SVMs (Continued): Sequential Minimal Optimization (SMO)

- Intuition: remember Perceptrons…
- Output for example **x** is sign($\mathbf{w}^T\mathbf{x}$)
- Candidate Hypotheses: real-valued weight vectors **w**
- Training: Update **w** for each misclassified example **x** (target class *y*, predicted *o*) by:
  - $w_i \leftarrow w_i + \eta(y\text{-}o)x_i$
  - η is learning rate parameter
- Let E be the error *o*- *y*. The above can be rewritten as

$$\mathbf{w} \leftarrow \mathbf{w} - \eta E\mathbf{x}$$

- So final weight vector is a sum of weighted contributions from each example. Predictive model can be rewritten as weighted combination of examples rather than features, where the weight on an example is sum (over iterations) of terms $-\eta E$.

Corresponding Use in Prediction

- To use perceptron, prediction is $\mathbf{w}^T\mathbf{x}$ - b. May treat -b as one more coefficient in **w** (and omit it here), may take sign of this value
- In alternative view of last slide, prediction can be written as $\sum_i a_j'(\mathbf{x}_j^T \mathbf{x}) - b$ or, revising the weight appropriately: $\sum_i y_j a_j(\mathbf{x}_i^T \mathbf{x}) - b$
- Prediction in SVM is:

$$u = \sum_{j=1}^{N} y_j \alpha_j K(\vec{x}_j, \vec{x}) - b$$

From Perceptron Rule to SMO Rule

- Recall that SVM optimization problem has the added requirement that: $\sum_{i=1}^{N} y_i \alpha_i = 0$
- Therefore if we increase one α by an amount η, in either direction, then we have to change another α by an equal amount in the opposite direction (relative to class value). We can accomplish this by changing: $\mathbf{w} \leftarrow \mathbf{w} - \eta E \mathbf{x}$ also viewed as:

$$\alpha \leftarrow \alpha - y \eta E$$ to:

$$\alpha_1 \leftarrow \alpha_1 - y_1 \eta (E_1 - E_2)$$

or equivalently: $\alpha_1 \leftarrow \alpha_1 + y_1 \eta (E_2 - E_1)$

We then also adjust α_2 so that $y_1\alpha_1 + y_2\alpha_2$ stays same

First of Two Other Changes from Perceptron Rule

- Instead of learning rate of $\frac{1}{20}$ or some similar constant, we use $\frac{1}{x_1 \cdot x_1 + x_2 \cdot x_2 - 2x_1 \cdot x_2}$

  or more generally $\frac{1}{K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2)}$
- For a given difference in error between two examples, the step size is larger when the two examples are more similar. When x_1 and x_2 are similar, $K(x_1,x_2)$ is larger (for typical kernels, including dot product). Hence the denominator is smaller and step size is larger.

Second of Two Other Changes

- Recall our formulation had an additional constraint:

$$0 \leq \alpha_i \leq C, \forall i$$

- Therefore, we "clip" any change we make to any α to respect this constraint

- This yields the following SMO algorithm. (We have motivated it intuitively. Can prove it minimizes our objective.)

Updating Two α's: One SMO Step

- Given examples e1 and e2, set

$$\alpha_2^{\text{new}} = \alpha_2 + \frac{y_2(E_1 - E_2)}{\eta}$$

where $\eta = K(x_1, x_1) + K(x_2, x_2) - 2K(x_1, x_2)$

- Clip this value in the natural way:

$$\alpha_2^{\text{new,clipped}} = \begin{cases} H & \text{if} \quad \alpha_2^{\text{new}} \geq H; \\ \alpha_2^{\text{new}} & \text{if} \quad L < \alpha_2^{\text{new}} < H; \\ \text{L} & \text{if} \quad \alpha_2^{\text{new}} \leq L. \end{cases}$$

if y1 = y2 then:

$$L = \max(0, \alpha_2 + \alpha_1 - C)$$
$$H = \min(C, \alpha_2 + \alpha_1)$$

otherwise:

$$L = \max(0, \alpha_2 - \alpha_1)$$
$$H = \min(C, C + \alpha_2 - \alpha_1)$$

- Set $\alpha_1^{\text{new}} = \alpha_1 + s(\alpha_2 - \alpha_2^{\text{new,clipped}})$ where $s = y_1 y_2$

Karush-Kuhn-Tucker (KKT) Conditions

- It is necessary and sufficient for a solution to our objective that all α's satisfy the following:

$$\alpha_i = 0 \Leftrightarrow y_i u_i \geq 1,$$
$$0 < \alpha_i < C \Leftrightarrow y_i u_i = 1,$$
$$\alpha_i = C \Leftrightarrow y_i u_i \leq 1.$$

- An α is 0 iff that example is correctly labeled with room to spare
- An α is C iff that example is incorrectly labeled or in the margin
- An α is properly between 0 and C (is "unbound") iff that example is "barely" correctly labeled (is a support vector)

- We just check KKT to within some small epsilon, typically 10^{-3}

SMO Algorithm

- *Input: C, kernel, kernel parameters, ε*
- *Initialize b and all α's to 0*
- *Repeat until KKT satisfied (to within ε):*
  - *Find an example **e1** that violates KKT (prefer unbound examples here, choose randomly among those)*
  - *Choose a second example **e2**. Prefer one to maximize step size (in practice, faster to just maximize $|E_1 - E_2|$). If that fails to result in change, randomly choose unbound example. If that fails, randomly choose example. If that fails, re-choose **e1**.*
  - *Update α_1 and α_2 in one step, compute new threshold b*

Comments on SVMs

- we can find solutions that are globally optimal (maximize the margin)
  - because the learning task is framed as a convex optimization problem
  - no local minima, in contrast to multi-layer neural nets

- there are two formulations of the optimization: *primal* and *dual*
  - dual represents classifier decision in terms of support vectors
  - dual enables the use of kernel functions

- we can use a wide range of optimization methods to learn SVM
  - standard quadratic programming solvers
  - SMO [Platt, 1999]
  - linear programming solvers for some formulations
  - etc.

Comments on SVMs

- kernels provide a powerful way to
  - allow nonlinear decision boundaries
  - represent/compare complex objects such as strings and trees
  - incorporate domain knowledge into the learning task
- using the kernel trick, we can implicitly use high-dimensional mappings without explicitly computing them
- one SVM can represent only a binary classification task; multi-class problems handled using multiple SVMs and some encoding
  - one class vs. rest
  - ECOC
  - etc.
- empirically, SVMs have shown state-of-the art accuracy for many tasks
- the kernel idea can be extended to other tasks (anomaly detection, regression, etc.)