These are the slides of the lecture

Pattern Recognition
Winter term 2020/21
Friedrich-Alexander University of Erlangen-Nuremberg.

Erlangen, January 8, 2021
Prof. Dr.-Ing. Andreas Maier

Pattern Recognition (PR)

Prof. Dr.-Ing. Andreas Maier
Pattern Recognition Lab (CS 5), Friedrich-Alexander-Universität Erlangen-Nürnberg
Winter Term 2020/21

Kernels

Motivation

Linear decision boundaries in its current form have serious limitations:

Motivation

Linear decision boundaries in its current form have serious limitations:

- too simple to provide good decision boundaries

Motivation

Linear decision boundaries in its current form have serious limitations:

- too simple to provide good decision boundaries
- non-linearly separable data cannot be classified

Motivation

Linear decision boundaries in its current form have serious limitations:

- too simple to provide good decision boundaries
- non-linearly separable data cannot be classified
- noisy data cause problems

Motivation

Linear decision boundaries in its current form have serious limitations:

- too simple to provide good decision boundaries
- non-linearly separable data cannot be classified
- noisy data cause problems
- formulation deals with vectorial data only

Motivation

Linear decision boundaries in its current form have serious limitations:

- too simple to provide good decision boundaries
- non-linearly separable data cannot be classified
- noisy data cause problems
- formulation deals with vectorial data only

Possible solution:

- Map data into a higher dimensional feature space using a non-linear feature transform, then use a linear classifier.

Dual Representation

- The SVM decision boundary can be rewritten in dual form:

$$f(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{x} + \alpha_0 = \sum_i \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + \alpha_0$$

  where we have used the identity:

$$\boldsymbol{\alpha} = \sum_i \lambda_i y_i \boldsymbol{x}_i \ .$$

Dual Representation

- The SVM decision boundary can be rewritten in dual form:

$$f(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{x} + \alpha_0 = \sum_i \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + \alpha_0$$

  where we have used the identity:

$$\boldsymbol{\alpha} = \sum_i \lambda_i y_i \boldsymbol{x}_i \ .$$

- The Lagrange dual problem is given by the optimization problem:

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_i \lambda_i \\ \text{subject to} & \boldsymbol{\lambda} \succeq 0, \quad \sum_i \lambda_i y_i = 0 \end{array}$$

Dual Representation

- The SVM decision boundary can be rewritten in dual form:

$$f(\boldsymbol{x}) = \boldsymbol{\alpha}^T \boldsymbol{x} + \alpha_0 = \sum_i \lambda_i y_i \boldsymbol{x}_i^T \boldsymbol{x} + \alpha_0$$

  where we have used the identity:

$$\boldsymbol{\alpha} = \sum_i \lambda_i y_i \boldsymbol{x}_i \ .$$

- The Lagrange dual problem is given by the optimization problem:

$$\begin{array}{ll} \text{maximize} & -\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \boldsymbol{x}_i^T \boldsymbol{x}_j + \sum_i \lambda_i \\ \text{subject to} & \boldsymbol{\lambda} \succeq 0, \quad \sum_i \lambda_i y_i = 0 \end{array}$$

Conclusion: feature vectors $\boldsymbol{x}_i$, $\boldsymbol{x}_j$, and $\boldsymbol{x}$ only appear in inner products, both in the learning and the classification phase.

Inner Product and the Perceptron

The decision boundary that we get for the perceptron can also be written in terms of inner products:

$$F(\boldsymbol{x})$$

Inner Product and the Perceptron

The decision boundary that we get for the perceptron can also be written in terms of inner products:

$$F(\boldsymbol{x}) \quad = \quad \left(\sum_{i \in \mathscr{E}} y_i \cdot \boldsymbol{x}_i\right)^T \boldsymbol{x} + \sum_{i \in \mathscr{E}} y_i$$

Inner Product and the Perceptron

The decision boundary that we get for the perceptron can also be written in terms of inner products:

$$
\begin{aligned}
F(\boldsymbol{x}) &= \left(\sum_{i \in \mathscr{E}} y_i \cdot \boldsymbol{x}_i \right)^T \boldsymbol{x} + \sum_{i \in \mathscr{E}} y_i \\
&= \sum_{i \in \mathscr{E}} y_i \cdot \langle \boldsymbol{x}_i, \boldsymbol{x} \rangle + \sum_{i \in \mathscr{E}} y_i
\end{aligned}
$$

Feature Transforms

We select a feature transform $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$, $D \geq d$, such that the resulting features $\phi(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, m$, are linearly separable.

Feature Transforms

We select a feature transform $\phi : \mathbb{R}^d \to \mathbb{R}^D$, $D \geq d$, such that the resulting features $\phi(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, m$, are linearly separable.

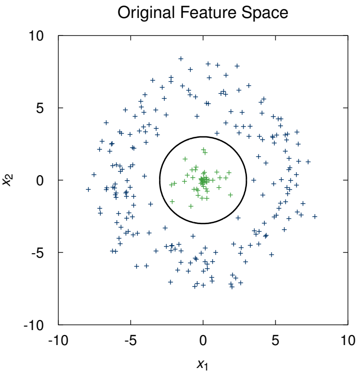


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = ({x_1}^2, {x_2}^2)^T$.

Feature Transforms

We select a feature transform $\phi : \mathbb{R}^d \rightarrow \mathbb{R}^D$, $D \geq d$, such that the resulting features $\phi(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, m$, are linearly separable.

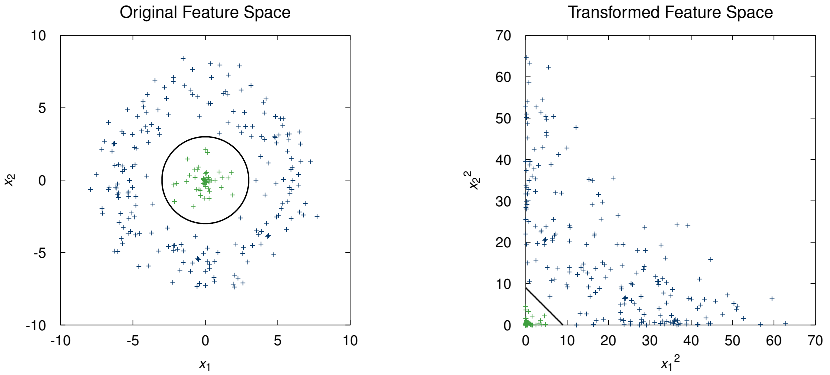
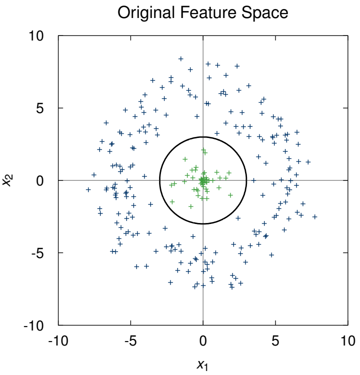


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = (x_1{}^2, x_2{}^2)^T$.

Feature Transforms

We select a feature transform $\phi : \mathbb{R}^d \to \mathbb{R}^D$, $D \geq d$, such that the resulting features $\phi(\boldsymbol{x}_i)$, $i = 1, 2, \ldots, m$, are linearly separable.



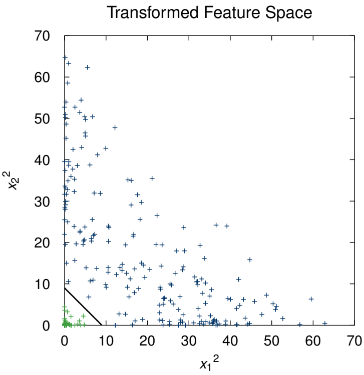


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = (x_1{}^2, x_2{}^2)^T$.

Feature Transforms (cont.)

Second Example: data is not centered

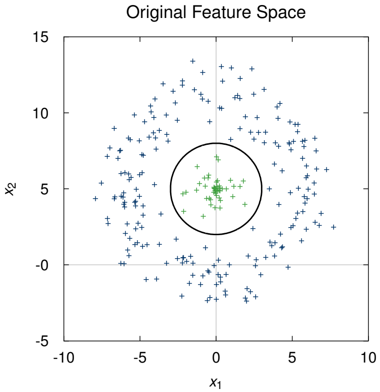
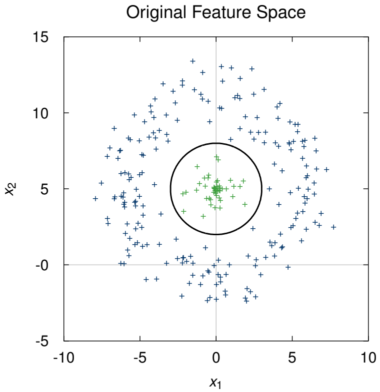


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = ({x_1}^2, {x_2}^2)^T$.

Feature Transforms (cont.)

Second Example: data is not centered



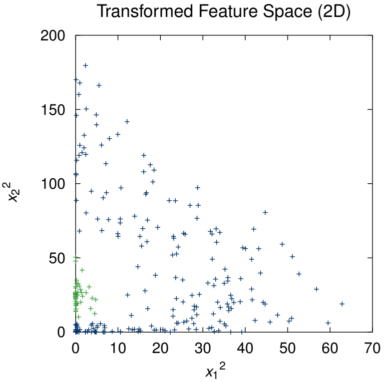
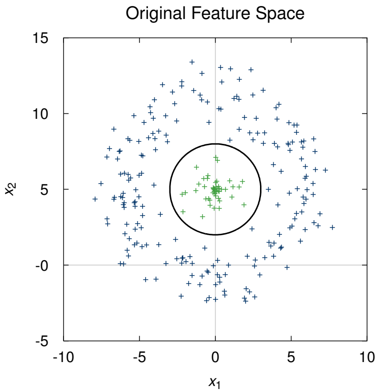


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = ({x_1}^2, {x_2}^2)^T$.

Feature Transforms (cont.)

Second Example: data is not centered



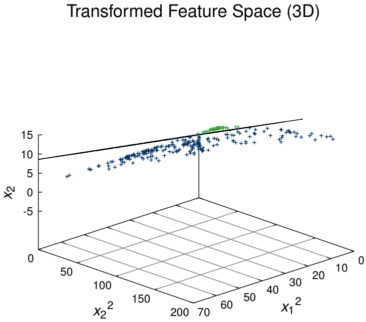


Fig.: Application of the feature transform $\phi(\boldsymbol{x}) = (x_1{}^2, x_2{}^2, x_2)^T$.

Feature Transforms (cont.)

Example

Assume the decision boundary is given by the quadratic function

$$f(\boldsymbol{x}) = a_0 + a_1 {x_1}^2 + a_2 {x_2}^2 + a_3 x_1 x_2 + a_4 x_1 + a_5 x_2 .$$

Obviously this is not a linear decision boundary.

Feature Transforms (cont.)

Example

Assume the decision boundary is given by the quadratic function

$$f(\boldsymbol{x}) = a_0 + a_1 {x_1}^2 + a_2 {x_2}^2 + a_3 x_1 x_2 + a_4 x_1 + a_5 x_2 .$$

Obviously this is not a linear decision boundary.

By the following mapping, we get features that have a linear decision boundary:

$$\phi(\boldsymbol{x}) = \begin{pmatrix} {x_1}^2 \\ {x_2}^2 \\ x_1 \cdot x_2 \\ x_1 \\ x_2 \end{pmatrix}$$

Feature Transforms (cont.)

Consider distances in the transformed feature space:

$$\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2^2$$

Feature Transforms (cont.)

Consider distances in the transformed feature space:

$$\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2^2 = \left\langle \left(\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\right), \left(\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\right) \right\rangle$$

Feature Transforms (cont.)

Consider distances in the transformed feature space:

$$\begin{aligned}\|\phi(\boldsymbol{x})-\phi(\boldsymbol{x}')\|_2^2 &= \Big\langle\big(\phi(\boldsymbol{x})-\phi(\boldsymbol{x}')\big),\big(\phi(\boldsymbol{x})-\phi(\boldsymbol{x}')\big)\Big\rangle \\ &= \langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x})\rangle-2\langle\phi(\boldsymbol{x}),\phi(\boldsymbol{x}')\rangle+\langle\phi(\boldsymbol{x}'),\phi(\boldsymbol{x}')\rangle\end{aligned}$$

Feature Transforms (cont.)

Consider distances in the transformed feature space:

$$
\begin{aligned}
\|\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\|_2^2 &= \big\langle \big(\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\big), \big(\phi(\boldsymbol{x}) - \phi(\boldsymbol{x}')\big) \big\rangle \\
&= \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}) \rangle - 2\langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle + \langle \phi(\boldsymbol{x}'), \phi(\boldsymbol{x}') \rangle
\end{aligned}
$$

Conclusion: Distances can be computed by just evaluating inner products.

Feature Transforms (cont.)

These feature transforms can be easily incorporated into SVMs:

- Decision boundary:

$$f(\boldsymbol{x}) = \sum_i \lambda_i \cdot y_i \cdot \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \rangle + \alpha_0$$

Feature Transforms (cont.)

These feature transforms can be easily incorporated into SVMs:

- Decision boundary:

$$f(\boldsymbol{x}) = \sum_i \lambda_i \cdot y_i \cdot \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}) \rangle + \alpha_0$$

- The Lagrange dual problem is given by the optimization problem:

$$\text{maximize} \qquad -\frac{1}{2} \sum_i \sum_j \lambda_i \lambda_j y_i y_j \cdot \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle + \sum_i \lambda_i$$

$$\text{subject to} \qquad \boldsymbol{\lambda} \succeq 0, \quad \sum_i \lambda_i y_i = 0$$

Kernel Functions

Definition

A *kernel function* $k : \mathscr{X} \times \mathscr{X} \rightarrow \mathbb{R}$ is a symmetric function that maps a pair of features to a real number. For a kernel function the following property holds:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

for any feature mapping ϕ.

Kernel Functions

Definition

A *kernel function* $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ is a symmetric function that maps a pair of features to a real number. For a kernel function the following property holds:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \langle \phi(\boldsymbol{x}), \phi(\boldsymbol{x}') \rangle$$

for any feature mapping ϕ.

Note:

Usually the evaluation of the kernel function is much easier than the computation of transformed features followed by the inner product.

Kernel Functions (cont.)

Definition

For a given set of feature vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m$, we define the *kernel matrix*

$$\boldsymbol{K} = [K_{i,j}]_{i,j=1,2,\dots,m}\,, \quad \text{where} \quad K_{i,j} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle .$$

Kernel Functions (cont.)

Definition

For a given set of feature vectors $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m$, we define the *kernel matrix*

$$\boldsymbol{K} \quad = \quad [K_{i,j}]_{i,j=1,2,\dots,m}\ , \quad \text{where} \quad K_{i,j} = \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle .$$

Note:

The entries of the matrix are similarity measures for transformed feature pairs.

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x} : \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x}: \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x}$$

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x} : \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} = \sum_{i,j=1}^{m} x_i x_j K_{i,j}$$

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x} : \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} = \sum_{i,j=1}^{m} x_i x_j K_{i,j} = \sum_{i,j=1}^{m} x_i x_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle$$

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x}: \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$
\begin{aligned}
\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} &= \sum_{i,j=1}^{m} x_i x_j K_{i,j} = \sum_{i,j=1}^{m} x_i x_j \left\langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \right\rangle \\
&= \sum_{i,j=1}^{m} \left\langle x_i \phi(\boldsymbol{x}_i), x_j \phi(\boldsymbol{x}_j) \right\rangle
\end{aligned}
$$

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x}: \ \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$\begin{aligned}
\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} &= \sum_{i,j=1}^{m} x_i x_j K_{i,j} = \sum_{i,j=1}^{m} x_i x_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle \\
&= \sum_{i,j=1}^{m} \langle x_i \phi(\boldsymbol{x}_i), x_j \phi(\boldsymbol{x}_j) \rangle \\
&= \left\langle \sum_{i=1}^{m} x_i \phi(\boldsymbol{x}_i), \sum_{j=1}^{m} x_j \phi(\boldsymbol{x}_j) \right\rangle
\end{aligned}$$

Kernel Functions (cont.)

Lemma

The kernel matrix is positive semidefinite.

Proof: We need to show $\forall \boldsymbol{x} : \; \boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} \geq 0$:

$$
\begin{aligned}
\boldsymbol{x}^T \boldsymbol{K} \boldsymbol{x} &= \sum_{i,j=1}^{m} x_i x_j K_{i,j} = \sum_{i,j=1}^{m} x_i x_j \langle \phi(\boldsymbol{x}_i), \phi(\boldsymbol{x}_j) \rangle \\
&= \sum_{i,j=1}^{m} \langle x_i \phi(\boldsymbol{x}_i), x_j \phi(\boldsymbol{x}_j) \rangle \\
&= \left\langle \sum_{i=1}^{m} x_i \phi(\boldsymbol{x}_i), \sum_{j=1}^{m} x_j \phi(\boldsymbol{x}_j) \right\rangle = \left\| \sum_{i=1}^{m} x_i \phi(\boldsymbol{x}_i) \right\|_2^2 \geq 0
\end{aligned}
$$

Kernel Functions (cont.)

Typical kernel functions:

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$
- Laplacian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_1}{\sigma^2}}$

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$
- Laplacian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_1}{\sigma^2}}$
- Gaussian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}{\sigma^2}}$

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$
- Laplacian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_1}{\sigma^2}}$
- Gaussian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}{\sigma^2}}$
- Sigmoid kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\alpha \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \beta)$

Kernel Functions (cont.)

Typical kernel functions:

- Linear: $k(\boldsymbol{x}, \boldsymbol{x}') = \langle \boldsymbol{x}, \boldsymbol{x}' \rangle$
- Polynomial: $k(\boldsymbol{x}, \boldsymbol{x}') = (\langle \boldsymbol{x}, \boldsymbol{x}' \rangle + 1)^d$
- Laplacian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_1}{\sigma^2}}$
- Gaussian radial basis function: $k(\boldsymbol{x}, \boldsymbol{x}') = e^{-\frac{\|\boldsymbol{x}-\boldsymbol{x}'\|_2^2}{\sigma^2}}$
- Sigmoid kernel: $k(\boldsymbol{x}, \boldsymbol{x}') = \tanh(\alpha \langle \boldsymbol{x}, \boldsymbol{x}' \rangle + \beta)$

Question:

Can we compute for any kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ a feature mapping ϕ such that the kernel function can be written as an inner product?

Kernel Functions (cont.)

Theorem (Mercer's Theorem)

For any symmetric function $k : \mathscr{X} \times \mathscr{X} \to \mathbb{R}$ that is square integrable on its domain and which satisfies

$$\int_{\mathscr{X} \times \mathscr{X}} f(\boldsymbol{x})\, f(\boldsymbol{x}')\, k(\boldsymbol{x}, \boldsymbol{x}')\, \mathrm{d}\boldsymbol{x}\, \mathrm{d}\boldsymbol{x}' \geq 0$$

for all square integrable functions f, there exist transforms $\phi_i : \mathscr{X} \to \mathbb{R}$ and $\lambda_i \geq 0$ such that:

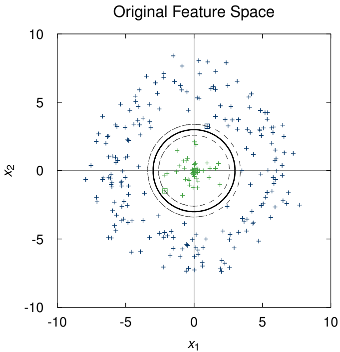
$$k(\boldsymbol{x}, \boldsymbol{x}') = \sum_i \lambda_i\, \phi_i(\boldsymbol{x})\, \phi_i(\boldsymbol{x}')$$

for all $\boldsymbol{x}$ and $\boldsymbol{x}'$.

Kernel Functions (cont.)

The Kernel Trick

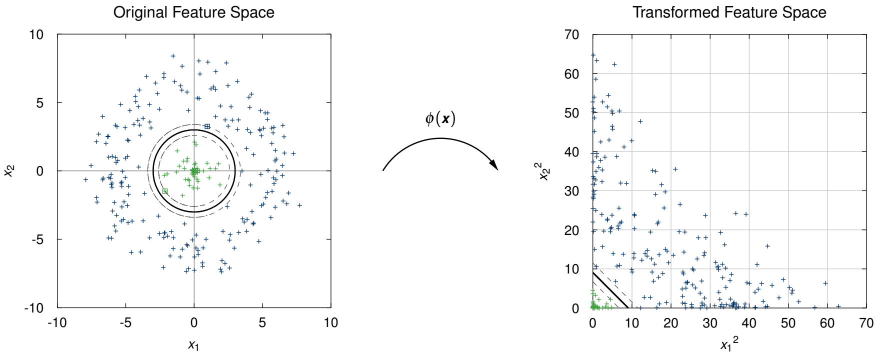
In *any* algorithm that is formulated in terms of a positive semidefinite kernel k, we can derive an alternative algorithm by replacing the kernel function k by another positive semidefinite kernel k'.



Kernel Functions (cont.)

The Kernel Trick

In *any* algorithm that is formulated in terms of a positive semidefinite kernel k, we can derive an alternative algorithm by replacing the kernel function k by another positive semidefinite kernel k'.



Kernel SVMs with Soft Margins

Linear kernel $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$:

- the complexity parameter C controls the number of support vectors and
- hence the width of the margin and
- the orientation of the decision boundary

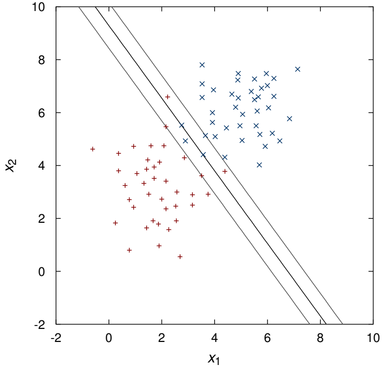


Fig.: $C = 10$: 8 support vectors, 3 misclassifications

Kernel SVMs with Soft Margins

Linear kernel $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$:

- the complexity parameter C controls the number of support vectors and
- hence the width of the margin and
- the orientation of the decision boundary

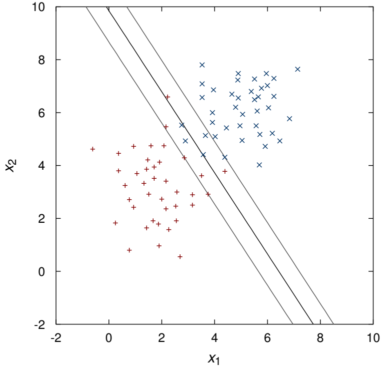


Fig.: $C = 1$: 11 support vectors, 4 misclassifications

Kernel SVMs with Soft Margins

Linear kernel $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle$:

- the complexity parameter C controls the number of support vectors and
- hence the width of the margin and
- the orientation of the decision boundary

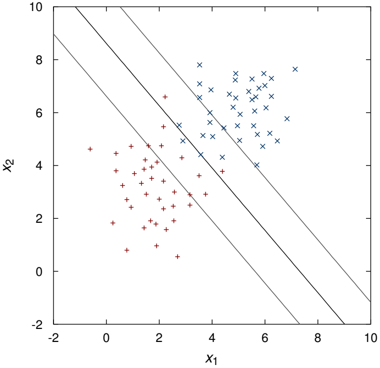


Fig.: $C = 0.1$: 17 support vectors, 3 misclassifications

Kernel SVMs with Soft Margins (cont.)

Polynomial kernel $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2$:

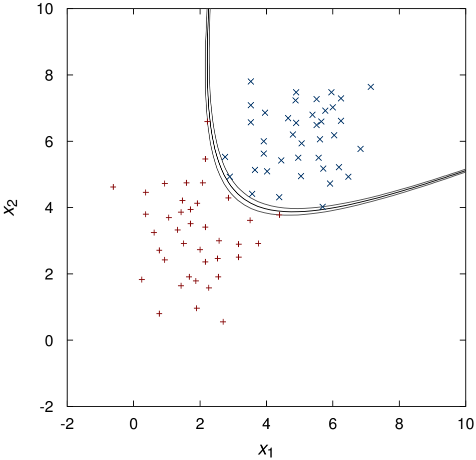


Fig.: $C = 10$: 4 support vectors, 0 misclassifications

Kernel SVMs with Soft Margins (cont.)

Polynomial kernel $\langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2$:

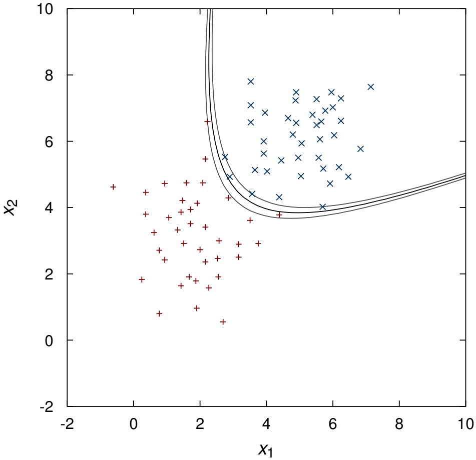


Fig.: $C = 1$: 5 support vectors, 0 misclassifications

Kernel SVMs with Soft Margins (cont.)

Gaussian RBF kernel $e^{-0.1 \cdot \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2}$:

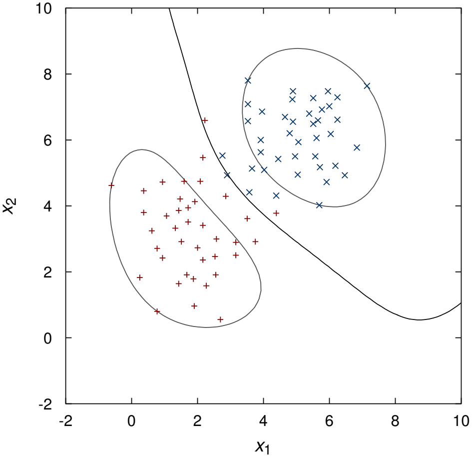


Fig.: $C = 10$: 18 support vectors, 3 misclassifications

Kernel SVMs with Soft Margins (cont.)

Gaussian RBF kernel $e^{-0.1 \cdot \langle \boldsymbol{x}, \boldsymbol{x}' \rangle^2}$:

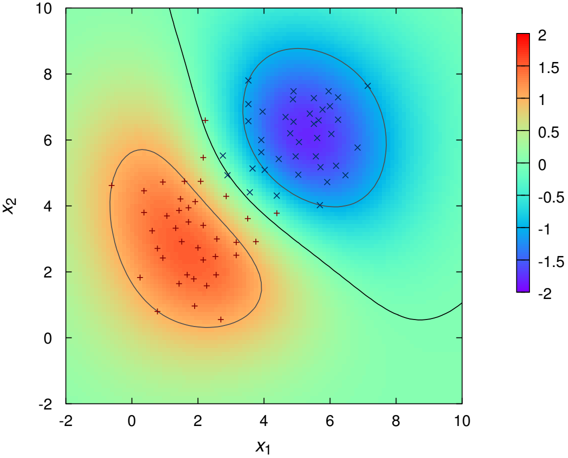


Fig.: $C = 10$: 18 support vectors, 3 misclassifications

Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Next Time in

Pattern Recognition

Kernel PCA

PCA revisited

Kernel PCA

PCA revisited

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ be the feature vectors with zero mean.

Kernel PCA

PCA revisited

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ be the feature vectors with zero mean.
- Compute the scatter matrix (covariance matrix):

$$\boldsymbol{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T \quad \in \quad \mathbb{R}^{d \times d}$$

Kernel PCA

PCA revisited

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ be the feature vectors with zero mean.
- Compute the scatter matrix (covariance matrix):

$$\boldsymbol{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T \quad \in \quad \mathbb{R}^{d \times d}$$

- Compute the eigenvectors and eigenvalues:

$$\boldsymbol{\Sigma} \boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$$

Kernel PCA

PCA revisited

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_m \in \mathbb{R}^d$ be the feature vectors with zero mean.
- Compute the scatter matrix (covariance matrix):

$$\boldsymbol{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T \quad \in \quad \mathbb{R}^{d \times d}$$

- Compute the eigenvectors and eigenvalues:

$$\boldsymbol{\Sigma} \boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$$

- Sort eigenvectors with decreasing eigenvalues.

Kernel PCA

PCA revisited

- Let $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_m \in \mathbb{R}^d$ be the feature vectors with zero mean.
- Compute the scatter matrix (covariance matrix):

$$\boldsymbol{\Sigma} = \frac{1}{m} \sum_{i=1}^{m} \boldsymbol{x}_i \boldsymbol{x}_i^T \quad \in \quad \mathbb{R}^{d \times d}$$

- Compute the eigenvectors and eigenvalues:

$$\boldsymbol{\Sigma} \boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$$

- Sort eigenvectors with decreasing eigenvalues.
- Subsequent projection of features to the eigenvectors.

Kernel PCA (cont.)

Facts from linear algebra:

Kernel PCA (cont.)

Facts from linear algebra:

- The eigenvectors $\boldsymbol{e}_i$ span the same space as the feature vectors.

Kernel PCA (cont.)

Facts from linear algebra:

- The eigenvectors $\boldsymbol{e}_i$ span the same space as the feature vectors.
- Each eigenvector $\boldsymbol{e}_i$ can be written as a linear combination of feature vectors:

$$\boldsymbol{e}_i = \sum_k \alpha_{i,k} \boldsymbol{x}_k$$

Kernel PCA (cont.)

The eigenvector/-value problem for the PCA computation can now be rewritten:

$$\Sigma \boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$$

Kernel PCA (cont.)

The eigenvector/-value problem for the PCA computation can now be rewritten:

$$\mathbf{\Sigma} \boldsymbol{e}_i = \lambda_i \boldsymbol{e}_i$$

$$\left(\frac{1}{m} \sum_{j=1}^{m} \boldsymbol{x}_j \boldsymbol{x}_j^T \right) \cdot \sum_k \alpha_{i,k} \boldsymbol{x}_k = \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_k$$

Kernel PCA (cont.)

The eigenvector/-value problem for the PCA computation can now be rewritten:

$$
\begin{aligned}
\boldsymbol{\Sigma} \boldsymbol{e}_i &= \lambda_i \boldsymbol{e}_i \\
\left(\frac{1}{m} \sum_{j=1}^{m} \boldsymbol{x}_j \boldsymbol{x}_j^T \right) \cdot \sum_k \alpha_{i,k} \boldsymbol{x}_k &= \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_k \\
\sum_{j,k} \alpha_{i,k} \boldsymbol{x}_j \boldsymbol{x}_j^T \boldsymbol{x}_k &= m \cdot \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_k
\end{aligned}
$$

Kernel PCA (cont.)

- The following equations have to be fulfilled for all projections on $\boldsymbol{x}_l$ for all indices l:

$$\sum_{j,k} \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_j \boldsymbol{x}_j^T \boldsymbol{x}_k = m \cdot \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_k$$

Kernel PCA (cont.)

- The following equations have to be fulfilled for all projections on $\boldsymbol{x}_l$ for all indices l:

$$\sum_{j,k} \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_j \boldsymbol{x}_j^T \boldsymbol{x}_k = m \cdot \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_k$$

- Wow – now all feature vectors show up in terms of inner products and the kernel trick can be applied, if *transformed* features $\phi(\boldsymbol{x})$ have zero mean.

Kernel PCA (cont.)

- The following equations have to be fulfilled for all projections on $\boldsymbol{x}_l$ for all indices l:

$$\sum_{j,k} \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_j \boldsymbol{x}_j^T \boldsymbol{x}_k = m \cdot \lambda_i \sum_k \alpha_{i,k} \boldsymbol{x}_l^T \boldsymbol{x}_k$$

- Wow – now all feature vectors show up in terms of inner products and the kernel trick can be applied, if *transformed* features $\phi(\boldsymbol{x})$ have zero mean.

- For *any* kernel $k(\boldsymbol{x}, \boldsymbol{x}')$, we get the key equation for Kernel PCA:

$$\sum_{j,k} \alpha_{i,k} \cdot k(\boldsymbol{x}_l, \boldsymbol{x}_j) \cdot k(\boldsymbol{x}_j, \boldsymbol{x}_k) = m \cdot \lambda_i \cdot \sum_k \alpha_{i,k} \cdot k(\boldsymbol{x}_l, \boldsymbol{x}_k)$$

Kernel PCA (cont.)

This can be written in matrix notation using the symmetric, positive semi-definite kernel matrix $\boldsymbol{K} \in \mathbb{R}^{m \times m}$ and the vector $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,m})^T$:

$$\begin{aligned} \boldsymbol{K}^2 \boldsymbol{\alpha}_i &= m \cdot \lambda_i \boldsymbol{K} \boldsymbol{\alpha}_i \\ \boldsymbol{K}(\boldsymbol{K}\boldsymbol{\alpha}_i) &= m \cdot \lambda_i (\boldsymbol{K}\boldsymbol{\alpha}_i) \end{aligned}$$

Kernel PCA (cont.)

This can be written in matrix notation using the symmetric, positive semi-definite kernel matrix $\boldsymbol{K} \in \mathbb{R}^{m \times m}$ and the vector $\boldsymbol{\alpha}_i = (\alpha_{i,1}, \alpha_{i,2}, \ldots, \alpha_{i,m})^T$:

$$\begin{aligned} \boldsymbol{K}^2 \boldsymbol{\alpha}_i &= m \cdot \lambda_i \boldsymbol{K} \boldsymbol{\alpha}_i \\ \boldsymbol{K}(\boldsymbol{K}\boldsymbol{\alpha}_i) &= m \cdot \lambda_i (\boldsymbol{K}\boldsymbol{\alpha}_i) \end{aligned}$$

This is equivalent to

$$\boldsymbol{K}(\boldsymbol{K}\boldsymbol{\alpha}_i - m \cdot \lambda_i \boldsymbol{\alpha}_i) = 0$$

- $\boldsymbol{K}\boldsymbol{\alpha}_i$ is an eigenvector of $\boldsymbol{K}$
- $\boldsymbol{\alpha}_i$ is an eigenvector of $\boldsymbol{K}$

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i\,\boldsymbol{\alpha}_i$$

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i \, \boldsymbol{\alpha}_i$$

Note:

- Kernel PCA (and thus the classical PCA as well) can be computed by solving an eigenvector/-value problem for an $(m \times m)$-matrix, where m is the cardinality of the training feature set.

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i \boldsymbol{\alpha}_i$$

Note:

- Kernel PCA (and thus the classical PCA as well) can be computed by solving an eigenvector/-value problem for an $(m \times m)$-matrix, where m is the cardinality of the training feature set.
- The principal components cannot be computed easily, because only the kernel is known, but not $\phi(\boldsymbol{x})$.

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i\,\boldsymbol{\alpha}_i$$

Note:

- Kernel PCA (and thus the classical PCA as well) can be computed by solving an eigenvector/-value problem for an $(m \times m)$-matrix, where m is the cardinality of the training feature set.
- The principal components cannot be computed easily, because only the kernel is known, but not $\phi(\boldsymbol{x})$.
- However, the projection c of the transformed feature vector $\phi(\boldsymbol{x})$ on principal component $\boldsymbol{e}_i = \sum_k \alpha_{i,k}\phi(\boldsymbol{x}_k)$ is easily computed by:

$$c = \phi(\boldsymbol{x})^T \boldsymbol{e}_i$$

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i \, \boldsymbol{\alpha}_i$$

Note:

- Kernel PCA (and thus the classical PCA as well) can be computed by solving an eigenvector/-value problem for an $(m \times m)$-matrix, where m is the cardinality of the training feature set.
- The principal components cannot be computed easily, because only the kernel is known, but not $\phi(\boldsymbol{x})$.
- However, the projection c of the transformed feature vector $\phi(\boldsymbol{x})$ on principal component $\boldsymbol{e}_i = \sum_k \alpha_{i,k}\phi(\boldsymbol{x}_k)$ is easily computed by:

$$c = \phi(\boldsymbol{x})^T \boldsymbol{e}_i = \sum_k \alpha_{i,k}\phi(\boldsymbol{x})^T \phi(\boldsymbol{x}_k)$$

Kernel PCA (cont.)

The coefficient vector $\boldsymbol{\alpha}_i$ for the principal components can be computed by solving the eigenvalue/-vector problem for i:

$$\boldsymbol{K}\boldsymbol{\alpha}_i = m\lambda_i\, \boldsymbol{\alpha}_i$$

Note:

- Kernel PCA (and thus the classical PCA as well) can be computed by solving an eigenvector/-value problem for an $(m \times m)$-matrix, where m is the cardinality of the training feature set.
- The principal components cannot be computed easily, because only the kernel is known, but not $\phi(\boldsymbol{x})$.
- However, the projection c of the transformed feature vector $\phi(\boldsymbol{x})$ on principal component $\boldsymbol{e}_i = \sum_k \alpha_{i,k}\phi(\boldsymbol{x}_k)$ is easily computed by:

$$c \;=\; \phi(\boldsymbol{x})^T\boldsymbol{e}_i \;=\; \sum_k \alpha_{i,k}\phi(\boldsymbol{x})^T\phi(\boldsymbol{x}_k) \;=\; \sum_k \alpha_{i,k}k(\boldsymbol{x},\boldsymbol{x}_k)$$

Kernel PCA (cont.)

It is assumed that the transformed features have zero mean:

$$\frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k) = 0.$$

Kernel PCA (cont.)

It is assumed that the transformed features have zero mean:

$$\frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k) = 0.$$

This can be enforced by the normalization step:

$$\tilde{\phi}(\boldsymbol{x}_i) = \phi(\boldsymbol{x}_i) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)$$

Kernel PCA

Implication for the components of the centered kernel matrix $\tilde{\boldsymbol{K}}$:

$$\tilde{K}_{i,j} = \tilde{\phi}(\boldsymbol{x}_i)^T \tilde{\phi}(\boldsymbol{x}_j)$$

Kernel PCA

Implication for the components of the centered kernel matrix $\tilde{\boldsymbol{K}}$:

$$\begin{aligned}\tilde{K}_{i,j} &= \tilde{\phi}(\boldsymbol{x}_i)^T \tilde{\phi}(\boldsymbol{x}_j) \\ &= \left(\phi(\boldsymbol{x}_i) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)\right)^T \left(\phi(\boldsymbol{x}_j) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)\right)\end{aligned}$$

Kernel PCA

Implication for the components of the centered kernel matrix $\tilde{\boldsymbol{K}}$:

$$\begin{aligned}
\tilde{K}_{i,j} &= \tilde{\phi}(\boldsymbol{x}_i)^T \tilde{\phi}(\boldsymbol{x}_j) \\
&= \left(\phi(\boldsymbol{x}_i) - \frac{1}{m}\sum_{k=1}^{m} \phi(\boldsymbol{x}_k)\right)^T \left(\phi(\boldsymbol{x}_j) - \frac{1}{m}\sum_{k=1}^{m} \phi(\boldsymbol{x}_k)\right) \\
&= \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j) - \frac{1}{m}\sum_{k=1}^{m} \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_k) - \frac{1}{m}\sum_{k=1}^{m} \phi(\boldsymbol{x}_k)^T \phi(\boldsymbol{x}_j) + \\
&\quad + \frac{1}{m^2}\sum_{k,l=1}^{m} \phi(\boldsymbol{x}_k)^T \phi(\boldsymbol{x}_l)
\end{aligned}$$

Kernel PCA

Implication for the components of the centered kernel matrix $\tilde{\boldsymbol{K}}$:

$$\begin{aligned}
\tilde{K}_{i,j} &= \tilde{\phi}(\boldsymbol{x}_i)^T \tilde{\phi}(\boldsymbol{x}_j) \\
&= \left(\phi(\boldsymbol{x}_i) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)\right)^T \left(\phi(\boldsymbol{x}_j) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)\right) \\
&= \phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_j) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_i)^T\phi(\boldsymbol{x}_k) - \frac{1}{m}\sum_{k=1}^{m}\phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_j) + \\
&\quad + \frac{1}{m^2}\sum_{k,l=1}^{m}\phi(\boldsymbol{x}_k)^T\phi(\boldsymbol{x}_l) \\
&= K_{i,j} - \frac{1}{m}\sum_{k=1}^{m}K_{i,k} - \frac{1}{m}\sum_{k=1}^{m}K_{k,j} + \frac{1}{m^2}\sum_{k,l=1}^{m}K_{k,l}
\end{aligned}$$

Kernel PCA (cont.)

Example: classical vs. kernel PCA

Consider $m = 50$ images with 1024^2 pixels. The pixels define 1024^2-dimensional feature vectors:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_{50} \in \mathbb{R}^{2^{20}}$$

Kernel PCA (cont.)

Example: classical vs. kernel PCA

Consider $m = 50$ images with 1024^2 pixels. The pixels define 1024^2-dimensional feature vectors:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{50} \in \mathbb{R}^{2^{20}}$$

The kernel PCA using the linear kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$$

requires the eigenvalue/-vector decomposition of the (50×50) kernel matrix.

Kernel PCA (cont.)

Example: classical vs. kernel PCA

Consider $m = 50$ images with 1024^2 pixels. The pixels define 1024^2-dimensional feature vectors:

$$\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_{50} \in \mathbb{R}^{2^{20}}$$

The kernel PCA using the linear kernel

$$k(\boldsymbol{x}_i, \boldsymbol{x}_j) = \boldsymbol{x}_i^T \boldsymbol{x}_j$$

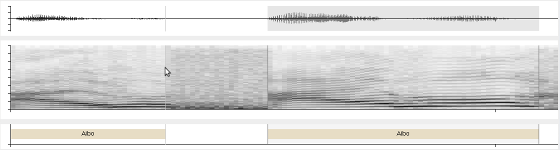
requires the eigenvalue/-vector decomposition of the (50×50) kernel matrix.

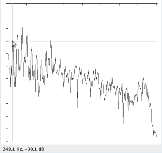
The classical PCA requires the eigenvalue/-vector decomposition of a $(2^{20} \times 2^{20})$ matrix.

Kernels for Feature Sequences

Example: string kernels

- In speech recognition we do not have feature vectors but sequences of feature vectors.
- In order to use kernel methods we need a kernel for time series.





Kernels for Feature Sequences (cont.)

Example: string kernels (cont.)

- Feature vectors are considered in $\mathbb{R}^d = \mathscr{X}$.
- Sequences of feature vectors are elements of $\mathscr{X}^*$.
- Problem: How to define a kernel over the sequence space $\mathscr{X}^*$?

Implications:

- PCA on feature sequences – COOL!
- SVM for feature sequences – EVEN COOLER!

Kernels for Feature Sequences (cont.)

Example: string kernels (cont.)

Comparison of sequences via *dynamic time warping* (DTW):

Given the feature sequences $(p, q \in \{1, 2, \dots\})$:

$$\begin{aligned} \langle \boldsymbol{x}_1, \boldsymbol{x}_2, \dots, \boldsymbol{x}_p \rangle \quad &\in \quad \mathscr{X}^* \\ \langle \boldsymbol{y}_1, \boldsymbol{y}_2, \dots, \boldsymbol{y}_q \rangle \quad &\in \quad \mathscr{X}^* \end{aligned}$$

Kernels for Feature Sequences (cont.)

Example: string kernels (cont.)

- Distance is computed by DTW:

$$D(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p \rangle, \langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_q \rangle) = \frac{1}{p} \sum_{k=1}^{p} \| \boldsymbol{x}_{v(k)} - \boldsymbol{y}_{w(k)} \|_2$$

where v, w define the mapping of indices to indices.

Kernels for Feature Sequences (cont.)

Example: string kernels (cont.)

- Distance is computed by DTW:

$$D(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p \rangle, \langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_q \rangle) = \frac{1}{p} \sum_{k=1}^{p} \| \boldsymbol{x}_{v(k)} - \boldsymbol{y}_{w(k)} \|_2$$

  where v, w define the mapping of indices to indices.

- The DTW kernel can be defined as:

$$k(\boldsymbol{x}, \boldsymbol{y}) = e^{-D(\langle \boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_p \rangle, \langle \boldsymbol{y}_1, \boldsymbol{y}_2, \ldots, \boldsymbol{y}_q \rangle)}$$

Fisher Kernels

Now we design kernels building on probability density functions $p(\boldsymbol{x};\boldsymbol{\theta})$.

- Fisher score:

$$\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) = -\frac{\partial}{\partial\boldsymbol{\theta}} \log p(\boldsymbol{x};\boldsymbol{\theta})$$

Fisher Kernels

Now we design kernels building on probability density functions $p(\boldsymbol{x};\boldsymbol{\theta})$.

- Fisher score:

$$\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) = -\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\boldsymbol{x};\boldsymbol{\theta})$$

- Fisher information matrix:

$$\boldsymbol{I}(\boldsymbol{x}) = E_{\boldsymbol{x}}[\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{\theta}}^{T}(\boldsymbol{x})]$$

Fisher Kernels

Now we design kernels building on probability density functions $p(\boldsymbol{x};\boldsymbol{\theta})$.

- Fisher score:

$$\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}) = -\frac{\partial}{\partial \boldsymbol{\theta}} \log p(\boldsymbol{x};\boldsymbol{\theta})$$

- Fisher information matrix:

$$\boldsymbol{I}(\boldsymbol{x}) = E_{\boldsymbol{x}}[\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{\theta}}^T(\boldsymbol{x})]$$

Note:

The Fisher information matrix is the curvature of the Kullback-Leibler divergence.

Fisher Kernels (cont.)

The Fisher kernel can be defined in two different ways:

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{J}_{\boldsymbol{\theta}}^T(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}')$$

or

$$k(\boldsymbol{x}, \boldsymbol{x}') = \boldsymbol{J}_{\boldsymbol{\theta}}^T(\boldsymbol{x})\boldsymbol{I}^{-1}(\boldsymbol{x})\boldsymbol{J}_{\boldsymbol{\theta}}(\boldsymbol{x}')$$

Fisher Kernels (cont.)

Application: learning from partially labeled data

Fisher Kernels (cont.)

Application: learning from partially labeled data

- Some classification approaches require huge collections of data (e. g. for text or speech recognition).

Fisher Kernels (cont.)

Application: learning from partially labeled data

- Some classification approaches require huge collections of data (e. g. for text or speech recognition).
- Labeling of the data can be time-consuming and costly.

Fisher Kernels (cont.)

Application: learning from partially labeled data

- Some classification approaches require huge collections of data (e. g. for text or speech recognition).
- Labeling of the data can be time-consuming and costly.
- If the data can be modeled with a small number of well separated components (with each component corresponding to a distinct category), little labeled data would suffice to assign a proper label to each of them.

Fisher Kernels (cont.)

Application: learning from partially labeled data

- Some classification approaches require huge collections of data (e. g. for text or speech recognition).
- Labeling of the data can be time-consuming and costly.
- If the data can be modeled with a small number of well separated components (with each component corresponding to a distinct category), little labeled data would suffice to assign a proper label to each of them.
- A machine learning approach that makes use of only partially labeled data usually achieves much better classification performance than using only the labeled data alone.

Fisher Kernels (cont.)

Application: learning from partially labeled data

- Some classification approaches require huge collections of data (e. g. for text or speech recognition).
- Labeling of the data can be time-consuming and costly.
- If the data can be modeled with a small number of well separated components (with each component corresponding to a distinct category), little labeled data would suffice to assign a proper label to each of them.
- A machine learning approach that makes use of only partially labeled data usually achieves much better classification performance than using only the labeled data alone.
- Fisher kernels describe a generative model that can be used in a discriminative approach (e. g. SVM).

Lessons Learned

- Limitations of linear decision boundaries
- Non-linear feature transforms
- Kernel function and kernel matrix
- Kernel trick
- Probabilities and kernels

Pattern Recognition Lab

FRIEDRICH-ALEXANDER UNIVERSITÄT ERLANGEN-NÜRNBERG
TECHNISCHE FAKULTÄT

Next Time in

Pattern Recognition

Further Readings

- Bernhard Schölkopf, Alexander J. Smola:
  Learning with Kernels,
  The MIT Press, Cambridge, 2003.
- Vladimir N. Vapnik:
  The Nature of Statistical Learning Theory,
  Information Science and Statistics, Springer, Heidelberg, 2000.
- John Shawe-Taylor, Nello Cristianini:
  ☞ Kernel Methods for Pattern Analysis,
  Cambridge University Press, Cambridge, 2004

Comprehensive Questions

- What are the properties of kernel functions?
- What is the kernel matrix?
- What is the kernel trick?
- How can we use kernels for string comparison?