Learning and Intelligent Systems

Non-linear prediction

Gunnar Rätsch
Learning and Adaptive Systems (las.ethz.ch)
(Material: Andreas Krause)
Linear classifiers

Data set

\((x_1, y_1), \ldots, (x_n, y_n)\)
The Perceptron problem

- Solve
  \[ \mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_{i=1}^{n} \ell_P(\mathbf{w}; y_i, \mathbf{x}_i) \]

  where \( \ell_P(\mathbf{w}; y_i, \mathbf{x}_i) = \max(0, -y_i \mathbf{w}^T \mathbf{x}_i) \)

- Optimize via Stochastic Gradient Descent
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
Linear functions for polynomials

- We can fit non-linear functions via linear regression, using nonlinear features of our data (basis functions)

\[ f(x) = \sum_{i=1}^{d} w_i \phi_i(x) \]

- For example: polynomials of degree m (in 1-D)

\[ f(x) = \sum_{i=0}^{m} w_i x^i \]
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
Example

\[ \Phi : \mathbb{R}^2 \rightarrow \mathbb{R}^3 \]

\[
(x_1, x_2) \mapsto (z_1, z_2, z_3) := (x_1^2, \sqrt{2} x_1 x_2, x_2^2)
\]
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 $\sim 100M$ dimensions

- In the following, we can see how we can efficiently implicitly operate in such high-dimensional feature spaces
Revisiting the Perceptron

**Fundamental insight:** Optimal hyperplane can be expressed in span of the data

\[ \mathbf{w}^* = \sum_{i=1}^{n} (\alpha_i y_i) \mathbf{x}_i \]

**(Handwavy) proof:** (Stochastic) gradient descent starting from 0 constructs such a representation

\[ \mathbf{w}_{t+1} = \mathbf{w}_t + \eta_t y \mathbf{x} \quad \text{if} \quad y \mathbf{w}^T \mathbf{x} < 0 \]

**More abstract proof:** Follows from the "representer theorem" (not discussed here)
 Reformulating the Perceptron

\[
\min_{w \in \mathbb{R}^d} \sum_{i=1}^n \max \left( 0, -y_i w^T x_i \right)
\]

Ansatz: \( w = \sum_{\delta=1}^n a_\delta y_\delta x_\delta \)

\[
\begin{align*}
\min_{a_1, a_2, \ldots, a_n} & \sum_{i=1}^n \max \left( 0, -y_i \left( \sum_{\delta=1}^n a_\delta y_\delta x_\delta \right)^T x_i \right) \\
\end{align*}
\]

\[
\begin{align*}
\min_{a_1, a_2, \ldots, a_n} & \sum_{i=1}^n \max \left( 0, -y_i \sum_{\delta=1}^n a_\delta y_\delta \left( x_\delta^T x_i \right) \right) \\
\end{align*}
\]
Advantage of reformulation

\[
\min_{\alpha_1:n} \sum_{i=1}^{n} \max \{ 0, - \sum_j \alpha_j y_i x_i^T x_j \}
\]

**Key observation**: Objective only depends on inner products of data points

Thus, we can *implicitly* work in high-dimensional spaces, as long as we can do inner products efficiently

\[
x \leftrightarrow \phi(x) \\
x^T x' \leftrightarrow \phi(x)^T \phi(x') = k(x, x')
\]
“Kernels = efficient inner products“

- Often times, $k(x, x')$ can be computed much more efficiently than $\phi(x)^T \phi(x')$

- Simple example: Polynomial kernel of degree 2
Polynomial kernels (degree 2)

Suppose $\mathbf{x} = [x_1, \ldots, x_d]^T$ and $\mathbf{y} = [y_1, \ldots, y_d]^T$

Then $(\mathbf{x}^T \mathbf{y})^2 = \left( \sum_{i=1}^{d} x_i y_i \right)^2 \leq \text{can compute in } O(d) \text{ ops}$
Polynomial kernels: Fixed degree

- The kernel $k(x, x') = (x^T x')^m$ implicitly represents all monomials of degree $m$

  Monomials of vars $x_1, \ldots, x_d$ of order $m$ are all products of $m$ vars, (with repetition)

  \[
  \{ x_1^m, x_2^m, \ldots, x_d^m, x_1^{m-1} x_2, \ldots, x_d^{m-1}, x_3, \ldots \} 
  \]

- How can we get monomials up to order $m$?
Polynomial kernels

- The polynomial kernel $k(x, x') = (1 + x^T x')^m$ implicitly represents all monomials of up to degree $m$.

\[ k(x, x') = (1 + x^T x')^m \]

- Representing the monomials (and computing inner product explicitly) is \textit{exponential} in $m$!!
The „Kernel Trick“

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

Example: Perceptron
The „Kernel Trick“

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

Example: Perceptron

$$\min_{\alpha_1:n} \sum_{i=1}^{n} \max\{0, - \sum_j \alpha_j y_i x_i^T x_j \}$$

$$\min_{\alpha_1:n} \sum_{i=1}^{n} \max\{0, - \sum_j \alpha_j y_i k(x_i, x_j) \}$$

- This „trick“ is much more generally applicable
Derivation: Kernelized Perceptron
Kernelized Perceptron

- 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 $\alpha_i \leftarrow \alpha_i + \eta_t$
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 $k : X \times X \to \mathbb{R}$

- Can we use any function?

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

- $k$ must be symmetric!

$$\forall x, x' \in X : \quad k(x, x') = \phi(x)^\top \phi(x') = \phi(x')^\top \phi(x) = k(x', x)$$

- Are there other properties that it must satisfy?
Positive semi-definite matrices

Symmetric matrix $M \in \mathbb{R}^{n \times n}$ is positive semidefinite iff

(i): $\forall x \in \mathbb{R}^n \quad x^T M x \geq 0$

(ii): all eigenvalues of $M$ (which are real due to symmetry) are $\geq 0$
Kernels \( \Rightarrow \) semi-definite matrices

- Data space \( X \)
- 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

\[
K = \begin{pmatrix}
k(x_1, x_1) & \cdots & k(x_1, x_n) \\
\vdots & \ddots & \vdots \\
k(x_n, x_1) & \cdots & k(x_n, x_n)
\end{pmatrix} = \begin{pmatrix}
\phi(x_1)^T \phi(x_1) & \cdots & \phi(x_1)^T \phi(x_n) \\
\vdots & \ddots & \vdots \\
\phi(x_n)^T \phi(x_1) & \cdots & \phi(x_n)^T \phi(x_n)
\end{pmatrix}
\]

is positive semidefinite

\[
K = \Phi^T \Phi \quad \text{where} \quad \Phi = \begin{bmatrix}
\phi(x_1) \\
\vdots \\
\phi(x_n)
\end{bmatrix}
\]

\[
\forall x \in \mathbb{R}^n : x^T K x = (x^T \Phi^T)(\Phi \Phi^T)x = v^T v \geq 0
\]
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 $k : X \times X \rightarrow \mathbb{R}^n$ 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
**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
  \[
  K = \begin{pmatrix}
  k(x_1, x_1) & \ldots & k(x_1, x_n) \\
  \vdots & \ddots & \vdots \\
  k(x_n, x_1) & \ldots & k(x_n, x_n)
  \end{pmatrix}
  \]

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): $k(x, x') = \exp(-||x - x'||^2/2h^2)$
- Sigmoid (tanh) kernel: $k(x, x') = \tanh \kappa x^T x' - b$
Examples of (non)-kernels

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

\[ k(x, x') = x^T M x' \]
Kernels as *similarity functions*

- Recall Perceptron (and SVM) classification rule:

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

- Consider Gaussian kernel \( k(x, x') = \exp\left(-\frac{||x - x'||^2}{h^2}\right) \)
Recall: Kernelized Perceptron

- 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 $\alpha_i \leftarrow \alpha_i + \eta_t$
Side note: Nearest-neighbor classifiers

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

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

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

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

- 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:**

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

  \(k = \text{Gaussian kernel}\)

  \[
  k = \exp\left(-\frac{\text{dist}^2}{h^2}\right)
  \]

  ‘bandwidth’

  \(\alpha_i \geq 0\) if \(x_i \text{ and } x\) are ‘close’

  \(\alpha_i \leq 1\) if \(x_i \text{ and } x\) are ‘far’
Side note: Nearest-neighbor classifiers

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

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

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

$$y = \text{sign} \left( \sum_{i=1}^{n} y_i \alpha_i k(x_i, x) \right)$$
## 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>Advantages</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>Disadvantages</td>
<td>Depends on all data ➔ inefficient</td>
<td>Training requires optimization</td>
</tr>
</tbody>
</table>
Preview: Kernelized Linear Regression

- From linear to **nonlinear regression**:

- Will see how we can kernelize linear regression

- Predictor will have the form

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

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

can also be kernelized

This requires convex duality (discussed in Advanced Machine Learning class).
Kernels beyond $\mathbb{R}^d$

- Can define kernels on a variety of objects:
  - Sequence kernels
  - Graph kernels
  - Diffusion kernels
  - Kernels on probability distributions
  - …
Example: Graph kernels

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

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

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

Then the following functions are valid kernels:

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

where \( f \) is a polynomial with positive coefficients or
the exponential function