Frequent Itemsets

1. Overview

The market-basket model

- Describe a common form of many-to-many relationship between two kinds of objects.
  - A large set of items. e.g.: things sold in a supermarket.
  - A large set of baskets, each of which is a small set of items. e.g.: things one customer buys on one trip to the supermarket.
    - Number of baskets cannot fit into memory.

Definition of a frequent itemsets

- A set of items that appears in many baskets is said to be frequent.
- Assume a value s: *support threshold*.
- If I is a set of items.
  - The support for I is the number of baskets in which I is a subset.
- I is frequent if its support is s or higher.

Example: frequent itemsets

- items = {milk, coke, pepsi, beer, juice}
  - B1: m,c,b
  - B2: m,p,j
  - B3: m,b
  - B4: c,j
  - B5: m,p,b
  - B6: m,c,b,j
  - B7: c,b,j
  - B8: b,c
- Support value s = 3 (three baskets)
- Frequent itemsets:
  - {m}, {c}, {b}, {j}, {m,b}, {b,c}, {c,j}

Applications

- Items: products; Baskets: sets of products.
  - Given that many people buy beer and diapers together: run a sale on diapers and raise price of beer.

  - Given that many people buy hotdog and mustards together: run a sale of hotdog and raise price of mustards.
- Items = documents; baskets = sentences/phrases.
  - Items that appear together too often could represent plagiarism.
- Items = words, basket = documents.
  - Unusual words appearing together in large number of documents indicating interesting relationship.

Scale of the problem

- Walmart sells hundreds of thousands of items, and has billions of transactions (shopping basket/cart at checkout).
- The Web has billions of words and many billions of pages.

2. Association Rules:

Definition

- If-then rules about the contents of baskets.
- {i_1, i_2, ..., i_k} -> j means: "If a basket contains all of i_1,…,i_k then it is likely to contain j."
- Confidence of this association rule is the probability of j given {i_1,…,i_k}.
- The fraction of the basket with {i_1,…,i_k} that also contain j.
- Example:
  - B1: m,c,b
  - B2: m,p,j
  - B3: m,b
  - B4: c,j
  - B5: m,p,b
  - B6: m,c,b,j
  - B7: c,b,j
  - B8: b,c
- An association rule: {m,b} -> c
  - Basket contains m and b: B1, B3, B5, B6
  - Basket contains m, b, and c: B1, B6
  - C = 2 / 4 = 50%

Finding association rules

- **Find all association rules with support >= s and confidence >=c**
- Hard part: finding the frequent itemsets.

Computation model

- Data is stored in flat files on disk.
- Most likely basket-by-basket.
- Expand baskets into pairs, triples, etc as you read the baskets.
    - Use k nested loops to generate all sets of size k.
- I/O cost: per passes (all baskets read).

Main memory bottleneck

- For many frequent-itemset algorithms, main memory is the critical resource.
- We need to keep count of things (occurrences of pairs/triples/…) when we read baskets.
- The number of different things we can count is limited by main memory.
- Swapping counts is going to be horrible.

3. Algorithms

Naive algorithm

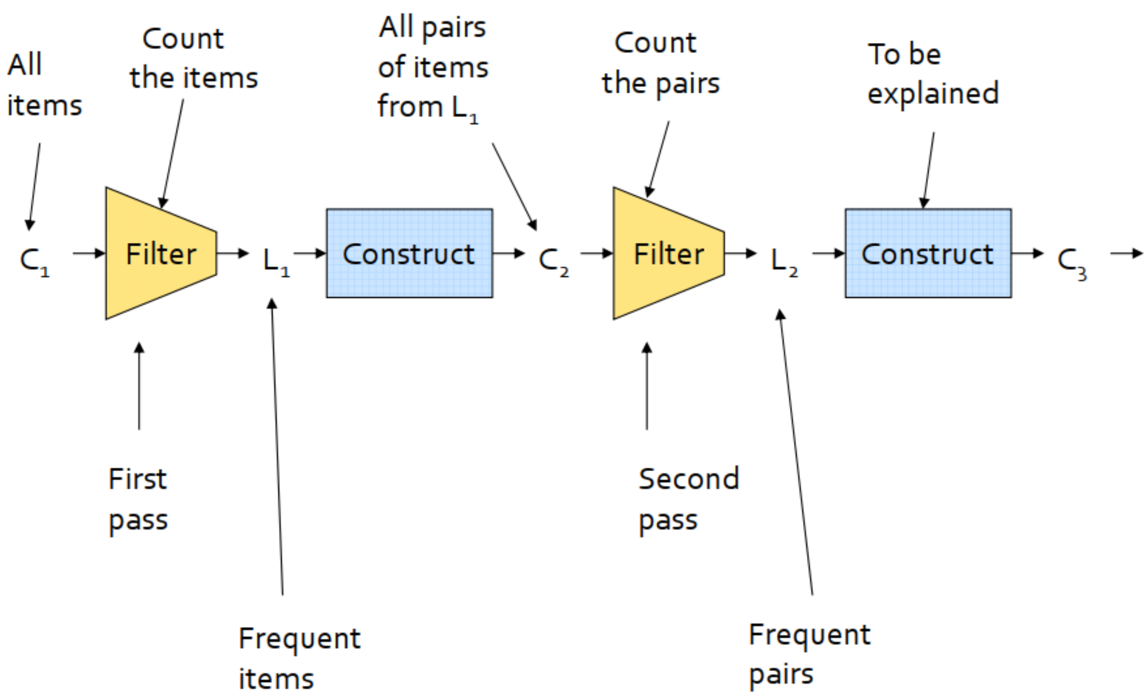
- Hardest problem is finding frequent pairs because they are the most common.
- Read file once, counting in main memory the occurences of each pair.
- For each basket of n items, there will be n(n-1)/2 pairs, generated by double-nested loops.
- If n^2 exceeds main memory, we fails.

Naive algorithm: how do we count

- Count all pairs using a triangular matrix.
    - Requires 4 bytes per pair for all possible pairs: 2n^2
- Keep a table ot triples [i, j, c] with c is the count of pair {i,j}.
    - Requires 12 bytes only for pairs with count > 0: 12p with p is the number of pairs that actually occur.

A-Priori algorithm

- Limit the need for main memory.
- Key idea: monotonicity
  - If a set of items appears at least s times, so does every subset of this set.
- Contrapositive: If an item i does not appear in s baskets, then no pair containing i can appear in s baskets.
- A-Priori algorithm:
  - Pass 1: read baskets and count the item occurrences. Only keep items that appear at least s times - frequent items.
  - Pass 2: read baskets again and only count in main memory only those pairs whose both items were found to be frequent from Pass 1.
  - Repeat the process with increasing number of items added to only sets found to be frequent.



- C1 = all items
- In general, L_k are members of C_k with support greater than or equal to s.
- C_(k+1) includes (k+1) sets, each k of which is in L_k.

A-Priori at scale

- Under two passes.
- SON: Savasere-Omiecinski-Navathe
- Adaptable to a distributed data model (mapreduce).
    - Repeatedly read small subsets of the baskets into main memory and perform a-priori on these subsets, using a support that is equal to the main support divided by the total numbers of subsets.
    - Aggregate all candidate itemsets and determine which are frequent in the entire set.

1. Overview

General problem statement

- Given a **set of data points**, with a notion of **distance** between points, **group the points** into some number of **clusters** so that:
  - Members of a cluster are close/similar to each other.
  - Members of different clusters are dissimilar.
- Usually
  - Points are in high-dimensional space (observations have many attributes).
  - Similarity is defined using a distance measure: Euclidean, Cosine, Jaccard, edit distance …

Clustering is a hard problem

- Clustering in two dimensions looks easy.
- Clustering small amounts of data looks easy.
- In most cases, looks are **not** deceiving.
- But:
  - Many applications involve not 2, but 10 or 10,000 dimensions.
  - High-dimensional spaces look different.

Example

- Clustering Sky Objects:
  - A catalog of 2 billion **sky objects** represents objects by their radiation in 7 dimensions (frequency bands)
  - Problem: cluster into similar objects, e.g., galaxies, stars, quasars, etc.
- Clustering music albums
  - Music divides into **categories**, and customer prefer a few categories

- Are **categories** simply genres?
- Similar Albums have similar sets of customers, and vice-versa

- Clustering documents
  - Group together documents on the same topic.
  - Documents with similar sets of words maybe about the same topic.
  - Dual formulation: a topic is a group of words that co-occur in many documents.

Distance measures: Cosine, Jaccard, Euclidean

- Different ways of representing documents or music albums lead to different distance measures.
- Document as set of words
  - Jaccard distance
- Document as point in space of words.
  - x_i = 1 if i appears in doc.
  - Euclidean distance
- Document as vector in space of words.
  - Vector from origin to …
  - Cosine distance.

2. Methods of clustering

Overview

- Hierarchical:
  - Agglomerative (bottom up): each point is a cluster, repeatedly combining two nearest cluster.
  - Divisive (top down): start with one cluster and recursively split it.
- Point assignment:
  - Maintain a set of clusters
  - Points belong to nearest cluster
- The curse of dimensionality:

    - In high dimensions, almost all pairs of points are equally far away from one another.
    - Almost any two vectors are almost orthogonal.

Hierarchical clustering

- Initial decisions
    - How will clusters be represented?
    - When to merge/split clusters?
    - When to stop?
- Basic algorithm is not very efficient ($O(n^3)$)
- A slightly more efficient approach
    - Calculate distance of all pairs ($O(n^2)$)
    - Create a priority queue for all pairs and their distance ($O(n^2)$)
    - Remove all entries in the queue that were merged (($O(\log n)$))
    - Calculate distances based on new clusters' centroids.

Point assignment: K-means clustering

- Assumes Euclidean space/distance
- Pick k, the number of clusters.
- Initialize clusters by picking on point per cluster.
- Until converge
    - For each point, place it in the cluster whose current centroid it is nearest.
        - A cluster centroid has its coordinates calculated as the averages of all its points' coordinates.
    - After all points are assigned, update the locations of centroids of the k clusters.
    - Reassign all points to their closest centroid.

The big question

- How to select k?
- Try different k, looking at the change in the average distance to centroid, as k increases.
- Approach 1: sampling
  - Cluster a sample of the data using hierarchical clustering, to obtain k clusters.
  - Pick a point from each clsuter (e.g. point closest to centroid)
  - Sample fits in main memory.
- Approach 2: Pick dispersed set of points
  - Pick first point at random
  - Pick the next point to be the one whose minimum distance from the selected points is as large as possible.
  - Repeat until we have k points.

3. Extensions to large data: BFR

Overview

- Bradley-Fayyad-Reina: a variant of k-means designed to handle very large (disk-resident) data sets.
- Assumes clusters are normally distributed around a centroid in a Euclidean space
- Key approach: summarize clusters instead of keep track of points.

BFR: k selection

- Select k centroids from the initial data set
  - Take k random points, or
  - Take a small random sample and cluster optimally, or

- Take a sample, pick a random point, then k-1 more points, each as far from the previously selected points as possible.

BFR: storing data points

- Keep track of three sets of points
  - Discard set (DS)
    - Points close enough to a centroid to be summarized
  - Compression set (CS)
    - Group of points that are close together but not close to any existing centroid
    - These points are summarized but not assigned to a cluster
    - Points are discarded after summarized.
  - Retained set (RS)
    - Isolated points waiting to be assigned to a CS
    - Actual data points storage here
  - For each cluster (a DS), all the points in the DS is summarized by
    - The number of points, N
    - The vector SUM, whose i_{th} component is the sum of the coordinates of the points in the i_{th} dimension.
    - The vector SUMSQ, similar to SUM, but is sum of squares instead.
    - Recalling movie rating: to find average we need sum (SUM) and count (N).
      - SUMQ is for variance (tightness of cluster).
    - $2d+1$ storage requiment to represent any cluster of any size

        ■ d is the number of dimensions

BFR: actual clustering

- Load a memory-full of points
- Find points that are sufficiently close to a cluster centroid and add those points to that cluster and the DS.
    - Adjust statistics of clusters in DS
    - Implied discard
- Use any main-memory clustering algorithm to cluster the remaining points from the current memory load **and** the RS.
    - Summarized clusters go to CS
    - Consider merging clusters in CS
    - Outliers (points) go to RS
- If this is last iterations, merge all compressed sets in CS and RS points to their nearest cluster.

BFR: questions

- How close is close enough
    - Mahalanobis distance is less than a threshold.
- Should two clusters be combined?
    - Combine statistics (N, SUM, SUMQ)
    - Combine if the combined variance is small (below some threshold)

4. Extensions to large data: CURE

Overview

- BFR problems
    - Normal distribution assumtion for all dimensions
    - Axes are fixed
- CURE: Clustering Using Representatives
    - Assume Euclidean distance
    - Allow clusters to assume any shape

- Use a collection of representative points to represent clusters

CURE Pass 1

- Pick a sample of points that fit in main memory
- Cluster these points **hierarchically**.
- Pick a sample of points from each cluster, as dispersed as possible
  - Pick representatives from sample by, for example, moving them 20% toward the centroids.

CURE Pass 2

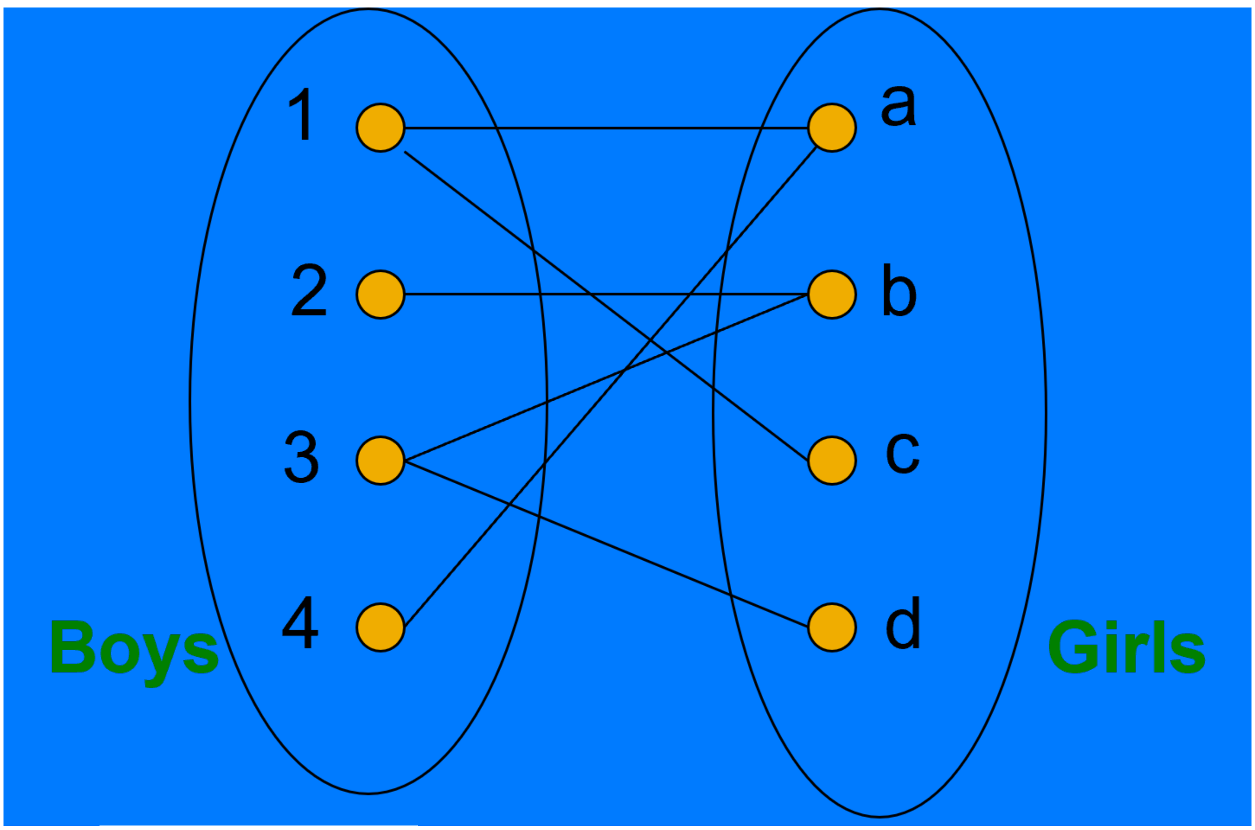
- Rescan the dataset and visit each point p.
- Place it in the **closet** cluster
  - Find the nearest representative to p and assign p to that representative's cluster.

1. Online algorithms

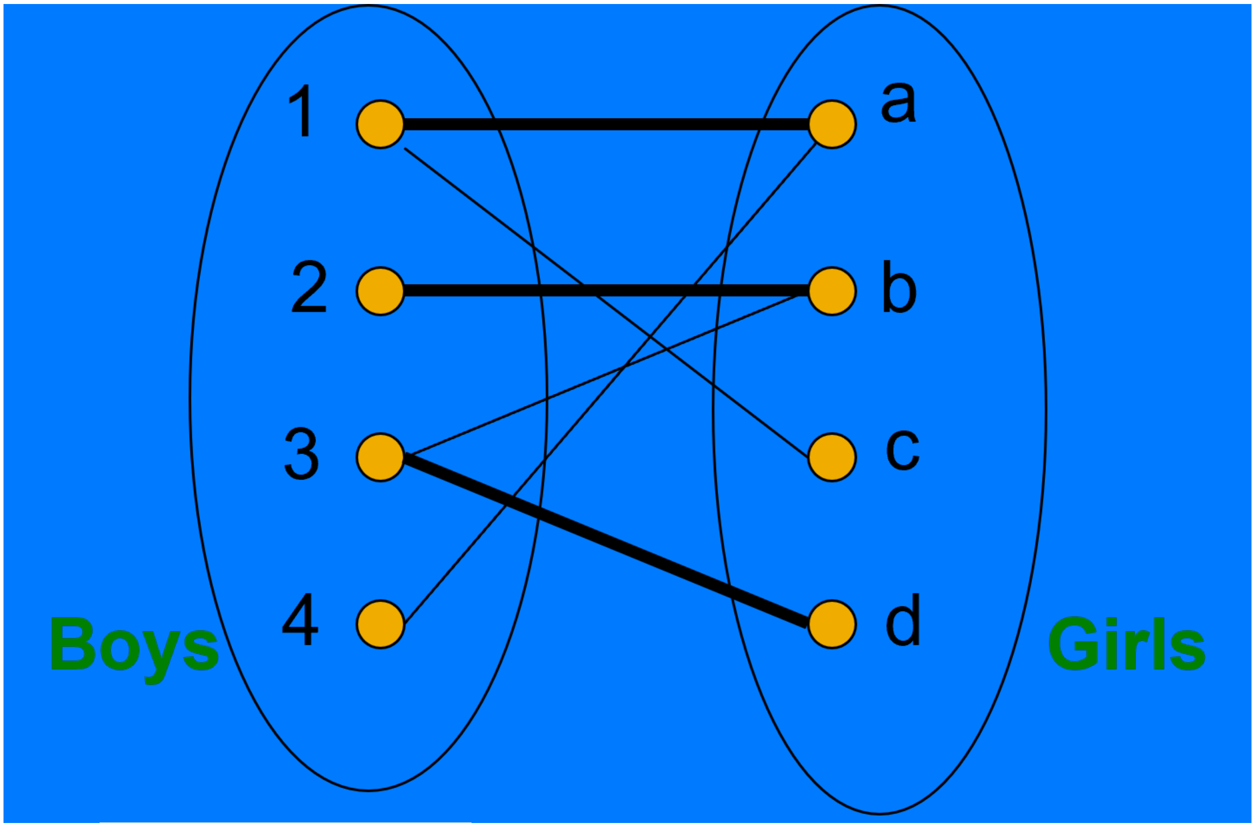
Overview

- Classic model
  - All inputs are observable and can be used in computation
  - `offline algorithm`.
- Online algorithms:
  - Input are observed one at a time.
  - Decision are made based on observed input and cannot be changed.
  - Similar to data streaming model.

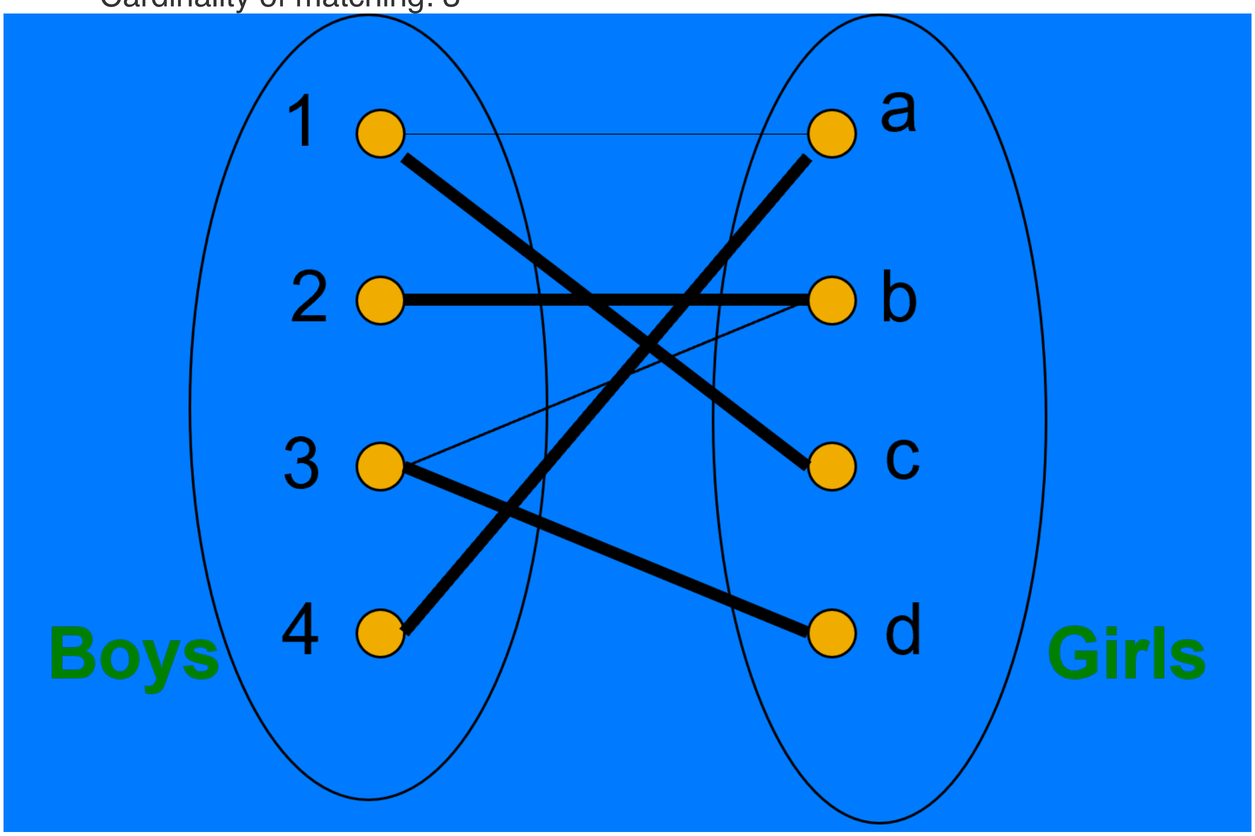
Example: Bipartitate Matching



- Nodes: Boys and Girls
- Edges: Preferences
- Goal: Match boys to girls so that maximum number of preferences is satisfied.



- Matches: (1,a), (2,b), (3,d)
- Cardinality of matching: 3



- Matches: (1,c), (2,b), (3,d), (4, a)
- Cardinality of matching: 4 (perfect matching)

Matching algorithm

- Offline: polynomial-time
  - Hopcroft-Karp algorithm
  - But what if we don't know the entire graph upfront?
- Online: greedy algorithm
  - Given: set of boys
  - Each round a new girl's preferences are revealed
    - Do we pair immediately or not?
  - Similar applications:
    - Scheduling tasks on servers
- Key question for online performance:
  - Don't expect for best performance similar to offline algorithms
  - How close can you get?

Web Advertising

History of Web Advertising

- Banner ads (1995-2001)
- Initial form of web advertising
- Popular websites charged `X$` for every 1,000 `impressions` of the ad
  - Called `CPM` rate (Cost per thousand (`mille` in Latin) impressions)
  - Modeled similar to TV, magazine ads
- From untargeted to demographically targeted
- Low click-through rates (CTR)
  - Low ROI for advertisers

Performance-based Advertising

- Introduced by Overture around 2000
  - Advertisers bid on search keywords
  - When someone searches for that keyword, the highest bidder's ad is shown
  - Advertiser is charged only if the ad is clicked on
- Similar model adopted by Google with some changes around 2002
  - Called Adwords
  - `Sponsored Links`

Web 2.0

- Performance-based advertising works!
    - Multi-billion-dollar industry
- Interesting problem:
    - What ads to show for a given query? (this lecture)
    - If I am an advertiser, which search terms should I bid on and how much should I bid? (not in scope for the class)

Adwords Problem

Overview

- Given:
    1. A set of bids by advertisers for search queries
    2. A click-through rate (CTR) for each advertiser-query pair
    3. A budget for each advertiser (say for 1 month)
    4. A limit on the number of ads to be displayed with each search query
- Respond to each search query with a set of advertisers such that:
    1. The size of the set is no larger than the limit on the number of ads per query
    2. Each advertiser has bid on the search query
    3. Each advertiser has enough budget left to pay for the ad if it is clicked upon

More specifics

- A stream of queries arrives at the search engine: $q1, q2, \ldots$
- Several advertisers bid on each query
- When query qi arrives, search engine must pick a subset of advertisers whose ads are shown
- Goal: **Maximize search engine's revenues**
    - Simple solution: Instead of raw bids, use the `expected revenue per click` (i.e., Bid*CTR)
- Advertisers' list

Advertisers	**Bid**	**CTR**	**Bid * CTR**
A	$1.00	1%	1 cent

B	$0.75	2%	1.5 cent
C	$0.50	2.5%	1.125 cents

- Maximum profits

Advertisers	Bid	CTR	Bid * CTR
B	$0.75	2%	1.5 cent
C	$0.50	2.5%	1.125 cents
A	$1.00	1%	1 cent

- Clearly we need an online algorithm!
- Complications:
    - Budget: each advertiser has a limited budget (Greedy anyone?)
    - CTR of an add is unknown

Complications: CTR

- CTR: Each ad has a different likelihood of being clicked
    - Advertiser 1 bids $2, click probability = 0.1
    - Advertiser 2 bids $1, click probability = 0.5
- Clickthrough rate (CTR) is measured historically
- Very hard problem: Exploration vs. exploitation
    - Exploit: Should we keep showing an ad for which we have good estimates of click-through rate
    - Explore: Shall we show a brand new ad to get a better sense of its click-through rate

Algorithms

Greedy Algorithm

- Our setting: Simplified environment
    - There is 1 ad shown for each query
    - All advertisers have the same budget B
    - All ads are equally likely to be clicked
    - Value of each ad is the same (1)
- Simplest algorithm is greedy:
    - For a query pick any advertiser who has bid 1 for that query
    - Competitive ratio of greedy is 1/2

Greedy Algorithm: Bad scenario

- Two advertisers A and B
  - A bids on query x, B bids on x and y
  - Both have budgets of $4
- Query stream: x x x x y y y y
  - Worst case greedy choice: B B B B _ _ _ _
  - Optimal: A A A A B B B B
  - Competitive ratio: $1/2$
- This is the worst case!
  - Note: Greedy algorithm is deterministic – It always resolves draws in the same way

BALANCE algorithm (MSVV)

- Earlier result: An Optimal Algorithm for On-line Bipartite Matching by Karp, Vazirani, and Vazirani
  - Lower bound for competitive rate: $1-1/e$
- Original work: An optimal deterministic algorithm for online b-matching by Kalyanasundaram and Pruhs.
  - Unweighted bipartite graph G = (S,R,E) where S and R are the two vertex partitions and E is the edge set.
  - At the ith unit of time, $1/leqi/leqn$, vertex $ri∈R$ and all edges incident to $r\ i$ are revealed to the algorithm A.
  - A must then either decline to ever service ri or irrevocably select a site $S\ k$ adjacent to $r\ i$ in G to service $r\ i$
  - Also achieve lower bound $1-1/e$
- Adwords and Generalized online matching by Mehta, Saberi, Vazirani, and Vazirani
  - Common generalization of RANKING and BALANCE
  - Also achieve lower bound $1-1/e$
- For each query, pick the advertiser with the largest unspent budget.
  - Break ties arbitrarily

- More specifics:
    - A bidder pays only if the user clicks on his ad.
    - Advertisers have different daily budgets.
    - Instead of charging a bidder his actual bid, the search engine company charges him the next highest bid.
    - Multiple ads can appear with the results of a query.
    - Advertisers enter at different times.

Example BALANCE

- Two advertisers A and B
    - A bids on query x, B bids on x and y
    - Both have budgets of $4
- Query stream: x x x x y y y y
- BALANCE choice: A B A B B B _ _
    - Optimal: A A A A B B B B
- In general: For BALANCE on 2 advertisers Competitive ratio: $3/4$

BALANCE: General result

- In the general case, worst competitive ratio of BALANCE is $1-1/e$ = approx. 0.63
    - No online algorithm has a better competitive ratio!
- Can be generalized to arbitrary bids.
- Key issue: finding the correct trade-off between the bid and (fraction of) the unspent budget.
- BALANCE tradeoff function:
    - $\psi(x)=1-e^{(x-1)}$
- BALANCE algorithm: Allocate the next query to bidder i maximizing the product of their bid and $\psi(T(i))$ where $T(i)$ is the fraction of the bidder budget that has been spent so far