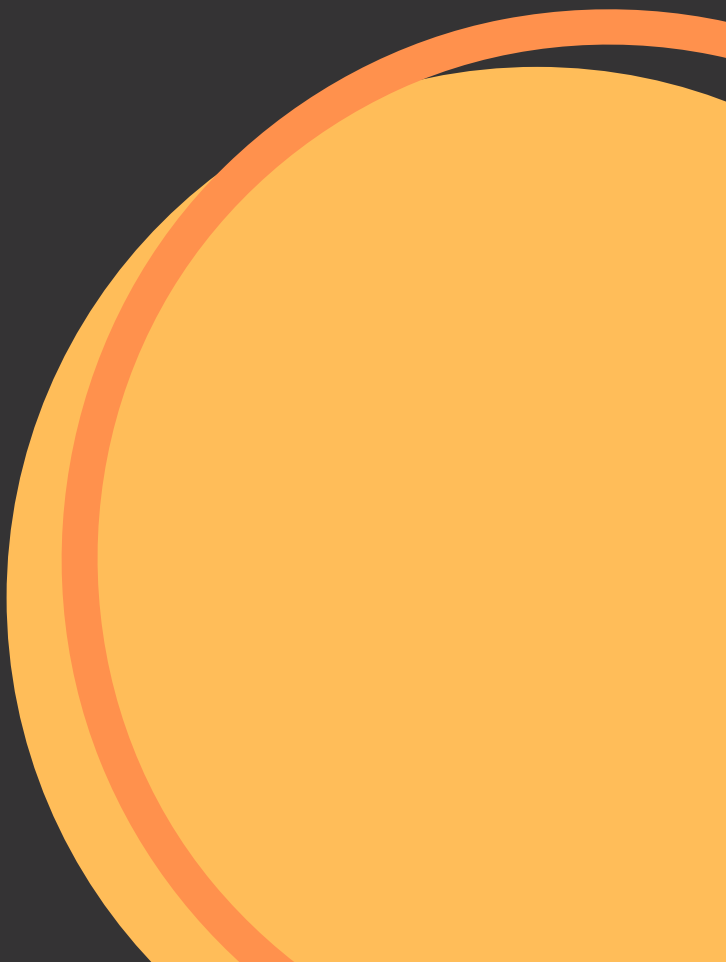
Andriy Burkov's

THE HUNDRED-PAGE MACHINE LEARNING BOOK



"All models are wrong, but some are useful."
— George Box

The book is distributed on the "read first, buy later" principle.

9 Unsupervised Learning

Unsupervised learning deals with problems in which your dataset doesn't have labels. This property is what makes it very problematic for many practical applications. The absence of labels which represent the desired behavior for your model means the absence of a solid reference point to judge the quality of your model. In this book, I only present unsupervised learning methods that allow building models that can be evaluated based on data as opposed to human judgment.

9.1 Density Estimation

Density estimation is a problem of modeling the probability density function (pdf) of the unknown probability distribution from which the dataset has been drawn. It can be useful for many applications, in particular for novelty or intrusion detection. In Chapter 7, we already estimated the pdf to solve the one-class classification problem. To do that, we decided that our model would be *parametric*, more precisely a multivariate normal distribution (MVN). This decision was somewhat arbitrary because if the real distribution from which our dataset was drawn is different from the MVN, our model will be very likely far from perfect. We also know that models can be nonparametric. We used a nonparametric model in kernel regression. It turns out that the same approach can work for density estimation.

Let $\{x_i\}_{i=1}^N$ be a one-dimensional dataset (a multi-dimensional case is similar) whose examples were drawn from a distribution with an unknown pdf f with $x_i \in \mathbb{R}$ for all $i = 1, \dots, N$. We are interested in modeling the shape of this function f. Our kernel model of f, let's denote it as $\hat{f}$, is given by,

$$\hat{f}_b(x) = \frac{1}{Nb} \sum_{i=1}^{N} k\left(\frac{x - x_i}{b}\right), \tag{1}$$

where b is a hyperparameter that controls the tradeoff between bias and variance of our model and k is a kernel. Again, like in Chapter 7, we use a Gaussian kernel:

$$k(z) = \frac{1}{\sqrt{2\pi}} \exp\left(\frac{-z^2}{2}\right).$$

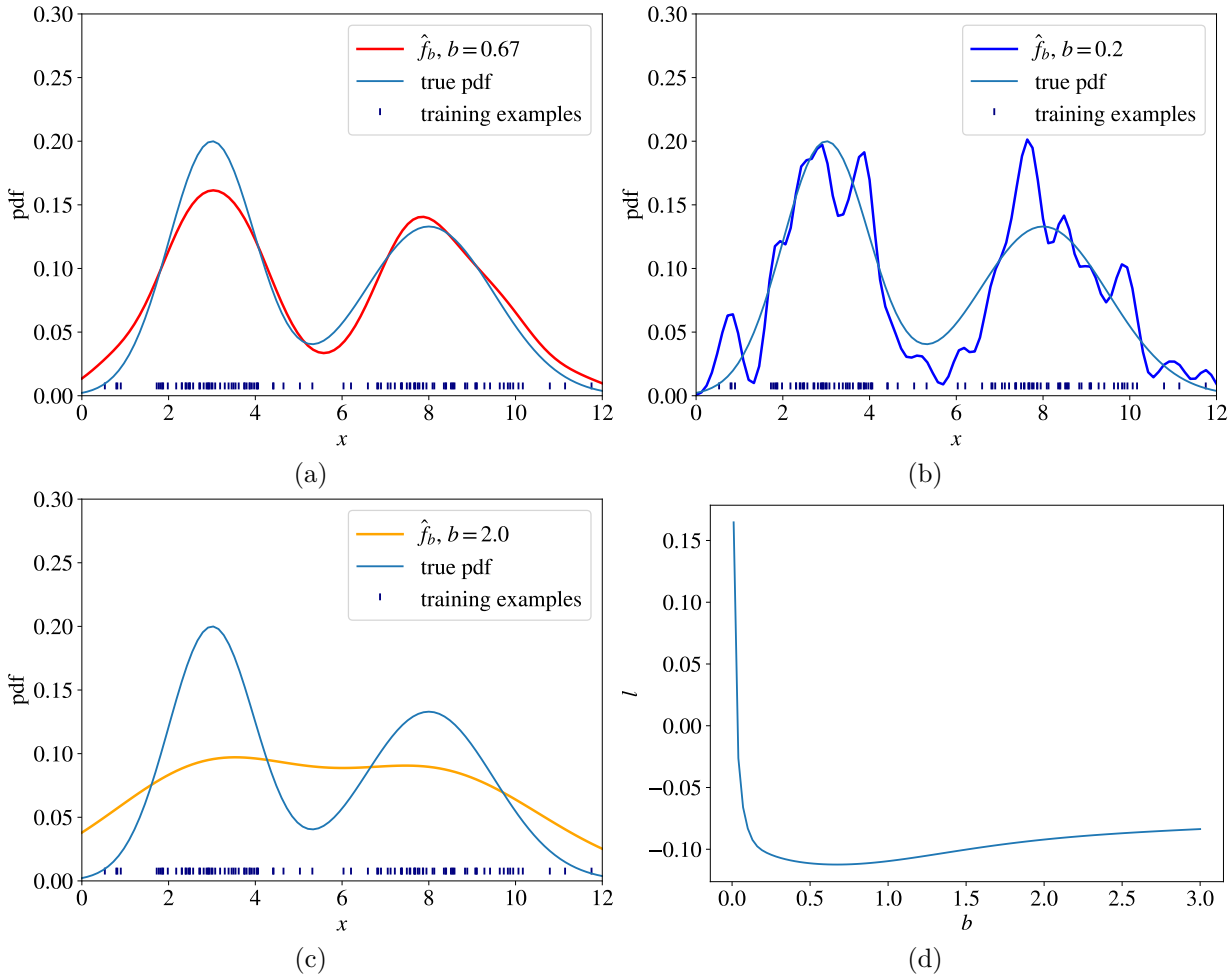


Figure 1: Kernel density estimation: (a) good fit; (b) overfitting; (c) underfitting; (d) the curve of grid search for the best value for b.

We look for such a value of b that minimizes the difference between the real shape of f and the shape of our model $\hat{f}_b$. A reasonable choice of measure of this difference is called the mean integrated squared error:

$$\text{MISE}(b) = \mathbb{E}\left[\int_{\mathbb{R}} (\hat{f}_b(x) - f(x))^2 \, dx\right]. \tag{2}$$

Intuitively, you see in eq. 2 that we square the difference between the real pdf f and our model of it $\hat{f}$. The integral $\int_{\mathbb{R}}$ replaces the summation $\sum_{i=1}^{N}$ we employed in the average

squared error, while the expectation operator $\mathbb{E}$ replaces the average $\frac{1}{N}$.

Indeed, when our loss is a function with a continuous domain, such as $(\hat{f}_b(x) - f(x))^2$, we have to replace the summation with the integral. The expectation operation $\mathbb{E}$ means that we want b to be optimal for all possible realizations of our training set $\{x_i\}_{i=1}^N$. That is important because $\hat{f}_b$ is defined on a *finite* sample of some probability distribution, while the real pdf f is defined on an infinite domain (the set $\mathbb{R}$).

Now, we can rewrite the right-hand side term in eq. 2 like this:

$$\mathbb{E}\left[\int_{\mathbb{R}} \hat{f}_b^2(x)dx\right] - 2\mathbb{E}\left[\int_{\mathbb{R}} \hat{f}_b(x)f(x)dx\right] + \mathbb{E}\left[\int_{\mathbb{R}} f(x)^2 dx\right].$$

The third term in the above summation is independent of b and thus can be ignored. An unbiased estimator of the first term is given by $\int_{\mathbb{R}} \hat{f}_b^2(x)dx$ while the unbiased estimator of the second term can be approximated by $-\frac{2}{N}\sum_{i=1}^N \hat{f}_b^{(i)}(x_i)$, where $\hat{f}_b^{(i)}$ is a kernel model of f computed on our training set with the example x_i excluded.

The term $\sum_{i=1}^N \hat{f}_b^{(i)}(x_i)$ is known in statistics as the *leave one out estimate*, a form of cross-validation in which each fold consists of one example. You could have noticed that the term $\int_{\mathbb{R}} \hat{f}_b(x)f(x)dx$ (let's call it a) is the expected value of the function $\hat{f}_b$, because f is a pdf. It can be demonstrated that the leave one out estimate is an unbiased estimator of $\mathbb{E}a$.

Now, to find the optimal value b^* for b, we want to minimize the cost defined as:

$$\int_{\mathbb{R}} \hat{f}_b^2(x)dx - \frac{2}{N}\sum_{i=1}^N \hat{f}_b^{(i)}(x_i).$$

We can find b^* using grid search. For D-dimensional feature vectors $\mathbf{x}$, the error term $x - x_i$ in eq. 1 can be replaced by the Euclidean distance $\|\mathbf{x} - \mathbf{x}_i\|$. In Figure 1 you can see the estimates for the same pdf obtained with three different values of b from a dataset containing 100 examples, as well as the grid search curve. We pick b^* at the minimum of the grid search curve.

9.2 Clustering

Clustering is a problem of learning to assign a label to examples by leveraging an unlabeled dataset. Because the dataset is completely unlabeled, deciding on whether the learned model is optimal is much more complicated than in supervised learning.

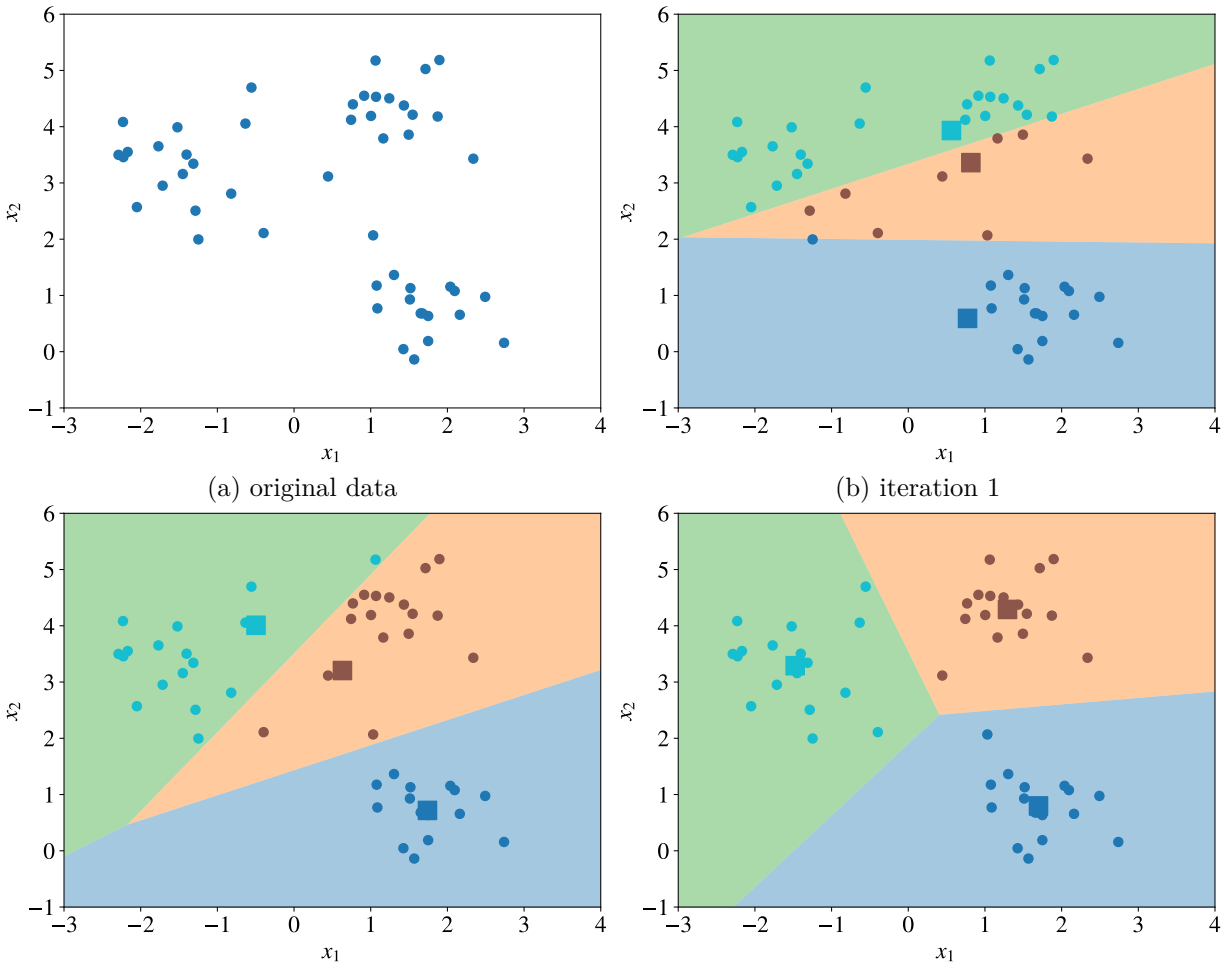


Figure 2: The progress of the k-means algorithm for $k = 3$. The circles are two-dimensional feature vectors; the squares are moving centroids.

There is a variety of clustering algorithms, and, unfortunately, it's hard to tell which one is better in quality for your dataset. Usually, the performance of each algorithm depends on the unknown properties of the probability distribution the dataset was drawn from.

9.2.1 K-Means

The **k-means** clustering algorithm works as follows. First, the analyst has to choose k — the number of classes (or clusters). Then we randomly put k feature vectors, called **centroids**, to the feature space[1]. We then compute the distance from each example $\mathbf{x}$ to each centroid $\mathbf{c}$ using some metric, like the Euclidean distance. Then we assign the closest centroid to each example (like if we labeled each example with a centroid id as the label). For each centroid, we calculate the average feature vector of the examples labeled with it. These average feature vectors become the new locations of the centroids.

We recompute the distance from each example to each centroid, modify the assignment and repeat the procedure until the assignments don't change after the centroid locations were recomputed. The model is the list of assignments of centroids IDs to the examples.

The initial position of centroids influence the final positions, so two runs of k-means can result in two different models. One run of the k-means algorithm is illustrated in Figure 2. Different background colors represent regions in which all points belong to the same cluster.

The value of k, the number of clusters, is a hyperparameter that has to be tuned by the data analyst. There are some techniques for selecting k. None of them is proven optimal. Most of them require from the analyst to make an "educated guess" by looking at some metrics or by examining cluster assignments visually. Later in this chapter, we consider one technique which allows choosing a reasonably good value for k without looking at the data and making guesses.

9.2.2 DBSCAN and HDBSCAN

While k-means and similar algorithms are centroid-based, **DBSCAN** is a density-based clustering algorithm. Instead of guessing how many clusters you need, by using DBSCAN, you define two hyperparameters: ϵ and n. You start by picking an example $\mathbf{x}$ from your dataset at random and assign it to cluster 1. Then you count how many examples have the distance from $\mathbf{x}$ less than or equal to ϵ. If this quantity is greater than or equal to n, then you put all these ϵ-neighbors to the same cluster 1. You then examine each member of cluster 1 and find their respective ϵ-neighbors. If some member of cluster 1 has n or more ϵ-neighbors, you expand cluster 1 by putting those ϵ-neighbors to the cluster. You continue expanding cluster 1 until there are no more examples to put in it. In the latter case, you pick from the dataset another example not belonging to any cluster and put it to cluster 2. You continue like this until all examples either belong to some cluster or are marked as outliers. An outlier is an example whose ϵ-neighborhood contains less than n examples.

The advantage of DBSCAN is that it can build clusters that have an arbitrary shape, while k-means and other centroid-based algorithms create clusters that have a shape of a hypersphere. An obvious drawback of DBSCAN is that it has two hyperparameters and choosing good

[1]Some variants of k-means compute the initial positions of centroids based on some properties of the dataset.

values for them (especially ϵ) could be challenging. Furthermore, having ϵ fixed, the clustering algorithm cannot effectively deal with clusters of varying density.

HDBSCAN is the clustering algorithm that keeps the advantages of DBSCAN, by removing the need to decide on the value of ϵ. The algorithm is capable of building clusters of varying density. HDBSCAN is an ingenious combination of multiple ideas and describing the algorithm in full is out of the scope of this book.

HDBSCAN only has one important hyperparameter: n, that is the minimum number of examples to put in a cluster. This hyperparameter is relatively simple to choose by intuition. HDBSCAN has very fast implementations: it can deal with millions of examples effectively. Modern implementations of k-means are much faster than HDBSCAN though, but the qualities of the latter may outweigh its drawbacks for many practical tasks. It is recommended to always try HDBSCAN on your data first.

9.2.3 Determining the Number of Clusters

The most important question is how many clusters does your dataset have? When the feature vectors are one-, two- or three-dimensional, you can look at the data and see "clouds" of points in the feature space. Each cloud is a potential cluster. However, for D-dimensional data, with $D > 3$, looking at the data is problematic[2].

There's one practically useful method of determining the reasonable number of clusters based on the concept of **prediction strength**. The idea is to split the data into training and test set, similarly to how we do in supervised learning. Once you have the training and test sets, $\mathcal{S}_{tr}$ of size N_{tr} and $\mathcal{S}_{te}$ of size N_{te} respectively, you fix k, the number of clusters, and run a clustering algorithm C on sets $\mathcal{S}_{tr}$ and $\mathcal{S}_{te}$ and obtain the clustering results $C(\mathcal{S}_{tr}, k)$ and $C(\mathcal{S}_{te}, k)$.

Let A be the clustering $C(\mathcal{S}_{tr}, k)$ built using the training set. Note that the clusters in A can be defined by some regions. If an example falls within one of those regions, then this example belongs to some specific cluster. For example, if we apply the k-means algorithm to some dataset, it results in a partition of the feature space into k polygonal regions, as we saw in Figure 2.

Define the $N_{te} \times N_{te}$ *co-membership matrix* $\mathbf{D}[A, \mathcal{S}_{te}]$ as follows: $\mathbf{D}[A, \mathcal{S}_{te}]^{(i,i')} = 1$ if and only if examples $\mathbf{x}_i$ and $\mathbf{x}_{i'}$ from the test set belong to the same cluster according to the clustering A. Otherwise $D[A, \mathcal{S}_{te}]^{(i,i')} = 0$.

Let's take a break and see what we have here. We have built, *using the training set* of examples, a clustering A that has k clusters. Then we have built the co-membership matrix that indicates whether two examples *from the test set* belong to the same cluster in A.

[2]Some analysts look at multiple two-dimensional scatter plots, in which only a pair of features are present at the same time. It might give an intuition about the number of clusters. However, such an approach suffers from subjectivity, is prone to error and counts as an educated guess rather than a scientific method.

Intuitively, if the quantity k is the reasonable number of clusters, then two examples that belong to the same cluster in clustering $C(\mathcal{S}_{te}, k)$ will most likely belong to the same cluster in clustering $C(\mathcal{S}_{tr}, k)$. On the other hand, if k is not reasonable (too high or too low), then training data-based and test data-based clusterings will likely be less consistent.

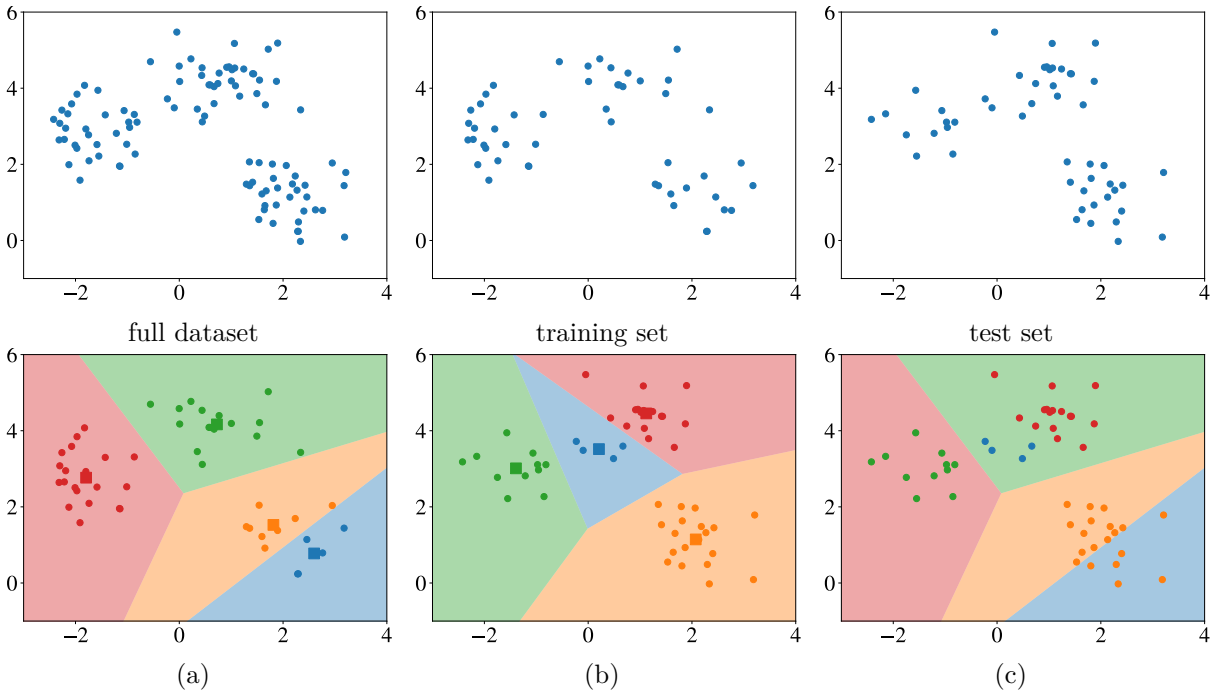


Figure 3: The clustering for $k = 4$: (a) training data clustering; (b) test data clustering; (c) test data plotted over the training clustering.

The idea is illustrated in Figure 3. The plots in Figure 3a and 3b show respectively $C(\mathcal{S}_{tr}, 4)$ and $C(\mathcal{S}_{te}, 4)$ with their respective cluster regions. Figure 3c shows the test examples plotted over the training data cluster regions. You can see in 3c that orange test examples don't belong anymore to the same cluster according to the clustering regions obtained from the training data. This will result in many zeroes in the matrix $\mathbf{D}[A, \mathcal{S}_{te}]$ which, in turn, is an indicator that $k = 4$ is likely not the best number of clusters.

More formally, the prediction strength for the number of clusters k is given by,

$$\text{ps}(k) \stackrel{\text{def}}{=} \min_{j=1,\ldots,k} \frac{1}{|A_j|(|A_j|-1)} \sum_{i,i' \in A_j} \mathbf{D}[A, \mathcal{S}_{te}]^{(i,i')},$$

where $A \stackrel{\text{def}}{=} C(\mathcal{S}_{tr}, k)$, A_j is j^{th} cluster from the clustering $C(\mathcal{S}_{te}, k)$ and $|A_j|$ is the number

of examples in cluster A_j.

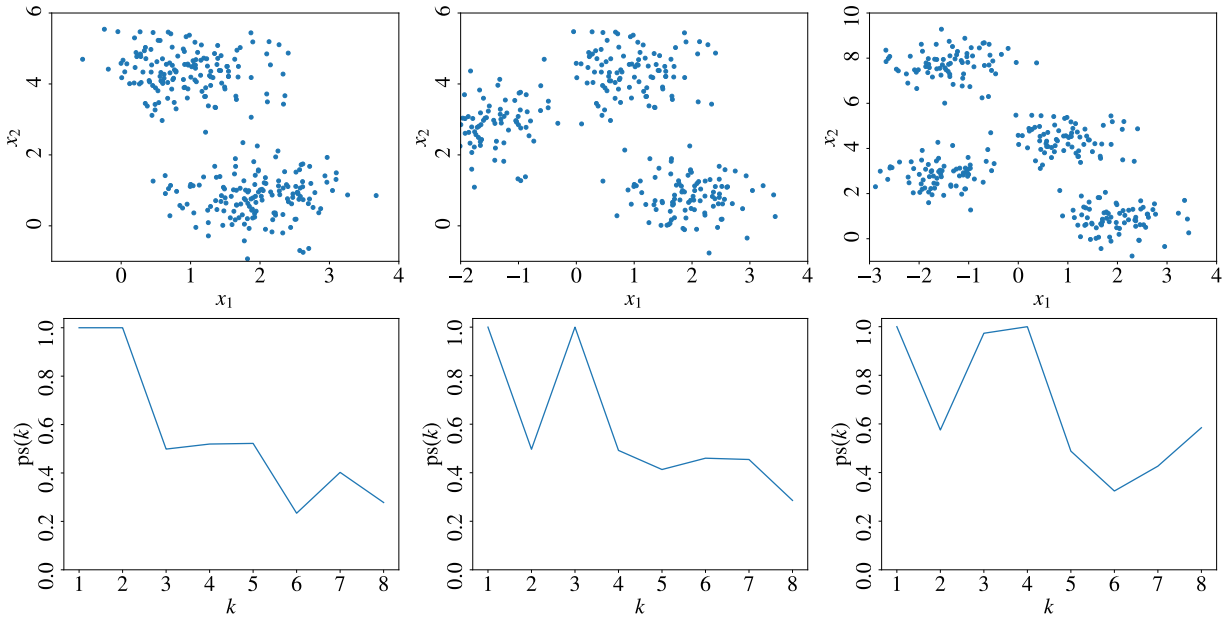


Figure 4: Predictive strength for different values of k for two-, three- and four-cluster data.

Given a clustering $C(\mathcal{S}_{tr}, k)$, for each test cluster, we compute the proportion of observation pairs in that cluster that are also assigned to the same cluster by the training set centroids. The prediction strength is the minimum of this quantity over the k test clusters.

Experiments suggest that a reasonable number of clusters is the largest k such that ps(k) is above 0.8. You can see in Figure 4 examples of predictive strength for different values of k for two-, three- and four-cluster data.

For non-deterministic clustering algorithms, such as k-means, which can generate different clusterings depending on the initial positions of centroids, it is recommended to do multiple runs of the clustering algorithm for the same k and compute the average prediction strength $\bar{\text{ps}}(k)$ over multiple runs. Another effective method to estimate the number of clusters is the **gap statistic** method. Other, less automatic methods, which some analysts still use, include the **elbow method** and the **average silhouette method**.

9.2.4 Other Clustering Algorithms

DBSCAN and k-means compute so-called **hard clustering**, in which each example can belong to only one cluster. **Gaussian mixture model** (GMM) allow each example to be a member of several clusters with different **membership score** (HDBSCAN allows that too, by the way). Computing a GMM is very similar to doing model-based density estimation. In GMM, instead of having just one multivariate normal distribution (MND), we have a weighted sum of several MNDs:

$$f_X = \sum_{j=1}^{k} \phi_j f_{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j},$$

where $f_{\boldsymbol{\mu}_j, \boldsymbol{\Sigma}_j}$ is a MND j, and ϕ_j is its weight in the sum. The values of parameters $\boldsymbol{\mu}_j$, $\boldsymbol{\Sigma}_j$, and ϕ_j, for all $j = 1, \ldots, k$ are obtained using the **expectation maximization algorithm** (EM) to optimize the **maximum likelihood** criterion.

Again, for simplicity, let us look at the one-dimensional data. Also assume that there are two clusters: $k = 2$. In this case, we have two Gaussian distributions,

$$f(x \mid \mu_1, \sigma_1^2) = \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x-\mu_1)^2}{2\sigma_1^2}} \text{ and } f(x \mid \mu_2, \sigma_2^2) = \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(x-\mu_2)^2}{2\sigma_2^2}}, \tag{3}$$

where $f(x \mid \mu_1, \sigma_1^2)$ and $f(x \mid \mu_2, \sigma_2^2)$ are two parametrized pdf defining the likelihood of $X = x$.

We use the EM algorithm to estimate μ_1, σ_1^2, μ_2, σ_2^2, ϕ_1, and ϕ_2. The parameters ϕ_1 and ϕ_2 of the GMM are useful for the density estimation task and less useful for the clustering task, as we will see below.

EM works like follows. In the beginning, we guess the initial values for μ_1, σ_1^2, μ_2, and σ_2^2, and set $\phi_1 = \phi_2 = \frac{1}{2}$ (in general, it's $\frac{1}{k}$ for each ϕ_k).

At each iteration of EM, the following four steps are executed:

1. For all $i = 1, \ldots, N$, calculate the likelihood of each x_i using eq. 3:

$$f(x_i \mid \mu_1, \sigma_1^2) \leftarrow \frac{1}{\sqrt{2\pi\sigma_1^2}} e^{-\frac{(x_i-\mu_1)^2}{2\sigma_1^2}} \text{ and } f(x_i \mid \mu_2, \sigma_2) \leftarrow \frac{1}{\sqrt{2\pi\sigma_2^2}} e^{-\frac{(x_i-\mu_2)^2}{2\sigma_2^2}}.$$

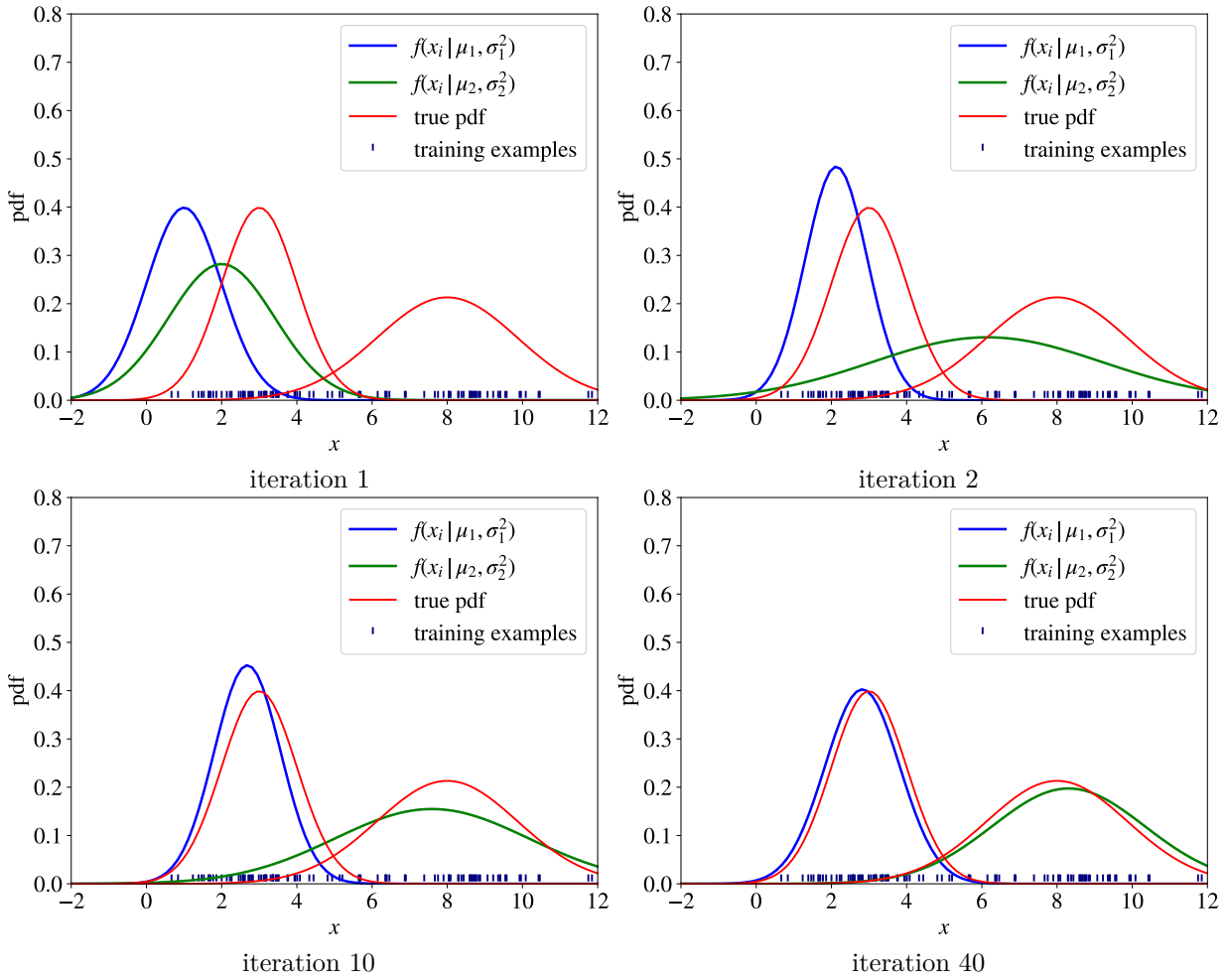


Figure 5: The progress of the Gaussian mixture model estimation using the EM algorithm for two clusters ($k = 2$).

2. Using **Bayes' Rule**, for each example x_i, calculate the likelihood $b_i^{(j)}$ that the example belongs to cluster $j \in \{1, 2\}$ (in other words, the likelihood that the example was drawn from the Gaussian j):

$$b_i^{(j)} \leftarrow \frac{f(x_i \mid \mu_j, \sigma_j^2)\phi_j}{f(x_i \mid \mu_1, \sigma_1^2)\phi_1 + f(x_i \mid \mu_2, \sigma_2^2)\phi_2}.$$

The parameter ϕ_j reflects how likely is that our Gaussian distribution j with parameters μ_j and γ_j may have produced our dataset. That is why in the beginning we set $\phi_1 = \phi_2 = \frac{1}{2}$: we

don't know how each of the two Gaussians is likely, and we reflect our ignorance by setting the likelihood of both to one half.

3. Compute the new values of μ_j and σ_j^2, $j \in \{1, 2\}$ as,

$$\mu_j \leftarrow \frac{\sum_{i=1}^{N} b_i^{(j)} x_i}{\sum_{i=1}^{N} b_i^{(j)}} \text{ and } \sigma_j^2 \leftarrow \frac{\sum_{i=1}^{N} b_i^{(j)} (x_i - \mu_j)^2}{\sum_{i=1}^{N} b_i^{(j)}}. \tag{4}$$

4. Update ϕ_j, $j \in \{1, 2\}$ as,

$$\phi_j \leftarrow \frac{1}{N} \sum_{i=1}^{N} b_i^{(j)}.$$

The steps $1 - 4$ are executed iteratively until the values μ_j and σ_j^2 don't change much: for example, the change is below some threshold ϵ. Figure 5 illustrates this process.

You may have noticed that the EM algorithm is very similar to the k-means algorithm: start with random clusters, then iteratively update each cluster's parameters by averaging the data that is assigned to that cluster. The only difference in the case of the GMM is that the assignment of an example x_i to the cluster j is soft: x_i belongs to cluster j with probability $b_i^{(j)}$. This is why we calculate the new values for μ_j and σ_j^2 in eq. 4 not as an average (used in k-means) but as a *weighted average* with weights $b_i^{(j)}$.

Once we have learned the parameters μ_j and σ_j^2 for each cluster j, the membership score of example x in cluster j is given by $f(x \mid \mu_j, \sigma_j^2)$.

The extension to D-dimensional data ($D > 1$) is straightforward. The only difference is that instead of the variance σ^2, we now have the covariance matrix $\mathbf{\Sigma}$ that parametrizes the multinomial normal distribution (MND). The advantage of GMM over k-means is that the clusters in GMM can have a form of an ellipse that can have an arbitrary elongation and rotation. The values in the covariance matrix control these properties.

How to choose k in GMM? Unfortunately, there's no universally recognized method. What is usually recommended to do is to split your dataset into training and test set. Then try different k and build a different model f_{tr}^k for each k on the training data. Then choose the model that maximizes the likelihood of examples in the test set:

$$\arg\max_k \prod_{i=1}^{N_{te}} f_{tr}^k(\mathbf{x}_i),$$

where N_{te} is the size of the test set.

There is a variety of clustering algorithms described in the literature. Worth mentioning are **spectral clustering** and **hierarchical clustering**. For some datasets, you may find those more appropriate. However, in most practical cases, kmeans, HDBSCAN and the Gaussian mixture model would satisfy your needs.

9.3 Dimensionality Reduction

Many modern machine learning algorithms, such as ensemble algorithms and neural networks handle well very high-dimensional examples, up to millions of features. With modern computers and graphical processing units (GPUs), dimensionality reduction techniques are used much less in practice than in the past. The most frequent use case for dimensionality reduction is data visualization: humans can only interpret on a plot the maximum of three dimensions.

Another situation in which you could benefit from dimensionality reduction is when you have to build an interpretable model and to do so you are limited in your choice of learning algorithms. For example, you can only use decision tree learning or linear regression. By reducing your data to lower dimensionality and by figuring out which quality of the original example each new feature in the reduced feature space reflects, one can use simpler algorithms. Dimensionality reduction removes redundant or highly correlated features; it also reduces the noise in the data — all that contributes to the interpretability of the model.

The three most widely used techniques of dimensionality reduction are **principal component analysis** (PCA), **uniform manifold approximation and projection** (UMAP), and **autoencoders**.

I already explained autoencoders in Chapter 7. You can use the low-dimensional output of the **bottleneck layer** of the autoencoder as the vector of reduced dimensionality that represents the high-dimensional input feature vector. You know that this low-dimensional vector represents the essential information contained in the input vector because the autoencoder is capable of reconstructing the input feature vector based on the bottleneck layer output alone.

9.3.1 Principal Component Analysis

Principal component analysis or PCA is one of the oldest methods. The math behind it involves operation on matrices that I didn't explain in Chapter 2, so I leave the math of PCA for your further reading. Here, I only provide intuition and illustrate the method on an example.

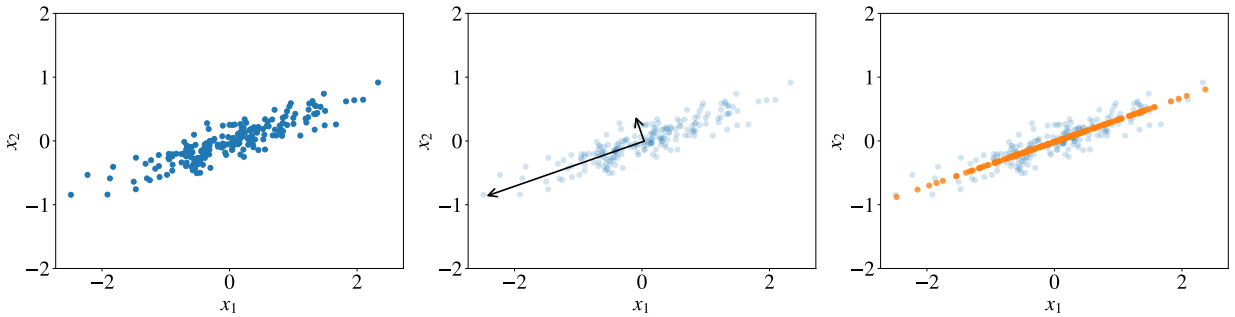


Figure 6: PCA: (a) the original data; (b) two principal components displayed as vectors; (c) the data projected on the first principal component.

Consider a two-dimensional data as shown in Figure 6a. Principal components are vectors that define a new coordinate system in which the first axis goes in the direction of the highest variance in the data. The second axis is orthogonal to the first one and goes in the direction of the second highest variance in the data. If our data was three-dimensional, the third axis would be orthogonal to both the first and the second axes and go in the direction of the third highest variance, and so on. In Figure 6b, the two principal components are shown as arrows. The length of the arrow reflects the variance in this direction.

Now, if we want to reduce the dimensionality of our data to $D_{new} < D$, we pick D_{new} largest principal components and project our data points on them. For our two-dimensional illustration, we can set $D_{new} = 1$ and project our examples to the first principal component to obtain the orange points in Figure 6c.

To describe each orange point, we need only one coordinate instead of two: the coordinate with respect to the first principal component. When our data is very high-dimensional, it often happens in practice that the first two or three principal components account for most of the variation in the data, so by displaying the data on a 2D or 3D plot we can indeed see a very high-dimensional data and its properties.

9.3.2 UMAP

The idea behind many of the modern dimensionality reduction algorithms, especially those designed specifically for visualization purposes, such as **t-SNE** and **UMAP**, is basically the same. We first design a similarity metric for two examples. For visualization purposes, besides the Euclidean distance between the two examples, this similarity metric often reflects

some local properties of the two examples, such as the density of other examples around them.

In UMAP, this similarity metric w is defined as follows,

$$w(\mathbf{x}_i, \mathbf{x}_j) \stackrel{\text{def}}{=} w_i(\mathbf{x}_i, \mathbf{x}_j) + w_j(\mathbf{x}_j, \mathbf{x}_i) - w_i(\mathbf{x}_i, \mathbf{x}_j) w_j(\mathbf{x}_j, \mathbf{x}_i). \tag{5}$$

The function $w_i(\mathbf{x}_i, \mathbf{x}_j)$ is defined as,

$$w_i(\mathbf{x}_i, \mathbf{x}_j) \stackrel{\text{def}}{=} \exp\left(-\frac{d(\mathbf{x}_i, \mathbf{x}_j) - \rho_i}{\sigma_i}\right),$$

where $d(\mathbf{x}_i, \mathbf{x}_j)$ is the Euclidean distance between two examples, ρ_i is the distance from $\mathbf{x}_i$ to its closest neighbor, and σ_i is the distance from $\mathbf{x}_i$ to its k^{th} closest neighbor (k is a hyperparameter of the algorithm).

It can be shown that the metric in eq. 5 varies in the range from 0 to 1 and is symmetric, which means that $w(\mathbf{x}_i, \mathbf{x}_j) = w(\mathbf{x}_j, \mathbf{x}_i)$.

Let w denote the similarity of two examples in the original high dimensional space and let w' be the similarity given by the same eq. 5 in the new low-dimensional space. Because the values of w and w' lie in the range between 0 and 1, we can see them as two probability distributions. A widely used metric of similarity between two probability distributions is **cross-entropy**:

$$C(w, w') = \sum_{i=1}^{N} \sum_{j=1}^{N} w(\mathbf{x}_i, \mathbf{x}_j) \ln\left(\frac{w(\mathbf{x}_i, \mathbf{x}_j)}{w'(\mathbf{x}'_i, \mathbf{x}'_j)}\right) + (1 - w(\mathbf{x}_i, \mathbf{x}_j)) \ln\left(\frac{1 - w(\mathbf{x}_i, \mathbf{x}_j)}{1 - w'(\mathbf{x}'_i, \mathbf{x}'_j)}\right), \tag{6}$$

where $\mathbf{x}'$ is the low-dimensional "version" of the original high-dimensional example $\mathbf{x}$.

As you can see from eq. 6, when $w(\mathbf{x}_i, \mathbf{x}_j)$ is similar to $w'(\mathbf{x}'_i, \mathbf{x}'_j)$, for all pairs (i, j), then $C(w, w')$ is minimized. And this is precisely what we want: for any two examples $\mathbf{x}_i$ and $\mathbf{x}_j$, we want their similarity metric in the original and the lower-dimensional spaces to be as similar as possible.

In eq. 6 the unknown parameters are $\mathbf{x}'_i$ (for all $i = 1, \ldots, N$) and we can compute them by gradient descent by minimizing $C(w, w')$.

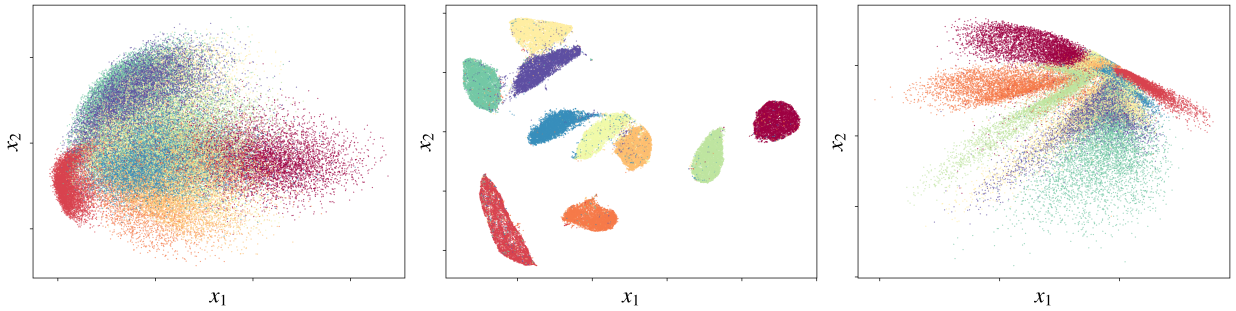


Figure 7: Dimensionality reduction of the MNIST dataset using three different techniques.

In Figure 7, you can see the result of dimensionality reduction applied to the MNIST dataset of handwritten digits. MNIST is commonly used for benchmarking various image processing systems; it contains 70,000 labeled examples. Ten different colors on the plot correspond to ten classes. Each point on the plot corresponds a specific example in the dataset. As you can see, UMAP separates examples visually better (remember, it doesn't have access to labels). In practice, UMAP is slightly slower than PCA but faster than autoencoder.

9.4 Outlier Detection

Outlier detection is the problem of detecting in the dataset the examples that are very different from what a typical example in the dataset looks like. We have already seen several techniques that could help to solve this problem: autoencoder and one-class classifier learning. If we use autoencoder, we train it on our dataset. Then, if we want to predict whether an example is an outlier, we can use the autoencoder model to reconstruct the example from the bottleneck layer. The model will unlikely be capable of reconstructing an outlier.

In one-class classification, the model either predicts that the input example belongs to the class, or it's an outlier.