Chapter 10: Covering problems

10. Covering problems

- Given a set of concepts (rules etc.) that apply to examples (rows of data etc.)
- The concept *covers* the examples
- How to find good small collections of concepts?
- Not all concepts satisfying certain conditions

Example: association rules

- For fixed B
- Given a bunch of rules of the form $W \Rightarrow B$
- Each rule applies (covers) the rows t such that $t[W] = 1$, i.e., $W \subseteq t$
- Find a set of rules which cover all examples in a given subset of the data
- Find a set of rules which cover as much of the data as possible

Prototype problems

- **Set cover problem**: find a small set of concepts such that all examples are covered by some concept in the set
- **Best collection problem**: find a set of size k of concepts that covers as many examples as possible
- Both problems are NP-complete
- Simple approximation algorithms with provable properties

Set cover problem

- Given a universe $X = p_1, \ldots, p_n$
- Sets $S_1, S_2, \ldots, S_m \subseteq X$, $\cup S_i = X$
- $\mathcal{F} = \{S_1, S_2, \ldots, S_m\}$.
- Question: find the smallest number of set from $\mathcal{F}$ whose union is X
- I.e., find a smallest subcollection $\mathcal{C} \subseteq \mathcal{F}$ such that

$$\cup_{S \in \mathcal{C}} S = X.$$

- NP-complete (what does it mean?)

Trivial algorithm

- Try all subcollections of $\mathcal{F}$
- Select the smalles one that covers X
- Running time $\mathcal{O}(2^{|\mathcal{F}|}|X|)$
- Too slow

Greedy algorithm for set cover

- Select first the largest set S
- Remove the elements of S from X
- Recompute the sizes of the sets
- Go back to the first step

As an algorithm

1. $U = X$;
2. $\mathcal{C} = \emptyset$;
3. While U is not empty do
   - For all $S \in \mathcal{F}$ let $a_S = |Y_i \cap U|$
   - Let S be such that a_S is maximal;
   - $\mathcal{C} = \mathcal{C} \cup \{S\}$
   - $U = U \setminus S$

How can this go wrong?

- No global consideration of how good or bad the set will be

Weighted version

- Each set $S \in \mathcal{F}$ has a cost $c(S)$
- Compute the number of elements per unit cost: $(S \cap U)/c(S)$
- At each step, select the S for which this is maximal

Running time of the algorithm

- Polynomial in $|X|$ and $\mathcal{F}$
- At most $min(|X|, |\mathcal{F}|)$ iterations of the loop
- Loop body takes time $\mathcal{O}(|X||\mathcal{F}|)$
- Running time $\mathcal{O}(|X||\mathcal{F}|min(|\chi|, |\mathcal{F}|))$
- Can be implemented in linear time $\mathcal{O}(\sum_{S\in\mathcal{F}} |S|)$

Related problems

- Given a graph $G = (V, E)$
- Independent set: find the largest set V' of vertices such that there is no edge between any two vertices in V'
- Glique: find the largest set of vertices V' such that all pairs of vertices of V' are connected by an edge
- Clique in G is an independent set in $\overline{G} = (V, V \times V \backslash E)$

Related problems

- Given a graph $G = (V, E)$
- **Vertex cover**: find the smallest subset $V' \subseteq V$ such that for each edge $(u, v) \in E$ we have $u \in V'$ or $v \in V'$

Approximation algorithms

- Consider a minimization problem
- Instance I, cost of optimal solution $\alpha^*(I)$, approximate solution with cost $\alpha(I)$
- The algorithm has approximation ratio $c(n)$, if for all instances of size at most n we have $\alpha(I) \leq c(n)\alpha^*(I)$
- The greedy algorithm has approximation ratio $\mathcal{O}(log\ n)$ (i.e., d log n for some d).

Approximation algorithm for vertex cover

- $C = \emptyset$;
- Select a random edge (u, v)
- $C = C \cup \{u, v\}$;
- Remove all edges that are incident either with u or with v
- Repeat until no edges remain

Approximation guarantee

- The result is a vertex cover (why?)
- No two selected edges share an endpoint
- For any edge (u, v) at least one of u and v has to belong to any vertex cover
- For any edge (u, v) at least one of u and v has to belong to the optimal vertex cover
- Thus $\alpha(G) \leq 2\alpha^*(G)$ for all G

Analysis of the greedy approximation algorithm for set cover

- $H(d) = \sum_{i=1}^{d} 1/i$: the ith harmonic number
- Greedy approximation algorithm has approximation ratio $H(s)$, where s is the size of the largest set in $\mathcal{F}$
- (Trivial bound; s)
- $H(s) \approx ln\ s$, i.e., the bound is quite good $(s \leq |X|)$

Proof.

- Source: Cormen et al., Introduction to Algorithms
- Optimal set cover $\mathcal{C}^*$ and the cover $\mathcal{C}$ produced by the greedy algorithm
- S_i: the ith set selected by an algorithm
- Cost of 1 (counting the number of sets)
- Spread this among the elements covered for the first time by S_i
- Cost to item x is $c_x = (|S_i \setminus\ (S_1 \cup \ldots S_{i-1})|)^{-1}$
- Costs by set S_i sum up to one
- $|\mathcal{C}| = \sum_{x \in X} c_x$

$\mathcal{C}^*$ covers X

$$|\mathcal{C}| = \sum_{x \in X} c_x \leq \sum_{S \in \mathcal{C}^*} \sum_{x \in S} c_x$$

For any set $S \in \mathcal{F}$ we have (proof separate)

$$\sum_{x \in S} c_x \leq H(|S|)$$

Thus

$$|\mathcal{C}| \leq \sum_{S \in \mathcal{C}^*} H(|S|)$$

and hence i

$$|\mathcal{C}| \leq \mathcal{C}^* H(s)$$

where $s = max\{|S| : S \in \mathcal{F}\}$

Best collection problem

- Given some concepts (rules etc.)
- Find the best collection of k rules
- For example, find the k sets whose union has maximum size
- Maximization problem, quality $f(\mathcal{C})$ of the result = number of elements in $\cup_{S \in \mathcal{C}} S$
- Simple approximation algorithm has bound

$$\alpha \geq \frac{e-1}{e} \alpha^*$$

Submodular functions

- The result is very general
- Concepts $\mathcal{F}$, task fo find the best subcollection $\mathcal{C}^* \subseteq \mathcal{F}$ of size k
- Solution function f should satisfy for all $\mathcal{C}$

$$f(\mathcal{C}) \geq 0$$

  and for all $\mathcal{C} \subseteq \mathcal{D} \subseteq \mathcal{F}$ and $S \in \mathcal{F}$

$$f(\mathcal{C} \cup \{S\}) - f(\mathcal{C}) \geq f(\mathcal{D} \cup \{S\}) - f(\mathcal{D})$$

  i.e., the improvement obtained by adding S may not increase when moving to a larger solution
- f is submodular; the result holds for all such functions

Greedy approximation algorithm

- $\mathcal{C} = \emptyset$
- Gain of S in the context of $\mathcal{C}$ is $f(\mathcal{C} \cup \{S\}) - f(\mathcal{C})$
- Select the concept S that has the highest gain
- $\mathcal{C} := \mathcal{C} \cup \{S\}$
- Repeat until $\mathcal{C}$ has k elements

Basic theorem

Let $\mathcal{C}_k^*$ be the optimal set of k concepts

Let $\mathcal{C}_i$ be the ith set formed by the greedy algorithm.

Assume

$$f(\mathcal{C}_i) - f(\mathcal{C}_{i-1}) \geq \frac{1}{k}(f(\mathcal{C}_k^*) - f(\mathcal{C}_{i-1}))$$

Then

$$f(\mathcal{C}_k) \geq \frac{e-1}{e} f(\mathcal{C}_k^*)$$

Proof. Separate.

Why does the assumption hold?

$$f(\mathcal{C}_i) - f(\mathcal{C}_{i-1}) \geq \frac{1}{k}(f(\mathcal{C}_k^*) - f(\mathcal{C}_{i-1}))$$

f is submodular
the greedy approximation algorithm
The concept $\mathcal{C}_i \setminus \mathcal{C}_{i-1}$ is the one that maximizes the gain.
(Something open here.)

Applications

- Which functions are submodular?
- The concepts should not have interaction
- Variable selection?

Chapter 11: Clustering

Clustering

- Task: group observations into groups so that the observations belonging to the same group are similar, whereas observations in different groups are different
- Lots and lots of research in various areas
- Just scratching the surface here

Basic questions?

- What does "similar" mean?
- What is a good partition of the objects? I.e., how is the quality of a solution measured?
- How to find a good partition of observations?

What does "similar" mean?

- Some function of the attribute values of the observations
- Usual approach: L_p distance

$$L((x_1, \dots, x_n), (y_1, \dots, y_n)) = (\sum_i (x_i - y_i)^p)^{1/p}$$

- Easy in 1-dimensional real case
- Already 2 dimensions cause problems: how to weigh the different dimensions?
- Lots of problems

Partition-based clustering

- Data mining algorithms: task; model; score function; search
- Task: partition the data into K disjoint sets of points
- The points within each set are as homogeneous as possible
- Measured by score function
- Often no clear model

Score functions for clustering

- $d(x, y)$: distance between points $x, y \in D$
- Assume d is a metric
- $\mathcal{C} = (C_1, C_2, \ldots, C_K)$
- Clusters should be compact
- Clusters should be as far from each other as possible
- Within cluster variation $wc(\mathcal{C})$ and between cluster variation $bc(\mathcal{C})$

- Cluster centers $r_1, \ldots, r_K$: representative points from each cluster, e.g., the centroid of the points
- Simple measure for within cluster variation

$$wc(\mathcal{C}) = \sum_{k=1}^{K} wc(C_k) = \sum_{k=1}^{K} \sum_{x \in C_k} d(x, r_k)^2$$

- Between cluster variation

$$bc(\mathcal{C}) = \sum_{1 \le j < k \le K} d(r_j, r_k)^2$$

- $wc(\mathcal{C})$ leads to spherical clusters

- Evaluation of $bc(\mathcal{C})$ and $wc(\mathcal{C})$?
- $\mathcal{O}(n)$ and $\mathcal{O}(K^2)$ operations
- Variations abound: define $wc(C_k)$ as the maximum of the minimum distance to another point in the same cluster
- Leads to elongated clusters

The K-means algorithm

- randomly pick K cluster centers
- assign each point to the cluster whose mean is closest in a Euclidean distance sense
- compute the mean vectors of the points assigned to each cluster
- use these as new centers
- repeat until convergence

As an algorithm

data points $D = \{\mathbf{x}_1, \ldots, \mathbf{x}_n\}$
find K clusters $\{C_1, \ldots, C_K\}$:

for $k = 1, \ldots, K$ let $\mathbf{r}_k$ be a randomly chosen point from D;
while changes in clusters C_k happen **do**
  form clusters:
  for $k = 1, \ldots, K$ **do**
   $C_k = \{\mathbf{x} \in D \mid d(\mathbf{r}_k, \mathbf{x}) \leq d(\mathbf{r}_j, \mathbf{x})$ for all $j = 1, \ldots, K, j \neq k\}$;
  od;
  compute new cluster centers:
  for $k = 1, \ldots, K$ **do**
   $\mathbf{r}_k =$ the vector mean of the points in C_k
  od;
od;

Properties of the K-means algorithm

- Finds a local optimum
- Converges often quite quickly
- Sometimes slow convergence
- For high dimensions the initial points can have a large influence

Hierarchical clustering

- Merge sets of points or divide sets of points
- Agglomerative or divisive
- Dendrograms (figures)

Agglomerative clustering

for $i = 1, \ldots, n$ let $C_i = \{\mathbf{x}(i)\}$;
while there is more than one cluster left **do**
  let C_i and C_j be the clusters
    minimizing the distance $\mathcal{D}(C_k, C_h)$ between
    any two clusters;
  $C_i = C_i \cup C_j$;
  remove cluster C_j ;
od;

Complexity

- Quadratic, at least, in the number of points
- Not usable for large sets of data

What is the distance?

- How to define the distance between two clusters?
- Two sets of points
- Lots of alternatives
- Actually quite difficult to define a metric

Single-link distance

$d(\mathbf{x}, \mathbf{y})$ the distance between objects $\mathbf{x}$ and $\mathbf{y}$

$$\mathcal{D}_{sl}(C_i, C_j) = \min_{\mathbf{x},\mathbf{y}} \{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}, \tag{1}$$

chaining: long, elongated clusters

Complete link

Furthest distance

$$\mathcal{D}_{fl}(C_i, C_j) = \max_{\mathbf{x},\mathbf{y}} \{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x} \in C_i, \mathbf{y} \in C_j\}, \tag{2}$$

leads to equal colume (or at least diameter)

Other measures

- For vectors
- the centroid measure (the distance between two clusters is the distance between their centroids)
- the group average measure (the distance between two clusters is the average of all the distances between pairs of points, one from each cluster)
- Ward's measure for vector data (the distance between two clusters is the difference between the total within cluster sum of squares for the two clusters separately, and the within cluster sum of squares resulting from merging the two clusters discussed above)

Divisive methods

- start with a single cluster composed of all of the data points
- split this into components
- continue recursively
- *Monothetic* divisive methods split clusters using one variable at a time
- *Polythetic* divisive methods make splits on the basis of all of the variables together
- any intercluster distance measure can be used
- computationally intensive, less widely used than agglomerative methods

Kleinberg's impossibility theorem

- John Kleinberg, An impossibility theorem for clustering, NIPS 2002
- Clustering methods based on pairwise distances
- Three properties for clustering methods
- No algoritm can have all three

Computational task

- A clustering function operates on a set S of n points
- No underlying space; $S = \{1, 2, \ldots, n\}$
- Distance function: $d : S \times S \to \mathbf{R}$ with $d(i, j) \geq 0$, $d(i, j) = d(j, i)$, and $d(i, j) = 0$ only if $i = j$
- (Metric: additionally have $d(i, j) + d(j, k) \geq d(i, k)$)
- Clustering function $f : f(S, d) = \Gamma$, where Γ is a partition of S
- (A partition)

Scale invariance

$\alpha > 0$; distance function αd has values $(\alpha d)(i, j) = \alpha d(i, j)$
For any d and for any $\alpha > 0$ we have $f(d) = f(\alpha d)$

Richness

The range of f is equal to the set of partitions of S

I.e., for any S and any partition Γ of S there is a distance function d on S such that $f(S, d) = \Gamma$

Consistency

Shrinking distances between points inside a cluster and expanding distances between points in different clusters does not change the result.

Γ a partition of S

$d,\ d'$ two distance functions on S

d' is a Γ-transformation of d, if

- for all $i, j \in S$ in the same cluster of Γ we have $d'(i,j) \leq d(i,j)$
- for all $i, j \in S$ in the different cluster of Γ we have $d'(i,j) \geq d(i,j)$

Consistency: if $f(S,d) = \Gamma$ and d' is a Γ-transformation of d, then $f(S,d') = \Gamma$

Examples

- Agglomerative clustering with single-link
- Repeatedly merge cluster whose distance is minimum
- Continue until a stopping criterion is met
  - k-cluster stopping criterion: continue until there are k connected components
  - distance-r stopping criterion: continue until all distances between clusters are larger than r
  - scale-α stopping criterion: let ρ^* be the maximum pairwise distance; continue until all distances are larger than $\alpha\rho^*$

Examples, cont.

- Single link with k-cluster stopping criterion satisfies scale-invariance and consistency
- Single link with distance-r stopping criterion satisfies richness and consistency
- Sigle link with scale-α stopping criterion satisfies richness and scale-invariance

Theorem

For each $n \geq 2$ there is no clustering function that satisfies scale-invariance, richness, and consistency

Proof of theorem

A partition Γ' is a refinement of partition Γ, if each set $C' \in \Gamma'$ is included in some set $C \in \Gamma$

A partial order between partitions: $\Gamma' \leq \Gamma$

Antichain of partitions: collection of partitions such than no one is a refinement of others

Theorem: If a clustering function f satisfies scale-invariance and concistency, then the range of f is an antichain

Γ-forcing

- partition Γ
- d (a, b)-conforms to Γ, if for all points i, j in the same cluster of Γ $d(i, j) \leq a$, and for all points i, j in different clusters of Γ $d(i, j) \geq b$
- given a clustering function f
- (a, b) is Γ-forcing, if for all d that (a, b)-conform to Γ we have $f(S, d) = \Gamma$

Forcing, cont.

- Assume f satisfies consistency; let $\Gamma \in Range(f)$
- Claim: there are $a < b$ such that (a, b) is Γ-forcing
- Γ in range of f: there is d such that $f(S, d) = \Gamma$
- a'=minimum distance of points in the same cluster in Γ
- b'=maximum distance of points in different clusters in Γ
- Choose $a < b$ such that $a < a'$ and $b < b'$
- If d' (a, b)-conforms to Γ, then d is a Γ-transformation of d
- By consistency $f(d') = \Gamma$
- Thus (a, b) is Γ-forcing

Antichains

- Assume f satisfies scale-invariance
- Let Γ_0 and Γ_1 be possible results of f, and let Γ_0 be a refinement of Γ_1
- Show that this leads to contradiction
- (a_0, b_0) Γ_0-forcing, (a_1, b_1) Γ_1-forcing
- Let $a_2 < a_1$, choose ϵ so that $0 < \epsilon < a_0 a_2 b_0^{-1}$
- Construct a d such that
  - For i, j in same cluster of Γ_0 we have $d(i, j) \leq \epsilon$
  - For i, j in same cluster of Γ_1 but not in Γ_0 we have $a_2 \leq d(i, j) \leq a_1$
  - For i, j in different clusters of Γ_1 $d(i, j) \geq b_1$

- d (a,b)-conforms to Γ_1, and thus $f(S,d) = \Gamma_1$
- $\alpha = b_0 a_2^{-1}$, and let $d' = \alpha d$
- scale-invariance: $f(d') = f(d) = \Gamma_1$
- i, j in same cluster of Γ_0 we have

$$d'(i,j) \leq \epsilon b_0 a_2^{-1} < a_0$$

- i, j in different clusters of Γ_0 we have

$$d'(i,j) \geq a_2 b_0 a_2^{-1} = b_0$$

- d' (a_0, b_0) conforms to Γ_0, and thus $f(S, d') = \Gamma_0 \neq \Gamma_1$, contradiction