Statistical Techniques in Robotics (16-831, F12) Lecture #22 (Nov 19, 2012)

Kernel Machines/ Functional Gradient Descent

Lecturer: Drew Bagnell *Scribe: Liz Cha*[1]

1 Review of Kernels and Functional Gradients

1.1 Kernels

- Ultimately, we wish to learn a function $f : \mathbb{R}^n \rightarrow \mathbb{R}$ that assigns a meaningful score given a data point. For example, in binary classification, we would like an $f(\cdot)$ to return positive and negative values, given positive and negative samples, respectively.
- A kernel $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$ intuitively measures the *correlation* between $f(\mathbf{x_i})$ and $f(\mathbf{x_j})$. Considering a matrix $\mathbf{K}$ with entries $K_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$, then matrix $\mathbf{K}$ must satisfy the properties:
  - $\mathbf{K}$ is symmetric ($K_{ij} = K_{ji}$)
  - $\mathbf{K}$ is positive-definite ($\forall \mathbf{x} \in \mathbb{R}^n : \mathbf{x} \neq \mathbf{0}, \mathbf{x^T K x} > 0$)

  Hence, a valid kernel is the inner product: $K_{ij} = \langle \mathbf{x_i}, \mathbf{x_j} \rangle$.
- A function can be considered that is a weighted composition of many kernels centered at various locations $\mathbf{x_i}$:

$$f(\cdot) = \sum_{i=1}^{Q} \alpha_i K(\mathbf{x_i}, \cdot), \tag{1}$$

  where Q is the number of kernels that compose $f(\cdot)$ and $\alpha_i \in \mathbb{R}$ is each kernel's associated weight. All functions $f(\cdot)$ with kernel K that satisfy the above properties and can be written in the form of Equation 1 are said to lie in a *Reproducing Kernel Hilbert Space* (RKHS) $\mathcal{H}_K$: $f \in \mathcal{H}_K$

However to do gradient descent on the space of such functions, we need the notion of a distance, norm and an inner product. We formalize this by introducing the Reproducing Kernel Hilbert Space.

1.2 Functional gradient

A gradient can be thought of as:

- Vector of partial derivatives

[1] Based on the scribe work of Abhinav Shrivastava, Varun Ramakrishna, Dave Rollinson, Daniel Munoz, Tomas Simon, Jack Singleton and Sergio Valcarcel

- Direction of steepest ascent
- Linear approximation of the function (or functional), ie. $f(x_0+\epsilon) = f(x_0)+\epsilon\cdot\underbrace{\nabla f(x_0)}_{gradient}+O(\epsilon^2)$.

We will use the third definition. A *functional* $E : f \to \mathbb{R}$ is a function of functions $f \in \mathcal{H}_K$. As an example let us write the terms of our loss function from above as functionals:

- $E_1[f] = ||f||^2$
- $E_2[f] = (y - f(\mathbf{x}))^2$
- $E[f] = \sum_i (y_i - f(\mathbf{x}_i))^2 + \lambda||f||^2$

A functional gradient $\nabla E[f]$ is defined implictly as the linear term of the change in a function due to a small perturbation ϵ in its input: $E[f + \epsilon g] = E[f] + \epsilon\langle\nabla E[f], g\rangle + O(\epsilon^2)$.

Before computing the gradients for these functionals, let us look at a few tools that will help us derive the gradient of the loss functional

1.3 Chain rule for functional gradients

Consider *differentiable* functions $C : \mathbb{R} \to \mathbb{R}$ that are functions of functionals G, $C(G[f])$. Our cost function $L[f]$ from before was such a function, these are precisely the functions that we are interested in minimizing.

The derivative of these functions follows the chain rule:

$$\nabla C(G[f]) = \frac{\partial C(G[f])}{\partial \lambda}|_{G(f)} \nabla G[f] \tag{2}$$

Example: If $C = (||f||^2)^3$, then $\nabla C = 3(||f||^2)^2(2f)$

1.4 Another useful functional gradient

Another useful functio nthat we come across often is the evaluation functional. The *evaluation functional* evaluates f at the specified x: $E_x[f] = f(x)$

- Gradient is $\nabla E_x = K(x, \cdot)$

$$\begin{aligned} E_x[f + \epsilon g] &= f(x) + \epsilon g(x) + 0 \\ &= f(x) + \epsilon\langle K(x, \cdot), g\rangle + 0 \\ &= E_x[f] + \epsilon\langle\nabla E_x, g\rangle + O(\epsilon^2) \end{aligned}$$

- It is called a *linear functional* due to the lack of a multiplier on perturbation ϵ.

1.5 Functional gradient of the regularized least squares loss function

- Let's look at the functional gradient of the second term of the loss function:

$$\nabla E[f] = \nabla ||f||^2 \tag{3}$$

Expanding it out using a Taylor's series type expansion

$$\begin{aligned} E[f + \epsilon g] &= \langle f + \epsilon g, f + \epsilon g \rangle \\ &= ||f|| + 2\langle f, \epsilon g \rangle + \epsilon^2 ||g|| \\ &= ||f|| + \epsilon \langle 2f, g \rangle + O(\epsilon^2) \end{aligned}$$

We observe that

$$\nabla E[f] = \nabla ||f||^2 = 2f \tag{4}$$

- Now for the first term of the loss function

$$E[f] = \sum_i (y_i - f(\mathbf{x}_i))^2 \tag{5}$$

Using the chain rule we have

$$\nabla E[f] = -2(y_i - f(\mathbf{x}_i)) \nabla (f(x_i)) \tag{6}$$

We observe that $\nabla(f(x_i))$ is the functional gradient of the evaluation functional. Substituting in the gradient of the evaluaton functional as computed in the previous section we have :

$$\nabla E[f] = -2(y_i - f(\mathbf{x}_i)) K(\mathbf{x}_i, \cdot) \tag{7}$$

2 Functional gradient descent

- Regularized least squares loss function $L[f]$

$$\begin{aligned} L[f] &= (y_i - f(\mathbf{x}_i))^2 + \lambda ||f||^2 \\ L[f] &= (y_i - E_{\mathbf{x}_i}[f])^2 + \lambda ||f||^2 \\ \nabla L[f] &= -2(y_i - f(\mathbf{x}_i)) K(\mathbf{x}_i, \cdot) + 2\lambda f \end{aligned}$$

Update rule for the regularized least squares loss function:

$$\begin{aligned} f_{t+1} &\leftarrow f_t - \eta_t \nabla L \\ &\leftarrow f_t - \eta_t(-2(y_t - f_t(\mathbf{x}_t)) K(\mathbf{x}_t, \cdot) + 2\lambda f_t) \\ &\leftarrow f_t(1 - 2\eta_t \lambda) + 2\eta_t (y_t - f_t(\mathbf{x}_t)) K(\mathbf{x}_t, \cdot) \end{aligned}$$

where η_t is the learning rate at time step t.

The update rule is equivalent to:

  - Adding a kernel $K(\mathbf{x}_t, \cdot)$ weighted by $2\eta_t(y_t - f_t(\mathbf{x}_t))$.
  - Shrinking all other weights by $(1 - 2\eta_t \lambda)$ multiplier.

- SVM loss function $L(f)$

$$L(f(\mathbf{x}_t), y_t) = \max(0, 1 - y_t f(\mathbf{x}_t)) + \lambda ||f||^2 \tag{8}$$

The sub-gradient ∇L has two cases. One where the prediction is correct by margin = 1, and the other where is not correct by margin = 1 (margin error).

$$\nabla L((x_t), y_t) = \begin{cases} 0 & \text{if } (1 - y_i f(\mathbf{x}_i)) \leq 0 \text{, correct by margin} \\ L'(f(\mathbf{x}_t), y_t) f'(\mathbf{x}_t) = -y_t K(\mathbf{x}_t, \cdot) & \text{else margin error} \end{cases} \tag{9}$$

The update rule is equivalent to:

  - Adding a kernel $K(\mathbf{x}_t, \cdot)$ weighted by $\eta_t y_t$ in case of margin error.
  - Shrinking all other weights by $(1 - 2\eta_t \lambda)$ multiplier.

What is the square loss for the linear predictor?

$$L_t(w) = \lambda ||w||_2 + (w^t x_t - y_t)^2 \tag{10}$$

We want to control complexity- aka penalize the size of the function? What does the loss function for SVM look like– hinge loss.

If we run this repeatedly with different L's,

  - only get kernels when we make mistakes
  - once you start getting it right, weights shrink
  - only end up iwth kernels at places called support vectors

3 Online Kernel Machine

- Initialize the function $f = 0$.
- For t = 1 to T:
  1. Observe some measurement over some set of features x_t
  2. Predict the class using $f(x_t) = \sum_{i=1}^{n} \alpha_i K(x_i, x_t)$
  3. Receive loss based on the prediction from $f(x_t)$ and the true class y_t

     $$L(f(x_t), y_t)$$

  4. Update f based on the gradient of the loss function L and learning rate η_t depending on the chosen algorithm (examples in previous section).

3.1 Discussion

- **Representer Theorem** (informally): Given a loss function and regularizer objective with many data points $\{x_i\}$, the minimizing solution f^* can be represented as
$$f^*(\cdot) = \sum_i \alpha_i K(x_i, \cdot) \tag{11}$$

- This algorithm qualitatively corresponds to adding weighted 'bumps' that predicts some value based on the kernel function in each new observation's neighborhood of the feature space in x. For example: Figure 1 shows an update over 3 points $\{(x_1,+),(x_2,-),(x_3,+)\}$. The individual kernels centered at the points are **independently** drawn with colored lines. After 3 updates, the function f looks like the solid black line.

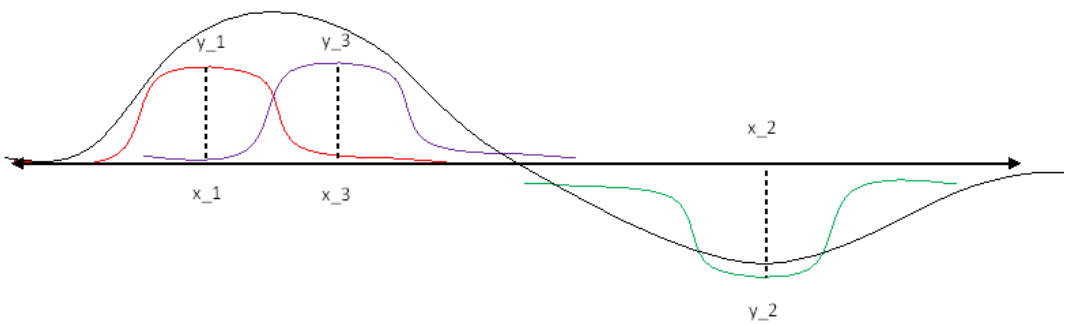

Figure 1: Illustration of function after 3 updates

- Need to perform $O(T)$ work at each time step. As time progresses and the data set grows, the prediction step will take longer and longer to compute. To shorten this computation time you may want to throw out old data points by weight or age. If interested, there are some papers on that use tricks to find sparse solutions to large-scale problems:
  - Rahimi and Recht - *Random Features for Large-Scale Kernel Machines* 2007
  - Dekel, Shalev-Shwartz and Singer *The Forgetron: A Kernel-Based Perceptron on a Budget* 2007

- The regret is computed as:
$$Regret = \sum_t (C_t(f_t(x_t)) - C_t(f^*(x_t))) \mid f^* \in H_k$$
The regret bound:
$$Regret = ||\nabla C_t(f)||_k \cdot ||f^*||_k \sqrt{T}$$
$||f^*||$ is the size of the function. $||\nabla C_t(f)||$ can get as big as $\alpha^T K \alpha$.

- The choice and tuning of the kernel and their corresponding bandwidth parameters are what affect the bias-variance tradeoff. These are parameters that need to be tuned in addition to the learning rate η and decay rate λ from the update equations.
  - Often simple kernels work quite well. When approaching a new problem it is usually a good idea start with linear or polynomial kernels. Radial basis functions are another good kernel to try early on. Note that any kernels K_1 and K_2 that satisfy the conditions mentioned in Section 2 can be summed to form a new valid kernel.