Prof. Dr. Michael Griebel
Prof. Dr. Jochen Garcke
Dr. Bastian Bohn
Jannik Schürg

3 PRINCIPAL COMPONENT ANALYSIS

Send your solutions to this chapter's tasks until June 5th.

A typical phenomenon of datasets is *redundancy*. From a statistical perspective one could say, that the measured variables in our data are typically not independent but correlated. The consequence is, that knowing a small number of values—derived from our variables—might already describe the full dataset. Thus, the allegedly high-dimensional representation of many datasets is actually not their most compact form.[1] This is a common practical observation, and it motivates several questions. Vaguely speaking one tries to find a mapping into a lower dimensional representation space which preserves (or selects) the relevant information. In this and in the following chapter we will look into two methods which are examples for such mappings. Both belong to the domain of *dimensionality reduction methods*, which are important tools in data science.

A difference in methodology to the previous chapters is, that no absolute solution criterion is usually known. While any good solution for a regression problem will provide a good approximation to the ground truth f, in dimensionality reduction there is usually no single best answer. Two representations could capture or highlight different aspects of the data and might be useful for different applications. One could say that the condition "preserves information" or "reduces redundancy" leaves more space for different objective functions to minimize, which might also depend on the task at hand.

3.1 REASONS FOR REDUCING THE DIMENSION

The dimension d of a set of measurements can be too big by several means. For example it is *too big to understand.* A typical task for a given dataset is to answer a certain question or gaining new insights about inner workings. Here, a good visualization is a powerful tool to develop an idea or intuition on what is happening. The more dimensions our data has the more challenging it is to visualize it. While finding a

[1] If you think of data compression you are not too far off, e.g. the term *redundancy* is also defined in information theory—which by the way provides a more rigorous investigation into terms we here use vaguely.

suitable visualization is a topic on its own,[2] reducing the dimension can be a helpful start.

A computational reason for reducing d is scaling behavior. The runtime of many algorithms scales polynomially or even exponentially in the dimension of the input data. Furthermore, some methods do not perform well in high dimensions. We already saw for example that k-nearest neighbors does only work for low dimensional data.

Therefore dimensionality reduction as a preprocessing step can provide more expressive features for a learning algorithm. Another example would be non-linearly separable input whose low dimensional representation is linearly separable which then can used in a linear model. This is comparable to the kernel trick allowing SVMs to separate non-linear data but in the other direction.[3] Also keep in mind that a low dimensional representation can be a regularization entailed by the reduction algorithm. The reason is that a certain model assumption is usually enforced.

In this sheet we look at representations gained through a linear transformation.

LINEAR REPRESENTATIONS

Suppose we conducted an experiment and gained measurements in two variables, X_1 and X_2. The plotted values might look as in fig. 3.1. The data is two dimensional but visibly correlated: If X_1 is large so is X_2 (i.e. the covariance is positive). In this case one might reduce the dimension to 1 and use a fitted line where each measurement is projected onto the line.

Then, the 1D values will be the points on this line, where we think of the line as a new axis. There is some freedom in this representation, namely in picking the center of this axis. Different centers will shift the 1D values.

We might be interested in two things in the calculation. The first could be the line itself corresponding to the mapping into a low dimensional representation, and the second would be the projections of our data onto this line, i.e. the coordinates w.r.t. the new axis.

In higher dimensions it might not be sufficient to have only one axis (e.g. a plane in 3D). So generally we are interested in the set of orthogonal axes which can represent the most variation in the data.

[2] The books of Edward R. Tufte might be useful for more information on great visualization from graphics design perspective.

[3] Often dimensionality reduction algorithms are thought of as way of finding a feature map, does this ring a bell with regard to kernels?

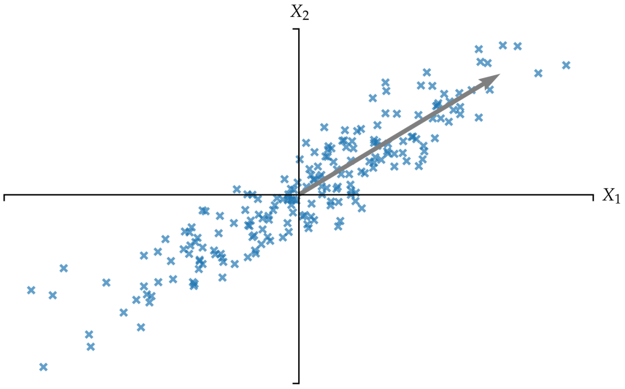

Figure 3.1: Plot of two measured variables in an experiment.

3.2 DERIVATION OF PCA

In this chapter we look into the classic method of *principal component analysis* (PCA). Despite its age [5, 9] it is the most popular method for linear dimensionality reduction.

Skip to theorem 3.1.

The goal is to find a linear transformation $P\colon \mathbb{R}^d \to \mathbb{R}^q$, with $q < d$, for a set of data points $\{\mathbf{x}_i\}_{i=1}^n \subset \mathbb{R}^d$ such that the new coordinates in $\mathbb{R}^q$ describe the data (approximately).

More precisely we try to find a linear affine model $f\colon \mathbb{R}^q \to \mathbb{R}^d$ s.t.

$$f(\boldsymbol{\lambda}_i) = \boldsymbol{\mu} + \mathbf{V}_q\boldsymbol{\lambda}_i, \tag{3.1}$$

where $\mathbf{V}_q \in \mathbb{R}^{d\times q}$ is a matrix whose columns form an orthonormal basis, $\boldsymbol{\mu} \in \mathbb{R}^d$ is a location vector, and $\boldsymbol{\lambda}_i = P(\mathbf{x}_i)$ are q dimensional coordinate vectors for the inputs $\mathbf{x}_i$, $i = 1, \dots, n$.

We fit the model using the least squares error

$$\sum_{i=1}^n \|\mathbf{x}_i - (\boldsymbol{\mu} + \mathbf{V}_q\boldsymbol{\lambda}_i)\|^2$$

with respect to $\boldsymbol{\mu}$, $\mathbf{V}_q$ and $\boldsymbol{\lambda}_i$.

To solve the system, first split up the optimization problem as

$$\min_{\boldsymbol{\mu},\mathbf{V}_q,\{\boldsymbol{\lambda}_i\}} \sum_{i=1}^n \|\mathbf{x}_i - (\boldsymbol{\mu} + \mathbf{V}_q\boldsymbol{\lambda}_i)\|^2 = \min_{\mathbf{V}_q} \min_{\boldsymbol{\mu},\{\boldsymbol{\lambda}_i\}} \sum_{i=1}^n \|\mathbf{x}_i - (\boldsymbol{\mu} + \mathbf{V}_q\boldsymbol{\lambda}_i)\|^2.$$

We solve the inner problem by using the first order necessary optimality condition

$$\begin{aligned} &\sum_{i=1}^n (\boldsymbol{\mu} - \mathbf{x}_i + \mathbf{V}_q\boldsymbol{\lambda}_i) = 0, \\ &\mathbf{V}_q^\top \mathbf{V}_q\boldsymbol{\lambda}_i - \mathbf{V}_q^\top(\mathbf{x}_i - \boldsymbol{\mu}) = 0 \quad \text{for} \quad i = 1, \dots, n, \end{aligned}$$

which is also a sufficient optimality criterion in this case since the objective function is convex.

With $\mathbf{V}_q^\top \mathbf{V}_q = \mathbf{I}_q$ this is equivalent to

Here $\bar{\mathbf{x}} = \frac{1}{n}\sum_{i=1}^n \mathbf{x}_i$ denotes the mean.

$$\begin{aligned}(\mathbf{I}_d - \mathbf{V}_q\mathbf{V}_q^\top)\boldsymbol{\mu} &= (\mathbf{I}_d - \mathbf{V}_q\mathbf{V}_q^\top)\bar{\mathbf{x}}, \\ \boldsymbol{\lambda}_i &= \mathbf{V}_q^\top(\mathbf{x}_i - \boldsymbol{\mu}) \quad \text{for} \quad i = 1, \dots, n.\end{aligned} \tag{3.2}$$

The system in eq. (3.2) is under-determined (for $q < d$). The matrix $\mathbf{I}_d - \mathbf{V}_q\mathbf{V}_q^\top$ is the projection[4] on the orthogonal complement $\mathbf{V}_q^\perp$. Since $\bar{\mathbf{x}}$ is a solution, all solutions to eq. (3.2) are of the form $\boldsymbol{\mu} = \bar{\mathbf{x}} + \operatorname{span}(\mathbf{V}_q)$. So a solution to the inner minimization problem is

$$\boldsymbol{\mu} = \bar{\mathbf{x}} \quad \text{and} \quad \boldsymbol{\lambda}_i = \mathbf{V}_q^\top(\mathbf{x}_i - \bar{\mathbf{x}}) \quad \text{for} \quad i = 1, \dots, n.$$

This leaves us with

$$\min_{\mathbf{V}_q} \sum_{i=1}^n \|(\mathbf{I}_d - \mathbf{V}_q\mathbf{V}_q^\top)(\mathbf{x}_i - \bar{\mathbf{x}})\|^2 \quad \text{subject to} \quad \mathbf{V}_q^\top\mathbf{V}_q = \mathbf{I}_q.$$

So we minimize the projection error onto the subspace[5] spanned by the columns in $\mathbf{V}_q$. It is straightforward to calculate that the above minimization is equivalent to

$$\max_{\mathbf{V}_q} \sum_{i=1}^n \|\mathbf{V}_q^\top(\mathbf{x}_i - \bar{\mathbf{x}})\|^2 \quad \text{subject to} \quad \mathbf{V}_q^\top\mathbf{V}_q = \mathbf{I}_q. \tag{3.3}$$

Let $\mathbf{X} \in \mathbb{R}^{n\times d}$ be the matrix with rows $(\mathbf{x}_i - \bar{\mathbf{x}})^\top$, and denote the columns of $\mathbf{V}_q$ by $\mathbf{v}_i$.

Clearly, for any solution $\mathbf{V}_q$ we would obtain new solutions by composing with an orthogonal transformation. To encode further semantics we look for a solution $\mathbf{V}_q$ such that the matrices $\mathbf{V}_k$ containing the first $k \leq q$ columns are a solution to the corresponding smaller problems

$$\max_{\mathbf{V}_k} \sum_{i=1}^n \|\mathbf{V}_k^\top(\mathbf{x}_i - \bar{\mathbf{x}})\|^2 \quad \text{subject to} \quad \mathbf{V}_k^\top\mathbf{V}_k = \mathbf{I}_k.$$

In this way, the columns are ordered w.r.t. their contribution to the error reduction.

Using Lagrange multipliers and induction over q one can show that such an optimal solution must fulfill

$$(\mathbf{X}^\top\mathbf{X})\mathbf{v}_k = \lambda_k\mathbf{v}_k \quad \text{for} \quad k = 1, \dots, q,$$

for some $\lambda_k \in \mathbb{R}$, $k = 1, \dots, q$. So the columns $\mathbf{v}_i$ of $\mathbf{V}_q$ must be eigenvectors of $\mathbf{X}^\top\mathbf{X}$.

[4] This is exactly the term in Gram-Schmidt.

[5] This can be also seen as the reconstruction error from going to the low dimensional representation and back, as in the original introduction of PCA [9].

Now consider the *singular value decomposition* (SVD) of $\mathbf{X}$:

$$\mathbf{X} = \mathbf{U}\mathbf{D}\mathbf{W}^\top \quad \text{with} \quad \mathbf{D} = \operatorname{diag}(\sigma_1, \dots, \sigma_d),$$
$$(\text{such that } \sigma_1 \geq \sigma_2 \geq \dots \geq \sigma_d \geq 0).$$

One can see that the columns of $\mathbf{W}$ are eigenvectors of $\mathbf{X}^\top\mathbf{X}$ with eigenvalues $\sigma_1^2, \dots, \sigma_d^2$.

Since the columns of $\mathbf{W}$ and of $\mathbf{V}_q$ form an orthonormal basis, we can find a family of orthogonal maps[6] $F_{\mathbf{V}_q}$ such that the image of all basis vectors from $\mathbf{W}$ includes the basis vectors of $\mathbf{V}_q$, and all eigenspaces map into themselves. We denote with $\hat{\mathbf{W}} := F_{\mathbf{V}_q}(\mathbf{W})$ the matrix whose columns are mapped column–wise and obtain the SVD

Effectively, we now pick a SVD matching our orthonormal basis vectors from $\mathbf{V}_q$*. The SVD is not unique in* $\mathbf{W}$ *and* $\mathbf{U}$*.*

$$\mathbf{X} = \hat{\mathbf{U}}\mathbf{D}\hat{\mathbf{W}}^\top,$$

for an orthogonal matrix $\hat{\mathbf{U}}$ which can be derived from $\mathbf{D}$ and $\hat{\mathbf{W}}$.

So $\mathbf{V}_q^\top\hat{\mathbf{W}} \in \mathbb{R}^{q\times d}$ is a permutation matrix. For any vector $\mathbf{z}$ follows

$$\|\mathbf{V}_q^\top\hat{\mathbf{W}}\mathbf{z}\|^2 = \sum_{k\in\Gamma} \mathbf{z}_k^2 \tag{3.4}$$

for certain indices $\Gamma \subseteq \{1, \dots, d\}$. We obtain

$$\begin{aligned}\sum_{i=1}^n \|\mathbf{V}_q^\top(\mathbf{x}_i - \bar{\mathbf{x}})\|^2 &= \sum_{i=1}^n \|\mathbf{V}_q^\top\hat{\mathbf{W}}\mathbf{D}^\top\hat{\mathbf{U}}^\top\mathbf{e}_i\|^2 \\ &= \sum_{i=1}^n \sum_{k\in\Gamma} (\sigma_k \hat{\mathbf{U}}_{i,k})^2 \qquad \text{(use eq. (3.4))} \\ &= \sum_{k\in\Gamma} \sigma_k^2. \qquad \text{(columns of } \hat{\mathbf{U}} \text{ have norm 1)}\end{aligned}$$

Therefore we can solve the problem from eq. (3.3) by letting the columns of $\mathbf{V}_q$ be orthogonalized unit eigenvectors belonging to the q largest eigenvalues.

Let us summarize this in the following Theorem.

Theorem 3.1. *Let* $\{\mathbf{x}_1, \dots, \mathbf{x}_n\}$ *be a set of vectors from* $\mathbb{R}^d$*, and let* $q \in \mathbb{N}$ *with* $q \leq d$*. The minimization problem*

$$\min_{\substack{\boldsymbol{\mu}\in\mathbb{R}^d, \boldsymbol{\lambda}_i\in\mathbb{R}^q,\\ \mathbf{V}_q\in\mathbb{R}^{d\times q}}} \sum_{i=1}^n \|\mathbf{x}_i - (\boldsymbol{\mu} + \mathbf{V}_q\boldsymbol{\lambda}_i)\|^2 \quad \textit{subject to} \quad \mathbf{V}_q^\top\mathbf{V}_q = \mathbf{I}_q$$

is solved for $\boldsymbol{\mu} = \bar{\mathbf{x}}$*,* $\boldsymbol{\lambda}_i = \mathbf{V}_q^\top(\mathbf{x}_i - \bar{\mathbf{x}})$*, and* $\mathbf{V}_q$ *contains column-wise* q *orthogonalized unit eigenvectors of* $\mathbf{X}^\top\mathbf{X}$ *for the* q *largest eigenvalues.*

Remark. The solution is not unique. The orthonormal basis of each eigenspace is only unique up to an orthogonal transformation (rotation).

[6] Complete the basis $\mathbf{V}_q$, then the linear map that maps the corresponding basis vectors on to each other does the job.

3.2.1 *Numerical Computation*

We have seen two basic problems we could solve to get a PCA:

1. Compute eigenvectors of $\mathbf{X}^\top \mathbf{X}$.
2. Compute a singular value decomposition of $\mathbf{X}$.

Of course both are related and several solvers exist for each. An optimal choice might depend on the the size of n, p, and d, and if speed is more important than accuracy.

Let us implement the PCA and test it on a toy example we understand. The accompanying material includes a Jupyter notebook which should be used as a template for your solution. The material also includes the datasets and some Python code you may use. Please check the `README` file for further details.

Task 3.1. *Implement a PCA routine whose inputs are* $\{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$ *and* q*. Use a NumPy/SciPy routine to compute either eigenvectors directly or use a SVD. In the case of eigenvectors, make sure your routine does actually return a orthonormal basis (consult the documentation of your solver).*

The toy example is a 2D dataset which is embedded into 4D by rotating and translating it, also noise was added. We would like to recover the 2D representation. Each point in the dataset is labeled among five categories. We also check the results we get for non-linear data.

Task 3.2. *Test your PCA implementation on the provided toy dataset.*

a) *Plot a slice of the 4D toy data. Compute the PCA representation for* $q = 2$ *and plot it as described in the notebook. To check you result, the plot should reveal a perfectly round and familiar shape.*

b) *Map the 2D representation onto a distorted ellipse in 3D, the code for the transformation is provided. Do a PCA of this 3D data for* $q = 2$ *and plot the result. Repeat this for a few ellipses and describe in words how the PCA does pick the coordinate system* $\mathbf{V}_q$*.*

3.3 STATISTICAL PERSPECTIVE

The method we described is known by several names. The name PCA is especially popular in statistics [5]. We quickly sketch the statistics approach to get another perspective of the method.

Suppose we have a vector–valued random variable $\hat{\mathbf{X}} \in \mathbb{R}^d$. The covariance matrix is a real $d \times d$ matrix and it is defined as

$$\operatorname{Cov}[\hat{\mathbf{X}}]_{ij} := \mathbb{E}\big[(\hat{\mathbf{X}}_i - \mathbb{E}[\hat{\mathbf{X}}_i])(\hat{\mathbf{X}}_j - \mathbb{E}[\hat{\mathbf{X}}_j])\big].$$

To simplify notation we assume that $\hat{\mathbf{X}}$ has zero mean, so

$$\operatorname{Cov}[\hat{\mathbf{X}}] = \mathbb{E}[\hat{\mathbf{X}}^\top \hat{\mathbf{X}}].$$

Let $\mathbf{v}_1 \in \mathbb{R}^d$ be a vector with unit-norm. We consider the real random variable Y_1 defined by $Y_1 := \mathbf{v}_1^\top \hat{\mathbf{X}}$. We now wish to maximize its variance

$$\mathrm{Var}[Y_1] := \mathbb{E}\left[(Y_1 - \mathbb{E}[Y_1])^2\right] = \mathbb{E}[Y_1^2]$$

w.r.t. to the direction $\mathbf{v}_1$. This is motivated by the idea, that since the variance is a measure of how much a random variable varies, we want to find the direction along which our data varies the most. This direction is defined to be a the *first principal component*.

The idea is iteratively continued by searching normed directions $\mathbf{v}_2, \mathbf{v}_3, \ldots,$ perpendicular to all previous ones, and maximizing the variance subject to

$$\mathrm{Var}[\mathbf{v}_1^\top \hat{\mathbf{X}}] \geq \mathrm{Var}[\mathbf{v}_2^\top \hat{\mathbf{X}}] \geq \ldots.$$

It turns out that a solution is given by picking eigenvectors of $\mathbb{E}[\hat{\mathbf{X}}^\top \hat{\mathbf{X}}]$ belonging to the largest eigenvalues. Moreover, the eigenvalues λ_i are equal to the variances of the random variables $Y_i := \mathbf{v}_i^\top \hat{\mathbf{X}}$.

The dots to our derivation can be connected by switching to approximations of the expected values using observed values (samples) for $\hat{\mathbf{X}}$, which we shall denote by $\mathbf{x}_1, \ldots, \mathbf{x}_n$. Following this idea we notice that the singular values from above fullfil $\sigma_i^2 \approx \lambda_i = \mathrm{Var}[Y_i]$.

A common approach on how to decide how large q should be chosen is to look at the variances σ_i^2. One usually picks q as the smallest number such that σ_i^2 is below a given threshold, or such that the *percentage of captured variance*[7]

$$\frac{\sum_{i=1}^q \mathrm{Var}[\mathbf{v}_i^\top \hat{\mathbf{X}}]}{\mathrm{Var}[\hat{\mathbf{X}}]} = \frac{\sum_{i=1}^q \lambda_i}{\sum_{i=1}^d \lambda_i} \approx \frac{\sum_{i=1}^q \sigma_i^2}{\sum_{i=1}^d \sigma_i^2} \tag{3.5}$$

is above a given threshold.

We revisit the Iris dataset to check how PCA can be utilized there.

Task 3.3. *Use PCA for the Iris dataset:*

a) Compute all (four) singular values of $\mathbf{X}$ *using a suitable function from* NumPy*. Compute the captured variance percentage from eq. (3.5) when using only the first principal component, and for the first two principal components.*

b) Compute the PCA transformation onto the first two principal components of the Iris dataset. Plot the transformed data in a 2D scatter plot such that the three labels are distinguishable. Do the same for 1D and use the provided function to plot it. What do you observe?

[7] Also known as percentage of explained variance.

c) *Using the insights from the visualization in b) build classifiers for the whole Iris dataset which use a 4D, 2D, or 1D PCA respectively as the first step, and two linear SVMs as a second and third step to classify a data-point as one of the three labels. You can copy your SVM code from sheet 2, but we recommend using* SCIKIT-LEARN.

With a better understanding of PCA we now turn our attention to the analysis of a larger dataset.

3.4 PEDESTRIAN CLASSIFICATION

Detecting people in video footage is a topic interesting for many entities. One important application would be in the self-driving car industry. The task can be split broadly into several connected steps:

- How to gather the image material.
- Find region proposals in a image which could be interesting.
- Decide if a given region shows a pedestrian.

We dive deeper into the last problem. The second step is known as a *segmentation problem* in computer vision, it is also tackled as an unsupervised learning problem.

Our dataset consists of labeled gray-scale pictures of size 25x50 pixel, it is derived from the Daimler dataset [4, 8]. Half of the pictures show a pedestrian, the other half does not. A separation into training and test data is provided. We begin by preparing the data.

Task 3.4. *Prepare the data.*

a) *The data is given as a MATLAB data file, which can be read using* SciPy. *The data is structured in the MATLAB file using matrices with the following names:*

   - `ped_train_int_25x50`: *Training data of 1500 images showing a pedestrian.*
   - `garb_train_int_25x50`: *Training data of 1500 images not showing a pedestrian.*
   - `ped_test_int_25x50`: *Test data of 500 images showing a pedestrian.*
   - `garb_train_int_25x50`: *Test data of 500 images not showing a pedestrian.*

   Each matrix has 1+25·50 columns corresponding to a label (-1 or 1) at the beginning and the grayscale values for each pixel. Read the matrices into NumPy *arrays.*

b) *Normalize the pixel values to* $[0, 1]$, *i.e. apply a suitable affine transformation.*

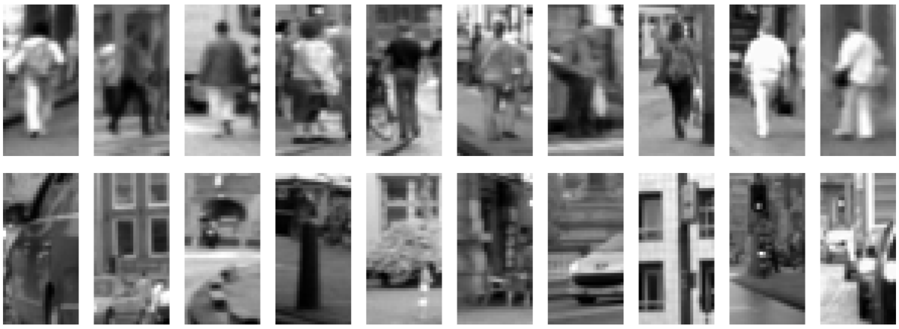

Figure 3.2: Ten pedestrian images and ten garbage (non-pedestrian) images from the dataset.

c) *Write a routine* `plot_im` *to plot a gray-scale image using* MatPlotLib*'s* `imshow` *(to get consistent contrast, provide constant values for its arguments* `vmin` *and* `vmax`*). Create a plot with ten randomly chosen training images showing a pedestrian and ten randomly chosen training images not showing a pedestrian. You can use the* `subplot` *method*[8] *from* MatPlotLib*.*

Our training data consists of $n = 3000$ points, with dimension $d = 1250$ (the pixels of an image). Compared to what we saw so far, one could say that the dimension is quite large. Arguably, visualization of a single data-point, i.e. image, is straight-forward, but our goal is to classify the images using the grayscale values of the pixels. Trying to find an algorithm by just looking at the numbers is hard.[9]

Unsurprisingly, our next step is to compute a PCA in order to reduce d to a much smaller number. Note that the coordinate axes computed by the PCA can be interpreted as images and we represent the data in terms of coefficients for the corresponding eigenvectors, which are called *eigenpedestrians*.

Task 3.5. *Take a look at the eigenpedestrians. From now on use the PCA implementation of* scikit-learn*.*

a) *Compute the PCA with $q = d$ for the full training set (i.e. with pedestrian and non-pedestrian samples combined).*

b) *Plot the first 20 eigenpedestrians, as well as eigenpedestrians 50 to 60 and eigenpedestrians 100 to 110. What do you observe? Provide a guess on what these eigenpedestrians might encode.*

We now use our PCA representation to train a linear SVM.

Task 3.6. *Train a linear SVM (use* `LinearSVC` *from* scikit-learn*) using the PCA representation of the full training dataset for 50 equally spaced values for q between 10 and 1250. For each q compute and store the prediction accuracy*

[8] You should implement an optional argument `ax` for `plot_im`, so you can use it for the subplot, see `plt.gca()`.

[9] This is the problem fundamental to the field of computer vision.

(use the `score` *method) on the training and test dataset. Plot the scores for q. Which q seems the best choice? Compare the situation to task 3.3 c) w.r.t. q.*

The prediction results are not bad, but for self–driving cars probably not acceptable.

3.4.1 *Histogram of Oriented Gradients*

In order to improve the prediction quality we use a handcrafted feature map as an additional step before applying PCA. In particular we will use the so called *Histogram of Oriented Gradients* (HOG), which became quite popular after the well–received experiments by Dalal and Triggs [2] were published. Before we explain what HOG is we take a quick ex-course into computer vision.

A very common tool in computer vision are image gradients. Starting from the difference quotient

$$\lim_{\varepsilon\to 0}\frac{f(x+\varepsilon)-f(x)}{\varepsilon}$$

we can define a gradient for images by using a discrete approximation. Let $\mathbf{I}\in\mathbb{R}^{h\times w}$ be a matrix representing an image. The entries of the matrix shall be gray-scale values. We can compute the partial derivatives in y- and x-direction by using a centered difference quotient, for example

$$\frac{\mathbf{I}_{y+1,x}-\mathbf{I}_{y-1,x}}{1}\text{ and }\frac{\mathbf{I}_{y,x+1}-\mathbf{I}_{y,x-1}}{1}. \tag{3.6}$$

One can represent this operation by a *convolution*—whose continuous counterpart might be known from integration theory. Given a *filter* matrix $\mathbf{K}\in\mathbb{R}^{p\times q}$ we define the convolution $\mathbf{I}*\mathbf{K}$ of $\mathbf{K}$ and $\mathbf{I}$ point-wise by

We use zero-based indexing

$$(\mathbf{I}*\mathbf{K})_{y,x} := \sum_{\substack{0\le i<p\\0\le j<q}}\mathbf{K}_{i,j}\mathbf{I}_{y+k_y-i,x+k_x-j}$$

with *shifts*[10] k_x and k_y. The result is a new image whose entries were computed using weighted sums of the surroundings of each pixel of $\mathbf{I}$. Another example for a popular filter is Gaussian smoothing, here the entries of $\mathbf{K}$ are computed using a Gaussian kernel.

The new image is either smaller, or one has to specify how missing image values outside of the border of $\mathbf{I}$ are extrapolated. In our case we are extending $\mathbf{I}$ by the constant value 0. The y- and x-derivatives can be computed using `scipy.ndimage.convolve` with the filters $[-1,0,1]^\top$ and $[1,0,-1]$.

In HOG the gradient norm and direction is computed for each pixel. Each (y,x)-position is *binned* into square *cells* of size $|c|\in\mathbb{N}$ spanning

[10] In SciPy by default these are chosen to center the filter.

over $|c|$ pixels in both directions and forming a regular grid. Similarly, the orientations are binned into $\#b \in \mathbb{N}$ equally sized intervals which partition either the full circle $\alpha := 2\pi$ or only the half circle $\alpha := \pi$, in which case one calls the orientations *unsigned*. The intervals are

$$\left[(\alpha/\#b)i, (\alpha/\#b)(i+1)\right), \quad \text{for } i = 0, \dots, \#b - 1.$$

In other words, the gradient norms are accumulated into a histogram[11] $H(c_y, c_x, b_i)$ where we collect

- the y- and x-positions into cells, indexed by (c_y, c_x), and
- the orientations into the former intervals for each cell, indexed by b_i.

More precisley, for a gradient norm $\|\nabla_{y,x}\|$ and direction $\phi \in [-\pi, \pi]$ at position (y, x) we consider the interval whose center is below or equal to

$$\hat{\phi} := \phi \mod \alpha,$$

and the interval whose center is above $\hat{\phi}$ (wrapping around), as well as preceding and succeding cells in y- and x-direction in the same fashion, i.e. with respect to their center[12] (out of bound cells are ignored). The gradient norms $H(c_y, c_x, b_i)$ for those neighbors are then updated by adding a fraction of the gradient norm at position (y, x). The fraction for each $H(c_y, c_x, b_i)$ is defined by the coefficients of a convex combination whose terms can be found in algorithm 3.3.

The last step is to combine the cells into blocks, which are then normalized and clipped. If $|B| \in \mathbb{N}$ is the block size, then a block consists of $|B|$ consecutive cells in y-direction and $|B|$ consecutive cells in x-direction. The blocks do overlap, but only full blocks are considered. Consequently, $|B|$ has to chosen less than or equal to the minimum number of cells in y- and x-direction respectively. The HOG feature vector consists of the entries of all blocks (in any order).

Task 3.7. *Test the HOG features for the dataset.*

a) Implement algorithm 3.3 either yourself in Python, or build a Python binding to the provided C++ implementation.

b) Repeat the experiment from task 3.6 for 25 values of q between 10 and 360 but use the HOG features as input for the PCA instead.

Be aware that task 3.7 is expected to be challenging. If you want to implement algorithm 3.3 in Python you should try to avoid stating the loops over all pixels as Python loops. Instead, a suitable NumPy routine should be used. This means the algorithm as stated here should not be implemented in a straightforward way with NumPy. A way to use

[11] The histogram was introduced by Karl Pearson, the inventor of PCA [9].

[12] The NumPy routines `np.digitze` and `np.arange` can be useful for these steps.

Algorithm 3.3 Computation of HOG-features.

```
Input: An image $\mathbf{I} \in \mathbb{R}^{h \times w}$, a number of bins $\#b$ (default: 9), a cell size $|c|$ (default: 8), a block size $|B|$ (default: 2), whether to use unsigned directions (default: yes), and a clip value $C$ (default: 0.2).
Output: A feature vector.

$\alpha \leftarrow \begin{cases} \pi & \text{if use unsigned directions,} \\ 2\pi & \text{else.} \end{cases}$
$|b| \leftarrow \alpha / \#b$
Initialize $H(c_y, c_x, b_i)$ to zero for all cell indices $c_y$, $c_x$ and all bin indices $b_i$.
for all pixel positions $y, x$ do
    $\mathrm{d}y$, $\mathrm{d}x \leftarrow$ derivatives at $y, x$ using zero boundary conditions
    $\hat{\varphi} \leftarrow \operatorname{atan2}(\mathrm{d}y, \mathrm{d}x) \mod \alpha$
    $b_\text{prec} \leftarrow$ index of orientation interval preceding the orientation $\hat{\varphi}$ w.r.t. the interval's center (can be -1)
    $c_{x,\text{prec}} \leftarrow$ index of horizontally preceding cell w.r.t. its center (can be -1)
    $c_{y,\text{prec}} \leftarrow$ index of vertically preceding cell w.r.t. its center (can be -1)
    $f_b \leftarrow \frac{\hat{\varphi} - [b_\text{prec}|i| + \frac{1}{2}|i|]}{|i|}$
    $f_x \leftarrow \frac{x - [(c_{x,\text{prec}}+1)|c| - \frac{1}{2}|c| - 0.5]}{|c|}$
    $f_y \leftarrow \frac{y - [(c_{y,\text{prec}}+1)|c| - \frac{1}{2}|c| - 0.5]}{|c|}$
    $\|\nabla_{y,x}\| \leftarrow \sqrt{(\mathrm{d}y)^2 + (\mathrm{d}x)^2}$
    $b_\text{succ} \leftarrow b_\text{prec} + 1 \mod \#b,\ b_\text{prec} \leftarrow b_\text{prec} \mod \#b$
    $c_{x,\text{succ}} \leftarrow c_{x,\text{prec}} + 1,\ c_{y,\text{succ}} \leftarrow c_{y,\text{prec}} + 1$
    (The cells with indices -1 and $\lfloor h/|c| \rfloor$ or $\lfloor w/|c| \rfloor$ can be stored in $H$ but must not be considered for the blocks)
    $H(c_{y,\text{prec}}, c_{x,\text{prec}}, b_\text{prec}) \leftarrow \text{add } \|\nabla_{y,x}\|\,(1-f_x)\,(1-f_y)\,(1-f_b)$
    $H(c_{y,\text{prec}}, c_{x,\text{prec}}, b_\text{succ}) \leftarrow \text{add } \|\nabla_{y,x}\|\,(1-f_x)\,(1-f_y)\,f_b$
    $H(c_{y,\text{succ}}, c_{x,\text{prec}}, b_\text{prec}) \leftarrow \text{add } \|\nabla_{y,x}\|\,(1-f_x)\,f_y\,(1-f_b)$
    $H(c_{y,\text{succ}}, c_{x,\text{prec}}, b_\text{succ}) \leftarrow \text{add } \|\nabla_{y,x}\|\,(1-f_x)\,f_y\,f_b$
    $H(c_{y,\text{prec}}, c_{x,\text{succ}}, b_\text{prec}) \leftarrow \text{add } \|\nabla_{y,x}\|\,f_x\,(1-f_y)\,(1-f_b)$
    $H(c_{y,\text{prec}}, c_{x,\text{succ}}, b_\text{succ}) \leftarrow \text{add } \|\nabla_{y,x}\|\,f_x\,(1-f_y)\,f_b$
    $H(c_{y,\text{succ}}, c_{x,\text{succ}}, b_\text{prec}) \leftarrow \text{add } \|\nabla_{y,x}\|\,f_x\,f_y\,(1-f_b)$
    $H(c_{y,\text{succ}}, c_{x,\text{succ}}, b_\text{succ}) \leftarrow \text{add } \|\nabla_{y,x}\|\,f_x\,f_y\,f_b$
end for
for all blocks do
    Normalize the block w.r.t. the Euclidean (vector) norm.
    Clip the block entries to be below $C$, normalize again.
    Add the block entries to the feature vector.
end for
```

NumPy more efficiently is to compute all inner variables (f_x, $c_{x,\text{prec}}$, ...) as matrices containing the results for all pixels. Then each update of the histogram can be done cell–wise, here `scipy.ndimage.sum` could be handy. To test your implementation values for intermediate steps of the algorithm are available, see the notebook for details.

Building a C++ binding is a quite technical task and therefore challenging as well. Some experience in programming Python and C, C++, or Fortran could be very helpful. A skeleton of the binding with learning resources in the README is provided in the accompanying material.

3.5 OUTLOOK

The HOG features managed to increase the prediction rate notably. This was around 2006 and since then development did not stop. First, the Daimler dataset is no longer used as a benchmark for recent pedestrian classification algorithms, presumably because other datasets are larger and more challenging, also color information proved to be valuable for achieving better results. Still, the HOG features are used as a step in some of the most successful methods. For further details we refer to the reviews [1, 3].

Finally, more information on PCA can be found in the book by Joliffe [6]. For dimensionality reduction in general see [7], which also covers PCA.

REFERENCES

[1] Rodrigo Benenson, Mohamed Omran, Jan Hosang, and Bernt Schiele. "Ten years of pedestrian detection, what have we learned?" In: *European Conference on Computer Vision*. Springer. 2014, pp. 613–627.

[2] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection." In: *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*. Vol. 1. IEEE. 2005, pp. 886–893.

[3] Piotr Dollar, Christian Wojek, Bernt Schiele, and Pietro Perona. "Pedestrian detection: An evaluation of the state of the art." In: *IEEE transactions on pattern analysis and machine intelligence* 34.4 (2012), pp. 743–761.

[4] Markus Enzweiler and Dariu M Gavrila. "Monocular pedestrian detection: Survey and experiments." In: *IEEE transactions on pattern analysis and machine intelligence* 31.12 (2009), pp. 2179–2195.

[5] Harold Hotelling. "Analysis of a complex of statistical variables into principal components." In: *Journal of Educational Psychology* 24.6 (1933), p. 417.

[6] Ian T Jolliffe. “Principal component analysis and factor analysis.” In: *Principal component analysis* (2002), pp. 150–166.

[7] John A Lee and Michel Verleysen. *Nonlinear dimensionality reduction*. Springer Science & Business Media, 2007.

[8] Stefan Munder and Dariu M Gavrila. “An experimental study on pedestrian classification.” In: *IEEE transactions on pattern analysis and machine intelligence* 28.11 (2006), pp. 1863–1868.

[9] Karl Pearson. “On lines and planes of closest fit to a system of points in space.” In: *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science* 6.2 (1901), pp. 559–571.