Topic 6: KERNEL METHODS

STAT 37710/CAAM 37710/CMSC 35400 Machine Learning

Risi Kondor, The University of Chicago

General form of kernel methods

$k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel and $\mathcal{H}_k$ is the associated RKHS.

General form of kernel methods

$k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel and $\mathcal{H}_k$ is the associated RKHS. Kernel methods (aka. Hilbert space learning algorithms) solve the RRM problem

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \left[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}_k})}_{\text{regularizer}} \right].$$

General form of kernel methods

$k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel and $\mathcal{H}_k$ is the associated RKHS. Kernel methods (aka. Hilbert space learning algorithms) solve the RRM problem

$$\widehat{f} = \underset{f \in \mathcal{H}_k}{\operatorname{argmin}} \left[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}_k})}_{\text{regularizer}} \right].$$

- Ω can be any increasing function $\mathbb{R}^+ \to \mathbb{R}^+$.
- The final hypothesis is $\widehat{h}(x) = \sigma(\widehat{f}(x))$. For example, in the SVM, simply $\widehat{h}(x) = \operatorname{sgn}(\widehat{f}(x))$.

General form of kernel methods

$k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel and $\mathcal{H}_k$ is the associated RKHS. Kernel methods (aka. Hilbert space learning algorithms) solve the RRM problem

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \Bigg[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}_k})}_{\text{regularizer}} \Bigg].$$

- Ω can be any increasing function $\mathbb{R}^+ \to \mathbb{R}^+$.
- The final hypothesis is $\widehat{h}(x) = \sigma(\widehat{f}(x))$. For example, in the SVM, simply $\widehat{h}(x) = \operatorname{sgn}(\widehat{f}(x))$.
- ℓ is the **surrogate loss.** For example, in classification, the hinge loss is a surrogate for the zero-one loss.

General form of kernel methods

$k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is a PSD kernel and $\mathcal{H}_k$ is the associated RKHS. Kernel methods (aka. Hilbert space learning algorithms) solve the RRM problem

$$\widehat{f} = \underset{f \in \mathcal{H}_k}{\operatorname{argmin}} \left[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}_k})}_{\text{regularizer}} \right].$$

- Ω can be any increasing function $\mathbb{R}^+ \to \mathbb{R}^+$.
- The final hypothesis is $\widehat{h}(x) = \sigma(\widehat{f}(x))$. For example, in the SVM, simply $\widehat{h}(x) = \operatorname{sgn}(\widehat{f}(x))$.
- ℓ is the **surrogate loss**. For example, in classification, the hinge loss is a surrogate for the zero-one loss.

Instead of actually having to search over a function space, all such problems reduce to m dimensional optimization thanks to the reproducing property $f(x) = \langle f, k_x \rangle$ and the Representer Theorem.

The modularity of kernel methods

Regularized risk minimization in RKHSs is a powerful paradigm because it has distinct moving parts:

- **The loss**
    - Reflects the nature of the problem (classification/regression/ranking/...).

The modularity of kernel methods

Regularized risk minimization in RKHSs is a powerful paradigm because it has distinct moving parts:

- **The loss**
  - Reflects the nature of the problem (classification/regression/ranking/...).
  - Determines exactly what type of optimization problem we end up with.

The modularity of kernel methods

Regularized risk minimization in RKHSs is a powerful paradigm because it has distinct moving parts:

- **The loss**
  - Reflects the nature of the problem (classification/regression/ranking/...).
  - Determines exactly what type of optimization problem we end up with.
- **The kernel**
  - Regulates overfitting by determining the regularization term.

The modularity of kernel methods

Regularized risk minimization in RKHSs is a powerful paradigm because it has distinct moving parts:

- **The loss**
  - Reflects the nature of the problem (classification/regression/ranking/...).
  - Determines exactly what type of optimization problem we end up with.
- **The kernel**
  - Regulates overfitting by determining the regularization term.
  - Reflects our prior knowledge about the problem.

Can dream up virtually any kernel machine and solve it efficiently as long as

1. The loss only involves function evaluations $f(x) = \langle f, k_x \rangle$ at data points;

The modularity of kernel methods

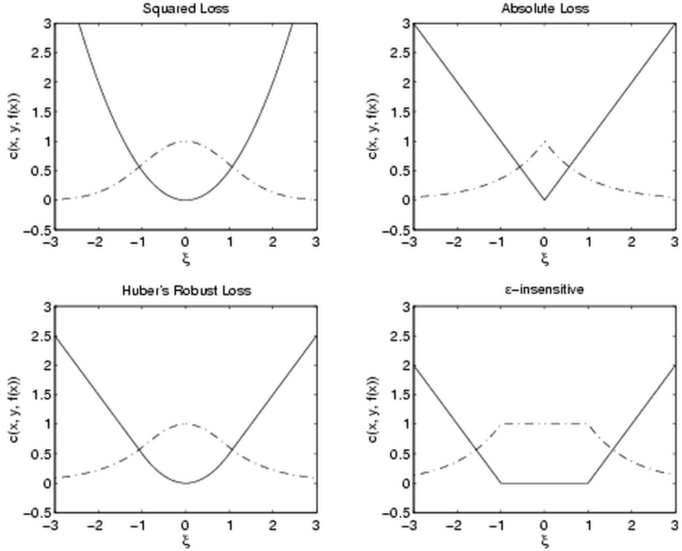
Regularized risk minimization in RKHSs is a powerful paradigm because it has distinct moving parts:

- **The loss**
  - Reflects the nature of the problem (classification/regression/ranking/...).
  - Determines exactly what type of optimization problem we end up with.
- **The kernel**
  - Regulates overfitting by determining the regularization term.
  - Reflects our prior knowledge about the problem.

Can dream up virtually any kernel machine and solve it efficiently as long as

1. The loss only involves function evaluations $f(x) = \langle f, k_x \rangle$ at data points;
2. The regularizer is an increasing function of $\| f \|_{\mathcal{F}}$.

Loss functions for regression



1. The kernel perceptron

The vanilla perceptron

$\mathbf{w} \leftarrow 0$;
$t \leftarrow 1$;
`while(true){`
  `if` $\mathbf{w} \cdot \mathbf{x}_t \geq 0$ `predict` $\hat{y}_t = 1$; `else predict` $\hat{y}_t = -1$;
  `if` $((\hat{y}_t = -1)$ `and` $(y_t = 1))$ `let` $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_t$;
  `if` $((\hat{y}_t = 1)$ `and` $(y_t = -1))$ `let` $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_t$;
  $t \leftarrow t + 1$;
`}`

The vanilla perceptron

$\mathbf{w} \leftarrow 0$;
$t \leftarrow 1$;
`while(true){`
  `if` $\mathbf{w} \cdot \mathbf{x}_t \geq 0$ `predict` $\hat{y}_t = 1$; `else predict` $\hat{y}_t = -1$;
  `if` $((\hat{y}_t = -1)$ `and` $(y_t = 1))$ `let` $\mathbf{w} \leftarrow \mathbf{w} + \mathbf{x}_t$;
  `if` $((\hat{y}_t = 1)$ `and` $(y_t = -1))$ `let` $\mathbf{w} \leftarrow \mathbf{w} - \mathbf{x}_t$;
  $t \leftarrow t + 1$;
`}`

At any t, the weight vector is of the form

$$\mathbf{w} = \sum_{i=1}^{t-1} c_i \, \mathbf{x}_i \qquad \text{where} \qquad c_i \in \{-1, 0, +1\} .$$

The kernel perceptron

```
t ← 1 ;
while(1){
  if  Σ_{i=1}^{t-1} c_i k(x_i, x_t) ≥ 0  predict  ŷ_t = 1 ; else  ŷ_t = −1 ;
  c_t ← 0 ;
  if  ((ŷ_t = −1)  and  (y_t = 1))  let  c_t = 1 ;
  if  ((ŷ_t = 1)  and  (y_t = −1))  let  c_t = −1 ;
  t ← t + 1 ;
}
```

$t \leftarrow 1$;

`while(1){`

`if` $\sum_{i=1}^{t-1} c_i k(\mathbf{x}_i, \mathbf{x}_t) \geq 0$ `predict` $\hat{y}_t = 1$; `else` $\hat{y}_t = -1$;

$c_t \leftarrow 0$;

`if` $((\hat{y}_t = -1)$ `and` $(y_t = 1))$ `let` $c_t = 1$;

`if` $((\hat{y}_t = 1)$ `and` $(y_t = -1))$ `let` $c_t = -1$;

$t \leftarrow t + 1$;

`}`

2. Kernel PCA

PCA in feature space

Recall that in $\mathbb{R}^D$ (after centering), the first principal component is given by

$$\mathbf{v}_1 = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{m}\sum_{i=1}^{m}(\mathbf{x}_i \cdot \mathbf{v})^2.$$

PCA in feature space

Recall that in $\mathbb{R}^D$ (after centering), the first principal component is given by

$$\mathbf{v}_1 = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i \cdot \mathbf{v})^2.$$

Clearly, $\mathbf{v}_1$ lies in the span, i.e., $\mathbf{v}_1 = \sum_{i=1}^{m} \alpha_i \, \mathbf{x}_i$.

PCA in feature space

Recall that in $\mathbb{R}^D$ (after centering), the first principal component is given by

$$\mathbf{v}_1 = \arg\max_{\|\mathbf{v}\|=1} \frac{1}{m} \sum_{i=1}^{m} (\mathbf{x}_i \cdot \mathbf{v})^2.$$

Clearly, $\mathbf{v}_1$ lies in the span, i.e., $\mathbf{v}_1 = \sum_{i=1}^{m} \alpha_i \, \mathbf{x}_i$.

Kernel analog:

$$f_1 = \underset{f \in \mathcal{F} \quad \|f\|=1}{\operatorname{argmax}} \sum_{i=1}^{m} \langle f, \phi(x_i) \rangle^2 .$$

Once again, $f = \sum_{i=1}^{m} \alpha_i \, \phi(x_i)$ for some $\alpha_1, \ldots, \alpha_m \in \mathbb{R}$.

Kernel PCA

As in $\mathbb{R}^D$, f will be the highest e-value e-vector of the sample covariance operator

$$\Sigma(f) = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) \left\langle f, \phi(x_i) \right\rangle .$$

Kernel PCA

As in $\mathbb{R}^D$, f will be the highest e-value e-vector of the sample covariance operator

$$\Sigma(f) = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) \langle f, \phi(x_i) \rangle .$$

Plugging in $f = \sum_{\ell=1}^{m} \alpha_\ell \, \phi(x_\ell)$ and multiplying from the right by any $\phi(x_j)$:

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{\ell=1}^{m} \langle \phi(x_j), \phi(x_i) \rangle \langle \phi(x_i), \phi(x_\ell) \rangle \alpha_\ell = \lambda \sum_{\ell=1}^{m} \langle \phi(x_j), \phi(x_\ell) \rangle \alpha_\ell .$$

Kernel PCA

As in $\mathbb{R}^D$, f will be the highest e-value e-vector of the sample covariance operator

$$\Sigma(f) = \frac{1}{m} \sum_{i=1}^{m} \phi(x_i) \langle f, \phi(x_i) \rangle .$$

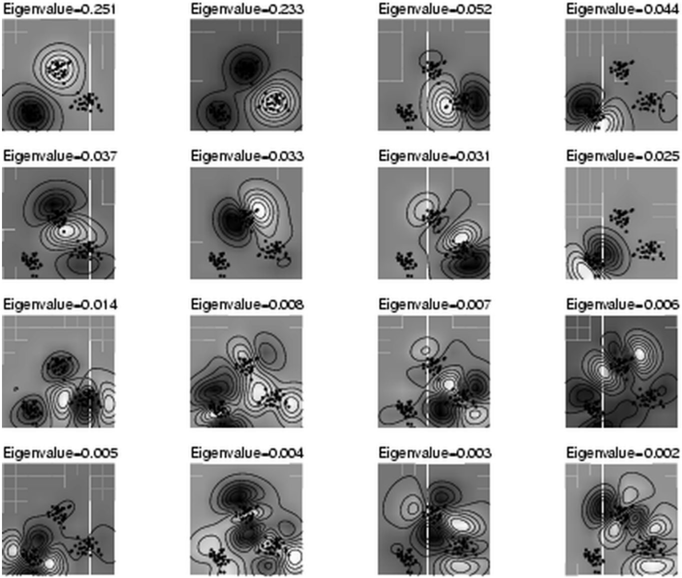
Plugging in $f = \sum_{\ell=1}^{m} \alpha_\ell \, \phi(x_\ell)$ and multiplying from the right by any $\phi(x_j)$:

$$\frac{1}{m} \sum_{i=1}^{m} \sum_{\ell=1}^{m} \langle \phi(x_j), \phi(x_i) \rangle \langle \phi(x_i), \phi(x_\ell) \rangle \alpha_\ell = \lambda \sum_{\ell=1}^{m} \langle \phi(x_j), \phi(x_\ell) \rangle \alpha_\ell.$$

Using $\langle \phi(x_j), \phi(x_i) \rangle = k(x_i, x_j)$ and letting K be the Gram matrix,

$$K^2 \boldsymbol{\alpha} = m \lambda K \alpha \qquad \Longrightarrow \qquad K \boldsymbol{\alpha} = m \lambda \boldsymbol{\alpha},$$

so kernel PCA reduces to just finding the first eigenvector of the Gram matrix!


Eigenvalue-0.251
Eigenvalue-0.233
Eigenvalue-0.052
Eigenvalue-0.044
Eigenvalue-0.037
Eigenvalue-0.033
Eigenvalue-0.031
Eigenvalue-0.025
Eigenvalue-0.014
Eigenvalue-0.008
Eigenvalue-0.007
Eigenvalue-0.006
Eigenvalue-0.005
Eigenvalue-0.004
Eigenvalue-0.003
Eigenvalue-0.002

3. Ridge Regression

Ridge Regression

Using squared error loss and setting $\lambda = m/2C$,

$$\widehat{f} = \underset{f \in \mathcal{H}_k}{\operatorname{argmin}} \Bigg[\underbrace{\sum_{i=1}^{m} (f(x_i) - y_i)^2 + \lambda \, \|f\|_{\mathcal{H}_k}^2}_{\mathcal{R}[f]} \Bigg].$$

Ridge Regression

Using squared error loss and setting $\lambda = m/2C$,

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \Bigg[\underbrace{\sum_{i=1}^{m} (f(x_i) - y_i)^2 + \lambda \, \|f\|^2_{\mathcal{H}_k}}_{\mathcal{R}[f]} \Bigg].$$

By the Representer Theorem, $f(x) = \sum_{i=1}^{m} \alpha_i k(x_i, x)$, so

$$\mathcal{R}[f] = \sum_{i=1}^{m} \Big(\sum_{j=1}^{m} \alpha_j \, k(x_i, x_j) - y_i \Big)^2 + \lambda \sum_{i=1}^{m} \sum_{j=1}^{m} \alpha_i \alpha_j \, k(x_i, x_j).$$

Ridge Regression

Letting $\mathbf{y} = (y_1, \ldots, y_m)$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$ and $K_{i,j} = k(x_i, x_j)$,

$$\mathcal{R}(\boldsymbol{\alpha}) = \| \boldsymbol{K}\boldsymbol{\alpha} - \mathbf{y} \|^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K} \boldsymbol{\alpha}.$$

Ridge Regression

Letting $\mathbf{y} = (y_1, \ldots, y_m)$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$ and $K_{i,j} = k(x_i, x_j)$,

$$\mathcal{R}(\boldsymbol{\alpha}) = \| \boldsymbol{K}\boldsymbol{\alpha} - \mathbf{y} \|^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K}\boldsymbol{\alpha}.$$

At the optimum,

$$\frac{\partial R(\boldsymbol{\alpha})}{\partial \alpha_i} = [\boldsymbol{K}(\boldsymbol{K}\boldsymbol{\alpha} - \mathbf{y})]_i + \lambda [\boldsymbol{K}\boldsymbol{\alpha}]_i = 0,$$

Ridge Regression

Letting $\mathbf{y} = (y_1, \ldots, y_m)$, $\boldsymbol{\alpha} = (\alpha_1, \ldots, \alpha_m)^\top$ and $K_{i,j} = k(x_i, x_j)$,

$$\mathcal{R}(\boldsymbol{\alpha}) = \| \boldsymbol{K\alpha} - \mathbf{y} \|^2 + \lambda \boldsymbol{\alpha}^\top \boldsymbol{K\alpha}.$$

At the optimum,

$$\frac{\partial R(\boldsymbol{\alpha})}{\partial \alpha_i} = [\boldsymbol{K}(\boldsymbol{K\alpha} - \mathbf{y})]_i + \lambda [\boldsymbol{K\alpha}]_i = 0,$$

so

$$\boldsymbol{K}(\boldsymbol{K\alpha} - \mathbf{y}) + \lambda \boldsymbol{K\alpha} = 0 \qquad \Longrightarrow \qquad \boldsymbol{\alpha} = (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}.$$

Ridge Regression

Defining $\boldsymbol{k}_x = k(x_i, x)$, the final solution is

$$\widehat{f}(x) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}.$$

Ridge Regression

Defining $\boldsymbol{k}_x = k(x_i, x)$, the final solution is

$$\widehat{f}(x) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}.$$

- In this case RRM reduced to just inverting a matrix.

Ridge Regression

Defining $\boldsymbol{k}_x = k(x_i, x)$, the final solution is

$$\widehat{f}(x) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}.$$

- In this case RRM reduced to just inverting a matrix.
- In fact, this is just **ridge regression**, which is a classical method in statistics, and the simplest non-linear regression/interpolation method possible.

Ridge Regression

Defining $\boldsymbol{k}_x = k(x_i, x)$, the final solution is

$$\widehat{f}(x) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \lambda \boldsymbol{I})^{-1} \mathbf{y}.$$

- In this case RRM reduced to just inverting a matrix.
- In fact, this is just **ridge regression**, which is a classical method in statistics, and the simplest non-linear regression/interpolation method possible.
- Ridge regression is the same as the MAP of a Gaussian Process with mean zero and covariance function k .

3. Gaussian Processes

Bayesian nonparametric regression

The canonical regression problem: learn a function $f\colon \mathbb{R}^d \to \mathbb{R}$ from a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$.

Bayesian nonparametric regression

The canonical regression problem: learn a function $f : \mathbb{R}^d \to \mathbb{R}$ from a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$.

The Bayesian way:

1. Assume that $f \sim p_0(f)$ for some appropriate prior p_0

Bayesian nonparametric regression

The canonical regression problem: learn a function $f : \mathbb{R}^d \to \mathbb{R}$ from a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$.

The Bayesian way:

1. Assume that $f \sim p_0(f)$ for some appropriate prior p_0
2. Assume that $y_i \sim p(y_i | f(x_i))$ for some distribution p

Bayesian nonparametric regression

The canonical regression problem: learn a function $f : \mathbb{R}^d \to \mathbb{R}$ from a training set $\mathcal{D} = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$.

The Bayesian way:

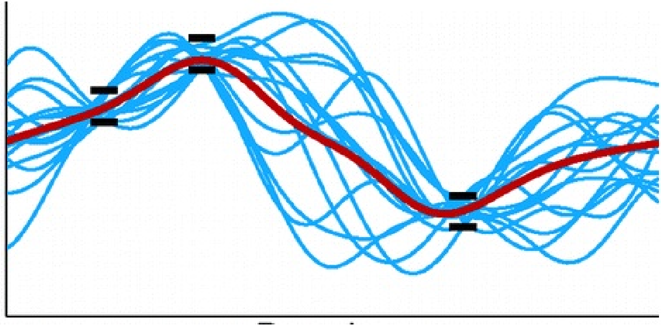
1. Assume that $f \sim p_0(f)$ for some appropriate prior p_0
2. Assume that $y_i \sim p(y_i | f(x_i))$ for some distribution p
3. Use Bayes' rule

$$p(f|\mathcal{D}) = \frac{p(\mathcal{D}|f)\, p_0(f)}{\int_{f'} p(\mathcal{D}|f')\, p_0(f')}$$

with $p(\mathcal{D}|f) = \prod_{i=1}^{m} p(y_i|f(x_i))$.

A prior over functions

The prior p_0 should capture that f is expected to be smooth.



Question: But how does one define a distribution over *functions*?

A prior over functions

IDEA: Assuming that the training points $\{x\}_{i=1}^{m}$ and testing points $\{x'\}_{i=1}^{p}$ are known, just focus on the *marginals*

$$p_0(f(x_1), \ldots, f(x_m), f(x'_1), \ldots, f(x'_p))$$

$$p(f(x_1), \ldots, f(x_m), f(x'_1), \ldots, f(x'_p)|\mathcal{D}).$$

A prior over functions

IDEA: Assuming that the training points $\{x\}_{i=1}^{m}$ and testing points $\{x'\}_{i=1}^{p}$ are known, just focus on the *marginals*

$$p_0(f(x_1), \ldots, f(x_m), f(x'_1), \ldots, f(x'_p))$$

$$p(f(x_1), \ldots, f(x_m), f(x'_1), \ldots, f(x'_p)|\mathcal{D}).$$

A **stochastic process** is a distrubition over functions, usually defined by specifying all possible finite dimensional marginals. $\rightarrow$ Bayesian nonparametrics

Gaussian Processes

Given any (suitably smooth) $\mu\colon \mathcal{X} \to \mathbb{R}$ and a p.s.d. $k\colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $GP(\mu, k)$ is a distribution over functions $f\colon \mathcal{X} \to \mathbb{R}$ such that for any $x_1, \ldots, x_m \in \mathcal{X}$, if $f \sim GP(\mu, k)$, then

$$(f(x_1), \ldots, f(x_m))^\top \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\mu_i = \mu(x_i)$ and $\Sigma_{i,j} = k(x_i, x_j)$.

Gaussian Processes

Given any (suitably smooth) $\mu \colon \mathcal{X} \to \mathbb{R}$ and a p.s.d. $k \colon \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, $GP(\mu, k)$ is a distribution over functions $f \colon \mathcal{X} \to \mathbb{R}$ such that for any $x_1, \dots, x_m \in \mathcal{X}$, if $f \sim GP(\mu, k)$, then

$$(f(x_1), \dots, f(x_m))^\top \sim \mathcal{N}(\boldsymbol{\mu}, \boldsymbol{\Sigma})$$

where $\mu_i = \mu(x_i)$ and $\Sigma_{i,j} = k(x_i, x_j)$.

μ and k are caled the mean and covariance functions of the GP since

$$\mathbb{E}[f(x)] = \mu(x)$$
$$\text{Cov}(f(x), f(x')) = k(x, x')$$

Gaussian Processes

Assume for simplicity that $y \sim \mathcal{N}(f(x), \sigma^2)$.

Gaussian Processes

Assume for simplicity that $y \sim \mathcal{N}(f(x), \sigma^2)$. Then, after observing $\{(x_1, y_1), \ldots, (x_m, y_m)\}$,

$$\mathbb{E}(f(x)) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \sigma^2 I)^{-1} \mathbf{y}$$
$$\text{Var}(f(x)) = \kappa_x - \boldsymbol{k}_x^\top (\boldsymbol{K} + \sigma^2 I)^{-1} \boldsymbol{k}_x$$

where $\mathbf{y} = (y_1, \ldots, y_m), \quad K_{i,j} = k(x_i, x_j), \quad [\boldsymbol{k}_x]_i = k(x_i, x)$, and $\kappa_x = k(x_i, x_i).$

Gaussian Processes

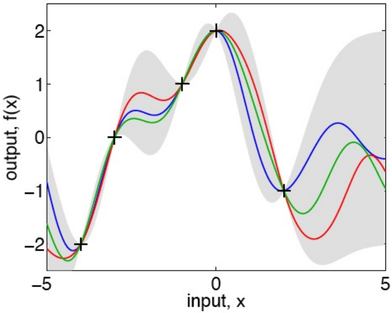
Assume for simplicity that $y \sim \mathcal{N}(f(x), \sigma^2)$. Then, after observing $\{(x_1, y_1), \ldots, (x_m, y_m)\}$,

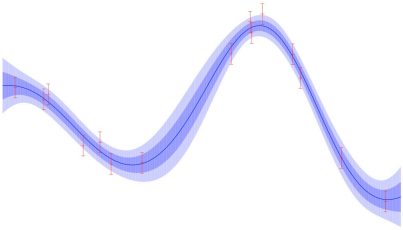
$$\mathbb{E}(f(x)) = \boldsymbol{k}_x^\top (\boldsymbol{K} + \sigma^2 I)^{-1} \mathbf{y}$$
$$\text{Var}(f(x)) = \kappa_x - \boldsymbol{k}_x^\top (\boldsymbol{K} + \sigma^2 I)^{-1} \boldsymbol{k}_x$$

where $\mathbf{y} = (y_1, \ldots, y_m), \quad K_{i,j} = k(x_i, x_j), \quad [\boldsymbol{k}_x]_i = k(x_i, x)$, and $\kappa_x = k(x_i, x_i).$

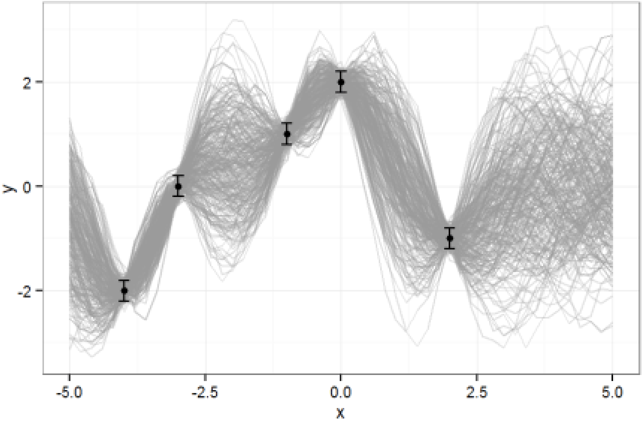
$\rightarrow$ GPs are very easy to use because the maringals and conditionals of Gaussians are also Gaussian.

Gaussian Process





Gaussian Process



One-class SVM and Multiclass SVM

The one-class SVM (outlier detection)

RKHS primal form

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \left[\frac{1}{m} \sum_{i=1}^{m} (1 - f(x_i))_{\geq 0} + \frac{1}{2C} \|f\|^2_{\mathcal{H}_k} \right].$$

The one-class SVM (outlier detection)

RKHS primal form

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \left[\frac{1}{m} \sum_{i=1}^{m} (1 - f(x_i))_{\geq 0} + \frac{1}{2C} \|f\|^2_{\mathcal{H}_k} \right].$$

Tries to peg $f(x_i) \geq 0$ for all points $x_1, \ldots, x_m$ in the training set $\rightarrow$ outlier detector.

The one-class SVM (outlier detection)

RKHS primal form

$$\widehat{f} = \underset{f \in \mathcal{H}_k}{\operatorname{argmin}} \left[\frac{1}{m} \sum_{i=1}^{m} (1 - f(x_i))_{\geq 0} + \frac{1}{2C} \|f\|^2_{\mathcal{H}_k} \right].$$

Tries to peg $f(x_i) \geq 0$ for all points $x_1, \ldots, x_m$ in the training set $\rightarrow$ outlier detector.

Dual form

$$\text{maximize}_{\alpha_1,\ldots,\alpha_m} L(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2} \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j)$$

$$\text{subject to} \quad 0 \leq \alpha_i \leq \frac{C}{m} \ \forall i$$

The Multiclass SVM

- Defining $f_z(x) = zf(x)/2$ for $z = \pm 1$,

$$\ell_{\text{hinge}}(f(x), y) = (1 - yf(x))_{\geq 0} = (1 - (f_y(x) - f_{-y}(x)))_{\geq 0},$$

i.e., the correct answer is supposed to beat the incorrect answer by at least a margin of 1.

The Multiclass SVM

- Defining $f_z(x) = zf(x)/2$ for $z = \pm 1$,

$$\ell_{\text{hinge}}(f(x), y) = (1 - yf(x))_{\geq 0} = (1 - (f_y(x) - f_{-y}(x)))_{\geq 0},$$

  i.e., the correct answer is supposed to beat the incorrect answer by at least a margin of 1.

- This inspires the **multiclass hinge loss**

$$\ell(f_1(x), \ldots f_k(x), y) = \sum_{y' \in \{1,2,\ldots,k\} \setminus \{y\}} \big(1 - (f_y(x) - f_{y'}(x))\big)_{\geq 0},$$

  which is the basis of the k-class SVM ($f_j(x)$ is a bit like a “score”). This is essentially the same notion of multiclass margin as in the k-class perceptron. Predict $\widehat{y} = \operatorname{argmax}_{j \in \mathcal{Y}} f_j(x, j)$.

RKHS form of Multiclass SVM

The loss now depends on not just $f_y(x)$, but also $f_{y'}(x)$ for all $y' \neq y$, so the RKHS form also needs to be generalized slightly:

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \Bigg[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f_1(x_i), f_2(x_i), \ldots, f_k(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}})}_{\text{regularizer}} \Bigg].$$

RKHS form of Multiclass SVM

The loss now depends on not just $f_y(x)$, but also $f_{y'}(x)$ for all $y' \neq y$, so the RKHS form also needs to be generalized slightly:

$$\widehat{f} = \operatorname*{argmin}_{f \in \mathcal{H}_k} \left[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell(f_1(x_i), f_2(x_i), \ldots, f_k(x_i), y_i)}_{\text{training error}} + \underbrace{\Omega(\|f\|_{\mathcal{H}})}_{\text{regularizer}} \right].$$

The corresponding generalized Representer Theorem will say that

$$f_j(x) = \sum_{i=1}^{m} \alpha_{i,j} k(x_i, x)$$

for all $j \in \{1, \ldots, k\}$, so now we have many more coefficients to optimize.

Structured prediction

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$?

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x, y) - f(x, y'))\big)_{\geq 0}.$$

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x,y) - f(x,y'))\big)_{\geq 0}.$$

IDEA: Use this to search for f in a **joint RKHS** of functions $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

- Kernel becomes $k((x,y),(x',y'))$.

Multiclass to Structured Prediction

What if we combine $f_1, \dots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \dots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x, y) - f(x, y'))\big)_{\geq 0}.$$

IDEA: Use this to search for f in a **joint RKHS** of functions $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

- Kernel becomes $k((x, y), (x', y'))$.
- We can now put structure on $\mathcal{Y}$ as well as $\mathcal{X}$.

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x, y) - f(x, y'))\big)_{\geq 0}.$$

IDEA: Use this to search for f in a **joint RKHS** of functions $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

- Kernel becomes $k((x, y), (x', y'))$.
- We can now put structure on $\mathcal{Y}$ as well as $\mathcal{X}$.
- We are now learning a single mapping $f \colon \mathcal{X} \to \mathcal{Y}$ directly.

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x, y) - f(x, y'))\big)_{\geq 0}.$$

IDEA: Use this to search for f in a **joint RKHS** of functions $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

- Kernel becomes $k((x, y), (x', y'))$.
- We can now put structure on $\mathcal{Y}$ as well as $\mathcal{X}$.
- We are now learning a single mapping $f \colon \mathcal{X} \to \mathcal{Y}$ directly.
- At the extreme, the distinction between $\mathcal{X}$ and $\mathcal{Y}$ is blurred.

Multiclass to Structured Prediction

What if we combine $f_1, \ldots, f_k \colon \mathcal{X} \to \mathbb{R}$ in the k-class SVM into a single function $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$ where $\mathcal{Y} = \{1, \ldots, k\}$? The loss becomes

$$\ell(f, x, y) = \sum_{y' \in \mathcal{Y} \setminus \{y\}} \big(1 - (f(x, y) - f(x, y'))\big)_{\geq 0}.$$

IDEA: Use this to search for f in a **joint RKHS** of functions $f \colon \mathcal{X} \times \mathcal{Y} \to \mathbb{R}$:

- Kernel becomes $k((x, y), (x', y'))$.
- We can now put structure on $\mathcal{Y}$ as well as $\mathcal{X}$.
- We are now learning a single mapping $f \colon \mathcal{X} \to \mathcal{Y}$ directly.
- At the extreme, the distinction between $\mathcal{X}$ and $\mathcal{Y}$ is blurred.

$\to$ **structured prediction**

RRM form of Structured Prediction

Let k be a psd kernel $k\colon (\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}$, let $\mathcal{H}_k$ be the corresponding RKHS, and Ω a monotonically increasing function. Solve

$$\widehat{f} = \arg\min_{f\in\mathcal{H}_k}\Big[\underbrace{\frac{1}{m}\sum_{i=1}^{m}\ell((f(x_i,y))_{y\in\mathcal{Y}},y_i)}_{\text{training error}} + \underbrace{\Omega\|f\|_{\mathcal{F}}}_{\text{regularizer}}\Big],$$

and predict $\widehat{y} = \operatorname{argmax}_{y\in\mathcal{Y}}\widehat{f}(x,y)$.

RRM form of Structured Prediction

Let k be a psd kernel $k\colon (\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}$, let $\mathcal{H}_k$ be the corresponding RKHS, and Ω a monotonically increasing function. Solve

$$\widehat{f} = \arg\min_{f\in\mathcal{H}_k}\Big[\underbrace{\frac{1}{m}\sum_{i=1}^{m}\ell((f(x_i,y))_{y\in\mathcal{Y}},y_i)}_{\text{training error}} + \underbrace{\Omega\|f\|_{\mathcal{F}}}_{\text{regularizer}}\Big],$$

and predict $\widehat{y} = \operatorname{argmax}_{y\in\mathcal{Y}}\widehat{f}(x,y)$.

The loss (in principle) now depends on $f(x_i,y)$ for all possible y.

RRM form of Structured Prediction

Let k be a psd kernel $k\colon (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$, let $\mathcal{H}_k$ be the corresponding RKHS, and Ω a monotonically increasing function. Solve

$$\widehat{f} = \arg\min_{f \in \mathcal{H}_k} \Bigg[\underbrace{\frac{1}{m} \sum_{i=1}^{m} \ell((f(x_i, y))_{y \in \mathcal{Y}}, y_i)}_{\text{training error}} + \underbrace{\Omega \|f\|_{\mathcal{F}}}_{\text{regularizer}} \Bigg],$$

and predict $\widehat{y} = \mathrm{argmax}_{y \in \mathcal{Y}} \widehat{f}(x, y)$.

The loss (in principle) now depends on $f(x_i, y)$ for all possible y. Therefore, the Representer Theorem says that f is of the form

$$f(x, y) = \sum_{i=1}^{m} \sum_{y^* \in \mathcal{Y}} \alpha_{i,y^*} k((x_i, y^*), (x, y)).$$

RRM form of Structured Prediction

Let k be a psd kernel $k\colon (\mathcal{X}\times\mathcal{Y})\times(\mathcal{X}\times\mathcal{Y})\to\mathbb{R}$, let $\mathcal{H}_k$ be the corresponding RKHS, and Ω a monotonically increasing function. Solve

$$\widehat{f} = \arg\min_{f\in\mathcal{H}_k}\Bigg[\underbrace{\frac{1}{m}\sum_{i=1}^{m}\ell((f(x_i,y))_{y\in\mathcal{Y}},y_i)}_{\text{training error}} + \underbrace{\Omega\|f\|_{\mathcal{F}}}_{\text{regularizer}}\Bigg],$$

and predict $\widehat{y} = \mathrm{argmax}_{y\in\mathcal{Y}}\,\widehat{f}(x,y)$.

The loss (in principle) now depends on $f(x_i,y)$ for all possible y. Therefore, the Representer Theorem says that f is of the form

$$f(x,y) = \sum_{i=1}^{m}\sum_{y^*\in\mathcal{Y}}\alpha_{i,y^*}\,k((x_i,y^*),(x,y)).$$

In practice, this is usually unfeasible, so only add α_{i,y^*} coefficients to the optimization on the fly “as needed”.

Kernels for Structured Learning

The simplest way to get a kernel $k\colon (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$:

- Get a kernel $k_{\mathcal{X}}$ that quantifies similarity between the x 's.

Kernels for Structured Learning

The simplest way to get a kernel $k\colon (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$:

- Get a kernel $k_{\mathcal{X}}$ that quantifies similarity between the x 's.
- Get a kernel $k_{\mathcal{Y}}$ that quantifies similarity between the y 's.

Kernels for Structured Learning

The simplest way to get a kernel $k\colon (\mathcal{X} \times \mathcal{Y}) \times (\mathcal{X} \times \mathcal{Y}) \to \mathbb{R}$:

- Get a kernel $k_{\mathcal{X}}$ that quantifies similarity between the x 's.
- Get a kernel $k_{\mathcal{Y}}$ that quantifies similarity between the y 's.
- Define

$$k((x, y), (x', y')) = k_{\mathcal{X}}(x, x') \cdot k_{\mathcal{Y}}(y, y').$$

Question: Is this a valid kernel? What is its RKHS?