P-packSVM: Parallel Primal grAdient desCent Kernel SVM

Zeyuan A. Zhu (Tsinghua University)
Weizhu Chen (MSRA)
Gang Wang (MSRA)
Chenguang Zhu (Tsinghua University)
Zheng Chen (MSRA)
Content

- Introduction
- Related Works
- Sequential packSVM
- Parallel packSVM
  - Packing Strategy
- Comparison
- Experimental Results
- Further Discussion

November 13, 2009
Content

- Introduction
- Related Works
- Sequential packSVM
  - Packing Strategy
- Parallel packSVM
- Comparison
- Experimental Results
- Further Discussion
Content

- Introduction
- Related Works
- Sequential packSVM
- Parallel packSVM
  - Packing Strategy
- Comparison
- Experimental Results
- Further Discussion
Content

- Introduction
- Related Works
- Sequential packSVM
  - Packing Strategy
- Parallel packSVM
- Comparison
- Experimental Results
- Further Discussion
E.g.

- CCAT dataset 781,265 / 23,149
- SVM-light?
  - 219,744s
  - 95.18%
- PSVM? **256 processors**
  - 18,173s
  - 89.04%
E.g.

- CCAT dataset 781,265 / 23,149
  - SVM-light?
    - 219,744s
    - 95.18%
  - PSVM?
    - 256 processors
    - 18,173s
    - 89.04%
  - P-packSVM?
    - 256 processors
    - 918s
    - 95.07%
Introduction

Given a set of training data:
\[ \Psi = \{(x_i, y_i) | x_i \in \mathbb{R}^n, y_i \in \{-1, 1\}\}_{i=1}^m \]

Primal SVM objective
\[ f(w) = \frac{\sigma}{2} \|w\|_2^2 + \frac{1}{m} \sum_{i=1}^m \max\{0, 1 - y_i \langle w, \phi(x_i) \rangle\} \]

Kernel trick (non-linear)
\[ \phi(.): \text{feature space} \rightarrow \text{Reproducing Kernel Hilbert Space (RKHS)}, \text{satisfying} \]
\[ \langle \phi(x_i)^T, \phi(x_j) \rangle = \mathcal{K}(x_i, x_j) \]
for some Mercer kernel \( \mathcal{K}(., .) \)
**Kernel trick (non-linear)**

\( \phi(\cdot) \): feature space \( \rightarrow \) Reproducing Kernel Hilbert Space (RKHS), satisfying

\[
\langle \phi(x_i)^T, \phi(x_j) \rangle = \mathcal{K}(x_i, x_j)
\]

for some Mercer kernel \( \mathcal{K}(\cdot, \cdot) \)
## Related Works

<table>
<thead>
<tr>
<th>Linear?</th>
<th>Well-studied</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Refer to Pegasos, and Inverse Time Dependency</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Kernel?</th>
<th>Interior Point Method (IPM)</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>Sequential Minimal Optimization (SMO)</td>
</tr>
<tr>
<td></td>
<td>Stochastic Gradient Descent (SGD)</td>
</tr>
</tbody>
</table>
Dual objective:
\[ \min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0 \]
where \([Q]_{ij} = y_i y_j K(x_i, x_j)\), and \(\alpha \in \mathbb{R}^m\) is the vector of the *Lagrangian dual variable*, also known as the *support vector* in SVM.
Kernel?

Interior Point Method (IPM)

Dual objective:
\[
\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0
\]

where \([Q]_{ij} = y_i y_j K(x_i, x_j)\), and \(\alpha \in \mathbb{R}^m\) is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

Newton methods
convergence rate: \(\log(1/\epsilon)\)
Kernel?

Interior Point Method (IPM)

Dual objective:

$$\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0$$

where $[Q]_{ij} = y_i y_j \mathcal{K}(x_i, x_j)$, and $\alpha \in \mathbb{R}^m$ is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

memory requirements $O(m^2)$

computational cost $O(m^3)$ for each iteration
**Kernel?**

**Interior Point Method (IPM)**

Dual objective:

\[
\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t.} \ 0 \leq \alpha \leq C, y^T \alpha = 0
\]

where \([Q]_{ij} = y_i y_j K(x_i, x_j)\), and \(\alpha \in \mathbb{R}^m\) is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

E. Y. Chang *et al* proposed an algorithm called **PSVM**

+ Incomplete Cholesky Factorization (ICF) \(Q \approx HH^T\)

+ \(H\) has the dimension \(m \times m'\) (in general, \(m' = m^{0.5}\))

+ time: \(O(m^2/p)\) for each iteration

+ space: \(O(m^{1.5}/p)\)
Dual objective:
\[
\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0
\]

where \( [Q]_{ij} = y_i y_j \mathcal{K}(x_i, x_j) \), and \( \alpha \in \mathbb{R}^m \) is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

E. Y. Chang et al proposed an algorithm called PSVM
+ 500 processors in experiments
+ parallel speed-up of up to 169 times
+ 800k training samples
Dual objective:

\[
\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0
\]

where \( [Q]_{ij} = y_i y_j K(x_i, x_j) \), and \( \alpha \in \mathbb{R}^m \) is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

E. Y. Chang et al proposed an algorithm called PSVM
+ 500 processors in experiments
+ parallel speed-up of up to 169 times
+ 800k training samples
Kernel?

Sequential Minimal Optimization (SMO)

Dual objective:

$$\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t.} \quad 0 \leq \alpha \leq C, \ y^T \alpha = 0$$

where $[Q]_{ij} = y_i y_j \mathcal{K}(x_i, x_j)$, and $\alpha \in \mathbb{R}^m$ is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

- decompose the QP problem into an inactive part and an active part – a so called “working set”
- The fastest sequential Kernel SVM Solver.
- SVM-light, libSVM, etc.
Related Works

Kernel?

Sequential Minimal Optimization (SMO)

Dual objective:

$$\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0$$

where $[Q]_{ij} = y_i y_j K(x_i, x_j)$, and $\alpha \in \mathbb{R}^m$ is the vector of the Lagrangian dual variable, also known as the support vector in SVM.

Parallel?

+ Hard to parallel (high number of iterations -> high communication cost)
+ Zanghirati and Zanni: a parallel implementation of SVM-light, especially effective for Gaussian kernels
+ Cao et al also parallelized a slightly modified SMO algorithm
Kernel?

Sequential Minimal Optimization (SMO)

Dual objective:

$$\min \frac{1}{2} \alpha^T Q \alpha - \alpha^T 1, \quad \text{s.t. } 0 \leq \alpha \leq C, y^T \alpha = 0$$

where $$[Q]_{ij} = y_i y_j \mathcal{K}(x_i, x_j)$$, and $$\alpha \in \mathbb{R}^m$$ is the vector of the Lagrangian dual variable, also known as the support vector in SVM.
Related Works

**Kernel?**

**Stochastic Gradient Descent (SGD)**

- Optimize on the primal directly.

- T. Zhang proved that a constant learning rate will achieve $O(1/\epsilon^2)$ convergence rate.

- Shai Shalev-Shwartz *et al* upgraded it to $O(1/\epsilon)$ convergence rate.

- Most of the previous works pay attention only to linear SVM.

- It was believed hard to generalize to kernel SVM due to high complexity.
**Sequential packSVM**

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $\mathbf{w} = 0$
3. **FOR** $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(x) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. **IF** $yy' < 1$ **THEN** $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(x)$
8. $\mathbf{w} \leftarrow \min \left\{1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. **RETURN** $\mathbf{w}$
Sequential packSVM

1. **INPUT**: $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE**: $\mathbf{w} = 0$
3. **FOR** $t = 1, 2, ..., T$
4. Randomly pick up $(\mathbf{x}, y) \in \Psi$
5. **Predict** $y' \leftarrow \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. **IF** $y y' < 1$ **THEN** $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(\mathbf{x})$
8. $\mathbf{w} \leftarrow \min\left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. **RETURN** $\mathbf{w}$

$$f(\mathbf{w}) = \frac{\sigma}{2} \|\mathbf{w}\|_2^2 + \frac{1}{m} \sum_{i=1}^{m} \max\{0, 1 - y_i \langle \mathbf{w}, \phi(\mathbf{x}_i) \rangle\}$$
Sequential packSVM

1. INPUT: $\sigma, T$, training sample space $\Psi$
2. INITIALIZE: $\mathbf{w} = 0$
3. FOR $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(x) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. IF $yy' < 1$ THEN $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(x)$
8. $\mathbf{w} \leftarrow \min \left\{1, \frac{1}{\sqrt{\sigma}} \right\} \mathbf{w}$
9. RETURN $\mathbf{w}$
Sequencial packSVM

1. **INPUT**: $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE**: $w = 0$
3. **FOR** $t = 1, 2, ..., T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. If $yy' < 1$ THEN $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|w\|_2} \right\} w$
9. **RETURN** $w$
Sequential packSVM

1. INPUT: $\sigma, T$, training sample space $\Psi$
2. INITIALIZE: $\mathbf{w} = 0$
3. FOR $t = 1, 2, \ldots, T$
   4. Randomly pick up $(\mathbf{x}, y) \in \Psi$
   5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$
   6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
   7. IF $yy' < 1$ THEN $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(\mathbf{x})$
   8. $\mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{||\mathbf{w}||_2} \right\} \mathbf{w}$
9. RETURN $\mathbf{w}$
Sequential packSVM

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $w = 0$
3. **FOR** $t = 1, 2, ..., T$
4. Randomly pick up $(x, y) \in \Psi$
5. **Predict** $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{||w||_2} \right\} w$
9. **RETURN** $w$
1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $\mathbf{w} = 0$
3. FOR $t = 1, 2, \ldots, T$
4. Randomly pick up $(\mathbf{x}, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. IF $yy' < 1$ THEN $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(\mathbf{x})$
8. $\mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. RETURN $\mathbf{w}$
Sequential packSVM

1. **INPUT:** $\sigma, T, \text{training sample space } \Psi$
2. **INITIALIZE:** $\mathbf{w} = 0$
3. **FOR** $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(x) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. **IF** $yy' < 1$ **THEN** $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(x)$
8. $\mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. **RETURN** $\mathbf{w}$

write $\mathbf{w}$ as a superposition of samples 
$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(x_i)$
Sequential packSVM

1. INPUT: $\sigma, T$, training sample space $\Psi$
2. INITIALIZE: $\mathbf{w} = 0$
3. FOR $t = 1, 2, \ldots, T$
4. Randomly pick up $(\mathbf{x}, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(\mathbf{x}) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. IF $yy' < 1$ THEN $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(\mathbf{x})$
8. $\mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. RETURN $\mathbf{w}$
Sequential packSVM

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $\mathbf{w} = 0$
3. FOR $t = 1,2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle \mathbf{w}, \phi(x) \rangle$
6. $\mathbf{w} \leftarrow (1 - 1/t)\mathbf{w}$
7. IF $yy' < 1$ THEN $\mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(x)$
8. $\mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w}$
9. **RETURN** $\mathbf{w}$

**Non-linear Extension to PEGASOS**

- Convergence rate of $1/\epsilon$

**Profound mathematical background hidden behind the simply algorithm**

- Online Strongly Convex Optimization
- Fenchel Conjugate
Sequential packSVM

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $w = 0$
3. **FOR** $t = 1, 2, ..., T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min \left\{1, \frac{1/\sqrt{\sigma}}{\|w\|_2} \right\} w$
9. **RETURN** $w$

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$
Parallel packSVM

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $w = 0$
3. **FOR** $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min \left\{1, \frac{1}{\|w\|_2} \right\}w$
9. **RETURN** $w$

\[ y' \leftarrow s\langle v, \phi(x) \rangle \]
\[ s \leftarrow (1 - 1/t)s \]
\[ v \leftarrow v + \frac{y}{\sigma ts} \phi(x) \]
\[ s \leftarrow \min \left\{1, \frac{1}{\|w\|_2} \right\} s \]

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$

**Parallelization:** Save $v$ in a distributed manner!
## Parallel packSVM

1. **INPUT:** $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE:** $w = 0$
3. **FOR** $t = 1, 2, \ldots, T$
   4. Randomly pick up $(x, y) \in \Psi$
   5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
   6. $w \leftarrow (1 - 1/t)w$
   7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
   8. $w \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{||w||_2} \right\} w$
9. **RETURN** $w$

### Algorithm Steps

- **Predict:** $y' = \langle w, \phi(x) \rangle$
- **Update:** $w = (1 - 1/t)w$
- **Update if necessary:**
  - $w = w + \frac{y}{\sigma t} \phi(x)$
  - $w = \min \left\{ 1, \frac{1/\sqrt{\sigma}}{||w||_2} \right\} w$

### Time Complexity

- $y' = O(1)$
- $s = O(1)$
- $v = O(1)$
- $w = O(1)$

---

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$

**Parallelization:** Save $v$ in a distributed manner!
1. **INPUT**: $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE**: $w = 0$
3. **FOR** $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{||w||_2} \right\} w$
9. **RETURN** $w$

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$.

Parallelization: Save $v$ in a distributed manner!

A single iteration can be highly parallelized.
1. **INPUT**: $\sigma, T$, training sample space $\Psi$
2. **INITIALIZE**: $w = 0$
3. **FOR** $t = 1, 2, \ldots, T$
4. Randomly pick up $(x, y) \in \Psi$
5. Predict $y' \leftarrow \langle w, \phi(x) \rangle$
6. $w \leftarrow (1 - 1/t)w$
7. **IF** $yy' < 1$ **THEN** $w \leftarrow w + \frac{y}{\sigma t} \phi(x)$
8. $w \leftarrow \min\left\{1, \frac{1}{\|w\|_2} \right\} w$
9. **RETURN** $w$

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$

**Parallelization**: Save $v$ in a distributed manner!

**Two many iterations exist**
1. **INPUT**: \( \sigma, T, \) training sample space \( \Psi \)
2. **INITIALIZE**: \( \mathbf{w} = 0 \)
3. **FOR** \( t = 1, 2, \ldots, T \)
4. Randomly pick up \( (x, y) \in \Psi \)
5. Predict \( y' \leftarrow \langle \mathbf{w}, \phi(x) \rangle \)
6. \( \mathbf{w} \leftarrow (1 - 1/t)\mathbf{w} \)
7. **IF** \( yy' < 1 \) **THEN** \( \mathbf{w} \leftarrow \mathbf{w} + \frac{y}{\sigma t} \phi(x) \)
8. \( \mathbf{w} \leftarrow \min \left\{ 1, \frac{1/\sqrt{\sigma}}{\|\mathbf{w}\|_2} \right\} \mathbf{w} \)
9. **RETURN** \( \mathbf{w} \)

In the implementation of S-packSVM, we express \( \mathbf{w} = s\mathbf{v} \), where \( s \in \mathbb{R} \) is a scalar, and \( \mathbf{v} = \sum_i \beta_i \phi(x_i) \)

Parallelization: Save \( \mathbf{v} \) in a distributed manner!

Two many iterations exist
In the implementation of S-packSVM, we express $\mathbf{w} = s\mathbf{v}$, where $s \in \mathbb{R}$ is a scalar, and $\mathbf{v} = \sum_i \beta_i \phi(x_i)$.

Parallelization: Save $\mathbf{v}$ in a distributed manner!

**Two many iterations exist**
In the implementation of S-packSVM, we express $\mathbf{w} = s \mathbf{v}$, where $s \in \mathbb{R}$ is a scalar, and $\mathbf{v} = \sum_i \beta_i \phi(x_i)$.

Parallelization: Save $\mathbf{v}$ in a distributed manner!

Two many iterations exist.
Two many iterations exist

In the implementation of S-packSVM, we express $\mathbf{w} = s \mathbf{v}$, where $s \in \mathbb{R}$ is a scalar, and $\mathbf{v} = \sum_i \beta_i \phi(x_i)$

Parallelization: Save $\mathbf{v}$ in a distributed manner!

Pack the consecutive $r$ iterations

$y_t' = \langle \mathbf{w}_t, \phi(x_t) \rangle$

$y_{t+1}' = \mathbf{w}_t \phi(x_{t+1}) + \mathcal{K}(x_t, x_{t+1})$

In the implementation of S-packSVM, we express $\mathbf{w} = s \mathbf{v}$, where $s \in \mathbb{R}$ is a scalar, and $\mathbf{v} = \sum_i \beta_i \phi(x_i)$
Two many iterations exist

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum \beta_i \phi(x_i)$.

Parallelization: Save $v$ in a distributed manner!
Two many iterations exist

In the implementation of S-packSVM, we express \( \mathbf{w} = s \mathbf{v} \), where \( s \in \mathbb{R} \) is a scalar, and \( \mathbf{v} = \sum_i \beta_i \phi(x_i) \).

Parallelization: Save \( \mathbf{v} \) in a distributed manner!
In the implementation of S-packSVM, we express \( \mathbf{w} = s \mathbf{v} \), where \( s \in \mathbb{R} \) is a scalar, and \( \mathbf{v} = \sum_i \beta_i \phi(x_i) \).

**Parallelization:** Save \( \mathbf{v} \) in a distributed manner!

Two many iterations exist
Packing Strategy

<table>
<thead>
<tr>
<th>$t$</th>
<th>$t+1$</th>
<th>…</th>
<th>$t+r-1$</th>
</tr>
</thead>
</table>

Pack the consecutive $r$ iterations

\[
y_t' = \langle w_t, \phi(x_t) \rangle
\]
\[
y_{t+1}' = \langle w_t, \phi(x_{t+1}) \rangle + \mathcal{K}(x_t, x_{t+1})
\]
\[
y_{t+2}' = \langle w_t, \phi(x_{t+2}) \rangle + \mathcal{K}(x_t, x_{t+2}) + \mathcal{K}(x_{t+1}, x_{t+2})
\]
\[
\vdots
\]
\[
y_{t+r-1}' = \langle w_t, \phi(x_{t+r-1}) \rangle + \mathcal{K}(x_t, x_{t+r-1}) + \ldots + \mathcal{K}(x_{t+r-2}, x_{t+r-1})
\]

After the above pre-calculation, each individual processor can learn these coefficients without any communication.

In the implementation of S-packSVM, we express $w = sv$, where $s \in \mathbb{R}$ is a scalar, and $v = \sum_i \beta_i \phi(x_i)$

Parallelization: Save $v$ in a distributed manner!

Two many iterations exist

November 13, 2009
Packing Strategy

<table>
<thead>
<tr>
<th></th>
<th>(t)</th>
<th>(t + 1)</th>
<th>(\ldots)</th>
<th>(t + r - 1)</th>
</tr>
</thead>
</table>

Pack the consecutive \(r\) iterations

\[
y'_t = \langle w_t, \phi(x_t) \rangle
\]

\[
y'_{t+1} = \mathbf{K}(w_t, \phi(x_{t+1})) + \mathbf{K}(x_t, x_{t+1})
\]

\[
y'_{t+2} = \mathbf{K}(w_t, \phi(x_{t+2})) + \mathbf{K}(x_t, x_{t+2}) + \mathbf{K}(x_{t+1}, x_{t+2})
\]

\[\vdots\]

\[
y'_{t+r-1} = \mathbf{K}(w_t, \phi(x_{t+r-1})) + \mathbf{K}(x_t, x_{t+r-1}) + \ldots + \mathbf{K}(x_{t+r-2}, x_{t+r-1})
\]

After the above pre-calculation, each individual processor can learn these coefficients without any communication.

In the implementation of S-packSVM, we express \(w = sv\), where \(s \in \mathbb{R}\) is a scalar, and \(v = \sum \beta_i \phi(x_i)\).

Parallelization: Save \(v\) in a distributed manner!

The number of communications divided by \(r\).
Hush et al proved that the dual approach converges slowly in the desired primal objective.

IPM/SMO: Dual

SGD: Primal!

IPM/SMO guarantees the descent

SGD: the accuracy fluctuate as the number of iterations increases

SVM

how well we optimize the objective

how well the objective is related to the accuracy

Optimization Effectiveness

Accuracy
Comparison

Speed

On a single machine

- IPM: slowest (matrix operation)
- SMO: fastest for kernel SVM
- SGD: fastest for linear SVM

Parallel Speed-up

- The communication cost
  - IPM: \( \log(1/\epsilon) \) iter.
  - SMO: bad.
  - SGD: bad, but we improved it!

- The parallel efficiency - Amdahl’s law
  - IPM/SMO: hard to parallel.
  - SGD: easy! Only a distributed inner-product.
**Experimental Results**

Fix $r = 100$

$T=m, T=1.5m, T=2m.$

<table>
<thead>
<tr>
<th>Data set</th>
<th>#samples(train/test)</th>
<th>#features</th>
<th>#processors</th>
<th>SVM-light</th>
<th>PSVM</th>
<th>P-pack 1</th>
<th>P-pack 1.5</th>
<th>P-pack 2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Splice</td>
<td>1,000 / 2,175</td>
<td>60</td>
<td>8</td>
<td>0.3s</td>
<td>0.6s</td>
<td>2s</td>
<td>3s</td>
<td>4s</td>
</tr>
<tr>
<td>Adult</td>
<td>32,561 / 16,281</td>
<td>123</td>
<td>128</td>
<td>1103s</td>
<td>12s</td>
<td>5s</td>
<td>8s</td>
<td>12s</td>
</tr>
<tr>
<td>Web</td>
<td>49,749 / 14,951</td>
<td>300</td>
<td>128</td>
<td>2483s</td>
<td>17s</td>
<td>8s</td>
<td>14s</td>
<td>19s</td>
</tr>
<tr>
<td>CovType</td>
<td>522,910 / 58,102</td>
<td>54</td>
<td>256</td>
<td>280101s</td>
<td>748s</td>
<td>321s</td>
<td>574s</td>
<td>864s</td>
</tr>
<tr>
<td>CCAT</td>
<td>781,265 / 23,149</td>
<td>47,236</td>
<td>256</td>
<td>219744s</td>
<td>18173s</td>
<td>918s</td>
<td>1741s</td>
<td>2739s</td>
</tr>
<tr>
<td>RCV1-All</td>
<td>781,265 / 23,149</td>
<td>47,236</td>
<td>256</td>
<td>3819441s</td>
<td>-</td>
<td>32363s</td>
<td>55323s</td>
<td>79686s</td>
</tr>
<tr>
<td>MNIST8m</td>
<td>8,000,000 / 10,000</td>
<td>784</td>
<td>512</td>
<td>-</td>
<td>-</td>
<td>12880s</td>
<td>41866s</td>
<td>145248s</td>
</tr>
</tbody>
</table>

Fix $r = 100$

$T=m, T=1.5m, T=2m.$

Fix $r = 100$

$T=m, T=1.5m, T=2m.$

Fix $r = 100$

$T=m, T=1.5m, T=2m.$
Experimental Results

Comparisons on the accuracy

- CCAT
- CovType
- Web
- Adult
- Splice

Comparisons on RCV1-All

- F1
- Accuracy
Experimental Results

CovType speed-up

CCAT speed-up
Further Discussion

L2-Logistic Regression?
Yes!

Bias term?
Can be added, but needs an addition term $\frac{1}{2}b^2$ in the regularizer

How to set $T$?
SGD algorithms lack good convergence criteria. However, it can be practically set, like: stop the program when the misclassification rate is numerically stable.
Thanks

Questions:
zhuzeyuan@hotmail.com
wzchen@microsoft.com
Experimental Results

Number of iterations $T$ x 100000

November 13, 2009