Fast Training of SVMs using Sequential Minimal Optimization

Presented by
Sudip Khanna & Ajit Datar

The Sequential Minimal Optimization (SMO) Algorithm

SMO solves the SVM QP problem by decomposing it into QP sub-problems and solving the smallest possible optimization problem, involving two Lagrange multipliers, at each step.
Part One - Background

- QP Problems
- SVM QP problem
- Lagrange Multipliers
- KKT Conditions

What’s a QP problem?

Maximize/Minimize

a Quadratic Objective Function

subject to a Set of Linear Constraints
The SVM QP problem

Maximize margin

\[
\frac{2|k|}{\|w\|}
\]

subject to

\[(w.x + b) \geq k,\]

Vx of class 1

\[(w.x + b) \leq -k,\]

Vx of class 2
The SVM QP problem

We can scale the data so that $k = 1$

The problem now reduces to

$$\min \frac{1}{2} \|w\|^2$$

$$\text{s.t. } y_i(w \cdot x_i + b) \geq 1, \forall x_i$$

where, $y_i$ is the classification for example $x_i$ (1 or -1)

The Lagrangian

The objective function and the constraints are combined in a single function

$$L(w, b, \alpha) = \frac{1}{2} \|w\|^2 - \sum \alpha_i (y_i \cdot ((x_i \cdot w) + b) - 1)$$

where, Lagrange multiplier, $\alpha_i \geq 0$

For $L$ to be maximized, only training examples with

$$y_i \cdot ((x_i \cdot w) + b) - 1 = 0 \quad \text{(support vectors)}$$

will have $\alpha_i \neq 0$
BTW what’s a Lagrange Multiplier?

- It is the ratio
  \[
  \frac{\text{Gradient of Objective Function}}{\text{Gradient of Constraint Function}}
  \]
  At the solution of the problem
- In SVM context it allows us to simplify the constraints.
  When the problem is expressed with Lagrangian multipliers \((\alpha_i)\) the only constraints are non-negative \(\alpha_i\).

KKT conditions – what they mean

- The solution which satisfies the KKT conditions is an **optimal solution**
- In SVM equations this means
  \[
  \sum \alpha_i y_i = 0
  \]
  \(\Rightarrow\) Only support vectors contribute to the constraints on the margin
Part Two: SMO algorithm

- Why do we need SMO?
  - Previous methods (chunking, decomposition)
  - Numerical Vs Analytical Methods of optimization
  - SMO: 3 part solution

Why do we need SMO?

- Current methods are based on Numerical Optimization
- Require calling library routines for solving optimization problems.
- Manipulation of large matrices => more numerical precision errors.
- Exponential memory requirements.
Previous methods

- **Chunking**: Optimizes chunks of examples at a time:
  - With non-zero Lagrange multipliers from last step
  - M worst examples (violators of KKT conditions)

- **Decomposition**: At each step add one violator example and optimize the new set.
  - Osuna’s optimization: const size matrix, delete an example whenever a new one is added.

Numerical Vs Analytical solvers

- **Numerical**
  - QP sub-problem solved iteratively
  - Subject to precision errors due to large matrices
  - Smaller number of QP sub-problems

- **Analytical**
  - QP sub-problem solved in fixed number of steps.
  - Small matrices => lesser precision errors.
  - Large number of smaller QP sub-problems
Sequential Minimal Optimization

- Not parallel
- Optimize in sets of 2 Lagrange multipliers

Satisfy the constraints for the chosen pair of Lagrange multipliers.

Optimize smallest possible sub-problem at each step.

---

SMO components

- Heuristics for choosing Lagrange multipliers
- Analytical method for 2 Lagrange multipliers
- Compute $b$ such both examples satisfy KKT

Do this until the entire training set obeys the KKT conditions
Solving for two Lagrange Multipliers

- **Constraints on the Lagrange Multipliers**
  - Bound constraints: $0 \leq \alpha_i \leq C$
  - Linear equality constraint: $\sum \alpha_i y_i = 0$

![Constraint Diagrams](image)

Choosing the multipliers

- All or no examples seen?
  - Yes: Pick a KKT violator from ALL examples
  - No: Pick a KKT violator from non-bound examples

- Choose 1st LM
  - Choose an example which maximizes step size (approx $|E1 - E2|$

- Choose 2nd LM
  - Pass to the Analytical solver
Calculating threshold ‘b’

- If the data is linearly separable, there is a unique value of b that maximizes margin
- b is recomputed after each step such that KKT conditions are fulfilled for both optimized examples

Speeding things up …

- Store the error associated with each example in cache
- Store and update a single weight vector which represents all examples
- SMO can take advantage of sparse input data
Relationship to previous algorithms

- Can be considered to be a special case of Osuna’s algorithm
- Similar to Bregman methods of optimization

### SMO & PCG Chunking on Adult dataset

<table>
<thead>
<tr>
<th>Training-Set Size</th>
<th>SMO-Time (CPU sec)</th>
<th>PCG-Time (CPU sec)</th>
<th>SMO Iterations</th>
<th>PCG Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>1605</td>
<td>0.4</td>
<td>37.1</td>
<td>3230</td>
<td>1328</td>
</tr>
<tr>
<td>2265</td>
<td>0.9</td>
<td>228.3</td>
<td>4635</td>
<td>3964</td>
</tr>
<tr>
<td>3185</td>
<td>1.8</td>
<td>596.2</td>
<td>6950</td>
<td>6742</td>
</tr>
<tr>
<td>4781</td>
<td>3.6</td>
<td>1954.2</td>
<td>9847</td>
<td>10550</td>
</tr>
<tr>
<td>6414</td>
<td>5.5</td>
<td>3684.6</td>
<td>10669</td>
<td>12263</td>
</tr>
<tr>
<td>11221</td>
<td>17</td>
<td>20711.3</td>
<td>17128</td>
<td>25400</td>
</tr>
<tr>
<td>16101</td>
<td>35.3</td>
<td>N/A</td>
<td>22770</td>
<td>N/A</td>
</tr>
<tr>
<td>22697</td>
<td>85.7</td>
<td>N/A</td>
<td>35822</td>
<td>N/A</td>
</tr>
<tr>
<td>32562</td>
<td>163.6</td>
<td>N/A</td>
<td>44774</td>
<td>N/A</td>
</tr>
</tbody>
</table>
SMO & PCG Chunking on web dataset

<table>
<thead>
<tr>
<th>Training-Set Size</th>
<th>SMO-Time (CPU sec)</th>
<th>PCG-Time (CPU sec)</th>
<th>SMO Iterations</th>
<th>PCG Iterations</th>
</tr>
</thead>
<tbody>
<tr>
<td>2477</td>
<td>2.2</td>
<td>13.1</td>
<td>25296</td>
<td>1929</td>
</tr>
<tr>
<td>3470</td>
<td>4.9</td>
<td>16.1</td>
<td>46830</td>
<td>2379</td>
</tr>
<tr>
<td>4912</td>
<td>8.1</td>
<td>40.6</td>
<td>66890</td>
<td>4110</td>
</tr>
<tr>
<td>7366</td>
<td>12.7</td>
<td>140.7</td>
<td>88948</td>
<td>7416</td>
</tr>
<tr>
<td>9888</td>
<td>24.7</td>
<td>239.3</td>
<td>141538</td>
<td>8700</td>
</tr>
<tr>
<td>17188</td>
<td>65.4</td>
<td>1633.3</td>
<td>268907</td>
<td>27074</td>
</tr>
<tr>
<td>24692</td>
<td>104.9</td>
<td>3369.7</td>
<td>345736</td>
<td>32014</td>
</tr>
<tr>
<td>49749</td>
<td>268.3</td>
<td>17164.7</td>
<td>489302</td>
<td>63817</td>
</tr>
</tbody>
</table>

Conclusions

- SMO has potential for speed-up
- Scales well – memory footprint grows linearly with training set size
- Easier to implement – does not require a QP library