exercise 5

regularization and the kernel trick

solutions due

until **January 20, 2026** at **23:30** via **ecampus**

general remarks

The first three practical tasks of this exercise sheet should be easy and straightforward. For the last two practical tasks, we suggest you implement code similar to what we discussed in lecture 11 . . .

task 5.1 [10 points]

regularized least squares

Recall the basic setting for uni-variate least squares regression: Given training data $\mathcal{D} = \big\{\big(x_j, y_j\big)\big\}_{j=1}^{n}$, fit a model $f : \mathbb{R} \to \mathbb{R}$ such that $y_j \approx f\big(x_j\big)$. In this task, we will be working with polynomial models

$$f(x) = \sum_{p=0}^{d} w_p \, x^p = \boldsymbol{w}^\intercal \boldsymbol{\varphi}(x)$$

where function $\boldsymbol{\varphi} : \mathbb{R} \to \mathbb{R}^{d+1}$ is a feature map such that

$$\boldsymbol{\varphi}(x) = \begin{bmatrix} x^0 \\ x^1 \\ \vdots \\ x^d \end{bmatrix}$$

task 5.1.1 [2 points]

Implement a function that takes an argument $x \in \mathbb{R}$ and a parameter $d \in \mathbb{N}$ and returns $\boldsymbol{\varphi}(x)$ as defined above.

task 5.1.2 [3 points]

Read the training data $\mathcal{D}$ in file

```
noisyCubicPoly.csv
```

into am $m = 2$ by $n = 11$ *numpy* array. The first row of this array contains training inputs x_j and the second row contains training outputs y_j.
Collect the y_j into a target vector $\boldsymbol{y}$ and use your function from task 5.1.1 to compute a feature matrix

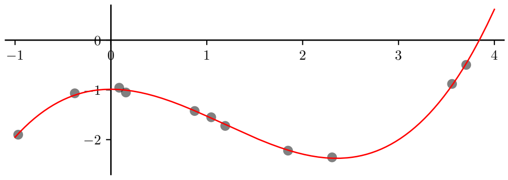
$$\boldsymbol{\Phi} = \begin{bmatrix} | & | & & | \\ \boldsymbol{\varphi}(x_1) & \boldsymbol{\varphi}(x_2) & \cdots & \boldsymbol{\varphi}(x_n) \\ | & | & & | \end{bmatrix}$$

Use a numerically stable method to compute the *ordinary least squares* solution

$$\hat{\boldsymbol{w}} = \big[\boldsymbol{\Phi}\boldsymbol{\Phi}^\intercal\big]^{-1} \boldsymbol{\Phi}\, \boldsymbol{y}$$

Finally, plot the training data together with your fitted model $\hat{f}(x) = \boldsymbol{\varphi}(x)^\intercal \hat{\boldsymbol{w}}$.

For example, for a polynomial model with $d = 3$, your plot should look like



This looks good, because the data in this task were actually sampled from a noisy third degree polynomial. **However: We deliberately want you to fit a polynomial of degree**

$$d = 9$$

Compare your fitted model to the one shown above. What are obvious differences? Which model would you say gives a better fit to the data? Which model would you say gives a more reasonable description of the data?

task 5.1.3 [5 points]

Use a numerically stable method to compute the *regularized least squares* solution

$$\hat{\boldsymbol{w}} = \left[\boldsymbol{\Phi}\boldsymbol{\Phi}^\intercal + \lambda\, \boldsymbol{I}\right]^{-1} \boldsymbol{\Phi}\, \boldsymbol{y}$$

Work with

$$d = 9$$
$$\lambda = 0.5$$

and plot the training data together with your fitted model.

Compare your result to that from task 5.1.2 and discuss what you observe.

Which model would you say provides a “subjectively” more reasonable description of the data?

For the fun of it, experiment with different choices of the *regularization parameter* λ (smaller and bigger) and see what effect these have . . .

Please read and internalize the following . . .

discussion

The model you fitted in task 5.1.2 is *over-parameterized*, i.e. too flexible for the data at hand. Adjusting it to the data therefore leads to a phenomenon called overfitting.

The model you fitted in task 5.1.3 is still *over-parameterized* but it is also *regularized*. Regularization techniques reduce the tendency of overly flexible models to overfit their training data.

If a model overfits a given set of training data, it basically learns these data by heart and makes very good prediction for them. This typically comes at the cost of poor *generalization*, i.e. of making less suitable predictions for previously unseen data points.

In the olden days, *overfitting* or *poor generalization* used to be the enemy of machine learners. Nowadays, in the age of (neural) models of billions of parameters which are trained on petabytes of data, nobody seems to worry about overfitting anymore.

The rational goes something like this: If your training data is so massive that it contains every foreseeable constellation of possible inputs, then it is a good thing to have a model that can learn all this data by heart. After all, if the training data is so massive that it covers every possible input, then who cares about generalization anymore?

However, this kind of thinking is dangerous! There can always be situations where overfitting (on massive data) will violate *decision theory* and rational thinking and we will see (didactic) examples in lecture 12.

Also, from a generative AI perspective, overfitting prevents *creativity*. Indeed, modern LLMs are often criticized for being mere stochastic parrots incapable of producing original results. Then again, image/video genAIs produce outputs nobody has seen before. But, and this is crucial, they do this using a kind of guided randomness build on top of fitted (density) models . . .

task 5.2 [10 points]

kernel least squares

In lecture 09, we learned about the *dual* least squares solution

$$\hat{\boldsymbol{w}} = \boldsymbol{\Phi}\big[\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi}\big]^{-1}\boldsymbol{y}$$

What we did not yet see is that it, too, can be regularized and then reads

$$\hat{\boldsymbol{w}} = \boldsymbol{\Phi}\big[\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi} + \lambda\,\boldsymbol{I}\big]^{-1}\boldsymbol{y} \tag{1}$$

where $\boldsymbol{I}$ is now the $n \times n$ identity matrix. In this task, you are supposed to work with the solution in (1).

Note that the (regularized) dual LSQ solution allows for invoking the kernel trick for regression. Indeed, it allows us to rewrite a fitted model

$$\hat{f}(x) = \boldsymbol{\varphi}(x)^\intercal\hat{\boldsymbol{w}} = \boldsymbol{\varphi}(x)^\intercal\boldsymbol{\Phi}\big[\boldsymbol{\Phi}^\intercal\boldsymbol{\Phi} + \lambda\,\boldsymbol{I}\big]^{-1}\boldsymbol{y}$$

in terms of a kernelized expression

$$\hat{f}(x) = \boldsymbol{k}(x)^\intercal\big[\boldsymbol{K} + \lambda\,\boldsymbol{I}\big]^{-1}\boldsymbol{y} \tag{2}$$

here is your task

Fit the model in (2) to the data in

`noisyCubicPoly.csv`

Work with a polynomial kernel matrix and a polynomial kernel vector where

$$\big[\boldsymbol{K}\big]_{ij} = \big(b + x_i\,x_j\big)^d$$
$$\big[\boldsymbol{k}(x)\big]_j = \big(b + x\,x_j\big)^d$$

Good choices for the model parameters are

$$\lambda = 0.5$$
$$b = 1$$
$$d = 3$$

Plot the training data together with your fitted model and discuss what you observe.

For the fun of it, experiment with different choices of parameters and see what kind of effect they have on the model. Especially, **see what happens if you choose** $d = 9$ . . .

Also, answer this: **Do you recognize a connection between kernel least squares and Gaussian processes?**

task 5.3 [10 points]

least squares SVMs for regression

In the lectures, we used support vector machines for classification and trained them to predict class labels $y \in \{\pm 1\}$. But we may just as well train them to predict any $y \in \mathbb{R}$ and thus use them for regression.

This is particularly easy, if we work with least squares SVMs for which the (primal) regression training problem reads

$$\begin{aligned} \underset{\boldsymbol{w},\, b,\, \boldsymbol{\xi}}{\operatorname{argmin}} \;\; & \frac{1}{2}\big[\boldsymbol{w}^\intercal \boldsymbol{w} + C\, \boldsymbol{\xi}^\intercal \boldsymbol{\xi}\big] \\ \text{s.t.} \;\; & \boldsymbol{y} = \boldsymbol{\Phi}^\intercal \boldsymbol{w} + b\,\mathbf{1} + \boldsymbol{\xi} \end{aligned}$$

Recall or observe that the solution to this training problem amounts to

$$\begin{aligned} \begin{bmatrix} \hat{\boldsymbol{\lambda}} \\ \hat{b} \end{bmatrix} &= \begin{bmatrix} \boldsymbol{\Phi}^\intercal \boldsymbol{\Phi} + \frac{1}{C}\boldsymbol{I} & \mathbf{1} \\ \mathbf{1}^\intercal & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix} \\ \hat{\boldsymbol{w}} &= \boldsymbol{\Phi}\,\hat{\boldsymbol{\lambda}} \end{aligned}$$

This way, a trained least squares SVM regression model is given by

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

Further observe that training process and model can easily be kernelized. We just need to compute

$$\begin{bmatrix} \hat{\boldsymbol{\lambda}} \\ \hat{b} \end{bmatrix} = \begin{bmatrix} \boldsymbol{K} + \frac{1}{C}\boldsymbol{I} & \mathbf{1} \\ \mathbf{1}^\intercal & 0 \end{bmatrix}^{-1} \begin{bmatrix} \boldsymbol{y} \\ 0 \end{bmatrix}$$

and then work with

$$\hat{f}(x) = \boldsymbol{k}(x)^\intercal \hat{\boldsymbol{\lambda}} + \hat{b} \tag{3}$$

here is your task

Fit the model in (3) to the data in

```
noisyCubicPoly.csv
```

Work with a polynomial kernel matrix and a polynomial kernel vector where

$$\begin{aligned}
\big[\boldsymbol{K}\big]_{ij} &= \big(b + x_i\, x_j\big)^d \\
\big[\boldsymbol{k}(x)\big]_j &= \big(b + x\, x_j\big)^d
\end{aligned}$$

Good choices for the model parameters are

$$\begin{aligned}
C &= 2 \\
b &= 1 \\
d &= 3
\end{aligned}$$

Plot the training data together with your fitted model and discuss what you observe.

For the fun of it, experiment with different choices of parameters and see what kind of effect they have on the model . . .

task 5.4 [10 points]

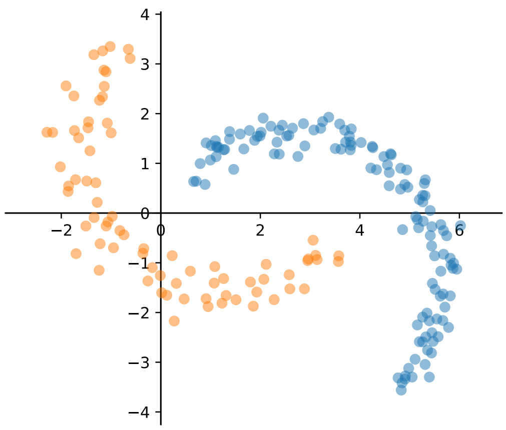
kernel SVMs for binary classification

In lecture 11, we saw how to train and to apply a kernelized L_2 SVM for binary classification.

Apply what you learned there to the training data

```
twoMoons-X-trn.csv
twoMoons-y-trn.csv
```

which when plotted may look something like this



However: Rather than working with Gaussian kernels as we did in lecture 11, please **work with polynomial kernels**

$$k(\boldsymbol{u}, \boldsymbol{v}) = \left(b + \boldsymbol{u}^\intercal \boldsymbol{v}\right)^d$$

Consider different choices of d, say 3, 4, 5, $\ldots$, and visualize the training data together with the learned decision functions.

In lecture 11, we promised to provide you with the two functions `compBBox` and `plot2dDataFnct` which will facilitate this visualization. You can find them in file

```
task5plot.py
```

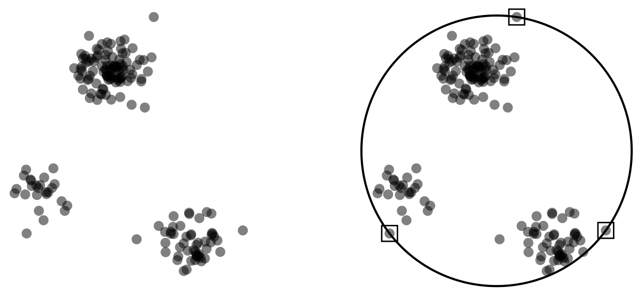
task 5.5 [10 points] + [10 bonus points]

minimum enclosing balls

The minimum enclosing ball of a set $\mathcal{X} = \{\boldsymbol{x}_j\}_{j=1}^{n} \subset \mathbb{R}^m$ of Euclidean data points is the *smallest* Euclidean m-ball

$$\mathcal{B}(\boldsymbol{c}, r) = \Big\{ \boldsymbol{x} \in \mathbb{R}^m \;\Big|\; \|\boldsymbol{x} - \boldsymbol{c}\| \leq r \Big\}$$

that contains every point in $\mathcal{X}$. While there exist numerous balls $\mathcal{B}(\boldsymbol{c}, r)$ which contain $\mathcal{X}$, there is always just one MEB $\mathcal{B}(\hat{\boldsymbol{c}}, \hat{r})$ and the following figure exemplifies that its surface is *supported* by some of the $\boldsymbol{x}_j \in \mathcal{X}$.



175 data points $\boldsymbol{x}_j \in \mathbb{R}^2$ . . .

and their MEB $\mathcal{B}$

The problem of estimating the center point $\hat{\boldsymbol{c}} \in \mathbb{R}^m$ and radius $\hat{r} \in \mathbb{R}$ of the MEB of a given set $\mathcal{X} \subset \mathbb{R}^m$ is a constrained optimization problem whose primal form reads

$$\begin{aligned} \hat{\boldsymbol{c}}, \hat{r} = \underset{\boldsymbol{c},\, r}{\operatorname{argmin}} \quad & r^2 \\ \text{s.t.} \quad & \big\|\boldsymbol{x}_j - \boldsymbol{c}\big\|^2 - r^2 \leq 0 \qquad j = 1, \ldots, n \end{aligned} \tag{4}$$

By now, we shouldn't be surprised that (4) has a dual form which is easier to solve. To write it compactly, we gather all the $\boldsymbol{x}_j \in \mathcal{X}$ in a data matrix

$$\boldsymbol{X} = \big[\, \boldsymbol{x}_1 \; \boldsymbol{x}_2 \; \cdots \; \boldsymbol{x}_n \,\big] \in \mathbb{R}^{m \times n}$$

and furthermore introduce the vector

$$\boldsymbol{z} = \operatorname{diag}\big[\boldsymbol{X}^\intercal \boldsymbol{X}\big] \in \mathbb{R}^n$$

Given these definitions, the dual MEB problem reads

$$\begin{aligned} \hat{\boldsymbol{\mu}} = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \quad & \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X} \, \boldsymbol{\mu} - \boldsymbol{\mu}^\intercal \boldsymbol{z} \\ \text{s.t.} \quad & \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \\ & \boldsymbol{\mu} \geq \mathbf{0} \end{aligned} \tag{5}$$

Once $\hat{\boldsymbol{\mu}}$ is available, we can compute center point and radius of the ball as

$$\begin{aligned} \hat{\boldsymbol{c}} &= \boldsymbol{X} \, \hat{\boldsymbol{\mu}} \\ \hat{r} &= \sqrt{\hat{\boldsymbol{\mu}}^\intercal \boldsymbol{z} - \hat{\boldsymbol{\mu}}^\intercal \, \boldsymbol{X}^\intercal \boldsymbol{X} \, \hat{\boldsymbol{\mu}}} \end{aligned}$$

task 5.5.1 [2 points]

Implement code that solves the dual MEB problem in (5). Which algorithm screams to be used for this ?

Test your code on the data in

```
threeBlobs.csv
```

Plot this data and highlight the support vectors of the surface of its MEB.

task 5.5.2 [2 points]

Let's dial things up a notch! To begin with, we note that we can determine if an arbitrary point $\boldsymbol{x} \in \mathbb{R}^m$ lies inside of $\mathcal{B}$, because if it does then

$$\left\|\boldsymbol{x} - \hat{\boldsymbol{c}}\right\|^2 \leq \hat{r}^2$$

But this is to say that function

$$\chi_{\mathcal{B}}(\boldsymbol{x}) = \left\|\boldsymbol{x} - \hat{\boldsymbol{c}}\right\|^2 - \hat{r}^2$$

which we may also write as

$$\chi_{\mathcal{B}}(\boldsymbol{x}) = \boldsymbol{x}^\intercal \boldsymbol{x} - 2\,\boldsymbol{x}^\intercal \boldsymbol{X}\,\hat{\boldsymbol{\mu}} + \hat{\boldsymbol{\mu}}^\intercal\,\boldsymbol{X}^\intercal \boldsymbol{X}\,\hat{\boldsymbol{\mu}} - \hat{\boldsymbol{\mu}}^\intercal \boldsymbol{z} + \hat{\boldsymbol{\mu}}^\intercal\,\boldsymbol{X}^\intercal \boldsymbol{X}\,\hat{\boldsymbol{\mu}} \tag{6}$$

characterizes the ball in the following sense

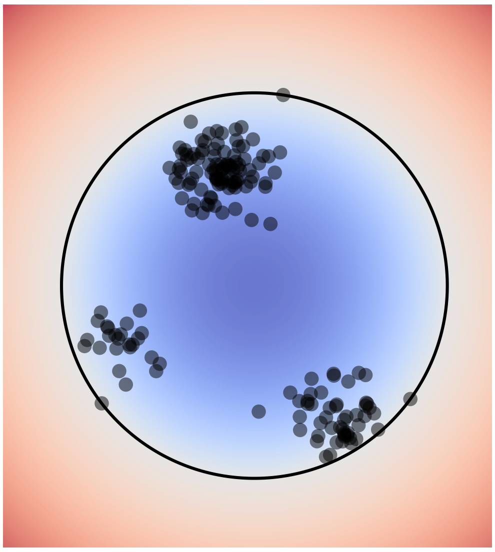
$$\chi_{\mathcal{B}}(\boldsymbol{x}) = \begin{cases} < 0 & \text{if } \boldsymbol{x} \in \mathcal{B} \setminus \partial\mathcal{B} \\ = 0 & \text{if } \boldsymbol{x} \in \partial\mathcal{B} \\ > 0 & \text{if } \boldsymbol{x} \notin \mathcal{B} \end{cases}$$

Now, reuse your code and the data from the previous task. Similar to function *compDecFnct* which you know from lecture 11 and likely used in task 5.4, implement a function *compChiFnct* that computes the values of $\chi_{\mathcal{B}}(\boldsymbol{x}) = \chi_{\mathcal{B}}(x, y)$ on a grid of 2D input points $\boldsymbol{x} = (x, y)$.

Assuming you have read the data in `threeBlobs.csv` into an array `matX`, have determined its bounding box `bbox`, and computed $\chi_{\mathcal{B}}(\boldsymbol{x})$ as we did in lecture 11 and stored it in `chiFnct`, you can use the following snippet to visualize your result

```
plot2dDataFnct([matX], bbox, fctF=chiFnct, showCont=True,
               cmap=cm.coolwarm, cmapalph=0.75)
```

If all goes well, this should produce a figure like this



If you can successfully reproduce this figure, i.e. if your code seems to work properly, you can move on to the grand finale . . .

task 5.5.3 [6 points]

Kernelize everything you did in tasks 5.5.1 and 5.5.2! **Work with Gaussian kernels**

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

and create visualizations of $\chi_{\mathcal{B}}(\boldsymbol{x})$ for the following choices of the kernel parameter

$$\sigma \in \big\{4, 2, 1, 0.5\big\}$$

Try to explain your results. That is, try to explain why kernel MEBs look the way they do . . .

task 5.5.4 [10 bonus points, not mandatory]

Use pen and paper to derive the dual in (5) from the primal in (4). **If you work with AI assistance, please tell us about your experience!**

task 5.6

submission of presentation and code

As always, prepare a presentation / set of slides on your solutions for all the mandatory tasks and submit them to eCampus. Also as always, put your code into a ZIP file and submit it to eCampus.