PART ONE

Fundamentals

The first part of the book focuses on domain-independent techniques and overall process. While the importance of domain expertise tends to be highlighted in machine learning tasks, many times careful data analysis can steer us away from bad assumptions and yield high-performing models without domain expertise.

Whether quality feature engineering (FE) can be achieved without tapping into deep domain expertise is a topic of debate in the field. In his talk "Data Agnosticism, Feature Engineering Without Domain Expertise" expert online competitions champion Nicholas Kridler[193] argues that "responsible data analysis and quick iterations produce high-performing predictive models" and that models will help us find features without domain expertise. We will explore this viewpoint in this part of the book.

Even if you lack domain expertise, do not underestimate the dataset expertise you will gain by working with it. In their book *Communicating with Today's Patient*, Desmond and Copeland[81] argue that in a doctor visit there are two experts at work: the expert in medicine (the doctor) and the expert on the patient's body (the patient themselves). If you follow the data analysis-centric process sketched in Chapter 1 and exemplified in Part Two, you will, over time, become an expert in your dataset, even if you do not know the domain.

1

Introduction

You have some data. You have trained a model. The results are below of what you need. You believe more work should help. Now what? You can try to improve the model. You can try to collect more data. Both are good avenues for improved results. A third avenue is to modify the features to better capture the nature of your problem. This process, feature engineering (FE), is partly an art and partly a palette of tricks and recipes. In the following chapters, I hope to expand your palette with new ideas to improve the performance of your machine learning solution.

To understand the importance of FE, I would like to draw an analogy to the techniques for solving word problems covered in mathematics textbooks. Take the following problem:

> A dog runs at 10 mph back and forth between two spouses while they run into each other for 100 feet at 5 mph. What is the total distance the dog runs?

Depending on the framing, solving this problem requires an integral (adding all the distances run by the dog) or elementary school arithmetic (calculating the time it takes the spouses to meet and the distance travelled at the dog speed for that period of time). The importance of framing is easy to overlook and hard to teach. Machine learning (referred as ML throughout this book) encounters a similar situation: most ML algorithms take a representation of the reality as vectors of "features," which are aspects of the reality over which the algorithm operates. First, choosing the right representation is key. Second, sometimes the features can be preprocessed outside of the algorithm, incorporating insights from the problem domain to better solve it. This type of operations, FE, tends to bring out performance gains beyond tweaking the algorithms themselves. This book is about these techniques and approaches.

Book Structure. This book is structured into two parts. In the first part, I[†] have sought to present FE ideas and approaches that are as much domain independent as FE can possibly be. The second part exemplifies the different techniques as used in key domains (graph data, time series, text processing, computer vision and others) through case studies. All the code and data for these case studies is available under open-source licenses at

`http://artoffeatureengineering.com`

This chapter covers definitions and processes. The key to FE is expanding the ML cycle (Section 1.3.1) to accommodate FE (Section 1.3.2) and, among other things, to include a data release schedule to avoid overfitting, a matter of evaluation (Section 1.2). Two types of analysis are central to this cycle, one to be done before the ML starts (Exploratory Data Analysis, Section 1.4.1) and another one after one ML cycle has concluded (Error Analysis, Section 1.4.2), which will inform the next steps in your FE process. Then we will look into two other processes related to FE: domain modelling, which helps with feature ideation (Section 1.5.1) that then results in different techniques for feature construction (Section 1.5.2). The chapter concludes with general discussions about FE particularly where it falls with respect to hyperparameter fitting and when and why to engage in a FE process (Section 1.6).

Chapter 2 discusses FE techniques that modify the features based on their behaviour as a whole. Techniques such as normalization, scaling, dealing with outliers and generating descriptive features are covered. Chapter 3 deals with the topic of feature expansion and imputation with an emphasis on computable features. Chapter 4 presents a staple of FE: the automatic reduction of features, either by pruning or by projection onto a smaller feature space. Chapter 5 concludes Part One by presenting advanced topics, including dealing with variable-length feature vectors, FE for deep learning (DL) and automatic FE (either supervised or unsupervised).

Part Two presents case studies on domains where FE is well understood and common practice. Studying these techniques can help readers working on new domains where the domain lacks such maturity. Neither of the case studies is to be taken as comprehensive instructional material on the domain; you would be better served by specific books on each domain, some of which are mentioned at the end of each chapter. Instead, the case studies are intended to help you brainstorm ideas for FE in your domain. As such, the nomenclature used might

[†] FE has plenty of topics still left open for debate. I have tried to separate my opinions from more established topics by using first person singular in cases where I felt you might want to take my comments with extra care. The use of first person singular is not to be less welcoming, it is to warn you to be specially critical of what you are about to read.

differ slightly from what is usual for each domain. A contribution of this book is also a dataset shared by the first four chapters specifically built to teach FE, which contains graph data, textual data, image data and timestamped data. The task is that of predicting the population of 80,000 cities and towns around the world based on different available data, and it is described in detail in Chapter 6. The domains studied are graph data, timestamped data (Chapter 7), textual data (Chapter 8), image data (Chapter 9) and other domains in Chapter 10, including video, geographical data and preference data. The chapters refer to accompanying source code implemented as Python notebooks; however, studying the code is not required to understand or follow the case studies.

How to Read this Book. This book has been written with practitioners in mind, people who have already trained models over data. With such readers in mind, there are two different situations this book can help you with:

You want to get better at FE. You have done some light FE and felt your efforts were lacking. A full light reading of Part One will give you fresh ideas of things to try. Pay special attention to the cycle proposed in Section 1.3.2 in this chapter and see if it makes sense to you. You could adapt it or develop your own cycle. It is good to have a process when doing FE; this way you can decide when to stop and how to allocate efforts and evaluation data. Then, move to Part Two and tear apart the case studies. The work presented in Part Two is intended to get the conversation started; your own opinions on the data and domains should give you plenty of ideas and criticism. I disagree with many of my own decisions in each of these case studies, as shown in the postmortems. Enticing you to come up with better ways to approach the case studies could be your fastest route to excel at FE. Hopefully, you will feel energized to give your ideas a try on the datasets and code released with the book.

You have a dataset and problem and need help with FE for it. This requires more precise reading of specific sections. If your domain is structured, approach the case study in Chapter 6 and read the linked material in Part One as needed. If your domain is sensor data, look at Chapter 9. If it is discrete data, look at Chapter 8. If it has a time component, look at Chapter 7. Alternatively, if you have too many features, look at Chapter 4. If you feel your features have a signal that is poorly captured by the ML algorithm, try a feature drill-down using the ideas in Chapter 3. If the relation of a feature value to the rest of the values and features might be important, look into Chapter 2. Finally, if you have variable length features, Section 5.1 in Chapter 5 can help you.

Background Expected from the Reader. ML practitioners come from a variety of backgrounds. The same can be said about ML researchers, which in turn means a wide variety of methods in existing work. There are many techniques that require advanced mathematics to understand them but not necessarily to use them. The explanations in these pages try to stay away from advanced topics as much as possible but the following subjects are assumed: knowledge of ML algorithms (decision trees, regression, neural networks, k-means clustering and others), knowledge of linear algebra (matrix inversion, eigenvalues and eigenvectors, matrix decomposition) and probability (correlation, covariance, independence). The last section of the chapter contains pointers to material suitable to cover these topics, if needed. Practitioners tend to be very strategic with their learning as their time is limited. Many of the techniques described in this part exceed this basic background. If the technique ends up in your critical path, references to all the source material are included in case you need to drill down deeper.

Throughout the book, I use the following abbreviations: ML for machine learning, NN for neural networks, IR for information retrieval, NLP for natural language processing and CV for computer vision.

1.1 Feature Engineering

The input to a supervised ML system is represented as a set of training examples called **instances**. Each instance in a classification or regression problem has a **target class**, or **target value,** which can be of discrete size (classification) or continuous (regression). This discussion refers to target class and classification but it also applies to target value and regression. Besides its target class, each instance contains a fixed-size vector of **features**, specific information about the instance that the practitioner doing ML expects will be useful for learning.

When approaching a ML problem, target classes and instances are usually given beforehand as part of the problem definition. They are part of what I call **raw data** (other authors use the term *variable*[134] or *attribute* vs. feature to make a similar distinction). Such raw data is normally the result of a data collection effort, sometimes through data collection hooks on a live system, with target classes obtained from the system or through human annotation (with suitable guidelines[262] and cross-annotator agreement[57]). Features themselves are not so clear cut, going from raw data to features involves extracting features following a **featurization** process (Section 1.5.2) on a **data pipeline**. This process goes hand in hand with **data cleaning** and enhancement.

Distinguishing raw data from features makes explicit the modelling decision involved in picking and assembling feature sets. If the raw data is tabular, each row can be an instance and there would be a temptation to consider each column a feature. However, deciding which columns are features and what type of preprocessing (including clustering, etc.) ought to be done on them to obtain features is a task closely tied to the problem sought to be solved. These decisions are better addressed through exploratory data analysis (Section 1.4.1), and featurization (Section 1.5.2). Therefore, a feature is defined as any value that can be computed from the raw data for the purpose of modelling the problem for a ML algorithm.[51] What makes good features and how to come up with them is discussed in the domain modelling section later in this chapter (Section 1.5.1).

The distinction between raw data and features is key and it enables the type of decisions behind successful FE. In the second part of the book, we will study examples where raw data includes graphs with hundreds of thousands of nodes, texts with millions of words and satellite images with hundreds of millions of pixels with features such as the average population of cities in a given country or whether the word "congestion" appears in a text.

Given these definitions, we are ready to define FE. The term means slightly different things for different people and I have not found an existing definition that captures the intuitions followed in this book. I therefore wrote my own definition, which follows, but beware the term might mean different things to other practitioners:

> Feature engineering is the process of representing a problem domain to make it amenable for learning techniques. This process involves the initial discovery of features and their stepwise improvement based on domain knowledge and the observed performance of a given ML algorithm over specific training data.

At its core, FE is a representation problem,[51] that is, it is the process of adjusting the representation of the data to improve the efficacy of the ML algorithms. It uses domain knowledge[195] and it might also use knowledge about the ML method itself. It is difficult, expensive and time consuming.

FE is referred to by many names, such as *data munging* or *data wrangling*,[49] or sometimes as a synonym of feature selection (which is of course limiting, as discussed in Section 4.1, in Chapter 4).

In the words of Jason Brownlee:[51]

> [Feature engineering] is an art like engineering is an art, like programming is an art, like medicine is an art. There are well defined procedures that are methodical, provable and understood.

I follow other authors[132] in considering FE as an encompassing term that includes feature generation (producing features from raw data), feature transformation (evolving existing features), feature selection (picking most important features), feature analysis (understanding feature behaviour), feature evaluation (determining feature importance) and automatic feature engineering methods (performing FE without human intervention). Note that in many circumstances, the term "feature engineering" will be used as a synonym for only one of such activities.

Examples of FE include normalizing features (Section 2.1 in Chapter 2), computing histograms (Section 2.3.1), using existing features to compute new ones (Section 3.1 in Chapter 3), imputing missing features (Section 3.2), selecting relevant features (Section 4.1 in Chapter 4), projecting the features into a smaller dimension (Section 4.3) and the rest of the techniques discussed in this book.

There is wide consensus in the field that FE is the place to add domain knowledge[49,341]; therefore, half this book describes FE in the context of domains where it is understood to be a key ingredient in learning. For example, realizing that certain feature values are not useful after a threshold (Section 6.4 in Chapter 6), or computing averages that take into account the cyclic nature of the data (Section 7.3 in Chapter 7) or grouping together words that start with the same letters (Section 8.5.2 in Chapter 8). Even custom signal processing (Section 9.8.1 in Chapter 9) is an example of using domain knowledge to modify the feature representation to present to the ML algorithm.

In the words of Yoshua Bengio:[337]

> Good input features are essential for successful ML. Feature engineering is close to 90% of effort in industrial ML.

These intuitions can also be captured at the feature ideation level, as discussed in Section 1.5.1. For example, if you think that emails that include a person in their lead photo convert better (i.e., they generate more sales), you can create a binary feature that records whether a person appears in the lead photo (how to compute such a feature is a different problem and you might need to rely on a separate ML system for it). You (if you are a domain expert) or **consulting with a domain expert** can provide information about the need to expand your available raw data. For example, if there are reasons to believe power consumption may be related to server outages in a data centre, you might want to request measurements for power consumption begin to be recorded and made available for learning and so on. As part of the FE process, the raw data should be transformed into features highlighting their relation to the target class (e.g., transform weight into BMI[171] in a health assessment task).

In the words of Andrew Ng:[337]

> Coming up with features is difficult, time-consuming, requires expert knowledge. "Applied machine learning" is basically feature engineering.

FE is sensitive to the ML algorithm being used as there are certain types of features (e.g., categorical) that fare better with some algorithms (e.g., decision trees) than others (e.g., SVMs). In general, the hope is that better features will improve the performance of any ML algorithm but certain operations are more useful for certain algorithms. Whenever possible, I have tried to signal them throughout the book.

The reasons for doing FE are usually reactive: an initial transformation of the raw data into features (featurization) did not render the expected results or did not render results good enough to put them into production use. At this stage, it is common to embark on what I call *model shopping*, or what has been called by other authors *data dredging*[204] or *a random walk through algorithm land*,[193] that is, to try different ML algorithms without much intuition, just out of convenience with the ML software package. In general, the difference between ML models with similar possible decision boundaries is not substantial and after repeating this process a number of times, the chosen algorithm will most definitely overfit (even when doing cross-validation because the model will not be overfit but the decision of picking the model will be too tied to the training data). In my experience, a well-orchestrated FE process can highlight much value in the raw data, sometimes even driving its expansion (for example, adding geolocation by IP[49]).

In the words of Pedro Domingos:[84]

> . . . some machine learning projects succeed and some fail. What makes the difference? Easily the most important factor is the features used.

A separate reason to do FE is to put the features into a more understandable light from a human perspective, with the purpose of having interpretable models. Such is the case when doing inference rather than prediction.

There are two other, more abstract, reasons. First, as Dr. Ursula Franklin said on her 1989 CBC Massey Lectures,[310] "Tools often redefine a problem." FE allows to maintain a focus on problem-solving grounded in the domain at hand. ML does not exist in isolation. Second, having a suitable toolbox can prove to be a phenomenal boost in self-confidence, a fact highlighted by Stephen King in his non-fiction work *On Writing*:[183]

> [C]onstruct your own toolbox [. . .] Then, instead of looking at a hard job and getting discouraged, you will perhaps seize the correct tool and get immediately to work.

Finally, many types of raw data require significant FE before they can be used with ML. The domains in Part Two fall into this category. The same goes for raw data with very large number of attributes.

I will conclude with a quote from Dima Korolev regarding over-engineering:[72]

> The most productive time during the feature engineering phase is spent at the whiteboard. The most productive way to make sure it is done right is to ask the right questions about the data.

1.2 Evaluation

Before looking into ML and FE cycles, let us spend some time looking into the issue of evaluating the performance of your trained model. They say it is better to crawl in the right direction than to run in the wrong one. This applies also to ML. How you will evaluate your trained model has deep implications on your choice of model and the type of FE you can perform. Centering the evaluation metric decision based on which metrics are easily available in your ML toolkit can be a great mistake, particularly as many toolkits allow you to plug in your own metric.

We will briefly discuss metrics next, about which many books have been devoted.[169,350] We will then look into how cross-validation relates to evaluation (Section 1.2.2) and at issues related to overfitting (Section 1.2.3), before concluding with a discussion about the curse of dimensionality.

1.2.1 Metrics

As part of the problem definition, it is important to spend some time thinking about the different metrics that will be used to evaluate the results for a trained algorithm. The metrics are deeply tied to the underlying use for which you are training the model. Not all errors will have the same impact on your application. Different metrics can penalize specific errors differently. Familiarizing yourself with them can help you pick the right one for your task. We will start by looking into metrics for classification before discussing regression metrics.

A great way to understand errors and metrics is through a **contingency table** (also known as a cross-classification table), which for the case of predicting a binary class becomes the following: