{(N-1)} \sum_{i=1}^{N} \big(L\big(\widehat{X}_i(t), X_i(t)\big) - \widehat{\varepsilon}(t) \big)^2$ – results in that estimator for $v(t)$. To show that this estimator is unbiased,

$$\mathbb{E}[\underbrace{N\widehat{v}(t)}_{s^2}] = \underbrace{Nv(t)}_{\sigma^2} \qquad \text{by Theorem 1 Part 3} \tag{5.15}$$

$$\Leftrightarrow \quad \mathbb{E}[\widehat{v}(t)] = v(t) \qquad \text{by linearity of Expectation} \tag{5.16}$$

□

Lemma 5.12. *An unbiased estimator for $v = \text{Var}[\widehat{\varepsilon}]$ can be written in terms of the unbiased sample covariance matrix Σ as*

$$\widehat{v} = \frac{1}{NK^2} \sum_{i=1}^{K} \sum_{j=1}^{K} \Sigma_{ij} \tag{5.17}$$

Proof.

$$v = \text{Var}[\widehat{\varepsilon}] \tag{5.18}$$

$$= \text{Var}\left[\frac{1}{NK} \, \mathbb{1}_{1\times N} \, \boldsymbol{M} \, \mathbb{1}_{K\times 1} \right] \qquad \text{rewriting Lemma 5.9 using Notation 5.3} \tag{5.19}$$

$$= \frac{1}{K^2} \, \text{Var}\left[\frac{1}{N} \, \mathbb{1}_{1\times N} \, \boldsymbol{M} \, \mathbb{1}_{K\times 1} \right] \qquad \text{since } \text{Var}[a\boldsymbol{x}] = a^2 \text{Var}[\boldsymbol{x}] \tag{5.20}$$

$$= \frac{1}{K^2} \, \text{Var}\big[\, \overline{\boldsymbol{M}} \, \mathbb{1}_{K\times 1} \, \big] \qquad \text{rewriting sample means using Lemma 5.5} \tag{5.21}$$

$$= \frac{1}{K^2} \, \mathbb{1}_{1\times K} \, \text{Var}\big[\overline{\boldsymbol{M}}\big] \, \mathbb{1}_{K\times 1} \qquad \text{since } \text{Var}[\boldsymbol{A}\boldsymbol{x}] = \boldsymbol{A}\text{Var}[\boldsymbol{x}]\boldsymbol{A}^T \tag{5.22}$$

$$= \frac{1}{NK^2} \, \mathbb{1}_{1\times K} \, \text{Var}\big[\boldsymbol{M}\big] \, \mathbb{1}_{K\times 1} \qquad \text{by Theorem 1 Part 2} \tag{5.23}$$

According to Lemma 5.5, Σ is an unbiased estimator for $\text{Var}\big[\boldsymbol{M}\big]$. Therefore we plug it in to derive $\widehat{v}$ as an unbiased estimator for v:

$$\widehat{v} = \frac{1}{NK^2} \, \mathbb{1}_{1\times K} \, \Sigma \, \mathbb{1}_{K\times 1} \qquad \text{by Lemma 5.5} \tag{5.24}$$

$$= \frac{1}{NK^2} \sum_{i=1}^{K} \sum_{j=1}^{K} \Sigma_{ij} \qquad \text{in element-wise form} \tag{5.25}$$

□

Remark 5.13. *The above estimator of the variance of the estimator of the generalisation error over all time points is valid in the general case where observations at different time points are correlated with one other. If we were to assume independence between different time points, the resulting estimator*

$$\widehat{v}_{independent} = \frac{1}{NK\underbrace{(NK-1)}_{unbiased}} \sum_{i=1}^{N}\sum_{j=1}^{K}\Big(L_i(t_j)-\widehat{\varepsilon}\Big)^2 \tag{5.26}$$

would underestimate the true variance v in the presence of non-negatively correlated error terms. This would lead to Type I errors when comparing the uncertainties of different prediction methods, so we cannot make this assumption.

5.2.3. Central Limit Theorems

Proposition 5.14. *For a large number of samples N,*

$$\frac{\widehat{\varepsilon}(t)-\varepsilon(t)}{\sqrt{v(t)}} \xrightarrow{D} \mathcal{N}(0,1) \tag{5.27}$$

$$\frac{\widehat{\varepsilon}-\varepsilon}{\sqrt{v}} \xrightarrow{D} \mathcal{N}(0,1) \tag{5.28}$$

Proof. Our estimators $\widehat{\varepsilon}(t)$ and $\widehat{\varepsilon}$ are sample means, so we apply the CLT (Theorem 1 Part 4) to them directly:

$$\frac{\sqrt{N}\big(\widehat{\varepsilon}(t)-\varepsilon(t)\big)}{\sqrt{N\ v(t)}} \xrightarrow{D} \mathcal{N}(0,1) \tag{5.29}$$

$$\frac{\sqrt{N}\big(\widehat{\varepsilon}-\varepsilon\big)}{\sqrt{N\ v}} \xrightarrow{D} \mathcal{N}(0,1) \tag{5.30}$$

and then simplify the left-hand sides. □

Proposition 5.15. *For a large number of samples N,*

$$\sqrt{N}\Big(\widehat{v}(t)-v(t)\Big) \xrightarrow{D} \mathcal{N}\Big(0\ ,\ (\gamma+2)[v(t)]^2\Big), \tag{5.31}$$

where γ is the kurtosis of $L(t)$.

Proof. Our estimator $\widehat{v}(t)$ is $1/N$ the sample variance, so we apply the sample variance CLT (Theorem 1 Part 5) to $N\widehat{v}(t)$ as the estimator for $Nv(t) = \mathrm{Var}[L(t)]$:

$$\sqrt{N}\Big(N\widehat{v}(t)-Nv(t)\Big) \xrightarrow{D} \mathcal{N}\Big(0\ ,\ (\gamma+2)[Nv(t)]^2\Big) \qquad \text{by Theorem 1 Part 5} \tag{5.32}$$

$$\iff \sqrt{N}\Big(N\widehat{v}(t)-Nv(t)\Big) \xrightarrow{D} N\ \mathcal{N}\Big(0\ ,\ (\gamma+2)[v(t)]^2\Big) \qquad \text{property of a Gaussian} \tag{5.33}$$

$$\iff \sqrt{N}\Big(\widehat{v}(t)-v(t)\Big) \xrightarrow{D} \mathcal{N}\Big(0\ ,\ (\gamma+2)[v(t)]^2\Big) \qquad \text{by Lemma 5.7 Part 1} \tag{5.34}$$

□

5.2.4. Consistency

We will now show that the 4 unbiased estimators that we have derived are also consistent[4].

Proposition 5.16.

$$\widehat{\varepsilon}(t) \xrightarrow{P} \varepsilon(t) \tag{5.35}$$

$$\widehat{\varepsilon} \xrightarrow{P} \varepsilon \tag{5.36}$$

$$\widehat{v}(t) \xrightarrow{P} v(t) \tag{5.37}$$

Proof. We have used the CLT to show that the estimators $\widehat{\varepsilon}(t)$, $\widehat{\varepsilon}$ and $\widehat{v}(t)$ are asymptotically Gaussian: see Eqns. 5.27, 5.28 & 5.31, respectively. Therefore, this implies consistency of the estimators as means of those distributions. Refer to Casella and Berger [10] pp. 472-473 for a more detailed reasoning: in short, as $N \to \infty$ the estimators will converge in distribution to the constant means of the asymptotic distributions, and thence converge in probability. This special case is essentially a partial converse of Lemma 5.7 Part 2. □

Lemma 5.17.

$$\Sigma \xrightarrow{P} \mathrm{Var}[\boldsymbol{L}] \tag{5.38}$$

Proof. Johnson and Wichern [22] pp. 175 shows that any multidimensional empirical covariance matrix (including our unbiased definition Σ) will converge in probability to the true population variance. □

Proposition 5.18.

$$\widehat{v} \xrightarrow{P} v \tag{5.39}$$

Proof. Although $\widehat{v}$ is not directly amenable to a CLT, we can show it is consistent by leaning on the consistency of the empirical covariance matrix Σ and the preservation of consistency under a continuous transformation:

$$\Sigma \xrightarrow{P} \mathrm{Var}[\boldsymbol{L}] \qquad \text{by Lemma 5.17} \tag{5.40}$$

$$\Rightarrow \quad \frac{1}{N}\frac{1}{K^2}\,\mathbb{1}_{1\times K}\,\Sigma\,\mathbb{1}_{K\times 1} \xrightarrow{P} \frac{1}{N}\frac{1}{K^2}\,\mathbb{1}_{1\times K}\,\mathrm{Var}\left[\boldsymbol{L}\right]\,\mathbb{1}_{K\times 1} \qquad \text{by Lemma 5.7 Part 1} \tag{5.41}$$

$$\Leftrightarrow \quad \widehat{v} \xrightarrow{P} v \qquad \text{by definition} \tag{5.42}$$

□

5.3. Cross-validated estimation across multiple splits

Cross validation (CV) estimates the generalisation error by splitting our dataset into training & test sets (each combination of which is known as a *fold*), repeating on varied combinations, and then taking the average of the metric over those folds to reduce the variance of the estimate. CV is a popular empirical technique. It makes efficient use of the full training data and directly estimates the generalisation error.

5.3.1. Approach

In our setting, each sample is a time series of data, so the series i.i.d. assumption of Definition 2.4 is consistent with the CV assumption of i.i.d. samples. Figure 5.2 shows a stylised example of one such split into a training set $X_{\mathcal{T}}$ and validation set $X_{\mathcal{V}}$; this split is orthogonal to the training and prediction times $\boldsymbol{T}_{\mathcal{T}}$ & $\boldsymbol{T}_{\mathcal{V}}$, which remain constant throughout.

[4] A consistent estimator converges in probability as the number of data points increases without bound.

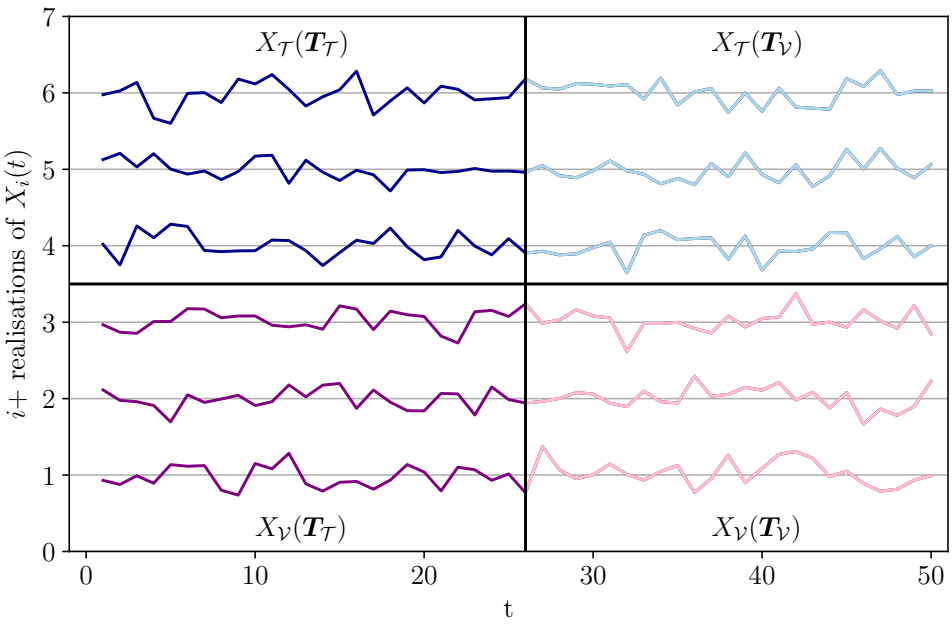

Figure 5.2: Stylised example of splitting multiple series over multiple timestamps into training and test sets. $X_{\mathcal{T}} = \{X_1, X_2, X_3\}$, $X_{\mathcal{V}} = \{X_4, X_5, X_6\}$, $\boldsymbol{T}_{\mathcal{T}} = \{1, \ldots, 25\}$, $\boldsymbol{T}_{\mathcal{V}} = \{26, \ldots, 50\}$.

We will use *k-fold CV* because of its good balance between bias, variance and computational time. Chapter 7 of Hastie et al. [19] suggests using 5 or 10 folds as a rule of thumb to balance bias against variance. Refer to Arlot and Celisse [2] for an extensive survey of different CV methods and their properties.

Chapter 12 of Efron and Hastie [16] raises a subtle objection to our approach – that our decision to use CV is not in itself cross-validated – but otherwise supports the use of CV to choose among competing prediction strategies. Cawley and Talbot [11] also support our approach – calling it "more pragmatic than some evaluation methodologies" and "based on much weaker assumptions" – while highlighting a potential issue that we will address in §5.3.3.

5.3.2. Estimators in the CV case

All the estimators that we have so far derived have been for the case of a single training-test split. We now extend them to K cross-validated splits.

Definition 5.19. *K-fold cross-validation estimators of our single-fold estimators* $\widehat{\varepsilon}, \widehat{\varepsilon}(t), \widehat{v}, \widehat{v}(t)$ *are*

$$\widehat{\varepsilon}_{CV} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\varepsilon}[\mathcal{D}_k], \tag{5.43}$$

$$\widehat{\varepsilon}(t)_{CV} = \frac{1}{K}\sum_{k=1}^{K} \widehat{\varepsilon}(t)[\mathcal{D}_k], \tag{5.44}$$

$$\widehat{v}_{CV} = \frac{1}{K}\sum_{k=1}^{K} \widehat{v}[\mathcal{D}_k], \tag{5.45}$$

$$\widehat{v}(t)_{CV} = \frac{1}{K}\sum_{k=1}^{K} \widehat{v}(t)[\mathcal{D}_k], \tag{5.46}$$

where the notation $[\mathcal{D}]$ *denotes an estimator that has been trained on a single training set* $\mathcal{D}$*, and there are* K *such folds of training-test splits.*

The definition above actually applies to any type of resampling scheme where the folds are identically sampled. We will now discuss a major implication for our setting.

Proposition 5.20. *Taking the mean is a conservative way of aggregating variance estimates. In our setting,*

$$\mathrm{Var}\left[\widehat{\varepsilon}_{CV}\right] \leq \widehat{v}_{CV}, \tag{5.47}$$

$$\mathrm{Var}\left[\widehat{\varepsilon}(t)_{CV}\right] \leq \widehat{v}(t)_{CV}. \tag{5.48}$$

Proof. This follows from Nadeau and Bengio [31] Lemma 1 (the *Variance Reduction Lemma*): for any number of (correlated) random variables $Z_1, \ldots, Z_M$ with $\mathrm{Var}[Z_i] = \sigma^2$, $\mathrm{Corr}(Z_i, Z_j) = \rho$ for $i \neq j$, we have

$$\mathrm{Var}\left[\frac{1}{M}\sum_{i=1}^{M} Z_i\right] = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2. \tag{5.49}$$

For the estimator $\widehat{\varepsilon}$, and assuming the fold errors $\widehat{\varepsilon}[\mathcal{D}]$ are identical and exchangeable for the sake of argument,

$$\underbrace{\mathrm{Var}\left[\widehat{\varepsilon}_{CV}\right]}_{\text{Var of CV estimate}} = \underbrace{\left(\rho + \frac{1-\rho}{K}\right)}_{\leq 1} \underbrace{\mathrm{Var}\left[\widehat{\varepsilon}[\mathcal{D}]\right]}_{\text{Var of one-split estimate}} \tag{5.50}$$

$$\iff \quad \mathrm{Var}\left[\widehat{\varepsilon}_{CV}\right] \leq \mathrm{Var}\left[\widehat{\varepsilon}[\mathcal{D}]\right]. \tag{5.51}$$

Importantly, $\widehat{v}_{CV}$ is an estimator of $\mathrm{Var}\left[\widehat{\varepsilon}[\mathcal{D}]\right]$ and not of $\mathrm{Var}\left[\widehat{\varepsilon}_{CV}\right]$. Since we assumed the fold errors $\widehat{\varepsilon}[\mathcal{D}]$ are identical and exchangeable, $\widehat{v}_{CV}$ will be an unbiased estimator and so the result follows. A similar argument applies to the estimator $\widehat{v}(t)_{CV}$. □

We have just shown that our cross-validation estimators of the variances will conservatively overestimate the true variances, under reasonable assumptions. Such conservatism is an acceptable result for our setting.

Also, the inequality of Eqn. 5.51 is a statement of the well-known argument that taking the mean of the generalisation error estimate over several folds should reduce the overall variance. This is often the primary justification for adopting CV estimates in the first place.

5.3.3. The need for nested CV

Cawley and Talbot [11] discuss a potential issue for our approach of using CV for model comparison: unless we treat "model selection" (i.e. *tuning* of predictors' hyperparameters) as an integral part of the fitting process, our approach would be susceptible to "selection bias" caused by overfitting. Varma and Simon [44] relate this to the estimate of the generalisation error: this "re-substitution estimate" would provide a falsely low estimate of the true quantity, especially when there are few samples. They also explain how to mitigate this potential issue by applying *nested cross validation* with 2 nested iteration loops:

- Outer loop: Estimate the true error for the optimised predictor (i.e. *generalisation error*).
- Inner loop: Perform the optimisation. Train the predictor by computing the CV error estimate for different values of the tuning parameters, then select the parameters with the smallest CV error estimate and use them to train the predictor on all the data within the whole fold.

We will follow their recommendation and design our software package to use nested CV.

6. The pysf package

TODO: UML diagram for compositor pattern, lozenge-ended arrow

TODO: the xpandas reference is appropriate when talking about the data container; may be worth also talking about column-typed data containers, which Markus will probably add to sktime.

So far, we have put in place a conceptual and mathematical framework to define and evaluate prediction strategies in the multi-series supervised forecasting setting. In practical terms, we demonstrate our ideas through the **pysf**[5] Python package, which implements a supervised forecasting data science workflow. The package is open-source and freely available[6]. This section will describe the package's functionality and some of the design choices we have made. Refer to Appendix A for a code demonstration of the package.

6.1. Use cases

Users of our package should be able to:

(U1) Define supervised forecasting prediction strategies based on the ideas in this manuscript, with consistent usage patterns.

(U2) Define other custom prediction strategies to aid in experimentation, including wrapped self-tuning and pipelining predictors, much like **scikit-learn**'s.

(U3) Instantiate baseline supervised forecasting prediction strategies to compare user-defined predictors against.

(U4) Estimate the generalisation error of a set of user-defined predictors for a user-specified error metric over a user-specified set of prediction times for a given multi-series data set.

6.2. Requirements

The package itself fulfils the following core requirements:

(R1) Provide a data container for efficiently storing & safely manipulating multi-series time- and series-labelled data in a single object.

(R2) Enable resampling to be conducted along series indices, particularly using k-fold CV, in an object-oriented manner.

(R3) Within the same code framework, provide an implementation of the conceptual framework of §3.5 for defining supervised forecasting predictors as wrappers around classical predictors.

(R4) Provide an object-oriented framework that implements the statistically-safe evaluation framework of §5, including single-fold and cross-validated estimators for various prediction metrics.

(R5) Provide implementations of self-tuning cross-validating predictors as wrappers, particularly:

  (a) one that optimises over all timestamps.

  (b) one that optimises per prediction timestamp (by selecting the optimal hyperparameters and/or predictor for each) and that multiplexes the outputs together.

(R6) Provide other common components of a data science workflow, as described in Buitinck et al. [9]:

[5] Abbreviates "python supervised forecasting".

[6] The source code is available at `https://github.com/alan-turing-institute/pysf` and documentation at `https://alan-turing-institute.github.io/pysf`. Also, the package can be installed using the popular **pip** package management system; see `https://pypi.org/project/pysf`.

(a) a predictor that pipelines others, by passing the output of one to the input of the next in sequence, thus enabling joint fitting and prediction.

(b) a data transformer framework that enabled transformers to be composed with predictors, with a reference implementation of a smoothing spline transformer.

(R7) Provide a convenient entry point to the user for defining combinations of multiple prediction strategies and sets of training & prediction features, and then performing fair evaluation of each over a common set of prediction times. The cross-validated evaluation metrics of (R4) should be reported to the user, who should also be able to easily visualise these results.

6.3. Software features

In this section, we will review the main features of the **pysf** package, making references to other packages when appropriate.

6.3.1. Predictors & their fit-predict-score workflow

In code, prediction strategies are objects that extend an `AbstractPredictor`, which in turn provides `fit` & `predict` methods that correspond to the theoretical interface methods of §3.2.3. As well as following our interface design, this has direct parallels to **scikit-learn**'s own time-tested `fit-predict` workflow. Buitinck et al. [9] justifies this design from an object-oriented programming (OOP) perspective.

In addition, we provide a `score` method for convenience in the case where test samples are known and we wish to quantify the accuracy of our predictor against them. Internally, it calls `predict` on the known test samples and returns various result objects that allow the user to access residuals and view calculated error metrics (of the 3 types described in Appendix C).

Predictors may inherit from each other; indeed, this is encouraged as it allows them to extend & reuse one another's functionality. For example, some predictors inherit from an `AbstractWindowedPredictor`, which allows them to benefit from common code to store sliding window-related hyperparameters, whether they are single- or multi-series predictors.

A user may define their own prediction strategy, providing they extend & implement `AbstractPredictor`. A number of predefined predictors are provided in the `predictors` module. Figure 6.1 is an inheritance diagram that lists these in a family tree. The predefined predictors fall into 4 categories: baselines (implementations of §3.6), pipelines, wrapping predictors (both to be discussed shortly) and a native supervised forecasting prediction strategy (`MultiCurveKernelsPredictor`, an implementation of Gaussian Process-based regression using the series kernel of **?**].

All predictors solve the task of supervised forecasting. Being able to define them & use them to make predictions using the fit-predict-score workflow are key deliverables of the **pysf** package. We will see related objects that interact with them, drill down into how to implement some, and see how they are used in a wider workflow.

6.3.2. Transformers & pipelining

Transformers do not provide predictions, but instead modify the data in some manner when their `transform` methods are called on a data container. A predefined B-spline smoothing transformer is provided in the `transformers` module, and users may implement their own by extending `AbstractTransformer`.

Transformers are useful as a means to preprocess data being input into a predictor (whether for training or prediction). The `PipelinePredictor` *pipelines* transformers and predictors into a sequence of processing steps for a given data input. It is *initialised with*[7] a chain of transformers and predictors; and when its `fit` or `predict` methods are called it steps through each element in the chain, calling the same method on predictors and `transform` on transformers, and passing the data between each step.

[7] i.e. it is created by calling the class constructor, and this constructor has a parameter that accepts this object as an input.

6.3.3. Data container

Prediction strategies need training data samples and test samples at different points of a workflow. We store multiple samples together in a single `MultiSeries` data container object, with the container providing methods to safely access parts of that data (for example, during resampling operations like cross validation).

Internally, a `MultiSeries` object holds series labels and time labels in a pair of indexed data frames, and maintains references to subsets of the indices that it is allowed to access. It can extract and return these subsets in a variety of different forms. This frees users from having to perform data manipulation on raw arrays or data frames, other than the very first step[8] of instantiating a `MultiSeries` object with an appropriately-formatted pair of **pandas** `DataFrames`. This empowers users who lack machine learning programming experience to use this package.

It is also possible to take copies of data container instances, with or without modifying them, and update the internal state of the instances, but it is strongly discouraged[9] to access the internal state in other ways. These methods are used by the internals of predictors and transformers for the purpose of producing output predictions or transformed datasets; the typical end-user should not need to call them unless they are implementing a new predictor themselves.

6.3.4. Defining prediction strategies through interfacing, wrapping & composition

In Table 3 specifically, and §3 more broadly, we saw 4 ways to wrap a classical prediction strategy in order to transform it to a supervised forecasting prediction strategy.

Our package provides code implementations of these wrappers: `SingleCurveSeriesPredictor`, `MultiCurveTabularPredictor`, `SingleCurveTabularWindowedPredictor` and `MultiCurveTabularWindowedPredictor`. Their constructors expect to be instantiated with a `classical_estimator`; i.e. some object that implements a classical supervised learning or classical forecasting model. Taking advantage of Python's *duck typing* OOP principle to enable compatibility with popular Python packages[10], they simply assume that any such classical estimators have appropriate `fit` and `predict` methods of their own, and make use of those internally.

In this way, we have used the OOP principles of *composition* (between supervised forecasting predictors and inner classical estimators) and *interfaces* (which we expect the inner classical estimators to conform to) in order to implement first-order type *wrapping* of prediction strategies (as introduced in §3.3.1).

6.3.5. Tuning hyperparameters

The other first-order type wrapping in §3.3.1 is of one supervised forecasting predictor around another supervised forecasting predictor. The clearest application of that is for hyperparameter tuning.

We wish to maintain predictors as the key building block of our data science workflow, without the intermediation of other objects. It is thus natural to define predictors that can tune themselves when the `fit` method is called. The procedure is to iterate through or sample some predefined hyperparameter values, and – for each of those values – perform some sort of resampling scheme on the given data to evaluate which set of hyperparameters minimises the prediction error on the data. Once that optimal set has been found, retrain on the full given data with those settings in force.

When `predict` is called, therefore, predictions can be carried out just like for any other predictor. More importantly, such *self-tuning* predictors can also be used as inputs to the performance evaluation workflow.

All the required components of this tuning procedure should be supplied when the self-tuning predictor is instantiated: the inner predictor to wrap around/tune, the feature & error metric to score for, some

[8] To circumvent even this initial step, the package provides convenience methods to download and format some of the datasets described in this dissertation.

[9] Python's OOP framework does not permit us to completely forbid this, unlike many other OOP languages.

[10] **scikit-learn** & **statsmodels** follow these conventions. Buitinck et al. [9] Section 3.4 discusses duck typing.

parameter iterator and some series iteration scheme. The latter two are used internally to perform the iteration and resampling; they are expected to conform (via duck typing) to interfaces used by **scikit-learn** in its `model_selection` module.

In code, the `TuningOverallPredictor` tunes a single wrapped/inner predictor and picks the hyperparmeters that minimise the overall error across all prediction timestamps. The `TuningTimestampMultiplexerPredictor` tunes multiple given predictors and memorises the best combination of prediction strategy and parameter settings per timetstamp; whenever `predict` is called later, the best such combination per prediction timestamp is used to predict for that particular timestamp. The latter predictor is an example of a *multiplexer*.

6.3.6. Workflow to estimate generalisation error

We saw that the fit-predict-score workflow enables a user to train prediction strategies on training series and make predictions for test series. The other major data science workflow of interest is for a user to evaluate the performance of multiple prediction strategies on the same dataset. We studied this problem in §5 and now have all the code we need to implement a workflow to solve it in our package.

The first two ingredients to this workflow are a dataset comprising full series (i.e. with observations available at prediction times) and a set of prediction strategies to be evaluated (including self-tuning predictors wrapping around any predictors that require hyperparameters). The third ingredient is a collection of `Target` objects: each is a set of input & output features and the scoring feature & metric to use in evaluating particular prediction strategies. There is a many-to-many mapping between targets and predictors. A target can be thought of as a descriptor of the fields that should be fed into a predictor, predicted by it, and used to judge its performance against.

The `GeneralisationPerformanceEvaluator` implements this workflow: the method `add_to_targets` defines and caches mappings of various target-predictor combinations, ready to be evaluated, and the `evaluate` method performs the evaluation. The latter involves resampling the full dataset that the object is initialised with (using 5-fold CV as a default scheme), iterating through every target-predictor combination, evaluating the current predictor on each split of the resamples[11], and storing the results. Convenience methods are also provided to chart results and persist them to disk.

[11] We must ensure that the same splits are used for each target-predictor pair, even when resampling is random. This ensures a fair evaluation.

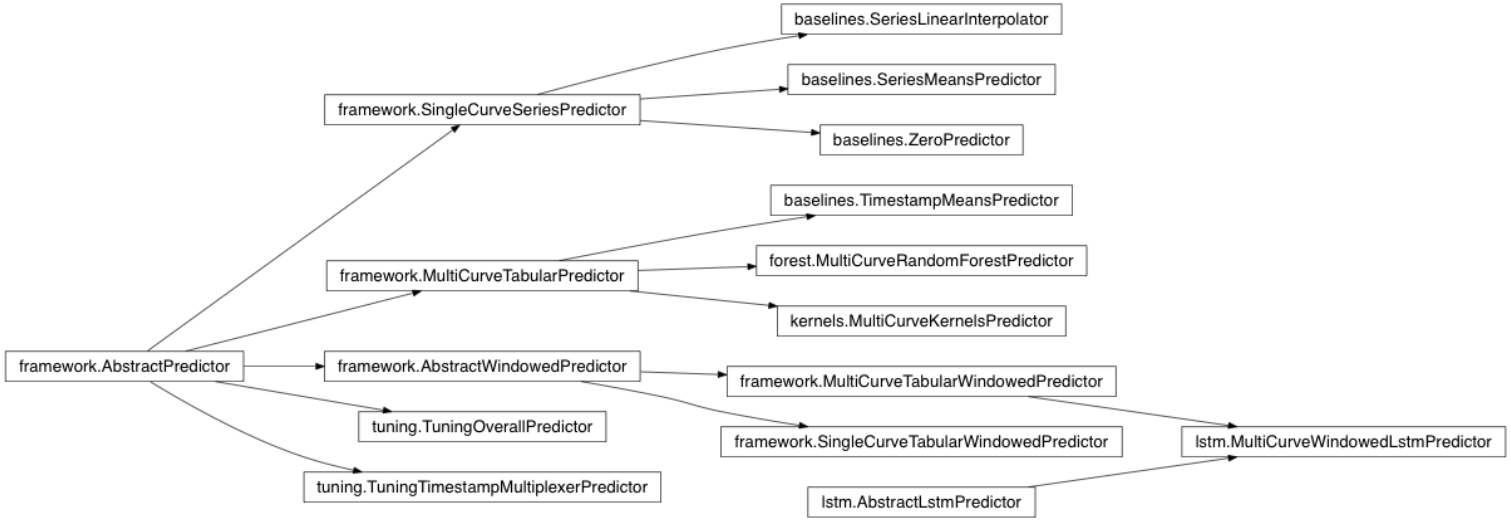

Figure 6.1: Inheritance diagram for objects in the `predictors` module.

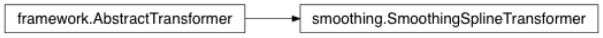

Figure 6.2: Inheritance diagram for objects in the `transformers` module.

6.4. Discussion

To design a software package's internals and API is to balance various tradeoffs in support of one's objectives. Our design decisions have been guided by the following self-imposed principles, which led to certain choices:

(P1) The user should not have to undertake messy and error-prone data manipulation: we have implemented a data container to allow safe views and resampling.

(P2) Our package deals with prediction, so predictors should be the fundamental building block of any wider workflows: we have provided functionality to perform hyperparameter tuning in the form as wrapping self-tuning predictors.

(P3) Our package should be compatible with popular and well-designed machine learning & statistical packages: we have used duck typing and assumed the widely-used `fit-predict` API in external prediction models to enable compatibility with other packages.

(P4) Good OOP practice and *design patterns*[12] should be adopted wherever practicable: we have made extensive use of inheritance & composition, and have used the *flyweight*, *template*, *prototype* & *bridge* patterns to design parts of the API.

In §1.3 we described some relevant contributions of the **mlr** R package to FDA-based prediction[13]. In particular, **mlr** provides a "task transformation" that transforms functional data into features that can be used by classical prediction models, albeit only using domain transformations. In principle, we could implement similar transformers that act on a `MultiSeries` data container instance to transform a dataset in a similar manner. We also benefit from various methods on a `MultiSeries` instance that can extract matrix representations of the data: these representations (of tabular high-dimensional samples, samples of windows, and so on) are compatible with classical predictors without necessitating a change of domain. While the packages' solutions may differ, the underlying philosophy of data transformation is similar, and in principle each could be extended to incorporate the other's transformations.

pandas data frames are too limiting to effectively represent the multi-series data that our supervised forecasting task acts on, so our solution to representing the data has been to implement our own data container. **mlr** is able to rely on R's native data frames to represent functional data because their columns can support a wider range of types. The recent **xpandas**[14] general-purpose data container package may render our specialised data container unnecessary if integrated together; the data extraction functionality and predictors' internals should then be refactored accordingly.

[12] Gamma et al. [17] is the standard reference for design patterns.

[13] We also reviewed the **fda** package, but its basis-focussed representation of functional data is not applicable to other situations and is thus of little relevance.

[14] Available at `https://alan-turing-institute.github.io/xpandas/index.html`.

7. Experiments

We now have all the conceptual and practical tools we need to be able to conduct experiments. We hope these will validate our approach to the supervised forecasting task. We intend to gain some insight into the performance of various classes of prediction strategy compared to common baselines, and to formalise this using statistical hypothesis testing. The results will also illustrate the relative performances of various classes of prediction strategy, within the limits of our experimental setup.

We will maintain the exact same setup for each experiment, including the prediction strategies to be tested, varying only the datasets and prediction times. We will describe this common methodology before presenting the experimental results in the form of tables.

7.1. Methodology

7.1.1. Software packages

All experiments are conducted using the **pysf** package, in particular its infrastructure for wrapping & tuning prediction strategies, and estimating their generalisation error. Predictive models from other packages are used as building blocks: **scikit-learn** & **keras** for classical supervised learning models and **statsmodels** for a classical forecasting model.

7.1.2. Estimating the generalisation error

The results for each experiment consist of a set of generalisation errors for each prediction strategy, which have been estimated using cross validation according to the methodology of §5. They are measured using the Root Mean Squared Error (RMSE) metric of Appendix C.

Most experiments utilise 5-fold CV, where the training/test split in each fold is 80/20% of available series. The exceptions to this are the 3 experiments conducted on power data: these utilise *reverse* 5-fold CV (training/test split of 20/80%) in order to obtain narrower estimates of $\widehat{\nu}_{CV}$ & $\widehat{\nu}(t)_{CV}$

7.1.3. Prediction strategies being evaluated

Table 5 details all of the non-baseline predictors that we have evaluated. The first column of that table maps each non-baseline predictor to one of the 4 supervised forecasting prediction strategies of Table 3.

The kernel predictor (described in **??**) is the only predictor that is native to our supervised forecasting setting. All others are built by wrapping around an inner predictive model; these are drawn from each major class of predictive strategy, as reviewed in §1.3. Of note:

- The ARIMA-based strategy defaults to predicting the series means if its fitting process (which assumes stationarity) proves unstable for a specific set of hyperparameters.
- The LSTM-based strategies' samples consist of windows pooled from all training series.
- *Principal Components Regression* (PCR) & *Partial Least Squares* (PLS) can be considered FDA-based approaches to supervised prediction. Together with the *Principal Components Analysis* (PCA) algorithm, the approaches are described extensively in Hastie et al. [19] Chapters 14.5 & 18.6, and by Ramsay and Silverman [35]. The inner prediction model for any PCR-based strategy is a pipeline, consisting of a transformer that applies the PCA algorithm to reduce the input samples into a number of principal components, followed by a linear regression model whose covariates are the data's projections onto those principal components.
- Smoothing-based strategies are built out of a pipeline whose first component is a transformer that smooths the input series, before passing them downstream to a non-smoothing predictor. *B-splines*, the basis system used to carry out the smoothing, are described by Hastie et al. [19] Chapter 5 and Ramsay and Silverman [35] Chapter 3.

- Most strategies require data to be *standardised*; i.e. transformed so that the training samples have zero mean and unit variance. This is applied in code by means of pipeline strategies where necessary, but is not detailed explicitly for brevity.

All hyperparameters listed in Table 5 are tuned for the prediction feature's overall predictive RMSE using nested 5-fold cross validation, unless they are listed as being fixed[15]. For legibility we have split each predictor's hyperparameters into common and specific sets, but all hyperparameters are nevertheless tuned jointly, including hyperparameters that are specific to each component in a pipeline of predictors. The tuning is implemented by wrapping the predictor (once more) in a self-tuning predictor and specifying ranges from which it may sample hyperparameter values.

[15] Only the (resource-intensive) LSTM-based predictors have some exposed hyperparameters that are fixed to constant values, with other hyperparameters allowed to vary. This is because nested cross validation is computationally intensive even for simple predictors – and more so for the LSTM-based ones.

Class	Class Hyperparams	Strategy Name	Strategy Hyperparams
Single-series	n/a	Linear	n/a
		ElasticNet	regularization parameters `alpha`, `l1_ratio`
		ARIMA	autoregressive order p, differencing degree d, moving-average order q
Single-series Tabular Windowed	input width, output width	Linear Windowed	n/a
		ElasticNet Windowed	regularization parameters `alpha`, `l1_ratio`
Multi-series Tabular	n/a	Linear	n/a
		ElasticNet	regularization parameters `alpha`, `l1_ratio`
		Kernel	λ_{KRR} , $\lambda_{\text{prediction}}$, names of the within-series and between-series kernels, dynamic hyperparams for each of those 2 kernels (as per Table **??**)
		Random Forest	# trees, maximum tree depth, proportion of candidate features to consider
		PCR	# components to use for the PCA step
		PLS	# components to use
		Smoothing Random Forest	Same as the non-smoothing versions above, with the addition of 2 smoothing params: spline degree, smoothing factor
		Smoothing PCR	
		Smoothing PLS	
Multi-series Tabular Windowed	input width, output width, train over prediction times	Linear Windowed	n/a
		ElasticNet Windowed	regularization parameters `alpha`, `l1_ratio`
		LSTM Windowed	Values are fixed for our experiments: # hidden units per layer (64 per layer, with 2 such layers for the Deep LSTM), # training epochs (100), input dropout rate (50%), recurrent dropout rate (0%)
		Deep LSTM Windowed	

Table 5: Non-baseline prediction strategies that we have evaluated in the experiments.

7.1.4. Baselines being evaluated

For comparison, we will also evaluate the 4 baseline predictors described in §3.6.

7.1.5. Datasets

The same experiment is run on the 7 datasets that are described in Table 6. The table also shows the splits between training and prediction timestamps, which are fixed for each dataset. Figures 7.1, 7.2, 7.3, 7.4 & 7.5 are series plots of these datasets.

As structured, all datasets consist of a single (non-time) input feature and a single prediction feature, with the notable exceptions of the Berkeley growth data (3 input features, 2 of which are one-hot encodings of the gender series label) and Canadian weather data (2 output features).

Estimates of the generalisation error are reported separately for each of the 2 individual features of the Canadian weather data. This mirrors the way the predictors are tuned – optimising separately for each (single) output feature, rather than some aggregated value.

Dataset Name		Description	Features		Series Count	Timestamps		Source
			X_i	Z_i		Count	Train/ Predict	
Berkeley growth		Quarterly & semi-annual height readings over 18 years	height (cm)	boy (boolean) & girl (boolean)	93 children	31 ages	22/9	Tuddenham and Snyder [43]
Canadian weather		Year-long daily averages of readings per weather station	temperature (°C) & precipitation (mm)	n/a	35 stations	365 days	300/65	Ramsay and Silverman [35]
ECG		One patient's cardiac activity, including both healthy & pathological cardiac cycles	potential difference (mV)	n/a	200 heartbeats	96 timestamps	48/48	Olszewski [32]
Power	**multiple sites & multiple days**	Day-long series of half-hourly energy consumption by SMEs	consumption (kWh)	n/a	300 combinations	48 half-hours	24/24	Sampled from Sidebotham [42] dataset "TC1b"
	multiple sites for a single day	Day-long series of half-hourly energy consumption by SMEs	consumption (kWh)	n/a	300 sites	48 half-hours	24/24	
	multiple days for a single site	Day-long series of half-hourly energy consumption by SMEs	consumption (kWh)	n/a	300 days	48 half-hours	24/24	
Starlight		Aligned photometric readings of the brightness of periodic stars (i.e. light-curves)	aligned magnitude (log-brightness)	n/a	100 stars	1000 aligned timestamps	500/500	Sampled from Rebbapragada et al. [38]

Table 6: Side-by-side comparison of the 7 datasets upon which we have conducted experiments.

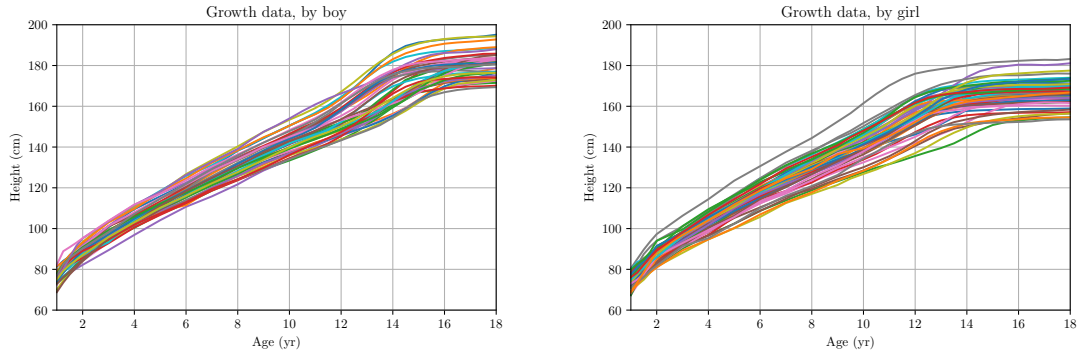

Figure 7.1: Series plots of the Berkeley growth data. Each series is labelled as a boy or girl, so the data has been divided into 2 corresponding plots. Height is the only time-indexed feature.

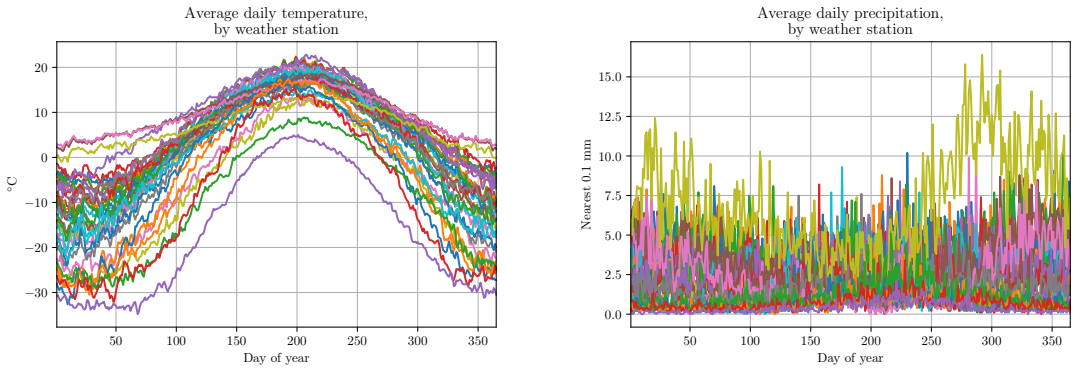

Figure 7.2: Series plots of the Canadian weather data, with each of the 2 features (average temperature & average precipitation) plotted separately.

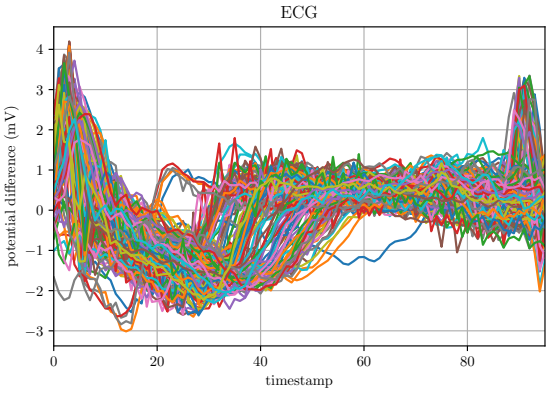

Figure 7.3: Series plot of the ECG dataset.

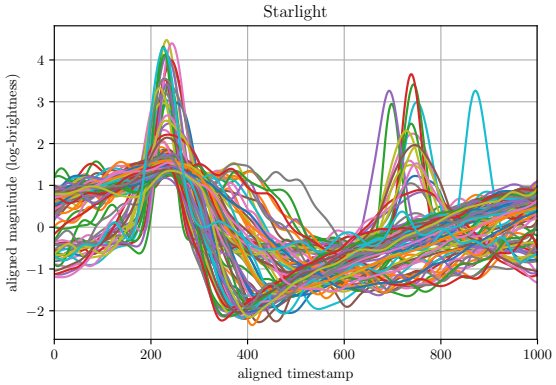

Figure 7.4: Series plot of the Starlight dataset.

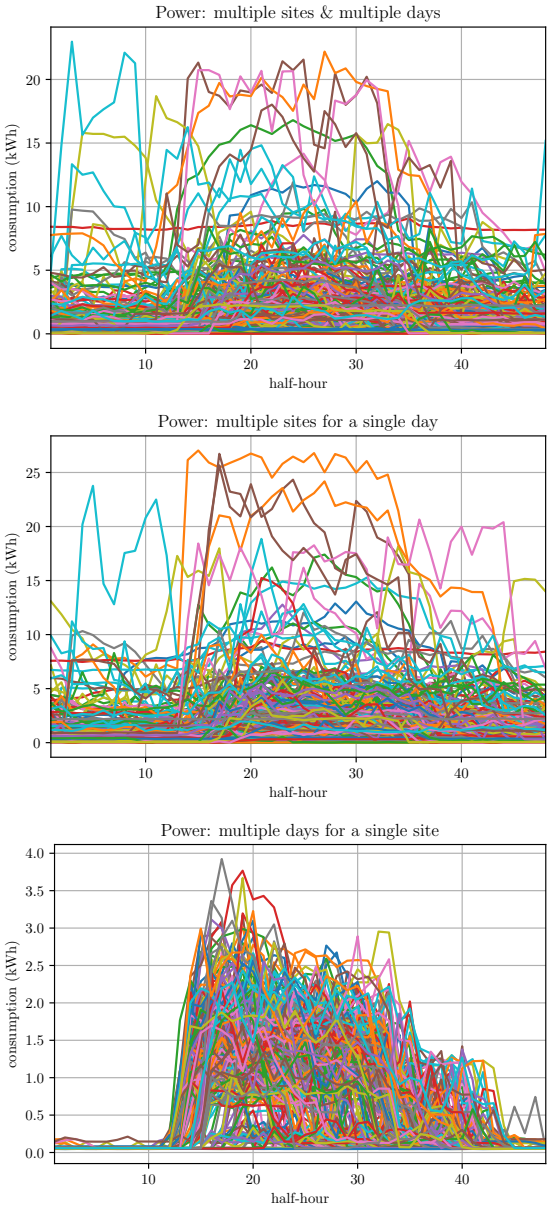

Figure 7.5: Series plots of all 3 Power datasets.

7.1.6. Significance testing

Although we cannot soundly test whether single- or multi-series predictors in general perform best[16], we do have the data to test whether a particular individual predictor is better than the best baseline for a particular dataset. The p-values produced by this hypothesis testing procedure are presented in Table 9, with associated significance levels in Table 10.

We conduct the significance testing by selecting the best baseline for each experiment and comparing it to each of the non-baseline predictors using one-tailed two-sample t-tests. Our t-statistic

$$t = \frac{\widehat{\varepsilon}_{CV}^{\text{predictor}} - \widehat{\varepsilon}_{CV}^{\text{best baseline}}}{\sqrt{(\widehat{v}_{CV}^{\text{predictor}})^2 + (\widehat{v}_{CV}^{\text{best baseline}})^2}} \tag{7.1}$$

assumes unequal population variances, with S.E. estimates given by $\widehat{v}_{CV}^{\text{predictor}}$ & $\widehat{v}_{CV}^{\text{best baseline}}$. The one-tailed null & alternative hypotheses

$$\begin{aligned} \mathrm{H}_0 &: \quad \widehat{\varepsilon}_{CV}^{\text{predictor}} - \widehat{\varepsilon}_{CV}^{\text{best baseline}} \geq 0 \\ \mathrm{H}_1 &: \quad \widehat{\varepsilon}_{CV}^{\text{predictor}} < \widehat{\varepsilon}_{CV}^{\text{best baseline}} \end{aligned}$$

should give us greater power to detect H_1 than a two-tailed test would have provided.

The degrees of freedom parameter of the null t-distribution is the number of series in the full dataset that we use to calculate our generalisation error estimates from. For the majority of our datasets we calculate these using 5-fold CV, so the d.f. is $1/5$ the total number of series of the relevant dataset. The exceptions are the 3 Power datasets, to which we apply reverse 5-fold CV, resulting in a d.f. parameter of $4/5$ the total number of Power series.

As a precedent, Nadeau and Bengio [31] studied the performance of t-tests on differences of generalisation errors (much like the numerator in our t-statistic), albeit in the differing situation of varying training set sizes. An alternative approach would have been to conduct a Wilcoxon signed rank test; this would not have imposed any parametric assumptions on the data but would also not have taken advantage of the variance estimates that we have available.

[16] To do so, we would need to rerun the experiments with all strategies of the appropriate class wrapped in a *multiplexing* predictor – i.e. one that uses the best-performing sub-strategy on a training set to predict on a test set – within the outer CV iteration, and then compare the estimated generalisation errors of the single-series multiplexing predictor against the multi-series multiplexing predictor.

7.2. Results

As well as the statistical testing tables already mentioned, Table 7 presents the estimated overall generalisation errors and Table 8 ranks them for convenience. Appendix D presents the same results in the form of charts, together with additional per-timestamp results.

Some results are not available and are denoted by ... in the tables. These fall into 2 categories:

- Failure to complete: on the Starlight dataset, generalisation error estimation for the LSTM-based prediction strategies failed to complete even after being run for several weeks on a desktop PC. This is directly due to the resource-hungriness of LSTM-based predictors, and is most apparent on the Starlight dataset because this dataset has the most timestamps and hence longest sequential samples. The use of nested CV only adds to this demand for computational resources.

- Failure to converge: on the 3 Power datasets, a number of strategies failed to converge on at least 1 CV fold during generalisation error estimation. Recall that we applied reverse 5-fold CV to these 3 datasets alone, in order to reduce the variance of the estimates. It is possible that by reducing the number of training samples in each fold (by a factor of 4 compared to regular 5-fold CV), we starved certain prediction strategies of training data that would have allowed them to generalise better on the test series.

Recall that the Berkeley growth & Canadian weather datasets each possess multiple input or output features. We call a specific configuration of input & output features a *target*. Where multiple targets are possible for a given dataset, we evaluate them all and report the best results. Target details are listed in the charts in Appendix D but omitted from the tables in this section, for conciseness.

Estimated generalisation errors for the 4 baselines are included at the bottom of Table 7.

Predictor			Canadian weather			Power			
Series	Description	Berkeley growth	precipitation feature	temperature feature	ECG	many days, single site	many sites and days	many sites, single day	Starlight
Multi	Deep LSTM WD	31.35 ± 0.9	2.26 ± 0.73	16.44 ± 2.45	0.76 ± 0.04	0.56 ± 0.02	1.93 ± 0.19	2.46 ± 0.27	—
Multi	ElasticNet	2.43 ± 0.43	1.08 ± 0.26	1.38 ± 0.22	0.29 ± 0.03	0.33 ± 0.01	1.37 ± 0.13	1.58 ± 0.19	0.26 ± 0.06
Multi	ElasticNet WD	4.57 ± 0.44	0.99 ± 0.25	4.32 ± 0.34	0.45 ± 0.03	0.51 ± 0.02	1.69 ± 0.18	2.03 ± 0.25	0.69 ± 0.04
Multi	Kernel	5.44 ± 0.68	1.09 ± 0.27	1.91 ± 0.36	0.29 ± 0.03	0.33 ± 0.01	1.62 ± 0.19	1.62 ± 0.18	0.26 ± 0.06
Multi	Linear	2.43 ± 0.36	1.04 ± 0.25	1.08 ± 0.2	0.33 ± 0.03	1.43 ± 0.31	3.48 ± 0.72	4.78 ± 1.06	0.57 ± 0.08
Multi	Linear WD	1.66 ± 0.19	0.99 ± 0.21	2.74 ± 0.4	0.6 ± 0.07	0.47 ± 0.01	1.66 ± 0.18	2.08 ± 0.27	2.65 ± 0.31
Multi	LSTM WD	30.39 ± 0.86	2.13 ± 0.58	14.77 ± 2.47	0.74 ± 0.04	0.58 ± 0.02	1.98 ± 0.2	2.39 ± 0.3	—
Multi	PCR	2.74 ± 0.42	1.29 ± 0.32	1.77 ± 0.32	0.29 ± 0.03	0.35 ± 0.01	1.45 ± 0.16	1.72 ± 0.21	0.27 ± 0.06
Multi	PLS	7.25 ± 0.78	1.53 ± 0.36	5.52 ± 0.94	0.41 ± 0.04	—	1.92 ± 0.19	2.18 ± 0.23	0.61 ± 0.1
Multi	RF	3.27 ± 0.48	1.31 ± 0.51	3.43 ± 0.7	0.28 ± 0.03	0.33 ± 0.01	1.42 ± 0.14	1.76 ± 0.21	0.29 ± 0.06
Multi	SM PCR	1 ± 0.25	1.16 ± 0.29	1.86 ± 0.32	0.3 ± 0.03	0.23 ± 0.01	—	—	1.2 ± 0.92
Multi	SM PLS	7.62 ± 0.79	1.49 ± 0.34	5.51 ± 1.01	0.48 ± 0.04	1.24 ± 0.04	1.82 ± 0.17	2.11 ± 0.22	0.6 ± 0.06
Multi	SM RF	3.4 ± 0.5	1.3 ± 0.5	3.66 ± 0.71	0.28 ± 0.03	0.29 ± 0.01	1.37 ± 0.13	1.72 ± 0.21	0.31 ± 0.06
Single	ARIMA	13.77 ± 1.87	1.28 ± 0.36	6.05 ± 1.88	0.95 ± 0.07	1.4 ± 0.64	—	4.26 ± 2.05	1.09 ± 0.1
Single	ElasticNet	10.29 ± 0.82	1.38 ± 0.41	12.07 ± 1.16	0.94 ± 0.05	0.55 ± 0.03	1.73 ± 0.33	2.11 ± 0.51	1.09 ± 0.1
Single	ElasticNet WD	6.84 ± 0.56	1.33 ± 0.32	5.81 ± 0.57	1.16 ± 0.07	0.57 ± 0.03	2.63 ± 1.32	2.1 ± 0.51	1.28 ± 0.19
Single	Linear	14.88 ± 1.04	1.85 ± 0.41	30.38 ± 2.56	2.56 ± 0.23	2.22 ± 0.13	4.64 ± 1.24	5.29 ± 1.44	2.59 ± 0.25
Single	Linear WD	7.41 ± 0.44	2.8 ± 1.3	12.83 ± 2.35	1.73 ± 0.14	—	4.56 ± 2.05	—	1.49 ± 0.13
Multi	BL TS means	9.59 ± 1.38	2.28 ± 0.56	8.61 ± 1.34	0.43 ± 0.03	0.45 ± 0.01	2.3 ± 0.2	2.78 ± 0.29	0.59 ± 0.06
Single	BL 0 values	169.78 ± 1.58	3.15 ± 0.96	10.72 ± 1.91	0.59 ± 0.03	0.69 ± 0.02	2.76 ± 0.23	3.38 ± 0.32	0.78 ± 0.04
Single	BL LIP	8.07 ± 0.74	1.35 ± 0.34	9.97 ± 1.04	0.87 ± 0.08	0.94 ± 0.03	2.31 ± 0.32	2.67 ± 0.36	1.14 ± 0.07
Single	BL means	46.45 ± 0.83	1.38 ± 0.41	12.07 ± 1.16	0.94 ± 0.05	0.55 ± 0.01	1.75 ± 0.17	2.06 ± 0.22	1.09 ± 0.1

Table 7: Estimated overall generalisation RMSE with 1 s.e. (i.e. $\widehat{\varepsilon}_{CV} \pm \widehat{\nu}_{CV}$), aggregated over time by an unweighted mean. This table shows the 4 baselines' results at the bottom. New abbreviations: "WD"="Windowed", "RF"="Random Forest", "SM"="Smoothing", "BL"="Baseline", "TS"="Timestamp", "LIP"="Linear Interpolator". Units are listed in the third column of Table 6.

Predictor			Canadian weather			Power			
Series	Description	Berkeley growth	precipitation feature	temperature feature	ECG	many days, single site	many sites and days	many sites, single day	Starlight
Multi	Deep LSTM Windowed	18	17	17	13	10	11	13	—
Multi	ElasticNet	3	4	2	5	4	2	1	2
Multi	ElasticNet Windowed	8	2	9	9	8	7	6	9
Multi	Kernel	9	5	5	4	5	5	2	1
Multi	Linear	4	3	1	7	15	14	15	6
Multi	Linear Windowed	2	1	6	11	7	6	7	16
Multi	LSTM Windowed	17	16	16	12	12	12	12	—
Multi	PCR	5	8	3	3	6	4	4	3
Multi	PLS	11	14	11	8	—	10	11	8
Multi	Random Forest	6	10	7	1	3	3	5	4
Multi	Smoothing PCR	1	6	4	6	1	—	—	12
Multi	Smoothing PLS	13	13	10	10	13	9	10	7
Multi	Smoothing Random Forest	7	9	8	2	2	1	3	5
Single	ARIMA	15	7	13	15	14	—	14	11
Single	ElasticNet	14	12	14	14	9	8	9	10
Single	ElasticNet Windowed	10	11	12	16	11	13	8	13
Single	Linear	16	15	18	18	16	16	16	15
Single	Linear Windowed	12	18	15	17	—	15	—	14

Table 8: Relative ranking of the non-baseline predictors in Table 7 by RMSE value. 1=best.

Predictor			Canadian weather			Power			
Series	Description	Berkeley growth	precipitation feature	temperature feature	ECG	many days, single site	many sites and days	many sites, single day	Starlight
Multi	Deep LSTM Windowed	1.00	0.85	0.99	1.00	1.00	0.76	0.87	—
Multi	ElasticNet	0.00	0.28	0.00	0.00	0.00	0.04	0.05	0.00
Multi	ElasticNet Windowed	0.00	0.22	0.01	0.65	1.00	0.42	0.46	0.91
Multi	Kernel	0.01	0.29	0.00	0.00	0.00	0.31	0.06	0.00
Multi	Linear	0.00	0.24	0.00	0.01	1.00	0.99	0.99	0.44
Multi	Linear Windowed	0.00	0.20	0.00	0.98	0.90	0.37	0.52	1.00
Multi	LSTM Windowed	1.00	0.86	0.97	1.00	1.00	0.81	0.81	—
Multi	PCR	0.00	0.46	0.00	0.00	0.00	0.10	0.14	0.00
Multi	PLS	0.23	0.64	0.05	0.39	—	0.75	0.65	0.56
Multi	Random Forest	0.00	0.47	0.01	0.00	0.00	0.07	0.17	0.00
Multi	Smoothing PCR	0.00	0.35	0.00	0.00	0.00	—	—	0.74
Multi	Smoothing PLS	0.34	0.61	0.05	0.84	1.00	0.63	0.57	0.57
Multi	Smoothing Random Forest	0.00	0.47	0.01	0.00	0.00	0.04	0.14	0.00
Single	ARIMA	0.99	0.45	0.15	1.00	0.93	—	0.86	1.00
Single	ElasticNet	0.97	0.52	0.95	1.00	1.00	0.49	0.54	1.00
Single	ElasticNet Windowed	0.10	0.48	0.05	1.00	1.00	0.75	0.53	1.00
Single	Linear	1.00	0.81	1.00	1.00	1.00	0.99	0.99	1.00
Single	Linear Windowed	0.23	0.84	0.92	1.00	—	0.91	—	1.00

Table 9: p-values for the one-tailed two-sample t-tests detailed in §7.1.6.

Predictor		Canadian weather				Power			
Series	Description	Berkeley growth	precipitation feature	temperature feature	ECG	many days, single site	many sites and days	many sites, single day	Starlight
Multi	Deep LSTM Windowed								—
Multi	ElasticNet	***		***	***	***	**	*	***
Multi	ElasticNet Windowed	***		***					
Multi	Kernel	***		***	***	***		*	***
Multi	Linear	***		***	**				
Multi	Linear Windowed	***		***					
Multi	LSTM Windowed								—
Multi	PCR	***		***	***	***			***
Multi	PLS			*		—			
Multi	Random Forest	***		***	***	***	*		***
Multi	Smoothing PCR	***		***	***	***	—	—	
Multi	Smoothing PLS			*					
Multi	Smoothing Random Forest	***		***	***	***	**		***
Single	ARIMA						—		
Single	ElasticNet								
Single	ElasticNet Windowed			**					
Single	Linear								
Single	Linear Windowed					—		—	

Table 10: Results of testing the hypotheses detailed in §7.1.6. Stars indicate the null hypothesis was rejected with a given level of confidence: ***=99%, **=95%, *=90%.

7.3. Discussion

When considering the relative rankings of the predictors' performance across all 7 datasets (Table 8), one obvious interpretation might be that multi-series methods seem to outperform single-series ones, although we should take care in making that statement[17]. We are on safer ground when we state that single-series methods almost always underperform the best baseline, leading us to conclude that there is very little evidence for single-series predictors performing well[18] for our supervised forecasting task and that – by contrast – we have widespread evidence for multi-series predictors performing well, in all but one[19] of our experiments..

These results provide an *ex post* justification to studying the problem of supervised forecasting, at the very heart of which is to take advantage of the presence of multiple series for prediction. Indeed, these results indicate that both approaches we took to solving the task (interface-based and native kernel-based) are justified, and any differences in performance between the two seem immaterial.

When considering individual strategies, PLS-based methods tend to rank poorly in the multi-series case, while LSTM-based methods are the only multi-series methods to consistently underperform the best baselines, often to a large extent. The excitement around sequential neural networks notwithstanding, LSTM-based techniques are very resource-intensive and have performed poorly in our experiments. While it could be argued that we did not sufficiently explore the parameter space due to resource constraints, it is worth pointing out that a literature review did not find many instances of nested CV being performed on LSTM models, which suggests others may also have found their resource-intensiveness a barrier to successful study.

Recall that the 3 Power datasets have been sampled from a single large panel; this allows us a potential insight into the effectiveness of different data gathering procedures for forecasting purposes. By counting the number of predictors that perform well on each sampled dataset, we hypothesise that restricting the number of sites (i.e. subjects in the wider population panel) improves predictability, while mixing sites but fixing the day degrades predictability. That would have interesting practical implications for forecasting commercial electricity demand, by suggesting that historical data for a given site is of more use in making forecasts than data for the current day drawn from other sites is.

A final element of interest is whether the inclusion of series labels Z_i as covariates improves predictive performance for multi-series predictors[20]. Figure D.1 in the Appendix shows generalisation error estimates and target details for the individual Berkeley growth experiment, our only dataset where series labels are available. For almost all predictors where the additional gender (series) labels are available, including most autoregressive linear-like ones, the presence of these additional inputs does indeed improve the predictive accuracy, compared to predicting from height (time labels) alone.

[17] We should re-emphasise that we cannot state that multi-series methods outperform single-series ones in a statistically sound manner. This is because our hypothesis testing was not set up to make a direct comparison between the 2 classes. See an earlier footnote on page 51.

[18] We interpret a predictor outperforming its best baseline as "performing well".

[19] That one exception is the precipitation feature of the Canadian weather data. Figure 7.6 indicates that this feature is relatively noisy and difficult to decompose into principal components, both compared to the temperature feature of the same dataset and to the features of another dataset. This noisiness appears to have made the temperature feature less predictable. In fact, many predictors do have lower generalisation error estimates than the best baseline, but the variances of those estimates are not low enough in order for the differences to be statistically significant.

[20] Such series labels are not relevant to single-series prediction strategies.

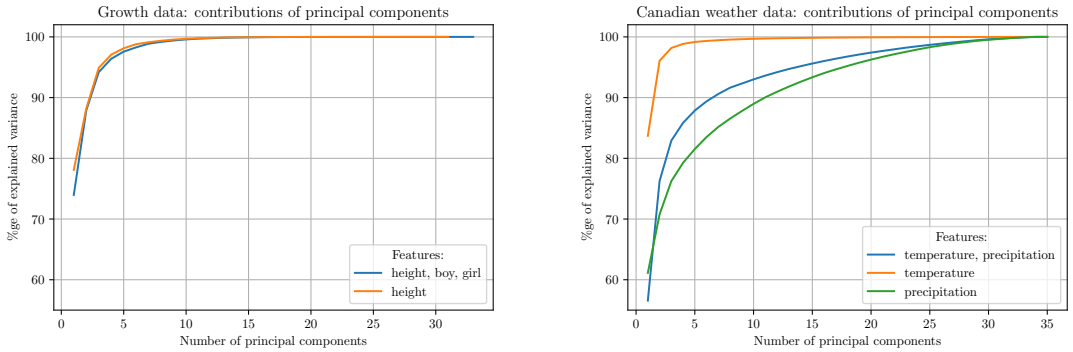

Figure 7.6: Scree plots: percentage of variance explained by the first given principal components derived from a tabular representation of the univariate Berkeley growth data and the multivariate Canadian weather data. Note the noisiness of the precipitation feature (green, on the right).

8. Conclusion

8.1. Contributions & findings

Our aim in writing this dissertation has been to investigate whether we can make better forecasts by taking advantage of the presence of additional series, when they are available.

Our approach has been to frame this question as a predictive task to be solved by an appropriate prediction strategy. In designing such strategies, we took two parallel approaches: wrapping around existing models (to be drawn from a variety of classes) and a native solution based around a kernel that we derived for this purpose. Furthermore, in order to test whether multi-series prediction strategies outperform baselines, we developed a statistically-sound framework for model evaluation, through an empirical estimation of the generalisation error. These contributions form the theoretical deliverable of our research project.

In addition, the open-source **pysf** Python package is our project's practical, code deliverable. By releasing it publicly, we hope not only to give life to our theoretical contributions, but also to contribute to the emergent ecosystem of machine learning software packages. In §6.4 we discussed the design choices we made, articulated some principles and related our package to the wider ecosystem; we hope this might inform further developments in related areas.

With the 2 core deliverables in place, we turned to the question of empirically validating our approach. We collected 7 panel datasets and built prediction strategies based on the major classes of machine learning regression-type models, FDA-based & stochastic statistical models, and our native kernel-based strategy. In the hope of answering our initial question, these included both single- and multi-series strategies.

We discussed the experimental results in detail in §7.3; in short, we are satisfied that many multi-series predictors outperform their baselines while almost no single-series predictors manage to achieve that. We believe the results validate our initial research question, while providing some initial data on which classes of model work best. Gratifyingly, both our native kernel-based predictor and various composite interface-based predictors perform well on a range of datasets.

8.2. Future work

We hope that the encouraging results might stimulate some additional work on the task of supervised forecasting. We suggest the following avenues of theoretical research:

- What new interface-based and native strategies could be designed? These could be evaluated against our results.
- What prediction strategies might continue to perform well if our assumption of i.i.d. series is relaxed? We suggest our kernel-based strategy would be able to model such dependencies between series. How would we adapt the generalisation error estimators? Perhaps they would necessitate the imposition of some parametric model?
- When observations are missing, and somehow interpolated, how might we adapt the generalisation error estimates? Perhaps weighting or imputation schemes might be used, as in Little and Rubin [26]?
- How might we better conduct significance testing of the difference between predictors' performances? Perhaps a nonparametric model is more efficient for this task?

We also hope that our release of the open-source **pysf** package might encourage suggestions and contributions to the package:

- One fruitful area is likely to be the implementation of more data transformers: these might include domain transformations (like **mlr**'s), transformations to make large data sets more tractable, or transformations to enable prediction on sparse data. For the latter, experimenting on sparse data using the pre-existing smoothing transformer is likely to be a useful starting point.

- As discussed in §6.4, another contribution might be to integrate **pysf** with a more general-purpose data container, such as the one provided by the **xpandas** pacakge. Indeed, this could be an intermediate step towards a higher goal of integrating **pysf** into some wider data science workflow.
- Other contributions might be software-focussed in nature, such as parallelising iterations where possible.

Finally, we can suggest some further experiments and empirical research questions to tackle:

- Which predictors perform best with a smaller/larger number of training series? Which predictors perform best on a smaller/larger number of prediction time points? Generalisation error estimates could be computed for a range of scenarios to answer these questions.
- Would LSTM-based strategies benefit from additional searching of the hyperparameter space? Researchers with access to sufficient computational resources might simply rerun existing experiments on a wider search space.
- Can we directly say that multi-series predictors perform better than single-series predictors? As discussed in §7.1.6, building multiplexing predictors for each major class and rerunning experiments using those wrappers would help to answer this.
- Similar multiplexing predictors could be used to compare any class of predictors to any other; for example, do smoothing strategies perform better than non-smoothing ones?
- As discussed in §7.3, what sampling schemes from larger datasets might be best to apply supervised forecasting predictors to? This might have particular relevance to electricity demand forecasting.

References

[1] A Aizerman, EM Braverman, and LI Rozoner. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

[2] Sylvain Arlot and Alain Celisse. A survey of cross-validation procedures for model selection. *Statistics Surveys*, 4:40–79, 2010. doi: 10.1214/09-SS054. URL http://dx.doi.org/10.1214/09-SS054.

[3] Badi Baltagi. *Econometric Analysis of Panel data*. Wiley, 3 edition, 2008.

[4] P. Billingsley. *Probability and Measure*. Wiley, 3 edition, 1995. ISBN 9780471007104.

[5] Bernd Bischl, Michel Lang, Lars Kotthoff, Julia Schiffner, Jakob Richter, Erich Studerus, Giuseppe Casalicchio, and Zachary M. Jones. mlr: Machine learning in r. *Journal of Machine Learning Research*, 17(170):1–5, 2016. URL http://jmlr.org/papers/v17/15-066.html.

[6] Christopher M Bishop et al. *Pattern recognition and machine learning*, volume 4. springer New York, 2006.

[7] Edwin V Bonilla, Kian M Chai, and Christopher Williams. Multi-task gaussian process prediction. In *Advances in neural information processing systems*, pages 153–160, 2008.

[8] George E. P. Box, Gwilym M. Jenkins, and Gregory C. Reinsel. *Time Series Analysis, Forecasting, and Control*. Francisco Holden-Day, 1970.

[9] Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pages 108–122, 2013.

[10] George Casella and Roger L Berger. *Statistical Inference*. Duxbury, 2 edition, 2002.

[11] Gavin C. Cawley and Nicola L.C. Talbot. On over-fitting in model selection and subsequent selection bias in performance evaluation. *Journal of Machine Learning Research*, 11:2079–2107, 2010. ISSN 1532-4435. URL http://jmlr.csail.mit.edu/papers/volume11/cawley10a/cawley10a.pdf.

[12] Marco Cuturi, Jean-Philippe Vert, Oystein Birkenes, and Tomoko Matsui. A kernel for time series based on global alignments. In *Acoustics, Speech and Signal Processing, 2007. ICASSP 2007. IEEE International Conference on*, volume 2, pages II–413. IEEE, 2007.

[13] Vitaly Davydov and Franz J Király. xpandas-python data containers for structured types and structured machine learning tasks. 2018.

[14] Thomas G. Dietterich. Machine learning for sequential data: A review. In Terry Caelli, Adnan Amin, Robert P. W. Duin, Dick de Ridder, and Mohamed Kamel, editors, *Structural, Syntactic, and Statistical Pattern Recognition: Joint IAPR International Workshops SSPR 2002 and SPR 2002 Windsor, Ontario, Canada, August 6–9, 2002 Proceedings*, pages 15–30. Springer, 2002. ISBN 978-3-540-70659-5. doi: 10.1007/3-540-70659-3_2. URL https://doi.org/10.1007/3-540-70659-3_2.

[15] Peter Diggle, Peter J Diggle, Patrick Heagerty, Patrick J Heagerty, Kung-Yee Liang, Scott Zeger, et al. *Analysis of Longitudinal Data*. Oxford University Press, 2 edition, 2013.

[16] Bradley Efron and Trevor Hastie. *Computer Age Statistical Inference: Algorithms, Evidence, and Data Science*. Institute of Mathematical Statistics Monographs. Cambridge University Press, 2016. doi: 10.1017/CBO9781316576533.

[17] Erich Gamma, Richard Helm, Ralph Johnson, and John Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Professional Computing Series. Addison-Wesley, 1994.

[18] Jan G. De Gooijer and Rob J. Hyndman. 25 years of time series forecasting. *International Journal of Forecasting*, 22(3):443 – 473, 2006. ISSN 0169-2070. doi: https://doi.org/10.1016/j.ijforecast.2006.01.001. URL http://www.sciencedirect.com/science/article/pii/S0169207006000021.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2 edition, 2009. URL http://www-stat.stanford.edu/~tibs/ElemStatLearn/.

[20] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural Computation*, 9(8): 1735–1780, 1997. ISSN 0899-7667. doi: 10.1162/neco.1997.9.8.1735. URL http://dx.doi.org/10.1162/neco.1997.9.8.1735.

[21] Stephan Hoyer and Joe Hamman. xarray: Nd labeled arrays and datasets in python. *Journal of Open Research Software*, 5(1), 2017.

[22] Richard A. Johnson and Dean W. Wichern. *Applied Multivariate Statistical Analysis*. Pearson, 6 edition, 2007. ISBN 0131877151.

[23] James Max Kanter and Kalyan Veeramachaneni. Deep feature synthesis: Towards automating data science endeavors. In *2015 IEEE International Conference on Data Science and Advanced Analytics, DSAA 2015, Paris, France, October 19-21, 2015*, pages 1–10. IEEE, 2015.

[24] Joel Kariel. Prediction in functional data analysis: a new kernel approach and an application in quantifying human strength. MRes dissertation, 2016.

[25] John Langford, Roberto Oliveira, and Bianca Zadrozny. Predicting conditional quantiles via reduction to classification. In *Proceedings of the Twenty-Second Conference Annual Conference on Uncertainty in Artificial Intelligence (UAI-06)*, pages 257–264, Arlington, Virginia, 2006. AUAI Press.

[26] Roderick JA Little and Donald B Rubin. *Statistical Analysis with Missing Data*. John Wiley & Sons, 2 edition, 2014.

[27] Wes McKinney. Data structures for statistical computing in python. In Stéfan van der Walt and Jarrod Millman, editors, *Proceedings of the 9th Python in Science Conference*, pages 51 – 56, 2010.

[28] Paul Meier. Variance of a weighted mean. *Biometrics*, 9(1):59–73, 1953. ISSN 0006341X, 15410420. URL http://www.jstor.org/stable/3001633.

[29] Charles A Micchelli and Massimiliano Pontil. Kernels for multi–task learning. In *Advances in neural information processing systems*, pages 921–928, 2005.

[30] Kevin P. Murphy. *Machine Learning: A Probabilistic Perspective*. MIT Press, 2012. ISBN 0262018020, 9780262018029.

[31] Claude Nadeau and Yoshua Bengio. Inference for the generalization error. *Machine Learning*, 52(3): 239–281, 2003. ISSN 1573-0565. doi: 10.1023/A:1024068626366. URL https://doi.org/10.1023/A:1024068626366.

[32] Robert Thomas Olszewski. *Generalized Feature Extraction for Structural Pattern Recognition in Time-series Data*. PhD thesis, 2001.

[33] E Omey and S Van Gulck. Central Limit Theorems for variances and correlation coefficients, 2008.

[34] J. Ramsay. When the data are functions. *Psychometrika*, 47(4):379–396, 1982. ISSN 1860-0980. doi: 10.1007/BF02293704. URL http://dx.doi.org/10.1007/BF02293704.

[35] J. Ramsay and B. Silverman. *Functional Data Analysis*. Springer, 2 edition, 2005.

[36] C. Radhakrishna Rao. Some statistical methods for comparison of growth curves. *Biometrics*, 14(1):1–17, 1958. ISSN 0006341X, 15410420. URL http://www.jstor.org/stable/2527726.

[37] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. Adaptive Computation and Machine Learning. MIT Press, 2006.

[38] Umaa Rebbapragada, Pavlos Protopapas, Carla E. Brodley, and Charles Alcock. Finding anomalous periodic time series. *Machine Learning*, 74(3):281–313, Mar 2009. ISSN 1573-0565. doi: 10.1007/s10994-008-5093-3. URL https://doi.org/10.1007/s10994-008-5093-3.

[39] Philip T Reiss and R. Todd Ogden. Functional principal component regression and functional partial least squares. *Journal of the American Statistical Association*, 102(479):984–996, 2007. doi: 10.1198/016214507000000527.

[40] Bernhard Schölkopf and Alexander J Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2001.

[41] Hiroshi Shimodaira, Ken-ichi Noma, Mitsuru Nakai, and Shigeki Sagayama. Dynamic time-alignment kernel in support vector machine. In *Advances in neural information processing systems*, pages 921–928, 2002.

[42] Liz Sidebotham. Customer-led network revolution project closedown report. Technical report, Northern Powergrid (Northeast) Ltd, 2015. URL https://www.ofgem.gov.uk/sites/default/files/docs/2015/05/clnr-g026_project_closedown_report_final_v2.pdf.

[43] Read D Tuddenham and Margaret M Snyder. Physical growth of california boys and girls from birth to eighteen years. *Publications in child development. University of California, Berkeley*, 1(2):183, 1954.

[44] Sudhir Varma and Richard Simon. Bias in error estimation when using cross-validation for model selection. *BMC Bioinformatics*, 7(1):91, 2006.

[45] Ronald J. Williams and David Zipser. Backpropagation. In Yves Chauvin and David E. Rumelhart, editors, *Gradient-based Learning Algorithms for Recurrent Networks and Their Computational Complexity*, pages 433–486. L. Erlbaum Associates Inc., Hillsdale, NJ, USA, 1995. ISBN 0-8058-1259-8. URL http://dl.acm.org/citation.cfm?id=201784.201801.

A. Walkthrough of the pysf package

This appendix aims to introduce the reader to the **pysf** package in an accessible manner by means of a walkthrough of a typical data science workflow.

Our goal is to estimate the generalisation error of a multi-series multivariate tabular Smoothing PCR predictor (one of a class of FDA-related methods) in relation to some baselines, on the last 65 days of the annual weather data.

We begin by loading and visualising the data. The results of this snippet are displayed in Fig. A.1:

```
1 from pysf.data import download_ramsay_weather_data_dfs, MultiSeries
2
3 (weather_vs_times_df, weather_vs_series_df) = download_ramsay_weather_data_dfs()
4 data_weather = MultiSeries(data_vs_times_df=weather_vs_times_df,
      data_vs_series_df=weather_vs_series_df, time_colname='day_of_year',
      series_id_colnames='weather_station')
5
6 data_weather.visualise()
7 data_weather.visualise_moments()
```

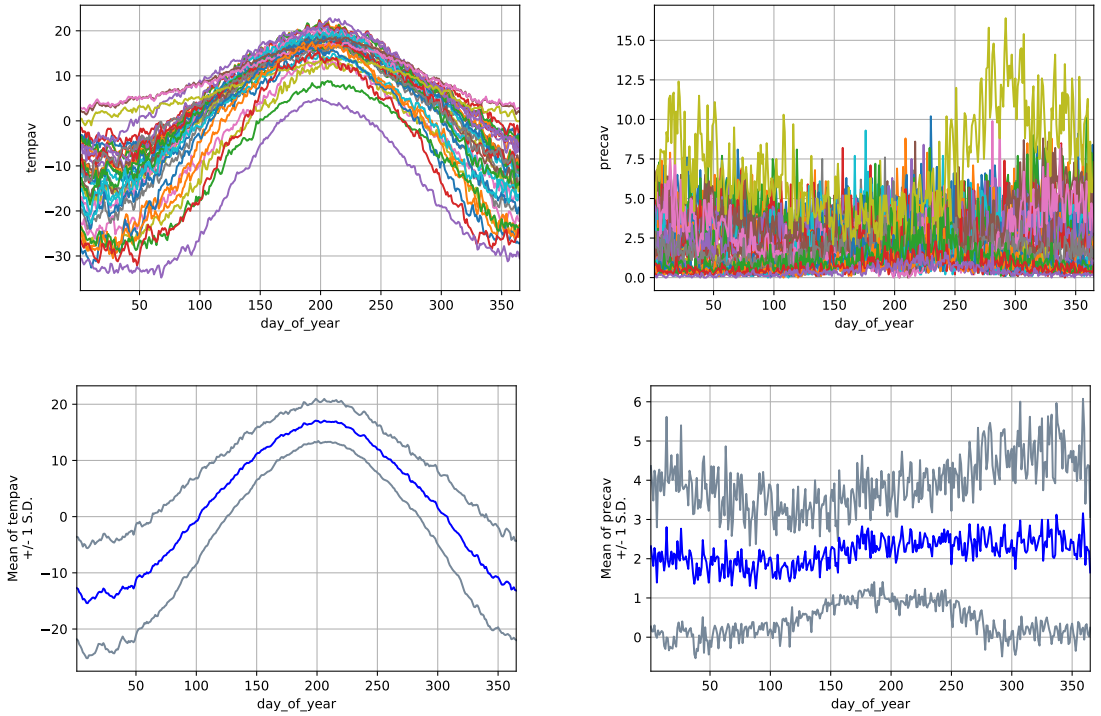

Figure A.1: Results of visualising the weather data using methods on the `MultiSeries` instance. The top 2 images are output by `visualise()` and the bottom 2 by `visualise_moments()`.

Before proceeding to the topics of tuning and generalisation evaluation, we will first show a simple fit-predict-score workflow.

In order to do so, we split the multi-series data into training and validation sets and visualise the results. In this case, it is a randomised 70%/30% split. Note that the splits are by series instance and not time/sequential index. The results of this snippet are displayed in Fig. A.2:

```
from sklearn.model_selection import ShuffleSplit

splits = list(data_weather.generate_series_folds(series_splitter=ShuffleSplit(
    test_size=0.30, n_splits=1)))
(training_set, validation_set) = splits[0]

training_set.visualise(title='Training set: Y_true')
validation_set.visualise(title='Validation set: Y_true')
```

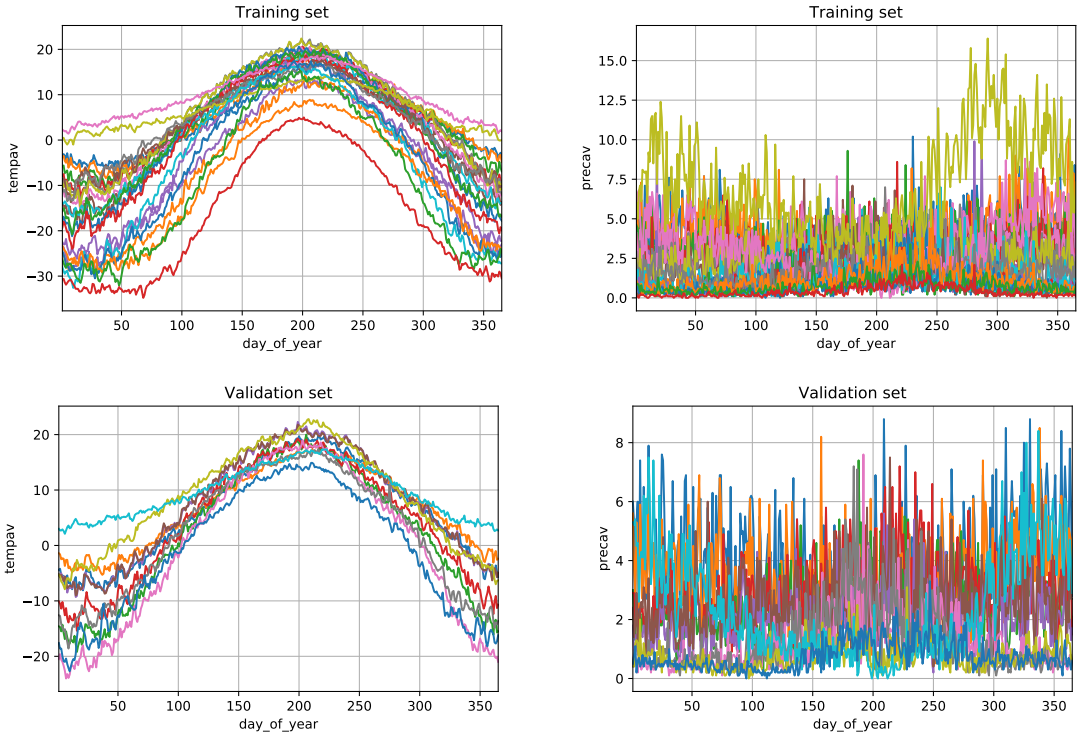

Figure A.2: Results of visualising the weather data using methods on the `MultiSeries` instance. The top 2 images represent the bivariate features of the training set, and the bottom 2 represent the validation set.

We now define our prediction strategy, resulting in a composite pipelined predictor that takes 3 hyper-parameters, each of which we fix for now. Note that there are 2 levels of pipelining in our strategy, all of which can be composed together because our predictors hew to a fit-predict interface:

- The **pysf** pipeline is a smoothing transformation followed by a multi-series tabular predictor, wrapped around an inner **scikit-learn** estimator.
- The inner **scikit-learn** estimator is itself a pipeline of a standardisation transformation, followed by a PCA decomposition and finally a linear regression.

```
from pysf.predictors.framework import PipelinePredictor,
    MultiCurveTabularPredictor
from pysf.transformers.smoothing import SmoothingSplineTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.linear_model import LinearRegression
```

```
7
8  sklearn_estimator_pipeline = Pipeline(steps=[('scale', StandardScaler()), ('pca',
       PCA()), ('ols', LinearRegression())])
9  multiseries_smoothing_pcr_predictor = PipelinePredictor(chain = [
       SmoothingSplineTransformer() , MultiCurveTabularPredictor(classic_estimator=
       sklearn_estimator_pipeline)  ])
10 multiseries_smoothing_pcr_predictor.set_parameters({ 'pca__n_components' : 3 , '
       spline_degree' : 5 , 'smoothing_factor' : 'default' })
```

We now fit our composite prediction strategy on the training set, predict out-of-sample on the validation set, and then calculate prediction residuals $\epsilon_i(\boldsymbol{T}_*)$ and various prediction metrics $\widehat{\varepsilon}(\boldsymbol{T}_*)$, with standard error bars derived from $\widehat{v}(\boldsymbol{T}_*)$. The results of this snippet are displayed in Fig. A.3:

```
1  from pysf.errors import ErrorCurve
2  import numpy as np
3
4  common_prediction_times = np.arange(301, 366)
5  common_input_time_feature = False
6  common_input_non_time_features = ['tempav', 'precav']
7  common_prediction_features = ['tempav', 'precav']
8
9  multiseries_smoothing_pcr_predictor.fit(X=training_set, prediction_times=
       common_prediction_times, input_time_feature=common_input_time_feature,
       input_non_time_features=common_input_non_time_features, prediction_features=
       common_prediction_features)
10
11 Y_hat = multiseries_smoothing_pcr_predictor.predict(X=validation_set,
       prediction_times=common_prediction_times, input_time_feature=
       common_input_time_feature, input_non_time_features=
       common_input_non_time_features, prediction_features=
       common_prediction_features)
12 Y_hat.visualise(title='Validation set: Y_hat')
13
14 Y_true = validation_set.subset_by_times(common_prediction_times)
15 Y_true.visualise(title='Validation set: Y_true')
16
17 for prediction_feature in common_prediction_features:
18         residuals = Y_true.get_raw_residuals(Y_hat=Y_hat,
               value_colnames_vs_times_filter=prediction_feature)
19         residuals.visualise()
20         err = ErrorCurve.init_from_raw_residuals(raw_residuals_obj=residuals)
21         err.visualise_per_timestamp(title=prediction_feature)
```

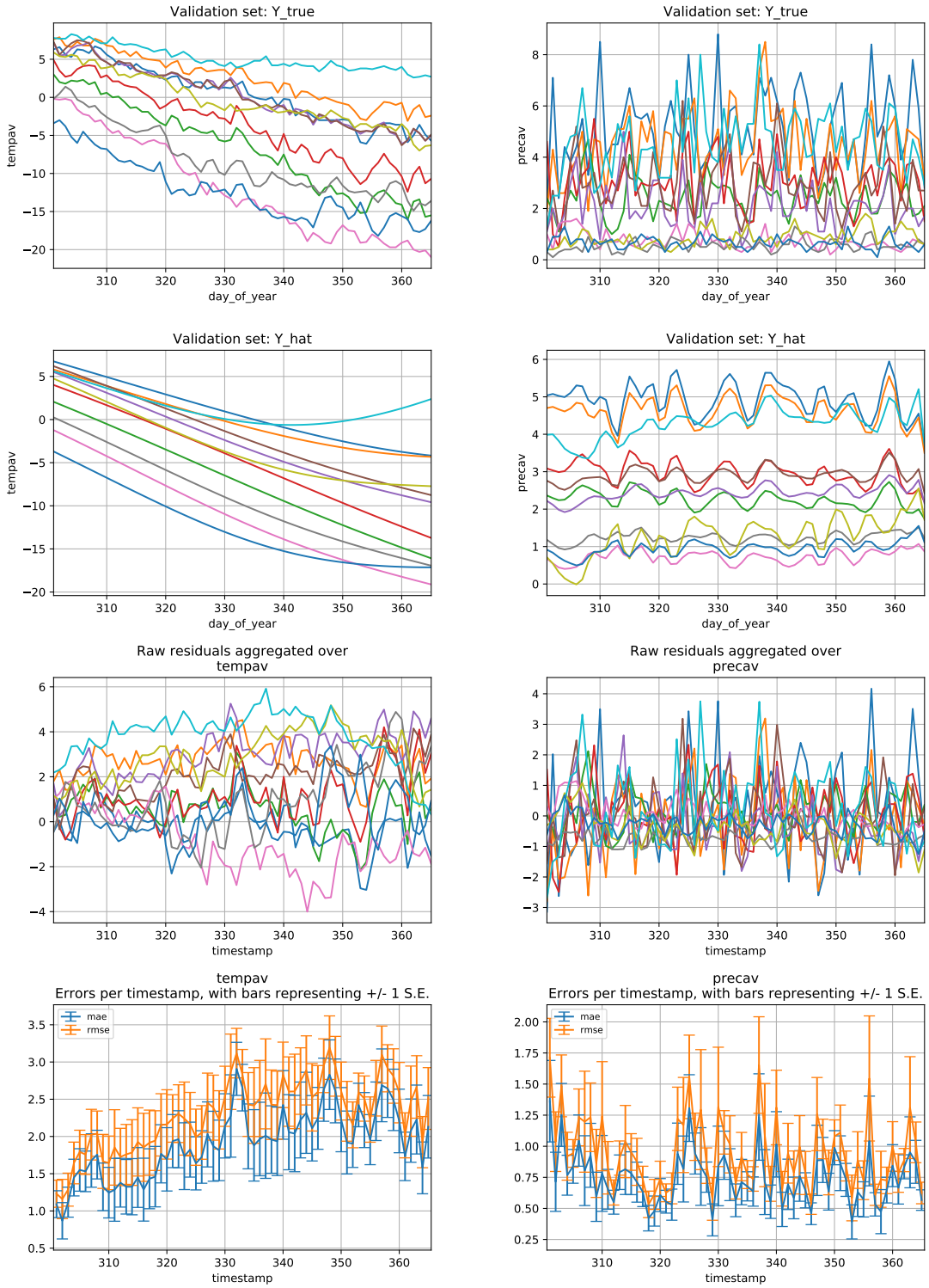

Figure A.3: Results of predicting & scoring on the validation set for the Smoothing PCR predictor with fixed hyperparameters. Each column represents a feature. Row 1 represents actual/true observations $X_{\mathcal{V}}(\boldsymbol{T}_*)$. Row 2 represents predicted values $\widehat{X}_{\mathcal{V}}(\boldsymbol{T}_*)$. Row 3 represents the raw residuals $\epsilon_i(\boldsymbol{T}_*)$. Row 4 represents the per-timestamp errors $\widehat{\varepsilon}(\boldsymbol{T}_*)$ for the RMSE & MAE metrics, with error bars derived from $\widehat{\nu}(\boldsymbol{T}_*)$.

The above predictor had a fixed set of non-optimal hyperparameters. We will now define another predictor that wraps around it and tunes for an (estimated) optimal set of hyperparameters on the training set. According to our definition,

- it will sample 5 parameter sets at random from the parameter space that we have defined,
- the fitting step will perform 5-fold cross-validation (the default), and
- it will tune for the overall RMSE on the average temperature feature (even though we may choose to predict for both features concurrently).

```
from pysf.predictors.tuning import TuningOverallPredictor
from sklearn.model_selection import ParameterSampler

tuning_overall_multiseries_smoothing_pcr_predictor = TuningOverallPredictor(
    predictor_template=multiseries_smoothing_pcr_predictor, scoring_metric='rmse'
    , scoring_feature_name='tempav'
, parameter_iterator=ParameterSampler(n_iter=5, param_distributions={  '
    pca__n_components' : [ 3, 5, 7, 9 ]
, 'spline_degree' : [ 3, 5 ]
, 'smoothing_factor' : [ 'default', '0', '50', '100', '150', '200' ]
}))
```

Now that we have defined our self-tuning predictor, we repeat our experiment on the same 70%/30% training/validation set split as before. We can see that the prediction accuracy has improved, indicating we have reached a local optimum. The results of this snippet are displayed in Fig. A.4:

```
import random
random.seed(777) # for reproducibility

tuning_overall_multiseries_smoothing_pcr_predictor.fit(X=training_set,
    prediction_times=common_prediction_times, input_time_feature=
    common_input_time_feature, input_non_time_features=
    common_input_non_time_features, prediction_features=
    common_prediction_features)

Y_hat = tuning_overall_multiseries_smoothing_pcr_predictor.predict(X=
    validation_set, prediction_times=common_prediction_times, input_time_feature=
    common_input_time_feature, input_non_time_features=
    common_input_non_time_features, prediction_features=
    common_prediction_features)
Y_hat.visualise(title='Validation set: Y_hat')

for prediction_feature in common_prediction_features:
        residuals = Y_true.get_raw_residuals(Y_hat=Y_hat,
            value_colnames_vs_times_filter=prediction_feature)
        residuals.visualise()
        err = ErrorCurve.init_from_raw_residuals(raw_residuals_obj=residuals)
        err.visualise_per_timestamp(title=prediction_feature)
```

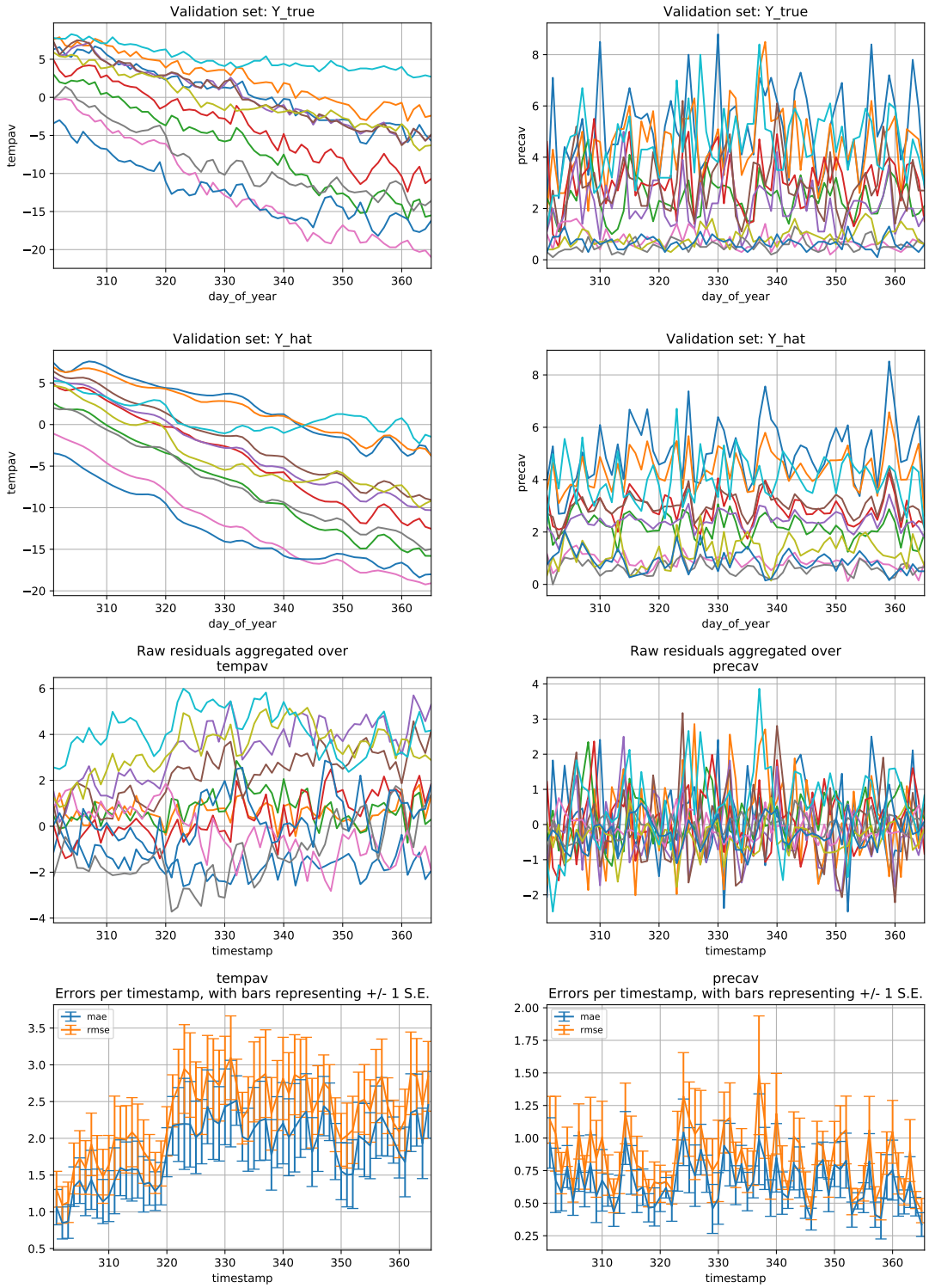

Figure A.4: Results of predicting & scoring on the validation set for the Smoothing PCR predictor wrapped in a tuning predictor, as described in the text. Chart descriptions are identical to those of Fig. A.3 .

Now that we have demonstrated the essential building blocks of our workflow, we will discard the

70%/30% training/validation set split and estimate the generalisation error of our self-tuning prediction strategy using nested cross-validation on the full data set. We will also compare our predictor against a set of baselines. The evaluator object will prepare identical CV folds and then iterate over various targets. Each target is defined by a predictor and the particular input and output fields to use. The results of this snippet are shown in Fig. A.5:

```
from pysf.generalisation import GeneralisationPerformanceEvaluator
from pysf.predictors.baselines import SeriesMeansPredictor, ZeroPredictor,
    TimestampMeansPredictor, SeriesLinearInterpolator

evaluator_weather = GeneralisationPerformanceEvaluator(data=data_weather,
    prediction_times=common_prediction_times)

evaluator_weather.add_to_targets(   combos_of_input_time_column=[ True ],
    combos_of_input_value_colnames=[ None ], combos_of_output_value_colnames=[
    common_prediction_features ]
, predictor_templates={   'Baseline single-series series means' :
    SeriesMeansPredictor()
, 'Baseline 0 values' : ZeroPredictor()
, 'Baseline multi-series timestamp means' : TimestampMeansPredictor()
, 'Baseline single-series series linear interpolator' : SeriesLinearInterpolator
    ()
})

evaluator_weather.add_to_targets(   combos_of_input_time_column=[ False ],
    combos_of_input_value_colnames=[ common_prediction_features ],
    combos_of_output_value_colnames=[ common_prediction_features ]
, predictor_templates={   'Multi-series self-tuning Smoothing PCR' :
    tuning_overall_multiseries_smoothing_pcr_predictor })

random.seed(777) # for reproducibility
results_df = evaluator_weather.evaluate()

evaluator_weather.chart_overall_performance(feature_name='tempav', metric='rmse')
evaluator_weather.chart_per_timestamp_performance(feature_name='tempav', metric='
    rmse')

evaluator_weather.chart_overall_performance(feature_name='precav', metric='rmse')
evaluator_weather.chart_per_timestamp_performance(feature_name='precav', metric='
    rmse')
```

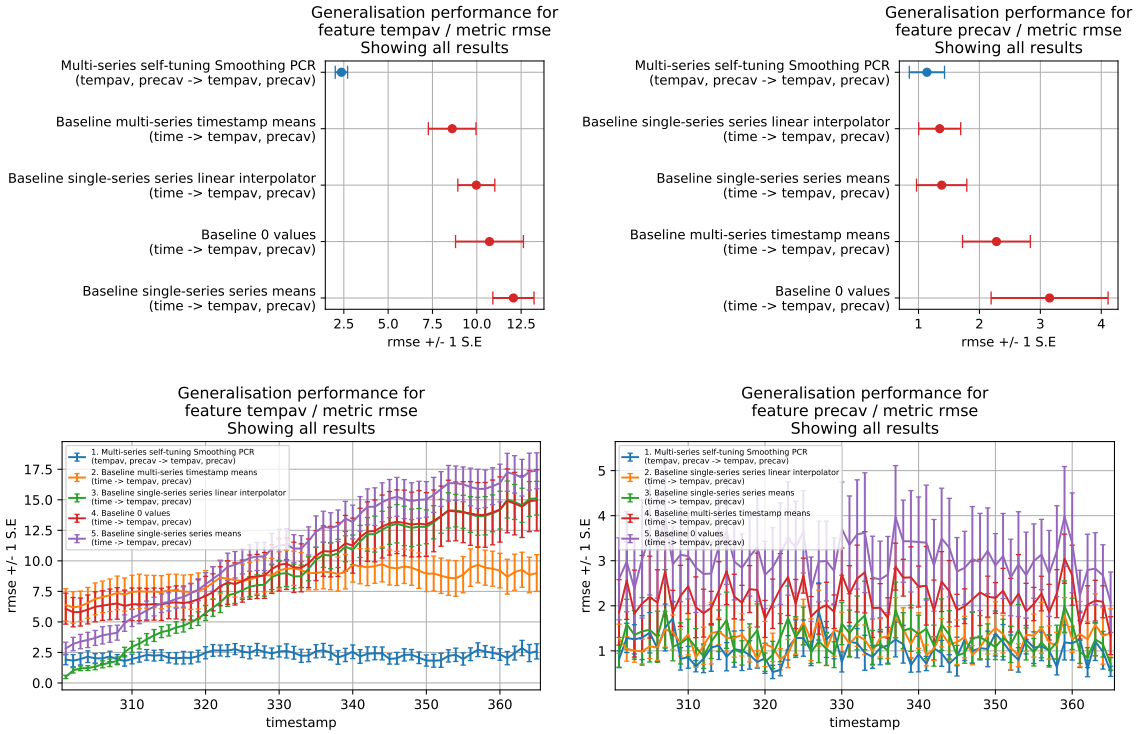

Figure A.5: Results of estimating the generalisation error of the self-tuning Smoothing PCR predictor against a set of baseline predictors, for both features. Each column represents a feature being studied. Row 1 shows cross-validated overall errors $\widehat{\varepsilon}_{\text{CV}}$. Row 2 shows cross-validated per-timestamp errors $\widehat{\varepsilon}_{\text{CV}}(t)$. Each uses the RMSE metric and draws error bars using the respective estimators of variance $\widehat{\nu}_{\text{CV}}$ and $\widehat{\nu}_{\text{CV}}(t)$.

B. Example Gram matrices of series kernels

Figures B.2 & B.3 show plots of Gram matrices for multiple example series kernels, over all series within the Berkeley growth & Canadian weather datasets, respectively. The Gram matrices are generally similar in appearance within each dataset.

Interpreting the series kernel as a measure of similarity, the Gram matrices show that series are clustered into groups of similar series according to the individual series kernel values. There is one highly-dissimilar series in the weather dataset that can be interpreted as an outlier and Figure B.1 highlights that particular series in relation to the others, with the precipitation series in particular being an obvious outlier.

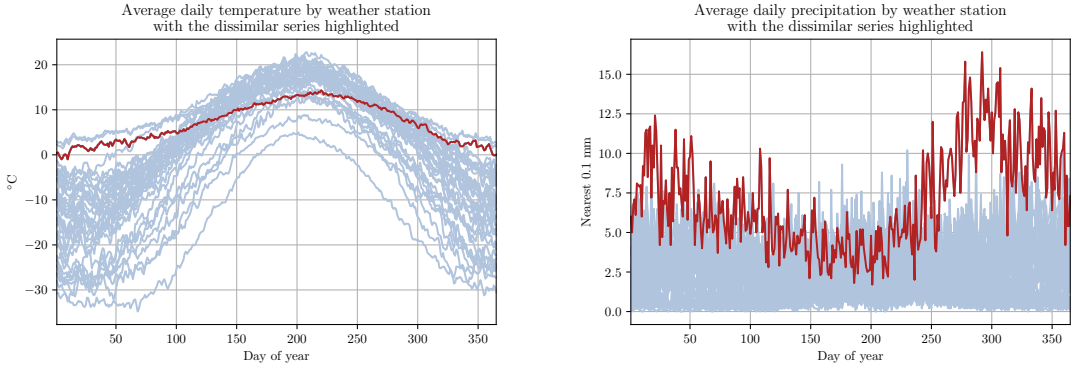

Figure B.1: Series plots of the multivariate Canadian weather data, with the dissimilar series highlighted in each feature plot.

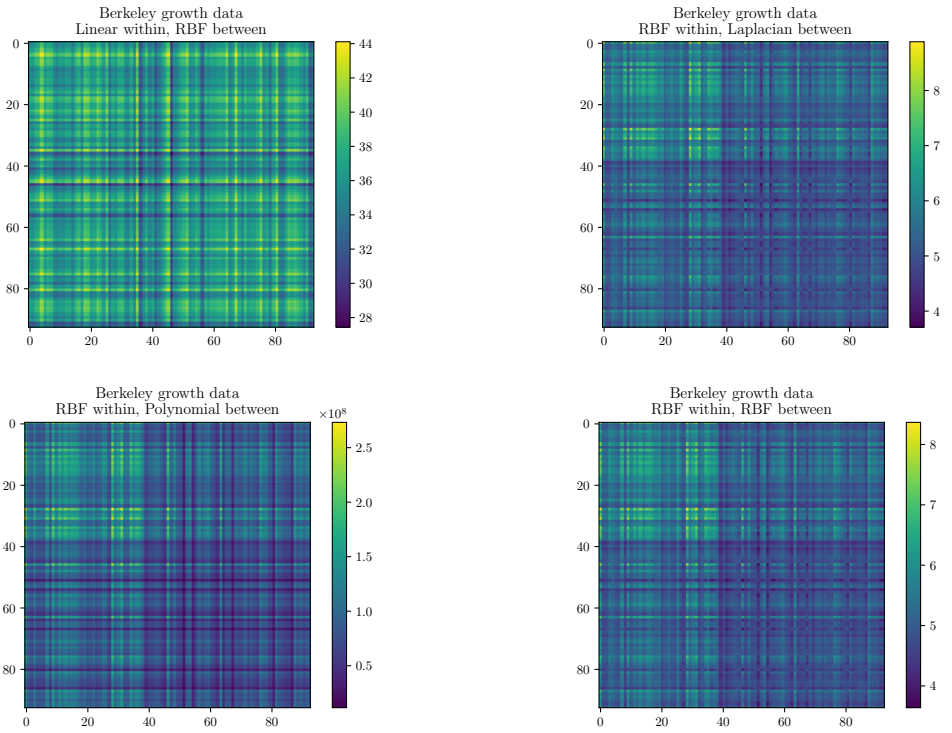

Figure B.2: Example series kernels for the univariate Berkeley growth dataset.

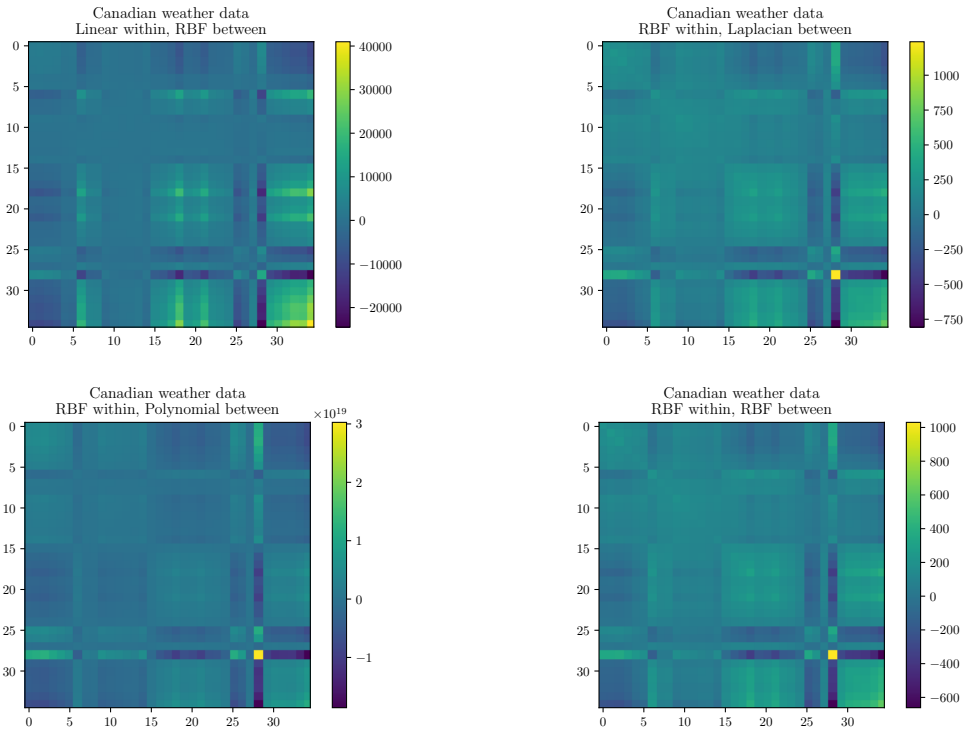

Figure B.3: Example series kernels encoding both the temperature & precipitation features for the Canadian weather dataset.

C. Prediction error metrics

In §5 we derived a number of estimators for the generalisation error (and estimators for their variances) without assuming a particular form of loss function L. This allows us leeway in our choice of loss function L. In this section, we provide 3 such choices of L that we have found informative when evaluating supervised forecasting prediction strategies. For each, we work backwards from a particular well-known *error metric*.

We found metrics on the same scale as the input data (RMSE & MAE) to be particularly useful. No matter the choice of metric, though, it is important to report the variance of the estimates in order to quantify the degree of uncertainty in obtaining them. At the very least, these standard errors (or approximations thereof) can be used to draw error bars on charts. We have also used them in significance testing.

C.1. Mean Squared Error

Definition C.1. *The Mean Squared Error (MSE) of predictions at a time point t_* is the mean of loss observations*

$$MSE(t_*) = \frac{1}{N}\sum_{i=1}^{N}[L_i(t_*)]^2 \tag{C.1}$$

for the squared error loss function

$$L\big(\widehat{X}_*(t_*), X_*(t_*)\big) = \Big[\widehat{X}_*(t_*) - X_*(t_*)\Big]^2. \tag{C.2}$$

Since the MSE is a sample mean, its variance is the standard error of the loss values.

C.2. Mean Absolute Error

Definition C.2. *The Mean Absolute Error (MAE) of predictions at a time point t_* is the mean of loss observations*

$$MAE(t_*) = \frac{1}{N}\sum_{i=1}^{N}[L_i(t_*)]^2 \tag{C.3}$$

for the absolute error loss function

$$L\big(\widehat{X}_*(t_*), X_*(t_*)\big) = \Big|\widehat{X}_*(t_*) - X_*(t_*)\Big|. \tag{C.4}$$

Since the MAE is a sample mean, its variance is the standard error of the loss values.

C.3. Root Mean Squared Error

Definition C.3. *The Root Mean Squared Error (RMSE) of predictions at a time point t_* is simply the square root of the MSE:*

$$RMSE(t_*) = \sqrt{MSE(t_*)} \tag{C.5}$$

We can see that, although the RMSE is on the same scale as the MAE, it is more affected by the presence of outliers as it is a function of the MSE.

However – because it is a function – calculating its variance and standard error is no longer straightforward. We could use the Delta Method, the jackknife or the bootstrap to approximate it, and choose the Delta Method because it results in a closed-form expression.

Theorem 2. *Delta Method: Let Y_n be a sequence of random variables that satisfies*

$$\sqrt{n}(Y_n - \theta) \xrightarrow{D} \mathcal{N}(0, \sigma^2). \tag{C.6}$$

For a given function g and a specific value of θ, if the first derivative $g'(\theta)$ exists and is not 0, then

$$\sqrt{n}\big(g(Y_n) - g(\theta)\big) \xrightarrow{D} \mathcal{N}\big(0, \sigma^2[g'(\theta)]^2\big) \tag{C.7}$$

Proof. See Casella and Berger [10] pp. 243. □

Proposition C.4. *The standard error (s.e.) of the RMSE can be approximated as*

$$s.e.\big[RMSE(t_*)\big] \approx \frac{s.e.\big[MSE(t_*)\big]}{2\,RMSE(t_*)} \tag{C.8}$$

Proof. Applying Theorem 2 to our situation, we can approximate the true/population variance as follows:

$$\mathrm{Var}[g(\theta)] \approx [g'(\theta)]^2\ \mathrm{Var}[\theta] \tag{C.9}$$

where $g(\theta) = \sqrt{\theta} \Rightarrow g'(\theta) = 1/2\sqrt{\theta}$ in our situation.

Applying the above to the population variances and thence to the sample variance,

$$\mathrm{Var}\big[\mathrm{RMSE}(t_*)\big] = \mathrm{Var}\big[\sqrt{\mathrm{MSE}(t_*)}\big] \tag{C.10}$$

$$\Rightarrow \quad \mathrm{s.e.}\big[\mathrm{RMSE}(t_*)\big] \approx \frac{1}{2\sqrt{\mathrm{MSE}(t_*)}}\,\mathrm{s.e.}\big[\mathrm{MSE}(t_*)\big] \tag{C.11}$$

$$= \frac{\mathrm{s.e.}\big[\mathrm{MSE}(t_*)\big]}{2\,\mathrm{RMSE}(t_*)} \tag{C.12}$$

□

D. Individual experimental results

This appendix presents the results of §7 in the form of charts, broken down by individual dataset. The charts show either *overall* estimated generalisation errors $\left(\widehat{\varepsilon}_{CV} \pm \widehat{v}_{CV}\right)$ or *per-timestamp* ones $\left(\widehat{\varepsilon}(t)_{CV} \pm \widehat{v}(t)_{CV}\right)$.

D.1. On Berkeley growth data

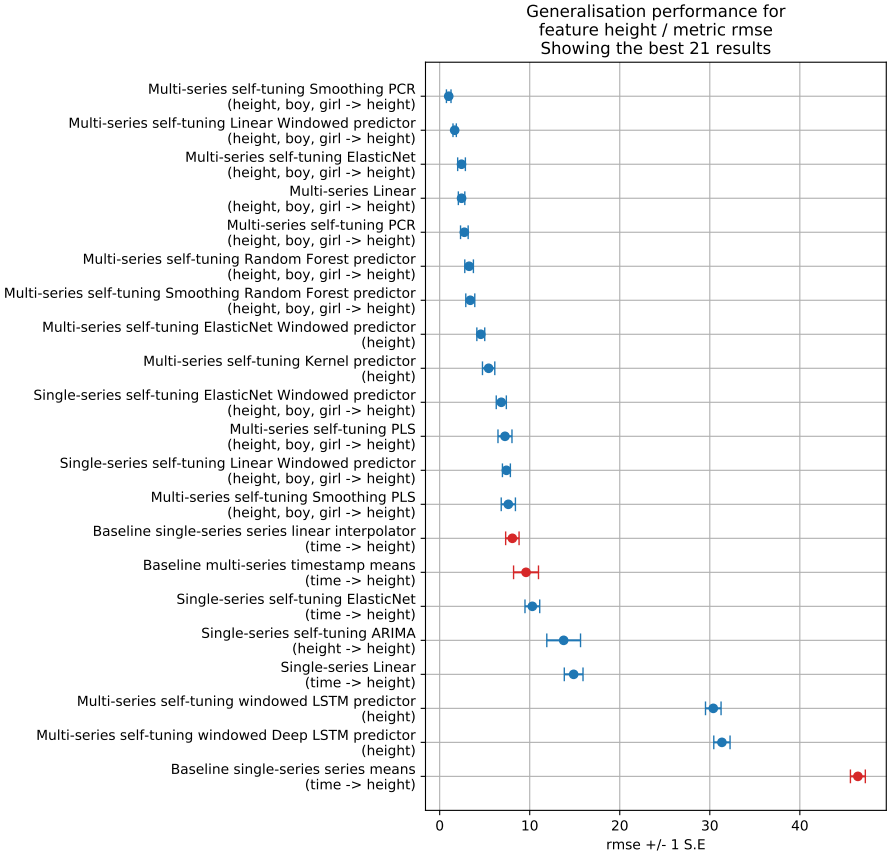

Figure D.1: Overall estimated generalisation errors for various prediction strategies on the Berkeley growth dataset.

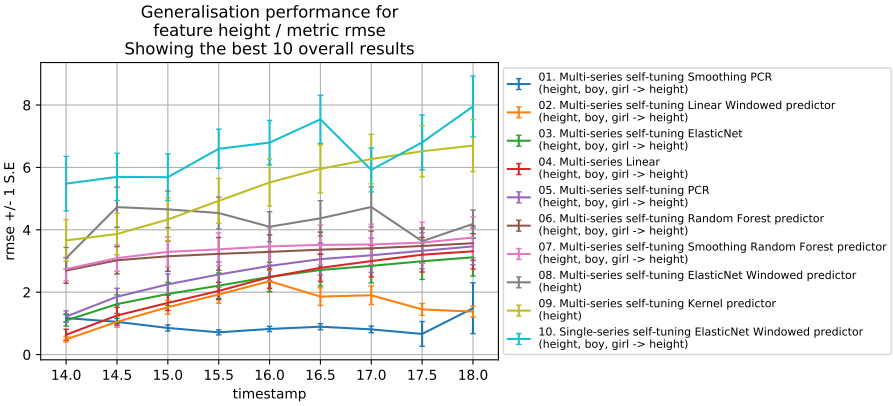

Figure D.2: Per-timestamp estimated generalisation errors for various prediction strategies on the Berkeley growth dataset.

D.2. On Canadian weather data

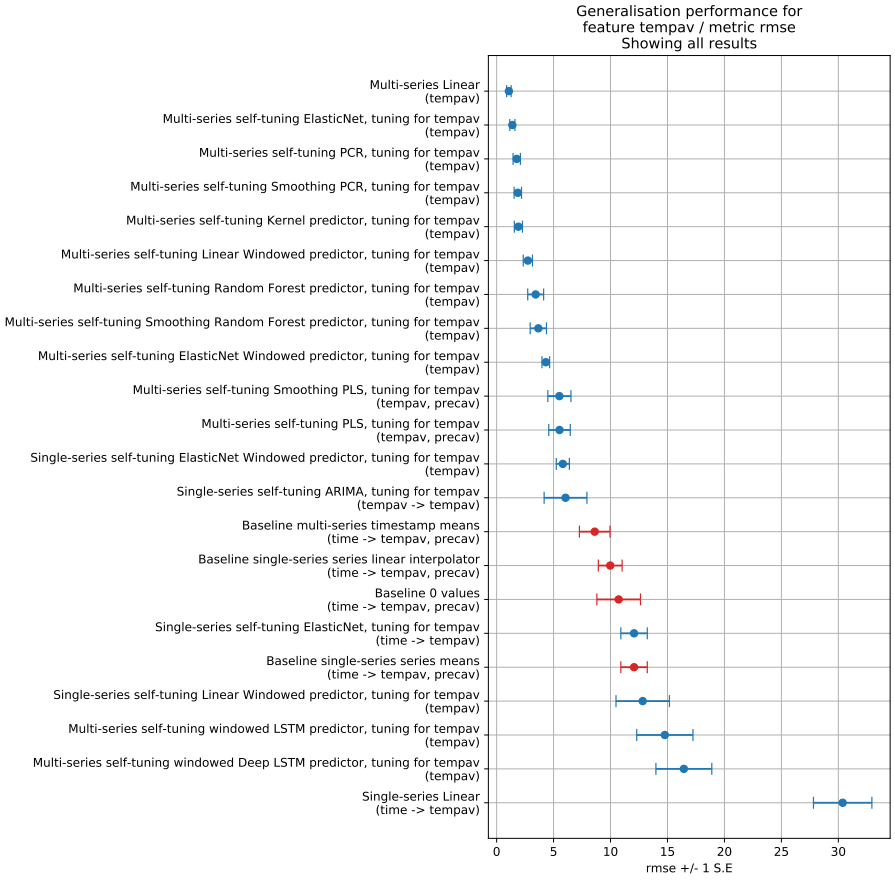

Figure D.3: Overall estimated generalisation errors for various prediction strategies on the Canadian weather dataset, specifically for the average temperature feature.

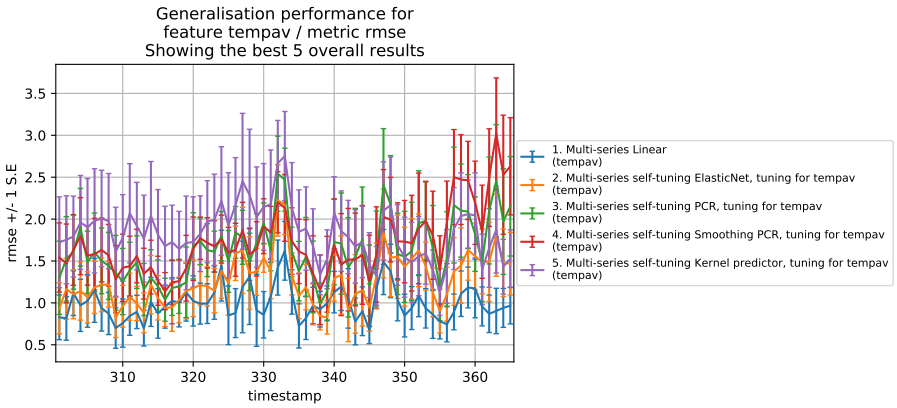

Figure D.4: Per-timestamp estimated generalisation errors for various prediction strategies on the Canadian weather dataset, specifically for the average temperature feature.

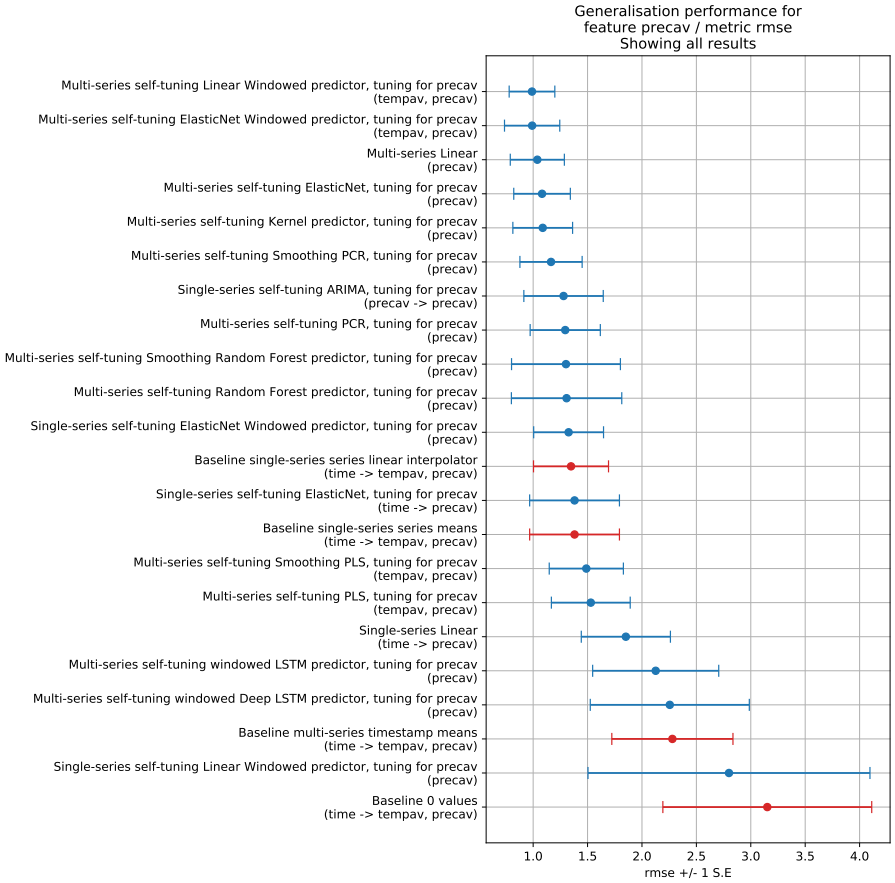

Figure D.5: Overall estimated generalisation errors for various prediction strategies on the Canadian weather dataset, specifically for the average precipitation feature.

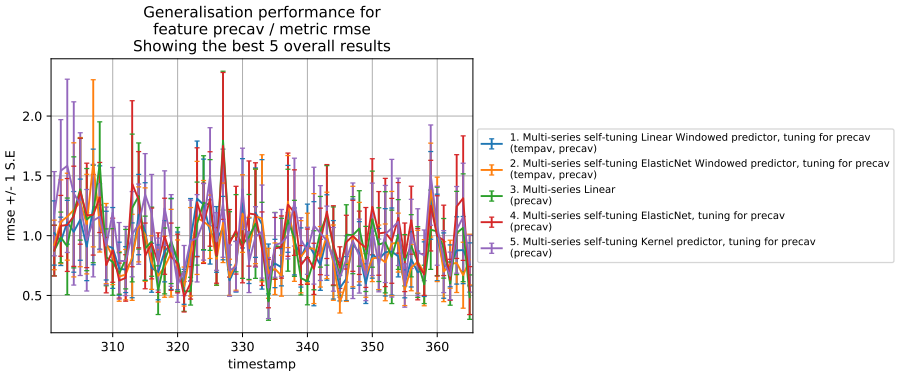

Figure D.6: Per-timestamp estimated generalisation errors for various prediction strategies on the Canadian weather dataset, specifically for the average precipitation feature.

D.3. On ECG data

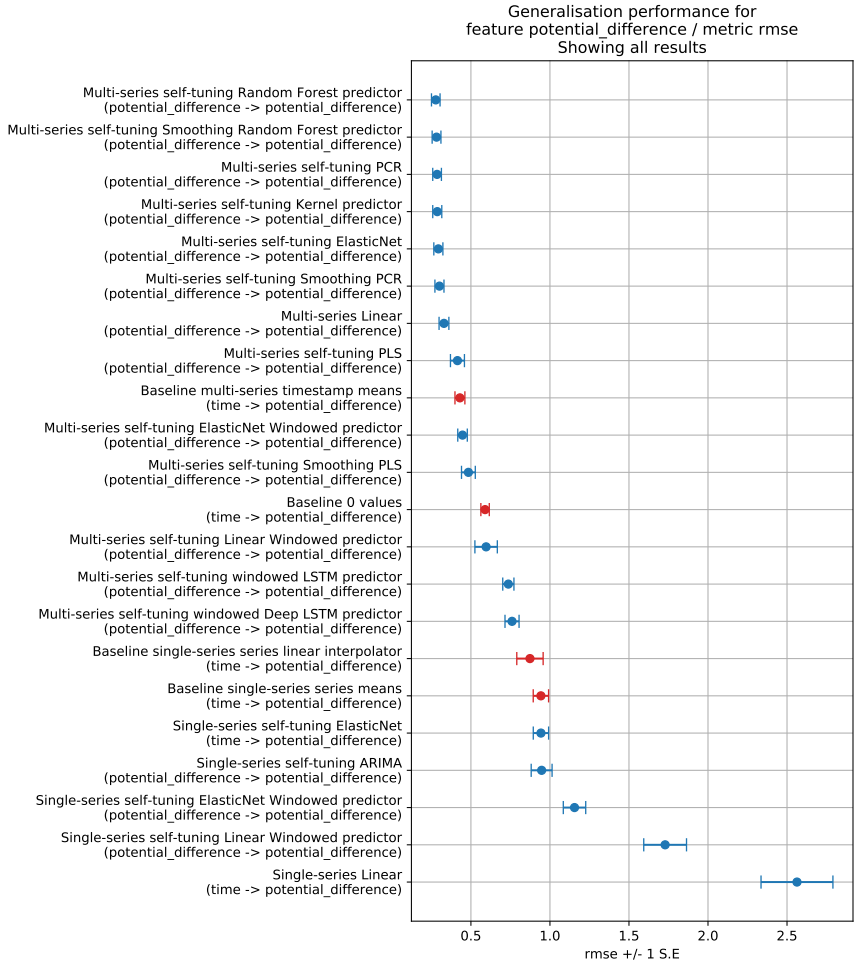

Figure D.7: Overall estimated generalisation errors for various prediction strategies on the ECG dataset.

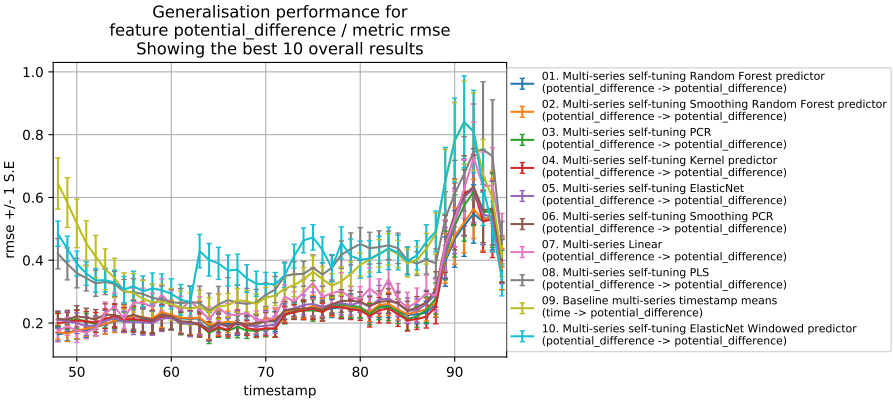

Figure D.8: Per-timestamp estimated generalisation errors for various prediction strategies on the ECG dataset.

D.4. On Power data: multiple days for a single site

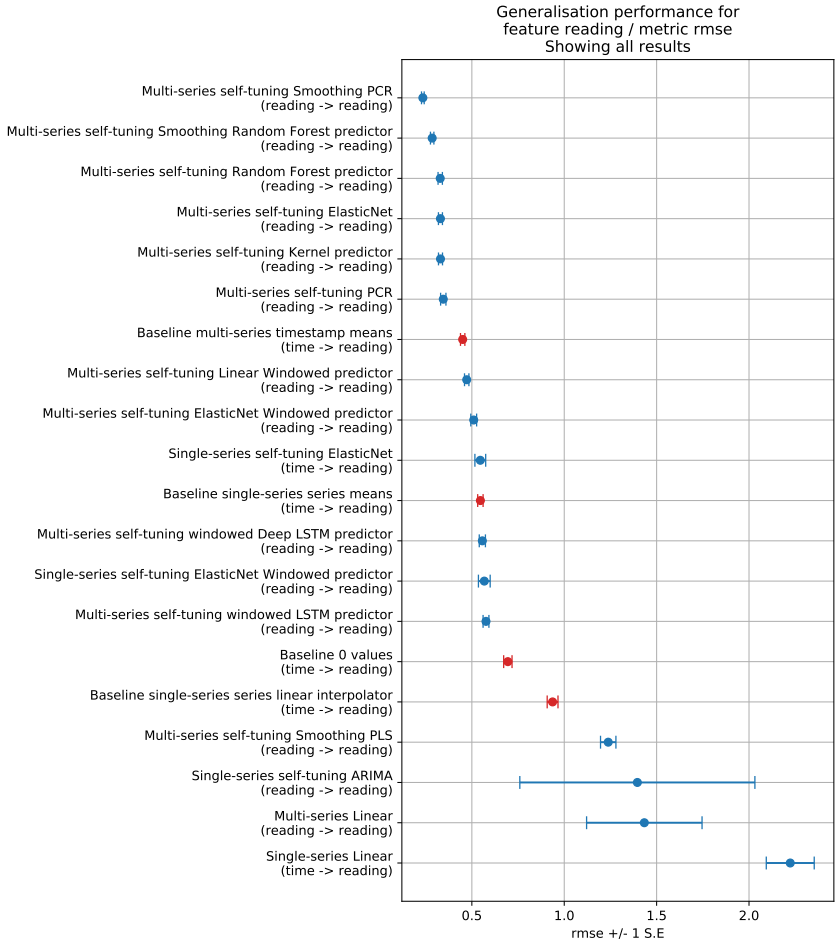

Figure D.9: Overall estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple days for a single site.

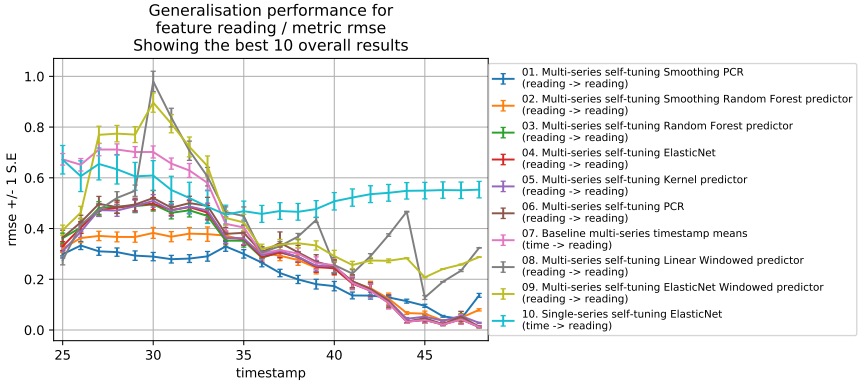

Figure D.10: Per-timestamp estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple days for a single site.

D.5. On Power data: multiple sites & multiple days

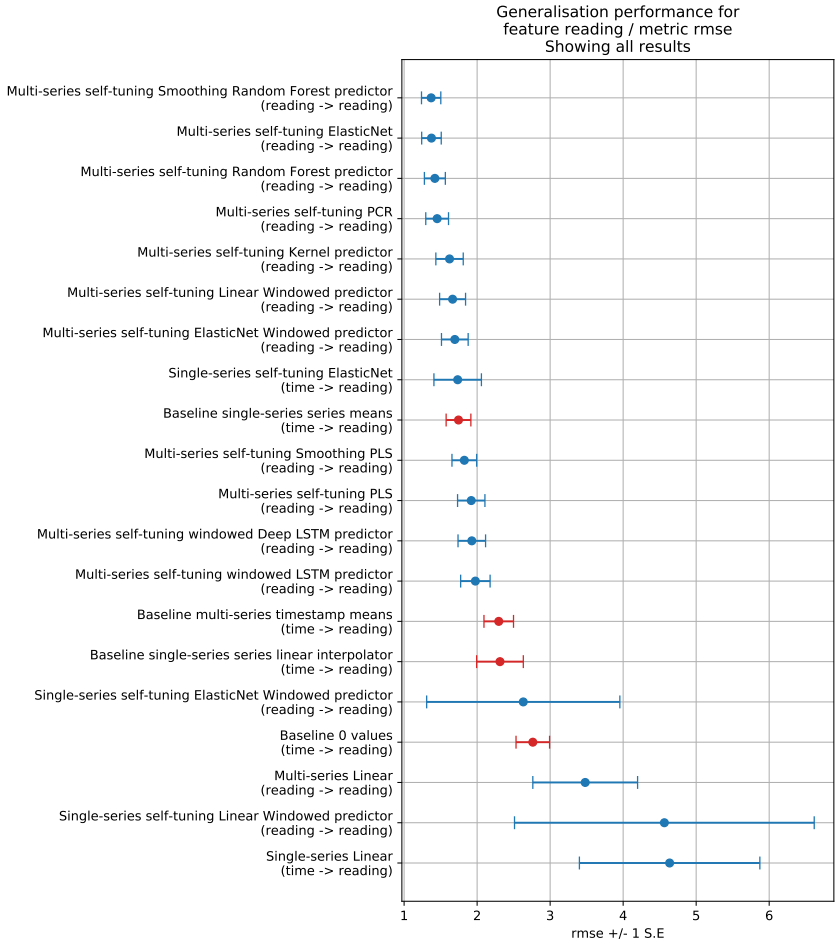

Figure D.11: Overall estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple sites & multiple days.

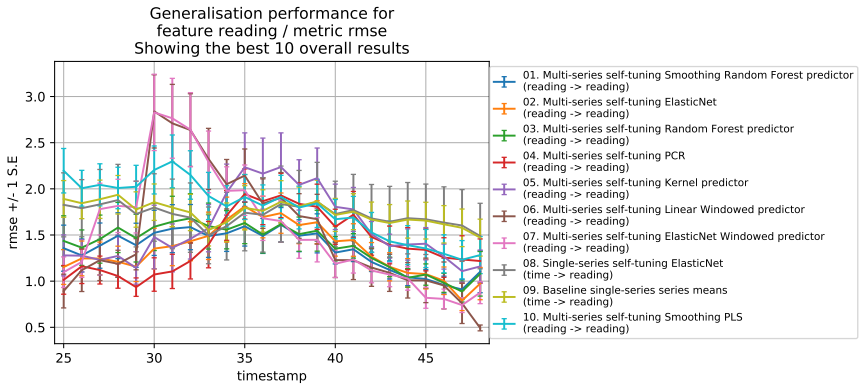

Figure D.12: Per-timestamp estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple sites & multiple days.

D.6. On Power data: multiple sites for a single day

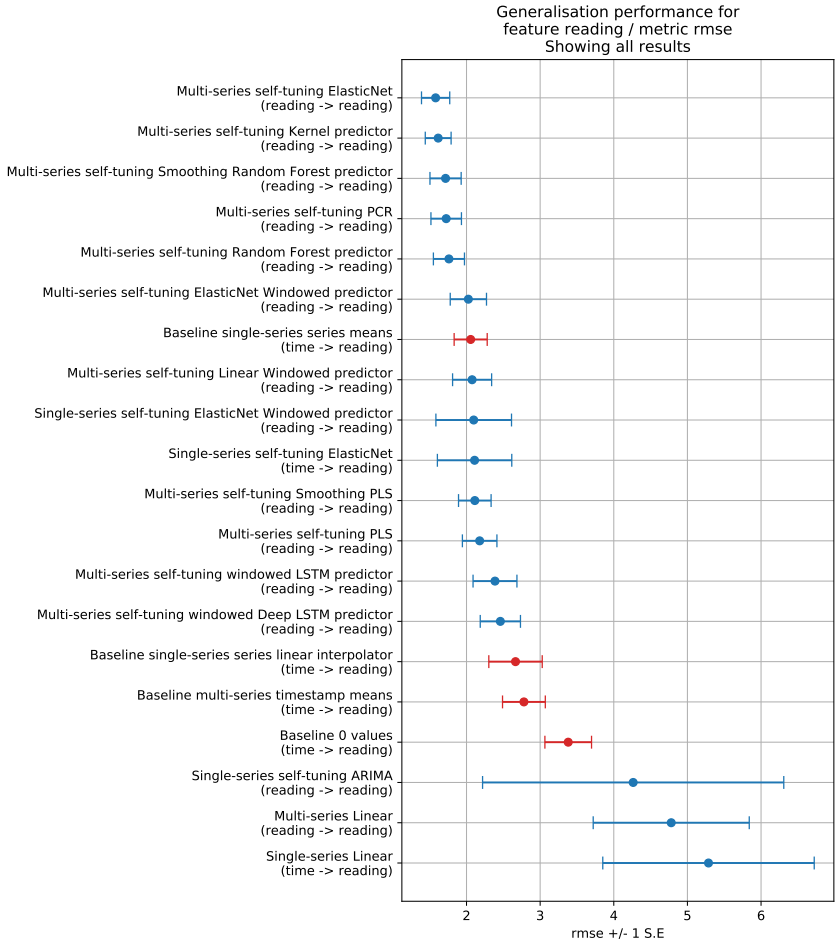

Figure D.13: Overall estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple sites for a single day.

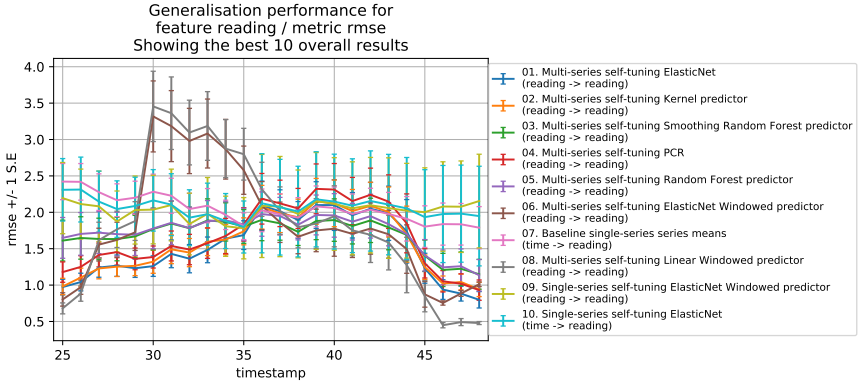

Figure D.14: Per-timestamp estimated generalisation errors for various prediction strategies on the Power dataset that combines multiple sites for a single day.

D.7. On Starlight data

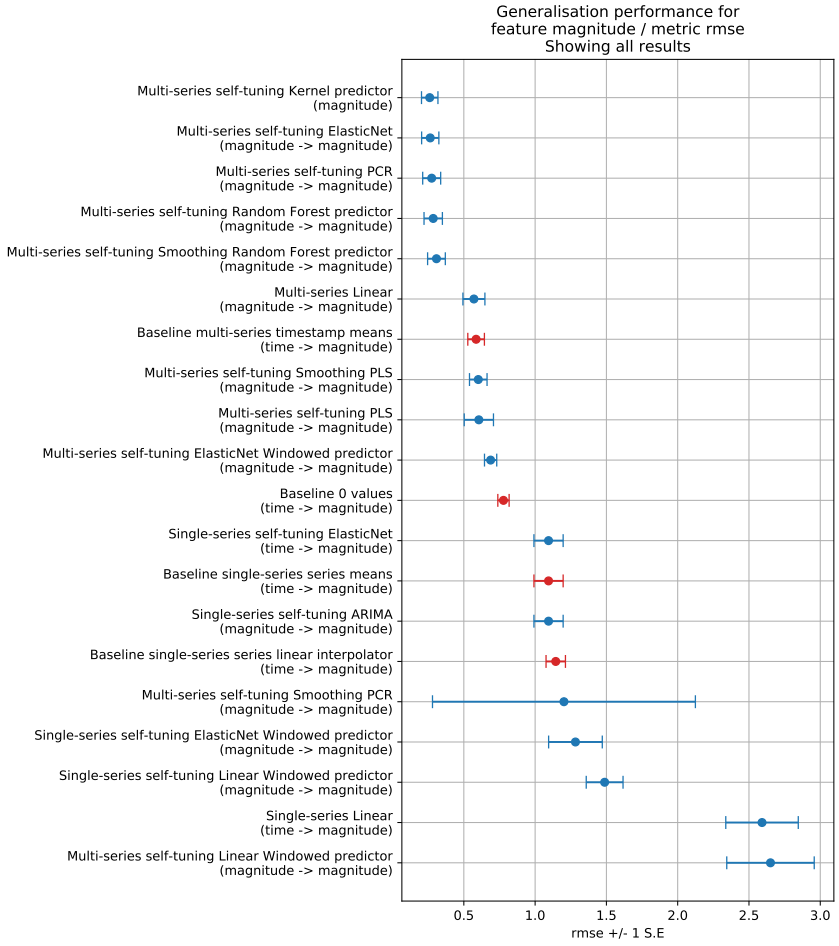

Figure D.15: Overall estimated generalisation errors for various prediction strategies on the Starlight dataset.

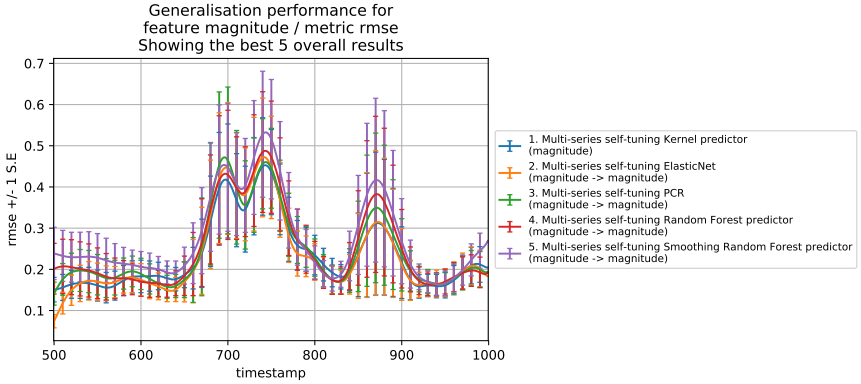

Figure D.16: Per-timestamp estimated generalisation errors for various prediction strategies on the Starlight dataset. Error bars are sampled for legibility.