PATTERN RECOGNITION REPORT

Maximilian Thiessen

University of Bonn

ABSTRACT

This report describes our approaches to solve the three projects for the lecture series Pattern Recognition in the winter semester 2017/18 at the University of Bonn by Prof. Bauckhage. The tasks included fitting models to data, implementing the nearest neighbour algorithm with kD-Trees, various clustering approaches, algorithms for dimensionality reduction, an L2-SVM and applications to cellular automata and the approximation of the fractal dimension of images.

1. INTRODUCTION

In recent decades machine learning become very popular by the availability of huge amounts of data and fast processors [1]. The lecture series Pattern Recognition was an introduction to various statistical and numerical techniques to analyse data, which can be regarded as the backbone of almost all machine learning algorithms. During the lecture period, we had to work on three projects, which involved understanding the theoretical background of the tasks, choosing reasonable approaches, implementing the needed algorithms and visualizing the results. In the first project, we had to fit a Gaussian and a Weibull distribution to data, gain some intuition with unit circles and approximate the fractal dimension of images. The second project dealt with different approaches to missing value prediction, the boolean Fourier transform and the nearest neighbour algorithm with kD-trees. Finally, in the last project, we implemented different clustering and dimensionality reduction methods, got some insights into perceptrons, trained an SVM and learnt about some typical numerical issues.

Our report starts with the basics of the least squares principle, as we recurrently used it for many different tasks, and continues by giving the main ideas of our results for each individual task.

2. LEAST SQUARES

Least squares is a standard technique in statistical learning [2]. The main idea is to fit a function to data points, which minimizes the squared distance between the function and the data points. In the simplest setting, we want to fit a hyperplane to given data. If we treat the data as pairs of (continuous) labels and values, arrange the values as rows of a so-called *design* matrix $\mathbf{X}$ and regard all labels as a vector $\mathbf{y}$, the hyperplane, which minimizes the squared distances to those points is given by:

$$\arg\min_{\mathbf{w}} ||\mathbf{X}\mathbf{w} - \mathbf{y}||^2.$$

The analytical solution to this problem is

$$\mathbf{w} = (\mathbf{X}^T\mathbf{X})^{-1}\mathbf{X}^T\mathbf{y}.$$

We can generalize this approach to non-linear functions, like polynomials, as long as the function is linear in the parameters $\mathbf{w}$ [2].

3. DATA FITTING WITH THE MAXIMUM LIKELIHOOD PRINCIPLE

Maximum likelihood is a basic principle for finding the parameters of a statistical model given some data. Informally this method finds the most *likely* parameters of a chosen distribution, which produced the data. Expressed mathematically:

$$\arg\max_{\mathbf{p}} P(X|\mathcal{D}_{\mathbf{p}})$$

with given data X, an assumed probability distribution $\mathcal{D}$ with to be determined parameters $\mathbf{p}$. It is part of the more general maximum a-posteriori framework, which yields the most likely distribution given the data (i.e. $P(\mathcal{D}_{\mathbf{p}}|X)$). Indeed if we assume a uniform prior on all possible parameters both approaches result in the same parameters.

As this is a maximization problem it is useful to inspect the parameters where the derivative of the *likelihood*-function $L(\mathbf{p}) = P(X|\mathcal{D}_{\mathbf{p}})$ is zero. Furthermore, the data X is often given as a set of *independent and identically distributed* random variables $x_1, \ldots, x_n$ according to the distribution $\mathcal{D}_{\mathbf{p}}$. In this situation the likelihood becomes

$$L(\mathbf{p}) = P(X|\mathcal{D}_{\mathbf{p}}) = \prod_{i=1}^{n} P(x_i|\mathcal{D}_{\mathbf{p}}).$$

As products are not convenient to differentiate we often use the *log-likelihood* $\mathcal{L}(\mathbf{p}) = logL(\mathbf{p})$ which becomes $\sum_{i=1}^{n} logP(x_i|\mathcal{D}_{\mathbf{p}})$ in the described situation. It has the same maxima, since the logarithm is a monotonic function.

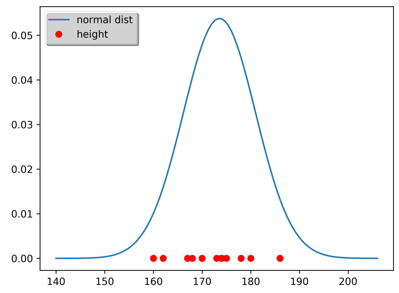

Fig. 1. Gaussian fit to one dimensional data

In the first project we had to apply the maximum likelihood technique to find a suitable probability density describing the given data. In the first task we assumed a Gaussian distribution and in the second a Weibull distribution.

3.1. Gaussian Fit

The Gaussian distribution $\mathcal{N}_{\mu,\sigma}$ with probability density function

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

is completely described by its mean μ and its standard deviation σ. Maximizing the log-likelihood $\mathcal{L}(\mu, \sigma) = \sum_{i=1}^{n} log p_{\mu,\sigma}$ by setting the derivative to zero leads to the parameters:

$$\mu_{ML} = \frac{1}{n}\sum_{i=1}^{n} x_i \quad \sigma_{ML}^2 = \frac{1}{n}\sum_{i=1} n(x_i - \mu)^2.$$

The problem is, that the estimator σ_{ML} is *biased* (i.e. makes a systematic error). Maximum likelihood estimators are not guaranteed to be unbiased. We used μ_{ML} and the unbiased estimator $\frac{n}{n-1}\sigma_{ML}^2$, which are the sample mean and the (corrected) sample variance, to fit a Gaussian to the data. Fig. 1 shows the resulting plot.

3.2. Weibull Fit

The Weibull distribution has the probability density function

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

We again estimate the parameters α and κ by the maximum likelihood approach. Instead of using the data $x_1, \ldots, x_n$ as given, we first transformed the data in histogram form: $h(v_j) = h_j$ means the value v_j appeared h_j times in the sample. The log-likelihood then becomes:

$$\begin{aligned}\mathcal{L}(\alpha, \kappa) =& N(log\kappa - \kappa log\alpha) + \\ & (\kappa - 1)\sum_j v_j log h_j - \sum_j \left(\frac{v_j h_j}{\alpha}\right)^{\kappa}\end{aligned}$$

To find the roots of the derivative of the log-likelihood we use Newton's method, which iteratively computes a better approximation of a root [2]. Fig. 2 shows the result after 20 iterations.

Additionally, we used *gradient ascent* instead of Newton's method to find the maximum likelihood. Gradient ascent starts with any point in the solution space and iteratively takes steps towards the gradient, which points to the direction of steepest ascent. This procedure eventually can lead to a local maximum of the function [2]. Furthermore gradient ascent

$$x_{k+1} = x_k + \eta\nabla f(x_k)$$

can be seen as a special case of Euler's method for solving ordinary differential equations [2] and because of that the python solver `scipy.integrate.odeint` [3] can be used to get the same solution as well.

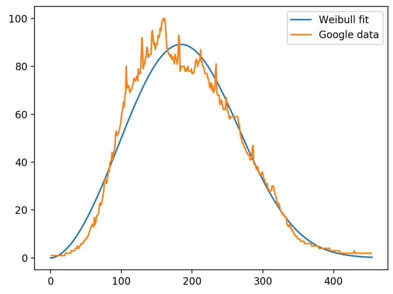

Fig. 2. Weibull fit to two dimensional data

4. UNIT CIRCLES

The next task dealt with functions of the form

$$||\mathbf{x}||_p = \left(\sum_{i=1}^{n} |x_i|^p\right)^{\frac{1}{p}}.$$

For $p \geq 1$ this is a norm. For $p < 1$ this is not a norm, since it is violating the triangle inequality (e.g. with the standard basis vectors $(1, 0, \ldots, 0), (0, 1, 0, \ldots, 0)$). Fig. 3 shows how this function looks in comparison to the standard euclidean norm.

5. FRACTALS

The last task of the first project introduced the concept of *fractal dimension* [4]. We analysed two given pictures by the box counting method to approximately determine their fractal dimension. The basic idea is as follows: Split the binarized image into 4 sub-images and count the number of images containing at least one white pixel. Recursively split those subimages until pixel level is reached. The resulting ratio at each

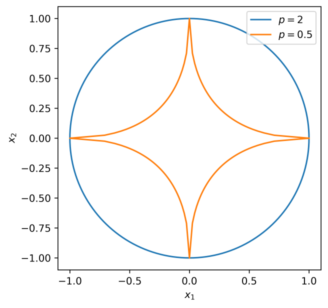

Fig. 3. $p = 0.5$ results in a so-called *quasi-norm* in contrast to an proper norm ($p = 2$).

step can be plotted. The slope of a line fitted to those points in a log-log plot yields the approximation for the fractal dimension [4]. The line can be fitted with a standard least squares approach. Fig. 4 shows the fitting for one of the pictures.

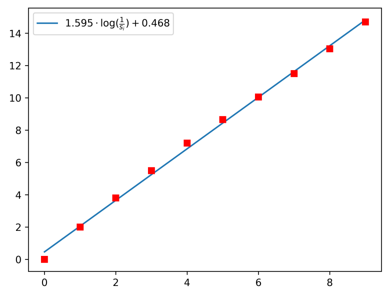

Fig. 4. Example plot for approximating the fractal dimension of an image.

6. MISSING VALUE PREDICTION

One main part of the second project was to predict missing values with different techniques.

6.1. Least Squares Approach

With this approach, we first fitted polynomials with different degrees to the data and then used the value of those to predict the missing points. To fit a polynomial of any degree we can use the least squares approach, as mentioned. The computation stays the same, we just have to use the so-called Vandermonde as a design matrix (raise each data point to the power of 1 to k for a degree k polynomial) [2]. Fig. 5 shows the result for a degree of 1, 5 and 10.

Even in this small toy-example caution is required. Normalization and the direct computation of the pseudo-inverse instead of first the standard inverse are helpful to prevent numerical issues, which can lead to wrong results.

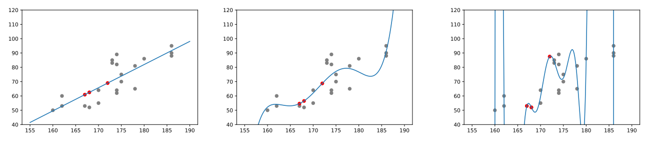

Fig. 5. Results of the polynomial fitting for degree $= 1, 5, 10$ via least squares. Predictions are marked in red. The error is decreasing in the degree, but we clearly overfit the data with the tenth order polynomial.

6.2. Gaussian Conditional Expectation Approach

In this approach, we first compute the sample mean and sample variance and use them as in section 3.1 to fit a 2d-Gaussian to the data. By marginalizing the probability density on the x-values, where we want to predict the missing y-values, we get an univariate Gaussian. We can use the mean of this marginalized Gaussian as the prediction. This is exactly what the expectation conditioned on a certain value would give us. Fig. 6 shows the result.

6.3. Bayesian Approach

As a last possibility, we tried a Bayesian approach [13] by assuming that the data come from a 5th-degree polynomial and that the coefficients (i.e. the weights) are normally distributed with given mean and variance. This prior knowledge can be incorporated into the least squares computation and leads to an adjusted formula:

$$w = (\mathbf{X}^T\mathbf{X} + \frac{\sigma^2_{sample}}{\sigma^2_{weights}}\mathbf{I})^{-1}\mathbf{X}^T\mathbf{y}.$$

A comparison between the Bayesian and the normal least squares result is shown in fig. 7.

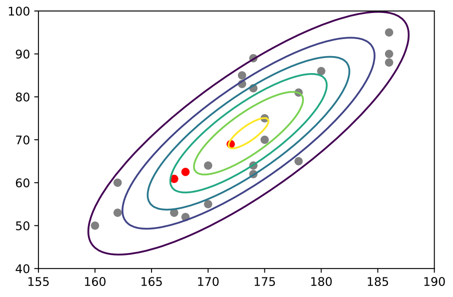

Fig. 6. Contour plot of a fitted two dimensional Gaussian.

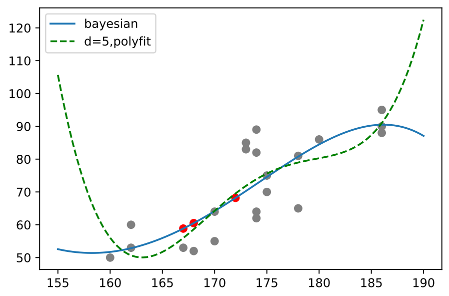

Fig. 7. Bayesian fit with prior knowledge in comparison to the normal least squares solution

6.4. Comparison

All three methods yield quite good results regarding the prediction performance, besides the first order polynomial (i.e. line) which suffers from underfitting and the tenth order polynomial which overfits the sample. If we additionally know the context of the data (height and weight values of humans), we can choose the Gaussian approach as the most reasonable one, since we know that this kind of measurements tend to be Gaussian distributed. Furthermore, it even generalizes beyond the sample area appropriately in contrast to the other two approaches, which quickly become useless for height values outside of the 155-185cm interval.

7. BOOLEAN FOURIER TRANSFORM

In this task, we learned about an application of least squares with the *Boolean Fourier Transform* to *cellular automata* [5]. A cellular automaton is an infinite array of binary cells and rules how the state of each cell is changed in one iteration. In our case, we assumed that the next state is determined by the current state of the cell itself and its two direct neighbours. It can be modelled with an 8×3 transition matrix $\mathbf{X}$ and a resulting state vector $\mathbf{y}$.

The goal is to learn a vector $\mathbf{w}$ such that the transition can be applied as a linear function (i.e. $\mathbf{Xw} = \mathbf{y}$). Since this equation could be unsolvable we resort to the best approximating solution by computing the least squares problem

$$\arg\min_{\mathbf{w}} ||\mathbf{Xw} - \mathbf{y}||^2.$$

But as it turns out the solution vector is worthless.

The idea is to represent the boolean function by its Boolean Fourier Transform with *parity basis* functions [6]

$$\phi_{\mathcal{S}}(\mathbf{x}) = \prod_{i \in \mathcal{S}} x_i,$$

for some subset $\mathcal{S}$ of the index set of boolean variables $x_1, \ldots, x_m$. We can now transform the transition matrix $\mathbf{X}$ into a feature design matrix $\mathbf{\Phi}$ by applying each of the 2^m possible ϕ to every row of X. As $\mathbf{\Phi}$ is a matrix of all basis vectors, $\mathbf{\Phi w}$ can represent any function, especially $\mathbf{y}$. So $\mathbf{\Phi w}$ = $\mathbf{y}$ and consequently $\arg\min_{\mathbf{w}} ||\mathbf{\Phi w} - \mathbf{y}||^2$ has a solution.

8. NEAREST NEIGHBOUR

The last task was to classify data with the kNN-algorithm. The main difference to the other approaches we used is, that this technique does *not* learn any explicit model of the data but instead just classifies new data by most similar known data points. Specifically, for a new data point, we find the k closest points to it and then apply majority voting (i.e. use the most frequent label of those closest points). The resulting plots for $k = 1, 3, 5$ are shown in fig. 8

Furthermore, we used `sklearn.metrics.pairwise` [7] to speed up the process of computing the distances between all test points to all training points.

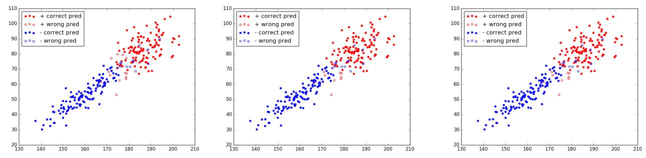

Fig. 8. Result of the kNN-algorithm for $k = 1, 3, 5$. Empty dots are misclassified. The error decreases with rising k.

8.1. KD-Tree Approach

A weakness of the naive kNN-algorithm is the fact, that it computes the distances to all training points all over again. kD-trees solve this issue [8]. A kD-tree is basically a tree of axis-parallel halfspaces separating the data space into boxes. It is guaranteed to have logarithmic height in the training data size. As one can show this leads to a logarithmic average running time for one query instead the naive linear time [9]. So by constructing the kD-tree in the beginning once, we can achieve a much better performance on average.

In the task we had implemented some variations of kD-trees: Different splitting strategies regarding which axis to pick and where to split (e.g. in the median or in the middle). Empirically the kD-Tree approach was one order of magnitude faster than the naive algorithm. Two resulting kD-trees are shown in fig. 9.

9. CLUSTERING

Clustering is an unsupervised method to group similar data together. One very popular technique for clustering is k-means. We want to find k means in the data space so that the sum of all distances between each data point and its closest mean is

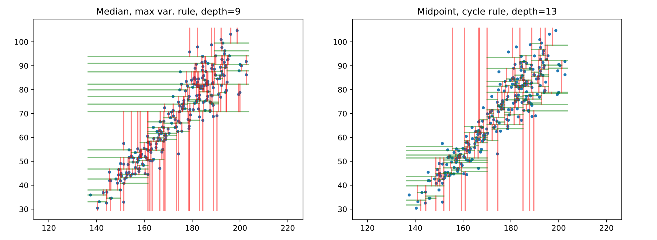

Fig. 9. Two different resulting kD-trees. The left one picks the axis which maximizes the variance and then the median of the data. The right one alternates between the axes and uses the midpoint of the data.

minimal. Hence we want to solve

$$\underset{\boldsymbol{\mu}_1,\ldots,\boldsymbol{\mu}_k}{\arg\min} \sum_{i=1}^{k} \sum_{\mathbf{x}\in C_i} ||\mathbf{x}-\boldsymbol{\mu}_i||^2,$$

where $\boldsymbol{\mu}_i$ is a mean and C_i are the data points, which have $\boldsymbol{\mu}_i$ as the closest mean. This problem is *NP*-hard [10] and therefore all known polynomial-time algorithms for this problems are heuristics. In the third, project we had to implement three different algorithms to solve this problem.

Lloyd's algorithm Lloyd's algorithm is the usual way to solve the k-means problem. We start by randomly choosing k data points and set them as the means. Now we iterate the following two steps until converges (i.e. no changes in the means). First: Compute the closest mean of each data point. Second: Move each mean to the centre of gravity of all points closest to it.

We cannot make any guarantees about the quality of the solution or the time until converges, but empirically it is usually fast [11].

Hartigan's algorithm Hartigan's algorithm tackles the problems with Lloyd's algorithm. In theory, it is more stable then Lloyd's one and has good convergence rates [12]. But empirically, at least in our experiments, the running time was two orders of magnitude slower than Lloyd's version.

MacQueen's algorithm MacQueen's algorithm is designed for a different perspective of the same problem. Instead of a static database of data it assumes that the data points arrive one by one and iteratively updates the means. It turns out that this approach is even reasonable for the static case. We could achieve competitive running times in comparison with Lloyd's algorithm.

Experiments Fig. 10 shows the clusterings of some test runs with Lloyd's algorithm. As one can see, the solution quality depends on the chosen initialization points. The other two algorithms produced similar results.

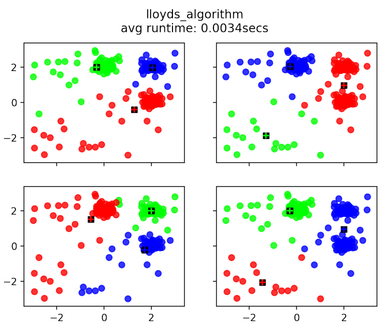

Fig. 10. Results of four test runs with Lloyd's algorithm with different starting means. The left two are the expected results.

9.1. Spectral Clustering

k-means has a big weakness, as it only can cluster ellipsoidal shaped data groups, specifically it finds a Gaussian mixture which fits the data [13]. Therefore it cannot correctly cluster groups of other forms (e.g. fig. 11). To overcome this issues different algorithms are needed. Spectral clustering is such a technique. It regards the data points as a graph with weighted edges corresponding to some measure of similarity between the points. The eigenvectors of the so-called Laplacian matrix of this graph can be regarded as *cuts* in the graph, which can be used to cluster the data [14]. Fig. 12 shows some clusterings with different parameters.

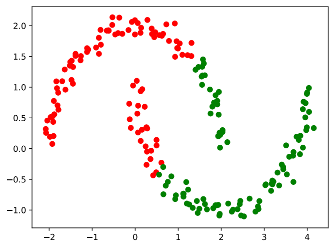

Fig. 11. k-means yields a poor result on this example.

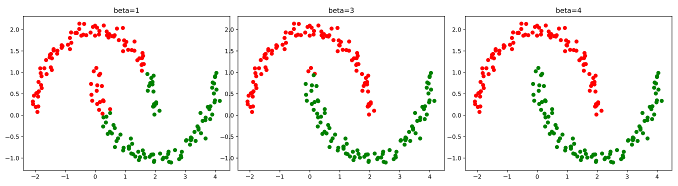

Fig. 12. Results of the spectral clustering with different similarity parameters.

10. DIMENSIONALITY REDUCTION

A usual step before we fit models to our data is dimensionality reduction. Real life datasets are often very high dimensional and hardly to accessible as a human. One idea to nevertheless get a first impression of the data is to somehow *reasonably* transform it into a 2d or a 3d representation and visualize it.

Another big advantage of dimensionality reduction is to find attributes (or combinations of attributes) which best describe the data at hand. PCA and LDA are two different methods to perform dimensionality reduction.

10.1. PCA

PCA (Principal Component Analysis) is a method which finds those axes, where the data varies the most. First, we find one axis, where the variance in the data sample along this axis is maximal. Afterwards, we find a second axis with maximal variance, but this time it must be orthogonal to the first one. We could apply this procedure until we spanned the complete data space and would get a linear transformation of it without any reduction. If we stop this procedure early, we can drastically reduce the dimension and simultaneously keep the most important structures of the data. Furthermore, it turns out that computing an eigenvalue decomposition of the covariance matrix of the data yields exactly the desired result [13].

10.2. LDA

LDA (Linear Discriminant Analysis) as unlike PCA is a supervised technique (i.e. it uses labelled data). Nevertheless, the main idea is similar to PCA. We also want to find a set of axes, which best describe the data, but this time we want to explicitly make use of the class labels. We achieve that by finding an axis which maximizes the separation between the means of the different classes and simultaneously minimizes the inner class variance. Again the axes can be computed by performing an eigenvalue decomposition

Furthermore it is noteworthy that LDA can be generalized to more than two classes [13] .

10.3. Experiments

Fig. 13 shows the visualization of a high dimensional dataset via PCA and LDA.

11. PERCEPTRONS WITH NON-MONOTONIC ACTIVATION FUNCTIONS

In the last main task of the third project we learned how to overcome typical issues in *perceptron* learning for classification. Perceptrons represent a function of the form:

$$f(\mathbf{x}) = \sigma(\mathbf{w}^T\mathbf{x} + w_0),$$

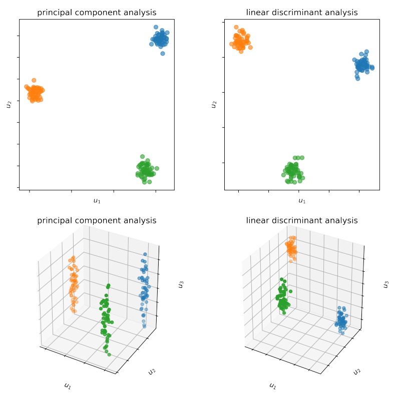

Fig. 13. Reduction of 500 dimensional data into 2 or 3 dimensions.

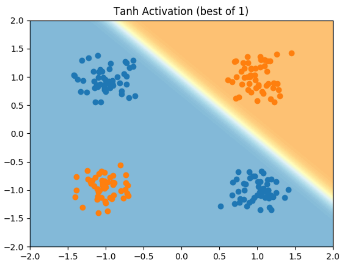

Fig. 14. A perceptron with a monotonic activation like $tanh$ cannot separate XOR shaped data like this.

where σ is called *activation function*. A perceptron can be used for binary classification by computing if $f(\mathbf{x})$ is below or above a certain threshold [13].

Given a sample $\mathbf{x_1}, \ldots, \mathbf{x_n}$ with binary labels $\mathbf{y}$ we can *train* the perceptron (i.e. determine the *weight* vector $\mathbf{w}$) to classify the sample. We start with a random initialization of $\mathbf{w}$ and then perform gradient descent on the squared error function:

$$E(\mathbf{w}) = \sum_{i=1}^{n}(f(x_i) - y_i).$$

It is well-known that a sample in the form as in fig. 14 (often called the XOR-problem) is not learnable, if the activation function σ is monotonic. In that case the learned classification just corresponds to a separating hyperplane. The problem is that the XOR-problem is not *linearly separable* [15]. If we try it nevertheless with a monotonic activation function like $tanh$, it results in a poor classification (see fig. 14).

But if we try to learn the same dataset with a non-monotonic activation function, like the Gaussian-shaped

$$\sigma(x) = 2e^{-\frac{1}{2}x^2} - 1,$$

the classification becomes possible (see fig. 15).

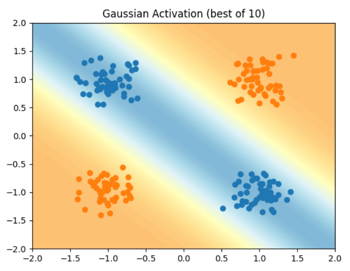

Fig. 15. A single perceptron with a non-monotonic Gaussian shaped activation can separate this dataset.

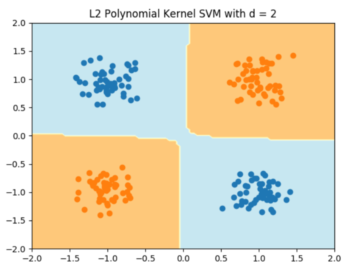

Fig. 16. Classification of the XOR-problem with a polynomial L2-SVM. A quadratic polynomial ($d = 2$) is enough to separate this sample properly.

Additionally, instead of choosing appropriate step sizes for gradient descent we decided to implement gradient descent with line search to choose the step sizes dynamically on its own [2]. Line search picks the step size which would lead to the biggest decrease in the error function along a list of possible step sizes.

Furthermore, it is reasonable to start with many different random initializations and pick the overall best, to prevent the possibility of getting stuck in a not optimal local minimum of the error function E.

12. SVM

A subtask was to classify the previous dataset with a support vector machine (SVM). SVMs are an improvement over regular perceptrons in the sense that they not only find any separating hyperplane, but the one with the largest *margin* to the closest points of it [16].

Moreover, SVMs can be used with so-called *kernel* functions instead of the usual inner product $\mathbf{w}^T\mathbf{x}$ to become highly expressive. A kernel function

$$\kappa(\mathbf{x}, \mathbf{y}) =< \phi(\mathbf{x}), \phi(\mathbf{y}) >$$

is an inner product in a (typically higher dimensional) space with an *embedding* function ϕ, which maps from the data space into this new space. We can now learn the linear classifier in that new space. The important fact is, that the separation now can be non-linear in the original space.

Additionally, we do not have to compute or even have to know the embedding function ϕ explicitly. If we can show that a function is a kernel (e.g. by Mercer's Theorem [16]), then we can use it in the SVM.

In practical situations it is useful to use the *soft-margin* SVM. Instead of finding a separating hyperplane, which is often not possible, we want to find a hyperplane which *best* separates the data, by penalizing data points on the wrong side of the hyperplane. If we quadratically penalize wrongly classified points the resulting optimization problem (soft-margin L2-SVM) becomes [17]:

$$\begin{aligned} \underset{\mathbf{w},\boldsymbol{\xi}}{\arg\min}\ ||\mathbf{w}||^2 + C\sum_{i=1}^{n}\xi_i^2 \\ \text{s.t. } y_i\mathbf{w}^T\mathbf{x_i} \geq 1 - \xi_i\ \forall i \\ \xi_i \geq 0\ \forall i \end{aligned}$$

where the data is given as $(x_1, y_1), \dots, (x_n, y_n)$ and ξ is the penalization. It turns out that this quadratic problem can be efficiently solved with the Frank-Wolfe algorithm [18]. It reduces the problem to iteratively finding the maximum of a linear objective function (e.g. $\mathbf{c}^T\mathbf{x}$) in a *unit-simplex* (i.e. the convex hull of all standard basis vectors). In such a situation the optimum must be in one of the vertices of the simplex (i.e. the optimum is attained by one of the standard basis vectors), as it is known from the field of linear programming [19]. So with the help of the Frank-Wolfe algorithm the problem of training an L2-SVM reduces to iteratively computing the maximum component of the linear objective vector $\mathbf{c}$.

We implemented the training soft-margin L2-SVM with a *polynomial* kernel $(\mathbf{w}^T\mathbf{x} + b)^d$ to learn the XOR-problem as shown in fig. 16

13. NUMERICAL INSTABILITIES

In the end of the third projects, we revisited the task of fitting a polynomial to a given dataset. We emphasized that numerical errors can occur even in professional software libraries and may lead to unnoticed wrong results. Even in toy example as in fig. 17. Usual numerical methods to stabilize the computation, like normalizing the data or using the pseudo-inverse instead of the inverse can improve the results.

14. CONCLUSION

In this report, we described our solutions to the projects. We furthermore sketched our solution processes and argued why we made certain decisions.

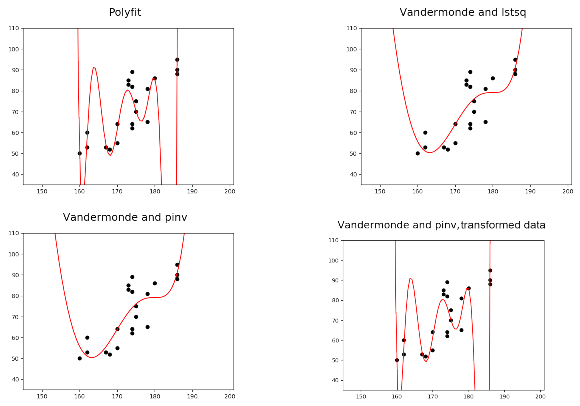

Fig. 17. Tenth order polynomial fitting with different techniques. Only the first and last approach yield the correct result.

The projects dealt with different basic tasks in pattern recognition such as density estimation via maximum likelihood and different techniques for missing value predictions. Also, we implemented many algorithms, like SVM, nearest neighbour with kD-trees, different clustering and dimensionality reduction methods, and evaluated them on datasets. Furthermore, we learnt how to overcome the limitation of perceptrons by using non-monotonic activation functions. Finally we applied some of the learnt techniques by estimating the fractal dimension of images and finding boolean functions with the Boolean Fourier Transform in the context of cellular automata.

A central theme of the tasks were challenging numerical issues, but we found methods to avoid those.

15. REFERENCES

[1] Foster Provost and Tom Fawcett, "Data science and its relationship to big data and data-driven decision making," *Big data*, vol. 1, no. 1, pp. 51–59, 2013.

[2] Uri M Ascher and Chen Greif, *A first course on numerical methods*, vol. 7, Siam, 2011.

[3] Eric Jones, Travis Oliphant, Pearu Peterson, et al., "SciPy: Open source scientific tools for Python," 2001–.

[4] Nirupam Sarkar and BB Chaudhuri, "An efficient differential box-counting approach to compute fractal dimension of image," *IEEE Transactions on systems, man, and cybernetics*, vol. 24, no. 1, pp. 115–120, 1994.

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

[6] Yishay Mansour, "Learning boolean functions via the fourier transform," in *Theoretical advances in neural computation and learning*, pp. 391–424. Springer, 1994.

[7] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay, "Scikit-learn: Machine learning in Python," *Journal of Machine Learning Research*, vol. 12, pp. 2825–2830, 2011.

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

[9] Jerome H Friedman, Jon Louis Bentley, and Raphael Ari Finkel, "An algorithm for finding best matches in logarithmic expected time," *ACM Transactions on Mathematical Software (TOMS)*, vol. 3, no. 3, pp. 209–226, 1977.

[10] Daniel Aloise, Amit Deshpande, Pierre Hansen, and Preyas Popat, "Np-hardness of euclidean sum-of-squares clustering," *Machine learning*, vol. 75, no. 2, pp. 245–248, 2009.

[11] David Arthur and Sergei Vassilvitskii, "k-means++: The advantages of careful seeding," in *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*. Society for Industrial and Applied Mathematics, 2007, pp. 1027–1035.

[12] Noam Slonim, Ehud Aharoni, and Koby Crammer, "Hartigan's k-means versus lloyd's k-means-is it time for a change?," 2013.

[13] Christopher M Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[14] Ulrike Von Luxburg, "A tutorial on spectral clustering," *Statistics and computing*, vol. 17, no. 4, pp. 395–416, 2007.

[15] Tom M Mitchell et al., *Machine learning. WCB*, McGraw-Hill Boston, MA:, 1997.

[16] Bernhard Schlkopf and Alexander J Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*, MIT press, 2001.

[17] Yichuan Tang, "Deep learning using linear support vector machines," *arXiv preprint arXiv:1306.0239*, 2013.

[18] Martin Jaggi, "Revisiting frank-wolfe: Projection-free sparse convex optimization.," in *ICML (1)*, 2013, pp. 427–435.

[19] Bernhard Korte, Jens Vygen, B Korte, and J Vygen, *Combinatorial optimization*, vol. 2, Springer, 2012.