REPORT: PATTERN RECOGNITION

Leif Van Holland

University of Bonn

1. INTRODUCTION

In the last decades, unprecedented amounts of data are collected in various places throughout the analog and digital world. The amount of information renders manual interpretation unfeasible if not impossible. Therefore the topic of *pattern recognition* plays an increasingly important role in modern technology. [1]

In the university lecture "Pattern Recognition," basic concepts of the field of research are presented. During the lecture, we worked on three projects that touched on some recurring problems.

Following we will represent our results chronologically, starting with a "warm-up" project about elementary model fitting. Next, the second project involved more approaches for regression as well as a simple algorithm for classification. Finally, in PROJECT 3, we shed light on different clustering techniques and ultimately revisited linear classifiers equipped with methods to handle non-linearly separable data.

2. REGRESSION

The term *regression* refers to a statistical method of estimating relationships between variables. Given a set of data $D = \{(x_i, y_i)\}_{i=1}^N$, where $x_i \in \mathbb{R}^n$ and $y_i \in \mathbb{R}$, the goal is to find a set of parameters $\theta \in \mathbb{R}^k$ of a given model $y : \mathbb{R}^N \times \mathbb{R}^k \to \mathbb{R}$, such that *$y$ predicts* the values y_i based on x_i as an input, with minimal error. In other words, we look for parameter values $\hat{\theta}$ such that

$$\hat{\theta} = \underset{\theta}{\operatorname{argmin}}\, E(\theta). \tag{1}$$

E denotes the *objective function* that depends on the problem at hand. Most times however, E measures a distance between the target output y_i and the model prediction $y(x_i, \hat{\theta})$.

2.1. Maximum Likelihood Estimation (MLE)

If we choose a probability distribution $\mathcal{N}$ based on parameters θ as our model, i.e. we suspect our data to be realizations of random variables $X_i \sim \mathcal{N}[\theta]$, we can calculate the probability of any possible realization x_i of X_i depending on θ. We define the *likelihood* L that parameters θ are responsible for generating the set of data D as a a function

$$L(\theta, D) := P(\theta|D) = \frac{P(D|\theta)P(\theta)}{P(D)}.$$

If we assume that the random variables X_i are i.i.d., we can further simplify that definition and rewrite the joint density $P(D)$ as a product and thus get $L(\theta, D) = \prod_{x_i \in D} P(\theta|x_i)$. For numerical stability, one often considers the *log-likelihood* function

$$\mathcal{L}(\theta, D) = \ln L(\theta, D) = \sum_{x_i \in D} \ln P(\theta|x_i).$$

To find the parameters θ that are most likely to have generated D, we look for the *maximum likelihood estimate*

$$\hat{\theta} = \underset{\theta}{\operatorname{argmax}}\, L(\theta, D) = \underset{\theta}{\operatorname{argmin}} -L(\theta, D)$$

2.2. Normal and Weibull Distribution

In the first project we looked at two distributions applied to 1-dimensional data. The *1D normal distribution* $N[\mu, \sigma^2]$ is depending on two parameters $\mu, \sigma^2 \in \mathbb{R}$ with the density function

$$f(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{\frac{1}{2}(\frac{x-\mu}{\sigma})^2}.$$

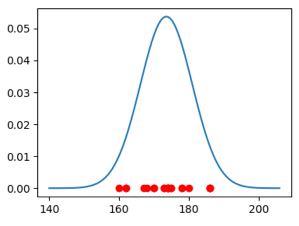
Using the method of maximum likelihood estimation, we can specify the optimal choices $\hat{\mu}$ and $\hat{\sigma}^2$ for both parameters directly, which coincide with the *sample mean* and *population variance* of the given set of data respectively:

$$\hat{\mu} = \frac{1}{n}\sum_{i=1}^n x_i \text{ and } \hat{\sigma}^2 = \frac{1}{n}\sum_{i=1}^n (x_i - \hat{\mu})^2$$

We applied this estimation to the body sizes of a set of data containing weights and heights of several attendees of an earlier installment of the "Pattern Recognition" lecture. The result can be seen in Fig. 1(a).

The *Weibull distribution* on the other hand uses parameters $\kappa, \alpha \in \mathbb{R}$ and is defined by the density

$$f(x) = \frac{\kappa}{\alpha}\left(\frac{x}{\alpha}\right)^{\kappa-1} e^{-(\frac{x}{\alpha})^\kappa}.$$


(a)

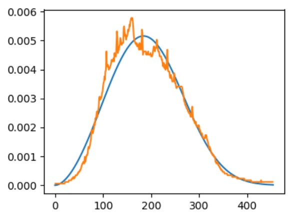

(b)

Fig. 1. (a) Resulting PDF of a normal distribution maximizing the likelihood w.r.t. the body sizes on the weight-height data; (b) MLE-fitted PDF of a Weibull distribution on Google Trends data.

Estimating the parameters here is more involved, as there is no closed-form solution for the estimates $\hat{\kappa}$ and $\hat{\alpha}$. A maximum of the log-likelihood function

$$L(\alpha,\kappa) = N(\log\kappa - \kappa\log\alpha) + (\kappa-1)\sum_i \log d_i - \sum_i \left(\frac{d_i}{\alpha}\right)^{\kappa}$$

has therefore to be found numerically and we used *Newton's method* initialized with $\kappa = 1$ and $\alpha = 1$. To optimize the calculation time we rewrote the log-likelihood using a histogram $h(x_j) = h_j$ for all occurring values x_j in the data, which reduces the number of elements that the sums in L are iterating over. $L(\alpha,\kappa)$ then equals the term

$$N(\log\kappa - \kappa\log\alpha) + (\kappa-1)\sum_j x_j \log h_j - \sum_j \left(\frac{x_j \cdot h_j}{\alpha}\right)^{\kappa}.$$

We used the described method on Google Trends data about global interest in the search term *"myspace,"* measured every week between January 1, 2003 and March 16, 2012. Fig. 1(b) shows that assuming a Weibull distribution reasonably describes the overall trend, although one can clearly observe a deviation from the actual graph between week 100 and 200.

2.3. Least Squares

A different but also common technique is to formulate the problem as a distance $\|Xw - y\|_2$ between a design matrix $X \in \mathbb{R}^{N\times d}$ times the solution vector $w \in \mathbb{R}^d$ and a target vector $y \in \mathbb{N}$ where

$$X = \begin{pmatrix} x_1^T \\ \vdots \\ x_N^T \end{pmatrix} \quad y = \begin{pmatrix} y_1 \\ \vdots \\ y_N \end{pmatrix}.$$

This distance is to be minimized, so as an objective function we choose

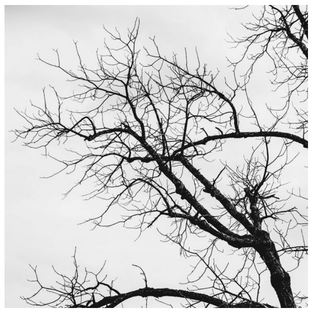
$$E(w) = \|Xw - y\|_2^2 \tag{2}$$

whereby the norm is often squared for simplicity.
If X has full rank, the problem becomes convex and therefore has a unique solution. It can be shown that this solution is determined by the product

$$w = X^+ y \text{ where } X^+ := (X^T X)^{-1} X^T$$

denotes the *pseudoinverse* of X. [2]

2.4. Excursion: Fractal Dimensions


(a) "tree" input image
$D = 1.827$

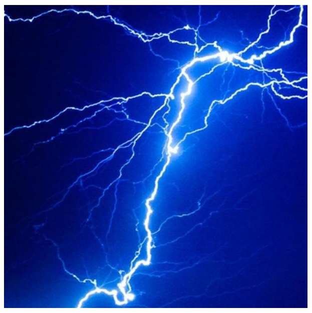

(b) "lightning" input image
$D = 1.595$

Fig. 2. Images of which we estimated the fractal dimension using the box-counting method.

In TASK 1.5 we took a detour and estimated the *fractal dimension* of objects in 2D images. Fractals are objects that exhibit a certain *self-similarity* regardless of scale. Loosely speaking, the fractal dimension is a measure of change in complexity with respect to change in scale. One idea to define this mathematically is to count, for a given scale s, how often we need the pattern/fractal of scale 1 to reproduce it. Say we counted that the object contains n versions of itself at scale s, the fractal dimension D amounts to

$$D = -\frac{\log n}{\log s}.$$

The method of *box counting* is putting that abstract definition in concrete terms. For the discretized 2D images we were given (Fig. 2) we first generated a binarized version of these images. For a set of scaling factors

$$S = \left\{ \frac{1}{2^i} \middle| i \in \{1, ..., 9\} \right\}$$

the images were split into 2^{2i} equally sized boxes at every scale factor s_i. Next we determine n_i by counting how many boxes at scale s_i contain at least one foreground pixel. The box-counting dimension D of the image is then equal to the slope of the line

$$D \cdot \log \frac{1}{s_i} + b = D \cdot i + b = \log n_i.$$

We can estimate the parameters D and b via least squares using the design matrix $X = \begin{pmatrix} 1 & \dots & 9 \\ 1 & \dots & 1 \end{pmatrix}^T$ and target vector

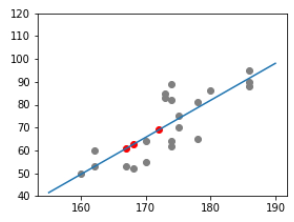
$y = (n_1, ..., n_9)^T$. The estimated dimensions for the images in Fig. 2 were: (a) $D = 1.827$ and (b) 1.595. This can be interpreted as (a) having more similarity with a plane whereas (b) being closer to being a line-like shape. [3, 4]

2.5. Least Squares Polynomials

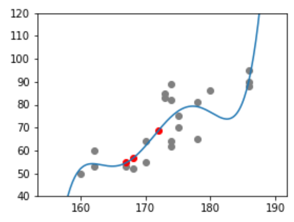
Fitting a d-dimensional polynomial $p(x) = \sum_{j=1}^{d} w_j x$ to a two-dimensional set of data can also be realized solving a linear system of equations. Recall the definition $D = \{(x_i, y_i)\}_{i=1}^{N}$, albeit this time $(x_i, y_i) \in \mathbb{R}^2$. Evidently, the objective function to be minimized is

$$E(w) = \sum_{i=1}^{N} \|p(x_i) - y_i\|_2^2. \tag{3}$$

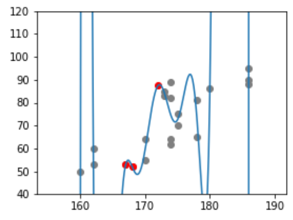
In fact, we can rewrite this as a matrix product employing the so-called *Vandermonde matrix* $X \in \mathbb{R}^{N \times d}$ with $x_{ij} = x_i^{j-1}$ as the design matrix and the coefficients $w = (w_1, ...w_d)^T$ of the polynomial as the parameters to be determined. The target vector is chosen as in Section 2.3. Equation (3) then is equivalent to the least squares objective function (2).
In TASK 2.1 we applied this idea to the aforementioned weight-height data and fitted polynomials for $d \in \{1, 5, 10\}$. The resulting fits can then be used to predict the missing weight values in the data. See Fig. 3 for details.



(a) $d = 1$



(b) $d = 5$



(c) $d = 10$

Fig. 3. Polynomials of degrees 1 (a), 5 (b) and 10 (c) fitted to the data containing height and weight of subjects. Undisclosed weight data of three subjects are estimated using the model output, marked as red.

The approach seen above was revisited in TASK 3.5 to gain awareness of numerical instabilities that are caused by poorly conditioned matrices. Several methods can improve the stability, including (i) the selection of a robust algorithm to solve systems of equations and (ii) normalizing the data to zero mean and unit standard deviation. We used both (i) and (ii) to improve the results in TASK 2.1.

2.6. Probabilistic Model Fitting

In the beginning of Section 2, we defined the model as a function so that the equality $y_i = f(x_i) + \epsilon_i$ holds for all data points $(x_i, y_i) \in D$ (ϵ_i denotes a noise term). Taking the weight-height data as an example, this assumption might be unreasonable as there are multiple subjects of a body height of 186 cm that have different body weights. In the lecture we looked at a reformulation of the model as a conditional expectation

$$f(x) = \mathbb{E}[y|x] = \int y \, p(y|x) \, dy. \tag{4}$$

In order to compute the target model $p(y|x) = \frac{p(x,y)}{p(x)}$ we first have to determine the joint probability $p(x, y)$ by guessing a model for the data and finding the parameters for an optimal fit. Next, we can either guess a model for the distribution $p(x)$ or by integrating $p(x, y)$ over all possible values of y: $p(x) = \int p(x, y) \, dy$. Consequently, we can now calculate the expected value in (4).

The goal of TASK 2.3 was to utilize this method on the weight-height data. As a model for the joint probability, a bi-variate Gaussian was used. Following the result in Section 2.2, the mean vector μ and the covariance matrix Σ of the distribution are based on 1D Gaussians $N[\mu_h, \sigma_h^2]$ for the height values and $N[\mu_w, \sigma_w^2]$ for the weight values, respectively. We get

$$\mu = \begin{pmatrix} \mu_h \\ \mu_w \end{pmatrix} \text{ and } \Sigma = \begin{pmatrix} \sigma_h^2 & \rho \, \sigma_h \sigma_w \\ \rho \, \sigma_h \sigma_w & \sigma_w^2 \end{pmatrix}$$

where $\rho := \frac{\text{Cov}(h,w)}{\sigma_h \sigma_w}$ is the *correlation coefficient* of h and w. In the lecture it was shown that in the case of a bi-variate Gaussian, (4) simplifies to

$$\mathbb{E}[y|x] = \mu_w + \rho \frac{\sigma_w}{\sigma_h}(x - \mu_h).$$

This led to our prediction result seen in Fig. 4(a). [3]

2.7. Bayesian Regression

To go even further, we can assume a probability distribution $p(\theta)$ over the model parameters θ. This idea leads to the *maximum a posteriori estimate* $\theta_{MAP} = \text{argmax}_\theta \, p(\theta|D)$, based on the posterior probability $p(\theta|D)$ given a set of data D.

Again using a linear model to fit a 5th-degree polynomial to the weight-height data, as we did in Section 2.5, one can show that an optimal choice $\hat{w}$ for the coefficient vector is

$$\hat{w} = (X^T X + \frac{\sigma^2}{\sigma_0^2})^{-1} X^T y \tag{5}$$

if we assume a Gaussian prior $p(w) \sim N(w|\mu_0, \sigma_0^2 I)$. [3]

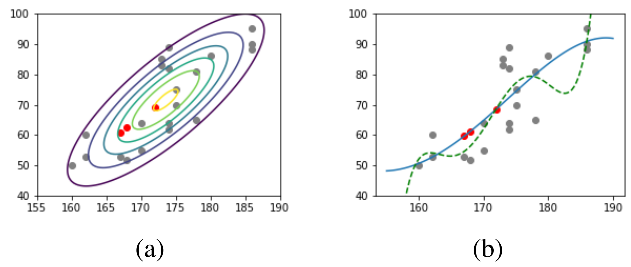


Fig. 4. (a) Bi-variate Gaussian fitted to the weight-height data. (b) 5th-degree polynomial fitted to the weight-height data using Bayesian regression (blue curve) compared to the least squares regression from Fig. 3 (dashed green curve). Predicted values marked as red.

We compared the results with the polynomial fit we computed in TASK 2.1 (see Section 2.5). One can observe in Fig. 4(b) that the Bayesian regression (5) provides a smoother fit throughout the data. In contrast, for example around 183 cm, the curve estimated by least squares drops because no data points are providing contrary information. In our opinion, the Bayes estimate is more reasonable, considering we would expect the relation between height and weight to be monotonically increasing instead of exhibiting multiple modes around certain height values.

2.8. Excursion: Boolean Functions

For TASK 2.4, we applied least squares to the concept of *cellular automata*; a concept that is extensively discussed by Wolfram in [5]. In short, a one-dimensional cellular automaton consists of infinitely many cells $x_i \in \{-1, +1\}$ and a rule $f(x_{i-1}, x_i, x_{i+1})$ to update the state of each cell in the next epoch $t + 1$, based on the value of the cell and its two neighbors in epoch t. This definition implies that there is a total of $2^{2^3} = 256$ different rules that can be defined and Wolfram enumerated them by their byte representation.

As every rule provides an output for any of the eight possible input configurations, we can represent every rule as a matrix $X \in \mathbb{R}^{8\times3}$ and a target vector $y \in \mathbb{R}^8$. Firstly, we tried to run least squares on this setting, i.e. computing $w \in \mathbb{R}^3$ as a minimum of (2) like we have seen in Section 2.3. w gives us coefficients of a linear combination of columns in X to reconstruct the target vector y. No perfect reconstruction was possible for the tested rule 110 and rule 126.

Using the notion of *Boolean Fourier series expansion* (see e.g. [6] for details), we can rewrite any rule as an inner product $f(x) = w^T\varphi$. The vector φ is determined for every input combination (x_1, x_2, x_3) as

$$\varphi(x_1, x_2, x_3) = (1,\, x_1,\, x_2,\, x_3,\, x_1x_2,\, x_1x_3,\, x_2x_3,\, x_1x_2x_3)$$

and we used this transformation on every row of X to get a matrix $\Phi \in \mathbb{R}^{8\times8}$. This time, we minimized $\|\Phi w - y\|_2$ for x and our results show that we can perfectly reconstruct rule 110 and rule 126 from $(\Phi \cdot w)$. That is a hint to our observation in Section 6, where we used transformations into higher dimensions to solve a non-linearly separable problem.

3. CLASSIFICATION

Up until now, we tried to predict a real-valued target $y = f(x)$ given our model f. The idea of classification is to group data points into n classes $\Omega_1, ..., \Omega_n$ and therefore the model f is to predict the class membership expressed as a so-called *label*, e.g. $y \in \{1, ..., n\}$. Often the data is split into a training set D_{train} and a validation set D_{test} to measure the performance of a certain classification algorithm. It may only use the known labels for data points in D_{train} for training. A good performance will be achieved if the algorithm predicts the labels in D_{test} reasonably well.

3.1. k-Nearest-Neighbors

A simple approach to tackle the classification problem is to let neighboring training points vote for the class of an unseen data point. More explicitly, for a new data point $x \in \mathbb{R}^d$ we determine k points $x_{i_1}, ..., x_{i_k}$ so that $\|x - x_{i_j}\|_2 \leq \|x - x_{i_l}\|_2$ for all $j \leq k < l$. The resulting prediction $f(x)$ is the most frequent label in $\{y_{i_1}, ..., y_{i_k}\}$. As seen in the lecture, the choice of k influences the smoothness of the class boundary the model is implying (a higher k causing the boundary to be smoother). [3]

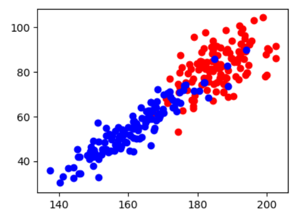
In TASK 2.4 we were given a set of data points labeled as members of one of two classes. The data was split into training and test sets as described above. Running the k-Nearest-Neighbors algorithm for $k \in \{1, 3, 5\}$ yielded the results shown in Fig. 5.

We measured the accuracy as the percentage of correctly predicted labels in D_{test} and, as expected, the accuracy increases for bigger neighborhoods. We suspect that for $k > 5$ the algorithm will not generate much higher accuracy values, as some points in the data are surrounded by points of the opposite label. Thus, further smoothing the class boundary will not improve the classification of these outliers.

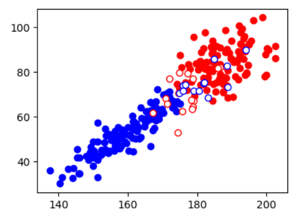
Our naïve implementation of the algorithm determines the neighborhood of an unseen point by measuring the distance to all training points and then taking the k smallest distances. This approach took about 0.02 seconds for determining the distances needed (independently of k).

3.2. k-d Trees

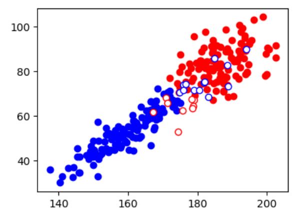
To improve the naïve implementation, we resorted to an efficient data structure called k-dimensional tree, or *k-d-tree* for short, that was first introduced by Bently in [7]. Its idea is to partition the space using $(k-1)$-dimensional axis-aligned hyperplanes, that are represented by nodes in the tree. One can



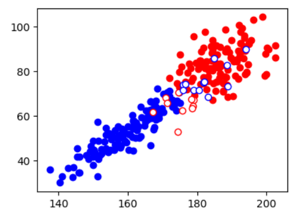
(a)



(b) $k = 1$ (77% acc.)



(c) $k = 3$ (82% acc.)



(d) $k = 5$ (83% acc.)

Fig. 5. Results of the k-Nearest-Neighbors algorithm on a (a) set of data for (b) $k = 1$, (c) $k = 3$, and (d) $k = 5$. Filled circles denote a correct prediction of the label. Unfilled circle are incorrectly predicted points (the color denotes the true label).

show that this leads to an average run time of $O(\log N)$ for determining the nearest neighbor of an unseen query point. See [3] or [7] for details.

In fact, there are several possibilities for building the k-d tree from a given set of points. For TASK 2.5 we were to employ two different methods for selecting the splitting dimension of the hyperplane, (i) alternating between x- and y-axis, or (ii) splitting along dimension with higher variance, and two different methods to determining the split point by either taking (a) the median of the data, or (b) the (arithmetic) mean of the data.

The choice of these methods will have an impact on the number of search steps needed to find the nearest neighbor.

Replacing our previous method from Section 3.1 with a 2-d tree, we measured the run times seen in Table 1. Evidently, the new data structure is an overall improvement compared to the average runtime of 0.02 sec measured for our naïve approach. Selecting the mean value together with cycling through the split axes seems to be the preferred choice for the given data.

	(a) median	(b) mean
(i) cycle	0.0142	0.0099
(ii) max. var	0.0136	0.0105

Table 1. Average runtimes in seconds for the combinations of building methods of a k-d-tree.

4. CLUSTERING

The type of data introduced in chapter 3 implied that the training set contains label information for every data point. To add another constraint, we drop this assumption, and even though we have no information about any class membership, we suppose there are k classes, and every point is a member of one of these. The goal of *clustering* techniques is to determine a "best guess" for a membership function f. The presumed classification can then be used to gain further information about the set of data.

4.1. k-Means Clustering

A popular algorithm used for these kinds of problems is the *k-means clustering* algorithm (k being the number of distinct clusters to find). For data points $D = \{x_i\}_{i=1}^{N}$, the goal is to determine k centroids $\mu_1, ..., \mu_k$, such that the objective function

$$E(k) = \sum_{j=1}^{k} \sum_{x i \in C_j} \|x_i - \mu_j\|_2 \tag{6}$$

is minimized. The membership of any given point x_i is determined by the closest centroid, i.e.

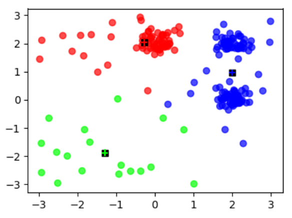
$$C_j := \{x_i \mid \|x_i - \mu_j\|_2 < \|x_i - \mu_k\|_2 \text{ for } k \neq j\}.$$

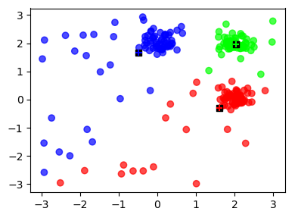
According to Hartigan in [8], the k-means algorithm consists of three components:

1. A rule for initializing centroid positions,
2. a movement rule determining the reassignment of points to a different cluster, and
3. an update rule for the centroid positions.

The algorithm runs iteratively, applying rules 2. and 3. until some chosen convergence criteria are met. In the lecture, we have seen three approaches that define these rules differently; (a) Lloyd's algorithm, (b) Hartigan's algorithm and (c) MacQueen's algorithm. One can show that none of the above are guaranteed to find the global minimum of (6), though they will minimize it locally. [3]

In task 3.1 we ran the three flavors of k-means on a given set of data for $k = 3$ and compared the variance of the results as well as the runtime. Fig. 6 shows example clusterings from one of several runs of the algorithms. We observed that (a) was the fastest implementation taking 0.003 sec, followed by (c) with 0.005 sec and (b) being much slower with 0.428 sec on average. On the other hand, (b) seemed to produce much less variation in its results, whereas (a) and (c) displayed multiple local optima in relatively equal frequency.


(a) Lloyd


(b) Hartigan

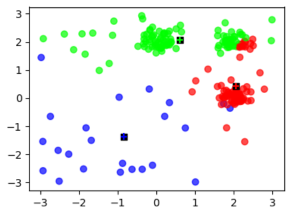

(c) MacQueen

Fig. 6. Exemplary clustering results of three different k-means approaches for $k = 3$ clusters. The colors indicate the membership of every point, plus symbols denote the centroids.

4.2. Spectral Clustering

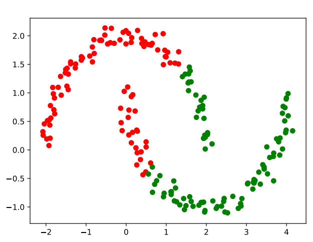
The k-means clustering algorithm will only yield acceptable results for data that cluster in spheres around centroids, whereas it fails for differently aligned data. Therefore, methods of transforming the input space were developed and the *spectral clustering* as a generalization of k-means may be one step towards the issue. [9]

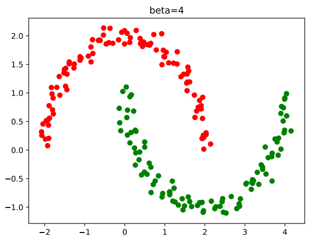
To simplify the definitions, we assume that the goal is to find two clusters in an unlabeled set of data D. First of all, we create a fully-connected weighted graph $G = (V, E)$, where the vertices $x_i \in V$ are representing the data points and the edge weights $w(i, j) \geq 0$ are expressing the *similarity* between vertices x_i and x_j. To determine two clusters in that graph is to find a cut that is partitioning the vertices into two disjoint sets A and B. To measure the (dis)similarity for a given partition, Shi et al. proposed the *normalized cut* which sums the weights of cut edges and normalizes it to be robust against outliers. Finding the minimal (non-zero) normalized cut yields a desired solution to the clustering problem, as it minimizes the similarity between the classes while maintaining a high within-class similarity. [10]

Based on this, we define the *Laplacian matrix* $L = D - W$ as the difference of a diagonal matrix D with entries $d_{ii} := \sum_j w(i, j)$ and W, which is containing the edge weights $w(i, j)$. One can further show that finding the minimal normalized cut is approximated by the eigenvalue problem

$$Ly = \lambda y. \tag{7}$$

For our problem, solutions $y = (y_1, ..., y_N)$ of (7) can be interpreted as partitions where x_i belongs to A if $y_1 > 0$.


(a)



(b)

Fig. 7. A non-linearly separable set of data separated by k-means for $k = 2$ (a) and by spectral clustering (b). The colors indicate the membership of each data point to one of the two classes.

Otherwise it belongs to B. In fact, the eigenvector of the smallest eigenvalue represents the trivial cut $V = A \dot{\cup} B$. Therefore we are interested in the eigenvector of the second-smallest eigenvalue, sometimes called *Fiedler vector*. [10]

In TASK 3.2, we applied this method to a constructed set of data that implies an apparent separation into two classes. Choosing the similarity $w(i, j) = \exp(-\beta \|x_i - x_j\|^2)$, we found that for $\beta = 4$, the spectral clustering method yields the apparent separation, while evidently k-means clustering (for $k = 2$) cannot come close to this result (see Fig. 7).

5. DIMENSIONALITY REDUCTION

The lecture briefly discussed the so-called *curse of dimensionality*, stating that for randomly distributed points in high-dimensional spaces, the distances are likely to be almost equal. This fact renders clustering algorithms depending on distances, like the ones we have seen in Section 3, less useful.

A simple idea is to reduce the dimension of the input space by determining an appropriate projection onto a lower-dimensional space. Finding such a projection is in itself a problem that can be solved with different objectives in mind.

5.1. Principal Component Analysis (PCA)

Intuitively, one may consider the variance of the data to determine the most important dimensions. The *principal components* of D are defined as orthogonal directions of highest variance. These can be obtained by calculating an eigen-decomposition $\Sigma = V \Lambda V^{-1}$ of the data covariance matrix Σ. As it can be shown that Σ is positive semi-definite, such a decomposition always exists. In fact, for such matrices, the eigenvectors of different eigenvalues are orthogonal and can be ordered because all eigenvalues are real and positive. Thereby an eigen-decomposition of Σ naturally yields principal components. To reduce the dimension to k, we then project the data onto the eigenvectors of the k highest eigenvalues. [11]

We were given a labeled 500-dimensional set of data in

TASK 3.3 and applied this method to reduce the dimension to $k \in \{2, 3\}$. The visualized projections in Fig. 8(a) and 8(c) indeed reveal clusters. We will now compare this to another method and see if we can improve the results.

5.2. Linear Discriminant Analysis (LDA)

As the data might be labeled, we can incorporate this information into the process. Suppose we are given data from k classes, we calculate the mean μ_j and covariances Σ_j inside each class as well as the overall mean μ. Next, we define the within-class scatter matrix S_W and the between-class scatter matrix S_B:

$$S_W = \sum_{j=1}^{k} \Sigma_j \qquad S_B = \sum_{j=1}^{k} (\mu_j - \mu)(\mu_j - \mu)^T.$$

To find a suitable projection W, in the lecture we showed one has to solve the following constrained minimization problem

$$\min_W W^T S_B W \text{ s.t. } W^T S_W W = I. \tag{8}$$

Fisher proved in [12] that a matrix W, consisting of eigenvectors of the k largest eigenvalues of $S_W^{-1} S_B$, is a solution of (8).

Applying this to the 500-dimensional data from Section 5.1, we obtain the projections visualized in Fig. 8(b) and 8(d). Again, we can observe that points from the same class are spatial close. Because the LDA-projection is using the labels to minimize the within-class scatter, the points are close even along the third axis, unlike when projected via PCA.

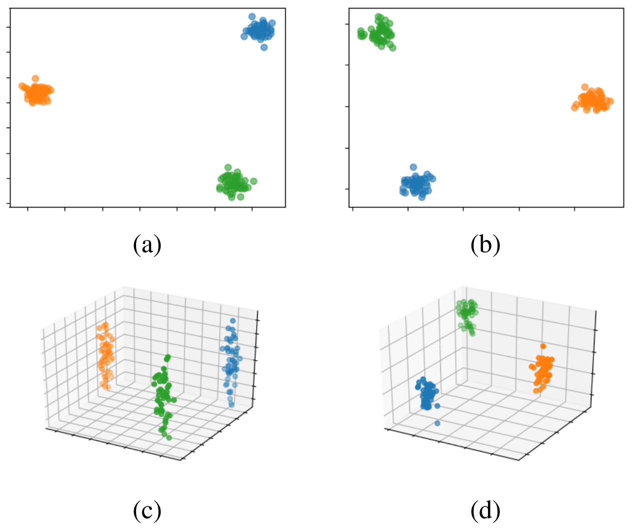


Fig. 8. Projections of a labeled 500-dimensional set of data into 2D (a, b) and 3D (c, d) using PCA (a, c) and LDA (b, d). Colors denote the class membership of the points.

6. EXTENDING LINEAR CLASSIFIERS

Lastly, we investigated the capabilities of traditionally linear classifiers, namely *perceptrons* and *support vector machines* extended with non-linear functions, and tested their performance on the XOR-problem. We now briefly discuss our findings.

6.1. Non-Monotonous Neurons

A perceptron with parameters w and θ calculates the projection $y(x) = f(w^T x - \theta)$ for a given activation function f. If f is chosen to be monotonous, y can only solve linearly separable problems, so for TASK 3.4 a non-monotonous activation function was selected, resulting in

$$y(x) = 2 \exp\left(-\frac{1}{2}(w^T x - \theta)^2\right) - 1.$$

To find the optimal parameters for the given data, we performed gradient descend over the loss function

$$E = \frac{1}{2} \sum_{i=1}^{N} (y(x_i) - y_i)^2.$$

This yielded a perfect classification of the XOR-problem as seen in Fig. 9(b). For a comparison, in Fig. 9(a) we visualized the result using the monotonous activation function $f(x) = \tanh(x)$ with which it is impossible to classify the data correctly.

6.2. Support Vector Machines (SVM)

A different linear classifier is the so-called *support vector machine*. Its goal is to maximize the margin between two linearly separable classes and the separating hyperplane. Though this optimization seems involved, we can consider the dual constrained optimization problem

$$\operatorname*{argmax}_{\mu} -\frac{1}{2}\mu^T G \mu + 1^T \mu \text{ s.t. } y^T \mu = 0 \text{ and } \mu \geq 0.$$

Its solution then yields the parameters for the linear classification $w = \sum_{i=1}^{N} \mu_i y_i x_i$. The calculation simplifies drastically when following the ideas in [13], as seen in the lecture. [3]

To "non-linearize" this approach we can utilize the *kernel trick*. The idea behind it is to project the data into a higher-dimensional space where it becomes linearly separable using a non-linear transformation φ. Finding (and potentially also computing) φ is generally a very difficult problem, but Mercer first proved in [14] that it suffices to find a positive semidefinite *kernel function* $k(a, b)$ in the input space as it implies the existence of a transformation φ such that $k(a, b) = \varphi(a)^T \varphi(b)$. [15]

As seen in the lecture, the kernel trick can be easily applied to support vector machines. Using a quadratic kernel $k(a, b) = (a^T b + c)^2$ we were in fact able to separate the XOR-problem using the modified SVM (see Fig. 9(c)). [3]

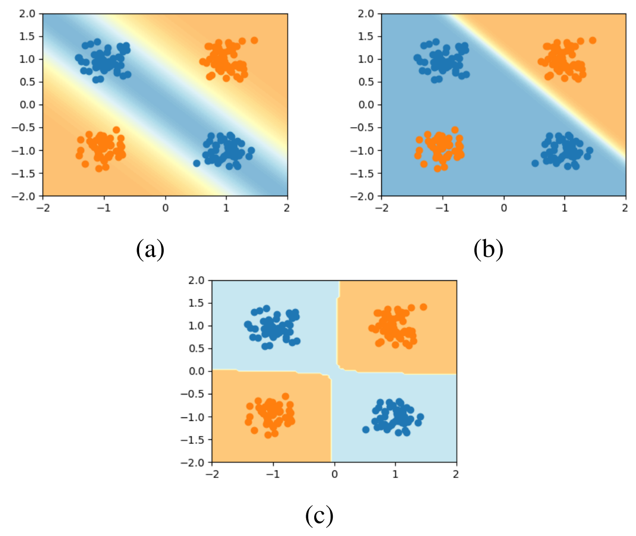


Fig. 9. Outputs on an XOR-problem of classifiers with (a) non-linear activation function, (b) linear activation function, and (c) of a modified SVM.

7. CONCLUSION

In this report, we presented our solutions for the three different projects we worked on in the course of the lecture. In PROJECT 1 one we have seen how to fit simple and more complex probability distributions to data using maximum likelihood estimation. We also applied linear regression to estimate the fractal dimension of given images.

PROJECT 2 was about different methods to predict missing values, namely via least squares polynomials, conditional expectation, and Bayesian regression. We concluded that, though the computation is more involved, the Bayesian approach yielded a superior model compared to the other methods. Next, we looked at the nearest-neighbor classifier and improved the runtime using k-d-trees.

Finally in PROJECT 3, we applied two clustering algorithms: k-means and spectral clustering. As they are prone to deliver suboptimal results for high-dimensional input spaces, we used principal component analysis and Fisher's linear discriminant analysis for dimensionality reduction. Furthermore, we took two classic linear classifiers, (i) the perceptron and (ii) the support vector machine, and implemented approaches to be capable of handling non-linearly separable data. For (i) we used a non-linear activation function and for (ii) we utilized the kernel trick to accomplish the goal. Both techniques were then able to solve the XOR-problem.

8. REFERENCES

[1] Richard L Villars, Carl W Olofson, and Matthew Eastwood, "Big data: What it is and why you should care," *White Paper, IDC*, vol. 14, 2011.

[2] Lloyd N Trefethen and David Bau III, *Numerical linear algebra*, vol. 50, Siam, 1997.

[3] Christian Bauckhage, "lectures on pattern recognition," 2017, notes can be found at http://www.researchgate.net/project/lectures-on-pattern-recognition, accessed on 2018-02-20.

[4] Kenneth Falconer, *Fractal geometry: mathematical foundations and applications*, John Wiley & Sons, 2004.

[5] Stephen Wolfram, "Statistical mechanics of cellular automata," *Reviews of modern physics*, vol. 55, no. 3, pp. 601, 1983.

[6] Ryan O'Donnell, *Analysis of boolean functions*, Cambridge University Press, 2014.

[7] Jon Louis Bentley, "Multidimensional binary search trees used for associative searching," *Communications of the ACM*, vol. 18, no. 9, pp. 509–517, 1975.

[8] John A Hartigan, *Clustering algorithms*, John Wiley & Sons, 1975.

[9] Inderjit S Dhillon, Yuqiang Guan, and Brian Kulis, "Kernel k-means: spectral clustering and normalized cuts," *Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pp. 551–556, 2004.

[10] Jianbo Shi and Jitendra Malik, "Normalized cuts and image segmentation," *IEEE Transactions on pattern analysis and machine intelligence*, vol. 22, no. 8, pp. 888–905, 2000.

[11] Hervé Abdi and Lynne J Williams, "Principal component analysis," *Wiley interdisciplinary reviews: computational statistics*, vol. 2, no. 4, pp. 433–459, 2010.

[12] Ronald A Fisher, "The statistical utilization of multiple measurements," *Annals of Human Genetics*, vol. 8, no. 4, pp. 376–386, 1938.

[13] Philip Wolfe, "The simplex method for quadratic programming," *Econometrica: Journal of the Econometric Society*, pp. 382–398, 1959.

[14] BA J Mercer, "Xvi. functions of positive and negative type, and their connection the theory of integral equations," *Phil. Trans. R. Soc. Lond. A*, vol. 209, no. 441-458, pp. 415–446, 1909.

[15] Christian Bauckhage, "Lecture notes on the kernel trick (i)," 2013.