Memory-efficient Large-scale Linear SVM

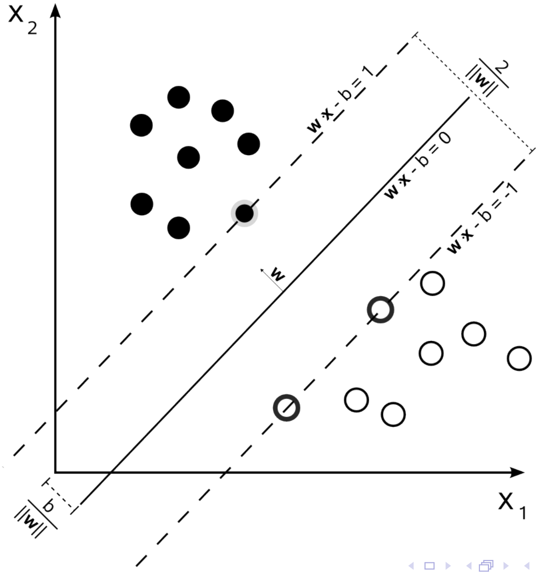
Abdullah Alrajeh, **Akiko Takeda** and **Mahesan Niranjan**

The 7th International Conference on Machine Vision
November 19-21, 2014
Milan, Italy

Outline

Support Vector Machines

SVMs were invented by Boser, Guyon and Vapnik (1992) [figure taken from Wikipedia]



Support Vector Machines

Given a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in R^n$ and $y_i \in \{-1, +1\}$, the maximum margin hyperplane $(\mathbf{w}, b)$ requires solving the following optimization problem:

$$\underset{\mathbf{w}}{\text{minimize}} \quad \frac{1}{2}\mathbf{w}^T\mathbf{w},$$

$$\text{subject to} \quad \mathbf{w}^T\mathbf{x}_i + b \geq +1\,, \text{for } \mathbf{x}_i \text{ of the positive class}$$

$$\mathbf{w}^T\mathbf{x}_i + b \leq -1\,, \text{for } \mathbf{x}_i \text{ of the negative class}$$

OR

$$\text{subject to} \quad y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1\,, \forall i$$

Soft Margin Version

Given a training set $S = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_l, y_l)\}$ where $\mathbf{x}_i \in R^n$ and $y_i \in \{-1, +1\}$, the maximum margin hyperplane $(\mathbf{w}, b)$ requires solving the following optimisation problem:

$$
\begin{aligned}
\underset{\mathbf{w}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{w}^T\mathbf{w} + C\sum_{i=1}^{l}\xi_i, \\
\text{subject to} \quad & y_i(\mathbf{w}^T\mathbf{x}_i + b) \geq 1 - \xi_i \,, \forall i, \\
& \xi_i \geq 0 \,, \forall i,
\end{aligned}
$$

where the slack variable ξ_i measures the margin violation by each data point $\mathbf{x}_i$ and $C \geq 0$ is a penalty parameter for this violation. They were introduced by Cortes and Vapnik (1995).

Dual Form

Introducing Lagrange multipliers $\boldsymbol{\alpha}$ to the primal form, the hyperplane can be found in the dual form as follows:

$$\begin{aligned} \underset{\boldsymbol{\alpha}}{\text{maximize}} \quad & \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j, \\ \text{subject to} \quad & \sum_{i=1}^{l} y_i \alpha_i = 0, \\ & C \geq \alpha_i \geq 0 \,, \forall i, \end{aligned}$$

where the norm of **w** is realised by the second term:

$$\mathbf{w}^T \mathbf{w} = \sum_{i=1}^{l} \sum_{j=1}^{l} \alpha_i \alpha_j y_i y_j \mathbf{x}_i^T \mathbf{x}_j.$$

Discriminative Function

$$f(\mathbf{x}) = \operatorname{sgn}\left(\mathbf{w}^T\mathbf{x} + b\right)$$
$$= \operatorname{sgn}\left(\sum_{i \in SV} \alpha_i y_i \mathbf{x}_i^T \mathbf{x} + b\right)$$

where the bias parameter b is obtained by any support vector $\mathbf{x}_m$ lies on the margin (i.e. $C > \alpha_m > 0$) as follows:

$$b = y_m - \sum_{i \in SV} \alpha_i y_i \mathbf{x}_i^T \mathbf{x}_m$$

Kernel Trick

The **kernel trick** is mapping input space into an inner product space without having to compute the mapping explicitly.

A kernel is defined as follows:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \phi(\mathbf{x}_i)^T \phi(\mathbf{x}_j)$$

Examples:

$$k(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T \mathbf{x}_j \qquad \text{Linear Kernel}$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = (\mathbf{x}_i^T \mathbf{x}_j + c)^d \qquad \text{Polynomial Kernel}$$

$$k(\mathbf{x}_i, \mathbf{x}_j) = \exp\left(-\frac{||\mathbf{x}_i - \mathbf{x}_j||^2}{2\sigma^2}\right) \qquad \text{Gaussian Kernel}$$

Solving SVM - Quadratic Programming

$$\begin{aligned} \underset{\mathbf{x}}{\text{minimize}} \quad & \frac{1}{2}\mathbf{x}^T H\mathbf{x} + \mathbf{f}^T\mathbf{x} \\ \text{subject to} \quad & A\mathbf{x} \le \mathbf{b} \qquad \text{(inequality constraints)} \\ & A_{eq}\mathbf{x} = \mathbf{b}_{eq} \qquad \text{(equality constraints)} \\ & \mathbf{l}_b \le \mathbf{x} \le \mathbf{u}_b \qquad \text{(lower and upper bounds)} \end{aligned}$$

SVM as QP problem:

$$\mathbf{x} = \boldsymbol{\alpha} \quad , \quad H = \mathbf{y}\mathbf{y}^T \circ K(\mathbf{x}_i, \mathbf{x}_j) \quad , \quad \mathbf{f} = -\mathbf{1}$$

$$A_{eq} = \mathbf{y}^T \quad , \quad \mathbf{b}_{eq} = 0 \quad , \quad \mathbf{l}_b = \mathbf{0} \quad , \quad \mathbf{u}_b = C\mathbf{1}$$

QP problem in MATLAB:

```
x = quadprog(H,f,A,b,Aeq,beq,lb,ub,x0)
x = quadprog(H,f,[],[],Aeq,beq,lb,ub,x0)
```

$O(n^3)$ time and $O(n^2)$ space complexities (Tsang et al., 2005, JMLR)

Stochastic Gradient Method

$$\underset{\boldsymbol{\alpha}}{\text{maximize}} \quad W(\boldsymbol{\alpha}) = \sum_i \alpha_i - \frac{1}{2}\sum_i \sum_j \alpha_i \alpha_j y_i y_j k(\mathbf{x}_i^T \mathbf{x}_j),$$

$$\text{subject to} \quad \sum_i y_i \alpha_i = 0,$$

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

The partial derivative is (Friess et al., 1998):

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{l} \alpha_j y_j k(\mathbf{x}_i, \mathbf{x}_j),$$

and the update rule is:

$$\alpha_i \leftarrow \min(C, \max(0, \alpha_i + \eta_i \frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i}))$$

Stochastic Gradient Method

In the case of linear kernel, we can rearrange the gradient as follows:

$$\frac{\partial W(\boldsymbol{\alpha})}{\partial \alpha_i} = 1 - y_i \sum_{j=1}^{l} \alpha_j y_j \, \mathbf{x}_i^T \mathbf{x}_j$$

$$= 1 - y_i \, \mathbf{x}_i^T \sum_{j=1}^{l} \alpha_j y_j \mathbf{x}_j$$

Storing the summation in a vector **w** accelerates linear SVM and can be updated as follows (Hsieh et al., 2008):

$$\begin{aligned}
\mathbf{w}_{\text{new}} &= \alpha_1 y_1 \mathbf{x}_1 + \cdots + \alpha_{\text{new}} y_i \mathbf{x}_i + \cdots + \alpha_l y_l \mathbf{x}_l \\
&= \alpha_1 y_1 \mathbf{x}_1 + \cdots + \alpha_{\text{old}} y_i \mathbf{x}_i + \cdots + \alpha_l y_l \mathbf{x}_l + \alpha_{\text{new}} y_i \mathbf{x}_i - \alpha_{\text{old}} y_i \mathbf{x}_i \\
&= \mathbf{w}_{\text{old}} + (\alpha_{\text{new}} - \alpha_{\text{old}}) y_i \mathbf{x}_i .
\end{aligned}$$

Shrinking Heuristic

Assumption: Dual variables are unlikely to change when become 0 or C.

1. Compute the new α_i for each data point and update $\boldsymbol{\alpha}$.
2. Store in a binary file only data points with $0 < \alpha_i < C$.
3. Read data points from the file until memory is full.
4. Do step 1 and 2.
5. Repeat step 3 and 4 until we reach end of the file.
6. Repeat steps 3-5 until $\boldsymbol{\alpha}$ converge.

Experiments

Table: Fields with brackets ( ) is based on 25% of the data.

Data Set	LLSVM (our method)		LIBLINEAR		Pegasos		SVM^{pref}	
	Time	Accuracy	Time	Accuracy	Time	Accuracy	Time	Accuracy
URL	**2m19s**	**99.54%**	**6m21s**	**99.59%**	1m56s	97.51%	15m4s	99.18%
RCV1	0m22s	97.67%	0m25s	97.78%	0m46s	96.03%	1m9s	97.72%
WEBSPAM	10m44s	99.00%	(2m43s)	(98.30%)	N/A	N/A	N/A	N/A
EPSILON	5m29s	89.75%	(1m48s)	(89.30%)	(4m10s)	(82.91%)	(3m5s)	(89.29%)
MNIST	0m33s	92.07%	0m41s	92.18%	N/A	N/A	N/A	N/A
MNIST8M	174m25s	90.86%	(50m28s)	(88.95%)	N/A	N/A	N/A	N/A
KDD 1999	0m30s	92.29%	0m37s	92.05%	0m41s	86.80%	1m31s	92.07%

Table: Block minimization method (Yu et al., 2012).

Data Set	WEBSPAM	EPSILON
Split	19m12s	14m10s
Train	4m48	3m26
Total Time	24m0s	17m36s
Accuracy	99.04 %	89.75 %

Experiments

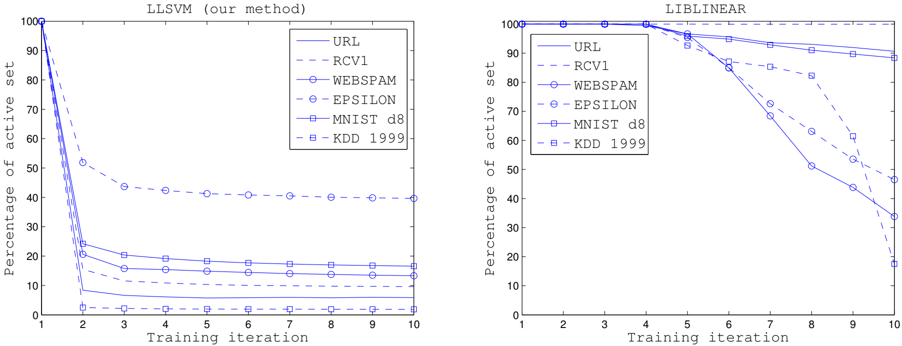


Figure: The active set in our solver and in LIBLINEAR.

Experiments

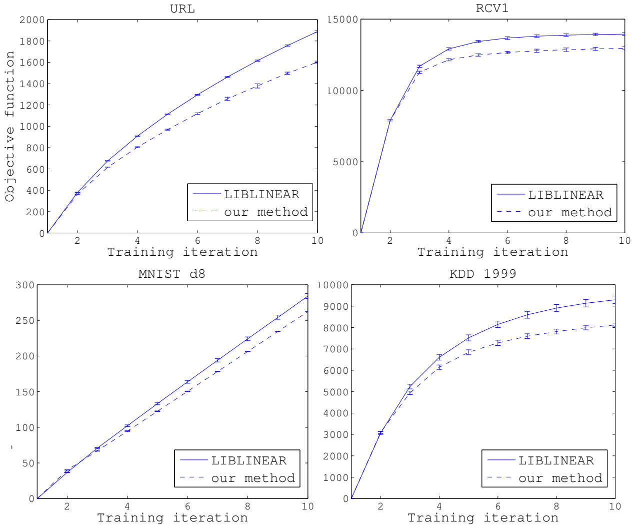


Figure: The objective function in our solver and in LIBLINEAR.

Conclusion

- Our method is comparable to the state-of-the-art solvers in terms of accuracy but faster.
- The shrinking heuristic could reduce up to 99% of some data sets from first iteration.
- The shrinking is very beneficial when data cannot fit in memory. Otherwise, the solver will take long time due to severe disk-swapping