Statistical Techniques in Robotics (16-831, F09) Lecture #23 11/10/2009

Kernel Review and Novelty Detection

Lecturer: Drew Bagnell *Scribe: Jack Singleton*

1 Review Online Kernel Machine

- inner product:
$$< f, g >= \sum_i \sum_j \alpha_i \beta_j K(x_i, x_j) = \alpha^T K \beta$$
- $f \epsilon H_k, f = \sum \alpha_i K(x_i, \cdot)$
- K is the Kernel Matrix
  - K is positive definite (ex: $K(x_1, x_2) = x_1^T x_2$)
  - The bigger the kernel width, the smoother the output and larger the computations
  - The smaller the kernel width, the more regret we will pay because all points will be constrained.
- Do online gradient descent with functions:
  1. Initialize
  $$f = 0$$
  2. Receive
  $$x_t$$
  3. Predict using
  $$f(x_t) = \sum_{i=1}^{n} \alpha_i K(x_i, x_t)$$
  4. Receive
  $$y_t = C_t(f(x_t))$$
  5. Update
  $$f_{t+1} \leftarrow f_t + \eta_{t+1} C'(f(x_t)) K(x_i, x_t)$$
  Math sidenote: derivative computed with chain rule:
  $$\frac{dC}{dx} = C'(f(x_t)) \cdot f'(x_t) = C'(f(x_t)) \cdot K(x_i, x_t)$$
  6. Shrink weights by $(1 - \eta_t \lambda)$
- 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. In class we talked about throwing out points by age or by weight.

- The regret is computed with:

$$Regret = \sum_t (C_t(f_t(x_t)) - C_t(f^*(x_t)))|f^* \in H_k$$

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 $\alpha^T K \alpha$.

- **Representer Theorem**: For every loss function of the form $\sum C(f(x_i)) + \gamma||f||^2$ the optimal solution, f^*, is in the span of $K(x_i, \cdot)$.

(informally): Given a loss function and regularizer objective with many data points $\{x_i\}$, the minimizing solution f^* can be represented as

$$f^*(\cdot) = \sum_i \alpha_i K(x_i, \cdot) \tag{1}$$

- Alternate idea from class: perform gradient descent in the space of α coefficients: $\nabla_\alpha L$
  - Takes n^2 iterations to get same performance (n = number of iterations of functional gradient descent)
  - Every iteration is $O(T^2)$

2 Novelty Detection

- Call something "not novel" if $f \geq threshold$
- SVM novelty loss

$$\mathcal{L} = max(0, \gamma - f(x))$$

if $f(x) > \gamma$

$$\nabla \mathcal{L} = 0$$

else

$$f = f + \alpha K(x_i, \cdot)$$

Tricks

1. Use supervised learning to pick kernel
2. Race to be not novel: if you run out of time to declare something not novel, then just call it novel and move on.
3. kd-tree
4. move-to-front