Abstract

We study the design of interactive clustering algorithms for data sets satisfying natural stability assumptions. Our algorithms start with any initial clustering and only make local changes in each step; both are desirable features in many applications. We show that in this constrained setting one can still design provably efficient algorithms that produce accurate clusterings. We also show that our algorithms perform well on real-world data.

1. Introduction

Clustering is usually studied in an unsupervised learning scenario where the goal is to partition the data given pairwise similarity information. Designing provably-good clustering algorithms is challenging because given a similarity function there may be multiple plausible clusterings of the data. Traditional approaches resolve this ambiguity by making assumptions on the data-generation process. For example, there is a large body of work that focuses on clustering data that is generated by a mixture of Gaussians (Achlioptas & McSherry, 2005; Kannan et al., 2005; Dasgupta, 1999; Arora & Kannan, 2001; Brubaker & Vempala, 2008; Kalai et al., 2010; Moitra & Valiant, 2010; Belkin & Sinha, 2010). Although this helps define the “right” clustering one should be looking for, real-world data rarely comes from such well-behaved probabilistic models. An alternative approach is to use limited user supervision to help the algorithm reach the desired answer. This approach has been facilitated by the availability of cheap crowd-sourcing tools in recent years. In certain applications such as search and document classification, where users are willing to help a clustering algorithm arrive at their own desired answer with a small amount of additional prodding, interactive algorithms are very useful. Hence, the study of interactive clustering algorithms has become an exciting new area of research.

In many practical settings we already start with a fairly good clustering computed with semi-automated techniques. For example, consider an online news portal that maintains a large collection of news articles. The news articles are clustered on the “back-end,” and are used to serve several “front-end” applications such as recommendations and article profiles. For such a system, we do not have the freedom to compute arbitrary clusterings and present them to the user, which has been proposed in prior work. But it is still feasible to get limited feedback and locally edit the clustering. In particular, we may only want to change the “bad” portion revealed by the feedback without changing the rest of the clustering. Motivated by these observations, in this paper we study the problem of designing local algorithms for interactive clustering.

We propose a theoretical interactive model and provide strong experimental evidence supporting the practical applicability our algorithms. In our model we start with an initial clustering of the data. The algorithm then interacts with the user in stages. In each stage the user provides limited feedback on the current clustering in the form of split and merge requests. The algorithm then makes a local edit to the clustering that is consistent with user feedback. Such edits are aimed at improving the problematic part of the clustering pointed out by the user. The goal of the algorithm is to quickly converge (using as few requests as possible) to a clustering that the user is happy with - we call this clustering the target clustering.

In our model the user may request a certain cluster to be split if it is overclustered (intersects two or more clusters in the target clustering). The user may also request to merge two given clusters if they are underclustered (both intersect the same target cluster). Note that the user may not tell the algorithm how to perform the split or the merge; such in-
Local algorithms for interactive clustering

put is infeasible because it requires a manual analysis of all
the objects in the corresponding clusters. We also restrict
the algorithm to only make local changes at each step, i.e.,
in response we may change only the cluster assignments
of the points in the corresponding clusters. If the user requests
to split a cluster $C_i$, we may change only the cluster assign-
ments of points in $C_i$, and if the user requests to merge $C_i$
and $C_j$, we may only reassign the points in $C_i$ and $C_j$.

The split and merge requests described above are a natu-
ral form of feedback. It is easy for users to spot over/underclus-
tering issues and request the corresponding splits/merges (without having to provide any additional infor-
mation about how to perform the edit). For our model to be
practically applicable, we also need to account for noise in
the user requests. In particular, if the user requests a merge,
only a fraction or a constant number of the points in
the two clusters may belong to the same target cluster. Our
model (See Section 2) allows for such noisy user responses.

We study the complexity of algorithms in the above model
(the number of edits requests needed to find the target clus-
tering) as a function of the error of the initial cluster-
ing. The initial error may be evaluated in terms of under-
clustering error $\delta_u$ and overclustering error $\delta_o$ (See
Section 2). The initial error may be fairly small: given two
$k$-clusterings, $\delta_u$ and $\delta_o$ is at most $k(k-1)$. Therefore
we would like to develop algorithms whose complexity de-
pends polynomially on $\delta_u$, $\delta_o$ and only logarithmically on
$n$, the number of data points. We show that this is indeed
possible given that the target clustering satisfies a natural
stability property (see Section 2). We also develop algo-
rimths for the well-known correlation-clustering objective
function (Bansal et al., 2004), which considers pairs of
points that are clustered inconsistently with respect to the
target clustering (See Section 2).

As a pre-processing step, our algorithms compute the
average-linkage tree of all the points in the data set. Note
that if the target clustering $C^*$ satisfies our stability as-
sumption, then the average-linkage tree must be consist-
ent with $C^*$ (see Section 3). However, in practice this average-
linkage tree is much too large to be directly interpreted by
the users. Still, given that the edit requests are somewhat
consistent with $C^*$, we can use this tree to efficiently com-
pute local edits that are consistent with the target clustering.
Our analysis then shows that after a limited number of edit
requests we must converge to the target clustering.

Our Results
In Section 3 we study the $\eta$-merge model. We assume that
the user may request to split a cluster $C_i$ only if $C_i$ contains
points from several ground-truth clusters. The user may
request to merge $C_i$ and $C_j$ only if an $\eta$-fraction of points
in each $C_i$ and $C_j$ are from the same ground-truth cluster.

For this model for $\eta > 0.5$, given an initial clustering with
overclustering error $\delta_o$ and underclustering error $\delta_u$,
we present an algorithm that requires $\delta_u$ split requests and
$2(\delta_u + k) \log \frac{1}{\epsilon} n$ merge requests to find the target clus-
tering, where $n$ is the number of points in the dataset. For $\eta > 2/3$, given an initial clustering with correlation-
clustering error $\delta_{cc}$, we present an algorithm that requires
at most $\delta_{cc}$ edit requests to find the target clustering.

In Section 4 we relax the condition on the merges and al-
low the user to request a merge even if $C_i$ and $C_j$ only
have a single point from the same target cluster. We call
this the unrestricted-merge model. Here the requirement
on the accuracy of the user response is much weaker and
we need to make further assumptions on the nature of the
requests. More specifically, we assume that each merge re-
quest is chosen uniformly at random from the set of feas-
ible merges. Under this assumption we present an algorithm
that with probability at least $1 - \epsilon$ requires $\delta_u$ split requests and $O(\log^2 \frac{1}{\epsilon} \delta_u^2)$ merge requests to find the target clustering.

We develop several algorithms for performing the split and
merge requests under different assumptions. Each algo-

rithm uses the global average-linkage tree $T_{avg}$ to com-
pute a local clustering edit. Our splitting procedure finds
the node in $T_{avg}$ where the corresponding points are first
split in two. It is more challenging to develop a correct
merge procedure, given that we allow “impure” merges,
where one or both clusters have points from another tar-
get cluster (other than the one that they both intersect). To
perform such merges, in the $\eta$-merge model we develop a
procedure to extract the “pure” subsets of the two clus-
ters, which must only contain points from the same tar-
get cluster. Our procedure searches for the deepest node
in $T_{avg}$ that has enough points from both clusters. In the
unrestricted-merge model, we develop another merge pro-
cedure that either merges the two clusters or merges them
and splits them. This algorithm always makes progress if
the proposed merge is “impure,” and makes progress on
average if it is “pure” (both clusters are subset of the same
target cluster).

In Section 5 we demonstrate the effectiveness of our algo-
rithms on real data. We show that for the purposes of
splitting known over-clusters, our splitting procedure com-
putes the best splits, when compared to other well-known
techniques. We also test the entire proposed framework
on newsgroup documents data, which is quite challenging
for traditional unsupervised clustering methods (Telgarsky
& Dasgupta, 2012; Heller & Ghahramani, 2005; Dasgupta
& Hsu, 2008; Dai et al., 2010; Boulis & Ostendorf, 2004;
Zhong, 2005). Still, we find that our algorithms perform
fairly well; for larger settings of $\eta$ we are able find the tar-
get clustering after a limited number of edit requests.

Related work
Interactive models for clustering studied in previous works (Balcan & Blum, 2008; Awasthi & Zadeh, 2010) were inspired by an analogous model for learning under feedback (Angluin, 1998). In this model, the algorithm can propose a hypothesis to the user (in this case, a clustering of the data) and get some feedback regarding the correctness of the current hypothesis. As in our model, the feedback considered is split and merge queries. The goal is to design efficient algorithms which use very few queries to the user. A critical limitation in prior work is that the algorithm has the freedom to choose any arbitrary clustering as the starting point and can make arbitrary changes at each step. Hence these algorithms may propose a series of “bad” clusterings to the user to quickly prune the search space and reach the target clustering. Our interactive clustering model is in the context of an initial clustering; we are restricted to only making local changes to this clustering to correct the errors pointed out by the user. This model is well-motivated by several applications, including the Google application described in the experimental section.

Basu et al. (Basu et al., 2004) study the problem of minimizing the $k$-means objective in the presence of limited supervision. This supervision is in the form of pairwise must-link and cannot-link constraints. They propose a variation of the Lloyd’s method for this problem and show promising experimental results. The split/merge requests that we study are a more natural form of interaction because they capture macroscopic properties of a cluster. Getting pairwise constraints among data points involves much more effort on the part of the user and is unrealistic in many scenarios.

The stability property that we consider is a natural generalization of the “stable marriage” property (see Definition 2.2) that has been studied in a variety of previous works (Balcan et al., 2008; Bryant & Berry, 2001). It is the weakest among the stability properties that have been studied recently such as strict separation and strict threshold separation (Balcan et al., 2008; Krishnamurthy et al., 2012). This property is known to hold for real-world data. In particular, (Voevodskii et al., 2012) observed that this property holds for protein sequence data, where similarities are computed with sequence alignment and ground truth clusters correspond to evolutionary-related proteins.

### 2. Notation and Preliminaries

Given a data set $X$ of $n$ points we define $C = \{C_1, C_2, \ldots C_k\}$ to be a $k$-clustering of $X$ where the $C_i$’s represent the individual clusters. Given two clusterings $C$ and $C'$, we define the distance between a cluster $C_i \in C$ and the clustering $C'$ as:

$$\text{dist}(C_i, C') = |\{C'_j \in C' : C'_j \cap C_i \neq \emptyset\}| - 1.$$ 

This distance is the number of additional clusters in $C'$ that contain points from $C_i$; it evaluates to 0 when all points in $C_i$ are contained in a single cluster in $C'$. Naturally, we can then define the distance between $C$ and $C'$ as:

$$\text{dist}(C, C') = \sum_{C_i \in C} \text{dist}(C_i, C').$$

Notice that this notion of clustering distance is asymmetric: $\text{dist}(C, C') \neq \text{dist}(C', C)$. Also note that $\text{dist}(C, C') = 0$ if and only if $C$ refines $C'$. Observe that if $C$ is the ground-truth clustering, and $C'$ is a proposed clustering, then $\text{dist}(C, C')$ can be considered an underclustering error, and $\text{dist}(C', C)$ an overclustering error.

An underclustering error is an instance of several clusters in a proposed clustering containing points from the same ground-truth cluster; this ground-truth cluster is said to be underclustered. Conversely, an overclustering error is an instance of points from several ground-truth clusters contained in the same cluster in a proposed clustering; this proposed cluster is said to be overclustered. In the following sections we use $C^* = \{C^*_1, C^*_2, \ldots C^*_k\}$ to refer to the ground-truth clustering, and use $C$ to refer to the initial clustering. We use $\delta_u$ to refer to the underclustering error of the initial clustering, and $\delta_o$ to refer to the overclustering error. In other words, we have $\delta_u = \text{dist}(C^*, C)$ and $\delta_o = \text{dist}(C, C^*)$.

We also observe that we can naturally define the distance between two clusterings using the correlation-clustering objective function. Given a proposed clustering $C'$, and a ground-truth clustering $C^*$, we define the correlation-clustering error $\delta_{cc}$ as the number of (ordered) pairs of points that are clustered inconsistently with $C^*$:

$$\delta_{cc} = |\{(u, v) \in X \times X : c(u, v) \neq c^*(u, v)\}|,$$

where $c(u, v) = 1$ if $u$ and $v$ are in the same cluster in $C$, and 0 otherwise; $c^*(u, v) = 1$ if $u$ and $v$ are in the same cluster in $C^*$, and 0 otherwise. Note that we may divide the correlation-clustering error $\delta_{cc}$ into overclustering component $\delta_{cco}$ and underclustering component $\delta_{cuc}$:

$$\delta_{cco} = |\{(u, v) \in X \times X : c(u, v) = 1 \text{ and } c^*(u, v) = 0\}|$$

$$\delta_{cuc} = |\{(u, v) \in X \times X : c(u, v) = 0 \text{ and } c^*(u, v) = 1\}|$$

In our formal analysis we model the user as an oracle that provides edit requests.

**Definition 2.1 (Local algorithm).** We say that an interactive clustering algorithm is local if in each iteration only the cluster assignments of points involved in the oracle request may be changed. If the oracle requests to split $C_i$, the algorithm may only reassign the points in $C_i$. If the oracle requests to merge $C_i$ and $C_j$, the algorithm may only reassign the points in $C_i \cup C_j$.

We next formally define the stability property of a clustering that we study in this work.
Definition 2.2 (Stability). Given a clustering \( C = \{C_1, C_2, \cdots C_k\} \) over a domain \( X \) and a similarity function \( S : X \times X \rightarrow \mathbb{R} \), we say that \( C \) satisfies stability if for all \( i \neq j \), and for all \( A \subset C_i \) and \( A' \subset C_j \), \( S(A, C_i \setminus A) > S(A, A') \), where for any two sets \( A, A' \), \( S(A, A') = E_{x \in A, y \in A'} S(x, y) \).

In the following sections we will assume that the ground-truth clustering of the data set satisfies this stability property, and we have access to the corresponding similarity function. In order for our algorithms to make progress, the oracle requests must be somewhat consistent with the target clustering.

Definition 2.3 (\( \eta \)-merge model). In the \( \eta \)-merge model the oracle requests have the following properties

- \( \text{split}(C_i): C_i \) contains points from two or more target clusters.
- \( \text{merge}(C_i, C_j): \) At least an \( \eta \)-fraction of the points in each \( C_i \) and \( C_j \) belong to the same target cluster.

Definition 2.4 (Unrestricted-merge model). In the unrestricted-merge model the oracle requests have the following properties

- \( \text{split}(C_i): C_i \) contains points from two or more target clusters.
- \( \text{merge}(C_i, C_j): \) At least 1 point in each \( C_i \) and \( C_j \) belongs to the same target cluster.

Note that the assumptions about the nature of the split requests are the same in both models. In the \( \eta \)-merge model, the oracle may request to merge two clusters if both have a constant fraction of points from the same target cluster. In the unrestricted-merge model, the oracle may request to merge two clusters if both have some points from the same target cluster.

3. The \( \eta \)-merge model

In this section we describe and analyze the algorithms in the \( \eta \)-merge model. As a pre-processing step for all our algorithms, we first run the average-linkage algorithm on all the points in the data set to compute the global average-linkage tree, which we denote by \( T_{\text{avg}} \). The leaf nodes in this tree contain the individual points, and the root node contains all the points. The tree is computed in a bottom-up fashion: starting with the leaves in each iteration the two most similar nodes are merged, where the similarity between two nodes \( N_1 \) and \( N_2 \) is the average similarity between points in \( N_1 \) and points in \( N_2 \).

We assign a label “impure” to each cluster in the initial clustering; these labels are used by the merge procedure. Given a split or merge request, a local clustering edit is computed from the average-linkage tree \( T_{\text{avg}} \) as described in Figure 1 and Figure 2.

Figure 1. Split procedure

<table>
<thead>
<tr>
<th>Algorithm: SPLIT PROCEDURE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Cluster ( C_i ), global average-linkage tree ( T_{\text{avg}} ).</td>
</tr>
<tr>
<td>1. Search ( T_{\text{avg}} ) to find the node ( N ) at which the set of points in ( C_i ) are first split in two.</td>
</tr>
<tr>
<td>2. Let ( N_1 ) and ( N_2 ) be the children of ( N ). Set ( C_{i,1} = N_1 \cap C_i ), ( C_{i,2} = N_2 \cap C_i ).</td>
</tr>
<tr>
<td>3. Delete ( C_i ) and replace it with ( C_{i,1} ) and ( C_{i,2} ). Mark the two new clusters as “impure”.</td>
</tr>
</tbody>
</table>

Figure 2. Merge procedure

<table>
<thead>
<tr>
<th>Algorithm: MERGE PROCEDURE</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Input:</strong> Clusters ( C_i ) and ( C_j ), global average-linkage tree ( T_{\text{avg}} ).</td>
</tr>
<tr>
<td>1. If ( C_i ) is marked as “pure” set ( \eta_i = 1 ) else set ( \eta_i = \eta ). Similarly set ( \eta_j ) for ( C_j ).</td>
</tr>
<tr>
<td>2. Search ( T_{\text{avg}} ) for a node of maximal depth ( N ) that contains enough points from ( C_i ) and ( C_j ): (</td>
</tr>
<tr>
<td>3. Replace ( C_i ) by ( C_i \setminus N ), replace ( C_j ) by ( C_j \setminus N ).</td>
</tr>
<tr>
<td>4. Add a new cluster containing ( N \cap (C_i \cup C_j) ), mark it as “pure”.</td>
</tr>
</tbody>
</table>

To implement Step 1 in Figure 1, we start at the root of \( T_{\text{avg}} \) and “follow” the points in \( C_i \) down one of the branches until we find a node that splits them. In order to implement Step 2 in Figure 2, it suffices to start at the root of \( T_{\text{avg}} \) and perform a post-order traversal, only considering nodes that have “enough” points from both clusters, and return the first output node.

The split procedure is fairly intuitive: if the average-linkage tree is consistent with the target clustering, it suffices to find the node in the tree where the corresponding points are first split in two. It is more challenging to develop a correct merge procedure: note that Step 2 in Figure 2 is only correct if \( \eta > 0.5 \), which ensures that if two nodes in the tree have more than an \( \eta \)-fraction of the points from \( C_i \) and \( C_j \), one must be an ancestor of the other. If the average-linkage tree is consistent with the ground-truth, then clearly the node equivalent to the corresponding target cluster (that \( C_i \) and \( C_j \) both intersect) will have enough points from \( C_i \) and \( C_j \); therefore the node that we find in Step 2 must be this node or one of its descendants. In addition, because our merge procedure replaces two clusters with three, we require pure/impure labels for the merge requests to terminate: “pure” clusters may only have other points added to them, and retain this label throughout the execution of the algorithm.
We now state the performance guarantee for these split and merge algorithms.

**Theorem 3.1.** Suppose the target clustering satisfies stability, and the initial clustering has over-clustering error \( \delta_o \) and under-clustering error \( \delta_u \). In the \( \eta \)-merge model, for any \( \eta > 0.5 \), the algorithms in Figure 1 and Figure 2 require at most \( \delta_o \) split requests and \( 2(\delta_u + k) \log \frac{1}{\Delta^2} \eta \) merge requests to find the target clustering.

In order to prove the theorem, we observe that if the target clustering satisfies stability, then every node of the average-linkage tree must be laminar (consistent) with respect to the ground-truth clustering. We next formally state these observations.

**Definition 3.2 (Laminar).** A node \( N \) is laminar with respect to the ground-truth clustering \( C^* \) if for each cluster \( C_i^* \in C^* \) we have either \( N \cap C_i^* = \emptyset \), \( N \subset C_i^* \), or \( C_i^* \subset N \).

**Lemma 3.3.** Suppose the ground-truth clustering \( C^* \) over a domain \( X \) satisfies stability with respect to a similarity function \( S \). Let \( T_{avg} \) be the average-linkage tree for \( X \) constructed with \( S \). Then every node in \( T_{avg} \) is laminar w.r.t. \( C^* \).

It follows that the split computed by the algorithm in Figure 1 must also be consistent with the target clustering; we call such splits clean.

**Definition 3.4 (Clean split).** A partition (split) of a cluster \( C_i \) into clusters \( C_{i,1} \) and \( C_{i,2} \) is said to be clean if \( C_{i,1} \) and \( C_{i,2} \) are non-empty, and for each ground-truth cluster \( C_j^* \) such that \( C_j^* \cap C_i \neq \emptyset \), either \( C_j^* \cap C_{i,1} = C_j^* \cap C_{i,2} \) or \( C_j^* \cap C_i = C_j^* \cap C_{i,1} \cup C_{i,2} \).

We now prove the correctness of the split/merge procedures.

**Lemma 3.5.** If the ground-truth clustering satisfies stability and \( \eta > 0.5 \) then,

- **a.** The split procedure in Figure 1 always produces a clean split.
- **b.** The new cluster added in Step 4 in Figure 2 must be “pure”, i.e., it must contain points from a single ground-truth cluster.

**Proof.** **a.** For purposes of contradiction, suppose the returned split is not clean: \( C_{i,1} \) and \( C_{i,2} \) contain points from the same ground-truth cluster \( C_j^* \). It must be the case that \( C_i \) contains points from several ground-truth clusters, which implies that w.l.o.g. \( C_{i,1} \) contains points from some other ground-truth cluster \( C_{l,j}^* \). This implies that \( N_1 \) is not laminar w.r.t. \( C^* \), which contradicts Lemma 3.3. **b.** By our assumption, at least \( \frac{1}{2} |C_i| \) points from \( C_i \) and \( \frac{1}{2} |C_j| \) points from \( C_j \) are from the same ground-truth cluster \( C_j^* \). Clearly, the node \( N' \) in \( T_{avg} \) that is equivalent to \( C_j^* \) (which contains all the points in \( C_j^* \) and no other points) must contain enough points from \( C_i \) and \( C_j \), and only ascendants and descendants of \( N' \) may contain more than an \( \eta \) fraction of points from both clusters. Thus the node \( N \) that we find with a depth-first search must be \( N' \) or one of its descendants, and will only contain points from \( C_j^* \).

Using the above lemma, we can prove the bounds on the split and merge requests stated in Theorem 3.1.

**Proof Sketch:** To bound the number of split requests, we observe that each clean split reduces the over-clustering error by exactly 1. Let us call a requested merge impure if one or both of the clusters are marked “impure,” and pure otherwise. To bound the number of impure merge requests, consider the pure sets \( P = \{ C_i \cap C_j \mid C_i \text{ is marked “impure” and } C_i \cap C_j \neq \emptyset \} \), which correspond to the pure subsets of clusters that are marked “impure”. We observe that each impure merge must reduce the size of one of the sets in \( P \) by an \( \eta \)-fraction. The number of pure merges is upper bounded by the number of pure clusters, which is at most the number of impure merge requests.

To bound the number of edit requests with respect to the correlation clustering objective, we must use a different merge procedure, which is described in Figure 3. Here in-

![Figure 3](image.png)

**Algorithm:** MERGE PROCEDURE  
**Input:** Clusters \( C_i \) and \( C_j \), global average-linkage tree \( T_{avg} \)  
Search \( T_{avg} \) for a node of maximal depth \( N \) that contains enough points from \( C_i \) and \( C_j \):  
- \( |N \cap C_i| \geq \eta |C_i| \) and \( |N \cap C_j| \geq \eta |C_j| \)  
- if \( |C_i| \geq |C_j| \) then  
  Replace \( C_i \) by \( C_i \cup (N \cap C_j) \)  
  Replace \( C_j \) by \( C_j \setminus N \)  
else  
  Replace \( C_i \) by \( C_i \setminus N \)  
  replace \( C_j \) by \( C_j \cup (N \cap C_i) \)  
end if

Instead of creating a new “pure” cluster, we add these points to the larger of the two clusters in the merge. Using this merge procedure and the split procedure presented earlier gives the following performance guarantee.

**Theorem 3.6.** Suppose the target clustering satisfies stability, and the initial clustering has correlation-clustering error of \( \delta_{cc} \). In the \( \eta \)-merge model, for any \( \eta > 2/3 \), using the split and merge procedures in Figures 1 and 3 requires at most \( \delta_{cc} \) edit requests to find the target clustering.
Proof Sketch: We can verify that splitting \( C_i \) may not increase \( \delta_{ccu} \), but it must decrease \( \delta_{cco} \) by at least \( |C_i| \). For the merge, given that the reassigned points must be “pure” (all come from the same target cluster), we can verify that \( \delta_{ccu} \) must decrease if \( \eta > 2/3 \).

When the data satisfies stronger stability properties we may simplify the presented algorithms and/or obtain better performance guarantees. In particular, if the data satisfies the strict separation property from (Balcan et al., 2008), we may change the split and merge algorithms to use the local single-linkage tree (constructed from only the points in the edit request). In addition, if the data satisfies strict threshold separation, we may remove the restriction on \( \eta \) and use a different merge procedure that is correct for any \( \eta > 0 \).

4. The unrestricted-merge model

In this section we further relax the assumptions about the nature of the oracle requests. As before, the oracle may request to split a cluster if it contains points from two or more target clusters. For merges, now the oracle may request to merge \( C_i \) and \( C_j \) if both clusters contain only a single point from the same ground-truth cluster. We note that this is a minimal set of assumptions for a local algorithm to make progress, otherwise the oracle may always propose irrelevant splits or merges that cannot reduce clustering error.

For this model we propose the merge algorithm described in Figure 4. The split algorithm remains the same as in Figure 1. To provably find the ground-truth clustering in this setting we require that each merge request must be chosen uniformly at random from the set of feasible merges. This assumption is consistent with the observation in (Awasthi & Zadeh, 2010) that in the unrestricted-merge model with arbitrary request sequences, even very simple cases (ex. union of intervals on a line) require a prohibitively large number of requests. We do not make additional assumptions about the nature of the split requests; in each iteration any feasible split may be proposed by the oracle. In this setting our algorithms have the following performance guarantees.

Theorem 4.1. Suppose the target clustering satisfies stationarity, and the initial clustering has overclustering error \( \delta_0 \) and underclustering error \( \delta_u \). In the unrestricted-merge model, with probability at least \( 1 - \epsilon \), the algorithms in Figure 1 and Figure 4 require \( \delta_0 \), split requests and \( O(\log \frac{\log \frac{1}{\epsilon}}{\epsilon^2} \delta_u^2) \) merge requests to find the target clustering.

The above theorem is proved in a series of lemmas. We first state a lemma regarding the correctness of the Algorithm in Figure 4. We argue that if the algorithm merges \( C_i \) and \( C_j \), it must be the case that both \( C_i \) and \( C_j \) contain points from the same ground-truth cluster. The proofs of the lemmas below are omitted due to space constraints.

Lemma 4.2. If the algorithm in Figure 4 merges \( C_i \) and \( C_j \) in Step 3, it must be the case that \( C_i \subseteq C_i^* \) and \( C_j \subseteq C_j^* \) for some ground-truth cluster \( C_i^* \).

The \( \delta_0 \) bound on the number of split requests follows from the observation that each split reduces the overclustering error by exactly 1 (as before), and the fact that the merge procedure does not increase overclustering error.

Lemma 4.3. The merge algorithm in Figure 4 does not increase overclustering error.

The following lemmas bound the number of impure and pure merges. Here we call a proposed merge pure if both clusters are subsets of the same ground-truth cluster, and impure otherwise.

Lemma 4.4. The merge algorithm in Figure 4 requires at most \( \delta_u \) impure merge requests.

Lemma 4.5. The probability that the algorithm in Figure 4 requires more than \( O(\log \frac{\log \frac{1}{\epsilon}}{\epsilon^2} \delta_u^2) \) pure merge requests is less than \( \epsilon \).

5. Experimental Results

We perform two sets of experiments: we first test the proposed split procedure on the clustering of business listings maintained by Google, and also test the proposed framework in its entirety on the much smaller newsgroup documents data set.

5.1. Clustering business listings

Google maintains a large collection of data records representing businesses. These records are clustered using a similarity function; each cluster should contain records about the same distinct business; each cluster is summarized and served to users online via various front-end applications. Users report bugs such as “you are displaying the name of one business, but the address of another” (caused by over-clustering), or “a particular business is shown multiple times” (caused by under-clustering). These bugs are routed to operators who examine the contents of the corresponding clusters, and request splits/merges accordingly. The clusters involved in these requests may be
Local algorithms for interactive clustering

quite large and usually contain records about several businesses. Therefore automated tools that can perform the requested edits are very helpful.

In particular, here we evaluate the effectiveness of our split procedure in computing correct cluster splits. We consider a binary split correct iff the two resulting sub-clusters are “clean” using Definition 3.4. Note that a clean split is sufficient and necessary for reducing the clustering error (measured by $\delta_u + \delta_o$, see Section 2). To compute the splits, we use a “local” variation of the algorithm in Figure 1, where we use the average-linkage tree built only from the points in the cluster (referred to as Clean-Split). This variation is easier to implement and run, but still gives a provably “clean” split for stronger assumptions on the data.

For comparison purposes, we use two well-known techniques for computing binary splits: the optimal 2-median (2-Median), and a “sweep” of the second-smallest eigenvector of the corresponding Laplacian matrix. Let $\{v_1, \ldots, v_n\}$ be the order of the vertices when sorted by their eigenvector entries, we compute the partition $\{v_1, \ldots, v_i\}$ and $\{v_{i+1}, \ldots, v_n\}$ such that its conductance is smallest (Spectral-Balanced), and a partition such that the similarity between $v_i$ and $v_{i+1}$ is smallest (Spectral-Gap). We compare the split procedures on

<table>
<thead>
<tr>
<th></th>
<th>Clean-Split</th>
<th>2-Median</th>
<th>Spectral-Gap</th>
<th>Spectral-Balanced</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>22</td>
<td>18</td>
<td>16</td>
<td>4</td>
</tr>
</tbody>
</table>

We compute an initial clustering by perturbing the ground-truth. In each iteration, we compute the set of all feasible splits and merges: a split of a cluster is feasible if it contains points from 2 or more ground-truth clusters, and a merge is feasible if at least an $\eta$-fraction of points in each cluster are from the same ground-truth cluster. Then, we choose one of the feasible edits uniformly at random, and ask the algorithm to compute the corresponding edit. We continue this process until we find the ground-truth clustering or we reach 20000 iterations. Our initial clusterings have over-clustering error of about 100, under-clustering error of about 100; and correlation-clustering error of about 5000.

We notice that for newsgroup documents it is difficult to compute average-linkage trees that are very consistent with the ground-truth. This observation was also made in other clustering studies that report that the hierarchical trees constructed from these data have low purity (Telgarsky & Dasgupta, 2012; Heller & Ghahramani, 2005). These observations suggest that these data are quite challenging for clustering algorithms. To test how well our algorithms can perform with better data, we prune the data sets by repeatedly finding the outlier in each target cluster and removing it, where the outlier is the point with minimum sum-similarity to the other points in the target cluster. For each data set, we perform experiments with the original (unpruned) data set, a pruned data set with 2 points removed per target cluster, and a pruned data set with 4 points removed per target cluster, which prunes 40 and 80 points, respectively (given that we have 20 target clusters).

5.2.2. Experiments in the $\eta$-merge model

We first experiment with local clustering algorithms in the $\eta$-restricted merge setting. Here we use the algorithm in Figure 1 to perform the splits, and the algorithm in Figure 2 to perform the merges. We show the results of running our algorithm on data set A in Figure 5. We find that for larger settings of $\eta$, the number of edit requests (necessary to find the target clustering) is very favorable and is consistent with our theoretical analysis. The results are better for pruned datasets, where we get very good performance regardless of the setting of $\eta$. The results for algorithms in Figure 1 and Figure 3 (for the correlation-clustering objec-

1http://people.csail.mit.edu/jrennie/20Newsgroups/
tive) are very favorable as well.

5.2.2. Experiments in the Unrestricted-Merge model

We also experiment with algorithms in the unrestricted merge model. Here we use the same algorithm to perform the splits, but use the algorithm in Figure 4 to perform the merges. We show the results on dataset A in Figure 5. We find that for larger settings of η our results are better than our theoretic analysis (we only show results for η ≥ 0.5), and performance improves further for pruned datasets. Our investigations show that for unpromised datasets and smaller settings of η, we are still able to quickly get close to the target clustering, but the algorithms are not able to converge to the target due to inconsistencies in the average-linkage tree. We can address some of these inconsistencies by constructing the tree in a more robust way, which indeed gives improved performance for unpromised data sets.

6. Discussion

In this work we motivated and studied new models and algorithms for interactive clustering. Several directions come out of this work. It would be interesting to relax the condition on η in the η-merge model, and the assumption about the request sequences in the unrestricted-merge model. It is important to study additional properties of an interactive clustering algorithm. In particular, it is often desirable that the algorithm never increase the error of the current clustering. Our algorithms in Figures 1, 3 and 4 have this property, but the algorithm in Figure 2 does not. It is also relevant to study more generalized notions of clustering error. In particular, we can define a small set of intuitive properties of a clustering error, and then prove that the algorithms in Figure 1 and Figure 2 are correct for any clustering error that satisfies these properties; a similar analysis is possible for the unrestricted-merge model as well.

![Figure 5. Results for data set A. The second chart corresponds to algorithms for correlation clustering error.](image)

![Figure 6. Results for initial clusterings with small error. Results presented for pruned data sets (4 points per cluster). The second chart corresponds to algorithms for correlation clustering error.](image)
References


Balcan, Maria-Florina and Blum, Avrim. Clustering with interactive feedback. In ALT, 2008.


