Part I

Introduction

Welcome

Welcome to *Deep Learning for Time Series Forecasting*. Deep learning methods, such as Multilayer Perceptrons, Convolutional Neural Networks, and Long Short-Term Memory Networks, can be used to automatically learn the temporal dependence structures for challenging time series forecasting problems. Neural networks may not be the best solution for all time series forecasting problems, but for those problems where classical methods fail and machine learning methods require elaborate feature engineering, deep learning methods can be used with great success. This book was designed to teach you, step-by-step, how to develop deep learning methods for time series forecasting with concrete and executable examples in Python.

Who Is This Book For?

Before we get started, let's make sure you are in the right place. This book is for developers that may know some applied machine learning. Maybe you know how to work through a predictive modeling problem end-to-end, or at least most of the main steps, with popular tools. The lessons in this book do assume a few things about you, such as:

- You know your way around basic Python for programming.
- You know some basic time series forecasting with classical methods such as naive forecasts, ARIMA, or exponential smoothing.
- You know how to work through a predictive modeling problem using machine learning or deep learning methods and libraries such as scikit-learn or Keras.
- You have some background knowledge on how neural network models work, perhaps at a high level.
- You want to learn how to develop a suite of deep learning methods to get more from new or existing time series forecasting problems.

This guide was written in the top-down and results-first machine learning style that you're used to from Machine Learning Mastery.

About Your Outcomes

This book will teach you the practical deep learning skills that you need to know for time series forecasting. After reading and working through this book, you will know:

- About the promise of neural networks and deep learning methods in general for time series forecasting.
- How to transform time series data in order to train a supervised learning algorithm, such as deep learning methods.
- How to develop baseline forecasts using naive and classical methods by which to determine whether forecasts from deep learning models have skill or not.
- How to develop Multilayer Perceptron, Convolutional Neural Network, Long Short-Term Memory Networks, and hybrid neural network models for time series forecasting.
- How to forecast univariate, multivariate, multi-step, and multivariate multi-step time series forecasting problems in general.
- How to transform sequence data into a three-dimensional structure in order to train convolutional and LSTM neural network models.
- How to grid search deep learning model hyperparameters to ensure that you are getting good performance from a given model.
- How to prepare data and develop deep learning models for forecasting a range of univariate time series problems with different temporal structures.
- How to prepare data and develop deep learning models for multi-step forecasting a real-world household electricity consumption dataset.
- How to prepare data and develop deep learning models for a real-world human activity recognition project.

This new understanding of applied deep learning methods will impact your practice of working through time series forecasting problems in the following ways:

- Confidently use naive and classical methods like SARIMA and ETS to quickly develop robust baseline models for a range of different time series forecasting problems, the performance of which can be used to challenge whether more elaborate machine learning and deep learning models are adding value.
- Transform native time series forecasting data into a form for fitting supervised learning algorithms and confidently tune the amount of lag observations and framing of the prediction problem.
- Develop MLP, CNN, RNN, and hybrid deep learning models quickly for a range of different time series forecasting problems, and confidently evaluate and interpret their performance.

This book is not a substitute for an undergraduate course in deep learning or time series forecasting, nor is it a textbook for such courses, although it could be a useful complement. For a good list of top courses, textbooks, and other resources on statistics, see the Further Reading section at the end of each tutorial.

How to Read This Book

This book was written to be read linearly, from start to finish. That being said, if you know the basics and need help with a specific method or type of problem, then you can flip straight to that section and get started. This book was designed for you to read on your workstation, on the screen, not on a tablet or eReader. My hope is that you have the book open right next to your editor and run the examples as you read about them.

This book is not intended to be read passively or be placed in a folder as a reference text. It is a playbook, a workbook, and a guidebook intended for you to learn by doing and then apply your new understanding with working Python examples. To get the most out of the book, I would recommend playing with the examples in each tutorial. Extend them, break them, then fix them. Try some of the extensions presented at the end of each lesson and let me know how you do.

About the Book Structure

This book was designed around major deep learning techniques that are directly relevant to time series forecasting. There are a lot of things you could learn about deep learning and time series forecasting, from theory to abstract concepts to APIs. My goal is to take you straight to developing an intuition for the elements you must understand with laser-focused tutorials. I designed the tutorials to focus on how to get results with deep learning methods. The tutorials give you the tools to both rapidly understand and apply each technique or operation. There is a mixture of both tutorial lessons and projects to both introduce the methods and give plenty examples and opportunity to practice using them.

Each of the tutorials are designed to take you about one hour to read through and complete, excluding the extensions and further reading. You can choose to work through the lessons one per day, one per week, or at your own pace. I think momentum is critically important, and this book is intended to be read and used, not to sit idle. I would recommend picking a schedule and sticking to it. The tutorials are divided into five parts; they are:

- **Part 1: Foundations**. Provides a gentle introduction to the promise of deep learning methods for time series forecasting, a taxonomy of the types of time series forecasting problems, how to prepare time series data for supervised learning, and a high-level procedure for getting the best performing model on time series forecasting problems in general.

- **Part 2: Deep Learning Modeling**. Provides a step-by-step introduction to deep learning methods applied to different types of time series forecasting problems with additional tutorials to better understand the 3D-structure required for some models.

- **Part 3: Univariate Forecasting**. Provides a methodical approach to univariate time series forecasting with a focus on naive and classical methods that are generally known to out-perform deep learning methods and how to grid search deep learning model hyperparameters.

- **Part 4: Multi-step Forecasting**. Provides a step-by-step series of tutorials for working through a challenging multi-step time series forecasting problem for predicting household electricity consumption using classical and deep learning methods.

- **Part 5: Time Series Classification**. Provides a step-by-step series of tutorials for working through a challenging time series classification problem for predicting human activity from accelerometer data using deep learning methods.

Each part targets a specific learning outcome, and so does each tutorial within each part. This acts as a filter to ensure you are only focused on the things you need to know to get to a specific result and do not get bogged down in the math or near-infinite number of digressions. The tutorials were not designed to teach you everything there is to know about each of the methods. They were designed to give you an understanding of how they work, how to use them, and how to interpret the results the fastest way I know how: to learn by doing.

About Python Code Examples

The code examples were carefully designed to demonstrate the purpose of a given lesson. Code examples are complete and standalone. The code for each lesson will run as-is with no code from prior lessons or third-parties required beyond the installation of the required packages. A complete working example is presented with each tutorial for you to inspect and copy-paste. All source code is also provided with the book and I would recommend running the provided files whenever possible to avoid any copy-paste issues.

The provided code was developed in a text editor and intended to be run on the command line. No special IDE or notebooks are required. If you are using a more advanced development environment and are having trouble, try running the example from the command line instead. All code examples were tested on a POSIX-compatible machine with Python 3.

About Further Reading

Each lesson includes a list of further reading resources. This may include:

- Books and book chapters.
- API documentation.
- Articles and Webpages.

Wherever possible, I try to list and link to the relevant API documentation for key functions used in each lesson so you can learn more about them. I have tried to link to books on Amazon so that you can learn more about them. I don't know everything, and if you discover a good resource related to a given lesson, please let me know so I can update the book.

About Getting Help

You might need help along the way. Don't worry; you are not alone.

- **Help with a Technique?** If you need help with the technical aspects of a specific operation or technique, see the *Further Reading* section at the end of each tutorial.

- **Help with APIs?** If you need help with using the Keras library, see the list of resources in the *Further Reading* section at the end of each lesson, and also see *Appendix A*.
- **Help with your workstation?** If you need help setting up your environment, I would recommend using Anaconda and following my tutorial in *Appendix B*.
- **Help with practice problems?** If you are looking for test sets on which to practice developing deep learning models, you can see a prepared list of standard time series forecasting problems in *Appendix A*.
- Help in general? You can shoot me an email. My details are in *Appendix A*.

Summary

Are you ready? Let's dive in! Next up you will discover a gentle introduction to the promise of deep learning methods for time series forecasting.

Part II

Foundations

Overview

This part motivates the use of deep learning for time series forecasting and provides frameworks to allow you to work through a new forecasting problems systematically and ensure that your final model is robust and defensible. After reading the chapters in this part, you will know:

- The promise that the capabilities of deep learning methods offer for addressing challenging time series forecasting problems (Chapter 1).
- How to systematically identify the properties of a time series forecasting problem using a taxonomy and framework of questions (Chapter 2).
- How to systematically work through a new time series forecasting problem to ensure that you are getting the most out of naive, classical, machine learning and deep learning forecasting methods (Chapter 3).
- How to generally transform a sequence of observations in a time series into samples with input and output elements suitable for training a supervised learning algorithm (Chapter 4).
- How naive and the top performing classical methods such as SARIMA and ETS work and how to use them prior to exploring more advanced forecasting methods (Chapter 5).

Chapter 1

Promise of Deep Learning for Time Series Forecasting

Deep learning neural networks are able to automatically learn arbitrary complex mappings from inputs to outputs and support multiple inputs and outputs. These are powerful features that offer a lot of promise for time series forecasting, particularly on problems with complex-nonlinear dependencies, multivalent inputs, and multi-step forecasting. These features along with the capabilities of more modern neural networks may offer great promise such as the automatic feature learning provided by convolutional neural networks and the native support for sequence data in recurrent neural networks. In this tutorial, you will discover the promised capabilities of deep learning neural networks for time series forecasting. After reading this tutorial, you will know:

- The focus and implicit, if not explicit, limitations on classical time series forecasting methods.
- The general capabilities of Multilayer Perceptrons and how they may be harnessed for time series forecasting.
- The added capabilities of feature learning and native support for sequences provided by Convolutional Neural Networks and Recurrent Neural Networks.

Let's get started.

1.1 Time Series Forecasting

Time series forecasting is difficult. Unlike the simpler problems of classification and regression, time series problems add the complexity of order or temporal dependence between observations. This can be difficult as specialized handling of the data is required when fitting and evaluating models. This temporal structure can also aid in modeling, providing additional structure like trends and seasonality that can be leveraged to improve model skill. Traditionally, time series forecasting has been dominated by linear methods like ARIMA because they are well understood and effective on many problems. But these classical methods also suffer from some limitations, such as:

- **Focus on complete data**: missing or corrupt data is generally unsupported.

- **Focus on linear relationships**: assuming a linear relationship excludes more complex joint distributions.
- **Focus on fixed temporal dependence**: the relationship between observations at different times, and in turn the number of lag observations provided as input, must be diagnosed and specified.
- **Focus on univariate data**: many real-world problems have multiple input variables.
- **Focus on one-step forecasts**: many real-world problems require forecasts with a long time horizon.

Machine learning methods can be effective on more complex time series forecasting problems with multiple input variables, complex nonlinear relationships, and missing data. In order to perform well, these methods often require hand-engineered features prepared by either domain experts or practitioners with a background in signal processing.

> Existing techniques often depended on hand-crafted features that were expensive to create and required expert knowledge of the field.
>
> — *Deep Learning for Time-Series Analysis*, 2017.

1.2 Multilayer Perceptrons for Time Series

Simpler neural networks such as the Multilayer Perceptron or MLP approximate a mapping function from input variables to output variables. This general capability is valuable for time series for a number of reasons.

- **Robust to Noise**. Neural networks are robust to noise in input data and in the mapping function and can even support learning and prediction in the presence of missing values.
- **Nonlinear**. Neural networks do not make strong assumptions about the mapping function and readily learn linear and nonlinear relationships.

> ... one important contribution of neural networks - namely their elegant ability to approximate arbitrary nonlinear functions. This property is of high value in time series processing and promises more powerful applications, especially in the subfield of forecasting ...
>
> — *Neural Networks for Time Series Processing*, 1996.

More specifically, neural networks can be configured to support an arbitrary defined but fixed number of inputs and outputs in the mapping function. This means that neural networks can directly support:

- **Multivariate Inputs**. An arbitrary number of input features can be specified, providing direct support for multivariate forecasting.

- **Multi-step Forecasts**. An arbitrary number of output values can be specified, providing direct support for multi-step and even multivariate forecasting.

For these capabilities alone, feedforward neural networks may be useful for time series forecasting. Implicit in the usage of neural networks is the requirement that there is indeed a meaningful mapping from inputs to outputs to learn. Modeling a mapping of a random walk will perform no better than a persistence model (e.g. using the last seen observation as the forecast). This expectation of a learnable mapping function also makes one of the limitations clear: the mapping function is fixed or static.

- **Fixed Inputs**. The number of lag input variables is fixed, in the same way as traditional time series forecasting methods.
- **Fixed Outputs**. The number of output variables is also fixed; although a more subtle issue, it means that for each input pattern, one output must be produced.

> Sequences pose a challenge for [deep neural networks] because they require that the dimensionality of the inputs and outputs is known and fixed.
>
> — *Sequence to Sequence Learning with Neural Networks*, 2014.

Feedforward neural networks do offer great capability but still suffer from this key limitation of having to specify the temporal dependence upfront in the design of the model. This dependence is almost always unknown and must be discovered and teased out from detailed analysis in a fixed form.

1.3 Convolutional Neural Networks for Time Series

Convolutional Neural Networks or CNNs are a type of neural network that was designed to efficiently handle image data. They have proven effective on challenging computer vision problems both achieving state-of-the-art results on tasks like image classification and providing a component in hybrid models for entirely new problems such as object localization, image captioning and more.

They achieve this by operating directly on raw data, such as raw pixel values, instead of domain-specific or handcrafted features derived from the raw data. The model then learns how to automatically extract the features from the raw data that are directly useful for the problem being addressed. This is called representation learning and the CNN achieves this in such a way that the features are extracted regardless of how they occur in the data, so-called transform or distortion invariance.

> Convolutional networks combine three architectural ideas to ensure some degree of shift and distortion invariance: local receptive fields, shared weights (or weight replication), and, sometimes, spatial or temporal subsampling.
>
> — *Convolutional Networks for Images, Speech, and Time-Series*, 1998.

The ability of CNNs to learn and automatically extract features from raw input data can be applied to time series forecasting problems. A sequence of observations can be treated like a one-dimensional image that a CNN model can read and distill into the most salient elements.

> The key attribute of the CNN is conducting different processing units [...] Such a variety of processing units can yield an effective representation of local salience of the signals. Then, the deep architecture allows multiple layers of these processing units to be stacked, so that this deep learning model can characterize the salience of signals in different scales.

— *Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition*, 2015.

This capability of CNNs has been demonstrated to great effect on time series classification tasks such as automatically detecting human activities based on raw accelerator sensor data from fitness devices and smartphones.

> The key advantages of [CNNs for activity recognition] are: i) feature extraction is performed in task dependent and non hand-crafted manners; ii) extracted features have more discriminative power w.r.t. the classes of human activities; iii) feature extraction and classification are unified in one model so their performances are mutually enhanced.

— *Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition*, 2015.

CNNs get the benefits of Multilayer Perceptrons for time series forecasting, namely support for multivariate input, multivariate output and learning arbitrary but complex functional relationships, but do not require that the model learn directly from lag observations. Instead, the model can learn a representation from a large input sequence that is most relevant for the prediction problem.

- **Feature Learning**. Automatic identification, extraction and distillation of salient features from raw input data that pertain directly to the prediction problem that is being modeled.

1.4 Recurrent Neural Networks for Time Series

Recurrent neural networks like the Long Short-Term Memory network or LSTM add the explicit handling of order between observations when learning a mapping function from inputs to outputs, not offered by MLPs or CNNs. They are a type of neural network that adds native support for input data comprised of sequences of observations.

- **Native Support for Sequences**. Recurrent neural networks directly add support for input sequence data.

The addition of sequence is a new dimension to the function being approximated. Instead of mapping inputs to outputs alone, the network is capable of learning a mapping function for the inputs over time to an output.

> Long Short-Term Memory (LSTM) is able to solve many time series tasks unsolvable by feedforward networks using fixed size time windows.

— *Applying LSTM to Time Series Predictable through Time-Window Approaches*, 2001.

This capability of LSTMs has been used to great effect in complex natural language processing problems such as neural machine translation where the model must learn the complex interrelationships between words both within a given language and across languages in translating form one language to another. This capability can be used in time series forecasting. In addition to the general benefits of using neural networks for time series forecasting, recurrent neural networks can also automatically learn the temporal dependence from the data.

- **Learned Temporal Dependence**. The most relevant context of input observations to the expected output is learned and can change dynamically.

In the simplest case, the network is shown one observation at a time from a sequence and can learn what observations it has seen previously are relevant and how they are relevant to forecasting. The model both learns a mapping from inputs to outputs and learns what context from the input sequence is useful for the mapping, and can dynamically change this context as needed.

> Because of this ability to learn long term correlations in a sequence, LSTM networks obviate the need for a pre-specified time window and are capable of accurately modelling complex multivariate sequences.
>
> — *Long Short Term Memory Networks for Anomaly Detection in Time Series*, 2015.

1.5 Promise of Deep Learning

The capabilities of deep learning neural networks suggest a good fit for time series forecasting. By definition and with enough resources, neural networks in general should be able to subsume the capabilities of classical linear forecasting methods given their ability to learn arbitrary complex mapping from inputs to outputs.

- Neural networks learn arbitrary mapping functions.

It is good practice to manually identify and remove systematic structures from time series data to make the problem easier to model (e.g. make the series stationary), and this may still be a best practice when using recurrent neural networks. But, the general capability of these networks suggests that this may not be a requirement for a skillful model. Technically, the available context of the sequence provided as input may allow neural network models to learn both trend and seasonality directly.

- Neural networks may not require a scaled or stationary time series as input

Each of the three classes of neural network models discussed, MLPs, CNNs and RNNs offer capabilities that are challenging for classical time series forecasting methods, namely:

- Neural networks support multivariate inputs.
- Neural networks support multi-step outputs.

Although MLPs can operate directly on raw observations, CNNs offer efficiency and much greater performance at automatically learning to identify, extract and distill useful features from raw data.

- Convolutional neural networks support efficient feature learning.

Although MLPs and CNNs can learn arbitrary mapping functions, the explicit addition of support for input sequences in RNNs offers efficiency and greater performance for automatically learning the temporal dependencies both within the input sequence and from the input sequence to the output.

- LSTM networks support efficient learning of temporal dependencies.

These capabilities can also be combined, such as in the use of hybrid models like CNN-LSTMs and ConvLSTMs that seek to harness the capabilities of all three model types.

- Hybrid models efficiently combine the diverse capabilities of different architectures.

Traditionally, a lot of research has been invested into using MLPs for time series forecasting with modest results (covered in Chapter 10). Perhaps the most promising area in the application of deep learning methods to time series forecasting are in the use of CNNs, LSTMs and hybrid models. These areas will be our primary focus throughput this book.

1.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Your Expectations**. List three of your own expectations of deep learning methods for time series forecasting, perhaps expectations you had before reading this tutorial.
- **Known Limitations**. Research and list or quote three limitations of deep learning methods for time series forecasting listed in the literature.
- **Successful Example**. Find one research paper that presents a successful application of deep learning methods for time series forecasting.

If you explore any of these extensions, I'd love to know.

1.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- *Deep Learning for Time-Series Analysis*, 2017.
  https://arxiv.org/abs/1701.01887
- *Neural Networks for Time Series Processing*, 1996.
  http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.45.5697

- *Sequence to Sequence Learning with Neural Networks*, 2014.
  https://arxiv.org/abs/1409.3215
- *Applying LSTM to Time Series Predictable through Time-Window Approaches*, 2001.
  https://link.springer.com/chapter/10.1007/3-540-44668-0_93
- *Long Short Term Memory Networks for Anomaly Detection in Time Series*, 2015.
  https://www.elen.ucl.ac.be/Proceedings/esann/esannpdf/es2015-56.pdf
- *Convolutional Networks for Images, Speech, and Time-Series*, 1998.
  https://dl.acm.org/citation.cfm?id=303704
- *Deep Convolutional Neural Networks On Multichannel Time Series For Human Activity Recognition*, 2015.
  https://dl.acm.org/citation.cfm?id=2832806

1.8 Summary

In this tutorial, you discovered the promised capabilities of deep learning neural networks for time series forecasting. Specifically, you learned:

- The focus and implicit, if not explicit, limitations on classical time series forecasting methods.
- The general capabilities of Multilayer Perceptrons and how they may be harnessed for time series forecasting.
- The added capabilities of feature learning and native support for sequences provided by Convolutional Neural Networks and Recurrent Neural Networks.

1.8.1 Next

In the next lesson, you will discover a taxonomy that you can use to quickly learn a lot about your time series forecasting problem.

Chapter 2

Taxonomy of Time Series Forecasting Problems

When you are presented with a new time series forecasting problem, there are many things to consider. The choice that you make directly impacts each step of the project from the design of a test harness to evaluate forecast models to the fundamental difficulty of the forecast problem that you are working on. It is possible to very quickly narrow down the options by working through a series of questions about your time series forecasting problem. By considering a few themes and questions within each theme, you narrow down the type of problem, test harness, and even choice of algorithms for your project. In this tutorial, you will discover a framework that you can use to quickly understand and frame your time series forecasting problem. After reading this tutorial, you will know:

- A structured way of thinking about time series forecasting problems.
- A framework to uncover the characteristics of a given time series forecasting problem.
- A suite of specific questions, the answers to which will help to define your forecasting problem.

Let's get started.

2.1 Framework Overview

Time series forecasting involves developing and using a predictive model on data where there is an ordered relationship between observations. Before you get started on your project, you can answer a few questions and greatly improve your understanding of the structure of your forecast problem, the structure of the model requires, and how to evaluate it. The framework presented in this tutorial is divided into seven parts; they are:

1. Inputs vs. Outputs.
2. Endogenous vs. Exogenous.
3. Unstructured vs. Structured.

4. Regression vs. Classification.
5. Univariate vs. Multivariate.
6. Single-step vs. Multi-step.
7. Static vs. Dynamic.
8. Contiguous vs. Discontiguous.

I recommend working through this framework before starting any time series forecasting project. Your answers may not be crisp on the first time through and the questions may require to you study the data, the domain, and talk to experts and stakeholders. Update your answers as you learn more as it will help to keep you on track, avoid distractions, and develop the actual model that you need for your project.

2.2 Inputs vs. Outputs

Generally, a prediction problem involves using past observations to predict or forecast one or more possible future observations. The goal is to guess about what might happen in the future. When you are required to make a forecast, it is critical to think about the data that you will have available to make the forecast and what you will be guessing about the future. We can summarize this as what are the inputs and outputs of the model when making a single forecast.

- **Inputs**: Historical data provided to the model in order to make a single forecast.
- **Outputs**: Prediction or forecast for a future time step beyond the data provided as input.

The input data is not the data used to train the model. We are not at that point yet. It is the data used to make one forecast, for example the last seven days of sales data to forecast the next one day of sales data. Defining the inputs and outputs of the model forces you to think about what exactly is or may be required to make a forecast. You may not be able to be specific when it comes to input data. For example, you may not know whether one or multiple prior time steps are required to make a forecast. But you will be able to identify the variables that could be used to make a forecast.

What are the inputs and outputs for a forecast?

2.3 Endogenous vs. Exogenous

The input data can be further subdivided in order to better understand its relationship to the output variable. An input variable is endogenous if it is affected by other variables in the system and the output variable depends on it. In a time series, the observations for an input variable depend upon one another. For example, the observation at time t is dependent upon the observation at $t-1$; $t-1$ may depend on $t-2$, and so on. An input variable is an exogenous variable if it is independent of other variables in the system and the output variable depends upon it. Put simply, endogenous variables are influenced by other variables in the system (including themselves) whereas as exogenous variables are not and are considered as outside the system.

- **Endogenous**: Input variables that are influenced by other variables in the system and on which the output variable depends.
- **Exogenous**: Input variables that are not influenced by other variables in the system and on which the output variable depends.

Typically, a time series forecasting problem has endogenous variables (e.g. the output is a function of some number of prior time steps) and may or may not have exogenous variables. Often, exogenous variables are ignored given the strong focus on the time series. Explicitly thinking about both variable types may help to identify easily overlooked exogenous data or even engineered features that may improve the model.

What are the endogenous and exogenous variables?

2.4 Regression vs. Classification

Regression predictive modeling problems are those where a quantity is predicted. A quantity is a numerical value; for example a price, a count, a volume, and so on. A time series forecasting problem in which you want to predict one or more future numerical values is a regression type predictive modeling problem. Classification predictive modeling problems are those where a category is predicted. A category is a label from a small well-defined set of labels; for example `hot`, `cold`, `up`, `down`, and `buy`, `sell` are categories. A time series forecasting problem in which you want to classify input time series data is a classification type predictive modeling problem.

- **Regression**: Forecast a numerical quantity.
- **Classification**: Classify as one of two or more labels.

Are you working on a regression or classification predictive modeling problem?

There is some flexibility between these types. For example, a regression problem can be reframed as classification and a classification problem can be reframed as regression. Some problems, like predicting an ordinal value, can be framed as either classification and regression. It is possible that a reframing of your time series forecasting problem may simplify it.

What are some alternate ways to frame your time series forecasting problem?

2.5 Unstructured vs. Structured

It is useful to plot each variable in a time series and inspect the plot looking for possible patterns. A time series for a single variable may not have any obvious pattern. We can think of a series with no pattern as unstructured, as in there is no discernible time-dependent structure. Alternately, a time series may have obvious patterns, such as a trend or seasonal cycles as structured. We can often simplify the modeling process by identifying and removing the obvious structures from the data, such as an increasing trend or repeating cycle. Some classical methods even allow you to specify parameters to handle these systematic structures directly.

- **Unstructured**: No obvious systematic time-dependent pattern in a time series variable.
- **Structured**: Systematic time-dependent patterns in a time series variable (e.g. trend and/or seasonality).

Are the time series variables unstructured or structured?

2.6 Univariate vs. Multivariate

A single variable measured over time is referred to as a univariate time series. Univariate means one variate or one variable. Multiple variables measured over time is referred to as a multivariate time series: multiple variates or multiple variables.

- **Univariate**: One variable measured over time.
- **Multivariate**: Multiple variables measured over time.

Are you working on a univariate or multivariate time series problem?

Considering this question with regard to inputs and outputs may add a further distinction. The number of variables may differ between the inputs and outputs, e.g. the data may not be symmetrical. For example, you may have multiple variables as input to the model and only be interested in predicting one of the variables as output. In this case, there is an assumption in the model that the multiple input variables aid and are required in predicting the single output variable.

- **Univariate and Multivariate Inputs**: One or multiple input variables measured over time.
- **Univariate and Multivariate Outputs**: One or multiple output variables to be predicted.

2.7 Single-step vs. Multi-step

A forecast problem that requires a prediction of the next time step is called a one-step forecast model. Whereas a forecast problem that requires a prediction of more than one time step is called a multi-step forecast model. The more time steps to be projected into the future, the more challenging the problem given the compounding nature of the uncertainty on each forecasted time step.

- **One-step**: Forecast the next time step.
- **Multi-step**: Forecast more than one future time steps.

Do you require a single-step or a multi-step forecast?

2.8 Static vs. Dynamic

It is possible to develop a model once and use it repeatedly to make predictions. Given that the model is not updated or changed between forecasts, we can think of this model as being static. Conversely, we may receive new observations prior to making a subsequent forecast that could be used to create a new model or update the existing model. We can think of developing a new or updated model prior to each forecasts as a dynamic problem.

For example, it if the problem requires a forecast at the beginning of the week for the week ahead, we may receive the true observation at the end of the week that we can use to update the model prior to making next weeks forecast. This would be a dynamic model. If we do not get a true observation at the end of the week or we do and choose to not re-fit the model, this would be a static model. We may prefer a dynamic model, but the constraints of the domain or limitations of a chosen algorithm may impose constraints that make this intractable.

- **Static**. A forecast model is fit once and used to make predictions.
- **Dynamic**. A forecast model is fit on newly available data prior to each prediction.

Do you require a static or a dynamically updated model?

2.9 Contiguous vs. Discontiguous

A time series where the observations are uniform over time may be described as contiguous. Many time series problems have contiguous observations, such as one observation each hour, day, month or year. A time series where the observations are not uniform over time may be described as discontiguous. The lack of uniformity of the observations may be caused by missing or corrupt values. It may also be a feature of the problem where observations are only made available sporadically or at increasingly or decreasingly spaced time intervals. In the case of non-uniform observations, specific data formatting may be required when fitting some models to make the observations uniform over time.

- **Contiguous**. Observations are made uniform over time.
- **Discontiguous**. Observations are not uniform over time.

Are your observations contiguous or discontiguous?

2.10 Framework Review

To review, the themes and questions you can ask about your problem are as follows:

1. **Inputs vs. Outputs**: What are the inputs and outputs for a forecast?
2. **Endogenous vs. Exogenous**: What are the endogenous and exogenous variables?
3. **Unstructured vs. Structured**: Are the time series variables unstructured or structured?

4. **Regression vs. Classification**: Are you working on a regression or classification predictive modeling problem? What are some alternate ways to frame your time series forecasting problem?
5. **Univariate vs. Multivariate**: Are you working on a univariate or multivariate time series problem?
6. **Single-step vs. Multi-step**: Do you require a single-step or a multi-step forecast?
7. **Static vs. Dynamic**: Do you require a static or a dynamically updated model?
8. **Contiguous vs. Discontiguous**: Are your observations contiguous or discontiguous?

2.11 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Apply Taxonomy**. Select a standard time series dataset and work through the questions in the taxonomy to learn more about the dataset.
- **Standard Form**. Transform the taxonomy into a form or spreadsheet that you can re-use on new time series forecasting projects going forward.
- **Additional Characteristic**. Brainstorm and list at least one additional characteristic of a time series forecasting problem and a question that you might used to identify it.

If you explore any of these extensions, I'd love to know.

2.12 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- *Machine Learning Strategies for Time Series Forecasting*, 2013.
  http://link.springer.com/chapter/10.1007%2F978-3-642-36318-4_3
- *Recursive and direct multi-step forecasting: the best of both worlds*, 2012.
  https://econpapers.repec.org/paper/mshebswps/2012-19.htm

2.13 Summary

In this tutorial, you discovered a framework that you can use to quickly understand and frame your time series forecasting problem. Specifically, you learned:

- A structured way of thinking about time series forecasting problems.
- A framework to uncover the characteristics of a given time series forecasting problem.
- A suite of specific questions, the answers to which will help to define your forecasting problem.

2.13.1 Next

In the next lesson, you will discover a systematic process to ensure that you get better than average performance on your next time series forecasting project.

Chapter 3

How to Develop a Skillful Forecasting Model

"Here is a dataset, now develop a forecast." This is the normal situation that most practitioners find themselves in when getting started on a new time series forecasting problem. You are not alone if you are in this situation right now. In this tutorial, I want to give you a specific and actionable procedure that you can use to work through your time series forecasting problem and get better than average performance from your model. After reading this tutorial, you will know:

- A systematic four-step process that you can use to work through any time series forecasting problem.
- A list of models to evaluate and the order in which to evaluate them.
- A methodology that allows the choice of a final model to be defensible with empirical evidence, rather than whim or fashion.

Let's get started.

3.1 The Situation

You are handed data and told to develop a forecast model. *What do you do?* This is a common situation; far more common than most people think.

- Perhaps you are sent a CSV file.
- Perhaps you are given access to a database.
- Perhaps you are starting a competition.

The problem can be reasonably well defined:

- You have or can access historical time series data.
- You know or can find out what needs to be forecasted.

- You know or can find out how what is most important in evaluating a candidate model.

So how do you tackle this problem? Unless you have been through this trial by fire, you will struggle.

- You may struggle because you are new to the fields of machine learning and time series.
- You may struggle even if you have machine learning experience because time series data is different.
- You may struggle even if you have a background in time series forecasting because machine learning methods may outperform the classical approaches on your data.

In all of these cases, you will benefit from working through the problem carefully and systematically with a reusable process.

3.2 Process Overview

The goal of this process is to get a *good enough* forecast model as fast as possible. This process may or may not deliver the best possible model, but it will deliver a good model: a model that is better than a baseline prediction, if such a model exists. Typically, this process will deliver a model that is 80% to 90% of what can be achieved on the problem.

The process is fast. As such, it focuses on automation. Hyperparameters are searched rather than specified based on careful analysis. You are encouraged to test suites of models in parallel, rapidly getting an idea of what works and what doesn't. Nevertheless, the process is flexible, allowing you to circle back or go as deep as you like on a given step if you have the time and resources. This process is divided into four parts; they are:

1. Define Problem.
2. Design Test Harness.
3. Test Models.
4. Finalize Model.

You will notice that the process is different from a classical linear work-through of a predictive modeling problem. This is because it is designed to get a working forecast model fast and then slow down and see if you can get a better model.

3.3 How to Use This Process

The biggest mistake is skipping steps. For example, the mistake that almost all beginners make is going straight to modeling without a strong idea of what problem is being solved or how to robustly evaluate candidate solutions. This almost always results in a lot of wasted time. Slow down, follow the process, and complete each step.

I recommend having separate code for each experiment that can be re-run at any time. This is important so that you can circle back when you discover a bug, fix the code, and re-run an

experiment. You are running experiments and iterating quickly, but if you are sloppy, then you cannot trust any of your results. This is especially important when it comes to the design of your test harness for evaluating candidate models. Let's take a closer look at each step of the process.

3.4 Step 1: Define Problem

Define your time series problem. Some topics to consider and motivating questions within each topic (taken from the taxonomy in Chapter 2) are as follows:

1. **Inputs vs. Outputs**: What are the inputs and outputs for a forecast?
2. **Endogenous vs. Exogenous**: What are the endogenous and exogenous variables?
3. **Unstructured vs. Structured**: Are the time series variables unstructured or structured?
4. **Regression vs. Classification**: Are you working on a regression or classification predictive modeling problem? What are some alternate ways to frame your time series forecasting problem?
5. **Univariate vs. Multivariate**: Are you working on a univariate or multivariate time series problem?
6. **Single-step vs. Multi-step**: Do you require a single-step or a multi-step forecast?
7. **Static vs. Dynamic**: Do you require a static or a dynamically updated model?
8. **Contiguous vs. Discontiguous**: Are your observations contiguous or discontiguous?

Some useful tools to help get answers include:

- Data visualizations (e.g. line plots, etc.).
- Statistical analysis (e.g. ACF/PACF plots, etc.).
- Domain experts.
- Project stakeholders.

Update your answers to these questions as you learn more.

3.5 Step 2: Design Test Harness

Design a test harness that you can use to evaluate candidate models. This includes both the method used to estimate model skill and the metric used to evaluate predictions. Below is a common time series forecasting model evaluation scheme if you are looking for ideas:

1. Split the dataset into a train and test set.
2. Fit a candidate approach on the training dataset.

3. Make predictions on the test set directly or using walk-forward validation.
4. Calculate a metric that compares the predictions to the expected values.

The test harness must be robust and you must have complete trust in the results it provides. An important consideration is to ensure that any coefficients used for data preparation are estimated from the training dataset only and then applied on the test set. This might include mean and standard deviation in the case of data standardization.

3.6 Step 3: Test Models

Test many models using your test harness. I recommend carefully designing experiments to test a suite of configurations for standard models and letting them run. Each experiment can record results to a file, to allow you to quickly discover the top three to five most skillful configurations from each run. Some common classes of methods that you can design experiments around include the following:

1. **Baseline**. Simple forecasting methods such as persistence and averages.
2. **Autoregression**. The Box-Jenkins process and methods such as SARIMA.
3. **Exponential Smoothing**. Single, double and triple exponential smoothing methods.
4. **Linear Machine Learning**. Linear regression methods and variants such as regularization.
5. **Nonlinear Machine Learning**. kNN, decision trees, support vector regression and more.
6. **Ensemble Machine Learning**. Random forest, gradient boosting, stacking and more.
7. **Deep Learning**. MLPs, CNNs, LSTMs, and Hybrid models.

This list is based on a univariate time series forecasting problem, but you can adapt it for the specifics of your problem, e.g. use VAR/VARMA/etc. in the case of multivariate time series forecasting. Slot in more of your favorite classical time series forecasting methods and machine learning methods as you see fit. Order here is important and is structured in increasing complexity from classical to modern methods. Early approaches are simple and give good results fast; later approaches are slower and more complex, but also have a higher bar to clear to be skillful.

The resulting model skill can be used in a ratchet. For example, the skill of the best persistence configuration provide a baseline skill that all other models must outperform. If an autoregression model does better than persistence, it becomes the new level to outperform in order for a method to be considered skillful. Ideally, you want to exhaust each level before moving on to the next. E.g. get the most out of Autoregression methods and use the results as a new baseline to define *skillful* before moving on to Exponential Smoothing methods. The more time and resources that you have, the more configurations that you can evaluate. For example, with more time and resources, you could:

- Search model configurations at a finer resolution around a configuration known to already perform well.
- Search more model hyperparameter configurations.
- Use analysis to set better bounds on model hyperparameters to be searched.
- Use domain knowledge to better prepare data or engineer input features.
- Explore different potentially more complex methods.
- Explore ensembles of well performing base models.

I also encourage you to include data preparation schemes as hyperparameters for model runs. Some methods will perform some basic data preparation, such as differencing in ARIMA, nevertheless, it is often unclear exactly what data preparation schemes or combinations of schemes are required to best present a dataset to a modeling algorithm. Rather than guess, grid search and decide based on real results. Some data preparation schemes to consider include:

- Differencing to remove a trend.
- Seasonal differencing to remove seasonality.
- Standardize to center.
- Normalize to rescale.
- Power Transform to make normal.

This large amount of systematic searching can be slow to execute. Some ideas to speed up the evaluation of models include:

- Use multiple machines in parallel via cloud hardware (such as Amazon EC2).
- Reduce the size of the train or test dataset to make the evaluation process faster.
- Use a more coarse grid of hyperparameters and circle back if you have time later.
- Perhaps do not refit a model for each step in walk-forward validation.

3.7 Step 4: Finalize Model

At the end of the previous time step, you know whether your time series is predictable. If it is predictable, you will have a list of the top 5 to 10 candidate models that are skillful on the problem. You can pick one or multiple models and finalize them. This involves training a new final model on all available historical data (train and test). The model is ready for use; for example:

- Make a prediction for the future.
- Save the model to file for later use in making predictions.

- Incorporate the model into software for making predictions.

If you have time, you can always circle back to the previous step and see if you can further improve upon the final model. This may be required periodically if the data changes significantly over time.

3.8 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Metrics**. List 3 or more metrics that you could use to evaluate the performance of evaluated time series forecasting models.
- **Modeling Algorithms**. Pick a standard time series forecasting dataset and list 3 or more algorithms that you could evaluate for each level in the *Test Models* section of the framework.
- **Framework Automation**. Design a framework that could use that automates one part or multiple parts of this process.

If you explore any of these extensions, I'd love to know.

3.9 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- *Forecasting: principles and practice*, 2013.
  http://amzn.to/2lln93c
- *Practical Time Series Forecasting with R: A Hands-On Guide*, 2016.
  http://amzn.to/2k3QpuV
- *Statistical and Machine Learning forecasting methods: Concerns and ways forward*, 2018.
  http://journals.plos.org/plosone/article?id=10.1371/journal.pone.0194889

3.10 Summary

In this tutorial, you discovered a systematic process that you can use to quickly discover a skillful predictive model for your time series forecasting problem. Specifically, you learned:

- A systematic four-step process that you can use to work through any time series forecasting problem.
- An list of models to evaluate and the order in which to evaluate them.
- A methodology that allows the choice of a final model to be defensible with empirical evidence, rather than whim or fashion.

3.10.1 Next

In the next lesson, you will discover why and how to transform a time series dataset into samples for training a supervised learning model.

Chapter 4

How to Transform Time Series to a Supervised Learning Problem

Time series forecasting can be framed as a supervised learning problem. This re-framing of your time series data allows you access to the suite of standard linear and nonlinear machine learning algorithms on your problem. In this lesson, you will discover how you can re-frame your time series problem as a supervised learning problem for machine learning. After reading this lesson, you will know:

- What supervised learning is and how it is the foundation for all predictive modeling machine learning algorithms.
- The sliding window method for framing a time series dataset and how to use it.
- How to use the sliding window for multivariate data and multi-step forecasting.

Let's get started.

4.1 Supervised Machine Learning

The majority of practical machine learning uses supervised learning. Supervised learning is where you have input variables (X) and an output variable (y) and you use an algorithm to learn the mapping function from the input to the output.

$$Y = f(X) \tag{4.1}$$

The goal is to approximate the real underlying mapping so well that when you have new input data (X), you can predict the output variables (y) for that data. Below is a contrived example of a supervised learning dataset where each row is an observation comprised of one input variable (X) and one output variable to be predicted (y).

```
X,  y
5,  0.9
4,  0.8
5,  1.0
3,  0.7
4,  0.9
```

Listing 4.1: Example of a small contrived supervised learning dataset.

It is called supervised learning because the process of an algorithm learning from the training dataset can be thought of as a teacher supervising the learning process. We know the correct answers; the algorithm iteratively makes predictions on the training data and is corrected by making updates. Learning stops when the algorithm achieves an acceptable level of performance. Supervised learning problems can be further grouped into regression and classification problems.

- **Classification**: A classification problem is when the output variable is a category, such as `red` and `blue` or `disease` and `no disease`.
- **Regression**: A regression problem is when the output variable is a real value, such as `dollars` or `weight`. The contrived example above is a regression problem.

4.2 Sliding Window

Time series data can be phrased as supervised learning. Given a sequence of numbers for a time series dataset, we can restructure the data to look like a supervised learning problem. We can do this by using previous time steps as input variables and use the next time step as the output variable. Let's make this concrete with an example. Imagine we have a time series as follows:

```
time,  measure
1,     100
2,     110
3,     108
4,     115
5,     120
```

Listing 4.2: Example of a small contrived time series dataset.

We can restructure this time series dataset as a supervised learning problem by using the value at the previous time step to predict the value at the next time step. Re-organizing the time series dataset this way, the data would look as follows:

```
X,   y
?,   100
100, 110
110, 108
108, 115
115, 120
120, ?
```

Listing 4.3: Example of time series dataset as supervised learning.

Take a look at the above transformed dataset and compare it to the original time series. Here are some observations:

- We can see that the previous time step is the input (X) and the next time step is the output (y) in our supervised learning problem.
- We can see that the order between the observations is preserved, and must continue to be preserved when using this dataset to train a supervised model.

- We can see that we have no previous value that we can use to predict the first value in the sequence. We will delete this row as we cannot use it.
- We can also see that we do not have a known next value to predict for the last value in the sequence. We may want to delete this value while training our supervised model also.

The use of prior time steps to predict the next time step is called the sliding window method. For short, it may be called the window method in some literature. In statistics and time series analysis, this is called a lag or lag method. The number of previous time steps is called the window width or size of the lag. This sliding window is the basis for how we can turn any time series dataset into a supervised learning problem. From this simple example, we can notice a few things:

- We can see how this can work to turn a time series into either a regression or a classification supervised learning problem for real-valued or labeled time series values.
- We can see how once a time series dataset is prepared this way that any of the standard linear and nonlinear machine learning algorithms may be applied, as long as the order of the rows is preserved.
- We can see how the width sliding window can be increased to include more previous time steps.
- We can see how the sliding window approach can be used on a time series that has more than one value, or so-called multivariate time series.

We will explore some of these uses of the sliding window, starting next with using it to handle time series with more than one observation at each time step, called multivariate time series.

4.3 Sliding Window With Multiple Variates

The number of observations recorded for a given time in a time series dataset matters. Traditionally, different names are used:

- **Univariate Time Series**: These are datasets where only a single variable is observed at each time, such as temperature each hour. The example in the previous section is a univariate time series dataset.
- **Multivariate Time Series**: These are datasets where two or more variables are observed at each time.

Most time series analysis methods, and even books on the topic, focus on univariate data. This is because it is the simplest to understand and work with. Multivariate data is often more difficult to work with. It is harder to model and often many of the classical methods do not perform well.

> Multivariate time series analysis considers simultaneously multiple time series. [...]
> It is, in general, much more complicated than univariate time series analysis

— Page 1, *Multivariate Time Series Analysis: With R and Financial Applications*, 2013.

The sweet spot for using machine learning for time series is where classical methods fall down. This may be with complex univariate time series, and is more likely with multivariate time series given the additional complexity. Below is another worked example to make the sliding window method concrete for multivariate time series. Assume we have the contrived multivariate time series dataset below with two observations at each time step. Let's also assume that we are only concerned with predicting `measure2`.

```
time,  measure1,  measure2
1,     0.2,       88
2,     0.5,       89
3,     0.7,       87
4,     0.4,       88
5,     1.0,       90
```

Listing 4.4: Example of a small contrived multivariate time series dataset.

We can re-frame this time series dataset as a supervised learning problem with a window width of one. This means that we will use the previous time step values of `measure1` and `measure2`. We will also have available the next time step value for `measure1`. We will then predict the next time step value of `measure2`. This will give us 3 input features and one output value to predict for each training pattern.

```
X1,   X2,   X3,   y
?,    ?,    0.2,  88
0.2,  88,   0.5,  89
0.5,  89,   0.7,  87
0.7,  87,   0.4,  88
0.4,  88,   1.0,  90
1.0,  90,   ?,    ?
```

Listing 4.5: Example of a multivariate time series dataset as a supervised learning problem.

We can see that as in the univariate time series example above, we may need to remove the first and last rows in order to train our supervised learning model. This example raises the question of what if we wanted to predict both `measure1` and `measure2` for the next time step? The sliding window approach can also be used in this case. Using the same time series dataset above, we can phrase it as a supervised learning problem where we predict both `measure1` and `measure2` with the same window width of one, as follows.

```
X1,   X2,   y1,   y2
?,    ?,    0.2,  88
0.2,  88,   0.5,  89
0.5,  89,   0.7,  87
0.7,  87,   0.4,  88
0.4,  88,   1.0,  90
1.0,  90,   ?,    ?
```

Listing 4.6: Example of a multivariate time series dataset as a multi-step or sequence prediction supervised learning problem.

Not many supervised learning methods can handle the prediction of multiple output values without modification, but some methods, like artificial neural networks, have little trouble. We can think of predicting more than one value as predicting a sequence. In this case, we were

predicting two different output variables, but we may want to predict multiple time steps ahead of one output variable. This is called multi-step forecasting and is covered in the next section.

4.4 Sliding Window With Multiple Steps

The number of time steps ahead to be forecasted is important. Again, it is traditional to use different names for the problem depending on the number of time steps to forecast:

- **One-step Forecast**: This is where the next time step (`t+1`) is predicted.
- **Multi-step Forecast**: This is where two or more future time steps are to be predicted.

All of the examples we have looked at so far have been one-step forecasts. There are a number of ways to model multi-step forecasting as a supervised learning problem. For now, we are focusing on framing multi-step forecast using the sliding window method. Consider the same univariate time series dataset from the first sliding window example above:

```
time, measure
1,    100
2,    110
3,    108
4,    115
5,    120
```

Listing 4.7: Example of a small contrived time series dataset.

We can frame this time series as a two-step forecasting dataset for supervised learning with a window width of one, as follows:

```
X1,   y1,   y2
?     100,  110
100,  110,  108
110,  108,  115
108,  115,  120
115,  120,  ?
120,  ?,    ?
```

Listing 4.8: Example of a univariate time series dataset as a multi-step or sequence prediction supervised learning problem.

We can see that the first row and the last two rows cannot be used to train a supervised model. It is also a good example to show the burden on the input variables. Specifically, that a supervised model only has $X1$ to work with in order to predict both $y1$ and $y2$. Careful thought and experimentation are needed on your problem to find a window width that results in acceptable model performance.

4.5 Implementing Data Preparation

In Chapter 6 we touch on data preparation an focus on how to transform time series data in order to meet the expectations of a three-dimensional structure by some deep learning methods. In Chapters 7, 8, and 9 we will explore how to implement deep learning methods for univariate,

multivariate and multi-step forecasting with a specific focus on how to prepare the data for model. In each chapter we will develop functions that can be reused to prepare time series data on future projects.

4.6 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Samples in Detail**. Explain how the choice of the number of time steps in a sample impacts training a model and making predictions with a trained model.
- **Work Through Dataset**. Select a standard time series forecasting dataset and demonstrate how it may be transformed into samples for supervised learning.
- **Develop Function**. Develop a function in Python to transform a time series into a samples for supervised learning.

If you explore any of these extensions, I'd love to know.

4.7 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

- *Multivariate Time Series Analysis: With R and Financial Applications*, 2013. https://amzn.to/2KD4rw7
- *Machine Learning for Sequential Data: A Review*, 2002. https://link.springer.com/chapter/10.1007/3-540-70659-3_2
- *Machine Learning Strategies for Time Series Forecasting*, 2013. https://link.springer.com/chapter/10.1007/978-3-642-36318-4_3

4.8 Summary

In this lesson, you discovered how you can re-frame your time series prediction problem as a supervised learning problem for use with machine learning methods. Specifically, you learned:

- Supervised learning is the most popular way of framing problems for machine learning as a collection of observations with inputs and outputs.
- Sliding window is the way to restructure a time series dataset as a supervised learning problem.
- Multivariate and multi-step forecasting time series can also be framed as supervised learning using the sliding window method.

4.8.1 Next

In the next lesson, you will discover naive and classical time series forecasting methods that you absolutely must evaluate before investigating more sophisticated deep learning methods.

Chapter 5

Review of Simple and Classical Forecasting Methods

Machine learning and deep learning methods can achieve impressive results on challenging time series forecasting problems. Nevertheless, there are many forecasting problems where classical methods such as SARIMA and exponential smoothing readily outperform more sophisticated methods. Therefore, it is important to both understand how classical time series forecasting methods work and to evaluate them prior to exploring more advanced methods. In this tutorial, you will discover naive and classical methods for time series forecasting.

- How to develop simple forecasts for time series forecasting problems that provide a baseline for estimating model skill.
- How to develop autoregressive models for time series forecasting.
- How to develop exponential smoothing methods for time series forecasting.

Let's get started.

5.1 Simple Forecasting Methods

Establishing a baseline is essential on any time series forecasting problem. A baseline in performance gives you an idea of how well all other models will actually perform on your problem. In this section, you will discover how to develop a simple forecasting methods that you can use to calculate a baseline level of performance on your time series forecasting problem.

5.1.1 Forecast Performance Baseline

A baseline in forecast performance provides a point of comparison. It is a point of reference for all other modeling techniques on your problem. If a model achieves performance at or below the baseline, the technique should be fixed or abandoned. The technique used to generate a forecast to calculate the baseline performance must be easy to implement and naive of problem-specific details. The goal is to get a baseline performance on your time series forecast problem as quickly as possible so that you can get to work better understanding the dataset and developing more advanced models. Three properties of a good technique for making a naive forecast are:

- **Simple**: A method that requires little or no training or intelligence.
- **Fast**: A method that is fast to implement and computationally trivial to make a prediction.
- **Repeatable**: A method that is deterministic, meaning that it produces an expected output given the same input.

5.1.2 Forecast Strategies

Simple forecast strategies are those that assume little or nothing about the nature of the forecast problem and are fast to implement and calculate. If a model can perform better than the performance of a simple forecast strategy, then it can be said to be skillful. There are two main themes to simple forecast strategies; they are:

- **Naive**, or using observations values directly.
- **Average**, or using a statistic calculated on previous observations.

Let's take a closer look at both of these strategies.

5.1.3 Naive Forecasting Strategy

A naive forecast involves using the previous observation directly as the forecast without any change. It is often called the persistence forecast as the prior observation is persisted. This simple approach can be adjusted slightly for seasonal data. In this case, the observation at the same time in the previous cycle may be persisted instead. This can be further generalized to testing each possible offset into the historical data that could be used to persist a value for a forecast. For example, given the series:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listing 5.1: Example of a univariate time series.

We could persist the last observation (relative index -1) as the value 9 or persist the second last prior observation (relative index -2) as 8, and so on.

5.1.4 Average Forecast Strategy

One step above the naive forecast is the strategy of averaging prior values. All prior observations are collected and averaged, either using the mean or the median, with no other treatment to the data. In some cases, we may want to shorten the history used in the average calculation to the last few observations. We can generalize this to the case of testing each possible set of n-prior observations to be included into the average calculation. For example, given the series:

```
[1, 2, 3, 4, 5, 6, 7, 8, 9]
```

Listing 5.2: Example of a univariate time series.

We could average the last one observation (9), the last two observations (8, 9), and so on. In the case of seasonal data, we may want to average the last n-prior observations at the same time in the cycle as the time that is being forecasted. For example, given the series with a 3-step cycle:

```
[1, 2, 3, 1, 2, 3, 1, 2, 3]
```

Listing 5.3: Example of a univariate time series with seasonal structure.

We could use a window size of 3 and average the last one observation (-3 or 1), the last two observations (-3 or 1, and -(3 × 2) or 1), and so on.

5.1.5 Implementing Simple Strategies

In Chapter 11 we will explore how to implement the naive and the average forecast strategy and how to grid search the hyperparameters of these strategies for univariate time series forecasting datasets. The results from these strategies provide the baseline by which the performance of more sophisticated models may be judged skillful, or not. In Chapter 17 we will explore how to develop domain-specific naive forecast strategies for making multi-step forecasts for predicting household electricity usage.

5.2 Autoregressive Methods

Autoregressive Integrated Moving Average, or ARIMA, is one of the most widely used forecasting methods for univariate time series data forecasting. Although the method can handle data with a trend, it does not support time series with a seasonal component. An extension to ARIMA that supports the direct modeling of the seasonal component of the series is called SARIMA. In this section, you will discover the Seasonal Autoregressive Integrated Moving Average, or SARIMA, method for time series forecasting with univariate data containing trends and seasonality.

5.2.1 Autoregressive Integrated Moving Average Model

An ARIMA model is a class of statistical models for analyzing and forecasting time series data. It explicitly caters to a suite of standard structures in time series data, and as such provides a simple yet powerful method for making skillful time series forecasts. ARIMA is an acronym that stands for AutoRegressive Integrated Moving Average. It is a generalization of the simpler AutoRegressive Moving Average or ARMA and adds the notion of integration. This acronym is descriptive, capturing the key aspects of the model itself. Briefly, they are:

- **AR**: *Autoregression*. A model that uses the dependent relationship between an observation and some number of lagged observations.
- **I**: *Integrated*. The use of differencing of raw observations (e.g. subtracting an observation from an observation at the previous time step) in order to make the time series stationary.
- **MA**: *Moving Average*. A model that uses the dependency between an observation and a residual error from a moving average model applied to lagged observations.

Each of these components are explicitly specified in the model as a parameter. A standard notation is used of `ARIMA(p,d,q)` where the parameters are substituted with integer values to quickly indicate the specific ARIMA model being used. A problem with ARIMA is that it does

not support seasonal data. That is a time series with a repeating cycle. ARIMA expects data that is either not seasonal or has the seasonal component removed, e.g. seasonally adjusted via methods such as seasonal differencing.

A linear regression model is constructed including the specified number and type of terms, and the data is prepared by a degree of differencing in order to make it stationary, i.e. to remove trend and seasonal structures that negatively affect the regression model. A value of 0 can be used for a parameter, which indicates to not use that element of the model. This way, the ARIMA model can be configured to perform the function of an ARMA model, and even a simple AR, I, or MA model. Adopting an ARIMA model for a time series assumes that the underlying process that generated the observations is an ARIMA process. This may seem obvious, but helps to motivate the need to confirm the assumptions of the model in the raw observations and in the residual errors of forecasts from the model.

5.2.2 What is Seasonal ARIMA

Seasonal Autoregressive Integrated Moving Average, SARIMA or Seasonal ARIMA, is an extension of ARIMA that explicitly supports univariate time series data with a seasonal component. It adds three new hyperparameters to specify the autoregression (AR), differencing (I) and moving average (MA) for the seasonal component of the series, as well as an additional parameter for the period of the seasonality.

> A seasonal ARIMA model is formed by including additional seasonal terms in the ARIMA [...] The seasonal part of the model consists of terms that are very similar to the non-seasonal components of the model, but they involve backshifts of the seasonal period.
>
> — Page 242, *Forecasting: Principles and Practice*, 2013.

5.2.3 How to Configure SARIMA

Configuring a SARIMA requires selecting hyperparameters for both the trend and seasonal elements of the series.

Trend Elements

There are three trend elements that require configuration. They are the same as the ARIMA model; specifically:

- **p**: Trend autoregression order.
- **d**: Trend difference order.
- **q**: Trend moving average order.

Seasonal Elements

There are four seasonal elements that are not part of ARIMA that must be configured; they are:

- **P**: Seasonal autoregressive order.
- **D**: Seasonal difference order.
- **Q**: Seasonal moving average order.
- **m**: The number of time steps for a single seasonal period.

Together, the notation for an SARIMA model is specified as: `SARIMA(p,d,q)(P,D,Q)m`. Where the specifically chosen hyperparameters for a model are specified. Importantly, the m parameter influences the P, D, and Q parameters. For example, an m of 12 for monthly data suggests a yearly seasonal cycle. A $P = 1$ would make use of the first seasonally offset observation in the model, e.g. $t - (m \times 1)$ or $t - 12$. A $P = 2$, would use the last two seasonally offset observations $t - (m \times 1)$, $t - (m \times 2)$. Similarly, a D of 1 would calculate a first order seasonal difference and a $Q = 1$ would use a first order errors in the model (e.g. moving average).

> A seasonal ARIMA model uses differencing at a lag equal to the number of seasons (s) to remove additive seasonal effects. As with lag 1 differencing to remove a trend, the lag s differencing introduces a moving average term. The seasonal ARIMA model includes autoregressive and moving average terms at lag s
>
> — Page 142, *Introductory Time Series with R*, 2009.

The trend elements can be chosen through careful analysis of ACF and PACF plots looking at the correlations of recent time steps (e.g. 1, 2, 3). Similarly, ACF and PACF plots can be analyzed to specify values for the seasonal model by looking at correlation at seasonal lag time steps.

> Seasonal ARIMA models can potentially have a large number of parameters and combinations of terms. Therefore, it is appropriate to try out a wide range of models when fitting to data and choose a best fitting model using an appropriate criterion ...
>
> — Pages 143-144, *Introductory Time Series with R*, 2009.

Alternately, a grid search can be used across the trend and seasonal hyperparameters.

5.2.4 Implementing ARIMA and SARIMA Models

In Chapter 13 we will look at how to develop a reusable framework for automatically grid searching the hyperparameters for the SARIMA model on a range of standard univariate time series forecasting problems. In Chapter 18 we will analyze ACF and PACF plots of a real-world household electricity usage dataset and develop an ARIMA to make multi-step time series forecasts.

5.3 Exponential Smoothing Methods

Exponential smoothing is a time series forecasting method for univariate data that can be extended to support data with a systematic trend or seasonal component. It is a powerful forecasting method that may be used as an alternative to the popular Box-Jenkins ARIMA family of methods. In this section, you will discover the exponential smoothing method for univariate time series forecasting.

5.3.1 What Is Exponential Smoothing?

Exponential smoothing is a time series forecasting method for univariate data. Time series methods like the Box-Jenkins ARIMA family of methods develop a model where the prediction is a weighted linear sum of recent past observations or lags. Exponential smoothing forecasting methods are similar in that a prediction is a weighted sum of past observations, but the model explicitly uses an exponentially decreasing weight for past observations. Specifically, past observations are weighted with a geometrically decreasing ratio.

> Forecasts produced using exponential smoothing methods are weighted averages of past observations, with the weights decaying exponentially as the observations get older. In other words, the more recent the observation the higher the associated weight.
>
> — Page 171, *Forecasting: Principles and Practice*, 2013.

Exponential smoothing methods may be considered as peers and an alternative to the popular Box-Jenkins ARIMA class of methods for time series forecasting. Collectively, the methods are sometimes referred to as ETS models, referring to the explicit modeling of Error, Trend and Seasonality. There are three main types of exponential smoothing time series forecasting methods. A simple method that assumes no systematic structure, an extension that explicitly handles trends, and the most advanced approach that add support for seasonality.

5.3.2 Single Exponential Smoothing

Single Exponential Smoothing, SES for short, also called Simple Exponential Smoothing, is a time series forecasting method for univariate data without a trend or seasonality. It requires a single parameter, called alpha (a or α), also called the smoothing factor or smoothing coefficient. This parameter controls the rate at which the influence of the observations at prior time steps decay exponentially. Alpha is often set to a value between 0 and 1. Large values mean that the model pays attention mainly to the most recent past observations, whereas smaller values mean more of the history is taken into account when making a prediction.

> A value close to 1 indicates fast learning (that is, only the most recent values influence the forecasts), whereas a value close to 0 indicates slow learning (past observations have a large influence on forecasts).
>
> — Page 89, *Practical Time Series Forecasting with R*, 2016.

Hyperparameters:

- **Alpha** (α): Smoothing factor for the level.

5.3.3 Double Exponential Smoothing

Double Exponential Smoothing is an extension to Exponential Smoothing that explicitly adds support for trends in the univariate time series. In addition to the alpha parameter for controlling smoothing factor for the level, an additional smoothing factor is added to control the decay of the influence of the change in trend called beta (b or β). The method supports trends that change in different ways: an additive and a multiplicative, depending on whether the trend is linear or exponential respectively. Double Exponential Smoothing with an additive trend is classically referred to as Holt's linear trend model, named for the developer of the method Charles Holt.

- **Additive Trend**: Double Exponential Smoothing with a linear trend.
- **Multiplicative Trend**: Double Exponential Smoothing with an exponential trend.

For longer range (multi-step) forecasts, the trend may continue on unrealistically. As such, it can be useful to dampen the trend over time. Dampening means reducing the size of the trend over future time steps down to a straight line (no trend).

> The forecasts generated by Holt's linear method display a constant trend (increasing or decreasing) indecently into the future. Even more extreme are the forecasts generated by the exponential trend method [...] Motivated by this observation [...] introduced a parameter that "dampens" the trend to a flat line some time in the future.
>
> — Page 183, *Forecasting: Principles and Practice*, 2013.

As with modeling the trend itself, we can use the same principles in dampening the trend, specifically additively or multiplicatively for a linear or exponential dampening effect. A damping coefficient Phi (p or ϕ) is used to control the rate of dampening.

- **Additive Dampening**: Dampen a trend linearly.
- **Multiplicative Dampening**: Dampen the trend exponentially.

Hyperparameters:

- **Alpha** (α): Smoothing factor for the level.
- **Beta** (β): Smoothing factor for the trend.
- **Trend Type**: Additive or multiplicative.
- **Dampen Type**: Additive or multiplicative.
- **Phi** (ϕ): Damping coefficient.

5.3.4 Triple Exponential Smoothing

Triple Exponential Smoothing is an extension of Exponential Smoothing that explicitly adds support for seasonality to the univariate time series. This method is sometimes called Holt-Winters Exponential Smoothing, named for two contributors to the method: Charles Holt and Peter Winters. In addition to the alpha and beta smoothing factors, a new parameter is added called gamma (g or γ) that controls the influence on the seasonal component. As with the trend, the seasonality may be modeled as either an additive or multiplicative process for a linear or exponential change in the seasonality.

- **Additive Seasonality**: Triple Exponential Smoothing with a linear seasonality.
- **Multiplicative Seasonality**: Triple Exponential Smoothing with an exponential seasonality.

Triple exponential smoothing is the most advanced variation of exponential smoothing and through configuration, it can also develop double and single exponential smoothing models.

> Being an adaptive method, Holt-Winter's exponential smoothing allows the level, trend and seasonality patterns to change over time.
>
> — Page 95, *Practical Time Series Forecasting with R*, 2016.

Additionally, to ensure that the seasonality is modeled correctly, the number of time steps in a seasonal period (Period) must be specified. For example, if the series was monthly data and the seasonal period repeated each year, then the `Period=12`. Hyperparameters:

- **Alpha** (α): Smoothing factor for the level.
- **Beta** (β): Smoothing factor for the trend.
- **Trend Type**: Additive or multiplicative.
- **Dampen Type**: Additive or multiplicative.
- **Phi** (ϕ): Damping coefficient.
- **Gamma** (γ): Smoothing factor for the seasonality.
- **Seasonality Type**: Additive or multiplicative.
- **Period**: Time steps in seasonal period.

5.3.5 How to Configure Exponential Smoothing

All of the model hyperparameters can be specified explicitly. This can be challenging for experts and beginners alike. Instead, it is common to use numerical optimization to search for and fund the smoothing coefficients (alpha, beta, gamma, and phi) for the model that result in the lowest error.

> a more robust and objective way to obtain values for the unknown parameters included in any exponential smoothing method is to estimate them from the observed data. [...] the unknown parameters and the initial values for any exponential smoothing method can be estimated by minimizing the SSE [sum of the squared errors].
>
> — Page 177, *Forecasting: Principles and Practice*, 2013.

The parameters that specify the type of change in the trend and seasonality, such as weather they are additive or multiplicative and whether they should be dampened, must be specified explicitly.

5.3.6 Implementing ETS Models

In Chapter 12 we will look at how to develop a reusable framework for automatically grid searching the hyperparameters for the ETS model on a range of standard univariate time series forecasting problems. You can adapt and reuse this framework on your own time series forecasting problems going forward.

5.4 Extensions

This section lists some ideas for extending the tutorial that you may wish to explore.

- **Summarize Methods**. Write a one line summary of naive, autoregressive and exponential smoothing time series forecasting methods.
- **Examples of Algorithms**. List 3 examples of algorithms that may be used for each of the naive, autoregressive and exponential smoothing time series forecasting methods.
- **API Wrappers**. Develop a reusable wrapper function for fitting and making predictions with one time of classical time series forecasting method provided by the Statsmodels library.

If you explore any of these extensions, I'd love to know.

5.5 Further Reading

This section provides more resources on the topic if you are looking to go deeper.

5.5.1 Simple Methods

- Chapter 2, The forecaster's toolbox, *Forecasting: Principles and Practice*, 2013.
  https://amzn.to/2xlJsfV
- Forecasting, Wikipedia.
  https://en.wikipedia.org/wiki/Forecasting

5.5.2 Autoregressive Methods

- Chapter 8, ARIMA models, *Forecasting: Principles and Practice*, 2013.
  https://amzn.to/2xlJsfV
- Chapter 7, *Non-stationary Models, Introductory Time Series with R*, 2009.
  https://amzn.to/2smB9LR
- Autoregressive integrated moving average on Wikipedia.
  https://en.wikipedia.org/wiki/Autoregressive_integrated_moving_average

5.5.3 Exponential Smoothing

- Chapter 7, Exponential smoothing, *Forecasting: Principles and Practice*, 2013.
  https://amzn.to/2xlJsfV
- Section 6.4. Introduction to Time Series Analysis, *Engineering Statistics Handbook*, 2012.
  https://www.itl.nist.gov/div898/handbook/
- *Practical Time Series Forecasting with R*, 2016.
  https://amzn.to/2LGKzKm
- Exponential smoothing, Wikipedia.
  https://en.wikipedia.org/wiki/Exponential_smoothing

5.6 Summary

In this tutorial, you discovered naive and classical methods for time series forecasting. Specifically, you learned:

- How to develop simple forecasts for time series forecasting problems that provide a baseline for estimating model skill.
- How to develop autoregressive models for time series forecasting.
- How to develop exponential smoothing methods for time series forecasting.

5.6.1 Next

This is the final lesson of this part, the next part will focus on how to develop deep learning models for time series forecasting in general.