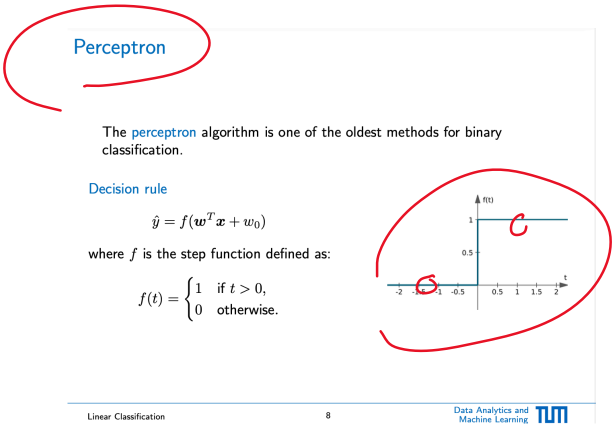
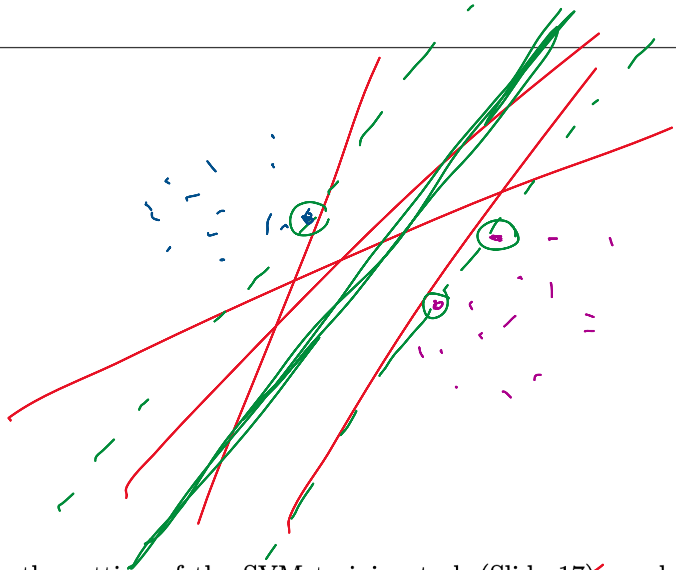
ML exercise 9 - SVM and Kernels

Tuesday, 26. January 2021 10:08

Problem 6: Explain the similarities and differences between the SVM and perceptron algorithms.

Both supervised classification methods separate two classes by a hyperplane. The difference is that SVM also tries to maximize the margin between the hyperplane and data, while perceptron only cares about separation.





Problem 7: Recall that the dual function in the setting of the SVM training task (Slide ~~17~~ 19) can be written as

$$\underbrace{1\times n \quad n\times n \quad n\times 1}_{} \Rightarrow 1\times 1$$

$$g(\boldsymbol{\alpha}) = \frac{1}{2}\boldsymbol{\alpha}^T\boldsymbol{Q}\boldsymbol{\alpha} + \boldsymbol{\alpha}^T\mathbf{1}_N\,.$$

(a) Write down the matrix $\boldsymbol{Q}$ using the vector of labels $\boldsymbol{y}$ and feature matrix $\boldsymbol{X}$. Denote the element-wise product between two matrices (in case you want to use it) by $\odot$ (also known as Hadamard product or Schur product).

(b) Prove that we can search for a *local* maximizer of g to find its *global* maximum (don't forget to prove properties of $\boldsymbol{Q}$ that you decide to use in this task).

a) slide 18

$$g(\alpha) = -\frac{1}{2}\sum_{i=1}^{N}\sum_{j=1}^{N}\alpha_i \underbrace{y_i y_j}_{\geq 0} \vec{x}_i^T\vec{x}_j \alpha_j + \sum_{i=1}^{N}\alpha_i$$

$(XX^T)_{ij} = B_{ij}$ $[n,n]$

$A_{ij} = (\vec{y}\vec{y}^T)_{ij}$ $[n,n]$ $\quad Q_{ij}$ $\quad \geq 0$ $\quad \alpha_i\alpha_j \geq 0$

$$A_{ij} = \begin{bmatrix} y_1y_1 & y_1y_2 & \cdots \\ y_2y_1 & \ddots & \\ y_ny_1 & \cdots & \end{bmatrix}$$

$$= \frac{1}{2}\vec{\alpha}^T\left[-A\odot B\right]\vec{\alpha} + \alpha^T\vec{1}_N$$

$$\Rightarrow Q = -A\odot B = -\vec{y}\vec{y}^T\odot XX^T$$

b) Positive Semidefinite: $\alpha^T(A\odot B)\alpha \geq 0$ for any $\alpha\in\mathbb{R}^n\setminus\vec{0}_n$

$$= \sum^{n}\sum^{n}(\alpha_i y_i \vec{x}_i^T)(\alpha_j y_j x_j) = \sum^{n}\sum^{n}\alpha_i\alpha_j y_i y_j \vec{x}_i^T\vec{x}_j$$

$\geq 0 \quad \geq 0 \quad \geq 0 \quad \geq 0$

$\geq 0 \Rightarrow (A \odot B)$ is positive semi definite

$-(A \odot B)$ is negative semidefinite

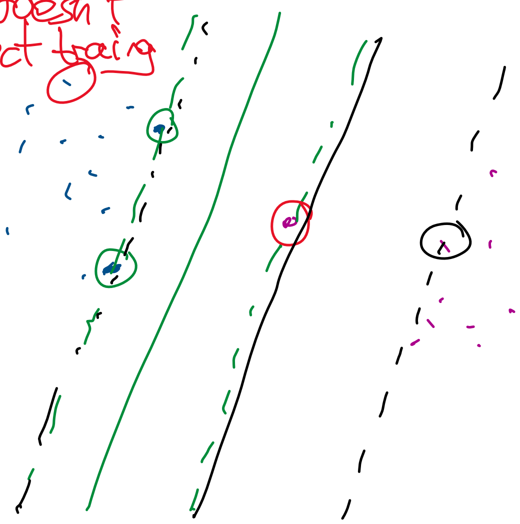
$\Rightarrow g(\alpha) = \vec{\alpha}^T Q \vec{\alpha}$ is concave! (local = global optimum)

Problem 8: Consider training a standard hard-margin SVM on a linearly separable training set of N samples. Let s denote the number of support vectors we would obtain if we would train on the entire dataset. Furthermore, let ε denote the leave-one-out cross validation (LOOCV) misclassification rate. Prove that the following relation holds:

$$\varepsilon \leq \frac{s}{N}.$$

it doesn't affect traing

leave out



$N-s$ folds, the data is perfectly seperated

if we leave out a support vector

$\Rightarrow$ we might have a misclassification

can happen at most s times

$\Rightarrow \varepsilon \leq \frac{s}{N}$

Problem 9: Load the notebook `09_homework_svm_kernels.ipynb` from Piazza. Fill in the missing code and run the notebook. Convert the evaluated notebook to pdf and add it to the printout of your homework.

all eigenvalues ≥ 0

$(A \odot B)\alpha = \lambda \alpha$

$\vec{\alpha}^T \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} \vec{\alpha} = \lambda_1 \alpha_1 \alpha_1 + \dots \lambda_n \alpha_n \alpha_n$

is concave if this holds:

$\frac{\partial^2}{\partial^2 \alpha_i} = \lambda_i \leq 0$

exercise_solution_09_notebook

January 26, 2021

1 Programming assignment 4: SVM

```
[1]: import numpy as np
```

```
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.datasets import make_blobs

from cvxopt import matrix, solvers
```

1.1 Your task

In this sheet we will implement a simple binary SVM classifier. Your task is to complete the functions where required. You are only allowed to use built-in Python functions, as well as any `numpy` functions. No other libraries / imports are allowed.

To solve optimization tasks we will use **CVXOPT** http://cvxopt.org/ - a Python library for convex optimization. If you use `Anaconda`, you can install it using

`conda install cvxopt`

1.2 Exporting the results to PDF

Once you complete the assignments, export the entire notebook as PDF and attach it to your homework solutions. The best way of doing that is 1. Run all the cells of the notebook. 2. Export/download the notebook as PDF (File -> Download as -> PDF via LaTeX (.pdf)). 3. Concatenate your solutions for other tasks with the output of Step 2. On a Linux machine you can simply use `pdfunite`, there are similar tools for other platforms too. You can only upload a single PDF file to Moodle.

Make sure you are using **nbconvert** Version 5.5 or later by running `jupyter nbconvert --version`. Older versions clip lines that exceed page width, which makes your code harder to grade.

1.3 Generate and visualize the data

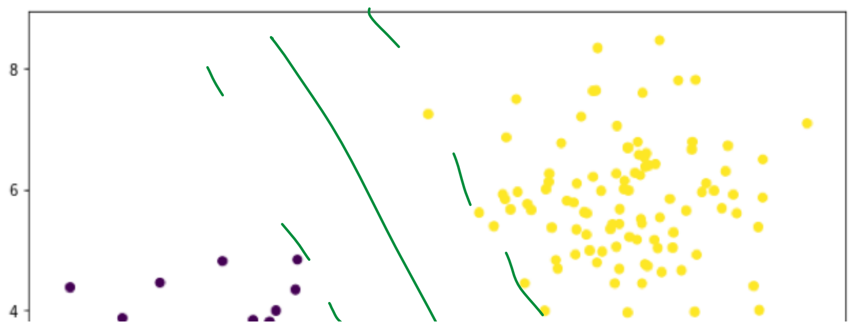
[3]:
```
N = 200  # number of samples
D = 2  # number of dimensions
C = 2  # number of classes
```

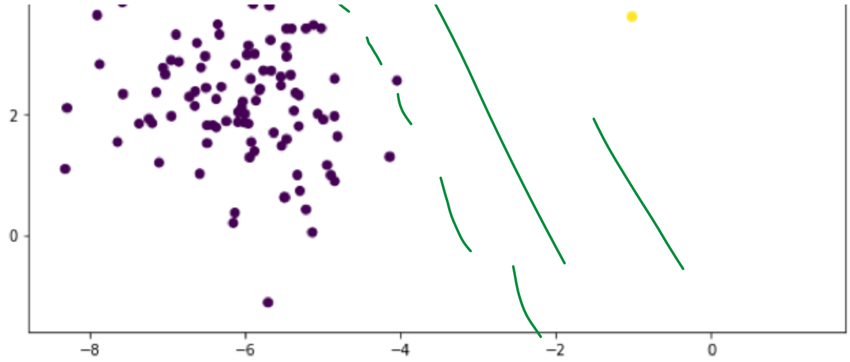


```
seed = 1234  # for reproducible experiments

alpha_tol = 1e-4 # threshold for choosing support vectors

X, y = make_blobs(n_samples=N, n_features=D, centers=C, random_state=seed)
y[y == 0] = -1  # it is more convenient to have {-1, 1} as class labels
↪(instead of {0, 1})
y = y.astype(np.float)
plt.figure(figsize=[10, 8])
plt.scatter(X[:, 0], X[:, 1], c=y)
plt.show()
```





1.4 Task 1: Solving the SVM dual problem

Remember, that the SVM dual problem can be formulated as a Quadratic programming (QP) problem. We will solve it using a QP solver from the `CVXOPT` library.

It has also "bindings" for Pytorch & Tensorflow

We use the following form of a QP problem:

$$\text{minimize}_{\mathbf{x}} \quad \frac{1}{2}\mathbf{x}^T\mathbf{P}\mathbf{x} + \mathbf{q}^T\mathbf{x}\text{subject to} \quad \mathbf{G}\mathbf{x} \leq \mathbf{h} \text{ and } \mathbf{A}\mathbf{x} = \mathbf{b}\,.$$

From task 7: $\cancel{\max}\ \min\ -\frac{1}{2}\vec{x}^T Q \vec{x} + \vec{x}^T \vec{1}_N$

are not the same

$\Rightarrow P = -Q = \vec{y}\vec{y}^T \odot XX^T$

$q = -\vec{1}_N$

Recall constraints (slide 18): $\sum_{i=1}^{N} \alpha_i y_i = 0$ and $\alpha_i \geq 0$ for $i = 1, \dots, N$

$A = \vec{y}^T$ and $b = 0$

$G = -I_N$ and $\vec{h} = \vec{0}$

$$\begin{bmatrix} -1 & 0 & \cdots & 0 \\ 0 & -1 & & \vdots \\ \vdots & & \ddots & \vdots \\ 0 & \cdots & & -1 \end{bmatrix} \begin{bmatrix} \alpha_1 \\ \alpha_2 \\ \vdots \\ \alpha_n \end{bmatrix} = \begin{bmatrix} -\alpha_1 \\ -\alpha_2 \\ -\alpha_3 \\ \vdots \\ -\alpha_n \end{bmatrix} \leq \begin{array}{c} \begin{bmatrix} 0 \\ 0 \\ \vdots \\ 0 \end{bmatrix} \end{array} \begin{array}{l} i=1 \\ i=2 \\ \\ \\ \end{array}$$

$\Rightarrow \vec{\alpha}^*$

Slide 20

"is a linear combination of...

$w = \sum^{N} \alpha^*_i y_i \vec{x}_i$ (slide 17)

$w = \sum_{i=1} \alpha_i y_i x_i$ (slide 14)

for any support vector ($\alpha_i^* > 0$):

$$b = y_i - w^T x_i$$

$$\underbrace{w^T x_i + b}_{\{-1,1\}} = \underbrace{y_i}_{\{-1,1\}}$$

Your task is to formulate the SVM dual problems as a QP of this form and solve it using `CVXOPT`, i.e. specify the matrices $\mathbf{P}, \mathbf{G}, \mathbf{A}$ and vectors $\mathbf{q}, \mathbf{h}, \mathbf{b}$.

[4]:
```
def solve_dual_svm(X, y):
    """Solve the dual formulation of the SVM problem.

    Parameters
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).

    Returns
    -------
    alphas : array, shape [N]
        Solution of the dual problem.
    """
    # TODO
    N, D = X.shape
    P = matrix(np.einsum("i,j,ik,jk->ij", y, y, X, X))
    # Also possible:
    # K = y[:, None] * X
    # P = matrix(K.dot(K.T))
    q = matrix(-np.ones([N, 1]))   matrix of shape [N,1]
    G = matrix(-np.eye(N))
    h = matrix(np.zeros(N))
    A = matrix(y.reshape(1, -1))
    b = matrix(np.zeros(1))
    solvers.options['show_progress'] = False
    solution = solvers.qp(P, q, G, h, A, b)
    alphas = np.array(solution['x'])
    return alphas.reshape(-1)
```

1.5 Task 2: Recovering the weights and the bias

[5]:
```
def compute_weights_and_bias(alpha, X, y):
    """Recover the weights w and the bias b using the dual solution alpha.

    Parameters
    ----------
    alpha : array, shape [N]
        Solution of the dual problem.
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).
```

```
    Returns
    -------
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    w = np.einsum("i,i,ij->j", alpha, y, X)
    # Also possible: w = np.dot(X.T, alpha * y)
    support_vecs = (alpha > alpha_tol)
    biases = y[support_vecs] - np.dot(X[support_vecs, :], w)
    #b = np.mean(biases)
    b = np.sum(alpha[support_vecs]*biases) / np.sum(alpha[support_vecs])  #␣
↪more numerically stable solution, see Bishop (Eq.7.18)
    return w, b
```

1.6 Visualize the result (nothing to do here)

[6]:
```
def plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b):
    """Plot the data as a scatter plot together with the separating hyperplane.

    Parameters
    ----------
    X : array, shape [N, D]
        Input features.
    y : array, shape [N]
        Binary class labels (in {-1, 1} format).
    alpha : array, shape [N]
        Solution of the dual problem.
    w : array, shape [D]
        Weight vector.
    b : float
        Bias term.
    """
    plt.figure(figsize=[10, 8])
    # Plot the hyperplane
    slope = -w[0] / w[1]
    intercept = -b / w[1]
    x = np.linspace(X[:, 0].min(), X[:, 0].max())
    plt.plot(x, x * slope + intercept, 'k-', label='decision boundary')
    plt.plot(x, x * slope + intercept - 1/w[1], 'k--')
    plt.plot(x, x * slope + intercept + 1/w[1], 'k--')
    # Plot all the datapoints
    plt.scatter(X[:, 0], X[:, 1], c=y)
    # Mark the support vectors
```



```
    support_vecs = alpha > alpha_tol
```

```
    plt.scatter(X[support_vecs, 0], X[support_vecs, 1], c=y[support_vecs],
 ↪s=250, marker='*', label='support vectors')
    plt.xlabel('$x_1$')
    plt.ylabel('$x_2$')
    plt.legend(loc='upper left')
```

The reference solution is

```
w = array([0.73935606 0.41780426])

b = 0.919937145
```

Indices of the support vectors are

```
[ 78 134 158]
```

[7]:
```
alpha = solve_dual_svm(X, y)
w, b = compute_weights_and_bias(alpha, X, y)
print("w =", w)
print("b =", b)
print("support vectors:", np.arange(len(alpha))[alpha > alpha_tol])
```

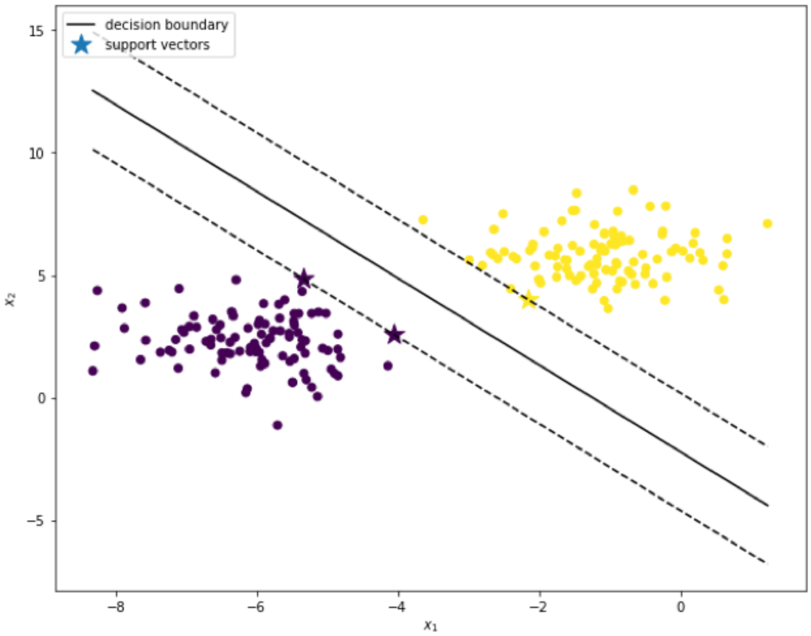
```
w = [0.73935606 0.41780426]
b = 0.9199371344144426
support vectors: [ 78 134 158]
```

[8]:
```
plot_data_with_hyperplane_and_support_vectors(X, y, alpha, w, b)
plt.show()
```

```
alpha
```

```
array([3.93025192e-10, 4.08362586e-10, 2.58659277e-09, 6.34201914e-10,
       1.22177769e-09, 6.10164590e-10, 6.80668606e-10, 4.27744471e-10,
       4.64884594e-10, 3.97070913e-10, 4.45316138e-10, 5.30926634e-10,
       5.01901154e-10, 5.85627374e-10, 4.71664314e-10, 4.92456797e-10,
       6.61337588e-10, 2.94726610e-09, 1.05147458e-09, 4.40692729e-10,
       4.03132563e-10, 7.92744191e-10, 5.81928533e-10, 6.65531094e-10,
       5.61127996e-10, 7.18900560e-10, 8.35560781e-10, 2.51021530e-08,
       9.32920115e-10, 4.96114843e-10, 5.09514243e-10, 5.57273002e-10,
       7.03766707e-10, 8.37350834e-10, 7.26742431e-08, 1.78206019e-09,
       5.06686134e-10, 1.02437311e-09, 4.21063117e-10, 8.30267616e-10,
       5.93588579e-10, 6.10531697e-10, 1.65878310e-09, 4.35779495e-10,
       7.12102437e-10, 4.79614578e-10, 7.47674018e-10, 6.25038985e-10,
       7.24676283e-10, 6.37298627e-10, 5.70553868e-10, 4.78245050e-10,
       5.06374553e-10, 5.96326384e-10, 6.08877197e-10, 7.61174973e-10,
       6.87035475e-10, 9.35534636e-10, 4.07552762e-10, 3.47841691e-10,
       1.04021661e-09, 3.94633169e-10, 9.53281296e-10, 3.01867621e-09,
       8.99550966e-10, 4.15781610e-10, 6.41639323e-10, 5.41964646e-10,
       7.89943724e-10, 1.92755250e-09, 5.10567974e-10, 5.69577773e-10,
       4.24913564e-10, 1.24363334e-09, 7.14446157e-10, 4.92657951e-10,
       1.02517236e-09, 4.99104067e-10, 4.12842777e-02, 3.22290950e-09,
       5.08556790e-10, 4.11316062e-10, 9.57131748e-10, 5.16877982e-10,
       3.67839658e-10, 5.50682787e-10, 1.71806277e-09, 6.31397847e-10,
       1.07768857e-09, 2.09792187e-09, 5.51879954e-10, 5.32138134e-10,
       4.18662240e-10, 8.87039252e-10, 1.51845626e-09, 7.15755180e-10,
       1.16470633e-09, 5.00253545e-10, 8.31061694e-10, 4.97741214e-09,
       4.88721691e-10, 3.68298106e-10, 4.53580323e-10, 5.65476924e-10,
       5.87116081e-10, 1.05679795e-09, 1.12039997e-09, 4.35616126e-10,
       8.90471237e-10, 7.03896610e-10, 4.29023744e-10, 4.33839757e-10,
       4.91949815e-10, 7.66852990e-10, 1.52689334e-09, 7.42394925e-10,
       1.21443501e-09, 4.27188797e-10, 1.96514637e-09, 4.86645377e-10,
       5.29894534e-10, 4.32185859e-10, 1.20709507e-09, 3.99503608e-10,
       3.78532607e-10, 7.96252945e-10, 1.23955801e-09, 5.41312255e-10,
       5.17864236e-10, 1.40314937e-09, 4.33033431e-10, 7.68387557e-10,
       5.06064383e-10, 1.25037701e-09, 3.19319427e-01, 2.61874303e-09,
       4.09629672e-10, 5.24943554e-10, 2.89383560e-09, 3.98202072e-10,
       1.49318095e-09, 1.11382281e-09, 7.90869660e-10, 5.33162586e-10,
       4.53421446e-10, 9.75001640e-10, 1.62031482e-09, 5.44104126e-10,
       1.02396370e-09, 7.44576916e-10, 7.51920174e-10, 8.79391627e-10,
       5.25697633e-10, 1.64010983e-09, 6.06230100e-10, 5.24147370e-10,
       7.11765701e-10, 5.88934391e-10, 3.60581688e-01, 5.33518069e-10
```

```
6.72761290e-10, 4.42991829e-10, 9.90379264e-10, 1.28212828e-09,
5.17064928e-10, 4.03924695e-10, 4.90782013e-10, 1.97621611e-09,
1.68824703e-09, 6.31063060e-10, 4.73339580e-10, 1.24339255e-09,
4.83158497e-10, 8.29564202e-10, 5.06686610e-10, 7.63798516e-10,
1.04407816e-09, 1.25445606e-09, 5.25538888e-09, 8.54745354e-10,
```



Kernel construction rules:

Let $k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ and $k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ be kernels, with $\mathcal{X} \subseteq \mathbb{R}^N$.
Then the following functions k are kernels as well:

- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2) + k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$ (1)
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = c \cdot k_1(\boldsymbol{x}_1, \boldsymbol{x}_2)$, with $c > 0$ (2)
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_1(\boldsymbol{x}_1, \boldsymbol{x}_2) \cdot k_2(\boldsymbol{x}_1, \boldsymbol{x}_2)$ (3)
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = k_3(\phi(\boldsymbol{x}_1), \phi(\boldsymbol{x}_2))$, with the kernel k_3 on $\mathcal{X}' \subseteq \mathbb{R}^M$ and $\phi : \mathcal{X} \to \mathcal{X}'$ (4)
- $k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \boldsymbol{x}_1 \boldsymbol{A} \boldsymbol{x}_2$, with $\boldsymbol{A} \in \mathbb{R}^N \times \mathbb{R}^N$ symmetric and positive semidefinite (5)

4 Kernels

Problem 10: Show that for $N \in \mathbb{N}$ and $a_i \geq 0$ for $i = 0, \ldots, N$ the following function k is a valid kernel.

$$k(\boldsymbol{x}_1, \boldsymbol{x}_2) = \underbrace{\sum_{i=1}^{N} \underbrace{a_i \left(\boldsymbol{x}_1^T \boldsymbol{x}_2\right)^i}_{\text{rule 2}}}_{\text{rule 1}} + a_0, \text{ with } \boldsymbol{x}_1, \boldsymbol{x}_2 \in \mathbb{R}^d.$$

$$\phi(x_1)\phi(x_2) \quad \& \quad \phi(x) = \begin{bmatrix} \sqrt{a_0} \\ \sqrt{a_1}\, x \\ \sqrt{a_2}\, x^2 \\ \vdots \\ \sqrt{a_N}\, x^N \end{bmatrix} \leftarrow 4$$

Problem 11: Find the feature transformation $\boldsymbol{\phi}(x)$ corresponding to the kernel

$$k(x_1, x_2) = \frac{1}{1 - x_1 x_2}, \text{ with } x_1, x_2 \in (0, 1).$$

Hint: Consider an infinite-dimensional feature space.

Geometric series:

$$s = \frac{1}{1-r} = 1 + r + r^2 + \ldots \quad \text{if } 0 < 1 < r$$

Proof $\quad s_n = 1 + r + r^2 + \ldots + r^n \qquad | \cdot r$

$$r s_n = r + r^2 + \ldots \quad r^{n+1}$$

$$s_n - r = 1 - r^{n+1}$$

$$s_n(1-r) = 1 - r^{n+1} \qquad | : (1-r)$$

$$s_n = \frac{1 - r^{n+1}}{\ldots}$$

$$s_n = \frac{\cdots}{1-r} \qquad r \neq 1$$

$$\to 0 \qquad |r| < 1$$

$$s = \lim_{n\to\infty} s_n = \lim_{n\to\infty} \frac{1 - r^{n+1}}{\underbrace{1 - r}_{\text{const.}}} = \frac{1}{1-r}$$

$$h(x_1, x_2) = \frac{1}{1 - \underbrace{x_1 x_2}_{r}} = \sum_{i=0}^{\infty} (x_1 x_2)^i = \sum_{i=1}^{\infty} x_1^i x_2^i$$

$$\phi(x) = \begin{bmatrix} 1 \\ x \\ x^2 \\ \vdots \\ x^{\infty} \end{bmatrix}$$