Additional Topics: Big Data

Lecture #2

Algorithms for Big Data

Joseph Bonneau
jcb82@cam.ac.uk
April 30, 2012

Today's topic: algorithms

Do we need new algorithms?

Quantity is a quality of its own

-Joseph Stalin, apocryphal

- can't always store all data
  - online/streaming algorithms
- memory vs. disk becomes critical
  - algorithms with limited passes
- N^2 is impossible
  - approximate algorithms
- human insight is limited
  - algorithms for high-dimensional data

Simple algorithms, more data

- *Mining of Massive Datasets*
    - Anand Rajaraman, Jeffrey Ullman 2010
    - Plus Stanford course, pieces adapted here
- "Synopsis Data Structures for Massive Data Sets"
    - Phillip Gibbons, Yossi Mattias, 1998
- "The Unreasonable Effectiveness of Data"
    - Alon Halevy, Peter Norvig, Fernando Perreira, 2010

The “easy” cases

- sorting
  - Google 1 trillion items, (1 PB) sorted in 6 hours
- searching
  - hashing & distributed search

Streaming algorithms

Have we seen x before?

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
  - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 0000 0000

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
    - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 0000 0000

$\mathbf{H}(jcb82)$ = 4, 7, 5, 2

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
  - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 0101 1010

$\mathbf{H}(jcb82) = 4, 7, 5, 2$

$\mathbf{H}(rja14) = 2, 4, 3, 8$

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
  - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 0111 1011

$\mathbf{H}(jcb82) = 4, 7, 5, 2$

$\mathbf{H}(rja14) = 2, 4, 3, 8$

$\mathbf{H}(mgk25) = 8, 3, 7, 1$

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
  - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 1111 1011

$\mathbf{H}(jcb82) = 4, 7, 5, 2$
$\mathbf{H}(rja14) = 2, 4, 3, 8$
$\mathbf{H}(mgk25) = 8, 3, 7, 1$
$\mathbf{H}(fms27) = 1, 5, 2, 4$

Bloom filter (Bloom 1970)

- assume we can store n bits $b_1, b_2, \ldots b_n$
- use a hash function $\mathbf{H}(x) = h_1, h_2, \ldots h_k$
  - each $h_i \in [1, n]$
- when we see x, set b_i, for each i in $\mathbf{H}(x)$

n = 8, k = 4

B = 1111 1011

$\mathbf{H}(jcb82)$ = 4, 7, 5, 2
$\mathbf{H}(rja14)$ = 2, 4, 3, 8
$\mathbf{H}(mgk25)$ = 8, 3, 7, 1
$\mathbf{H}(fms27)$ = 1, 5, 2, 4

false positive

Bloom filters

- constant time lookup/insertion
- no false negatives
- false positives from N items: $\ln p = (-N/n) / (\ln 2)^2$
    - at 1% p, 1 GB of RAM ≈ 77 billion unique items!
- once full, can't expand
- counting bloom filter: store integers instead of bits

Application: market baskets

- assume TESCO stocks 100k items
    - Amazon stocks many more
- market basket:
    - {ravioli, pesto sauce, olive oil, wine}
- perhaps 1 billion baskets per year
    - $2^{100,000}$ possible baskets...
- what items are "frequently" purchased together?
    - more frequent than predicted by chance

Application: market baskets

- pass #1: add all sub-baskets to counting BF
  - with 8 GB of RAM: 25B baskets
  - restrict basket size $\binom{100{,}000}{3} > 100$ trillion
- pass #2: check all sub-baskets
  - check $\{x\}, \{y_1, \ldots, y_n\}\ \{x, y_1, \ldots\ y_n\}$
  - store possible rules $\{y_1, \ldots, y_n\} \rightarrow \{x\}$
- pass #3: eliminate false positives
- for better approaches see Ramarajan/Ullman

Other streaming challenges

- rolling average of previous k items
  - sliding window of traffic volume
- hot list – most frequent items seen so far
  - probabilistically start tracking new items
- histogram of values

Querying data streams

```
Select segNo, dir, hwy
From SegSpeedStr [Range 5 Minutes]
Group By segNo, dir, hwy
Having Avg(speed) < 40
```

- Continuous Query Language (CQL)
    - Oracle now shipping version 1
- based on SQL...

Similarity metrics

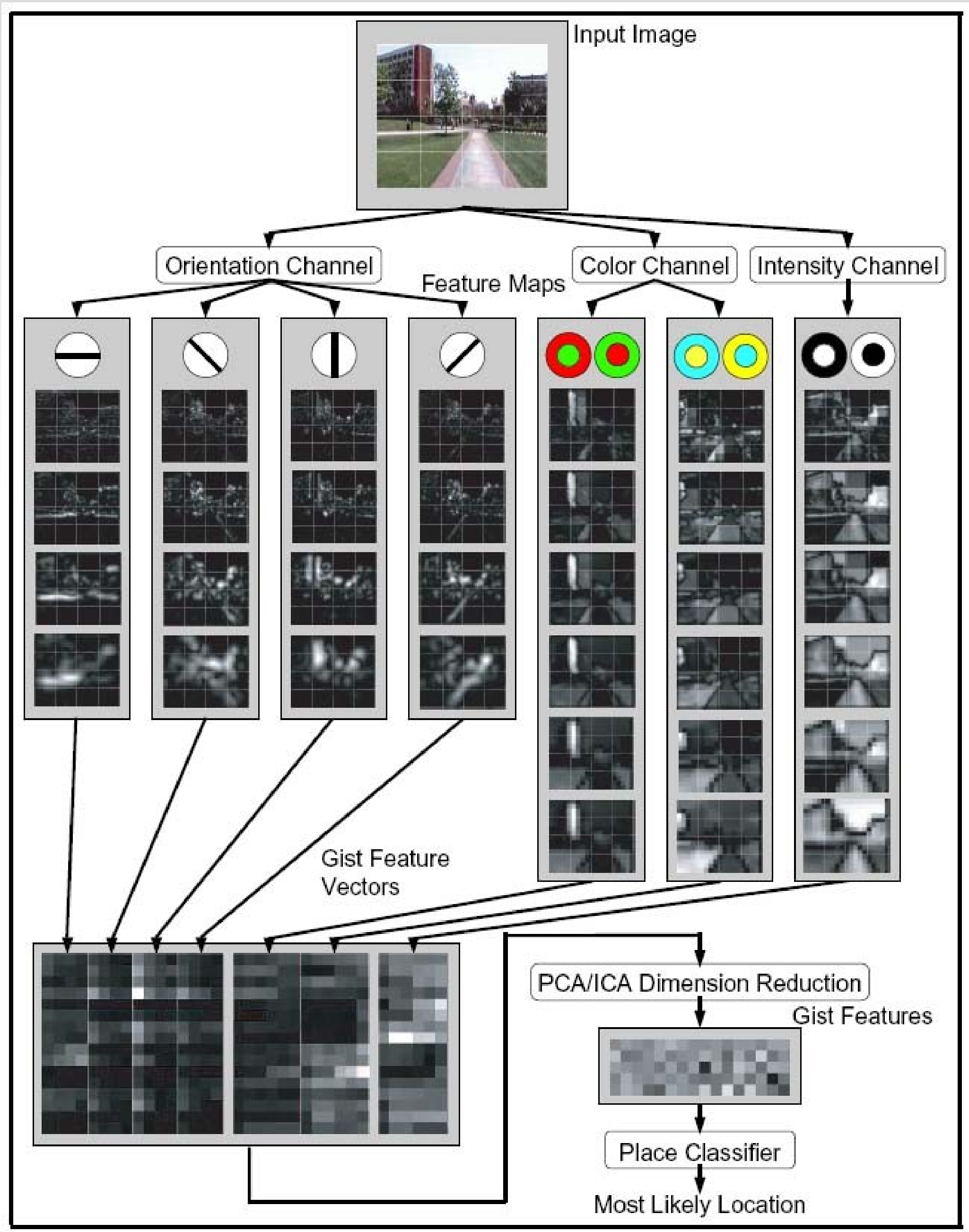
Which items are similar to x?

Similarity metrics

Which items are similar to x?

- reduce x to a high-dimensional vector
- features:
  - words in a document (bag of words)
  - *shingles* (sequences of w characters)
  - various body part measurements (faces)
  - edges, colours in a photograph

Feature extraction - images

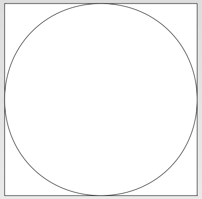


Gist
Siagan, Itti, 2007

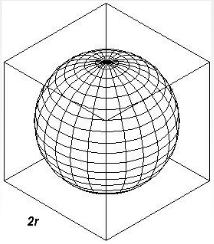
Distance metrics

- Chebyshev distance $max_i(x_i - y_i)$
- Manhattan distance $\sum |x_i - y_i|$
- Hamming distance for binary vectors
- Euclidean distance $\sqrt{\sum (x_i - y_i)^2}$
- Mahalanobis distance $\sqrt{\sum \frac{(x_i - y_i)^2}{\sigma^2}}$
    - Adjusts for different variance
- Cosine distance $\cos\theta = \frac{\sum x_i \cdot y_i}{|X| \cdot |Y|}$
- Jaccard distance (binary) $\frac{|X \cup Y| - |X \cap Y|}{|X \cup Y|}$

The curse of dimensionality



- d = 2: area(○) = π, area(□) = 4
  - ratio = **0.78**



- d = 3: area(○) = (4/3) π, area(□) = 8
  - ratio = **0.52**
- d → ∞:
  - ratio → **0**!
- all points are “far” by Euclidean distance

The curse of dimensionality

- space is typically *very* sparse
- most dimensions are semantically useless
    - hard for humans to tell which ones
- need **dimension reduction**

Singular value decomposition

$$D_{[m \times n]} = U_{[m \times n]} \cdot \Sigma_{[n \times n]} \cdot V^T_{[n \times n]}$$

books to words
(input data)

books to genres

genre strength
(diagonal)

genres to words

see Baker '05

Singular value decomposition

books to words
(input data)

books to words
(approximate)

$$\begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix} \approx \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix}$$

$$\approx \qquad\qquad =$$

$$\begin{bmatrix} -.54 & .07 & .82 \\ -.1 & -.59 & -.11 \\ -.53 & .06 & -.21 \\ -.65 & .07 & -.51 \\ -.06 & -.8 & .09 \end{bmatrix} \cdot \begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix} \cdot \begin{bmatrix} -.46 & .02 & -.87 & 0 & .17 \\ -.07 & -.76 & .06 & .6 & .23 \\ -.74 & 0.1 & .28 & .22 & -.56 \end{bmatrix}$$

books to genres

genre strength
(diagonal)

genres to words

see Baker '05

Singular value decomposition

books to words
(input data)

books to words
(approximate)

$$\begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix} \approx \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix}$$

$\approx$

books to genres

$=$

$$\boxed{\begin{bmatrix} -.54 & .07 & .82 \\ -.1 & -.59 & -.11 \\ -.53 & .06 & -.21 \\ -.65 & .07 & -.51 \\ -.06 & -.8 & .09 \end{bmatrix} \cdot \begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix}} \cdot \begin{bmatrix} -.46 & .02 & -.87 & 0 & .17 \\ -.07 & -.76 & .06 & .6 & .23 \\ -.74 & 0.1 & .28 & .22 & -.56 \end{bmatrix}$$

books to genres

genre strength
(diagonal)

genres to words

see Baker '05

Singular value decomposition

books to words
(input data)

books to words
(approximate)

$$\begin{bmatrix} 2 & 0 & 8 & 6 & 0 \\ 1 & 6 & 0 & 1 & 7 \\ 5 & 0 & 7 & 4 & 0 \\ 7 & 0 & 8 & 5 & 0 \\ 0 & 10 & 0 & 0 & 7 \end{bmatrix} \approx \begin{bmatrix} 2.29 & -0.66 & 9.33 & 1.25 & -3.09 \\ 1.77 & 6.76 & 0.90 & -5.50 & -2.13 \\ 4.86 & -0.96 & 8.01 & 0.38 & -0.97 \\ 6.62 & -1.23 & 9.58 & 0.24 & -0.71 \\ 1.14 & 9.19 & 0.33 & -7.19 & -3.13 \end{bmatrix}$$

$\approx$ $=$

genres to words

$$\begin{bmatrix} -.54 & .07 & .82 \\ -.1 & -.59 & -.11 \\ -.53 & .06 & -.21 \\ -.65 & .07 & -.51 \\ -.06 & -.8 & .09 \end{bmatrix} \cdot \boxed{\begin{bmatrix} 17.92 & 0 & 0 \\ 0 & 15.17 & 0 \\ 0 & 0 & 3.56 \end{bmatrix} \cdot \begin{bmatrix} -.46 & .02 & -.87 & 0 & .17 \\ -.07 & -.76 & .06 & .6 & .23 \\ -.74 & 0.1 & .28 & .22 & -.56 \end{bmatrix}}$$

books to genres

genre strength
(diagonal)

genres to words

see Baker '05

Singular value decomposition

- computation takes $O(mn^2)$, with $m > n$
- useful but out-of-reach for largest datasets
- implemented in most statistics packages
  - R, MATLAB, etc
- (often) better linear algebra approaches exist
  - CUR, CMD decomposition

Locality-sensitive hashing

- replace SVD with simple probabilistic approach
- choose a family of hash functions **H** such that:
    - distance(X, Y) ≈ distance(**H**(X), **H**(Y))
    - **H** can produce any number of bits
- compute several different **H**
- investigate further if **H**(X) = **H**(Y) exactly
    - scale output size of **H** to minimise cost

MinHash implementation

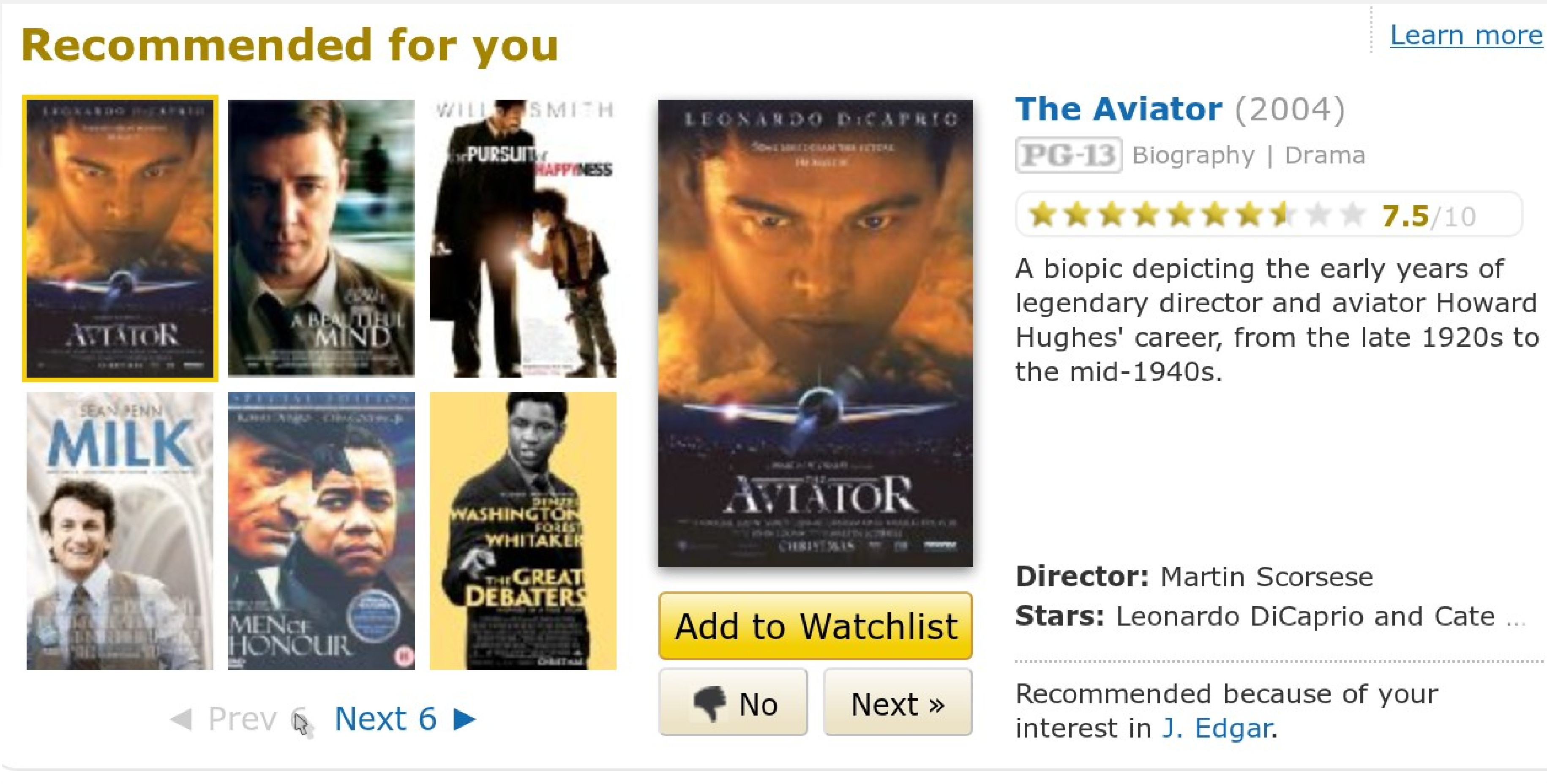
- compute a random permutation $\sigma_R(X)$
- count the number of 0's before a 1 appears
- **theorem:**
  - $pr[MH(X) = MH(Y)] = 1 – Jaccard(X, Y)$
- combine multiple permutations to add bits

LSH example – Flickr photos

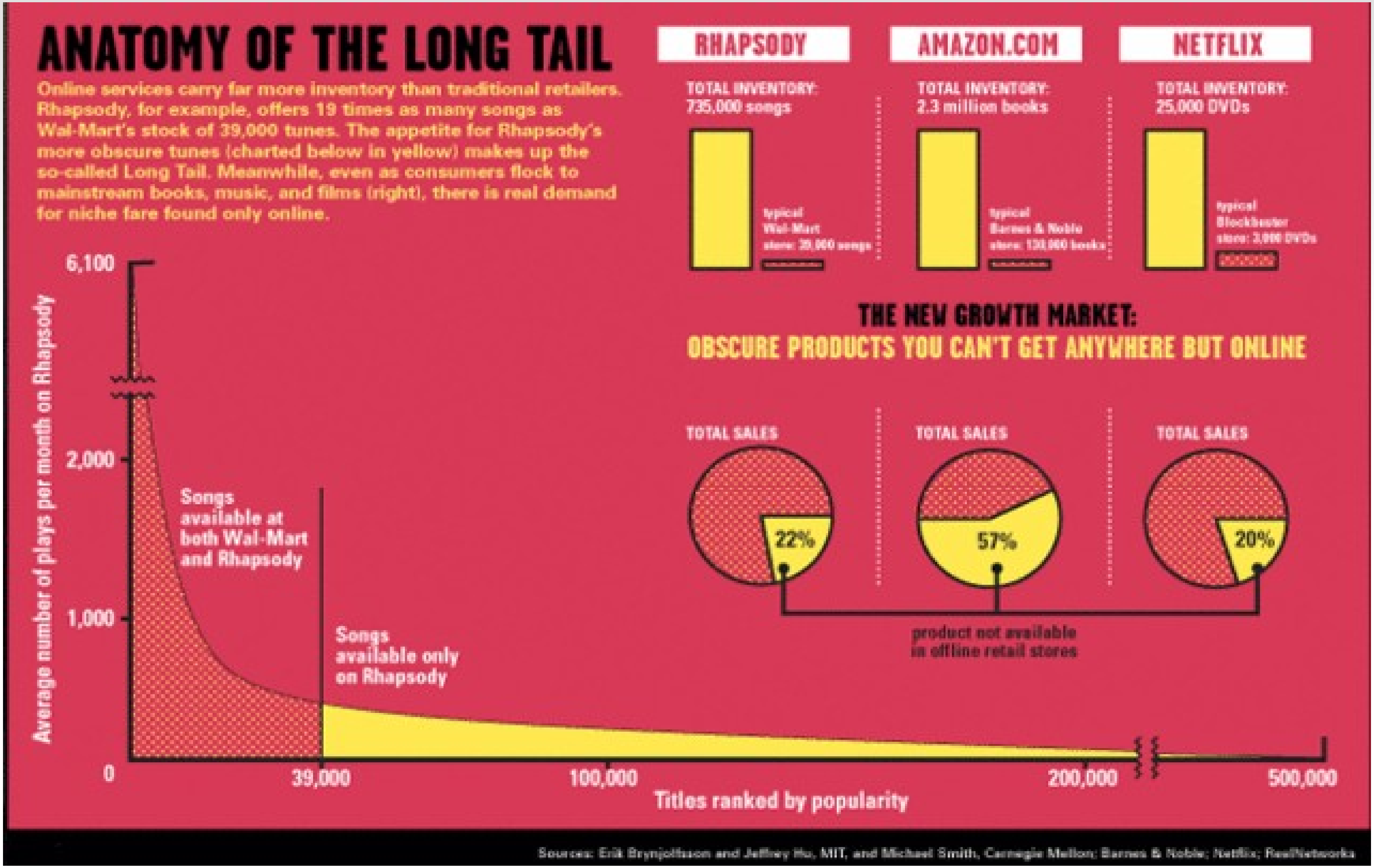
Kulis and Grauman, 2010

Recommendation systems

Can we recommend books, films, products to users based on their personal tastes?



Recommendation systems



Anderson 2004

Content-based filtering

- extract features from content
    - actors
    - director
    - genre
    - keywords in plot summary
    - etc.
- find nearest-neighbours to what a user likes or buys

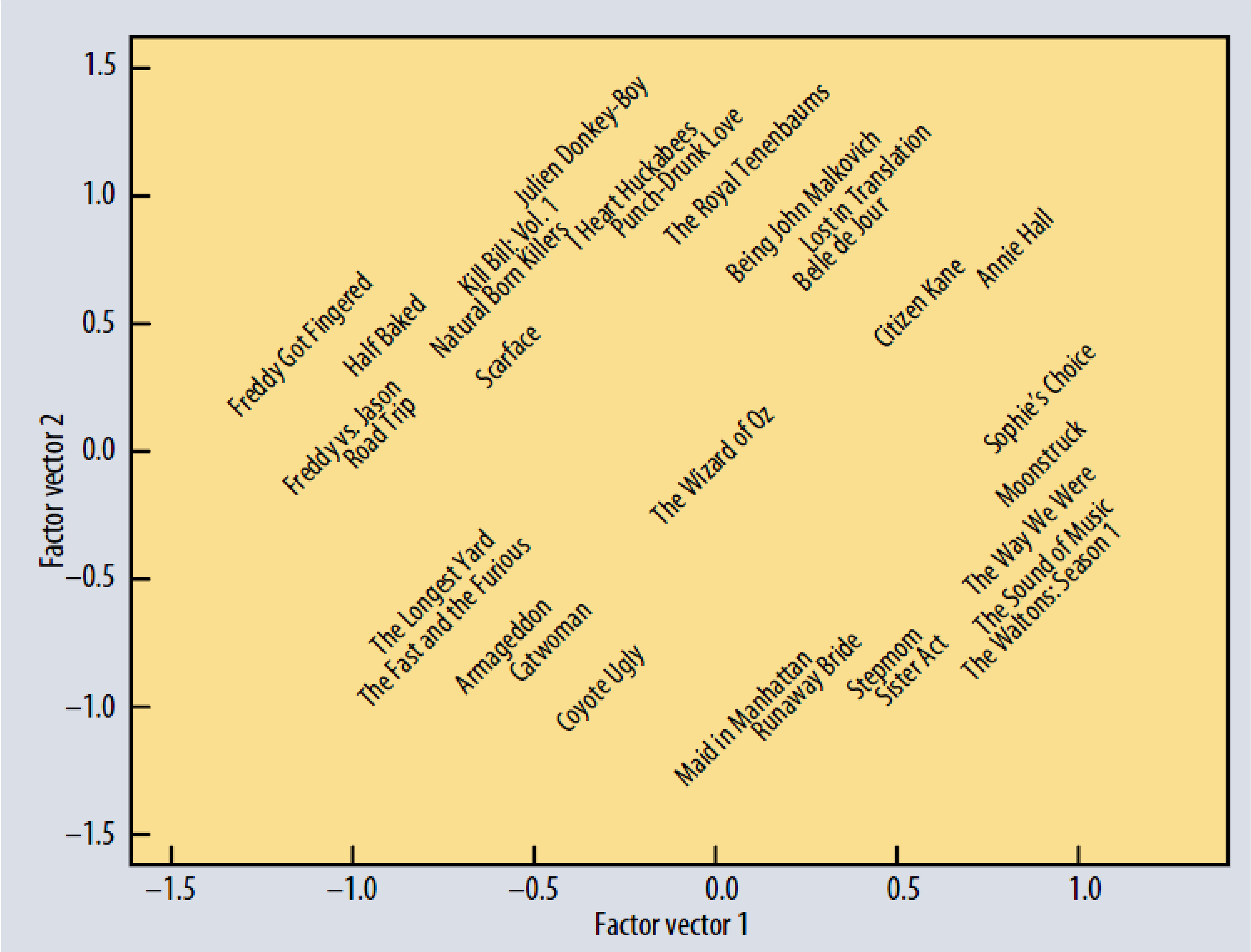
Content-based filtering

- PROS
    - rate new items
    - no herding
    - no user information exposed to others
- CONS
    - features may not be relevant
    - recommendations may be boring
        - “filter bubble”

Collaborative filtering

- features are user/item interactions
  - purchases
  - explicit ratings
    - need lots of clean-up, scaling
- user-user filtering: find similar users
  - suggest their top ratings
  - scale for each user's rating style
- item-item filtering: find similar items
  - suggest most similar items

Item-item filtering preferred



Koren, Bell, Volinksy, 2009

Collaborative filtering

- PROS
  - automatic feature extraction
  - surprising recommendations
- CONS
  - ratings for new users/items
  - herding
  - privacy

The Netflix Prize, 2006-2009

- **100M** film ratings made available
  - **480k** users
  - **17k** films
  - (shoddily) anonymised
- **3M** ratings held in reserve
  - goal: improve predictions by 10%
  - measure by RMS error
- US **$1 M** prize
  - attracted over **2500** teams

Netflix Prize insights

- need to heavily normalise user ratings
  - some users more critical than others
  - user rating style
  - temporal bias
- latent item factors is strongest approach

Netflix Prize league table

Netflix Prize COMPLETED

Home | Rules | Leaderboard | Update

Leaderboard

Showing Test Score. Click here to show quiz score

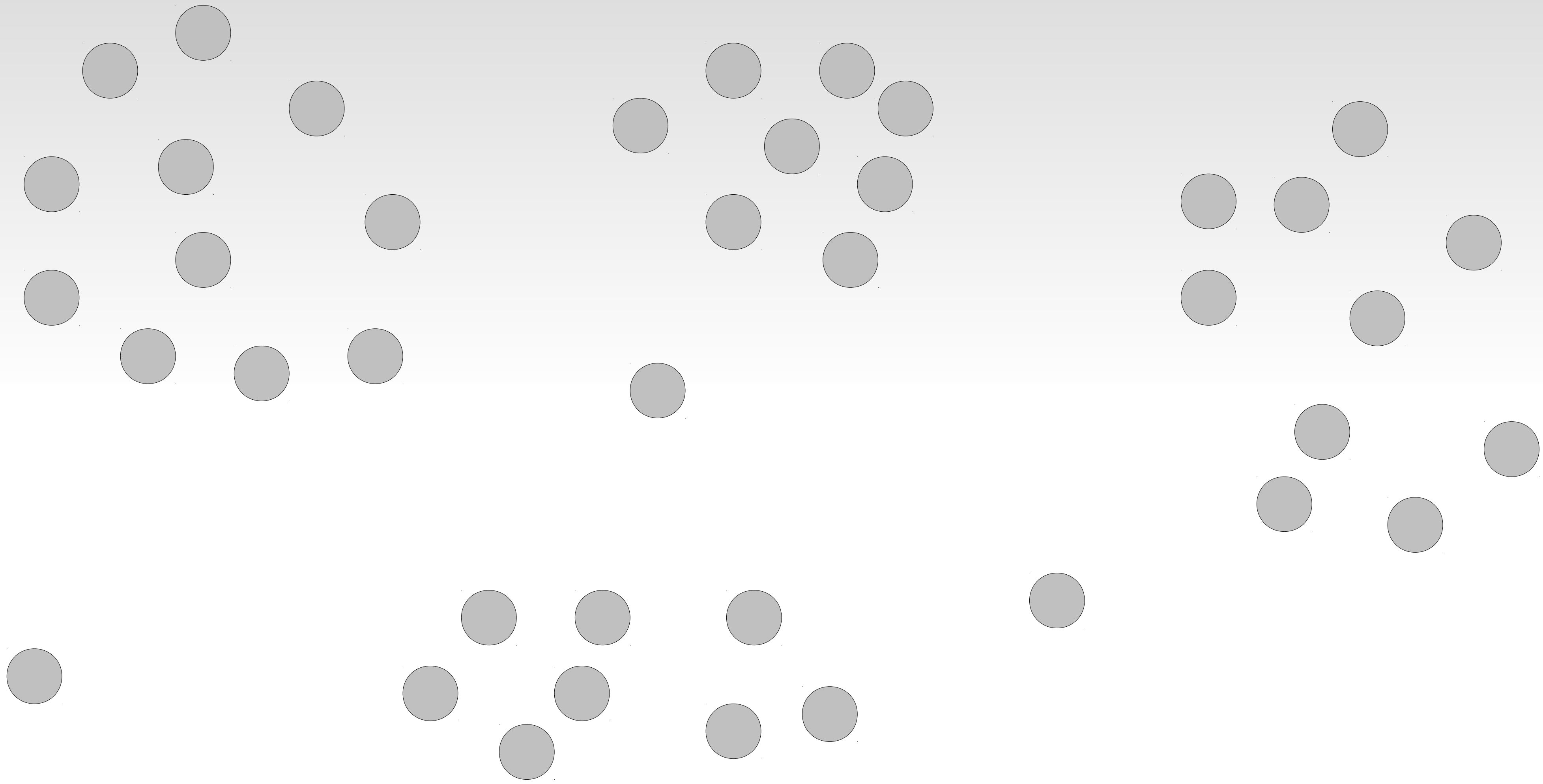
Display top 20 **leaders.**

Rank	Team Name	Best Test Score	% Improvement	Best Submit Time
Grand Prize - RMSE = 0.8567 - Winning Team: BellKor's Pragmatic Chaos				
1	BellKor's Pragmatic Chaos	0.8567	10.06	2009-07-26 18:18:28
2	The Ensemble	0.8567	10.06	2009-07-26 18:38:22
3	Grand Prize Team	0.8582	9.90	2009-07-10 21:24:40
4	Opera Solutions and Vandelay United	0.8588	9.84	2009-07-10 01:12:31
5	Vandelay Industries !	0.8591	9.81	2009-07-10 00:32:20
6	PragmaticTheory	0.8594	9.77	2009-06-24 12:06:56
7	BellKor in BigChaos	0.8601	9.70	2009-05-13 08:14:09
8	Dace	0.8612	9.59	2009-07-24 17:18:43
9	Feeds2	0.8622	9.48	2009-07-12 13:11:51
10	BigChaos	0.8623	9.47	2009-04-07 12:33:59
11	Opera Solutions	0.8623	9.47	2009-07-24 00:34:07
12	BellKor	0.8624	9.46	2009-07-26 17:19:11

Netflix Prize aftermath

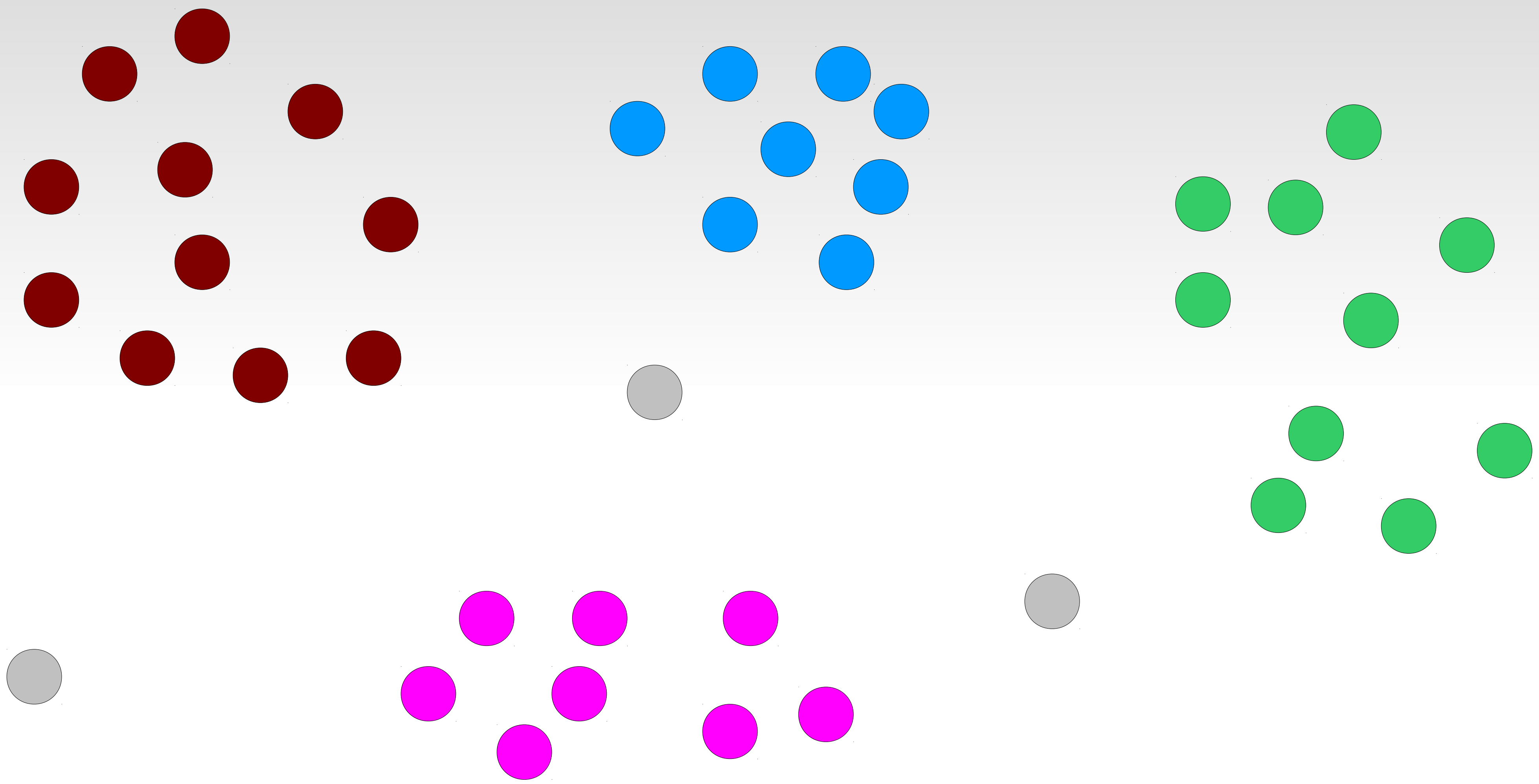
- winning solution is a messy hybrid
- never implemented in practice!
    - too much pain for little gain...

Clustering



Can we identify *groups* of similar items?

Clustering

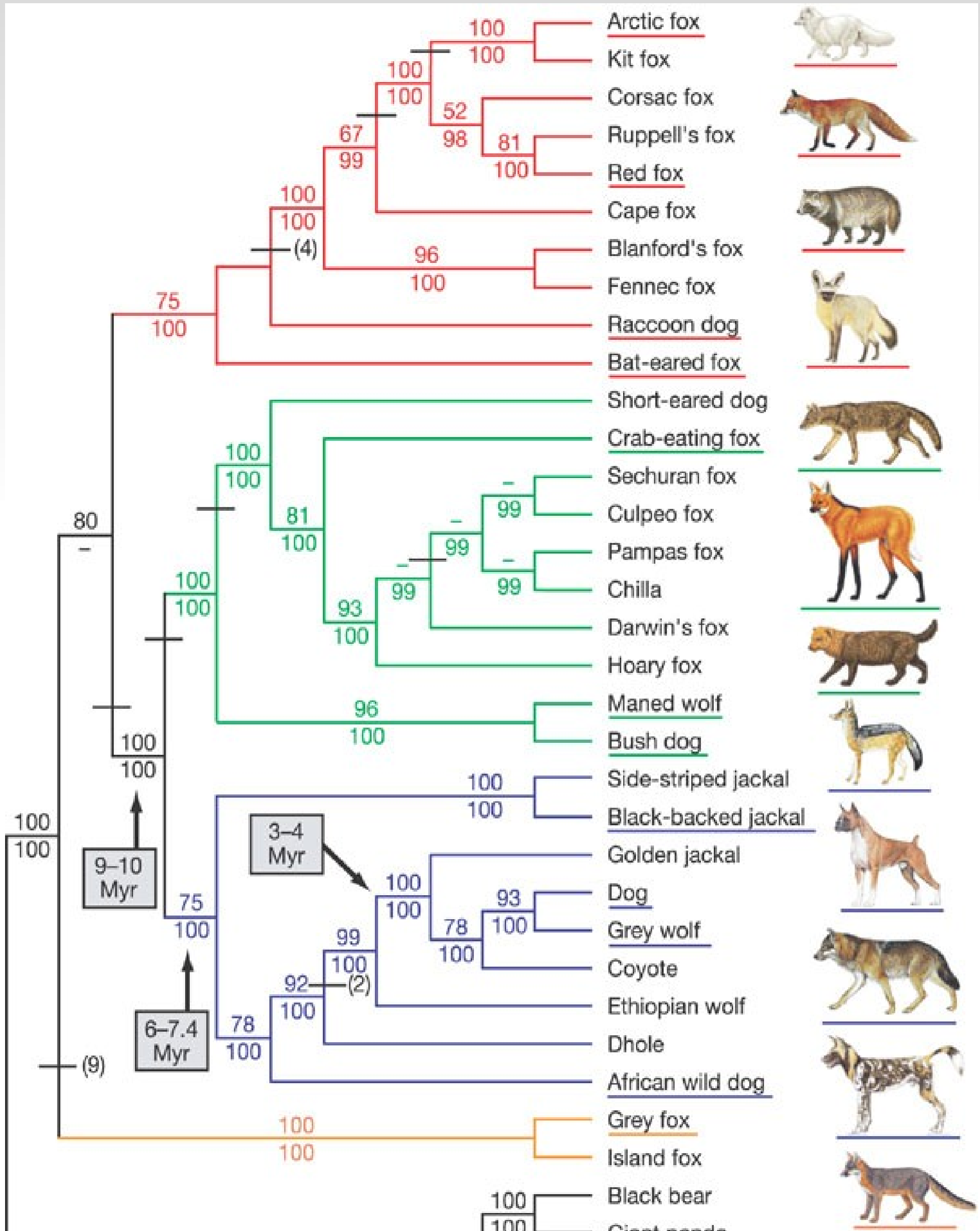


Can we identify *groups* of similar items?

Clustering

- identify categories of:
  - organisms (species)
  - customers (tastes)
  - graph nodes (community detection)
- develop cluster scores based on item similarity
  - diameter of clusters?
  - radius of clusters?
  - average distance?
  - distance from other *clusters*?

Hierarchical clustering



☹

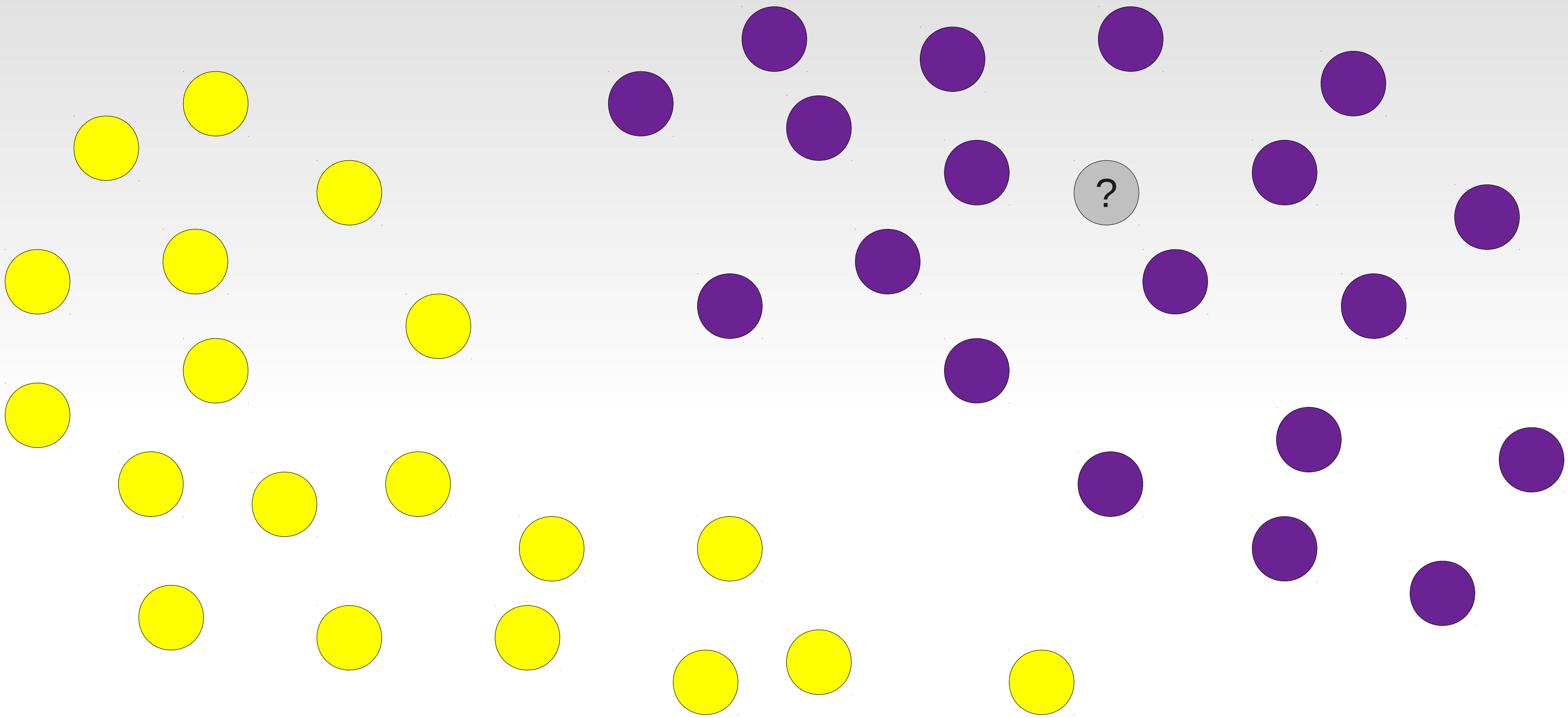
$O(N^2 \lg N)$

Lindblad-Toh et al. 2005

Approximate *k*-means

- choose # of categories k in advance
  - can test various values
- draw a random "RAM-full" of items
- cluster them "optimally" into k clusters
- identify the "centre"
  - *centroid*: average of points (Euclidean)
  - *clustroid*: closest to other points
- assign remaining points to closest centre
  - $O(N)$ time

Classification

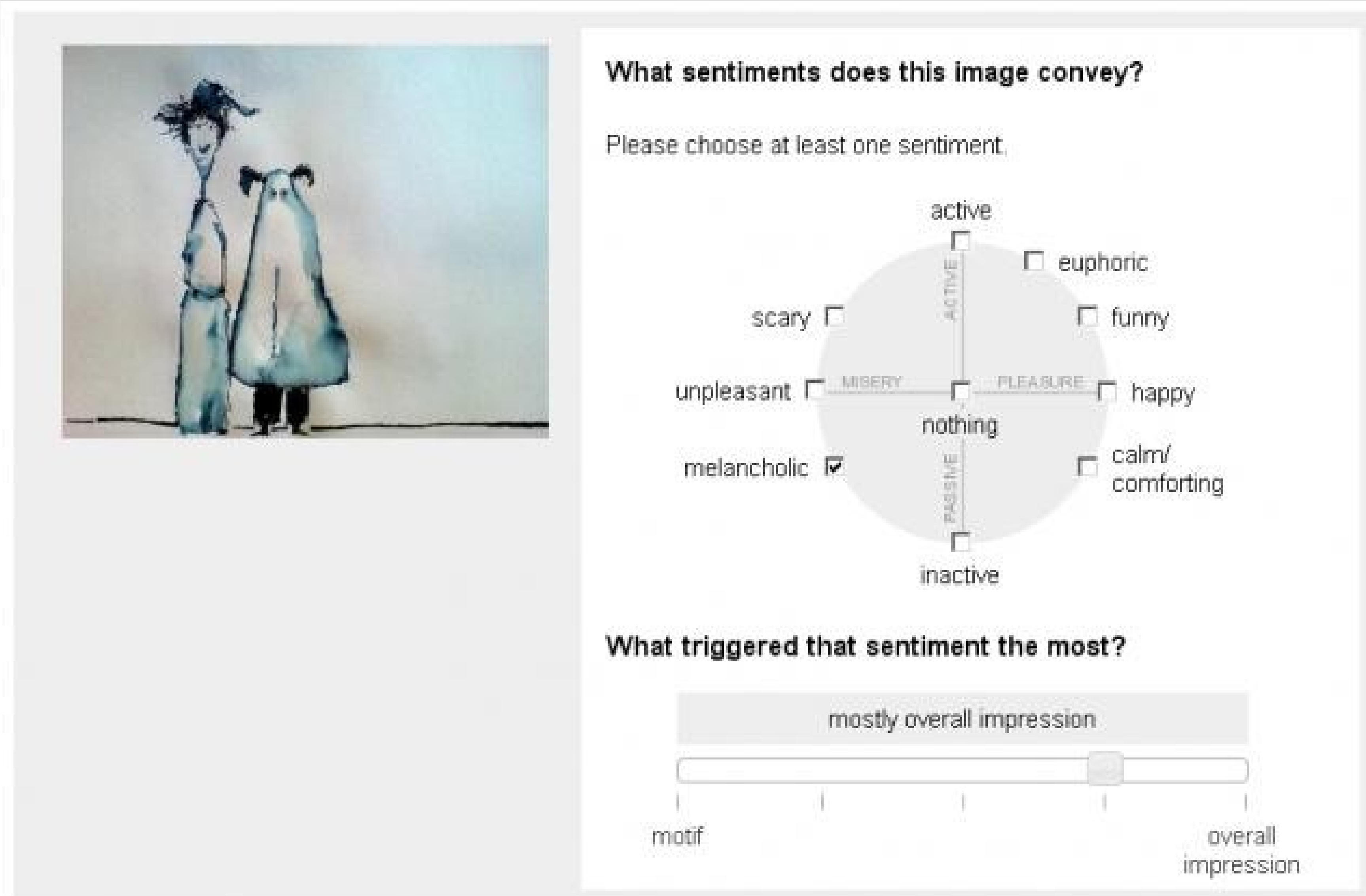


Given some examples, can we classify new items?

Classification

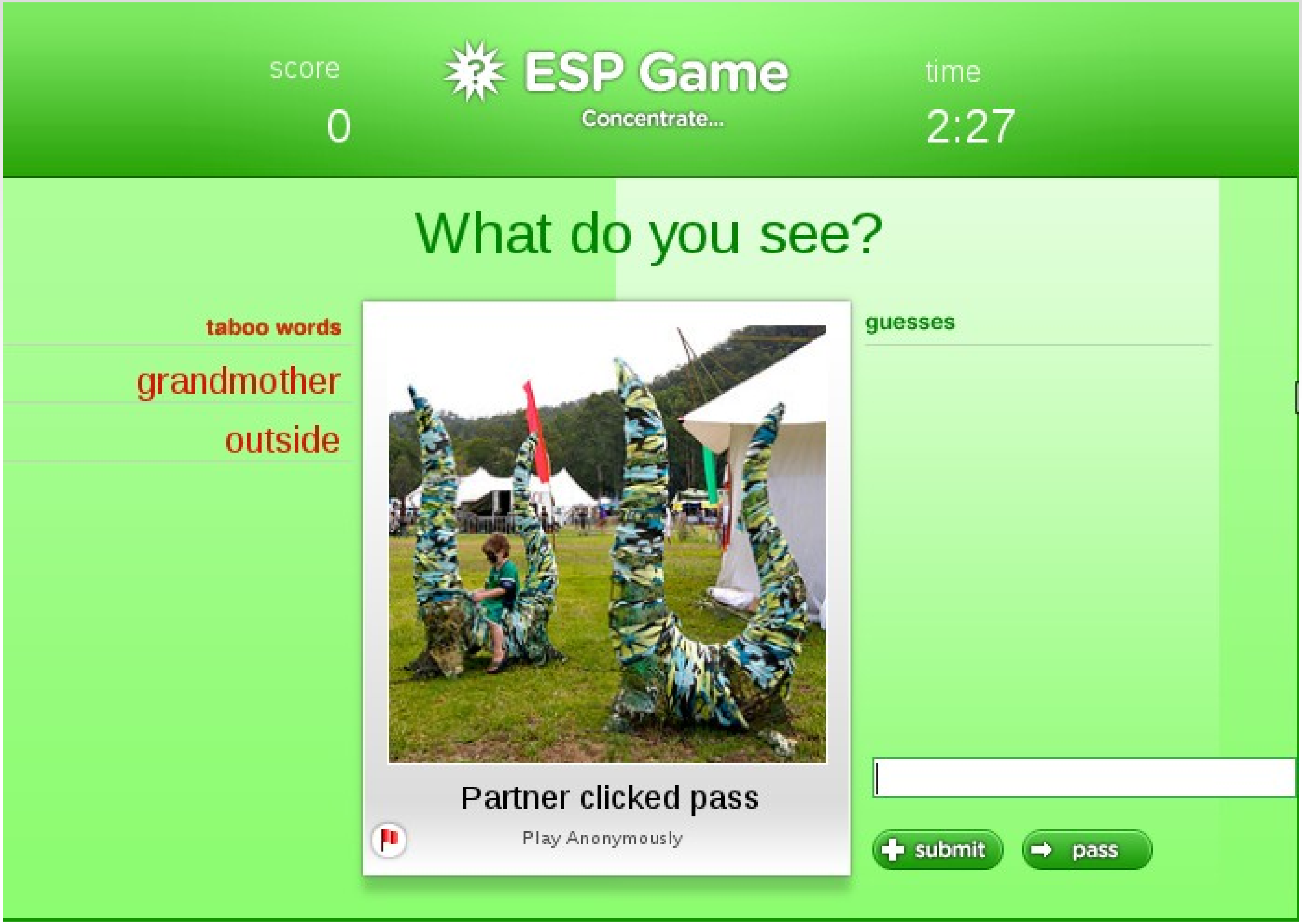
- is this item...
  - a spam email?
  - a cancerous cell?
  - the letter 'J'?
- many approaches exist:
  - neural networks
  - Bayesian decision trees
  - domain-specific probabilistic models

Manual classification



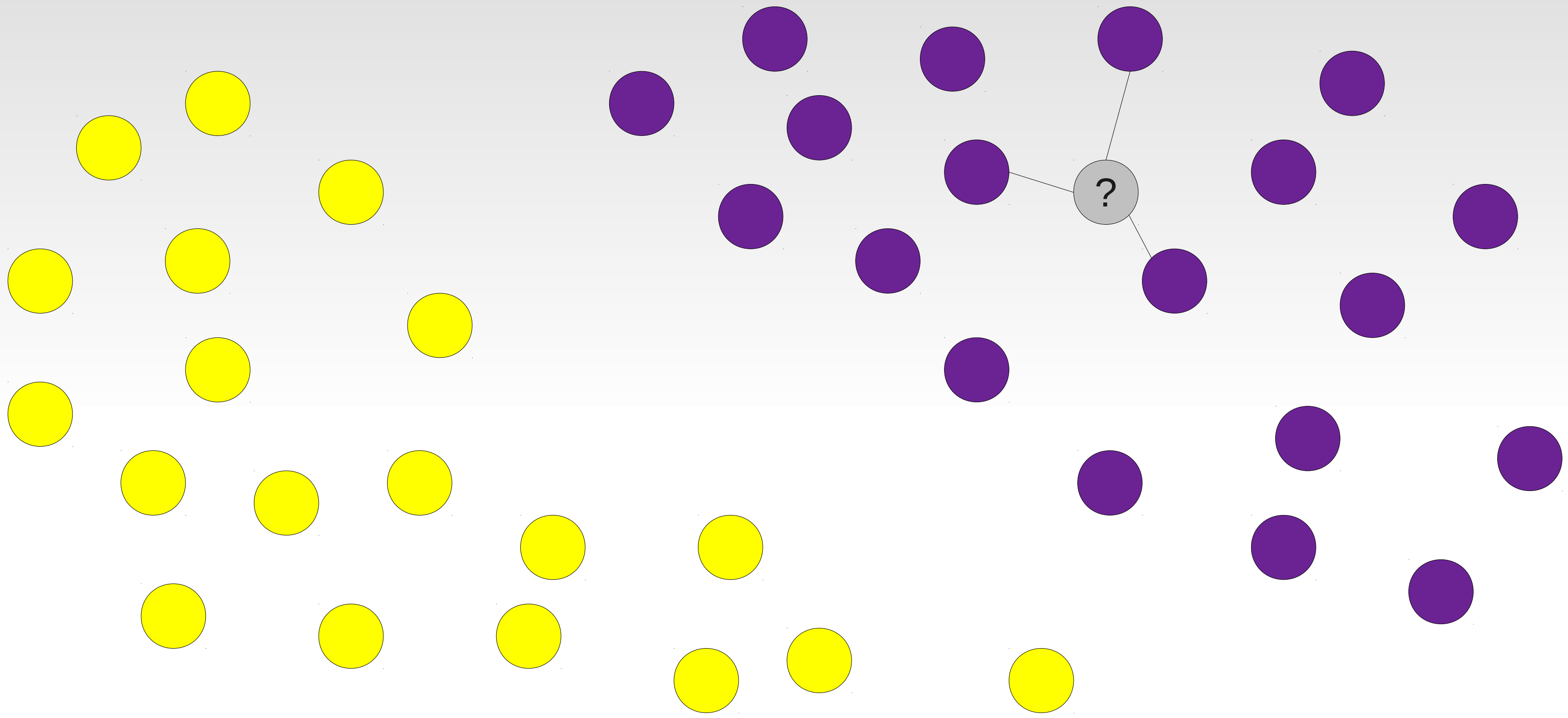
Nowak et al. 2011

Manual classification



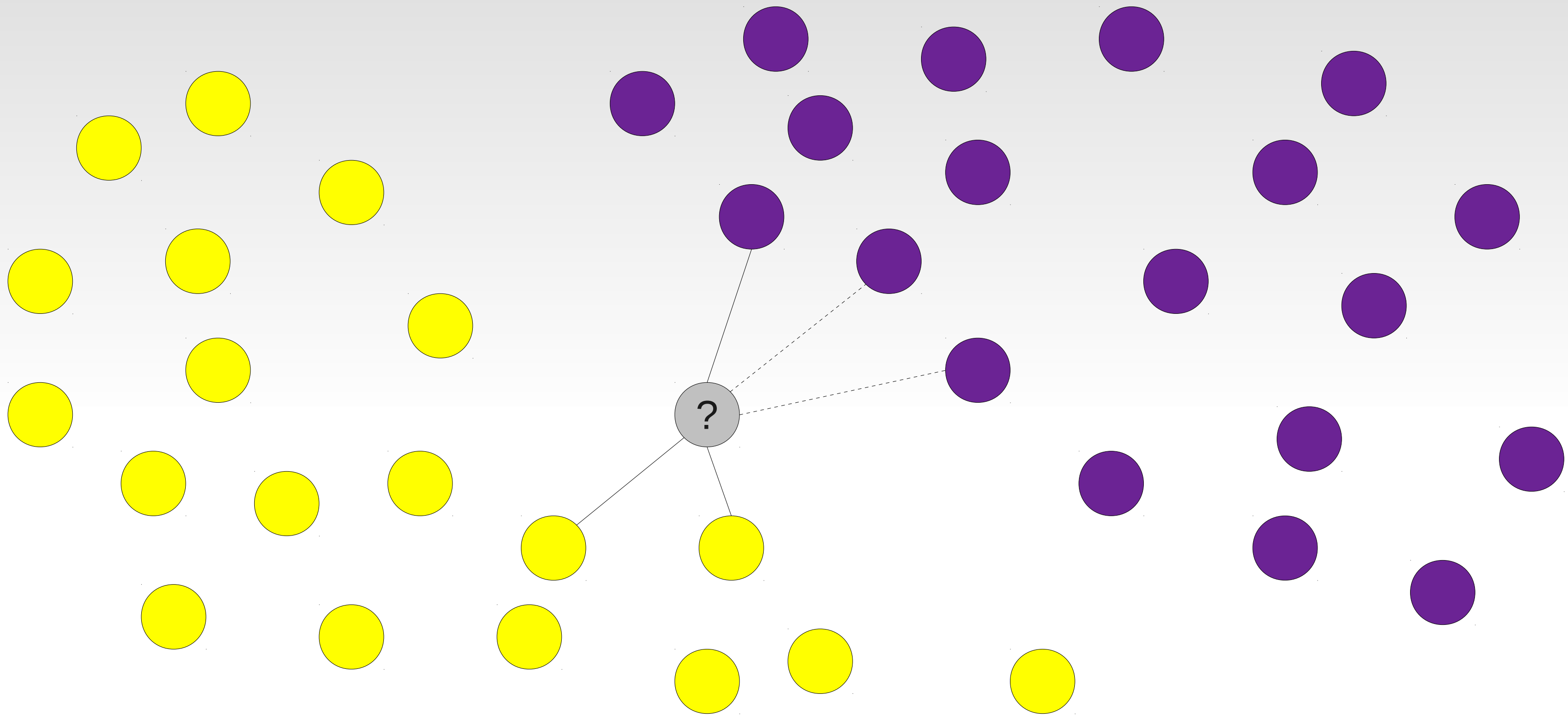
ESP Game, Von Ahn et al.

k-nearest neighbour classifier



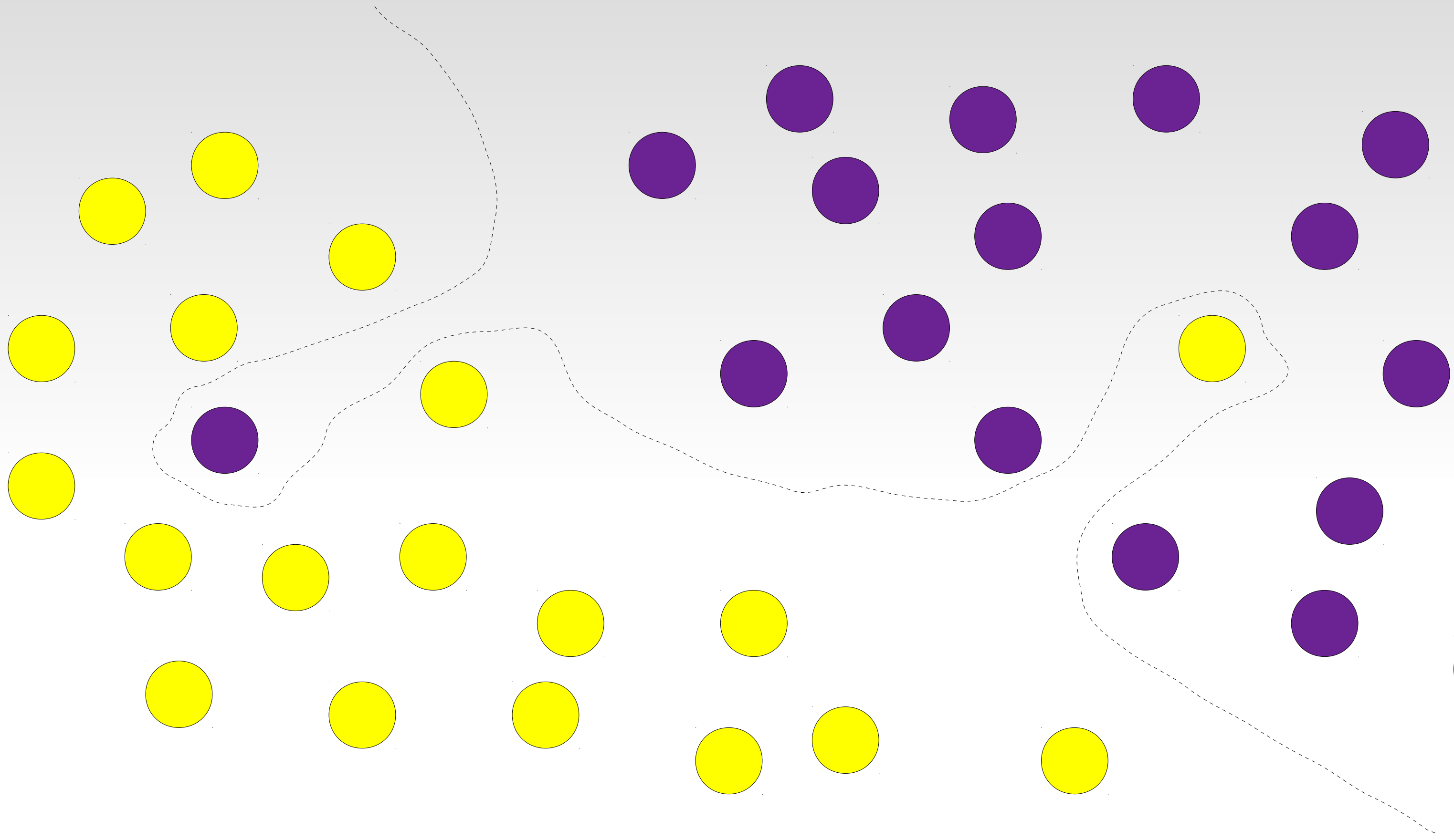
Classify a point by majority vote of its *k* nearest neighbors

k-nearest neighbour classifier



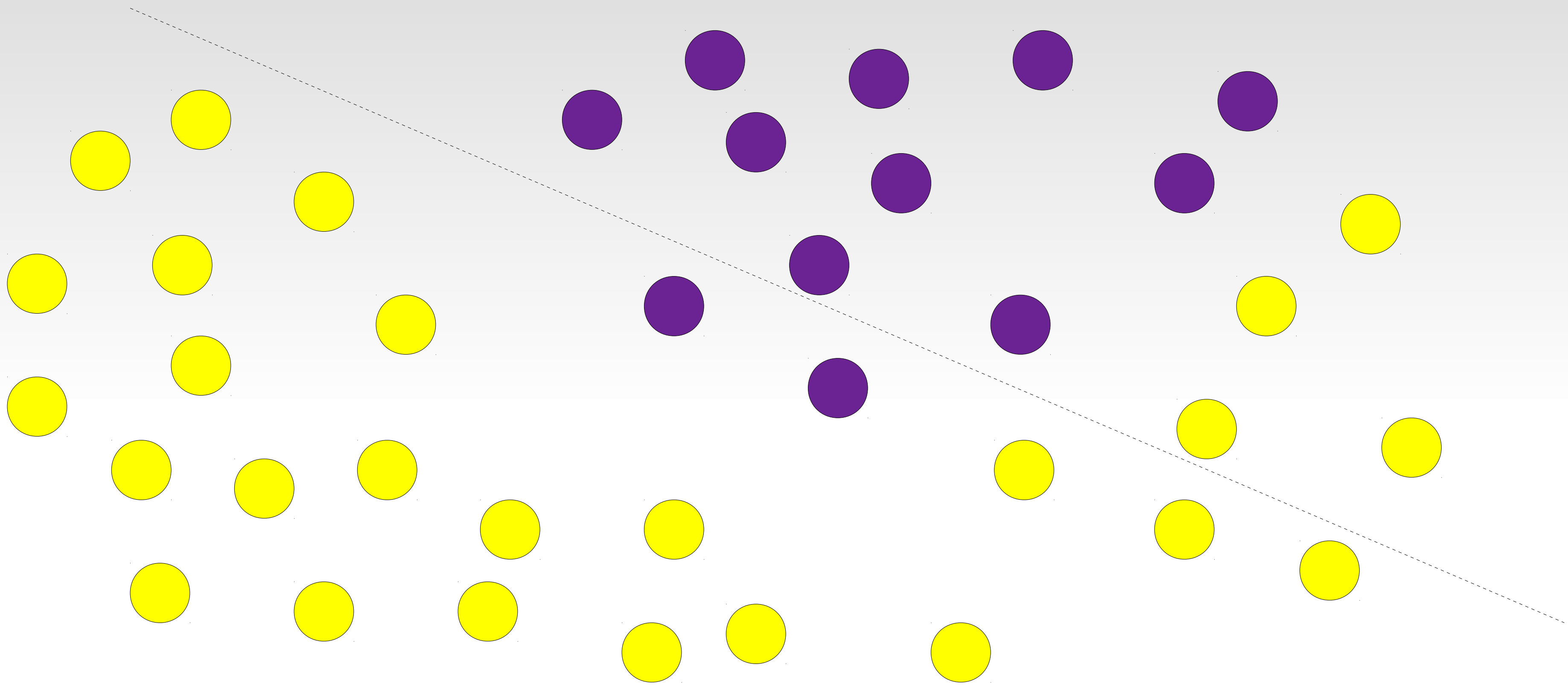
k can make a big difference!

Bias-variance tradeoff



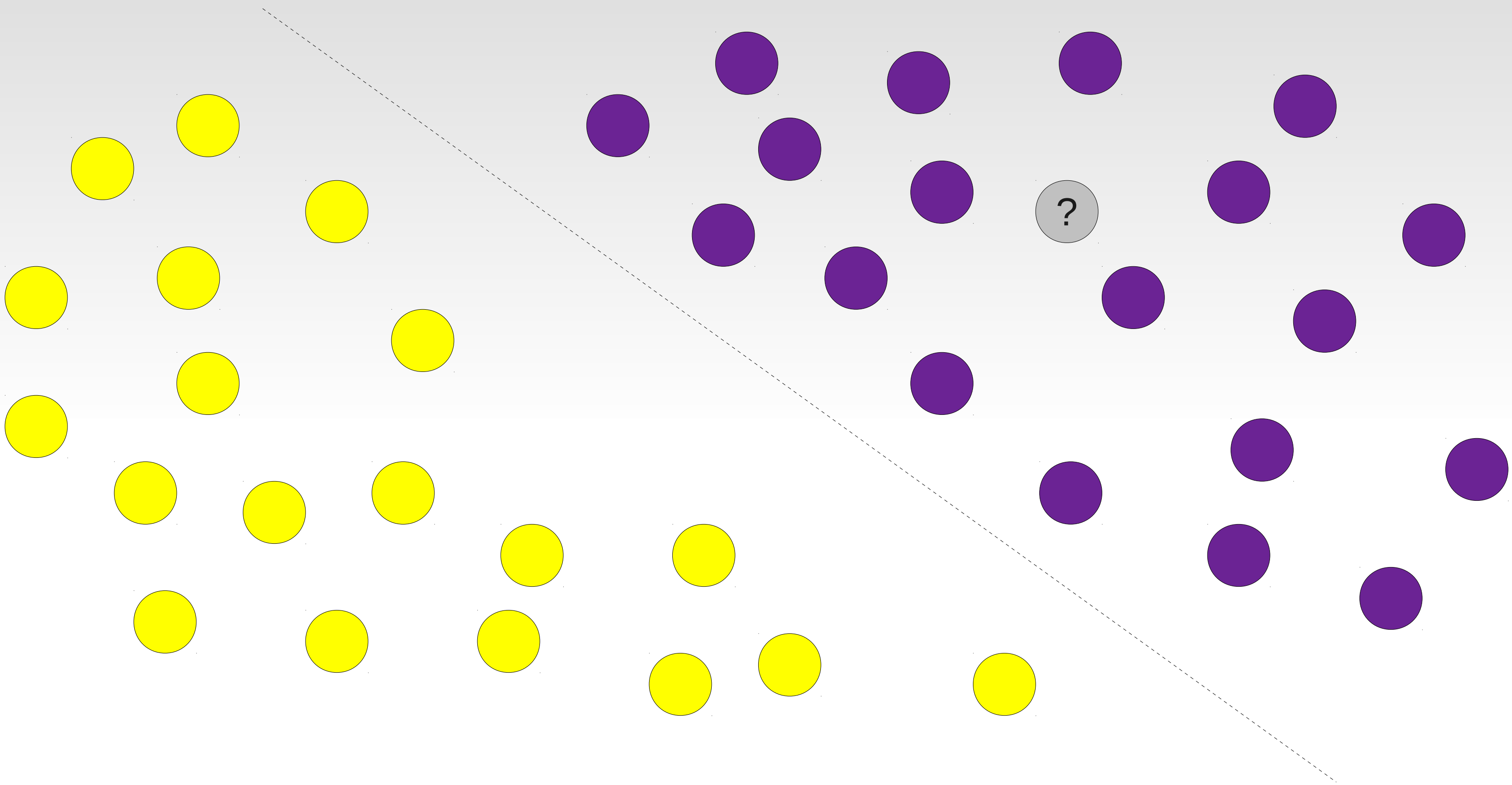
too much variance

Bias-variance tradeoff



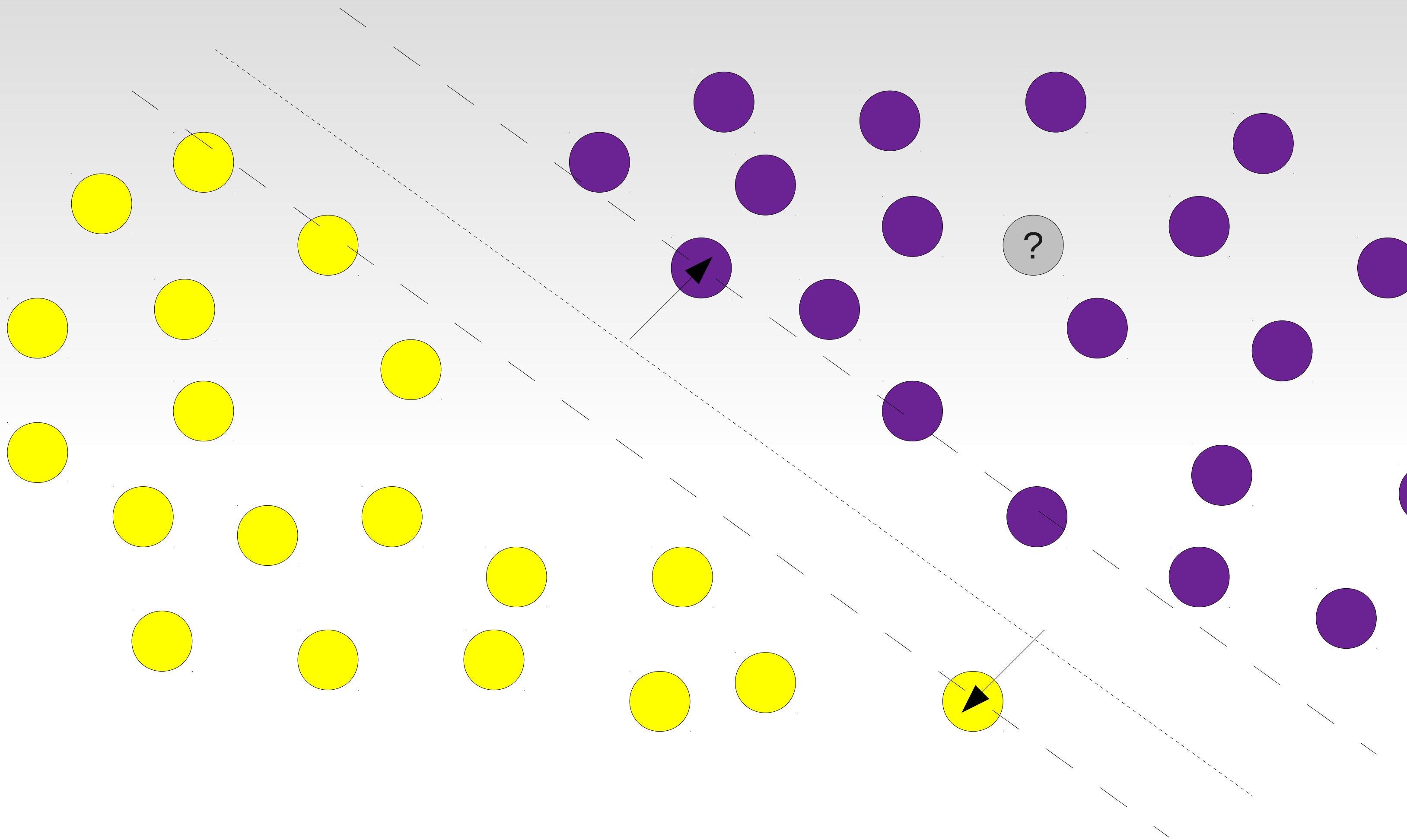
too much bias

Linear classifiers



In high-dimensional space, linear is often best
(If not? Map to a different space and linearize-*kernel machines*)

Support vector machines (SVM)



find hyperplane with maximal distance between any points

- closest points define the plane (support vectors)

SVM example – cat or dog?

Golle 2007

Machine learning for dummies

- many other algorithms for classifying/clustering
  - learn the concepts and what you can tweak
- let others do the hard work
  - libraries: Shogun, libsvm, scikit-learn
  - Apache Mahout: works with Hadoop!
  - outsource to Kaggle...
    - more in the next lecture

Thank you