Notes on Balcan and Blum (2008)

Karl Stratos

1 Setting

S denotes a set of m points $x_1 \dots x_m$, and C denotes a concept class. A concept $c \in C$ selects a subset of S: for instance, an interval selects a subset of points in $\mathbb{R}$, a disjunction/conjunction selects a subset of Boolean assignments in $\{0,1\}^n$. For convenience, we will interchangeably write $c \in C$ to denote either the concept or the subset of S that c determines.

Target clustering. There is a particular **target** k-clustering $c_1 \dots c_k \in C$ of S. A hypothesis clustering $h_1 \dots h_k \in C$ is **correct** if, for some permutation σ over $[k]$, the subset of S determined by h_i is the same as the subset of S determined by $c_{\sigma(i)}$ for all $i \in [k]$.

Clustering restrictions. The clusters must be in C and exactly partition S. This requirement makes the model inflexible (Appendix A). The authors alleviates the problem somewhat by introducing the "extended" model where the final cluster is always set to cover remaining points (Appendix B): $c_k := S \backslash c_1 \cup \dots \cup c_{k-1}$.

Interactive clustering. Given S and an unknown target k-clustering $c_1 \dots c_k \in C$ of S, our goal is to obtain a correct hypothesis clustering $h_1 \dots h_k \in C$ by using as few of the following requests (from the user) as possible:

- `split`(h) is issued when there are $x, x' \in S$ such that $x, x' \in h$ but $x \in c_i$ and $x' \in c_j$ for $i \neq j$.
- `merge`(h, h') is issued when $h \subseteq c_i$ and $h' \subseteq c_i$ for some $i \in [k]$.

The hypothesis clustering must be correct when none of these requests is issued. So we can start out with some initial $h_1 \dots h_{k'} \in C$ and continue applying operations until these requests are not issued.

Flexibility in hypothesis clusterings. We consider two sources of flexibility.

1. We allow $k' \neq k$.
2. We allow $h_i \cap h_j \neq \varnothing$ for $i \neq j$.

The paper emphasizes that Flexibility 1 is important.

- Let S be m points on a circle and C be intervals on the circle. Choose a random halving of S as the target 2-clustering in C. If we insist that $k' = k$, then `split` or `merge` cannot lead to any meaningful operations, and all we can do is guess a random halving. So the expected number of requests is $\Omega(m)$.

- If we allow $k' \neq k$, we can initialize h_1, h_2 to be a random halving of S, and find each of the two target boundary by halving h every time `split`(h) is issued. The number of `split` or `merge` requests is therefore $O(\log_2(m))$.

Lemma D.1 gives an example of a cluster class and a dataset such that no algorithm exists that can use only $k' = \text{poly}(\log m, k)$ clusters and find the correct clustering in $\text{poly}(\log m, k)$ requests. The benefit of Flexibility 2 will be evident in the following algorithms.

2 Interval Clustering

Let C be the class of intervals: $c_1 \dots c_k \in C$ is a partition of m points S on the line determined by some k consecutive intervals. Here is an obvious havling algorithm that maintains disjoint hypothesis clusters (so Flexibility 2 is not needed):

1. Initialize $h = \mathbb{R}$.
2. Until no request is made,
   (a) `split`(h): Split h to two intervals each with half the points.
   (b) `merge`(h, h'): Merge h and h'.

This is such a straightforward result, but a proof is still useful because it shows how to be careful with arguments on `split` and `merge`.

Proposition 2.1. *The above algorithm finds the target interval clustering in $O(k \log m)$ requests.*

Proof. For each $i \in [k-1]$, let $x < x'$ be the points in S such that $x \in c_i$ and $x' \in c_{i+1}$. We know they exist because the target clustering must cover all points in S. Set $a_i = (x + x')/2$ as the decision boundary between c_i and c_{i+1} (thus $a_i \notin c_j$ for all $j \in [k]$). At any point in the algorithm with hypothesis intervals $h_1 \dots h_{k'} \in C$, define $\text{size}(a_i) = |h_j|$ if $a_i \in h_j$ for some $j \in [k']$ and $\text{size}(a_i) = 0$ otherwise.

- In the beginning, we must have $\text{size}(a_i) = m$ for all $i \in [k-1]$.
- When we have the correct clustering, we must have $\text{size}(a_i) = 0$ for all $i \in [k-1]$. But this is not a *sufficient* condition for having the correct clustering as some hypotheses clusters may need to be merged.
- We will now argue that we will have split the initial hypothesis clustering $h = \mathbb{R}$ at most $O(m \log k)$ times to reach this point. If this is true, then we need at most $O(m \log k)$ merges to recover the correct clustering, hence the claim follows.

`split`(h) is requested only if $a_i \in h$ for some $i \in [k-1]$, since otherwise h will contain only points in a single target cluster. Furthermore, the havling operation ensures that $\text{size}(a_i)$ is halved. Therefore, for each $i \in [k-1]$, we make at most $O(\log m)$ splits to reach $\text{size}(a_i) = 0$. Thus we make at most $O(k \log m)$ splits to reach $\text{size}(a_i) = 0$ for all $i \in [k-1]$. □

Extended model. The extended model can be treated as the non-extended model with $2k-1$ intervals ($k-1$ intervals plus k intervals to catch the rest), so it can also be learned in $O(k \log m)$ requests.

3 Disjunction Clustering

Let C be the class of disjunctions: $c_1 \dots c_k \in C$ is a partition of m Boolean assignments S in $\{0,1\}^n$ determined by some k disjuctions. WLOG we assume that there is no negated literal (just introduce more dimensions).

Remark. Partitioning Boolean assignments using disjunctions is awkward and unnatural (see Appendix A). Here, however, we don't worry about this part. We simply assume the existence of some valid target disjunction clustering, and exploit this assumption to derive an analyzable interactive clustering algorithm.

3.1 Overlapping Hypothesis Clusters

We exploit the limitation of C that if a literal z_i is used in any target disjuction $c \in C$, this c must contain all assignments $x \in S$ with $x_i = 1$.

1. Initialize $h_i = z_i$ for $i \in [n]$ (and remove empty clusters).
2. Until no request is made,
   (a) `split`(h): Remove h.
   (b) `merge`(h, h'): Remove h and h' and introduce $h \vee h'$.

Let's analyze what happens at the initialization.

- We have at most n hypothesis clusters each of which corresponds to some literal z_i. Denote the cluster corresponding to z_i by h_i. They may overlap, so Flexibility 2 is used.
- If z_i is used in a target disjunction c, then all assignments in h_i must belong to c. This is because c is a disjunction and we assume a valid target clustering (in particular, it must be the case that z_i is used only in c in the target clustering).
- Thus if assignments in h_i are not pure, then z_i is not used in any target disjunction and h_i can be removed. Note that each $x \in S$ stays covered by some hypothesis cluster.
- `merge`(h, h') is issued only for clusters h, h' that cannot be split, and the merged cluster $h \vee h'$ will never be split in the future.

Since the number of hypothesis clusters is reduced by one at either `split` or `merge`, we find the correct clustering in at most $n - k$ requests.

Extended model. The algorithm is the same except that it's possible for some $x \in S$ to become uncovered. Since this happens only if x is in c_k, so we can simply maintain a default bucket to put these uncovered points as we proceed. We still find the correct clustering in at most $n - k$ requests.

3.2 Disjoint Hypothesis Clusters

To maintain disjoint hypothesis clusters during interactive clustering, we have to change the algorithm slightly.

- A quick way to achieve disjoint initial clusters is to define n clusters $h_1 \dots h_n$ where now h_i contains all $x \in S$ such that $x_i = 1$ and $x_j = 0$ for $j \in [i-1]$. For example, if $S = \{0,1\}^3 \setminus \{000\}$,

$$\begin{aligned} 100, 101, 110, 111 &\in h_1 \\ 010, 011 &\in h_2 \\ 001 &\in h_3 \end{aligned}$$

- Again, if $\texttt{split}(h_i)$ is issued, it must be that z_i is never used in any target disjunction c (otherwise, if z_i is a literal in c, then every point in h_i must be in c). So we can try to remove this cluster.
- However, upon removal of h_i all $x \in h_i$ becomes uncovered. It's not obvious how to reassign them so that we can guarantee that a merged cluster will never be split in the future (we want this invariant).
- The simplest fix is to remove the i-th *variable* z_i in all S and restart.

Here's the resulting algorithm:

1. Initialize $h_i = \{x \in S : \ x_i = 1, \ x_j = 0 \ \forall j < i\}$.
2. Until no request is made,
   (a) $\texttt{split}(h_i)$: Remove z_i from all S and restart the algorithm.
   (b) $\texttt{merge}(h_i, h_j)$: Remove h and h' and introduce $h \cup h'$.

Each episode of the algorithm consists purely of $O(n)$ merges, so the total runtime is $O(n^2)$.

Extended model. We can maintain a default bucket to put all-zero-literal assignments as we proceed (an assignment $x \in S$ becomes all-zero only if it's in c_k). We still find the correct clustering in at most $O(n^2)$ requests.

4 Conjunction Clustering

Let C be the class of conjunctions: $c_1 \dots c_k \in C$ is a partition of m Boolean assignments S in $\{0,1\}^n$ determined by some k conjuctions. As before, we disallow negated literals.

Remark. Partitioning $S \subset \{0,1\}^n$ using conjunctions is strictly more expressive than disjunctions. Suppose $S = \{10, 01, 11\}$ and we want to have each point contained in its own cluster. This clustering is not possible with disjunctions (in the non-extended model). With conjunctions, we can set $c_1 = (z_1 \wedge \bar{z}_2)$, $c_2 = (\bar{z}_1 \wedge z_2)$, and $c_3 = (z_1 \wedge z_2)$. Without negated literals, we can equivalently introduce $z_3 = \bar{z}_1$ and $z_4 = \bar{z}_2$ and cluster $S = \{1001, 0110, 1100\}$ using $c_1 = (z_1 \wedge z_4)$, $c_2 = (z_2 \wedge z_3)$, and $c_3 = (z_1 \wedge z_2)$.

4.1 Reduction to Disjunctions

Because we assume that the target clustering is a valid clustering that partitions S (with no empty cluster), the same clustering is given by $(k-1)$-DNFs

$$c_i = \bigwedge_{j=1:\ j\neq i}^{k} \bar{c}_j \tag{1}$$

where the negated conjunction c_j is a disjunction of negated literals (well, their non-negated counterparts):

$$\bar{c}_j = \bigvee_{l\in L_j} \bar{z}_l$$

(the set L_j is some subset of $[n]$). Distributing disjunctions across $k-1$ conjunctions, we can write (1) as

$$\begin{aligned} c_i &= \bigwedge_{j=1:\ j\neq i}^{k} \left(\bigvee_{l\in L_j} \bar{z}_l \right) \\ &= \bigvee_{t\in T} \left(\bar{z}_{t_1} \wedge \cdots \wedge \bar{z}_{t_{k-1}} \right) \\ &= \bigvee_{t\in T} y_{t_1 \ldots t_{k-1}} \end{aligned}$$

where T is some selection of $k-1$ elements from $[n]$. Thus each c_i is now a disjunction over n^{k-1} variables $y_{t_1\ldots t_{k-1}} := \bar{z}_{t_1} \cdots \bar{z}_{t_{k-1}}$, and we can use the interactive clustering algorithm for disjunctions to find the correct clustering in $O(n^{k-1})$ requests ($O(n^{2(k-1)})$ if overlapping clusters are disallowed).

Extended model. In the extended model, c_k is not necessarily expressed as a conjunction over $z_1 \ldots z_n$, thus $\bar{c}_k$ is not necessarily a disjunction over $\bar{z}_1 \ldots \bar{z}_n$. Consequently, c_i is *not* a $(k-1)$-DNF except when $i = k$, so we can't just run the disjunction algorithm as before. What we can do is to exploit the fact that c_k *is* a $(k-1)$-DNF and can be written as a disjunction

$$c_k = \bigvee_{t\in T} y_{t_1\ldots t_{k-1}}$$

1. Initialize $h_{t_1\ldots t_{k-1}} = y_{t_1\ldots t_{k-1}}$ for all n^{k-1} selections of $k-1$ integers from $[n]$ (and remove empty clusters).
2. Keep a default bucket B.
3. Until no request is made,
   (a) `split`(h): Remove h and put all points in h to B. Run $(k-1)$-disjunction algorithm inside B.
   (b) `merge`(h, h'): Merge h and h'. If one of them is in B then put the merged cluster in B and run $(k-1)$-disjunction algorithm inside B.

This works because any $y_{t_1\ldots t_{k-1}}$ used in c_k is pure and cannot be split. Thus only points that belong to $c_1 \cup \cdots \cup c_{k-1}$ will be sent to B, in which the problem is reduced to the non-extended $(k-1)$-disjunction model. Since there can be at most n^{k-1} calls to either the $O(n^{k-2})$ or $O(n^{2k-4})$ algorithms, the total runtim is $O(n^{2k-3})$ or $O(n^{3k-5})$.

5 Generic Clustering

Let C be any finite concept class (e.g., if C is the class of conjunctions over $\{0,1\}^n$, then $|C| = 2^n$). Let C^{VS} denote all possible k-clusterings with C: note that $\left|C^{VS}\right| \leq |C|^k$. Our target clustering is just some $c \in C^{VS}$. The authors provide a "halving" algorithm for interactive clustering in this general scenario.

1. Initialize the version space C^{VS}.
2. While $\left|C^{VS}\right| > 1$,
   (a) $(h_1 \dots h_k) \leftarrow$ **GenerateInterestingClustering**(C^{VS}, S)
   (b) Receive a single request on $h_1 \dots h_k$. Remove all clusterings in C^{VS} that conflict with this request:
   - If $\texttt{split}(h)$: Remove all clusterings in C^{VS} such that the points in h belong to the same cluster.
   - If $\texttt{merge}(h, h')$: Remove all clusterings in C^{VS} such that the points in h and h' do not belong to the same cluster.

We will not describe **GenerateInterestingClustering**, but the claim is that Step 2b removes $1/k^2$ portion of the version space C^{VS}. Then the number of requests r required to obtain $\left|C^{VS}\right| \leq 1$ is

$$\left(1 - \frac{1}{k^2}\right)^r \left|C^{VS}\right| \leq 1 \quad \iff \quad r \leq -\frac{1}{\log\left(1 - \frac{1}{k^2}\right)} \log\left|C^{VS}\right|$$
$$\leq k^2 \log\left|C^{VS}\right|$$
$$\leq k^3 \log|C|$$

A Valid Clusterings

Let C be the class of disjunctions over $z_1, z_2, z_3, z_4 \in \{0,1\}$. Suppose $S = \{1100, 0100, 0010\}$ and $k = 2$. Invalid target clusterings of S include:

- (Multiple-membership) $c_1 = (z_1 \vee z_2)$ and $c_2 = (z_2 \vee z_3)$

  $1100 \in c_1 \cap c_2$ $\quad$ $0100 \in c_1 \cap c_2$ $\quad$ $0010 \in c_2$

- (Incomplete coverage) $c_1 = (z_1)$ and $c_2 = (z_3)$

  $1100 \in c_1$ $\quad$ $0100 \notin c_1 \cup c_2$ $\quad$ $0010 \in c_2$

- (Empty cluster) $c_1 = (z_2 \vee z_3)$ and $c_2 = (z_4)$

  $1100 \in c_1$ $\quad$ $0100 \in c_1$ $\quad$ $0010 \in c_1$

Valid target clusterings of S include:

- $c_1 = (z_1 \vee z_2)$ and $c_2 = (z_3)$

  $1100 \in c_1$ $\quad$ $0100 \in c_1$ $\quad$ $0010 \in c_2$

- $c_1 = (z_1)$ and $c_2 = (z_2 \vee z_3)$

  $1100 \in c_1$ $\quad$ $0100 \in c_2$ $\quad$ $0010 \in c_2$

B Extended Model

Let C be the class of disjunctions over $z_1, z_2, z_3 \in \{0,1\}$. Suppose $S = \{101, 011, 001\}$. We wish to put each point in a separate cluster (so set $k = 3$), but this is not possible with C. Under the extended model, we can set $c_1 = (z_1)$ for 101 and $c_2 = (z_2)$ for 011, whereupon the default cluster c_3 handles 001.

As another example, let C be the class of conjunctions over $z_1, z_2, z_3 \in \{0,1\}$. Suppose $S = \{111, 011, 001\}$. We want two clusters $\{111, 001\}$ and $\{011\}$ but this is not possible with C: $z_3 = 1$ is the only common factor between 111 and 001, but it is also shared by 011. Under the extended model, we can set $c_1 = (\bar{z}_1 \wedge z_2)$ and let the default cluster c_2 handle 111 and 001.

C The Fourier Basis

Consider the vector space of functions from n Boolean variables to $\mathbb{R}$:

$$\mathcal{F} := \{f : \{0,1\}^n \to \mathbb{R}\}$$

The **inner product** between $f, g \in \mathcal{F}$ is

$$\langle f,\ g\rangle := \mathbf{E}\left[f(x)g(x)\right] = \frac{1}{2^n}\sum_{x\in\{0,1\}^n} f(x)g(x)$$

The **norm** of $f \in \mathcal{F}$ is given by $\sqrt{\langle f,\ f\rangle}$.

Lemma C.1. *$S := \{\bar{e}_\alpha : \alpha \in \{0,1\}^n\}$ is a basis of $\mathcal{F}$, where $\bar{e}_\alpha \in \mathcal{F}$ is defined as $\bar{e}_\alpha(x) := [[\alpha = x]]$. In particular, $\dim(\mathcal{F}) = 2^n$.*

Proof. S is orthogonal (though not normal). Any $f \in \mathcal{F}$ can be written as

$$f(x) = \sum_{\alpha \in \{0,1\}^n} f(\alpha)[[\alpha = x]]$$

□

Define $P := \{\chi_\alpha \in \mathcal{F} : \alpha \in \{0,1\}^n\}$ as the set of **parity functions**:

$$\chi_\alpha(x) := (-1)^{\sum_{i:\ \alpha_i = 1} x_i}$$

Note that

$$\mathbf{E}\left[\chi_\alpha(x)\right] = \begin{cases} 0 & \text{if } \alpha \neq 0 \\ 1 & \text{otherwise} \end{cases} \tag{2}$$

$$\chi_\alpha(x)\chi_\beta(x) = \chi_{\alpha \oplus \beta}(x) \tag{3}$$

where $\oplus$ is the bit-wise XOR operation. From (2) and (3), we see that

$$\langle \chi_\alpha,\ \chi_\beta \rangle = \mathbf{E}\left[\chi_{\alpha\oplus\beta}(x)\right] = \begin{cases} 1 & \text{if } \alpha = \beta \\ 0 & \text{otherwise} \end{cases} \tag{4}$$

Proposition C.1 (The Fourier basis)**.** *P is an orthonormal basis of $\mathcal{F}$.*

Proof. P is orthonormal by (4); in particular, $\dim(P) = 2^n$. Since $\dim(\mathcal{F}) = 2^n$ by Lemma C.1, the claim follows. □

Thus any $f \in \mathcal{F}$ can be written as

$$f(x) = \sum_{\alpha \in \{0,1\}^n} \langle \chi_\alpha,\ f \rangle\ \chi_\alpha(x)$$

from which the following claim can be easily verified.

Proposition C.2 (Perseval's identity)**.** *For any $f \in \mathcal{F}$,*

$$\sum_{\alpha \in \{0,1\}^n} \langle \chi_\alpha,\ f \rangle^2 = \boldsymbol{E}\left[f(x)^2\right]$$

Corollary C.2. *For any $f : \{0,1\}^n \to \{\pm 1\}$,*

$$\sum_{\alpha \in \{0,1\}^n} \langle \chi_\alpha,\ f \rangle^2 = 1$$

Perseval's identity can be used to argue that any given $f : \{0,1\}^n \to \{0,1\}$ cannot correlate strongly with too many parity functions. Concretely, the number of $\alpha \in \{0,1\}^n$ such that $|\langle \chi_\alpha,\ f \rangle| \geq C$ cannot exceed $1/C^2$, since otherwise

$$\sum_{\alpha \in \{0,1\}^n} \langle \chi_\alpha,\ f \rangle^2 > \frac{1}{C^2} C^2 = 1$$

Reference Learning Boolean Functions via the Fourier Transform (Mansour, 1994)

D Lemmas

Lemma D.1. *Consider the setting:*

- *C is the class of parity functions c and their negations $\neg c(x) := -c(x)$ over $\{0,1\}^n$. Given $x \in \{0,1\}^n$ and $c \in C$, we say $x \in c$ if $c(x) = 1$ and $x \notin c$ if $c(x) = -1$.*
- *$S = \{0,1\}^n$. That is, we consider all $m = 2^n$ points for clustering.*
- *$k = 2$. Denote the target 2-clustering of S by $c \in C$ and $\neg c = S \backslash c$. Assume that c is picked uniformly at random from 2^n possible parity functions.*

Then for all interactive clustering algorithms using only $k' = poly(k, \log m)$ clusters, the expected number of requests needed to recover c is exponential in $poly(k, \log m)$.

Proof. Let $c' \in C$ denote the largest hypothesis cluster. If $|c'| = \alpha m$, we must have $\alpha \geq 1/k' = 1/\text{poly}(k, \log m)$. We will show that the probability that $\texttt{split}(c')$ cannot be issued is at most $1/(2^n 4\alpha^2)$ using the properties of parity functions. This probability is then bounded as

$$\frac{1}{2^n 4\alpha^2} \leq \frac{\text{poly}(k, \log m)}{2^n 4} = \frac{\text{poly}(k, n)}{2^n 2^k}$$

which is exponentially small in $n = \log m$ and k. Since this is the case at any point in the algorithm, the number of requests we need to cluster correctly is exponential in $\log m$ and k as argued.

The only way $\texttt{split}(c')$ cannot be issued is either $c' \subseteq c$ or $c' \subseteq \neg c$. Define a parity function $h : \{0,1\}^n \to \{\pm 1\}$ associated with c': $h(x) = 1$ if $x \in c'$ and $h(x) = -1$ otherwise. We will show that if $\texttt{split}(c')$ cannot be issued, we must have $|\langle c,\ h\rangle| = 2\alpha$. Then we invoke Parseval's identity to state that there are at most $1/4\alpha^2$ such target parity functions c. Since we choose c at random, the probability that such c is picked is at most $1/(2^n 4\alpha^2)$.

All that remains to show is that when $c' \subseteq c$ or $c' \subseteq \neg c$,

$$|\langle c,\ h\rangle| = |\mathbf{E}\left[h(x)c(x)\right]| = |\Pr(h(x) = c(x)) - \Pr(h(x) \neq c(x))|$$

is equal to 2α

- Suppose $c' \subseteq c$. Then

$$\Pr(c'(x) = -1, c(x) = -1) = \Pr(c'(x) = -1 | c(x) = -1) \times \Pr(c(x) = -1) = \frac{1}{2}$$
$$\Pr(c'(x) = 1, c(x) = 1) = \Pr(c(x) = 1 | c'(x) = 1) \times \Pr(c'(x) = 1) = \alpha$$

  so that $\Pr(h(x) = c(x)) = \frac{1}{2} + \alpha$. Then $|\langle c,\ h\rangle| = |2\alpha|$.

- Suppose $c' \subseteq \neg c$. Then similarly $\Pr(h(x) = \neg c(x)) = \frac{1}{2} + \alpha$ and $|\langle c,\ h\rangle| = |-2\alpha|$.

□