

SDS 385: Stat Models for Big Data

Lecture 6: Support Vector Machines

Purnamrita Sarkar
Department of Statistics and Data Science
The University of Texas at Austin
<https://psarkar.github.io/teaching>

Support Vector Machines

- Given training data $(x_i, y_i)_{i=1}^n \in \mathbb{R}^p \times \{-1, 1\}$, we want to minimize:

$$\min_w \frac{w^T w}{2} + C \sum_i \max(0, 1 - y_i w^T x_i)$$

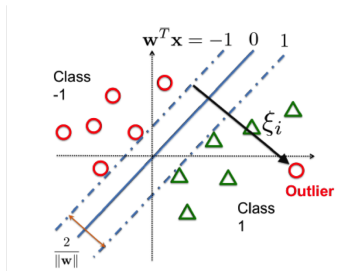


Figure 1: Courtesy Cho-Jui Hsieh's class

- Define:

$$f(w) = \frac{1}{n} \sum_i \left(\underbrace{\frac{w^T w}{2} + nC \max(0, 1 - y_i w^T x_i)}_{f_i(w)} \right)$$

- For $t = 1 \dots$
 - Pick j uniformly at random.
 - Compute $\nabla f_j(w)$
 - Update $w = w - \eta_t \nabla f_j(w)$

- In this case, the hinge loss is not differentiable.
- A subgradient of the hinge loss $\max(0, 1 - y_i w^T x_i)$

$$\begin{cases} -y_i x_i & \text{if } 1 - y_i w^T x_i > 0 \\ 0 & \text{if } 1 - y_i w^T x_i < 0 \\ 0 & \text{if } 1 - y_i w^T x_i = 0 \end{cases}$$

- For $t = 1 \dots$
 - Pick j uniformly at random.
 - If $y_j w^T x_j < 1$
 - $w_{t+1} = w_t(1 - \eta_t) + \eta_t C n y_j x_j$
 - Else update $w_{t+1} = w_t(1 - \eta_t)$
 - If you store w as a scalar vector pair $w = \gamma v$, then just updating γ leads to $O(1)$ computation.
- This is in “Pegasos: primal estimated subgradient solver for SVM”, ICML 2007, Shalev-Schwartz et al.

- Remember the primal problem?

$$\begin{aligned} \min_{w, \xi} \quad & \frac{w^T w}{2} + C \sum_i \xi_i \\ \text{s.t.} \quad & y_i w^T x_i - 1 + \xi_i \geq 0, \xi_i \geq 0, i = 1, \dots, n \end{aligned}$$

- Add lagrange multipliers:

$$\min_{w, \xi} \max_{\alpha \geq 0, \beta \geq 0} \frac{w^T w}{2} + C \sum_i \xi_i - \sum_i \alpha_i (y_i w^T x_i - 1 + \xi_i) - \sum_i \beta_i \xi_i$$

- Under Slater's condition, exchanging min and max does not change the optimal solution.

- The dual is:

$$\max_{\alpha \geq 0, \beta \geq 0} \min_{w, \xi} \frac{w^T w}{2} + C \sum_i \xi_i - \sum_i \alpha_i (y_i w^T x_i - 1 + \xi_i) - \sum_i \beta_i \xi_i$$

- Differentiate w.r.t w .

$$w^* = \sum_i \alpha_i^* y_i x_i$$

- Differentiate w.r.t ξ_j .

$$C = \alpha_i + \beta_i$$

- Substituting

$$\max_{0 \leq \alpha \leq C} -\frac{1}{2} \sum_{ij} \alpha_i \alpha_j y_i y_j x_i^T x_j + \sum_i \alpha_i$$

SVM: the dual problem

- The dual of SVM is given by:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \sum_i \alpha_i \\ \text{s.t.} \quad & \alpha_i \in [0, C] \end{aligned}$$

Where $Q_{ij} = y_i y_j x_i^T x_j$.

- The primal solution can be written in terms of the dual solution as:

$$w^* = \sum_i y_i \alpha_i^* x_i$$

Stochastic Dual coordinate descent

- Consider the one variable problem:

$$\begin{aligned}f(\alpha + \delta e_i) &= \frac{1}{2}(\alpha + \delta e_i)^T Q(\alpha + \delta e_i) - \sum_i \alpha_i - \delta \\&= \frac{1}{2}\alpha^T Q\alpha + \delta\alpha^T Qe_i + \delta^2 \frac{Q_{ii}}{2} - \sum_i \alpha_i - \delta\end{aligned}$$

- Set the gradient to zero:

$$(Q\alpha)_i + Q_{ii}\delta^* - 1 = 0 \rightarrow \delta^* = \frac{1 - (Q\alpha)_i}{Q_{ii}}$$

- But we have the constraint $0 \leq \alpha_i + \delta \leq C$, so we have:

$$\alpha_i + \delta^* = \begin{cases} \alpha_i + \frac{1 - (Q\alpha)_i}{Q_{ii}} & \text{If } \alpha_i + \delta \in [0, C] \\ 0 & \text{If } \alpha_i + \delta < 0 \\ C & \text{If } \alpha_i + \delta > C \end{cases}$$

Stochastic Dual coordinate descent

- For $t = 1 \dots$
 - Pick a coordinate i at random.
 - Compute

$$\delta^* = \arg \min_{0 < \alpha_i + \delta < C} f(\alpha + \delta e_i)$$

- Update $\alpha_i = \alpha_i + \delta^*$
- Update $w = w + \delta^* y_i x_i$ (time complexity $O(\text{nnz}(x_i))$)
- After convergence this gives $w^* = \sum_i \alpha_i^* y_i x_i$

Fast computation

- Main computational bottleneck $Q\alpha$
- Write $Q = \underbrace{\text{diag}(y)X}_R \underbrace{X^T \text{diag}(y)}_{R^T}$

- Note that:

$$(Q\alpha)_i = R_i \underbrace{R^T \alpha}_w = y_i x_i^T w$$

- If you maintain w through the steps, computational complexity becomes $O(\text{nnz}(x_i))$
- After each $\alpha_i \leftarrow \alpha_i + \delta^* e_i$ update, you have:

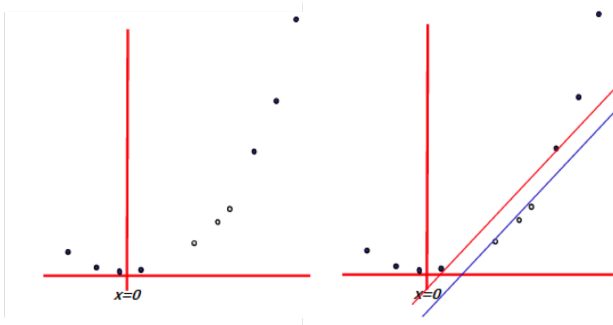
$$w \leftarrow w + \delta^* Qe_i = w + \delta^* Q(:, i)$$

SVM: the kernel trick



- How do we use an SVM here?
- What if we use more than one dimensions?
- Say $z = (x, x^2)$

SVM: the kernel trick



- So using a different mapping to a higher dimensional space helped.
- So if I want to map my features to a quadratic space, I will have coefficients: $(1, x_1, x_2, \dots, x_1^2, x_2^2, \dots, x_1 x_2, x_1 x_3, \dots)$
- So a total of $O(p^2)$ terms. If it is a cubic, then $O(p^3)$ terms. Wow! storage increases exponentially with the dimensionality.

SVM: the kernel trick

- So in a nutshell:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \sum_k \alpha_k \\ \text{s.t.} \quad & 0 \leq \alpha \leq C \end{aligned}$$

where

$$Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$$

SVM: the kernel trick

- So in a nutshell:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \sum_k \alpha_k \\ \text{s.t.} \quad & 0 \leq \alpha \leq C \end{aligned}$$

where

$$Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$$

- Building Q needs $n^2 m^2$ time, where m is the number of dimensions of the projection.

SVM: the kernel trick

- So in a nutshell:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \sum_k \alpha_k \\ \text{s.t.} \quad & 0 \leq \alpha \leq C \end{aligned}$$

where

$$Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$$

- Building Q needs $n^2 m^2$ time, where m is the number of dimensions of the projection.
- $w = \sum_{k: \alpha_k > 0} \alpha_k y_k \phi(x_k)$. This will take $O(nm)$ time.

SVM: the kernel trick

- So in a nutshell:

$$\begin{aligned} \min_{\alpha} \quad & \frac{1}{2} \alpha^T Q \alpha - \sum_k \alpha_k \\ \text{s.t.} \quad & 0 \leq \alpha \leq C \end{aligned}$$

where

$$Q_{ij} = y_i y_j \phi(x_i)^T \phi(x_j)$$

- Building Q needs $n^2 m^2$ time, where m is the number of dimensions of the projection.
- $w = \sum_{k: \alpha_k > 0} \alpha_k y_k \phi(x_k)$. This will take $O(nm)$ time.
- Classification rule for datapoint x :

$$\text{Predict sign}(w^T \phi(x))$$

See next slide.

SVM: the kernel trick

- But, how do you predict the class of a new example?
 - You would need to compute $w^T \phi(x)$, where $\phi(x)$ is the high dimensional mapping.
 - This is proportional to the length of $\phi(x)$
 - But remember the form of w ?
 - $w^T \phi(x) = \sum_i \alpha_i y_i \phi(x_i)^T \phi(x) = \sum_i \alpha_i y_i K(x_i, x)$

SVM: the kernel trick

$$\begin{pmatrix} 1 \\ \sqrt{2}a_1 \\ \sqrt{2}a_2 \\ \vdots \\ \sqrt{2}a_m \\ a_1^2 \\ a_2^2 \\ \vdots \\ a_m^2 \\ \sqrt{2}a_1a_2 \\ \sqrt{2}a_1a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \sqrt{2}a_2a_3 \\ \vdots \\ \sqrt{2}a_1a_m \\ \vdots \\ \sqrt{2}a_{m-1}a_m \end{pmatrix} \cdot \begin{pmatrix} 1 \\ \sqrt{2}b_1 \\ \sqrt{2}b_2 \\ \vdots \\ \sqrt{2}b_m \\ b_1^2 \\ b_2^2 \\ \vdots \\ b_m^2 \\ \sqrt{2}b_1b_2 \\ \sqrt{2}b_1b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \sqrt{2}b_2b_3 \\ \vdots \\ \sqrt{2}b_1b_m \\ \vdots \\ \sqrt{2}b_{m-1}b_m \end{pmatrix}$$

Diagram illustrating the dot product of two vectors. The first vector has components: 1, $\sqrt{2}a_1$, $\sqrt{2}a_2$, ..., $\sqrt{2}a_m$, a_1^2 , a_2^2 , ..., a_m^2 , $\sqrt{2}a_1a_2$, $\sqrt{2}a_1a_3$, ..., $\sqrt{2}a_1a_m$, $\sqrt{2}a_2a_3$, ..., $\sqrt{2}a_1a_m$, $\sqrt{2}a_{m-1}a_m$. The second vector has components: 1, $\sqrt{2}b_1$, $\sqrt{2}b_2$, ..., $\sqrt{2}b_m$, b_1^2 , b_2^2 , ..., b_m^2 , $\sqrt{2}b_1b_2$, $\sqrt{2}b_1b_3$, ..., $\sqrt{2}b_1b_m$, $\sqrt{2}b_2b_3$, ..., $\sqrt{2}b_1b_m$, $\sqrt{2}b_{m-1}b_m$. The dot product is shown as a sum of products of corresponding components, grouped by color brackets on the right:

- Red bracket: 1
- Green bracket: $+$
- Green bracket: $\sum_{i=1}^m 2a_i b_i$
- Green bracket: $+$
- Purple bracket: $\sum_{i=1}^m a_i^2 b_i^2$
- Purple bracket: $+$
- Purple bracket: $\sum_{i=1}^m \sum_{j=i+1}^m 2a_i a_j b_i b_j$

- But this dot product is the same as $(a^T b + 1)^2$!
- Same for cubic maps. So instead of doing $O(p^k)$ computation to compute a dot product in degree k polynomial, you can compute it in $O(p)$ time!
- So plug in $K(x, x')$ instead of $\phi(x)^T \phi(x')$

Acknowledgment

Cho-Jui Hsieh's class notes at UC Davis. Andrew Moore's notes on SVM.