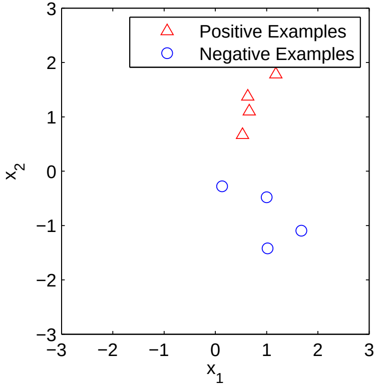
Cognitive Algorithms
Lecture 5

Kernel Methods

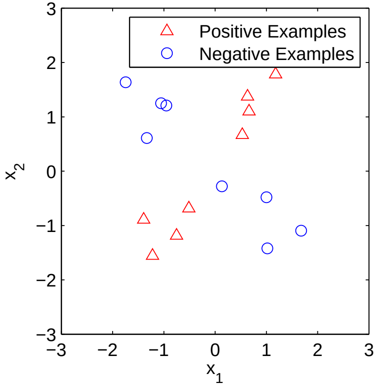
Klaus-Robert Müller, **Wojciech Samek**
Stephanie Brandl

Berlin Institute of Technology
Dept. Machine Learning

Categorization Revisited

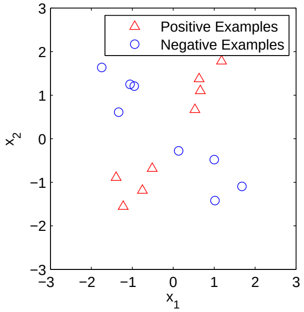


Some data is linearly separable



Other data is not separable

Categorization Revisited



Linearly non-separable data

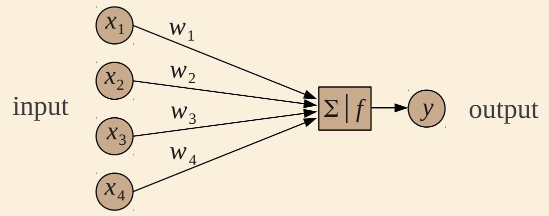
Multilayer Perceptrons (MLP) can learn linearly non separable problems

Kernel methods can *implicitly* model arbitrarily complex hidden layers

Overview

- Very brief introduction to Multilayer Perceptrons
- Overview of Kernel methods
- Non-linear Regression with Kernel Ridge Regression
- Generalization and Model Selection

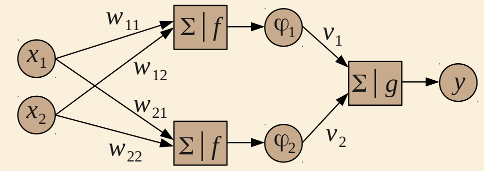
Recall the Perceptron



$$y = f(x_1 w_1 + x_2 w_2 + x_3 w_3 + x_4 w_4 + b) = f(\mathbf{w} \cdot \mathbf{x} + b)$$

- Realises linear function
- Trained by stochastic gradient descent

The Multilayer Perceptron



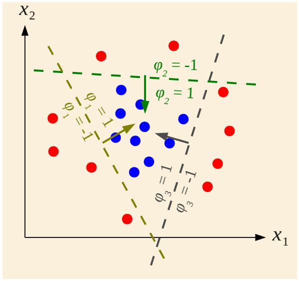
$$\varphi_1 = f(x_1 w_{11} + x_2 w_{12} + b_1)$$
$$\varphi_2 = f(x_1 w_{21} + x_2 w_{22} + b_2)$$
$$y = g(\varphi_1 v_1 + \varphi_2 v_2 + c)$$

Matrix form:
$$y = g(V \cdot f(W \cdot x))$$

- Typically uses the tangens hyperbolicus, $f(\cdot) = \tanh(\cdot)$
- Error function $\sum_t [y_t - g(V \cdot f(W \cdot \mathbf{x}_t))]^2$ has local minima
- Trained by stochastic gradient descent (compute gradient by backpropagation)

The Multilayer Perceptron

Each intermediate variable $\varphi_1, \varphi_2, \varphi_3$ is the output of a perceptron that encodes certain aspect of the input. These intermediate variables can be combined in order to form the desired output.

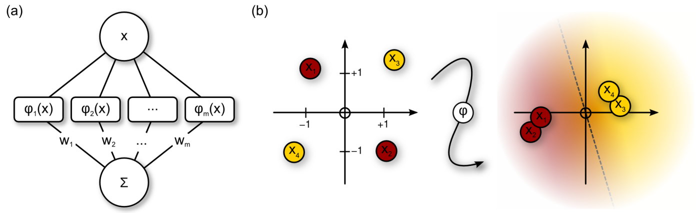


$$y = \text{sign}(\varphi_1 + \varphi_2 + \varphi_3 - 2)$$

Universal Approximation Theorem

With enough intermediate variables, a network with a single hidden layer can approximate any reasonable function of the input. [Cybenko, 1989; Hornik, 1991]

Kernelizing linear methods



[Jäkel et al., 2009]

1. Map the data into a (high dimensional) feature space, $\mathbf{x} \mapsto \varphi(\mathbf{x})$
2. Look for linear relations in the feature space
   - Work in that space by considering scalar product of data points, $k(\mathbf{x}_i, \mathbf{x}_j) = \langle \varphi(\mathbf{x}_i), \varphi(\mathbf{x}_j) \rangle$
   - Many linear models have a *dual representation* that only uses scalars products between the data points
   - k is called the *kernel function*

Kernel Trick

Kernel Trick

Any algorithm for vectorial data that can be expressed only in terms of scalar products between vectors can be performed implicitly in the feature space associated with any kernel, by replacing each scalar product by a kernel evaluation.

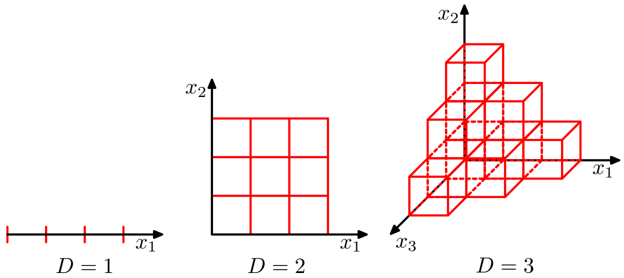
We can kernelize: Linear Discriminant Analysis, the Perceptron, Principal Component Analysis, Ridge Regression, . . .

Kernels are also used to handle symbolic objects, such as sequences, text data or chemical structures.

But: the curse of dimensionality

A big problem with high dimensional features spaces

When the dimensionality increases, the volume of the space increases so fast that the available data becomes sparse



[Bishop, 2007]

The amount of data needed for a reliable result often grows exponentially with the dimensionality

What about the curse of dimensionality?

Representer Theorem

In a regularized learning problem, the optimal weights $\mathbf{w}$ in feature space are a linear combination of the training examples in feature space $\varphi(\mathbf{x}_i)$:

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i).$$

Mathematical intuition: Optimal solution has to lie in the space spanned by the data.

Kernels as Similarity Measures

$$\mathbf{w} = \sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i).$$

Prediction $\hat{y}$ for a new data point $\mathbf{x}_{new}$ can be expressed as a linear combination of similarities to the training points $\mathbf{x}_i$:

$$\begin{aligned}\hat{y} =& f(\mathbf{x}_{\text{new}}) = \mathbf{w}^T \varphi(\mathbf{x}_{\text{new}}) \\ =& \left(\sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i) \right)^T \varphi(\mathbf{x}_{\text{new}}) = \sum_{i=1}^{N} \alpha_i \varphi(\mathbf{x}_i)^T \varphi(\mathbf{x}_{\text{new}}) \\ =& \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_i, \mathbf{x}_{\text{new}})\end{aligned}$$

where $i \in \{1, \dots, N\}$ are the indices of previously seen data points

Kernels as Similarity Measures

$$\hat{y} = \sum_{i=1}^{N} \alpha_i k(\mathbf{x}_{\text{new}}, \mathbf{x}_i)$$

Kernel methods are *memory-based* methods:

- store the entire training set
- define similarity of data points by kernel function

$$k(\cdot, \cdot) : \mathbb{R}^d \times \mathbb{R}^d \mapsto \mathbb{R}$$

- new predictions require comparision with previously learned examples

When do humans percieve stimuli as similar?

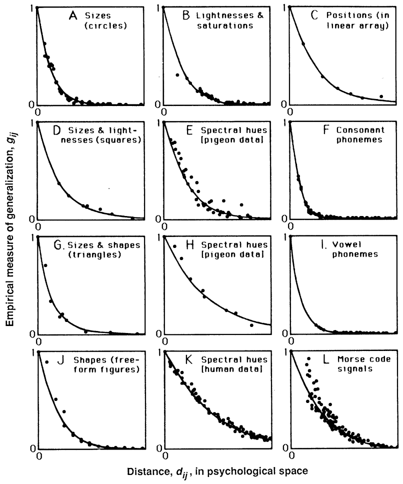


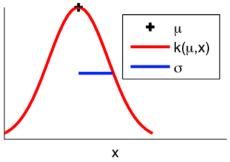
Fig. 1. Twelve gradients of generalization. Measures of generalization between stimuli are plotted against distances between corresponding points in the psychological space that renders the relation most nearly monotonic.

Generalization gradients are obtained by conditioning on a certain stimulus and measuring the response to related, but different stimuli

$\rightarrow$ **Perceptual Similarity** of new data $\mathbf{x}$ decays exponentially with distance from prototype μ [Shepard, 1987]

Shepard's Law of Universal Generalization

→ **Perceptual Similarity** of new data **x** decays exponentially with distance from prototype μ

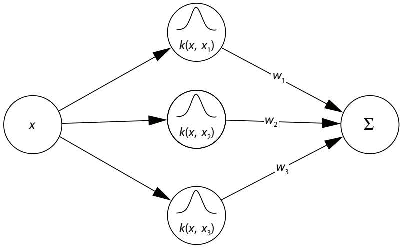


Gaussian Kernel in $\mathbb{R}^1$

Gaussian Kernel

$$k(\mu, \mathbf{x}) = e^{\frac{-(\mu - \mathbf{x})^2}{\sigma^2}} \qquad (1)$$

Remember the Multilayer Perceptrons



Imagine one hidden layer with neurons storing copies of all *training data* and the similarity kernel (eq. 1) as a gaussian of width σ on each data point

Any non-linear function (e.g. category borders) can be implemented as a linear combination of similarities to all previously seen data points

Summary Kernel Trick

The kernel trick is one of the most powerful machine learning tools [Aizerman et al., 1964; Müller et al., 2001; Schölkopf and Smola, 2002; Shawe-Taylor and Cristianini, 2004]

- Psychological intuition [Jäkel, 2007]:
  New predictions of arbitrarily complex categorizations require **only comparisons with previously learned examples**
- Mathematical intuition:
  Optimal solution has to lie in the space spanned by the data

Some Popular Kernel Functions

Linear Kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^\top \mathbf{x}_j$$

Polynomial Kernel

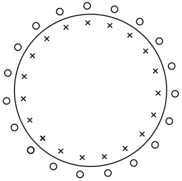
$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^\top \mathbf{x}_j + c)^p$$

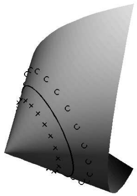
Gaussian Kernel

$$k(\mathbf{x}_i, \mathbf{x}_j) = e^{||\mathbf{x}_i - \mathbf{x}_j||^2 / -2\sigma^2}$$

A simple example

$$\varphi : \mathbf{x} = (x_1, x_2)^\top \mapsto \varphi(\mathbf{x}) = (x_1^2, x_2^2, \sqrt{2}x_1x_2)^\top$$





The corresponding kernel:

$$\begin{aligned} k(\mathbf{x}, \mathbf{y}) &= \varphi(\mathbf{x})^\top \cdot \varphi(\mathbf{y}) \\ &= (x_1^2, x_2^2, \sqrt{2}x_1x_2)^\top \cdot (y_1^2, y_2^2, \sqrt{2}y_1y_2) \\ &= x_1^2y_1^2 + x_2^2y_2^2 + 2x_1x_2y_1y_2 \\ &= (x_1y_1 + x_2y_2)^2 = (\mathbf{x}^\top\mathbf{y})^2 \end{aligned}$$

Kernel Methods - Pros and Cons

+ Powerful Modeling Tool
  (non-linear problems become linear in kernel space)
+ Omnipurpose Kernels
  (Gaussian works well in many cases)
+ Kernel methods can handle symbolic objects
+ When you have less data points than your data has dimensions kernel methods can offer a dramatic speedup

– Difficult to understand what's happening in kernel space
– Model complexity increases with number of data points

$\rightarrow$ If you have too much data, kernel methods can be slow

Kernel Methods - Training and Testing in Practice

Input: Training Data $X \in \mathbb{R}^{D\times N},\ Y \in \mathbb{R}^{1\times N}$

- **Compute Kernel Marix $K \in \mathbb{R}^{N\times N}$ on Training Data**

$$K = k(X_{\text{train}}, X_{\text{train}})$$

  $K_{ij} = k(x_i, x_j)$: similarity between ith and jth data point

- **Training**
  Compute weights $\alpha \in \mathbb{R}^{N\times 1}$ for each training sample

$$\alpha = \text{someKernelAlgorithm}(K, Y_{\text{train}})$$

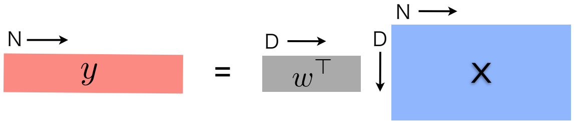
- **Testing**
  Compute predictions on test data X_{test}

$$\hat{Y}_{\text{test}} = (k(X_{\text{test}}, X_{\text{train}})\alpha)^{\top}$$

Recap: Linear Regression

Let N be the number of samples, so $y \in \mathbb{R}^{1\times N}$ and $X \in \mathbb{R}^{D\times N}$. The Linear Regression model in matrix notation then becomes

$$y = \mathbf{w}^\top X.$$



Linear Regression minimizes the least-squares loss function

$$\mathcal{E}_{lsq}(\mathbf{w}) = \sum_{n=1}^{N} (y_n - \mathbf{w}^\top X_n)^2 = \|y - \mathbf{w}^\top X\|^2$$

Recap: Ridge Regression

Linear ridge regression looks for a linear combination of features **w** that minimizes the prediction error and has a small norm

$$\begin{aligned}\mathcal{E}_{RR}(\mathbf{w}) =& ||y - \mathbf{w}^\top X||^2 + \lambda ||\mathbf{w}||^2 \\ =& yy^\top - 2\mathbf{w}^\top X y^\top + \mathbf{w}^\top X X^\top \mathbf{w} + \lambda \mathbf{w}^\top \mathbf{w}\end{aligned}$$

Recap: Ridge Regression

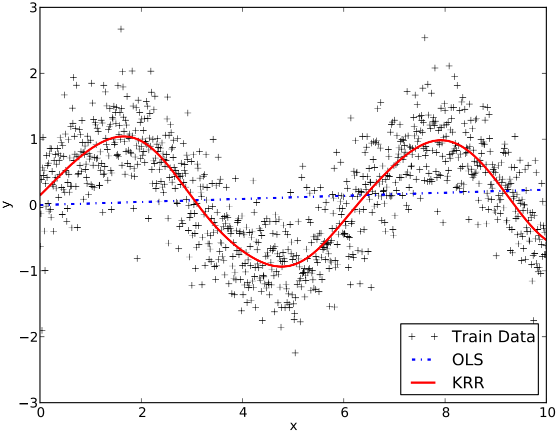
Computing the derivative w.r.t. w yields

$$\frac{\partial \mathcal{E}_{RR}(\mathbf{w})}{\partial \mathbf{w}} = -2Xy^\top + 2XX^\top \mathbf{w} + \lambda 2\mathbf{w}.$$

Setting the gradient to zero and rearranging terms the optimal **w** is

$$\begin{aligned} 2XX^\top \mathbf{w} + \lambda 2\mathbf{w} =& 2Xy^\top \\ (XX^\top + \lambda I)\mathbf{w} =& Xy^\top \\ \mathbf{w} =& (XX^\top + \lambda I)^{-1} Xy^\top \end{aligned}$$

From Linear to Kernel Ridge Regression



From Linear to Kernel Ridge Regression

Setting the derivative to **0**:

$$\begin{aligned}\frac{\partial \mathcal{E}_{RR}(\mathbf{w})}{\partial \mathbf{w}} =& -2Xy^\top + 2XX^\top\mathbf{w} + \lambda 2\mathbf{w} \overset{!}{=} \mathbf{0} \\ \rightarrow \mathbf{w} =& X \underbrace{\frac{1}{\lambda}(y^\top - X^\top \mathbf{w})}_{\alpha} \\ =& X\alpha\end{aligned}$$

The optimal **w** is a **linear combination** α **of all data points** X.

From Linear to Kernel Ridge Regression

$$\begin{aligned}
\alpha =& \frac{1}{\lambda}(y^\top - X^\top \mathbf{w}) \\
\lambda\alpha =& y^\top - X^\top X\alpha \\
(X^\top X + \lambda I)\alpha =& y^\top \\
\rightarrow \alpha =& (\underbrace{X^\top X}_{K} + \lambda I)^{-1} y^\top
\end{aligned}$$

$K \in \mathbb{R}^{N\times N}$ is called **kernel matrix** - here just a linear kernel
K_{ij} denotes similarity between data point x_i and x_j
Other kernel functions include the gaussian kernel in eq. 1

From Linear to Kernel Ridge Regression

Predictions for new data $\mathbf{x}_{new}$:

$$\begin{aligned} y_{new} &= \mathbf{w}^\top \mathbf{x}_{new} \\ &= (X\alpha)^\top \mathbf{x}_{new} \\ &= \alpha^\top X^\top \mathbf{x}_{new} \\ &= (\underbrace{\mathbf{x}_{new}^\top X}_{k(\mathbf{x}_{new}, X_{train})} \alpha)^\top \end{aligned}$$

Kernel Ridge Regression Algorithm

Algorithm 1: Gaussian Kernel

```
Require: Data $x_1, \dots, x_n = X$, $\hat{x}_1, \dots, \hat{x}_m = \hat{X}$ , kernel width $\sigma$
1: $H = repmat(sum((X. * X), 2), 1, m) - 2X^\top \hat{X} + repmat(sum((\hat{X}. * \hat{X}), 2)^\top, n, 1)$
2: $K = exp(-H./2./\sigma^2)$;
3: return $K$
```

Algorithm 2: Kernel Ridge Regression - Training

```
Require: Kernel matrix $K \in \mathbb{R}^{N_{\text{train}} \times N_{\text{train}}}$, labels $[y_1, \dots, y_N]$, ridge $\lambda$
1: # Compute dual coefficients
2: $\alpha = (K + \lambda I)^{-1} y^\top$
3: return $\alpha$
```

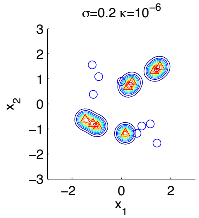
Algorithm 3: Kernel Ridge Regression - Prediction

```
Require: Kernel matrix $K \in \mathbb{R}^{N_{\text{test}} \times N_{\text{train}}}$, weights $\alpha$
1: return $(K\alpha)^\top$
```

Generalization and Model Selection

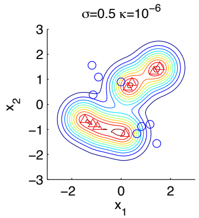
The best model is the model that *generalizes best*

Overfitting



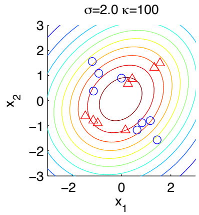
Model is too complex
→ Bad generalization

Better fit



Appropriate complexity
→ Good generalization

Underfitting



Model is too simple
→ Bad generalization

How to Achieve Good Generalization?

When using powerful algorithms (MLPs, KRR, . . .)
every data set can be modeled perfectly! (overfitting)

But we want to model new data well (generalization)

Cross-validation can be used for:

- **Model evaluation**
  Test how good an algorithm actually is
- **Model selection**
  Optimize parameters of a model for generalization performance

Cross-Validation

Split data set in F different **training** and **test** data

$$\text{fold } 1 \; [\; \underbrace{x_1,\ x_2,\ x_3,\ x_4}_{\mathcal{F}_1^{\text{train}}},\ \underbrace{x_5,\ x_6}_{\mathcal{F}_1^{\text{test}}} \;]$$

$$\text{fold } 2 \; [\; \underbrace{x_1,\ x_2,}_{\mathcal{F}_1^{\text{test}}}\ \underbrace{x_3,\ x_4,\ x_5,\ x_6}_{\mathcal{F}_1^{\text{train}}} \;]$$

fold 3 . . .

For each fold:

Train your model on the training data

Test your model on the test data

Cross-Validation: Model Selection or Model Evaluation

Model Evaluation
Report **mean evaluation score** – e.g. accuracy – across folds

Model Selection
Take that parameter with the highest mean score across folds

Combined Model Selection and Evaluation
$\rightarrow$ **Nested Cross-Validation**

Nested Cross-Validation

Algorithm 4: Cross-Validation for Model Selection and Evaluation

```
Require: Data $(x_1, y_1) \dots, (x_N, y_N)$, parameters $\sigma_1, \dots, \sigma_S$, Number of CV folds $F$
 1: # Split data in $F$ disjunct folds
 2: for Outer folds $f_{\text{outer}} = 1, \dots, F$ do
 3:     # Pick folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ for Model Selection
 4:     # Model Selection
 5:     for Fold $f_{\text{inner}} = 1, \dots, F-1$ do
 6:         for Parameter $s = 1, \dots, S$ do
 7:             # Train model on folds $\{1, \dots, F\} \setminus \{f_{\text{outer}}, f_{\text{inner}}\}$ with parameter $\sigma_s$
 8:             # Compute prediction on fold $f_{\text{inner}}$
 9:         end for
10:     end for
11:     # Pick best parameter $\sigma_s$ for all $f_{\text{inner}}$
12:     # Model Evaluation
13:     # Train model on folds $\{1, \dots, F\} \setminus f_{\text{outer}}$ with parameter $\sigma_s$
14:     # Test model on fold $f_{\text{outer}}$
15: end for
```

Summary

Kernel Ridge Regression

- Non-linear regression
- Predictions involve comparison of new and old data
- Predictions based on linear combination of (non-linear) similarity measures (e.g. eq.1)
- Optimization requires inversion of kernel matrix ($N \times N$, N=number of examples)
- $\rightarrow$ Difficult for very large data sets

Generalization and Model Selection

- Good prediction on new data is called generalization
- Good algorithms maximize generalization performance
- Cross-Validation is a simple and powerful framework for model selection

References

A. Aizerman, E. Braverman, and L. Rozonoer. Theoretical foundations of the potential function method in pattern recognition learning. *Automation and Remote Control*, 25:821–837, 1964.

C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. 2007.

G. Cybenko. Approximations by superpositions of sigmoidal functions. *Mathematics of Control, Signals, and Systems*, 2(4):303–314, 1989.

K. Hornik. Approximation capabilities of multilayer feedforward networks. *Neural Networks*, 4(2):251–257, 1991.

F. Jäkel. *Some Theoretical Aspects of Human Categorization Behaviour: Similarity and Generalization*. PhD thesis, 2007.

F. Jäkel, B. Schölkopf, and F. A. Wichmann. Does cognitive science need kernels? *Trends Cogn Sci*, 13(9):381–388, 2009. doi: 016/j.tics.2009.06.002.

K.-R. Müller, S. Mika, G. Ratsch, K. Tsuda, and B. B. Schölkopf. An introduction to kernel-based learning algorithms. *IEEE Transactions on Neural Networks*, 12(2):181–201, 2001. doi: 10.1109/72.914517.

B. Schölkopf and A. J. Smola. *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond*. MIT Press, 2002.

J. Shawe-Taylor and N. Cristianini. *Kernel methods for pattern analysis*. Cambridge University Press, 2004.

R. N. Shepard. Toward a universal law of generalization for psychological science. *Science*, 237(4820):1317–23, 1987.