Statistical Techniques in Robotics (16-831, F10) Lecture #13 (Tuesday October 5th)

Learning by Constraints and SVMs

Lecturer: Drew Bagnell *Scribe: Chris Skonieczny*[1]

We have seen that Online Convex Programming (OCP) has many applications, from optimizing investment portfolios to inferring terrain supporting surfaces in front of robots. We shall now explore "support vector" methods, which are OCP for learning. The connection will be made by:

- Thinking of data as (soft) constraints
- Turning these constraints into objective functions

1 Support Vector Applications

1.1 "Little Dog" walking robot

One example of a support vector application is crossing terrain with the robot Little Dog. At a high level, a cost function can be defined that considers how long it takes the robot to cross the terrain and how close it comes to tipping over. At the level of planning individual footsteps, though, such a cost function does not apply. It can, in fact, be difficult to determine the cost function for footstep planning.

One method for determining the cost function is to utilize human expertise. Since guessing the cost function is hard, a human user is given two pieces of terrain for comparison with the option of choosing which would be better for robot foot placement. An steep slope would be considered expensive, while shallow convex divets, that can serve as stable footholds, would be cheap. A cost function can then be generated from the learned data, over a few hundred examples, using regression. Possible features in this particular example include terrain filter response, slope and curvature, triangle of support, etc.

1.2 Sports Examples

This technique for learning cost functions can be extended to other problems. For example, there are continuous or discrete rankings of teams in sports. In football and basketball rankings are created to decide which team might be 'better', or more likely to win. Cost features can be generated for two teams through their pairwise comparison.

2 Implementation

For each example where one instance is judged to be preferable to another, a constraint can be formed as $w^T f^p \leq w^T f^n$ where w is a vector of weights, f is a feature set, and the preferred and

[1]Some content adapted from previous scribes: Lindsey Hines

not preferred instances are denoted by superscripts p and n, respectively.

Multiple constraints, indexed $i = 1, ..., T$, can then be compiled in the form:
$$\begin{aligned} w^T f_1^p &\leq w^T f_1^n \\ w^T f_2^p &\leq w^T f_2^n \\ &\dots \\ w^T f_T^p &\leq w^T f_T^n \end{aligned}$$

There are several problems, however, with the constraints as they are now written.

1. There may not be a set of weights that satisfy all of the constraints. The human trainer may not have been consistent. (Eg. non-transitive ranking: $w^T f^1 \leq w^T f^2$, $w^T f^2 \leq w^T f^3$, $w^T f^3 \leq w^T f^1$)
2. There can be multiple solutions; weights can be scaled and still satisfy the constraints.
3. There is a trivial solution of $w = 0$. (Note: we don't want to constrain $\|w\| \geq r$, as this results in a non-convex problem)

Maximum Margin Approach
Both problem 2 and problem 3 can be addressed with a maximum margin approach. By adding a constant to the inequalities, one must not only satisfy them, but do so by a margin. One can then attempt to maximize the size of the margin as suggested, subject to all the constraints and $\|w\|^2 \leq 1$. It is equivalent, though, and simpler, to fix the margins to a constant value and minimize the weights. The constraints can then be reformatted as follows:

Objective function: $min||w||^2$, subject to:

$$\begin{aligned} &\dots \\ w^T f_i^p &\leq w^T f_i^n - 1 \\ w^T f_{i+1}^p &\leq w^T f_{i+1}^n - 1 \\ &\dots \\ w^T f_T^p &\leq w^T f_T^n - 1 \end{aligned}$$

Constraint Softening
Problem number 1 can be addressed with the addition of a slack variable ξ_i, where $\xi_i \geq 0$. If a judgment is wrong, the slack variable can be increased until the constraint is satisfied, though a price is paid with an increase of the total cost. The objective function and constraints are now:

Objective function: $\min\limits_{w,\xi} \frac{\lambda}{2}||w||^2 + \sum\limits_{i=1}^{T} \xi_i$, subject to:

$$\begin{aligned} &\dots \\ w^T f_i^p &\leq w^T f_i^n - 1 + \xi_i \\ w^T f_{i+1}^p &\leq w^T f_{i+1}^n - 1 + \xi_{i+1} \\ &\dots \\ w^T f_T^p &\leq w^T f_T^n - 1 + \xi_T \end{aligned}$$

where lambda trades off cost of violating constraints vs. having high weights:
$\lambda \leftarrow$ smaller = larger (growing) set of weights, longer to learn but violate less contraints.
$\lambda \leftarrow$ larger = smaller (constrained) set of weights, learn initially faster.
Larger weights = smaller margin, and smaller weights = larger margin.

Note that we can also write each constraint in terms of the slack variable:

$\xi_i \geq w^T \Delta f_i + 1$ where $\Delta f_i = f_i^p - f_i^n$.

Quadratic Programming Considered Harmful: Creating Online Support Vector Ranking

The maximum margin approaches discussed above (for either hard or softened constraints) minimize an objective function quadratic in w (and ξ) subject to constraints linear in w (and ξ). This is the classical setup of a quadratic programming problem. In practice, solving such problems using QP can take a considerable amount of time. We can, instead, turn it into an online problem, eliminating the constraints. The resulting approach, called Support Vector Ranking, is easy to implement and very fast; it can also be applied in an online setting, if required.

Considering again cases with conflicting constraints, we can define a bound on our slack variables ξ_i. If a constraint is met within the margin, ξ_i is not needed and is equal to zero. If $w^T(\Delta f_i) + 1 > 0$, where the judgement is wrong or not within the margin, ξ_i is required, but will never need to be greater than $\xi_i = w^T(\Delta f_i) + 1$. At that point, the constraint is already met with margin, and any further increase would only increase cost. The objective function can now be written as:

$$\min_w \{\frac{\lambda}{2}||w||^2 + \sum_{i=1}^{T} max(0, w^T \Delta f_i + 1)\}$$

or equivalently:

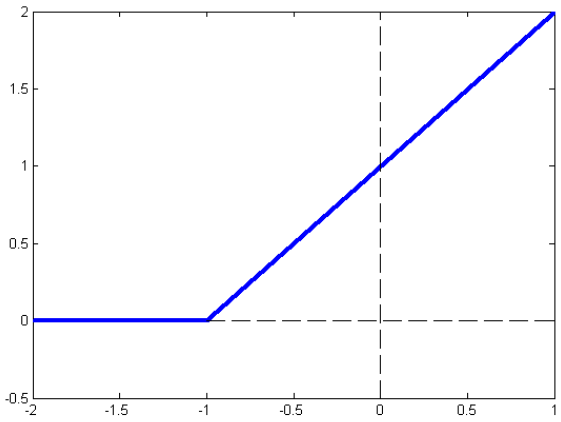
$$\min \sum_{i=1}^{T} \{\frac{\lambda^*}{2}||w||^2 + max(0, w^T \Delta f_i + 1)\}$$

Where $\lambda^* = \frac{\lambda}{T}$. We can drop the star notation and note that λ has absorbed T.

At each time step (i.e. for each training example), our loss function is now:

$l_t = \frac{\lambda}{2}||w||^2 + max(0, w^T \Delta f_t + 1)$

This is called the hinge loss, and is convex (in fact, it is strongly convex) and entirely unconstrained! It is not differentiable, but subgradients exist everywhere. As can be seen in figure below, the subgradient of the hinge loss can fall within one of three cases. When $w^T \Delta f_i < -1$ (corresponding to correctly ranked examples, constraint i is satisfied with margin), the subgradient is λw. When $w^T \Delta f_i > -1$ the subgradient is $\lambda w + \Delta f_i$. The third case, $w^T \Delta f_i = -1$ can be placed in either of the two previous cases, since there are multiple valid subgradients.



3 Online Support Vector Ranking Algorithm Outline

- Step 1: $w- = \alpha_t \lambda w$ (This step attempts to increase margin)
- Step 2: If a misranking occurred (or the constraint wasn't satisfied by the margin),
  $w- = \alpha_t(f_t^p - f_t^n)$ (where we recall that p denotes the preferred instance, and n the not-preffered instance)
  Else, nothing
- Projection is not required because there are no contraints left in the reformulated problem.
- Iterate - May need to pass through data multiple times and data order should be randomized.