Introduction to Machine Learning

Non-linear prediction with kernels

Prof. Andreas Krause
Learning and Adaptive Systems (las.ethz.ch)
Solving non-linear classification tasks

- How can we find nonlinear classification boundaries?
- Similar as in regression, can use non-linear transformations of the feature vectors, followed by linear classification.
Avoiding the feature explosion

Need \(O(d^k)\) dimensions to represent (multivariate) polynomials of degree \(k\) on \(d\) features

**Example:** \(d=10000, k=2\) \(\Rightarrow\) Need \(~100M\) dimensions

In the following, we can see how we can efficiently implicitly operate in such high-dimensional feature spaces (i.e., without ever explicitly computing the transformation)
The „Kernel Trick“

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

- Example: Perceptron

\[
\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j (x_i^T x_j)\}
\]

\[
\hat{\alpha} = \arg\min_{\alpha_{1:n}} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -\sum_{j=1}^{n} \alpha_j y_i y_j k(x_i, x_j)\}
\]

- Will see further examples later
Kernelized Perceptron

Training
- Initialize \( \alpha_1 = \cdots = \alpha_n = 0 \)
- For \( t=1,2,... \)
  - Pick data point \((x_i, y_i)\) uniformly at random
  - Predict
    \[
    \hat{y} = \text{sign} \left( \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_i) \right)
    \]
  - If \( \hat{y} \neq y_i \) set \( \alpha_i \leftarrow \alpha_i + \eta_t \)

Prediction
- For new point \( x \), predict
  \[
  \hat{y} = \text{sign} \left( \sum_{j=1}^{n} \alpha_j y_j k(x_j, x) \right)
  \]
Questions

- What are valid kernels?
- How can we select a good kernel for our problem?
- Can we use kernels beyond the perceptron?
- Kernels work in very high-dimensional spaces. Doesn’t this lead to overfitting?
Definition: kernel functions

- Data space $X$
- A kernel is a function $k : X \times X \rightarrow \mathbb{R}$ satisfying
  
  1) Symmetry: For any $x, x' \in X$ it must hold that
     
     $$k(x, x') = k(x', x)$$

  2) Positive semi-definiteness: For any $n$, any set
     
     $$S = \{x_1, \ldots, x_n\} \subseteq X$$
     
     the kernel (Gram) matrix
     
     $$K = \begin{pmatrix}
     k(x_1, x_1) & \ldots & k(x_1, x_n) \\
     \vdots & \ddots & \vdots \\
     k(x_n, x_1) & \ldots & k(x_n, x_n)
     \end{pmatrix}$$

     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)**: 
  \[ k(x, x') = \exp\left(-\frac{||x - x'||^2}{2h^2}\right) \]

- **Laplacian kernel**: 
  \[ k(x, x') = \exp\left(-\frac{||x - x'||_1}{h}\right) \]
Examples of (non)-kernels

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

Not symmetric: \( k(x, x') \neq k(x', x) \) e.g. for \( x = 0, x' = \frac{\pi}{2} \)

\[ \Rightarrow \text{not a valid kernel on } \mathbb{R} \]

\[ k(x, x') = x^T M x' \]

\[ x, x' \in \mathbb{R}^d, \ M \in \mathbb{R}^{d \times d} \]

If \( M \) symmetric:
\[ k(x, x') = x^T M x' = x^T M^T x' = x^T M x = k(x, x') \]

If \( M \) not symmetric, \( k \) in general not sym.

Claim: \( k \) s.p.d. \( \Leftrightarrow \) \( M \) s.p.d.

If \( M \) not pos-def:
E.g. in 1-dim: \( M = -I \): \( x^T M x = -x^2 < 0 \) if \( x \neq 0 \)

If \( M \) positive-definit, \( M = U D^{\frac{1}{2}} D^{\frac{1}{2}} U^T = V^T V \) for \( V = (U D^{\frac{1}{2}})^T \)

then: \( k(x, x') = x^T M x' = x^T V^T V x = (V x)^T (V x') = \Phi(x)^T \Phi(x') \)

for \( \Phi(x) = V x \)
Effect of kernel on function class

Given kernel $k$, predictors (for kernelized classification) have the form

$$\hat{y} = \text{sign}\left(\sum_{j=1}^{n} \alpha_j y_j k(x_j, x)\right)$$
Example: Gaussian kernel

\[ k(x, x') = \exp\left(-\frac{||x - x'||^2}{2h^2}\right) \]

\[ f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \]

Bandwidth h=.3

Bandwidth h=.1
Example: Laplace/Exponential kernel

\[ k(x, x') = \exp\left(-\|x - x'\|_1 / h\right) \]

\[ f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x) \]

Bandwidth \( h=1 \)  

Bandwidth \( h=.3 \)
Demo: Effect on decision boundary
Kernels beyond $\mathbb{R}^d$

Can define kernels on a variety of objects:

- Sequence kernels
- Graph kernels
- Diffusion kernels
- Kernels on probability distributions
- ...
Can define a kernel for measuring similarity between graphs by comparing random walks on both graphs (not further defined here)

[Borgwardt et al.]
Example: Diffusion kernels on graphs

\[ K = \exp(-\beta L) \]

- Can measure similarity among nodes in a graph via diffusion kernels (not defined here)
Kernel engineering (composition rules)

Suppose we have two kernels \( k_1(x) = \phi(x)^T \phi(x') \)

\( k_1 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \)

\( k_2 : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \)

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

Then the following functions are valid kernels:

\[
\begin{align*}
k_1(x, x') &= k_1(x, x') + k_2(x, x') \\
k_2(x, x') &= k_1(x, x') k_2(x, x') \\
k_3(x, x') &= c k_1(x, x') \quad \text{for } c > 0 \\
k_4(x, x') &= f(k_1(x, x'))
\end{align*}
\]

where \( f \) is a polynomial with positive coefficients or the exponential function
Example: ANOVA kernel

\[ h(x, x') = \sum_{j=1}^{d} k_j(x_j, x'_j) \]

where \( x, x' \in \mathbb{R}^d \)

\[ k : \mathbb{R}^d \times \mathbb{R}^d \rightarrow \mathbb{R} \]

\[ k_j : \mathbb{R} \times \mathbb{R} \rightarrow \mathbb{R} \text{ kernel} \]

\[ \Rightarrow \text{ } k \text{ is a valid kernel} \]

What functions does \( k \) model?

\[ f(x) = \sum_{i=1}^{n} a_i y_i k(x^{(i)}, x) \]

\[ = \sum_{i=1}^{n} a_i y_i \sum_{j=1}^{d} k_j(x^{(i)}_j, x_j) \]

\[ = \sum_{j=1}^{d} \left( \sum_{i=1}^{n} a_i y_i k_j(x^{(i)}_j, x_j) \right) f_j(x_j) \]
Example: Modeling pairwise data

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

\[ k((x_i, z_i), (x'_i, z'_i)) = k_x(x_i, x'_i) \cdot k_z(z_i, z'_i) \]

\[ k((x_i, z_i), (x'_i, z'_i)) = k_x(x_i, x'_i) + k_z(z_i, z'_i) \]
Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

Next questions:
- What kind of predictors / decision boundaries do kernel methods entail?
- Can we use the kernel trick beyond the perceptron?
Kernels as *similarity functions*

- Recall Perceptron (and SVM) classification rule:

  \[ y = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) \right) \approx \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i \left[ x_i \text{ "close to" } x \right] \right) \]

- Consider Gaussian kernel \( k(x, x') = \exp\left( -\frac{||x - x'||^2}{\gamma^2} \right) \)
Side note: Nearest-neighbor classifiers

For data point $x$, predict majority of labels of $k$ nearest neighbors

$$y = \text{sign} \left( \sum_{i=1}^{n} y_i [x_i \text{ among } k \text{ nearest neighbors of } x] \right)$$
Demo: k-NN
Nearest-neighbor classifiers

For data point $x$, predict majority of labels of $k$ nearest neighbors

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

How to choose $k$?

Cross-validation! 😊
K-NN vs. Kernel Perceptron

- **k-Nearest Neighbor:**

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

- **Kernel Perceptron:**

\[ y = \text{sign} \left( \sum_{i=1}^{n} y_i \alpha_i k(x_i, x) \right) \]
## 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>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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
Where are we?

- We’ve seen how to kernelize the perceptron
- Discussed properties of kernels, and seen examples

Next question:
- Can we use the kernel trick beyond the perceptron?
Kernelized SVM

The support vector machine

\[ \hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \max \{0, 1 - y_i w^T x_i \} + \lambda \| w \|_2^2 \]

can also be kernelized
How to kernelize the objective?

\[
\hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \max \{0, 1 - y_i w^T x_i\} + \lambda \|w\|_2^2
\]

\[(k) = \max \{0, 1 - y_i \left( \frac{1}{n} \sum_{j=1}^{n} \alpha_j y_j x_j \right)^T x_i \}
\]

\[
= \max \{0, 1 - y_i \sum_{j=1}^{n} \alpha_j y_j k(x_j, x_i) \}
\]

\[
= \max \{0, 1 - y_i \alpha^T k(x_i) \}
\]

\[
\alpha = \left[ \begin{array}{c} \alpha_1 \kappa(x_1, x_1) \\ \vdots \\ \alpha_n \kappa(x_n, x_n) \end{array} \right]^T, \quad k_i = \left[ y_1 \kappa(x_i, x_1), \ldots, y_n \kappa(x_i, x_n) \right]^T
\]
How to kernelize the regularizer?

\[ \hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\} + \lambda \|w\|^2_2 \]

\( (4) \)

\[ = \mathbf{D}_y^T \mathbf{D}_y \mathbf{k} \mathbf{D}_y \mathbf{a} \]
Learning & prediction with kernel classifier

- **Learning**: Solve the problem

  \[
  \arg \min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, -y_i \alpha^T k_i\} \quad \text{or:}
  \]

  \[
  \text{Perceptron:} \quad \arg \min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T k_i\}
  \]

  \[
  \text{SVM:} \quad \arg \min_{\alpha} \frac{1}{n} \sum_{i=1}^{n} \max\{0, 1 - y_i \alpha^T k_i\} + \lambda \alpha^T D_y K D_y \alpha
  \]

  \[
  k_i = [y_1 k(x_i, x_1), \ldots, y_n k(x_i, x_n)]
  \]

- **Prediction**: For data point \(x\) predict label \(y\) as

  \[
  \hat{y} = \text{sign} \left( \sum_{i=1}^{n} \alpha_i y_i k(x_i, x) \right)
  \]
Demo: Kernelized SVM
Kernelized Linear Regression

- From linear to **nonlinear regression**:

- Can also kernelize linear regression

- Predictor has the form

\[
f(x) = \sum_{i=1}^{n} \alpha_i k(x_i, x)
\]
Example: Kernelized linear regression

- Original *(parametric)* linear optimization problem

\[ \hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \left( w^T x_i - y_i \right)^2 + \lambda \| w \|_2^2 \]

- Similar as in perceptron, optimal lies in span of data:

\[ \hat{w} = \sum_{i=1}^{n} \alpha_i x_i \]
Kernelizing linear regression

\[
\hat{w} = \arg \min_w \frac{1}{n} \sum_{i=1}^{n} \left( \mathbf{w}^T \mathbf{x}_i - y_i \right)^2 + \lambda \| \mathbf{w} \|^2_2
\]

\[
\hat{w} = \sum_{i=1}^{n} \alpha_i \mathbf{x}_i
\]

\[\begin{align*}
(4) &= \left( \sum_{j=1}^{n} \alpha_j \left( \mathbf{x}_j \cdot \mathbf{x}_i \right) - y_i \right)^2 \\
&= \left( \sum_{j=1}^{n} \alpha_j \left( \mathbf{x}_j \cdot \mathbf{x}_i \right) \right)^2 - 2 \alpha_i \mathbf{x}_i \cdot \mathbf{k}(\mathbf{x}_i) + \mathbf{k}(\mathbf{x}_i) \cdot \mathbf{k}(\mathbf{x}_i)
\end{align*}\]
**Learning & Predicting with KLR**

- **Learning**: Solve least squares problem

\[
\hat{\alpha} = \arg \min_{\alpha} \frac{1}{n} \left\| \alpha^T K - y \right\|^2_2 + \lambda \alpha^T K \alpha
\]

Closed-form solution: \[\hat{\alpha} = (K + n\lambda I)^{-1} y\]

- **Prediction**: For data point \( x \) predict response \( y \) as

\[
\hat{y} = \sum_{i=1}^{n} \hat{\alpha}_i k(x_i, x)
\]