Support Vector Machines

Alan Yuille

Feb 5 2024

Support Vector Machines: Overview

- SVM's were originally formulated for binary classification. They could be interpreted geometrically in terms of finding the separating hyperplane with the maximum margin and with slack variables.
- They are formulated in terms of minimizing an energy function of the weights which was quadratic. This ensured convexity, a primal dual formulation, enabled the kernel trick, and showed that the solution was a function only of a limited number of support vectors.
- This could be solved by exploiting primal-duality or simply by steepest descent of the energy function. The motivation is that you should concentrate your resources on the decision boundary.

Support Vector Machines: Overview

- SVM's could be re-derived and generalized by first formulating an empirical loss function and a quadratic regularizer for the weights, Then obtaining a quadratic upper bound.
- This could be applied to energy functions defined over graphs (similar to exponential probability models but without the normalization terms). This formulation has the same advantages as the original SVMs (primal-dual, support vectors, etc.). When defined over graphs it requires inference algorithms, like dynamic programming or belief propagation.

Support Vector Machines: Overview

- SVMs can be extended to have internal hidden/latent variables. This can also be formulated in terms of an empirical loss function and then replaced by an upper bound. This difference is that the upper bound is the sum of a convex and a concave term. CCCP can be applied to get a learning algorithm which has a natural interpretation in terms of alternatively estimating the hidden variables and then solving for the weights by minimizing the SVM energy function, hence we retain support vectors and prime-dual formulations.
- Dynamic programming, belief propagation, or alternatives will be needed to perform inference over the graphs.
- For some problems (Girschick et al.) you can relate latent SVMs to Deep Network models. This is done by observing that the latent variables can be estimated by max-operations, and then requires generalizing the concept of max-pooling.
- It is possible to derive these SVM formulations as computationally tractable approximations to probability models (Yuille and He).

Basic Support Vector Machines

- Support Vector Machine (SVM) is a modern approach to linear separation. Suppose you have
  Data: $\{(\vec{x}_\mu, y_\mu) : \mu = 1 \text{ to } N\}$, $\quad y_\mu \in \{-1, 1\}$
  Hyperplane: $< \vec{x} : \vec{x} \cdot \vec{a} + b = 0 >$ $\quad |\vec{a}| = 1$
- The signed distance of a point $\vec{x}$ to the plane is $\vec{a} \cdot \vec{x} + b$.
- Proof. If we project the line $\vec{x}(\vec{\lambda}) = \vec{x} + \lambda\vec{a}$, it hits plane when $\vec{a} \cdot (\vec{x} + \lambda\vec{a}) = -b$. Follows that $\lambda = -(\vec{a} \cdot \vec{x} + b)/|\vec{a}|^2$, and if $|\vec{a}| = 1$, then $\lambda = -(\vec{a} \cdot \vec{x} + b)$.

Basic Support Vector Machines

- In SVM we seek a classifier with biggest margin:

$$\max_{\vec{a}, b, |\vec{a}|=1} C \quad \text{s.t.} \quad y_\mu(\vec{x}_\mu \cdot \vec{a} + b) \geq C, \ \forall \mu \geq 1 \text{ to } N$$

  I.e, the positive examples are at least distance C above the plane, and negative examples are at least C below the plane.

- Having a large margin is good for generalization. This can be shown intuitively.

Basic Support Vector Machines (2)

- Perfect separation is not always possible. Let's allow for some data points to be misclassified. We define the *slack variables* $\{z_1, ..., z_n\}$ allow data points to move in direction $\vec{a}$, so that they are on the right side of the margin.
- Criterion:
  $$\max_{a,b,|\vec{a}|=1} C \quad \text{s.t.} \quad y_\mu(\vec{x}_\mu \cdot \vec{a} + b) \geq C(1 - z_\mu), \forall \mu \in \{1, N\} \quad \text{s.t.} \quad z_\mu \geq 0, \forall \mu.$$
- Alternately, $y_\mu\{(\vec{x}_\mu + Cz_\mu\vec{a}) \cdot \vec{a} + b\} \geq C$, which is like moving $\vec{x}_\mu$ to $\vec{x}_\mu + z_\mu\vec{a}$.
- We pay a penalty $\sum_{\mu=1}^{N} z_\mu$ for the slack variables. If $z_\mu = 0$ then the data point is correctly classified and is on the correct side of the margin. If $z_\mu > 0$, then the data point is on the wrong side of the margin and the slack variable is needed to move it to the correct side.

The Max-Margin Criterion

- Here the task is to estimate several quantities simultaneously: (1) The plane $\vec{a}, b$. (2) The margin C. (3) The slack variables $\{z_\mu\}$.
- We need a criterion that maximizes the margin and minimizes the amount of slack variables used. We absorb C into $\vec{a}$ by $\vec{a} \to a/c$ and remove the constraint $|a| = 1$. Hence, $C = \frac{1}{|\vec{a}|}$.
- The Max-Margin Criterion:
  $\min \frac{1}{2} \sum \vec{a} \cdot \vec{a} + \gamma \sum_\mu z_\mu \text{ s.t. } y_\mu(\vec{x}_\mu \cdot \vec{a} + b) \geq 1 - z_\mu, \qquad \forall\ \mu,\ z_\mu \geq 0.$
- We need to solve the *Quadratic Primal Problem* using Lagrange multipliers: $L_p(\vec{a}, b, z; \alpha, \tau) =$
  $\frac{1}{2}\vec{a} \cdot \vec{a} + \gamma \sum_\mu z_\mu - \sum_\mu \alpha_\mu \{y_\mu(\vec{x}_\mu \cdot \vec{a} + b) - (1 - z_\mu)\} - \sum_\mu \tau_\mu z_\mu$. The $\{\alpha_\mu\}$ and $\{\tau_\mu\}$ are Lagrange parameters needed to enforce the inequality constraints. We require that $\alpha_\mu \geq 0, \tau_\mu \geq 0, \forall \mu$.

The Max-Margin Criterion

- The function $L_p(\vec{a}, b, z; \alpha, \tau)$ should be *minimized* with respect to the *primal variables* $\vec{a}, z$ and *maximized* with respect to the dual variables α, τ.
- This means that if the constraints are satisfied then we need to set the corresponding lagrange parameter to be zero (to maximize).
- For example, if $y_\mu(\vec{x}_\mu \cdot \vec{a} + b) - (1 - z_\mu) > 0$ for some μ then we set $\alpha_\mu = 0$ because the term $-\alpha_\mu\{y_\mu(\vec{x}_\mu \cdot \vec{a} + b) - (1 - z_\mu)\}$ is non-positive, and so the maximum value occurs when $\alpha_\mu = 0$. But if the constraint is not satisfied – e.g., $y_\mu(\vec{x}_\mu \cdot \vec{a} + b) - (1 - z_\mu) \leq 0$ – then the lagrange parameter will be positive.
- The lagrange parameters will only be positive (non-zero) if the data lies on the margin after the use of slack variables.

The Support Vectors

- L_p is a function of the primal variable $\vec{a}, b, \{z_\mu\}$ and the Lagrange parameters $\{\alpha_\mu, \tau_\mu\}$. There is no analytic solution for these variables, but we can use analytic techniques to get some understanding of their properties.

$$\frac{\partial L_p}{\partial \vec{a}} = 0 \Rightarrow \hat{\vec{a}} = \sum_\mu \alpha_\mu y_\mu \vec{x}_\mu$$

$$\frac{\partial L_p}{\partial b} = 0 \Rightarrow \sum_\mu \alpha_\mu y_\mu = 0$$

$$\frac{\partial L_p}{\partial z_\mu} = 0 \Rightarrow \alpha_\mu = \gamma - \hat{\tau}_\mu, \forall \mu$$

The Support Vectors

- The classifier is: $sign < \hat{\vec{a}} \cdot \vec{x} + \hat{b} >= sign < \sum_\mu \alpha_\mu y_\mu \vec{x}_\mu \cdot \vec{x} + b >$, by using the equation $\frac{\partial L_p}{\partial \vec{a}} = 0$.
- Given that the solution depends only on the vectors $\vec{x}_\mu$ for which $\alpha_\mu \neq 0$, we call them *support vectors*.
- The constraints are $y_\mu(\vec{x}_\mu \cdot \tilde{\vec{a}} + \tilde{b}) \geq 1 - \hat{z_\mu}$, $\hat{z_\mu} \geq 0$, and $\hat{\tau_\mu} \geq 0$.
- By theory of Quadratic Programming, $\alpha_\mu > 0$, only if either:
  (i) $z_\mu > 0$, i.e, slack variable is used.
  (ii) z_μ, but $y_\mu(\vec{x}_\mu \cdot \tilde{\vec{a}} + \tilde{b}) = 1$, i.e. data point is on the margin.
- The classifier depends only on the support vectors, the other data points do not matter. This is intuitively reasonable - the classifier must pay close attention to the data that is difficult to classify - the data near the boundary. By contrast the probabilistic approach models all the data which may be a waste of resources.

The Dual and its Relation to the Primal

- We can solve the problem more easily in the dual formulation – this is a function of Lagrange multipliers only.
  $L_p = \sum_\mu \alpha_\mu - \frac{1}{2}\sum_{\mu,\nu} \alpha_\mu \alpha_\nu y_\mu y_\nu \vec{x}_\mu \vec{x}_\nu$ s.t $0 \leq \alpha_\mu \leq \tau, \sum_\mu \alpha_\mu y_\mu = 0$.
- There are standard packages to solve this. (Although they get slow if you have a very large amount of data). Knowing $\{\hat{\alpha}_\mu\}$, will give us the solution $\hat{\vec{\alpha}} = \sum_\mu \hat{\alpha}_\mu y_\mu \vec{x}_\mu$, (only a little more work needed to get $\hat{b}$).
- Now we show how to obtain the dual formulation from the primal. The method we use is only correct if the primal is a convex function (but it is a positive quadratic function with linear constraints, which is convex).
- Start with the dual formulation L_p. Rewrite it as
  $L_p =$
  $-\frac{1}{2}\vec{a}\cdot\vec{a} + \sum_\mu \alpha_\mu + \vec{a}\cdot(\vec{a} - \sum_\mu \alpha_\mu y_\mu \vec{x}_\mu) + \sum_\mu z_\mu(\gamma - \tau_\mu - \alpha_\mu) - b\sum_\mu \alpha_\mu y_\mu$.
- Extremize w.r.t. $\vec{a}, b, \{z_\mu\}$. The result is:
  $\hat{\vec{a}} = \sum_\mu \alpha_\mu y_\mu \vec{x}_\mu, \sum_\mu \alpha_\mu y_\mu = 0, \gamma - \tau_\mu - \alpha_\mu = 0$
  Substituting back into L_p gives:
  $L_p = -\frac{1}{2}\sum_{\mu,\nu} \alpha_\mu \alpha_\nu y_\mu y_\nu \vec{x}_\mu \vec{x}_\nu + \sum_\mu \alpha_\mu$,
  which has to be maximized w.r.t. $\{\alpha_\mu\}$.

Reformulation of the Perceptron Algorithm

- The original Perceptron algorithm can be reformulated as follows. By SVM theory, the weight hypothesis will always be of form: $\vec{a} = \sum_\mu \alpha_\mu y_\mu \vec{x}_\mu$.
- The Perceptron update rule is: If data $\vec{x}_\mu$ is misclassified (i.e, $y_\mu(\vec{a} \cdot \vec{x}_\mu + b) \leq 0$), then set
  $\vec{\alpha}_\mu \to \vec{\alpha}_\mu + 1$
  $b \to b + y_\mu K^2$,
  where K is the radius of the smallest ball containing the data.
- So the Perceptron algorithm is an update on the dual variables $\vec{\alpha}$.

Max Margin from Empirical Risk

- We re-express the SVM primal function in a way that relates it to minimizing the empirical risk.
- The primal function L_p includes the constraint $y_\mu(\vec{x}_\mu \cdot \vec{a} + b) - 1 > 0$. If this constraint is satisfied, then it is best to set the slack variable $z_\mu = 0$ (because otherwise we pay a penalty γ for it). If the constraint is not satisfied, then we set the slack variable to be $z_\mu = 1 - y_\mu(\vec{x}_\mu \cdot \vec{a})$ because this is the smallest value of the slack variable which satisfies the constraint.
- We can summarize this by paying a *Hinge Loss* penalty $\max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\}$ – if the constraint is satisfied, then the maximum is 0 but, if not, the maximum is $1 - y_\mu(\vec{x}_\mu \cdot \vec{a})$ which is minimum value of the slack variable (to make the constraint satisfied).
- We re-express L_p as the (convex) loss function: $L(\vec{a}, b) = \frac{1}{2}\vec{a} \cdot \vec{a} + \gamma \sum_\mu \max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\}$.

Max Margin from Empirical Risk

- We re-express L_p as the sum of the empirical risk (with hinge loss function) plus a term $\frac{1}{2}|\vec{a}|^2$, multiplied by $\frac{1}{\gamma N}$:
  $\frac{L_p}{\gamma N} = \frac{1}{2\gamma N}|\vec{a}|^2 + \frac{1}{N}\sum_{\mu=1}^{N} \max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\}$.
- The first term is a *regularizer*. It penalizes decision rules $\hat{y}(\vec{x}) = \text{sign}(\vec{x} \cdot \vec{a} + b)$ which have large $|\vec{a}|$. This is done in order to help generalization.
- If we only minimize the loss function, we may overfit the data, because the space of possible decision rules is very big (all values of $\vec{a}$ and b). If we penalize those rules with big $|\vec{a}|$, then we restrict our set of rules and are more likely to generalize to new data.

Online Learning: AKA Steepest Descent

- We can do online learning (update with new data) using the cost function $L(\vec{a}, b) = \frac{1}{2}\vec{a} \cdot \vec{a} + \gamma \sum_\mu \max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\}$.
  Consider the second term for one datapoint $\vec{x}_\mu, y_\mu$:
  $\frac{\partial}{\partial \vec{a}} \max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\} = -y_\mu \vec{x}_\mu, \quad \text{if} \quad y_\mu(\vec{x}_\mu \cdot \vec{a} + b) < 1,$
  $= 0 \quad \text{otherwise.}$
- Online learning requires: (i) selecting the data $(\vec{x}_\mu, y_\mu)$ at random, (ii) computing $\frac{\partial}{\partial \vec{a}} \max\{0, 1 - y_\mu(\vec{x}_\mu \cdot \vec{a} + b)\}$, (iii) if $_\mu(\vec{x}_\mu \cdot \vec{a} + b) < 1$, updating $\vec{a}^t \to \vec{a}^t - \frac{1}{2N}\vec{a}^t - \gamma\{-y_\mu \vec{x}_\mu\}$,
  or if $_\mu(\vec{x}_\mu \cdot \vec{a} + b) > 1$, updating $\vec{a}^t \to \vec{a}^t - \frac{1}{2N}\vec{a}^t - \gamma\{0\}$.
- This is almost exactly the 1950's perceptron algorithm. The $-y_\mu$ term is like converting negative examples to positive ones. The $\frac{1}{2N}\vec{a}^t$ is from the regularizer.

Structure SVM

- Structure Max-Margin extends binary-classification methods so they can be applied to learn the parameters of an MRF, HMM, SCFG or other methods. Recall standard SVM, for binary classification, $R(\lambda) = \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M} \max\langle 0, 1 - y_i\lambda \cdot \phi(x_i)\rangle$ where $\{(y_i, x_i)\}$is training data, and $y_i \in \{\pm 1\}$,
- The goal is a Decision rule $\hat{y}_i(\lambda) = \arg\max_y y\lambda \cdot \phi(x_i) = sign(\lambda \cdot \phi(x_i))$
- The task is to minimize $R(\lambda)$ w.r.t λ which maximize the "margin" $\frac{1}{||\lambda||}$.

Structure SVM

- Here is a more general formulation that can be used if the output variable y is a vector $y = (y_1, \ldots, y_n)$. i.e. it could be the state of an MRF, or HMM, or a SCFG. $R(\lambda) = \frac{1}{2}||\lambda||^2 + c\sum_{i=1}^{M} \Delta(y_i, \hat{y}_i(\lambda))$. The decision rule: $\hat{y}_i(\lambda) = \arg\max_y \lambda \cdot \phi(x_i, y)$. The error function $\Delta(y_i, \hat{y}_i(\lambda))$ is any measure of distance between the true solution y_i and the estimate $\hat{y}_i(\lambda)$,
- Binary is a special case:(i) set $y_i \in \{-1, 1\}$, (ii) $\phi(x, y) = y\phi(x)$, (iii) $\Delta(y_i, \hat{y}_i(\lambda)) = \max\langle 0, 1 - y_i\lambda \cdot \phi(x_i)\rangle$. This is the *hinge loss* because the function is 0 if $y_i\lambda \cdot \phi(x_i) > 1$ i.e. point is on the right side of the margin and the function increases linearly with $\lambda \cdot \phi(x_i)$. (iv) $\hat{y}_i(\lambda) = \arg\max_y y\lambda \cdot \phi(x)$.

Structure SVM: Convex Upper Bound

- This more general formulation is $R(\lambda) = \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M}\Delta(y_i, \hat{y}_i(\lambda))$ with $\hat{y}_i(\lambda) = \arg\max y\lambda\phi(y, x_i)$
- But this is non-convex and hard to minimize. The non-convexity is due to the error term $\Delta(y_i, \hat{y}_i(\lambda))$ which a highly complicated function of λ. Instead we replace $R(\lambda)$ by a convex upper bound $\bar{R}(\lambda)$.
- $\bar{R}(\lambda) = \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M}\max_{\hat{y}}\{\Delta(y_i, \hat{y}) + \lambda\cdot\phi(x_i, \hat{y}) - \lambda\cdot\phi(x_i, y_i)\}$ which is convex in λ.
- We obtain this bound in two steps: Step 1: $\max_{\hat{y}}\{\Delta(y_i, \hat{y}) + \lambda\cdot\phi(x_i, \hat{y})\} \geq \Delta(y_i, \hat{y}_i(\lambda)) + \lambda\cdot\phi(x_i, \hat{y}_i(\lambda))$. Step 2: $\lambda\cdot\phi(x_i, \hat{y}_i(\lambda)) \geq \lambda\cdot\phi(x_i, y_i)$.
- Note: these bounds are "tight" because if we can find a good solution then $y_i \approx \hat{y}_i(\lambda)$.

Structure SVM: How to Minimize $R(\lambda)$.

- How to minimize $R(\lambda)$?
- (1) Solve in the Dual Space. Like the original SVM for the binary problem.
- (2) Stochastic gradient descent. Pick example (x_i, y_i), take derivative of $R(\lambda)$ w.r.t λ

$$\lambda^{t+1} = \lambda^t - \beta^t(\phi(x_i, \hat{y}^t) - \phi(x_i, y_i))$$

where $\hat{y}^t = \arg\max_{\hat{y}} \Delta(y_i, \hat{y}) + \lambda \cdot \phi(x_i, \hat{y}))$

- To compute $\arg\max \hat{y}$ will require an inference algorithm. This depends on whether this is a graph with closed loops or not. If no closed loops, we can use dynamical programming. If closed loops we would need an approximate algorithm like mean field theory or belief propagation.

Latent SVM

- How to extend to module with latent (hidden) variables? Denote these variables by h with decision rule $(\hat{y}, \hat{h}) = \arg\max_{(y,h)\in\mathcal{Y},\mathcal{H}} \lambda \cdot \phi(x, y, h)$
- Training data $\langle (x_i, y_i); i = 1, \ldots, M \rangle$. The hidden variables are not known.
- The Loss function $\Delta(y_i, \hat{y}_i(\lambda), \hat{h}_i(\lambda))$ depends on the truth y_i, the estimate of $\hat{y}_i(\lambda), \hat{h}_i(\lambda)$ from the model

$$R(\lambda) = \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M} \Delta(y_i; \hat{y}_i(\lambda), \hat{h}_i(\lambda))$$

which is typically a highly non-convex function of λ.

Latent SVM

- Replace $R(\lambda)$ by an upper bound

$$\bar{R}(\lambda) = \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M} \max_{(\hat{y},\hat{h})}(\Delta(y_i;\hat{y},\hat{h}) + \lambda\cdot\phi(x_i,\hat{y},\hat{h})) - \max_{h} \lambda\cdot\phi(x_i,y_i,h)$$

-

$$f(\lambda) = \max_{(\hat{y},\hat{h})}(\Delta(y_i;\hat{y},\hat{h}) + \lambda\cdot\phi(x_i,\hat{y},\hat{h}))$$

$$g(\lambda) = -\max_{h} \lambda\cdot\phi(x_i,y_i,h)$$

- This is an upper bound but it is not convex, From general principles, it is impossible to get a useful convex upper bound for any problem that involves learning with hidden variables.

Latent SVM: Concavity and Convexity

- Here $f(\cdot)$ is convex and $g(\cdot)$ is concave.
- To show convexity and concavity. Let

$$\tau(\lambda) = \sum_{i=1}^{M} \max_{\hat{y}_i} \lambda \cdot \phi(x_i, \hat{y}_i)$$

- This is convex if $\tau(\alpha\lambda_1 + (1-\alpha)\lambda_2) \leq \alpha\tau(\lambda_1) + (1-\alpha)\tau(\lambda_2)$ for all $\lambda_1, \lambda_2, \alpha$.

Latent SVM: Concavity and Convexity

- Now

$$\tau(\alpha\lambda_1 + (1-\alpha)\lambda_2) = \alpha \sum_{i=1}^{M} \max_{\hat{y}_i} \alpha\lambda_1 + (1-\alpha)\lambda_2), \phi(x_i, \hat{y}_i)$$

$$\alpha\tau(\lambda_1)+(1-\alpha)\tau(\lambda_2) = \alpha \sum_{i=1}^{M} \max_{y_i} \{\lambda_1, \phi(x_i, \hat{y}_i)\rangle + (1-\alpha) \sum_{i=1}^{M} \max_{y_i} \lambda_2 \phi(x_i, \hat{y}_i)\}$$

- and the result follows from

$$\max_{\hat{y}_i} \alpha\lambda_1\phi(x_i, \hat{y}_i) + \max_{\hat{y}_i} \{(1-\alpha)\lambda_2\phi(x_i, \hat{y}_i)\} \geq \max_{\hat{y}_i} \{(\alpha\lambda_1+(1-\alpha)\lambda_2)\phi(x_i, \hat{y}_i)\}$$

Latent SVM: CCCP

- There are two steps.
- Step 1: involves estimating the hidden state h_i^*

$$\frac{\partial g(\lambda^t)}{\partial \lambda} = -\phi(x_i, y_i, h^*)$$

where $h^* = \arg\max h\lambda^t \phi(x_i, y_i, h)$, λ^t is the current estimate of λ. This reduces to a modified SVM with known state:

$$\min_{\lambda} \frac{1}{2}||\lambda||^2 + C\sum_{i=1}^{M} \max_{(y,h)}\{\lambda \cdot \phi(x_i, y_i, h) + \Delta(y_i, y, h)\} - C\sum_{i=1}^{M} \lambda \cdot \phi(x_i, y_i, h_i^*)$$

- Step 2 estimate λ. This is equivalent to minimizing a structured SVM (now that h is known).
- Repeat steps 1 and 2 until convergence. This is the CCCP algorithns so it guaranteed to converge to a local minimum.