Kernels

Jiali Lin

Virginia Tech

January 15, 2017

Outline

Kernel functions

The kernel trick

Support vector machines

Introduction

- **Goal:** measure the similarity between objects, that doesn't require preprocessing them into feature vector format.
- E.g. when comparing strings, we can compute the edit distance between them.
- **Kernel function** $\kappa(\boldsymbol{x}, \boldsymbol{x'})$: some measure of similarity between objects $\boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}$, where $\mathcal{X}$ is some abstract space.

Mercer Kernel Functions

- A kernel function maps pairs of inputs to real numbers

$$\kappa : \mathcal{X} \times \mathcal{X} \to \mathbb{R} \qquad \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \kappa(\boldsymbol{x}_j, \boldsymbol{x}_i)$$

  Intuition: Larger values indicate inputs are"more similar".

- A kernel function is positive semidefinite if and only if for any $n \geq 1$, and any $x = \{\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\}$, the Gram matrix is positive semidefinite

$$\boldsymbol{K} \in \mathbb{R}^{n \times n} \qquad K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j)$$

- **Mercer's Theorem**: Assuming certain technical conditions, every positive definite kernel function can be represented as

$$\kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \sum_{\ell=1}^{d} \phi_\ell(\boldsymbol{x}_i) \phi_\ell(\boldsymbol{x}_j)$$

- **Motivation**: Can be faster to compute kernel than features.

RBF kernels

- The **squared exponential kernel** (SE kernel) or **Gaussian kernel**

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\frac{1}{2}(\boldsymbol{x} - \boldsymbol{x'})^T \boldsymbol{\Sigma}^{-1}(\boldsymbol{x} - \boldsymbol{x'}))$$

- If $\boldsymbol{\Sigma}^{-1}$ is diagonal, this can be written as

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\frac{1}{2}\sum_{j=1}^{D} \frac{1}{\sigma_j^2}(\boldsymbol{x}_j - \boldsymbol{x}'_j)^2)$$

- We can interpret the σ_j as defining the **characteristic length scale** of dimension j.
- **ARD kernel**: If $\sigma_j = \infty$, the corresponding dimension is ignored.
- **Radial basis function:** If $\boldsymbol{\Sigma}$ is spherical, we get the isotropic kernel

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = \exp(-\frac{\|\boldsymbol{x} - \boldsymbol{x'}\|^2}{2\sigma^2})$$

σ^2 is **bandwidth**. **RBF** kernel is only a function of $\|\boldsymbol{x} - \boldsymbol{x'}\|$.

Polynomial Kernels

- $\mathcal{X} \rightarrow$ real vectors of some fixed dimension.
- **Polynomial kernel**

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = (\gamma \boldsymbol{x}^T \boldsymbol{x}' + r)^M, \quad \text{where} \quad r > 0$$

- If $M = 2, \gamma = r = 1$ and $x, x' \in \mathbb{R}^2$, we have

$$\begin{aligned}
(1 + \boldsymbol{x}^T \boldsymbol{x'})^2 &= (1 + \boldsymbol{x}_1 \boldsymbol{x}_1' + \boldsymbol{x}_2 \boldsymbol{x}_2')^2 \\
&= 1 + 2\boldsymbol{x}_1 \boldsymbol{x}_1' + 2\boldsymbol{x}_2 \boldsymbol{x}_2' + (\boldsymbol{x}_1 \boldsymbol{x}_1')^2 + (\boldsymbol{x}_2 \boldsymbol{x}_2')^2 + 2\boldsymbol{x}_1 \boldsymbol{x}_1' \boldsymbol{x}_2 \boldsymbol{x}_2' \\
&= \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x})
\end{aligned}$$

where $\boldsymbol{\phi}(\boldsymbol{x}) = [1, \sqrt{2}\boldsymbol{x}_1, \sqrt{2}\boldsymbol{x}_2, \boldsymbol{x}_1^2, \boldsymbol{x}_2^2, \sqrt{2}\boldsymbol{x}_1 \boldsymbol{x}_2]^T$

- This is equivalent to working in a 6 dimensional feature space.
- In the case of a Gaussian kernel, the feature map lives in an infinite dimensional space.

Linear kernels

- Deriving the feature vector implied by a kernel is in general quite difficult, and only possible if the kernel is Mercer.
- However, deriving a kernel from a feature vector is easy: we just use

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{\phi}(\boldsymbol{x})^T \boldsymbol{\phi}(\boldsymbol{x}) =< \boldsymbol{\phi}(\boldsymbol{x}), \boldsymbol{\phi}(\boldsymbol{x}') >$$

- If $\boldsymbol{\phi}(\boldsymbol{x}) = \boldsymbol{x}$, we get the linear kernel, defined by

$$\kappa(\boldsymbol{x}, \boldsymbol{x'}) = \boldsymbol{x}^T \boldsymbol{x'}$$

Outline

Kernel functions

The kernel trick

Support vector machines

Kernelized ridge regression

The primal problem:

- Let $\boldsymbol{x} \in \mathbb{R}^D$ be some feature vector, and $\boldsymbol{X}$ be the corresponding $N \times D$ design matrix. We want to minimize

$$J(\boldsymbol{w}) = (\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^T(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) + \lambda\|\boldsymbol{w}\|^2$$

- The optimal solution is given by

$$\boldsymbol{w} = (\boldsymbol{X}^T\boldsymbol{X} + \lambda\boldsymbol{I}_D)^{-1}\boldsymbol{X}^T\boldsymbol{y} = (\sum_i \boldsymbol{x_i}\boldsymbol{x_i^T} + \lambda\boldsymbol{I}_D)^{-1}\boldsymbol{X}^T\boldsymbol{y}$$

Kernelized ridge regressionc(cont'd)

The dual problem:

- Rewrite the ridge estimate by the matrix inversion lemma

$$\boldsymbol{w} = \boldsymbol{X}^T(\boldsymbol{X}\boldsymbol{X}^T + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{y}$$

- Why?
  - This can be computationally advantageous if D is large.
  - Partially kernelize this, by replacing $\boldsymbol{X}\boldsymbol{X}^T$ with the Gram matrix $\boldsymbol{K}$.
- What about the leading $\boldsymbol{X}^T$ term?
  - Define the following **dual variables**

$$\alpha = (\boldsymbol{K} + \lambda \boldsymbol{I}_N)^{-1}\boldsymbol{y}$$

  - Then we can rewrite the primal variables as follows

$$\boldsymbol{w} = \boldsymbol{X}^T\alpha = \sum_{i=1}^{N} \alpha_i \boldsymbol{x_i}$$

  - Plug this in at test time to compute the predictive mean, we get

$$\hat{f}(\boldsymbol{x}) = \boldsymbol{w^T}\boldsymbol{x} = \sum_{i=1}^{N} \alpha_i \boldsymbol{x_i^T}\boldsymbol{x} = \sum_{i=1}^{N} \alpha_i \kappa(\boldsymbol{x}, \boldsymbol{x_i})$$

Outline

Kernel functions

The kernel trick

Support vector machines

Introduction

- Consider the ℓ_2 regularized empirical risk function

$$J(\boldsymbol{w}, \lambda) = \sum_{i=1}^{N} L(y_i, \hat{y}_i) + \lambda \|\boldsymbol{w}\|^2$$

- **Support vector machine**: use a modified loss function to ensure that the solution is sparse, so that predictions only depend on a subset of the training data.
- SVMs are very unnatural from a probabilistic point of view.
  - SVMs encode sparsity in the loss function rather than the prior.
  - SVMs encode kernels by using an algorithmic trick, rather than being an explicit part of the model.
  - SVMs do not result in probabilistic outputs (nonparametric).

Losses for Binary Classification

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} L(\tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)), \quad \text{where} \quad \tilde{y}_i \in \{+1, -1\}$$

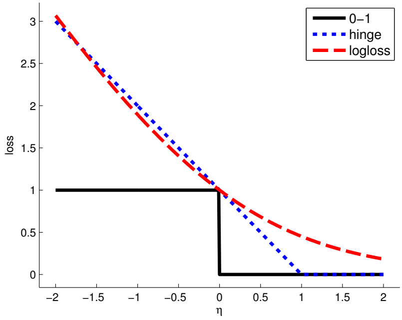

Figure: Illustration of various loss functions for binary classification. The horizontal axis is the margin $y\eta$, the vertical axis is the loss. The log loss uses log base 2. Figure generated by `HingeLossPlot`.

Losses for Binary Classification (cont'd)

- Training Error Rate (0-1 loss)
  - For many classifications, the objective we really care about.
  - Hard to optimize (gradients zero almost everywhere).
  - Cannot distinguish top-performing training classifiers.
- Logistic Regression (logarithmic loss)
  - Estimates label probabilities for calibrated decision-making.
  - Easy to optimize (convex, smooth bound on 0-1 loss).
  - Scalability problems with large datasets and many features.
- Support Vector Machine (hinge loss)
  - Does not estimate valid probability distribution on labels.
  - Possible to optimize (convex, non-smooth bound on $0 - 1$ loss).
  - Chooses boundary which maximizes margin of training data.
  - Gives sparse solutions, allowing greater scalability.

Support Vector Machines (SVMs)

- **Goal**

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} L(\tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))$$

- **Hinge loss**

$$L_{\mathsf{hinge}}(y, \eta) = \max(0, 1 - y\eta) = (1 - y\eta)_+$$

- **SVM Penalized Objective**

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} (1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$

- **SVM Constrained Objective**

$$\operatorname*{argmin}_{\boldsymbol{w}, \xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C \sum_{i=1}^{N} (\xi_i)$$

$$\text{s.t.} \quad \xi_i \geq 0, \quad \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \geq 1 - \xi_i$$

  - Quadratic Program: Quadratic function with linear constraints.
  - **Slack Variables**: $\boldsymbol{x}_i$ penalize misclassified training examples.

Maximum Margin Hyperplanes

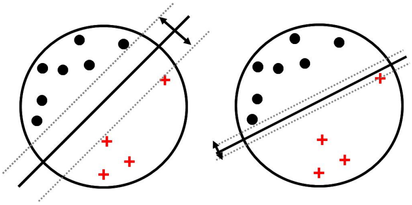

Figure: Illustration of the large margin principle. Left: a separating hyper-plane with large margin. Right: a separating hyper-plane with small margin.

- **Margin**: For a hyperplane which perfectly separates training data, orthogonal distance of closest training example to plane.
- **Intuition**: Expect similar features to have similar labels, so would like decision boundary as far as possible from data.
- Statistical Learning Theory: Formal bounds on generalization performance (test error) of large-margin classifiers.

Maximum Margin Hyperplanes (cont'd)

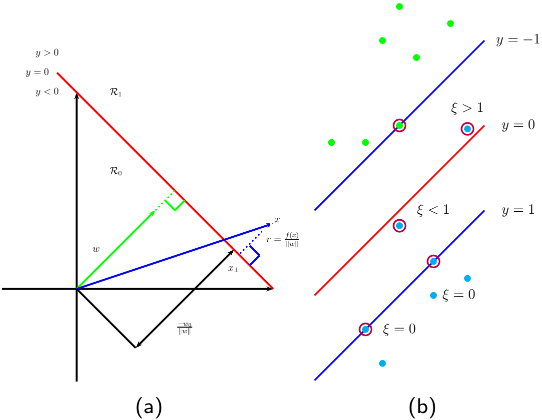

Figure: (a) Geometry of a linear decision boundary. (b) Soft margin principle.

Maximum Margin Hyperplanes (cont'd)

- Referring to Figure, $\boldsymbol{w}$ is perpendicular to the boundary, so $\overrightarrow{\boldsymbol{x}_\perp \boldsymbol{x}}$ is parallel to $\boldsymbol{w}$, i.e. $\overrightarrow{\boldsymbol{x}_\perp \boldsymbol{x}} = r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$.
- Thus, $\boldsymbol{x} = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$, where r is the distance of $\boldsymbol{x}$ from the decision boundary whose normal vector is $\boldsymbol{w}$, and $\boldsymbol{x}_\perp$ is the orthogonal projection of $\boldsymbol{x}$ onto this boundary.
- Multiply both sides by $\boldsymbol{w}^T$ and plus w_0 to get $f(\boldsymbol{x}) = f(\boldsymbol{x}_\perp) + r\|\boldsymbol{w}\|$. Now $f(\boldsymbol{x}_\perp) = 0$, thus $r = \frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|}$.

Margins and SVMs

- Let $\phi(\boldsymbol{x}) = \boldsymbol{x}_\perp + r\frac{\boldsymbol{w}}{\|\boldsymbol{w}\|}$.
- Now make this distance $r = \frac{f(\boldsymbol{x})}{\|\boldsymbol{w}\|}$ as large as possible.
- Accuracy: Classify all training data correctly by enforcing

$$\tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) > 0$$

- Margin: Maximize distance of closest point to boundary

$$\max_{w_0, \boldsymbol{w}} \min_{i=1,\ldots n} \frac{\tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|}$$

- Invariance: Scale $\boldsymbol{w}$ so closest point distance 1 from boundary

$$\operatorname*{argmin}_{\boldsymbol{w}, \xi} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}(\xi_i)$$
$$\text{s.t.} \quad \xi_i \geq 0, \quad \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \geq 1 - \xi_i$$

Support Vectors and Sparsity

$$\hat{\boldsymbol{w}} = \underset{\boldsymbol{w}}{\operatorname{argmin}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n}(1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$

- Optimal solution takes following form:

$$\hat{\boldsymbol{w}} = \sum_{i=1}^{n} \alpha_i \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i) \quad \alpha_i > 0$$

- Here, the α_i are Lagrange multipliers for constrained QP.
- Because loss exactly zero for arguments greater than one, only a sparse subset of training examples have $\alpha_i > 0$.
- Training examples with non-zero weight are support vectors.
- **Optimization**: quadratic program solver.
- Improvement: **sequential minimal optimization** or **SMO** algorithm.

SVMs and Kernels

- Optimal weights always the form, with non-zero weights only for support vectors
$$\boldsymbol{w} = \sum_{j=1}^{n} \beta_j \phi(\boldsymbol{x}_j)$$
- Kernel Tricks: for $\boldsymbol{K} \in \mathbb{R}^{n\times n}$, $K_{ij} = \kappa(\boldsymbol{x}_i, \boldsymbol{x}_j) = \phi(\boldsymbol{x}_i)^T \phi(\boldsymbol{x}_j)$.
- Then $\boldsymbol{f} = \boldsymbol{K}\boldsymbol{\beta}$, $f_i = \boldsymbol{w}^T \phi(\boldsymbol{x}_i)$. Thus, $\|\boldsymbol{w}\|^2 = \boldsymbol{f}^T \boldsymbol{K}^{-1} \boldsymbol{f}$.
- Dual SVM
$$\hat{\boldsymbol{f}} = \operatorname*{argmin}_{\boldsymbol{f}} \frac{\lambda}{2} \boldsymbol{f}^T \boldsymbol{K}^{-1} \boldsymbol{f} + \sum_{i=1}^{n} (1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$
- Primal SVM
$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{\lambda}{2} \|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} (1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$

SVMs and Gaussian Processes

- Logistic Regression

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} \log(1 + e^{\tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i)})$$

- GP Classification

$$\hat{\boldsymbol{f}} = \operatorname*{argmin}_{\boldsymbol{f}} \frac{\lambda}{2}\boldsymbol{f}^T \boldsymbol{K}^{-1}\boldsymbol{f} + \sum_{i=1}^{n} \log(1 + e^{\tilde{y}_i f_i})$$

- Dual SVM

$$\hat{\boldsymbol{f}} = \operatorname*{argmin}_{\boldsymbol{f}} \frac{\lambda}{2}\boldsymbol{f}^T \boldsymbol{K}^{-1}\boldsymbol{f} + \sum_{i=1}^{n} (1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$

- Primal SVM

$$\hat{\boldsymbol{w}} = \operatorname*{argmin}_{\boldsymbol{w}} \frac{\lambda}{2}\|\boldsymbol{w}\|^2 + \sum_{i=1}^{n} (1 - \tilde{y}_i \boldsymbol{w}^T \phi(\boldsymbol{x}_i))_+$$