Understanding Deep Learning

Application in Rare Event Prediction

Chitta Ranjan

Understanding Deep Learning Application in Rare Event Prediction

Chitta Ranjan, Ph.D.

First Edition

Chitta Ranjan, Ph.D.
Director of Science, ProcessMiner Inc.
cranjan@processminer.com
https://medium.com/@cran2367

ISBN: 9798586189561

Printed in the United States of America.

Dedicated to my parents,
Dr. Radha Krishna Prasad and Chinta Prasad.

With love, Bharti.

Preface

Deep learning is an art. But it has some boundaries. Learning the boundaries is essential to develop working solutions and, ultimately, push them for novel creations.

For example, cooking is an art with some boundaries. For instance, most cooks adhere to: salt stays out of desserts. But chefs almost always add salt to desserts because they know the edge of this boundary. They understand that while salt is perceived as a condiment it truly is a flavor enhancer; when added to a dessert it enhances the flavor of every ingredient.

Some chefs push the boundaries further. Claudia Fleming, a distinguished chef from New York, went to an extreme in her *pineapple* dessert. Each component in it is heavily salted. Yet the dessert is not salty. Instead, each flavor feels magnified. The salt makes the dessert an extraordinary dish.

The point is that an understanding of the constructs of food allows one to create extraordinary recipes. Similarly, an understanding of deep learning constructs enables one to create extraordinary solutions.

This book aims at providing such a level of understanding to a reader on the primary constructs of deep learning: multi-layer perceptrons, long- and short-term memory networks from the recurrent neural network family, convolutional neural networks, and autoencoders.

Further, to retain an understanding, it is essential to develop a mastery of (intuitive) visualizations. For example, Viswanathan Anand, a chess grandmaster, and a five-time world chess champion is quoted in *Mind Master: Winning Lessons from a Champion's Life*, "chess players

visualize the future and the path to winning." There, he alludes that a player is just as good as (s)he can visualize.

Likewise, the ability to intuitively visualize a deep learning model is essential. It helps to see the flow of information in a network, and its transformations along the way. A visual understanding makes it easier to build the most appropriate solution.

This book provides ample visual illustrations to develop this skill. For example, an LSTM cell, one of the most complex constructs in deep learning is visually unfolded to vividly understand the information flow within it in Chapter 5.

The understanding and visualizations of deep learning constructs are shrouded by their (mostly) abstruse theories. The book focuses on simplifying them and explain to a reader **how** and **why** a construct works. While the "how it works" makes a reader learn a concept, the "why it works" helps the reader unravel the concept. For example, Chapter 4 explains how dropout works followed by why it works?

The teachings in the book are solidified with implementations. This book solves a rare event prediction problem to exemplify the deep learning constructs in every chapter. The book explains the problem formulation, data preparation, and modeling to enable a reader to apply the concepts to other problems.

This book is appropriate for graduate and Ph.D. students, as well as researchers and practitioners.

To the practitioners, the book provides complete illustrative implementations to use in developing solutions. For researchers, the book has several research directions and implementations of new ideas, e.g., custom activations, regularizations, and multiple pooling.

In graduate programs, this book is suitable for a one-semester introductory course in deep learning. The first three chapters introduce the field, a working example, and sets up a student with TensorFlow. The rest of the chapters present the deep learning constructs and the concepts therein.

These chapters contain exercises. Most of them illustrate concepts in the respective chapter. Their practice is encouraged to develop a

stronger understanding of the subject. Besides, a few advanced exercises are marked as optional. These exercises could lead a reader to develop novel ideas.

Additionally, the book illustrates how to implement deep learning networks with TensorFlow in Python. The illustrations are kept verbose and, mostly, verbatim for readers to be able to use them directly. The link to the code repository is also available at `https://github.com/cran2367/deep-learning-and-rare-event-prediction`.

My journey in writing this book reminded me of a quote by Dr. Frederick Sanger, a two-time Nobel laureate, "it is like a voyage of discovery, seeking not for new territory but new knowledge. It should appeal to those with a good sense of adventure."

I hope every reader enjoys this voyage in deep learning and find their adventure.

Chitta Ranjan

Acknowledgment

A trail of learning experiences ultimately leads to writing a book. I have drawn from many people during my experiences to whom I am very thankful.

I thank Dr. Kamran Paynabar and Dr. Jhareswar Maiti who instilled in me the passion for learning and discovery during my academic life.

I also want to thank Dr. David Hardtke, Dr. Ronald Menich, and Karim Pourak for giving me an intellectually rich environment at my past and current work. The professional and sometimes philosophical conversations with them gave me insightfulness to write this book.

My father, Dr. Radha Krishna Prasad, who is an inspiration in my life helped me get started with the book. His guidance during the writing invaluably enriched it.

Besides, bringing this book to its completion would not have been possible without the support of my wife, Bharti. I am grateful for her immeasurable support and motivation when writing was difficult.

I also thank Dr. Shad Kirmani, eBay Inc., Zhen Zhu, Stanford University, and Jason Wu, who helped improve the theoretical content in the book. I am also grateful to Professor Mary Orr and Grace Gibbs for their inputs.

Lastly, I want to thank my family and friends for their perennial encouragement.

Website

This book provides several implementations of deep learning networks in TensorFlow. Additionally, video lectures are provided for most of the sections in the book. The website provides the link to the code repository, links to the lectures, and other resources useful for both readers and instructors.

www.understandingdeeplearning.com

Contents

This page is intentionally left blank.

Chapter 1

Introduction

Data is important to draw inferences and predictions. As obvious as it may sound today, it has been a collective work over the centuries that has brought us this far. Francis Bacon (1561-1626) proposed a Baconian method to propagate this "concept" in the 16th century. His book *Novum Organum* (1620) advanced Aristotle's *Organon* to advocate data collection and analysis as the basis of knowledge.

Centuries have passed since then. And today, data collection and analysis has unarguably become an important part of most processes.

As a result, data corpuses are multiplying. Appropriate use of these abundant data will make us potently effective. And a key to this is **recognizing predictive patterns from data** for better decision making.

Without this key, data by itself is a dump. But, at the same time, drawing the valuable predictive patterns from this dump is a challenge that we are facing today.

> "We are drowning in information while starving for knowledge. The world henceforth will be run by synthesizers, people able to put together the right information at the right time, think critically about it, and make important choices wisely." – E.O. Wilson, *Consilience: The Unity of Knowledge* (1998).

John Naisbitt stated the first part of this quote in Megatrends (1982)

which was later extended by Dr. Edward Osborne Wilson in his book *Consilience: The Unity of Knowledge* (1998). Both of them have emphasized the importance of data and the significance of drawing patterns from it.

Humans inherently learn patterns from data. For example, as a child grows she learns touching a hot cup will burn. She would learn this after doing it a few times (collecting data) and realizing the touch burns (a pattern). Over time, she learns several other patterns that help her to make decisions.

However, as problems become more complex humans' abilities become limited. For example, we might foretell today's weather by looking at the morning sun but cannot predict it for the rest of the week.

This is where Artificial Intelligence (AI) comes into the picture. AI enables an automatic derivation of predictive patterns. Sometimes the patterns are interpretable and sometimes otherwise. Regardless, these automatically drawn patterns are usually quite predictive.

In the last two decades, AI has become one of the most studied fields. Some of the popular texts in AI are, *Pattern recognition and machine learning* by Bishop, C. Bishop 2006, *The elements of statistical learning* by Friedman, J., Hastie, T., and Tibshirani, R. Hastie, Tibshirani, and Friedman 2009, and *Deep Learning* by LeCun, Y., Bengio, Y., and Hinton, G. LeCun, Bengio, and G. Hinton 2015.

In this book, we will go a little further than them to understand the constructs of deep learning. A rare event prediction problem is also solved side-by-side to learn the application and implementation of the constructs.

Rare event prediction is a special problem with profound importance. **Rare events are the events that occur infrequently**. Statistically, if an event constitutes less than 5% of the data set, it is categorized as a rare event. In this book, even rarer events that **occur less than 1%** are discussed and modeled.

Despite being so rare when these events occur, their consequences can be quite dramatic and often adverse. Due to which, such problems are sometimes also referred to as adverse event prediction.

Rare event problem has been categorized under various umbrellas.

For instance, "mining needle in a haystack," "chance discovery," "exception mining," and so on. The rare event prediction problem in this book is, however, different from most of these categories. It is predicting a rare event in advance. For example, predicting a tsunami before it hits the shore.

Rare event prediction is predicting a rare event in advance.

In the next section, a few motivating rare event examples are posed. Thereafter, a dialogue on machine learning versus deep learning approaches and the reasoning for selecting deep learning is made in § 1.3. Lastly, a high-level overview of the rest of the book is given in § 1.4.

1.1 Examples of Application

Rare event problems surround all of us. A few motivating examples are presented below.

> *Before going further, take a moment and think of the rare event problems that you see around. What are their impacts? How would it be if we could predict them? Proceed with the thought in mind.*

1.1.1 Rare Diseases

There are 5,000 to 8,000 known rare diseases. Based on World Update Report (2013)[1] by WHO, 400 million people worldwide of which 25 million in the US are affected by a rare disease.

Some rare diseases are chronic and can be life-threatening. An **early detection** and diagnosis of these diseases will significantly improve the patients' health and may save lives.

[1] `https://www.who.int/medicines/areas/priority_medicines/Ch6_19Rare.pdf`

The International Rare Diseases Research Consortium (IRDiRC) was launched in 2011 at the initiative of the European Commission and the US National Institutes of Health to foster international collaboration in rare diseases research. However, despite these developments, rare disease diagnosis and early detection is still a major challenge.

1.1.2 Fraud Detection

Digital frauds, such as credit card and online transactions, are becoming a costly problem for many business establishments and even countries. Every year billions of dollars are siphoned in credit card frauds. These frauds have been growing year after year due to the growth in online sales. A decade ago the estimated loss due to online fraud was $4 billion in 2008, an increase of 11% from $3.6 billion in 2007.

The fraud's magnitude is large in dollars but constitutes a fraction of all the transactions. This makes it extremely challenging to detect. For example, a credit card fraud data set provided by Kaggle has 0.172% of the samples labeled as fraud[2].

An **early detection** of these frauds can help in a timely prevention of the fraudulent transactions.

1.1.3 Network Intrusion Detection

Attacks on computer systems and computer-networks are not unheard-of. Today most organizations, including hospitals, traffic control, and sensitive government bodies, are run on computers. Attacks on such systems can prove to be extremely fatal.

Due to its importance, this is an active field of research. To bolster the research, KDD-CUP'99 contest provided a network intrusion data set[3] from Defense Advanced Research Projects Agency (DARPA), an agency of the United States Department of Defense responsible for the development of emerging technologies for use by the military. The data included a wide variety of intrusions simulated in a military network

[2] `https://www.kaggle.com/mlg-ulb/creditcardfraud/`

[3] `http://kdd.ics.uci.edu/databases/kddcup99/kddcup99`

environment. Few examples of network attacks were: denial-of-service (dos), surveillance (probe), remote-to-local (r2l), and user-to-root (u2r). Among which, the u2r and r2l categories are intrinsically rare but fatal.

The objective in such systems are quick or **early detection** of such intrusions or attacks to prevent any catastrophic breach.

1.1.4 Detecting Emergencies

Audio-video surveillance is generally accepted by society today. The use of cameras for live traffic monitoring, parking lot surveillance, and public gatherings, is getting accepted as they make us feel safer. Similarly, placing cameras for video surveillance of our homes is becoming common.

Unfortunately, most such surveillance still depends on a human operator sifting through the videos. It is a tedious and tiring job—monitoring for events-of-interest that rarely occur. The sheer volume of these data impedes easy human analysis and, thus, necessitates automated predictive solutions for assistance.

TUT Rare Sound events 2017, a development data set[4], provided by the Audio Research Group at Tampere University of Technology is one such data set of audio recordings from home surveillance systems. It contains labeled samples for baby crying, glass breaking, and gunshot. The last two events being the rare events of concern (housebreak) requiring **early detection**.

1.1.5 Click vis-à-vis churn prediction

Digital industries, such as Pandora, eBay, and Amazon, rely heavily on subscriptions and advertisements. To maximize their revenue, the objective is to increase the clicks (on advertisements) while minimizing customers' churn.

A simple solution to increasing customer clicks would be to show more advertisements. But it will come at the expense of causing cus-

[4] `http://www.cs.tut.fi/sgn/arg/dcase2017/challenge/task-rare-sound-event-detection`

tomer churn. A churn is a customer leaving the website which could be in the form of exiting a transaction page or sometimes even unsubscribing. Churns, therefore, cost dearly to the companies.

To address this problem, companies attempt at a targeted marketing that maximizes the click probability while minimizing the churn probability by modeling customer behaviors.

Typically, customer behavior is a function of their past activities. Due to this, most research attempts at developing predictive systems that can **early predict** the possibility of click and churn to better capitalize on the revenue opportunities. However, both of these events are "rare" which makes it challenging.

1.1.6 Failures in Manufacturing

Today most manufacturing plants whether continuous or discrete run 24x7. Any downtime causes a significant cost to the plant. For instance, in 2006 the automotive manufacturers in the US estimated production line downtime costs at about $22,000 per minute or $1.3 million per hour (Advanced Technology Services 2006). Some estimates ran as high as $50,000 per minute.

A good portion of these downtimes is caused due to a *failure*. Failures occur in either machine parts or the product. For example, predicting and preventing machine bearing failure is a classical problem in the machining and automotive industries.

Another common problem on the product failures are found in pulp-and-paper industries. These plants continuously produce paper sheets. Sometimes sheet breakage happens that causes the entire process to stop. According to Bonissone et. al. (2002) Bonissone, Goebel, and Y.-T. Chen 2002, this causes an expected loss of $3 billion every year across the industry.

More broadly, as per a report by McKinsey in 2015 Manyika 2015, predictive systems have the potential to save global manufacturing businesses up to $630 billion per year by 2025.

These failures are a rare event from a predictive modeling point-of-view. For which, an **early prediction** of the failures for a potential

prevention is the objective.

Outside of these, there are several other applications, such as stock market collapse, recession prediction, earthquake prediction, and so on.

None of these events should occur in an ideal world. We, therefore, **envision to have prediction systems to predict them in advance to either prevent or minimize the impact**.

1.2 A Working Example

A paper sheet-break problem in paper manufacturing is taken from Ranjan et al. 2018 as a working example in this book. The methods developed in the upcoming chapters will be applied on a sheet-break data set.

The problem formulation in the upcoming chapter will also refer to this problem. The sheet-break working example is described and used for developing a perspective. However, the formulation and developed methods apply to other rare event problems.

1.2.1 Problem Motivation

Paper sheet-break at paper mills is a critical problem. On average, a mill witnesses more than one sheet-break every day. The average is even higher for fragile papers, such as paper tissues and towels.

Paper manufacturing is a continuous process. These sheet-breaks are unwanted and costly hiccups in the production. Each sheet-break can cause downtime of an hour or longer. As mentioned in the previous section, these downtime cause loss of millions of dollars at a plant and billions across the industry. Even a small reduction in these breaks would lead to significant savings.

More importantly, fixing a sheet-break often requires an operator to enter the paper machine. These are large machines with some hazardous sections that pose danger to operators' health. Preventing sheet-break via predictive systems will make operators' work condition better and

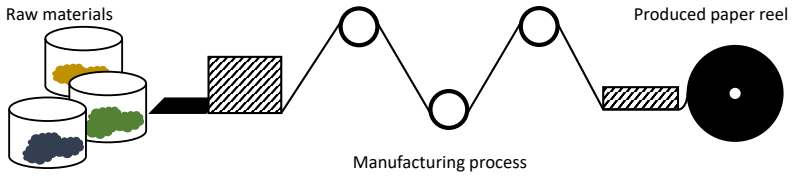


Figure 1.1. *A high-level schematic outline of a continuous paper manufacturing process.*

the paper production more sustainable.

1.2.2 Paper Manufacturing Process

Paper manufacturing machines are typically half a mile long. A condensed illustration of the machine and the process is given in Figure 1.1. As shown, the raw materials get into the machine from one end. They go through manufacturing processes of forming, press, drying, and calendaring in the same order. Ultimately, a large reel of the paper sheet is yielded at the other end. This process runs continuously.

In this continuous process, sometimes sheet tears are called **sheet-breaks** in the paper industry. There are several possible causes for a break. They could be instantaneous or gradual. Breaks due to instantaneous causes like something falling on the paper are difficult to predict. But gradual effects that lead to a break can be caught in advance. For example, in the illustrative outline in Figure 1.1 suppose one of the rollers on the right starts to rotate asynchronously faster than the one on its left. This asynchronous rotation will cause the sheet to stretch and eventually break. When noticed in time, an imminent break can be predicted and prevented.

1.2.3 Data Description

The sheet-break data set in consideration has observations from several sensors measuring the raw materials, such as the amount of pulp fiber, chemicals, etc., and the process variables, such as blade type, couch

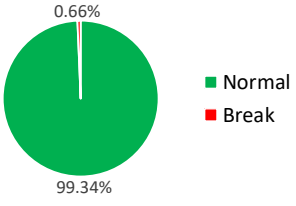


Figure 1.2. *Class distribution in paper sheet-break data.*

vacuum, rotor speed, etc.

The data contains the process status marked as: normal or break. The class distribution is extremely skewed in this data. Shown in Figure 1.2, the data has only 0.66% positive labeled samples.

1.3 Machine Learning vs. Deep Learning

Machine Learning is known for its simplicity especially in regards to its interpretability. Machine learning methods are, therefore, usually the first choice for most problems. Deep learning, on the other hand, provides more possibilities. In this section, these two choices are debated.

As noticed in § 1.2.3 we have an imbalanced binary labeled multivariate time series process. For over two decades, imbalanced binary classification has been actively researched. However, there are still several open challenges in this topic. Krawczyk 2016 discussed these challenges and the research progresses.

One major open challenge for machine learning methods is an apparent absence of a robust and simple modeling framework. The summarization of ML methods by Sun, Wong, and Kamel 2009 in Figure 1.3 makes this more evident.

As seen in the figure, there are several disjoint approaches. For each of these approaches, there are some unique methodological modifications. These modifications are usually not straightforward to implement.

Another critical shortcoming of the above methods is their limitations with multivariate time series processes.

To address this, there are machine learning approaches for multi-

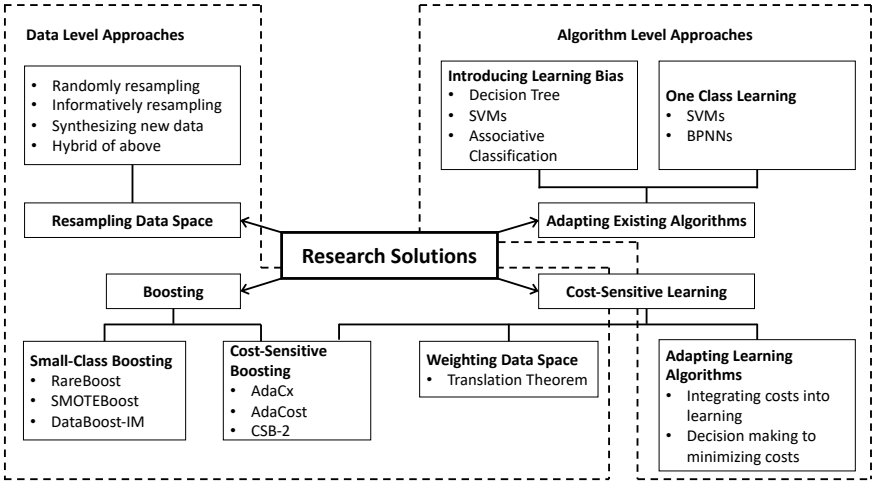


Figure 1.3. *A summary of research solutions in machine learning for imbalanced data classification (Sun, Wong, and Kamel 2009).*

variate time series classification. For example, see Batal et al. 2009; Orsenigo and Vercellis 2010; Górecki and Łuczak 2015. Unfortunately, these methods are not directly applicable to imbalanced data sets.

In sum, most of the related machine learning approaches are solving a part of an, "*imbalanced multivariate time series problem.*" A robust and easy-to-implement solution framework to solve the problem is, therefore, missing in machine learning.

Deep learning, on the other hand, provides a better possibility.

It is mentioned in § 2.2.3 that traditional oversampling and data augmentation techniques do not work well with extremely rare events. Fortunately, in the rest of the book, it is found that deep learning models do not necessarily require data augmentation.

Intuitively this could be logical. Deep learning models are inspired by the mechanisms of human brains. We, humans, do not require oversampled rare events or objects to learn to distinguish them. For example, we do not need to see several Ferrari cars to learn how one looks like.

Similarly, deep learning models might learn to distinguish rare events

from a few samples present in a large data set. The results in this book empirically support this supposition. But this is still a conjecture and the true reason could be different.

Importantly, there are architectures in deep learning that provide a simpler framework to solve a complex problem such as an imbalanced multivariate time series.

Given the above, deep learning methods are developed for rare event prediction in the subsequent chapters of this book.

1.4 In this Book

In the next chapter, a rare event problem is framed by elucidating its underlying process, the problem definition, and the objective. Thereafter, before getting to deep learning constructs, a reader is set up with TensorFlow in Chapter 3. Both these chapters can be skipped if the reader intends to only learn the deep learning theories.

The book then gets into deep learning topics starting with deconstructing multi-layer perceptrons in Chapter 4. They are made of *dense* layers—the most fundamental deep learning construct. More complex and advanced long- and short- term memory (LSTM) constructs are in Chapter 5. Convolutional layers, considered as the workhorse of deep learning, are in Chapter 6. Lastly, an elemental model in deep learning, autoencoders, are presented in Chapter 7.

The chapters uncover the core concepts behind these constructs. A simplified mathematics behind dense and other layers, state information flow in LSTM, filtration mechanism in convolution, and structural features in autoencoder are presented.

Some of the concepts are even rediscovered. For example,

- Chapter 4 draws parallels between a simple regression model and a multi-layer perceptron. It shows how nonlinear activations differentiates them by implicitly breaking down a complex problem into small pieces. The essential gradient properties of activations are laid. Moreover, customized activation is also implemented for re-

search illustrations. Besides, the *dropout* concept which enhances MLP (and most other networks) is explained along with answering how to use it?

- LSTM mechanisms are one of the most mystical in deep learning. Chapter 5 deconstructs an LSTM cell to visualize the state information flow that preserves the memory. The secret behind it is the gradient transmission which is explained in detail. Moreover, the variants of LSTM available in TensorFlow such as *backward* and *bi-directional* are visually explained. Besides, LSTMs have a rich history with several variants. They are tabulated to compare against the variant in TensorFlow. This gives a perspective of the LSTM commonly in-use versus the other possibilities.

- Convolutional networks use *convolution* and *pooling* operations. They are simple operations yet make convolutional networks one of the most effective models. Chapter 6 explains both concepts with visual exemplifications and theoretical clarifications. The chapter goes into detail to explain convolutional and pooling properties such as parameter sharing, filtration, and invariance. Besides, the need for pooling to summarize convolution is statistically concretized. The statistics behind pooling also answer questions like why max-pool generally outperforms others, and what other pooling statistics are viable?

- Autoencoders are constructs for unsupervised, and semi-supervised learning. Chapter 7 explains them by drawing parallels with principal component analysis in machine learning. The comparison makes it easier to understand their internal mechanisms. Autoencoders' ability to learn specific tasks such as denoising or feature learning for classification by incorporating regularization are also presented. Besides, classifiers trained on encodings are developed.

Every chapter explains the concepts along with illustrating model implementations in TensorFlow. The implementation steps are also explained in detail to help a reader learn model building.

This book acknowledges that there is no perfect model. And, therefore, the book aims at showing **how** to develop the models to enable a reader to build his own best model.

> *Think of deep learning as an art of cooking. One way to cook is to follow a recipe. But when we learn* ***how*** *the food, the spices, and the fire behave, we make our creation. And an understanding of the* ***how*** *transcends the creation.*
>
> *Likewise, an understanding of the* ***how*** *transcends deep learning. In this spirit, this book presents the deep learning constructs, their fundamentals, and how they behave. Baseline models are developed alongside, and concepts to improve them are exemplified.*

Besides, the enormity of deep learning modeling choices can be overwhelming. To avoid that, we have a few rules-of-thumb before concluding a chapter. It lists the basics of building and improving a deep learning model.

In summary, deep learning offers a robust framework to solve complex problems. It has several constructs. Using them to arrive at the best model is sometimes difficult. The focus of this book is to understand these constructs to have a direction to develop effective deep learning models. Towards the end of the book, a reader would understand every construct and how to put them together.

Chapter 2

Rare Event Prediction

2.1 Rare Event Problem

The previous chapter emphasized the importance of rare event problems. This chapter formulates the problem by laying out the underlying statistical process, problem definition, and objectives. Moreover, the challenges in meeting the objectives are also discussed. The working example in § 1.2 is referred to during the formulation.

2.1.1 Underlying Statistical Process

First, the statistical process behind a rare event problem is understood. This helps in selecting an appropriate approach (see the discussion in § 1.3).

The process and data commonalities in the rare event examples in § 1.1 are

- time series,
- multivariate, and
- imbalanced binary labels.

Consider our working example of a sheet-break problem. It is from a continuous paper manufacturing process that generates a data stream.

This makes the underlying process a **stochastic time series**.

Additionally, this is a **multivariate** data streamed from multiple sensors placed in different parts of the machine. These sensors collect the process variables, such as temperature, pressure, chemical dose, etc.

Thus, at any time t, a vector of observations $\boldsymbol{x}_t$ is recorded. Here, $\boldsymbol{x}_t$ is a vector of length equal to the number of sensors and x_{it} the reading of the i-th sensor at time t. Such a process is known as a **multivariate time series**.

In addition to the process variables $\boldsymbol{x}_t$, a binary label y_t denoting the status of the process is also available. A positive y_t indicates an occurrence of the rare event. Also, due to the rareness of positive y_t's, the class distribution is imbalanced.

For instance, the labels in the sheet-break data denote whether the process is running normal ($y_t = 0$), or there is a sheet-break ($y_t = 1$). The samples with $y_t = 0$ and $y_t = 1$ is referred to as *negatively* and *positively* labeled data in the rest of the book. The former is the majority class and the latter minority.

Putting them together, we have an **imbalanced multivariate stochastic time series** process. Mathematically represented as, $(y_t, \boldsymbol{x}_t), t = 1, 2, \ldots$ where $y_t \in \{0, 1\}$ with $\sum \mathbb{1}\{y_t = 1\} \ll \sum \mathbb{1}\{y_t = 0\}$ and $\boldsymbol{x}_t \in \mathbb{R}^p$ with p being the number of variables.

2.1.2 Problem definition

Rare event problems demand an early detection or prediction to prevent the event or minimize its impact.

In literature, detection and prediction are considered as different problems. However, in the problems discussed here, early detection eventually becomes a prediction. For example, early detection of a condition that would lead to a sheet-break is essentially predicting an imminent sheet-break. This can, therefore, be formulated as a "prediction" problem.

An early prediction problem is predicting an event in advance. Suppose the event prediction is needed k time units in advance. This k should be chosen such that the prediction gives sufficient time to take

an action against the event.

Mathematically, this can be expressed as estimating the probability of $y_{t+k} = 1$ using the information at and until time t, i.e.,

$$\Pr[y_{t+k} = 1 | \boldsymbol{x}_{t^-}] \tag{2.1}$$

where $\boldsymbol{x}_{t^-}$ denotes $\boldsymbol{x}$ before time t, i.e., $\boldsymbol{x}_{t^-} = \{\boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots\}$.

Equation 2.1 also shows that this is a classification problem. Therefore, prediction and classification are used interchangeably in this book.

2.1.3 Objective

The objective is to **build a binary classifier to predict a rare event in advance**. To that end, an appropriate loss function and accuracy measures are selected for predictive modeling.

Loss function

There are a variety of loss functions. Among them, binary cross-entropy loss is chosen here.

Cross-entropy is intuitive and has the appropriate theoretical properties that make it a good choice. Its gradient lessens the vanishing gradients issue in deep learning networks. Moreover, from the model fitting standpoint, cross-entropy approximates the Kullback-Leibler divergence. Meaning, minimizing cross-entropy yields an approximate estimation of the "true" underlying process distributions[1].

It is defined as,

$$\begin{aligned} \mathcal{L}(\theta) = &-y_{t+k} \log(\Pr[y_{t+k} = 1 | \boldsymbol{x}_{t-}, \theta]) - \\ &(1 - y_{t+k}) \log(1 - \Pr[y_{t+k} = 1 | \boldsymbol{x}_{t-}, \theta]) \end{aligned} \tag{2.2}$$

where θ denotes the model.

[1] Refer to a recent paper at NIPS 2018 Zhang and Sabuncu 2018 for more details and advancements in cross-entropy loss.

Entropy means randomness. The higher the entropy the more the randomness. More randomness means a less predictable model, i.e., if the model is random it will make poor predictions.

Consider an extreme output of an arbitrary model: an absolute opposite prediction, e.g., estimating $\Pr[y = 1] = 0$ when $y = 1$. In such a case, the loss in Equation 2.2 will be, $\mathcal{L} = -1*\log(0)-(1-1)*\log(1-0) = -1 * -\infty - 0 * 0 = +\infty$.

On the other extreme, consider an oracle model: makes absolute true prediction, i.e. $\Pr[y = 1] = 1$ when $y = 1$. In this case, the cross-entropy loss will become, $\mathcal{L} = -1 * \log(1) - (1 - 1) * \log(1 - 1) = 0$.

During model training, any arbitrary model is taken as a starting point. The loss is, therefore, high at the beginning. The model then trains itself to lower the loss. This is done iteratively to bring the cross-entropy loss from $+\infty$ towards 0.

Accuracy measures

Rare event problems have extremely imbalanced class distribution. The traditional *misclassification* accuracy metric does not work here.

This is because more than 99% of our samples are negatively labeled. A model that predicts everything, including all the minority $< 1\%$ positive samples, as negative is still $> 99\%$ accurate. Thus, a model that cannot predict any rare event would appear accurate. The area under the ROC curve (AUC) measure is also unsuitable for such extremely imbalanced problems.

In building a classifier, if there is any deviation from the usual it is useful to fall back to the *confusion matrix* and look at other accuracy measures drawn from it. A confusion matrix design is shown in Table 2.1.

The "actuals" and the "predictions" are along the rows and columns of the matrix. Their values are either negative or positive. As mentioned before, negative corresponds to the normal state of the process and is the majority class. And, positive corresponds to the rare event and is the minority class.

The actual negative or positive samples predicted as the same go on

the diagonal cells of the matrix as *true negative* (TN) and *true positive* (TP), respectively. The other two possibilities are if an actual negative is predicted as a positive and the vice versa denoted as *false positive* (FP) and *false negative* (FN), respectively.

In rare event classifiers, the goal is inclined towards maximizing the true positives while ensuring it does not lead to excessive false predictions. In light of this goal, the following accuracy measures are chosen and explained vis-à-vis the confusion matrix.

- **recall**: the percentage of positive samples correctly predicted as one, i.e., $\frac{\text{TP}}{\text{TP + FN}}$. It lies between 0 and 1. A high recall indicates the model's ability to accurately predicting the minority class. A recall equal to one means the model could detect all of the rare events. However, this can also be achieved with a dummy model that predicts everything as a rare event. To counterbalance this, we also use f1-score.

- **f1-score**: a combination (harmonic mean) of precision[2] and recall. This score indicates the model's overall ability in predicting most of the rare events with as few false alerts as possible. It is computed as, $\frac{2}{\left(\frac{\text{TP}}{\text{TP + FN}}\right)^{-1}+\left(\frac{\text{TP}}{\text{TP + FP}}\right)^{-1}}$. The score lies between 0 and 1 with higher the better. If we have the dummy model favoring high recall by predicting all samples as positive (a rare event), the f1-score will counteract it by getting close to zero.

- **false positive rate (fpr)**: lastly, it is also critical to measure the false positive rate. Fpr is the percentage of false alerts, i.e., $\frac{\text{FP}}{\text{FP + TN}}$. An excess of false alerts makes us insensitive to the predictions. It is, therefore, imperative to keep the fpr as close to zero as possible.

[2] A ratio of the true positives overall predicted positives. The ratio lies between 0 to 1 with higher the better. This measure shows the model performance w.r.t. high true positives and low false positives. High precision is indicative of this and vice versa.

Table 2.1. Confusion matrix.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative (TN)	False Positive (FP)
	Positive	False Negative (FN)	True Positive (TP)
	** Negative: Normal process and the majority class.*		
	*** Positive: Rare event and the minority class.*		

2.2 Challenges

The step after formulating a problem is to identify the modeling challenges. Challenges, if identified, enable a better modeling direction. It tells a practitioner the issues to address during the modeling.

A few acute challenges posed by a rare event problem are,

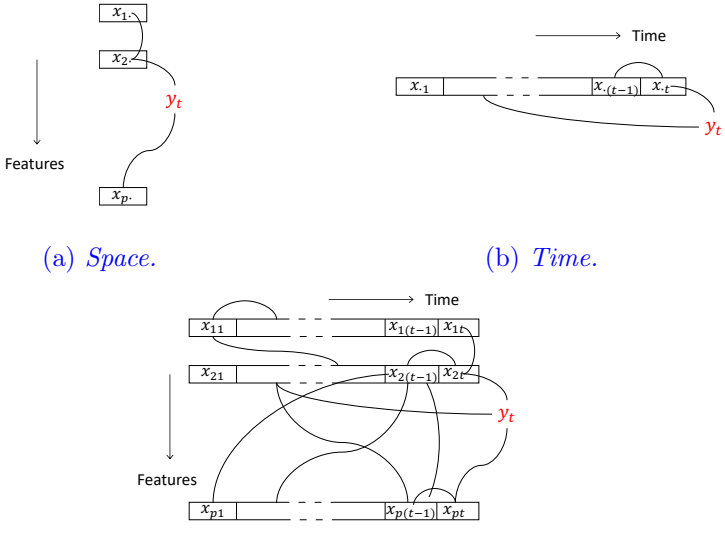
- high-dimensional multivariate time series process,
- early prediction, and
- imbalanced data.

2.2.1 High-dimensional Multivariate Time Series

This is a mouthful and, hence, broken down to its elements for clarity. Earlier, § 2.1.1 mentioned that a rare event process is a multivariate time series. A multivariate process has multiple features (variables). Rare event problems typically have 10s to 100s of features which categorizes them as a high-dimensional process.

A **high-dimensional** process poses modeling challenges due to "spatial" relationships between the features. This is also known as *cross-correlations* in *space*. The term "space" is used because the features mentioned here are spread in a *space*.

While this space is in a mathematical context, for an intuitive understanding, think of the sensors placed at different locations in space on a paper manufacturing machine and how they correlate with each other.



(a) *Space.* (b) *Time.*

(c) *Space and time.*

Figure 2.1. *Spatio-temporal relationships.*

Figure 2.1a shows a visual illustration of spatial correlations and dependencies of features with the response. Since every feature can be related to each other, the possible spatial relationships increase exponentially with the number of features, p.

The issue with excess spatial relationships with large p is that they may induce spurious dependencies in the model. For example, in the figure x_1 and x_2 are correlated and x_2 is related to y. But the model may select x_1 instead of x_2 or both x_1 and x_2.

Such spurious dependencies often cause high model variance and overfitting, and therefore, should be avoided.

Time series, also referred to as **temporal processes**, pose another critical challenge. Temporal features exhibit correlations with themselves (autocorrelation[3]) and long-short term dependencies with

[3]The correlation of a variable with a lagged value of itself. For example, x_t being correlated with its previous value x_{t-1}.

the response.

For illustration, Figure 2.1b shows x_{t-1} and x_t are autocorrelated. It is important to isolate and/or account for these dependencies in the model to avoid high model variance.

Additionally, we see that x_t and an early observation close to x_1 are related to y_t indicative of short- and long-term dependencies, respectively. While estimating the short-term dependencies are relatively simpler, long-term dependencies are quite challenging to derive.

But long-term dependencies are common and should not be ignored. For example, in a paper manufacturing process, a few chemicals that are fed at an early stage of the production line affects the paper quality hours later. In some processes, such long-term relationships are even a few days apart.

A major issue in drawing these long-term dependencies is that any prior knowledge on the lag with which a feature affects the response may be unavailable. In absence of any prior knowledge, we have to include all the feature lags in a model. This blows up the size of the model and makes its estimation difficult.

A high-dimensional and time series process are individually challenging. If they are together, their challenges get multiplied.

Figure 2.1c is visually illustrating this complexity. As we can see here, in a high-dimensional time series process the features have, a. relationships with themselves, b. long- and short-term dependencies, and c. cross-correlations in space and time. Together, all of these are called **spatio-temporal relationships**.

As shown in the figure, the dependencies to estimate in a spatio-temporal structure grows exponentially with more features and a longer time horizon. Modeling such processes is, therefore, extremely challenging.

2.2.2 Early Prediction

As mentioned in § 2.1.2, we need to predict an event in advance. Ideally, we would like to predict it well in advance. However, the farther we are

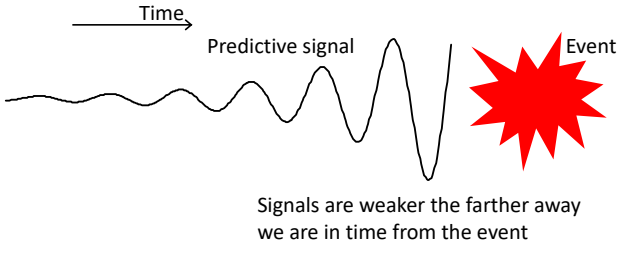


Figure 2.2. *Challenge with early prediction.*

from an event the weaker are the predictive signals and, therefore, results in poor predictions.

Visually illustrated in Figure 2.2, the red zagged mark indicates the occurrence of an event and the horizontal axis is time. As shown, the predictive signal will be the most dominant closest to the event. The farther we are in a time the weaker is the signal.

This is generally true for any event irrespective of whether it is rare or not. However, due to the dramatic adverse impact of a rare event, it is critical to be able to predict it well in advance. And the challenge is the more in advance prediction we want the harder it gets for a model.

2.2.3 Imbalanced Data

A rare event problem by definition has imbalanced data. The issue with imbalanced data sets is that the model is heavily skewed towards the majority class. In such situations, learning the underlying patterns predictive of the rare event becomes difficult.

A typical resolution to address the imbalanced data issue is oversampling. Oversampling, however, does not work in most rare event problems. This is because, a. extreme imbalance in class distribution, and b. time series data.

The former typically make resampling approaches inapplicable because it requires excessive resampling to balance the data. This causes the model to get extremely biased.

The latter prohibits the usage of more sophisticated oversampling

techniques, such as SMOTE. This is because samples "interpolation" for data synthesis done in SMOTE takes into account the spatial aspect of the process but not temporal (refer to Appendix L). Due to this, the synthesized data does not necessarily accentuate or retain the underlying predictive patterns.

The temporal aspect of our problem also prohibits the use of most other data augmentation approaches. In problems like image classification, techniques such as reorientation and rotation augment the data.

But with temporal features, any such augmentation distorts the data and, consequently, the inherent dependencies. Time series data augmentation methods using slicing and time warping are available but they do not work well with multivariate time series.

In the rest of the book, appropriate modeling directions to address these challenges are discussed.

Chapter 3

Setup

TensorFlow and the working example is set up in this chapter. The chapter begins with the reasoning for choosing TensorFlow followed by laying out the steps to set it up.

After this setup, the chapter describes the paper sheet-break problem, a data set for it, and a few basic preprocessing steps. The following sections are intentionally succinct to keep the setup process short.

3.1 TensorFlow

There are a bunch of platform choices for deep learning. For example, Theano, PyTorch, and TensorFlow. Among them, the book uses the recent TensorFlow 2x. The section begins with its reasoning and then the installation steps in Ubuntu, Mac, and Windows.

Why TensorFlow 2x?

TensorFlow 2x was released in 2019 and is expected to change the landscape of deep learning. It has made,

- model building simpler,
- production deployment on any platform more robust, and

- enables powerful experimentation for research.

With these, TF 2x is likely to propel deep learning to mainstream applications in research and industry alike.

TF 2x has Keras API integrated into it. Keras is a popular high-level API for building and training deep learning models. In regards to TF 2x and Keras, it is important to know,

- TF 1.10+ also supports Keras. But in TF 2x Keras is also integrated with the rest of the TensorFlow platform. TF 2x is providing a single high-level API to reduce confusion and enable advanced capabilities.
- `tf.keras` in TF 2x is different from the `keras` library from `www.keras.io`. The latter is an independent open source project that precedes TF 2x and was built to support multiple backends, viz. TensorFlow, CNTK, and Theano.

It is recommended to read TensorFlow 2018 and TensorFlow 2019 from the TensorFlow team for more details and benefits of TF 2x. In summary, they state that TF 2x has,

- brought an ease-of-implementation,
- immense computational efficiency, and
- compatibility with most mobile platforms, such as Android and iOS, and embedded edge systems like Raspberry Pi and edge TPUs.

Achieving these was difficult before. As TF 2x brings all of these benefits, this book chose it.

Fortunately, the installation has become simple with TF 2x. In the following, the installation prerequisites, the installation, and testing steps are given.

> **Note**: Using Google Colab environment is an alternative to this installation. Google Colab is generally an easier way to work with TensorFlow. It is a notebook on Google Cloud with all the TensorFlow requisites pre-installed.

3.1.1 Prerequisites

TensorFlow is available only for Python 3.5 or above. Also, it is recommended to use a virtual environment. Steps for both are presented here.

Install Python

Anaconda

Anaconda with Jupyter provides a simple approach for installing Python and working with it.

Installing Anaconda is relatively straightforward. Follow this link `https://jupyter.org/install` and choose the latest Python.

System

First, the current Python version (if present) is looked for.

```
$ python --version
Python 3.7.1
```

or,

```
$ python3 --version
Python 3.7.1
```

It this version is less than 3.5, Python can be installed as shown in the listing below.

Listing 3.1. Install Python.

```
$ brew update
$ brew install python # Installs Python 3
$ sudo apt install python3-dev python3-pip
```

Install Virtual Environment

An optional requirement is using a virtual environment. Its importance is in the next section, § 3.1.2. It can be installed as follows[1].

[1]Refer: `https://packaging.python.org/guides/installing-using-pip-and-virtual-environments/`

Mac/Ubuntu

```
$ python3 -m pip install --user virtualenv
```

Windows

```
py -m pip install --user virtualenv
```

Now the system ready to install TensorFlow.

3.1.2 TensorFlow 2x Installation

Instantiate and activate a virtual environment

The setup will begin by creating a virtual environment.

Why is the virtual environment important? A virtual environment is an isolated environment for Python projects. Inside a virtual environment one can have a completely independent set of packages (dependencies) and settings that will not conflict with anything in other virtual environment or with the default local Python environment.

This means that different versions of the same package can be used in different projects at the same time on the same system but in different virtual environments.

Mac/Ubuntu

Create a virtual environment called `tf_2`.

```
$ virtualenv --system-site-packages -p python3 tf_2
$ source tf_2/bin/activate  # Activate the
    virtualenv
```

The above command will create a virtual environment `tf_2`. In the command,

- `virtualenv` will create a virtual environment.
- `-system-site-packages` allow the projects within the virtual environment `tf_2` to access the global site-packages. The default

setting does not allow this access (`-no-site-packages` were used before for this default setting but now deprecated.)

- `-p python3` is used to set the Python interpreter for `tf_2`. This argument can be skipped if the `virtualenv` was installed with Python 3. By default, that is the python interpreter for the virtual environment. Another option for setting Python 3.x as an interpreter is `$ virtualenv -system-site-packages -python=python3.7 tf_2`. This gives more control.

- `tf_2` is the name of the virtual environment created. Any other name of a user's choice can also be given. The physical directory created at the location of the virtual environments will bear this name. The (`/tf_2`) directory will contain a copy of the Python compiler and all the packages installed afterward.

Anaconda

With Anaconda, the virtual environment is created using Conda as,

```
$ conda create -n tf_2
$ conda activate tf_2 # Activate the virtualenv
```

The above command will also create a virtual environment `tf_2`. Unlike before, pre-installation of the virtual environment package is not required with Anaconda. The in-built `conda` command provides this facility.

Understanding the command,

- `conda` can be used to create virtual environments, install packages, list the installed packages in the environment, and so on. In short, `conda` performs operations that `pip` and `virtualenv` do. While `conda` replaces `virtualenv` for most purposes, it does not replace `pip` as some packages are available on `pip` but not on `conda`.

- `create` is used to create a virtual environment.

- `-n` is an argument specific to `create`. `-n` is used to name the virtual environment. The value of `n`, i.e. the environment name, here is `tf_2`.

- Additional useful arguments: similar to `--system-site-packages` in `virtualenv`, `--use-local` can be used.

Windows

Create a new virtual environment by choosing a Python interpreter and making a `.\tf_2` directory to retain it.

```
C:\> virtualenv --system-site-packages -p python3 ./
    tf_2
```

Activate the virtual environment:

```
C:\> .\tf_2\Scripts\activate
```

Install packages within a virtual environment without affecting the host system setup. Start by upgrading pip:

```
(tf_2) C:\> pip install --upgrade pip
(tf_2) C:\> pip list  # show packages installed
    within the virtual environment
```

After the activation in any system, the terminal will change to this `(tf_2) $`.

Install TensorFlow

Upon reaching this stage, TensorFlow installation is a single line.

```
(tf_2) $ pip install --upgrade tensorflow
```

For a GPU or any other version of TensorFlow replace `tensorflow` in this listing with one of the following.

- `tensorflow` âĂŤLatest stable release (2.x) for CPU-only (recommended for beginners).
- `tensorflow-gpu` âĂŤLatest stable release with GPU support (Ubuntu and Windows).
- `tf-nightly` âĂŤPreview build (unstable). Ubuntu and Windows include GPU support.

3.1.3 Testing

Quick test

An instant test for the installation through the terminal is,

```
(tf_2) $ python -c "import tensorflow as tf; x =
    [[2.]]; print('tensorflow version', tf.
    __version__); print('hello, {}'.format(tf.matmul
    (x, x)))"
tensorflow version 2.0.0
hello, [[4.]]
```

The output should have the TensorFlow version and a simple operation output as shown here.

Modeling test

More elaborate testing is done by modeling a simple deep learning model with MNIST (`fashion_mnist`) image data.

```
import tensorflow as tf
layers = tf.keras.layers
import numpy as np
print(tf.__version__)
```

The `tf.__version__` should output `tensorflow 2.x`. If the version is older, check the installation, or the virtual environment should be revisited.

In the following, the `fashion_mnist` data is loaded from the TensorFlow library and preprocessed.

```
mnist = tf.keras.datasets.fashion_mnist
(x_train, y_train), (x_test, y_test) = mnist.
    load_data()
x_train, x_test = x_train / 255.0, x_test / 255.0
```

A simple deep learning model is now built and trained on the data.

```
model = tf.keras.Sequential()
model.add(layers.Flatten())
model.add(layers.Dense(64, activation='relu'))
```

```
model.add(layers.Dense(64, activation='relu'))
model.add(layers.Dense(10, activation='softmax'))
model.compile(optimizer='adam',
 loss='sparse_categorical_crossentropy',
 metrics=['accuracy'])
model.fit(x_train, y_train, epochs=5)
```

Note that this model is only for demonstration and, therefore, trained on just five epochs.

Lastly, the model is evaluated on a test sample.

```
predictions = model.predict(x_test)
predicted_label = class_names[np.argmax(predictions
    [0])]
print('Actual label:', class_names[y_test[0]])
print('Predicted label:', predicted_label)
# Actual label: Ankle boot
# Predicted label: Ankle boot
```

The prediction output confirms a complete installation.

3.2 Sheet Break Problem Dataset

An anonymized data from the paper manufacturing plant for sheet break is taken from link[2] provided in Ranjan et al. 2018.

It contains data at two minutes frequency with the time information present in the `DateTime` column. The system's status with regards to *normal* versus *break* is present in the `SheetBreak` column with the corresponding values as 0 and 1.

These remaining columns include timestamped measurements for the predictors. The predictors include raw materials, such as the amount of pulp fiber, chemicals, etc., and the process variables, such as blade type, couch vacuum, rotor speed, etc.

The steps for loading and preparing this data is shown in Listing 3.2 below.

Listing 3.2. Loading data.

[2] `http://bit.ly/2uCIJpG`

	DateTime	SheetBreak	RSashScanAvg	CT#1 BLADE PSI	P4 CT#2 BLADE PSI	Bleached GWD Flow	ShwerTemp	BlndStckFloTPD	C1 BW SPREAD CD	RS BW SPREAD CD	...	1PrsTopSpd	4PrsBotSpd	WtN
0	5/1/99 0:00	0	0.376665	-4.596435	-4.095756	13.497687	-0.118830	-20.669883	0.000732	-0.061114	...	10.091721	0.053279	-4
1	5/1/99 0:02	0	0.475720	-4.542502	-4.018359	16.230658	-0.128733	-18.758079	0.000732	-0.061114	...	10.095871	0.062801	-4
2	5/1/99 0:04	0	0.363848	-4.681394	-4.353147	14.127997	-0.138636	-17.836632	0.010803	-0.061114	...	10.100265	0.072322	-4
3	5/1/99 0:06	0	0.301590	-4.758934	-4.023612	13.161566	-0.148142	-18.517601	0.002075	-0.061114	...	10.104660	0.081600	-4
4	5/1/99 0:08	0	0.265578	-4.749928	-4.333150	15.267340	-0.155314	-17.505913	0.000732	-0.061114	...	10.109054	0.091121	-4

5 rows × 63 columns

Figure 3.1. *A snapshot of the sheet-break data.*

```
1  import pandas as pd
2
3  df = pd.read_csv("data/processminer-sheet-break-rare
       -event-dataset.csv")
4  df.head(n=5)  # visualize the data.
5
6  # Hot encoding
7  hotencoding1 = pd.get_dummies(df['Grade&Bwt'])
8  hotencoding1 = hotencoding1.add_prefix('grade_')
9  hotencoding2 = pd.get_dummies(df['EventPress'])
10 hotencoding2 = hotencoding2.add_prefix('eventpress_'
       )
11
12 df = df.drop(['Grade&Bwt', 'EventPress'], axis=1)
13
14 df = pd.concat([df, hotencoding1, hotencoding2],
       axis=1)
15
16 # Rename response column name for ease of
       understanding
17 df = df.rename(columns={'SheetBreak': 'y'})
```

A snapshot of the data is shown in Figure 3.1.

The data contains two categorical features which are hot-encoded in Line 7-10 followed by dropping the original variables. The binary response variable `SheetBreak` is renamed to `y`.

The processed data is stored in a `pandas` data frame `df`. This will be further processed with *curve shifting* and *temporalization* for modeling

in the next chapters.

Deactivate Virtual Environment

Before closing, deactivate the virtual environment.

Mac/Ubuntu

```
(tf_2) C:\> deactivate
```

Windows

```
(tf_2) $ deactivate  # don't exit until you're done
    using TensorFlow
```

Conda

```
(tf_2) $ source deactivate
```

or,

```
(tf_2) $ source deactivate
```

This concludes the TensorFlow setup. Now we will head to model building.

Chapter 4

Multi-layer Perceptrons

4.1 Background

Perceptrons were built in the 1950s. And they proved to be a powerful classifier at the time.

A few decades later, researchers realized stacking multiple perceptrons could be more powerful. That turned out to be true and multi-layer perceptron (MLP) was born.

A single perceptron works like a neuron in a human brain. It takes multiple inputs and, like a neuron emits an electric pulse, a perceptron emits a binary pulse which is treated as a response.

The **neuron**-like behavior of perceptrons and an MLP being a **network** of perceptrons perhaps led to the term *neural networks* come forth in the early days.

Since its creation, neural networks have come a long way. Tremendous advancements in various architectures, such as convolutional neural networks (CNN), recurrent neural networks (RNN), etc., have been made.

Despite all the advancements, MLPs are still actively used. It is the "hello world" to deep learning. Similar to *linear regression* in machine learning, MLP is one of the immortal methods that remain active due to its robustness.

It is, therefore, logical to start exploring deep learning with multi-layer perceptrons.

Multi-layer perceptrons are complex nonlinear models. This chapter unfolds MLPs to simplify and explain its fundamentals in § 4.2. The section shows that an MLP is a collection of simple regression models placed on every node in each layer. How they come together with non-linear activations to deconstruct and solve complex problems becomes clearer in this section.

An end-to-end construction of a network and its evaluation is then given in § 4.3-4.4. § 4.3 has granular details on data preparation, viz. curve shifting for early prediction, data splitting, and features scaling. Thereafter, every construction element, e.g., layers, activations, evaluation metrics, and optimizers, are explained in § 4.4.

Dropout is a useful technique (not limited to multi-layer perceptrons) that resolves **co-adaptation** issue in deep learning. The co-adaptation issue is explained in § 4.5.1. How dropout addresses it and regularizes a network is in § 4.5.2. The use of dropout in a network is then illustrated in § 4.5.3.

Activation functions elucidated in § 4.7 are one of the most critical constructs in deep learning. Network performances are usually sensitive to activations due to their vanishing or exploding gradients. An understanding of activations is, therefore, essential for constructing any network. § 4.7.1 and 4.7.2 explain vanishing and exploding gradients issues, and their connection with activations. The story of activations laying discoveries such as non-decaying gradient, saturation region, and self-normalization is in § 4.7.3 and 4.7.4.

Besides, a few customizations in TensorFlow implementations are sometimes required to attempt novel ideas. § 4.8.1 shows an implementation of a new *thresholded exponential linear unit* (`telu`) activation. Moreover, metrics such as f1-score and false positive rate useful for evaluating imbalanced classifiers are unavailable in TensorFlow. They are custom implemented in § 4.8.2. This section also clarifies that metrics available outside TensorFlow such as in `sklearn` cannot be directly used during model training.

Lastly, deep learning networks have several configurations and nu-

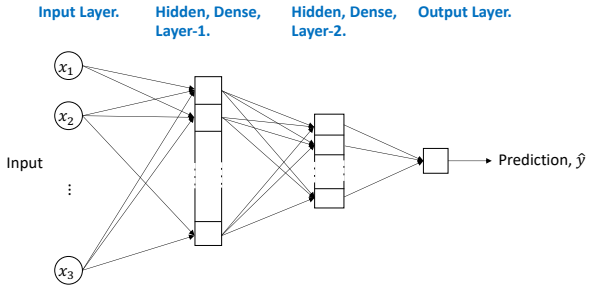


Figure 4.1. *A high-level representation of a multi-layer perceptron.*

merous choices for them, e.g., number of layers, their sizes, activations on them, and so on. To make a construction simpler, the chapter concludes with a few rules-of-thumb in § 4.10.

4.2 Fundamentals of MLP

Multi-layer perceptrons are possibly one of the most visually illustrated neural networks. Yet most of them lack a few fundamental explanations. Since MLPs are the foundation of deep learning, this section attempts at providing a clearer perspective.

A typical visual representation of an MLP is shown in Figure 4.1. This high-level representation shows the feed-forward nature of the network. In a feed-forward network, information between layers flows in only forward direction. That is, information (features) learned at a layer is not shared with any prior layer[1].

The abstracted network illustration in Figure 4.1 is unwrapped to its elements in Figure 4.2. Each element, its interactions, and implementation in the context of TensorFlow are explained step-by-step in the following.

1. The process starts with a data set. Suppose there is a data set

[1]Think of a bi-directional LSTM discussed in § 5.6.4 as a counter example to a feed-forward network.

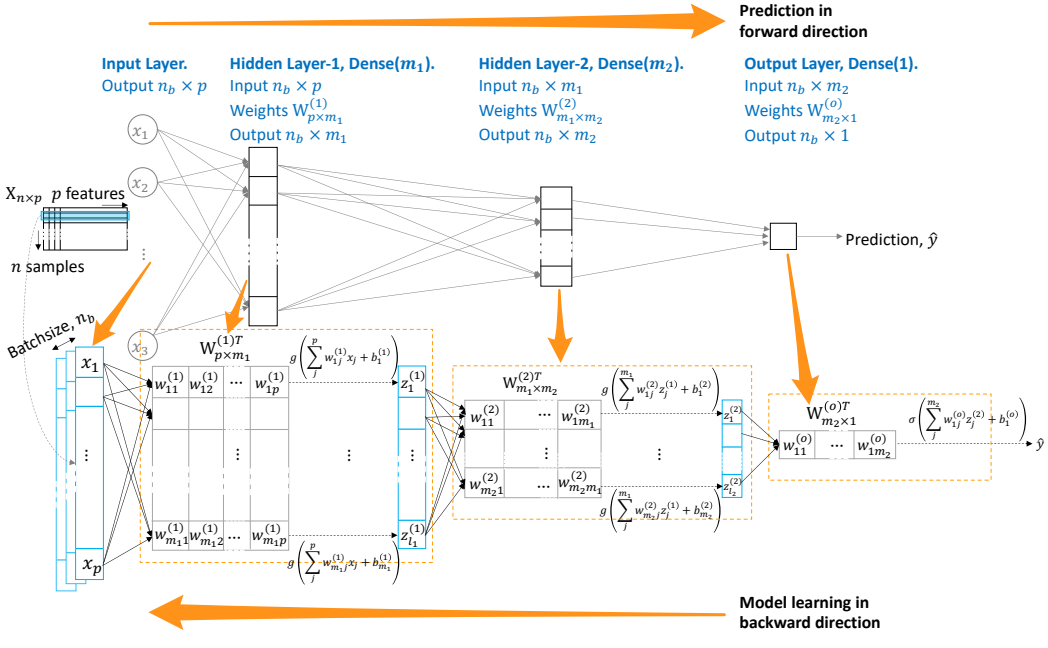


Figure 4.2. *An unwrapped visual of a multi-layer perceptron. The input to the network is a batch of samples. Each sample is a feature vector. The hidden layers in an MLP are* `Dense`*. A* `Dense` *layer is characterized by a weight matrix W and bias b. They perform simple affine transformations (dot product plus bias: $XW + b$). The affine transforms extract features from the input. The transforms are passed through an activation function. The activations are nonlinear. Its nonlinearity enables the network to implicitly divide a complex problem into arbitrary sub-problems. The outputs of these sub-problems are put together by the network to infer the final output $\hat{y}$.*

shown at the top left, $X_{n \times p}$, with n samples and p features.

2. The model ingests a randomly selected batch during training. The batch contains random samples (rows) from X unless otherwise mentioned. The batch size is denoted as n_b here[2].

3. By default, the samples in a batch are processed independently. Their sequence is, therefore, not important.

4. The input batch enters the network through an input layer. Each node in the input layer corresponds to a sample feature. Explicitly defining the input layer is optional but it is done here for clarity.

5. The input layer is followed by a stack of hidden layers till the last (output) layer. These layers perform the "complex" interconnected nonlinear operations. Although perceived as "complex," the underlying operations are rather simple arithmetic computations.

6. A hidden layer is a stack of computing nodes. Each node extracts a feature from the input. For example, in the sheet-break problem, a node at a hidden-layer might determine whether the rotations between two specific rollers are out-of-sync or not[3]. A node can, therefore, be imagined as solving one arbitrary sub-problem.

7. The stack of output coming from a layer's nodes is called a *feature map* or *representation*. The size of the feature map, also equal to the number of nodes, is called the layer size.

8. Intuitively, this feature map has results of various "sub-problems" solved at each node. They provide predictive information for the next layer up until the output layer to ultimately predict the response.

9. Mathematically, a node is a perceptron made of weights and bias parameters. The weights at a node are denoted with a vector $\boldsymbol{w}$ and a bias b.

[2]Batch size is referred to as `None` or `?` in `model.summary()`, e.g., see Figure 4.4.

[3]Another example from the face-recognition problem about solving an arbitrary sub-problem at a node is determining whether eyebrows are present or not.

10. All the input sample features go to a node. The input to the first hidden layer is the input data features $\boldsymbol{x} = \{x_1, \ldots, x_p\}$. For any intermediate layer it is the output (feature map) of the previous layer, denoted as $\boldsymbol{z} = \{z_1, \ldots, z_m\}$, where m is the size of the prior layer.

11. Consider a hidden layer l of size m_l in the figure. A node j in the layer l performs a feature extraction with a dot product between the input feature map $\boldsymbol{z}^{(l-1)}$ and its weights $\boldsymbol{w}_j^{(l)}$, followed by an addition with the bias b_j. Generalizing this as,

$$z_j^{(l)} = \sum_i^{m_{l-1}} z_i^{(l-1)} w_{ij}^{(l)} + b_j, \; j = 1, \ldots, m_l \tag{4.1}$$

where $z_i^{(l-1)}, i = 1, \ldots, m_{l-1}$ is a feature outputted from the prior layer $l-1$ of size m_{l-1}.

12. The step after the linear operation in Equation 4.1 is applying a nonlinear *activation* function, denoted as g. There are various choices for g. Among them, a popular activation function is *rectified linear unit* (`relu`) defined as,

$$g(z) = \begin{cases} z, & \text{if } z > 0. \\ 0, & \text{otherwise.} \end{cases} \tag{4.2}$$

As shown in the equation, the function is nonlinear at 0.

13. The operations in Equation 4.1 and 4.2 can be combined for every nodes in a layer as,

$$\boldsymbol{z}^{(l)} = g(\boldsymbol{z}^{(l-1)} W^{(l)} + \boldsymbol{b}^{(l)}) \tag{4.3}$$

where $\boldsymbol{z}^{(l)}$ is the feature map, $W^{(l)} = [\boldsymbol{w}_1^{(l)}; \ldots; \boldsymbol{w}_{m_l}^{(l)}]$ is the stack of weights of all m_l nodes in the layer, $\boldsymbol{z}^{(l-1)}$ is the input to the layer which is $\boldsymbol{x}$, if $l = 1$, and $\boldsymbol{b}^{(l)} = \{b_1^{(l)}, \ldots, b_{m_l}^{(l)}\}$ is the bias.

14. Equation 4.3 is applied to the batch of n_b input samples. For a `Dense` layer this is a matrix operation shown in Equation 4.4 below,

$$Z^{(l)}_{n_b \times m_l} = g(Z^{(l-1)}_{n_b \times m_{l-1}} W^{(l)}_{m_{l-1} \times m_l} + \boldsymbol{b}^{(l)}_{m_l}). \tag{4.4}$$

The output $Z^{(l)}_{n_b \times m_l}$ of the equation is the g-activated *affine* transformation of the input features.

The operation here is called a `tensor` *operation. Tensor is a term used for any multi-dimensional matrix. Tensor operations are computationally efficient (especially in GPUs), hence, most steps in deep learning layers use them instead of iterative loops.*

15. It is the nonlinear activation in Equation 4.4 that **dissociates** the feature map of one layer from another. Without the activation, the feature map outputted from every layer will be just a **linear** transformation of the previous. This would mean the subsequent layers are **not** providing any additional information for a better prediction. An algebraic explanation of this is given in Equation A.1 in the Appendix.

16. Activation functions, therefore, play a **major** role. An appropriate selection of activation is critical. In addition to being nonlinear, an activation function should also have a non-decaying gradient, saturation region, and a few other criteria discussed in § 4.7.

17. The operation in Equation 4.4 is carried forward in each layer till the output layer to deliver a prediction. The output is delivered through a different activation denoted as σ. The choice of this activation is dictated by the response type. For example, it is *sigmoid* for a binary response.

18. The model training starts with randomly initializing the weights and biases at the layers. The response is predicted using these

parameters. The prediction error is then propagated back into the model to update the parameters to a value that reduces the error. This iterative training procedure is called *backpropagation*.

19. Put simply, backpropagation is an **extension** of the iterative stochastic gradient-descent based approach to train multi-layer deep learning networks. This is explained using a single-layer perceptron, also otherwise known as, **logistic regression**. A gradient-descent based estimation approach for logistic regression is shown in p.120-121 in Hastie, Tibshirani, and Friedman 2009[4]. The estimation equation in the reference[5] is rephrased to a simplified context here,

$$\begin{aligned} \theta^{new} &\leftarrow \theta^{old} - \eta \nabla_\theta \\ &\leftarrow \theta^{old} - \eta X^T (y - \hat{y}) \end{aligned} \tag{4.5}$$

where η is a multiplier[6], X is a random sample, ∇_θ is the first-order gradient of the loss with respect to the parameter θ, and θ is the weight and bias parameters. As shown, the gradient ∇_θ for the logistic regression contains $(y - \hat{y})$ which is the **prediction error**. This implies that the prediction error is propagated back to update the model parameters θ.

20. This estimation approach for logistic regression is extended in *backpropagation* for a multi-layer perceptron. In backpropagation, this process is repeated on every layer. It can be imagined as updating/learning one layer at a time in the reverse order of prediction.

21. The learning is done iteratively over a user-defined number of epochs. An epoch is a learning period. Within each epoch, the stochastic gradient-descent based learning is performed iteratively over randomly selected batches.

[4]The approach in Hastie, Tibshirani, and Friedman 2009 also used the second-derivative or the Hessian matrix. However, in most backpropagation techniques only first derivatives are used.

[5]Equation 4.26 in Hastie, Tibshirani, and Friedman 2009.

[6]Equal to the Hessian in Newton-Raphson algorithm.

22. After training through all the epochs the model is expected to have learned the parameters that have minimal prediction error. This minimization is, however, for the training data and is not guaranteed to be the global minima. Consequently, the performance of the test data is not necessarily the same.

The fundamentals enumerated above will be referred to during the modeling in the rest of the chapter.

Deep learning models are powerful because it breaks down a problem into smaller sub-problems and combines their results to arrive at the overall solution. This fundamental ability is due to the presence of the nonlinear activations.

Intuitively, ***backpropagation*** *is an extension of stochastic gradient-descent based approach to train deep learning networks.*

The next section starts the preparation for modeling.

4.3 Initialization and Data Preparation

4.3.1 Imports and Loading Data

Modeling starts with the ritualistic library imports. Listing 4.1 shows all the imports and also a few declarations of constants, viz. random generator seeds, the data split percent, and the size of figures to be plotted later.

Listing 4.1. Imports for MLP Modeling.

```
1 import tensorflow as tf
2
3 from tensorflow.keras import optimizers
```

```
4  from tensorflow.keras.models import Model
5  from tensorflow.keras.models import Sequential
6  from tensorflow.keras.layers import Input
7  from tensorflow.keras.layers import Dense
8  from tensorflow.keras.layers import Dropout
9  from tensorflow.keras.layers import AlphaDropout
10
11 import pandas as pd
12 import numpy as np
13
14 from sklearn.preprocessing import StandardScaler
15 from sklearn.model_selection import train_test_split
16
17 from imblearn.over_sampling import SMOTE
18 from collections import Counter
19
20 import matplotlib.pyplot as plt
21 import seaborn as sns
22
23 # user-defined libraries
24 import datapreprocessing
25 import performancemetrics
26 import simpleplots
27
28 from numpy.random import seed
29 seed(1)
30
31 from pylab import rcParams
32 rcParams['figure.figsize'] = 8, 6
33
34 SEED = 123 #used to help randomly select the data
       points
35 DATA_SPLIT_PCT = 0.2
```

A few user-defined libraries: `datapreprocessing`, `performancemetrics`, and `simpleplots` are loaded. They have custom functions for preprocessing, evaluating models, and visualizing results, respectively. These libraries are elaborated in this and upcoming chapters (also refer to Appendix B and C).

Next, the data is loaded and processed the same way as in Listing 3.2 in the previous chapter. The listing is repeated here to avoid

any confusion.

Listing 4.2. Loading data for MLP Modeling.

```
# Read the data
df = pd.read_csv("data/processminer-sheet-break-rare
    -event-dataset.csv")

# Convert Categorical column to hot dummy columns
hotencoding1 = pd.get_dummies(df['Grade&Bwt'])
hotencoding1 = hotencoding1.add_prefix('grade_')
hotencoding2 = pd.get_dummies(df['EventPress'])
hotencoding2 = hotencoding2.add_prefix('eventpress_'
    )

df=df.drop(['Grade&Bwt', 'EventPress'], axis=1)

df=pd.concat([df, hotencoding1, hotencoding2], axis
    =1)

# Rename response column name for ease of
    understanding
df=df.rename(columns={'SheetBreak':'y'})
```

4.3.2 Data Pre-processing

The objective, as mentioned in Chapter 2, is to predict a rare event in advance to prevent it, or its consequences.

From a modeling standpoint, this translates to teaching the model to identify a *transitional* phase that would lead to a rare event.

For example, in the sheet-break problem, a transitional phase could be the speed of one of the rollers (in Figure 1.1) drifting away and rising in comparison to the other rollers. Such asynchronous change stretches the paper sheet. If this continues, the sheet's tension increases and ultimately causes a break.

The sheet break would typically happen a few minutes after the drift starts. Therefore, if the model is taught to identify the start of the drift it can predict the break in advance.

One simple and effective approach to achieve this is *curve shifting*.

Curve Shifting

Curve Shifting here should not be confused with *curve shift* in Economics or *covariate shift* in Machine Learning. In Economics, a curve shift is a phenomenon of the Demand Curve changing without any price change. Covariate shift or data shift in ML implies a change in data distribution due to a shift in the process. Here it means aligning the predictors with the response to meet a certain modeling objective.

For early prediction, curve shifting moves the labels early in time. Doing so, the samples before the rare event get labeled as one. These prior samples are assumed to be the transitional phase that ultimately leads to the rare event.

Providing a model with these positively labeled transitional samples teaches it to identify the "harbinger" of a rare event in time. This, in effect, is an early prediction.

For early prediction, teach the model to identify the transitional phase.

Given the time series sample, $(y_t, \boldsymbol{x}_t), t = 1, 2, \ldots$, curve shifting will,

1. label the k prior samples to a positive sample as one. That is, $y_{t-1}, \ldots, y_{t-k} \leftarrow 1$, if $y_t = 1$.

2. Followed by dropping the sample $(y_t, \boldsymbol{x}_t)$ for the $y_t = 1$ instance.

Due to the relabeling in the first bullet, the model learns to predict the rare-event up to $t + k$ times earlier.

Besides, the curve shifting drops the original positive sample to avoid teaching the model to predict the rare event when it has already happened. Referring to Figure 2.2, the signal is highest when the event has occurred. Keeping these samples in the training data will be overpowering on the transitional samples. Due to this, it is likely that the model does not learn predicting the event ahead.

The `curve_shift` function in the `datapreprocessing` library performs this operation. It labels the samples adjacent to the positive labels as one. The number of adjacent samples to label is equal to the

argument `shift_by`. A negative `shift_by` relabels the preceding samples[7]. Moreover, the function drops the original positive sample after the relabeling.

Listing 4.3. Curve-shifting.

```
1  # Sort by time.
2  df['DateTime'] = pd.to_datetime(df.DateTime)
3  df = df.sort_values(by='DateTime')
4
5  # Shift the response column y by 2 rows to do a 4-
       min ahead prediction.
6  df = datapreprocessing.curve_shift(df, shift_by=-2)
7
8  # Drop the time column.
9  df = df.drop(['DateTime'], axis=1)
10
11 # Converts df to numpy array
12 X = df.loc[:, df.columns != 'y'].values
13 y = df['y'].values
14
15 # Axes lengths
16 N_FEATURES = X.shape[1]
```

Line 6 in Listing 4.3 applies the curve shift with `shift_by=-2`. This relabels two samples prior to a sheet break as positive, i.e., the transitional phase leading to a break. Since the samples are at two minutes interval, this shift is of four minutes. Thus, the model trained on this curve-shifted data can do up to 4-minute ahead sheet break prediction.

While this is reasonable for this problem, the requirements could be different for different problems. The `shift_by` parameter should be set accordingly. Furthermore, for advanced readers, the curve shift definition is given in Appendix B for details and customization, if needed.

The effect of the curve shifting is visualized in Figure 4.3. The figure shows sample 259 is originally a positive sample. After applying `curve_shift` with `shift_by=-2`, the preceding two samples 257-258 are relabeled as `1`. And, the original positive sample 259 is dropped.

Thereafter, the `DateTime` column is not needed and, therefore, dropped.

[7] A positive `shift_by` relabels the succeeding samples to $y_t = 1$ as one.

Before shifting

	DateTime	y	RSashScanAvg	CT#1 BLADE PSI	P4 CT#2 BLADE PSI
256	5/1/99 8:32	0	1.016235	-4.058394	-1.097158
257	5/1/99 8:34	0	1.005602	-3.876199	-1.074373
258	5/1/99 8:36	0	0.933933	-3.868467	-1.249954
259	5/1/99 8:38	1	0.892311	-13.332664	-10.006578
260	5/1/99 10:50	0	0.020062	-3.987897	-1.248529

After shifting

	y	DateTime	RSashScanAvg	CT#1 BLADE PSI	P4 CT#2 BLADE PSI
255	0.0	5/1/99 8:30	0.997107	-3.865720	-1.133779
256	0.0	5/1/99 8:32	1.016235	-4.058394	-1.097158
257	1.0	5/1/99 8:34	1.005602	-3.876199	-1.074373
258	1.0	5/1/99 8:36	0.933933	-3.868467	-1.249954
260	0.0	5/1/99 10:50	0.020062	-3.987897	-1.248529

Figure 4.3. *An illustration of Curve Shifting. In this example, a dataframe is curve shifted (ahead) by two time units (`curve_shift(df, shift_by=-2)`). After the shift, the updated dataframe has the labels of two prior samples to a positive sample updated to `1` and the original positive sample is dropped. This procedure can also be interpreted as treating the samples prior to a positive sample as transitional phases. Using `curve_shift()`, these transitional samples are re-labeled as positives.*

The dataframe is partitioned into the features array `X` and the response `y` in lines 12-13. Lastly, the shape of the features array is recorded in `N_FEATURES`. This becomes a global constant that will be used in defining the `input_shape` during modeling.

Data Splitting

The importance of splitting a data set into *train*, *valid*, and *test* sets is well known. It is a necessary modeling tradition for the right reasons, which are briefly described below.

With the split data,

1. a model is trained on the *train* set, and
2. the model's performance is validated on the *valid* set.
3. Steps 1-2 are repeated for a variety of models and/or model configurations. The one yielding the best performance on the *valid* set is chosen as the final model. The performance of the final model on the *test* set is then recorded.

The *test* set performance is a "robust" indicator. While the *train* set performance is unusable due to a usual overfitting, the *valid* set is used for model selection and, therefore, is biased towards the selected model. Consequently, only the *test* set performance gives a reliable estimate.

In Listing 4.4, the data is randomly split into a *train*, *valid*, and *test* making 64%, 16%, and 20% of the original data set, respectively.

Listing 4.4. Data splitting.

```
1  # Divide the data into train, valid, and test
2  X_train, X_test, y_train, y_test =
3      train_test_split(np.array(X),
4                       np.array(y),
5                       test_size=DATA_SPLIT_PCT,
6                       random_state=SEED)
7  X_train, X_valid, y_train, y_valid =
8      train_test_split(X_train,
9                       y_train,
10                      test_size=DATA_SPLIT_PCT,
11                      random_state=SEED)
```

A common question in splitting a time series is: should it be at random or in the order of time?

Splitting time series in the order of time is appropriate when,

1. the data set is arranged as time-indexed tuples ($response : y_t$, $features : \boldsymbol{x}_t$). And,

2. the model has temporal relationships, such as $y_t \sim y_{t-1} + \ldots + x_t + x_{t-1} + \ldots$. That is, the y_t is a function of the prior y's and x's.

In such a model, maintaining the time order is necessary for the data splits.

However, suppose the temporal (time-related) features are included in the tuple as ($response : y_t$, $features : \{y_{t-1}, \ldots, \boldsymbol{x}_t, \boldsymbol{x}_{t-1}, \ldots\}$). This makes a self-contained sample with the response and (temporal) features. With this, the model does not require to keep track of the time index of the samples.

For multivariate time series, it is recommended to prepare such self-contained samples. In MLP, the temporal features can be added during the data preprocessing. But learning the temporal patterns are left to recurrent and convolutional neural networks in the upcoming chapters where temporally contained samples are used.

Here, the data samples are modeled in their original form $(y_t, \boldsymbol{x}_t)$, where y_t is one in the transitional phase and zero, otherwise. No additional time-related features are to be modeled. Therefore, the model is agnostic to the time order. And, hence, the data set can be split at random.

Features Scaling

Virtually every problem has more than one feature. The features can have a different range of values. For example, a paper manufacturing process has temperature and moisture features. Their units are different due to which their values are in different ranges.

These differences may not pose theoretical issues. But, in practice, they cause difficulty in model training typically by converging at local

minimas.

Feature scaling is, therefore, an important preprocessing step to address this issue. Scaling is generally linear[8]. Among the choices of linear scaling functions, the standard scaler shown in Listing 4.5 is appropriate for the unbounded features in our problem.

Listing 4.5. Features Scaling.

```
# Scaler using the training data.
scaler = StandardScaler().fit(X_train)

X_train_scaled = scaler.transform(X_train, scaler)
X_valid_scaled = scaler.transform(X_valid, scaler)
X_test_scaled = scaler.transform(X_test, scaler)
```

As shown in the listing, the `StandardScaler` is fitted on the *train* set. The fitting process computes the means ($\bar{\boldsymbol{x}}_{train}$) and standard deviation ($\boldsymbol{\sigma}_{train}$) from the *train* samples for each feature in $\boldsymbol{x}$. This is used to transform each of the sets as per Equation 4.6.

$$\boldsymbol{x} \leftarrow \frac{\boldsymbol{x} - \bar{\boldsymbol{x}}_{train}}{\boldsymbol{\sigma}_{train}} \tag{4.6}$$

Another popular scaling method is `MinMaxScaler`. However, they work better in and became more popular through the image problems in deep learning. A feature in an image has values in a fixed range of $(0, 255)$. For such bounded features, `MinMaxScaler` is quite appropriate.

`StandardScaler` is appropriate for unbounded features such as sensor data. However, `MinMaxScaler` is better for bounded features such as in image detection.

[8]There are a few nonlinear feature scaling methods to deal with feature outliers. However, it is usually recommended to deal with the outliers separately and work with linear scaling to not disturb the original distribution of the data.

4.4 MLP Modeling

In this section, a multi-layer perceptron model is constructed step-by-step. Each modeling step is also elucidated conceptually and programmatically.

4.4.1 Sequential

TensorFlow provides a simple-to-implement API for constructing deep learning models. There are three general approaches,

- sequential,
- functional, and
- model sub-classing.

The ease of their use is in the same order. Most of the modeling requirements are covered by *sequential* and *functional*.

Sequential is the simplest approach. In this approach, models that have a linear stack of layers and the layers communicate sequentially are constructed. Models in which layers communicate non-sequentially (for example, residual networks) cannot be modeled with a sequential approach. Functional or model sub-classing is used in such cases.

MLPs are sequential models. Therefore, a sequential model is initialized as shown in Listing 4.6.

Listing 4.6. Creating a Sequential object.

```
model = Sequential()
```

The initialization here is creating a `Sequential` object. `Sequential` inherits the `Model` class in TensorFlow, and thereby, inherits all the training and inference features.

4.4.2 Input Layer

The model starts with an input layer. No computation is performed at this layer. Still, this plays an important role.