Learning and Intelligent Systems

Non-linear prediction

Prof. Gunnar Rätsch
Biomedical Informatics (bmi.inf.ethz.ch)
(Material: Andreas Krause)
Nonlinear Classification

\[ \Phi : \mathbb{R}^2 \to \mathbb{R}^3 \]
\[ (x_1, x_2) \mapsto (z_1, z_2, z_3) := \left( x_1^2, \sqrt{2} x_1 x_2, x_2^2 \right) \]
"Kernel Trick" Summary

- Representer Theorem: \( w = \sum_{i=1}^{N} \alpha_i \Phi(x_i) \).
- Hyperplane in \( \mathcal{F} \): \( y = \text{sgn} \left( \langle w, \Phi(x) \rangle + b \right) \)
- Putting things together

\[
\begin{align*}
    f(x) &= \text{sgn} \left( \langle w, \Phi(x) \rangle + b \right) \\
    &= \text{sgn} \left( \sum_{i=1}^{N} \alpha_i \langle \Phi(x_i), \Phi(x) \rangle + b \right) \\
    &= \text{sgn} \left( \sum_{i: \alpha_i \neq 0} \alpha_i k(x_i, x) + b \right) \quad \text{sparse!}
\end{align*}
\]

- Trick: \( k(x, y) = \langle \Phi(x), \Phi(y) \rangle \), i.e. do not use \( \Phi \), but \( k \)!
“Kernel Trick”

- Express problem s.t. it only depends on inner products
- Replace inner products by kernels

Example: Perceptron

$$\begin{align*}
\min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, - \sum_{j} \alpha_j y_i x_i^T x_j\} \\
\min_{\alpha_{1:n}} \sum_{i=1}^{n} \max\{0, - \sum_{j} \alpha_j y_i k(x_i, x_j)\}
\end{align*}$$

This „trick“ is much more generally applicable
Definition: kernel functions

- Data space $X$
- A **kernel** is a function $k : X \times X \rightarrow \mathbb{R}$ satisfying
  - **Symmetry:** For any $x, x' \in X$ it must hold that $k(x, x') = k(x', x)$
  - **Positive semi-definiteness:** For any $n$, any set $S = \{x_1, \ldots, x_n\} \subseteq X$, the kernel (Gram) matrix
    
    $\begin{bmatrix}
    k(x_1, x_1) & \cdots & k(x_1, x_n) \\
    \vdots & \ddots & \vdots \\
    k(x_n, x_1) & \cdots & k(x_n, x_n)
    \end{bmatrix} \in \mathbb{R}^{n \times n}$

    must be positive semi-definite
Examples of kernels on $\mathbb{R}^d$

- Linear kernel: $k(x, x') = x^T x'$
- Polynomial kernel: $k(x, x') = (x^T x' + 1)^d$
- Gaussian (RBF, squared exp. kernel): $k(x, x') = \exp\left(-\frac{||x - x'||^2}{2h^2}\right)$
- Sigmoid (tanh) kernel: $k(x, x') = \tanh \kappa x^T x' - b$
Polynomials in higher dimensions

- Suppose we wish to use polynomial features, but our input is higher-dimensional.
- Can still use monomial features.
- **Example**: Monomials in 3 variables

\[ x = \begin{bmatrix} x_1 & x_2 \end{bmatrix} \quad \mapsto \quad \phi(x) = \begin{bmatrix} 1, x_1, x_2, x_1^2, x_2^2, x_1 x_2, x_1^3, x_2^3, x_1^2 x_2, x_1 x_2^2, x_1 x_2 x_3, \ldots \end{bmatrix} \]

\[ x = \begin{bmatrix} x_1 & x_2 & x_3 \end{bmatrix} \quad \mapsto \quad \phi = \begin{bmatrix} 1, x_1, x_2, x_3, x_1^2, x_2^2, x_3^2, x_1 x_2, x_1 x_3, x_2 x_3, x_1^3, x_2^3, x_3^3, \ldots \end{bmatrix} \]
Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Kernels work in very high-dimensional spaces. Doesn’t this lead to overfitting?
Properties of kernel functions

- Data space $X$
- A kernel is a function $k : X \times X \rightarrow \mathbb{R}$
- Can we use any function?

- $k$ must be an **inner product** in a suitable space

$\Rightarrow$ $k$ must be **symmetric**!

$$\forall x, x' \in X : \quad k(x, x') = \phi(x)^T \phi(x')$$

$$= \phi(x')^T \phi(x) = k(x', x)$$

$\Rightarrow$ Are there other properties that it must satisfy?
Kernels $\rightarrow$ semi-definite matrices

- Data space $X$ (possibly infinite)
- Kernel function $k : X \times X \rightarrow \mathbb{R}$
- Take any finite subset of data $S = \{x_1, \ldots, x_n\} \subseteq X$
- Then the kernel (gram) matrix
  \[
  K = \begin{pmatrix}
  k(x_1, x_1) & \ldots & k(x_1, x_n) \\
  \vdots & \ddots & \vdots \\
  k(x_n, x_1) & \ldots & k(x_n, x_n)
  \end{pmatrix}
  \]
  is positive semidefinite

\[
\forall x \in \mathbb{R}^n : \quad x^T K x = \left(x^T \Phi^T \Phi \right) x = v^Tv \geq 0
\]
Semi-definite matrices ➔ kernels

- Suppose the data space $X=\{1,\ldots,n\}$ is finite, and we are given a positive semidefinite matrix $K \in \mathbb{R}^{n \times n}$.

- Then we can always construct a feature map $\phi: X \rightarrow \mathbb{R}^{n}$ such that $K_{i,j} = \phi(i)^T \phi(j)$. 

Outlook: Mercer’s Theorem

Let $X$ be a compact subset of $\mathbb{R}^n$ and $k : X \times X \to \mathbb{R}^n$ a kernel function.

Then one can expand $k$ in a uniformly convergent series of bounded functions $\phi_i$ s.t.

$$k(x, x') = \sum_{i=1}^{\infty} \lambda_i \phi_i(x)\phi_i(x')$$

Can be generalized even further.
Examples of (non)-kernels

\[ k(x, x') = \sin(x) \cos(x') \]

\[ k(x, x') = x^T M x' \]
**Side note: Nearest-neighbor classifiers**

- Very simple classifier: k-Nearest Neighbor (k-NN)
- For data point $\mathbf{x}$, predict majority of labels of $k$ nearest neighbors

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i [\mathbf{x}_i \text{ among } k \text{ nearest neighbors of } \mathbf{x}]\right)$$

- How does one choose $k$?
- Thus, kernelized Perceptron can be understood as nearest neighbor prediction (with optimized weights)

$$y = \text{sign}\left(\sum_{i=1}^{n} y_i \alpha_i k(\mathbf{x}_i, \mathbf{x})\right)$$
### Comparison: k-NN vs Kernelized Perceptron

<table>
<thead>
<tr>
<th>Method</th>
<th>k-NN</th>
<th>Kernelized Perceptron</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Advantages</strong></td>
<td>No training necessary</td>
<td>Optimized weights can lead to improved performance</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Can capture „global trends“ with suitable kernels</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Depends on „wrongly classified“ examples only</td>
</tr>
<tr>
<td><strong>Disadvantages</strong></td>
<td>Depends on all data inefficient</td>
<td>Training requires optimization</td>
</tr>
</tbody>
</table>
Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:
  - Sequence kernels
  - Graph kernels
  - Diffusion kernels
  - Kernels on probability distributions
  - ...
Kernel engineering (composition rules)

- Suppose we have two kernels

\[ k_1 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \quad k_2 : \mathcal{X} \times \mathcal{X} \rightarrow \mathbb{R} \]

defined on data space \( \mathcal{X} \)

- Then the following functions are valid kernels:

\[ k(x, x') = k_1(x, x') + k_2(x, x') \]
\[ k(x, x') = k_1(x, x') \cdot k_2(x, x') \]
\[ k(x, x') = c \cdot k_1(x, x') \text{ for } c > 0 \]
\[ k(x, x') = f(k_1(x, x')) \]

where \( f \) is a polynomial with positive coefficients or the exponential function
Example: Modeling pairwise data

- May want to use kernels to model pairwise data
  (users x products; genes x patients; ...)

![Graph 1](image1.png)

![Graph 2](image2.png)
Example: Kernels for Sequence Data

Given: Potential acceptor splice sites

Goal: Rule that distinguishes true from false ones

- e.g. exploit that exons have higher GC content
- or
- that certain motifs appear near splice site
Example: Kernels for Sequence Data

- Given: Potential acceptor splice sites
- Goal: Rule that distinguishes true from false ones

More realistic problem!? 
- Not linearly separable!
- Need nonlinear separation!?
- Need more features!?  
  \[ \Rightarrow \text{Need new kernel!} \]
Considerations for Creating new Kernels

At least two ways to get to a kernel
- Construct $\Phi$ and think about efficient ways to compute $\langle \Phi(x), \Phi(y) \rangle$
- Construct similarity measure, show positiveness and think about what it means

What is a good kernel?
- It should be mathematically valid (symmetric, pos. def.),
- fast to compute and
- adapted to the problem (yields good performance).

Approaches:
- Manually generate a few strong features
  - Requires background knowledge
  - Nonlinear decisions often beneficial
- Include many potentially useful weak features
  - Requires more training examples

Best in practice: Combination of both
Spectrum Kernel for Sequences

- **General idea** [Leslie et al., 2002]
  - For each $k$-mer $s \in \Sigma^k$, the coordinate indexed by $s$ will be the number of times $s$ occurs in sequence $x$.
  - Then the $k$-spectrum feature map is
    \[
    \Phi_k^{\text{Spectrum}}(x) = (\phi_s(x))_{s \in \Sigma^k}
    \]
  - Here $\phi_s(x)$ is the \# occurrences of $s$ in $x$.
  - The spectrum kernel is now the inner product in the feature space defined by this map:
    \[
    k^{\text{Spectrum}}(x, x') = \langle \Phi_k^{\text{Spectrum}}(x), \Phi_k^{\text{Spectrum}}(x') \rangle
    \]
  - Dimensionality: Exponential in $k$: $|\Sigma|^k$
Spectrum Kernel for Sequences

- **Principle**
  - Spectrum kernel: Count shared $k$-mers

Protein A: ILVFMC

Protein B: WLVFQC

- $\Phi(x)$ has only very few non-zero dimensions
  $\implies$ Efficient kernel computations possible

Common 1-mers
L, V, F, C

Common 2-mers
LV
VF

Common 3-mers
LVF
Weighted Degree Kernel for Sequences

- Equivalent to a mixture of spectrum kernels (up to order $K$) at every position for appropriately chosen $\beta$'s:

$$k(x_i, x_j) = \sum_{k=1}^{K} \sum_{l=1}^{L-k+1} \beta_k k_k^{\text{Spectrum}}(u_{l:l+k}(x_i), u_{l:l+k}(x_j))$$

where $\beta_k = \frac{K-k+1}{\sum_k (K-k+1)} = 2 \frac{K-k+1}{k(k+1)}$.

- Can be equivalently computed by

$$k(x_i, x_j) = \sum_{k=1}^{K} \sum_{l=1}^{L-k+1} \beta_k I(u_{l:l+k}(x_i) = u_{l:l+k}(x_j))$$
Weighted Degree Kernel for Sequences

- **Without shifts:** Compare two sequences by identifying the largest matching blocks:

  \[ k(s_1, s_2) = w_7 + w_1 + w_2 + w_2 + w_3 \]

  where a matching block of length \( k \) implies many shorter matches:

  \[ w_k = \sum_{j=1}^{\min(k, K)} \beta_j \cdot (k - j + 1). \]

  Complexity of kernel computation: \( O(n) \)
  Complexity of dot product: \( O(n^* |\Sigma|^k) \)
Example: Sequence Kernel Comparison

- Linear kernel on GC-content features
- Spectrum kernel
- Weighted degree kernel
- Weighted degree kernel with shifts
Parametric vs nonparametric learning

- **Parametric** models have finite set of parameters
  - **Example**: Linear regression, linear Perceptron, ...

- **Nonparametric** models grow in complexity with the size of the data
  - Potentially much more expressive
  - But also more computationally complex – **Why?**
  - **Example**: Kernelized Perceptron, k-NN, ...

- Kernels provide a principled way of deriving non-parametric models from parametric ones
Example: Kernelized Linear Regression

- Original (parametric) linear optimization problem

\[ w^* = \arg\min_w \sum_i (w^T x_i - y_i)^2 + \lambda \|w\|^2 \]

- Similar as in perceptron, optimal \( w^* \) lies in span of data:

\[ w^* = \sum_i \alpha_i x_i \]
Kernelized Linear Regression (KLR)

\[
\min_{\alpha_1, \ldots, \alpha_n} \sum_{i=1}^{n} \left( \sum_{j=1}^{n} \alpha_j k(x_i, x_j) - y_i \right)^2 + \lambda \alpha^T K \alpha
\]

\[
K = \begin{pmatrix}
    k(x_1, x_1) & \ldots & k(x_1, x_n) \\
    \vdots & \ddots & \vdots \\
    k(x_n, x_1) & \ldots & k(x_n, x_n)
\end{pmatrix}
\]
Learning & Predicting with KLR

- **Learning**: Solve least squares problem
  
  $$\min_{\alpha} \|\alpha^T K - y\|_2^2 + \lambda \alpha^T K \alpha$$

  Closed-form solution: $$\alpha^* = (K + \lambda I)^{-1} y$$

- **Prediction**: For data point \(x\) predict response \(y\) as
  
  $$y = \sum_{i=1}^{n} \alpha_i^* k(x_i, x)$$
KLR for the linear kernel

What if \( k(x, x') = x^T x' \)?
Application: semi-parametric regression

- Often, parametric models are too "rigid", and non-parametric models fail to extrapolate
- **Solution**: Use additive combination of linear and non-linear kernel function

\[
k(x, x') = c_1 \exp(||x - x'||^2_2/h^2) + c_2 x^T x'
\]
Example

Training data

Function we wish to learn
Example fits
Choosing kernels

- For a given kernel, how should we choose parameters?
  - Cross-validation!

- How should we select suitable kernels?
  - Domain knowledge (dependent on data type)
  - «Brute force» (or heuristic) search
  - Use cross-validation

- Learning kernels
  - Much research on automatically selecting good kernels
    (Multiple Kernel Learning; Hyperkernels; etc.)
What about overfitting?

• Kernels map to (very) high-dimensional spaces.
• Why don’t we overfit??

• Overfitting can of course happen
  (if we choose poor parameters)
• Can combat overfitting by regularization
  • This is already built into kernelized linear regression
    (and SVMs), but not the kernelized Perceptron

\[
\min_{\alpha} \| \alpha^T K - y \|^2_2 + \lambda \alpha^T K \alpha
\]
What you need to know

- Kernels are
  - (efficient, implicit) inner products
  - Positive (semi-)definite functions
  - Many examples (linear, polynomial, Gaussian/RBF, ...)

- The „Kernel trick“
  - Reformulate learning algorithm so that inner products appear
  - Replace inner products by kernels

- K-Nearest Neighbor classifier (and relation to Perceptron)

- How to choose kernels (kernel engineering etc.)

- **Applications**: Kernelized Perceptron / SVM; kernelized linear regression
Supervised learning big picture so far

Kernelized Regression

Kernels

Ridge Regression

L2-regularizer

Least squares Regression

Loss funct.

Perceptron

Loss funct.

Kernelized Perceptron

Kernels

k-NN

"Special case"

Kernelized SVM

Kernels

Linear SVM

Loss funct.