Hierarchical cluster kernels for supervised and semi-supervised learning

Zalán Bodó
Department of Mathematics and Computer Science
Babeş–Bolyai University
RO-400084, Cluj-Napoca, Romania
zbodo@cs.ubbcluj.ro

Abstract

Semi-supervised learning became an important subdomain of machine learning in the last years. These methods try to exploit the information provided by the large and easily gathered unlabeled data besides the labeled training set. Analogously, many semi-supervised kernels appeared which determine similarity in feature space considering also the unlabeled data points. In this paper we propose a novel kernel construction algorithm for supervised and semi-supervised learning, which actually constitutes a general frame of semi-supervised kernel construction. The technique is based on the cluster assumption: we cluster the labeled and unlabeled data by an agglomerative clustering technique, and then we use the linkage distances induced by the clustering hierarchy to construct our kernel. The hierarchical cluster kernel is then compared to other existing techniques and evaluated on synthetic and real data sets.

1. Introduction

In the last years semi-supervised learning obtained considerable attention in machine learning community, and most of the supervised methods used in various applications were shifted to their semi-supervised version. Semi-supervision brought the ease of including a huge amount of unlabeled data – usually gathered from the internet – eliminating the costly step of human labeling. The unlabeled data helps to obtain a better approximation of the decision function. For example, suppose we are dealing with textual data, and that the word ”professor” turns out to be a good predictor for the positive examples based on the labeled data. Then if the unlabeled data shows that the words ”professor” and ”university” are usually correlated, then using both of these words better predictions can be made.

In this paper we propose a kernel construction technique based on hierarchical clustering, using the cluster and manifold assumption of semi-supervised learning, and test some of these kernels on synthetic and real data.

The paper is structured as follows: Section 2 introduces semi-supervised learning and kernels used in machine learning. Section 3 introduces hierarchical clustering, ultrametric matrices and trees, ISOMAP kernel, and finally presents the proposed kernel. In Section 4 the experiment results on the *two-moons* and the *USPS* data set are shown, and the last section, Section 5 concludes the paper.

Throughout the paper by using the expressions “supervised” and “semi-supervised learning” we refer to supervised and semi-supervised classification, respectively.

2. Semi-supervised learning and kernels

In semi-supervised learning the training data is divided into two parts: labeled and unlabeled data, $\mathcal{D} = \{(\mathbf{x}_1, y_1), \ldots, (\mathbf{x}_\ell, y_\ell), \mathbf{x}_{\ell+1}, \ldots, \mathbf{x}_{N=\ell+u}\}$, $\mathbf{x}_i \in \mathbb{R}^n$, $i = 1, \ldots, N$, $y_j \in \{-1, +1\}$, $j = 1, \ldots, \ell$. Usually $\ell \ll u$, because of the complex and time consuming labeling, which involves precise human decision-making, and since gathering huge amounts of unlabeled data from various sources is quite a simple task.

One of the simplest and commonly used techniques is self-training, where the classifier is trained on the labeled data, and then the learned model is used to predict the unknown labels of the yet unlabeled data. Using only the most confident predictions, the classifier is retrained, and the whole process is repeated several times. Another simple method is to cluster the labeled and unlabeled data first, then try to label the clusters, that is determine the cluster–class relations. In most cases there are no one-to-one connections between the clusters and the classes. Our kernel construction method is somewhat similar to this approach, its main step being a hierarchical clustering method.

For a comprehensive survey of the semi-supervised literature and recent advances in semi-supervised learning see [16] and [5].

In order to advantageously use the unlabeled data some assumptions have to be made. There are three main assumptions commonly used in semi-supervised learning: the smoothness, the cluster and the manifold assumption [5]. By the smoothness assumption, if two points are close in a high density region, then their labels should be similar. The cluster assumption says that if two points are in the same cluster, then they are likely to be in the same class, that is the decision boundary should lie in a low density region. The simple method mentioned above, which first clusters the whole data set, then assigns the classes, clearly uses the cluster assumption. The manifold assumption presumes that the data lie on a low-dimensional manifold. It is basically an assumption regarding dimensionality but, if one considers the manifold as the approximation of the high-dimensional region, then it is equivalent to the smoothness assumption.

Kernels were proposed for learning non-linear decision boundaries in 1964 by Aizerman, Braverman and Rozonoer, however they became famous and widely studied and applied in the nineties with support vector machines by the famous article of Boser, Guyon and Vapnik. Kernel functions can be seen as two-dimensional symmetric functions, which return the "similarity" of two points in some high dimensional space, without actually mapping the points to that space. More precisely, kernel functions return the dot product of two vectors in that space,

$$k(\mathbf{x}, \mathbf{z}) = \phi(\mathbf{x})' \cdot \phi(\mathbf{z})$$

Thus using an algorithm where the training instances appear only in forms of dot product, without modifying the algorithm itself, the same method can be used for learning non-linear decision functions by simply using the kernel function instead of the inner product of the vectors. The "kernel trick" says that in any algorithm in which the data points appear as dot products, the dot product is interchangeable with any positive semi-definite kernel. The matrix constructed using the kernel function and the data points is called the kernel matrix or the Gram matrix. A kernel function is said to be positive semi-definite if the induced Gram matrix is positive semi-definite. A matrix is called positive semi-definite if and only if all of its eigenvalues are non-negative.

Three of the widely used general-purpose kernels are the linear (dot product, inner product, etc.), polynomial, and the Gaussian (RBF) kernel functions:

$$k_{lin}(\mathbf{x}, \mathbf{z}) = \mathbf{x}'\mathbf{z}$$

$$k_{poly}(\mathbf{x}, \mathbf{z}) = (a\mathbf{x}'\mathbf{z} + b)^k$$

$$k_{rbf}(\mathbf{x}, \mathbf{z}) = \exp\left(-\frac{1}{2\sigma^2}\|\mathbf{x} - \mathbf{z}\|_2^2\right)$$

For a detailed analysis of these and other general-purpose kernels see the book [12].

Since semi-supervised learning obtained notable attention, semi-supervised kernels appeared which determine the similarities of the data points using also some information obtained from the whole data set including labeled and unlabeled points.

Our kernel construction method can be considered a generalization of the connectivity kernel introduced in [7]. We shortly describe and discuss the connectivity kernel in Section 4.

3. The hierarchical cluster kernel

3.1. Hierarchical clustering

Clustering means partitioning a set of points into "separate" groups or components called clusters. We call it a nested clustering if a partition is formed by merging components of the other; then we say that the latter one is nested into the first one. Hierarchical clustering consists in a sequence of partitions where every partition is nested into the next partition in the sequence. Agglomerative (bottom-up) methods start with as many clusters as points there are, and at every step the most similar pair of clusters is merged until one big cluster is obtained. Divisive (top-down) methods however start with the whole data set as a single cluster, and splits the best separable clusters recursively until the one-point clusters are reached. For a detailed introduction to hierarchical clustering see [11].

The general agglomerative clustering algorithm producing a binary tree is the following:

1. Define the initial clusters as the points themselves.
2. Find the most similar pair of clusters.
3. Merge them thus creating a new cluster.
4. Repeat from step 2. until a single cluster is obtained.

One can design a large variety of methods by simply choosing a different function for determining cluster similarity.

From the huge literature of hierarchical clustering we mention three widely known methods, namely the single linkage, the complete linkage and average linkage clustering.

Single linkage clustering uses the following cluster distance function:

$$D(C_1, C_2) = \min\{d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_1, \mathbf{z} \in C_2\} \tag{1}$$

The complete linkage clustering defines the distance between clusters as

$$D(C_1, C_2) = \max\{d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_1, \mathbf{z} \in C_2\} \tag{2}$$

In average linkage clustering the average of pointwise distances between all the elements of the two clusters is taken:

$$D(C_1, C_2) = \frac{1}{|C_1| \cdot |C_2|} \sum_{i=1}^{|C_1|} \sum_{j=1}^{|C_2|} d(\mathbf{x}_i, \mathbf{z}_j) \qquad (3)$$

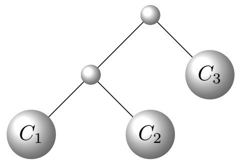

Figure 1. Merging 3 clusters.

We call the function $D(\cdot,\cdot)$ the *linkage distance*, while $d(\cdot,\cdot)$ is called the *pointwise distance*.

All the above-mentioned linkage functions possess the following property: suppose in two consecutive steps we merge three clusters, C_1, C_2 and C_3; first we merge C_1 with C_2 resulting in C_{12}, and then we merge it with C_3 (see Figure 1). That is if

$$\begin{aligned} D(C_1, C_2) &\leq D(C_1, C_3) \\ D(C_1, C_2) &\leq D(C_2, C_3) \end{aligned}$$

then

$$D(C_1, C_2) \leq D(C_{12}, C_3)$$

This can be proved as follows:

(1) We know that $D(C_1, C_2) \leq D(C_1, C_3)$ and $D(C_1, C_2) \leq D(C_2, C_3)$, which is equivalent to saying that $D(C_1, C_2) \leq \min\{d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_1, \mathbf{z} \in C_3\}$ and $D(C_1, C_2) \leq \min\{d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_2, \mathbf{z} \in C_3\}$. Consequently $D(C_1, C_2) \leq \min\{d(\mathbf{x}, \mathbf{z}) | \mathbf{x} \in C_1 \cup C_2, \mathbf{z} \in C_3\} = D(C_{12}, C_3)$.

(2) The proof is similar to the previous.

(3) It is simplest in this case if we prove it for three points: $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$. Suppose (i) $D(\mathbf{x}_1, \mathbf{x}_2) \leq D(\mathbf{x}_1, \mathbf{x}_3)$ and (ii) $D(\mathbf{x}_1, \mathbf{x}_2) \leq D(\mathbf{x}_2, \mathbf{x}_3)$ which are equivalent to the same relations with exchanging $D(\cdot,\cdot)$ by $d(\cdot,\cdot)$. We have to prove that $D(\mathbf{x}_1, \mathbf{x}_2) \leq D(\mathbf{x}_{12}, \mathbf{x}_3)$ which is equivalent to $d(\mathbf{x}_1, \mathbf{x}_2) \leq (1/2)(d(\mathbf{x}_1, \mathbf{x}_3) + d(\mathbf{x}_2, \mathbf{x}_3))$. Summing up the corresponding sides of (i) and (ii), and then dividing by 2, we obtain the desired relation.

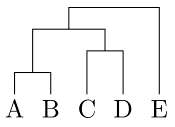

Figure 2. Dendrogram resulting from the clustering of 5 points: A,B,C,D and E.

The result of a hierarchical clustering method is called a dendrogram, which is actually a binary tree. On Figure 2 a dendrogram is shown. If there are N points to cluster, the resulting tree will have N leaves and $N-1$ internal nodes or vertices, including the root.

In order to produce a dendrogram, equivalently to perform hierarchical clustering one can use an arbitrary non-hierarchical method in a divisive manner: split the initial cluster of the points into two, and then apply the same method recursively to split the resulting clusters into smaller ones, until one-point clusters are reached.

3.2. Ultrametric matrices and trees

In this section we cite some definitions and theorems from different sources to introduce ultrametric trees and matrices.

Definition 1 ([14]) *An $N \times N$ matrix $\mathbf{M}$ is said to be ultrametric iff*

$$M_{ij} \leq \max\{M_{ik}, M_{jk}\}, \quad \forall i, j, k \in \{1, 2, \ldots, N\}$$

The following theorem establishes the connection between ultrametric matrices and trees.

Theorem 1 ([14]) *A distance matrix is ultrametric iff it can be realized by a unique ultrametric tree.*

Definition 2 ([8]) *A tree T is called ultrametric if the nodes are labeled with with some values such that any path from the root to a leaf results in a decreasing / non-increasing sequence.*

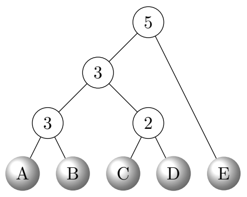

Figure 3. Example for an ultrametric tree.

On Figure 3 an ultrametric tree is shown with 5 leaves and 4 internal nodes. One can observe that every path – starting from the roof and leading to a leaf – produces a decreasing sequence. The values attached to the internal nodes can be considered as distances: for every pair of leaves, their distance is equal to the value attached to their lowest common subsumer. For example the distance between A and B is 3, between B and D is 3, and between C and E is 5. Moreover, this construction defines an 5×5 ultrametric matrix $\mathbf{M}$, with the property $M_{ij} \leq \max\{M_{ik}, M_{kj}\}$, $\forall i, j, k \in \{1, \ldots, 5\}$.

3.3. The ISOMAP kernel

In the second version of our cluster kernel we will use distances induced by the data graph of the labeled and unlabeled points. This will be accomplished by using the ISOMAP kernel [13], [10] which we shortly describe here. We also compare our kernel to the ISOMAP kernel.

The ISOMAP kernel is

$$\mathbf{K}_{\text{isomap}} = -(1/2)\mathbf{J}\mathbf{G}^{(2)}\mathbf{J}$$

where $\mathbf{G}^{(2)}$ contains the squared graph distances (shortest paths) and $\mathbf{J}$ is the centering matrix, $\mathbf{J} = \mathbf{I} - \frac{1}{N} \cdot \mathbf{1} \cdot \mathbf{1}'$, $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is the $N \times 1$ vector of 1's. If the points were centered at each dimension, then the ISOMAP kernel shown above is positive semi-definite. Unfortunately we do not know the representation of the points, so we are not able to center the data. But because only the large eigenvalues and the corresponding eigenvectors are important we proceed in the following way. The above kernel matrix is factorizable into $\mathbf{USU}'$, where $\mathbf{U}$ contains the eigenvectors, while the diagonal matrix $\mathbf{S}$ holds the eigenvalues of the decomposed matrix [9, p. 393]. Then the ISOMAP kernel we will use is $\mathbf{K}_{\text{isomap}} = \mathbf{U}\widetilde{\mathbf{S}}\mathbf{U}'$, where $\widetilde{\mathbf{S}}$ is the diagonal matrix of the eigenvalues, where each negative eigenvalue was set to zero.

Informally, the ISOMAP kernel maps the points to the space, where their pointwise distances equal to the shortest path distances on the data graph in the input space. If the points are centered at each dimension, then $-(1/2)\mathbf{J}\mathbf{G}^{(2)}\mathbf{J}$ is equal to the dot products of the vectors mapped to the above-mentioned space [2, p. 262].

3.4. The cluster kernel

For the construction of our cluster kernel we need the following theorems.

Theorem 2 ([7]) *For every ultrametric* $\mathbf{M}$*,* $\sqrt{\mathbf{M}}$ *is* ℓ_2 *embeddable.*

The euclidean embedding of vectors means that the dissimilarities contained in some matrix can be interpreted as euclidean distances. If this is possible, then it is easy to reconstruct the vectors from this distance matrix. For euclidean embeddings and multidimensional scaling methods see [2].

Theorem 3 ([7]) *For a metric* $\mathbf{M}$*,* $\sqrt{\mathbf{M}}$ *is* ℓ_2 *embeddable iff* $\mathbf{M}^c = \mathbf{JMJ}$ *is negative semi-definite.*

The matrix $\mathbf{J}$ used in the above theorem is called the *centering matrix*, and is defined as $\mathbf{J} = \mathbf{I} - \frac{1}{N} \cdot \mathbf{1} \cdot \mathbf{1}'$, where $\mathbf{I}$ is the identity matrix and $\mathbf{1}$ is the $N \times 1$ vector of 1's.

Theorem 4 ([7]) *Given an ultrametric* $\mathbf{M}$*, the matrix* $-\frac{1}{2}\mathbf{M}^c = -\frac{1}{2}\mathbf{JMJ}$ *is a Gram matrix containing the dot products of some vectors* $\mathbf{z}_i$*,* $i = 1, 2, \ldots, N$.

We construct the cluster kernel using linkage distances from hierarchical clustering. Thus we map the points to a feature space where the pointwise distances are equal to the cluster distances in the input space. The steps are the following:

1. Perform an agglomerative clustering on the labeled and unlabeled data – for example using one of the linkage functions from equations (1), (2) and (3).

2. Define matrix $\mathbf{M}$ with entries M_{ij} = linkage distance in the resulting ultrametric tree at the lowest common subsumer of i and j; $M_{ii} = 0, \forall i$.

3. Define the kernel matrix as $\mathbf{K} = -\frac{1}{2}\mathbf{JMJ}$.

One can ask why we call $\mathbf{K}$ a semi-supervised kernel. Semi-supervised methods use the information provided by the unlabeled data too. Here we use the unlabeled data in the clustering step to determine "better" pointwise distances, based on which we construct the kernel. We expect to obtain better similarities as if only the labeled training set was used. In order to compute the kernel function values for the test points, we include them into the unlabeled set. For every new test point, unavailable at training time the whole process of clustering must be performed again. In the next subsection we present a more efficient technique which can be used in these cases. However, in all the experiments we made, we considered the test set to be equal to the unlabeled data set.

The resulting kernel can be also used for embedding. Since $\mathbf{K}$ is positive semi-definite, by the spectral theorem [9, p. 393] it can be decomposed into $\mathbf{USU}'$, where $\mathbf{U}$ is a unitary matrix, i.e. $\mathbf{UU}' = \mathbf{I}$, containing the eigenvectors of $\mathbf{K}$, and $\mathbf{S}$ is a diagonal matrix containing the corresponding eigenvalues. Thus $\mathbf{Z} = \mathbf{U}\sqrt{\mathbf{S}}$ gives the low-dimensional representation of the vectors, and they are placed in the rows of $\mathbf{Z}$.

A slightly modified version of the algorithm uses also the manifold assumption of semi-supervised learning; the above algorithm is augmented with the following three preceding steps:

-2. Determine the k nearest neighbors or an ϵ-neighborhood of each point and take all the distances to other points equal to zero.

-1. Compute shortest paths for every pair of points – using for example Dijkstra's algorithm.

0. Use these distances in clustering for the pointwise distance $d(\cdot, \cdot)$ in equations (1), (2) and (3).

We use here the shortest path distances computed on the k nearest neighbor or the ϵ-neighborhood graph of the data, thus – if the data lie on a low-dimensional manifold – approximating pointwise distances on this manifold. For the proof of the previous statement see [1]. The very first technique using shortest path graph distances for dimensionality reduction was the famous ISOMAP algorithm [13]. We are going to compare our method to the ISOMAP kernel too.

However, the resulting graph is not necessarily connected; by choosing a small k or ϵ, the data graph could contain several connected components. We follow the idea described in [15] to obtain one connected component. Let $\mathbf{D}$ denote the pointwise distance matrix we obtain after sparsifying the neighborhood matrix by choosing the k nearest neighbors or an ϵ-neighborhood of each point. Similarly $\mathbf{G}$ denotes the all-pair shortest path matrix computed using $\mathbf{D}$. Then $G_{ij} = \infty$ for some i and j means that the graph contains more than one connected component. Then we choose those two unconnected points i and j, which has a minimal euclidean distance. We connect these points using a relatively large distance,

$$L_{ij} = g_{max} + \frac{d_{min}}{d_{max}} \tag{4}$$

where g_{max} denotes the maximal path in the whole graph, while d_{min} and d_{max} could be the minimal and maximal either euclidean or graph distances between the data points. When connecting these two points, only those distances must be updated in the distance matrix, whose value could not be calculated (∞). For every such pair of points k and ℓ we calculate the minimal distance as

$$G_{k\ell} = \min\{G_{ik} + L_{ij} + G_{j\ell}, G_{kj} + L_{ij} + G_{i\ell}\}$$

Unfortunately, after the insertion of such a connection, there could still remain unconnected components. In such a case the above procedure must be repeated until one connected component is obtained. For every such an iteration we increment L_{ij} by d_{min}/d_{max}.

3.5. New test points

The cluster kernel we have presented does not allow simple kernel calculations for new, unseen points, which were not available in the training phase together with the labeled and unlabeled data. This means that all the steps must be performed again whenever a new test point arrives, which is very inefficient. Hence we follow here the approach of [6], that is we represent each test point as a linear combination of the labeled and unlabeled data, thus to compute $k(\mathbf{x}, \mathbf{x}_j)$ we will need kernel function calculations only between labeled and unlabeled examples, which are already computed. First we solve the optimization problem

$$\operatorname*{argmin}_{\boldsymbol{\alpha}} \left\| \mathbf{x} - \sum_{i=1}^{\ell+u} \alpha_i \mathbf{x}_i \right\|^2$$

where we obtain the optimal coefficients $\boldsymbol{\alpha}^0 = (\mathbf{X}\mathbf{X}')^{-1}\mathbf{X}\mathbf{x} = \widetilde{\mathbf{K}}_{NN}^{-1}\widetilde{\mathbf{k}}_{\mathbf{x}}$ for a test point $\mathbf{x}$. Thus we can say that

$$\mathbf{x} \approx \sum_{i=1}^{N} \alpha_i^0 \mathbf{x}_i$$

Calculating the dot products of the feature-space vectors we obtain

$$\phi(\mathbf{x})'\phi(\mathbf{x}_j) = k(\mathbf{x}, \mathbf{x}_j) = \sum_{i=1}^{N} \alpha_i^0 k(\mathbf{x}_i, \mathbf{x}_j) = \widetilde{\mathbf{k}}_{\mathbf{x}}' \widetilde{\mathbf{K}}_{NN}^{-1} \mathbf{k}_{\mathbf{x}_j}$$

where $\widetilde{\mathbf{k}}_{\mathbf{x}} = [\widetilde{k}(\mathbf{x}_1, \mathbf{x}) \ \ldots \ \widetilde{k}(\mathbf{x}_N, \mathbf{x}))]'$, where $\widetilde{k}(\cdot,\cdot)$ denotes the dot product in the input space, while $\mathbf{k}_{\mathbf{x}_j} = [k(\mathbf{x}_1, \mathbf{x}_j) \ \ldots \ k(\mathbf{x}_N, \mathbf{x}_j))]'$, where $k(\cdot,\cdot)$ is our cluster kernel. We put subscripts to the kernel matrices, indicating their size.

Suppose we are given t test points. In kernel methods, for predicting the label of a new test point we need to perform kernel function calculations only between the test points and the (labeled) training points. Thus we will need the matrix $\mathbf{K}_{t\ell}$, which – by the above – is computed as

$$\mathbf{K}_{t\ell} = \widetilde{\mathbf{K}}_{tN} \widetilde{\mathbf{K}}_{NN}^{-1} \mathbf{K}_{N\ell}$$

$\mathbf{K}_{N\ell}$ is calculated from $\mathbf{K}_{NN}$ which is the kernel matrix of the labeled and unlabeled training points.

In the second case, where the graph-based distances are used, we actually first map the points to a feature space, and then we implicitly map them to another feature space by the cluster kernel. Thus the kernel function approximation proceeds in two steps:

$$\mathbf{K}_{tN}^1 = \widetilde{\mathbf{K}}_{tN} \widetilde{\mathbf{K}}_{NN}^{-1} \mathbf{K}_{N\ell}^1$$

and

$$\mathbf{K}_{t\ell}^2 = \mathbf{K}_{tN}^1 (\mathbf{K}_{NN}^1)^{-1} \mathbf{K}_{N\ell}^2$$

where $\mathbf{K}^1$ denotes the ISOMAP kernel, while $\mathbf{K}^2$ is our cluster kernel. Hence the needed kernel function values will be in the matrix $\mathbf{K}_{t\ell}^2$.

When the inverse of some matrices do not exist, we use the pseudoinverse instead.

4. Experiments

For using kernel methods, only the values of the kernel function are required: $k(\mathbf{x}_i, \mathbf{x}_j)$ for $i, j \in \{1, \ldots, \ell\}$ and $k(\mathbf{x}_i, \mathbf{x}_j)$ for $i \in \{\ell + 1, \ldots, N\}$, $j \in \{1, \ldots, \ell\}$. Thus we calculate the $N \times N$ Gram matrix for the whole data set using the presented techniques, thus we obtain all the above values. Here we consider the test set to be equal to the unlabeled data set.

In the experiments we tested our cluster kernel on a toy data set and on the modified USPS data set used in the benchmarks of the book [5].

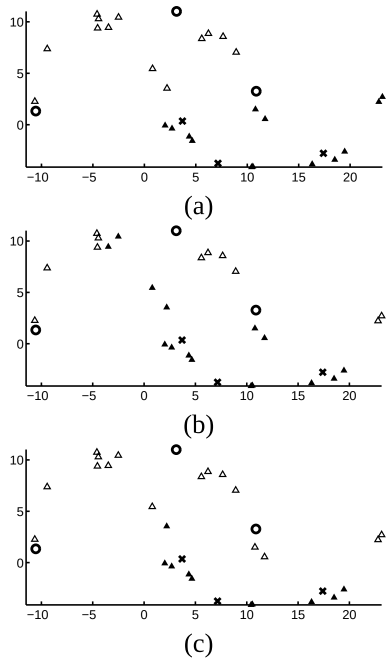

Figure 4. Learning the decision boundary for the two-moons data set with support vector machines: (a) linear kernel; (b) polynomial kernel with $a = b = 1$, $k = 3$; (c) RBF kernel with $\sigma^2 = 1$.

We used the proposed cluster kernel for classification. For classification we used support vector machines [12], [3]. The tests were performed using the LIBSVM package [4], version 2.85.

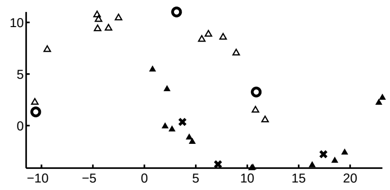

Figure 5. Learning decision boundary using support vector machines with the proposed cluster kernel with euclidean distances (no graph construction).

The toy data set is the two-moons data set, used for the testing and illustration of many semi-supervised algorithms. Our two-moons data set consists of a total of 33 points: 6 labeled points, 3 for each class, and 27 unlabeled points, 14 for the upper crescent and 13 for the lower one. On Figure 4 the separation of the classes is shown using only the labeled points for training the support vector machine. The best decision boundary is obtained using the 3rd order polynomial kernel. On Figure 5 the proposed hierarchical cluster kernel is applied using complete linkage clustering, and a perfect separation is obtained. The circles and crosses denote the training points from the upper and lower classes respectively, the empty triangles denote the points classified as belonging to the upper crescent, while full triangles represent the points classified into the lower crescent.

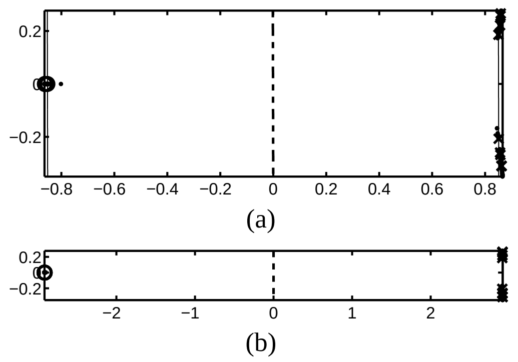

Figure 6. The low-dimensional representation and separation of the modified two-moons data set: (a) single linkage hierarchical clustering with euclidean distances; (b) single linkage clustering with graph distances, $k = 7$.

On Figure 6 the low-dimensional representation of a modified version of the two-moons data set is shown, and also the maximum margin hyperplane obtained with support vector machines, separating the classes: the circles and crosses denote the labeled, while the dots represent the unlabeled points; the midmost line shows the separating hyperplane, while the slimmer outside lines are the marginal hyperplanes. The modification consisted in adding more labeled and unlabeled points to the crescents: 8 and 9 points in the upper and lower crescent respectively, and 383 unlabeled points. We made this broadening to better emphasize the cluster and manifold structure of the data.

We also wanted to test our method on real data. For this end we used the modified USPS handwritten digit recognition data set, which – along 8 other data sets – is freely downloadable from the web page of the book [5]. These data sets contain $1\,500$ points, and 2×12 different splits is given for each set: 12 splits with 10 labeled points ($1\,490$

unlabeled) and also 12 splits with 100 labeled ($1\,400$ unlabeled) data points.

The modified USPS data set was constructed selecting 150 samples of each of the ten digits – stored as a 16×16 pixel image –, then the digits "2" and "5" were put in the positive class, while the remaining samples formed the negative class, thus the classification algorithm became imbalanced. In addition the data were obscured with the following algorithm (using $\sigma = 0.1$):

1. Randomly select and permute 241 features (dimensions).
2. Add to each feature a random bias from $N(0, 1)$.
3. Multiply each column by a uniform random value from $[-1, -0.5] \cup [0.5, 1]$.
4. Add independent noise from $N(\mathbf{0}, \sigma^2\mathbf{I})$ to each data point.

In every experiment we performed only the first split was used from the 12 available, that is our results are not averaged over the different splits. The evaluation measure used in the experiments is accuracy, that is the ratio of the number of correctly classified data and the size of the whole data set. The unlabeled data was used as the test data. Thus we calculated the training and test accuracy too.

%	linear	polynomial	RBF
USPS/10	100 ⋄ 72.82	90 ⋄ **83.69**	90 ⋄ 80.13
USPS/100	100 ⋄ 86.43	92 ⋄ 89.57	98 ⋄ **89.64**

Table 1. Accuracy results (in percentage) of experiments on the USPS data set with support vector machines using only the labeled points for training; the unlabeled part is used only for testing. The columns denote the type of the kernel we used.

During the experiments for connecting unconnected graph parts in (4) we used the ratio of euclidean distances. Also we set the k-neighborhood of the points to 7. USPS/10 and USPS/100 denotes the type of the data set: having 10 labeled and $1\,490$ unlabeled, and similarly 100 labeled and $1\,400$ unlabeled points.

The diamonds between the results separate training and test accuracies.

On Table 1 the accuracy results are shown, treating the problem as a supervised learning task: the training data consists of the labeled points, while the unlabeled points constitute the test data. These experiments were performed to show that semi-supervised methods can help to find a better separation. Although we did not make any cross validation, we found the following "best" parameters by choosing some values randomly, and testing the learned model: polynomial: $a = 0.0041$, $b = 1$, $k = 3$; RBF: $\sigma = 4.082$.

%	single	complete	average
USPS/10 euclidean	100 ⋄ 80.07	100 ⋄ 82.01	100 ⋄ 81.48
USPS/10 graph	100 ⋄ 80.07	100 ⋄ 88.26	100 ⋄ **89.26**
USPS/100 euclidean	100 ⋄ 81.79	100 ⋄ 89.50	100 ⋄ 92.86
USPS/100 graph	100 ⋄ 81.79	100 ⋄ **95.64**	100 ⋄ **95.64**

Table 2. Accuracy results (in percentage) of experiments on the USPS data set using support vector machines with our hierarchical cluster kernel.

%	ISOMAP kernel
USPS/10	100 ⋄ **85.10**
USPS/100	100 ⋄ **86.71**

Table 3. Accuracy results (in percentage) of experiments on the USPS data set using support vector machines with the ISOMAP kernel.

On Table 2 the results obtained using our kernel are shown. The columns indicate the type of hierarchical clustering used: single, complete and average linkage. As the table shows, we tested our kernel using euclidean as well as graph distances.

On Table 3 the accuracies obtained using ISOMAP kernel are shown.

Finally we compare our kernels to the connectivity kernel [7]. The proposed hierarchical cluster kernel is actually a generalization of the connectivity kernel. The connectivity kernel calculates the path-specific effective dissimilarities as

$$M_{ij} = \min_{p \in P_{ij}} \left\{ \max_{1 \le k \le |p|-1} d(p[k], p[k+1]) \right\}$$

where p denotes a path from all the existing paths P_{ij} between nodes i and j, and $p[k]$ denotes the kth node on this path. The authors prove that this metric can be approximated using the complete input graph of pairwise (euclidean) distances, and applying a single linkage hierarchical clustering on it. Thus the connectivity kernel results can be read off the table (Table 2) by considering the single linkage clustering results using euclidean distances.

In all the above tables the best test results obtained for the two data sets (10 and 100 labeled points) are formatted with boldface.

5. Conclusions

In this paper we proposed the hierarchical cluster kernel for supervised and semi-supervised learning. Our cluster kernel calculates similarities in feature space by taking distances between data points as distances between the clusters they belong to. If the cluster (and manifold) assumption holds, the cluster kernel can produce significant improvement in finding a better decision function. For its effectiveness compare the results from tables 1, 2 and 3. The proposed kernel can be used for supervised learning and dimensionality reduction too, however, we expect to be more efficient in semi-supervised settings. Similarly to a few semi-supervised methods, e.g. label propagation, all the points are needed, and the test points are put into the unlabeled set.

The cluster kernel can be considered similar to the connectivity kernel proposed in [7], however the freedom in choosing the linkage function makes it a more flexible technique for constructing kernels for different data settings.

The proposed hierarchical cluster kernel offers a frame for constructing different cluster kernels, by using different linkage functions and clustering techniques. Non-hierarchical methods can be used as well in kernel construction, by applying divisive clustering and using an arbitrary non-hierarchical method. The single condition that has to be fulfilled by the clustering method is the following relation regarding cluster dissimilarities: if $D(C_1, C_2) \leq D(C_1, C_3)$ and $D(C_1, C_2) \leq D(C_2, C_3)$ then $D(C_1, C_2) \leq D(C_{12}, C_3)$.

As for further research on hierarchical cluster kernels we would like to construct more powerful kernels and compare their efficiency using different approaches, e.g. kernel alignment. We also want to compare the produced Gram matrices with included test data and with test data available only after training. Furthermore, we would like to empirically compare our kernel with other cluster kernels.

Acknowledgement

We acknowledge the support of the grant CNCSIS/TD-35, nr. 485/1.10.2007 by the Romanian Ministry of Education and Research. We also thank reviewers for their useful comments and suggestions.

References

[1] M. Bernstein, V. de Silva, J. C. Langford, and J. B. Tenenbaum. Graph approximations to geodesics on embedded manifolds. Technical report, 2000.

[2] I. Borg and P. J. F. Groenen. *Modern multidimensional scaling, 2nd edition*. Springer-Verlag, New York, 2005.

[3] C. J. C. Burges. A tutorial on support vector machines for pattern recognition. In *Knowledge Discovery and Data Mining*, volume 2, pages 121–167. 1998.

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

[5] O. Chapelle, B. Schölkopf, and A. Zien. *Semi-Supervised Learning*. MIT Press, Sept. 2006. Web page: http://www.kyb.tuebingen.mpg.de/ssl-book/.

[6] O. Chapelle, J. Weston, and B. Schölkopf. Cluster kernels for semi-supervised learning. In S. Becker, S. Thrun, and K. Obermayer, editors, *NIPS*, pages 585–592. MIT Press, 2002.

[7] B. Fischer, V. Roth, and J. M. Buhmann. Clustering with the connectivity kernel. In S. Thrun, L. K. Saul, and B. Schölkopf, editors, *NIPS*. MIT Press, 2003.

[8] I. P. Gent, P. Prosser, B. M. Smith, and W. Wei. Supertree construction with constraint programming. In *ICCP: International Conference on Constraint Programming (CP), LNCS*, 2003.

[9] G. H. Golub and C. F. Van Loan. *Matrix Computations, 3nd Edition*. The Johns Hopkins University Press, Baltimore, MD, 1996.

[10] J. Ham, D. D. Lee, S. Mika, and B. Schölkopf. A kernel view of the dimensionality reduction of manifolds. In C. E. Brodley, editor, *ICML*, volume 69 of *ACM International Conference Proceeding Series*. ACM, 2004.

[11] A. K. Jain and R. C. Dubes. *Algorithms for Clustering Data*. Prentice-Hall, 1988.

[12] B. Schölkopf and A. J. Smola. *Learning with Kernels*. The MIT Press, Cambridge, MA, 2002.

[13] J. B. Tenenbaum, V. de Silva, and J. C. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, Dec. 2000.

[14] B. Y. Wu and K.-M. Chao. *Spanning Trees and Optimization Problems*. Chapman and Hall/CRC, Boca Raton, Florida, 2004.

[15] Q. Yong and Y. Jie. Geodesic distance for support vector machines. *Acta Automatica Sinica*, 31(2):202–208, 2005.

[16] X. Zhu. Semi-supervised learning literature survey. Technical Report 1530, Computer Sciences, University of Wisconsin-Madison, 2007.