Kernel Methods and Nonlinear Classification

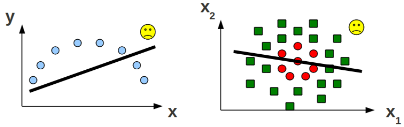
Piyush Rai

CS5350/6350: Machine Learning

September 15, 2011

Kernel Methods: Motivation

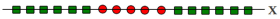
- Often we want to capture nonlinear patterns in the data
  - Nonlinear Regression: Input-output relationship may not be linear
  - Nonlinear Classification: Classes may not be separable by a linear boundary
- Linear models (e.g., linear regression, linear SVM) are not just rich enough



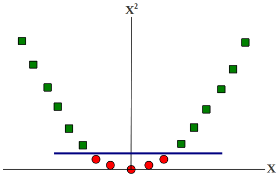
- Kernels: Make linear models work in nonlinear settings
  - By mapping data to higher dimensions where it exhibits linear patterns
  - Apply the linear model in the new input space
  - Mapping $\equiv$ changing the feature representation
- **Note:** Such mappings can be expensive to compute in general
  - Kernels give such mappings for (almost) free
    - In most cases, the mappings need not be even computed
    - .. using the Kernel Trick!

Classifying non-linearly separable data

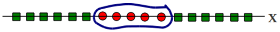
- Consider this binary classification problem



  - Each example represented by a single feature x
  - No linear separator exists for this data
- Now map each example as $x \rightarrow \{x, x^2\}$
  - Each example now has two features ("derived" from the old representation)
- Data now becomes linearly separable in the new representation

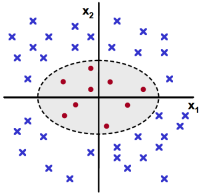


- Linear in the new representation $\equiv$ nonlinear in the old representation

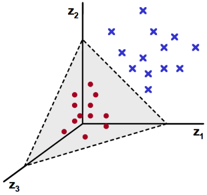


Classifying non-linearly separable data

- Let's look at another example:



  - Each example defined by a two features $\mathbf{x} = \{x_1, x_2\}$
  - No linear separator exists for this data
- Now map each example as $\mathbf{x} = \{x_1, x_2\} \rightarrow \mathbf{z} = \{x_1^2, \sqrt{2}x_1x_2, x_2^2\}$
  - Each example now has three features ("derived" from the old representation)
- Data now becomes linearly separable in the new representation



Feature Mapping

- Consider the following mapping ϕ for an example $\mathbf{x} = \{x_1, \ldots, x_D\}$

$$\phi : \mathbf{x} \rightarrow \{x_1^2, x_2^2, \ldots, x_D^2, , x_1x_2, x_1x_2, \ldots, x_1x_D, \ldots\ldots, x_{D-1}x_D\}$$

- It's an example of a quadratic mapping
  - Each new feature uses a pair of the original features
- Problem: Mapping usually leads to the number of features blow up!
  - Computing the mapping itself can be inefficient in such cases
  - Moreover, *using* the mapped representation could be inefficient too
    - e.g., imagine computing the similarity between two examples: $\phi(\mathbf{x})^\top \phi(\mathbf{z})$
- Thankfully, Kernels help us avoid both these issues!
  - The mapping doesn't have to be explicitly computed
  - Computations with the mapped features remain efficient

Kernels as High Dimensional Feature Mapping

- Consider two examples $\mathbf{x} = \{x_1, x_2\}$ and $\mathbf{z} = \{z_1, z_2\}$
- Let's assume we are given a function k (kernel) that takes as inputs $\mathbf{x}$ and $\mathbf{z}$

$$\begin{aligned} k(\mathbf{x}, \mathbf{z}) &= (\mathbf{x}^\top \mathbf{z})^2 \\ &= (x_1 z_1 + x_2 z_2)^2 \\ &= x_1^2 z_1^2 + x_2^2 z_2^2 + 2 x_1 x_2 z_1 z_2 \\ &= (x_1^2, \sqrt{2} x_1 x_2, x_2^2)^\top (z_1^2, \sqrt{2} z_1 z_2, z_2^2) \\ &= \phi(\mathbf{x})^\top \phi(\mathbf{z}) \end{aligned}$$

- The above k implicitly defines a mapping ϕ to a higher dimensional space

$$\phi(\mathbf{x}) = \{x_1^2, \sqrt{2} x_1 x_2, x_2^2\}$$

- Note that we didn't have to define/compute this mapping
- Simply defining the kernel a certain way gives a higher dim. mapping ϕ
- Moreover the kernel $k(\mathbf{x}, \mathbf{z})$ also computes the dot product $\phi(\mathbf{x})^\top \phi(\mathbf{z})$
  - $\phi(\mathbf{x})^\top \phi(\mathbf{z})$ would otherwise be much more expensive to compute explicitly
- All kernel functions have these properties

Kernels: Formally Defined

- Recall: Each kernel k has an associated feature mapping ϕ
- ϕ takes input $\mathbf{x} \in \mathcal{X}$ (input space) and maps it to $\mathcal{F}$ ("feature space")
- Kernel $k(\mathbf{x}, \mathbf{z})$ takes two inputs and gives their similarity in $\mathcal{F}$ space

$$\begin{aligned} \phi &: \mathcal{X} \to \mathcal{F} \\ k &: \mathcal{X} \times \mathcal{X} \to \mathbb{R}, \quad k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})^\top \phi(\mathbf{z}) \end{aligned}$$

- $\mathcal{F}$ needs to be a *vector space* with a *dot product* defined on it
  - Also called a *Hilbert Space*
- Can just *any* function be used as a kernel function?
  - No. It must satisfy **Mercer's Condition**

Mercer's Condition

- For k to be a kernel function
  - There must exist a Hilbert Space $\mathcal{F}$ for which k defines a dot product
  - The above is true if K is a positive definite function

$$\int d\mathbf{x} \int d\mathbf{z} f(\mathbf{x}) k(\mathbf{x}, \mathbf{z}) f(\mathbf{z}) > 0 \quad (\forall f \in L_2)$$

  - This is Mercer's Condition
- Let k_1, k_2 be two kernel functions then the following are as well:
  - $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) + k_2(\mathbf{x}, \mathbf{z})$: direct sum
  - $k(\mathbf{x}, \mathbf{z}) = \alpha k_1(\mathbf{x}, \mathbf{z})$: scalar product
  - $k(\mathbf{x}, \mathbf{z}) = k_1(\mathbf{x}, \mathbf{z}) k_2(\mathbf{x}, \mathbf{z})$: direct product
  - Kernels can also be constructed by composing these rules

The Kernel Matrix

- The kernel function k also defines the Kernel Matrix $\mathbf{K}$ over the data
- Given N examples $\{\mathbf{x}_1, \ldots, \mathbf{x}_N\}$, the (i, j)-th entry of $\mathbf{K}$ is defined as:

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

- K_{ij}: Similarity between the i-th and j-th example in the feature space $\mathcal{F}$
- $\mathbf{K}$: $N \times N$ matrix of pairwise similarities between examples in $\mathcal{F}$ space
- $\mathbf{K}$ is a symmetric matrix
- $\mathbf{K}$ is a positive definite matrix (except for a few exceptions)
- For a P.D. matrix: $\mathbf{z}^\top \mathbf{K} \mathbf{z} > 0, \quad \forall \mathbf{z} \in \mathbb{R}^N$ (also, all eigenvalues positive)
- The Kernel Matrix $\mathbf{K}$ is also known as the Gram Matrix

Some Examples of Kernels

The following are the most popular kernels for real-valued vector inputs

- Linear (trivial) Kernel:

  $$k(\mathbf{x}, \mathbf{z}) = \mathbf{x}^\top \mathbf{z} \text{ (mapping function } \phi \text{ is identity - no mapping)}$$

- Quadratic Kernel:

  $$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^2 \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^2$$

- Polynomial Kernel (of degree d):

  $$k(\mathbf{x}, \mathbf{z}) = (\mathbf{x}^\top \mathbf{z})^d \quad \text{or} \quad (1 + \mathbf{x}^\top \mathbf{z})^d$$

- Radial Basis Function (RBF) Kernel:

  $$k(\mathbf{x}, \mathbf{z}) = \exp[-\gamma ||\mathbf{x} - \mathbf{z}||^2]$$

  - γ is a hyperparameter (also called the kernel bandwidth)
  - The RBF kernel corresponds to an infinite dimensional feature space $\mathcal{F}$ (i.e., you can't actually write down the vector $\phi(\mathbf{x})$)

Note: Kernel hyperparameters (e.g., d, γ) chosen via cross-validation

Using Kernels

- Kernels can turn a linear model into a nonlinear one
- Recall: Kernel $k(\mathbf{x}, \mathbf{z})$ represents a dot product in some high dimensional feature space $\mathcal{F}$
- Any learning algorithm in which examples only appear as dot products $(\mathbf{x}_i^\top \mathbf{x}_j)$ can be kernelized (i.e., non-linearlized)
  - .. by replacing the $\mathbf{x}_i^\top \mathbf{x}_j$ terms by $\phi(\mathbf{x}_i)^\top \phi(\mathbf{x}_j) = k(\mathbf{x}_i, \mathbf{x}_j)$
- Most learning algorithms are like that
  - Perceptron, SVM, linear regression, etc.
  - Many of the unsupervised learning algorithms too can be kernelized (e.g., K-means clustering, Principal Component Analysis, etc.)

Kernelized SVM Training

- Recall the SVM dual Lagrangian:

$$\boxed{\begin{array}{l}\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n (\mathbf{x}_m^T \mathbf{x}_n) \\ \text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N\end{array}}$$

- Replacing $\mathbf{x}_m^T \mathbf{x}_n$ by $\phi(\mathbf{x}_m)^\top \phi(\mathbf{x}_n) = k(\mathbf{x}_m, \mathbf{x}_n) = K_{mn}$, where $k(.,.)$ is some suitable kernel function

$$\boxed{\begin{array}{l}\text{Maximize} \quad L_D(\mathbf{w}, b, \xi, \alpha, \beta) = \sum_{n=1}^{N} \alpha_n - \frac{1}{2} \sum_{m,n=1}^{N} \alpha_m \alpha_n y_m y_n K_{mn} \\ \text{subject to} \quad \sum_{n=1}^{N} \alpha_n y_n = 0, \quad 0 \leq \alpha_n \leq C; \quad n = 1, \ldots, N\end{array}}$$

- SVM now learns a linear separator in the kernel defined feature space $\mathcal{F}$
  - This corresponds to a non-linear separator in the original space $\mathcal{X}$

Kernelized SVM Prediction

- Prediction for a test example $\mathbf{x}$ (assume $b = 0$)

$$y = sign(\mathbf{w}^\top \mathbf{x}) = sign(\sum_{n \in SV} \alpha_n y_n \mathbf{x}_n{}^\top \mathbf{x})$$

- SV is the set of support vectors (i.e., examples for which $\alpha_n > 0$)
- Replacing each example with its feature mapped representation ($\mathbf{x} \rightarrow \phi(\mathbf{x})$)

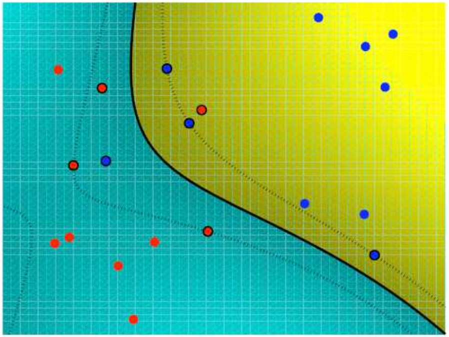
$$y = sign(\sum_{n \in SV} \alpha_n y_n \phi(\mathbf{x}_n)^\top \phi(\mathbf{x})) = sign(\sum_{n \in SV} \alpha_n y_n k(\mathbf{x}_n, \mathbf{x}))$$

- The weight vector for the kernelized case can be expressed as:

$$\mathbf{w} = \sum_{n \in SV} \alpha_n y_n \phi(\mathbf{x}_n) = \sum_{n \in SV} \alpha_n y_n k(\mathbf{x}_n, .)$$

- **Important:** Kernelized SVM needs the support vectors at the test time (except when you can write $\phi(\mathbf{x}_n)$ as an explicit, reasonably-sized vector)
  - In the unkernelized version $\mathbf{w} = \sum_{n \in SV} \alpha_n y_n \mathbf{x}_n$ can be computed and stored as a $D \times 1$ vector, so the support vectors need not be stored

SVM with an RBF Kernel



- The learned decision boundary in the original space is nonlinear

Kernels: concluding notes

- Kernels give a modular way to learn nonlinear patterns using linear models
  - All you need to do is replace the inner products with the kernel
- All the computations remain as efficient as in the original space
- Choice of the kernel is an important factor
- Many kernels are tailor-made for specific types of data
  - Strings (string kernels): DNA matching, text classification, etc.
  - Trees (tree kernels): Comparing parse trees of phrases/sentences
- Kernels can even be learned from the data (**hot research topic!**)
  - Kernel learning means learning the similarities between examples (instead of using some pre-defined notion of similarity)
- A question worth thinking about: Wouldn't mapping the data to higher dimensional space cause my classifier (say SVM) to overfit?
  - The answer lies in the concepts of large margins and generalization

Next class..

- Intro to probabilistic methods for supervised learning
  - Linear Regression (probabilistic version)
  - Logistic Regression