[1. Distributed file systems: 2](#_Toc36044089)

[2. Association Rules + Frequent Itemsets 3](#_Toc36044090)

[3. Finding similar items: Locality Sensitive Hashing (LSH) 4](#_Toc36044091)

[5. Clustering 6](#_Toc36044092)

[6. Dimensionality Reduction 7](#_Toc36044093)

[7. Recommender Systems 1 9](#_Toc36044094)

[8. Recommender Systems 2: Latent Factor 9](#_Toc36044095)

[9. Pagerank 10](#_Toc36044096)

[10. Link Analysis 10](#_Toc36044097)

[11. Community detection in Graphs 11](#_Toc36044098)

[12. Graph Representation Learning 11](#_Toc36044099)

[13. Decision Trees 12](#_Toc36044100)

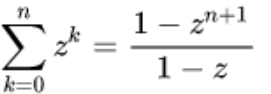
[14. Support Vector Machine 13](#_Toc36044101)

[15. Mining Data Streams 1 14](#_Toc36044102)

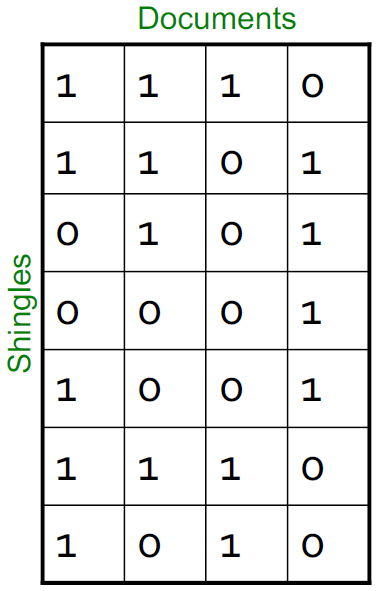
[16. Mining Data Streams 2: 14](#_Toc36044103)

[17. Advertising 16](#_Toc36044104)

[18. Bandits 17](#_Toc36044105)

**Geometric series**

1. Distributed file systems: [01-intro](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\01-intro.pdf)
2. MapReduce: p.37
   1. Failures p.44
   2. Problems suitable for MapReduce: p.58, 59
   3. Join by Map-Reduce: p.60, 61
   4. Problems not suitable for MapReduce: p.62
   5. Cost of running MapReduce: p.64+66
3. RDD: p.50
   1. Operations: p.51
4. PySpark DataFrame: p.53
5. Association Rules + Frequent Itemsets: [02-assocrules.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\02-assocrules.pdf)
6. Examples of items & baskets: p.5+7
7. **Support** of itemset, association rule = number of times it appears in baskets
8. Items appearing in at least “s (**support threshold**)” baskets: **frequent** itemsets (p.9)
9. **Association** rule: {i1, i2, …, ik} -> j (p.11)
10. **Confidence** of association rule: conf(I🡪j) = support(I U j) / support(I) (p.11)
11. **Interest** of association rule I🡪j: 𝐼𝑛𝑡𝑒𝑟𝑒𝑠𝑡(I→𝑗) = |𝑐𝑜𝑛𝑓(𝐼→𝑗) − Pr[𝑗]| (interesting rules: have Interest > 0.5) (p.12)
12. **Maximal frequent itemsets**: immediate supersets are not frequent (p.17, example: p.18)
13. **Closed itemsets**: immediate supersets have smaller supports (p.17, example: p.18)
14. We measure the cost by the **number of passes** an algorithm makes over the data (p.21)
15. **Counting pairs** in memory: p.26 + p.28
16. **Counting pairs with A-Priori** alg: p.32
    1. Extension to frequent k-itemsets: p.35 🡪 example: p.36
17. **PCY** algorithm:
    1. 1st pass: p.40 (top of the page);
    2. Transition: p.42;
    3. 2nd pass: p.43
18. Finding similar items: Locality Sensitive Hashing (LSH): [03-lsh.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\03-lsh.pdf)
    1. Naïve approach to finding all pairs of data (xi, xj) with distance <= s: O(N2) (p.8)
    2. **Min-Hash / LSH** steps: p.13+**52** [**downside**: can produce false negatives: pairs of similar items not detected]
       1. **Shingling**: convert document to set of k-shingles then hash the shingles to 4-byte integers (p.16)
          1. **Jaccard** similarity (p.18) for Ci = S(Di):  
             sim(D1, D2) = |C1 C2|/|C1 C2| [numerator: dot product of two columns in matrix below]

 (p.19)

* + - 1. Jaccard distance: d(C1, C2) = 1 - |C1C2|/|C1C2|
    1. **Min-Hashing (suitable for Jaccard similarity):** h(C) s.t. sim(C1, C2) is high 🡪 P(h(C1) = h(C2)) is high + vice versa (p.22) 🡪 **P(h(C1)=h(C2))=sim(C1,C2)**
       1. Minhash function for permutation : (C) = [first 1 in permuted column C] 🡪 example: p.25
       2. **Pr[*h*(C1) = *h*(C2)] = *sim*(C1, C2)** [proof: p.27]
          1. One-pass implementation: p.31
          2. Universal hashing:

ha,b(x) = ((a.x+b) mod p) mod N [a,b: rnd int; p: prime > N]

* + 1. **Locality Sensitive Hashing:** (p.38)
       1. Divide signature matrix M into b bands of r rows [M = b r]
       2. Hash all bands to a hash table with k buckets (large k)
       3. Candidate pairs: hash to same bucket for >= 1 bands
       4. Tune b&r to catch most similar pairs but few non-similar pairs
       5. [assuming same bucket == identical bands]
       6. Example calculation of false negatives ((1-Sr)b): p.41 + 47
       7. Example calculation of false positives 1-((1-Sr)b): p.42 + 47
       8. **S curve** (probability of sharing a bucket vs similarity of two sets)  
          (1- (1-tr)b**)**: p.48 + 49 (example for b = 20, r = 5)

1. LSH Theory: [04-lsh\_theory.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\04-lsh_theory.pdf)
   * + 1. Probability that C1 and C­2 are candidate pairs: 1 – (1-Sr)b (P.10)
       2. **Distance measures: Jaccard, Cosine, L1 (Manhattan), L2 (Euclidean)** (p.16)
          1. Cosine distance (p.43): d(A,B) = [range: 0 to ]
          2. Cosine similarity (p.43): 1 – d(A,B) 🡪 or 🡪 cos() =
       3. **Locality-Sensitive (LS) Families**: family H of hash functs (d1, d2, p1, p2)-sensitive [d(x,y) <= d1 🡪 Pr(h(x)=h(y)) >= p1] [d(x,y) >= d2 🡪 Pr(h(x)=h(y)) <= p2] (p.18)
          1. Example for min-hashing with Jaccard distance (d):  
             Pr[h(x) = h(y)] = 1 – d(x,y) for (p.20)
          2. **Min-Hashing for Jaccard similarity: (d1, d2, (1-d1), (1-d2))-sensitive family for any d1 < d2** (p.21)
          3. **LSH for Cosine Distance:** *(d1, d2, (1-d1/*𝝅*), (1-d2/*𝝅*))-sensitive*

Random hyperplanes: p.45

* + - * 1. LSH for Euclidean distance: p.50, 55,
        2. **AND-Construction**: e.g. rows in a band (p.22)

H(x) = h(y) if and only if all hi(x) = hi(y)  
🡪 (d1, d2, p1^r, p2^r) (p.24)

* + - * 1. **OR-Construction**: e.g. many bands (p.22)

H(x) = h(y) if and only if hi(x) = hi(y) for at least 1 i  
🡪 (d1, d2, 1-(1-p1)^b, 1-(1-p2)^b) (p.26)

[Assumption: hi’s are independent]

* + - * 1. **Impacts of AND vs OR**: AND shrinks all probs.; OR grows all probs. (p.27) 🡪 Summary: p.39 **[ideal: get p1 to 1, p2 to 0]**

\*\* **False positive and false negative**: P.32 --> (increasing b: reduces false negative+increases false positives, increasing r: reduces false positives+increases false negatives [p.27])

* + - * 1. Composing constructions: AND 🡪 OR or vice versa (p.29)

**r-way AND followed by b-way OR** (p.29):  
Pr[min 1 shared bucket=candidate pair] = 1-(1-sr)b

Threshold t: 1-(1-tr)b = t  
we’re improving the sensitivity ala low prob less than t, high prob greater than t (p.37)

**b-way OR followed by r-way AND** (p.34):

Pr[min 1 shared bucket=candidate pair] = (1-(1-s)b)r

(3; 4; 5) way OR-AND-OR --> 1 - (1 - (1 - (1 - s)3)4)5 [p.31 exam 2019 solution]

**Example calculating number of hash functions** required: P.36

1. Clustering: [05-clustering.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\05-clustering.pdf)
   1. Representing documents (p.9):
      1. Vectors 🡪 cosine distance for similarity
      2. Sets 🡪 Jaccard distance
      3. Points 🡪 Euclidean distance
   2. Curse of dimensionality: must traverse (0.001)1/d of space to capture 0.1% of data (p.12)
   3. Clustering (p.14):
      1. **Agglomerative (bottom-up; hierarchical):** each point a cluster 🡪 combine nearest every time
         1. Good for non-convex shapes [p.17]
         2. Dendrogram: p.19
      2. **Divisive (top-down):** start one cluster, split gradually
      3. **Centroid**: avg of all data points in cluster VS **clustroid**: one point from cluster closest to all others (p.21)
      4. Merging clusters:
         1. Based on proximity of centroids: Good for convex clusters (p.26)
         2. Based on proximity of nearest points b/w two clusers: good for concentric clusters (p.27)
      5. **K-Means Clustering**: uses Euclidean distance/space [assumes clusters normally distributed in each dimension + axes are fixed, no tilted ellipses allowed!]
         1. K-means++: p.30
         2. Choosing right k: p.35
         3. **BFR Algorithm:** [assumes clusters normally distributed in each dimension + axes are fixed, no tilted ellipses allowed!]
            1. Overview: p.41

Step 1: p.42

Step 2: p.43 & 45

Step 3-4: p.47

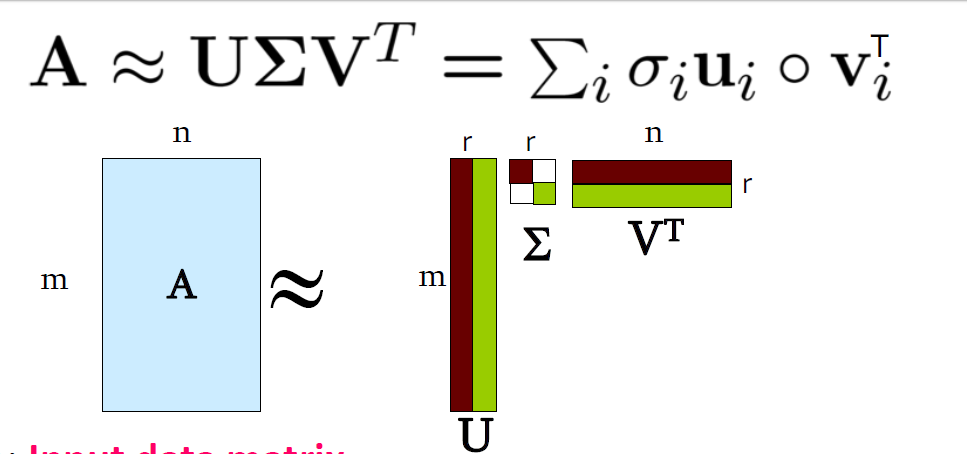
Step 5: p.48

* + - * 1. Three classes of points: p.43,44

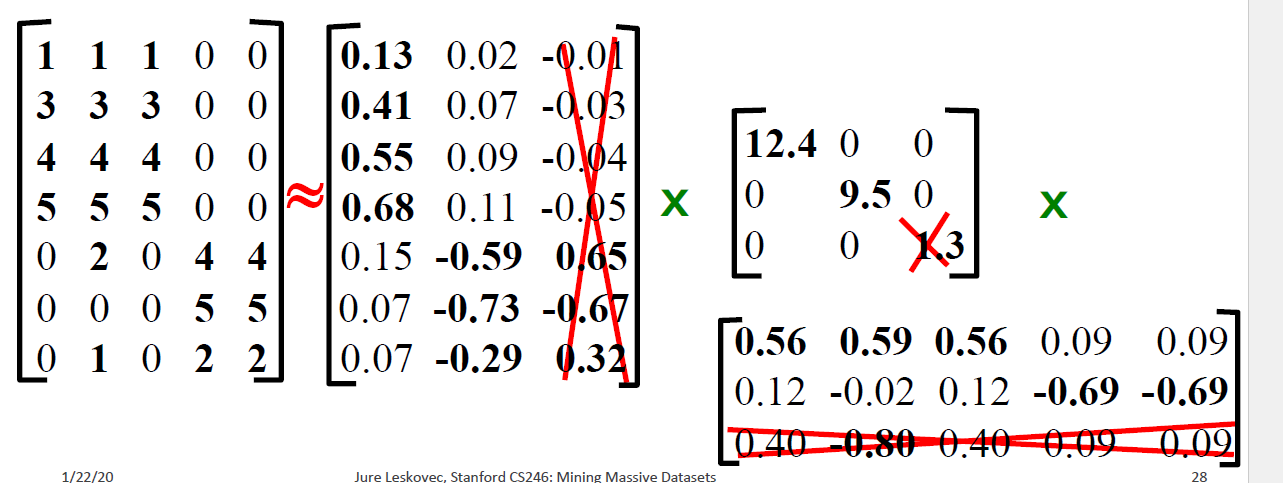
DS: p.45 & 46

* + - * 1. Proximity criteria for adding a point to a cluster: **Mahalanobis Distance** (p.52, 53)
        2. Merging criteria for two clusters: variance of combined below a threshold (p.55)
    1. Cure Algorithm: Euclidean distance + clusters of any shape:
       1. Step 1: p.59; Step 2: p.63

1. Dimensionality Reduction [06-dim\_red.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\06-dim_red.pdf)
   1. First dimension: direction with greatest variance; second: second largest (p.5)
   2. Rank of matrix A: Number of linearly independent rows of A (p.6)
   3. **SVD**: A (m docs x n terms, input data matrix) ~ U (m x r concepts [=latent dimensions, latent factors], left singular vectors, **user-to-concept factor matrix**) . (rxr, singular values, diagonal, **strength of each concept**, sorted in decreasing order) . VT (rxn, right singular vectors, **movie-to-concept factor matrix**) [p.10] [example: p.22]



* + 1. Real matrix A can be uniquely decomposed as A = U VT
       1. U, V: column orthonormal 🡪 UT U = I
    2. First right singular vector = first row of VT (p. 22)
    3. Variance (spread on the v1 axis (element 1,1 from . (p.23)
    4. gives coords of points along projection axis [first column: projection of users on a concept] (p.24)
    5. Dimensionality reduction: set (r-k) smallest singular value (in ) to zero (i.e. finding the *rank k* approximation of A) [pick r s.t. the retained singular values have at least 90% of the total energy, i.e. sum of their squares (p.33)]



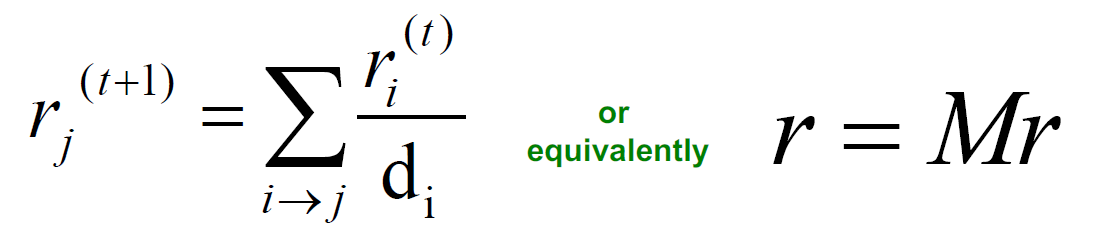


* + 1. **Reconstruction error**: Frobenius Norm (p.30+31)
    2. Computing SVD: finding principal eigenvector and eigenvalue (p.35-39)
       1. Steps: P.40
       2. Complexity: p.41
    3. QUERY using SVD: map query into concept space: q\*V (p.45,46,47[find user d’s similarity with query])
    4. Drawbacks of SVD: p.49
  1. **CUR**: p.56 + p.58 (pick 4k points for a rank-k approximation)
     1. SVD vs CUR: p. 60-61

1. Recommender Systems 1: [07-recsys1.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\07-recsys1.pdf)
   1. X = set of customers; S = set of items
   2. u: X x S 🡪 R (p.9)
      1. R = set of ratings (ordered set)
   3. **Content-based Recommendation:** Recommend items to customer x like previous items rated highly by x
      1. Creating item profiles for text mining: p.17-18
      2. Create user profiles: p.19
      3. Pros: p.20, cons: p.21
   4. **Collaborative Filtering**:
      1. **Cosine Similarity+Pearson correlation coefficient**: p.24
      2. **User-user collaborative filtering (prediction)**: p.26
      3. **Item-item collaborative filtering (prediction)**: p.27+example: p.28-32 🡪 OFTEN WORKS BETTER THAN USER-USER
         1. Common practice formula: p.33
         2. Pors/Cons: p.35 [no feature selection needed unlike user-user]
      4. Complexity: **finding k nearest customers**: p.42
   5. Evaluating predictions: p.40 [Root-mean-square error **RMSE**]
2. Recommender Systems 2: Latent Factor Models [08-recsys2.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\08-recsys2.pdf)
   1. Root mean square error (RMSE): p.5
   2. Estimation considering local and global effects (deviations): p.10 (definition) and p.15 (method)
   3. Using SVD for recommendation (needs fully defined R):
      1. Rating of user x for item i: A = R, Q = U, PT = VT 🡪 (p.25)
      2. SVD minimizes sum of squared error (=reconstruction error) 🡪  
         𝑹𝑴𝑺𝑬 = 𝟏/𝒄 \* (SVD also minimizes SSE: p.26)
   4. **Latent Factor Model**:
      1. Regularization of objective function (defined in p.29)+gradient descent: p.36
         1. Stochastic GD: p.39
      2. With biases: p.44 and p.45
3. Pagerank [09-pagerank.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\09-pagerank.pdf)
   1. Definition: page j, importance rj 🡪 n out-links 🡪 rj/n each  
      rj = (p.21)

r = stationary distribution of random walk (p.31) 🡪 it is unique and it exists (p.32)

* 1. **Stochastic adjacency matrix M**: (p.23) & r = M.r (p.23) 🡪 r = an eigenvector of M with eigenvalue = 1 (p.26)

 (p.34)

* + 1. Power iteration for finding r: p.27 🡪 example: p.29
  1. Problems with pagerank:  
     spider traps (absorb pagerank, pagerank not what we want) p.37 🡪 solution: teleport with prob 1- (p.39)   
     dead-ends (leak page rank, adjacency matrix not column-stochastic anymore [columns sum to zero]): p.37 🡪 solution: teleport with prob 1 (p.41)
     1. **Google’s** solution: p. 44 [+Google matrix 🡪 ***r***new = ***A* · *r***old] 🡪 0.8,0.9
        1. Full Algorithm: p.50 [more complete: in p.4 of [10-spam.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\10-spam.pdf)]
           1. Cost of power method: p.53
        2. Block-stripe update alg: p.56-57

1. Link Analysis [10-spam.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\10-spam.pdf)
   1. **Full pagerank algorithm**: p.4
   2. **Topic-specific (aka Personalized) pagerank**: p.9
   3. **SimRank** (fixed teleport set): p.17
   4. Pixie random walk algorithm (get top 1k pins with highest visit count): p.31
      1. Pros: p.34
   5. Comparison b/w all pageRank methods: p.35
   6. **TrustRank**:
      1. Term spam: add important words to your page (p.40) 🡪 Google’s solution: p.41 [what links to a page say about the page]
      2. Type of pages from spammer’s pov: p.48
         1. Pagerank of target page resulting from accessible and own pages: p.49
            1. Fighting spam farms: p.54, 55
            2. TrustRank == Pagerank with trusted pages as teleport set (p.55)

Justification: p.56

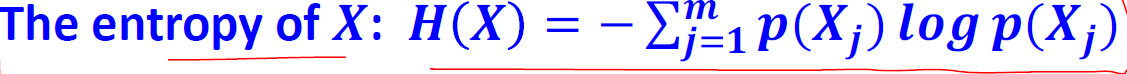
Picking seed set: p.58

Spam mass estimation: p.61

1. Community detection in Graphs [11-graphs1.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\11-graphs1.pdf)
   1. Finding densely linked clusters; using **Personalized PageRank** (PPR): p.10
   2. Cut of a cluster: p.13
   3. Graph partitioning criteria:
      1. **Conductance**: p.15
      2. **Algorithm**: p.17+18 (calculating using ) +
         1. Approximate PageRank (PageRank-Nibble):
            1. Undirected graph, lazy random walk: p.20
            2. Page rank vector: p.21
            3. ALGORITHM: p.25 🡪 example: p.26

Runtime + approximation guarantee: p.27

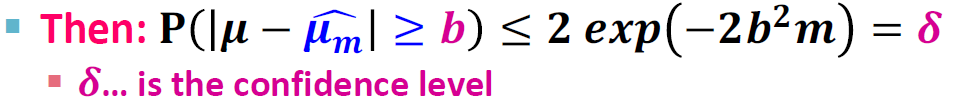
* 1. Motif-based clustering:
     1. **Conductance** for motifs: p.35
     2. Steps: p.36
  2. Network modularity:
     1. Measure of how well network is partitioned into communities [sets of tightly connected nodes] (p.40)
     2. Null model: configuration model 🡪 expected number of edges b/w nodes i and j of deg ki and kj in G’ (rewired network for G, with same degree distribution but random connections): ki kj / 2m (p.41)
     3. Sum of degrees of all nodes = 2\*m (number of edges)
     4. **Modularity of partitioning S of graph G**: p.42 🡪 p.43 for weighted graph
     5. Maximize Modularity 🡪 identify communities
  3. Louvain Algorithm for maximizing modularity: algorithm: p.51 [greedy strategy; greatly scalable]
     1. 1st phase: p.47
        1. Modularity gain: p.48
     2. 2nd phase: p.50

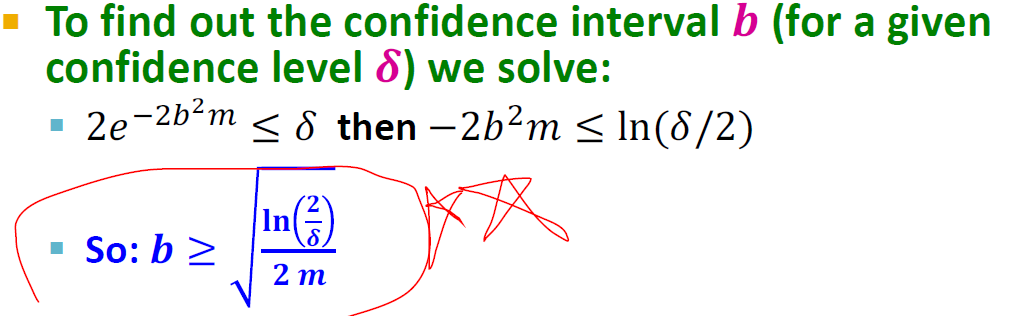
1. Graph Representation Learning [12-graphs2.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\12-graphs2.pdf)
   1. Encoder:
      1. Shallow encoding: p.17
      2. Similarity: random-walk embedding: p.22, 23
         1. Pros of random walk: 24
         2. Optimization objective: p.29
            1. Approximation: p.32-33
         3. ALGORITHM: p.34
   2. Generalized random walk (**node2vec**): biased random walk: p.39, 42,
      1. Algorithm: p.43 🡪 example: p.45
   3. Applications of embeddings [clustering/community detection, node classification, link prediction]: p.48
2. Decision Trees [13-dt.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\13-dt.pdf)
   1. Regression:
      1. Split criteria: **purity** (p.19)
      2. Prediction: p.30
      3. PLANET algorithm: settings: p.33
         1. Overview of steps: p.35 + p.53
         2. Master node: p.41
         3. MapReduce initialization: p.45-46
         4. MapReduce FindBestSplit: p.48,49, Map: p.50, reducer: p.51
   2. Classification:
      1. Split criteria: **information gain** (p.20)
         1. Entropy: p.21  
            
         2. Conditional entropy: p.25
         3. IG(Y|X) = H(Y) – H(Y|X) [p.26]
      2. Stopping criterion: p.29
      3. Prediction: p.30
   3. Ensembles, bagging, and RandomForest: p.58-61
3. Support Vector Machine (SVM) [14-svm.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\14-svm.pdf)
   1. **Margin** 𝜸: Distance of closest example from the decision line/hyperplane (basics: p.13; normalized: p.21)
   2. Distance from a point to the margin (: (p.16 )
   3. **Prediction** = sign(w.x+b); **confidence**: (w.x+b)y [p.17]
   4. OBJECTIVE: Maximizing the margin: basic: p.18 🡪 simplified for linearly separable data: p.23 + **p.30**
   5. Num of support vectors: d+1 (for d dimensional data) [p.20]
   6. **Regularization+slack variable ()**: p.25,**27**
   7. Finding w: Gradient descent: p.30 (cost function+gradient) + p.31(Gradient Descent)
      1. Stochastic Gradient Descent: p.32
   8. Multiclass SVM: p.40-42
4. Mining Data Streams 1: [15-streams1.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\15-streams1.pdf):
   1. Sampling data from a stream
      1. Random sample with fixed proportion
         1. Naïve approach: sample randomly (p.13) --> caveats: p.14
            1. Solution: sample users: p.16
      2. Random sample with fixed size
         1. Reservoir sampling: p.19
            1. Proof: p.20-21
   2. Queries over sliding windows
      1. Number of items of type x in the last k elements of the stream
         1. Assuming uniformity: p.27
         2. **DGIM**: O(log2N) storage+at most 50% error [proof: p.42] (p.28)
            1. Definition of bucket: p.34
            2. Rules of representing a stream by buckets: p.35+36(figure)
            3. **Updating buckets**: p.37+38 --> example: p.39
            4. Estimating number of 1s in N most recent bits: p.40
            5. Extensions

r or r-1 buckets: p.43

Stream of positive integers: p.45

1. Mining Data Streams 2: [16-streams2.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\16-streams2_READ%20AGAIN.pdf)
   1. **Filtering a data stream**
      1. Select elements with property x from the stream
         1. First cut solution: p.6
            1. Creates false positives but no false negatives: If the item is in S we surely output it, if not we may still output it
            2. Probability of m darts, n targets: a target getting hit by at least one dart: 1 – (1-1/n)m [1/n: probability of hitting a target with one dart] ~= 1-e-m/n = probability of false positives [p.10]
      2. **Bloom Filter**:
         1. Algorithm [false positive: (1-e-km/n)k: p.12-13 --> optimal k: n/m ln(2) [p.14]
         2. Guarantees no false negatives, use limited memory; 1 big B == k small Bs --> p.15
   2. Counting **distinct** elements **(Flajolet-Martin)**
      1. Number of distinct elements in the last k elements of the stream
         1. Hash N elements to at least log2N bits; 2maxar(a) with r(a) = position of first 1 counting from the right. [p.20] --> proof [2r will be around m]: p.23-24
            1. Debugging E[2r] --> : p.25
   3. Estimating **moments**
      1. Kth moment: p.28-29
      2. 2nd moment calculation: **AMS** method: p.31-32 --> proof: p.33-34
      3. **For all kth moments**: p.35
      4. Dealing w Endless streams: p.37 [reservoir sampling]
      5. Estimate avg./std dev of last k elements
   4. Counting Itemsets: [Finding frequent elements]
      1. Using DGIM: p.39
      2. Exponentially Decaying window: p.41-42; sum over all weights = 1/c [p.43] --> example: p.44
2. Advertising [17-advertising. pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\17-advertising_READ%20AGAIN%20THE%20BALANCE%20ALLOCATION%20AT%20THE%20END.pdf)
   1. Perfect matching: all vertices of the graph matched
   2. Maximum matching; matching with largest possible number of matches
   3. Competitive ratio: minall possible inputs I(|Mgreedy|/|Mopt|) (p.14)
   4. |*Mgreedy*|/|*Mopt*| >= ½ [proof: p.15-17]
   5. Cost per thousand impressions (CPM); click-through rates (CTR) [p.20]
   6. Goal of ads: max search engine’s revenues: expected revenue per click (Bid \* CTR)
   7. Simplified environment: p.33
   8. BALANCE Algorithm: p.35 (pick advertiser with largest unspent budget; ties broken alphabetically)
      1. Optimal exhausts all budgets of advertisers (p.37)
      2. Balance revenue: minimum for x = y = B/2 --> Min balance revenue: 3B/2 --> competitive ratio = 3/4
         1. Proof: pl.39-40
         2. Worst case competitive ratio: 0.63
            1. Proof: P.42-43, 45-46
            2. General case: p.48
3. Bandits (Learning through experimentation) [18-bandits.pdf](file:///D:\University\Stanf\Terms%20Letters%20and%20so%20on\Term%2012--Term%207%20PhD\CS%20246\LNs\18-bandits.pdf)
   1. K-armed (ad) bandit (query):
      1. win (reward = 1) with fixed (unknown) prob. [ad’s CTR: we want to estimate this]
      2. loss (reward = 0) with fixed (unknown) prob.
      3. all draws independent given
      4. Performance metric: regret: p.13
      5. Allocation strategy: finding out p.14
      6. **Epsilon-Greedy algorithm**: p.18
         1. **Confidence** **Intervals**: choose action with highest upper bound on confidence interval (p.22)
         2. **Hoeffding’s** inequality for upper bound on average deviation from expected value of (p.25-26)





* + 1. **Upper Confidence Sampling (UCB1 algorithm**): (p.27-28) --> performance: O(RT/T) <= k ln(T)/T [p.29]
    2. A/B Testing: p.35
       1. Thompson Sampling p.38-41 --> p.41: in general