2009 International Conference on Electronic Computer Technology

Training One-class Support Vector Machines in the Primal Space

Lei Wang
School of Economics Information Engineering
Southwestern University of Finance & Economics
Chengdu, 610074, China
wangle_t@swufe.edu.cn

Yong Yang
Suminet Communication Technology(Shanghai) Co., Ltd
Shanghai, 200127, China
dr.wangl@163.com

***Abstract*—In this paper, we proposed a novel Newton-type solver for one-class support vector machines in the primal space directly. Firstly, utilizing reproducing property of kernel and Huber regression function, original constrained quadratic programming is transformed into approximate unconstrained one, which is continuous and twice differentiable. Then, we give a Newton-type training algorithm to solve it. Further analysis shows the computation complexity of our algorithm is identical with theoretical lower bound for solving one-class support vector machines. In the end, experiments on 9 UCI datasets are done to validate the effectivity of proposed algorithm, and when comparing with dual method (LIBSVM), its produces comparative testing accuracy, better training speed, and less support vectors.**

Keywords—one-class support vector machines; primal space; Newton-type algorithm

I. INTRODUCTION

One-class support vector machine (OCSVM) is an unsupervised learning method for density support estimation, which was firstly introduced by Schölkopf [1], and has been successfully applied to numbers of areas, including novelty detection, fault diagnosis, unbalanced classification *etc*.

Like support vector machines, most literature on solving OCSVM concentrates on its dual optimization problem, such as [2][3], which is an linear constrained strictly convex quadratic programming (QP) in reproducing kernel Hilbert space (RKHS). Unfortunately, the computation and storage complexity of QP is cubic and quadratic with problem scale respectively. Although many heuristic strategies (such as working set selection, shrinking, kernel caching) have been used for smoothing this deficiency, training speed is still an unsettled bottleneck of OCSVM, especially in large-scale problem. Therefore, researches on novel approaches (other than dual) for solving OCSVM are quite meaningful, for example, solving it in primal space directly.

Primal optimization of support vector machine (SVM) has received much attention these years. [4] and [5] try to convert the non-differentiable original problem of SVM into an unconstrained and twice differentiable one, by defining a generalized Hessian matrix and using approximate function respectively. But none of their approaches have reduced the computation complexity. Recently, [6] develops a fast solver for linear SVM with L_2 loss function, which is much faster than dual methods, especially in large-scale problem. Wang et al. combines the merits of [5] and [6], propose a Newton-type algorithm for solving SVM with any kernel function in primal space directly, which produces the ε -approximate optimal solution quickly [7]. However, approaches of solving OCSVM in the primal space haven't been carefully researched up to now.

In this paper, we extend the ideas of [7], and put forward a Newton-type training algorithm for training OCSVM in the primal space directly. Analysis shows that the computation complexity is identical with theoretical lower bound for solving OCSVM. And experimental results on 9 UCI datasets prove the effectivity of our methods. As far as know, our algorithm is firstly proposed for solving OCSVM in primal space, and indicates one of potential directions in order to alleviate the deficiency of dual methods.

II. TRAINING OCSVM IN THE PRIMAL SPACE

Given a training set $S=\{\mathbf{x}_1,\mathbf{x}_2,...,\mathbf{x}_n;\ \mathbf{x}_i\in\mathbf{R}^d,\ i=1,...,n\}$, and each of them is independent identically distributed (*i.i.d.*) with unknown probability distribution $P(\mathbf{x})$. Then, the primal optimization problem of OCSVM is usually written as (nonlinear separable)

$$\min_{\mathbf{w},\xi,\rho} \ g(\mathbf{w},\rho)= \frac{1}{2}\mathbf{w}^T\mathbf{w}-\rho+\frac{1}{\nu n}\mathbf{e}^T\xi$$

$$\text{subject to} \quad \Phi(\mathbf{X})^T\cdot\mathbf{w}\ge\rho\cdot\mathbf{e}-\xi,\ \ \xi\ge 0 \qquad (1)$$

Where $\Phi(\mathbf{x}_i)$ is image of $\mathbf{x}_i$ in feature space $\mathbf{R}^N$, and reproducing kernel map $\Phi(\cdot):\mathbf{R}^d\mapsto\mathbf{R}^N$ is determined by kernel function $K(\mathbf{x},\mathbf{y})$ uniquely. With *Lagrange* multiplier $\boldsymbol{\beta}$, we differentiate the *Lagrangian* form of (1) with respect to $\mathbf{w}$, and easily get

$$\mathbf{w}=\sum_{i=1}^{n}\beta_i\cdot\Phi(\mathbf{x}_i) \qquad (2)$$

Obviously, $\mathbf{w}$ belongs to RKHS that determined by $K(\mathbf{x},\mathbf{y})$. According to the reproducing property of kernel, we have $K_{i,j}=K(\mathbf{x}_i,\mathbf{x}_j)=\Phi(\mathbf{x}_i)\cdot\Phi(\mathbf{x}_j)$, and the slack variables in (1) can be rewritten as

$$\xi=\max\{0,\ \rho\mathbf{e}-\mathbf{K}^T\boldsymbol{\beta}\} \qquad (3)$$

978-0-7695-3559-3/09 $25.00 © 2009 IEEE
DOI 10.1109/ICECT.2009.133

where, $\mathbf{K}=\left(K_{i,j}\right)$ is the kernel matrix. Let $C=1/\nu n$, $\mathbf{r}=\rho\mathbf{e}-\mathbf{K}^T\boldsymbol{\beta}$, and consider the fact $\max\{0,t\}=(t+|t|)/2$, we reformulate (1) into following unconstrained QP problem

$$\min_{\boldsymbol{\beta},\rho} \mathrm{g}(\boldsymbol{\beta},\rho)= \frac{1}{2}\boldsymbol{\beta}^T\mathbf{K}\boldsymbol{\beta}-\rho+\frac{C}{2}\mathbf{e}^T\cdot\mathbf{r}+\frac{C}{2}\|\mathbf{r}\|_1 \tag{4}$$

Unfortunately, the L_1 norm is not twice differentiable, we can't solve (4) with gradient methods directly. However, we can replace the L_1-norm by an approximate differentiable form, with help of famous *Huber* regression function [5,8]. Specifically, with $\gamma>0$, the *Huber* function is defined as $\pi_\gamma(t)=\begin{cases}|t|-\frac{\gamma}{2}, & |t|>\gamma\\ \frac{t^2}{2\gamma}, & |t|\le\gamma\end{cases}$. Obviously, $\pi_\gamma(t)$ is Lipschitz continuous and twice differentiable, and satisfies $\lim_{\gamma\to 0}\pi_\gamma(t)=|t|$.

Then, for a given $\gamma>0$, we define support vectors (*SVs*) in primal space as examples with $|r_i|\le\gamma$, outliers with $r_i>\gamma$, and non-support vectors (*NSVs*) with $r_i<\gamma$. Further, we rearrange training examples in an order of *SVs*, outliers, *NSVs*, supposing their numbers are n_{sv}, n_o, $n_{\overline{sv}}$ respectively. By defining diagonal matrix $\mathbf{D}^0=\mathrm{diag}(\underbrace{1,\ldots,1}_{n_{sv}},\underbrace{0,\ldots,0}_{n-n_{sv}})$ and symbol vector $\mathbf{s}=(\underbrace{0,\ldots,0}_{n_{sv}},\underbrace{s_1,s_2,\ldots}_{n-n_{sv}})^T$, $s_i=\mathrm{sign}(r_i)$, we can easily prove that $\lim_{\gamma\to 0}\left\{\frac{\mathbf{r}^T\mathbf{D}^0\mathbf{r}}{2\gamma}+\mathbf{s}^T(\mathbf{r}-\frac{\gamma}{2}\mathbf{s})\right\}=\|\mathbf{r}\|_1$.

Denote $\mathbf{z}=\begin{pmatrix}\boldsymbol{\beta}\\ \rho\end{pmatrix}$, $\mathbf{A}=\begin{pmatrix}\mathbf{K}\\ -\mathbf{e}^T\end{pmatrix}$, $\mathbf{G}=\begin{pmatrix}\mathbf{K} & \mathbf{0}\\ \mathbf{0} & 0\end{pmatrix}$, we have $\mathbf{r}=-\mathbf{A}^T\mathbf{z}$, and an approximate QP problem of (4) can be given as following, which satisfying $\lim_{\gamma\to 0}\mathrm{g}_\gamma(\mathbf{z})=g(\boldsymbol{\beta},\rho)$.

$$\min_{\mathbf{z}} \mathrm{g}_\gamma(\mathbf{z})= \frac{1}{2}\mathbf{z}^T\mathbf{G}\mathbf{z}-\rho+\frac{C}{2}\mathbf{e}^T\mathbf{r}+\frac{C}{2}\left(\frac{1}{2\gamma}\mathbf{r}^T\mathbf{D}^0\mathbf{r}+\mathbf{s}^T(\mathbf{r}-\frac{\gamma}{2}\mathbf{s})\right) \tag{5}$$

Obviously, (5) is twice differentiable, and gradient and *Hessian* of it are

$$\nabla g_\gamma=\frac{\partial g_\gamma(\mathbf{z})}{\partial\mathbf{z}}=\mathbf{G}\mathbf{z}+\begin{pmatrix}\mathbf{0}\\ -1\end{pmatrix}-\frac{C}{2}\mathbf{A}\left(\mathbf{e}+\frac{\mathbf{D}^0\mathbf{r}}{\gamma}+\mathbf{s}\right) \tag{6}$$

$$\mathbf{H}_\gamma=\frac{\partial g_\gamma(\mathbf{z})}{\partial\mathbf{z}\partial\mathbf{z}}=\mathbf{G}+\frac{C}{2\gamma}\mathbf{A}\mathbf{D}^0\mathbf{A}^T \tag{7}$$

For $\forall\,\mathbf{z}\in\mathbf{R}^{n+1}$, we have $\mathbf{z}^T\mathbf{H}_\gamma\mathbf{z}>0$ if $\mathbf{K}$ is positive definite, namely $\mathrm{g}_\gamma(\mathbf{z})$ is strictly convex function. So, Newton-type optimization techniques (or gradient methods) can be used to solve (5) directly. Update rule of $\mathbf{z}$ at each Newton step is

$$\mathbf{z}\leftarrow\mathbf{z}-\mathbf{H}_\gamma^{-1}\cdot\nabla\mathrm{g}_\gamma \tag{8}$$

From (6)(7), we have $\nabla\mathrm{g}_\gamma=\mathbf{H}_\gamma\cdot\mathbf{z}+\begin{pmatrix}\mathbf{0}\\ -1\end{pmatrix}-\frac{C}{2}\mathbf{A}(\mathbf{e}+\mathbf{s})$. Then, let $\tau=\frac{2\gamma}{C}$, $\mathbf{M}=\left(\tau\mathbf{K}+\mathbf{K}\mathbf{D}^0\mathbf{K}\right)^{-1}\mathbf{K}$ and $\phi=\mathbf{e}^T\mathbf{D}^0\mathbf{e}-\mathbf{e}^T\mathbf{D}^0\mathbf{K}\mathbf{M}\mathbf{D}^0\mathbf{e}$, the update of (8) can be rewritten as

$$\begin{aligned}\mathbf{z}&=-\mathbf{H}_\gamma^{-1}\cdot\begin{pmatrix}\mathbf{0}\\ -1\end{pmatrix}+\frac{C}{2}\mathbf{H}_\gamma^{-1}\cdot\mathbf{A}(\mathbf{e}+\mathbf{s})\\ &=\frac{2\gamma}{C}\begin{pmatrix}\tau\mathbf{K}+\mathbf{K}\mathbf{D}^0\mathbf{K} & -\mathbf{K}\mathbf{D}^0\mathbf{e}\\ -\mathbf{e}^T\mathbf{D}^0\mathbf{K} & \mathbf{e}^T\mathbf{D}^0\mathbf{e}\end{pmatrix}^{-1}\left[\frac{C}{2}\mathbf{A}(\mathbf{e}+\mathbf{s})-\begin{pmatrix}\mathbf{0}\\ -1\end{pmatrix}\right]\\ &=\frac{\gamma}{\phi}\begin{pmatrix}\phi\mathbf{M}+\mathbf{M}\mathbf{D}^0\mathbf{e}\mathbf{e}^T\mathbf{D}^0\mathbf{K}\mathbf{M}-\mathbf{M}\mathbf{D}^0\mathbf{e}\mathbf{e}^T\\ \mathbf{e}^T\mathbf{D}^0\mathbf{K}\mathbf{M}-\mathbf{e}^T\end{pmatrix}(\mathbf{e}+\mathbf{s})+\frac{\tau}{\phi}\begin{pmatrix}\mathbf{M}\mathbf{D}^0\mathbf{e}\\ 1\end{pmatrix}\end{aligned} \tag{9}$$

The key to solve (9) is the computation of $\mathbf{M}=\left(\tau\mathbf{K}+\mathbf{K}\mathbf{D}^0\mathbf{K}\right)^{-1}\mathbf{K}=\left(\tau\mathbf{I}_n+\mathbf{D}^0\mathbf{K}^T\right)^{-1}$. Consider the special structure of matrix $\tau\mathbf{I}_n+\mathbf{D}^0\mathbf{K}^T$, which can be written as $\begin{pmatrix}(\tau\mathbf{I}_{sv}+\mathbf{K}_{sv}) & \mathbf{K}_{sv,o+\overline{sv}}\\ \mathbf{0} & \tau\mathbf{I}_{o+\overline{sv}}\end{pmatrix}$ (kernel matrix $\mathbf{K}$ has been rearranged also, and subscripts $sv,o,\overline{sv}$ represent *SVs*, outliers, *NSVs* respectively), we can easily solve $\mathbf{M}$ as

$$\mathbf{M}=\begin{pmatrix}\mathbf{M}_{11} & \mathbf{M}_{12}\\ \mathbf{M}_{21} & \mathbf{M}_{22}\end{pmatrix}=\begin{pmatrix}(\tau\mathbf{I}_{sv}+\mathbf{K}_{sv})^{-1} & -\frac{1}{\tau}(\tau\mathbf{I}_{sv}+\mathbf{K}_{sv})^{-1}\mathbf{K}_{sv,o+\overline{sv}}\\ \mathbf{0} & \frac{1}{\tau}\mathbf{I}_{o+\overline{sv}}\end{pmatrix} \tag{10}$$

Let $\boldsymbol{\Theta}=(\tau\mathbf{I}_{sv}+\mathbf{K}_{sv})^{-1}$, and put (10) into (9), we can rewrite (9) as

$$\mathbf{z}=\frac{1}{\phi}\begin{pmatrix}\gamma\mathbf{q}_1\mathbf{e}_{sv}+2\gamma\mathbf{q}_2\mathbf{e}_o+\frac{2\gamma}{C}\boldsymbol{\Theta}\mathbf{e}_{sv}\\ C\phi\cdot\mathbf{e}_o\\ \mathbf{0}\\ \gamma\mathbf{q}_3\mathbf{e}_{sv}+2\gamma\mathbf{q}_4\mathbf{e}_o+\frac{2\gamma}{C}\end{pmatrix}=\begin{pmatrix}\boldsymbol{\beta}_{sv}\\ \boldsymbol{\beta}_o\\ \boldsymbol{\beta}_{\overline{sv}}\\ \rho\end{pmatrix} \tag{11}$$

where $\begin{cases}\mathbf{q}_1=\phi\boldsymbol{\Theta}+\boldsymbol{\Theta}\mathbf{e}_{sv}\mathbf{e}_{sv}^T\mathbf{K}_{sv}\boldsymbol{\Theta}-\boldsymbol{\Theta}\mathbf{e}_{sv}\mathbf{e}_{sv}^T\\ \mathbf{q}_2=-\frac{\phi}{\tau}\boldsymbol{\Theta}\mathbf{K}_{sv,o}-\frac{1}{\tau}\boldsymbol{\Theta}\mathbf{e}_{sv}\mathbf{e}_{sv}^T(\mathbf{K}_{sv}\boldsymbol{\Theta}\mathbf{K}_{sv,o}-\mathbf{K}_{sv,o}\mathbf{I}_o)-\boldsymbol{\Theta}\mathbf{e}_{sv}\mathbf{e}_o^T\\ \mathbf{q}_3=\mathbf{e}_{sv}^T\mathbf{K}_{sv}\boldsymbol{\Theta}-\mathbf{e}_{sv}^T\\ \mathbf{q}_4=\frac{1}{\tau}\mathbf{e}_{sv}^T(-\mathbf{K}_{sv}\boldsymbol{\Theta}\mathbf{K}_{sv,o}+\mathbf{K}_{sv,o}\mathbf{I}_o)-\mathbf{e}_o^T\end{cases}$.

So, if we can know support vectors and outliers in advance, the optimal solution of (5) can be computed immediately according to (11). Noticeably, up to now, all of our computations are in primal space only.

III. ALGORITHM AND COMPLEXITY ANALYSIS

As above analysis, we present a Newton-type algorithm for solving (5), which is called "**Primal Newton for OCSVM**" algorithm (or **PN-OCSVM**) in this paper.

Algorithm 1. PN-OCSVM algorithm

Input: A collection of training set S; kernel function $K(\mathbf{x},\mathbf{y})$ and corresponding kernel matrix $\mathbf{K}$; penalty parameter λ; small parameter $\gamma>0$ and $\varepsilon>0$;

(a) Select a subset S_0 from S randomly with size $m\,(m<n)$. Deem examples in S_0 as initial *SVs* (the rest are *NSVs*), and rearrange $\mathbf{K}$. Then, let $k=0$;

(b) $k=k+1$, update $\mathbf{z}_k$ according to (11), and calculate $\mathbf{r}_k=-\mathbf{A}^T\mathbf{z}_k$;

(c) Identify the *SVs* and outliers according to the value of $\mathbf{r}_k$, and rearrange kernel matrix $\mathbf{K}$;

(d) Calculate ∇g_γ^k according to (6). Then, if it satisfies $\left\|\nabla g_\gamma^k\right\|\le\varepsilon$, algorithm stops; otherwise, return to (b).

For positive definite of *Hessian*, we say that PN-OCSVM algorithm will converge in finite steps according to Newton convergence theorem , and the number of Newton steps is independent of problem scale. So, we can evaluate PN-OCSVM's computation complexity by that of one Newton step.

Analyze equation (11), computation complexity of calculation matrix $\mathbf{\Theta}$ is $O(n_{sv}^3)$, and that of calculation $\mathbf{q}_1,\mathbf{q}_2,\mathbf{q}_3,\mathbf{q}_4$ is at most $O(n_{sv}^2\times n_o)$ in step (b); Complexity of calculation $\mathbf{r}_k$ and ∇g_γ^k in step (b)(d) are all $O(n(n_{sv}+n_o))$; And that of rearranging matrix $\mathbf{K}$ is at most $O(n(n_{sv}+n_o))$ in step (c). Therefore, computation complexity of PN-OCSVM algorithm is $O(n(n_{sv}+n_o)+n_{sv}^3+n_{sv}^2\times n_o)$.

Noticeablely, the complexity becomes $O(n(n_{sv}+n_o)+n_{sv}^3)$ if there are few outliers in training examples [9], and it's exactly same with the theoretical lower bound of any algorithm for solving optimization (1). That is to say, PN-OCSVM algorithm has fast training speed theoretically when solving OCSVM in the primal space.

IV. Experimental and Results Analysis

Total 9 standard datasets drawn from the UCI machine learning repository are used in experiments [10], and the numbers of training and testing examples are listed in Table 1. These datasets are chosen with strong representation, including different scale, noise (diabetes, waveform, adult), and unbalanced classes (iris, DNA, waveform, for converting from multi-classification to bi-classification). Besides, the PN-OCSVM algorithm is implemented in Matlab 6.5, and runs on a computer with CPU P4 2.66MHz and memory 512Mb.

For examining the effectivity of PN-OCSVM algorithm when training in the primal space, we compare three indices (testing accuracy, training time and number of support vectors) of it on UCI datasets with that of *LIBSVM*, which is one of the most popular training toolbox for OCSVM up to now. Table 1 gives the experimental results (values are averaged 10 runs independently). In these experiments, each algorithm adopts RBF kernel function, and parameters are set with $\nu=0.05$ and $\gamma=0.01$.

From the statistical results of Table 1, the testing accuracy of PN-OCSVM algorithm is very close to *LIBSVM* in each case. The two approaches gain best accuracy 3/7 times respectively, while the difference is at most 0.74% (chess dataset). As to training time, they obtain the shortest training time 4/5 times respectively. Consider the fact that PN-OCSVM's implementation doesn't adopt any optimization while *LIBSVM* takes many heuristic strategies, we say that the training time of PN-OCSVM algorithm is no longer or better than that of *LIBSVM*. In addition, PN-OCSVM obtains less support vectors than *LIBSVM*, and support vectors of the former must be subset of the latter.

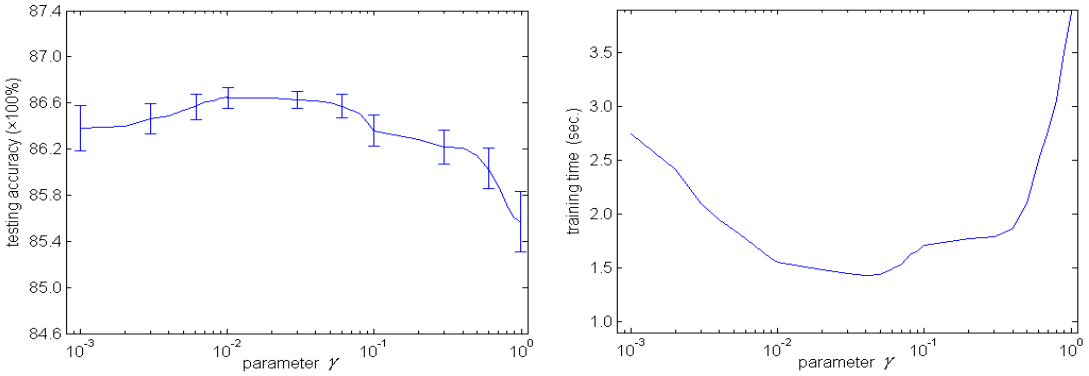
Table 1 Results of PN-OCSVM Algorithm on UCI Datasets and Comparison with LIBSVM

Datasets	# training/testing examples	PN-OCSVM Algorithm			LIBSVM		
		accuracy (×100%)	*training time (sec.)*	*# SVs*	*accuracy (×100%)*	*training time (sec.)*	*# SVs*
Iris	30/30	90.67	**0.01**	2	**95.33**	—	3
(**4D**)	30/30	**94.00**	0.01	3	**94.00**	—	3
	30/30	91.33	0.02	3	**93.33**	—	4
Diabetes	300/227	**71.57**	**0.18**	65	69.42	0.31	107
(**8D**)	161/157	73.06	**0.10**	43	**73.88**	0.23	74
Chess	1,001/821	93.44	**0.74**	97	**95.05**	1.06	183
(**36D**)	916/778	93.57	**0.62**	102	**94.15**	0.85	155
Waveform	994/997	86.60	**1.49**	156	**87.27**	2.83	279
(**21D**)	988 /994	**87.06**	**1.98**	144	86.44	2.91	250
	1,017/1,008	87.19	**1.85**	142	**87.83**	2.55	292
Mushroom	2,093/1,611	95.47	0.95	64	**96.21**	**0.41**	93
(**22D**)	1,294/1,211	**96.55**	0.71	62	**96.55**	**0.32**	87

* training set = 60% of examples belong to the class in a dataset
testing set = 40% of examples belong to the class (the rest) + 10% of examples other than the class in a dataset

Thus, when analyzing results of testing accuracy and number of support vectors, we can judge that PN-OCSVM yields an approximate optimal solution of OCSVM, and the generated separating hyperplane is more smooth (for less support vectors). Besides, from the results on diabetes and adult datasets, we observe that LIBSVM is more sensitive to noise data, and the trained separating hyperplane is very complex (the ratio of support vectors is 46.6% and 34.3% respectively). Oppositely, PN-OCSVM is less sensitive to noise, and the testing accuracy is even a little higher than LIBSVM.

Seen from Figure 1, parameter γ doesn't have significant impact on testing accuracy while influences greatly on training time. Specifically, when γ takes a small value, PN-OCSVM needs more Newton steps to reach to the approximate solution, and the training time becomes long; When γ takes a large value, the number of unbounded support vectors increases, the training time grows rapidly.



(a) (b)

Figure 1 Influences of parameters of PN-OCSVM algorithm to testing error and training time (waveform dataset, Class I) (a) testing accuracy (b) training time γ=0.01

V. CONCLUSION

This paper mainly proposes a new approach for training OCSVM in primal space directly. With the help of reproducing property of kernel and Huber regression function, we transform original OCSVM optimization into an unconstrained QP problem, which is convex and twice differentiable. Then, we give a Newton-type algorithm to solve it. Theoretical analysis shows that the algorithm produces an approximate solution of OCSVM, while complexity is almost identical with low-bound of it. Finally, we make several experiments on 9 UCI datasets, and compare the performance between PN-OCSVM and LIBSVM. Preliminary results show that the testing accuracy of PN-OCSVM is very close to LIBSVM, and training time is no longer than it, but the ratio of support vectors is much less.

ACKNOWLEDGEMENT

This work was supported by the Scientific Research of Southwestern University of Finance and Economics (QN0806).

REFERENCES

[1] B. SchÖlkopf, J. Platt, et al, "Estimating the support of high-dimensional distribution", Neural Comput., Vol.13, pp.1443-1471, 2001.

[2] D.M.J. Tax, R.P.W. Duin, "Support vector data description", Mach. Learn., Vol.54, pp. 45-66, 2004.

[3] C.C. Chang, C.J. Lin, LIBSVM: a library for support vector machines. 2001. Software available at http://www.csie.ntu.edu.tw/~cjlin/libsvm

[4] O.L. Mangasarian, "A finite Newton method for classification", Optim. Methods and Software, Vol.17, pp. 913-929, 2002.

[5] S.S. Zhou, H.S. Zhan, L.H. Zhou, "A Huber approximation method for training the support vector machines", Chinese J. of Comput. Vol. 28, pp.1664-1670, 2005. (in Chinese).

[6] S.S. Keerthi, D. DeCoste, "A modified finite Newton method for fast solution of large scale linear SVMs", J. of Mach. Learn. Res. Vol.6, pp.341-361, 2005.

[7] L. Wang, S.X. Sun, K.Zhang, "A fast approximate algorithm for training L1-SVMs in primal space", Neurocomputing, Vol. 70, pp1554-1560, 2007.

[8] K. Madsen, H.B. Nielsen, M.C. Pinar, "Bound constrained quadratic programming via piecewise quadratic functions", Math. Program., Vol. 85, pp.135-156, 1999.

[9] A. Bordes, S. Ertekin, J. Weston, et al., "Fast kernel classifiers with online and active learning". J. of Mach. Learn. Res., Vol. 6, pp.1579-1619, 2005.

[10] D.J. Newman, S. Hettich, et al., "UCI Repository of machine learning databases", [http://www.ics.uci.edu/ ~mlearn/MLRepository.html], 1998.