Large-Scale Clustering through Functional Embedding

Frédéric Ratle[1], Jason Weston[2], and Matthew L. Miller[2]

[1] IGAR, University of Lausanne, Amphipôle, 1015 Lausanne, Switzerland
frederic.ratle@unil.ch
[2] NEC Labs America, 4 Independence Way, Princeton NJ, USA

Abstract. We present a new framework for large-scale data clustering. The main idea is to modify functional dimensionality reduction techniques to directly optimize over discrete labels using stochastic gradient descent. Compared to methods like spectral clustering our approach solves a single optimization problem, rather than an ad-hoc two-stage optimization approach, does not require a matrix inversion, can easily encode prior knowledge in the set of implementable functions, and does not have an "out-of-sample" problem. Experimental results on both artificial and real-world datasets show the usefulness of our approach.

1 Introduction

Clustering, which aims at identifying groups in the data in an unsupervised manner, is one of the main tasks in the field of machine learning, and different approaches to tackle this problem have been popularized over the last decades, among them: k-means and hierarchical clustering, spectral clustering and its variants (e.g., [20,24]), support vector and maximum margin clustering [5,30,31].

Clustering and dimensionality reduction algorithms have, in recent years, grown to become a single field of study through spectral embedding methods. Indeed, both tasks aim at producing a compact and visually meaningful representation of data, and strong links between methods to achieve both tasks have been highlighted. For example, methods based on the analysis of the graph Laplacian of a similarity matrix have provided a common framework to perform clustering and embedding using the top eigenvectors of the Laplacian [3]. Typically methods like spectral clustering, however, require a two-stage approach of first embedding and then using k-means to cluster data points in the found embedding space.

In this paper we aim at providing a framework for data clustering based on large-scale direct optimization by stochastic gradient descent. We describe a general class of objective functions that are the analogue of nonlinear embedding pairwise loss functions, but using loss functions that are directly related to the task of clustering. After a review of existing embedding algorithms for clustering, we present an instance of our approach, the Ncut embedding (NCutEmb thereafter) method. We first describe a comparison of explicit and functional embedding methods in order to highlight the benefits of using a function-based approach. Experiments with NCutEmb clustering clearly show that we can achieve

W. Daelemans et al. (Eds.): ECML PKDD 2008, Part II, LNAI 5212, pp. 266–281, 2008.

error rates inferior or *at most* similar to those obtained with deterministic batch methods using stochastic gradient descent, while obtaining at the same time a model for new examples as they become available, thus avoiding the so-called "out-of-sample" problem.

2 Existing Methods: Embedding and Clustering

The problem of learning the underlying data structure from unlabeled examples has been intensively studied with both functional and explicit (point-based) methods. This problem is usually referred to as manifold learning or dimensionality reduction in the machine learning literature. Self-organizing maps and autoassociator networks are classical examples of function-based embedding methods. While these methods have traditionally been based on various heuristics, recent work [4,13,14] has shown that functional methods can benefit from the theoretical understanding of dimensionality reduction made since the nineties, and offer several practical advantages over explicit methods.

2.1 Point-Based Methods

Point-based, or explicit, methods are most commonly rooted either in principal component analysis or multidimensional scaling, two well-known linear embedding methods, which provide them a sound mathematical background. They provide a point-to-point correspondence between the input space and the intrinsic space in which the data lie. Explicit methods have been multiplied in recent years, and include the following methods and their variants: kernel PCA [23], Isomap [25], Locally Linear Embedding [21], Maximum Variance Unfolding [28] and Laplacian Eigenmaps [3]. Thorough reviews of this family of methods can be found in [19,22].

Laplacian Eigenmaps and Spectral Clustering. We will briefly describe here Laplacian Eigenmaps (LE), as it is central to the remainder of the paper. The idea behind LE is to map nearby inputs to nearby outputs, hence preserving the neighborhood relations between data points. This preservation of neighborhood structure renders the method insensitive to outliers, thus making it appropriate for clustering. LE finds the embedding that minimizes the following loss function:

$$\sum_{ij} L\left(f_i, f_j, W_{ij}\right) = \sum_{ij} W_{ij} ||f_i - f_j||^2 \tag{1}$$

where $f_k \in \mathbb{R}^d$ is the embedding of training example k. The algorithm finds an embedding of examples given a distance metric between the examples encoded in the graph Laplacian $L = D - W$, where W is a similarity ("affinity") matrix between examples and $D_{ii} = \sum_j W_{ij}$ is the (diagonal) degree matrix. The basis vectors of the embedding are given by the top eigenvectors of the graph Laplacian. LE can be summarized as follows:

1. Construct the **neighborhood graph**.
2. Choose **weights** on the edges (e.g., $W_{ij} = 1$ if i and j are neighbors, or $W_{ij} = e^{\frac{||x_i - xj||^2}{2\sigma^2}}$).
3. Solve the **eigenvalue problem** $L\mathbf{v} = \lambda D\mathbf{v}$, where $D_{ii} = \sum_j W_{ij}$ and $L = D - W$).

There is a direct connection between LE and spectral clustering (SC). Both methods are based on the graph Laplacian of a similarity matrix. Spectral clustering only involves a supplementary step, that is, clustering the spectral images using a conventional clustering algorithm such as k-means.

Explicit methods have a unique solution. However, their cost becomes prohibitive when confronted with large datasets. Moreover, a vector f_i has to be learnt in order to embed an example x_i into a K-dimensional space; each example is embedded separately, and after training, given a new example x^* there is no straightforward way to embed it. This is referred to as the "out-of-sample" problem, which several authors have tried to address with special techniques (e.g., [6,16,26]). Learning a function-based embedding might prove useful to solve the above problems.

2.2 Functional Methods

While explicit methods learn directly the underlying low-dimensional representation, functional methods learn a mapping from the input space to the low-dimensional space. In this paper, we show that with the latter approach, learning can often be faster, while being easily generalizable to new data points, as we obtain a function and thus avoid the so-called out-of-sample problem.

As proposed in [14], one can instead of learning vectors learn a function-based embedding, i.e., $y_i = f(x_i)$, using some family of functions to choose f. Adopting this approach can provide several gains:

- Training is faster since if two examples x_i and x_j are highly correlated, the embedding for $f(x_j)$ will be good before we have even seen it during training if we have trained well on $f(x_i)$ already.
- Prior knowledge can be expressed via the set of possible embedding functions, as noted in [14]. In their work they showed how choosing a set of functions based on convolutions exploiting prior knowledge of images they achieve improved embedding results. In the clustering algorithms we develop, we will be able to encode this type of prior in exactly the same way.
- There is no out-of-sample problem, as we have the embedding $y^* = f(x^*)$.
- By choosing $f(x)$ of sufficiently high capacity, we can find the solution provided by a point-based embedding. By capacity we mean the complexity of the class of functions $f(x)$, measured by, e.g., the VC dimension.

Several methods have been devised to provide a model that is based on an embedding criterion. In [4], the authors incorporate a LE-type of regularization (see Eq. 1) in the cost function of the SVM, which provides a functional version

of this embedding method, where the function used is a kernel-based classifier. This method is known as the Laplacian SVM (LapSVM).

The above mentioned algorithms are functional embedding algorithms. Of course there are many functional clustering algorithms as well, e.g. classical k-means. Several authors also showed how to implement functional clustering within the context of a support vector machine by proposing objectives whereby good clusterings have large margins [5,30,31].

DrLIM Embedding and Siamese Networks. Hadsell, Chopra and LeCun [14] recently suggested to minimize the following loss function for embedding:

$$\sum_{ij} L\left(f_i, f_j, W_{ij}\right) = \sum_{ij} W_{ij}||f_i - f_j||^2 + \sum_{ij}(1 - W_{ij})\max(0, m - ||f_i - f_j||)^2 \quad (2)$$

where $W_{ij} = 1$ if i and j are deemed similar and 0 otherwise, and m is the size of the margin. DrLIM encourages similar examples to be mapped closely, and dissimilar ones to be separated by at least the distance m. This is related to energy-based models in general, such as Ising models or Hopfield networks, but it is based on neighborhood relations rather than actual labels.

Another similar approach is the Normalized Embedding [9]. The loss function, independent of magnitude, encourages similar examples to be positively correlated:

$$\sum_{ij} L\left(f_i, f_j, W_{ij}\right) = \sum_{ij} -W_{ij}\frac{f_i \cdot f_j}{||f_i||||f_j||} + \sum_{ij}(1 - W_{ij})\frac{f_i \cdot f_j}{||f_i||||f_j||} \quad (3)$$

These two loss functions were implemented as instantiations of a general class of neural networks called *siamese networks*. However, potentially any class of functions can be used to implement f.

3 Functional Embeddings for Clustering

In the above we considered the task of embedding data in a K-dimensional space $K < p$, where p is the dimensionality of the input. This can be viewed as soft clustering; for instance, PCA and soft k-means are guaranteed to provide, under certain conditions, the same result (see e.g. [12]).

We now propose a general objective for using functional embeddings for clustering. In our framework, one optimizes the following loss:

$$\sum_{c}\sum_{ij} L(f(x_i), c)\; Y_c(f(x_i), f(x_j), W_{ij}) \quad (4)$$

- W is a pairwise similarity matrix defined *a priori* as in previous algorithms.
- $L(\cdot)$ is a classification based loss function such as the hinge loss:

$$L\left(f(x), y\right) = H(yf(x)) = \max\left(0, 1 - yf(x)\right)) \quad (5)$$

For the multiclass case we can use $L(f(x), y) = \sum_{c=1}^{K} H(y(c)f_c(x))$. Here, $y(c) = 1$ if $y = c$ and -1 otherwise.

– $Y_c(f(x_i), f(x_j), W_{ij})$ encodes the weight to assign to point i being in cluster c. In the general case it can be a function of $f(x_i)$, $f(x_j)$ and the similarity score W_{ij}. Different choices of $Y_c(\cdot)$ implement different algorithms.

This class of algorithms learns a clustering $c(x) = \text{argmax}\ f(x) \in \mathbb{R}^K$ (or $c(x) = \text{sign}(f(x))$ in the two-class case). Our objective differs from usual clustering algorithms in that we use *pairs of examples* to learn the clustering. This objective is similar to embedding algorithms in that we essentially embed the data *at the same time as performing a clustering* by viewing the K-dimensional output as K clusters.

Intuitively, we directly encode into our clustering algorithm that neighbors with $W_{ij} > 0$ should have the same cluster assignment. For example, one choice for $Y_c(\cdot)$ that we explore in this paper, which we call NCutEmb, is as follows (two-class case, $c \in \{\pm 1\}$):

$$Y_c(f_i, f_j, W_{ij}) = \begin{cases} \eta^{(+)} & \text{if } \text{sign}(f_i + f_j) = c \text{ and } W_{ij} = 1 \\ -\eta^{(-)} & \text{if } \text{sign}(f_j) = c \text{ and } W_{ij} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{6}$$

Equation (6) assigns a pair of neighbors to the cluster with the most confident label from the pair. Examples x_j that are not neighbors of x_i, i.e. when $W_{ij} = 0$, are encouraged to fall into different clusters. Multiclass versions of NCutEmb are described in Section 3.3.

In the following we describe in detail some particular specializations of this objective function, and our particular implementation of it by training online by stochastic gradient descent using a set of functions implementable by a neural network.

3.1 Balancing Constraint

Many unconstrained clustering objective functions lead to a trivial solution, e.g., all the points end up in one single cluster, therefore it is often necessary to enforce some constraints. Even though we do not have access to label information, we can use the predicted labels to implement such constraints.

Unsupervised and semi-supervised learning (SSL) both usually make use of balancing constraints, albeit different since in SSL we have partial label information. In an unsupervised setting, the usual assumption is that each cluster should be reasonably large. For instance, RatioCut [15] and NormalizedCut [24] weigh, respectively, the cut by the number of vertices $|c_i|$ in cluster c_i and its volume $vol(c_i)$, i.e., the sum of the weights on the edges. In [2], the authors simply impose that each cluster has at least $\frac{N}{K}$ examples, where N is the size of the training set and K, the number of clusters.

In a semi-supervised setting, the problem is somehow easier, as the labeled points provide an estimate of the class balance (if the examples are i.i.d.). In [17], the author imposes that the fraction of each class in the unlabeled set is the same as in the labeled set. A similar constraint is used with low-density

Algorithm 1. Two-class NCutEmb algorithm with hard constraints

for each iteration **do**
  Pick x_i and x_j such that $x_j \in neighb\,(x_i)$
  Find $g = \text{argmax}_{i,j}(|f(x_i)|, |f(x_j)|)$ and $h = \text{sign}\,(f(x_g))$
  if $seen(h) \leq \frac{N_t}{K} + \zeta$ **then**
    Update the network w.r.t. to $\eta^{(+)} \nabla L\,(f(x_i), h)$ and $\eta^{(+)} \nabla L\,(f(x_j), h)$
    Update $seen(h)$, we assigned an example to cluster h
  end if
  Pick x_m and x_n such that $x_n \notin neighb\,(x_m)$
  Find $p = \text{sign}\,(x_m)$ and $q = \text{sign}\,(x_n)$
  Update the network w.r.t. $\eta^{(-)} \nabla L\,(f(x_m), -q)$ and $\eta^{(-)} \nabla L\,(f(x_n), -p)$
end for

separation [10]. In [18], the authors ignore examples of a class that has been seen more than its expected frequency, as provided by the labeled set.

Generally speaking, we can classify the constraints used in clustering simply as "hard" or "soft". We have tried both types, in the following way:

1. At each iteration, control the number of examples assigned to cluster c, denoted as $seen(c)$. We then define $Y_c(\cdot)$ as

$$Y_c(f_i, f_j, W_{ij}) = \begin{cases} \eta^{(+)} & \text{if } \text{sign}(f_i + f_j) = c \text{ and } W_{ij} = 1 \text{ and} \\ & seen(c) \leq \frac{N_t}{K} + \zeta \\ -\eta^{(-)} & \text{if } \text{sign}(f_j) = c \text{ and } W_{ij} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{7}$$

   where N_t is the number of examples seen at time t, K is the number of clusters and ζ is a negative or positive factor allowing for some variation in cluster size. One can apply this constraint on a fixed-sized window of the last, e.g., 100 examples seen, which corresponds to a particular choice of the function $seen(\cdot)$.

2. Diminish the "learning rate" proportionally to the number of examples seen of each class. $Y_c(\cdot)$ then becomes

$$Y_c(f_i, f_j, W_{ij}) = \begin{cases} \frac{\eta^{(+)}}{seen(c)+1} & \text{if } \text{sign}(f_i + f_j) = c \text{ and } W_{ij} = 1 \\ -\eta^{(-)} & \text{if } \text{sign}(f_j) = c \text{ and } W_{ij} = 0 \\ 0 & \text{otherwise.} \end{cases} \tag{8}$$

   where again $seen(c)$ is the number of examples assigned to cluster c seen in the last 100 examples (+1 to avoid zeros).

Algorithm 2. Two-class NCutEmb algorithm with soft constraints

for each iteration **do**

Pick x_i and x_j such that $x_j \in neighb\,(x_i)$

Find $g = \text{argmax}_{i,j}(|f(x_i)|, |f(x_j)|)$ and $h = \text{sign}\,(f(x_g))$

Update the network w.r.t. to $\frac{\eta^{(+)}}{seen(h)+1}\nabla L\,(f(x_i), h)$ and $\frac{\eta^{(+)}}{seen(c)+1}\nabla L\,(f(x_j), h)$

Update $seen(h)$, we assigned an example to cluster h

Pick x_m and x_n such that $x_n \notin neighb\,(x_m)$

Find $p = \text{sign}\,(x_m)$ and $q = \text{sign}\,(x_n)$

Update the network w.r.t. $\eta^{(-)}\nabla L\,(f(x_m), -q)$ and $\eta^{(-)}\nabla L\,(f(x_n), -p)$

end for

The hard constraint is similar to the approaches described in [2,18], while the soft constraint is very similar to the optimal learning rate for stochastic k-means, as reported in [8]. These approaches will be referred to as NCutEmb^h and NCutEmb^s, respectively.

3.2 Two-Class Clustering

In a two-class setting, we employ a neural network $f(x)$ with one output $y \in \{\pm 1\}$, trained online via stochastic gradient descent. We employ either a hard constraint (Algorithm 1) or a soft constraint (Algorithm 2). In these algorithms, $neighb\,(x_i)$ refers to the neighborhood of x_i.

Simply put, similar points are pushed towards the same label. In the case of dissimilar points, the points are pulled in opposite classes. We use two different learning rates $\eta^{(+)}$ and $\eta^{(-)}$. The latter is used to pull non-neighbors in different classes and is typically smaller. The idea behind the method is illustrated in Figure 1.

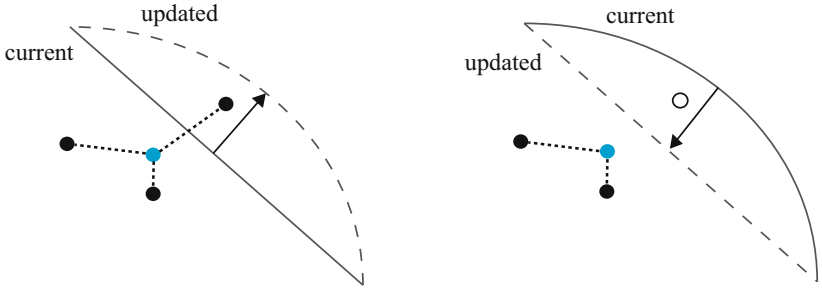

Fig. 1. Clustering using NCutEmb. A hyperplane that separates neighbors is pushed to classify them in the same class (**left**; the classifier cuts through an edge of the graph and is pushed upwards), whilst one that classifies non-neighbors in the same class is modified to separate them (**right**; the hyperplane is pushed to separate the unconnected points).

3.3 The Multiclass Case

In the multiclass setting, we also tried two alternative approaches.

1. The same strategy as binary case: push neighbors towards the **most confident label**, i.e., if we define the most confident example of a pair as

$$m(f_i, f_j) = \text{argmax}_{i,j} \left(\max(f_i), \max(f_j)\right))$$

then we can define the function $Y_c(\cdot)$ using:

$$Y_c(f_i, f_j, W_{ij}) = \begin{cases} \eta^{(+)} & \text{if } \text{argmax } f_{m(f_i,f_j)} = c \text{ and } W_{ij} = 1 \\ -\eta^{(-)} & \text{if } \text{argmax } f_j = c \text{ and } W_{ij} = 0 \\ 0 & \text{otherwise.} \end{cases} \quad (9)$$

The balancing constraint can then be enforced in a hard way by a fixed-sized window, as in the binary case.

2. Push towards the K clusters simultaneously, with one learning rate η per class, weighted by the outputs of the point with the most confident label. That is,

$$Y_c(f_i, f_j, W_{ij}) = \begin{cases} \eta_c & \text{if } W_{ij} = 1 \\ 0 & \text{otherwise} \end{cases} \quad (10)$$

where

$$\eta_c \leftarrow \eta^{(+)} f_c(x_i)$$

The balancing constraint is soft, i.e, the learning rate decreases with the size of the cluster.

These approaches will be referred to as NCutEmb^{max} and NCutEmb^{all}. In the latter approach, the softmax$(\cdot)$ function is used at the output of $f(\cdot)$ in order to have $f(x_i) \in [0, 1]$:

$$\text{softmax}\,(y_1, ..., y_k) = \frac{\exp(y_i)}{\sum_i \exp(y_i)}$$

This is a common technique in Neural Networks [7].

3.4 Large-Scale Extension

Building a graph *a priori* is not feasible when dealing with very large datasets (i.e., millions of examples). One can easily avoid this step by computing neighbors on the fly with epsilon-balls or local weighted distances. Indeed, whenever two points are picked, it is straightforward to verify a condition such as

$$||x_i - x_j|| < \epsilon$$

where ϵ is a neighborhood parameter. Note that any similarity measure can be used. For efficiency purposes, a table with the neighbors found up to date can be kept in memory.

4 Experiments

Table 1 summarizes the small-scale datasets that have been used throughout Section 4. `g50c`, `text` and `usps` are datasets that are often used in the semi-supervised learning literature (e.g., [10,11]). Numerous clustering techniques were applied to the `usps` digit dataset in [29]. The authors, however, studied two subsets of 2 and 4 digits. In this paper we consider the 10 `usps` classes.

The datasets `bcw` (Breast Cancer Wisconsin) and `glass` (Glass Identification) are taken from the UCI repository [1]. The `ellips` dataset consists of four 50-dimensional ellipses connected by one extremity.

In order to have a relative measure of class imbalance, which will be useful for the clustering section, we define the degree of class imbalance (DCI):

$$DCI = \frac{1}{N}\left(\frac{1}{K-1}\sum_{k}\left(|c_k| - \frac{N}{K}\right)^2\right)^{\frac{1}{2}} \tag{11}$$

This is the standard deviation of the clusters with respect to the "expected value" of the size of clusters c_k in a balanced problem, divided by N, the number of examples. A DCI of 0 thus represents a perfectly balanced problem. Simply taking the ratio between the biggest and smallest classes would not take into account the variability across all the classes in a multiclass setting.

4.1 Functional vs. Explicit Embedding

We first performed some experiments that aimed at comparing embedding methods as a justification of the use of functional embedding for clustering. To this end, we have used the `g50c`, `text` and `usps` datasets. We provide results for explicit and functional DrLIM, and these are compared with Laplacian Eigenmaps. The goal is to reach a value as close as possible to LE, as DrLIM is an approximate method. However, the latter has the advantage of being useful on much larger datasets as it does not require costly operations such as matrix inversion. The performance criterion that has been used is the fraction of embedded points

Table 1. Small-scale datasets used throughout the experiments. `g50c`, `text` and `usps` are used in Section 4.1, while all datasets are used in Section 4.2.

data set	classes	dims	points	DCI ×100
g50c	2	50	550	0
text	2	7511	1946	0.87
bcw	2	9	569	21.19
ellips	4	50	1064	12.21
glass	6	10	214	13.89
usps	10	256	2007	2.83

that do not share the same label with their nearest neighbor, i.e.,

$$L_{embed}\left(f_1, ..., f_n, \alpha\right) = \frac{1}{N}\sum_{i=1}^{n} \delta\left(f_i, f_{nn(i)}\right) \tag{12}$$

where $f_{nn(i)}$ is f_i's nearest neighbor in the embedding (the relation needs not be mutual), α are the parameters, N is the size of the training set, and

$$\delta\left(f_i, f_{nn(i)}\right) = \begin{cases} 0 \text{ if } y_i = y_{nn(i)} \\ 1 \text{ if } y_i \neq y_{nn(i)} \end{cases}$$

Here, y_i designates the label of f_i. We have chosen this criterion because unlike Eq. 1, it does not depend on the specific weights W_{ij} that are used for each dataset.

Table 2 reports the best values of the embedding criterion obtained. The number of iterations required to achieve that result is given in brackets. The results are compared against Laplacian Eigenmaps. Both LE and DrLIM are assessed using the criterion of Eq. 12. We have tried both a linear network ("Linear") and a one-hidden layer network ("NN"). The linear network performed best for the `g50c` dataset, while the NN performed better for the `text` and `usps` datasets. For all the methods, the number of neighbors was optimized. An appropriate learning rate was selected for nf-DrLIM and f-DrLIM.

It can be observed from Table 2 that functional DrLIM provides a gain of accuracy compared to its non-functional counterpart, and requires a smaller number of iterations. Using a functional model rather than a point-based one appears more important than the choice of the class of functions used within such a model. Indeed, even though there are differences specific to each dataset between a linear network and a one hidden layer network, the differences between functional and point-based methods are larger, in terms of both convergence speed and value of the embedding criterion.

Table 2. Lowest obtained value of the embedding criterion (Eq. 12) for LE, functional DrLIM and non-functional DrLIM, on an average of 10 runs. Low values indicate that points sharing the same label are mapped closely. The number of iterations (in iterations $\times 10^3$) to reach a minimum value for f-DrLIM and nf-DrLIM is given in brackets. Convergence is considered achieved when the value of the embedding criterion stops decreasing. The neural network has one hidden layer and 5, 40 and 40 hidden units for the three datasets, respectively. For every dataset, the appropriate learning rate was selected.

	g50c	text	usps
LE	7.45 [-]	10.69 [-]	7.28 [-]
nf-DrLIM	12.82 [100]	29.58 [70]	11.55 [80]
f-DrLIM (Linear)	**7.14** [50]	14.18 [50]	10.02 [70]
f-DrLIM (NN)	7.55 [80]	**12.00** [40]	**8.99** [70]

The network that has been trained using functional DrLIM can be applied to any other data point; we thus avoid the out-of-sample problem, as we have a functional relationship between the high-dimensional input and the output. The fact that functional DrLIM is also more accurate and faster than the point-based method further encourages the use of functional methods for embedding. Indeed, the difference with the values obtained with LE, towards which we wish to converge, is small. Note that LE optimizes the criterion of Eq. 1, but we make the assumption that a good embedding will minimize both Eq. 1 and 12.

4.2 Small-Scale Clustering Experiments

Here, all the datasets from Table 1 are used. We compare the NCutEmb algorithm to k-means and spectral clustering (the algorithm presented in [20]). For spectral clustering, a k-nearest neighbor graph (sc-knn) and a full radial basis function graph (sc-rbf) have been used for the calculation of the weights, and the corresponding parameters (k and σ) have been selected. For the NCutEmb method, we have selected for each experiment the appropriate number of neighbors for W (typically 10, 50 or 100) and learning rate (a value between 0.001 and 1).

Tables 3 and 4 report the cluster assignment error for two-class and multiclass problems, respectively. The cluster assignment error is simply the fraction of points for which $c_i \neq y_i$, where c_i is the cluster in which we assign x_i. This has to be calculated considering all permutations from clusters to indices. When confronted to a large number of classes, one can use optimization techniques to solve this problem, as underlined in [27]. For the small-scale experiments, training times ranged approximately from 5 seconds (`bcw`) to a minute (`usps`) on a standard PC.

The results show that the NCutEmb method performs either similarly or provides superior performance than conventional clustering algorithms. Since we have other benefits when using NCutEmb, such as the availability of a functional model and the scalability to large datasets, the approach is quite attractive. For

Table 3. Cluster assignment error (%) on two-class small-scale datasets with the NCutEmb method, compared to k-means and spectral clustering. The spectral clustering algorithms have been optimized over the choice of σ and k. Each value is averaged over 10 random replications of the initial dataset. A linear network was used for `bcw` and `text`, while a one-hidden layer NN with 5 hidden units was used for `g50c`.

	bcw	g50c	text
k-means	3.89	4.64	7.26
sc-rbf	3.94	5.56	6.73
sc-knn	3.60	6.02	12.9
NCutEmb^h	3.63	4.59	**7.03**
NCutEmb^s	**3.15**	**4.41**	7.89

Table 4. Cluster assignment error (%) on multiclass small-scale datasets with the NCutEmb method, compared to k-means and spectral clustering. The spectral clustering algorithms have been optimized over the choice of σ and k. Each value is averaged over 10 random replications of the initial dataset. A linear network was used for `ellips` while a one-hidden layer network with 3 hidden units was used for `glass` and 40 hidden units for `usps`.

	ellips	glass	usps
k-means	20.29	25.71	30.34
sc-rbf	10.16	39.30	32.93
sc-knn	**2.51**	40.64	33.82
NCutEmb^{max}	4.76	**24.58**	19.36
NCutEmb^{all}	2.75	24.91	**19.05**

the two-class datasets, the difference between the type of constraint used appears minor.

Regarding the multiclass experiments, it is interesting to study the performance of the methods with respect to the class imbalance of each dataset. NCutEmb^{all} outperforms the other methods (except sc-knn) for the `ellips` dataset, which is very imbalanced. However, the `glass` dataset is also imbalanced, and both NCutEmb^{all} and NCutEmb^{max} perform well. Consequently, it is hard to draw any conclusion regarding the superior ability of one method or another to handle clusters of different sizes; the two methods can handle them well given the right parameters.

On average, NCutEmb appears superior to both k-means and spectral clustering on the datasets that we have used. This is a surprising result, as we could expect spectral clustering to be the "ground truth" result, as with LE in Section 4.1. However, this could be the effect of the use of a one-stage approach for the optimization. Indeed, even if a good embedding is obtained with spectral clustering, the clustering stage might be poor (and vice versa). Performing the embedding and clustering tasks simultaneously seems to overcome this problem. We note, however, that similarly to k-means, the method performs better when clusters are close to Gaussian objects. Since NCutEmb is neighborhood based, we cannot expect good performance with highly complex clusters. If $||x_i - x_j|| < \epsilon$, we must have a high probability that x_i and x_j truly belong to the same cluster. In other words, the smoothness assumption must hold.

We then conducted a series of experiments where we split the datasets in order to obtain a test error, as an estimation of the out-of-sample error. In the following experiments, 70% of the data is used for training, and the error is calculated on the remaining 30%, which has not been seen. Tables 5 and 6 report this error for two-class and multiclass methods.

We note that the difference between clustering error and out-of-sample error is small for each dataset. A network that has been trained on a large enough set of examples can cluster new points accurately.

Table 5. Test error (i.e., out-of-sample error) for the two-class datasets (%)

	bcw	g50c	text
k-means	4.22	**6.06**	8.75
NCutEmb^h	**3.21**	**6.06**	7.68
NCutEmb^s	3.64	6.36	**7.38**

Table 6. Test error (i.e., out-of-sample error) for the multiclass datasets (%)

	ellips	glass	usps
k-means	20.85	28.52	29.44
NCutEmb^{max}	5.11	25.16	20.80
NCutEmb^{all}	**2.88**	**24.96**	**17.31**

4.3 Large-Scale Clustering Experiments

Here, we repeat the experiments of the previous section with the MNIST digits database, consisting of 60,000 training examples and 10,000 test examples of digits 0 to 9. Each image has 28×28 pixels. MNIST is a fairly well balanced dataset (DCI$\times$100 of 0.57 using Eq. 11). NCutEmb requires about 60 minutes to converge to an optimum on a database such as MNIST. For visualization purposes, projections of MNIST data are shown in Fig. 2 using Laplacian Eigenmaps. Only a subset of 3000 examples is projected in Fig. 2 because of computational limitations, as LE requires the eigenvectors of the similarity matrix of the data.

Table 7. Clustering the MNIST database with the NCutEmb^{max} and NCutEmb^{all} methods, compared to k-means, for different numbers of clusters. The training error (i.e., clustering error) and the test error (out-of-sample error) are provided. A one-hidden layer network with 40 units has been used for the 10 and 20 clusters problems, and 80 units for the 50 clusters problem.

# clusters	method	train	test
50	k-means	18.46	17.70
	NCutEmb^{max}	**13.82**	**14.23**
	NCutEmb^{all}	18.67	18.37
20	k-means	29.00	28.03
	NCutEmb^{max}	20.12	23.43
	NCutEmb^{all}	**17.64**	**21.90**
10	k-means	40.98	39.89
	NCutEmb^{max}	**21.93**	**24.37**
	NCutEmb^{all}	24.10	24.90

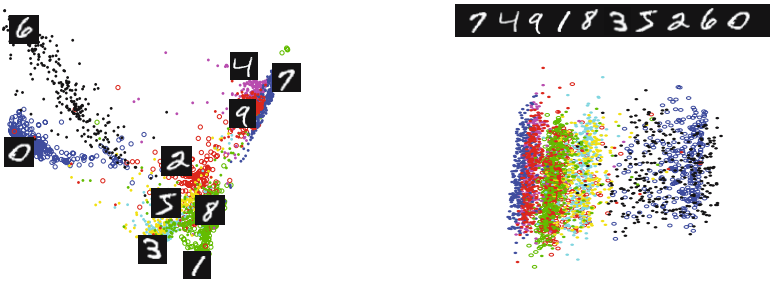

Fig. 2. Projections of an MNIST subset using Laplacian Eigenmaps with a nearest-neighbor graph

Table 7 presents the training and test error of the network used to cluster MNIST. We compare our method to k-means. For 10 and 20 clusters, NCutEmb^{max} and NCutEmb^{all} provide equivalent error rates. We seem to achieve a much better performance than k-means in most cases, except with NCutEmb^{all} using 50 clusters. This may be due to the fact that within this approach, we push neighbors towards all classes at each iteration. It is interesting to note that the gap with k-means is reduced when increasing the number of clusters. This could be expected; the more groups we use to divide the dataset, the more simple these groups become in their structure. With a very large number of clusters, NCutEmb and k-means should be expected to provide the same results.

5 Conclusion

We have presented a framework for data clustering based on functional models for dimensionality reduction optimized by stochastic gradient descent. Our framework suggests direct algorithms for clustering using an embedding approach rather than the two-stage approach of spectral clustering. The experiments have shown that using this approach brings about many benefits: (i) the algorithms scale well as we do not need to compute eigenvectors, (ii) a model is available and can be applied to new data points, and (iii) the method provides superior results on average than spectral clustering or k-means.

Some limitations were highlighted, however, in the way it can handle highly non-linear datasets. The more we violate the smoothness assumption, the more chance that pairs of neighbors do not belong to the same cluster and the less accurate the method is expected to be. However, this limitation is dependent on the method for choosing neighbors: if one has a way of choosing pairs of neighbors which can guarantee they belong to the same cluster, then the method can perform well.

We believe that this approach could be of great use in many important applications areas where new examples are continuously available.

Acknowledgements

The authors thank the anonymous reviewers for their comments. FR is funded by the Swiss National Science Foundation (grant no.105211-107862).

References

1. Asuncion, A., Newman, D.J.: UCI machine learning repository (2007)
2. Banerjee, A., Gosh, J.: Scalable clustering algorithms with balancing constraints. Data Mining and Knowledge Discovery 13(3), 365–395 (2006)
3. Belkin, M., Niyogi, P.: Laplacian eigenmaps for dimensionality reduction and data representation. Neural Computation 15(6), 1373–1396 (2003)
4. Belkin, M., Niyogi, P., Sindhwani, V.: Manifold regularization: a geometric framework for learning from Labeled and Unlabeled Examples. Journal of Machine Learning Research 7, 2399–2434 (2006)
5. Ben-Hur, A., Horn, D., Siegelmann, H.T., Vapnik, V.: Support vector clustering. Journal of Machine Learning Research 2, 125–137 (2001)
6. Bengio, Y., Delalleau, O., Le Roux, N., Paiement, J.-F., Vincent, P., Ouimet, M.: Learning eigenfunctions links spectral embedding and kernel PCA. Neural Computation 16(10), 2197–2219 (2004)
7. Bishop, C.M.: Neural Networks for Pattern Recognition. Oxford University Press, USA (1995)
8. Bottou, L.: Stochastic learning. In: Bousquet, O., von Luxburg, U., Rätsch, G. (eds.) Machine Learning 2003. LNCS (LNAI), vol. 3176, pp. 146–168. Springer, Heidelberg (2004)
9. Bromley, J., Bentz, J.W., Bottou, L., Guyon, I., LeCun, Y., Moore, C., Sackinger, E., Shah, R.: Signature verification using a siamese time delay neural network. International Journal of Pattern Recognition and Artificial Intelligence 7(4) (August 1993)
10. Chapelle, O., Zien, A.: Semi-supervised classification by low density separation. In: AISTATS, pp. 57–64 (January 2005)
11. Collobert, R., Sinz, F., Weston, J., Bottou, L.: Large scale transductive SVMS. Journal of Machine Learning Research 7, 1687–1712 (2006)
12. Ding, C., He, X.: K-means clustering via principal component analysis. In: Proc. of the Int. Conference on Machine Learning (ICML 2004) (2004)
13. Gong, H.F., Pan, C., Yang, Q., Lu, H.Q., Ma, S.: Neural network modeling of spectral embedding. In: BMVC 2006, p. I–227 (2006)
14. Hadsell, R., Chopra, S., LeCun, Y.: Dimensionality reduction by learning an invariant mapping. In: Proc. Computer Vision and Pattern Recognition Conference (CVPR 2006). IEEE Press, Los Alamitos (2006)
15. Hagen, L., Kahng, A.: New spectral methods for ratio cut partitioning and clustering. IEEE Trans. on Computer Aided-Design 11(9), 1074–1085 (1992)
16. He, X., Yan, S.C., Hu, Y., Niyogi, P., Zhang, H.J.: Face recognition using laplacianfaces. IEEE Trans. PAMI 27(3), 328
17. Joachims, T.: Transductive inference for text classification using support vector machines. In: International Conference on Machine Learning, ICML (1999)
18. Karlen, M., Weston, J., Erken, A., Collobert, R.: Large scale manifold transduction. In: Proc. of the Int. Conference on Machine Learning (ICML 2008) (2008)

19. Lee, J.A., Verleysen, M.: Nonlinear Dimensionality Reduction. Springer, New York (2007)
20. Ng, A.Y., Jordan, M., Weiss, Y.: On spectral clustering: analysis and an algorithm. In: Advances in Neural Information Processing Systems (NIPS 13) (2001)
21. Roweis, S.T., Saul, L.K.: Nonlinear dimensionality reduction by locally linear embedding. Science 290(5500), 2323–2326 (2000)
22. Saul, L.K., Weinberger, K.Q., Ham, J.H., Sha, F., Lee, D.D.: Spectral methods for dimensionality reduction. In: Semi-Supervised Learning. MIT Press, Cambridge (2006)
23. Schölkopf, B., Smola, A.J., Müller, K.R.: Nonlinear component analysis as a kernel eigenvalue problem. Neural Computation 10, 1299–1319 (1998)
24. Shi, J., Malik, J.: Normalized cuts and image segmentation. IEEE Trans. on Pattern Analysis and Machine Intelligence 22(8) (2000)
25. Tenenbaum, J.B., de Silva, V., Langford, J.C.: A global geometric framework for nonlinear dimensionality reduction. Science 290(5500), 2319–2323 (2000)
26. Trosset, M.W., Priebe, C.E.: The out-of-sample problem for multidimensional scaling. Technical Report 06-04, Dept. of Statistics, Indiana University (2006)
27. Verma, D., Meila, M.: Comparison of spectral clustering methods. In: Advances in Neural Information Processing Systems (NIPS 15) (2003)
28. Weinberger, K.Q., Saul, L.K.: Nonlinear dimensionality reduction by semidefinite programming and kernel matrix factorization. In: Proc. of the Tenth International Workshop on AI and Statistics (AISTATS 2005) (2005)
29. Wu, M., Schölkopf, B.: A local learning approach for clustering. In: Advances in Neural Information Processing Systems (NIPS 19) (2006)
30. Xu, L., Neufeld, J., Larson, B., Schuurmans, D.: Maximum margin clustering. In: Advances in Neural Information Processing Systems (NIPS 16) (2004)
31. Zhang, K., Tsang, I., Kwok, J.T.: Maximum margin clustering made practical. In: Proc. of the Int. Conference on Machine Learning (ICML 2007) (2007)