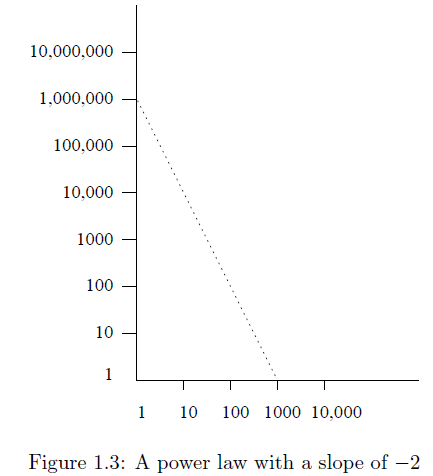
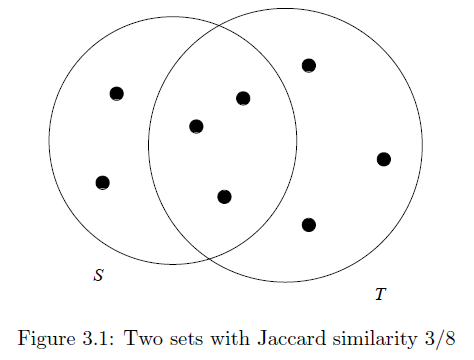
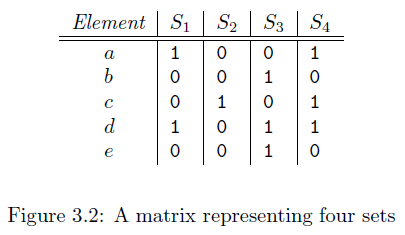
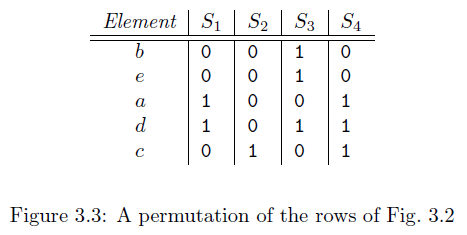
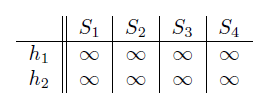
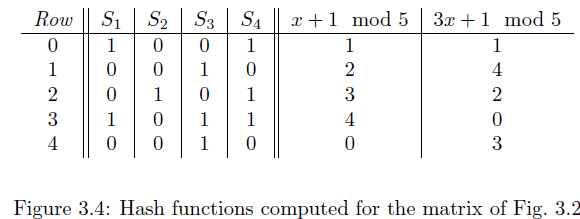
**Intelligent Data Management**

1. **Data Mining (Chapter1)**
   1. **What is data mining**

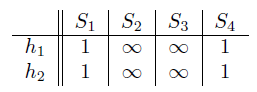
* **Data Mining**
  + Finding models for data.
  + Model either as a summary or as a set of most extreme features of the data
* **Statistical Modeling** 
  + underlying distribution, from which visible data is drawn
* **Machine Learning** 
  + especially useful if unclear what is searched for, i.e. movie recommendations
  + examples: Bayes nets, support-vector machines, decision trees, hidden Markov models...
* **Computational Approaches** 
  + data mining as algorithmic problem
  + model as answer to complex query
  + Summarization
    - represent / simplify data through a simpler form
    - e.g. "PageRank", clustering
  + Feature Extraction
    - find most extreme examples and use them to represent data
    - frequent item sets (hamburger and ketchup are frequently bought together) (see below)
    - similar items (recommendation based on customers with similar taste) (see below)
  1. **Statistical Limits on Data Mining**
     1. **Bonferroni’s principle**
* Not all assumptions that might make sense are statistically significant.
* Bonferroni correction: clear data to avoid 'bogus' data
* compare observations with expected number of positives (assuming randomness)
* only features that are very rare in random data can be assumed to be significant
* e.g. exercise with evil-doers
  1. **Things useful to know**
     1. **Importance of Words in Documents**
* **TF (Term-Frequency):** 
  + How often does a word occur in a document?
  + , with
  + Normalized by dividing it by the max number of occurrences of any term in document j
* **IDF (Inverse Document Frequency):** 
  + In how many documents does a word occur?
  + , with
* **TF.IDF (Term Frequency times Inverse Document Frequency):**
  + Term with highest TF.IDF score are terms that best characterize the topic of the document
  + What is the topic of a document? What words appear relatively rarely but if they do appear, they appear often within one document?
  + A term has high TF.IDF if it appears in relatively few documents, but appears in this one, and when it appears it appears in a document it tends to appear many times
    1. **Hash functions**
* map hash-keys (of any data type) to integer bucket numbers
* represent elements of interest as numbers (e.g. through ASCII-Code)
* create hash function that divides homogeneous data into equal "buckets"
* e.g. h(x) = x mod B
  + with B usually chosen as prime
  + consider that not all hash-keys (x) have B as factor
  + the amount of buckets shall not greater than the amount of hash-keys
    1. **Indexes**
* Index: Data structure that allows us to store and retrieve objects efficiently
* Object: Record with fields
* might be combined with hashing (hashing as a way to build indexes)
  + 1. **Secondary storage (not covered)**
* refers to secondary memory: disk
* relatively slow in comparison to primary memory: RAM
* big data requires "clever" algorithms that load data into RAM which is possibly needed soon.
  + 1. **The base of Natural logarithms**
* can be approximated by
* can be approximated by (with a being small)
* can be approximated by
  + 1. **Power Law**
* power laws describe linear relationships between logarithms of two variables x, y
* e.g.
  + (1,000,000 🡪 1,000) for and for
  + In general: and
* **Mathew Effect** :Strong features are likely to be strengthened further (“the rich get richer”)

1. **placeholder**
2. **Finding Similar Items** 
   1. **Applications of Near-Neighbor Search**
      1. **Jaccard Similarity of Sets**

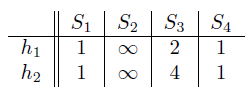
* (max: 1)
  + 1. **Jaccard Similarity of Bags**
* intersection size = minimum number of occurrences in both bags
* union size: sum of occurrences (intersections are counted twice)
* maximum similarity: 1/2
  + 1. **Similarity of Documents**
* only considering syntax level, not semantics
* used for finding (near) duplicates (e.g.)
* applications:
  + plagiarism (similarity between documents)
  + online shopping (similarity between customers)
  + movie ratings (similarity between movies)
    1. **Collaborative Filtering as a Similar-Sets Problem**
* recommend items based on similar user's preferences
* applications:
  + On-Line Purchases
    - Finding similar customers based on Jaccard similarity
    - due to large dimensionality, similarity need not be high to be significant
    - possible gain through additional clustering (e.g. sci-fi books)
  + Movie Ratings
    - Similar customers or movies
    - non-binary scale introduces complexity
    - e.g. star rating: put a (n\* rated) movie n times in a ‘bag’ and compute Jaccard similarity for bags
  1. **Shingling of Documents**
* simplest way to compare lexical similarity of textual documents
* represent documents as sets
  + 1. **K-Shingles**
* K-shingle of a document: any substring of length k found within the document
* using shingles on character level poses some difficulties (e.g. whitespaces), but has the advantage of ignoring possible typos
  + 1. **Choosing the Shingle Size**
* depends on size of documents and alphabet
* should be as small as possible to be little complex
* should be big enough so that the probability of any given shingle appearing in any given document is low
* consider average frequency of alphabet-parts
* rule of thumb for standard English: k=5
  + 1. **Hashing Shingles**
* Hash the string of shingle to an integer
* assumes that many shingles are unlikely (less buckets than possible shingles)
* reduces needed storage size
  + 1. **Shingles Build from Words**
* create shingle from stop words followed by the next x (e.g. 2) words (e.g. successful for finding similar news articles)
  1. **Similarity-Preserving Summaries of Sets**
* Applying Jaccard Similarity on shingles of documents might lead to too many large sets (4 times the space than the document alone)
* instead estimate similarity based on signatures
* 🡪 estimate Jaccard similarity by computing the fraction of rows in the signature matrix that agree
  + 1. **Matrix representation of Sets**
* **Characteristic Matrix**
  + columns ~ sets (e.g. documents)
  + rows ~ possible elements (e.g. shingles)
* sparse: many more 0s than 1s (usually stored in other format therefore)
  + 1. **Minhashing**
* **Calculate *Min-Hash*** 
  + permute rows randomly
  + minhash value of column is first row with element = 1
* **Example:** 
  + Permutation of rows: b,e,a,d,c
    1. **Minhashing and Jaccard Similarity**
* The probability that the minhash function for a random permutation of rows produces the same value for two sets equals the Jaccard similarity of those sets.
* Proof
  + 3 Types of rows: ***X*** if both 1, ***Y*** if one 1, ***Z*** if both 0
  + Most rows are of type *Z* (because sparse matrix)
  + ratio of type *X* to type *Y*
  + *x*: amount of type *X* rows; *y*: amount of type *Y* rows
  + Jaccard Similarity:
  + Consider:
  + Probability that we meet a *X* row before a type *Y* row (when going from the tp through permuted rows):
    1. **Minhash Signatures**
* **Minhash Signature** : vector of hash values for set S:
* **Minhash Matrix** : matrix of minhash signatures as columns (smaller than characteristic matrix, same rows but only n columns)
  + 1. **Computing Minhash Signatures**
* Not feasible/ time consuming to permute a large matrix 🡪 not implementable
* **Simulated Minhash**: compute minhash signatures by random hashes of row identifiers
  + Simulate permutation because k rows are mapped to at most k buckets
  + Permute row r to position h(r)
* **Computation:** 
  + 1) initialize:
  + 2) compute
  + 3) for each column c of do the following:
    - A) if c has 0 in row r, do nothing
    - B) if c has 1 in row r, then for each set to the smaller of the current value of and
  + 4) estimate Jaccard similarity by comparing columns
* **Example:** 
  + 1)



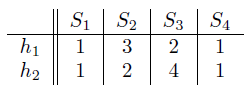
* + 2)
  + 3) row 0
    - A) for columns we do nothing because they have 0
    - B) rows 0 of have 1s so only these columns of signature matrix can change 🡪 since both column have ∞ they are changed to

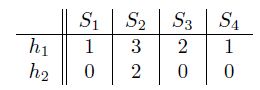


* + 3) row 1
    - B) only column can be changed 🡪 change and

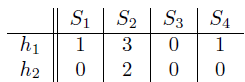


* + 3) row 2
    - B) only column can be changed; 🡪 values in are not changed because values [1,1]

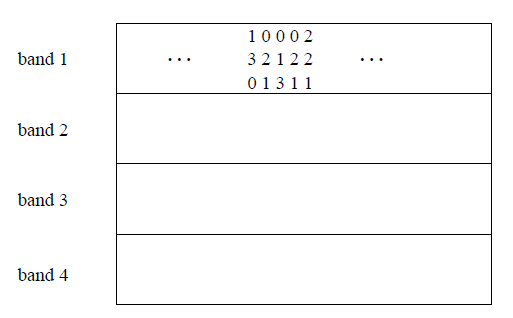
Are smaller than [3,2] 🡪 but replace column with [3,2]

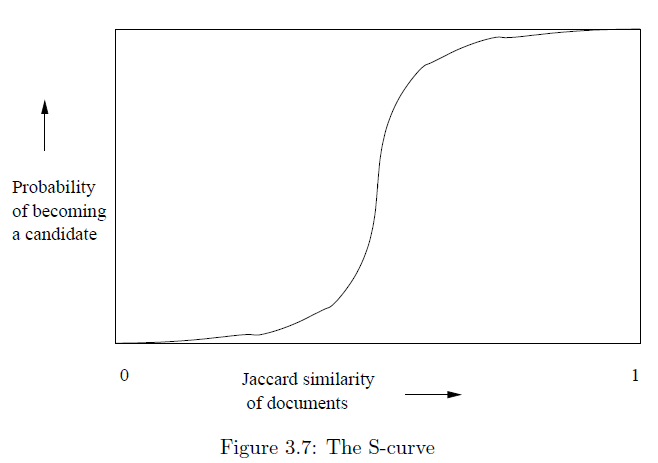


* + 3) row 3



* + 3) row 4
* 4) estimate Jaccard similarity:
  + 🡪 with more data more accurate estimated similarities
  1. **Locality-Sensitive Hashing for Documents**
* Minhashing for compressing large documents efficient
* But: find any pair of document with greatest similarity is still hard because too many possible pairs of documents
* find a way to filter pairs that are likely to be similar and compute similarity only for those
* **locality-sensitive hashing (LSH)** or **near-neighbor search**
  + 1. **LSH for Minhash Signatures**
* Hash items/ document that it is likely that similar items/documents are hashed in the same bucket
* Items in same bucket = **candidate pairs**
* Items in same bucket that are dissimilar = **false positives**
* Similar pairs that are not hashed to the same bucket by any of the hash functions = **false negatives**
* If we have the minhash signatures for the items, divide the characteristic matrix into b bands consisting of r rows each
  + For each band there is a hash function that takes the vector of r integers and hashes them to some bucket
* Example: a characteristic matric of 12 rows (and 5 columns shown) divided into bands with length 3 each
  + The more similar the columns are the more likely it is that the documents are hashed in the same bucket (best only if identical, as e.g. document 2 and 4)
  + Done for each band (e.g. with same hash function but different bucket array)



* + 1. **Analysis of the Banding Technique**
* B bands, r rows, a pair of documents has Jaccard similarity s
* The probability that the minhash signature for 2 documents agree in one particular row is s
* Probability that these 2 documents become a candidate pair is:
  + Prob. that the signatures agree in all rows of a band is:
  + Prob. that the signatures disagree in at least one row of a band:
  + Prob. that the signatures disagree in at least one row of each bands:
  + Prob. That signatures agree in all rows of at least one band

= prob. Of becoming a candidate pair:

* This function has an s-curve (regardless of b and r)
  + 1. **Combining the Techniques**
* Procedure to find most similar document pairs for several documents
* 1) pick k and compute k-shingles for all documents
* 2) sort document-shingle pairs and order them by shingle (characteristic matrix)
* 3) pick n, compute for each row/ shingle and compute signature matrix/ minhash signature for each document
* 4) choose threshold t that defines how similar documents have to be to become a candidate pair: pick so that and
  + If avoidance of false negatives: b and r should produce
  + If speed and avoidance of false positives:
* 5) Construct candidate pairs by using banding technique
* 6) examine candidates signatures and examine if the estimated similarity is > t
* 7) optionally check true document similarity of those whose
  1. **Distance Measures**
  2. **The Theory of Locality-Sensitive Functions**
  3. **LSH Families for Other Distance Measures**
  4. **Applications of Locality-Sensitive Hashing**
  5. **Methods for High Degrees of Similarity**

1. **Placeholder**
2. **Placeholder**
3. **Placeholder**
4. **Clustering** 
   1. **Introduction to Clustering Techniques**
   2. **Hierarchical Clustering**
   3. **K-Means Algorithms**
   4. **The CURE Algorithm**
   5. **Clustering in Non-Euclidean Spaces**
5. **Placeholder**
6. **Recommendation Systems**
   1. **A Model for Recommendation Systems**
   2. **Content-based Recommendations**
   3. **Collaborative Filtering**
   4. **Dimensionality Reduction**
   5. **The Netflix Challenge**
7. **Placeholder**
8. **Dimensionality Reduction**
   1. **Eigenvalues and Eigenvectors of Symmetric Matrices**
   2. **Principal Component Analysis (covered??)**
   3. **Singular-Value Decomposition**
   4. **CUR Decomposition**
9. **Large-Scale Machine Learning**
   1. **The Machine-Learning Model**
   2. **Perceptrons**
   3. **Support-Vector Machines**
   4. **Learning from Nearest Neighbors**

**Distance Measures**

* Jaccard similarity 0..1 --> Jaccard distance 0..1, = 1 - Jaccard similarity
* Defined as a set of points (*space*), with e.g. points x and y
* function
* Distance axioms
  + d(x,y) >= 0 (non-negativity)
  + d(x,y) = 0 <--> x = y (identity)
  + d(x,y) = d(y,x) (symmetry)
  + d(x,y) <= d(x,z) + d(z,y) (triangle-inequality)

**Euclidean Distance**

* L2-Norm
  + sqrt(sum(squared distance in each dimension))
  + shortest line between two points
* LR-Norm
  + like L2 norm but with exponent and root degree r instead of 2
* Manhattan Distance
  + = L1 Norm
  + sum(distance in each dimension)
* L-infinity Distance
  + with r getting larger, only the biggest difference matters
  + max(distance in each dimension)

**Cosine Distance**

* angle between two vectors (regardless of length) (between 0 and 180°)
* calculate through `=arc cos(x \* y / (|x| \* |y|))

**Edit Distance**

* only for strings
* smallest number of insertions and deletions that convert x to y

**Hamming Distance**

* = number of components that differ

**Theory of Locality - sensitive functions**

* function families like Jaccard Similarity have following properties:
  + close pairs more likely than distant pairs
  + statistically independent
  + efficient (better than manual checking)
* locality sensitive function: f(x) = f(y) <--> make x,y a candidate pair
* sensitive if for any function in function family
  + d(x,y) <= d\_1 --> p(f(x)=f(y)) >= p\_1
  + d(x,y) >= d\_2 --> p(f(x)=f(y)) >= p\_2
* combine multiple functions within family to *tune* S-curve
* is sensitive if p\_1 := 1 - d\_1 && p\_2 := 1 - d\_2

**Amplification of locality-sensitive families**

* given a sensitive family F, form new F' by the **AND**-construct so that for any function in F' such that it gives an equal result for x, y if and only if for any function in F, this is also the case
  + p\_1 := p\_1^r, where r is the number of rows in banding technique
  + similar with p\_2
* same with **OR**-construct if at least one function in F gives the same value
  + 1 - (1 - p\_1)^b, where b is the number of functions in F
* if cascading **AND** with **OR** for r = b = 4, false positives and false negatives are reduced
* if cascading **OR** with **AND** for r = b = 4, false positives increase

**LSH families for other distance measures**

**Hamming Distance**

* defined as number of component-wise-differences
* Family of functions is defined as creating one function for each component and f\_i(x) = f\_i(y) iff x,y agree in component i

**TODO**: integrate Summary of Chapter 3