DATA MINING
LECTURE 13

Pagerank, Absorbing Random Walks

Coverage Problems

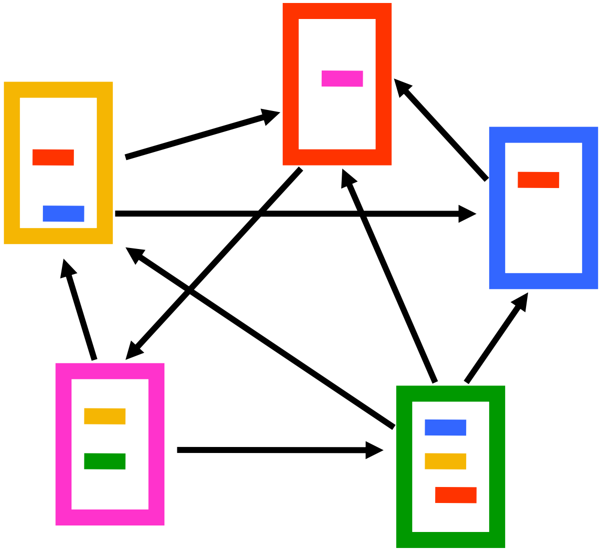
PAGERANK

PageRank algorithm

- The PageRank random walk
  - Start from a page chosen uniformly at random
  - With probability α follow a random outgoing link
  - With probability 1- α jump to a random page chosen uniformly at random
  - Repeat until convergence
- The PageRank Update Equations

$$PR(p) = \alpha \sum_{q \to p} \frac{PR(q)}{|Out(q)|} + (1-\alpha)\frac{1}{n}$$

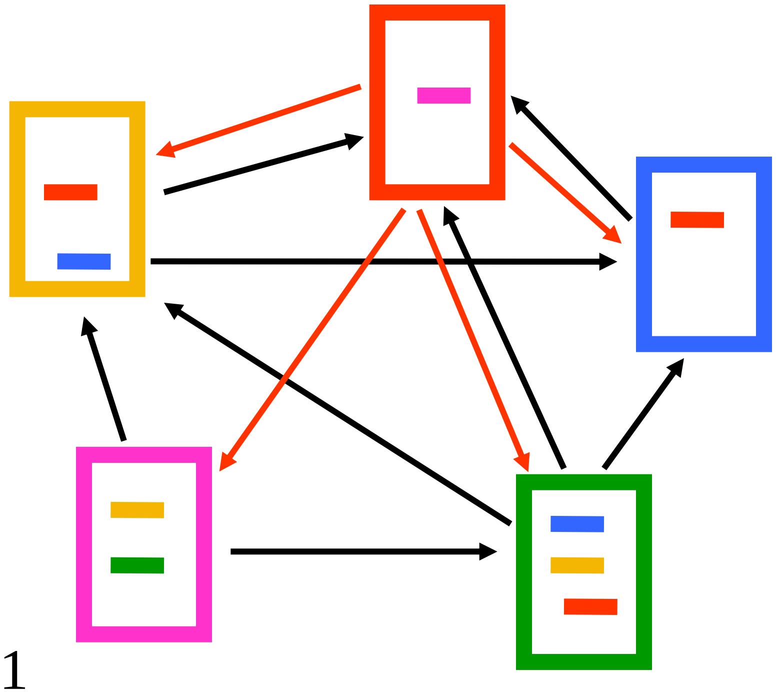
$\alpha = 0.85$ in most cases



The PageRank random walk

- What about sink nodes?
  - When at a node with no outgoing links jump to a page chosen uniformly at random

$$P' = \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 1/2 & 0 \end{bmatrix}$$



$$PR(p) = \alpha \sum_{q \to p} \frac{PR(q)}{|Out(q)|} + \alpha \sum_{q:q \text{ is sink}} \frac{PR(q)}{n} + (1-\alpha)\frac{1}{n}$$

The PageRank random walk

- The PageRank transition probability matrix
  - P was sparse, P'' is dense.

$$P'' = \alpha \begin{bmatrix} 0 & 1/2 & 1/2 & 0 & 0 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 0 & 1 & 0 & 0 & 0 \\ 1/3 & 1/3 & 1/3 & 0 & 0 \\ 1/2 & 0 & 0 & 0 & 1/2 \end{bmatrix} + (1-\alpha) \begin{bmatrix} 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \\ 1/5 & 1/5 & 1/5 & 1/5 & 1/5 \end{bmatrix}$$

$P'' = \alpha P' + (1-\alpha)uv^T$,
where u is the vector of all 1s, $u = (1,1,\ldots,1)$
and v is the uniform vector, $v = (1/n,1/n,\ldots,1/n)$

A PageRank implementation

- Performing vanilla power method is now too expensive – the matrix is not sparse

$q^0 = v$

$t = 1$

repeat

$q^t = (P'')^T q^{t-1}$

$\delta = \|q^t - q^{t-1}\|$

$t = t + 1$

until $\delta < \varepsilon$

Efficient computation of $q^t = (P'')^T q^{t-1}$

$$y = \alpha P^T q^{t-1}$$

$$\beta = 1 - \|y\|_1$$

$$q^t = y + \beta v$$

P = normalized adjacency matrix

$P' = P + dv^T$, where d_i is 1 if i is sink and 0 o.w.

$P'' = \alpha P' + (1-\alpha)uv^T$, where u is the vector of all 1s

A PageRank implementation

$$y = \alpha P^T q^{t-1}$$
$$\beta = 1 - \|y\|_1$$
$$q^t = y + \beta v$$

- For every node i:

$$y_i = \alpha \sum_{j:j \to i} \frac{q_j^{t-1}}{Out(j)}$$
$$\beta = 1 - \sum_i y_i$$
$$q_i^t = y_i + \beta \frac{1}{n}$$

Why does this work?

$$\sum_i y_i = \alpha \left(1 - \sum_i \sum_{j:j\ is\ a\ sink} \frac{q_j^{t-1}}{n} \right) = \alpha - \alpha \sum_{j:j\ is\ a\ sink} q_j^{t-1}$$

$$\beta = \alpha \sum_{j:j\ is\ a\ sink} q_j^{t-1} + (1 - \alpha)$$

$$q_i^t = \alpha \sum_{j:j \to i} \frac{q_j^{t-1}}{Out(j)} + \alpha \sum_{j:j\ is\ a\ sink} \frac{q_j^{t-1}}{n} + (1 - \alpha) \frac{1}{n}$$

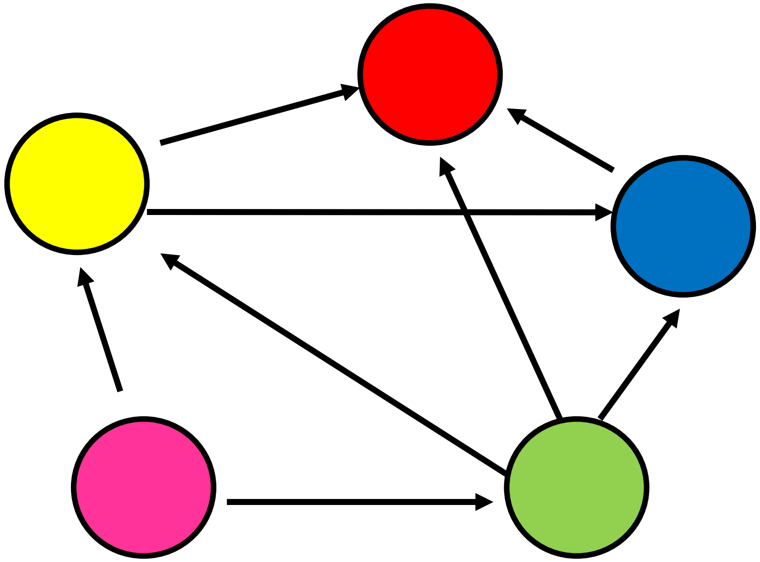
Implementation details

- If you use Matlab, you can use the matrix-vector operations directly.
- If you want to implement this at large scale
  - Store the graph as an adjacency list
  - Or, store the graph as a set of edges,
  - You need the out-degree Out(v) of each vertex v
  - For each edge $u \to v$add weight $\frac{q_u^{t-1}}{Out(u)}$ to the weight q_v^t
  - This way we compute vector y, andthen we can compute q^t

ABSORBING RANDOM WALKS

Random walk with absorbing nodes

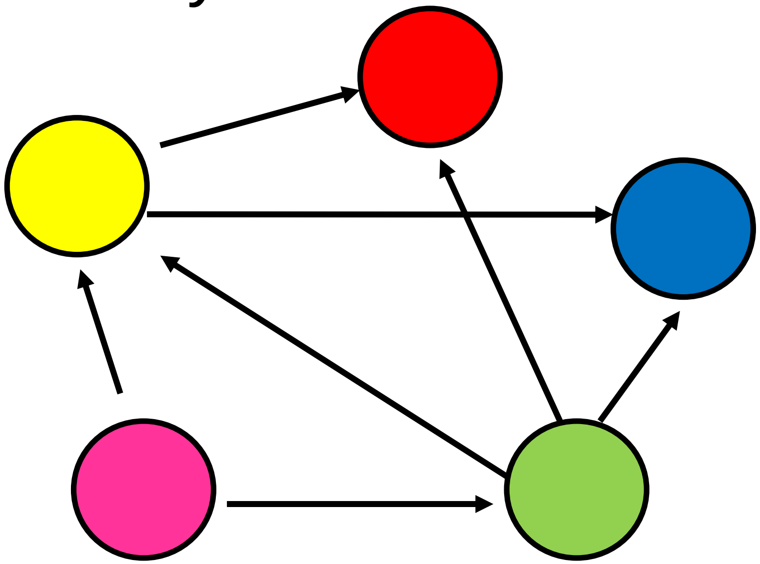
- What happens if we do a random walk on this graph? What is the stationary distribution?



- All the probability mass on the red sink node:
  - The red node is an absorbing node

Random walk with absorbing nodes

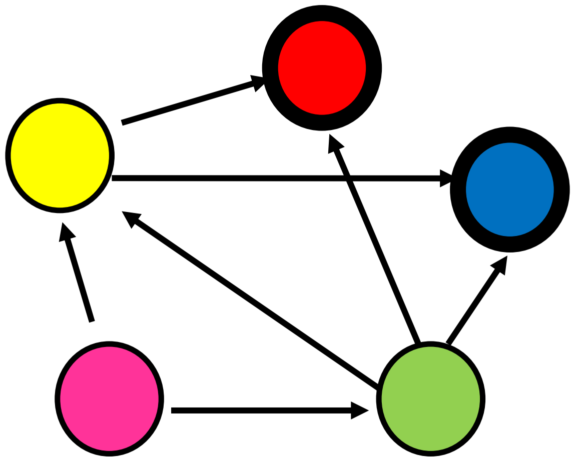
- What happens if we do a random walk on this graph? What is the stationary distribution?



- There are two absorbing nodes: the red and the blue.
- The probability mass will be divided between the two

Absorption probability

- If there are more than one absorbing nodes in the graph a random walk that starts from a non-absorbing node will be absorbed in one of them with some probability
  - The probability of absorption gives an estimate of how close the node is to red or blue



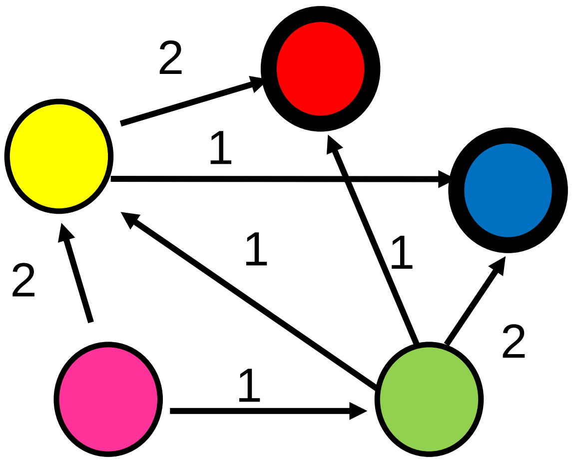
Absorption probability

- Computing the probability of being absorbed is very easy
  - Take the (weighted) average of the absorption probabilities of your neighbors
    - if one of the neighbors is the absorbing node, it has probability 1
  - Repeat until convergence (very small change in probs)
  - The absorbing nodes have probability 1 of being absorbed in themselves and zero of being absorbed in another node.

$$P(Red|Pink) = \frac{2}{3}P(Red|Yellow) + \frac{1}{3}P(Red|Green)$$

$$P(Red|Green) = \frac{1}{4}P(Red|Yellow) + \frac{1}{4}$$

$$P(Red|Yellow) = \frac{2}{3}$$



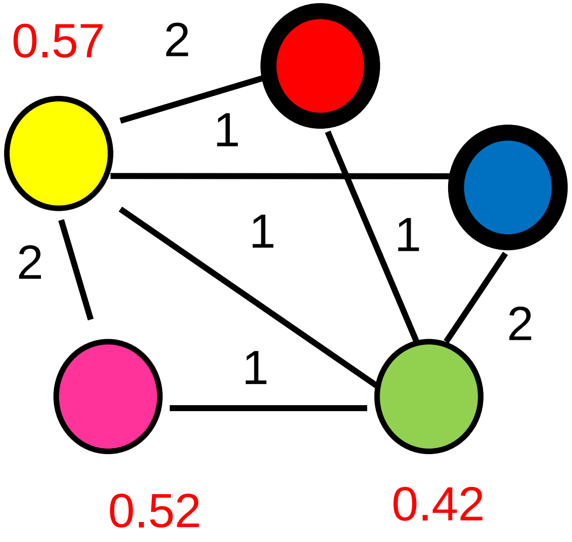
Absorption probability

- The same idea can be applied to the case of undirected graphs
  - The absorbing nodes are still absorbing, so the edges to them are (implicitly) directed.

$$P(Red|Pink) = \frac{2}{3}P(Red|Yellow) + \frac{1}{3}P(Red|Green)$$

$$P(Red|Green) = \frac{1}{5}P(Red|Yellow) + \frac{1}{5}P(Red|Pink) + \frac{1}{5}$$

$$P(Red|Yellow) = \frac{1}{6}P(Red|Green) + \frac{1}{3}P(Red|Pink) + \frac{1}{3}$$



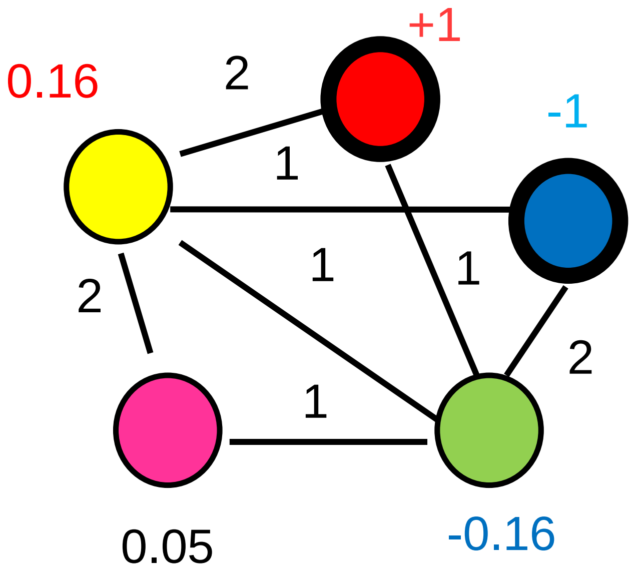
Propagating values

- Assume that Red has a positive value and Blue a negative value
  - Positive/Negative class, Positive/Negative opinion
- We can compute a value for all the other nodes in the same way
  - This is the expected value for the node

$$V(Pink) = \frac{2}{3}V(Yellow) + \frac{1}{3}V(Green)$$

$$V(Green) = \frac{1}{5}V(Yellow) + \frac{1}{5}V(Pink) + \frac{1}{5} - \frac{2}{5}$$

$$V(Yellow) = \frac{1}{6}V(Green) + \frac{1}{3}V(Pink) + \frac{1}{3} - \frac{1}{6}$$



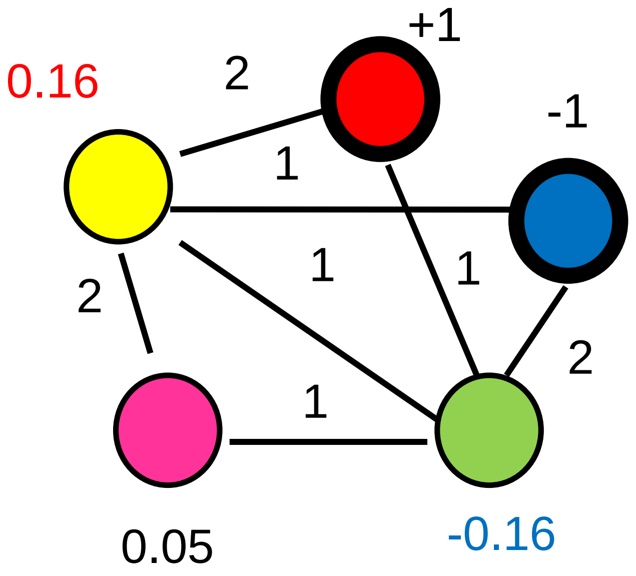
Electrical networks and random walks

- Our graph corresponds to an electrical network
- There is a positive voltage of +1 at the Red node, and a negative voltage -1 at the Blue node
- There are resistances on the edges inversely proportional to the weights (or conductance proportional to the weights)
- The computed values are the voltages at the nodes

$$V(Pink) = \frac{2}{3}V(Yellow) + \frac{1}{3}V(Green)$$

$$V(Green) = \frac{1}{5}V(Yellow) + \frac{1}{5}V(Pink) + \frac{1}{5} - \frac{2}{5}$$

$$V(Yellow) = \frac{1}{6}V(Green) + \frac{1}{3}V(Pink) + \frac{1}{3} - \frac{1}{6}$$



Transductive learning

- If we have a graph of relationships and some labels on these edges we can propagate them to the remaining nodes
  - E.g., a social network where some people are tagged as spammers
  - E.g., the movie-actor graph where some movies are tagged as action or comedy.
- This is a form of semi-supervised learning
  - We make use of the unlabeled data, and the relationships
- It is also called transductive learning because it does not produce a model, but just labels the unlabeled data that is at hand.
  - Contrast to inductive learning that learns a model and can label any new example

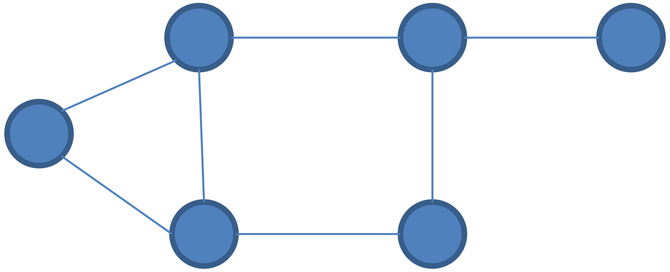
Implementation details

- Implementation is in many ways similar to the PageRank implementation
  - For an edge (u, v) instead of updating the value of v we update the value of u.
    - The value of a node is the average of its neighbors
  - We need to check for the case that a node u is absorbing, in which case the value of the node is not updated.
  - Repeat the updates until the change in values is very small.

COVERAGE

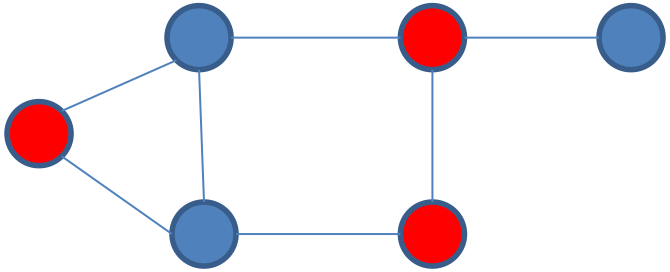
Example

- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has bought it.
  - We want to offer the product for free to some people such that every person in the graph is covered (they have a friend who has the product).
  - We want the number of free products to be as small as possible



Example

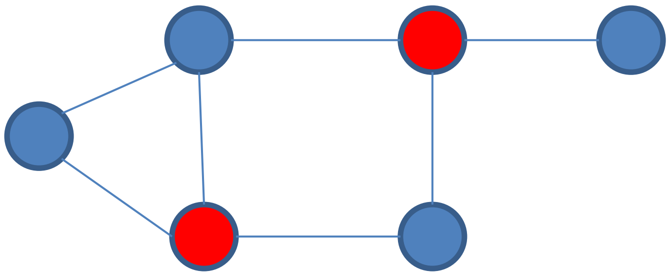
- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has bought it.
  - We want to offer the product for free to some people such that every person in the graph is covered (they have a friend who has the product).
  - We want the number of free products to be as small as possible



One possible selection

Example

- Promotion campaign on a social network
  - We have a social network as a graph.
  - People are more likely to buy a product if they have a friend who has bought it.
  - We want to offer the product for free to some people such that every person in the graph is covered (they have a friend who has the product).
  - We want the number of free products to be as small as possible



A better selection

Dominating set

- Our problem is an instance of the dominating set problem

- Dominating Set: Given a graph $G = (V, E)$, a set of vertices $D \subseteq V$ is a dominating set if for each node u in V, either u is in D, or u has a neighbor in D.

- The Dominating Set Problem: Given a graph $G = (V, E)$ find a dominating set of minimum size.

Set Cover

- The dominating set problem is a special case of the Set Cover problem

- The Set Cover problem:
  - We have a universe of elements $U = \{x_1, \dots, x_N\}$
  - We have a collection of subsets of U, $\boldsymbol{S} = \{S_1, \dots, S_n\}$, such that $\bigcup_i S_i = U$
  - We want to find the smallest subcollection $\boldsymbol{C} \subseteq \boldsymbol{S}$ of $\boldsymbol{S}$, such that $\bigcup_{S_i \in \boldsymbol{C}} S_i = U$
    - The sets in **C** cover the elements of U

Applications

- Dominating Set (or Promotion Campaign) as Set Cover:
  - The universe *U* is the set of nodes *V*
  - Each node u defines a set S_u consisting of the node u and all of its neighbors
  - We want the minimum number of sets S_u (nodes) that cover all the nodes in the graph.
- Document summarization
  - We have a document that consists of a set of terms *T* (the universe *U* of elements), and a set of sentenses***S***, where each sentence is a set of terms.
  - Find the smallest number of sentences *C*, that cover all the terms in the document.
- Many more…

Best selection variant

- Suppose that we have a budget K of how big our set cover can be
  - We only have K products to give out for free.
  - We want to cover as many customers as possible.

- Maximum-Coverage Problem: Given a universe of elements U, a collection of $\boldsymbol{S}$ of subsets of U, and a budget K, find a sub-collection $\boldsymbol{C} \subseteq \boldsymbol{S}$, such that $\bigcup_{S_i \in \boldsymbol{C}} S_i$ is maximized.

Complexity

- Both the Set Cover and the Maximum Coverage problems are NP-complete
  - What does this mean?
  - Why do we care?

- There is no algorithm that can guarantee to find the best solution in polynomial time
  - Can we find an algorithm that can guarantee to find a solution that is close to the optimal?
  - Approximation Algorithms.

Approximation Algorithms

- Suppose you have an (combinatorial) optimization problem
  - E.g., find the minimum set cover
  - E.g., find the set that maximizes coverage
- If X is an instance of the problem, let OPT(X) be the value of the optimal solution, and ALG(X) the value of an algorithm ALG.
- ALG is a good approximation algorithm if the ratio of OPT and ALG is bounded.

Approximation Algorithms

- For a minimization problem, the algorithm ALG is an α-approximation algorithm, for $\alpha > 1$, if for all input instances X,

$$ALG(X) \leq \alpha OPT(X)$$

- For a maximization problem, the algorithm ALG is an α-approximation algorithm, for $\alpha > 1$, if for all input instances X,

$$ALG(X) \geq \alpha OPT(X)$$

- α is the approximation ratio of the algorithm

Approximation ratio

- For a minimization problem (resp. maximization), we want the approximation ratio α to be as small (resp. as big) as possible.
  - Best case: $\alpha = 1 + \epsilon$(resp. $\alpha = 1 - \epsilon$) and $\epsilon \to 0$, as $n \to \infty$(e.g., $\epsilon = \frac{1}{n}$)
  - Good case: $\alpha = O(1)$is a constant
  - OK case: $\alpha = O(\log n)$ (resp. $\alpha = O\left(\frac{1}{\log n}\right)$)
  - Bad case $\alpha = O(n^{\epsilon})$ (resp. $\alpha = O(n^{-\epsilon})$)

A simple approximation ratio for set cover

- Any algorithm for set cover has approximation ratio $\alpha = |S_{max}|$, where S_{max} is the set in **S** with the largest cardinality

- Proof:
  - $OPT(X) \geq N/|S_{max}| \Rightarrow N \leq |s_{max}|OPT(I)$
  - $ALG(X) \leq N \leq |s_{max}|OPT(X)$

- This is true for any algorithm.
- Not a good bound since it can be that $|S_{max}|=O(N)$

An algorithm for Set Cover

- What is the most natural algorithm for Set Cover?

- Greedy: each time add to the collection ***C*** the set S_i from ***S*** that covers the most of the remaining elements.

The GREEDY algorithm

GREEDY(U,S)

X= U

C = {}

while X is not empty do

For all $S_i \in \boldsymbol{S}$ let $\mathrm{gain}(S_i) = S_i \cap X$

Let S_* be such that $gain(S_*)$ is *maximal*

C = C U {S_*}

X = X\ S_*

S = ***S***\ S_*

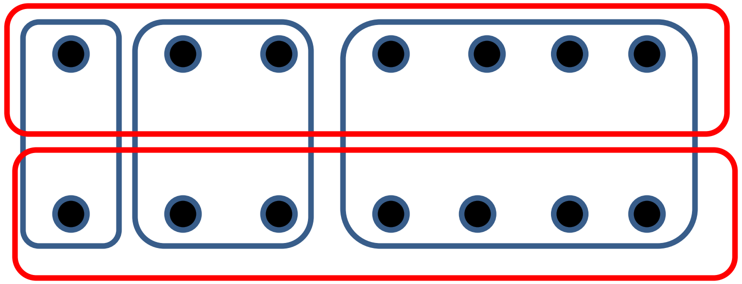
Approximation ratio of GREEDY

- Good news: the approximation ratio of GREEDY is

$$\alpha = H(|S_{\max}|) = 1 + \ln|S_{\max}|, \qquad H(n) = \sum_{k=1}^{n} \frac{1}{k}$$

$$GREEDY(X) \leq (1 + \ln|S_{\max}|)OPT(X),$$ for all X

- The approximation ratio is tight up to a constant (we can find a counter example)



OPT(X) = 2
GREEDY(X) = logN
α=½logN

Maximum Coverage

- What is a reasonable algorithm?

GREEDY(U,S,K)

X = U

C = {}

while |***C***| < K

For all $S_i \in \boldsymbol{S}$ let $\text{gain}(S_i) = S_i \cap X$

Let S_* be such that $gain(S_*)$ is *maximal*

C = C U {S_*}

X = X\ S_*

S= ***S***\ S_*

Approximation Ratio for Max-K Coverage

- Better news! The GREEDY algorithm has approximation ratio $\alpha = 1 - \frac{1}{e}$

$$GREEDY(X) \geq \left(1 - \frac{1}{e}\right) OPT(X),$$ for all X

Proof of approximation ratio

- For a collection C, let $F(C) = \bigcup_{S_i \in \boldsymbol{C}} S_i$ be the number of elements that are covered.
- The function F has two properties:

- F is monotone:

$$F(A) \leq F(B)\ if\ A \subseteq B$$

- F is submodular:

$$F(A \cup \{S\}) - F(A) \geq F(B \cup \{S\}) - F(B)\ \ if\ A \subseteq B$$

- Diminishing returns property

Optimizing submodular functions

- Theorem: A greedy algorithm that optimizes a monotone and submodularfunction F, each time adding to the solution C, the set S that maximizes the gain $F(C \cup \{S\}) - F(C)$has approximation ratio $\alpha = \left(1 - \frac{1}{e}\right)$

Other variants of Set Cover

- Hitting Set: select a set of elements so that you hit all the sets (the same as the set cover, reversing the roles)
- Vertex Cover: Select a subset of vertices such that you cover all edges (an endpoint of each edge is in the set)
  - There is a 2-approximation algorithm
- Edge Cover: Select a set of edges that cover all vertices (there is one edge that has endpoint the vertex)
  - There is a polynomial algorithm

Parting thoughts

- In this class you saw a set of tools for analyzing data
    - Association Rules
    - Sketching
    - Clustering
    - Classification
    - Signular Value Decomposition
    - Random Walks
    - Coverage
- All these are useful when trying to make sense of the data. A lot more variants exist.