Statistical Techniques in Robotics (16-831, F11) Lecture #14 (Monday October 31th)

Support Vector Machines and Bayes Regression

Lecturer: Drew Bagnell *Scribe: Carl Doersch* [1]

1 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 following 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.

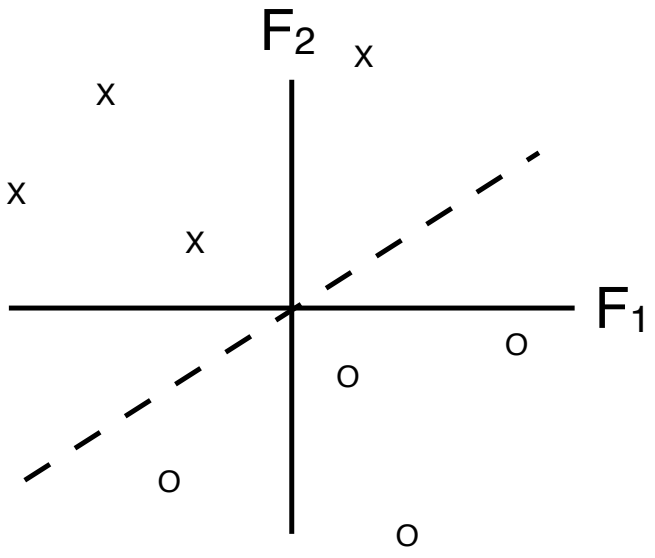

Figure 1: Linear binary classification

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

[1] Some content adapted from previous scribes: Alan Kraut, Robert Fisher, Heather Knight, Ben Xinjilefu, Kevin Lipkin, Alvaro Collet-Romea, Gandalf, Laura Lindzey and Hans Pirnay

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 following picture, the space between the dotted line and the decision boundary represents the margin.

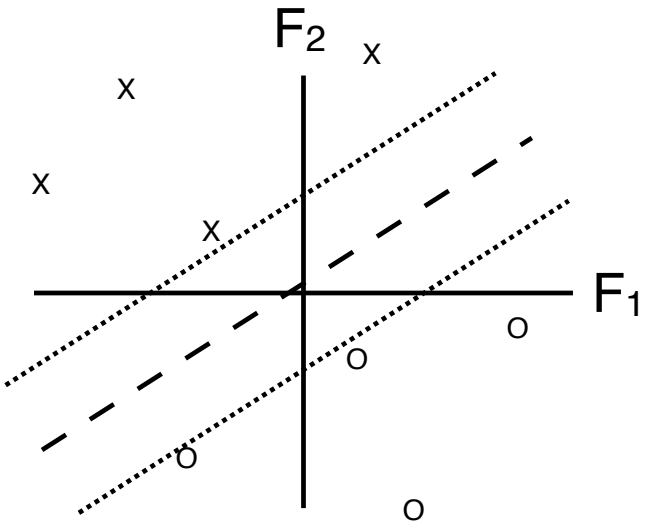

Figure 2: Linear binary classification with margins

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:

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

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:

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

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:

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

There will be one slack variable ξ_i to correspond to each of the T data-points that we are considering. 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.

1.1 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. 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 with the following figure, in which $Z = w^T f_i$:

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.

1.2 Selecting α_t

- First idea is to set α_t proportional to $\frac{1}{\sqrt{t}}$.
- If we have T elements, each with a maximum value of F, the maximum gradient, G, is $\sqrt{TF}$. This results in a regret that is $R \leq \sqrt{FGT}$.
- This is not as good as we could do.

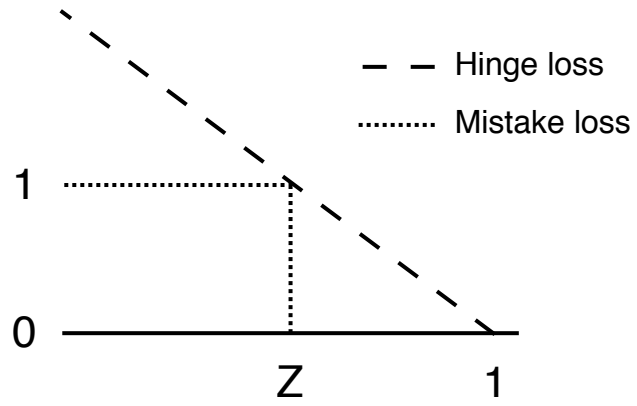

Figure 3: Loss functions

- 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.

2 Bayes' Linear Regression

In linear regression, the goal is to predict a continuous outcome variable. In particular, let:

- θ = parameter vector of the learned model
- $x_t \in \mathbb{R}^n$ = set of features at every timestep, used for prediction
- $y_t \in \mathbb{R}$ = true outcome

Then our model is as follows:

$$y_t = \theta x_t + \epsilon_t$$

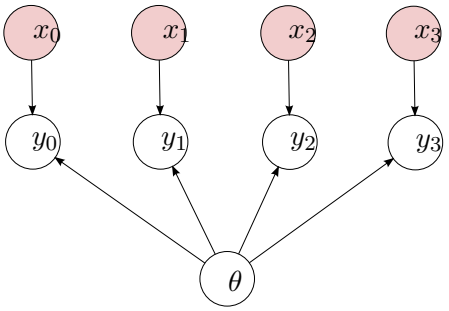

Figure 4: Graphic model of BLR

where $\epsilon_t \sim \mathcal{N}(0, \sigma^2)$ is noise that is independent of everything else. Thus the likelihood if θ is known may be written:

$$p(y_t|x_t, \theta) = \frac{1}{z} \exp(\theta x (2\sigma^2)^{-1})$$

In BLR, we maintain a distribution over the weight vector θ to represent our beliefs about what θ is likely to be. The math is easiest if we restrict this distribution to be a Gaussian: $\theta \sim N(\mu_0, \Sigma_0)$. This has the following form:

$$p(\theta) = \frac{1}{z} \exp\{-\frac{1}{2}(\theta - \mu_0)^T \Sigma_0^{-1} (\theta - \mu_0)\} \tag{7}$$

This is called the moment parameterization of a Guassian. This is the most intuitive parameterization, but it is actually more computationally convenient to use the natural parameterization. In the natural parameterization, the multiplication operation required in Bayes rule becomes a simple sum:

$$p(\theta) = \frac{1}{z} \exp\{J_0^T \theta - \frac{1}{2}\theta^T P_0 \theta\} \tag{8}$$

where

$$P_0 = \Sigma_0^{-1} \tag{9}$$
$$J_0 = P_0 \mu_0 \tag{10}$$

Note that for these to be probability distributions, Σ_0 must be positive-definite; otherwise the distribution will have an infinite integral and cannot be normalized. It is also generally constrained to be symmetric, as correlations between random variables are symmetric.

Recall that if we have a distribution over θ, and we recieve a new $\tilde{x}_{t+1}$, we can compute the distribution over $\tilde{y}_{t+1}$ by integrating out θ

$$p(\tilde{y}_{t+1}|x_{t+1}, D) = \int p(\tilde{y}_{t+1}|x_{t+1}, D, \theta) \cdot p(\theta|D) d\theta \tag{11}$$
$$= \int p(\tilde{y}_{t+1}|x_{t+1}, \theta) \cdot p(\theta|D) d\theta \tag{12}$$

We want to derive how to update our estimate of θ given a series of data $D = \{x, y\}$ up to timestep t and x_{t+1}. We can calculate $p(\theta|D)$ recursively using the Bayes' Theorem:

$$p(\theta|D) = \frac{1}{z} p(y_t|\theta, D) p(\theta|D) \tag{13}$$

where we know the likelihood $p(y_t|D, \theta) = p(y_t|x_t, \theta)$ is given by a Gaussian $\mathcal{N}(\theta^T x_t, \sigma_t^2)$

$$p(y_t|x_t, \theta) = \frac{1}{z} \exp\{\frac{-(\theta^T x_t - y_t)^2}{2\sigma^2}\} \tag{14}$$

The updating rule for $p(\theta|D)$ at timestep t is given by multiplication of two exponential functions. Adding the exponent of the prior to that of the likelihood yields

$$-\frac{1}{2\sigma^2} \left(\theta^T x_t - y_t\right)^2 + J_{t-1}^T \theta - \frac{1}{2} \theta^T P_{t-1} \theta \tag{15}$$

collecting terms to find updates J_t and P_t:

$$= -\frac{1}{2\sigma^2} \left(\theta^T x_t x_t^T \theta - 2\theta^T x_t y_t + y_t^2\right) + J_{t-1}^T \theta - \frac{1}{2} \theta^T P_{t-1} \theta \tag{16}$$

$$= \left(\frac{x_t^T y_t}{\sigma^2} + J_{t-1}^T\right) \theta - \frac{1}{2} \theta^T \left(\frac{x_t x_t^T}{\sigma^2} + P_{t-1}\right) \theta - \frac{y_t^2}{2\sigma^2} \tag{17}$$

Since this all happens in the exponent of an exponential function, the constant y_t^2/σ^2-term can be shifted into the regularizing constant z. Thus, the update rules for J_t and P_t are

$$J_t = \frac{y_t x_t}{\sigma^2} + J_{t-1} \tag{18}$$

$$P_t = \frac{x_t x_t^T}{\sigma^2} + P_{t-1} \tag{19}$$

1. In a gaussian model, a new datapoint always lowers the variance - this downgrading of the variance does not always make sense
2. If you believe there are outliers, this model won't work for you
3. The variance is not a function of y_t. The precision is only affected by input not output. This is a consequence of having the same σ (observation error) everywhere in space.