Outline

- Solving the primal problem: stochastic gradient method (SG)
- Solving the dual problem: dual coordinate descent (DCD)
- Solving kernel SVM by greedy coordinate descent
- Scaling to large-scale datasets
Support Vector Machines

- Given training data \( \mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d \) with labels \( y_i \in \{+1, -1\} \).
- SVM primal problem:

\[
\min_{\mathbf{w}} \frac{1}{2} \mathbf{w}^T \mathbf{w} + C \sum_{i=1}^{n} \max(0, 1 - y_i \mathbf{w}^T \mathbf{x}_i)
\]
Stochastic Gradient for SVM
Stochastic Gradient

- Decompose the problem into \( n \) parts:

\[
f(w) = \frac{1}{n} \sum_{i} \left( \frac{1}{2} \|w\|^2 + nC \max(0, 1 - y_i w^T x_i) \right)
\]

\[
:= \frac{1}{n} \sum_{i} f_i(w)
\]

- Stochastic Gradient (SG):

For \( t = 1, 2, \ldots \)

Randomly pick an index \( i \)

\[
w^{t+1} \leftarrow w^t - \eta_t \nabla f_i(w^t)
\]

- But \( f_i \) is non-differentiable.
A vector $g$ is a subgradient of $f$ at a point $x_0$ if

$$f(x) - f(x_0) \geq g^T (x - x_0) \quad \forall x$$

Stochastic Subgradient descent:

For $t = 1, 2, \ldots$

Randomly pick an index $i$

$$w^{t+1} \leftarrow w^t - \eta_t \nabla g_i,$$ where $g_i$ is a subgradient of $f_i$ at $w^t$
A subgradient of $\ell_i(w) = \max(0, 1 - y_i w^T x_i)$:

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

Stochastic Subgradient descent for SVM:

For $t = 1, 2, \ldots$
- Randomly pick an index $i$
- If $y_i w^T x_i < 1$, then
  $$
  w \leftarrow (1 - \eta_t)w + \eta_t n C y_i x_i
  $$
- Else (if $y_i w^T x_i \geq 1$):
  $$
  w \leftarrow (1 - \eta_t)w
  $$
Stochastic Subgradient Method

- Improve the time complexity when $\mathbf{x}_i$ is sparse:
  - $w \leftarrow (1 - \eta_t) w$: $O(d)$ time complexity
  - Instead, we maintain $w = a \mathbf{v}$, so this operation requires only $O(1)$ time
  - $w \leftarrow w - \eta n_C y_i x_i$: $O(n_i)$ time where $n_i$ is the number of nonzeroes in $\mathbf{x}_i$

- This algorithm was proposed in:
  
  Shalev-Shwartz et al., “Pegasos: Primal Estimated sub-GrAdient SOlver for SVM”, in ICML 2007
Dual Coordinate Descent for SVM
Dual Form of SVM

- Given training data \( \mathbf{x}_1, \cdots, \mathbf{x}_n \in \mathbb{R}^d \) with labels \( y_i \in \{+1, -1\} \).

- SVM dual problem:

  \[
  \min_{0 \leq \alpha \leq C} \frac{1}{2} \alpha^T \mathbf{Q} \alpha - \sum_{i=1}^{n} \alpha_i
  \]

  where \( \mathbf{Q} \) is an \( n \) by \( n \) matrix with \( Q_{ij} = y_i y_j \mathbf{x}_i^T \mathbf{x}_j \).

- After computing the dual optimal solution \( \alpha^* \), the primal solution is

  \[
  \mathbf{w}^* = \sum_{i=1}^{n} y_i \alpha^*_i \mathbf{x}_i
  \]
Stochastic Coordinate descent for solving $\min_{0 \leq \alpha \leq C} f(\alpha)$:

For $t = 1, 2, \ldots$
  Randomly pick an index $i$
  Compute the optimal one-variable update:

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

Update $\alpha_i \leftarrow \alpha_i + \delta^*$
One-variable subproblem:

\[
f(\alpha + \delta e_i) = \frac{1}{2}(\alpha + \delta e_i)^T Q(\alpha + \delta e_i) - \sum_{i=1}^{n} \alpha_i - \delta
\]

\[
= \frac{1}{2} \alpha^T Q \alpha + \alpha^T Q e_i \delta + \frac{Q_{ii}}{2} \delta^2 - \sum_{i=1}^{n} \alpha_i - \delta
\]

Compute \( \arg\min_{\delta} f(\alpha + \delta e_i) \): set gradient equals to zero

\[
(Q\alpha)_i + Q_{ii} \delta^* - 1 = 0 \implies \delta^* = \frac{1 - (Q\alpha)_i}{Q_{ii}}
\]

However, we require \( 0 \leq \alpha_i + \delta \leq C \), so the optimal solution is

\[
\delta^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - (Q\alpha)_i}{Q_{ii}} \right) \right)
\]
Main computation: the $i$-th element of $Q\alpha$

Time complexity:
Dual Coordinate Descent for SVM

- Main computation: the \(i\)-th element of \(Q\alpha\)
- Time complexity:
  - Assume each data point \(x_i\) has \(\bar{d}\) nonzero elements (in average), so \(\text{nnz}(X) = n\bar{d}\).
  - Assume we use Gaussian kernel, so each \(Q_{ij} = \exp(-\gamma\|x_i - x_j\|^2)\) requires \(O(\bar{d})\) computational time.
  - If \(Q \in \mathbb{R}^{n \times n}\) stored in memory: \(O(n)\) time complexity, \(O(n^2)\) space complexity.
  - If \(Q\) is not stored in memory:
    - Compute \(Q_{i,1}, \ldots, Q_{i,n}\): \(O(\text{nnz}(X))\) time complexity
    - Compute \(\sum_{j=1}^{n} Q_{i,j}\alpha_j\): \(O(n)\) time complexity
- What if \(n = 1,000,000\)?
A faster way to compute coordinate descent updates.

Since $Q_{ij} = y_i y_j x_i^T x_j$, we can define

$$Q = \bar{X} \bar{X}^T,$$

where the $i$-th row of $\bar{X}$ is $y_i x_i$.

Now, maintain $\mathbf{w} = \bar{X}^T \alpha$ during the whole procedure, then

$$(Q\alpha)_i = (\bar{X} \bar{X}^T \alpha)_i = y_i x_i^T \mathbf{w}$$

Time complexity: $O(d_i)$ (number of nonzeros of $x_i$)
After updating $\alpha_i \leftarrow \alpha_i + \delta^*$, we need to maintain $w$:

$$w \leftarrow w + \delta^* y_i x_i$$

Time complexity: $O(d_i)$

After convergence,

$$w^* = \sum_{i=1}^{n} \alpha_i^* y_i x_i$$

is the optimal primal solution.
Initial: $\alpha, \mathbf{w} = \sum_{i=1}^{n} \alpha_i y_i \mathbf{x}_i$

For $t = 1, 2, \ldots$

Randomly pick an index $i$

Compute the optimal one-variable update:

$$
\delta^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - y_i \mathbf{w}^T \mathbf{x}_i}{Q_{ii}} \right) \right)
$$

Update $\alpha_i \leftarrow \alpha_i + \delta^*$

Update $\mathbf{w} \leftarrow \mathbf{w} + \delta^* y_i \mathbf{x}_i$

(Hsieh et al., “A Dual Coordinate Descent Method for Large-scale Linear SVM”, ICML 2008)
Convergence of Dual Coordinate Descent

- Is the dual SVM problem strongly convex?
Is the dual SVM problem strongly convex?

\[ Q = \bar{X} \bar{X}^T \] may be low-rank

Sublinear convergence in dual objective function value.

Sublinear convergence in duality gap (primal obj - dual obj)

(Shown in Shalev-Shwartz and Zhang, “Stochastic Dual Coordinate Ascent Methods for Regularized Loss Minimization”. JMLR 2013).

Global linear convergence rate in terms of dual objective function value.

Experimental comparison

- RCV1: 677,399 training samples; 47,236 features; 49,556,258 nonzeroes in the whole dataset.

Time vs primal objective function value

(e) L1-SVM: rcv1
(f) L2-SVM: rcv1

Time vs primal objective function value
Experimental comparison

- RCV1: 677,399 training samples; 47,236 features; 49,556,258 nonzeros in the whole dataset.

Time vs prediction accuracy
Implemented in LIBLINEAR:

https://www.csie.ntu.edu.tw/~cjlin/liblinear/

Other functionalities:
- Logistic regression (L1 or L2 regularization)
- Multi-class SVM
- Support vector regression
- Cross-validation
Greedy Coordinate Descent for Kernel SVM
Nonlinear SVM

- SVM primal problem:

  \[
  \min_{w, \xi} \frac{1}{2} w^T w + C \sum_{i=1}^{n} \xi_i \quad \text{s.t.} \quad y_i (w^T \varphi(x_i)) \geq 1 - \xi_i, \quad \xi_i \geq 0, \ i = 1, \ldots, n,
  \]

  \(w\): an infinite dimensional vector, very hard to solve

- The dual problem for SVM:

  \[
  \min_{\alpha} \frac{1}{2} \alpha^T Q \alpha - e^T \alpha, \quad \text{s.t.} \quad 0 \leq \alpha_i \leq C, \ \text{for} \ i = 1, \ldots, n,
  \]

  where \(Q_{ij} = y_i y_j \varphi(x_i)^T \varphi(x_j)\) and \(e = [1, \ldots, 1]^T\).

- Kernel trick: define \(K(x_i, x_j) = \varphi(x_i)^T \varphi(x_j)\).

- Example: Gaussian kernel \(K(x_i, x_j) = e^{-\gamma \|x_i - x_j\|^2}\)
Can we solve the problem by dual coordinate descent?

The vector $\mathbf{w} = \sum_i y_i \alpha_i \varphi(x_i)$ may have infinite dimensionality:

Cannot maintain $\mathbf{w}$

Closed form solution needs $O(n)$ computational time:

$$
\delta^* = \max(-\alpha_i \min(C - \alpha_i, \frac{1 - (Q \alpha)_i}{Q_{ii}}))
$$

(Assume $Q$ is stored in memory)

Can we improve coordinate descent using the same $O(n)$ time complexity?
The Greedy Coordinate Descent (GCD) algorithm:

For $t = 1, 2, \ldots$

1. Compute $\delta^*_i := \arg\min_{\delta} f(\alpha + \delta e_i)$ for all $i = 1, \ldots, n$

2. Find the best $i^*$ according to the following criterion:

$$i^* = \arg\max_i |\delta^*_i|$$

3. $\alpha_{i^*} \leftarrow \alpha_{i^*} + \delta^*_{i^*}$
Greedy Coordinate Descent

Other variable selection criterion:

- The coordinate with the maximum step size:

\[ i^* = \arg\max_i |\delta_i^*| \]

- The coordinate with maximum objective function reduction:

\[ i^* = \arg\max_i (f(\alpha) - f(\alpha + \delta_i^* e_i)) \]

- The coordinate with the maximum projected gradient.

...
Greedy Coordinate Descent

How to compute the optimal coordinate?

Closed form solution of best $\delta$:

$$\delta^*_i = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - (Q\alpha)_i}{Q_{ii}} \right) \right)$$

Observations:

1. Computing all $\delta^*_i$ needs $O(n^2)$ time
2. If $Q\alpha$ is stored in memory, computing all $\delta^*_i$ only needs $O(n)$ time

Maintaining $Q\alpha$ also needs $O(n)$ time after each update
Greedy Coordinate Descent

Initial: $\alpha, z = Q\alpha$

For $t = 1, 2, \ldots$

For all $i = 1, \ldots, n$, compute

$$\delta^*_i = \max \left( -\alpha_i, \min(C - \alpha_i, \frac{1 - z_i}{Q_{ii}}) \right)$$

Let $i^* = \arg\max_i |\delta^*_i|$

$$\alpha \leftarrow \alpha + \delta^*_{i^*}$$

$$z \leftarrow z + q_{i^*} \delta^*_{i^*} \quad (q_{i^*} \text{ is the } i^*\text{-th column of } Q)$$

(This is a simplified version of the Sequential Minimal Optimization (SMO) algorithm proposed in Platt el al., 1998)

(A similar version is implemented in LIBSVM)
Why greedy coordinate descent?

- Let $\bar{d}$ be the average number of nonzeroes per instance, $n$ be the number of training samples.
- Usually, $\bar{d} \ll n$.
- If $Q$ can be stored in memory

<table>
<thead>
<tr>
<th></th>
<th>Nonlinear SVM</th>
<th>Linear SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Greedy CD</td>
<td>Cyclic CD</td>
</tr>
<tr>
<td>Update $\alpha_i$</td>
<td>$O(1)$</td>
<td>$O(n)$</td>
</tr>
<tr>
<td>Maintain $Q\alpha$</td>
<td>$O(n)$</td>
<td>NA</td>
</tr>
</tbody>
</table>

- If $Q$ cannot be stored in memory

<table>
<thead>
<tr>
<th></th>
<th>Nonlinear SVM</th>
<th>Linear SVM</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Greedy CD</td>
<td>Cyclic CD</td>
</tr>
<tr>
<td>Update $\alpha_i$</td>
<td>$O(1)$</td>
<td>$O(nd\bar{d})$</td>
</tr>
<tr>
<td>Maintain $Q\alpha$</td>
<td>$O(nd\bar{d})$</td>
<td>NA</td>
</tr>
</tbody>
</table>
How to solve problems with millions of samples? (*)

- $Q \in \mathbb{R}^{n \times n}$ cannot be fully stored
- Have a fixed size of memory to “cache” the computed columns of $Q$
- For each coordinate update:
  - If $q_i$ is in memory, directly use it to update
  - If $q_i$ is not in memory
    1. Kick out the “Least Recent Used” column
    2. Recompute $q_i$ and store it in memory
    3. Update $\alpha_i$
Implemented in LIBSVM:

https://www.csie.ntu.edu.tw/~cjlin/libsvm/

Other functionalities:

- Multi-class classification
- Support vector regression
- Cross-validation
Recent Research Topics
Asynchronous dual coordinate descent algorithm:

Each thread repeatedly performs the following updates:
For $t = 1, 2, \ldots$

Randomly pick an index $i$
Compute the optimal one-variable update:

$$
\delta_i^* = \max \left( -\alpha_i, \min \left( C - \alpha_i, \frac{1 - y_i w^T x_i}{Q_{ii}} \right) \right)
$$

Update $\alpha_i \leftarrow \alpha_i + \delta_i^*$
Update $w \leftarrow w + \delta_i^* y_i x_i$

Different mechanisms on accessing $w$ in the shared memory.

Other multi-core algorithms for SVM

- Other parallel primal solvers:
Question: how to solve linear SVM on a single machine when data cannot fit in memory?

Dual coordinate descent: need random access, not suitable when training samples are stored in disk
Large-scale linear: out-of-core version

- Block coordinate descent:
  - Partition data into blocks $S_1, \ldots, S_k$
  - For $t = 1, 2, \ldots$
    - Load a block $S_i$ into memory
    - Update this block of dual variables by dual coordinate descent

  (Yu et al., “Large Linear Classification when Data Cannot Fit In Memory”, in KDD 2010)

- Selective block coordinate descent: keep important samples in memory
  
  (Chang and Roth, “Selective Block Minimization for Faster Convergence of Limited Memory Large-scale Linear Models”, in KDD 2011)

- StreamSVM: one thread keep loading data, while another thread keep running coordinate descent updates

  (Matsushima et al., “Linear Support Vector Machines via Dual Cached Loops”, in KDD 2012)
Comparisons

On webspam dataset, 0.35 million samples, 16.61 million features, dataset size 20.03GB
Distributed Linear SVMs

- Each machine stores a subset of training samples
- Each machine conducts dual coordinate updates and has a local $w$
- How to communicate and synchronize the updates?

  Paper presentations next week
Next class: Matrix Completion for Recommender Systems

Questions?