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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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).

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

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:

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

		Real: +	Real: −
Predicted	+	true positives	false positives
	−	false negatives	true negatives

In general, it is good to distinguish **false positives** (type I errors), where the system is predicting something that it is not there, from **false negatives** (type II errors), where the system is missing to identify something that it should. Certain applications are more tolerant of one type of errors over the other. For example, prefiltering data can be quite tolerant of type I errors. On the other hand, an application that decides to single out a person for shoplifting, however, will have very little tolerance for type I errors.

Measuring how many times a classifier outputs the right answer (**accuracy**, true positives plus true negatives over the total number of points) is usually not enough, as many interesting problems are very biased towards a background class (and thus true negatives will dominate the computation). If 95% of the time something does not happen, saying it will never happen will make you only 5% wrong, but is not at all a useful classifier.

Type I and type II errors are usually summarized as ratios. For instance, the number of true positives over the total number of labelled examples. This metric has received many names including **precision** or PPV (positive predictive value):

$$precision = \frac{|correctly\ tagged|}{|tagged|} = \frac{tp}{tp + fp}$$

Alternatively, you can focus on the false negatives. This metric is called **recall**, TPR (true positive rate) or also sensitivity:

$$recall = \frac{|correctly\ tagged|}{|should\ be\ tagged|} = \frac{tp}{tp + fn}$$

This metric will give you an idea about whether the system is missing many labels.

Other metrics can be so defined, for example, NPV (negative predictive value) and TNR (true negative ratio). If you need to have only one metric that summarizes both numbers, you can take a weighted average of them, what is called the F_β**-measure**, where the β tells you whether to favour precision rather than recall:

$$F_\beta = (1 + \beta^2) \cdot \frac{P \cdot R}{\beta^2 P + R}$$

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

Setting β to 1 renders the metric as F_1 or simply just F and that favours both metrics equally ($2^{PR}/_{P+R}$). Another popular metric to summarize the behaviour is **area under the ROC curve** (AUC-ROC), that is, the curve under a recall vs. false positive rate (1-TNR) plot obtained by varying sensitivity parameters of the model.

Note that using F_2 versus using F_1 might significantly change the results of your evaluation. If you know nothing about your task, you can use F_1, but the more you know, the better metrics you can use (or even design). For example, in the question answering competition TREC, the script distributed by the organizers[53] computed a whopping 36 metrics on the results. Focusing only on one metric is usually preferred by higher-level decision makers, but as a practitioner working closely with ML algorithms and source data, you will be better served to consult a variety of metrics in order to build a better understanding of the behaviour of the trained model. Ultimately, I advocating for a full-fledged error analysis process (Section 1.4.2).

When comparing multiple annotations, as when having multiple annotators, metrics for inter-rater reliability can be used, for example Fleiss' kappa,[108] which measures the agreement that can be achieved above chance and the numerator measures the agreement actually observed above chance.

For regression problems, the error can be measured as a difference, but in that case negative errors might cancel with positive errors. Therefore, it is necessary to take the absolute value of the error. However, the absolute value does not produce a continuous derivative, and thus, it is customary to use the square of the errors instead, the mean squared (MSE). To have the metric in the same units as the original signal, you can take the square root of the mean of the squared errors, obtaining the RMSE. Other ways to weigh the errors are possible but less common. For example, you can use another exponent instead of powers of 2, or you can weigh negative errors different than positive errors. Note that the requirement of a continuous derivative is important if you are using the error directly as part of the optimization process of your ML algorithm (for example, to train a neural network). You can also use an error metric for algorithmic purposes and another to evaluate your results and see whether they are fit for their underlying purpose (**utility metrics**).

Finally, the metrics discussed before all deal with averages and try to summarize the behaviour of the model on its most representative case. Nevertheless, that does not address the **variance** of the results. This topic is well studied in ML, where we talk about the **bias** (learning the wrong thing; errors due to limitations of the model) versus **variance** (learning scattered points; errors due to finite data sampling and different samples produce different models).[84]

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

1.2.2 Cross-Validation

Cross-validation is a technique to deal with small datasets in model evaluation by reducing the loss of data allocated to testing the model. In general, this is done due to a lingering feeling that testing data is "wasted" as it is not used to estimate the parameters of your model. However, keeping data aside to understand how well your trained ML model performs on production is definitely more valuable than the marginal changes to the model that data will produce: if 20% more data produces drastic changes, then your model is not stable, and you do not have much of a model, really. Basically, you will be better served by a simpler model with fewer parameters that behaves in a stable fashion over your available data.

Moreover, the value in executing your model on test data goes hand in hand with using your model as a component inside a larger solution. A good understanding of how well your model performs on unseen data might enable you to address many of the model's shortcomings with business logic outside the model itself. Would you rather have a model that has a 5% lower RMSE but fails in obscure, not understandable ways, or one where its errors are understood, signed off and accepted by its potential users and clearly communicated to them? (Of course, if you do not know how either model behaves, you will choose the one with lower RMSE but I am trying to argue for deeper understanding of the behaviour of the models.) The process for leveraging the test data to gain such rich insights is called error analysis and it is discussed in Section 1.4.2.

In cross-validation, the rationale is to split the training data into N parts, taken at random. The system is then trained and tested N times: for each fold, the remaining $N - 1$ folds are used to train a model, which is then used to predict labels or values on the selected fold. In certain domains, care has to be taken when splitting the data so that each fold contains a full view of the data, and then the splitting is random over sets of instances rather than over all instances. For example, if training over multiple rounds of user logs, all rows for the same user should fall into the same fold. Otherwise, the evaluation will not be representative of the behaviour in production. Also, care should also be taken that all labels appear in the train set.

At the end of the process, the evaluation metric or metrics can be computed over the full, labelled dataset (micro evaluation) or as the average of the evaluation metrics over each fold (macro evaluation). I personally prefer micro evaluation as it does not depend on the number of folds but the variance of the macro evaluation is a good estimator of the stability of the model over the available data.[42]

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

1.2.2.1 Out-of-Fold Estimation

A great use of cross-validation, which is needed by many techniques presented in this book, including the case study in Chapter 6, is to use the fold system to compute feature transformation estimates that depend on the target variable. Without the use of cross-validation, such estimates constitute what is known as a **target leak**, a poorly constructed feature where the target class has been made available by mistake to the ML algorithm. Such target leaks are usually very unfortunate. The too-good-to-be-true evaluation numbers could get communicated widely before the target leak is discovered when trying to implement the production code that makes use of the model ("What do you mean I need the attrition rate to compute this feature? I don't know the attrition rate, that's why I'm calling your model!"). This technique of using folds for estimating feature transformation that require the target variable is known as **out-of-fold** estimation, and it is so common that it is abbreviated as OOF. Be careful to use a large number of folds, so as to have estimates that are stable enough. For an example, see Section 6.4 in Chapter 6.

1.2.3 Overfitting

Overfitting is a well-studied topic in ML. It is well-addressed in existing general ML books[234] and it is central to FE. When doing ML, we care about fitting a model to existing data, with the purpose of generalization, that is, extrapolation. Overfitting happens when the model follows too closely to the original training sample and it fails to generalize. It is the reason we always use a separate test set when training supervised learning models. Evaluating the train set will provide a view of the results that is too optimistic and not representative of the behaviour on new data.

Now, the training data and test data all constitute a *sample* of the overall population over which we plan to use the model. Sometimes, through a series of training and evaluation steps, we might gain insights about the full sample that will lead to overfitting of the ML process, not just on the testing-on-train-data sense but on trickier ways, such as choosing suboptimal models, model parameters or, central to this book, features that accidentally perform better on the sample but will underperform when applied over the actual population.

For example, imagine if you are trying to predict whether a drug will produce an adverse reaction and a patient should discontinue its use. Training data was collected during the winter and one of the questions asked was whether the patient was thirsty frequently. This question proved to be a very informative feature. You might encounter that this is positive (i.e., that the

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

feature fires) much more often during the summer months, producing many false positives in production. At that stage we can conclude the model features were overfit to the test set.

A common misconception that I have heard from practitioners of all levels of expertise is that cross-validation is not prone to overfitting as compared to using a held-out set. That is, of course, not true; if you quiz your data repeatedly, your results get tuned to the sample rather than the overall population. That is well understood in statistics and there are techniques like the Bonferroni correction[95] that says roughly that if you use the same data to answer N questions, the statistical significance threshold has to be reduced substantially, as much as dividing it by N (making rejecting the null hypothesis much harder and therefore requiring much more extraordinary evidence for significance). I have not yet encountered a similar principle for FE, but as I advocate in the next section to use a held-out set only once, the issue falls into a matter of personal taste.

1.2.4 Curse of Dimensionality

Care must be taken when designing feature sets to escape the **curse of dimensionality**. The main issue with higher-dimensional spaces is that everything starts to become very close, even if just by chance. That is the case as meaningful differences on a few key coordinates are drowned over similar values for the remaining coordinates. Moreover, the more dimensions present in the data, the more training data needed (a rule of thumb says an absolute minimum of five training instances per dimension.)[84]

1.3 Cycles

Given a ML algorithm and some identified features, the parameters for the algorithm can be estimated from the training data and then evaluated on **unseen data**. The purpose of this book is to provide you with tools to perform several cycles iterating over the process of finding better features, a process called **feature evolution**. If the test set is consulted multiple times, that will lead to overfitting (discussed in detail in the next section), that is, to select a suboptimal model that appears to perform better than what it will perform when applied to truly fresh data (which will be tragic when deployed on a production setting). To avoid this problem, it is of uttermost importance to have a development test set to use during feature evolution and to leave enough test data for a final evaluation. Such data might be new, held-out data or freshly

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

acquired data if you are looking into a process that continuously generates annotated data.[49]

1.3.1 ML Cycle

Building computer systems that adapt their behaviour based on available evidence is usually subdivided into several types of problems. By far, the most common type of ML problem is that of extrapolation of a function. In this context, extrapolation involves, given some known points of a function, to predict how the function will behave over a different set of points. This is the case of **supervised learning** of classifiers, discussed in the next section.

While supervised learning is the most common problem and technique, there are other problems and techniques popular within ML (see Table 1.1): **unsupervised learning** (which seeks to find structure on unannotated data), **reinforcement learning** (which uses delayed feedback to guess annotations over the data), **active learning** (that selects which data to be annotated next) and **semi-supervised learning** (that mixes annotated and unannotated data), to name a few. Most of these problems and techniques operate over representations of reality consisting of feature vectors and can benefit from the techniques discussed in this book.

At its core, a supervised ML algorithm is defined by a representation being learned, an objective function to be minimized or maximized and an

Table 1.1. *Types of ML.*

Learning Type	Goal	Example/Advantage
Supervised	Function extrapolation.	Given the behaviour of a website visitor, determine whether they are likely to purchase if offered a coupon.
Unsupervised	Find structure in unannotated data.	Group together different paradigmatic customer behaviours in a website, to help build marketing personas.
Others		
Reinforcement	Learning from past successes and mistakes.	Game AIs.
Active	Select data to be annotated next.	Needs less annotation effort than supervised.
Semi-supervised	Mix annotated and unannotated data.	Needs fewer annotations.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

optimizer that guides the search over the space of potential representations.[84] For example, in a decision tree, the representation is the tree being built, the objective function is a metric of how well the tree splits the training data into homogeneous sets and the optimizer picks the feature split that improves over the worse subset. There is a recent trend in moving to an explicit representation of these three components, led by NN frameworks like TensorFlow.[1]

The ML cycle presented here (Figure 1.1) follows consensus in the field but it expands the relation between raw data and featurization. It starts with

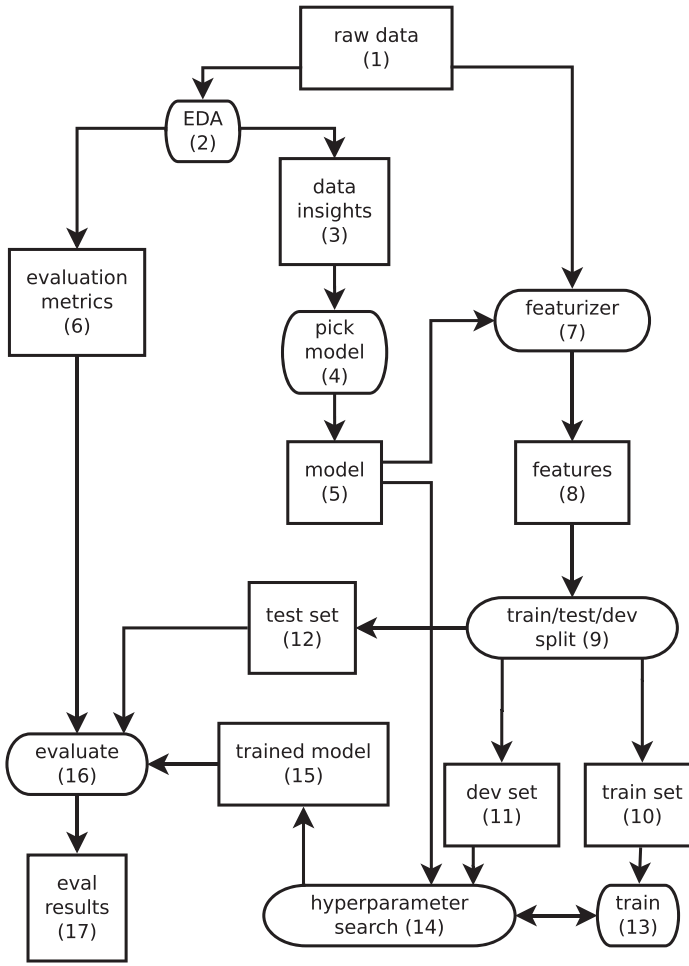

Figure 1.1 ML life cycle.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

an exploratory data analysis (EDA), discussed in Section 1.4.1 and numbered (2) in the figure, on the raw data (1). From the obtained data insights (3), you ought to be able to pick (4) an ML model (5) in a more informed fashion. At this stage, the evaluation metrics should also be decided[204] (6). With the model in hand, you can proceed to generate features (8) from the raw data (7), which is then split (9) into train (10), development (11) and test sets (12). The train and test sets ought to be completely separated as we want to know how the model will predict on new data.[49] The test set should be at least 10% of the total data, with some authors[204] recommending up to 30%.[†] The model can then be trained on the train set (13). The development set, if any, is used for a hyperparameter search, which is model dependent (14). The best model from the hyperparameter search is evaluated (16) on the test set (**held-out data**). This cycle concludes with a trained model (15) together with a believable evaluation (17). Note the held-out data can only be used to test the model once. Resist what has been called "double-dipping the training data,"[49] either explicitly or by applying cross-validation repeatedly. If you want to do several cycles doing variations of different parameters of the algorithm (or its features, see next section), you should use a different (new) held-out set. Alternatively, you can use a separate evaluation set ("development test set") and acknowledge you will be overfitting your results to that set, before performing a final evaluation. Such is the approach I advocate for a full-fledged FE cycle, as discussed in the next section.

The changes from raw data to features plus the different experiments, datasets being used for evaluation (and, thus, consumed), etc., become the pedigree of your models. Care must be taken to document this process, for example, in a **log book** or lab notebook.[204] Without this, for problems that undergo a long improvement process, it is not uncommon to encounter trained models that work well in practice and there is no recollection of what data or exact process was followed to obtain them. Trade speed in executing iterations with judicious documentation as training a model is something seldom done once. Always favour **reproducible builds** over a few extra points of performance.

1.3.2 Feature Engineering Cycle

My proposed FE cycle (Algorithm 1.1) has an emphasis on building an understanding of what features work and do not work and having a **final**

[†] Very large datasets can get away with using only 1% of the training data, as long it has low variance.[235]

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

evaluation on held-out data (generated in line 1 of Algorithm 1.1). That final evaluation set has also been called a *confirmation set*.[204] I am proposing this methodology as a compromise when no continuous stream of newly annotated (fresh) data is available. If fresh data is available (or if you can schedule batches of test data on a **data release schedule**), you can just cycle through the whole process altogether; each evaluation on fresh data is equivalent to a full evaluation. In the proposed framework, the training data is reused on a number of cycles (line 2 of Algorithm 1.1) and thus risks overfitting. In each cycle, a fixed development set may be used or a new one split into a train set and a development set (a **bootstrap**) can be used. Cross-validation is also an option but the extra running time might not be worth it, and it complicates the error analysis. Of course, the decision is yours.

FE is an iterative process where a set of features is defined, experimented upon, evaluated and refined. As part of that process, training data is consumed and hypotheses regarding changes to the feature set are built based on the result of the experiments. For the process to be successful, a stream of unseen data is needed, otherwise, the hypothesis generated from the experiments might lead to procedural overfitting, that is, to changes to the feature representation that will not help in the general case but help for the data analyzed as part of the FE cycle.

If you do not have enough data, you can use a different split. The FE process concludes with you building in your mind an understanding of the behaviour of the features for this data, problem, and ML model.

When doing repeated ML experiments, it is useful to have a development test set[43] – a test set, independent from the train set that is used during development. This test set is also independent from the final test set. It helps avoid procedural overfit. This final test speaks better about the generalization power of the model. While this is a good idea in general, when doing heavy FE, it is mandatory as the possibility for overfitting when doing FE is greatly increased.[17]

FE cycle is not unlike the ML cycle. The key difference is in the type of processes performed in the cycle: (1) identifying good features and expanding them[17] and (2) identifying redundant/uninformative features and dropping them. Moreover, it might not be necessary to run the full train cycle to do this. Sometimes, measuring the correlation between a feature and the target class is enough, that is, you might want to consider creating a contingency table for each feature and to look for patterns in that data.[43]

Ultimately, I believe that error analysis is the leading force in FE.[44] The key is improving the understanding of each feature, for example, revisiting EDA

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

but this time only for one feature or one feature and the target.[72] The final goal is to have a better framing of the raw data.[295]

The final output of the FE process is a data pipeline (or two if you do not have a stream of fresh data) that produces features from data.[49†] Such pipelines can be expressed as custom code or through feature manipulation formalisms such as the ones present in scikit-learn.[253]

While many quick iterations reduce the need for domain expertise,[193] if you have domain knowledge or access to people who do, this is also a great moment to incorporate domain knowledge (for example, realizing the category "late cretaceous" falls into the category "cretaceous"[267]).

If the FE process was performed without a fresh stream of test data, then I advocate for the construction of two feature sets, an optimized set (with a greater risk of overfitting) and a conservative set (maybe worse performing). In the final evaluation, both feature sets are evaluated separately (lines 5 and 6 in Algorithm 1.1). The conservative set will be used if the optimized set does not outperform it in a significant manner (line 7 in Algorithm 1.1).

Algorithm 1.1 Feature engineering life cycle. The ML cycle in the loop is the one in Figure 1.1.

Require: raw_data
Ensure: featurizer, model
1: $raw\_data_{final\_eval}, raw\_data_{feat\_eng}$ = final_eval_split(raw_data)
2: **while** not good results **do**
3: $featurizer_C, model_C, featurizer_O, model_O$ = ML_cycle($raw\_data_{feat\_eng}$)
4: **end while**
5: $results_O$ = evaluate($model_O$, $featurizer_O(raw\_data_{final\_eval})$)
6: $results_C$ = evaluate($model_C$, $featurizer_C(raw\_data_{final\_eval})$)
7: **if** $results_O > results_C + \delta$ **then return** $featurizer_O, model_O$
8: **else return** $featurizer_C, model_C$
9: **end if**

1.4 Analysis

The FE process proposed in the last section is heavily dependent on two types of analyses described next. Exploratory data analysis (Section 1.4.1) is helpful to understand the raw data, devise features and select suitable ML algorithm

† Chapter 7.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

and metrics. Error analysis (Section 1.4.2) seeks to understand the strength and weakness of your feature set to drill down and improve it. Note that there is a certain analyst bias in any of these tasks. Other practitioners doing similar analysis might arrive at different results.[284]

1.4.1 Exploratory Data Analysis

EDA refers to exploratory analysis of the raw data. Even without domain knowledge,[91] it is possible to analyze raw data and extract insights about the behaviour of the data, particularly as it relates to the target variable. I am among many authors[196,278] who find it to be a necessary ingredient for successful FE. You want to make sure that you have "rich enough data to distill meaningful features."[20] A good first step is to analyze the variety of values the different columns in the raw data take. Descriptive statistics such as mean, median, mode, extremes (max and min), variance, standard deviation, quartiles and visualizations such as box-plots are helpful at this stage.[239] Columns with very little variability tend to have very little explanatory power and are better off being ignored, unless the variability is highly correlated with some values of the target class. Therefore, it is good to plot correlations between the target variable and different columns. Other things to try include analyzing how your ongoing conclusions about the nature of the dataset change as you take different subsamples of the data, particularly samples segmented by time or type.[72] You might also want to check whether certain fields that look random are actually random by using standard randomness tests.[204] Again, random data ought to have no explanatory power and can be discarded. Outliers can also be found and discussed at this stage (cf., Section 2.4 in Chapter 2).[19] Similarly, you can start looking into missing values, although you might need to do a full featurization to find ways to deal with them in a principled way.

From there, you can look into the relationship between two columns or a column and the target class. Scatterplots help you visualize this graphically. Another alternative is to use summary tables (also known as pivot tables), where a categorical column is used to segment the values of a second column. Each cell in the table contains summary statistics about the values of the second column that fall into the first category value. If only counts are provided, it resembles a simplified contingency table, but other summaries (mean, standard deviation, etc.) are possible.[239] Finally, you can compute metrics about the relationships between pairs of columns by using correlation coefficients or any of the metrics presented in Section 4.1.1 in Chapter 4.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

If you possess domain knowledge or have access to people who do, this is a great time to make your assumptions about the domain as explicit as possible and test whether such assumptions hold on the data at hand. Do you expect certain distribution of values to follow a normal (or any other given) distribution? State that assumption and check whether it holds. For example, using histograms (cf., Section 2.3.1 in Chapter 2) or computing representative statistics such as skewness, etc. (cf., Section 2.3).[19] You do not want to build heavily on assumptions that ultimately do not hold. All throughout FE, you need to exercise an inquisitive mindset and drill down on anything that looks suspicious. If the domain knowledge indicates a certain column ought to behave in a certain way and it does not, do not let that fact pass by without further investigation. You might unveil data extraction errors or hidden biases in the sampling. Or you might just have the wrong domain assumptions that might have deeper implications than the data columns at hand. This will also be a good moment to engage in data cleaning, for example, to check that your values are all in the same units using box-plots.[204]

Useful tools for EDA include, of course, spreadsheets such as Excel,[18] Jupyter Notebooks[188] and OpenRefine.[316] The latter, formerly GoogleRefine, can find similar names, bin numerical data and do mathematical transforms, among many other capabilities.[17] You might even want to do a data survey computing key statistical descriptors following Pyle's method from chapter 11 of *Data Preparation for Data Mining*.[263] There are also ways to visualize observations by looking at multiple variables at once: based on data similarities (clustering); based on shallow data combination (association rules); and based on deeper data combination (decision trees).[239†]

All case studies in Part Two, Chapters 6 to 9 start with an EDA process. For example, in Chapter 7, we will see a feature heatmap over different versions of the feature, reproduced here in Figure 1.2. The graph aligns features for each year, uncovering the story of the data. Missing values are shown in a checkerboard pattern. The values of the features are clustered into six classes, represented as different gradations of gray. This allows you to see how the values of a feature evolve over time. See Section 7.1.1 for further details.

1.4.2 Error Analysis

The previous discussion about approaching the behaviour of the data with suspicion takes its full force when doing **error analysis**. Plenty of analysis can be done with aggregate metrics but I feel that falls in the direction of

† Chapter 5.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

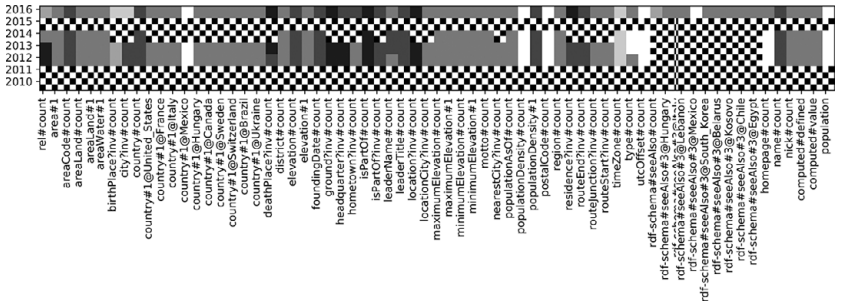

Figure 1.2 Historical features visualization using a feature heatmap (excerpt). The checkerboard patterns are missing values; the different shades of gray indicate different quantiles.

evaluation metrics, discussed in Section 1.2.1. To me, error analysis is all about the parochial, the anecdotal. Focusing on a few cases that capture your attention for important reasons then biting down on that problem and not letting it go, going deeper and deeper, trying to assemble a meaningful narrative behind the reasons why that error came to be. Of course, a great working understanding of the internals of the ML system helps quite a bit there, together with appropriate tooling, for example, mechanisms such as TensorBoard.[218] In general, to perform a reasonable error analysis, your ML algorithm will need to be instrumented so as to generate enough metadata to enable this type of drilling down. Looking at samples rather than aggregates and being skeptical are part of what has been referred as "responsible data analysis."[193]

For example, the first featurization shown in Chapter 6 has a few cities that contribute heavily to the overall error. For each of these "problematic" cities, Section 6.3.1.1 finds the feature that further contributes to the error by a process of feature elimination, reproduced in Table 1.2. From this analysis, the conclusion was to dampen the effect of the count features. See Chapter 6 for details.

From a successful error analysis, you might discover that certain features are hurting performance more than helping it (therefore deciding to incur in a session of feature selection, cf., Section 4.1, Chapter 4) or that certain features are really informative,[72] in which case you might want to further boost their signal engaging in some of the feature drill-down techniques discussed in Section 3.1 in Chapter 3. Single-feature visualization techniques might also prove useful.[352†]

† Chapter 2.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

Table 1.2. *Ablation study. A 5% sample was retrained while removing one feature at a time. The highest contributor is listed here.*

City	Improvement	Feature to remove
Dublin	0.85 to 0.6 (29.41%)	city?inv#count
Mexico_City	–0.53 to –0.92 (–73.81%)	seeAlso#3@List_of_tallest_buildings
Edinburgh	0.17 to –0.19 (212.89%)	leaderTitle#count
Algiers	–1.0 to –1.2 (–15.33%)	name#count
Prague	–1.3 to –1.7 (–30.56%)	seeAlso#count
Milan	–0.84 to –1.1 (–30.81%)	seeAlso#count
Amsterdam	–0.74 to –0.96 (–29.86%)	homepage#count
Lisbon	–0.6 to –0.88 (–46.54%)	seeAlso#3@Belarus
Tabriz	–0.75 to –1.2 (–57.17%)	seeAlso#count
Nicosia	0.3 to –0.028 (109.40%)	country#count
OVERALL	–4.5 to –6.3 (–40.69%)	seeAlso#count

While you might use specific examples to generate hypothesis for improvement, whether to pursue those hypothesis should be informed by their potential impact on the whole dataset. Otherwise, you will encounter a situation akin to optimizing rarely used code.[267] Following Amdahl's law,[9] the code that takes the most time is the code that ought to be optimized the most. In the same vein, errors that account for more of the performance loss ought to be addressed first.

When drilling down on errors, use the contingency table (cf., Section 1.2.1), maybe even on a per-feature basis,[43] and remember your chosen metrics for this problem and domain: if type I errors are more important, drill down more on them, accordingly.

Also keep an eye out for spotting software errors. As ML produces results with intrinsic error, debugging ML systems can be very hard. Do not take behaviour that counters intuition lightly, as it might indicate software bugs that can be much more easily addressed than ML modelling issues.

Another interesting tool for error analysis is the use of random data and random models. If your trained model is doing something sensical, it ought to outperform a random model. If a feature is meaningful, its value from actual data ought to outperform replacing it with a random value.[72] The last approach might help you unveil errors in your data, which can be particularly difficult to appreciate when you are tunnel focused on your model errors.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

1.5 Other Processes

I will conclude this introductory chapter with two highly related processes that wrap up the FE discussion: domain modelling, which in this book pertains to how to go from raw data to a first feature set, and featurization (Section 1.5.2), which deals with feature improvement and drill-down.

1.5.1 Domain Modelling

Domain modelling is separate from featurization to distinguish the more abstract from the more operational aspects. Representing a domain's raw data into features is an inherently creative approach, similar to modelling in any other area of mathematics and (software) engineering. Modelling and design in general are matters of decisions and approximations; as such, there is an inherent imperfection aspect to them. This discussion follows more abstract concepts while the next section covers more concrete terms, also related to how to generate follow-up featurizations, that is, featurization when certain features have already been identified. In the words of Max Kanter and colleagues from the Featuretools system:[175]

> Transforming raw data into features is often the part of the process that most heavily involves humans, because it is driven by intuition.

Let us see how to come up with features, and then discuss what makes particularly good features and list domains with a tradition of successful featurization. The process of coming up with features, **feature ideation,** is closely related to brainstorming, design, creativity and programming in general and thus benefits from general advice on brainstorming.[17,178] There are no hard truths here, just general guidelines and advice. A usual approach is to use anything remotely related to the target class as a feature (basically, all the raw data and variations), in what constitutes a **feature explosion**.[40] This rarely works in practice unless the raw data has very low dimensionality (5 or 6 attributes). You will be better off starting with features you believe will have predictive value.[49] But do not let the available raw data constrain you.

A very beneficial approach is also to think of the type of information *you* (as a human) would use to make a successful prediction.[127] Such "thought features"[72] can unveil a trove of ideas and lead to requests for expansions to be made to the original raw data. Of course, you do not need to reinvent the wheel; you can also build on the work of others[17] through published papers, blogs or source code. Post-mortem posts in online ML competition sites such as Kaggle.com can be very informative. Using features from existing work will

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

most likely require some feature adaptation[21] but it might save you a world of trial and error. Do not feel constrained by the existing attributes in the raw data and look to propose features that are computed from existing attributes (cf., Section 3.1 in Chapter 3, for computable features).

A common oversight at this stage is to forget to reach out to domain experts for insights on what they would consider good features for this problem and domain. Here traditional techniques for knowledge acquisition in expert system can be helpful.[52,269]

Finally, once you start converging into a feature set, you might want to step back and start looking for rare new features that can explore other parts of the problem space.[17] The case studies provide an example of this approach: once a baseline has been established using structured data, we move to look at a textual description of each city. Another example is presented in Chapter 9, where we move from colour histograms to local textures expressed as the number of corners in the image (cf., Section 9.6). That might be particularly productive if your chosen ML model is one based on ensemble learning, like random forests.[48]

As you come up with the features, the next question is: **what makes for a good feature?** Which one of these potential features should you address in your first **featurizer**?[41] While any piece of information that is potentially useful for a prediction may do as a feature,[342] you want features with these three properties:[72]

(1) **Informative**, the feature describes something that makes sense to a human, which is particularly important when doing inference and seeking out interpretable models, but also for productive error analysis,
(2) **Available**, the feature is defined for as many instances as possible; otherwise you will need to deal with missing data (cf., Section 3.2 in Chapter 3), and
(3) **Discriminant**, the feature divides instances into the different target classes or correlates with the target value. Basically, you want to seek out characteristics of objects that are available, easily modelled and have a relation to the target being predicted.

In the words of Peter Norvig:

> Good features allow a simple model to beat a complex model.[250]

Moreover, you want them also to be as independent from each other and simple as possible,[127] as better features usually mean simpler models.[51] How many features to extract is also an open question, as ML can be very effective given a

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

good feature vector.[202] In theory, the more features f you have, the more data you will need to give to your ML to achieve the same error ϵ, in the order of $O\left(\frac{1}{\epsilon^f}\right)$ amount of data.[206]

You can test your distilled features at this stage with statistical association tests (like the t-test, building an association table between the feature values and the target values),[23] and, in general, many of the feature utility metrics from Section 4.1.1.1 in Chapter 4 can be used. If needed, you can revisit this feature ideation process later on after a few cycles of FE and include more features. Brink, Richards, and Fetherolf propose[49] to start with the most predictive features first and stop there if they perform well; otherwise, try a feature explosion and if it does not work, prune the expanded feature set using feature selection (cf., Section 4.1).

Certain domains have standard representations by now. For new domains, studying the existing ones might give some ideas and guidance. Such domains include images, signals, time series, biological data, text data,[40] prognosis and health management,[337] speech recognition, business analytics and biomedical.[180] Part Two of this book includes seven domains.

Finally, in term of feature presentation, you want to make sure that related variables should be contiguous and that you use multiple levels of information (that is, you split features into sections), if needed.[324] This will not affect the ML but it will help you better visualize the feature vectors and ward off software errors.

1.5.2 Feature Construction

The last step in the process sketched in this chapter involves the actual execution of code (the featurizer) that assembles a feature vector from the raw data. The featurizer is algorithm dependent and it can change the types of features from the raw data. This process can be as simple as selecting columns from a DB but as the multiple cycles of FE start piling up, it can get quite complex. There are some software solutions for this problem (some of which I have built in the past[124]) but none that have attracted widespread use as of the writing of this book.

There seems to be some momentum within the academic community to express featurization as ETL (extract, transform and load) operations from data warehousing[167] or as SQL stored procedures[267] or UDFs (user-defined functions)[10] as a framework to standardize computable features. That seems like a good idea, as featurization is inherently a data manipulation activity, of which the DB community has a long and productive history to rely on. It is

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

to expect that somewhat complex operations such as OOF estimation might become standard operations available in DB systems in the future.

I will now discuss a particular featurization operation, feature templates, which is usually employed early in the process and important enough to deserve its own section.

1.5.2.1 Feature Types

Features as defined in Section 1.1 are values of a certain type, as used by ML algorithms. We will review the following types, as they are accepted by many algorithms in widespread use: binary, categorical, discrete, continuous and complex features. Table 1.3 contains examples for each of these feature types. In this discussion and throughout the book, x_i refers to a feature within the feature vector $\vec{x}$ and y refers to the target class or target value.

Binary Features. These are the simplest possible features. They are features that can take one of two values: true or false, present or absent, 1 or 0, –1 or 1, A or B. They are also known as *indicator features*. Some ML models (such as some implementations of maximum entropy[164] and certain SVMs) can only take such features. Many ML models (including SVMs and logistic regression) can only predict binary targets.

Categorical Features. These are features that can take one among many values, known as *categories, categorical values*, or *levels*. The categories themselves are known beforehand. In general, there is no ordering among the different categories. Usually, the categories themselves receive a meaningful name from the domain. The number of categories tends to be small. That is, using the whole English dictionary (say, 20,000 categories) as a categorical feature is possible but models will seldom benefit from such a representation.

Discrete Features. These are features that represent values that can be mapped to integers (or a subset of them). Because integers have an absolute order, discrete features are also ordered. Representing large number of categorical features as discrete features is tempting (and many times done in practice), but leaves the model with a false sense of ordering if the categorical features do not posses an ordering. For example, what is the meaning of saying that Visa is less than Mastercard, which is less than American Express?[204]

Continuous Features. These are features that can be mapped to a real number (or a non-discrete subset of the real numbers). Ratios and percentages are special subtypes of these features. Due to physical restrictions of digital computers, all continuous values are actually discrete, represented as a set of bit values on a particular encoding. While this might sound like a theoretical point, the representational richness of the internal floating point implementation might be a key problem with sensor data. You must take care

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

Table 1.3. *Feature types and examples.*

Type	Description / *Note*	Example
Binary	One of two values	Did the customer leave a tip? Did the turbine make noise? Did the student turn in the exam before the end of the allocated time? Does the person like cats?
	simplest	Did the licence plate have a match in the DMV database?
Categorical	One among many values	Colour of the car, Brand of the car, Credit card type, Number of bedrooms (*also a number*)
	values known beforehand	Type of mattress (*also mattress surface area*)
Discrete	Can be mapped to the integers	Number of pizzas eaten, Times visited the gas pump, Times the turbine got serviced last year, Steps walked last week (*compare: distance walked, which is continuous*), Number of people living in the house.
	order is important	
Continuous	Can be mapped to a real number	Engine temperature, Latitude, Longitude, Speech length, Colour intensity at centre of image, Distance walked, Temperature at centre of dish (*as measured by an IR camera*)
	representation issues	
Complex	Records, lists, sets	Date (year, month, day; note that it can be expressed as number of days from a fixed date), Product name (*it might include brand, product type, colour, size and variant*), Location (latitude and longitude), Complaint (a sequence of characters), Countries visited (it is a set of categories)
	challenging	

to avoid extreme situations; either your values might underflow, with all values mapping to zero, or overflow, with all values mapping to the maximum possible value.

As a refresher, floating points are usually represented using the IEEE 754 format, which uses scientific notation, for example, the number 56332.53 is represented as 5.633253×10^4. The actual representation consists of a sign bit, a fraction and an exponent. Single precision uses 8-bit exponent and 23-bit fraction, double precision uses 11-bit exponent and 52-bit fraction. Due to the fraction and exponent behaviour, not all **segments of the real line are sampled at the same resolution.** These topics are studied in detail in **numerical analysis** courses.[123,248]

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

Complex Features. Some ML algorithms can process features in what can be called non-normal form in databases: records, sets, lists.[48,68] Most systems benefit from decomposing those, and we will see some simple techniques in Section 3.3 in Chapter 3 and advanced techniques in Section 5.1 in Chapter 5.

1.5.2.2 Feature Templates

Feature templates are small programs that transform raw data into multiple (simpler) features at once. They have two objectives: first, to present data in a way better suited for the ML model; and, second, to incorporate domain knowledge about the nature of the problem. Many of the feature expansion techniques in Chapter 3 involve feature templates.

For a simple example, consider a date attribute in the raw data. A feature template can transform it into three discrete features (year, month and day). If the month itself is meaningful, this transformation will make that information easily available to the ML. If the original date attribute is expressed as the number of days counting from a fixed date, the model will have to learn all the months by itself, which will require substantive amount of training data and it will seldom generalize.

For a more complex example, consider the keywords4bytecodes project,[92] where I seek to predict the name of a Java method based on its compiled bytecodes. In this task, we could start adding features of the likes of "there is an **iadd** operation in the method," but that is very tedious and labour intensive. Instead, we can use feature templates to extract features from the data. If we take a particular instruction (e.g.,"`getfield org.jpc.emulator.f.i`") and generate three features: the instruction ("`getfield`"), instruction plus operand ("`getfield_org.jpc.emulator.f.i`") and instruction plus abbreviated operand ("`getfield_org.jpc.emulator`"), we can obtain a very large feature set as a first approximation. Further FE will be needed to reduce the set to a manageable size, if needed.

1.6 Discussion

In this otherwise very practical book, I would like to spend a paragraph on my opinions of the future of ML and FE. In a similar fashion that it is possible to trade time for space in many algorithmic problems, using more RAM to speed up the algorithm, it is my opinion that FE can trade domain knowledge for training data. Does that qualify for a learning machine? If you are helping the algorithm so earnestly, it is difficult to argue this is a reasonable direction for creating artificial minds (I think it is not). The interesting thing about the field

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

of AI is that in many aspects, it defines the boundaries of the computer science field. In the 1950s, search was squarely part of AI. Finding a node in a tree was considered an AI problem. Nowadays, nobody will put the two problems together. In the same vein, empirical nonparametric statistical modelling using heavy doses of domain knowledge, for example, through FE, might soon be an accepted method to solve many real-world problems. There are many real-world problems, such as in medical research, where collecting more data or wasting data on uninformed models is unacceptable. Whether these techniques remain to be called ML or not it remains to be seen. This is my view on the topic, the convergence towards an engineering contribution of the techniques presented in this book.

Looking ahead to the case studies and Part Two, I would like to reflect on a few issues that can be distilled from the work presented in those five chapters. The first is the fact that FE does not always succeed in improving the performance of the ML process. Embarking on a FE process is prone to hit dead-ends and regressions, something I have tried to exemplify throughout the case studies. Omitting the dead-ends and regressions I encountered working on these datasets would have been straightforward, but it will paint FE in unrealistic terms. **The fact that a particular technique did not work on a particular problem should not discourage you from trying it in your problem. The techniques showcased are valuable tools with a long track of success in the field.**

Second, there is the intertwining of FE and ML algorithm optimization, where ML optimization includes choosing the algorithm and tuning its hyperparameters. For the sake of covering more FE techniques, the case studies did not drill down into the ML algorithm optimization topic. Many applied ML books cover that topic quite well. The finer details of hyperparameter tuning jointly with FE are left cursorily discussed in the case studies. This issue is not only about obtaining better ML results. It does change the potential direction on which the FE process revolves around, as it changes the errors made by the ML. If the ML results change, it requires redoing error analysis. **Thus, error analysis for** FE **should be done at key points were the ML algorithm and its hyperparameters have been determined to have reached a local optimum.** And no technique can be expected *a priori* to consistenly outperform other techniques, what is called normally a "no free lunch" theorem.[332]

This brings us to the last topic I want to discuss: the human-in-the-loop nature of the FE process and experimental turnaround delays. When doing experiments for a publication or R&D for a production system, I would leave a hyperparameter search running for a week or use a cluster with hundreds of cores to speed it up. In a book format, my target has been to

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

have all case studies for a chapter run in a day, two days tops. Given the number of featurizations explored per chapter, that has limited the exploration of hyperparameters, as mentioned. But there is another, deeper issue with experimental turnaround time.† It has to do with the mental focus of the practitioner. If the experimental turnaround time is in the order of minutes, then you can wait, see the results and proceed. Larger turnaround times will imply a context switch on your side. You will go work on something else for the period of time that the ML training and tuning takes place. The longer the turnaround, the harder it will be to reconnect with your own thoughts and ideas regarding the FE process at hand.

Luckily, many of the FE tasks can build on work from earlier versions. As hinted in Section 10.1 in Chapter 10, it is possible to reuse computation successfully in FE. Some recent work on this direction includes systems built by Anderson and Cafarella[10] that adapt DB technologies to detect the parts of the train set changed by a proposed FE operation. Their system then updates a trained ML model only over those values. This requires ML models that are updateable and will not work on techniques like dimensionality reduction that produce substantive changes to the training data but it is an exciting new development in the field.

I would like to close with the following advice: try simpler ideas first. In the case of FE, that means not doing any FE at all. If your problem is similar to problems successfully addressed by DL and you have comparable training data, start with DL. If you do not have enough training data but can adapt pretrained models, then also try DL. If DL fails for you or you do not expect it will work given your problem and data size, then try AutoML frameworks such as Featuretools (discussed in Chapter 5, Section 5.4.1.2). At that stage, you can start FE, maybe on top of DL pretrained feature extractors and/or AutoML features. And do not forget to reach out to domain experts. They can save you tons of time and training data. Good luck, and I look forward to hearing about your experiences with feature engineering!

1.7 Learning More

FE is not covered in depth in regular ML books. Luckily, new books are being published that focus primarily on FE, some of which we will talk about here.

† This discussion is informed by a debate I had with Dr. David Gondek in 2015. He was arguing the ideas I talk here. At that time I was arguing against them. It took me some time to come around. Thank you, David.

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press

One of the earlier books is Alice Zheng's *Mastering Feature Engineering*,[351] which puts forward a mathematical formalism focusing in feature selection. It does a very good job explaining very complex mathematical concepts to a general public. The book has been expanded with the help of a coauthor, with more introductory chapters suitable for data science bootcamps (the book also has an emphasis on linear methods).[352] I highly recommend their appendix on linear algebra if you are missing that background, as the presentation is excellent and it is written from a FE perspective. The PCA chapter is also a great resource. Regarding the rest of the book, the emphasis on linear methods enables clear intutions but it concerns me that some of their intuitions only hold for linear methods and that might be lost to beginners. You might want to be extra careful and double-check their assumptions when reading it.

Dong and Liu[132] present a larger body of work than this current book, including 14 chapters authored by almost 40 different authors. It includes separate chapters on featurization for common data types (similar to the case studies in Part Two, but only focusing on featurization) and three case studies (without source code) on three novel domains (social bot detection, software analytics and extracting features from the Twitter API). Some of the chapters in the collection are quite introductory and others include advanced topics in active research. I have referenced the relevant chapters in this book, and I would advise an advanced and serious reader on FE to look at them in detail.

Regarding ML workflows and practical matters in ML, I recommend *Machine Learning for Hackers* by Drew Conway and John Myles White;[74] *Mahout in Action* by Owen, Anil, Dunning and Friedman;[249] *Real-World Machine Learning* by Brink, Richards and Fetherolf.[49] I am particularly fond of the last book and I have cited it quite a bit throughout this book. For a whole book on case studies, you might want to see *Fundamentals of Machine Learning for Predictive Data Analytics: Algorithms, Worked Examples, and Case Studies* by Kelleher, Mac Namee and D'arcy.[177]

In terms of EDA, I recommend *Doing Data Science: Straight Talk from the Frontline* by Schutt and O'Neil, as it includes EDA for many domains and problems in almost every chapter. The series of interviews with practitioners will prove valuable to anybody in the field, beginners and experts alike.

Finally, a great source of white papers related to practical ML lies in the KDD Cup series of ML competitions.[51]

https://doi.org/10.1017/9781108671682.003 Published online by Cambridge University Press