Chapter 9
Feature Learning

"Solving Problems By Changing the Viewpoint."

Chapter 2 discussed features as those properties of a data point that can be measured or computed easily. Sometimes the choice of features follows naturally from the available hard and software. For example, we might use the numeric measurement $z \in \mathbb{R}$ delivered by a sensing device as a feature. However, we could augment this single feature with new features such as the powers z^2 and z^3 or adding a constant $z + 5$. Each of these computations produces a new feature. Which of these additional features are most useful?

Feature learning methods automate the choice of finding good features. These methods learn a hypothesis map that reads in some representation of a data point and transforms it to a set of features. Feature learning methods differ in the precise format of the original data representation as well as the format of the delivered features. The focus of this chapter in on feature learning methods that require data points being represented by d numeric raw features and deliver a set of n new numeric features. We will denote the set of raw and new features by $\mathbf{z} = \big(z_1, \ldots, z_d\big)^T \in \mathbb{R}^d$ and $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$, respectively.

Many ML application domains generate data points for which can access a huge number of raw features. Consider data points being snapshots generated by a smartphone. It seems natural to use the pixel colour intensities as the raw features of the snapshot. Since modern smartphone have Megapixel cameras, the pixel intensities would provide us with millions of raw features. It might seem a good idea to use as many (raw) features of a data point as possible since more features should offer more information about a data point and its label y. There are, however, two pitfalls in using an unnecessarily large number of features. The first one is a computational pitfall and the second one is a statistical pitfall.

Computationally, using very large feature vectors $\mathbf{x} \in \mathbb{R}^n$ (with n being billions), might result in excessive resource requirements (bandwidth, storage, time) of the

A. Jung, *Machine Learning*, Machine Learning: Foundations, Methodologies, and Applications, https://doi.org/10.1007/978-981-16-8193-6_9

resulting ML method. Statistically, using a large number of features makes the resulting ML methods more prone to overfitting. For example, linear regression will typically overfit when using feature vectors $\mathbf{x} \in \mathbb{R}^n$ whose length n exceeds the number m of labeled data points used for training (see Chap. 7).

Both from a computational and a statistical perspective, it is beneficial to use only the maximum necessary amount of features. The challenge is to select those features which carry most of the relevant information required for the prediction of the label y. Finding the most relevant features out of a huge number of (raw) features is the goal of dimensionality reduction methods. Dimensionality reduction methods form an important sub-class of feature learning methods. Formally, dimensionality reduction methods learn a hypothesis $h(\mathbf{z})$ that map a long raw feature vector $\mathbf{z} \in \mathbb{R}^d$ to a short new feature vector $\mathbf{x} \in \mathbb{R}^n$ with $d \gg n$.

Beside avoiding overfitting and coping with limited computational resources, dimensionality reduction can also be useful for data visualization. Indeed, if the resulting feature vector has length $n = 2$, we depict data points in the two-dimensional plane in form of a scatterplot.

We will discuss the basic idea underlying dimensionality reduction methods in Sect. 9.1. Section 9.2 presents one particular example of a dimensionality reduction method that computes relevant features by a linear transformation of the raw feature vector. Section 9.4 discusses a method for dimensionality reduction that exploits the availability of labelled data points. Section 9.6 shows how randomness can be used to obtain computationally cheap dimensionality reduction.

Most of this chapter discusses dimensionality reduction methods that determine a small number of relevant features from a large set of raw features. However, sometimes it might be useful to go the opposite direction. There are applications where it might be beneficial to construct a large (even infinite) number of new features from a small set of raw features. Section 9.7 will showcase how computing additional features can help to improve the prediction accuracy of ML methods.

9.1 Basic Principle of Dimensionality Reduction

The efficiency of ML methods depends crucially on the choice of features that are used to characterize data points. Ideally we would like to have a small number of highly relevant features to characterize data points. If we use too many features we risk to waste computations on exploring irrelevant features. If we use too few features we might not have enough information to predict the label of a data point. For a given number n of features, dimensionality reduction methods aim at learning an (in a certain sense) optimal map from the data point to a feature vector of length n.

Figure 9.1 illustrates the basic idea of dimensionality reduction methods. Their goal is to learn (or find) a "compression" map $h(\cdot) : \mathbb{R}^d \to \mathbb{R}^n$ that transforms a (long) raw feature vector $\mathbf{z} \in \mathbb{R}^d$ to a (short) feature vector $\mathbf{x} = (x_1, \ldots, x_n)^T := h(\mathbf{z})$ (typically $n \ll d$).

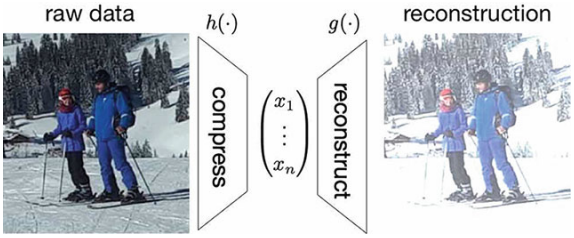

Fig. 9.1 Dimensionality reduction methods aim at finding a map h which maximally compresses the raw data while still allowing to accurately reconstruct the original data point from a small number of features $x_1, \ldots, x_n$

The new feature vector $\mathbf{x} = h(\mathbf{z})$ serves as a compressed representation (or code) for the original raw feature vector $\mathbf{z}$. We can reconstruct the raw feature vector using a reconstruction map $r(\cdot) : \mathbb{R}^n \to \mathbb{R}^d$. The reconstructed raw features $\widehat{\mathbf{z}} := r(\mathbf{x}) = r(h(\mathbf{z}))$ will typically by different from the original raw feature vector $\mathbf{z}$. Thus, we will obtain a non-zero reconstruction error

$$\underbrace{\widehat{\mathbf{z}}}_{=r(h(\mathbf{z}))} - \mathbf{z}. \tag{9.1}$$

Dimensionality reduction methods learn a compression map $h(\cdot)$ such that the reconstruction error (9.1) is minimized. In particular, for a dataset $\mathcal{D} = \{\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\}$, we measure the quality of a pair of compression map h and reconstruction map r by the average reconstruction error

$$\widehat{L}\big(h, r|\mathcal{D}\big) := (1/m) \sum_{i=1}^{m} L(\mathbf{z}^{(i)}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)). \tag{9.2}$$

Here, $L(\mathbf{z}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)$ denotes a loss function that is used to measure the reconstruction error $\underbrace{r\big(h\big(\mathbf{z}^{(i)}\big)\big)}_{\widehat{\mathbf{z}}} - \mathbf{z}$. Different choices for the loss function in (9.2) result in different dimensionality reduction methods. One widely-used choice for the loss is the squared Euclidean norm

$$L(\mathbf{z}, g\big(h\big(\mathbf{z}\big)\big)) := \big\|\mathbf{z} - g\big(h\big(\mathbf{z}\big)\big)\big\|_2^2. \tag{9.3}$$

Practical dimensionality reduction methods have only finite computational resources. Any practical method must therefore restrict the set of possible compression and reconstruction maps to small subsets $\mathcal{H}$ and $\mathcal{H}^*$, respectively. These subsets are the hypothesis spaces for the compression map $h \in \mathcal{H}$ and the reconstruction map $r \in \mathcal{H}^*$. Feature learning methods differ in their choice for these hypothesis spaces.

Dimensionality reduction methods learn a compression map by solving

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \min_{r \in \mathcal{H}^*} \widehat{L}\big(h, r | \mathcal{D}\big)$$
$$\stackrel{(9.2)}{=} \operatorname*{argmin}_{h \in \mathcal{H}} \min_{r \in \mathcal{H}^*} (1/m) \sum_{i=1}^{m} L(\mathbf{z}^{(i)}, r\big(h\big(\mathbf{z}^{(i)}\big)\big)). \tag{9.4}$$

We can interpret (9.4) as a (typically non-linear) approximation problem. The optimal compression map $\hat{h}$ is such that the reconstruction$r(\hat{h}(\mathbf{z}))$, with a suitably chosen reconstruction map r, approximates the original raw feature vector $\mathbf{z}$ as good as possible. Note that we use a single compression map $h(\cdot)$ and a single reconstruction map $r(\cdot)$ for all data points in the dataset $\mathcal{D}$.

We obtain variety of dimensionality methods by using different choices for the hypothesis spaces $\mathcal{H}, \mathcal{H}^*$ and loss function in (9.4). Section 9.2 discusses a method that solves (9.4) for $\mathcal{H}, \mathcal{H}^*$ constituted by linear maps and the loss (9.3). Deep autoencoders are another family of dimensionality reduction methods that solve (9.4) with $\mathcal{H}, \mathcal{H}^*$ constituted by non-linear maps that are represented by deep neural networks [1, Chap. 14].

9.2 Principal Component Analysis

We now consider the special case of dimensionality reduction where the compression and reconstruction map are required to be linear maps. Consider a data point which is characterized by a (typically very long) raw feature vector $\mathbf{z} \in \mathbb{R}^d$ of length d. The length d of the raw feature vector might be easily of the order of millions. To obtain a small set of relevant features $\mathbf{x} = \big(x_1, \ldots, x_n\big)^T \in \mathbb{R}^n$, we apply a linear transformation to the raw feature vector,

$$\mathbf{x} = \mathbf{W}\mathbf{z}. \tag{9.5}$$

Here, the "compression" matrix $\mathbf{W} \in \mathbb{R}^{n \times d}$ maps (in a linear fashion) the (long) raw feature vector $\mathbf{z} \in \mathbb{R}^d$ to the (shorter) feature vector $\mathbf{x} \in \mathbb{R}^n$.

It is reasonable to choose the compression matrix $\mathbf{W} \in \mathbb{R}^{n \times D}$ in (9.5) such that the resulting features $\mathbf{x} \in \mathbb{R}^n$ allow to approximate the original data point $\mathbf{z} \in \mathbb{R}^d$ as accurate as possible. We can approximate (or recover) the data point $\mathbf{z} \in \mathbb{R}^d$ back from the features $\mathbf{x}$ by applying a reconstruction operator $\mathbf{R} \in \mathbb{R}^{d \times n}$, which is chosen such that

$$\mathbf{z} \approx \mathbf{R}\mathbf{x} \stackrel{(9.5)}{=} \mathbf{R}\mathbf{W}\mathbf{z}. \tag{9.6}$$

The approximation error $\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big)$ resulting when (9.6) is applied to each data point in a dataset $\mathcal{D} = \{\mathbf{z}^{(i)}\}_{i=1}^m$ is then

$$\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big) = (1/m) \sum_{i=1}^{m} \|\mathbf{z}^{(i)} - \mathbf{R}\mathbf{W}\mathbf{z}^{(i)}\|^2. \tag{9.7}$$

One can verify that the approximation error $\widehat{L}\big(\mathbf{W}, \mathbf{R} \mid \mathcal{D}\big)$ can only by minimal if the compression matrix $\mathbf{W}$ is of the form

$$\mathbf{W} = \mathbf{W}_{\rm PCA} := \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\big)^T \in \mathbb{R}^{n \times d}, \tag{9.8}$$

with n orthonormal vectors $\mathbf{u}^{(j)}$, for $j = 1, \ldots, n$. The vectors $\mathbf{u}^{(j)}$ are the eigenvectors corresponding to the n largest eigenvalues of the sample covariance matrix

$$\mathbf{Q} := (1/m)\mathbf{Z}^T\mathbf{Z} \in \mathbb{R}^{d \times d}. \tag{9.9}$$

Here we used the data matrix $\mathbf{Z} = \big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big)^T \in \mathbb{R}^{m \times d}$.[1] It can be verified easily, using the definition (9.9), that the matrix $\mathbf{Q}$ is psd. As a psd matrix, $\mathbf{Q}$ has an eigenvalue decomposition (EVD) of the form [2]

$$\mathbf{Q} = \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}\big) \begin{pmatrix} \lambda^{(1)} & \ldots & 0 \\ 0 & \ddots & 0 \\ 0 & \ldots & \lambda^{(d)} \end{pmatrix} \big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}\big)^T$$

with real-valued eigenvalues $\lambda^{(1)} \geq \lambda^{(2)} \geq \ldots \geq \lambda^{(d)} \geq 0$ and orthonormal eigenvectors $\{\mathbf{u}_r\}_{r=1}^{d}$.

The feature vectors $\mathbf{x}^{(i)}$ are obtained by applying the compression matrix $\mathbf{W}_{\rm PCA}$ (9.8) to the raw feature vectors $\mathbf{z}^{(i)}$. We refer to the entries of the vector $\mathbf{x}^{(i)}$, obtained via the eigenvectors of $\mathbf{Q}$ (see (9.2)), as the **principal components (PC)** of the raw feature vectors $\mathbf{z}^{(i)}$. Algorithm 15 summarizes the overall procedure of determining the compression matrix (9.8) and computing the vectors $\mathbf{x}^{(i)}$ whose entries are the PC of the raw feature vectors. This procedure is known as **principal component analysis (PCA)**. Note that the length $n (\leq d)$ of the mew feature vector $\mathbf{x}$, which is also the number of PCs used, is an input (or hyper) parameter of Algorithm 15. The number n can be chosen between the two extreme cases $n = 0$ (maximum compression) and $n = d$ (no compression). We finally note that the choice for the orthonormal eigenvectors in (9.8) might not be unique. Depending on the sample covariance matrix $\mathbf{Q}$, there might different sets of orthonormal vectors that correspond to the same eigenvalue of $\mathbf{Q}$. Thus, for a given length n of the new feature vectors, there might be several different matrices $\mathbf{W}$ that achieve the same (optimal) reconstruction error $\widehat{L}^{(\rm PCA)}$.

From a computational perspective, Algorithm 15 essentially amounts to performing an EVD of the sample covariance matrix $\mathbf{Q}$ (see (9.9)). Indeed, the EVD of $\mathbf{Q}$ provides not only the optimal compression matrix $\mathbf{W}_{\rm PCA}$ but also the measure $\widehat{L}^{(\rm PCA)}$ for the information loss incurred by replacing the original data points $\mathbf{z}^{(i)} \in \mathbb{R}^d$ with the smaller feature vector $\mathbf{x}^{(i)} \in \mathbb{R}^n$. In particular, this information loss is measured by the approximation error (obtained for the optimal reconstruction matrix $\mathbf{R}_{\rm opt} = \mathbf{W}_{\rm PCA}^T$)

[1] Some authors define the data matrix as $\mathbf{Z} = \big(\widetilde{\mathbf{z}}^{(1)}, \ldots, \widetilde{\mathbf{z}}^{(m)}\big)^T \in \mathbb{R}^{m \times D}$ using "centered" raw feature vectors $\widetilde{\mathbf{z}}^{(i)} - \widehat{\mathbf{m}}$ obtained by subtracting the average $\widehat{\mathbf{m}} = (1/m) \sum_{i=1}^{m} \mathbf{z}^{(i)}$.

Algorithm 15 Principal Component Analysis (PCA)

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(i)} \in \mathbb{R}^d\}_{i=1}^m$; number n of PCs.

1: compute the EVD (9.2) to obtain orthonormal eigenvectors $\left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}\right)$ corresponding to (decreasingly ordered) eigenvalues $\lambda^{(1)} \geq \lambda^{(2)} \geq \ldots \geq \lambda^{(d)} \geq 0$

2: construct compression matrix $\mathbf{W}_{\rm PCA} := \left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}\right)^T \in \mathbb{R}^{n \times d}$

3: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}_{\rm PCA}\mathbf{z}^{(i)}$ whose entries are PC of $\mathbf{z}^{(i)}$

4: compute approximation error $\widehat{L}^{(\rm PCA)} = \sum_{r=n+1}^{d} \lambda^{(r)}$ (see (9.10)).

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and the approximation error $\widehat{L}^{(\rm PCA)}$.

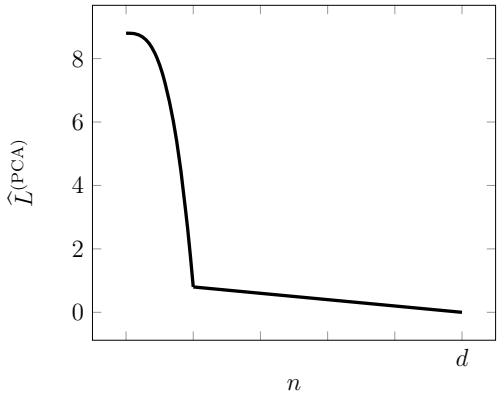

Fig. 9.2 Reconstruction error $\widehat{L}^{(\rm PCA)}$ (see (9.10)) of PCA for varying number n of PCs

$$\widehat{L}^{(\rm PCA)} := \widehat{L}\big(\mathbf{W}_{\rm PCA}, \underbrace{\mathbf{R}_{\rm opt}}_{=\mathbf{W}_{\rm PCA}^T} \mid \mathcal{D}\big) = \sum_{r=n+1}^{d} \lambda^{(r)}. \tag{9.10}$$

As depicted in Fig. 9.2, the approximation error $\widehat{L}^{(\rm PCA)}$ decreases with increasing number n of PCs used for the new features (9.5). For the extreme case $n = 0$, where we completely ignore the raw feature vectors $\mathbf{z}^{(i)}$, the optimal reconstruction error is $\widehat{L}^{(\rm PCA)} = (1/m)\sum_{i=1}^m \|\mathbf{z}^{(i)}\|^2$. The other extreme case $n = d$ allows to use the raw features directly as the new features $\mathbf{x}^{(i)} = \mathbf{z}^{(i)}$, which amounts to no compression at all, and trivially results in a zero reconstruction error $\widehat{L}^{(\rm PCA)} = 0$.

9.2.1 Combining PCA with Linear Regression

One important use case of PCA is as a pre-processing step within an overall ML problem such as linear regression (see Sect. 3.1). As discussed in Chap. 7, linear regression methods are prone to overfitting whenever the data points are characterized by feature vectors whose length D exceeds the number m of labeled data points used

for training. One simple but powerful strategy to avoid overfitting is to preprocess the original feature vectors (they are considered as the raw data points $\mathbf{z}^{(i)} \in \mathbb{R}^d$) by applying PCA in order to obtain smaller feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$ with $n < m$.

9.2.2 How to Choose Number of PC?

There are several aspects which can guide the choice for the number n of PCs to be used as features.

- for data visualization: use either $n = 2$ or $n = 3$
- computational budget: choose n sufficiently small such that the computational complexity of the overall ML method does not exceed the available computational resources.
- statistical budget: consider using PCA as a pre-processing step within a linear regression problem (see Sect. 3.1). Thus, we use the output $\mathbf{x}^{(i)}$ of PCA as the feature vectors in linear regression. In order to avoid overfitting, we should choose $n < m$ (see Chap. 7).
- elbow method: choose n large enough such that approximation error $\widehat{L}^{(\mathrm{PCA})}$ is reasonably small (see Fig. 9.2).

9.2.3 Data Visualisation

If we use PCA with $n = 2$, we obtain feature vectors $\mathbf{x}^{(i)} = \mathbf{W}_{\mathrm{PCA}}\mathbf{z}^{(i)}$ (see (9.5)) which can be depicted as points in a scatterplot (see Sect. 2.1.3). As an example, consider data points $\mathbf{z}^{(i)}$ obtained from historic recordings of Bitcoin statistics. Each data point $\mathbf{z}^{(i)} \in \mathbb{R}^d$ is a vector of length $d = 6$. It is difficult to visualise points in an Euclidean space $\mathbb{R}^d$ of dimension $d > 2$. Therefore, we apply PCA with $n = 2$ which results in feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^2$. These new feature vectors (of length 2) can be depicted conveniently as a scatterplot (see Fig. 9.3).

9.2.4 Extensions of PCA

There have been proposed several extensions of the basic PCA method:

- **Kernel PCA** [3, Chap. 14.5.4]**:** The PCA method is most effective if the raw feature vectors of data points are nearby a low-dimensional linear subspace of $\mathbb{R}^d$. Kernel PCA extends PCA to handle data points that are located near a low-dimensional manifold which might be highly non-linear. This is achieved by applying PCA to transformed feature vectors instead of the original feature vectors. Kernel PCA first applies a (typically non-linear) feature map to the original feature vectors

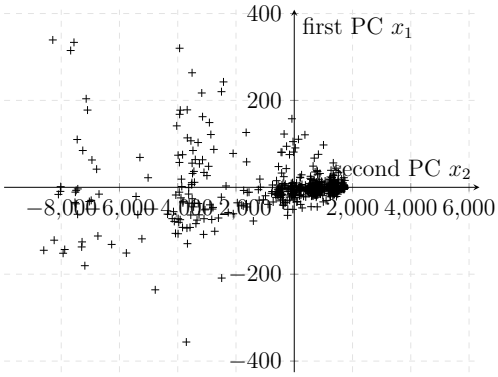

Fig. 9.3 A scatterplot of data points with feature vectors $\mathbf{x}^{(i)} = \left(x_1^{(i)}, x_2^{(i)}\right)^T$ whose entries are the first two PCs of the Bitcoin statistics $\mathbf{z}^{(i)}$ of the ith day

$\mathbf{x}^{(i)}$ resulting in new feature vectors $\mathbf{z}^{(i)}$ (see Sect. 3.9). We then apply PCA to the transformed feature vectors $\mathbf{z}^{(i)}$, for $i = 1, \ldots, m$.

- **Robust PCA** [4]**:** In its basic form, PCA is sensitive to outliers which are a small number of data points with fundamentally different statistical properties than the bulk of data points. This sensitivity might be attributed to the properties of the squared Euclidean norm (9.3) which is used in PCA to measure the reconstruction error (9.1). We have seen in Chap. 3 that linear regression (see Sect. 3.1 and 3.3) can be made robust against outliers by replacing the squared error loss with another loss function. In a similar spirit, robust PCA replaces the squared Euclidean norm with another norm that is less sensitive to having very large reconstruction errors (9.1) for a small number of data points (which are outliers).
- **Sparse PCA** [3, Chap. 14.5.5]**:** The basic PCA method transforms the raw feature vector $\mathbf{z}^{(i)}$ of a data point to a new (shorter) feature vector $\mathbf{x}^{(i)}$. In general each entry $x_j^{(i)}$ of the new feature vectors will depend on every raw feature. More precisely, the new feature $x_j^{(i)}$ depends on all raw features $z_{j'}^{(i)}$ for which the corresponding entry $W_{j,j'}$ of the matrix $\mathbf{W} = \mathbf{W}_{\mathrm{PCA}}$ (9.8) is non-zero. For most datasets, all entries of the matrix $\mathbf{W}_{\mathrm{PCA}}$ will typically be non-zero.
  In some applications of linear dimensionality reduction we would like to construct new features that depend only on a small subset of raw features. Equivalently we would like to learn a linear compression map $\mathbf{W}$ (9.5) such that each row of $\mathbf{W}$ contains only few non-zero entries. To this end, sparse PCA enforces the rows of the compression matrix $\mathbf{W}$ to contain only a small number of non-zero entries. This enforcement can be implement either using additional constraints on $\mathbf{W}$ or by adding a penalty term to the reconstruction error (9.7).
- **Probabilistic PCA** [5, 6]**:** We have motivated PCA as a method for learning an optimal linear compression map (matrix) (9.5) such that the compressed feature vectors allows to linearly reconstruct the original raw feature vector with minimum reconstruction error (9.7). Another interpretation of PCA is that of a method that

learns a subspace of $\mathbb{R}^d$ that best fits the set of raw feature vectors $\mathbf{z}^{(i)}$, for $i = 1, \ldots, m$. This optimal subspace is precisely the subspace spanned by the rows of $\mathbf{W}_{\mathrm{PCA}}$ (9.8).

Probabilistic principal component analysis (PPCA) interprets the raw feature vectors $\mathbf{z}^{(i)}$ as realizations of i.i.d. RVs. These realizations are modelled as

$$\mathbf{z}^{(i)} = \mathbf{W}^T \mathbf{x}^{(i)} + \boldsymbol{\varepsilon}^{(i)}, \text{ for } i = 1, \ldots, m. \tag{9.11}$$

Here, $\mathbf{W} \in \mathbb{R}^{n \times d}$ is some unknown matrix with orthonormal rows. The rows of $\mathbf{W}$ span the subspace around which the raw features are concentrated. The vectors $\mathbf{x}^{(i)}$ in (9.11) are realizations of i.i.d. RVs whose common probability distribution is $\mathcal{N}(\mathbf{0}, \mathbf{I})$. The vectors $\boldsymbol{\varepsilon}^{(i)}$ are realizations of i.i.d. RVs whose common probability distribution is $\mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I})$ with some fixed but unknown variance σ^2. Note that $\mathbf{W}$ and σ^2 parametrize the joint probability distribution of the feature vectors $\mathbf{z}^{(i)}$ via (9.11). Probabilistic principal component analysis (PPCA) amounts to maximum likelihood estimation (see Sect. 3.12) of the parameters $\mathbf{W}$ and σ^2. This maximum likelihood estimation problem can be solved using computationally efficient estimation techniques such as EM [6, Appendix B]. The implementation of PPCA via EM also offers a principled approach to handle missing data. Roughly speaking, the EM method allows to use the probabilistic model (9.11) to estimate missing raw features [6, Sect. 4.1].

9.3 Feature Learning for Non-numeric Data

We have motivated dimensionality reduction methods as transformations of (very long) raw feature vectors to a new (shorter) feature vector $\mathbf{x}$ such that it allows to reconstruct $\mathbf{z}$ with minimum reconstruction error (9.1). To make this requirement precise we need to define a measure for the size of the reconstruction error and specify the class of possible reconstruction maps. PCA uses the squared Euclidean norm (9.7) to measure the reconstruction error and only allows for linear reconstruction maps (9.6).

Alternatively, we can view dimensionality reduction as the generation of new feature vectors $\mathbf{x}^{(i)}$ that maintain the intrinsic geometry of the data points with their raw feature vectors $\mathbf{z}^{(i)}$. Different dimensionality reduction methods using different concepts for characterizing the "intrinsic geometry" of data points. PCA defines the intrinsic geometry of data points using the squared Euclidean distances between feature vectors. Indeed, PCA produces feature vectors $\mathbf{x}^{(i)}$ such that for data points whose raw feature vectors have small squared Euclidean distance, also the new feature vectors $\mathbf{x}^{(i)}$ will have small squared Euclidean distance.

Some application domains generate data points for which the Euclidean distances between raw feature vectors does not reflect the intrinsic geometry of data points. As a point in case, consider data points representing scientific articles which can be characterized by the relative frequencies of words from some given set of relevant

words (dictionary). A small Euclidean distance between the resulting raw feature vectors typically does not imply that the corresponding text documents are similar. Instead, the similarity between two articles might depend on the number of authors that are contained in author lists of both papers. We can represent the similarities between all articles using a similarity graph whose nodes represent data points which are connected by an edge (link) if they are similar (see Fig. 8.8).

Consider a dataset $\mathcal{D} = \big(\mathbf{z}^{(1)}, \ldots, \mathbf{z}^{(m)}\big)$ whose intrinsic geometry is characterized by an unweighted similarity graph $\mathcal{G} = \big(\mathcal{V} := \{1, \ldots, m\}\, \mathcal{E}\big)$. The node $i \in \mathcal{V}$ represents the ith data point, with raw feature vector $\mathbf{z}^{(i)}$. Two nodes are connected by an undirected edge if the corresponding data points are similar. We would like to find short feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, such that two data points i, i', whose feature vectors $\mathbf{x}^{(i)}, \mathbf{x}^{(i')}$ have small Euclidean distance, are well-connected to each other. To make this requirement precise we need to define a measure for how well two nodes of an undirected graph are connected. We refer the reader to literature on network theory for an overview and details of various connectivity measures [7].

Let us discuss a simple but powerful technique to map the nodes $i \in \mathcal{V}$ of an undirected graph $\mathcal{G}$ to (short) feature vectors $\mathbf{x}^{(i)} \in \mathbb{R}^n$. This map is such that the Euclidean distances between the feature vectors of two nodes reflect their connectivity within $\mathcal{G}$. This technique uses the Laplacian matrix $\mathbf{L} in \mathbb{R}^{(i)}$ which is defined for an undirected graph $\mathcal{G}$ (with node set $\mathcal{V} = \{1, \ldots, m\}$) element-wise

$$L_{i,j} := \begin{cases} -1 & \text{, if } \{i, j\} \in \mathcal{E} \\ d^{(i)} & \text{, if } i = j \\ 0 & \text{otherwise.} \end{cases} . \tag{9.12}$$

Here, $d^{(i)} := \big|\{j : \{i, j\} \in \mathcal{E}\}\big|$ denotes the degree, or the number of neighbours, of node $i \in \mathcal{V}$. It can be shown that the Laplacian matrix $\mathbf{L}$ is psd [8, Proposition 1]. Therefore we can find an orthonormal set of eigenvectors

$$\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(m)} \in \mathbb{R}^m \tag{9.13}$$

with corresponding (ordered in a non-decreasing fashion) eigenvalues $\lambda_1 \leq \ldots \leq \lambda_m$ of $\mathbf{L}$.

It turns out that, for a prescribed number n of numeric features, the entries $u_i^{(1)}, \ldots, u_i^{(n)}$ of the first n eigenvectors (9.13) result in feature vectors whose Euclidean distances reflect the connectivities of data points in the similarity graph $\mathcal{G}$. For a more precise statement of this informal claim we refer to the excellent tutorial [8]. Thus, we obtain a feature learning method for (non-numeric) data points via using the eigenvectors of the graph Laplacian associated with the similarity graph of the data points. Algorithm 16 summarizes this feature learning method which requires the similarity graph of the dataset and the desired number of new features as input. Note that Algorithm 16 does not make any use of the Euclidean distances between raw feature vectors and uses solely the similarity graph $\mathcal{G}$ for determining the intrinsic geometry of $\mathcal{D}$.

Algorithm 16 Feature Learning for Non-Numeric Data

Input: dataset $\mathcal{D} = \{\mathbf{z}^{(i)} \in \mathbb{R}^d\}_{i=1}^m$; similarity graph $\mathcal{G}$; number n of features to be constructed for each data point.

1: construct the Laplacian matrix $\mathbf{L}$ of the similarity graph (see ((9.12)))
2: compute EVD of $\mathbf{L}$ to obtain n orthonormal eigenvectors (9.13) corresponding to the smallest eigenvalues of $\mathbf{L}$
3: for each data point i, construct feature vector

$$\mathbf{x}^{(i)} := \left(u_i^{(1)}, \ldots, u_i^{(n)}\right)^T \in \mathbb{R}^n \tag{9.14}$$

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$

9.4 Feature Learning for Labeled Data

We have discussed PCA as a linear dimensionality reduction method. PCA learns a compression matrix that maps raw features $\mathbf{z}^{(i)}$ of data points to new (much shorter) feature vectors $\mathbf{x}^{(i)}$. The feature vectors $\mathbf{x}^{(i)}$ determined by PCA depend solely on the raw feature vectors $\mathbf{z}^{(i)}$ of the data points in a given dataset $\mathcal{D}$. In particular, PCA determines the compression matrix such that the new features allow for a linear reconstruction (9.6) with minimum reconstruction error (9.7).

For some application domains we might not only have access to raw feature vectors but also to the label values $y^{(i)}$ of the data points in $\mathcal{D}$. Indeed, dimensionality reduction methods might be used as pre-processing step within a regression or classification problem that involves a labeled training set. However, in its basic form, PCA (see Algorithm 15) does not allow to exploit the information provided by available labels $y^{(i)}$ of data points $\mathbf{z}^{(i)}$. For some datasets, PCA might deliver feature vectors that are not very relevant for the overall task of predicting the label of a data point.

Let us now discuss a modification of PCA that exploits the information provided by available labels of the data points. The idea is to learn a linear construction map (matrix) $\mathbf{W}$ such that the new feature vectors $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$ allow to predict the label $y^{(i)}$ as good as possible. We restrict the prediction to be linear,

$$\hat{y}^{(i)} := \mathbf{r}^T\mathbf{x}^{(i)} = \mathbf{r}^T\mathbf{W}\mathbf{z}^{(i)}, \tag{9.15}$$

with some weight vector $\mathbf{r} \in \mathbb{R}^n$.

While PCA is motivated by minimizing the reconstruction error (9.1), we now aim at minimizing the prediction error $\hat{y}^{(i)} - y^{(i)}$. In particular, we assess the usefulness of a given pair of construction map $\mathbf{W}$ and predictor $\mathbf{r}$ (see (9.15)), using the empirical risk

$$\widehat{L}\big(\mathbf{W}, \mathbf{r} \mid \mathcal{D}\big) := (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \hat{y}^{(i)}\big)^2$$
$$\stackrel{(9.15)}{=} (1/m) \sum_{i=1}^{m} \big(y^{(i)} - \mathbf{r}^T \mathbf{W} \mathbf{z}^{(i)}\big)^2. \tag{9.16}$$

to guide the learning of a compressing matrix $\mathbf{W}$ and corresponding linear predictor weights $\mathbf{r}$ (9.15).

The optimal matrix $\mathbf{W}$ that minimizes the empirical risk (9.16) can be obtained via the EVD (9.2) of the sample covariance matrix $\mathbf{Q}$ (9.9). Note that we have used the EVD of $\mathbf{Q}$ already for PCA in Sect. 9.2 (see (9.8)). Remember that PCA uses the n eigenvectors $\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(n)}$ corresponding to the n largest eigenvalues of $\mathbf{Q}$. In contrast, to minimize (9.16), we need to use a different set of eigenvectors in the rows of $\mathbf{W}$ in general. To find the right set of n eigenvectors, we need the sample cross-correlation vector

$$\mathbf{q} := (1/m) \sum_{i=1}^{m} y^{(i)} \mathbf{z}^{(i)}. \tag{9.17}$$

The entry q_j of the vector $\mathbf{q}$ estimates the correlation between the raw feature $z_j^{(i)}$ and the label $y^{(i)}$. We then define the index set

$$\mathcal{S} := \{j_1, \ldots, j_n\} \text{ such that } \big(q_j\big)^2/\lambda_j \geq \big(q_{j'}\big)^2/\lambda_{j'} \text{ for any } j \in \mathcal{S}, j' \in \{1, \ldots, d\} \notin \mathcal{S}. \tag{9.18}$$

It can then be shown that the rows of the optimal compression matrix $\mathbf{W}$ are the eigenvectors $\mathbf{u}^{(j)}$ with $j \in \mathcal{S}$. We summarize the overall feature learning method in Algorithm 17.

Algorithm 17 Linear Feature Learning for Labeled Data

Input: dataset $\big(\mathbf{z}^{(1)}, y^{(1)}\big), \ldots, \big(\mathbf{z}^{(m)}, y^{(m)}\big)$ with raw features $\mathbf{z}^{(i)} \in \mathbb{R}^d$ and numeric labels $y^{(i)} \in \mathbb{R}$; number n of new features.

1: compute EVD (9.10) of the sample covariance matrix (9.9) to obtain orthonormal eigenvectors $\big(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}\big)$ corresponding to (decreasingly ordered) eigenvalues $\lambda^{(1)} \geq \lambda^{(2)} \geq \ldots \geq \lambda^{(d)} \geq 0$

2: compute the sample cross-correlation vector (9.17) and, in turn, the sequence

$$\big(q_1\big)^2/\lambda_1, \ldots, \big(q_d\big)^2/\lambda_d \tag{9.19}$$

3: determine indices $i_1, \ldots, i_n$ of n largest elements in (9.19)

4: construct compression matrix $\mathbf{W} := \big(\mathbf{u}^{(i_1)}, \ldots, \mathbf{u}^{(i_n)}\big)^T \in \mathbb{R}^{n \times d}$

5: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$

Output: $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and compression matrix $\mathbf{W}$.

The main focus of this section was on regression problems involving data points with numeric labels. Given the raw features and labels of the data point in the dataset

$\mathcal{D}$, Algorithm 17 determines new feature vectors $\mathbf{x}^{(i)}$ that allow to linearly predict a numeric label with minimum squared error. A similar approach can be used for classification problems involving data points with discrete labels. The resulting linear feature learning methods are known as **linear discriminant analysis** or **Fisher discriminant analysis** [3].

9.5 Privacy-Preserving Feature Learning

Many important application domains of ML involve sensitive data that is subject to data protection law [9]. Consider a health-care provider (such as a hospital) holding a large database of patient records. From a ML perspective this databases is nothing but a (typically large) set of data points representing individual patients. The data points are characterized by many features including personal identifiers (name, social security number), bio-physical parameters as well as examination results. We could apply ML to learn a predictor for the risk of particular disease given the features of a data point.

Given large patient databases, the ML methods might not be implemented locally at the hospital but using cloud computing. However, data protection requirements might prohibit the transfer of raw patient records that allow to match individuals with bio-physical properties. In this case we might apply feature learning methods to construct new features for each patient such that they allow to learn an accurate hypothesis for predicting a disease but do not allow to identify sensitive properties of the patient such as its name or a social security number.

Let us formalize the above application by characterizing each data point (patient in the hospital database) using raw feature vector $\mathbf{z}^{(i)} \in \mathbb{R}^d$ and a sensitive numeric property $\pi^{(i)}$. We would like to find a compression map $\mathbf{W}$ such that the resulting features $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$ do not allow to accurately predict the sensitive property $\pi^{(i)}$. The prediction of the sensitive property is restricted to be a linear $\hat{\pi}^{(i)} := \mathbf{r}^T\mathbf{x}^{(i)}$ with some weight vector $\mathbf{r}$.

Similar to Sect. 9.4 we want to find a compression matrix $\mathbf{W}$ that transforms, in a linear fashion, the raw feature vector $\mathbf{z} \in \mathbb{R}^d$ to a new feature vector $\mathbf{x} \in \mathbb{R}^n$. However the design criterion for the optimal compression matrix $\mathbf{W}$ was different in Sect. 9.4 where the new feature vectors should allow for an accurate linear prediction of the label. In contrast, here we want to construct feature vectors such that there is no accurate linear predictor of the sensitive property $\pi^{(i)}$.

As in Sect. 9.4, the optimal compression matrix $\mathbf{W}$ is given row-wise by the eigenvectors of the sample covariance matrix (9.9). However, the choice of which eigenvectors to use is different and based on the entries of the sample cross-correlation vector

$$\mathbf{c} := (1/m) \sum_{i=1}^{m} \pi^{(i)} \mathbf{z}^{(i)}. \tag{9.20}$$

We summarize the construction of the optimal privacy-preserving compression matrix and corresponding new feature vectors in Algorithm 18.

Algorithm 18 Privacy Preserving Feature Learning

Input: dataset $\left(\mathbf{z}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{z}^{(m)}, y^{(m)}\right)$ with raw features $\mathbf{z}^{(i)} \in \mathbb{R}^d$ and (numeric) sensitive property $\pi^{(i)} \in \mathbb{R}$; number n of new features.

1: compute the EVD (9.10) of the sample-covariance matrix (9.9) to obtain orthonormal eigenvectors $\left(\mathbf{u}^{(1)}, \ldots, \mathbf{u}^{(d)}\right)$ corresponding to (decreasingly ordered) eigenvalues $\lambda_1 \geq \lambda_2 \geq \ldots \geq \lambda_d \geq 0$

2: compute the sample cross-correlation vector (9.20) and, in turn, the sequence

$$\left(c_1\right)^2/\lambda_1, \ldots, \left(c_d\right)^2/\lambda_d \tag{9.21}$$

3: determine indices $i_1, \ldots, i_n$ of n smallest elements in (9.21)

4: construct compression matrix $\mathbf{W} := \left(\mathbf{u}^{(i_1)}, \ldots, \mathbf{u}^{(i_n)}\right)^T \in \mathbb{R}^{n \times d}$

5: compute feature vector $\mathbf{x}^{(i)} = \mathbf{W}\mathbf{z}^{(i)}$

Output: privacy-preserving feature vectors $\mathbf{x}^{(i)}$, for $i = 1, \ldots, m$, and compression matrix $\mathbf{W}$.

Algorithm 18 learns a map $\mathbf{W}$ to extract privacy-preserving features out of the raw feature vector of a data point. These new features are privacy-preserving as they do not allow to accurately predict (in a linear fashion) a sensitive property π of the data point. Another formalization for the preservation of privacy can be obtained using information-theoretic concepts. This information-theoretic approach interprets data points, their feature vector and sensitive property, as realizations of RVs. It is then possible to use the mutual information between new features $\mathbf{x}$ and the sensitive (private) property π as an optimization criterion for learning a compression map h (Sect. 9.1). The resulting feature learning method (referred to as privacy-funnel) differs from Algorithm 18 not only in the optimization criterion for the compression map but also in that it allows it to be non-linear [10, 11].

9.6 Random Projections

Note that PCA involves an EVD of the sample covariance matrix $\mathbf{Q}$ (9.9). The computational complexity (e.g., measured by number of multiplications and additions) for computing this EVD is lower bounded by $\min\{D^2, m^2\}$ [12, 13]. This computational complexity can be prohibitive for ML applications with n and m being of the order of millions (which is already the case if the features are pixel values of a 512×512 RGB bitmap, see Sect. 2.1.1).

There is a computationally cheap alternative to PCA (Algorithm 15) for finding a useful compression matrix $\mathbf{W}$ in (9.5). This alternative is to construct the compression matrix $\mathbf{W}$ entry-wise

$$W_{i,j} := a^{(i,j)} \text{ with i.i.d. } a_{i,j} \sim p(a). \tag{9.22}$$

The entries of the matrix (9.22) are realizations of i.i.d. RVs $a_{i,j}$ with some common probability distribution $p(a)$. Different choices for the probability distribution $p(a)$ have been studied in the literature [14]. The Bernoulli distribution is used to obtain a compression matrix with binary entries. Another popular choice for $p(a)$ is the multivariate normal (Gaussian) distribution.

Consider data points whose raw feature vectors $\mathbf{z}$ are located near a s-dimensional subspace of $\mathbb{R}^d$. The feature vectors $\mathbf{x}$ obtained via (9.5) using a random matrix (9.22) allows to reconstruct the raw feature vectors $\mathbf{z}$ with high probability whenever

$$n \geq Cs \log d. \tag{9.23}$$

The constant C depends on the maximum tolerated reconstruction error η (such that $\|\widehat{\mathbf{z}} - \mathbf{z}\|_2^2 \leq \eta$ for any data point) and the probability that the features $\mathbf{x}$ (see)(9.22)) allow for a maximum reconstruction error η [14, Theorem 9.27.].

9.7 Dimensionality Increase

The focus of this chapter is on dimensionality reduction methods that learn a feature map delivering new feature vectors which are (significantly) shorter than the raw feature vectors. However, it might sometimes be beneficial to learn a feature map that delivers new feature vectors which are longer than the raw feature vectors. We have already discussed two examples for such feature learning methods in Sects. 3.2 and 3.9. Polynomial regression maps a single raw feature z to a feature vector containing the powers of the raw feature z. This allows to use apply linear predictor maps to the new feature vectors to obtain predictions that depend non-linearly on the raw feature z. Kernel methods might even use a feature map that delivers feature vectors belonging to an infinite-dimensional Hilbert space [15].

Mapping raw feature vectors into higher-dimensional (or even infinite-dimensional) spaces might be useful if the intrinsic geometry of the data points is simpler when looked at in the higher-dimensional space. Consider a binary classification problem where data points are highly inter-winded in the original feature space (see Fig. 3.7). Loosely speaking, mapping into higher-dimensional feature space might "flatten-out" a non-linear decision boundary between data points. We can then apply linear classifiers to the higher-dimensional features to achieve accurate predictions.

9.8 Exercises

Exercise 9.1 Computational Burden of Many Features Discuss the computational complexity of linear regression. How much computation do we need to compute the linear predictor that minimizes the average squared error on a training set?

Exercise 9.2 Power Iteration The key computational step of PCA amounts to an EVD of the psd matrix (9.9). Consider an arbitrary initial vector $\mathbf{u}^{(r)}$ and the sequence obtained by iterating

$$\mathbf{u}^{(r+1)} := \mathbf{Q}\mathbf{u}^{(r)} / \left\| \mathbf{Q}\mathbf{u}^{(r)} \right\|. \tag{9.24}$$

Under what (if any) conditions for the initialization $\mathbf{u}^{(r)}$ can be ensure that the sequence $\mathbf{u}^{(r)}$ converges to the eigenvector $\mathbf{u}^{(1)}$ of $\mathbf{Q}$ corresponding to its largest eigenvalue λ_1

Exercise 9.3 Linear Classifiers with High-Dimensional Features Consider a training set $\mathcal{D}$ consisting of $m = 10^{10}$ labeled data points $\left(\mathbf{z}^{(1)}, y^{(1)}\right), \ldots, \left(\mathbf{z}^{(m)}, y^{(m)}\right)$ with raw feature vectors $\mathbf{z}^{(i)} \in \mathbb{R}^{4000}$ and binary labels $y^{(i)} \in \{-1, 1\}$. Assume we have used a feature learning method to obtain the new features $\mathbf{x}^{(i)} \in \{0, 1\}^n$ with $n = m$ and such that the only non-zero entry of $\mathbf{x}^{(i)}$ is $x_i^{(i)} = 1$, for $i = 1, \ldots, m$. Can you find a linear classifier that perfectly classifies the training set?

References

1. I. Goodfellow, Y. Bengio, A. Courville, *Deep Learning* (MIT Press, Cambridge, 2016)
2. G. Strang, *Computational Science and Engineering* (Wellesley-Cambridge Press, MA, 2007)
3. T. Hastie, R. Tibshirani, J. Friedman, *The Elements of Statistical Learning*. Springer Series in Statistics (Springer, New York, 2001)
4. J. Wright, Y. Peng, Y. Ma, A. Ganesh, S. Rao, Robust principal component analysis: exact recovery of corrupted low-rank matrices by convex optimization, in *Neural Information Processing Systems, NIPS 2009* (2009)
5. S. Roweis, EM Algorithms for PCA and SPCA. *Advances in Neural Information Processing Systems* (MIT Press, Cambridge, 1998), pp. 626–632
6. M.E. Tipping, C. Bishop, Probabilistic principal component analysis. J. Roy. Stat. Soc. B **21**(3), 611–622 (1999)
7. M.E.J. Newman, *Networks: An Introduction* (Oxford University Press, Oxford, 2010)
8. U. von Luxburg, A tutorial on spectral clustering. Stat. Comput. **17**(4), 395–416 (2007)
9. S. Wachter, Data protection in the age of big data. Nat. Electron. **2**(1), 6–7 (2019)
10. A. Makhdoumi, S. Salamatian, N. Fawaz, M. Médard, From the information bottleneck to the privacy funnel, in *2014 IEEE Information Theory Workshop (ITW 2014)*, pp. 501–505 (2014)
11. Y.Y. Shkel, R.S. Blum, H.V. Poor, Secrecy by design with applications to privacy and compression. IEEE Trans. Inf. Theory **67**(2), 824–843 (2021)
12. Q. Du, J. Fowler, Low-complexity principal component analysis for hyperspectral image compression. Int. J. High Perform. Comput. Appl., pp. 438–448 (2008)
13. A. Sharma, K. Paliwal, Fast principal component analysis using fixed-point analysis. Pattern Recogn. Lett. **28**, 1151–1155 (2007)
14. S. Foucart, H. Rauhut, *A Mathematical Introduction to Compressive Sensing* (Springer, New York, 2012)
15. B. Schölkopf, A. Smola, *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond* (MIT Press, Cambridge, 2002)