CSC 411 Lecture 18: Kernels

Ethan Fetaya, James Lucas and Emad Andrews

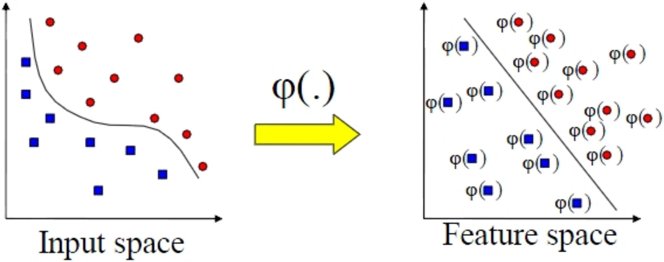
University of Toronto

Today

- Kernel trick
- Representer theorem

Non-linear decision boundaries

- We talk about SVM: max margin linear classifier
- Linear is limiting, how do we get non-linear decision boundaries?
- Feature mapping $\mathbf{x} \rightarrow \phi(\mathbf{x})$



- How do we find good features?
- If features are in a high dimension - high computational cost.

Motivation

- Let's say that we want a quadratic decision boundary
- What feature mapping do we need?
- One possibility (ignore arbitrary $\sqrt{2}$ for now)

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, ...\sqrt{2}x_{d-1}x_d, x_1^2, ..., x_d^2)$$

Pairwise is over $i < j$

- We have $dim(\phi(\mathbf{x})) = \mathcal{O}(d^2)$, could be problematic for large d.
- How can this be addressed?

Kernel Trick Idea

- Linear algorithms are based on inner-product
- What if you could compute the inner product without computing $\phi(\mathbf{x})$?
- Our previous example:

$$\phi(\mathbf{x}) = (1, \sqrt{2}x_1, ..., \sqrt{2}x_d, \sqrt{2}x_1x_2, \sqrt{2}x_1x_3, ...\sqrt{2}x_{d-1}x_d, x_1^2, ..., x_d^2)$$

- What is $K(\mathbf{x}, \mathbf{y}) = \langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle$?

$$\langle \phi(\mathbf{x}), \phi(\mathbf{y}) \rangle = 1 + \sum_{i=1}^{d} 2x_iy_i + \sum_{i,j=1}^{d} x_ix_jy_iy_j = (1 + \langle \mathbf{x}, \mathbf{y} \rangle)^2$$

- We can compute K in $\mathcal{O}(d)$ memory and compute time!
- K is called the (polynomial) kernel.

Kernel SVM

- SVM dual form objective: $w = \sum \alpha_i t^{(i)} \mathbf{x}^{(i)}$

$$\max_{\alpha_i \geq 0} \{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j (\mathbf{x}^{(i)^T} \cdot \mathbf{x}^{(j)}) \}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

- Non-linear SVM using kernel function $K()$:

$$\max_{\alpha_i \geq 0} \{ \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{N} t^{(i)} t^{(j)} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \}$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq C; \quad \sum_{i=1}^{N} \alpha_i t^{(i)} = 0$$

- Unlike linear SVM, cannot express **w** as linear combination of support vectors
  - now must retain the support vectors to classify new examples
- Final decision function:
$y = \text{sign}[b + \left\langle \sum_{i=1}^{N} t^{(i)} \alpha_i \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle] = \text{sign}[b + \sum_{i=1}^{N} t^{(i)} \alpha_i K(\mathbf{x}, \mathbf{x}^{(i)})]$

Kernels

- Examples of kernels: kernels measure similarity
  1. Polynomial
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = (\mathbf{x}^{(i)^T}\mathbf{x}^{(j)} + 1)^d$$
  where d is the degree of the polynomial, e.g., $d = 2$ for quadratic
  2. Gaussian/RBF
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \exp(-\frac{||\mathbf{x}^{(i)} - \mathbf{x}^{(j)}||^2}{2\sigma^2})$$
  3. Sigmoid
  $$K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) = \tanh(\beta(\mathbf{x}^{(i)^T}\mathbf{x}^{(j)}) + a)$$
- Kernel functions exist for non-vectorized data - string kernel, graph kernel, etc.
- Each kernel computation corresponds to a dot product
  - calculation for particular mapping $\phi(\mathbf{x})$ implicitly maps to high-dimensional space

Kernel Functions

- Mercer's Theorem (1909): any reasonable kernel corresponds to some feature space
- Reasonable means that the Gram matrix is positive semidefinite

$$K_{ij} = K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

- We can build complicated kernels so long as they are positive semidefinite.
- We can combine simple kernels together to make more complicated ones

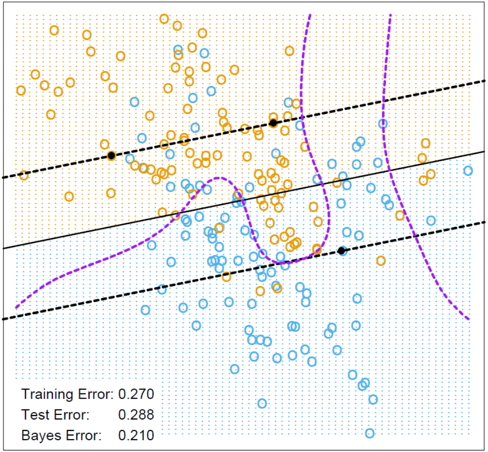
Basic Kernel Properties

- Positive constant function is a kernel: for $\alpha \geq 0$, $K'(x_1, x_2) = \alpha$
- Positively weighted linear combinations of kernels are kernels: if $\forall i, \alpha_i \geq 0$, $K'(x_1, x_2) = \sum_i \alpha_i K_i(x_1, x_2)$
- Products of kernels are kernels: $K'(x_1, x_2) = K_1(x_1, x_2) K_2(x_1, x_2)$
- The above transformations preserve positive semidefinite functions
- We can use kernels as building blocks to construct complicated feature mappings

Kernel Feature Space

- Kernels let us express very large feature spaces
  - polynomial kernel $(1 + (\mathbf{x}^{(i)})^T \mathbf{x}^{(j)})^d$ corresponds to feature space exponential in d
  - Gaussian kernel has infinitely dimensional features
- Linear separators in these super high-dimensional spaces correspond to highly non-linear decision boundaries in the input space

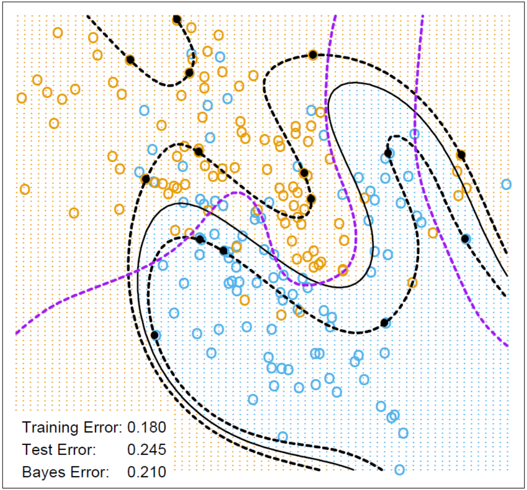
Example - linear SVM



- Solid line - decision boundary. Dashed - +1/-1 margin. Purple - Bayes optimal
- Solid dots - Support vectors on margin

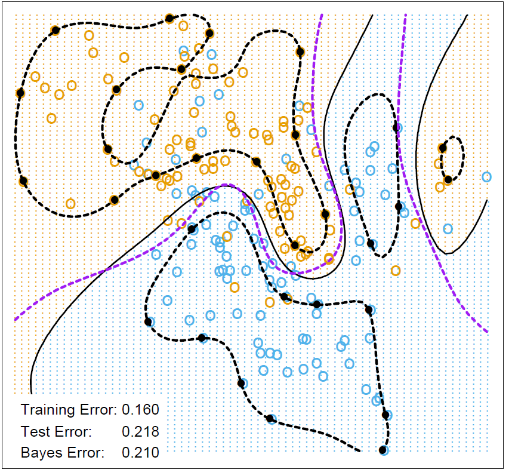
[Image credit: "Elements of statistical learning"]

Example - Deg 4 polynomial SVM



[Image credit: "Elements of statistical learning"]

Example - Gaussian SVM



[Image credit: "Elements of statistical learning"]

Kernel methods

- Kernels work well with SVM but not limited to it.
- When can we apply the kernel trick?

Representer Theorem:
If $\mathbf{w}^*$ is defined as

$$\mathbf{w}^* = \arg\min \sum_{i=1}^{N} L\left(\left\langle \mathbf{w}, \phi(\mathbf{x}^{(i)}) \right\rangle, t^{(i)}\right) + \lambda ||\mathbf{w}||^2$$

Then $\mathbf{w}^* \in span\{\phi(x_1), ..., \phi(x_N)\}$, i.e. $\exists \alpha : \mathbf{w}^* = \sum_{i=1}^{N} \alpha_i \phi(x_i)$

- Proof idea: The subspace that is orthogonal to the span doesn't impact the loss, but increases the norm $\Rightarrow$ Optimal thing is to set it to zero.
- We assume you can predict using inner-product.

Optimization

- We can compute

$$\langle \mathbf{w}, \phi(\mathbf{x}) \rangle = \left\langle \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^{N} \alpha_i \left\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}) \right\rangle = \sum_{i=1}^{N} \alpha_i K(\mathbf{x}^{(i)}, \mathbf{x})$$

- Similarly for the regularizer

$$\begin{aligned} ||\mathbf{w}||^2 &= \left\langle \sum_{i=1}^{N} \alpha_i \phi(\mathbf{x}^{(i)}), \sum_{j=1}^{N} \alpha_j \phi(\mathbf{x}^{(j)}) \right\rangle = \sum_{i,j=1}^{N} \alpha_i \alpha_j \left\langle \phi(\mathbf{x}^{(i)}), \phi(\mathbf{x}^{(j)}) \right\rangle \\ &= \sum_{i=1}^{N} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}) \end{aligned}$$

- We can optimize without computing $\phi(\mathbf{x})$.

$$\alpha = \arg\min \sum_{i=1}^{N} L\left(\sum_{j=1}^{N} \alpha_j k(\mathbf{x}^{(i)}, \mathbf{x}^{(j)}), t^{(i)} \right) + \lambda \sum_{i=1}^{N} \alpha_i \alpha_j K(\mathbf{x}^{(i)}, \mathbf{x}^{(j)})$$

Other Kernel methods

- Kernel Logistic regression
  - We can think of logistic regression as minimizing $\log(1 + \exp(-t^{(i)}\mathbf{w}^T\mathbf{x}^{(i)}))$
  - If you use L_2 regularization (Gaussian prior) this fits the representer theorem.
  - Performance is close to SVM
- PCA
  - A bit trickier to show how to only use kernels.
  - Equivalent to first using a non-linear transformation to high dimension then use linear projection to low dimension.
- Kernel Bayesian methods (not covered in this course)
  - Gaussian processes

Kernel and SVM

- The kernel trick is not limited to SVM, but is most common with it.
- Why do the kernel trick and SVM work well together?
- Generalization:
  - The kernel trick allows you to work in very high dimensions - what about overfitting?
  - SVM enjoys generalization bounds that don't depend on dimension (depend on margin or #support vectors).
  - Regularization is still very important to reduce overfitting.
- Computation:
  - In general $\mathbf{w}^*$ is a linear combination of the training data
  - SVM only need to save a (hopefully small) subset of support vectors - Less memory and faster predictions.

Summary

- Advantages:
  - Kernels allow very flexible hypotheses
  - Kernel trick allows us to work in very high (or infinite) dimensional space
  - Soft-margin extension permits mis-classified examples
  - Can usually outperform linear svm
- Disadvantages:
  - Must choose kernel parameters
  - Large number of support vector $\Rightarrow$ Computationally expensive to predict new points.
  - Can overfit.

More Summary

- Software:
  - Sklearn implementation is based on LIBSVM (SMO algorithm)
  - SVMLight is among the earliest implementations
  - svm-Perf uses Cutting-Plane Subspace Pursuit.
  - Several Matlab toolboxes for SVM are also available
- Key points:
  - Difference between logistic regression and SVMs
  - Maximum margin principle
  - Target function for SVMs
  - Slack variables for mis-classified points
  - Kernel trick allows non-linear generalizations