Perceptron Learning with Random Coordinate Descent

Ling Li and Hsuan-Tien Lin

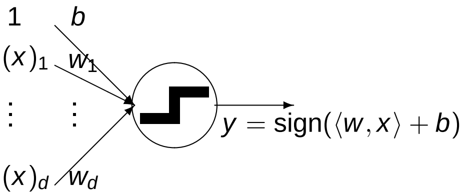
Learning Systems Group, California Institute of Technology, U.S.A.

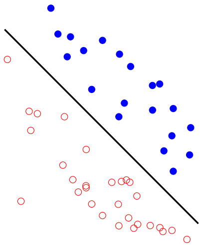
International Joint Conf. on Neural Networks, August 15, 2007

Perceptron

- proposed by Rosenblatt (1958)
- a single neuron;
  a linear threshold classifier;
  a hyperplane in $\mathbb{R}^d$
- define $(\mathbf{x})_0 \triangleq 1$ and $\mathbf{w}_0 \triangleq b$:

$$y = \mathrm{sign}(\langle \mathbf{w}, \mathbf{x} \rangle)$$



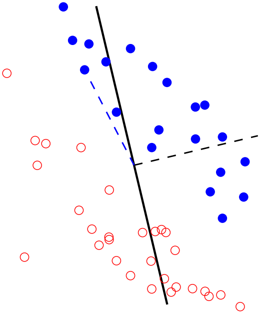


a simple but useful classifier, especially for building more complex systems

Perceptron Learning Rule (PLR)

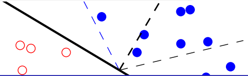
- an iterative optimization procedure to learn $\mathbf{w}$ from $\mathcal{S} = \{(\mathbf{x}_n, y_n)\}_{n=1}^{N}$ (Rosenblatt, 1962)
- repeatedly, for $(\mathbf{x}_n, y_n) \in \mathcal{S}$,

  1. if current $\mathbf{w}$ correctly classifies $\mathbf{x}_n$,
     do nothing;
  2. if current $\mathbf{w}$ wrongly classifies $\mathbf{x}_n$,
     $\mathbf{w}^{new} = \mathbf{w} + y_n \mathbf{x}_n$



- convergence proved for **separable** $\mathcal{S}$

but unstable for nonseparable cases



Minimum Training Error Perceptrons

$$\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w}} \sum_{n=1}^{N} [\![y_n \langle \mathbf{w}, \mathbf{x}_n \rangle \leq 0]\!]$$

Hard Optimization Problem

- numerically:
  0/1 loss $c(\rho) = [\![\rho \leq 0]\!]$ not convex, not continuous, with mostly 0 gradient
- combinatorially:
  NP-complete
  (Marcotte and Savard, 1992)

Useful Classifier

- theoretically:
  $\mathbf{w}^*$ converges to optimal linear classifier when $N \to \infty$
- practically:
  basic building blocks for networks/ensembles of neurons

goal: an efficient algorithm guaranteed to approach $\mathbf{w}^*$
even for nonseparable cases

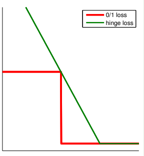
Two Existing Approaches for Nonseparable Sets

$$\mathbf{w}^* \in \operatorname*{argmin}_{\mathbf{w}} C(\mathbf{w}) = \sum_{n=1}^{N} c(y_n \cdot \langle \mathbf{w}, \mathbf{x}_n \rangle), \text{ where } c(\rho) = [\![\rho \le 0]\!]$$

pocket-PLR

- in addition to PLR, store the best $\mathbf{w}$ encountered
- guaranteed to locate $\mathbf{w}^*$ with high probability in the long run
- usually inefficient
  – PLR unstable and wastes iterations on bad candidates

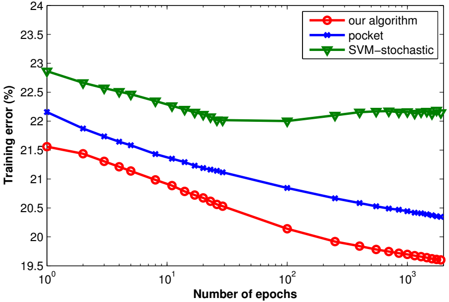
support vector machine (SVM)



- regularize $C(\mathbf{w})$; change $c(\rho)$ to hinge loss
- efficiently solved via quadratic programming
- no guarantee on getting $\mathbf{w}^*$
  – hinge loss different from $0/1$ loss

Our Contributions

- new perceptron algorithm to minimize $0/1$ loss
  – efficient with guarantee on approaching $\mathbf{w}^*$



- empirical study to understand $0/1$ loss
  – insights on dealing with nonseparable data sets
- better neural ensemble approach: AdaBoost + our algorithm
  – useful when modeling very complex data sets

Our Algorithm: Random Coordinate Descent

PLR

$$\mathbf{w}^{new} = \mathbf{w} + [\![y_n \langle \mathbf{w}, \mathbf{x}_n \rangle \leq 0]\!] \, (y_n \mathbf{x}_n)$$

$$\Downarrow \text{ generalized and improved}$$

Random Coordinate Descent (RCD)

$$\mathbf{w}^{new} = \mathbf{w} + \alpha \mathbf{d}$$

- instead of fixed directions $y_n \mathbf{x}_n$, use random directions $\mathbf{d}$
- instead of a fixed step size 0 or 1, use the optimal step size α with respect to $\mathbf{d}$

next: how to compute the optimal step size

Computing the Optimal Step Size α

$$\min_{\alpha \in \mathbb{R}} \sum_{n=1}^{N} [\![y_n \langle \mathbf{w} + \alpha \mathbf{d}, \mathbf{x}_n \rangle \le 0]\!]$$

Define

$$\delta_n \triangleq \langle \mathbf{d}, \mathbf{x}_n \rangle$$

when $\delta_n = 0$

$$\langle \mathbf{w}, \mathbf{x}_n \rangle$$

when $\delta_n \neq 0$

$$\delta_n \left(\delta_n^{-1} \langle \mathbf{w}, \mathbf{x}_n \rangle + \alpha \right)$$

- for those n with nonzero δ_n, let $(x'_n, y'_n) \leftarrow \left(\delta_n^{-1} \langle \mathbf{w}, \mathbf{x}_n \rangle, y_n \operatorname{sign}(\delta_n) \right)$

$$\min_{\alpha \in \mathbb{R}} \sum_{\delta_n \neq 0} [\![y'_n \left(x'_n + \alpha \right) \le 0]\!]$$

- optimal α can be computed from these new 1-D examples efficiently by sorting + dynamic programming

Choosing Update Directions $\mathbf{d}$

some natural candidates

1. coordinate directions $\mathbf{e}_i = (\ldots, 0, 1, 0, \ldots)^T$
2. PLR directions $y_n \mathbf{x}_n$
3. sufficiently random directions on the unit sphere $\|\mathbf{d}\| = 1$

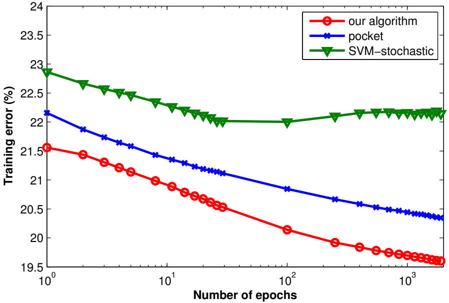
- recall: hard optimization problem
  – finite choices like coordinate or PLR stuck in local minima
- sufficiently random directions guarantee convergence to global minima $\mathbf{w}^*$ in the long run
- some even provably help with efficient local search

Putting Things Together

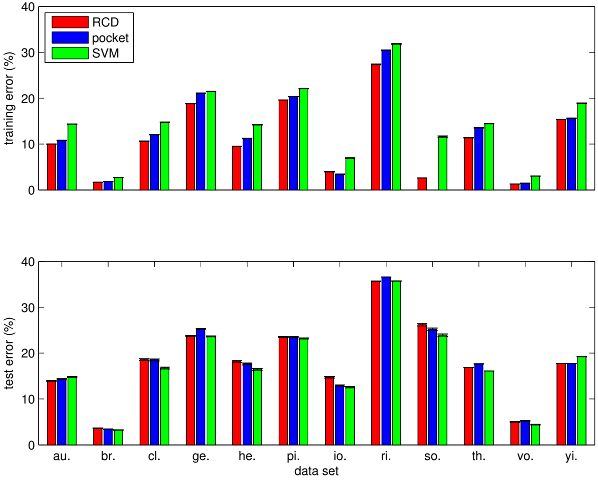
Random Coordinate Descent

iteratively,

1. pick a direction $\mathbf{d}$ from sufficiently random choices
2. transform $(\mathbf{x}_n, y_n)$ to (x'_n, y'_n) with $\mathbf{w}$ and $\mathbf{d}$
3. compute optimal step size α from (x'_n, y'_n)
4. $\mathbf{w}^{new} = \mathbf{w} + \alpha\mathbf{d}$



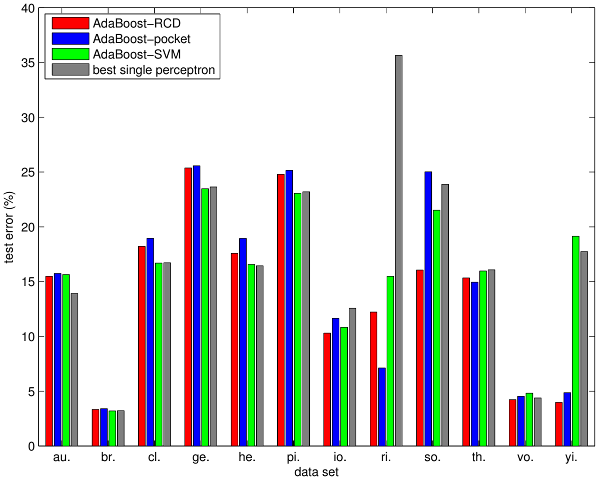
Comparison as Single Perceptron Algorithms



- training error (0/1 loss): RCD usually lowest; SVM highest
- test error: SVM often better
- pocket slow and not the sharpest in both cases

for a single perceptron, RCD does too good of a job for 0/1 loss and causes overfitting

Comparison When Coupled with AdaBoost



- single perceptron sufficient on $6/12$ sets
- AdaBoost-RCD significantly better than any single perceptron on the other half
- AdaBoost-SVM cannot improve; AdaBoost-pocket slow

for modeling very complex data sets with perceptron ensembles, AdaBoost-RCD is the best

Conclusion

- Random Coordinate Descent: an efficient algorithm guaranteed to minimize $0/1$ loss of perceptron
- theoretical analysis:
  proved to converge to $\mathbf{w}^*$ and to perform fast local search
- empirical study:
  - RCD the best training error minimizer
    – but can cause overfitting
  - AdaBoost-RCD the best perceptron ensemble approach in test performance

Thank you. Questions?