COMS 6998-4 Fall 2017 Presenter: Kiran Vodrahalli
October 16, 2017 Scribe: Rong Zhou

# Interactive Clustering

## 1 Introduction

Previously we have mostly been discussing binary classification problems, where the active learner gets feedback from a teacher to know if the predictions are correct. We now extend the active learner to solve clustering problems. Clustering algorithms group samples into clusters such that samples within the same cluster are more similar than samples that are not in the cluster. For example, documents are grouped by similar topics or similar writing styles. There has been a wide range of techniques that produce clustering results, but many of them rely on probabilistic assumptions such as Gaussian distributions. Interactive clustering algorithms, in contrast, do not have any distributional assumptions. They query the user who has the *target clustering* in mind to adjust to the "vague" feedback. We will show several examples of interactive clustering and their query complexity analysis below.

### 1.1 Interactive clustering basic setting

Given a set of samples $S$ with size $m$, and the hypothesis class $\mathcal{C}$, we are going partition $S$ to produce $k$ disjoint clusters $\{c_1, \ldots, c_k\}$ such that each $c_i \in \mathcal{C}$. The goal of interactive clustering is to identify the *target clustering* $\{c_1, \ldots, c_k\}$ with as few queries as possible.

### 1.2 Split-merge query feedback

In interactive clustering algorithms, the learner proposes a candidate clustering $\{c'_1, \ldots, c'_k\}$ to the user as a *query*. The user then provides vague responses based on the proposed clustering. Given a proposed clustering $\{c'_1, c'_2, \ldots, c'_k\}$ by the learner, the user's feedback falls into the following two kinds of responses:

- $Split(c'_i)$: The user asks the learner to split samples in cluster $c'_i$ since it contains samples from multiple target clusters.
- $Merge(c'_i, c'_j)$: The user asks the learner to merge the two clusters $c'_i$ and $c'_j$ since they are both subsets of the same target cluster.

Note that the user does not provide the learner counterexamples or any information about how to split the cluster. This is different from equivalence queries we talked about last week since equivalence queries provide specific counterexamples.

### 1.3 A basic example of interactive clustering

We now propose a toy example to show the baseline of the query complexity that an interactive clustering algorithm can achieve.

**Algorithm 1** Algorithm Toy Example

```
1: Begin the clustering $\{c_1, c_2, \ldots, c_m\}$ where each sample forms its own cluster for all $m$ points of $S$.
2: On a $Merge$ request, merge the two clusters indicated.
```

If the user provides correct feedback over time, the target clustering can be obtained by simply merging clusters. Each merge request will reduce the total number of clusters by 1. Since the target clustering has $k$ clusters, the toy algorithm will conduct overall $m - k$ merges, indicating that the number of queries will also be $m - k$.

The above example has query complexity $O(m)$, and we would like to have interactive clustering algorithms with even lower query complexity if the user's feedback has no noise. Ideally, we are looking for algorithms with query complexity $O(\log m)$ or $poly(k, \log m, \log |\mathcal{C}|)$.

## 2 Concrete interactive clustering examples

### 2.1 Clustering 1-D Intervals

Consider an interactive clustering algorithm that produces $k$ clusters on the line, which can be represented as $k$ disjoint intervals. The algorithm is shown below:

**Algorithm 2** Cluster-Intervals

```
1: Begin with a single cluster containing all $m$ points of $S$.
2: On a *split* to a cluster $c'$, partition $c'$ into two clusters of equal cardinality.
3: One a *merge* request, merge the two clusters indicated.
```

**Theorem 1.** *Algorithm 2 requires $O(k \log m)$ queries for clustering the class of intervals on the line.*

*Proof.* Let $a_1, a_2, \cdots a_{k-1}$ as decision boundaries that separate the line into $k$ segments in the target clustering. $a_i$ can be any point between the largest point in the interval $c_i$ and the smallest point in the interval $c_{i+1}$.

Define $size(a_i)$ to be as

$$size(a_i) = \begin{cases} |c'_t| & \text{if } a_i \text{ lies in interval } c'_t \\ 0 & \text{otherwise} \end{cases}$$

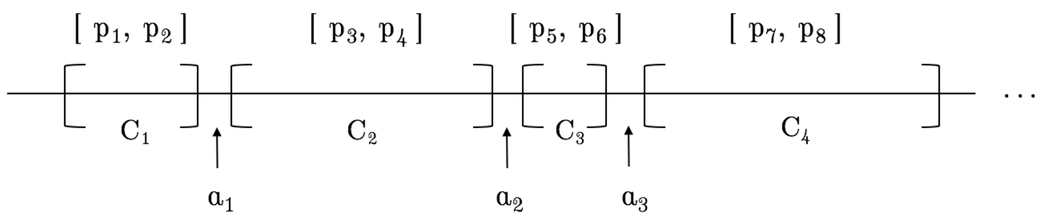

Figure 1: Clustering Intervals

When the target clustering is achieved, $size(a_i)$ will be 0 for all $a_i$. The algorithm starts with $size(a_i) = m$, and as the algorithm proceeds, $size(a_i)$ changes based on the following two requests:

- For a $split(c')$ request, the algorithm makes cluster $c'$ reduce its cardinality by a factor of 2, and for all decision boundaries $a_i$ in $c$, $size(a_i)$ reduces by a factor of 2.
- A $merge(c'_i, c'_j)$ request happens only when neither of the two clusters has any decision boundaries, thus a merge request will not increase or decrease $size(a_i)$ for any $a_i$.

For one decision boundary $a_i$, we need $\log m$ split requests to achieve $size(a_i) = 0$, and $k - 1$ such boundaries require $(k - 1)\log m$ split requests. This indicates that there are at most $(k - 1)\log m$ merge requests. □

## 2.2 Clustering Rectangles

Now we extend the problem to a 2-D space where each cluster is in shape of a rectangle. We parametrize each rectangle using four points $a_i, a_j, b_i$ and $b_j$ to represent the boundaries of a cluster $c_k$ such that if $(x, y) \in c_k$ , then $a_i < x < a_j$ and $b_i < y < b_j$.

The algorithm for clustering rectangles Algorithm 3 is shown below and we will show that the algorithm uses at most $O((k \log m)^2)$ queries. Note that Algorithm 3 can also be extended to clustering rectangles in a $d$-dimensional space. The extended algorithm has query complexity $O((k \log m)^d)$.

**Theorem 2.** *Algorithm 3 can cluster the class of rectangles in 2 dimensions using at most $O((k \log m)^2)$ queries.*

**Proposition 1.** *There is an algorithm which can cluster the class of rectangles in d dimensions using at most $O((kd \log m)^d)$ queries.*

**Algorithm 3** Cluster-Rectangles

```
Require: A graph $\mathcal{G}$ over $m$ points, two sets for maintaining points on the x-axis and the y-axis.
1: Begin with points $a'_{start}$, $a'_{end}$ on the x-axis and $b'_{start}$, $b'_{end}$ on the y-axis such that we have a single cluster containing all $m$ points of $S$.
2: At each step, cluster the $m$ points based on the region, if the points in two regions form a clique in $\mathcal{G}$ merge the regions. Repeat until no more regions can be merged.
3: On a *merge* request, create a clique in $\mathcal{G}$ corresponding the samples in the two clusters.
4: On a *split* to a cluster $a'_i$, $a'_j$, $b'_i$, and $b'_j$, create a new point $a'_r$ such that $a'_i < a'_r < a'_j$, and $a'_r$ divides the projection of all the samples on $(a'_i, a'_j)$ by half in the cluster. Similarly, create a new point $b'_r$ such that $b'_i < b'_r < b'_j$, and $b'_r$ divides the projection of all the samples on $(b'_i, b'_j)$ by half in the cluster.
```

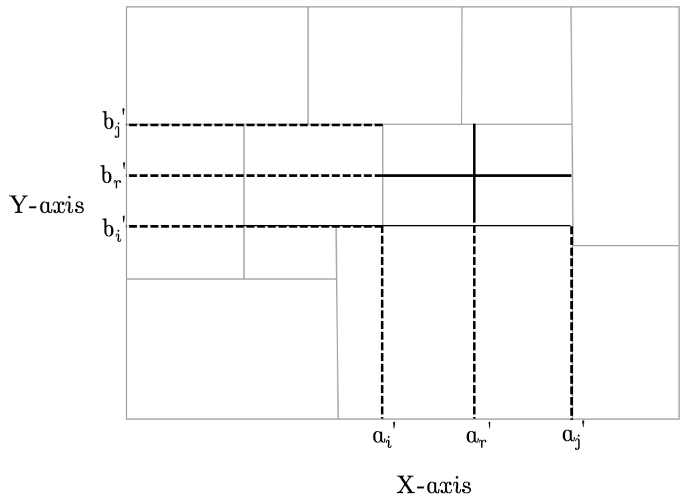

Figure 2: A split request in cluster rectangles

*Proof.* (Proof of Theorem 2) Since there are overall $k$ clusters in the target clustering, there will be at most $2k$ points on the x-axis and $2k$ points on the y-axis. Let $a_1, a_2, \dots$ be the points on x-axis representing the target clustering, and $b_1, b_2, \dots$ be the points on the y-axis representing the target clustering.

Each split of $a'_i, a'_j, b'_i, b'_j$ happens if there exist some target points $a_i$ or $b_j$ within the current region. After splitting, the number of samples in the region that still contains target point will be reduced by a factor of 2. Similar as the previous example, for each axis it requires $O(k \log m)$ splits to achieve $2k$ intervals. As there could be at most $O((k \log m)^2)$ clusters, the number of merges will be at most $((2k-1)\log m)^2$, which is $O((k \log m)^2)$. $\square$

## 3 A generic algorithm for finite class $\mathcal{C}$ and its query complexity

We now discuss a more general algorithm for interactive clustering. Using the idea similar to halving algorithm, the following generic algorithm guarantees a removal of at least $\frac{1}{2}$ of the version space for every *split* or *merge* request.

**Definition 1.** *Given a concept class $\mathcal{C}$, let $\mathcal{C}^{VS} = \{$all possible $k$ clusterings using concepts in the finite class $\mathcal{C}\}$. Let $\mathcal{R}$ be a subset of points in $\mathcal{S}$, a clustering is consistent with $\mathcal{R}$ if $\mathcal{R}$ appears as a subset of one of the clusters in the clustering. A Consistent Cluster Set (CCS) of $\mathcal{R}$ is a set of clusterings that are consistent with $\mathcal{R}$ in $\mathcal{C}^{VS}$. More formally,*

$$CSS(R) = \{\{c_1, c_2, \ldots, c_k\} \in \mathcal{C}^{VS} : \{c_1, c_2, \ldots, c_k\} \textit{ is consistent with } \mathcal{R}\}$$

**Algorithm 4** Generic Clustering

```
1: Initialize $i = 1$, empty cluster list $L$.
2: While $|\mathcal{C}^{VS}| > 1$:
    3: Find the largest set of points $R_i \subseteq S$ , s.t. $|CCS(R_i)| \geq \frac{1}{2}|\mathcal{C}^{VS}|$.
    4: Append $R_i$ into $L$ as a cluster, and increment $i$ by 1.
    5: Repeat step 3-4 until all points in $S$ are assigned to the cluster.
6: Output $L$ to the user.
7: On a split($R_i$) request, remove all the clusterings in $\mathcal{C}^{VS}$ that are consistent with $R_i$.
8: On a merge($R_i, R_j$) request, remove all the clusterings in $\mathcal{C}^{VS}$ that are inconsistent with
   $R_i \cup R_j$.
```

**Theorem 3.** *The generic clustering Algorithm 4 can cluster any finite concept class $\mathcal{C}$ with at most $k \log |\mathcal{C}|$ queries.*

*Proof.* The algorithm starts with the full version space $\mathcal{C}^{VS}$, and note that $|\mathcal{C}^{VS}| < |\mathcal{C}|^k$. For each *split*$(R_i)$ request, the clusterings in $\mathcal{C}^{VS}$ where $R_i$ appears to be a single cluster are incorrect since the user asks the learner to split it. Thus, Algorithm 4 removes all the clusterings that are consistent with $R_i$ in the current version space. This also indicates that at least half of the version space is removed based on line 3 in Algorithm 4.

Similarly, for each *merge*$(R_i, R_j)$ request, the clusterings in $\mathcal{C}^{VS}$ where $R_i$ and $R_j$ appear to be two different clusters are incorrect since the user asks to merge the two clusters. In this case, Algorithm 4 removes all the clusterings that are inconsistent with the idea of merging two clusters and forming $R_i \cup R_j$. This means at least a half of the current version space is removed based on line 3 in Algorithm 4.

Therefore, each request from the user helps the algorithm make progress by eliminating at least half of clusterings from the current version space. The query complexity will be at most $\log |\mathcal{C}^{VS}| \le \log |\mathcal{C}|^k = k \log |\mathcal{C}|$.

The query complexity in Theorem 3 can be further improved if the VC-dimension $d$ of the concept class $\mathcal{C}$ is much smaller than $\log |\mathcal{C}|$. The number of ways to split $m$ points using concept in $\mathcal{C}$ is bounded by $m^d$, and this gives us query complexity

$$\log |\mathcal{C}^{VS}| \le \log m^{kd} = kd \log(m)$$

□

## 4 Lower Bound

Notice that all algorithms mentioned above do not have any restrictions about the number of clusters in the proposed clustering. However, if the algorithm is restricted to produce clusterings with $ploy(k, \log m)$ clusters, then there exist classes that no algorithms can succeed. A simple example can be as follows:

**Theorem 4.** *There exist classes $\mathcal{C}$ of size $m$ such that, even for $k = 2$, any algorithm that is restricted to producing $k$-clusterings will require $\Omega(m)$ queries.*

*Proof.* Consider the situation where all the $m$ points lie on a circle, and the *target clustering* is a random partition with two equal sized intervals. The concept class $\mathcal{C}$ is the class of intervals. The algorithm proposes clusterings with 2 clusters all the time. The user will return *split* request each time when the proposed two intervals are in different sizes. If the two intervals have the same size, the user still returns *split* if the partition is not exactly correct. These feedbacks, on the other hand, do not provide any information to the learner to make progress. Since the target clustering is chosen by random, on expectation the algorithm will require $\Omega(m)$ queries. □

A more general result of the lower bound is proposed and proved in [2]:

**Theorem 5.** *There exist classes $\mathcal{C}$ 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 $\frac{1}{poly(k, \log m)}$ chance of success after $poly(k, \log m)$ queries.*

The analysis of the lower bounds further shows us that it is necessary for algorithms to produce a large number of small clusters, even if the *target clustering* has much smaller number of clusters.

## Bibliographic notes

The problem setting is based on [2]. The Cluster intervals algorithm was proposed by [2]. The Cluster rectangles algorithm is due to [1]. The original generic algorithm is from [2], and here we present an improved version from [1]. The upper bound of the query complexity with known VC-dimension $d$ in the generic algorithm is due to [3].

## References

[1] Pranjal Awasthi and Reza B Zadeh. Supervised clustering. In *Advances in neural information processing systems*, pages 91–99, 2010.

[2] Maria-Florina Balcan and Avrim Blum. Clustering with interactive feedback. In *ALT*, pages 316–328. Springer, 2008.

[3] Vladimir Naumovich Vapnik and Vlamimir Vapnik. *Statistical learning theory*, volume 1. Wiley New York, 1998.