

Principles of Machine Learning: Exercise 5

Alina Pollehn (3197257), Julian Litz (3362592), Manuel Hinz (3334548)
Felix Göhde (3336445), Felix Lehmann (3177181), Caspar Wiswesser (3221493)
Adrian Köring (3347785), Greta Günther (3326765), Linus Mallwitz (3327653)
Niklas Mueller-Goldingen (3363219), Jennifer Kroppen (2783393)

11.01.2024

Exercise 4.1: Overview

- ① Goal: Fitting a polynomial to noisy data via polynomial regression
- ② First step: Compute a feature map, where the i th column is composed of the powers x_i^0, \dots, x_i^d
- ③ Second step: Compute the weights: $\hat{w} = [\Phi\Phi^\top]^{-1}\Phi y$ using a numerically stable inversion algorithm (i.e. using QR decomposition)
- ④ Third step: Compute the fitted model: $\hat{f}(x) = \varphi(x)^\top \hat{w}$

Results: Plots

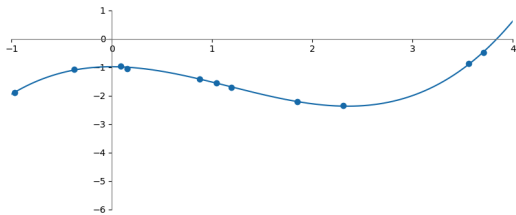


Figure: Polynomial fit for $d = 3$

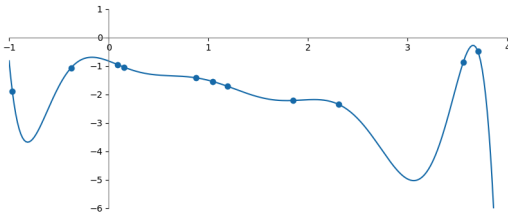


Figure: Polynomial fit for $d = 9$

Results: Discussion

- While $d = 9$ naturally gives a much better MSE, we not only know (because our data was generated by adding noise to a third degree polynomial), but can clearly see that we are overfitting!
- Therefore we would prefer our cubic fit, even if it isn't a perfect

Adding regularization

We now compute the solution to regularized least squares in the same way:

$\hat{w} = [\Phi\Phi^T + \lambda I]^{-1}\Phi y$ for different lambda:

landa = 0.005

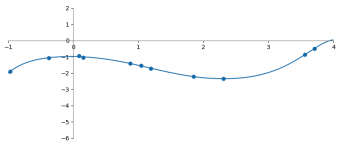


Figure: Polynomial fit for $\lambda = 0.005$

landa = 0.5

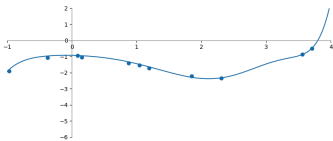


Figure: Polynomial fit for $\lambda = 0.5$

landa = 5

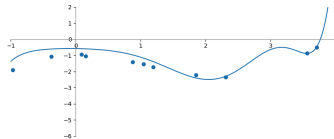


Figure: Polynomial fit for $\lambda = 5$

Regularized least squares

- Great results for smaller $\lambda = 0.5, 0.005$
- $\lambda = 5$ gives a worse result, which intuitively makes sense, because we are adding a larger value, decreasing the impact of our gram matrix before inverting!

Setting

- Lecture 07 showed that the dual least squares solution is given by $\hat{w} = \Phi[\Phi\Phi^\top]^{-1}y$
- After regularization this becomes $\hat{w} = \Phi[\Phi\Phi^\top + \lambda I]^{-1}y$
- We kernelize the expression $\hat{f}(x) = \phi^\top(x)\Phi[\Phi\Phi^\top + \lambda I]^{-1}y$ to get

$$\hat{f}(x) = k(x)^\top[K + \lambda I]^{-1}y$$

- Good choices for the model parameters where given: $\lambda = 0.5, b = 1, d = 3$

Results

- b translates the result,
- d is the degree of the polynomial, therefore influencing the shape of our model in the usual ways
- This, similarly to the previous task, leads to better results for $d = 9$, again because of the regularization

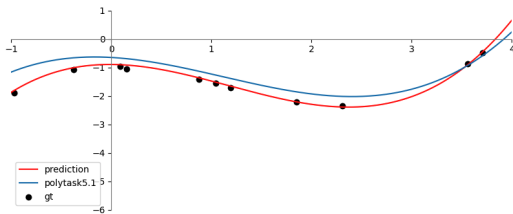


Figure: Polynomial fit for the given parameters

Exercise 5.3: Least squares SVMs for regression

- Adding to the lecture we can use SVMs for regression as well
- We want to build a least squares SVM regression model

$$\hat{f}(x) = \varphi(x)^\top \Phi \hat{\lambda} + \hat{b}$$

- Where

$$\begin{bmatrix} \hat{\lambda} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} \Phi^\top \Phi + \frac{1}{c} I & 1 \\ 1^\top & 0 \end{bmatrix} \begin{bmatrix} y \\ 0 \end{bmatrix}$$

- Which can easily be kernelized:

$$\begin{bmatrix} \hat{\lambda} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} K + \frac{1}{c} I & 1 \\ 1^\top & 0 \end{bmatrix} \begin{bmatrix} y \\ 0 \end{bmatrix}$$

and

$$\hat{f}(x) = k(x)^\top \hat{\lambda} + \hat{b}$$

- Good results for a wider range of parameters (keeping the d fixed)
- Results differ more when changing the degree d

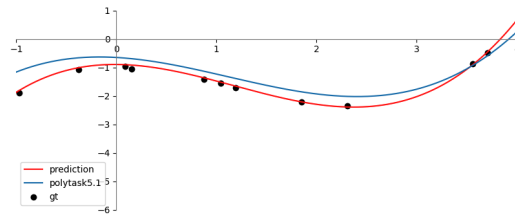


Figure: Polynomial fit for the given parameters

Other degrees

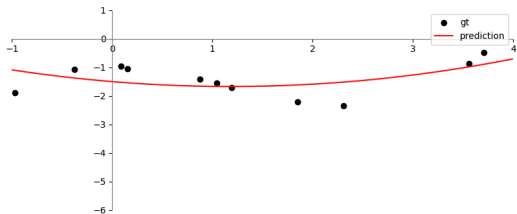


Figure: Polynomial fit for the given parameters
 $d = 2$

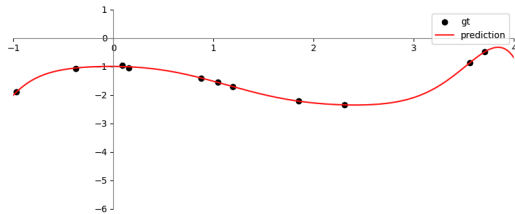


Figure: Polynomial fit for the given parameters
 $d = 9$

Exercise 5.4: Kernel SVM for binary classification

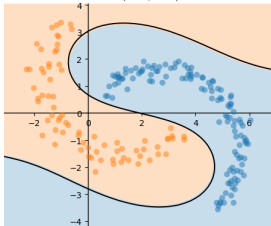
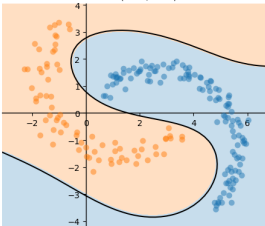
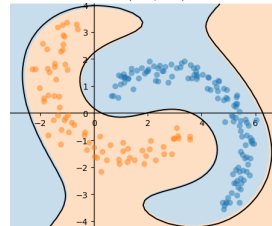
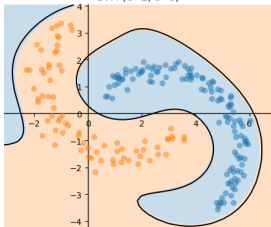
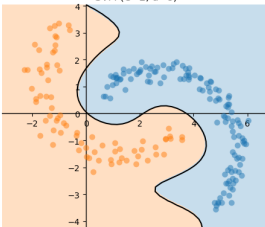
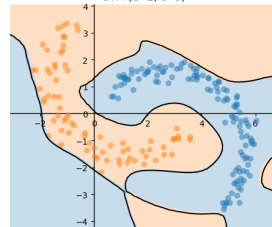
- Same math: $\begin{bmatrix} \hat{\lambda} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} K + \frac{1}{C}I & \mathbf{1} \\ \mathbf{1}^T & 0 \end{bmatrix}^{-1} \begin{bmatrix} y \\ 0 \end{bmatrix}$
- Different regression targets $y \in \{-1, 1\}$
- Polynomial Kernel $k(u, v) = (b + \mathbf{u}^T \mathbf{v})^d$

```
K = (b + (X.T @ X))**d
I = 1/C * np.eye(len(K))
One = np.ones((len(K), 1))

M = np.block([[K + I, One],
              [One.T, 0]])
t = np.block([y, 0])

params, *_ = np.linalg.lstsq(M, t)
lam, bias = params[:-1], params[-1]
```

Exercise 5.4: SVM Decision Boundary

SVM ($C=2$, $d=3$)SVM ($C=2$, $d=4$)SVM ($C=2$, $d=5$)SVM ($C=2$, $d=6$)SVM ($C=2$, $d=8$)SVM ($C=2$, $d=9$)

Minimum enclosing balls

- We already saw how to compute the minimal enclosing ball for a given dataset in lecture 08:

- Using Frank-Wolfe solve

$$\underset{\mu}{\operatorname{argmin}} \mu^{\top} X^{\top} X \mu - \mu^{\top} z$$

- where X denotes the dataset $X = [x_1, \dots, x_n] \in \mathbb{R}^{m \times n}$ and $z = \operatorname{diag}[X^{\top} X]$
 - under the constraints $1^{\top} \mu = 1$ and $\mu \geq 0$
 - given $\hat{\mu}$ we can then either compute the radius and the center of the ball

$$\hat{c} = X \hat{\mu} \text{ and } \hat{r} = \sqrt{\hat{\mu}^{\top} z - \hat{\mu}^{\top} X^{\top} X \hat{\mu}}$$

- which leads to a function

$$\chi_B(x) = \|x - \hat{c}\|^2 - \hat{r}^2$$

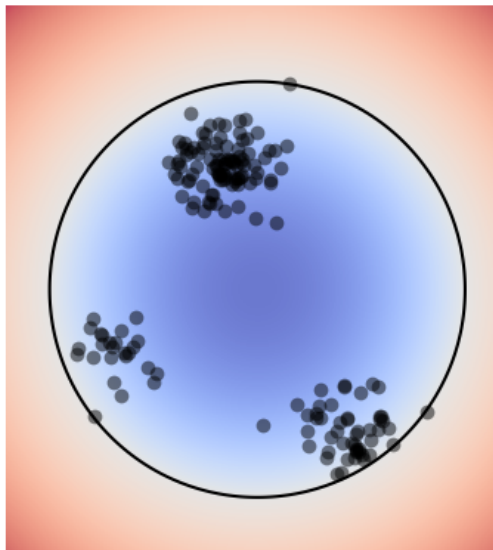
which is negative for $x \notin B \cup \partial B$!

- we can also rewrite

$$\chi_B(x) = x^{\top} x - 2x^{\top} X \hat{\mu} + \hat{\mu}^{\top} X^{\top} X \hat{\mu} - \hat{\mu}^{\top} z + \hat{\mu}^{\top} X^{\top} X \hat{\mu}$$

Result and Frank-Wolfe-Implementation

```
def fwDualMEB(matX,vecZ,T=100):  
    m, n = matX.shape  
    vecM = np.ones(n)/n  
    for t in range(T):  
        beta = 2 / (t+2)  
        vecG = 2 * matX.T @ \  
            matX @ vecM - vecZ  
        imin = np.argmin(vecG)  
        vecM *= (1-beta)  
        vecM[imin] += beta  
    return vecM
```



Kernel minimum enclosing balls

- Using our second formulation

$$\chi_B(x) = x^T x - 2x^T X \hat{\mu} + \hat{\mu}^T X^T X \hat{\mu} - \hat{\mu}^T z + \hat{\mu}^T X^T X + \hat{\mu}$$

we can kernalize everything:

- We get:

$$\chi_B(x) = K(x, x) - 2\kappa^T \hat{\mu} - \hat{\mu}^T k + 2\hat{\mu}^T K \hat{\mu}$$

- where $K(x, x) = \exp(0) = 1 \in \mathbb{R}$ and $k = 1 \in \mathbb{R}^n$, because $k_j = K(x_j, x_j) = \exp(0) = 1$.
- Using a Gaussian kernel:

$$k(u, v) = \exp\left(-\frac{1}{2\sigma^2} \|u - v\|^2\right)$$

and the following Frank-Wolfe-Algorithm:

Frank-Wolfe-Algorithm for kernel minimum enclosing balls

- Solving the minimization problem

$$\begin{aligned} & \underset{\mu}{\operatorname{argmin}} \mu^T K \mu - \mu^T k \\ & = \underset{\mu}{\operatorname{argmin}} \mu^T K \mu - \mu^T \mathbf{1} \end{aligned}$$

- under the constraints $\mathbf{1}^T \mu = 1$ and $\mu \geq 0$

```
def fwDualMEB2(K,k, T=100):
    m, n = K.shape
    vecM = np.ones(n)/n
    for t in range(T):
        beta = 2 / (t+2)
        vecG = K @ vecM - k
        imin = np.argmin(vecG)
        vecM *= (1-beta)
        vecM[imin] += beta
    return vecM
```

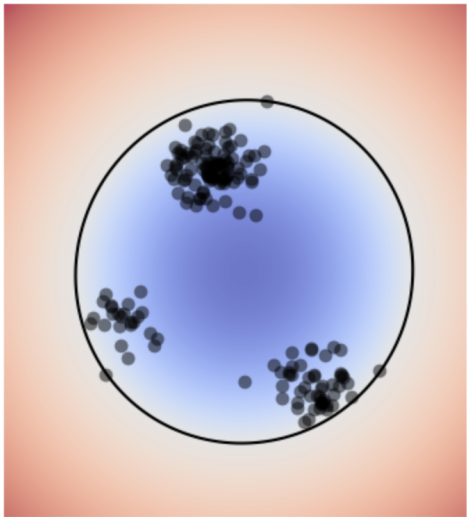


Figure: $\sigma = 4$

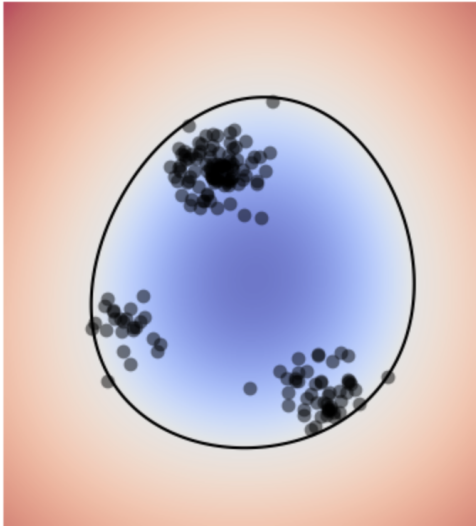


Figure: $\sigma = 2$

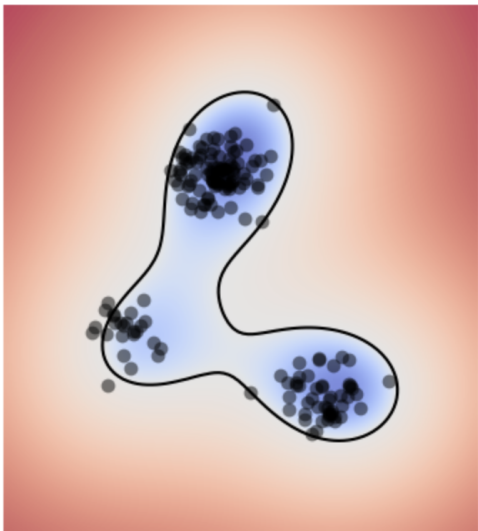


Figure: $\sigma = 1$

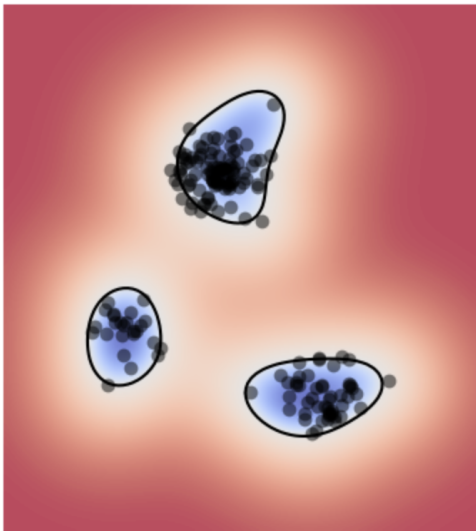


Figure: $\sigma = 0.5$