Clustering in a Boosting Framework

Amir Saffari and Horst Bischof

Institute for Computer Graphics and Vision, Graz University of Technology
saffari@icg.tu-graz.ac.at, bischof@icg.tu-graz.ac.at

Abstract *In this paper we present a novel approach for creating partitions of data space using simple clustering algorithms in a boosting framework. A general boosting algorithm for clustering tasks is proposed, and solutions for directly optimizing two loss functions according to this framework are obtained. Experimental results show how the performance of relatively simple and computationally efficient base clustering algorithms could be boosted using the proposed algorithm.*

1 Introduction

Without doubt, currently ensemble methods are amongst the best techniques in classification domain. These methods include particularly a very successful genre called as *boosting*, where any member of the ensemble of classifiers are trained sequentially to compensate the shortcomings of the previously trained models, usually using the notion of sample weights. Recently, there has been a few attempts to bring the same idea of ensemble learning to the clustering domain. However, this transition is not as straight-forward as seen at a first glance, mainly because of the vague nature of the clustering due to the lack of supervisory information commonly used in classification tasks. Even after almost 60 years of research in clustering domain, there are still debates on what really is a clustering algorithm supposed to perform [20, 12].

In general, ensemble clustering can be divided into two categories. In the first category, each member of the ensemble is generated independent of the other models, usually with different initial conditions or parameters, and the goal is usually to achieve a better clustering by just combining the outputs of a set of diversely generated base models [6, 7, 15, 2, 4, 16, 19]. Since each clustering algorithm usually results in a set of disjoint partitions of the data space, it is conventional to represent the output of the clustering model to an input data sample as an index to the corresponding cluster. While this is convenient for a single algorithm, it is a source of problems when we want to compare or combine the outputs of different clustering methods, mainly because the ordering of the clusters are arbitrary in every clustering algorithm. As a result, it is not meaningful to directly compare the output indexes of different partitioning methods. This problem is referred to as *cluster correspondence* problem in the clustering field. The correspondence problem becomes more complicated when different algorithms are allowed to deliver different number of clusters. As a result, the problem of finding or learning a *consensus function* for the clustering combination is usually the main concern of this category. We will refer to this set of methods as just *ensemble clustering*, because of their similarities with non-boosting classification methods and their fundamental differences with the next category.

The second category could be considered as the clustering counterpart of the boosting-based methods in classification domain, since the main concern of these algorithms is to create and add the new models to the ensemble based on the performance of the previously trained set of models. It should be noted that in the same way for the previous category, a meaningful combination of clustering methods can only be achieved with a proper consensus function, but here learning a consensus function is not the only goal of the algorithms. We will refer to this set of algorithms as *boosting-based clustering*. Tophcy et al. [18] incorporated this idea with sub-sampling of the dataset using the sample weights driven from a consistency index, which is a measure of how often a sample remain in the same cluster as the new models are introduced to the ensemble. A set of *k-means* algorithms is used as the base models, and after finishing the boosting iterations, a consensus function is calculated for the whole ensemble. They have shown that this method results in a better performance compared to an ensemble method [13] which uses a uniform sub-sampling of the dataset. Frossyniotis et al. [10] used the same concept of sub-sampling the dataset, by using two different performance measures for assessing the clustering quality for each sample, both based on the confidence (or membership) values of each sample to the individual clusters. Frossyniotis et al. incorporated a very similar approach used in the original Discrete AdaBoost [8] for updating the weights for both samples and base models. They compared the performance of *k-means* and *fuzzy c-means* to their boosted versions, and showed a better clustering results on a variety of datasets.

In order to deploy the concept of boosting to the clustering domain, there is a need for solving one main problem besides the consensus function calculations. This problem can be stated as finding a proper performance measure in order to assess the quality of both individual (local) clustering algorithms and later on the whole collection of them (global). Note that in this context we will use *local* referring to the base models and *global* for the ensemble model. In general, such a global objective function should be minimized by iterations of boosting algorithm based on the local performance of base models. As a result, there should be a

strong and direct connection between local and global objectives. However, this direct connection is not clearly described in [18, 10]. In details, Topchy et al. [18] used the consistency index as the global objective function, but it is not clear how this function is minimized by the iterative updates of the base models using the local performance measure described in their algorithm. The same sort of problems also can be seen in [10] where there exist two local performance measures to assess base models, but it is not clear what kind of global objective functions could be minimized using iterative application of such loss functions to base models and sample weights update.

Additionally, in both of the methods presented in [18, 10], there is a major problem with the local objective functions as the base models are not directly optimizing them. In other words, the base models are assessed on a criterion which they are not designed to directly minimize. As a result, there is no guarantee for the base algorithms to result in models which could optimize the local performance criterion. Furthermore, using the sub-sampling with replacement is not a proper way of introducing the sample weights to the base models, because in most cases there exist a few outliers in the dataset which are usually considered as hard samples. Typically after a few iterations, the weights of these outliers will grow. As a result if we subsample the dataset with replacement, we will generate fake clouds of data with repeated instances of these outliers, which will ultimately result in a very poor clustering base model.

Based on these facts, we present in this paper a boosting-based clustering algorithm which builds forward stage-wise additive models for data partitioning and overcomes previously explained problems in a theoretical framework. Needless to say, data clustering plays an important and essential role in many computer vision applications. For example in unsupervised or weakly-supervised object recognition problems, the visual words (or parts) are usually constructed by clustering a set of descriptor responses to some selected image regions (usually extracted by interest point/region detectors), for example refer to [3] and references therein. It is also well-known that many of classical segmentation algorithms relies on clustering to find visually similar regions in the images [5].

In Section 2 we will explain theoretical foundations for this algorithm and we will describe how the presented methods relates to minimization of two loss functions. We will present the experimental results of the proposed methods on a few datasets in Section 3.

2 Methods

2.1 Additive Modeling

In order to formulate the overall process, we define the outputs of a clustering method as a collection of two functions, i and f:

$$i : \mathbf{x} \in \mathbb{R}^D \rightarrow \mathbb{R}^K \tag{1}$$

$$f : \mathbf{x} \in \mathbb{R}^D \rightarrow \mathbb{R}^D \tag{2}$$

Roughly speaking, the function i determines the membership probability of the sample to each cluster, and the function f provides a prototype from the partition that the input sample belongs to, which is usually referred to as the *cluster center*. Note that for those clustering algorithms which do not incorporate directly the notion of cluster centers, the output of function f can be seen as an instance from those samples clustered together, usually represented as their mean.

Let $\mathbf{y} = i(\mathbf{x}) = [y_1, \ldots, y_K]^T$ be the membership probability vector where K is the number of partitions and the values of each y_k is given as:

$$y_k = P(\mathbf{x} \in C_k) \tag{3}$$

where C_k represents kth cluster, and we require that $\sum_{k=1}^{K} y_k = 1$. In the simplest case, if we assume that there is only one prototype per cluster, then we can represent all prototypes in a matrix format like $P = [\mathbf{p}_1 | \ldots | \mathbf{p}_K]$. Now we define the function f as:

$$f(\mathbf{x}) = Pi(\mathbf{x}) \tag{4}$$

In a more general case which we will consider later, it is possible that the columns of the matrix P to be dependent on the input sample, resulting in a general form of:

$$f(\mathbf{x}) = P(\mathbf{x})i(\mathbf{x}) \tag{5}$$

In other words, the prototype within a cluster is not fixed and depends on the input sample. Despite of its unusual usage here, dependence of P on $\mathbf{x}$ will ease the definition of additive model for the function f in the context of weighted voting scheme presented later.

Note that this general definition encompasses both discriminative and generative clustering algorithms [21]. Specially in the case of discriminative clustering (like *k-means*), exactly one location of vector $i(\mathbf{x})$ has a non-zero value equal to one, and P is the collection of cluster centers. In this case the representation given in (4) exactly corresponds to the cluster center returned by the algorithm.

Assume that we are given a set of M clustering algorithms, $\{(j_m(\mathbf{x}), g_m(\mathbf{x}))\}, m = 1, \ldots, M$, all trained over a common dataset with equal number of partitions, K. $j_m(\mathbf{x})$ and $g_m(\mathbf{x})$ represent the membership and prototype retrieval functions for the mth clustering algorithm, respectively. Now we present the cluster membership function, $i(\mathbf{x})$, using the additive form of:

$$i(\mathbf{x}) = \sum_{m=1}^{M} \beta_m j_m(\mathbf{x}) \tag{6}$$

where the function $i(\mathbf{x})$ is an additive construction of some M base functions, $\{j_m(\mathbf{x})\}$ and $\beta_m \in \mathbb{R}$ are multipliers used to indicate the influence of each individual base function and we require that $\sum_{m=1}^{M} \beta_m = 1$.

The presented additive model (6) could be thought as a weighted voting scheme, specially when the base algorithms are discriminative. Since the base functions are considered to be a clustering algorithm on their own, we require them to follow the same definitions given in (3) and (5).

Additionally based on definition given in (5), it is easy to show that there exist solutions for $P(\mathbf{x})$ given the base

prototype matrices as:

$$\sum_{m=1}^{M} \beta_m g_m(\mathbf{x}) = \sum_{m=1}^{M} \beta_m P_m(\mathbf{x}) j_m(\mathbf{x}) = \\ = P(\mathbf{x}) \sum_{m=1}^{M} \beta_m j_m(\mathbf{x}) = P(\mathbf{x}) i(\mathbf{x}) \tag{7}$$

where $g_m(\mathbf{x})$ is the prototype retrieval function, and $P_m(\mathbf{x})$ is the matrix of cluster prototypes of mth base clustering algorithm. So without loss of generality we can define the additive formulation for prototype retrieval function as:

$$f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m g_m(\mathbf{x}) \tag{8}$$

Note that the formulation presented above is true regardless of the cluster correspondence problem, however for a meaningful voting in an ensemble of clustering models, we need to establish the correspondences of clusters in base models. We will discuss this issue later.

2.2 Learning

In order to learn the parameters of the additive model described in previous section, we have to consider a loss (or objective) function to be optimized by the algorithm. Since clustering is an unsupervised method, which means that there is no additional information available about the data, such as labels, there is no direct assessment regarding the quality of a model. As a result, it is usually hard to justify whether an algorithm was successful in dealing with a dataset or not. But nevertheless, many clustering algorithms try to formulate the whole process into an optimization framework and find the best solution for the given dataset based on the proposed objective function. For example the classical k-means algorithm uses the following function as an optimization criteria in order to minimize the intra-cluster variance:

$$L = \sum_{k=1}^{K} \sum_{\mathbf{x}_n \in C_k} \|\mathbf{x}_n - \mathbf{p}_k\| \tag{9}$$

where K is the number of cluster centers, C_k is the set of samples assigned to the k^{th} center, and $\mathbf{p}_k$ is the mean point of the C_k.

We define a general empirical loss function as:

$$L(\mathbf{x}, i, f) = L(\mathbf{x}, \sum_{m=1}^{M} \beta_m g_m(\mathbf{x}), \sum_{m=1}^{M} \beta_m j_m(\mathbf{x})) \tag{10}$$

which measures the quality of a given clustering model or an ensemble of them based on the input samples. Depending on the loss function, it is possible to use both i and f functions or one of them in order to evaluate the quality of the clustering.

Now we can specify the forward stage-wise additive modeling as an algorithm that tries to fit a new model at each stage to the ensemble of previous models without altering the parameters of the ensemble units [9]. The general clustering algorithm based on such an approach is described in Algorithm (1).

Algorithm 1 General Forward Stage-wise Additive Modeling for Clustering

1: Input dataset: $X = \{\mathbf{x}_n\}, n = 1, \ldots, N$.
2: Initialize the model: $f^{(0)}(\mathbf{x}) = \mathbf{0}$, and $i^{(0)}(\mathbf{x}) = \mathbf{0}$.
3: **for** $m = 1$ to M **do**
4: Compute: $(\beta_m, j_m, g_m) = \arg\underset{\beta,j,g}{min} \sum_{n=1}^{N} L(\mathbf{x}_n, i^{(m-1)}(\mathbf{x}_n) + \beta j(\mathbf{x}_n), f^{(m-1)}(\mathbf{x}_n) + \beta g(\mathbf{x}_n))$
5: Set: $f^{(m)}(\mathbf{x}) = f^{(m-1)}(\mathbf{x}) + \beta_m g_m(\mathbf{x})$ and $i^{(m)}(\mathbf{x}) = i^{(m-1)}(\mathbf{x}) + \beta_m j_m(\mathbf{x})$
6: **end for**
7: Output the final model: $f(\mathbf{x}) = \sum_{m=1}^{M} \beta_m g_m(\mathbf{x})$ and $i(\mathbf{x}) = \sum_{m=1}^{M} \beta_m j_m(\mathbf{x})$.

Note that according to this algorithm the new model will be fitted to compensate the shortcomings of the previous set of models. Now the main question is that how we can define a proper loss function for a clustering algorithm and how this function can be optimized at each step of the algorithm. In next sections, we will provide solutions for the quantization and correlation loss functions.

2.2.1 Quantization Error As explained earlier, the objective function (9), also known as *quantization error*, is used widely in different center-based algorithms, like *k-means*, *VQ*, and *SOM* [11]. Here in order to facilitate the mathematical formulation, we redefine the inner part of (9) as:

$$J(\mathbf{x}, f) = \|\mathbf{x} - f(\mathbf{x})\|^2 = (\mathbf{x} - f(\mathbf{x}))^T (\mathbf{x} - f(\mathbf{x})) \tag{11}$$

We can simplify this equation into:

$$J(\mathbf{x}, f) = \mathbf{x}^T\mathbf{x} - 2\mathbf{x}^T f(\mathbf{x}) + f(\mathbf{x})^T f(\mathbf{x}) \\ = \|\mathbf{x}\|^2 + \|f(\mathbf{x})\|^2 - 2\mathbf{x}^T f(\mathbf{x}) \tag{12}$$

Since we are searching for a function $f(\mathbf{x})$ which minimizes this equation, one can simply ignore the first term which is independent of $f(\mathbf{x})$ and write the loss function as:

$$J(\mathbf{x}, f) \approx \|f(\mathbf{x})\|^2 - 2\mathbf{x}^T f(\mathbf{x}) \tag{13}$$

Now we can derive the loss function formulation for an additive model based on the same approaches used in (12) and (13):

$$J(\mathbf{x}, f + \beta g) = J(\mathbf{x}, f) + J(\mathbf{x}, \beta g) + 2\beta f(\mathbf{x})^T g(\mathbf{x}) \tag{14}$$

Since in a forward stage-wise additive modeling, the previous models are kept intact, it will be shown later that the first term of (14) is playing the role of sample's weight and represents the amount of loss of each sample up to the current stage of the algorithm. Now looking into the second and third terms, some interesting aspects of this loss function becomes clear. Firstly based on the second term, the function $\beta g(\mathbf{x})$ is needed to be a good clustering algorithm on its own. Secondly the third term encourages the new additional clustering algorithm to search for spaces which are not in the vicinity of the already explored data spaces by previous models.

But since many of existing clustering algorithms are designed to work on their own, they are only able to give optimization solutions to the second term of (14). As a result in order to utilize existing methods, we will ignore the last term of (14). Furthermore, for simplicity we give equal weights to all base models, resulting in $\beta_m = 1/M$.

Based on these facts, we define the exponential approximated version of this loss function as defined bellow:

$$L(\mathbf{x}, f) = \exp(J(\mathbf{x}, f)) \tag{15}$$

$$L(\mathbf{x}, f + g) \approx L(\mathbf{x}, f)L(\mathbf{x}, g) \tag{16}$$

Now by substituting the (16) into the optimization step of Algorithm (1) and defining the sample's weight as the value of the loss function at the current stage of the algorithm, we can write:

$$g_m = \arg\min_{g} \sum_{n=1}^{N} w_n L(\mathbf{x}_n, g(\mathbf{x}_n)) \tag{17}$$

and

$$w_n = L(\mathbf{x}_n, f^{(m-1)}(\mathbf{x}_n)) \tag{18}$$

where w_n shows the nth sample's weight at the $(m-1)$th stage of the algorithm. As a result the inner optimization of the algorithm, (17), can be seen as fitting a clustering model using the sample weights at that stage of the algorithm. Practically speaking, the samples which are receiving higher error rates, namely *hard samples*, will get more emphasis in next iteration, while those with lower errors will be considered as less important or *easy samples*.

It is straightforward to derive the iterative formulation for calculation of the value of sample weights in next iteration as:

$$w_n \leftarrow w_n L(\mathbf{x}_n, g_m(\mathbf{x}_n)) \tag{19}$$

It should be noted that since the quantization error, (11), is always greater or equal to zero, the minimum of the exponential loss is 1. As a result, solely based on the update formula (19) the samples weight will always increase, however the normalization step which always accompanies the weight updates solves this problem.

2.2.2 Correlation If one can normalize the data space, so that $\forall \mathbf{x} : \|\mathbf{x}\| = 1$, we can further simplify the vector quantization error function (13) by only considering the correlation of the data samples and their respective cluster centers. As a result, the correlation based error functions could be written as follow:

$$J(\mathbf{x}, f) = -\mathbf{x}^T f(\mathbf{x}) \tag{20}$$

It is very easy to show that the exponential version of this loss function has the same properties presented in (16). Additionally, derivation of the minimization and sample's weight update stages follows exactly the same procedure described in (17) and (19).

Algorithm 2 CBoost.VQ: Clustering in a Boosting Framework using Vector Quantization Loss Function

1: Input dataset: $X = \{\mathbf{x}_n\}, n = 1, \ldots, N$.
2: Set: $W = \{w_n = 1/N\}, n = 1, \ldots, N$.
3: Initialize the model: $f^{(0)}(\mathbf{x}) = \mathbf{0}$.
4: **for** $m = 1$ to M **do**
5: Compute: $g_m(\mathbf{x}) = \arg\min_{g} \sum_{n=1}^{N} w_n L(\mathbf{x}_n, g(\mathbf{x}_n))$.
6: Set: $w_n \leftarrow w_n L(\mathbf{x}_n, g_m(\mathbf{x}_n))$ and renormalize afterward.
7: **end for**
8: Find the cluster correspondence for $\{g_m(\mathbf{x})\}$, and rearrange the partitions orders.
9: Output the final model: $f(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} g_m(\mathbf{x})$ and $i(\mathbf{x}) = \frac{1}{M}\sum_{m=1}^{M} j_m(\mathbf{x})$.

2.3 CBoost.VQ

Based on these principles, we introduce the *CBoost.VQ* algorithm, which could be seen as the clustering counterpart of famous boosting classifiers, specially GentleAdaBoost [9]. Algorithm (2) represents the overall additive modeling and learning processes.

2.4 Base Models

It is clear that in order to have a successful clustering performance based on the *CBoost.VQ* algorithm, there is a need for base clustering models capable of working with weighted samples. Unfortunately, not many of famous clustering methods are capable of making benefits of this additional information. Like [18, 10], for such algorithms, sub-sampling with replacement according to the sample weights could be used to reflect the effect of sample importance, however this will result in problems mentioned in Section 1.

Additionally, because of the additive nature of the algorithm which tries to compensate the shortcomings of the previous models in each step, it is more desirable to utilize simpler and computationally efficient base algorithms. Note there is no clear definition of a *weak clustering* model as what is conventional in classification domain, mainly because of lack of a concrete loss function. However, one can still refer to very simple clustering algorithms which are performing better than random partitioning as *weak* models.

Based on these facts, we briefly present a simple and computationally efficient clustering algorithm for base models which is a combination and modification of two existing algorithms, namely *Leaders* and *Agglomerative* methods. Recently, Viswanath and Jayasurya [19] used *Leaders* in an ensemble clustering context, based on an older existing algorithm described in [14]. Since this algorithm just uses a one-time scanning of the data samples in order to accomplish the partitioning of the data space, it is computationally very efficient. However because of the random and simple nature of the method, its results are usually considered very poor compared to the performance of other algorithms.

Before proceeding into details of the proposed methods, we present our base model in Algorithms (3) and (4). The clustering approach for *Leaders for Weighted Samples* is a

Algorithm 3 Leaders for Weighted Samples

Input dataset and sample weights: $X = \{\mathbf{x}_n\}, W = \{w_n\}, n = 1, \ldots, N$.
Initialize the leaders (clusters): $P = \{\}$.
Initialize the leader weights: $C = \{\}$.
Select the distance threshold: τ.
for $n = 1$ to N **do**
  Randomly select one sample, $\mathbf{x}_i$, using the weights as probability distribution, and remove it from the list of samples to be chosen later.
  if P is empty **then**
    Add $\mathbf{x}_i$ as the first leader: $P = \{\mathbf{x}_i\}$ and $C = \{w_i\}$.
  else
    Find the first leader which $\|\mathbf{x}_i - \mathbf{p}_j\| \leq \tau, \quad \mathbf{p}_j \in P$.
    if There was no $\mathbf{p}_j$ satisfying the distance criterion **then**
      Add $\mathbf{x}_i$ as the next leader: $P \leftarrow P \cup \{\mathbf{x}_i\}$ and $C \leftarrow C \cup \{w_i\}$.
    else
      Update: $\mathbf{p}_j \leftarrow \frac{c_j \mathbf{p}_j + w_i \mathbf{x}_i}{c_j + w_i}$ and $c_j \leftarrow c_j + w_i$.
    end if
  end if
end for
Output the Final Partitions: (P, C).

very straightforward center-based partitioning method: for any input sample $\mathbf{x}_i$, it finds the *first* leader (cluster center) which is close enough to the sample. Note that the algorithm even does not search for best matching leader, just the first encounter is selected (based on the leaders order in the set P). If there is no such a center, $\mathbf{x}_i$ itself joins the group of leaders. Obviously the chosen leaders depends largely on the order in which the samples are presented to the algorithm, and with the use of sample weights, we bias it toward those areas in the data space where the previous algorithms had difficulties clustering them. Furthermore, we calculate the next location of a leader using the sample weights belonging to that particular leader. Additionally, because of the random nature of the *Leaders* algorithm we expect to observe a large variance in the number of leaders and their locations.

However, there exists one major problem with the presented *Leaders* algorithm which makes it alone not useful for *CBoost.VQ* base models: there is no control on the number of cluster centers (leaders). As shown previously in Section 2.1, the weighted voting scheme requires each base model to deliver a fixed number of cluster centers. Assume that the number of leaders is $\|P\| = K_P$ while the number of desired clusters is K. With *Leaders* as base models, there could be three different situations: $K_P < K$, $K_P = K$, and $K_P > K$. For the first case, we have to try again or lower the distance threshold value, τ, until we get into second or third conditions. Obviously the second case is the desired situation, so there is no need for any other action. In order to solve the third problem (which practically is usually the case), we perform another *agglomerative* clustering [9] over the leaders in order to get a fixed number of cluster centers, as presented in Algorithm (4). Note that this algorithm is very similar to the *agglomerative* clustering method using *weighted average distance* linkage.

Algorithm 4 Agglomerative Clustering of Leaders

Input the *Leaders* model (P, C).
Input the desired number of final clusters, K.
for $k = 1$ to $K_P - K$ **do**
  Choose the closest leaders: $(i, j) = \arg\min_{i,j} \|p_i - p_j\|$.
  Merge these leaders: $p_{new} = \frac{c_i p_i + c_j p_j}{c_i + c_j}$ and $c_{new} = c_i + c_j$.
  Remove ith and jth leaders from P and C, and add (P_{new}, C_{new}) to the list of leaders.
end for
Output the Final Partitions (Leaders): P.

After formation of the hierarchy of leaders by applying agglomerative clustering over them, during the query stage, we find the first leader close enough to the sample given the distance threshold τ, and then use the hierarchy to find its label.

2.5 Cluster Correspondence

As mentioned before, after finishing the boosting iterations, we have to solve the cluster correspondence problem between the base models, $\{j_m \mathbf{x}\}$. Following [17, 18], we rearrange the orders of labels in all of the base models according to the first one. Since the number of the partitions are fixed for the base models, we first count the number of shared samples between clusters of both the first model and the next one in the ensemble. This gives us a $K \times K$ similarity matrix between pairs of partitions in two clustering algorithms. Then we use the *Hungarian* algorithm to find the best assignment for the cluster correspondence using the similarity matrix.

3 Experiments

3.1 Methodology

Since the clustering algorithms are trained in an unsupervised manner, it is natural to assess their qualities in an unsupervised settings too, but since many other similar works in this area use the classification error as the final quality measure, we adopt a similar approach in our methodology. In particular, we set the K in the algorithms to be the number of classes in the datasets, and use the Normalized Mutual Information, *NMI*, [15] between the true labels and the labels returned by the clustering algorithms as the quality assessment measure. It should be noted that the *NMI* value is bounded between zero and one, which correspond to a pure random and a perfect classification results, respectively.

3.2 Datasets

In order to show the performance of the proposed methods, we conducted experiments on a few synthetic and real-world datasets, which are summarized in Table 1. Both of 3-Gaussians and Half-rings are synthetic datasets, while Iris

Dataset	Classes	Features	Samples	*k-means*
3-Gaussians	3	2	1500	0.846 ± 0.000
Half-rings	2	2	1000	0.645 ± 0.000
Iris	3	4	150	0.656 ± 0.015
Pendigit	10	16	7494	0.692 ± 0.017

Table 1: Datasets used in the experiments: The second column shows the number classes, the third column shows the number of features, the third column shows the number of samples in the dataset, and the last column indicated the NMI value of the traditional *k-means* algorithm.

Dataset	Leaders	CBoost.VQ+Leaders
3-Gaussians	0.572 ± 0.105	0.863 ± 0.021
Half-rings	0.624 ± 0.249	0.924 ± 0.000
Iris	0.523 ± 0.104	0.715 ± 0.005
Pendigit	0.546 ± 0.092	0.726 ± 0.089

Table 2: Experimental results of base models (second column) and boosted version of them (third column). All experiments has been conducted 50 times independently and the values shown in this table are the average and standard deviation of the *NMI* measure.

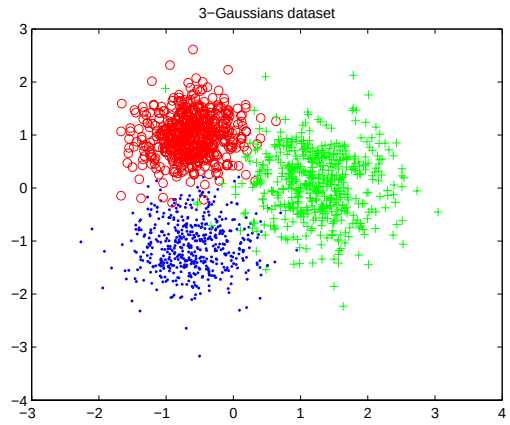
and Pendigit datasets are obtained from UCI machine learning repositories [1]. With the first two datasets we want to demonstrate the performance of the proposed algorithms on low dimensional data spaces, which enables a better visualizations, compared to later real world datasets which are high dimensional. The 3-Gaussians dataset is generated from samples drawn randomly from 3 different Gaussian distributions with different mean and variance parameters, and is shown in Figure 1(a). The Half-rings dataset is shown in Figure 1(b). All datasets have been standardized before performing any clustering over them, i.e. the mean of the features have been removed and then divided by their standard deviations.

3.3 Results

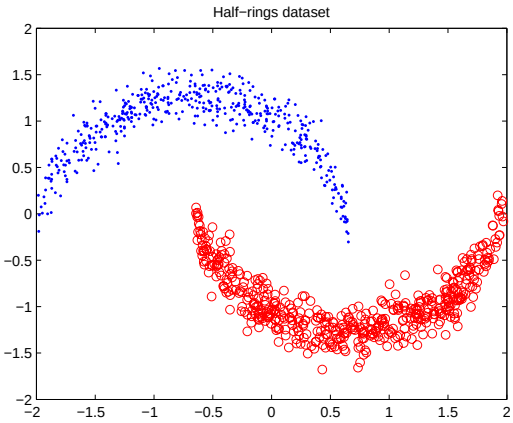
The experimental results of the described algorithms on these datasets are shown in Table 2, for 50 times independent simulations using the quantization error as the loss function. The second column of this table shows the mean and standard deviation of the performance of the individual *Leaders*. For these simulations we used a uniform distribution of the sample's weights, giving equal importance to every sample in the dataset. As it can be seen from the results, the performance of the *Leaders* has a very high variance and performs on average worse than *k-means*, Table 1.

Now looking at the third column of the Table 2 shows the results obtained by using *CBoost.VQ* algorithm incorporating the *Leaders* as base models. One can clearly observe the improvements both in terms of average performance and also stability of the results (very low variance).

Figure 2 shows the Half-rings dataset and the clustering results using *k-means*, *Leaders*, and *CBoost.VQ* with *Leaders* as base models. Note that this dataset is considered to be a difficult example for center-based algorithms, as it can be seen from the performance of the *k-means*.



(a) 3-Gaussians



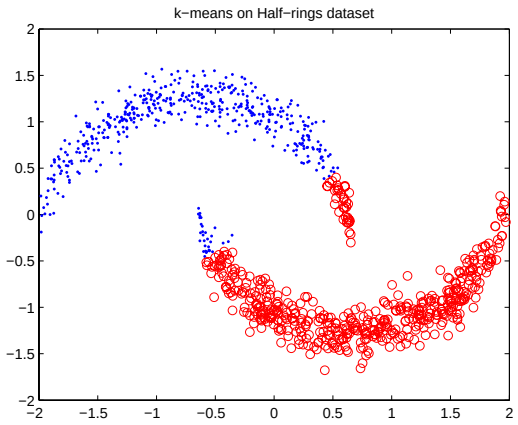
(b) Half-rings

Figure 1: Synthetic datasets used in experiments: (a): 3-Gaussians, and (b): Half-rings.

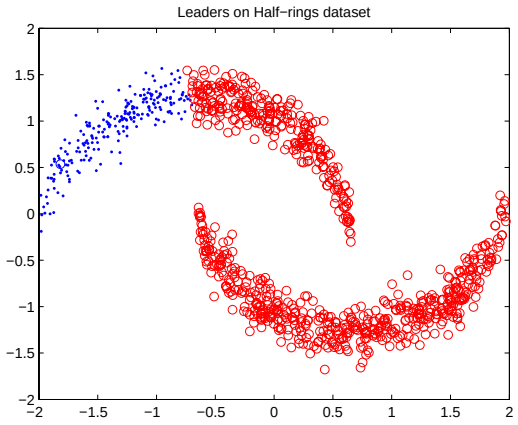
Figure 3 shows the average quantization error value (normalized with the number of samples in the dataset), with respect to the addition of each base model during the boosting iterations for the 3-Gaussians dataset. It is obvious that with each iteration of boosting and addition of a new base model the average error is decreasing, which demonstrates the success of the forward stage-wise optimization of the proposed loss function using the additive model proposed for the clustering tasks. Another observation is the convergence of the total quantization error to almost the same error value of a converged *k-means* (the red-dashed line in this figure).

4 Conclusion

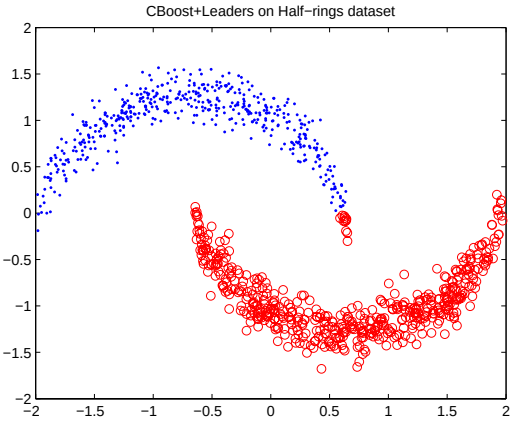
In this paper, we presented a novel approach for clustering datasets in a boosting framework. The proposed methods are based on theoretical concepts of optimization using additive models in a forward stage-wise approach. We provided solutions for optimizing the quantization loss function which is used widely in center-based clustering algorithms. The experimental results also suggest the good performance of these methods on a few synthetic and real world datasets. However, we would like to extend the application of these



(a) K-Means



(b) Leaders



(c) CBoost.VQ+Leaders

Figure 2: Results of (a): *k-means*, (b): *Leaders*, and (c): *CBoost.VQ+Leaders* on Half-rings dataset.

algorithms into partitioning large-scale datasets. The motivation behind moving to large-scale real-world datasets comes from the fact that the base models which we used in this paper, are computationally very efficient methods, and a clever combination of their results using the boosting framework could lead to a better performance than what

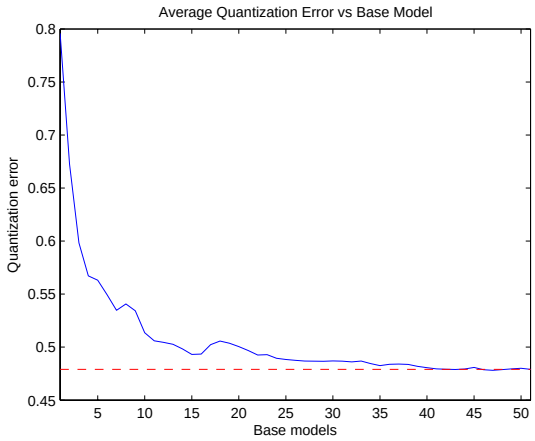


Figure 3: Average vector quantization error versus the number of base models in the boosting iterations. The red-dashed line is the average quantization error of the *k-means* after convergence.

we could expect from traditional algorithms, in both quality of the clustering and computational costs. On average, during the experiments for evaluation purposes, the *CBoost.VQ* algorithm was almost two times faster than the *k-means*, and it should be noted that our code is still in its non-optimized stage, so we expect to achieve a higher computational speed in future.

Additionally we would like to investigate the relationships between these methods and the other class of unsupervised methods named density estimator, since one can think of local base models as local density estimators, for example similar to local modes of mean shift algorithm. However further research into this direction is needed.

Acknowledgement

This work has been supported by the FWF Austrian Joint Research Project Cognitive Vision under projects S9103-N04 and S9104-N04 and the EU FP6-507752 NoE MUSCLE IST project.

References

[1] C.L. Blake D.J. Newman, S. Hettich and C.J. Merz. UCI repository of machine learning databases.

[2] S. Dudoit and J. Fridlyand. Bagging to improve the accuracy of a clustering procedure. *Bioinformatics*, 19(9):1090–1099, 2003.

[3] R. Fergus. *Visual Object Category Recognition*. PhD thesis, Robotics Research Group, Department of Engineering Science, University of Oxford, 2005.

[4] B. Fischer and J.M. Buhmann. Path-based clustering for grouping of smooth curves and texture segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 25(4):513–518, 2003.

[5] David A. Forsyth and Jean Ponce. *Computer Vision: A Modern Approach*. Prentice Hall, 2003.

[6] A. Fred. Finding consistent clusters in data partitions. In Josef Kittler and Fabio Roli, editors, *Proceedings of*

Third International Workshop on Multiple Classifier Systems, volume LNCS 2096, pages 309–318. Springer, 2001.

[7] A. Fred and A.K. Jain. Data clustering using evidence accumulation. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, pages 276–280, 2002.

[8] Y. Freund and R. Schapire. Experiments with a new boosting algorithm. In *Proceedings of the Thirteenth International Conference on Machine Learning (ICML)*, pages 148–156, 1996.

[9] Jerome Friedman, Trevor Hastie, and Robert Tibshirani. Additive logistic regression: a statistical view of boosting. *The Annals of Statistics*, 38(2):337–374, 2000.

[10] D. Frossyniotis, A. Likas, and A. Stafylopatis. A clustering method based on boosting. *Pattern Recognition Letters*, 25:641–654, 2004.

[11] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Springer-Verlag, 2001.

[12] Anil K. Jain and Martin H. C. Law. Data clustering : A user's dilemma. In *Proceedings of International Conference on Pattern recognition and Machine Intelligence (PReMI)*, 2005.

[13] B. Minaei-Bidgoli, A. Topchy, and W.F. Punch. Ensembles of partitions via data resampling. In *Proceedings of International Conference on Information Technology: Coding and Computing (ITCC)*, 2004.

[14] H. Spath. *Cluster Analysis Algorithms for Data Reduction and Classification*. Ellis Horwood, Chichester, UK., 1980.

[15] A. Strehl and J. Ghosh. Cluster ensembles-a knowledge reuse framework for combining multiple partitions. *Journal of Machine Learning Research*, 3:583–617, 2002.

[16] A. Topchy, A. K. Jain, and W. Punch. Clustering ensembles: Models of consensus and weak partitions. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 27(12):1866–1881, 2005.

[17] A.P. Topchy, M.H.C. Law, A.K. Jain, and A.L. Fred. Analysis of consensus partition in cluster ensemble. In *Proceedings of IEEE International Conference on Data Mining (ICDM)*, 2004.

[18] A. Topchy, B. Minaei-Bidgoli, A. K. Jain, and W. F. Punch. Adaptive clustering ensembles. In *Proceedings of the International Conference on Pattern Recognition (ICPR)*, 2004.

[19] P. Viswanath and K. Jayasurya. A fast and efficient ensemble clustering method. In *Proceedings of International Conference on Pattern Recognition (ICPR)*, pages 720–723, 2006.

[20] U. von Luxburg and S. Ben-David. Towards a statistical theory for clustering. In *PASCAL Workshop on Statistics and Optimization of Clustering, London, UK*, 2005.

[21] S. Zhong and J. Ghosh. A unified framework for model-based clustering. *Journal of Machine Learning Research*, 4:1001–1037, 2003.