CS246 Finals Review

The CS 246 Course Staff

MapReduce (Hadoop)

Programming model designed for:

- Large Datasets (HDFS)
  - Large files broken into chunks
  - Chunks are replicated on different nodes
- Easy Parallelization
  - Takes care of scheduling
- Fault Tolerance
  - Monitors and re-executes failed tasks

Dataflow

MapReduce operates exclusively on <key, value> pairs

Steps:

- Map:
  - Map function be applied independently to each unit of input
- Shuffle:
  - Redistributes data by output key of mappers
- Reduce:
  - Operates on full set of values for each key and produces a single output

Final output is the union of all the reducers
Multiple MapReduce jobs can be chained together
Degree of parallelism determined by # Mapper tasks and Reducer tasks

Coping with Failure

MapReduce is designed to deal with compute nodes failing

Output from previous phases is stored. Re-execute failed tasks, not whole jobs.

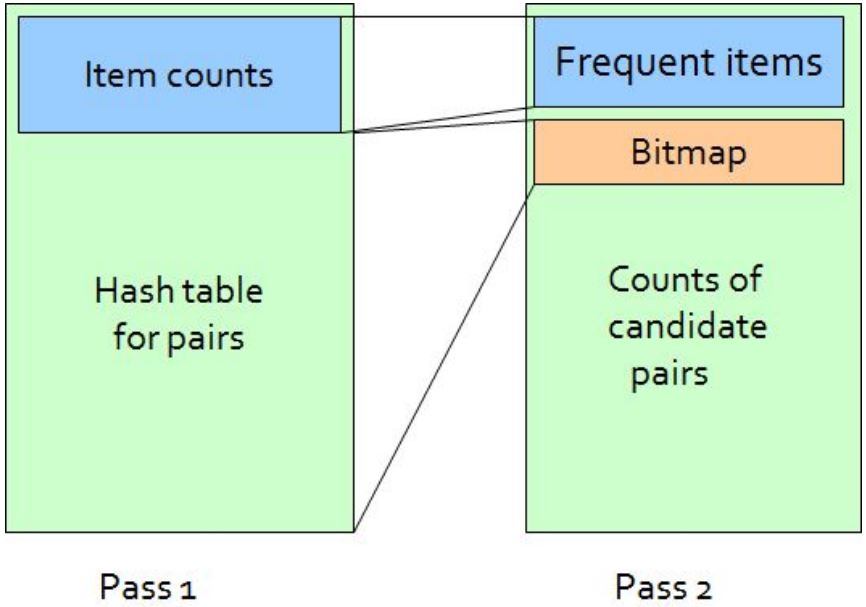
Blocking Property: no output is used until the task is complete. Thus, we can restart a Map task that failed without fear that a Reduce task has already used some output of the failed Map task.

Frequent Itemsets

- The Market-Basket Model
  - Items
  - Baskets
  - Count how many baskets contain an itemset
  - Support threshold => frequent itemsets
- Application
  - Confidence
    - Pr(D | A, B, C)

Computation Model

- Count frequent pairs
- Main memory is the bottleneck
- How to store pair counts?
  - Triangular matrix/Table
- Frequent pairs -> frequent items
- A-Priori Algorithm
  - Pass 1 - Item counts
  - Pass 2 - Frequent items + pair counts
- PCY
  - Pass 1 - Hash pairs into buckets
    - Infrequent bucket -> infrequent pairs
  - Pass 2 - Bitmap for buckets
    - Count pairs w/ frequent items and frequent bucket



All (Or Most) Frequent Itemsets

- Handle Large Datasets
- Simple Algorithm
  - Sample from all baskets
  - Run A-Priori/PCY in main memory with lower threshold
  - No guarantee
- SON Algorithm
  - Partition baskets into subsets
  - Frequent in the whole => frequent in at least one subset
- Toivonen’s Algorithm
  - Negative Border - not frequent in the sample but all immediate subsets are
  - Pass 2 - Count frequent itemsets and sets in their negative border
  - What guarantee?

Locality-Sensitive Hashing

Main idea:

- What: hashing techniques to map similar items to the same bucket
- Applications: similar documents, entity resolution, etc.

For the similar document application, the main steps are:

1. Shingling - converting documents to set representations
2. Minhashing - converting sets to short signatures using random permutations
3. Locality-sensitive hashing - applying the “b bands of r rows” technique on the signature matrix to an “s-shaped” curve

Locality-Sensitive Hashing

General Theory:

- Distance measures d (similar items are “close”):
  - Ex) Euclidean, Jaccard, cosine, edit, Hamming
- LSH families:
  - A family of hash functions H is (d_1, d_2, p_1, p_2)-sensitive if for any x and y:
    - If d(x, y) <= d_1, Pr [h(x) = h(y)] >= p_1; and
    - If d(x, y) >= d_2, Pr [h(x) = h(y)] <= p_2.
  - Ex) minhashing, random hyperplane
- Amplification of an LSH families (“bands” technique):
  - AND construction (“rows in a band”)
  - OR construction (“many bands”)
  - AND-OR/OR-AND compositions

Clustering

What: Given a set of points, group them in ‘clusters’ so that a point is more similar to other points within the cluster compared to points in other clusters (unsupervised learning - without labels)

How: Two types of approaches

- **Point assignments**: maintain a set of clusters, assign points, iteratively refine
- **Hierarchical**: each point is its own cluster, repeatedly combine nearest clusters

Point Assignment approaches

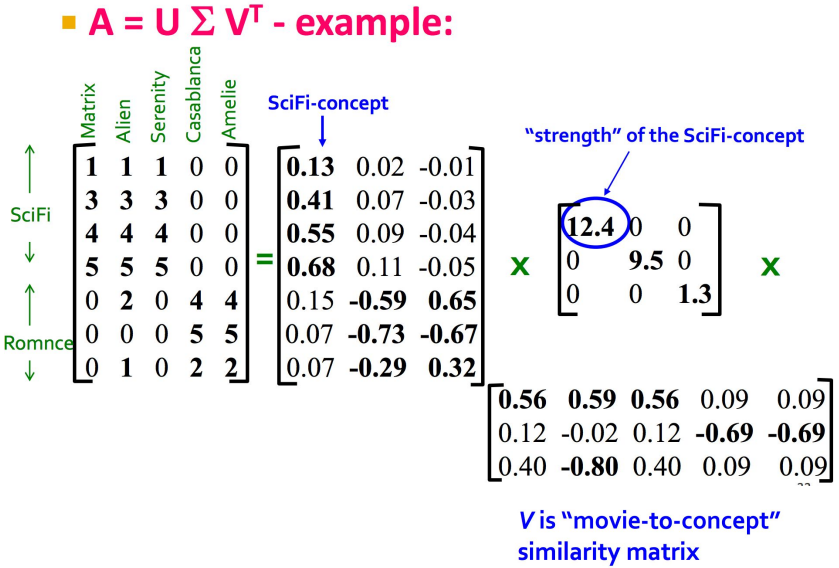
- Spherical/convex cluster shapes
- **k-means:** initialize cluster centroids, assign points to the nearest centroid, iteratively refine estimates of the centroids
  - Euclidean space
  - Sensitive to initialization (K-means++)
  - Good values of “k” empirically derived
  - Assumes dataset can fit in memory
- **BFR algorithm:** variant of k-means for very large datasets (residing on disk)
  - Keep running statistics of previous memory loads
  - Compute centroid, assign points to clusters in a second pass

Hierarchical clustering

- Works better when clusters have weird shapes (e.g. concentric)
- **General approach**:
  - Start with each point in its own cluster
  - Successively merge two “nearest” clusters until convergence
- **Important problems**:
  - Location of clusters: centroid in Euclidean spaces, clustroid in non-Euclidean spaces
  - Intercluster distance: smallest max distance, smallest average distance, cohesion. What works best depends on cluster shapes, often trial and error

Dimensionality Reduction

- Methods of Dimensionality Reduction
  - UV Decomposition
    - M = UV
  - SVD
    - $M = U\Sigma V^T$
  - CUR Decomposition
    - M = CUR
- Motivation
  - Discover hidden correlation
  - Remove redundant and noisy features
  - Easier storage and processing of the data



SVD Calculation

- $M = U\Sigma V^T$
  - Based on the orthonormal properties of U and V, it can be shown the columns of V are eigenvectors of M^TM
  - The columns of U are eigenvectors of MM^T
- Steps to calculate
  - Find Σ, V
    - Find eigenpairs of M^TM
    - Σ is square root of eigenvalues
    - V is the normalized eigenvectors
  - Similarly, to find U, just find eigenpairs of MM^T

PageRank

- PageRank is a method for determining the importance of webpages
  - Named after Larry Page
- Rank of a page depends on how many pages link to it
- Pages with higher rank get more of a vote
- The vote of a page is evenly divided among all pages that it links to

If $i \to j$, then $M_{ji} = \frac{1}{d_i}$ else $M_{ji} = 0$

PageRank

- **PageRank equation** [Brin-Page, '98]

$$r_j = \sum_{i \to j} \beta \frac{r_i}{d_i} + (1 - \beta) \frac{1}{N}$$

- **The Google Matrix *A*:**

$[1/N]_{NxN}$...N by N matrix where all entries are 1/N

$$A = \beta M + (1 - \beta) \left[\frac{1}{N}\right]_{N \times N}$$

- **We have a recursive problem: $r = A \cdot r$**

[Jure Leskovec, CS 246, Winter 2014]

Hubs and Authorities

- Similar to PageRank
- Every webpage gets two scores: an “authority” score, which measures the quality of the webpage, and a “hub” score, which measures how good it is at linking to good webpages
- Mutually recursive definition:
  - Good hubs link to good authorities
  - Good authorities are linked to by good hubs

Hubs and Authorities

- Adjacency matrix $\boldsymbol{A}$ (NxN): $\boldsymbol{A_{ij}} = 1$ if $i \rightarrow j$, 0 otherwise
  - Set: $\boldsymbol{a_i} = \boldsymbol{h_i} = \frac{\mathbf{1}}{\sqrt{n}}$

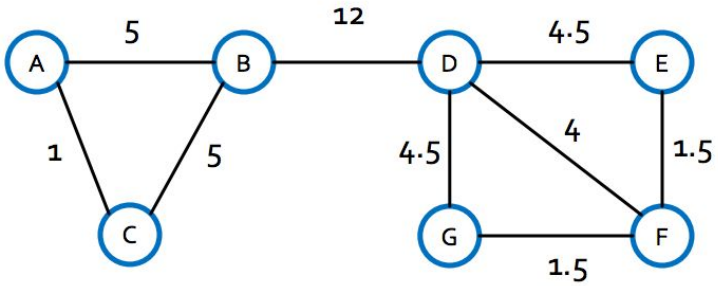
Repeat until convergence:

- $\boldsymbol{h} = \boldsymbol{A} \cdot \boldsymbol{a}$
- $\boldsymbol{a} = \boldsymbol{A^T} \cdot \boldsymbol{h}$
- Normalize $\boldsymbol{a}$ and $\boldsymbol{h}$

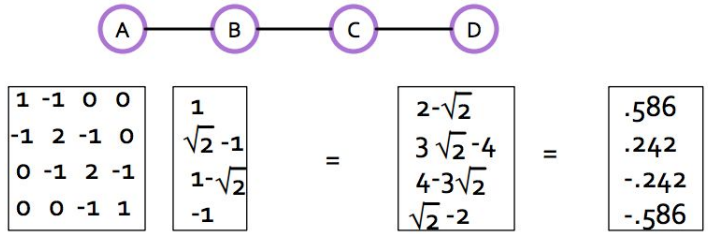
[Jure Leskovec, CS 246, Winter 2014]

Social Networks & Community Detection

- **Basic Terms:**
  - Locality, Community, Diameter, Small-world property
- **Betweenness:**
  - Edges of high betweenness separate communities
  - Girvin-Newman Algorithm

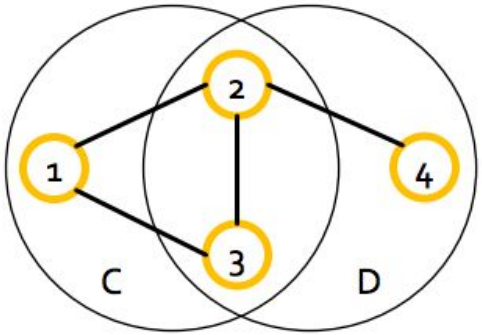


- **Cliques, bi-cliques**
  - Definition: Sets of nodes that are fully connected
  - Growing cliques + bi-cliques
- **Laplacian Matrices**
  - How to Construct a Laplacian Matrix
  - Using eigenvector with the second-smallest eigenvalue

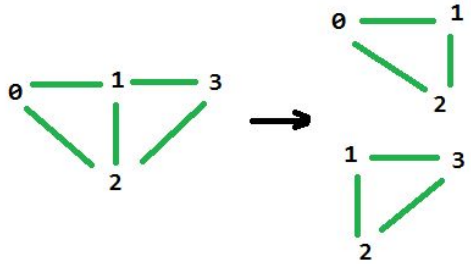


More Graph Algorithms

- **Affiliation-Graph Models(AGM):**
  - Model that best explains the edges in the graph, given a set of communities with associated probabilities
  - Gradient Descent optimizes graph



- **Triangle Counting via “Heavy Hitters”:**
  - “heavy hitter” – a node with degree at least sqrt(M)
  - Using heavy-hitter triangles to count triangles => O(M^1.5) algorithm
  - Potential speed-up compared to naive solution(O(MN) or O(N^3))



Graph with 2 triangles

Transitive Closure

- Definition: Given a directed graph, find out if a vertex j is reachable from another vertex i for all vertex pairs (i, j) in the given graph.
- O(NM) in general, but can parallelize depending on algorithm

Method	Total (Serial) Computation	Parallel Rounds
Warshall	$O(N^3)$	O(N)
Depth-First Search	O(NM)	O(M)
Breadth-First Search	O(NM)	O(D)
Linear + Seminaive	O(NM)	O(D)
Nonlinear + Seminaive	$O(N^3)$	O(log D)
Smart	$O(N^3)$	O(log D)

Seems odd. But in the worst case, almost all shortest paths can have a length that is a power of 2, so there is no guarantee of improvement for Smart.

Social Networks & Graph Algorithms: Cheat Sheet

- Property of Social Graphs: Locality, diameters
- Communities Detection: Betweenness, Girvan-Newman(GN) Algorithm
- Communities Detection: Cliques, Bi-Cliques, properties of Bi-Cliques
- Communities Detection: Using Laplacian Matrices
- The Affiliation-Graph Model(AGM), estimating maximum likelihoods
- Graph Algorithms: Using “Heavy Hitters” to count triangles in a large graph
- Graph Algorithms: Transitive Closure, algorithms on computing TC(focus on Semi-naive TC, Smart TC…)

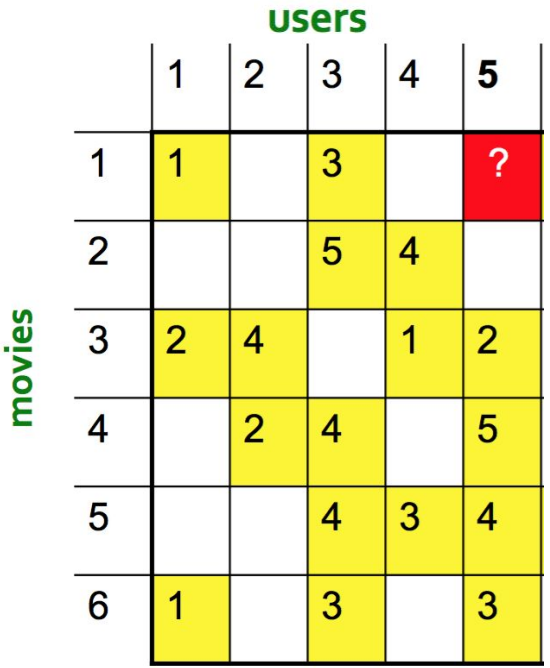
Recommender Systems: Content-Based

What: Given a bunch of users, items and ratings, want to predict missing ratings

How: two methods.

- Content-Based:
  - (1) Collect user profile ***x*** and item profile ***i***
  - (2) Estimate utility: $u(\boldsymbol{x},\boldsymbol{i}) = cos(\boldsymbol{x},\boldsymbol{i})$

- Collaborative Filtering (next slide)



[from lecture slides]

Recommender Systems: Collaborative Filtering

Collaborative Filtering:

- **user-user CF vs item-item CF**
  user-user CF: estimate a user's rating based on ratings of similar users **who have rated the item**. Similar definition for item-item CF.
- **Similarity metrics**
  Jaccard similarity: *binary*; Cosine similarity: *treats missing ratings as "negative"*; Pearson correlation coeff: *remove mean of non-missing ratings, zero-centered.*
- **Baseline estimate**: $b_{xi} = \mu + b_x + b_i$
  In CF, sometimes we remove baseline estimate and only model rating deviations from baseline estimate, so that we're not affected by user/item bias.

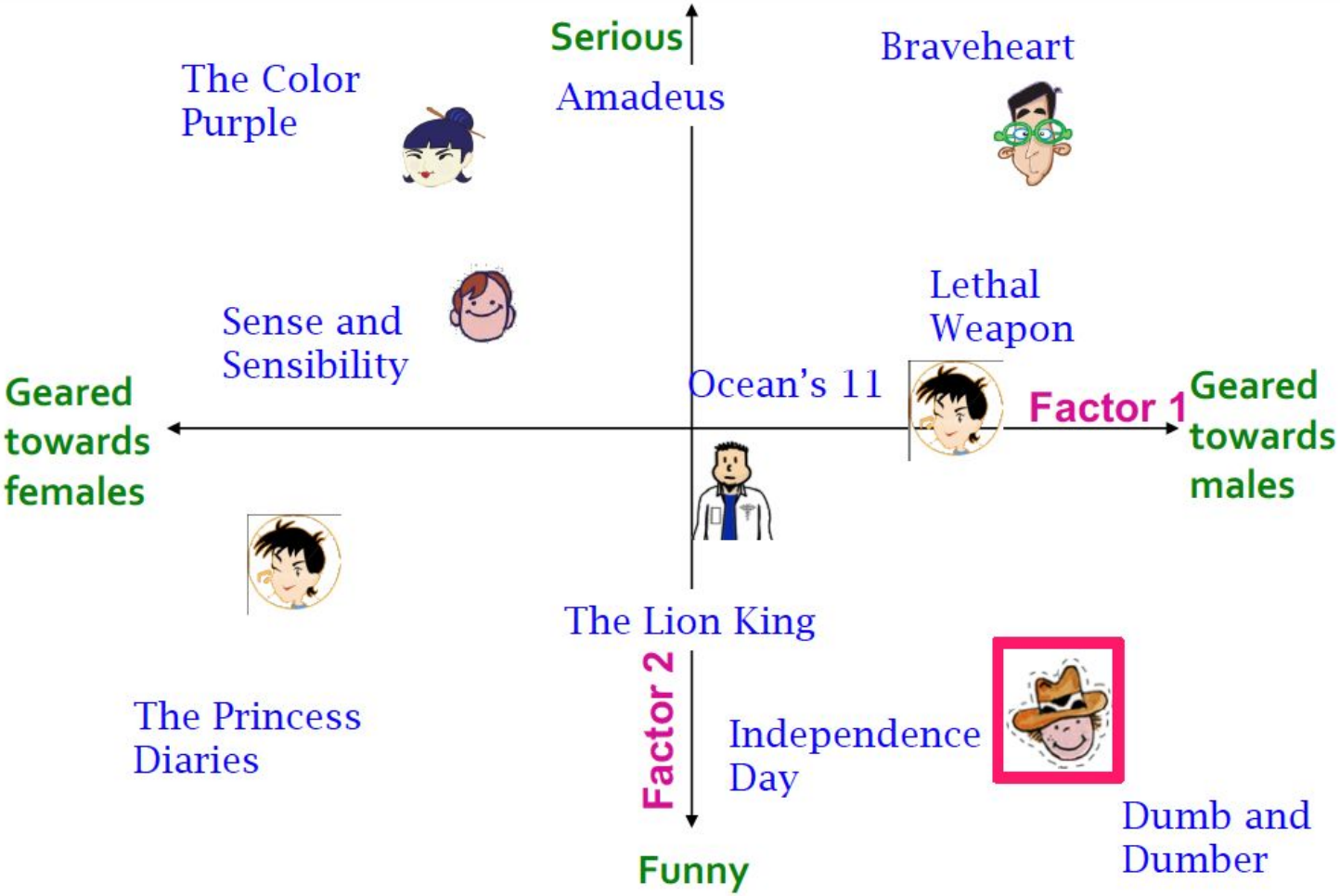
Evaluation: Root-Mean-Square error (RMSE), etc.

Recommender Systems: Latent Factor Models

Motivation: Collaborative filtering is a local approach to predicting ratings based on finding neighbors. Matrix factorization takes a more global view.

Intuition: We decompose users and movies based on a set of latent factors. Using these latent factors, we can make predictions. For example, if you like fantasy movies and Harry Potter has a big fantasy component, then the model will predict that you'll be more likely to like Harry Potter.

Model: $\hat{r}_{xi} = p_x \cdot q_i$ for user x and movie i


Serious
Amadeus
The Color Purple
Braveheart
Sense and Sensibility
Lethal Weapon
Ocean's 11
Geared towards females
Factor 1
Geared towards males
The Lion King
The Princess Diaries
Factor 2
Independence Day
Dumb and Dumber
Funny

Recommender Systems: Latent Factor Models

$$\min_{P,Q} \sum_{(x,i)\in \text{training}} (r_{xi} - p_x \cdot q_i)^2 + \lambda_1 \sum_x \|p_x\|^2 + \lambda_2 \sum_i \|q_i\|^2$$

- Note that we only sum over observed ratings in the training set
- Use regularization to prevent overfitting
- Can solve this via SGD
- Can be extended to include biases (and temporal biases)

$$\hat{r}_{xi} = \mu + b_x + b_i + p_x \cdot q_i$$

- Netflix Prize: get best performance using large ensemble of models

Machine Learning

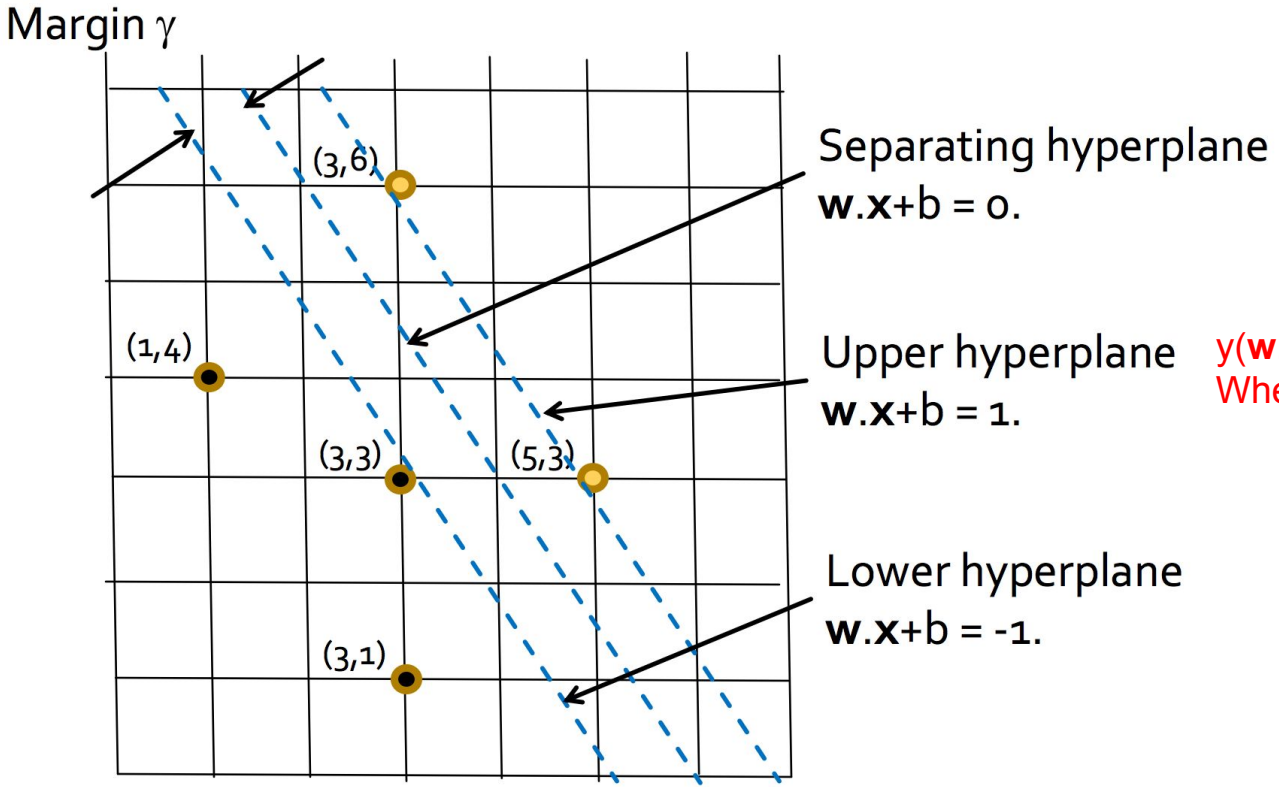
Training examples $\{(\mathbf{x}_1, y_1), (\mathbf{x}_2, y_2), ...\}$ where **x** is the type of item we wish to evaluate, and y is its label. If y belongs to a discrete set, this is a **classification** problem, and if y is a real number, it is a **regression** problem.

In binary (two classes) classification problems, y belongs to {-1, 1}. Example: spam classification, we might have an email **x** that is spam, and its label is y = 1.

Usually evaluated on a test set of the form $\{(\mathbf{x}_1, \mathbf{x}_2, \mathbf{x}_3, \mathbf{x}_4...\}$.

(Here we are only talking about supervised learning, where labels are given).

SVM: Maximum margin classifiers



y(**w**.**x** + b) >= 1
When we have linearly separable data

Measuring Impurity of a Set S

Let p1,..., pk be the fractions of measures of S with the k possible values of y.

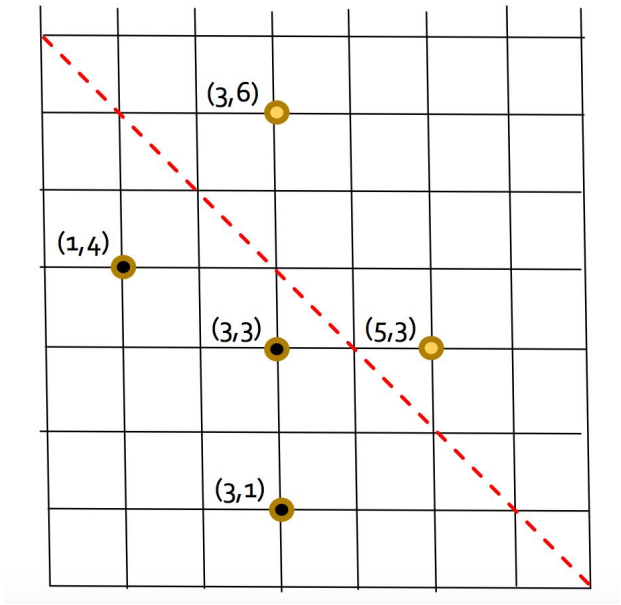
3 main measures of impurity:

1) Accuracy: if output is the most common value of y, what fraction of inputs in the set S are not given their correct output: 1 - max_i p_i
2) GINI Impurity: 1 - Σ_i (p_i)^2
3) Entropy: Σ_i -p_i log_2(p_i) or equivalently Σ_i p_i log_2(1/p_i)

We want low impurity!

Linear Separator

- A linear separator is a d-dimensional vector w and a threshold theta such that the hyperplane defined by w and theta separates the positive and negative examples.
- Given input x, the separator returns +1 If x.w > theta and returns -1 if not.
- The hyperplane is the set of points whose dot product with w is theta.



Black points = -1
Gold points = +1
w = (1,1)
$\theta = 7$

w

Hyperplane **x**.**w** = θ
If **x** = (a,b),
then a+b = 7

Perceptron

Given a set of training points (x, y) where:

1) X is a real valued vector of d dimensions (i.e. a point in a Euclidean space) and
2) y is a binary decision +1 or -1

A perceptron tries to find a linear separator between the positive and negative inputs.

Stochastic Gradient Descent

- performs a parameter update for each training example x(i) and label y(i) computes the gradient of the cost function w.r.t. to the parameters θθ for the entire training dataset

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta; x^{(i)}; y^{(i)}).$$

Bloom Filter [1970]

- Basic construction
  - Use one hash function
  - Errors caused by hash function collisions
  - Bounded error probability
- The Magic of independent trials
  - Use many (independent!) hash functions
  - Combine trials by noting if *all* corresponding cells have a 1
  - Exponentially reduce probability of error
- Tune number of buckets and number of hash functions
  - Query time, and insertion time increase linearly with number of hash functions
  - Trade off error probability with space usage
  - But, space usage does not depend on number of elements in stream!

Count-Min Sketch (HW 4) [2003]

- Similar in spirit to Bloom filter
- Basic construction
  - Use one hash function
  - Each cell stores an *overestimate* of the true count
    - Again, errors caused by hash function collisions
  - In expectation, overestimate not too bad
  - Bounded probability of overestimating
    - Use a tail bound

Count-Min Sketch (HW 4) [2003]

- The Magic of Independent Trials
  - Use many (independent!) hash functions
  - Combine trials by taking the *best* overestimate (the minimum!)
  - Exponential reduction in error probability
- Tune error tolerance ε and error probability δ
  - Space usage is $O(1/\varepsilon \log(1/\delta))$
  - Trade off both tolerance and error probability with space usage
  - Again, space usage does not depend on number of elements in stream!
  - For fixed tolerance and error probability, space is a constant
- This is a primitive at many large tech companies
  - Search queries
  - Web requests

Flajolet-Martin

- Problem: a data stream consists of elements chosen from a set of size n. Maintain a count of the number of distinct elements seen so far.
- Pick a hash function h that maps each of the n elements to at least $\log_2 n$ bits.
- For each stream element a, let r(a) be the number of trailing 0's in h(a).
  - Record R = the maximum r(a) seen for any a in the stream.
  - Also known as the "tail length"
- Estimate of distinct elements = 2^R .
- Intuitively, seeing r trailing 0s is "unusual"
  - More distinct elements leads to a higher chance of seeing this "unusual" event
- If we notice this "unusual" event, our estimate should be correspondingly higher

AMS Algorithm

- Problem: Suppose a stream has elements chosen from a set of n values. Let m_i be the number of times value i occurs. Find the k^{th} moment which is the sum of $(m_i)^k$ over all i.
  - 0^{th} moment = number of different elements in the stream.
  - 1^{st} moment = sum of counts of the numbers of elements = length of the stream.
  - 2^{nd} moment = measure of how uneven the distribution is.
- Algorithm for 2^{nd} moment:
  - Assume stream has n elements
  - Pick a random starting and let the chosen time have element a in the stream.
  - Let X = # times a is seen in the stream from that point onward
  - Estimate of 2^{nd} moment = n(2X -1)