Data Selection Using SASH Trees for Support Vector Machines

Chaofan Sun and Ricardo Vilalta

Department of Computer Science, University of Houston
4800 Calhoun Rd., Houston TX, 77204-3010
{cfsun, vilalta}@cs.uh.edu
Center for Research and Advanced Studies (CINVESTAV)
Av. Científica 1145, Guadalajara, México, 45010
rvilalta@gdl.cinvestav.mx

Abstract. This paper presents a data preprocessing procedure to select support vector (SV) candidates. We select decision boundary region vectors (BRVs) as SV candidates. Without the need to use the decision boundary, BRVs can be selected based on a vector's nearest neighbor of opposite class (NNO). To speed up the process, two spatial approximation sample hierarchical (SASH) trees are used for estimating the BRVs. Empirical results show that our data selection procedure can reduce a full dataset to the number of SVs or only slightly higher. Training with the selected subset gives performance comparable to that of the full dataset. For large datasets, overall time spent in selecting and training on the smaller dataset is significantly lower than the time used in training on the full dataset.

Keywords: sampling methods, support vector machines.

1 Introduction

Support vector machines (SVMs) [1] are a popular approach to machine learning because of their solid analytical foundation and frequent high generalization power. However, standard maximal margin SVMs face the difficulty of solving a quadratic programming (QP) problem with time and space complexities of $O(n^3)$ and $O(n^2)$ respectively (where n is the size of the input dataset). When the dataset is relatively large, training SVMs becomes intolerable slow and often results in memory shortage. Multiple efforts have been undertaken to overcome these difficulties. Previous work has focused on how to simplify or modify the QP problem (e.g., the sequential minimum optimization (SMO) algorithm [2]). Another line of research has focused on reducing the training dataset by selecting only a set of potential support vectors (SVs) to define the decision boundary.

In terms of data selection, existing methodologies can be roughly divided into three categories: (1) data sampling, (2) neighborhood-based selection, and (3) boundary-based selection. In the first category, statistical techniques such as random sampling and stratified sampling [3,4,5] are employed to do data

P. Perner (Ed.): MLDM 2007, LNAI 4571, pp. 286–295, 2007.

selection. These techniques are simple but tend to miss SVs and select non-SVs. In the second category, data selection is done by focusing on regions populated by k-mean clusters ([6,7]), or on vector neighborhoods (such as k-NNs [8,9], or Gabriel neighbors [10]). Using clustering to do data selection relies heavily on the existence of regions with high sample density; sparse data makes it difficult to identify clusters even when some of these data should be kept as SVs candidates. Hence, using clusters to select SVs candidates tends to be overly conservative. In addition, using k-NNs to select SVs candidates is based on the assumption that data close to the decision boundary should have a mixed number of class labels among their neighbors; a common research problem is to measure the degree of k-NN purity, but a major problem is the time used to find all k-NNs which is at least $O(n^2)$ without using an index. In the third category, data selection techniques find the decision boundary, such as [11,12,13]. The general procedure follows two steps: 1) find a tentative decision boundary; and 2) use the boundary to find potential SVs. The selected vectors are used again during the first step to update the decision boundary. The two steps are repeated until no more updating is necessary. Even though this approach will eventually generate a decision boundary, it may need a long search and do multiple scans over the dataset before it converges.

In this paper we propose the following two steps: (1) select the decision boundary region vectors (BRVs) as the SVs candidates without using iterative decision boundary, and (2) use the spatial approximation sample hierarchy (SASH [14]) tree structure to speed up this process. In section 2, we provide a detailed account of our approach followed by an empirical evaluation in Section 3. Section 4 gives conclusions and discusses future work.

2 Data Selection Based on Boundary Region Vectors

Most vectors far from the decision boundary are not SVs and can be safely removed. Without knowing the exact position of the decision boundary, however, we can always find a vector close to the boundary followed by a search for its nearest neighbor of opposite class (NNO); we can then consider them both as SV candidates. The concept of NNO is used for data condensing in instance-based learning (IBL). Data condensing aims at selecting the minimal subset of data that preserves the same accuracy to that obtained when invoking a 1-NN algorithm on the full dataset. Methodologies developed in data condensing mainly focus on the quality (minimal or not) and accuracy (consistent or not) of the condensed set. For example, Dasarathy's MCS (minimum consistent set [15]) condenses a dataset based on k-NNs of the same class and its nearest opposite-class neighbors.

Different from data condensing, our purpose using NNOs is to select SV candidates similar to those identified by SVMs. A positive vector $\mathbf{x}_i^+$'s NNO can be obtained by comparing the distances from this vector to all negative vectors $\mathbf{x}_j^-$:

$$NNO_i = \arg\min_{\mathbf{x}_j^-} ||\mathbf{x}_i^+ - \mathbf{x}_j^-||.$$

Algorithm 1. Finding BRVs using a distance matrix $D \in \mathbb{R}^{n_p \times n_n}$. In each step, the set of marked vectors M is reduced to size k_n or k_p, respectively.

```
Given: k_p ≥ 1, k_n ≥ 1
Output: B, the set of BRVs
B = {}
for 1 ≤ i ≤ n_p:
  d_i = k_n th smallest distance in row i
  M = {x_j | D_{i,j} ≤ d_i}
  B = B ∪ M
for 1 ≤ j ≤ n_n:
  d_j = k_p th smallest distance in column j
  M = {x_i | D_{i,j} ≤ d_j}, where
  B = B ∪ M
```

The same can be done to find a negative vector's NNO. We define the union of NNOs for both classes as boundary region vectors (BRVs); these are assumed to be within or close to the region around the decision boundary delineated by the margin.

2.1 Finding BRVs Using a Pairwise Matrix

Our approach begins by splitting the full dataset into positive and negative subsets, with size n_p and n_n respectively. A 2-d distance matrix D with n_p lines and n_n columns is then created. Along the i^{th} row of the matrix, the j^{th} element is the distance from positive vector $\mathbf{x}_i^+$ to negative vector $\mathbf{x}_j^-$. The $\mathbf{x}_i^+$'s NNO can be found by looking for the shortest distance in the i row. Suppose this shortest distance is found on the v^{th} column; vector $\mathbf{x}_v^-$ is then the NNO. The procedure iterates until all NNOs in both classes are found and selected as BRVs (see Algorithm 1). In this algorithm, more than one NNO is allowed to be selected for each vector, i.e. $k_p \geq 1$ and $k_n \geq 1$. We will discuss this in the next section.

Algorithm 1 is an effective approach to finding nearest neighbors across the decision boundary. The time and space complexities of this algorithm are both of $O(n^2)$ because we need to compute every pairwise distance and store that in memory. Large datasets demand an index to speed up the search for BRVs.

2.2 Approximating BRVs Using SASH Trees

There are many indexing structures that can be used to reduce the complexity of NN search, such as R-tree. In this study, we use SASH trees because they require tuning fewer parameters than other methods. Since SV data selection is a data preprocessing step, computing minimal and consistent subsets is unnecessary.

To guide the NN search, a SASH tree assumes that transitivity holds for the NN relation. Specifically, if vector $\mathbf{x}$ is a NN vector of $\mathbf{y}$, and $\mathbf{y}$ is a NN vector

Algorithm 2. Finding the approximate BRVs given positive and negative SASH trees T^+ and T^-. The dataset X is split into classes X^+ and X^-. The function k-CLOSEST(k, u, X) returns the k closest elements in the set X to the node u.

```
Given: number of NNOs in each class, k_+ ≥ 1, k_− ≥ 1
Output: approximate BRVs, S
S = {}
for c ∈ {+, −}:
  c̄ = {+, −} − c
  for u ∈ X^(c):
    P_1(u) = root of T^(c̄)
    for 2 ≤ i ≤ HEIGHT(T^(c̄)):
      C_i(u) = {v ∈ P_{i−1}(u)}
      P_i(u) = k-CLOSEST(p, u, C_i(u))
    P(u) = ⋃_i P_i(u)
    s(u) = k-CLOSEST(k_c̄, u, P(u))
    S = S ∪ s(u)
```

of $\mathbf{z}$, then $\mathbf{x}$ is likely to be a NN vector of $\mathbf{z}$. As a result, only approximate NNs in adjacent levels are connected to each other. Although a SASH tree cannot guarantee finding all exact NNs, it limits the NNs search in each level of the tree to a small number of candidates, thus significantly reducing search time. Similar work [16] uses a clustering feature hierarchy as index but guides the data selection process through a tentative decision boundary.

Construction of SASH trees. A SASH tree consists of n nodes, where n is the number of data vectors. Each vector is randomly assigned to a node, such that starting from the bottom, there are $\frac{n}{2}, \frac{n}{4}, \ldots, 1$ nodes in each level. Tree levels are numbered starting from 1 for the top level until h for the bottom level, where $h \approx \log_2 n$ is the height of the tree. Each node is allowed to have at most p parents and c children. The leaf nodes have no children. The root node has no parent and is fully connected to all nodes in level 2.

Links for the interior nodes are created through the following iterative process: (1) Interior nodes in levels $3 \leq i \leq h$ are only connected to p parent nodes in level $i-1$. Parent nodes are selected from a pool of pc candidates. The candidates are obtained by selecting at most pc nearest nodes in each level, starting from the root. (2) For each parent node, all but the c nearest child nodes are removed.

Because pc is much smaller than the actual number of nodes in most levels, finding the p nearest parents substantially reduces search time. The time complexity for connecting one node is $O(pch)$, or $O(pc \log_2 n)$. Tree construction has time complexity $O(n \log_2 n)$ and space complexity $O(n)$.

Approximating BRVs using SASH tress. To speed up our BRVs search, we need to construct positive and negative SASH trees. After each SASH tree is constructed, we find BRVs by querying the opposite tree for each node. Querying

is a top-down search similar to finding the parents for a child node. For each node in the i^{th} level of the positive SASH tree, searching starts from the root node of the negative SASH tree. In each level, at most pc distances are compared and only the p closest parents are collected. The final k_n NNOs of the positive node are selected from the p closest nodes obtained from the search. Through this process, BRVs are selected for all nodes, as described in Algorithm 2. The time complexity for BRVs querying is $O(n \log_2 n)$.

3 Experiments and Discussions

3.1 Experiment Setup

In this study, we consider only binary-class datasets. Data are taken from UCI repository with each variable scaled to $[-1, 1]$. For each dataset, we do the following: (1) select BRVs, (2) train on the selected subset of data and save the resulting model, (3) assess the model on the testing dataset and obtain accuracy results, and (4) stop if there is no performance improvement in two consecutive runs; otherwise go back to step (1) and select more data after increasing the value of k (parameter of nearest neighbor).

For our implementation, we used the publicly available LIBSVM library, C++ version 2.83, to train the SVMs [17]. SASH trees are also implemented in C++. Experiments were run on a PC with a 2.2 GHz Pentium CPU and 1GB of RAM. The classifier used is C-SVM with L_2 penalty for noisy data and radial basis functions (RBF) as kernel. With this kernel, there are two user-defined parameters: (C, γ). We invoked a grid search to find the best-performing (C, γ) values for each dataset.

3.2 Support Vector Recovery

We detail the BRV selection process on the breast-cancer dataset. To test our SV recovery ratio, we trained on the full dataset. The dataset consists of 683 vectors and 10 features each. Using (C, γ) values of (1.0, 0.022) we found 87 SVs. Of these, 44 are positive and 43 are negative, which accounts for only 12.7% of all vectors.

We use Algorithm 1 to select BRVs and compare them to the actual 87 SVs. We first select only one NNO for each vector. In this case, BRVs can only recover 34.5% of the set of SVs (see Table 1). The corresponding SVM's accuracy is very low. As we increase the number of NNOs for each vector, k (here $k_p = k_n = k$), the number of selected BRVs increases and the SV recovery ratio grows, while accuracy improves. When $k = 8$, the selected BRVs (74 positive and 69 negative vectors) recover 90.8% of SVs, and accuracy reaches its highest value. We find that selecting more data does not necessarily yield better accuracy even when the number of recovered SVs increases. When this occurs, the ratio of BRVs (selected to reach the same accuracy as SVM on the full dataset) is termed the *critical ratio*. In this case, the critical ratio is 20.9%. The remaining 79.1% of

Table 1. SV recovery from breast cancer dataset (683 vectors with 10 features)

$k : k_p(k_n)$	1(1)	3(3)	5(5)	7(7)	8(8)	10(10)	15(15)	-
BRVs: p(n)	17(18)	40(35)	54(52)	66(64)	74(69)	83(80)	95(96)	239(444)
selection ratio	5.1	11.0	15.5	19.0	20.9	23.1	22.1	100
SV recovered (%)	34.5	64.4	79.3	86.2	90.8	92.0	94.3	100
SVM test acc(%)	65.0	95.9	96.2	96.5	97.4	97.4	97.4	97.4

Table 2. *Critical ratios* and SVM performance for different datasets

datasets	full data set			selected BRVs		
$(n \times d)$	#SVs	SV ratio (%)	SVM acc. (%)	#BRVs	*critical ratio* (%)	SVM acc. (%)
breast cancer(683x10)	87	12.3	97.4	143	20.9	97.4
diabetes(768x8)	467	60.8	78.1	497	64.7	77.3
heart(270x13)	103	38.1	85.2	134	49.6	85.2
ionosphere (351x34)	194	55.3	100	225	64.1	98.9
mushrooms(8124x112)	313	3.8	100	1093	13.5	100

data are redundant and can be removed. This critical ratio is 8% higher than the SV ratio.

The critical ratio is slightly higher than the actual SV ratio for two main reasons. (1) some BRVs are not SVs but lie close to the decision boundary; (2) noisy data causes a vector to be selected as an NNO. Noise has two effects in our data selection process: *(i)* some NNOs of a noisy vector may be non-noisy. These are selected in our approach and cause the critical ratio to increase; *(ii)* a noisy vector may mislead the search of real NNOs if it is closer to a vector than the actual NNO. In this case we have to increase the k value to select more data; this helps to recover the real SVs, but increases the critical ratio.

Additional experimental results are shown in Table 2. Results show that critical ratios are 4-10% higher than SV ratios. For large datasets having low SVs ratio, the reduction is substantial. For example, the mushroom dataset can be reduced from 8124 vectors to 1093 vectors, where about 86% of the data are eliminated without performance degradation.

Our investigation also shows that data selected according to the critical ratio does not recover 100% of all SVs (again see Table 1). As we look for the best (C, γ), the highest performance contour lies in a flat region within which all SVMs have almost the same performance. This means that the best (C, γ) can change within small ranges without performance degradation. However, the SVs set is very sensitive to (C, γ) values both in terms of size and vectors contained. Within different sets of SVs, many SVs are shared; others are mutual NNs of common SVs. Shared SVs are essential to defining the decision boundary. Other SVs can be replaced by their neighbors without changing overall performance. This explains why a slight different BRVs set, even if it does not cover 100% SVs, does not affect SVMs consistency.

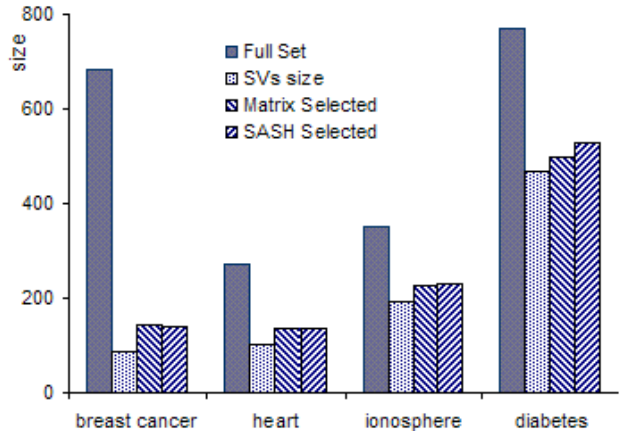

Fig. 1. Comparison of different sizes (SASHs with $p = 2$ and $c = 8$) for different datasets

3.3 Accuracy of SASH Approximation

As mentioned earlier, SASH only finds approximate nearest neighbors. There are two ways to obtain more accurate approximations. One way is to increase the values of p and c. This method will increase the time of SASH tree construction and the time of k-NNs querying. Alternatively, one can keep p and c fixed and select more SV candidates with a larger value of k-NNOs. This produces a larger critical ratio and will increase SVM's training time. Using either method can make the selected subset of data generate models comparable to those obtained with the full dataset. In practice, $p = 2$ and $c = 8$ results in a good approximation. Figure 1 compares the size of the full dataset to the subset selected by the matrix approach and the subset selected by SASHs for four datasets. Experiments show that subsets obtained from SASHs are slightly larger than those obtained from the matrix approach. However the difference is small, which leads us to conclude that SASH approximation is very similar to the matrix approach.

3.4 Overall Time Savings

To check the overall improvement in computational time, we compare the time it takes to train SVMs on a full dataset to the time it takes to do SASH data selection and training on the selected subsets. Our experiments test the Adult dataset with varying sizes (from 1600 to 32561 vectors, available from the LIBSVM website [17]). These data have 123 features (scaled to $[-1, 1]$). Experimental results are shown in Figure 2. This figure shows that when the size of the dataset is small, the overhead for data selection dominates. As the size of data increases, the time for data selection is shorter than the time needed to train on the full dataset. Training time on the reduced dataset is even shorter. Overall time savings are significant as the dataset becomes larger than 30,000 vectors. On

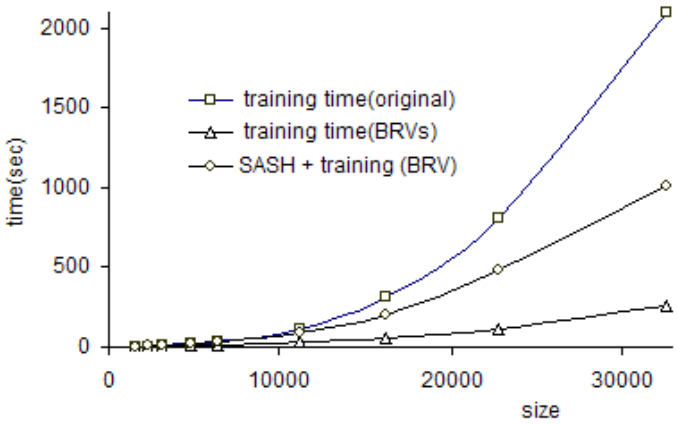

Fig. 2. Comparison of different CPU times

Table 3. Overall time comparison

dataset	full dataset			selected BRVs				overall time
$(n \times d)$	#SVs	t-time (sec)	SVM acc. (%)	#BRVs	s-time (sec)	t-time (sec)	SVM acc. (%)	saving (%)
mushrooms (8,124x112)	313	30	100	1,093	23	2	100	16.6
adult-9 (32,561x123)	1,1505	2,098	85.1	11,733	742	263	84.5	52.1
SensIT (30,000x100)	9,561	1,810	87.8	11,257	561	172	86.9	59.5
SensIT (40,000x100)	12,329	3,465	88.0	14,667	964	417	87.2	60.1
SensIT (50,000x100)	15,415	5,658	88.0	18,431	1360	823	87.2	61.4

(t-time: training time on 10-run average; s-time: selecting time on 10-run average)

additional datasets (see Table 3), we can observe that time savings are significant, particularly for over-sized datasets.

Part of our proposed strategy is based on computing and comparing the distance between two vectors in input space. When it comes to finding NNs, using Radial Basis Functions RBF as kernels does not change the NNOs nor BRVs. For two vectors $\mathbf{x}$ and $\mathbf{y}$, the distance over the input space is $d^2 = ||\mathbf{x} - \mathbf{y}||^2$. When the vectors are mapped to $\phi(\mathbf{x})$ and $\phi(\mathbf{y})$, the distance in kernel space can be written as $d_k^2 = ||\phi(\mathbf{x}) - \phi(\mathbf{y})||^2 = K(\mathbf{x}, \mathbf{x}) - 2K(\mathbf{x}, \mathbf{y}) + K(\mathbf{y}, \mathbf{y})$. Using RBF, $K(\mathbf{x}, \mathbf{y}) = e^{-\gamma||\mathbf{x}-\mathbf{y}||^2}$, we have $K(\mathbf{x}, \mathbf{x}) = K(\mathbf{y}, \mathbf{y}) = 1$. In that case the distance is $d_k^2 = 2 - 2e^{-\gamma||\mathbf{x}-\mathbf{y}||^2}$. This shows how distance order does not change in NN search. If we select k-NNOs, we will find the same vectors in input space as those

found in the kernel space. Thus we do not need to compute a kernel function to select data. This can save a lot of time as compared to the case where data selection is carried out in the kernel space.

As final remarks, we add that data selection is used to select potential SVs but will not produce gainings in time savings if a given dataset has a high SV ratio. Data with high noise ratio suffers of the same problem. For particular distributions such as equally spaced vectors, our approach is not an appropriate solution because we select data based on differences in distance.

4 Conclusions and Future Work

Data selection based on BRVs can recover most SVs when we select slightly more data than the number of actual SVs. Training on selected data subsets significantly reduces the amount of training time without degrading performance. Since the decision boundary can be defined with different set of SVs, we do not need to recover 100% SVs. Instead, selecting BRVs as SVs candidates produces almost-identical results. For large datasets, the SASH tree can be used to approximate all BRVs. Time saving are significant compared to training an SVM on the full dataset.

Future work will focus on studying performance on extremely large datasets, and in using fixed-sized SASH trees to approximate BRVs. As noisy data causes more data to be selected, we will consider different techniques to detect and remove noise vectors. For the case of highly unbalanced datasets, selecting the same k-NNOs for both classes can have a strong impact on the example distribution. Future work will study how to address the class imbalance problem using our proposed techniques for data selection.

References

1. Vapnik, V.N.: Statistical Learning Theory. Addison-Wiley, New York, NY (1998)
2. Platt, J.: Sequential minimal optimization: A fast algorithm for training support vector machines. In: Microsoft Research Technical Report MSR-TR-98-14 (1998)
3. Bazzani, A., Bevilacqua, A., Bollini, D., Brancaccio, R., Campanini, R., Lanconelli, N., Riccardi, A., Romani, D.: An SVM classifier to separate false signals from microcalcifications in digital mammograms. vol 46, pp. 1651–1663 (2001)
4. Schohn, G., Cohn, D.: Less is more: Active learning with support vector machines. In: Proceedings of the 17th International Conference on Machine Learning (ICML), pp. 839–846 (2000)
5. Milenova, Boriana, L., Yarmus, J.S., Campos, M.M.: SVM in oracle database 10g: Removing the barriers to widespread adoption of support vector machines. In: Proceedings of the 31st VLDB Conference, Trondheim, Norway (2005)
6. Almeida, M., Braga, A.P., Braga, J.P.: SVM-KM: Speeding SVMs learning with a priori cluster selection and k-means. In: Proceedings of the 6th Brazilian Symposium on Neural Networks, pp. 162–167 (2000)
7. Koggalage, R., Halgamuge, S.: Reducing the number of training samples for fast support vector machine classification. Neural Information Processing - Letters and Reviews 2, 57–65 (2004)

8. Shin, H., Cho, S.: Fast pattern selection for support vector classifiers. In: Proceedings of the 7th Pacific-Asia Conference on Knowledge Discovery and Data Mining. Lecture Notes in Artificial Intelligence (LNAI 2637), pp. 376–387 (2003)
9. Wang, J., Neskovic, P., Cooper, L.N.: Training data selection for support vector machines. In: Wang, L., Chen, K., Ong, Y.S. (eds.) ICNC 2005. LNCS, vol. 3610, pp. 554–564. Springer, Heidelberg (2005)
10. Zhang, W., King, I.: A study of the relationship between support vector machine and gabriel graph. In: IEEE, pp. 239–245 (2002)
11. Roobaert, D.: DirectSVM: A fast and simple support vector machine perceptron. In: Proceedings. IEEE Int. Workshop Neural Networks for Signal Processing, Sydney, Australia, pp. 356–365 (2000)
12. Vishwanathan, S.V.N., Murty, N.M.: A simple SVM algorithm. In: Proceedings 2002 International Joint Conference on Neural Networks. IJCNN '02. vol. 3, Honolulu, Hawaii, pp. 2393–2398 (2002)
13. Raicharoen, T., Lursinsap, C.: Critical support vector machine without kernel function. In: Proc. of 9th International Conference on Neural Information. vol. 5, pp. 2532–2536 (2002)
14. Houle, M.E.: SASH: A spatial approximation sample hierarchy for similarity search. Technical Report pages 16, IBM Tokyo Research Laboratory Report RT-0517 (March 5, 2003)
15. Dasarathy, B.V., Sanchez, J.S., Townsend, S.: Nearest neighbour editing and condensing tools-synergy exploitation. In: Pattern Analysis & Applications, vol. 3, pp. 19–30. Springer, London (2000)
16. Yu, H., Yang, J., Han, J.: Classifying large data sets using SVM with hierarchical clusters. In: SIGKDD '03 Washington, DC, USA (2003)
17. Chang, C.C., Lin, C.J.: LIBSVM: A library for support vector machines (2001)