Clustering with Interactive Feedback

Maria-Florina Balcan and Avrim Blum

Carnegie Mellon University
Pittsburgh, PA 15213-3891
{ninamf,avrim}@cs.cmu.edu

Abstract. In this paper, we initiate a theoretical study of the problem of clustering data under interactive feedback. We introduce a query-based model in which users can provide feedback to a clustering algorithm in a natural way via split and merge requests. We then analyze the "clusterability" of different concept classes in this framework — the ability to cluster correctly with a bounded number of requests under only the assumption that each cluster can be described by a concept in the class — and provide efficient algorithms as well as information-theoretic upper and lower bounds.

1 Introduction

Clustering is often a highly under-specified problem: given a set of data items, there may be many different possible clusterings a user might be interested in. For instance, given a set of documents or news articles, should all those about sports go into a single cluster or should there be different clusters for football, baseball, hockey and so on? Should articles on salaries paid to sports figures be classified under sports or under business? Or perhaps, completely orthogonally, the user wants articles clustered by length or by writing style and not by topic. Most theoretical work "wishes away" this under-specification by making strong distributional assumptions, such as the data distribution being a mixture of Gaussians with each Gaussian as one of the clusters (e.g., [9, 3]). In this work, we instead embrace the idea that a given set of data might have multiple plausible clusterings, and consider the problem of *clustering under feedback*. That is, we imagine users are willing to help a clustering algorithm arrive at their own desired answer with a small amount of additional prodding, and ask what kinds of algorithms can take advantage of such feedback and under what conditions can they succeed. In this paper, we consider the problem of clustering under a quite natural type of feedback in the form of split and merge requests. Specifically, given a proposed clustering of the dataset, the user responds with either:

Split: The user identifies a cluster c in the algorithm's clustering that contains points from multiple target clusters (and therefore should be split), or

Merge: The user identifies two clusters c, c' in the algorithm's clustering that are both subsets of the same target cluster and therefore should be merged.

Or, if neither of these applies, the user responds that the algorithm's clustering is correct. Thus, this model is similar to the standard model of *learning with equivalence queries* [10], except that rather than return a misclassified data point, the user instead responds with the more vague request that, say, some cluster should be split (but without saying *how* that split should be done, or where in that cluster a mistake was made). We then show, perhaps surprisingly, that a number of interesting positive results can be had in this model under only the assumption that each cluster in the target is a member of some given concept class C, without any distributional assumptions on the data. In contrast, as mentioned above, most work on clustering has focused on *generative* models, such as mixtures of Gaussian or logconcave distributions, in which the underlying data distribution is effectively committing to only a single answer [1, 8, 7, 9, 14, 12].

Our model can be illustrated by the following simple example. Suppose our given dataset S consists of m points on the real line. We are told that each cluster is an interval, and let us say for simplicity of discussion we also are told there are only $k \leq 2$ clusters. In this case we could begin by proposing a single cluster with all the points and proposing it to the user. If we are incorrect, the user will respond with a split request, in which case we split the cluster into two intervals of $m/2$ points each and present the result to the user again. In general, if the user asks us to split a cluster, we partition it exactly in half (by cardinality), and if the user asks us to merge two clusters, we merge them. Since at most one of the algorithm's intervals can contain points from both of the user's intervals, and since each split request causes the number of points in the offending interval to drop by a factor of 2, the total number of split requests is at most $\lg m$. Therefore, the total number of merge requests is at most $\lg m$ as well, and so the overall number of requests at at most $2 \lg m$.

Note that clustering in this model with m requests is trivial for any concept class C: just begin with each point in its own cluster and merge as requested. So, our goal will be to develop algorithms whose query complexity is logarithmic in m and polynomial in parameters such as $\log |C|$ and (ideally) k. Note also that if we strengthened the model to allow the algorithm to specify *which* cluster the user should focus on, then we could simulate membership queries [2, 11];[1] indeed, one of the key difficulties in our model will be designing algorithms that can make progress no matter which clusters are asked to be split or merged.

The main results we show in this model are as follows (here, m is the total number of data points and k is the number of clusters in the target):

1. For the case of points on the line and the class of intervals, we give a simple algorithm that requires only $O(k \log m)$ requests to cluster correctly.

[1] Suppose inductively we have determined points $x_1, \ldots, x_{k'}$ that are all known to be in different clusters. Then, in this strengthened model, given a new point x we could query the sequence of clusters $\{x, x_1\}$, $\{x, x_2\}$.... The first one of these that does not produce a split request is the label of x (or if all produce a split request, we assign x to label $k' + 1$).

2. For the case of points in $\{0,1\}^n$ and the class of conjunctions (each cluster in the target is specified by a conjunction, and each data point satisfies exactly one of these conjunctions) we give an efficient algorithm with query complexity $O(n^k)$. Thus, this is polynomial when k is constant. For the case of disjunctions, we give an efficient algorithm with query complexity $O(n)$, independent of k. We do not know if there is an efficient algorithm for conjunctions with query complexity $\text{poly}(n,k)$.
3. For a general class C, (each cluster is a member of C, so there are at most $|C|^k$ possible clusterings overall), we give a generic but computationally inefficient algorithm with query complexity $O(k^3 \log |C|)$.
4. The generic algorithm mentioned above requires the algorithm, as it is learning, to produce clusterings in which there are many more than k clusters. If instead we restrict the algorithm to producing clusterings with only $\text{poly}(k, \log m, \log |C|)$ clusters (e.g., we allow the user a third option of refusing to split or merge and instead just saying "way too many clusters") then even for simple classes no algorithm can succeed.

1.1 Related work and motivation

There has, of course, been substantial theoretical work on clustering, e.g., [3, 1, 5, 8, 7, 9, 14, 12]. Much of this work assumes a generative model in which the distribution of data contains enough information at least in principle to reconstruct the single correct answer. Our model is motivated by recent work [4] that considers a relaxation of this setting in which the assumption is only that the target clustering satisfies certain natural relations with respect to the data. For instance, the target clustering might have the property that data points are closer to points in their own cluster than to points in any other cluster. This condition and others considered in [4] are not sufficient to uniquely identify the target clustering directly, but they *are* sufficient as shown in [4] to produce a *tree* (a hierarchical clustering) such that the desired clustering is some pruning of this tree. The idea is then that a user, given such a tree, could begin at the root and "click" on any node that is too broad to navigate down. This model is similar to clustering with split requests only, since the tree can be viewed as a pre-specification of what the algorithm would do on any given such request. Unfortunately, this approach is not sufficient to handle even the case of intervals described above, since for intervals we have multiple different possible clusterings even for $k = 2$, and none of these are refinements of the others (so no single such tree is possible). By considering an interactive model with both split and merge requests, we are able to avoid this difficulty and analyze a broader class of settings.

2 Notation and Definitions

We are given a set S of m points from some instance space X, and we assume that a user has in mind some partition of S into k disjoint clusters $c_1, \ldots, c_k$.

We call $\{c_1, \ldots, c_k\}$ the *target clustering*. Our goal will be to identify this target clustering from a limited number of split and merge requests (so, in learning-theoretic terms, we are considering a *transductive* problem). In particular, given a proposed clustering $\{h_1, h_2, \ldots, h_{k'}\}$ of S produced by the algorithm, if this clustering is not correct then the user will respond with either split(h_i) for some h_i that contains points from multiple target clusters, or merge(h_i, h_j) for two clusters h_i, h_j that are both subsets of the same target cluster. Note that if none of these conditions hold for any of h_i then the proposed clustering must be correct (to be clear, we say that $\{h_1, \ldots, h_k\}$ is correct if there is some permutation σ on $\{1, \ldots, k\}$ such that $h_i = c_{\sigma(i)}$ for all i). We think of the act of proposing a clustering as making a "query" (much like the notion of an equivalence query in learning, except the response is split or merge rather than a labeled example), and our goal is to identify the target from a small number of queries. As in the standard equivalence query model, we view the user as adversarial in the sense that we want to identify the target with a small number amount of feedback no matter how the user chooses which request to make if multiple split or merge requests apply.

If C is a concept class, we say that the clustering $\{c_1, \ldots, c_k\}$ is in class C if each $c_i \in C$. We say that an algorithm clusters class C with Q queries if it is able to identify the target with at most Q queries for any clustering in C. Our goal will be to do this for a number of queries that is polynomial in k, $\log |C|$, and $\log m$. As noted above, clustering with m queries is trivial: simply begin with m clusters each containing a single point in S and then merge as requested.

Note that because we assume each point $x \in S$ belongs to exactly one cluster c_i of the target, not every clustering in a given class C may be consistent with a given dataset S and vice-versa. E.g., if C is the class of disjunctions and S contains the point 10010, then the target could not, for instance, have $x_1 \vee x_2$ as one cluster and $x_4 \vee x_5$ as another. In fact, because of this issue, we will also consider what we call the *extended* model, where we allow the final cluster c_k to equal $S - (c_1 \cup \ldots \cup c_{k-1})$, even if this cannot be written as a concept in C. So, for $k = 2$, the set of possible clusterings of the given dataset S in the extended model is exactly $\{(c \cap S, \overline{c} \cap S) : c \in C\}$. All of our results can be made to hold for the extended model as well.

We have not yet specified whether or not algorithms are allowed to produce clusterings $\{h_1, \ldots, h_{k'}\}$ in which the clusters h_i overlap. We find that in some cases (e.g., Section 4) it is easier to design algorithms if we allow overlap, but we are also able to remove this at the expense of somewhat worse bounds. In all cases, however, we require the hypothesis clustering to cover all the points of S.

One final point: we have not yet placed any conditions on the number of clusters k' that the algorithm may use in its clusterings. Ideally, an algorithm should use $k' = O(k)$ or perhaps $k' = poly(k, \log m)$ clusters, since this quantity in some sense determines the "cognitive load" on the user. In fact, for most of our algorithms this is indeed the case. However, our generic algorithm (Section 5) may need substantially more clusters, and in Section 6 we show that in fact this is necessary in our framework to achieve a good bound in general.

3 Intervals

To illustrate the general setting, in this section we present a simple algorithm for the class C of intervals on the line, that requires at most $k \log m$ requests to cluster correctly. Specifically, the algorithm is as follows.

Algorithm Cluster-Intervals:

- Begin with a single cluster containing all m points of S.
- On a split request to a cluster c, partition c into two clusters of equal cardinality.
- On a merge request, merge the two clusters indicated.

Theorem 1. *Algorithm* **Cluster-Intervals** *requires only $O(k \log m)$ requests to cluster the class of intervals on the line.*

Proof: Since the target consists of k intervals, we can identify $k-1$ decision boundaries $a_1, \ldots, a_{k-1}$, where a_i is an arbitrary position between the largest point in S in target interval i and the smallest point in S in target interval $i+1$. If a_i lies inside some cluster c of the algorithm's clustering, define size(a_i) to be the number of points of S in c; else define size$(a_i) = 0$. (For concreteness, if a hypothesis cluster c contains points $p_1 < p_2 < \ldots < p_t \in S$, we define $c = [p_1, p_t]$.) So, initially, we have size$(a_i) = m$ for all i.

Now, a split request can only be made to a cluster c that contains at least one of the a_i (otherwise c would contain points from only one target interval). Since the result of a split is to replace c with two clusters of half as many points, this means that each split request reduces size(a_i) by a factor of 2 for at least one of the decision boundaries a_i. Furthermore, a merge request can only be made on two clusters c, c' such that $c \cup c'$ does not contain any of the a_i. Therefore, the total number of split requests can be at most $k \log m$ total, which implies a total of at most $k \log m$ merge requests as well. □

Note that for intervals, the extended model with k clusters (where the last cluster can be a "default bucket") can be expressed using the standard model with $2k-1$ clusters, so Theorem 1 applies to the extended model as well.

4 Conjunctions and Disjunctions

We now present an algorithm for clustering the class C of disjunctions over $\{0,1\}^n$. That is, each target cluster can be described by some disjunction of literals, and each point $x \in S$ satisfies exactly one of the target disjunctions. We show we can do this making only $O(n)$ queries if we allow the clusters in the hypothesis clusterings to overlap. We then show how this algorithm can be adapted to remove this overlap (so that each proposed clustering is indeed a partition of S), but at the expense of now making $O(n^2)$ queries. We finally show how these can be used to yield algorithms for the class C of conjunctions that are polynomial for constant k. Specifically, these algorithms make $O(n^{k-1})$ queries

and $O(n^{2(k-1)})$ queries respectively depending on whether or not disjointness is required.

We begin with a simple $O(n)$-query algorithm for disjunctions, if we allow hypothesis clusters to overlap. Without loss of generality we may assume the target disjunctions are monotone (by the standard trick of introducing variables $y_i = 1 - x_i$ for every variable x_i).

Algorithm Cluster-Disjunction:

- Begin with one cluster for each variable x_i, containing all points $x \in S$ such that $x_i = 1$. (Note that these may overlap).
- On a split request to a cluster x_i, simply delete that cluster.
- On a merge request, merge the two clusters indicated.

Theorem 2. *Algorithm* **Cluster-Disjunction** *requires at most $n - k$ requests to cluster the class of disjunctions over $\{0,1\}^n$.*

Proof: First, notice that whenever two clusters are merged, they can never be split, since by definition, the result of a merge operation is pure (contains points from only one target cluster). Therefore, all split requests are made to clusters corresponding to single variables. Second, note that if x_i is a relevant variable (belongs to one of the target disjunctions), then its associated cluster is pure and so will not be split. Since each point $x \in S$ must satisfy one of the target disjunctions, this means we maintain the (crucial) invariant that our clustering is *legal*: every point $x \in S$ belongs to at least one cluster in the hypothesis. Thus, we ensure that if our current hypothesis is incorrect, there must be a split or merge request the user can make. Since each request results in reducing the number of clusters by 1, this means that the total number of requests is at most $n - k$. □

Notice that the above algorithm produces hypothesis clusterings in which the clusters are non-disjoint (because a given point $x \in S$ may have several variables set to 1). If we want the hypothesis clusters to be disjoint, a natural approach to doing so is to just arbitrarily remove each example x from all but one of the clusters i such that $x_i = 1$. However, this may cause the result of a split request to no longer be a legal clustering since some points x may no longer belong to any clusters, and arbitrarily re-assigning them may break the invariant that the clusters resulting from merge requests can never be split later. We can fix this problem, but at a loss of using $O(n^2)$ requests, as follows.

Algorithm Disjoint-Disjunction:

- Begin with one cluster for each variable x_i, containing all points $x \in S$ such that $x_i = 1$ *and* $x_j = 0$ for all $j < i$.
- On a split request to a cluster x_i, delete that variable from every point in S and restart the entire algorithm from the beginning (with $n \leftarrow n - 1$).
- On a merge request, merge the two clusters indicated.

Theorem 3. *Algorithm* **Disjoint-Disjunction** *maintains a disjoint clustering and requires at most $O(n^2)$ requests to cluster the class of disjunctions over $\{0,1\}^n$.*

Proof: The initialization step of the algorithm insures that clusters are disjoint and furthermore include all points $x \in S$. As before, since the result of a merge operation must contain points in only one target cluster, any split request must be to a cluster corresponding to a single variable x_i. By assumption that the target clusters are disjunctions, any hypothesis clusters corresponding to *relevant* variables are pure, and therefore any split request must be to an irrelevant variable. So, deleting such variables maintains the invariant that each target cluster can be expressed as a disjunction. Since each split request reduces the total number of variables by 1, and there can be at most $n - k$ merge requests in a row, the total number of requests is $O(n^2)$. □

We now show how the above algorithms can be used to cluster the class of conjunctions, making $O(n^{k-1})$ queries and $O(n^{2(k-1)})$ queries respectively depending on whether or not disjointness is required. In particular, let $c_1, c_2, \ldots, c_k$ denote the target clusters and assume without loss of generality that each conjunction is monotone. Because each point $x \in S$ lies in exactly one target cluster, this means we can equivalently write cluster c_i as $\overline{c}_1 \wedge \ldots \wedge \overline{c}_{i-1} \wedge \overline{c}_{i+1} \wedge \ldots \wedge \overline{c}_k$. Since each c_j is a conjunction, this means we can write c_i as a $(k-1)$-DNF, or equivalently as a disjunction over a space of n^{k-1} variables $y_{i_1 \cdots i_{k-1}} = \overline{x}_{i_1} \overline{x}_{i_2} \cdots \overline{x}_{i_{k-1}}$. Thus, we can cluster by expanding to this space of n^{k-1} variables and running the disjunction algorithms given above. The bounds then follow immediately.

4.1 Conjunctions and Disjunctions in the Extended Model

In the extended model (where we allow c_k to be a default cluster not necessarily in C), Algorithm **Cluster-Disjunction** "almost" works. The only problem is that deleting a cluster x_i may cause some points to become completely uncovered, because points in c_k do not have any relevant variables set to 1. However, since all points with no relevant variables must be in the same cluster c_k, we can fix this problem with a small modification to the algorithm. Specifically, we just create an extra "default bucket" containing all the points not covered by any of the other hypothesis clusters. This bucket will never be split (since all such points must be in c_k) so the analysis proceeds exactly as before. The same modification and analysis applies to Algorithm **Disjoint-Disjunction**: in particular, the default bucket will just contain all points that have no variables set to 1. Thus we have the following theorem.

Theorem 4. *The above modification to Algorithm* **Cluster-Disjunction** *requires at most $n - k + 1$ requests to cluster the class of disjunctions over $\{0,1\}^n$ in the extended model. The modification to Algorithm* **Disjoint-Disjunction** *maintains disjoint clusters and requires at most $O(n^2)$ requests to cluster disjunctions in the extended model.*

For conjunctions, the difficulty with the reduction given in the non-extended model is that the expression $\overline{c}_1 \wedge \ldots \wedge \overline{c}_{i-1} \wedge \overline{c}_{i+1} \wedge \ldots \wedge \overline{c}_k$ for cluster c_i is no longer a $(k-1)$-DNF, except for the case $i = k$. However, we can deal with this problem with the following procedure.

Algorithm Extended-Cluster-Conjunctions:

- Begin with one cluster for each term $y_{i_1 \cdots i_{k-1}} = \overline{x}_{i_1} \overline{x}_{i_2} \cdots \overline{x}_{i_{k-1}}$, containing all points in S satisfying this term. Instantiate a default bucket with all points not covered by any other cluster.
- On a split request to one of the y clusters, or a merge request to a y cluster and a cluster in the default bucket, delete the y cluster. If any points become uncovered, insert them into the default bucket and instantiate (or restart from scratch) a $(k-1)$-cluster conjunction algorithm for the non-extended model on the points in that bucket.
- On a merge request to two y clusters, merge the two clusters indicated.
- On a split or merge request to two clusters within the default bucket, send them to the conjunction-learning algorithm being run on the datapoints in that bucket.

Theorem 5. *Algorithm* **Extended-Cluster-Conjunctions** *requires at most* $O(n^{2k-3})$ *requests to cluster the class of conjunctions over* $\{0,1\}^n$ *in the extended model.*

Proof: Because cluster c_k can be written as a disjunction over the y variables, the relevant y variables for c_k will never receive split requests or be asked to be merged with clusters within the default bucket. Therefore, the above algorithm maintains the invariant that only points within $c_1 \cup \cdots \cup c_{k-1}$ are ever placed into the default bucket. This implies that at most $O(n^{k-2})$ requests will be made to clusters inside that bucket between any two consecutive restarts to the $(k-1)$-cluster conjunction-learning algorithm in that bucket. There can be at most n^{k-1} restarts, so overall the number of requests will be at most $O(n^{2k-3})$. □

One can also consider a disjoint-cluster version of Algorithm **Extended-Cluster-Conjunctions**, by deleting y variables and restarting on split requests as in Algorithm **Disjoint-Disjunction**, and by using a disjoint clustering algorithm for the default bucket. In this case, we have at most $O(n^{2(k-2)})$ requests to the default bucket between restarts, and at most n^{k-1} restarts, resulting in at most $O(n^{3k-5})$ requests total.

5 A General Upper Bound

We now describe a generic but computationally inefficient algorithm that will cluster any concept class C using $O(k^3 \log |C|)$ queries. The high-level idea is that ideally we would like to use some form of halving algorithm, except that because feedback is in the form of split and merge requests rather than labeled examples, it is not so clear how to do this directly. What we will do instead is carefully construct a clustering such that any split or merge request removes at least a $1/k^2$ fraction of the version space, leading to the bound given above. We point out that the clustering constructed in each step may have many more than k clusters in it. However, this is unavoidable: as we show in Section 6, it is not possible to achieve a query complexity polynomial in k, $\log |C|$ and $\log m$ if we require the algorithm to use only $\text{poly}(k, \log |C|, \log m)$ clusters.

The generic algorithm is as follows. We begin with the simple "wrapper" and then present the main "engine" of the algorithm.

Algorithm Generic-Clustering:

1. Let C^{VS} denote the current version space: the set of cluster*ings* consistent with the results of all queries so far. So, initially we have $|C^{VS}| \leq |C|^{k-1}$.
2. Run Algorithm **Generate-Interesting-Clustering**(C^{VS}, S) described below to produce a clustering that is guaranteed to have the property that any split or merge request removes a substantial fraction of the version space, and propose it to the user.
3. When a split or merge request is received, remove from C^{VS} all clusterings inconsistent with the request and go to (2).

We now give the main "engine" of the algorithm. Here we say that a cluster b is *consistent* with a clustering $\{c_1, \ldots, c_k\}$ if $b \subseteq c_i$ for some i.

Algorithm Generate-Interesting-Clustering(C^{VS}, S)**:**

1. Initialize k buckets $B_1, \ldots, B_k$ (initially all empty) and let $\alpha = 1/k^2$. We will maintain the invariant that each B_i is consistent with at least a $1 - \alpha$ fraction of the clusterings in C^{VS}.
   For each point $x \in S$ (in an arbitrary order) do:
   (a) Insert x into the bucket B_i of least index such that $B_i \cup \{x\}$ is consistent with at least an α fraction of the clusterings in C^{VS}. If no such B_i exists, then halt with failure.
   (b) If B_i is now consistent with less than a $1-\alpha$ fraction of clusterings in C^{VS}, then output B_i as one of the hypothesis clusters, and replace it with a new empty bucket, putting the new bucket at the end of the list. That is, let $B_i \leftarrow B_{i+1}, B_{i+1} \leftarrow B_{i+2}, \ldots, B_{k-1} \leftarrow B_k$ and call the new empty bucket B_k.
2. If we reach this point (no more points $x \in S$ and all buckets B_i are consistent with at least a $1 - \alpha$ fraction of C^{VS}), output the clusters $B_1, B_2, \ldots, B_k$, ignoring empty buckets.

Theorem 6. *Algorithm* **Generic-Clustering** *succeeds in clustering any class C using at most $O(k^3 \log |C|)$ requests, and furthermore this holds in the extended model as well.*

Proof: We show that Algorithm **Generate-Interesting-Clustering** outputs a clustering such that any split or merge request is guaranteed to remove at least a $1/k^2$ fraction of clusterings from the version space C^{VS}. This will immediately imply that the total number of queries of Algorithm **Generic-Clustering** is at most $O(k^2 \log |C^{VS}|) = O(k^3 \log |C|)$ (in both standard and extended models) as desired.

First, we argue that the algorithm will never halt with failure in Step 1(a). By construction, we maintain the invariant that each bucket B_i is consistent with at least a $1 - \alpha$ fraction of clusterings in C^{VS}, since otherwise we would

have already outputted it in Step 1(b) (and by definition, an empty bucket is consistent with the entire C^{VS}). Therefore, at least a $1 - k\alpha$ fraction of C^{VS} is consistent with all B_i simultaneously. In addition, for each pair $i < j$ for which B_j is non-empty, at most an α fraction of C^{VS} can be consistent with $B_i \cup B_j$ (because each point $x \in B_j$ has the property that at most an α fraction of C^{VS} is consistent with $B_i \cup \{x\}$, else it would have been inserted into B_i instead). Moreover, we only care about the case that no buckets are empty since otherwise we could always put x into the first empty bucket. Therefore at least a $1 - k\alpha - \binom{k}{2}\alpha$ fraction of C^{VS} is consistent with each B_i *and* has all k buckets B_i as subsets of *distinct* clusters. By definition of α, this is at least a $k\alpha$ fraction of C^{VS}. Now, each of these clusterings has x in one of its k clusters, and that cluster is associated with a single bucket B_i. So, there must exist an index i such that at least a $k\alpha/k = \alpha$ fraction of C^{VS} has x in a cluster consistent with B_i, and therefore Step 1(a) succeeds.

We now argue that the clustering produced has the desired property that any split or merge request removes at least an α fraction of the version space. First, if the algorithm outputs a cluster in Step 1(b), then the fraction of clusterings in C^{VS} consistent with the cluster B_i produced is in the range $[\alpha, 1 - \alpha]$, and therefore any split or merge request involving it removes at least an α fraction of the version space. So, all clusters produced in Step 1(b) have the desired property. Next, we need to consider the clusters output in Step 2. For those, the situation is even better: any split or merge request involving only these clusters removes at least a $1 - \alpha$ fraction of the version space. In particular, for split requests this is because each B_i is consistent with at least a $1 - \alpha$ fraction of C^{VS}, and for merge requests this is because at most an α fraction of C^{VS} is consistent with $B_i \cup \{x\}$ for all $x \in B_j, j > i$. Thus, overall the clustering produced has the property that any request removes at least an $\alpha = 1/k^2$ fraction of the version space as desired. □

6 Lower Bounds

We now show that if we restrict the algorithm to only producing clusterings with $\text{poly}(k, \log m)$ clusters, then there exist classes for which no algorithm can succeed. Thus, the use of what might potentially be a large number of small clusters in the generic algorithm of Section 5 is in fact necessary.

We begin first with a much easier statement to prove.

Theorem 7. *There exist classes C of size m such that, even for $k = 2$, any algorithm that is restricted to producing k-clusterings will require $\Omega(m)$ queries.*

Proof: Let S consist of m points on the circle, and let C be the class of intervals. Assume the target clustering is a random partition of S into two contiguous intervals of $m/2$ points. If the algorithm proposes a 2-clustering in which the two clusters have different sizes, then the larger cluster must contain points from both target clusters. So the user can issue a split request on that cluster providing the algorithm with no information (the algorithm could simulate the

user itself). Alternatively, if the algorithm proposes two equal-size clusters, the user can issue a split request on either cluster unless the algorithm is exactly correct. Since the target was chosen to be a random partition, this implies any algorithm must make $\Omega(m)$ queries in expectation. □

We now present our main result of this section.

Theorem 8. *There exist classes C of size $O(m)$ such that, even for $k = 2$, no algorithm that is restricted to producing clusterings with only poly$(k, \log m)$ clusters can have even a $1/\text{poly}(k, \log m)$ chance of success after poly$(k, \log m)$ queries.*

Proof: Consider the set $S = \{0,1\}^n$, so $m = 2^n$, and let C be the class of parity functions and their negations. (So, for some parity function c, one cluster will consist of all points $x \in \{0,1\}^n$ such that $c(x) = 1$ and the other will consist of all points $x \in \{0,1\}^n$ such that $c(x) = -1$.) We claim that for any query made by the algorithm, if the target is determined by a random parity c, the user will be able to issue a split request on the largest cluster with probability exponentially close to 1. Thus, the algorithm receives no information (since it could simulate such a user itself) and therefore exponentially many queries will be needed to cluster correctly (exponential in k and $\log m$).

Specifically, suppose the largest cluster c' in the algorithm's proposed clustering has size αm. Define the boolean function $h(x) = 1$ if $x \in c'$ and $h(x) = -1$ if $x \notin c'$. The user will be able to issue a split request on c' unless $c' \subseteq c$ or $c' \subseteq \neg c$; in the former case we have $\Pr[h(x) = c(x)] = 1/2 + \alpha$ and in the latter case we have $\Pr[h(x) = \neg c(x)] = 1/2 + \alpha$. Either of these cases imply that the magnitude of correlation between h and c satisfies:

$$\begin{aligned} |\mathbf{E}_x[h(x)c(x)]| &= |\Pr[h(x) = c(x)] - \Pr[h(x) \neq c(x)]| \\ &= |1/2 + \alpha - (1/2 - \alpha)| = 2\alpha, \end{aligned}$$

or in Fourier notation, we have $|\langle h, c\rangle| = 2\alpha$. However, by Parseval's identity [13], we know that h can have correlation of magnitude 2α with at most $1/(2\alpha)^2$ different parity functions. Thus, if c was chosen at random from among all 2^n parity functions, the probability that the algorithm's largest cluster c' truly is a subset of one of the target clusters is at most $1/(4\alpha^2 2^n)$. Since we assumed the algorithm produced clusterings with only $\text{poly}(k, \log m)$ clusters, it must be the case that $\alpha \geq 1/\text{poly}(k, \log m)$, and so the probability the user is not able to issue a split request on the largest cluster is exponentially small in k and $\log m$, as desired. □

7 Relation to Equivalence Query Model

In the standard model of learning with equivalence queries, any class C can be learned using at most $\log |C|$ queries via the halving algorithm. However, some classes can be learned with many fewer queries, such as the class of concepts

having at most one positive example which requires only one query to learn. In contrast, we show here that in our framework, for *any* class C, for the case $k = 2$ there is a lower bound of $\Omega(\log |C^{VS}|)$ on the number of queries needed to cluster, where C^{VS} is the initial version space. Thus, for the extended model, this gives a lower bound of $\Omega(\log |C|)$ on the number of queries needed to cluster.

Theorem 9. *For any class C, for $k = 2$, any clustering algorithm must make $\Omega(\log |C^{VS}|)$ queries in the worst case, where C^{VS} is the initial version space.*

Proof: Let $\{h_1, h_2, \ldots, h_t\}$ be a clustering produced by the clustering algorithm. We show there must exist a split or merge request that removes at most a 5/6 fraction of the version space.

First, suppose the algorithm's clustering has only two clusters ($t = 2$). In that case, every clustering in C^{VS} except for (a) the trivial clustering that puts all points into a single cluster or (b) a clustering identical to $\{h_1, h_2\}$ must split either h_1 or h_2 or both. Without loss of generality, say that a majority of those $|C^{VS}| - 2$ clusterings split h_1. Thus, a split request on h_1 must be consistent with at least $(|C^{VS}| - 2)/2$ clusterings in the version space.

Now, for the case $t > 2$, consider the first three clusters h_1, h_2, h_3 in the algorithm's clustering. Since all clusterings in C^{VS} have only two clusters, each must either split one of the h_i or else have two of the h_i inside the same cluster. Thus, for each clustering in C^{VS}, at least one of the 6 possible split or merge requests on $\{h_1, h_2, h_3\}$ must apply. Therefore there must exist some request that is consistent with at least a 1/6 fraction of C^{VS} as desired. □

8 Conclusions

In this paper we have analyzed the problem of determining the correct clustering of data from a bounded number of split and merge requests. We have provided efficient algorithms for several natural classes including disjunctions, conjunctions (for bounded k), and intervals, as well as a generic $O(k^3 \log |C|)$ upper bound for clustering any given class C. We also provide lower bounds for algorithms that use a bounded number of clusters and a separation result with respect to the standard model of learning with equivalence queries.

This model brings up several interesting open questions. First, can one improve the generic upper bound from $O(k^3 \log |C|)$ to $O(k \log |C|)$, i.e., gain a constant amount of information per query. Second, can one devise an efficient algorithm for conjunctions whose query complexity is polynomial in both k and n. Our generic algorithm implies this is possible information-theoretically but we do not know any efficient procedure. Finally, a natural domain for clustering with split and merge requests is image segmentation. From this perspective, it would be interesting to generalize the 1-dimensional results to 2 dimensions, ideally to the case where each cluster is a region defined by a limited number s of axis-parallel line segments as in the results on learning discretized geometric concepts using equivalence queries of Bshouty et al. [6].

More broadly, it would be interesting to further explore clustering with other natural forms of interactive feedback.

Acknowledgments: This work was supported in part by the National Science Foundation under grant CCF-0514922, by an IBM Graduate Fellowship, and by a Google Research Grant.

References

1. D. Achlioptas and F. McSherry. On spectral learning of mixtures of distributions. In *Proceedings of the 18th Annual Conference on Learning Theory*, 2005.
2. D. Angluin. Queries and concept learning. *Machine Learning*, 2:319–342, 1998.
3. S. Arora and R. Kannan. Learning mixtures of arbitrary gaussians. In *Proceedings of the 33rd ACM Symposium on Theory of Computing*, 2001.
4. M.-F. Balcan, A. Blum, and S. Vempala. A discriminative framework for clustering via similarity functions. In *Proceedings of the 40th ACM Symposium on Theory of Computing*, 2008.
5. S. C. Brubaker and S. Vempala. Isotropic PCA and affine-invariant clustering. In *Proceedings of the 49th ACM Symposium on Foundations of Computer Science*, 2008.
6. N. H. Bshouty, P. W. Goldberg, S. A. Goldman, and H. D. Mathias. Exact learning of discretized geometric concepts. *SIAM J. Computing*, 28(2):674–699, 1998.
7. A. Dasgupta, J. Hopcroft, J. Kleinberg, and M. Sandler. On learning mixtures of heavy-tailed distributions. In *46th IEEE Symposium on Foundations of Computer Science*, 2005.
8. A. Dasgupta, J. E. Hopcroft, R. Kannan, and P. P. Mitra. Spectral clustering by recursive partitioning. In *Proceedings of the 14th European Symposium on Algorithms*, pages 256–267, 2006.
9. S. Dasgupta. Learning mixtures of gaussians. In *Proceedings of the 40th Annual Symposium on Foundations of Computer Science*, 1999.
10. L. Hellerstein, K. Pillaipakkamnatt, V. V. Raghavan, and D. Wilkins:. How many queries are needed to learn? In *Proceedings of the 27th ACM Symposium on Theory of Computing*, 1995.
11. J. Jackson. An efficient membership-query algorithm for learning dnf with respect to the uniform distribution. *Journal of Computer and System Sciences*, 57(3):414–440, 1995.
12. R. Kannan, H. Salmasian, and S. Vempala. The spectral method for general mixture models. In *Proceedings of the 18th Annual Conference on Learning Theory*, 2005.
13. Y. Mansour. Learning boolean functions via the fourier transform. *Theoretical Advances in Neural Computation and Learning*, pages 391–424, 1994.
14. S. Vempala and G. Wang. A spectral algorithm for learning mixture models. *Journal of Computer and System Sciences*, 68(2):841–860, 2004.