Storage Fit Learning with Unlabeled Data*

Bo-Jian Hou Lijun Zhang Zhi-Hua Zhou
National Key Laboratory for Novel Software Technology
Nanjing University, Nanjing 210023, China
{houbj, zhanglj, zhouzh}@lamda.nju.edu.cn

Abstract

By using abundant unlabeled data, semi-supervised learning approaches have been found useful in various tasks. Existing approaches, however, neglect the fact that the storage available for the learning process is different under different situations, and thus, the learning approaches should be flexible subject to the storage budget limit. In this paper, we focus on graph-based semi-supervised learning and propose two *storage fit learning* approaches which can adjust their behaviors to different storage budgets. Specifically, we utilize techniques of low-rank matrix approximation to find a low-rank approximator of the similarity matrix to meet the storage budget. The first approach is based on stochastic optimization, which is an iterative approach that converges to the optimal low-rank approximator globally. The second approach is based on Nyström method, which can find a good low-rank approximator efficiently and is suitable for real-time applications. Experiments show that the proposed methods can fit adaptively different storage budgets and obtain good performances in different scenarios.

1 Introduction

During the past decade, smart mobile devices such as mobile phones are becoming increasingly popular. Mobile devices can easily generate an enormous amount of data, such as photos, texts, videos, etc. However, although most users prefer neatly arranged data, few users bother to label these data, for example, assigning labels to photos they took.

This problem naturally corresponds to semi-supervised learning where few photos labeled by users represent labeled data while large amounts of unlabeled photos can be regarded as unlabeled data. Semi-supervised learning utilizes these additional unlabeled data to enhance classification. Among many semi-supervised learning approaches, graph-based semi-supervised learning is one of the most important semi-supervised learning paradigms [Zhu, 2007; Liu *et al.*, 2012]. Graph-based semi-supervised methods define a graph where the nodes are all instances either labeled or unlabeled, and edges reflect the similarity of examples [Zhu *et al.*, 2005]. In graph-based methods, a large kernel matrix of size $n \times n$ will be calculated, where n is the number of instances, resulting in ineffectiveness in both computation and storage. During the past decade, various methods from different perspectives have been proposed in graph-based semi-supervised learning, including Mincut [Blum and Chawla, 2001], Gaussian Random Fields and Harmonic Functions [Zhu *et al.*, 2003; Grady and Funka-Lea, 2004; Levin *et al.*, 2004], Local and Global Consistency [Zhou *et al.*, 2003], Tikhonov Regularization [Belkin *et al.*, 2004], Graph Kernels [Chapelle *et al.*, 2002; Zhu *et al.*, 2004], Spectral Graph Transducer [Joachims, 2003], Tree-Based Bayes [Kemp *et al.*, 2003], etc.

There are some graph-based semi-supervised methods dealing with the large kernel matrix, however, previous studies on graph-based semi-supervised learning almost neglect the fact that the storage budget that can be used is different on different devices. Back to the photos labeling example, since the mobile phones have various limited memory, such as 500MB, 1000MB or 2000MB, we cannot exploit all the unlabeled photos to do semi-supervised learning, and the best to do is to fully exploit the storage budget rather than fully exploiting the unlabeled data. In this paper, we propose to study *storage fit learning*, that is, the learning process is designed to fully exploit a storage budget limit. For example, assume that a storage of $10^6 \times 10^6$ matrix is required for a concerned semi-supervised learning algorithms $Algo$ to exploit all available unlabeled data, yet the memory storage available is only able to accommodate a $10^3 \times 10^3$ matrix. Ideally, effective storage fit learning algorithms should be able to adjust their behaviors considering the given storage budgets. This problem was firstly studied by Zhou *et al.* [2009], where fully-connected affinity graph for *cluster kernel* [Chapelle *et al.*, 2002] was replaced by k-nearest neighbor graph for approximation. When k is small, abundant unlabeled data can be exploited, but the approximation would be poor; when k is large, the approximation can be good, but the computational load is too high to handle large-scale data.

In this paper, we will make the cluster kernel method adaptively fit the storage budget. Our basic ideas can also be generalized to other approaches relying on spectral analysis which suffer seriously from storage limitation. Inspired by

*This work was supported by the NSFC (61673201, 61603177), JiangsuSF (BK20160658), and the Collaborative Innovation Center of Novel Software Technology and Industrialization.

techniques of low-rank matrix approximation, we propose two simple but effective methods to solve the problem of storage fit learning. One is based on stochastic optimization and the other on Nyström. The stochastic optimization method can converge to the optimal low-rank approximator iteratively and the Nyström-based method can find a good low-rank approximator efficiently. Unlike most graph-based semi-supervised learning methods, the two proposed methods can adaptively adjust their memory requirements to fit different storage budgets as well as obtain good performances.

2 Background

Cluster kernel (abbreviated as ClusK) [Chapelle *et al.*, 2002; Zhou *et al.*, 2009] is a classic technique that has been popularly used in graph-based approaches. The details of cluster kernel method are given as follows.

Given labeled data $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_l, y_l)\}$ and unlabeled data$\{\boldsymbol{x}_{l+1}, \cdots, \boldsymbol{x}_{l+u}\}, l \ll u, n = l + u$ where $\boldsymbol{x} \in \mathbb{R}^d$ and $y \in \{-1, 1\}$. We can construct a kernel matrix K where nearby data points are assigned with relatively large edge weights. The procedure of cluster kernel algorithm is presented as follows:

1. Let D denote the diagonal matrix whose elements are $D_{ii} = \Sigma_j K_{ij}$, and construct the matrix $L = D^{-1/2} K D^{-1/2}$.
2. Compute the eigendecomposition $L = U \Sigma U^\top$, and assume that the eigenvalues are ordered as $\sigma_1 \geq \cdots \geq \sigma_n$.
3. Apply a transfer function φ on σ. Let $\tilde{\sigma}_i = \varphi(\sigma_i)$, and construct $\tilde{L} = U \tilde{\Sigma} U^\top$.
4. Let $\tilde{D}$ denote the diagonal matrix whose elements are $\tilde{D}_{ii} = 1/\tilde{L}_{ii}$, and compute $\tilde{K} = \tilde{D}^{1/2} \tilde{L} \tilde{D}^{1/2}$.

The transfer function φ can take different forms. Here, the *poly-step* transfer function which has achieved the best performance in [Chapelle *et al.*, 2002] is adopted:

$$\tilde{\sigma}_i = \begin{cases} \sqrt{\sigma_i} & i < h(= l + 9) \\ \sigma_i^2 & i \geq h(= l + 9). \end{cases} \tag{1}$$

The kernel $\tilde{K}$ obtained from cluster kernel method then could be used to do classification by kernel SVM. One problem of this method is that it has a storage requirement up to $O(n^2)$. It cannot work for those situations where we have limited resources which cannot meet the requirement, such as doing image labeling [Sánchez *et al.*, 2013] on smart phones. Another problem we would face is that mobile devices have different storage sizes. The effect of storage increasing is weakened since the number of instances used to calculate the kernel only increases by the square root of increased storage.

For the first problem, several large scale graph-based semi-supervised methods try to tackle it. Some of them focus on reducing time complexity. For example, Pfahringer *et al.* [2007] compute the large kernel matrix inverse; Fowlkes *et al.* [2004] calculate only the dominate eigenvalues; Kumar *et al.* [2009] and Zhang and Kwok [2010] use kernel approximation in manifold learning. There are some other methods reducing space complexity including the fixed-sized least-squares support vector machines which try to reduce space complexity by approximating kernels [Brabanter *et al.*, 2010; Jumutc *et al.*, 2013]. However, these methods focus on how to solve the large scale problem while we concentrate on the *fitting* problem which means even though the data is in regular scale, we cannot compute the full kernel matrix since the storage is limited. Moreover, due to different storage budgets, we need to adjust our methods to fit different scenarios. There are some other related works relying on spectral analysis which try to tackle the large-scale problem in clustering, such as [Li *et al.*, 2016; Han *et al.*, 2016]. However, they do not consider the situation where a storage budget is given and the methods should dynamically fit it. We notice that the key storage cost of cluster kernel method is owing to storing matrix K and L. Additionally, the main time cost is due to eigendecomposing L. Thus if we can efficiently obtain the eigensystem of K or L without storing them, we would have room to fit our methods for different storage budgets. In this paper, to solve the storage fit problem, we utilize low-rank matrix approximation techniques to find low-rank approximator of K or L and thus, we can use the top eigensystem to get surrogate "virtual" samples which can be used in linear SVM. By adjusting parameters according to different budgets, we can exploit all the unlabeled data and obtain good performances in different scenarios.

3 Our Proposed Approaches

In this section, we first describe the basic idea of how to perform kernel SVM to do classification without storing the whole $n \times n$ kernel. Then the specific methods which can adjust their bahaviors to defferent storage budgets are presented in accompany with complexity analysis and how to dynamically adjust parameters according to different budgets.

Originally, we calculate $\tilde{K}$ according to the steps mentioned in Section 2 to perform kernel SVM. But these steps cost nearly $O(n^3)$ time [Pan and Chen, 1999] and $O(n^2)$ space which cannot work in our setting. We don't need to compute $\tilde{K}$, instead, we can transform a kernel SVM into a linear SVM by decomposing kernel matrix on the training and testing data [Zhang *et al.*, 2012] which is showed in Proposition 1.

Proposition 1 *Given training data X_r and label $\boldsymbol{y}_r$, and test data X_e. A kernel SVM trained on X_r, $\boldsymbol{y}_r$, and tested on X_e is equivalent to a linear SVM trained on F_r, $\boldsymbol{y}_r$ and tested on F_e, where*

$$K = \begin{bmatrix} F_r \\ F_e \end{bmatrix} [F_r^\top \quad F_e^\top] \tag{2}$$

is any decomposition of the PSD kernel matrix K evaluated on (X_r,X_e), and the factor $F_r \in \mathbb{R}^{n \times p}$ and $F_e \in \mathbb{R}^{m \times p}$ can be deemed as "virtual samples" whose p is the rank of K.

Proposition 1 shows that any kernel SVM can be cast as an equivalent linear SVM by decomposing of the kernel matrix $K = FF^\top$, where F serves as an empirical kernel map or *virtual samples*. The positive semi-definiteness of the kernel matrix guarantees that decomposition (2) always exists.

In our setting, we would like to decompose $\tilde{K}$ to obtain the virtual examples without storing $\tilde{K}$. The basic idea is to exploit the top *eigensystem* to compute the virtual examples directly. Specifically, note that $\tilde{K} = \tilde{D}^{1/2} \tilde{L} \tilde{D}^{1/2}$, so that the

virtual samples can be represented as $F = \tilde{K}^{1/2} = \tilde{D}^{1/2}\tilde{L}^{1/2}$. Now, the task is transformed to compute $\tilde{D}^{1/2}$ and $\tilde{L}^{1/2}$. Assume that we already have the top-k eigensystem of K, namely, (U_k, Σ_k) where Σ_k is a diagonal matrix whose diagonal elements are the first k eigenvalues of K and U_k consists of the k corresponding eigenvectors. So we can approximate K by $K \approx U_k\Sigma_k(U_k)^\top$. According to Section 2, we have

$$L = D^{-1/2}KD^{-1/2} \approx D^{-1/2}U_k\Sigma_kU_k^\top D^{-1/2}. \quad (3)$$

Then apply the transfer function φ on every diagonal elements of Σ_k, we have

$$\tilde{L} \approx D^{-1/2}U_k\tilde{\Sigma}_kU_k^\top D^{-1/2}. \quad (4)$$

Therefore we can verify that

$$\tilde{L}^{1/2} \approx D^{-1/2}U_k(\tilde{\Sigma}_k)^{1/2} \quad \text{and} \quad \tilde{D}_{ii} = 1/\tilde{L}_{ii}. \quad (5)$$

Finally, the virtual samples are formed as

$$F \approx \tilde{D}^{1/2}\tilde{L}^{1/2}. \quad (6)$$

According to Proposition 1, we can directly perform linear SVM on F. Based on this idea, what we should do in the following is to obtain the *eigensystem* of K or equally L. We adopt two different kinds of methods to achieve this goal. One is an iterative approach which uses partial information of L in each round to estimate the top eigenvectors, and the other is an approximating approach which utilizes Nyström to directly approximate the original kernel matrix by sampling. Note that Mehrkanoon and Suykens [2014] also use Nyström approximation of feature map in semi-supervised scenario, but they only considers the *large scale* setting while we focus on the *fitting* problem where we adjust our methods to be fit for different storage budgets.

3.1 The SoCK Method

In this section, we propose an iterative approach: *Stochastic optimization for Cluster Kernel* (SoCK). Specifically, we use partial information of matrix L in each round to estimate the top eigenvectors. With appropriate initialization, the solution to SoCK will converge to the optimal low-rank approximator globally [Zhang *et al.*, 2016]. After obtaining virtual samples, through adjusting the corresponding parameters, we are able to adaptively fit SoCK for different storage budgets.

Denote by $U\Sigma U^\top$ the eigendecomposition of the matrix $L \in \mathbb{R}^{n\times n}$ mentioned in Section 2, where $U = [\boldsymbol{u}_1, \cdots, \boldsymbol{u}_n]$, $\Sigma = \mathrm{diag}[\sigma_1, \cdots, \sigma_n]$ and $\sigma_1 \geq \sigma_2 \geq \cdots \geq \sigma_n$. To obtain the top eigensystem of L, we need to find a low-rank matrix $\hat{L}$ to approximate L. We consider the Singular Value Thresholding (SVT) operator [Cai *et al.*, 2010] to L with threshold λ to obtain the low-rank matrix $\hat{L}$, i.e.,

$$\hat{L} = \mathcal{D}_\lambda[L] = \sum_{i:\sigma_i\geq\lambda}(\sigma_i - \lambda)\boldsymbol{u_i}\boldsymbol{u}_i^\top. \quad (7)$$

From (7), we can recover the top eigensystem of L (with eigenvalues larger than λ) from the eigendecomposition of $\hat{L}$ since the eigenvectors of $\hat{L}$ with nonzero eigenvalues are the top eigenvectors of L and nonzero eigenvalues of $\hat{L}$ are the top eigenvalues of L minus λ. Then SVT operation can be formulated as a Stochastic Composite Optimization (SCO) problem. First, we propose the following convex composite optimization problem

$$\min_{Z\in\mathbb{R}^{n\times n}} \frac{1}{2}\|Z - L\|_F^2 + \lambda\|Z\|_* \quad (8)$$

where $\|\cdot\|_*$ is the nuclear norm of matrices and note that $\hat{L}$ is the optimal solution to it. We can generate a low-rank random matrix ξ which is an unbiased estimate of K by the random Fourier features [Rahimi and Recht, 2007], then $L_\xi = D^{-1/2}\xi D^{-1/2}$ is an unbiased estimate of L, i.e., $L = E[L_\xi]$. Then (8) is equivalent to

$$\min_{Z\in\mathbb{R}^{n\times n}} \frac{1}{2}E[\|Z - L_\xi\|_F^2] + \lambda\|Z\|_*. \quad (9)$$

Due to the high space complexity of generic algorithms for stochastic optimization, we utilize an algorithm which is based on Stochastic Proximal Gradient Descent (SPGD) where the power of the non-smooth term is preserved to optimize (9) and take its last iteration as the final solution [Zhang *et al.*, 2016]. Denote by Z_t the solution at the t-th iteration. In this iteration, we first sample a random matrix $\xi_t \in \mathbb{R}^{n\times n}$. Let $L_t = D^{-1/2}\xi_tD^{-1/2}$ and it is easy to verify that $Z_t - L_t$ is an unbiased estimate of the gradient of $\frac{1}{2}E\left[\|Z - L_\xi\|_F^2\right]$. Then, we update the current solution by the SPGD, which is essentially a stochastic variant of composite gradient mapping [Nesterov, 2013]

$$\begin{aligned} Z_{t+1} &= \operatorname*{argmin}_{Z\in\mathbb{R}^{n\times n}} \frac{1}{2}\|Z - Z_t\|_F^2 \\ &\quad + \eta_t\langle Z - Z_t, Z_t - L_t\rangle + \eta_t\lambda\|Z\|_* \\ &= \operatorname*{argmin}_{Z\in\mathbb{R}^{n\times n}} \frac{1}{2}\|Z - [(1-\eta_t)Z_t + \eta_tL_t]\,\|_F^2 + \eta_t\lambda\|Z\|_* \\ &= \mathcal{D}_{\eta_t\lambda}\left[(1-\eta_t)Z_t + \eta_tL_t\right] \end{aligned} \quad (10)$$

where $\eta_t > 0$ is the step size. Let Z_{T+1} be the final solution obtained after T iterations. We eigendecompose Z_{T+1} and obtain its eigensystem $\{(\boldsymbol{u}_i, \sigma_i)\}_{i=1}^k$ with nonzero eigenvalues. Finally, we return $\{(\boldsymbol{u}_i, \sigma_i + \lambda)\}_{i=1}^k$ as the top eigensystem

Algorithm 1 SoCK

Input: labeled data $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_l, y_l)\}$,
unlabeled data $\{\boldsymbol{x}_{l+1}, \cdots, \boldsymbol{x}_{l+u}\}$, $l \ll u$, $n = l + u$;
The number of trials T, the regularization parameter λ;
Output: Predicted label $\boldsymbol{y} = [y_{l+1}, \cdots, y_{l+u}]^\top$ of unlabeled data.

```
Initialize Z_1 = 0 and calculate D;
for t = 1, 2, ..., T do
    Sample a random matrix ξ_t;
    let L_t = D^{-1/2} ξ_t D^{-1/2}, η_t = 2/t;
    Z_{t+1} = D_{η_t λ}[(1 − η_t) Z_t + η_t L_t];
end for
Compute eigensystem of Z_{T+1} : (U_k, Σ̂_k) or {(u_i, σ̂_i)}_{i=1}^k;
Let σ_i = σ̂_i + λ and ordered as σ_1 ≥ σ_2 ≥ ··· ≥ σ_k;
Let σ̃_i = φ(σ_i) to get diagonal matrix Σ̃_k;
L̃^{1/2} = U_k Σ̃_k^{1/2}, D̃_ii = 1/L̃_ii, F = D̃^{1/2} L̃^{1/2};
Regard every row of F as a new instance to perform linear SVM.
```

of L. After getting the eigensystem of matrix L, we can easily obtain the approximation of virtual samples F by following (4), (5) and (6). We conclude the procedure in Algorithm 1.

According to [Zhang *et al.*, 2016], we have the following theorem which shows that with a high probability, Z_{T+1} converges to $\hat{L}$, the optimal solution to (9), at an $O(1/T)$ rate.

Theorem 1 *Assume the Frobenius norm of the matrix $L_\xi = D^{-1/2}\xi D^{-1/2}$ is upper bounded by some constant $C > 0$, i.e., $\|L_\xi\|_F \leq C$. With a probability at least $1-\delta$, we have*

$$\begin{aligned}&\|Z_{T+1} - \hat{L}\|_F^2 \\ &\leq \frac{8}{T}\left[C\lambda \max_{t\in[T]} \sqrt{r_t} + C^2\left(8 + 6\log\frac{\lceil 2\log_2 T\rceil}{\delta}\right)\right] \\ &= O\left(\left(\log\log T\right)/T\right)\end{aligned}$$

where r_t is the rank of Z_t.

From the above theorem we observe that with the number of iterations increasing, the approximation error $\|\hat{L} - L\|_F/n$ will decrease. We will show that the classification performance is positively correlated with the approximation error in subsection 4.3. To discuss the space and time complexity, we have to mention the implementation issues. First, the random matrix L_t can always be represented by $L_t = \zeta_t \chi_t^\top$, where $\zeta_t, \chi_t \in \mathbb{R}^{n\times a_t}$ are two rectangular matrices with $a_t \ll n$. Now, suppose that Z_t is also represented by $Z_t = U_t V_t^\top$, where $U_t, V_t \in \mathbb{R}^{n\times b_t}$ are two rectangular matrices with $b_t \ll n$. Then, $Z_{t+1} = \mathcal{D}_{\eta_t\lambda}[(1-\eta_t)U_tV_t^\top + \eta_t\zeta_t\chi_t^\top]$ can be solved efficiently according to Lemma 2.4 of [Avron *et al.*, 2012]. From the above discussion, we know that the space complexity is $O(n(d+a_t+b_t))$. The running time from step 1 to step 7 in Algorithm 1 is dominated by computing D which takes $O(n^2d)$ time, calculating ξ_t, which takes $O(nda_t)$ time and eigendecomposition, which takes $O(n(a_t+b_t)^2)$ time. Other steps either take $O(nk)$ or $O(nk^2)$ time where k is the number of nonzero eigenvalues. As a result, the time complexity is $O(n[nd + da_t + (a_t+b_t)^2 + k^2])$.

We study the relationship between the storage budget and the value of parameters. Suppose each double float costs 8 bytes in storage which is popular in current machines. Given n examples with dimensionality d, the storage required for these examples is $dn \times 8$ bytes. After applying SoCK, we need to store the matrix U and F where they both contain $n(a_t+b_t)$ elements. So the storage for U and F is $2n(a_t+b_t)\times 8$ bytes. Other matrices such as D only have $n\times 1$ elements so that they can be omitted. Overall, the storage required by the SoCK method for exploiting all the labeled and unlabeled examples is $(d+2(a_t+b_t))n\times 8$ bytes. Since d and n are known, and a_t, the number of Fourier Features can be determined by experience, we can get the estimate of the largest b_t given the storage budget.

3.2 The NysCK Method

In this section, we propose an approximate approach: *Nyström Cluster Kernel* (NysCK). The basic idea is to obtain the eigensystem of the kernel matrix K by employing Nyström.

One of the main characteristics of the Nyström method is that it uses samples to reduce the decomposing of the given $n\times n$ kernel matrix in the original problem to the decomposing of an $s\times s$ matrix, where s is the number of samples, much smaller than n. Consider that we do rank-k approximation (with $k < s$), we sample s instances from data set $\mathcal{X} = \{\boldsymbol{x}_1, \cdots, \boldsymbol{x}_n\}$. Here, we simply use uniform sampling without replacement [Williams and Seeger, 2001; Kumar *et al.*, 2012]. Based on the instances we have sampled, we can calculate C and W. Let C denote the $n\times s$ matrix formed by corresponding s columns of original kernel matrix K and W the $s\times s$ matrix consisting of the intersection of these s columns with the corresponding s rows of K. Note that W is symmetric positive semidefinite (SPSD) since K is SPSD. Without loss of generality, the columns and rows of K can be rearranged based on this sampling so that K and C can be written as follows:

$$K = \begin{bmatrix} W & K_{21}^\top \\ K_{21} & K_{22}\end{bmatrix} \quad \text{and} \quad C = \begin{bmatrix} W \\ K_{21}\end{bmatrix}. \tag{11}$$

To compute the first k approximate eigenvalues Σ_k and the corresponding k approximate eigenvectors U_k of the kernel matrix K, we use SVD $W_k = U_{W,k}\Sigma_{W,k}U_{W,k}^\top$ where W_k is the best rank-k approximation of W. Similar to classical Nyström method [Williams and Seeger, 2001], we can use C and W from equation (11) to generate a rank-k approximation K_k of K for $k < n$ defined by

$$K_k = CW_k^\dagger C^\top \approx K. \tag{12}$$

where $W_k^\dagger$ denotes the pseudo-inverse of W_k. And the eigenvalues and eigenvectors are computed by

$$\Sigma_k = \left(\frac{n}{s}\right)\Sigma_{W,k} \quad \text{and} \quad U_k = \sqrt{\frac{s}{n}}CU_{W,k}\Sigma_{W,k}^\dagger \tag{13}$$

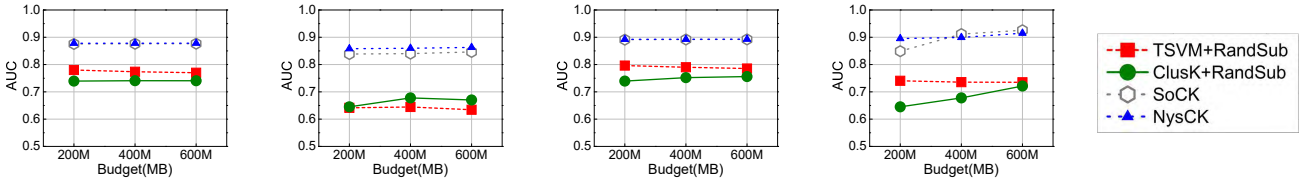
where $\Sigma_{W,k}^\dagger$ denotes the pseudo-inverse of $\Sigma_{W,k}$. After getting the approximate eigensystem of matrix K, we can easily obtain the approximation of virtual samples F by following (4), (5) and (6). The procedure is summarized in Algorithm 2.

Denote by n the number of all samples, and d the dimension. From the above discussion, it is clear that the space complexity is $O(n(d+k))$. The running time is dominated by calculating D, which takes $O(n^2d)$ time and the eigendecomposition which takes $O(nk^2)$ time. We can omit the time cost of other steps. In summary, the time complexity is $O(n(nd+k^2))$, and the space complexity is linear in n. We provide a summary of the space and time complexity of cluster kernel method, SoCK and NysCK in Table 1. Note that n is the number of

Algorithm 2 NysCK

Input: Labeled data $\{(\boldsymbol{x}_1, y_1), \cdots, (\boldsymbol{x}_l, y_l)\}$,
unlabeled data $\{\boldsymbol{x}_{l+1}, \cdots, \boldsymbol{x}_{l+u}\}$, $l \ll u$, $n = l+u$;
Output: Predicted label $\boldsymbol{y} = [y_{l+1}, \cdots, y_{l+u}]^\top$ of unlabeled data.

1: Sample s instances and calculate D, W and C;
2: Get $\Sigma_{W,k}$ and $U_{W,k}$ from SVD $W_K = U_{W,k}\Sigma_{W,k}U_{W,k}^\top$;
3: Compute Σ_k and U_k using (13);
4: Apply a transfer function φ on every element of Σ_k to get $\tilde{\Sigma}_k$;
5: Compute $\tilde{L}$, $\tilde{D}$ and F using (4), (5) and (6);
6: Regard every row of F as a new instance to perform linear SVM.



(a) 1K labeled training examples @adult-a (b) 1K labeled training examples @w8a (c) 3K labeled training examples @adult-a (d) 3K labeled training examples @w8a

Figure 1: Comparison among TSVM+RandSub, ClusK+RandSub, SoCK and NysCK on AUC. The original TSVM (kernel type) and ClusK cannot deal with such a large data set even when the largest budget (i.e., 600MB) was allocated.

Table 1: Space and time complexity comparisons.

Methods	Space	Time
ClusK	$O(n^2)$	$O(n^3)$
NysCK	$O(n(d+k))$	$O(n(nd+k^2))$
SoCK	$O(n(d+a_t+b_t))$	$O(n[nd+da_t+(a_t+b_t)^2+k^2])$

all instances, either labeled or unlabeled, d is the dimension of each instance, k is the number of nonzero eigenvalues and a_t and b_t are the temporary parameters generated during the SoCK implementation process.

Suppose each double float costs 8 bytes in storage. Given n examples with dimensionality d, the storage required for these examples is $dn \times 8$ bytes. After applying NysCK, we need to store a matrix C which contains $n \times s$ elements. So the required storage for C is $sn \times 8$ bytes. We also obtain matrix U_k and F. These two matrices both have n rows and k columns which take $2kn \times 8$ bytes. Other matrices generated during our process either have $s \times s$ elements or appear with k rows and k columns which can be ignored since s and k are very small compared with n. So, the overall storage required by the NysCK method for exploiting all the labeled and unlabeled examples is $(d+s+2k)n \times 8$ bytes. Since d and n are known, by assuming $s = k \log k$ [Boutsidis *et al.*, 2009], we can get the estimate of the largest k tolerated by the storage budget.

4 Experiments

In our experiments, we consider two scenarios, one is that algorithms are restricted to limited storage budgets, the other is that we have enough storage budgets to exploit all available unlabeled data. We also discuss the difference between SoCK and NysCK. All the experiments performed on the cores with CPU clocked at 2.53GHz. All the experiments are repeated for 30 times and the average results are presented.

4.1 With Storage Limit

First, we learn how algorithms work given storage budgets. In this setting, due to the storage budget, we can not include all the test data in the training process, thus transductive problem becomes an inductive one. Therefore all the classic transductive methods cannot be compared with. So we compare two inductive methods, namely Cluster Kernel method (ClusK) and Transductive SVM (TSVM) [Joachims, 1999]. According to [Chapelle *et al.*, 2002], σ of Gaussian kernel is set to 0.55 for both proposed algorithms as well as ClusK. We run experiments on two large scale UCI data sets, named as adult-a and w8a respectively. Adult-a has 32,561 samples with 123 dimensions, meanwhile w8a contains 49,749 samples with 300 dimensions. We randomly pick 1K or 3K examples to use as labeled training examples and regard the remaining ones as unlabeled examples. In the experiments we evaluate the performances under three storage budgets, i.e., 200MB, 400MB and 600MB. The original ClusK and kernel TSVM cannot deal with such two large data sets even when the largest budget (i.e., 600MB) is allocated. So we choose to do random sampling on unlabeled data to relieve this problem. ClusK and TSVM facilitated with random sampling [Delalleau *et al.*, 2006] are denoted by *ClusK+RandSub* and *TSVM+RandSub*, respectively. We gradually increase the sampling number until MATLAB is out of memory given the corresponding storage budget. In addition, we use the calculation described at the end of Section 3.1 and Section 3.2 to estimate b_t and k for SoCK and NysCK respectively. It is worth mentioning that the positive category ratio of adult-a is 0.2408 and w8a 0.0297. So we adopt AUC as our performance measure since AUC is insensitive to class imbalance data.

Table 2: Time used (measured in seconds) on adult-a and w8a datasets. TR is abbreviated for TSVM+RandSub, and CR is abbreviated for ClusK+RandSub. #labeled means the number of labeled instances and Budget is the memory budget measured by MB.

#labeled	Data	Budget	TR	CR	SoCK	NysCK
1,000	adult-a	200	4,940	13	2,230	2,175
		400	9,990	24	2,230	2,190
		600	12,800	39	2,230	2,240
	w8a	200	5,430	12	7,300	7,300
		400	10,600	20	7,400	7,300
		600	13,300	28	7,450	7,300
3,000	adult-a	200	2,722	13	728	700
		400	5,878	23	798	715
		600	9,043	84	821	765
	w8a	200	3,694	11	1,750	1,720
		400	7,353	19	1,755	1,770
		600	11,700	40	1,750	1,715

As can be seen from Figure 1, SoCK and NysCK outperform other two methods under all budgets. Note that some of the AUC value of TSVM+RandSub drop slightly with increasing memory budget. Actually, it has been reported in previous studies that the performance of TSVM may decrease when unlabeled data are used [Cozman *et al.*, 2003; Li and Zhou, 2015]. The reason is probably that there exist multiple Large Margin Separators (LMS) given few label examples and a

Table 3: AUC on UCI data sets without storage limit. The number in parentheses shows the relative rank of the algorithm on the corresponding data set. The smaller the rank, the better the relative performance. "size" means the number of instances; "dim" means the dimensionality. The best performance on each data set is bolded.

Dataset [size, dim]	KNN	Harmonic	CMN	TSVM	ClusK	SoCK	NysCK
australian[690,42]	.743(7)	.754(5)	.754(6)	.851(4)	.873(3)	**.902**(1)	.897(2)
credit-a[653,15]	.805(7)	.874(5)	.864(6)	.894(3)	**.901**(1)	.884(4)	.896(2)
credit-g[1000,20]	.591(7)	.670(5)	.670(6)	.700(4)	.711(2)	**.723**(1)	.705(3)
diabetes[768,8]	.640(7)	.737(5)	.737(6)	.781(2)	**.801**(1)	.775(3)	.757(4)
german[1000,59]	.587(7)	.665(4)	.665(5)	.655(6)	.691(2)	**.710**(1)	.669(3)
kr-vs-kp[3196,36]	.821(7)	.917(6)	.917(5)	.928(4)	**.990**(1)	.985(2)	.980(3)
splice[3175,60]	.678(7)	.782(5)	.782(6)	.825(3)	**.899**(1)	.891(2)	.823(4)
svmguide3[1284,22]	.605(7)	.645(4)	.643(5)	.629(6)	.769(2)	.701(3)	**.771**(1)
Total rank	56	39	45	32	13	17	22

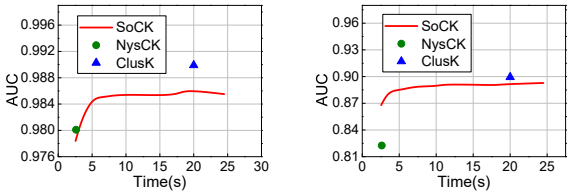
large amount of unlabeled data, and it is difficult to select a correct LMS without sufficient domain knowledge. Table 2 gives out the time using of those methods mentioned above on adult-a and w8a. As can be seen from Table 2, SoCK and NysCK are much faster than TSVM+RandSub even though TSVM+RandSub samples a small number of unlabeled data to do experiments while our methods utilize all the unlabeled data. Besides, NysCK does not appear much faster than SoCK because we set the number of iterations of SoCK to a very small value, say, 20 to decrease the running time. If the number of iterations of SoCK increases, the performance would be better. Finally, ClusK+RandSub is the fastest one due to the small data set by random sampling.

4.2 Without Storage Limit

In this section, we study the case without storage limit and show that even in this situation, our methods can still work well. The compared methods are listed as follows:

- Harmonic Gaussian Field method (Harmonic) which does not consider the class imbalance [Zhu *et al.*, 2003].
- Class Mass Normalization (CMN) which adjusts the class distributions to match the priors [Zhu *et al.*, 2003].
- Cluster Kernel method [Chapelle *et al.*, 2002].
- Transductive SVM [Joachims, 1999].
- The classic 1-Nearest Neighbour Classifier (1NN).

All the parameters are selected via five-fold cross-validation. We evaluate the proposed algorithms on 8 UCI data sets. On each data set we randomly pick 10% examples to use as labeled training examples and regard the remaining ones as unlabeled examples. The experiments are repeated 30 times and the average AUC values on the unlabeled data are reported. The results are showed in Table 3. As can be seen, the proposed algorithms achieve competitive performance on all data sets. In particular, ClusK obtains the best performance on 4 data sets in Table 3 as well as gets the highest score on total rank. This phenomenon is as expected because ClusK is our best baseline calculating the full kernel of all the data. Our methods are slightly worse than ClusK due to the approximation characteristic. Moreover, ClusK, SoCK and NysCK take the first three places on most data sets. You might notice that SoCK and NysCK perform not as well as ClusK on svmguide3 and splice, respectively. This is related to the distribution of the eigenvalues. Yang *et al.* [2012] find that when there is a large gap in the eigen-spectrum of the kernel matrix,



(a) AUC @kr-vs-kp (b) AUC @splice

Figure 2: Comparison between SoCK and NysCK. The performance is positively related to the approximate error in SoCK.

approaches based on the Nyström method can yield better generalization error bound than random Fourier features based approach. The spectrum in svmguide3 is highly skewed where the biggest eigenvalue is 1 and the others are almost 0 while in splice most eigenvalues are 1 and few are 0. NysCK is more effective when the eigenvalues of the kernel matrix are highly skewed just as that in svmguide3.

4.3 Comparison between SoCK and NysCK

Finally, we discuss the difference between SoCK and NysCK. We experiment on two UCI data sets, kr-vs-kp and splice with ClusK, SoCK and NysCK. We continually increase the running time of SoCK by controlling the number of iterations. As can be seen from Figure 2, NysCK gets an approximate solution with a not high AUC value in a short time while ClusK obtains a good solution with the highest AUC value but with longer time and larger memory. SoCK can refine its solution continuously with the decreasing of approximation error and outperforms NysCK after a few seconds. Thus SoCK is more effective while NysCK is more efficient.

5 Conclusion

In this paper, we focus on a new setting: storage fit learning with unlabeled data. The key to this setting is that, given different storage budgets, even for the same data, the behavior of the algorithm should be adjusted differently. Considering that those algorithms relying on spectral analysis suffer seriously from storage burden of kernel matrix, we utilize the techniques of low-rank approximation and present two simple yet effective techniques which are able to adapt such kind of algorithms to be fit for a given storage budget.

References

[Avron *et al.*, 2012] H. Avron, S. Kale, S. P. Kasiviswanathan, and V. Sindhwani. Efficient and practical stochastic subgradient descent for nuclear norm regularization. In *ICML*, 2012.

[Belkin *et al.*, 2004] M. Belkin, I. Matveeva, and P. Niyogi. Regularization and semi-supervised learning on large graphs. In *COLT*, 2004.

[Blum and Chawla, 2001] A. Blum and S. Chawla. Learning from labeled and unlabeled data using graph mincuts. In *ICML*, 2001.

[Boutsidis *et al.*, 2009] C. Boutsidis, M. Mahoney, and P. Drineas. An improved approximation algorithm for the column subset selection problem. In *SODA*, 2009.

[Brabanter *et al.*, 2010] K. De Brabanter, J. De Brabanter, J. Suykens, and B. De Moor. Optimized fixed-size kernel models for large data sets. *Computational Statistics & Data Analysis*, 54(6):1484–1504, 2010.

[Cai *et al.*, 2010] J.-F. Cai, E. Candès, and Z. Shen. A singular value thresholding algorithm for matrix completion. *SIAM Journ. on Opt.*, 20(4):1956–1982, 2010.

[Chapelle *et al.*, 2002] O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In *NIPS*, 2002.

[Cozman *et al.*, 2003] F. G. Cozman, I. Cohen, and M. C. Cirelo. Semi-supervised learning of mixture models. In *ICML*, 2003.

[Delalleau *et al.*, 2006] O. Delalleau, Y. Bengio, and N. L. Roux. *Large-Scale Algorithms*, pages 333–341. MIT Press, 2006.

[Fowlkes *et al.*, 2004] C. Fowlkes, S. Belongie, F. Chung, and J. Malik. Spectral grouping using the Nyström method. *IEEE TPAMI*, 26(2):214–225, 2004.

[Grady and Funka-Lea, 2004] L. Grady and G. Funka-Lea. Multilabel image segmentation for medical applications based on graph-theoretic electrical potentials. In *ECCV*, 2004.

[Han *et al.*, 2016] Y. Han, Y. Shen, and M. Filippone. Mini-batch spectral clustering. *CoRR*, abs/1607.02024, 2016.

[Joachims, 1999] T. Joachims. Transductive inference for text classification using support vector machines. In *ICML*, 1999.

[Joachims, 2003] T. Joachims. Transductive learning via spectral graph partitioning. In *ICML*, 2003.

[Jumutc *et al.*, 2013] V. Jumutc, X. Huang, and J. Suykens. Fixed-size Pegasos for hinge and pinball loss SVM. In *IJCNN*, 2013.

[Kemp *et al.*, 2003] C. Kemp, T. Griffiths, S. Stromsten, and J. Tenenbaum. Semi-supervised learning with trees. In *NIPS*, 2003.

[Kumar *et al.*, 2009] S. Kumar, M. Mohri, and A. Talwalkar. Ensemble nystrom method. In *NIPS*, 2009.

[Kumar *et al.*, 2012] S. Kumar, M. Mohri, and A. Talwalkar. Sampling methods for the Nyström method. *JMLR*, 13:981–1006, 2012.

[Levin *et al.*, 2004] A. Levin, D. Lischinski, and Y. Weiss. Colorization using optimization. *ACM Trans. on Graph.*, 23(3):689–694, 2004.

[Li and Zhou, 2015] Y.-F. Li and Z.-H. Zhou. Towards making unlabeled data never hurt. *IEEE TPAMI*, 37(1):175–188, 2015.

[Li *et al.*, 2016] Y. Li, J. Huang, and W. Liu. Scalable sequential spectral clustering. In *AAAI*, 2016.

[Liu *et al.*, 2012] W. Liu, J. Wang, and S.-F. Chang. Robust and scalable graph-based semisupervised learning. *Proceedings of the IEEE*, 100(9):2624–2638, 2012.

[Mehrkanoon and Suykens, 2014] S. Mehrkanoon and J. A. K. Suykens. Large scale semi-supervised learning using KSC based model. In *IJCNN*, 2014.

[Nesterov, 2013] Y. Nesterov. Gradient methods for minimizing composite functions. *Math. Prog.*, 140(1):125–161, 2013.

[Pan and Chen, 1999] V. Y. Pan and Z. Q. Chen. The complexity of the matrix eigenproblem. In *STOC*, 1999.

[Pfahringer *et al.*, 2007] B. Pfahringer, C. Leschi, and P. Reutemann. Scaling up semi-supervised learning: An efficient and effective LLGC variant. In *PAKDD*, 2007.

[Rahimi and Recht, 2007] A. Rahimi and B. Recht. Random features for large-scale kernel machines. In *NIPS*, 2007.

[Sánchez *et al.*, 2013] J. Sánchez, F. Perronnin, T. Mensink, and J. Verbeek. Image classification with the fisher vector: Theory and practice. *IJCV*, 105(3):222–245, 2013.

[Williams and Seeger, 2001] C. Williams and M. Seeger. Using the Nyström method to speed up kernel machines. In *NIPS*, 2001.

[Yang *et al.*, 2012] T. Yang, Y.-F. Li, M. Mahdavi, R. Jin, and Z.-H. Zhou. Nyström method vs random Fourier features: A theoretical and empirical comparison. In *NIPS*, 2012.

[Zhang and Kwok, 2010] K. Zhang and J. Kwok. Clustered Nyström method for large scale manifold learning and dimension reduction. *IEEE TNN*, 21(10):1576–1587, 2010.

[Zhang *et al.*, 2012] K. Zhang, L. Lan, Z. Wang, and F. Moerchen. Scaling up kernel SVM on limited resources: A low-rank linearization approach. In *AISTATS*, 2012.

[Zhang *et al.*, 2016] L. Zhang, T. Yang, J.-F. Yi, R. Jin, and Z.-H. Zhou. Stochastic optimization for kernel PCA. In *AAAI*, 2016.

[Zhou *et al.*, 2003] D. Zhou, O. Bousquet, T. N. Lal, J. Weston, and B. Schölkopf. Learning with local and global consistency. In *NIPS*, 2003.

[Zhou *et al.*, 2009] Z.-H. Zhou, M. K. Ng, Q.-Q. She, and Y. Jiang. Budget semi-supervised learning. In *PAKDD*, 2009.

[Zhu *et al.*, 2003] X. Zhu, Z. Ghahramani, and J. Lafferty. Semi-supervised learning using Gaussian fields and harmonic functions. In *ICML*, 2003.

[Zhu *et al.*, 2004] X. Zhu, J. Kandola, Z. Ghahramani, and J. Lafferty. Nonparametric transforms of graph kernels for semi-supervised learning. In *NIPS*, 2004.

[Zhu *et al.*, 2005] X. Zhu, J. Lafferty, and R. Rosenfeld. *Semi-supervised learning with graphs*. Carnegie Mellon University, Language Technologies Institute, School of Computer Science, 2005.

[Zhu, 2007] X. Zhu. Semi-supervised learning literature survey. Technical report, University of Wisconsin-Madison, 2007.