Statistical Techniques in Robotics (16-831, F14) Lecture #15 (Wednesday October 21$^{\text{st}}$)

Learning by Constraints and SVMs

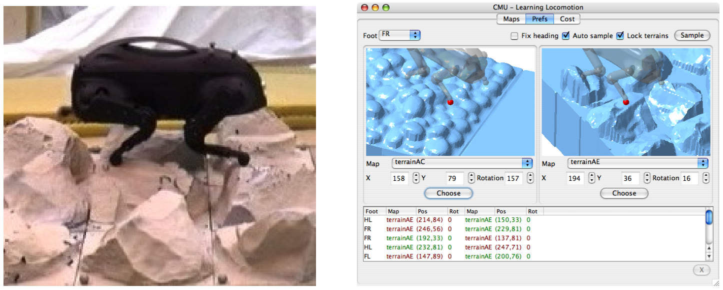
Lecturer: Drew Bagnell *Scribe: Annie Holladay, Sanjiban Choudhury* [1]

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 - A Ranking problem



Foot FR

□ Fix heading ☑ Auto sample ☑ Lock terrains

Map terrainAC — X 158 Y 79 Rotation 157

Map terrainAE — X 194 Y 36 Rotation 16

Foot	Map	Pos	Rot	Map	Pos	Rot
HL	terrainAE	(214,84)	0	terrainAE	(150,33)	0
FR	terrainAE	(246,56)	0	terrainAE	(229,81)	0
FR	terrainAE	(192,33)	0	terrainAE	(137,81)	0
HL	terrainAE	(232,81)	0	terrainAE	(247,71)	0
FL	terrainAE	(147,89)	0	terrainAE	(200,76)	0

(a) (b)

Figure 1: (a) Rough terrain navigation by Little Dog robot (b) Expert provides preference (left v/s right) of which instance should have lower cost and the system learns a cost function over terrain features

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 as shown in Figure **??** (a), though, such a cost function does not apply. It can, in fact, be difficult to determine the cost function for footstep planning.

A place to start would be to enlist the *features* on which the cost function may depend on such as

- Height of new location form current foot location
- Distance of new location from current foot location
- Stability of new configuration (using the support triangle)

[1] Some content adapted from previous scribes: Karthik Lakshmanan

- Local concavity of new location

These features can be stacked into a feature vector f. We can now assume a *linear* cost function structure $w^T f$. The problem then is to determine the right w such that better footstep locations score higher than worse ones. 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 as shown in Figure **??** (b). A cost function can then be generated from the learned data, over a few hundred examples, using regression. We try to find a w such that the rankings made by our goodness metric is as consistent as possible with the human's judgement.

1.2 Sports Examples (College Football)

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.

Example features, which could be used to predict the outcome of a game, could include:

- Number of wins in previous year
- Number of returning All-American players on the team
- Average points per game of the team
- Human ranking of the team determined by the Associated Press (AP) poll
- Booster club budget
- etc...

2 Implementation

For each example where one instance is judged to be preferable to another, a constraint can be formed as $w^T f^n \leq w^T f^p$ where w is a vector of weights, f is a feature set, and the preferred and not preferred instances are denoted by superscripts n and p, respectively.

Multiple constraints, indexed $i = 1, ..., T$, can then be compiled in the form:

$$\begin{gathered} w^T f_1^n \leq w^T f_1^p \\ w^T f_2^n \leq w^T f_2^p \\ \vdots \\ w^T f_T^n \leq w^T f_T^p \end{gathered}$$

There are several issues, 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: $w^T f^1 \leq w^T f^2$, $w^T f^2 \leq w^T f^3$, $w^T f^3 \leq w^T f^1$ Think of the case where Team A beats Team B, Team B beats C and C beats A.)
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)

You might be tempted to fix these problems with hacks. Drew does not like hacks.

1. "Can we make the inequality strict to alleviate problem 2?" This often leads to an ill-posed optimization problem. Often the objective might have a minima on the boundary of the feasible set - if the boundary is not included (as is the case with a hard inequality), the solution is undefined. For example, the problem $\min x^2$ such that $x > 0$ is ill-posed.
2. "Can we fix problem 2 by excluding 0?" This makes the set non-convex (the feasible space is $\mathbb{R}^n - \{0\}$).

Adding a margin

Both issue 2 and issue 3 can be addressed by introducing a margin of 1. This implies

$$w^T f^p \geq w^T f^n + 1$$

The choice of 1 here is arbitrary. Choosing another constant simply means rescaling w.

An open question at this point is what to optimize? If we go back to the OCP formulation, the regret bounded depended on the size of the convex set we were optimizing over. Minimizing the size of this set, minimizes the regret. Here the size is $||w||$. Hence a plausible formulation can be Objective function: $\min ||w||^2$, subject to:

$$\begin{aligned} w^T f_1^n &\leq w^T f_1^p - 1 \\ w^T f_2^n &\leq w^T f_2^p - 1 \\ &\vdots \\ w^T f_T^n &\leq w^T f_T^p - 1 \end{aligned}$$

This is known as Hard Margin Support Vector Ranking. Observe that the set of constraint forms the intersection of a unit ball and a number of half planes, which make it a convex problem.

Constraint Softening

Issue 1 is still an open question - how does one deal with violations of the constraint? One suggestion is to minimize the number of violations. This is a very hard combinatorial optimization problem which might be very expensive to solve. Also, it is not obvious that this is a criteria one would want to optimize. If two teams are very close to each other, who is better shouldnt change the ranking function dramatically.

Issue 1 can be addressed with the addition of a slack variable ξ_i, where $\xi_i \geq 0$. This says that it is ok for constraint i to be satisfied by a margin that is ξ_i less than 1. 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: $\min_{w,\xi}\left(\frac{\lambda}{2}||w||^2 + \sum_{i=1}^{T}\xi_i\right)$, subject to:

$$\begin{aligned} w^T f_1^n &\leq w^T f_1^p - 1 + \xi_1 \\ w^T f_2^n &\leq w^T f_2^p - 1 + \xi_2 \\ &\vdots \\ w^T f_T^n &\leq w^T f_T^p - 1 + \xi_T \end{aligned}$$

where lambda trades off cost of violating constraints vs. having high weights:
Smaller $\lambda \rightarrow$ allows larger (growing) set of weights, longer to learn but violate less constraints.
Larger $\lambda \rightarrow$ forces 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 $\Delta f_i = f_i^n - f_i^p$.

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 (QP). In practice, solving such problems using QP can take a considerable amount of time since the time scales cubically with the number of constraints. 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:

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

or equivalently:

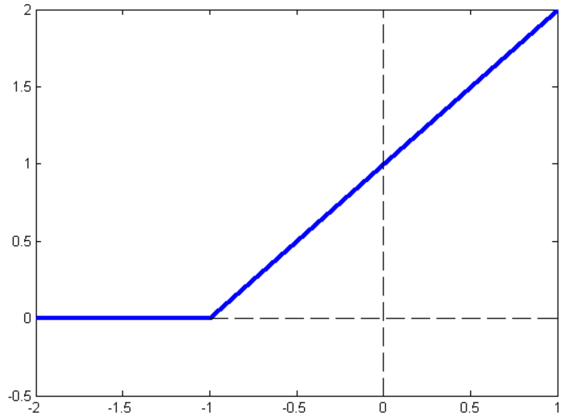
$$\min \sum_{i=1}^{T} \left(\frac{\lambda^*}{2} ||w||^2 + max(0, w^T \Delta f_i + 1) \right)$$

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

```
Set $w_0 = 0$
for $i$ = 1,2 ..... $T$ do
    $w_{t+1} = w_t - \alpha_t \lambda w_t$ (tries to increase margin)
    if a misranking occurred or constraint not satisfied by margin then
        $w_{t+1} = w^*_{t+1} - \alpha_t (f^n_t - f^p_t)$
    else
        $w_{t+1} = w^*_{t+1}$
    end
end
```

Note:

- Projection is not required because there are no constraints left in the reformulated problem.
- Iterate - May need to pass through data multiple times and data order should be randomized.

- $\alpha_t = \frac{1}{\sqrt{t}}$ gives us no average regret. In fact, this is strongly convex and $\alpha_t \propto \frac{1}{t}$ also gives no regret!

4 Linear SVMs

We begin by considering binary, linear classification. In this problem, we are trying to map elements from our feature space into the set $\{-1, 1\}$. When we have two features, $F_1, F_2 \in \mathbb{R}$, we can think of the problem graphically. In the Figure **??**, we represent the points with a label of 1 as O's, we the points with a label of -1 as X's. We wish to find a linear decision boundary (a straight line) that separates the points from each class. The decision boundary is shown as the dotted line.

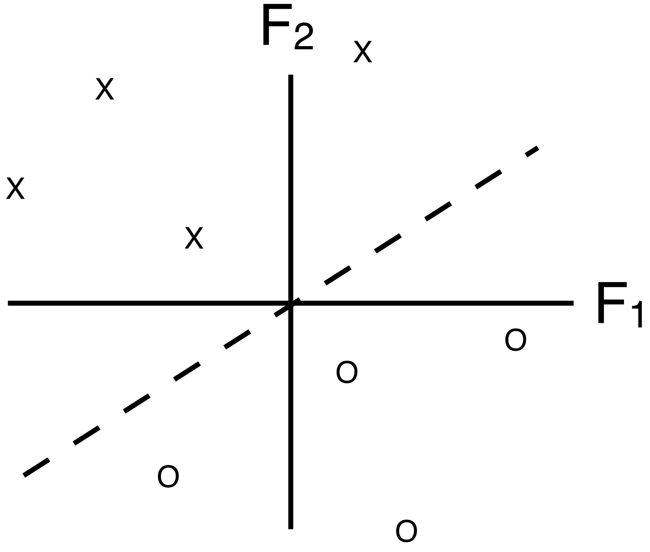


Figure 2: Linear binary classification

To represent this notion mathematically, we denote the i^{th} training point as f_i, and the class of this point as y_i. The linear classification problem requires us to find a set of weights w such that:

$$\forall i \ y_i w^T f_i \geq 0 \tag{1}$$

Note that the decision boundary need not pass through the origin. We often use a dummy variable $F_0 = c$ for constant $c \neq 0$, which allows us to use w_0 as an offset.

One issue with this formulation of the problem is that it has a trivial solution. Specifically, if we set all of our weights to 0, we will have $y_i w^T f_i = 0$ for all points. To address this issue, we modify our constraints to be:

$$\forall i \ y_i w^T f_i \geq margin \tag{2}$$

This can have the added benefit of improving generalization. Once again, we can represent this graphically. In the Figure **??**, the space between the dotted line and the decision boundary represents the margin.

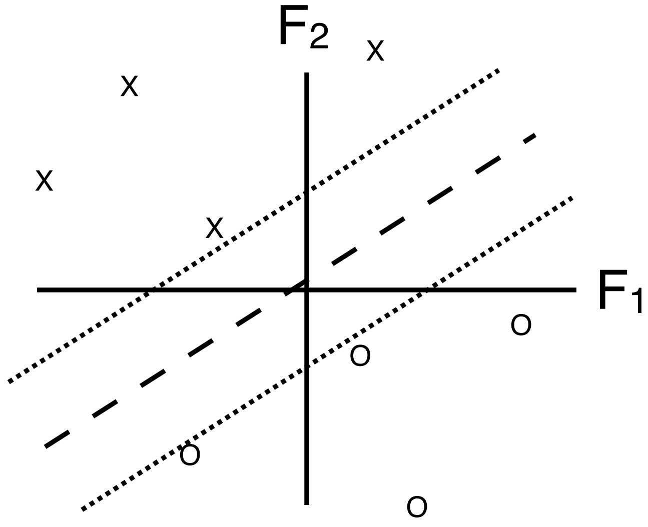


Figure 3: Linear binary classification with margins

We further see that the magnitude of our weight vector does not matter, only the orientation affects classification. Therefore we can restrict w to $||w||^2 = 1$. Taken together, this yields the following formulation:

$$\begin{array}{rl} \textbf{Maximize} & margin \\ \textbf{Such that} & \forall i \ y_i w^T f_i \geq margin \\ & ||w||^2 \leq 1 \end{array}$$

We use $||w||^2 \leq 1$ instead of $||w||^2 = 1$ to make sure that our problem remain convex. Note that we do not include a bias term; we can incorporate a bias by simply adding a 1 at the end of each input vector. This means we are regularizing the bias in this formulation of the SVM, though the original SVM papers did not.

This problem is generally solved in an equivalent form that is easier to optimize, where we hold the margin fixed and minimize the magnitude of the weights:

$$\begin{array}{rl} \textbf{Minimize} & ||w||^2 \\ \textbf{Such that} & \forall i \ y_i w^T f_i \geq 1 \\ & margin > 0 \end{array}$$

This last formulation is an example of a quadratic program, which we are able to solve efficiently. Unfortunately, the hard margin SVM requires that the data be linearly separable, which is almost never the case. To address this issue, we introduce a slack variable, ξ. The problem now becomes:

$$\begin{array}{rl} \textbf{Minimize} & \lambda ||w||^2 + \sum_i \xi_i \\ \textbf{Such that} & y_i w^T f_i \geq 1 - \xi_i \\ & \xi_i \geq 0 \\ & margin > 0 \end{array}$$

There will be one slack variable ξ_i to correspond to each of the T data-points that we are considering. In Figure **??**, the correctly labeled points have $\xi_i = 0$, and the incorrectly labeled x-point has $\xi = 1 - y_i w^T f_i$ shown by the red arrow.

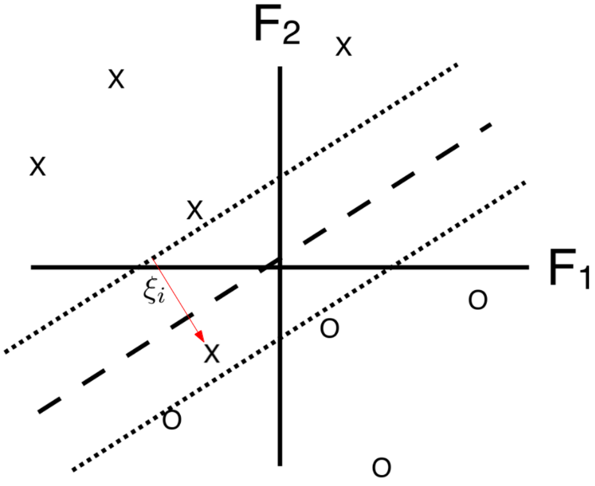


Figure 4: Linear binary classification with margins and slack variables.

Many SVM solvers further transform this optimization into a dual form, where the weight vector w is written as a linear combination of the support vectors, and this linear combination is optimized rather than optimizing w directly. This allows us to formulate nonlinear versions of the SVM, but this is beyond the scope of the class.

4.1 Gradient Descent

To perform gradient descent, we first observe that $\xi = \max(0, 1 - y_i w^T f_i)$, because the slack variable is 0 if this point is labeled correctly. If it is not labeled correctly (as described in previous sections), $\xi = 1 - y_i w^T f_i$. Any higher values of ξ are penalized. This allows us to generate the loss function

$$l_t = \lambda ||w||^2 + \max(0, 1 - y_t w^T f_t) \tag{3}$$

Our update for w is now as follows. If the output was correct by at least the margin, the only contribution to the loss is the magnitude of w:

$$w \leftarrow w - 2\alpha_t \lambda w \tag{4}$$

And if the output for this time step was incorrect or not outside the margin ($\xi > 0$),

$$w \leftarrow w - 2\alpha_t \lambda w + \alpha_t y_t f_t \tag{5}$$

We note that this loss function is **not** minimizing the number of mistakes made by the algorithm. In fact, solving this problem with a 0-1 loss function (which would minimize the number of mistakes) is known to be NP-hard. We can visualize the difference between these loss functions in Figure **??**, in which $Z = w^T f_i$:

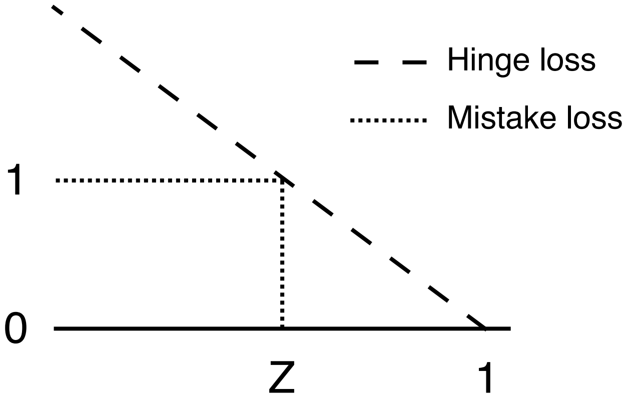


Figure 5: Loss functions

Our loss function, the hinge loss function, is convex, while the mistake loss (0-1 loss) function is not. However, we do see intuitively that the loss function we are using is the best "convexification" of the NP-hard problem. In practice the convexified loss is more appropriate anyway, since we generally want to minimize "how wrong" we are when we do mislabel a point.

4.2 Selecting α_t

- First idea is to set α_t proportional to $\frac{1}{\sqrt{t}}$.
- To analyze our regret: We have T elements, each with a maximum value of F (span of the convex set). The maximum gradient, G, is $\sqrt{TF}$. This results in a regret that is $R \leq \sqrt{FGT}$. Alternatively, let's assume $||w_t|| < C\gamma$, where C is some constant, and our features are normalized to $[-1.1]$ (so the max range of feature values is 2). Then, we note that $||\nabla l_t||^2 \leq 1/N\gamma + C4N$, where N is the number of data points. In general, $G \geq ||\nabla||^2$, therefore $G = 4N$.
- This is not as good as we could do.
- Notice that l_t is an extremely good convex function. It is a quadratic plus a convex function. In the same way all convex functions lie above a line (a subgradient) from every point, l_t lies above a quadratic from every point.
- Specifically, if it is always the case that

$$f(y) \geq f(x) + \frac{H}{2}(y-x)^2 + \nabla f_x^T (y-x) \tag{6}$$

  then $f(x)$ is said to be H-strongly convex.
- In this case l_t is λ-strongly convex.
- If $\alpha_t = \frac{G}{Ht}$, then regret$\leq \frac{G^2}{H}(1+\log t)$. $\log t$ is *really* good, and this learning rate and algorithm is essentially the current best for this class of problem.

5 Multiple Classes

What if we had 3 classes, say Apples, Oranges and Grapes. Given an item we would like to classify it into one of these classes. We maintain a weight vector w for each class, i.e. we have $w^{apple}, w^{orange}, w^{grapes}$.

Suppose instance i is an Apple, then we can generate two constraints to satisfy per datapoint:

$$\begin{aligned}
(w^{apple})^T f^i &\geq (w^{orange})^T f^i + 1 - \xi_i^1 \\
(w^{apple})^T f^i &\geq (w^{grapes})^T f^i + 1 - \xi_i^2 \\
&\vdots
\end{aligned}$$

6 Cost Weighted Learning

If you care more about satisfying some constraints, you can consider:

- Replicate the constraints so that they are accounted for multiple times (but adds computation time)
- Scale the margins
- Scale the slack variables ξ_i (this works best in practice)