2

Features, Combined: Normalization, Discretization and Outliers

This chapter discusses feature engineering (FE) techniques that look *holistically* at the feature set, that is, replacing or enhancing the features based on their relation to the whole set of instances and features. An ongoing issue in FE is how to provide more value to features by leveraging contextual information. In this chapter, we use the whole dataset to provide the context. The central question is how to consider all the features together. Note that humans, when faced with unfamiliar data, will usually look at the behaviour of a given feature value through all instances to gauge its impact. How does this value compare to others? Is it representative? Is it rather small? Rather large? The techniques described here seek to incorporate this type of intuition to the machine learning (ML) process.

The most common approach is to **scale** and **normalize** the feature values (Section 2.1), finding the maximum and minimum and changing the values to ensure they will lie in a given interval (e.g., $[0, 1]$ or $[-1, 1]$). The expectation is to let observed meaningful variabilities emerge without being drowned by the difference of behaviour for each different feature. Note that names such as "normalization" and "standardization" are very ambiguous, as they may mean different things for different communities and people. For example, calculating BMI (body mass index) is considered "standardization" by their community of users, when it is not a normalization technique in the sense used here (it is a great computable feature, though).[171] When in doubt, ask for the precise formula being used.

Another type of FE technique called **discretization** is discussed in Section 2.2 and involves dynamically finding thresholds to segment continuous features into intervals or categories: for example, to decide that all temperatures between -10 F and 50 F fall into a bin (which could be called "cold"), and between 80 F and 110 F fall into another bin (which could be called "hot")

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

and so on. This way, you signal to the ML that differences between 15 and 45 degrees are not important for your task and domain.

Section 2.3 discusses **descriptive features**, that is, the use of compact summary statistics. This information has the advantage of always being defined (no missing values) and of producing dense feature vectors. We will look into using tables of counts (**histograms**) in Section 2.3.1 and general descriptive features such as maximum, minimum and averages in Section 2.3.2. Histograms are a tool of choice in computer vision and text length tends to be a very informative feature in natural language processing.

When looking at the feature values across many instances, some values might present themselves far from the rest, which is what constitutes **outliers** (Section 2.4). The chapter closes with advanced topics, including using feature differentials as features and inducing features from Random Forests.

Note that many of the techniques presented in this chapter (if not all) fall into the category of what has been called "model-based FE"[352] that is, after the FE process, the featurizer will contain trained models and parameters. To use the trained ML model over new data, you will need to preserve the featurizer model and make it available with the trained ML model.

2.1 Normalizing Features

Features in isolation can be difficult to compare with each other. Certain ML algorithms perform simple mathematical operations with them that assume their values are comparable. Other algorithms try to find ways to make them comparable using simple approaches. That means that if two features (say, flight length in kilometres and passenger weight in kilograms) are of widely different dimensions, the ML algorithm will require training data just to learn to scale them accordingly. That unnecessary burden can be alleviated by normalizing the features in various forms. Just remember that by changing the features you might lose information. Whether that information is important for the task depends on your understanding of the problem and domain. Thus, these decisions are also places to add domain knowledge. Nevertheless, there are times where the unnormalized value is relevant; in those cases, it is beneficial to maintain the original feature as a separate feature. Ultimately, these techniques drop information; if this information were important for the algorithm, it will make worse predictions.

The simplest approach is to scale the features so all the feature values have the same magnitude and are centred on zero. But there is also plenty of

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

value in normalizing them according to their mean and standard deviation, so that they have a unit variance (**standardization**). Other approaches involve normalization according to a norm,[351] for example, L_2. We will also look into standardization and decorrelation techniques (Section 2.1.1), smoothing (Section 2.1.2) and feature weighting (Section 2.1.3).

Feature normalization is a great way of reducing the variations on feature values, that is, to have less **nuisance variations** and concentrate on variations that contain a signal to the target class. It also ensures that the feature values fall within a specific range during training. For example, to transform the features into floating-point numbers between 0 and 1 (or -1 and 1). This is crucial for support vector machines (SVMs) and neural networks (ANNs) that need input data scaled to certain specific intervals.[180] On the other hand, many ML algorithms are scale invariant (like decision trees) while others, like logistic regression, might have requirements of zero mean and unit variance if used with regularization. Care must be taken when working with sparse datasets. In the presence of a large number of zero entries, some of the techniques presented here, like centring, will change the zero entries to non-zero entries producing dense vectors that might be inefficient to use with many ML algorithms.[352†]

In general, when I talk about normalization, I am talking about normalizing against all instances in the training dataset. Normalizing against subsets is described as an *instance* engineering problem in Section 5.2 in Chapter 5. After the normalization parameters are computed over the training set, they are applied at runtime (and to the test set).[351] The parameters computed over the training set ought to be fixed, and may generate outputs on the test outside the desired range (for example, values over 1.0). It might be tempting to recompute the parameters on the test set, but doing so will give an incorrect portrayal of the behaviour of the algorithm in production. Normalizing the test set independently has been described as the most common amateur mistake in SVMs.[209] The normalized features could, however, be clipped to conform to the chosen interval (by taking the maximum or the minimum).

Scaling. The simplest normalization that can be done is to take the maximum and minimum values in the population, subtract the minimum to each value, then divide by range (maximum minus minimum):

$$x'_f = \frac{x_f - \min_{\hat{x}_f \in \text{trainset}}(\hat{x}_f)}{\max_{\hat{x}_f \in \text{trainset}}(\hat{x}_f) - \min_{\hat{x}_f \in \text{trainset}}(\hat{x}_f)}$$

† Chapter 2.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

This will result in a value scaled to (0,1). For example, for the feature values $\{5, 10, 2, 17, 6\}$, their maximum is 17 and their minimum is 2, with a range of $17 - 2 = 15$. The scaled values will be $\{{}^{(5-2)}/_{15}, {}^{(10-2)}/_{15}, {}^{(2-2)}/_{15}, {}^{(17-2)}/_{15}, {}^{(6-2)}/_{15}\} = \{0.2, 0.53, 0, 1, 0.27\}$.

This scaling is used with support vector regression in Chapter 6, Section 6.3. This approach has the problem that outliers might concentrate the values on a narrow segment, so when doing scaling it is recommended to perform outlier filtering first (Section 2.4). Alternatively, you can use standardization instead, as discussed later in this section.

Squashing functions, such as $\log(1+x)$ or the Box–Cox transformation, are sometimes called nonlinear scaling.[341] We will see these functions with other computable features in the next chapter (Section 3.1). While these functions squash the distribution of values, they do not take into account the whole distribution to perform the squashing, which is the focus of this chapter.

Centring. After scaling, it is also common to add or subtract a number to ensure a fixed value (e.g., 0) is the "centre" of the values. The centre might be the arithmetic mean, the median, the centre of mass, etc., depending on the nature of the data and problem. A reasonable new centre might be zero, one or e, depending on the nature of the ML algorithm. In general, you centre your data to align it with attractor points in the ML parameter space: if the origin is the default starting point for your ML parameters, it makes sense that your data has zero as the most representative value. Values scaled and centred to the (scaled) mean are called **mean normalized values**.[342]

For the previous example, as the scaled mean is 0.4, the mean normalized values will be $\{ -0.2, 0.13, -0.4, 0.6, -0.13 \}$. For a more comprehensive example, see Chapter 6, end of Section 6.2.

Scaling to Unit Length. This is a normalization method applied to multiple features at once. Given a norm definition, divide features by the result of calculating the said norm

$$\vec{x}' = \frac{\vec{x}}{\|\vec{x}\|} = \left\langle \frac{x_1}{\|\vec{x}\|}, \ldots, \frac{x_n}{\|\vec{x}\|}, \right\rangle$$

For example, given a feature vector $\langle 2, 1, 3 \rangle$, its Euclidean mean is $\sqrt{14}$, therefore, the unit-length scaled feature vector becomes $\langle {}^{2}/_{\sqrt{14}}, {}^{1}/_{\sqrt{14}}, {}^{3}/_{\sqrt{14}} \rangle$. The type of norm to be used is dependent on the type of features. Usually the Euclidean or L_2 norms are employed but histograms sometimes use the L_1 norm (also known as **Manhattan distance**). We will see norms in detail as a regularization technique in Chapter 4, Section 4.2.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

2.1.1 Standardization and Decorrelation

We will now look into operations over the whole dataset when seen as a matrix. For some of these transformations, you will need to estimate the **covariance matrix** of the training data. If you take the training data as a matrix M of size $n \times m$, for n instances and m features and centre it so that the mean of its columns is zero, you can then compute the covariance matrix as $C = {}^{M^T M}/_{n}$. This matrix can undergo a spectral decomposition into eigenvectors E and eigenvalues in diagonal D such that $C = EDE^T$. This is known as the principal components analysis (PCA) **decomposition** of the matrix M.

Standardization. This process transforms the features to have zero mean and unit variance. It is very useful for SVMs, logistic regression, and NNs, to the point that forgetting to normalize the data is considered one of the most common mistakes.[155]

Given the feature mean $\bar{x}_f$ and standard deviation σ_f, the standardized feature is defined as

$$x'_f = \frac{x_f - \bar{x}_f}{\sigma_f}$$

If you replace the denominator by the variance rather than the standard deviation, the normalization is called variance scaling[351] (not to be confused with the ANN initialization technique). Note that certain data will not play well with standardization, for example, latitude and longitude data.[171]

Decorrelation. For signals acquired from sensor data, it is common to have artifacts, like repetitions of the past data (or an echo in the case of acoustic data). Decorrelation is a technique to reduce such artifacts. If your understanding of the domain indicates that the relations of interest should *not* be of a linear nature, then linear influences between instances or features can be considered an artifact of the acquisition methodology (e.g., a voice from one speaker being captured from the microphone of another speaker, as discussed in the context of ICA in Section 4.3.6 in Chapter 4). Decorrelation is usually performed by discounting past versions of the data over the current data, which is a type of linear filter. We will see examples of such processes in the timestamped data case study, Section 7.6 in Chapter 7. Note that certain ML algorithms and techniques (like the dropout for ANN, discussed in Section 5.3, Chapter 5) need some redundancy to perform properly.

Mahalanobis Distance. A concept related to decorrelation is to scale the data using the inverse of the correlation matrix C:[217]

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

$$\text{Distance}_{\text{Mahalanobis}}(\vec{x}, \vec{y}) = \sqrt{(\vec{x} - \vec{y})^T \, C^{-1} \, (\vec{x} - \vec{y})}$$

If the instances are decorrelated, then C is the identity matrix and this distance is equivalent to the Euclidean distance. You can either decorrelate your instances or use the Mahalanobis distance in places you would normally use the Euclidean distance. A graphical interpretation of this distance has the intuition that the covariance is *stretching* the data in the direction of areas of higher covariance. By applying the inverse of the covariance matrix, the stretch is rectified.[45]

Whitening and ZCA. Standardization changes the data to have a unit variance; decorrelation removes correlation between the variables. Would it be possible to obtain both? That process is called **whitening**, as it transforms the data into white noise. Sphering transformation is another name for whitening, even though it is a linear transformation. Again, whitening destroys the linearities present in the data. You will only apply this process if you believe the signal in your domain is expressed through nonlinearities; otherwise, training on perfectly random data simply does not make sense.

Given the PCA decomposition discussed at the beginning of this section, the PCA whitening is $W_{PCA} = D^{-1/2}E^T$. The math is a little tedious but multiplying M by W_{PCA} obtains a matrix whose correlation matrix is a diagonal matrix (the cross-correlations are zero). Because the properties of a whitening transformation are invariant over rotation, there are infinite such transformations,[291] many of which are singled out for their special properties. This means that $W = RW_{PCA}$ with R orthonormal will equally be a whitening operation. Among them, **ZCA**[30] (also known as the Mahalanobis transformation) uses E, the eigenvector matrix, to be the R, then

$$W_{ZCA} = EW_{PCA} = ED^{-1/2}E^T = C^{-1/2}$$

ZCA has the property of obtaining a transformed data as close as possible to the original data. This is advantageous for computer vision as the transformed images will still resemble the sources images. For example, Figure 9.3, reproduced in Figure 2.1, shows a whitened version of satellite images. See Chapter 9 for details. Many ML problems, however, will benefit from whitening with any transformation. For those, the whitening based on PCA is a popular choice, as PCA is easily available in most statistical packages.

2.1.2 Smoothing

If your feature appears perturbed by errors unrelated to each other, then particular errors might push the observed value of the feature too far from

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

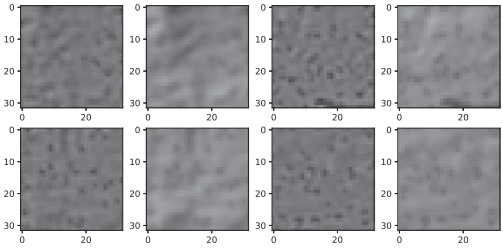

Figure 2.1 ZCA whitening results for eight settlements. The figure to the right is the whitened version of the figure to the left.

its true value and to the point of no longer being useful to the ML. To bring the value closer to its true value, you can compare it to other feature values in the *vicinity* of the instance and move it closer to other feature values in that vicinity. The intuition is that the aggregated independent errors will cancel each other out and the true signal will emerge. This process is called **smoothing**. Jeff Leek, in his data organization book, describes it as "one of the oldest ideas in statistics."[204]

What constitutes a "vicinity of the instance"? In the simplest case, if the instances contain a time or location, then the features could be smoothed taking a vicinity of instances, which are physically or temporally closer. This is succinctly put by Jeff Leek as "**smooth when you have data measured over space or time.**"[204]

Otherwise, you can consider the feature as a missing value, compute an imputed value per the techniques in Section 3.2 in Chapter 3 and average the imputed value with its existing value.

2.1.2.1 Probability Smoothing

A special type of normalization is used when dealing with sparse events where there are reasons to believe your training data constitute an imperfect sample, missing many naturally occurring combination of feature values. In those circumstances you want to discount your observed values and leave counts (probability mass) for unobserved events (feature value combinations). This smoothing is used with ML algorithms based on probabilistic methods, as the probability mass held aside needs to be included as input to the algorithm. It deals with unseen values at the sampling level, while in the next chapter (Section 3.2) we will see imputation, which is the problem of dealing with missing values explicitly marked in the training data.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

The most common situation to smooth is dealing with one feature value that never appears together with a particular value of the target class. Without smoothing, certain algorithms like naive Bayes end up unable to produce any results as their core multiplication always results in a joint probability of zero.

However, how many unseen events are out there? How will you distribute the probability mass among them? If possible, you can use your understanding of the nature of the events (domain knowledge). For example, if the events are word pairs, you can estimate how likely an unseen pair will be, given how frequent the words are that make up the pair.[220]

Simple Smoothing. Different smoothing techniques are widely in use, including Lagrangian smoothing (every new unseen event is considered to have occurred at least once), ELE smoothing (add 0.5 to all counts) or Add-Tiny smoothing (add a very small number to all counts). These simple smoothing techniques are quite straightforward, although I have seen Lagrangian smoothing consistently outperform Add-Tiny over natural language processing (NLP) data and there are theoretical reasons to believe it should.[220]

Simple Good–Turing. The problem with simple smoothing techniques is that they overestimate the amount of probability mass to keep aside. The final systems are then timid, not trusting their observed data enough, due to the uncertainty of the unseen data. A more complex technique involves fitting a curve of frequency of frequencies to the observed data, and then using the curve to estimate unseen events. A probability distribution is said to be of Zipfian nature if it can be expressed as the following:[236]

$$p(f) = \alpha f^{-1-\frac{1}{s}}$$

where α and s are parameters that define the distribution. Such distributions explain many events of interest, such as distribution of words. If you believe that your probability distribution is Zipfian, then you can decide what amount of probability mass to set aside in a more informed manner.[117]

2.1.3 Feature Weighting

For the last technique of feature normalization, let us discuss using weights computed or adjudicated for features in all instances, basically, adding a statistical prior to the different features.[180] This is particularly advantageous if this prior is rich in domain knowledge and you do not expect the ML algorithm to be able to reconstruct it from the available data. Therefore, if you

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

expect certain features to be more informative, you can weight them higher by multiplying them by a number. Alternatively, you can provide a feature-weighting vector directly to some ML algorithms.

While feature weighting can be used to add some meta-learning capabilities to a simpler ML (for example, by weighting up features highly correlated with the target class[55]), I find such techniques more related to ML algorithms than FE. An ideal feature weighting ought to bring information that exceeds the information available in the training data. You ought to be weighting a feature because of things you know about the domain, beyond the particular training data.

2.1.3.1 Inverse Document Frequency Weighting

One of the most popular feature-weighting schemes in the NLP world is known as **TF-IDF**, where in a Bag-of-Words representation of text (a word histogram, discussed in Section 2.3.1) the features (word counts) are scaled down by how popular the words are in a large collection of documents. A collection used to compute the IDF scores can exceed in many orders of magnitude the available training data, thus providing general knowledge about which words are too popular to be informative.

The weighting scheme has many flavours (aptly summarized on Wikipedia[329]); a popular one is

$$\text{idf}(t) = \log\left(\frac{N}{n_t + 1}\right)$$

where N is the total number of documents in the corpus and n_t is the number of documents where the term t appears (irrespective of how many times the term t appears in each document). When featurizing, if the term t appears $\text{freq}(t)$ times on the raw data (text) for a given instance, then setting the feature x_f (where f is the feature index associated with t in the feature vector) as $x_f = \text{freq}(t) \times \text{idf}(t)$ accomplishes this weighting scheme. For example, a word like "the" is very common; therefore, n_{the} will be very high and idf(the) will be very small. Compare that with a word like "scheme" that ought to appear rarely and will have a much higher IDF weight. Note that to use TF-IDF directly as a text-similarity metric you will need to penalize longer documents.[285] In the case study on textual data in Chapter 8, we use the IDF scores to combine the embedding representation for all the words in a Wikipedia page for a given city as extra features to calculate its population.

2.1.3.2 Camera Calibration

In computer vision, a common weighting scheme is camera calibration:[297] often optical elements and sensors exhibit variabilities in a predictable manner,

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

measured by capturing data for a calibration image. The calibration data obtained can then be turned into a feature-weighting scheme. For example, a camera might produce lighter pixels on its upper-left corner. That might confuse the ML algorithm if lighter pixels are a good signal for a particular class elsewhere in the image. By reducing the luminescence of the pixels in the upper-left corner, the effect will be reduced.

2.2 Discretization and Binning

Instances in the training data and their features are models of events and entities from reality. The features are always a simplification of reality. That simplification often poses challenges to ML. However, there are times where there is value behind simplifying a representation even further. That is the type of processing performed in **feature discretization**: reducing the number of possible values a feature can take, usually from a real-value number to a discrete quantity such as an integer. It is a synonym of **quantizing**, different from the term in physics. In its most common form, it refers to changing a continuous feature into an (ordered) categorical feature. Other possibilities include transforming a real-valued feature into an integer-valued feature, or reducing the number of categories in a categorical feature (coalescing categories).

Every discretization incurs a **discretization error**, but you can expect that discretization will result in fewer parameters for some ML models, thus boosting the signal in the training data and improving generalization. Your number of parameters might increase, however, if your ML model of choice cannot accommodate categorical features directly and you resort to using one-hot encoding. Discretization is also useful for error analysis and understanding the behaviour of the system, enabling you, for example, to build summary tables.[239†] It also enables you to squash differences on the actual numbers; a lower quantile in one feature is comparable to a lower quantile in another.[352‡]

This operation can be done over the feature values alone, in isolation from the target class values, in what is known as **unsupervised discretization** or it can be done relative to the target class (**supervised discretization**). It usually involves finding thresholds on which to partition the data (univariate), but you can discretize on more than one feature at a time (multivariate).

In discretization, the intuition is to find quality boundaries on the data such that the number of feature values that fall between two boundaries is

† Chapter 4.
‡ Chapter 2.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

reasonably well distributed. You use this approach if you suspect that small differences between values in dense parts of the space should be taken more seriously than differences in sparse areas. Age is a great example: a difference of one year in age at age 5 is much more important than a difference at age 85. In discretization we want similar instances to discretize to the same value,[41] including, for example, to deduplicate multiple aliases for the same person.[49] The positive impact of discretization has long been established in the field:[191]

> Many machine learning (ML) algorithms are known to produce better models by discretizing continuous attributes.

2.2.1 Unsupervised Discretization

Reducing the resolution of a set of numbers without referencing other data involves finding an underlying structure in them. This means realizing that, for example, the values are grouped around certain centres of mass or appear spaced regularly at a certain distance. Finding such a structure is the goal of unsupervised learning. We will see some simple yet popular techniques known as binning before discussing general clustering (Section 2.2.1.2). Note that unsupervised discretization is usually vulnerable to outliers[213] (Section 2.4). Also, these unsupervised discretization techniques lose classification information as they might merge many points with different values of the target class into the same discretized feature value.

2.2.1.1 Binning

Simple discretization techniques seeking to split the segment of observed feature values into equal segments either in size or length are called **binning** and have a long history in statistics. This is also known as discrete binning or bucketing. The original objective of binning was to reduce the error of observations by replacing them with a representative value (usually the centre) for the small interval (bin) on which the value is located. Intervals can be chosen so they have all the same size or they have the same number of observed values on each interval (quantiles). Alternatively, a bin size can be chosen and the full set of real numbers can be split into an integer number of intervals of the same size (rounding).

In the case study in Chapter 6, the target variable is binned using an equal frequency interval approach (Section 6.3.1.2). For 50,000 cities, their population range from 1,000 to 24,300,000. Splitting into 32 bins using a logarithmic scale, the first bin has boundaries (1,000; 1,126), and the last bin

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

is (264,716; 24,300,000). The total discretization error incurred is 300 million (an average of 6.5% per city).

Equal Interval Width. This binning approach involves taking the range and dividing it into k equal regions. It is very sensitive to outliers, so either remove them first or do not use them if you have many outliers. It is one of the easier approaches but if you know anything about your data, you can do better.

Equal Frequency Intervals. In this approach, you take m instances and divide them into m/k values (possibly duplicated). It helps when you have different levels of data density in different regions of the possible values. This approach is useful when equal intervals will result in many empty bins due to clustering of points.[352†] It operates by sorting the values (including duplicates) and picking the boundary items at the m/k-th position. Alternatively, the values can be recursively divided using the median element, which will be discussed in Chapter 6.

Rounding. A straightforward way to fully transform a real number into an integer is to multiply the real number by a fixed number, and then round it to a whole number on a given base (or just truncate it). For example, given the feature values $\{0.7, 0.9, 1.05, 0.25\}$ with multiplier 3 and flooring on base 10, we will obtain discretized features $\{2, 2, 3, 0\}$. The multiplier and base ought to be chosen based on domain knowledge. Otherwise, unsupervised learning can help, but in that case you will be better served using clustering, discussed in the next section.

Winsorising (Thresholding). A simple way to binarize or coalesce ordered feature values is to apply a threshold over them. Values below the threshold become an indicator feature with a value of false. Otherwise the feature value is true.[196] The threshold can be chosen in the middle of the range (maximum value minus minimum value), which is the mean, the median or such that the target class is evenly distributed (in a supervised variant of this approach). Winsorising is important for some ML algorithms that can only operate over binary features such as certain implementations of maximum entropy or certain types of SVMs. With an appropriate threshold, it is possible to boost the signal present in the data.

† Chapter 2.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

Other Techniques. Maximal marginal entropy adjusts the boundaries so they decrease the entropy on each interval (this is a variation of equal frequency binning discussed earlier).[86] If the total number of different values in your data is small, you can bin by the exact number (e.g., observed value "27" becomes "observed-category-27") and use the resulting bins as categories.[55] Binning can also be applied to existing categorical values, basically coalescing categories,[171] a topic discussed in the next chapter (Section 3.1) in the context of computable features.

2.2.1.2 Clustering

The simple techniques already discussed reduce the feature space considerably and might enable the use of categorical-based ML algorithms over continuous data. Still, there are better ways to capture the underlying distribution of the input features: to apply unsupervised learning over the features. A common technique is to use k-means clustering (discussed next) and then use the number of the cluster ("Cluster-ID") as the feature category[41] (therefore, if you cluster using $k = 20$, you will end up with a feature with 20 categorical classes). Alternatively, it is possible to have the distance to each cluster (or the top clusters, to obtain sparse vectors) as a separate feature.[352] Chapter 7 uses this approach to build feature heatmaps, for example, Figure 7.3, reproduced here as Figure 2.2. The feature values for each feature are split into six clusters, visualized as different levels of gray. The feature heatmap allows for comparison of different historical versions of the same instance, or comparison across different instances. See Section 7.1.1 for the fully worked-out example.

The algorithm k-means[142] is based on the concept of synthetic instances: each cluster is represented by a centroid, a fictitious instance (the synthetic instance). Instead of computing the distances among all instances in the cluster, k-means computes just the distances to the centroids. It receives as parameters the number k of target clusters (which can be estimated using other

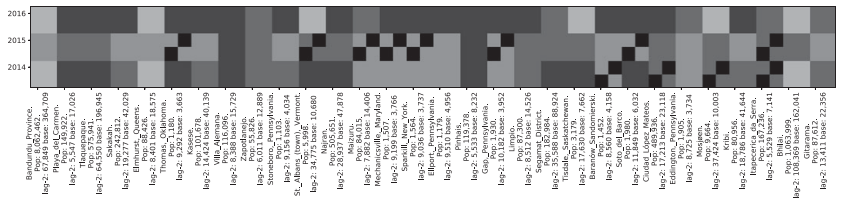

Figure 2.2 Historical features visualization using a feature heatmap. Different levels of gray indicate different feature clusters.

methods such as canopy clustering[226]), the distance function and a procedure to compute synthetic instances from a set of actual instances.

It starts by picking k instances at random as initial centroids (other initialization techniques are possible, for example, kmeans++[13]). In each iteration, it reclassifies each instance as belonging to the cluster that has its centroid closer to the instance. Then, for each cluster, it computes a new centroid based on the instances assigned to it.

This algorithm is an example of **expectation-maximization** (EM), a statistical estimation technique in which the algorithm switches between estimation and modelling steps.[80] Under certain circumstances, convergence is guaranteed, albeit quite slowly. Basically, you continue with the two steps until convergence, in one step you fix the model (assignment of each instance to a cluster), and estimate its parameters (the centroids) and in the other step you fix the parameters and estimate the model.

2.2.2 Supervised Discretization

Supervised discretization received plenty of attention in the 1990s and 2000s, most probably as the need to reduce parameter space was driven by memory bottlenecks. The different solutions proposed to the discretization problem can be organized around many facets and dimensions.[213] We will look into three algorithms of practical importance that are relatively easy to understand and implement and are also representative of many algorithms centred around similar ideas. These algorithms are ChiMerge, MDLP and CAIM. They all work on a single numeric feature, operating over its sorted distinct values.

What is the benefit of the added complexity of using supervised discretization? In the words of Alberto Bietti:[34]

> Unless you have good knowledge or intuition about the common values taken by the feature, hand-picked or equal-width intervals probably won't give good results.

Note that if you use the target class for discretization then you will need to find the boundaries over a held-out dataset that you will not be able to reuse in subsequent training. Otherwise, reusing the training data with the discretized features will result in a weak target leak: the ML will be misguided into trusting the discretized feature too much.

Discretization is a search problem where each algorithm is defined by (given candidate places to split into intervals) a criterion for determining how good a split is and a method to explore the space of partitions.[86]

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

Adaptive Quantizers. MDLP and CAIM, discussed next, both belong to the family of top-down **adaptive quantizers**, where the observed feature values are sorted and then split recursively, starting from an interval equal to the whole range. Each algorithm in the family has a different criteria to choose where to split an interval and whether to continue splitting. The candidate places (*cut points*) to split are the observed values or a "smart" subset of them. These are incremental algorithms and have the advantage that they do not require a number of internals provided beforehand. In its most computationally expensive version (and from which the name derives), you can train a full classifier on each partition and choose the cut point where the classifier performs better.[60]

2.2.2.1 ChiMerge

The ChiMerge discretizer[179] presents a bottom-up approach. Its starting point considers each observed feature value as a separate internal. At each step of the algorithm, it merges an interval with its neighbour depending on the χ^2 statistical test (discussed in the context of feature selection in Section 4.1.1 in Chapter 4) over the values of the target class associated with each interval. It merges an interval with its neighbour if the χ^2 test cannot reject the null hypothesis, that is, we cannot show the two sets to be statistically independent. Besides its simplicity and statistical justification, it can be applied to multiple features at once and can do joint discretization and feature selection.

2.2.2.2 MDLP

Fayyad and Irani[103] propose using the entropy of the target class to choose the optimal partition cut point. They define the entropy of the interval S as:

$$\tilde{H}(S) = -\sum_{i=1}^{k} \frac{\#(C_S = i)}{|S|} \log \frac{\#(C_S = i)}{|S|}$$

where k is the number of categorical values the target class can take and $\#(C_S = i)$ is the number of instances in S that have its target class equal to i. For a given cut point, we can define the split entropy defined as the weighted average (weighted on sizes) of the entropy for the two intervals obtained by splitting at the given cut point. If the splitting at a cut point is meaningful, the remaining intervals will have a more homogeneous class (which should facilitate learning and make the feature more informative). MDLP uses split entropy as the metric to score different cut points.

To speed up the algorithm, the authors prove a theorem that shows the only cut points that need to be considered are *class boundaries*. A class boundary is a feature value where the value of the target class has changed from its

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

adjacent neighbour. The number of class boundaries will hopefully be much smaller than the total number of observed feature values. As a stopping criteria, the algorithm uses a minimum description length (MDL) criteria that stops splitting when the description (in bits) for the class labels without splitting is shorter than encoding the labels for the two resulting intervals.

While the algorithm seems very similar to decision trees,[48] it produces a different discretization: decision trees are local methods while the discretization technique discussed here is a global method. Local methods suffer from "data fragmentation" where intervals may be split unnecessarily, producing suboptimal results.[16] Global discretization algorithms have been shown to help decision trees and outperform their embedded discretization functionality.[213]

2.2.2.3 CAIM

The CAIM algorithm[197] uses the mutual information between the feature intervals and the target class. The authors claim that it generates very few intervals. CAIM seeks to minimize the loss of feature-target class interdependency. It uses the confusion table ("quanta matrix" in the paper, see Figure 2.3) between the intervals found so far (starting with a single interval covering the whole range of observed values) and the different categories of the target class. From the confusion table they derive CAIM scores by looking at the maximum counts for a class on a given interval ($\max_r$), versus the rest:

$$\text{CAIM}(D) = \frac{\sum_{r=1}^{n} \frac{\max_r^2}{M_{\circ r}}}{n}$$

where the discretization D splits the feature into n intervals and $\max_r$ is the maximum value in the r-th column of the quanta matrix. The authors'

Class	Intervals					Class Total
	$[d_0, d_1]$	...	$(d_{r-1}, d_r]$	...	$(d_{n-1}, d_n]$	
C_1	q_{11}	...	q_{1r}	...	q_{1n}	$M_{1\circ}$
⋮	⋮	...	⋮	...	⋮	⋮
C_i	q_{i1}	...	q_{ir}	...	q_{in}	$M_{i\circ}$
⋮	⋮	...	⋮	...	⋮	⋮
C_S	q_{S1}	...	q_{Sr}	...	q_{Sn}	$M_{S\circ}$
Interval Total	$M_{\circ 1}$	...	$M_{\circ r}$	...	$M_{\circ n}$	M

Figure 2.3 CAIM quanta matrix, adapted from Kurgan et al.[197] It is a confusion matrix for a feature split into n intervals, for a target class with S categories.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

expectation is that "the larger the value of CAIM the higher the interdependence between the class labels and the discrete intervals."

The cut point with the highest CAIM score is chosen at each iteration. If no segment produces a CAIM score higher than the larger CAIM encountered (after k steps, where k is the number of values the target class can take), it stops. In their evaluation of 30 discretizers,[118] García and colleagues conclude:

> CAIM is one of the simplest discretizers and its effectiveness has also been shown in this study.

2.3 Descriptive Features

Sometimes the ML does not need the actual data but only a compact statistical summary: the task can be solved knowing general characteristics about the shape of the data distribution. The general term to refer to such indicators is **descriptive statistics**. We will see histograms next, the most common descriptive features, and then present other descriptive features in Section 2.3.2. This approach is particularly important when an instance contains a large number of similar and low information features, like pixels or other sensor data.

Dense vs. Sparse Feature Values. Using descriptive features has the advantage of producing reliable, dense features. For example, if a user review is in a language different from the language used in the train set, a text-length feature might still be informative while any indicator feature for particular words will most probably be all zeroes. Dense feature vectors, however, might slow down considerably ML algorithms designed to operate over sparse feature vectors, like SVMs.

In their winning submission to the KDD cup (cf., Section 1.7 in the previous chapter), Yu and others[341] combined two systems with different approaches, one using binarization and discretization, which produced a very sparse feature set, while the second approach used simple descriptive statistics and thus produced a dense feature set. Their results show the value of the dense approach.

2.3.1 Histograms

Another way to obtain a holistic view of the features is to summarize their behaviour by means of a **histogram**. A histogram is a simple representation of the distribution of values for a set of features. It is amply used in image

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

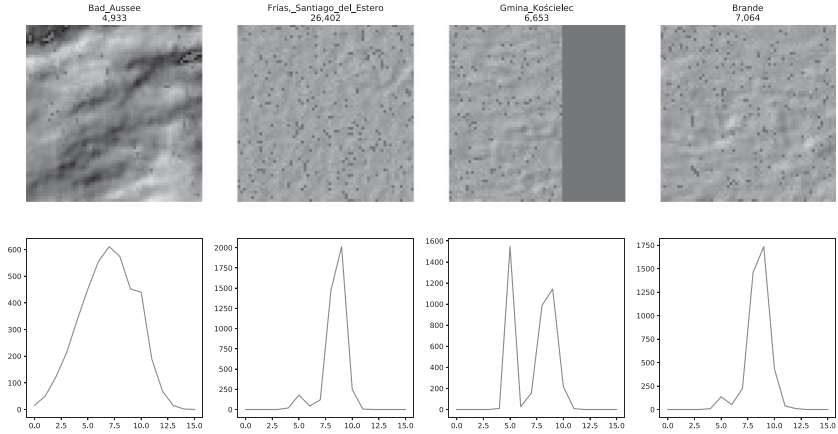

Figure 2.4 Four random settlements and histograms. The histograms show that Bad Aussee is in the Alps, Gmina Kościelec downtown is next to a small pond and the other two places are relatively flat.

processing. It transforms feature values to frequency values and can handle continuous data by using bins to define each of the frequency columns. The original feature vector then has multiple features of the same type. The histogram feature vector has one feature per bin with its value equal to the number of features in the original feature vector that fell into that bin.

This is a common technique when dealing with images. Chapter 9 uses histograms for image classification, Figure 9.5, reproduced here as Figure 2.4, shows how different images can have similar histograms (Frias and Brande in the figure), which can help reduce the nuisance variations in the domain. See Section 9.5 for the fully worked-out example. If you have a large number of related but otherwise identical features, instead of (or in addition to) using the features themselves, you can use the number of times a particular feature value appears in that large collection of features. In the case of a 300×300 black-and-white image, you can represent the whole image as two features: number of white pixels and number of black pixels (the simplest possible histogram). That reduces the number of features from 90,000 to 2 and it might contain enough information to solve a variety of classification problems (e.g., indoors vs. outdoors). Or, if you represent each pixel with 30 discernible shades of gray (5 bits), that accounts for 566,250 bytes of input vector. A histogram then contains 30 entries with a maximum value of 90,000 per entry (17 bits), for a total of 64 bytes. Now, if the problem is solvable from the histogram that is a reduction of almost 9,000 times.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

It is interesting to note that, given enough training data, many ML algorithms will learn a decision function that internally computes the histogram or a close approximation to it. If you believe the histogram is related to the problem you are trying to solve, you can spare the ML algorithm that unnecessary burden.

Bag of Words. A theme of this book is to look at techniques used in different fields and abstract them with the hope that you can apply them to new fields of your own. In that vein, while histograms are usually associated with computer vision, it is worth noting that a popular representation in NLP, bag-of-words (**BoW**), is a histogram of words. In this representation, a piece of text is presented to the ML as a fixed vector of size equal to the whole vocabulary observed during training. The feature values indicate the number of times the given word appears in the text corresponding to a given instance. Chapter 8 uses the bag-of-words representation, which for the sentence

> Its population was 8,361,447 at the 2010 census whom 1,977,253 in the built-up (or "metro") area made of Zhanggong and Nankang, and Ganxian largely being urbanized.

would appear to the ML as the token counts:

> ['its': 1, 'population': 1, 'was': 1, 'TOKNUMSEG31':1, 'at':1, 'the':2, 'TOKNUMSEG6':1, 'census':1, 'whom':1, 'TOKNUMSEG31':1, 'in':1, 'built':1, 'up':1, 'or':1, 'metro':1, 'area':1, 'made':1, 'of':1, 'zhanggong':1, 'and':2, 'nankang':1, 'ganxian':1, 'largely':1, 'being':1, 'urbanized':1, ... rest 0]

See Section 8.4 the fully worked-out example.

2.3.2 Other Descriptive Features

You can consider that the histogram is a particular type of summary for a set of data. More types of summaries are available, including the maximum, minimum, mean, median, mode, variance, length and sum.

Other descriptive features can be obtained by assuming a distribution of feature values. Then, for the values in the raw data for a particular instance, you can compute how close to the assumed distribution they are. By far the most popular distribution is the normal distribution, but other distributions are possible (good candidates include Poisson, bimodal distributions and distributions with "fat tails"). Assuming the data follows a normal distribution, the standard deviation captures a known percentage of the data as it measures

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

the spread of the dataset from the mean. Other metrics related to the normal are possible, here, I discuss skewness and kurtosis.

Skewness. This statistic measures the lack of symmetry in the distribution:

$$s = \frac{\sqrt{N\,(N-1)}}{N-2}\frac{\sum_{i=1}^{N}(Y_i-\overline{Y})^3/N}{\sigma^3}$$

This feature is to be applied similar to the histogram feature, given a large number of similar values in the raw data, the skewness value for them becomes the feature. A distribution with a high skewness will have the bulk of its elements to only one side of the mean.

Kurtosis. This statistic measures whether the data is heavy tailed compared to a normal distribution:

$$k = \frac{\sum_{i=1}^{N}(Y_i-\overline{Y})^4/N}{\sigma^4} - 3$$

Heavy-tailed distributions are common in human data and when mismodelled with normal distributions produce a large number of outliers, which is discussed in the next section. This feature is applied similar to skewness.

Quantiles and Percentiles. These numbers summarize the distribution by indicating the boundaries over which the bulk of the points fall, dividing them into equal number of segments. Q2 is the mean, the segment (Q1, Q2) has the same number of elements as (Q2, Q3) and ($-\infty$,Q1), (Q3, $+\infty$). Percentiles are similarly defined over 10 segments.

Text Length in NLP. Descriptive features are not known by that name for all domains and problems. A very common (and successful) feature in natural language processing is text length. It predicts correctly many classes of interest. For example, is the customer happy or unhappy? Unhappy customers tend to leave much longer reviews, with plenty of details to justify their unhappiness. Note that text length is the L_1 norm of the bag-of-words (the word histogram) if there are no out-of-vocabulary words present. Otherwise, text length is more informative.

Other Descriptive Features. In the general case, you can compute the KL divergence (cf., Section 4.3.7 in Chapter 4) between all similar features in the instance (e.g., pixels) and the full probability distribution induced by all the instances. Such feature will tell the ML how "likely" the features are, as compared to the data seen during training.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

2.4 Dealing with Outliers

In his book *The Analysis of Time Series,* Chris Chatfield[61†] mentions that

> The treatment of outliers is a complex subject in which common sense is as important as the theory.

Dealing with outliers might be the most domain knowledge–intensive task in FE. You can only drop outliers if you have a story behind them that can assure you they are invalid observations due to the idiosyncrasies of the domain. Indeed, the key issue in dealing with outliers is to differentiate errors from extreme observations. When analyzing the values for a feature, it is not unusual to find values (or small clumps of values) that clearly depart from the rest. The temptation is to remove these values, either the values themselves or to throw away the full instance, as there are few of them and the ML might work better if it focused itself on the more frequent "normal" cases. However, Dorian Pyle in his book *Data Preparation for Data Mining* exemplifies:[263] in insurance, most claims are small but a few are very large. Obviously removing the very large claims will completely invalidate an insurance model.

That is not to say that there is no value in realizing your data presents a large number of outliers; we will see outliers detection briefly at the end of this section. But if outliers are present, you might be restricted to ML algorithms that are robust in their presence. For instance, estimation for covariance matrices are very sensitive to outliers, and so is the mean (in the mean case, you can replace it with the median if there are many outliers).

For some examples of outliers, when working with sensor data, some equipment will produce rather large peaks on their signal when cables are being plugged in or the acquisition system is being switched on. Alternatively, you might know from domain knowledge that the encountered value is impossible (e.g., a patient of 999 years of age). Other examples include: a clerk making an error on entry, credit card fraud, unauthorized access, fault detection on engines, new features on satellite images and epilepsy seizures detection.[151]

From my perspective, outliers make it easier to identify potential data mistakes,[49] but you will have to do the legwork to find whether it is truly a mistake or not. Even though I am repeating myself, I will insist: do not throw out outliers unless you have definite evidence that these are erroneous observations. In his book *How to Lie with Statistics,* Darrell Huff[162] shows that even in normally distributed data, it is common to encounter "suspiciously extreme" values. Dropping them, however, will mislead you away from the true

† Chapter 1.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

mean and standard deviation of the distribution. *That outlier you are about to throw might be one of your most informative instances.*

If you have determined that certain feature values are outliers, you can drop them and consider them to be **missing values**. Imputation techniques to deal with them are discussed in the next chapter, in Section 3.2. Unless you have a fully automatic method for identifying, removing and imputing outliers, leave them in the test set as-is, otherwise you will not have a correct appraisal of the behaviour of your ML model on production.

2.4.1 Outlier Detection

Outlier detection – also known as novelty detection, anomaly detection, noise detection, deviation detection or exception mining [151] – pertains to identifying outliers, either on a fully assembled dataset or on newly arriving data. Defining what is an outlier is a challenging task. Bannet and Lewis define them as: [27]

> An observation (or subset of observations), which appears to be inconsistent with the remainder of that set of data.

Identifying outliers is important beyond removing them, as data with many outliers might be indicative of a non-normal "fat tail" distribution. Also, the outliers you have received might be clipped or distorted data (what is known as **censored data**). You might want to train a separate model on the outliers, as sometimes they exhibit what is known as the **king effect**, [199] where the top representatives of the distribution behave very differently from the rest of the instances.

In the case of FE, outlier detection interests us for learning a model over the training data and using it to identify unusual instances and their feature values. In the general case, outlier detection techniques can be supervised or unsupervised. In the case of FE, spending extra effort labelling outliers is unlikely, so we will focus on unsupervised techniques. The main techniques are clustering, density estimation and one-class SVMs. When doing unsupervised outlier detection, there are diagnosis and accommodation techniques, where diagnosis just finds the outliers while accommodation seeks to make the ML robust in their presence.

Unsupervised discriminative techniques use a similarity function and clustering. These techniques define an **outlier score** as the distance to the closest centroid. Unsupervised parametric techniques only model the normal class, the new data is anomalous if its probability of being generated from the model is low. [133] Using k-means helps to distinguish sparseness from isolation (sparse

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

means a cluster with a large distance overall, whereas isolated means a single item with large distance to others). Points in sparse areas are not outliers, while isolated points are. Large-scale clustering techniques like BIRCH[346] and DB-SCAN[101] handle outliers explicitly and can also be used but they do not provide a degree of outlier-ness. An intriguing method by Japkowicz and colleagues[168] is based on autoencoders (discussed in Chapter 5, Section 5.4.2). As autoencoders tend to behave poorly on novel data, their reconstruction error can be employed as an outlier score. Similarly, pruned nodes in a decision tree can be used as an outlier detector or, as kernel-based methods estimate the density of the distribution, they can detect outliers by identifying areas of low density.[170] Extreme values theory[272] models the outliers directly as a particular distribution and uses EM to estimate its parameters, including thresholds. If the data has spatio-temporal components, it is possible to leverage special outlier detection techniques.[133]

Finally, outlier detection over human data can have ethical implications that you need to factor into your decisions.[102]

2.5 Advanced Techniques

Many techniques in the following chapters also consider how the features behave when combined, particularly dimensionality reduction (Section 4.3 in Chapter 4) and certain computable features, particularly target rate encoding (Section 3.1 in Chapter 3). I have chosen to discuss them closer to other related techniques, such as feature selection.

There are two other techniques I would like to discuss here: using leaves from a Random Forest as features (discussed in the next section) and delta features, discussed next.

Delta Features. Also known as **correlation-based features**,[193] they involve building a model for the behaviour of the whole feature to produce a new feature that indicates how different the observed original feature in a given instance is from the model for the feature.

The simplest case is to take the average and to encode the difference to the average as the feature (the astute reader will realize that this achieves the same result as to centre the feature so its mean is zero). More interestingly, you can replace the feature with how likely are the feature values. For example, each feature value can be replaced with the percentile in the value distribution (is this feature value the top 10% more common values? the top 25%? etc.). You can see an example of this in the case study in Chapter 7, Section 7.1.1.

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

A similar concept are **z-scores**, the number of standard deviations that feature value is from the mean:[239]

$$z = \frac{f - \bar{f}}{\sigma}$$

As with other complex FE techniques, delta features will result in a complex featurizer that needs to be available together with the trained ML model for production deployment. That is, the means, histograms, etc., obtained from training data are then needed to compute these delta-features on test data and ought to be kept available together with the ML model.

Random Forests Feature Induction. Vens and Costa[315] present an exciting way of benefiting from the combined set of features: train a Random Forest and use the different paths as produced by the forest as features:

> Intuitively, two instances that are sorted into two nearby leaves share more similar features than two instances that are classified into leaves far apart.

The new feature vector is a concatenation of binary features obtained for each tree in the trained forest. For each internal node in the decision tree, a binary feature is generated, indicating whether the condition at the internal node holds or not over the instance. Notice how different trees intermix and combine different features, and how the Random Forest is normalizing, handling outliers and producing descriptive features automatically for us. In a way, it encapsulates all the techniques described in this chapter and more. If you train the random forest with many trees, you will achieve an expansion of your feature set. Or you can train a few shallow trees and obtain a reduction. You can combine the original features plus the tree features too.

2.6 Learning More

This chapter has described techniques usually considered part of a data preparation pipeline. General books on the topic such as Dorian Pyle's *Data Preparation for Data Mining*[263] or Jeff Leek's *The Elements of Data Analytic Style*[204] can be a good reference for this. In the same direction, the article *Tidy Data* by Hadley Wickham[324] can be of interest. For available tooling for binning and outlier detection, you might want to look into OpenRefine (formerly GoogleRefine).[22]

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press

Discretization is a well-studied topic. I have referenced four surveys in this chapter,[86,213,191,118] and each of them is worth studying. I found the one by Dougherty and colleagues[86] to be the most succinct and easy to read.

Finally, outlier detection is a topic with many books written about it, from which *Outlier Analysis* by Charu Aggarawal[2] is a great resource. For a recent survey, see Hodge and Austin.[151]

https://doi.org/10.1017/9781108671682.004 Published online by Cambridge University Press