Contents

1 Introduction to Interactive Split-Merge Clustering

In their original work, Balcan & Blum (2008) introduced the theoretical study of the interactive clustering problem via split-merge feedback. The central idea is as follows: We can denote a "clustering" as a set of hypotheses $\{c_1, \cdots, c_k\}$ from hypothesis class $\mathcal{C}$ such that when we apply the set of maps to some dataset $X = \{x_i\}_{i=1}^m$, the hypotheses partition X into disjoint sets. The goal is to identify a set of hypotheses such that we can achieve this goal by making a minimal number of split-merge queries to a user. In particular, one thing which differentiates this setting from other clustering settings is that we are looking for a kind of worst-case guarantee: We wish to make no assumptions about the frequencies of the various types of queries we will receive. This approach is distinct from Bayesian setting where typically strong (and Gaussian) assumptions are made on the data, as is typical in Gaussian mixture modeling. We instead specify two kinds of feedback which the algorithm receives after outputting a candidate clustering (a **query**). A **split** feedback specifies a hypothesis cluster in the algorithm's outputted clustering which needs to be split into one or more clusters, but does not specify how to split the offending cluster. A **merge** feedback specifies two hypothesis clusters in the clustering which should be merged together into a single cluster. Note that by definition, merging is a pure operation in the sense that the two hypotheses selected to merge must be in the same target cluster. Typically, algorithms in this setting involve

specifying a way to initialize clusters and how to react when receiving a split request between two hypotheses.

Since it is trivial to cluster in this model using only m queries (start with each point in its own cluster and receive merge requests), we are only interested in algorithms which depend sublinearly on m (hopefully, logarithmically). In general, we hope to find algorithms which cluster with a number of queries of order $\mathcal{O}(\text{poly}(k, \log|\mathcal{C}|, \log m))$. Using this model, it is possible to design clustering algorithms for specific hypothesis classes (e.g., intervals, disjunctions) which take advantage of the specific structure of the hypothesis class to get good algorithms.

1.1 Two Generic Inefficient Interactive Clustering Algorithms

However, we would like to design more general algorithms for interactive clustering oblivious to the specific concept class we are dealing with. One approach is to work explicitly over the version space of clusterings, and with each feedback response to a query, reduce the size of the version space by some fraction. In particular, this approach is intimately related to the halving algorithm and is common in interactive learning theory (for instance, the splitting index approach by Dasgupta (2005) is similar in general spirit). In particular, if we can guarantee that we can reduce the version space by a fixed fraction each iteration, *no matter what the result of the query was*, we will be able to obtain a query-efficient algorithm, providing an upper bound to the query complexity of interactive clustering for any hypothesis class for clusters. The first algorithm following this approach was given in Balcan & Blum (2008). Before giving the algorithm, we define an important notion:

Definition 1.1. α-consistent.
A set S of points is α-consistent for some $\alpha \in (0,1)$ with respect to a concept class $\mathcal{C}$ and a dataset of points X if for an α-fraction of all clusterings of concepts $(c_1, \cdots, c_k) \in \mathcal{C}^{\text{VS}}$ in the version space, it is true that $S \subseteq c_i(X)$ for some $i \in [k]$.

This notion is critical to defining to ensuring we make progress as use feedback to prune the version space. Algorithm 2 implements this strategy and yields a query complexity $\mathcal{O}(k^3 \log|\mathcal{C}|)$, where k is the number of clusters. We can see this fact since to reduce to the case where $|V| = 1$, we need T iterations of the while loop where we are only guaranteed to remove a $\frac{1}{k^2}$ fraction of the version space each round. Solving $\left(1 - \frac{1}{k^2}\right)^T |\mathcal{C}^{VS}| = 1$ yields $T = \frac{k \log|\mathcal{C}|}{\log \frac{k^2}{k^2-1}}$, noting that $|\mathcal{C}^{VS}| = |\mathcal{C}|^{k-1}$. Seeing that $\log^{-1} \frac{k^2}{k^2-1} = \mathcal{O}(k^2)$, we see the query complexity is $\mathcal{O}(k^3 \log|\mathcal{C}|)$.

However, we can improve this bound to $\mathcal{O}(k \log|\mathcal{C}|)$ using a modified, simpler algorithm due to Awasthi & Zadeh (2010), given in Algorithm 3. To succinctly describe the algorithm, we need the following definition:

Definition 1.2. Consistent Cluster Set.
For some set of points $s \subset S$, the **consistent cluster set**

$$\text{CCS}(s) := \left\{\{c_1, \cdots, c_k\} \in \mathcal{C}^{VS} | \{c_1, \cdots, c_k\} \text{ is consistent with } s\right\}$$

In the case of Algorithm 3, we are guaranteed to halve the version space each time no matter what the feedback is. Thus, the query complexity is $\mathcal{O}(\log|\mathcal{C}^{VS}|) = \mathcal{O}(k \log|\mathcal{C}|)$.

1.2 Inefficient Aspects of the Generic Algorithms

Despite the good query complexity given by the generic algorithm, there are many issues which prevent it from being used in practice. They are

(a) Requiring an instantiation of the version space.

(b) Requiring a potentially large number of clusters per clustering.

It turns out that the second restriction is inescapable in the general case for merge-split queries: There exist hypothesis classes such that algorithms which produce a polynomial number of clusters are insufficient to succeed with only polynomially many queries. In particular, the hypothesis class of parity functions over the Boolean hypercube is a failure case, due to an uncertainty principle argument derived from Boolean Fourier analysis.

Thus, for now, we restrict ourselves to improving the first issue in the merge-split setting. The version space is typically exponentially large, and thus infeasible to represent in full memory. We want to be able to execute one of the above algorithms (Algorithm 2 or 3) without having to represent the version space in full. This necessity results in two requirements if we use Algorithm 2:

(a) We must be able to check the α-consistency of a set with respect to a version space V efficiently without actually maintaining a full representation of V.

(b) We must be able to update our representation of V to V' so that in the future, the α-consistency check is updated to be with respect to V', again without having to actually iterate over the full version space.

Similarly, if we choose to go with Algorithm 3, we have two slightly different requirements:

(a) We must be able to optimize over sets of points s in the domain constrained by the size of the V-consistent cluster set with respect to s. In particular, we must be able to find the largest set of points s which obeys this constraint. This requirement again amounts to being able to check consistency with respect to the version space efficiently.

(b) Again, we must be able to update our representation of V to V' so that consistency checks are also updated.

We can choose to make either algorithm efficient: The algorithm which will overall be more efficient depends on how much harder optimization over consistency constraints is compared to just checking consistency constraints. As *constrained* optimization seems difficult, we will focus on making Algorithm 2 efficient.

2 Efficient Interactive Clustering

In this section, we efficiently implement a relatively general purpose interactive concept-specification algorithm, subject to some constraints on the hypothesis classes we consider.

2.1 The Algorithm

We will provide an alternative implementation of Balcan & Blum (2008)'s algorithm which is somewhat practical. The main insight is to use a sampling oracle which can be used to estimate α-consistency. See Algorithm 1. Correctness follows from the proof of Balcan & Blum (2008), as does query complexity $\mathcal{O}(k^3 d \log m)$. We choose $\epsilon = \frac{1}{k^3}, \delta = \frac{0.0001}{k \cdot m \cdot k^3 d \log m} = \frac{0.0001}{k^4 dm \log m}$. We require $\epsilon = o(\frac{1}{k^2})$ so that with high probability we get α-consistency with $\alpha \geq \frac{1}{k^2}$ as in Balcan & Blum (2008). We also require the chosen δ, which by a union bound ensures probability of success 0.9999 over the whole algorithm (the denominator is the number of times we must run the sampler with parameters δ, ϵ). We can always run the algorithm repeatedly to boost the probability of success arbitrarily high.

The computational complexity of the algorithm is $\mathcal{O}\left(k^3 d \log(m) \cdot \left(m \cdot k \cdot \frac{T_{\text{sample}}}{\epsilon^2} \log\left(\frac{1}{\delta}\right) + k \cdot T_{\text{update}}\right)\right)$, where T_{sample} is the time it takes to sample and T_{update} is the time it takes to update the cluster-sampler.

2.2 The Sampling Oracle

First, we want to determine whether or not a set S is α-consistent, or at least to determine a lower bound on the true α bounded away from 0. We then need an approach to retrieve such α-consistent S from the dataset of points P in an efficient manner. Finally, we need to be able to update the state of the version space, upon which the property of α-consistency depends, after receiving feedback.

2.2.1 Reduction to Sampling Clusters

In order to check α-consistency over the version space of clusterings, it suffices to sample from a distribution over $\mathcal{C}$, the concept class of clusters.

Lemma 2.1. *α-consistency over clusters $\implies$ at least α-consistency over k-clusterings.*

Proof. The set of clusterings which would not be consistent must consist of clusterings containing clusters drawn from exclusively the $1 - \alpha$ fraction of non-consistent clusters; this implies β-consistency over clusterings where $\beta = 1 - (1 - \alpha)^k \geq \alpha$, where k is the number of clusters in a clustering.

If we want to ensure that we have $1/k^2$-consistency over k-clusterings, then we need to solve $\beta = 1/k^2$:

$$1 - (1 - \alpha)^k = \frac{1}{k^2}$$

$$\alpha = 1 - \left(1 - \frac{1}{k^2}\right)^{1/k}$$

□

Algorithm 1 Efficient Generic Clustering

procedure EFFICIENTCLUSTER(X, `cluster-sampler`, ϵ, δ) ▷ X := input dataset of m points, `cluster-sampler` := sampling oracle for $\mathcal{C}$, ϵ := error tolerance, δ := probability of failure
  Initialize feedback set $F := \{\text{start}\}$.
  while $|F| > 0$ **do**
    Initialize buckets $B_1, \cdots, B_k := \{\}$.
    Initialize output cluster list $L = []$.
    for each point $x \in X$ in arbitrary order **do**
      for i in $[1, \cdots, k]$ **do**
        Sample $\frac{1}{\epsilon^2}\log(\frac{1}{\delta})$ clusters from `cluster-sampler`
        if $B_i \cup \{x\}$ consistent with at least $\frac{1}{k^2} + \epsilon$ of sampled clusters **then** $B_i := B_i \cup \{x\}$
          if B_i is consistent with at most $\left(1 - \frac{1}{k^2} - \epsilon\right)$ clusters **then**
            Append B_i to L.
            Delete B_i from the list of buckets.
            Append $\{\}$ to the end of the list of buckets.
          end if
          break
        end if
      end for
    end for
    Receive feedback set $F := F_u(L)$ from user on cluster list L.
    for feedback $f \in F$ **do**
      if $f = \text{merge}(c_i, c_j)$ **then**
        Let $R :=$ set of clusters inconsistent with $c_i \cup c_j$.
      else if $f = \text{split}(c_i)$ **then**
        Let $R :=$ set of clusters consistent with c_i.
      end if
      Update `cluster-sampler` so clusters in R are sampled with probability 0.
    end for
  end while
  return L as the clustering.
end procedure

2.2.2 α-Consistency Algorithm and its Sample Complexity

Now, given a distribution over concept clusters conditioned on the merge-split query results we have seen thus far, as well as the data, we can check whether a set of points (a candidate cluster c) is α-consistent with respect to the version space by iterating the following procedure $\mathcal{O}(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ times:

(a) Initialize a count variable to 0.

(b) Sample a cluster c uniformly from the current distribution over clusters.

(c) Check if the cluster is consistent with c. If it is, increment the count variable.

After all iterations, decide if c is α-consistent by checking if the count variable divided by the number of iterations is larger than α. If it is larger, output yes, otherwise output no. With probability at least $1-\delta$ the answer will be correct.

Lemma 2.2. *Sampling lemma.*
We require $\mathcal{O}(\frac{1}{\epsilon^2}\log(\frac{1}{\delta}))$ samples from a discrete distribution to determine the frequency of an element of the support with error $\leq \epsilon$ with probability $\geq 1-\delta$.

Proof. Let Y be the random variable corresponding to $\frac{1}{n}\sum_{i=1}^{n}\mathbf{1}\left[\text{cluster } i \text{ consistent with } c\right]$. Then,

$$\mathbb{P}\left\{|Y-\hat{Y}|>\epsilon\right\} < 2e^{-2n\epsilon^2}$$

by Hoeffding's inequality for sums of independent Bernoulli random variables. Setting the RHS to δ gives the desired result. □

The efficiency of this step could be improved if there were an approach to bias the distribution $\mathcal{P}\{c : c \in \mathcal{C}\}$ over the concept class of clusters so that more weight is placed on consistent clusters with respect to a given choice of hypothesis cluster.

2.2.3 Updating the Sampler with Constraints

We now know how to sample given a distribution over clusters in the concept class. However, how do we actually construct the distribution, and how do we update it upon seeing new constraints? We can update the sampling distribution over clusters once we receive feedback from the user in the following fashion:

$$\mathcal{P}\{c|E_{\text{feedback}}, X\} = \frac{\mathcal{P}\{E_{\text{feedback}}|c, X\}\,\mathcal{P}\{c|X\}}{Z} \tag{1}$$

where X is the data, E_{feedback} is merge-split feedback which imposes a constraint set, Z is the new normalization constant summing over the numerator for all possible c.

The first main challenge is to determine whether calculating $\mathcal{P}\{E_{\text{feedback}}|c, X\}$ and the normalization constant is tractable. The second is to figure out how to efficiently update the distribution after receiving more feedback. The third obstacle is to actually define an algorithm for sampling.

2.3 Calculating Probabilities by Memorizing Queries

2.3.1 Assumptions for Efficient Computation

Using merge-split feedback, we can determine whether or not a certain set is α-consistent. If we additionally impose constraints on the concept class so that

(a) $\mathcal{C}$ is efficiently *optimizable*, in the sense that we can efficiently find the *smallest* such concept in $\mathcal{C}$ such that it contains or does not contain some query set $Q \subseteq X$. In particular, we want to find

$$\bigcap_{c: Q \subseteq c, c \in \mathcal{C}} c$$

(b) $\mathcal{C}$ is efficiently *intersectable*, in the sense that we can calculate intersections between sets relatively quickly. This property will be used to define a function which can efficiently calculate whether a proposed cluster is valid.

We will shortly see that T_{update}, the time it takes to update the `cluster-sampler`, depends only upon these two steps.

One way to make $\mathcal{C}$ optimizable is to think in terms of added constraints upon the kinds of hypothesis classes we want to deal with.

Let us now define a function which can tell us whether a candidate concept (cluster) is valid. The idea will be to replace the $\mathcal{P}\{E_{\text{feedback}}|c, X\}$ term from before with a validity function.

2.3.2 The Validity Function

We continue to work in the merge-split setting. Our assumption will be that in the course of our algorithm, we will submit a list of potential clusters and recieve merge-split feedback over the clusters.

Definition 2.3. Validity measure.
Suppose that Q is one of the clusters in our merge-split query whether a set of points Q is contained in a cluster of the true clustering. Then, optimize over concepts to find the "smallest" concept cluster containing Q, call this cluster $\hat{c}_Q$.

Define the validity measure μ_v as a function of a concept cluster c to be

$$\mu_v(\hat{c}_Q, c) := \frac{|\hat{c}_Q \cap c|}{|\hat{c}_Q|}$$

We define the validity measure in this way because it has some nice properties which give us information about whether a concept is valid or not without having to maintain any list over the version space. Notably, converting the validity function into a determiniation of validity or invalidity of any concept/cluster c depends on the result of the split-merge query applied to a particular query point $\hat{c}_Q$.

Lemma 2.4. *Properties of the validity function.*
Let c^* be a relevant concept in the true clustering. Suppose we receive feedback α_Q relating to set Q, where α_Q can be split or merge feedback. There are two cases:

(a) α_Q *is* ***merge****-type feedback: Then,* $\hat{c}_Q \subset c^*$ *and* $\mu_v(\hat{c}_Q, c) \in (0,1)$ *implies that* c *is invalid, while* $\mu_v(\hat{c}_Q, c) \in \{0,1\}$ *does not rule out* c *as a valid cluster.*

(b) α_Q *is* ***split****-type feedback: Then,* $\hat{c}_Q \not\subset c^*$ *and we have that* $\mu_v(\hat{c}_Q, c) \in [0,1)$ *implies that* c *does not rule out* c *as a valid cluster, while* $\mu_v(\hat{c}_Q, c) = 1$ *implies* c *is invalid.*

Proof. If α_Q is merge feedback, then Q is contained inside a true cluster c^*. Since $\hat{c}_Q$ is a concept in $\mathcal{C}$, it also must be contained inside c^* (or is equal to c^*). Thus, the validity measure $\mu_v(\hat{c}_Q, c)$ for arbitrary c is 0 when c does not intersect $\hat{c}_Q$ (and thus we get no information and cannot rule c out), is 1 only when $\hat{c}_Q \subseteq c$ (and thus we cannot rule out that it is the correct cluster, since we know $\hat{c}_Q$ is contained inside the correct cluster). Otherwise, we know that c is not the correct cluster, since any cluster with non-trivial but incomplete intersection cannot be the correct cluster, as $\hat{c}_Q \subseteq c^*$.

If α_Q is split feedback, then Q contains parts of several of the clusters in the true clustering. Thus, so does $\hat{c}_Q$, since $Q \subseteq \hat{c}_Q$. Thus, $\mu_v(\hat{c}_Q, c)$ for arbitrary c is 0 when there is no relation and is fractional when there is a partial intersection. Both of these cases imply nothing about the validity of c, since both cases are acceptable for some true cluster in the clustering. However, if $\mu_v(\hat{c}_Q, c) = 1$, we can rule c out since any cluster which contains parts of several clusters is not pure, and is thus not a true cluster in the clustering. □

Therefore, we can define an indicator function which tells us whether or not any concept is valid given a merge-split query.

Definition 2.5. Validity indicator function.
Given query cluster Q, let $\hat{c}_Q$ be the result of the optimization and c be a input concept which we would like to determine is valid or not. Let $\alpha_Q \in \{\text{merge}, \text{split}\}$ denote the response of a merge-split query with respect to Q. Then, define

$$\mathbb{V}_{\text{merge}}(\hat{c}_Q, c) := \lfloor 2\mu_v(\hat{c}_Q, c) - 1 \rfloor$$

$$\mathbb{V}_{\text{split}}(\hat{c}_Q, c) := \lceil 1 - \mu_v(\hat{c}_Q, c) \rceil$$

and let $\mathbb{V}_{\alpha_Q}(\hat{c}_Q, c)$ be the *validity indicator function.* We can always relax this definition by removing the floor function.

Notably, we need to be able to update this validity function over many queries in an easy way which only involves solving an optimization problem at each new query. Ideally, we would be able to avoid storing too much data in order to compute the function. We can simply sum:

Definition 2.6. Validity indicator function for N queries.
Let $\{\hat{c}_{Q_i}\}_{i=1}^N$ be the result of the optimizations for N proposed clusters $\{Q_i\}_{i=1}^N$. Let $\{\alpha_{Q_i}\}_{i=1}^N$ be the merge-split feedback for each proposed cluster. Then, define the validity indicator function for input concept c to be

$$\mathbb{V}(c) := \left\lfloor \frac{1}{N} \sum_{i=1}^{N} \mathbb{V}_{\alpha_{Q_i}}(\hat{c}_{Q_i}, c) \right\rfloor$$

If any of the the functions resulted in an invalid query, then the input to the floor function will be less than 1, and thus taking the floor will result in zero. In this formulation, it is necessary to store $\{(\hat{c}_{Q_i}, \alpha_{Q_i})\}_{i=1}^N$ in order to calculate this function.

Therefore, we can now calculate

$$\begin{aligned} \mathcal{P}\left\{c|E_{\text{feedback}}, X\right\} &= \frac{\mathcal{P}\left\{E_{\text{feedback}}|c, X\right\}\mathcal{P}\left\{c|X\right\}}{Z} \\ &= \frac{\mathbb{V}(c, \{(\hat{c}_{Q_i}, \alpha_{Q_i})\}_{i=1}^N)\mathcal{P}\left\{c|X\right\}}{\sum_{c\in\mathcal{C}}\mathbb{V}(c, \{(\hat{c}_{Q_i}, \alpha_{Q_i})\}_{i=1}^N)\mathcal{P}\left\{c|X\right\}} \end{aligned} \tag{2}$$

Here, we will take $\mathcal{P}\left\{c|X\right\} = 1$ for any set of points $c \subseteq X$, where $|X| = m$. Thus, we allow improper cluster queries: This choice is fine since in the end, by realizable assumption we have a true clustering with individual clusters in $\mathcal{C}$. Thus, the final probability distribution we need to sample from and update is given by

$$\mathcal{P}\left\{c|E_{\text{feedback}}, X\right\} = \frac{\mathbb{V}(c, \{(\hat{c}_{Q_i}, \alpha_{Q_i})\}_{i=1}^N)}{\sum_{c\subseteq X}\mathbb{V}(c, \{(\hat{c}_{Q_i}, \alpha_{Q_i})\}_{i=1}^N)} \tag{3}$$

Note that this distribution is uniform over the remaining valid clusters. Also note that the geometry of the concept class is not being used to make the sampling algorithm efficient: We are treating the task of actually sampling as a procedure over all possible clusters, though our constraints will eventually ensure we end up only sampling clusters from the concept class.

2.4 Sampling Algorithm

We now consider T_{sample}, the time it takes to actually sample given the probability distribution over clusters. The difficulty of sampling in our setting is due to the fact that we need to calculate the normalization constant over a large discrete set of ones and zeros, as well as come up with an algorithm to sample from the distribution. Kim et al. (2016) directly solves this problem by using the Gumbel-max trick for sampling and formulating the problem as an integer linear program optimization problem. The method takes advantage of linear programming relaxations and branch-and-bound search heuristics, which have been optimized over the years and built into the CPLEX optimization language. This methodology is also capable of taking advantage of parallelism and can reduce runtime.

We provide a translation from our sampling setting to the terminology of Kim et al. (2016). Here, our finite set Σ is the set of possible clusters in X. Our $w : \Sigma \to \mathbb{R}^+$ is exactly $\mathbb{V}$, the validity indicator function.

2.5 Implications of the Results

Previously, the $\mathcal{O}(k^3 \log|\mathcal{C}|)$ query-complexity upper bound algorithm given by Balcan & Blum (2008) was only interesting as an upper bound: It was previously not possible to implement such an algorithm on any practical system. Thus, if one wanted a practical algorithm, it was necessary to construct alternative algorithms which typically depended on specific properties of the hypothesis class. We have identified a set of properties of concept classes which are considerably more general than specific geometric properties. Thus, we can apply our efficient version of Balcan & Blum

(2008)'s algorithm to get lower query complexity implementable algorithms in a variety of settings. Here, we present some examples of applying our method to a specific hypothesis class.

For any particular hypothesis class, we need to present efficient methods for

(a) Finding the smallest hypothesis in the class containing a given set of points.

(b) Calculating the validity indicator function.

2.5.1 d-Dimensional Rectangles

Our example is for the d-dimensional rectangles concept class. Awasthi & Zadeh (2010) provide an efficiently implementable algorithm which depends on the geometry of d-dimensional rectangles and which has a query complexity of $\mathcal{O}((kd\log m)^d)$.

Using Algorithm 1, we can learn a clustering of d-dimensional rectangles with query complexity $\mathcal{O}(k^3 d\log m)$, which is considerably better than the $\mathcal{O}((kd\log m)^d)$ queries attained by Awasthi & Zadeh (2010). Moreover, it is particularly easy to define the optimization and intersection operations efficiently for d-dimensional rectangles.

(a) Optimization: Given a set of points Q, our algorithm to find $\hat{c}_Q$ is to find the max and min value in each of the d dimensions. This takes time at worst $\mathcal{O}(md)$. We then have a representation of a d-dimensional rectangle with $2d$ real values, of the form $[a_1, b_1] \times \cdots \times [a_d, b_d]$.

(b) Intersection: We can calculate $|\hat{c}_Q \cap c| = \prod_{i=1}^d |[a_i, b_i] \cap [w_i, z_i]|$, where a, b denote intervals of $\hat{c}_Q$ and w, z denote intervals of c. Then, $|[a_i, b_i] \cap [w_i, z_i]| = \mathbf{1}\left[d_i > b_i, w_i < b_i\right](b_i - w_i) + \mathbf{1}\left[z_i < b_i, a_i < z_i\right](z_i - a_i)$. Overall, we have

$$\mu_v(\hat{c}_Q, c) = \prod_{i=1}^d \frac{\mathbf{1}\left[d_i > b_i, w_i < b_i\right](b_i - w_i) + \mathbf{1}\left[z_i < b_i, a_i < z_i\right](z_i - a_i)}{b_i - a_i}$$

This operation takes time $\mathcal{O}(d)$.

3 Conclusion and Future Work

In this work, we developed an efficient version of the interactive clustering algorithm of Balcan & Blum (2008) by using sampling to replace the version space. We develop a framework and conditions for which the algorithm can be made efficient, and demonstrate that for the concept class of d-dimensional rectangles, we achieve a merge-split query complexity bound better than that of Awasthi & Zadeh (2010) with a computationally tractable algorithm.

3.1 Implementing and Empirically Evaluating the Algorithm

It remains to actually implement the algorithm and test its effectiveness empirically. We will do this for several practical tasks, including the problem of evaluating generative adversarial networks (GANs) mentioned below.

3.2 Extension to Data-Conditioned Concept Classes

In practice, we may want to try to cluster objects like images or text documents. Geometric concept classes like rectangles and other common learning theoretic concept classes may not measure up as good measures of clusters for these kinds of objects. How can we fix this problem?

Let us consider concept classes for images. One approach might be to take several layers of a pre-trained convolutional neural network, and to then have a linear final layer which would output a 1 if the input image was in the cluster, and a 0 otherwise. Here, the linear final layer would constitute the parametrization of the concept class. The optimization part of the problem is easy: In order to find a concept that matches a cluster of points Q, simply apply supervised learning: Label every point in Q with a 1, and all other points 0 – then train the last layer with gradient descent.

However, it is considerably more difficult to identify the method for finding intersections of two concepts. One might consider trying to invert the network, but it is not clear how one would simply and efficiently calculate the overlap. We leave this task for future work.

3.3 Applications to Evaluating Generative Models

One interesting idea which comes to mind is framing the problem of interactive clustering as a method for sanity-checking the outputs of generative models (for instance, GANs) to see if they indeed have a diverse distribution which is not merely memorized from the data. In general, testing these methods often requires spot-checking and human evaluation – a perfect place for an interactive method to come in, which can help automate the labor of a human. This application is particularly suited to merge-split queries since one is not necessarily able to provide useful labels directly without seeing all of the data (one may not be able to know what the classes are in the beginning, just whether or not certain images should be grouped together). In fact, interactive methods have a place/application for any kind of learning algorithm whose results need to be checked by humans. I believe that interactive learning algorithms could in fact **be more principled** in some cases than just having a human go over the results, since a human may not necessarily realize all the implications of making certain judgements.

To make this concrete, let us return to the GAN example. Say we have a dataset of images upon which we train some GANs. We can then sample from the GANs to look at the output distribution. We would like to compare the distribution of the training data and the distribution of the learned model to see if 1) the output distribution in some way "generalizes" the training data distribution and 2) if the distribution is diverse. Since typically the training data distribution has some structure (for instance, perhaps the data is only of faces, or of rooms), which the human can typically easily recognize, one approach to comparing the distributions which makes sense is some kind of clustering (perhaps hierarchical in some settings). Then, we would reduce the problem to essentially comparing clusters. In the case of hierarchical clustering, we would get extra information which we could use to compare: We would be able to compare the distributions **at each level** of the hierarchy to see if they made sense at every level. Hierarchical clustering would not necessarily make sense for every data distribution naturally, but for certain settings, it seems it would make a lot of sense. Notably, it is difficult for humans to examine the low-resolution images which are typically used in these problems – this is the benefit of using an interactive algorithm. The idea is essentially to use the interactive algorithm and the inherent structure of the chosen clustering hypothesis class in order to impose consistency on the human's spot checks.

Notably, this kind of methodology can augment current approaches. The method of Arora & Zhang (2017) involves birthday paradox tests: Assuming that there are only so many kinds of faces in the world, one can estimate the probability that one would see the same face twice in a uniform distribution over faces. One can compare this "birthday paradox" estimate with the number of collisions one witnesses in practice on the generated dataset of images by the GAN. Since this method is somewhat crude and requires a human to actually check, interactive clustering could be a great alternative.

4 Appendix A: Algorithms

Algorithm 2 Generic Clustering (Inefficient)

procedure CLUSTER(X) ▷ X := input dataset of m points
  version space $V := \mathcal{C}^{VS}$ ▷ $\mathcal{C}^{VS}$:= the set of all k-clusterings on X
  while $|V| > 1$ **do**
    Initialize buckets $B_1, \cdots, B_k := \{\}$.
    Initialize output cluster list $L = []$.
    for each point $x \in X$ in arbitrary order **do**
      for i in $[1, \cdots, k]$ **do**
        if $B_i \cup \{x\}$ is at least $\frac{1}{k^2}$-consistent with V **then** $B_i := B_i \cup \{x\}$
          if B_i is at most $\left(1 - \frac{1}{k^2}\right)$-consistent with V **then**
            Append B_i to L.
            Delete B_i from the list of buckets.
            Append $\{\}$ to the end of the list of buckets.
          end if
          break
        end if
      end for
    end for
    Output cluster list L.
    Receive feedback F from user.
    if $F = \text{merge}(c_i, c_j)$ **then**
      Remove from V all clusterings inconsistent with $c_i \cup c_j$.
    else if $F = \text{split}(c_i)$ **then**
      Remove from V all clusterings consistent with c_i.
    end if
  end while
  return V as the clustering.
end procedure

Algorithm 3 Generic Clustering (Also Inefficient)

procedure CLUSTER(X) ▷ X := input dataset of m points
  version space $V := \mathcal{C}^{VS}$ ▷ $\mathcal{C}^{VS}$:= the set of all k-clusterings on X
  while $|V| > 1$ **do**
    Initialize output cluster list $L = []$.
    Initialize $i = 1$.
    while clusters in L do not cover X **do**
      $c_i = \underset{\substack{x \subset X \setminus L \\ |\text{CCS}(x)| \geq \frac{1}{2}|V|}}{\text{argmax}} |x|$.
      Append c_i to L.
      Update $i = i + 1$.
    end while
    Output cluster list L.
    Receive feedback F from user.
    if $F = \text{merge}(c_i, c_j)$ **then**
      Remove from V all clusterings inconsistent with $c_i \cup c_j$.
    else if $F = \text{split}(c_i)$ **then**
      Remove from V all clusterings consistent with c_i.
    end if
  end while
  return V as the clustering.
end procedure

References

Sanjeev Arora and Yi Zhang. Do gans actually learn the distribution? an empirical study. 2017. URL `https://arxiv.org/abs/1706.08224`. arXiv preprint.

Pranjal Awasthi and Reza Bosagh Zadeh. Supervised clustering. *Advances in Neural Information Processing Systems*, 2010.

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

Sanjoy Dasgupta. Coarse sample complexity bounds for active learning. *Advances in Neural Information Processing Systems*, pp. 235–242, 2005.

Carolyn Kim, Ashish Sabharwal, and Stefano Ermon. Exact sampling with integer linear programs and random perturbations. *Proceedings of AAAI*, 2016.