Cone Cluster Labeling for Support Vector Clustering

Sei-Hyung Lee
Department of Computer Science
University of Massachusetts Lowell
MA 01854, U.S.A.
slee@cs.uml.edu

Karen M. Daniels
Department of Computer Science
University of Massachusetts Lowell
MA 01854, U.S.A.
kdaniels@cs.uml.edu

Abstract

Clustering forms natural groupings of data points that maximize intra-cluster similarity and minimize inter-cluster similarity. Support Vector Clustering (SVC) is a clustering algorithm that can handle arbitrary cluster shapes. One of the major SVC challenges is a cluster labeling performance bottleneck. We propose a novel cluster labeling algorithm that relies on approximate coverings both in feature space and data space. Comparison with existing cluster labeling approaches suggests that our strategy improves efficiency without sacrificing clustering quality.

1 Introduction

1.1 Clustering Overview Clustering is a natural grouping or unsupervised classification of data into groups [6]. Most clustering algorithms use one or a combination of the following techniques: graph-based [5, 9, 10, 7], density-based [8], model-based methods using either a statistical approach or a neural network approach, or optimization of a clustering criterion function. Constructing cluster boundaries is another popular technique [10, 11]. SVC is similar to a boundary-finding clustering method except that it only finds certain points (called *support vectors*) on the boundary of each cluster.

1.2 Support Vector Clustering As with support vector machines [4], SVC uses a nonlinear mapping of the data from the data space into a high-dimensional feature space. Whereas support vector machines use a linear separator in feature space in order to separate and classify data points, SVC uses a minimal hypersphere encompassing feature space images of data points [2, 3]. The minimal hypersphere corresponds to several disjoint bounded regions in the data space that are contours and are interpreted as clusters. The cluster labeling challenge of SVC is to associate each data point with a cluster, using only the available operations, without explicitly constructing the contours in data space.

Given a finite set $\mathcal{X} \subseteq \mathcal{R}^d$ of N distinct data points, the minimal hypersphere of radius R enclosing all data points' images in the feature space can be described by the following, as in [2]:

$$(1.1) \qquad \|\Phi(x) - a\|^2 \leq R^2 \quad \forall x \in \mathcal{X},$$

where Φ is a nonlinear mapping from data space to feature space, $\Phi(x)$ is the feature space image of data point x, $\|\cdot\|$ is the Euclidean norm, and a is the center of the sphere. Therefore, data points can be categorized into three groups based on the location of their images in feature space: (1) points whose images are on the surface of the minimal hypersphere are *Support Vectors* (SVs), (2) points whose images are outside of the minimal hypersphere are *Bounded Support Vectors* (BSVs), and (3) points whose images are inside the minimal hypersphere.

The mapping from data space to feature space is determined by a kernel function, $K : \mathcal{X} \times \mathcal{X} \to \mathcal{R}$, that defines the inner product of image points. As in other SVC literature [1, 2], we use a Gaussian kernel given by Eq. 1.2 below:

$$(1.2) \quad K(x_i, x_j) = e^{-q\|x_i - x_j\|^2} = \langle \Phi(x_i) \cdot \Phi(x_j) \rangle,$$

where q is the width of the Gaussian kernel. From Eq. 1.2, it immediately follows that,

$$(1.3) \qquad K(x_i, x_i) = 1.$$

This implies that all data point images are on the surface of the unit ball in feature space.

Cluster labeling, the focus of this paper, has a goal of grouping together data points that are in the same contour obtained from the minimal hypersphere. Prior work is discussed in Section 1.3. In that work, cluster labeling has been an SVC performance bottleneck.

1.3 Cluster Labeling Traditional SVC cluster labeling algorithms are Complete Graph (CG) [2], Support Vector Graph (SVG) [2], Proximity Graph (PG)

[7], and Gradient Descent (GD) [1]. All these algorithms group together data points by representing pairs of data points using an adjacency structure (typically a matrix). Each element records whether there is sufficient evidence to conclude that the associated pair of data points is in the same contour and therefore the same cluster. Each connected component in the adjacency matrix is regarded as a cluster. CG requires a $O(N^2)$ sized adjacency structure because it represents all data point pairs. SVG represents pairs in which one point is a support vector, so its adjacency structure only uses $O(N_{sv}N)$ space, where N_{sv} is the number of support vectors. PG forms an adjacency structure from a proximity graph that has only $O(N)$ edges. The GD method finds Stable Equilibrium Points (SEPs) that are the nearest minimum point (of Eq. 1.4 below) for each data point and then tests pairs of SEPs. GD's adjacency structure therefore uses $O(N_{sep}^2)$ space, where N_{sep} is the number of SEPs.

$$
\begin{aligned}
R^2(x) &= \|\Phi(x) - a\|^2 \\
(1.4) \quad &= 1 - 2\sum_j \beta_j K(x_j, x) + \sum_{i,j} \beta_i \beta_j K(x_i, x_j),
\end{aligned}
$$

where $x \in \mathcal{R}^d$, β_i is the Lagrange multiplier for x_i, and $a = \sum_i \beta_i \Phi(x_i)$ is the center of the minimal hypersphere [2].

Deciding if a pair of data points x_i and x_j is in the same contour is problematic because data space contours cannot be explicitly constructed due to the nature of the feature space mapping Φ. The aforementioned algorithms therefore use an indirect strategy that relies on the fact that every path connecting two data points in different data space contours exits the contours in data space and the image of the path in feature space exits the minimal hypersphere. The algorithms use, as a path in data space, the straight line segment $\overline{x_i x_j}$ connecting x_i and x_j. The line segment $\overline{x_i x_j}$ is sampled. For sample point x' in data space, $\Phi(x')$ is outside the minimal hypersphere if $R^2(x') > R^2$, where $R^2(x')$ is defined by Eq. 1.4. If the image of every sample point along $\overline{x_i x_j}$ is inside the minimal hypersphere, then x_i and x_j are determined to be in the same contour (hence cluster).

Unfortunately, the sampling approach creates a running time versus accuracy tradeoff. If m is the number of sample points along a line segment, then solving Eq. 1.4 for each sample point introduces a multiplicative factor of mN_{sv} beyond the time proportional to the size of the adjacency structure. The algorithms limit m to be a small constant (typically 10 to 20) in order to limit running time. However, small values of m can also cause two types of errors, false positive and false negative, as shown in Figure 1.
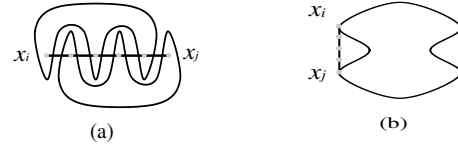

(a)   (b)

Figure 1: Problems in using line segment $\overline{x_i x_j}$, depicted in data space: (a) sample points on $\overline{x_i x_j}$ are all inside the minimal hypersphere although x_i and x_j are in different contours; (b) all sample points are outside the minimal hypersphere although x_i and x_j are in the same contour.

1.4   Overview This paper avoids the cluster labeling problems listed in Section 1.3 by using a novel approach that decides if two data points are in the same cluster without sampling a path between them. The main idea of this paper is to try to cover a key portion of the minimal hypersphere in feature space using cones that are anchored at each support vector in the feature space and also correspond to hyperspheres in data space. The union of the hyperspheres forms an approximate covering of the data space contours. The union need not be constructed; pairs of support vectors can be quickly tested during the cluster labeling process and then the remaining data points can be easily clustered. The algorithm[1], presented in Section 2, does not use sample points. It works for data sets of arbitrary dimension, and has been tested for up to 200 dimensions. The results in Section 3 show that our new algorithm is faster than traditional cluster labeling algorithms.

2   Cone Cluster Labeling

2.1   High-Level Approach We provide a new cluster labeling algorithm that is quite different from traditional methods. For a given kernel width value, our method does not sample a path between two data points in order to judge if they belong in the same cluster. Instead, we leverage the geometry of the feature space to help perform cluster labeling in the data space. First we find an approximate covering for the minimal hypersphere in feature space. This is described in Section 2.2. Strictly speaking, the covering is not for the minimal hypersphere but for the intersection P between the surface of the unit ball and the minimal hypersphere.

The approximate covering consists of a union of cone-shaped regions. One region is associated with each support vector's feature space image. Let $V = \{v_i | v_i \text{ is a support vector}, 1 \le i \le N_{sv}\}$ be the set of SVs for a given q value. The region for support vector v_i is called a support vector cone and is denoted by $\mathcal{E}_{v_i}$.

---

[1]An earlier version of this work appears in [14].

We call our algorithm 'Cone Cluster Labeling' (CCL) because of its reliance on cones.

Let B be the surface of the feature space unit ball. A hypersphere S_{v_i} centered on v_i in the data space maps to a subset of $\mathcal{E}_{v_i} \cap B$. Section 2.3 derives the radius of $\mathcal{S}_{v_i}$, which is shown to be the same for all support vectors. Having only one radius contributes to the speed of CCL. The union $\cup_i(\mathcal{S}_{v_i})$ is an approximate covering of the data space contours P', where $\Phi(P') \subseteq P$. However, the union is not explicitly constructed. Rather, cluster labeling is done in two phases, as described in Section 2.4. The first phase clusters support vectors while the second clusters the remaining data points. We regard two support vectors v_i and v_j as connected if their hyperspheres overlap: $(\mathcal{S}_{v_i} \cap \mathcal{S}_{v_j}) \neq \emptyset$. Forming the transitive closure of the connected relation yields a set of support vector clusters. The final step is a clustering for data points that are not support vectors. For each such data point, we assign the closest support vector's cluster label. Proofs are in [14].

2.2 Approximate Covering in Feature Space
This section forms a collection of support vector cones that approximately cover P. Let $\Theta_i = \angle(\Phi(v_i)Oa)$, where O is the feature space origin and a is the center of the minimal hypersphere (see Figure 2(a)). In feature space, each support vector has its own cone $\mathcal{E}_{v_i}$ that covers a part of P. We define the support vector cone $\mathcal{E}_{v_i}$ to be the infinite cone with axis $\overrightarrow{\Phi(v_i)}$ and base angle Θ_i. Lemma 2.1 below shows that $\Theta_i = \Theta_j = \Theta$ for all $v_i, v_j \in V$.

LEMMA 2.1. $\angle(\Phi(v_i)Oa) = \angle(\Phi(v_j)Oa), \forall v_i, v_j \in V$.

To approximately cover P, we denote by a' the intersection of $\overrightarrow{a}$ with the surface of the unit ball (see Figure 2(b)). The point a' is a common point of intersection for all the support vector cones. Thus, $((\cup_i(\mathcal{E}_{v_i})) \cap P) \approx P$.

2.3 Approximate Covering in Data Space
To approximately cover P' in data space using support vector cones, we find a hypersphere $\mathcal{S}_{v_i}$ in data space associated with each $\mathcal{E}_{v_i}$ in feature space. Since all support vector cones have the same base angle Θ, all $\mathcal{S}_{v_i}$ have the same radius and each is centered at v_i.

LEMMA 2.2. *Each data point whose image is inside $(\mathcal{E}_{v_i} \cap P)$ is at distance $\leq \|v_i - g_i\|$ from v_i, where $g_i \in \mathcal{R}^d$ is such that $\angle(\Phi(v_i)O\Phi(g_i)) = \Theta$.*

The claim implies that $(\mathcal{E}_{v_i} \cap P)$ corresponds to a hypersphere $\mathcal{S}_{v_i}$ in the data space centered at v_i with radius $\|v_i - g_i\|$. Since $(\cup_i(\mathcal{E}_{v_i})) \cap P$ approximately covers P, $\cup_i(\mathcal{S}_{v_i})$ approximately covers P'.

The next task is to obtain $\|v_i - g_i\|$ in data space. Because $\|\Phi(v_i)\| = 1 = \|a'\|$, $cos(\Theta) = cos(\angle(\Phi(v_i)O\Phi(g_i))) = cos(\angle(\Phi(v_i)Oa')) = \langle\Phi(v_i) \cdot a'\rangle$. Thus, we can solve for $\|v_i - g_i\|$ as follows:

$$(2.5) \qquad \|v_i - g_i\| = \sqrt{-\frac{ln(cos(\Theta))}{q}}.$$

Note that because $\Theta = \angle(\Phi(v_i)Oa')$ is the same for all $v_i \in V$, all $\mathcal{S}_{v_i}$ have the same radii. We therefore denote $\|v_i - g_i\|$ by Z. We now need to obtain $\langle\Phi(v_i)\cdot a'\rangle$ in order to calculate $cos(\Theta)$. To do this, we first show in Lemma 2.3 that $\langle\Phi(v_i)\cdot a'\rangle = \|a\|$, $\forall v_i \in V$ (see Figure 2 (c)). We then show that $\|a\| = \sqrt{1 - R^2}$ (see Figure 2 (d)).

LEMMA 2.3. $\langle\Phi(v_i) \cdot a'\rangle = \|a\|, \forall v_i \in V$.

COROLLARY 2.1. $\overrightarrow{\Phi(v_i)a}$ is orthogonal to $\overrightarrow{a}$.

LEMMA 2.4. $\langle\Phi(v_i) \cdot a\rangle = 1 - R^2 = \|a\|^2, \forall v_i \in V$.

Consequently, we have:

$$(2.6) \qquad Z = \sqrt{-\frac{ln(\sqrt{1 - R^2})}{q}}.$$

2.4 Assign Cluster Labels
Table 1 below shows the CCL algorithm. For the given q value, it first computes Z using Eq. 2.6. Next, support vectors are clustered by finding connected components in the resulting adjacency structure. Connected components can be efficiently found using a standard algorithm such as Depth First Search. Each connected component corresponds to a cluster. Therefore, the output of FindConnComponents() is an array of size N that contains cluster labels. Finally, the remaining data points are clustered. Each is assigned the cluster label of the nearest support vector.

```
CCL(X,q,V)
    compute Z for q using Eq. 2.6
    AdjacencyMatrix ← Construct connectivity matrix
    // assign cluster labels to the support vectors
    Labels ← FindConnComponents(AdjacencyMatrix)
    // assign cluster label to the rest of data
    for each x ∈ X, where x ∉ V
        idx ← find the nearest SV from x
        Labels[x] ← Labels[x_{idx}]
    end for
    return Labels
```
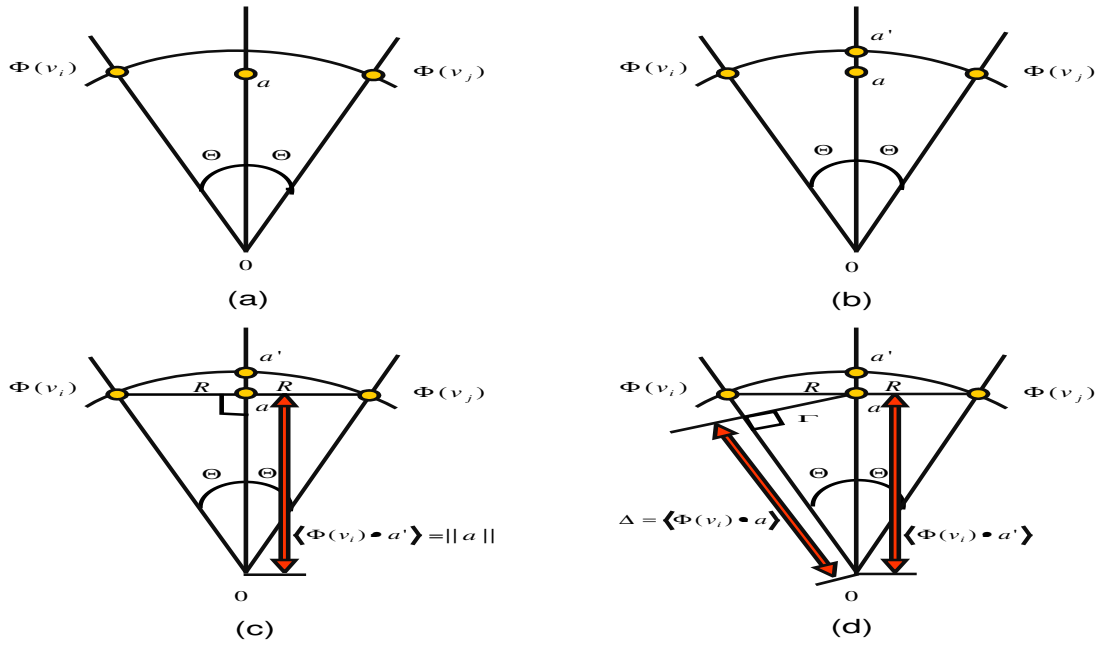

Table 1: Main algorithm of Cone Cluster Labeling

Figure 2: Developing an approximate cover of part of P, where v_i and v_j are support vectors and $\Phi(v_i)$ and $\Phi(v_j)$ are their feature space images, respectively. (a) $\Theta = \angle(\Phi(v_i)Oa)= \angle(\Phi(v_j)Oa)$. (b) a' is intersection of $\overrightarrow{a}$ with the surface of the unit ball. (c) the length $\|a\|$ is $< \Phi(v_i) \cdot a' >$. (d) $\Delta = < \Phi(v_i) \cdot a > = 1 - R^2 = \|a\|^2$. Note this is only a two-dimensional illustration; the actual feature space is high-dimensional.

The worst-case asymptotic running time complexity for FindConnComponents() is in $O(N_{sv}^2)$. The time complexity of the CCL **for** loop is in $O((N - N_{sv})N_{sv})$. Therefore, this algorithm uses time in $O(NN_{sv})$ for each q value. Note that, unlike previous cluster labeling algorithms, this time does not depend on a number of sample points. Detailed execution time comparisons are given in Section 3.

Figure 3: Left: dataset (N=98), right: cluster result (2 clusters). Data is a subset of data from the authors of [10].

3 Results

CCL provides high-quality clustering using less computation time than existing cluster labeling algorithms.

We compare three different types of execution times as well as total time of CCL with respect to 4 cluster labeling approaches introduced in Section 1.3: CG, SVG, PG, and GD. Times are for 1) preprocessing and constructing the adjacency matrix, 2) finding connected components from the adjacency matrix (this clusters some data points), and 3) clustering remaining data points. Worst-case asymptotic time complexity and actual running time comparisons are given in Table 2 with the data set of Figure 3.

The total of the three actual execution times is mea-

sured and divided by the number of q values[2] to compute average times. Since a cluster labeling algorithm receives q as an input and produces a cluster result, the average time is appropriate as a measurement for execution time comparison. To test high-dimensional datasets without outliers or strongly overlapping clusters, we created datasets with different dimensions. CCL worked well with these high-dimensional data. Details of these and all of our two and three-dimensional experiments can be found at [14].

---

[2]A list of q values is generated by our method in [13].

	CG	SVG	PG	GD	CCL
Adjacency Matrix Size	$O(N^2)$	$O(N_{sv}N)$	$O(N^2)$	$O(N_{sep}^2)$	$O(N_{sv}^2)$
Preprocessing			$O(NlogN)$	$O(mN^2k)$	$O(1)$
Adjacency	$O(mN^2N_{sv})$	$O(mNN_{sv}^2)$	$O(mNN_{sv})$	$O(mN_{sep}^2N_{sv})$	$O(N_{sv}^2)$
Connected Component	$O(N^2)$	$O(N_{sv}N)$	$O(N^2)$	$O(N_{sep}^2)$	$O(N_{sv}^2)$
Remaining Clustering			$O((N-N_{sv})N_{sv})$	$O(N-N_{sep})$	$O((N-N_{sv})N_{sv})$
Total Asymptotic Time	$O(mN^2N_{sv})$	$O(mNN_{sv}^2)$	$O(N^2+mNN_{sv})$	$O(mN^2(k+N_{sv}))$	$O(NN_{sv})$
Preprocessing	0	0	0.04	50.74	0.54
Adjacency	32.51	24.78	6.43	63.13	0.16
Connected Component	1.13	0.22	0.04	0.03	0.0005
Remaining Clustering	0	0	0.002	0.003	0.001
Total Execution Time	33.64	25.00	6.52	113.90	0.702

Table 2: Worst-case asymptotic running times for a single q value and average execution times for 22 q values in seconds, where k is the number of iterations for GD to converge to a SEP.

4 Conclusion and Future Work

Existing SVC cluster labeling algorithms, such as CG, SVG, PG, and GD, sample a line segment to decide whether a pair of data points is in the same cluster. This creates a tradeoff between cluster labeling time and clustering quality. Our Cone Cluster Labeling method uses a novel covering approach that avoids sampling. Using the geometry of the feature space, we find an approximate covering for the intersection of the minimal hypersphere and the surface of the unit ball. This maps to an approximate cover of the contours in data space. The cover uses hyperspheres in data space, centered on support vectors. Without constructing the union of these hyperspheres, data points are clustered quickly and effectively.

Cone Cluster Labeling quickly produces high-quality clusterings. Our experiments suggest that it operates well even in high dimensions. Future work will seek tighter and more complete coverage in the feature space and data space. We would also like to reduce the size of the kernel width value at which this method produces its best clustering.

References

[1] J. Lee and D. Lee, *An Improved Cluster Labeling Method for Support Vector Clustering*, IEEE Transactions on pattern analysis and machine intelligence, 27 (2005), pp. 461–464.

[2] A. Ben-Hur, D. Horn, H. Siegelmann, and V. Vapnik, *Support vector clustering*, Journal of Machine Learning Research 2, 2001, pp. 125–137.

[3] B. Schölkopf, O. Burges, and V. Vapnik, *Extracting support data for a given task*, First International Conference on Knowledge Discovery and Data Mining, 1995, pp. 252–257.

[4] C. J. Burges, *A Tutorial on Support Vector Machines for Pattern Recognition*, Data Mining and Knowledge Discovery, 2 (1998), pp. 121–167.

[5] E. Hartuv and R. Shamir, *A clustering algorithm based on graph connectivity*, Information Processing Letters, 76 (2000), pp. 175–181.

[6] A. Jain, N. Murty, and P. Flynn, *Data Clustering: A Review*, ACMCS, 31 (1999), pp. 264–323.

[7] J. Yang, V. Estivill-Castro, and S. Chalup, *Support Vector Clustering Through Proximity Graph Modeling*, Proceedings, 9th International Conference on Neural Information Processing (ICONIP'02), 2002, pp. 898–903.

[8] M. Ester, H. Kriegel, J. Sander, and X. Xu, *A density-based algorithm for discovering clusters in large spatial databases with noise*, Proc. 2nd Int. Conf. Knowledge Discovery and Data Mining(KDD'96), Portland, OR, 1996, pp. 226–231.

[9] I. Jonyer, L. Holder, and D. Cook, *Graph-Based Hierarchical Conceptual Clustering*, International Journal on Artificial Intelligence Tools, 10 (2001), pp. 107–135.

[10] D. Harel and Y. Koren, *Clustering spatial data using random walks*, Knowledge Discovery and Data Mining (KDD'01), 2001, pp. 281–286.

[11] V. Estivill-Castro and I. Lee, *Automatic Clustering via Boundary Extraction for Mining Massive Point-data sets*, In Proceedings of the 5th International Conference on Geocomputation, 2000.

[12] C. Blake and C. Merz, *UCI repository of machine learning databases*, 1998.

[13] S. Lee and K. Daniels, *Gaussian Kernel Width Generator for Support Vector Clustering*, International Conference on Bioinformatics and its Applications, 2004, pp. 151–162.

[14] S. Lee, *Gaussian Kernel Width Selection and Fast Cluster Labeling for Support Vector Clustering*, Doctoral Thesis and Tech. Report 2005-009, Department of Computer Science, University of Massachusetts Lowell, 2005.