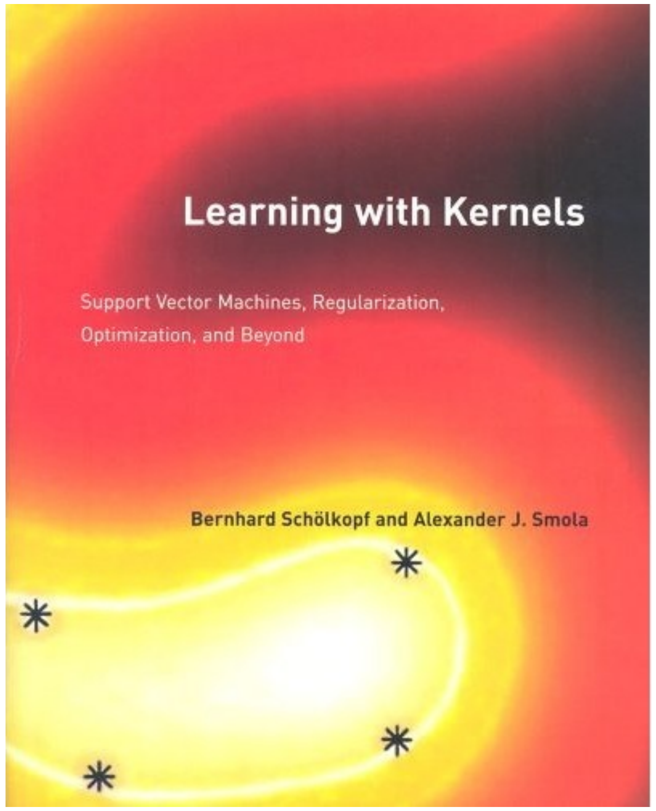
Scalable Machine Learning

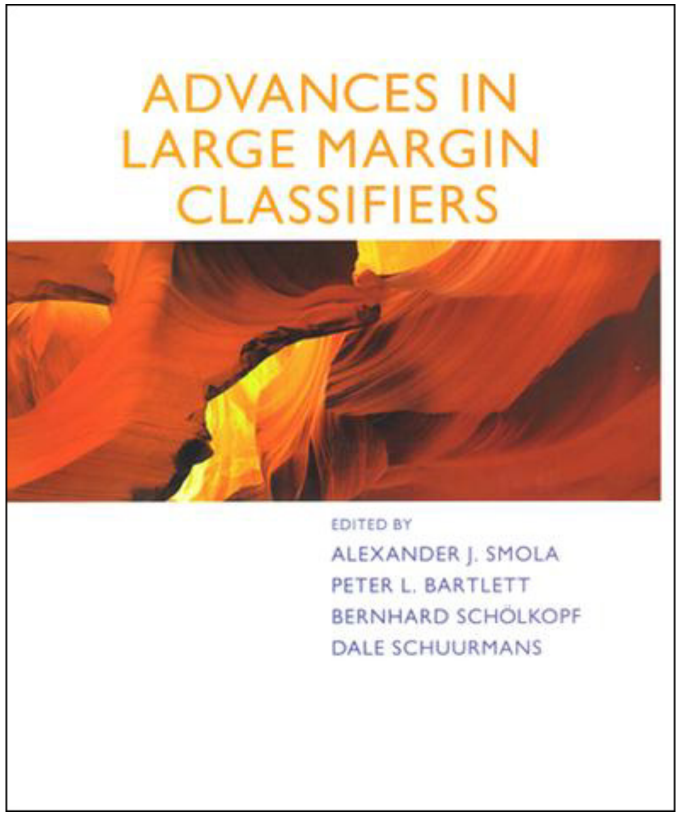
6. Kernels

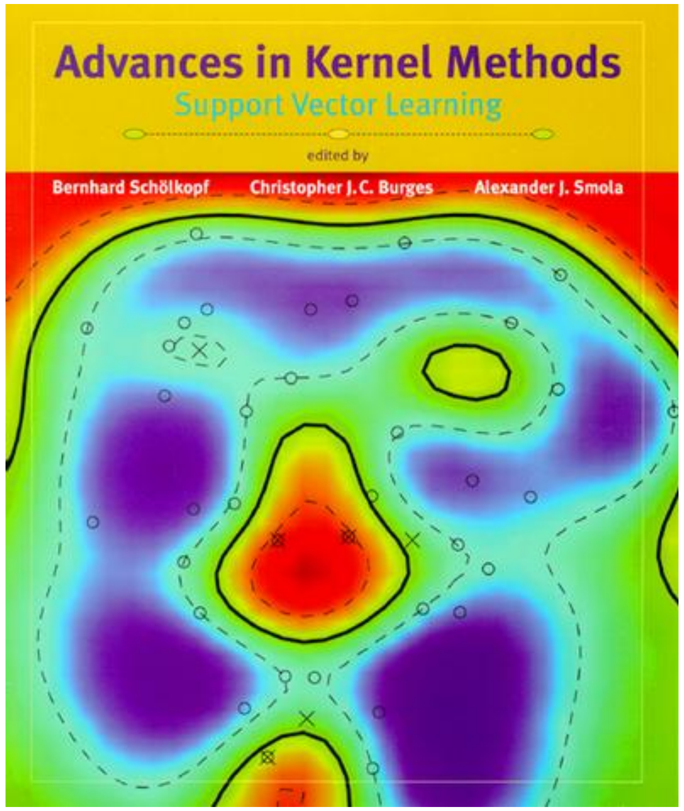
Alex Smola
Yahoo! Research and ANU

http://alex.smola.org/teaching/berkeley2012
Stat 260 SP 12

6. Kernels


Learning with Kernels
Support Vector Machines, Regularization, Optimization, and Beyond
Bernhard Schölkopf and Alexander J. Smola


ADVANCES IN LARGE MARGIN CLASSIFIERS
EDITED BY
ALEXANDER J. SMOLA
PETER L. BARTLETT
BERNHARD SCHÖLKOPF
DALE SCHUURMANS


Advances in Kernel Methods
Support Vector Learning
edited by
Bernhard Schölkopf
Christopher J.C. Burges
Alexander J. Smola

Outline

- Kernels
  - Hilbert Spaces
  - Regularization theory
  - Kernels on strings, sets, graphs, images
- Efficient algorithms
  - Dual space (using α)
  - Reduced dimensionality (low rank expanions)
  - Function space (using fast $K\alpha$)
  - Primal space (hashing & random kitchen sinks)
- Structured estimation
  - Sequence annotation and segmentation
  - Ranking and graph matching
  - Ramp loss, consistency, and invariances

Function classes

Functional Analysis Basics

Microsoft® Visual Basic® for Applications™

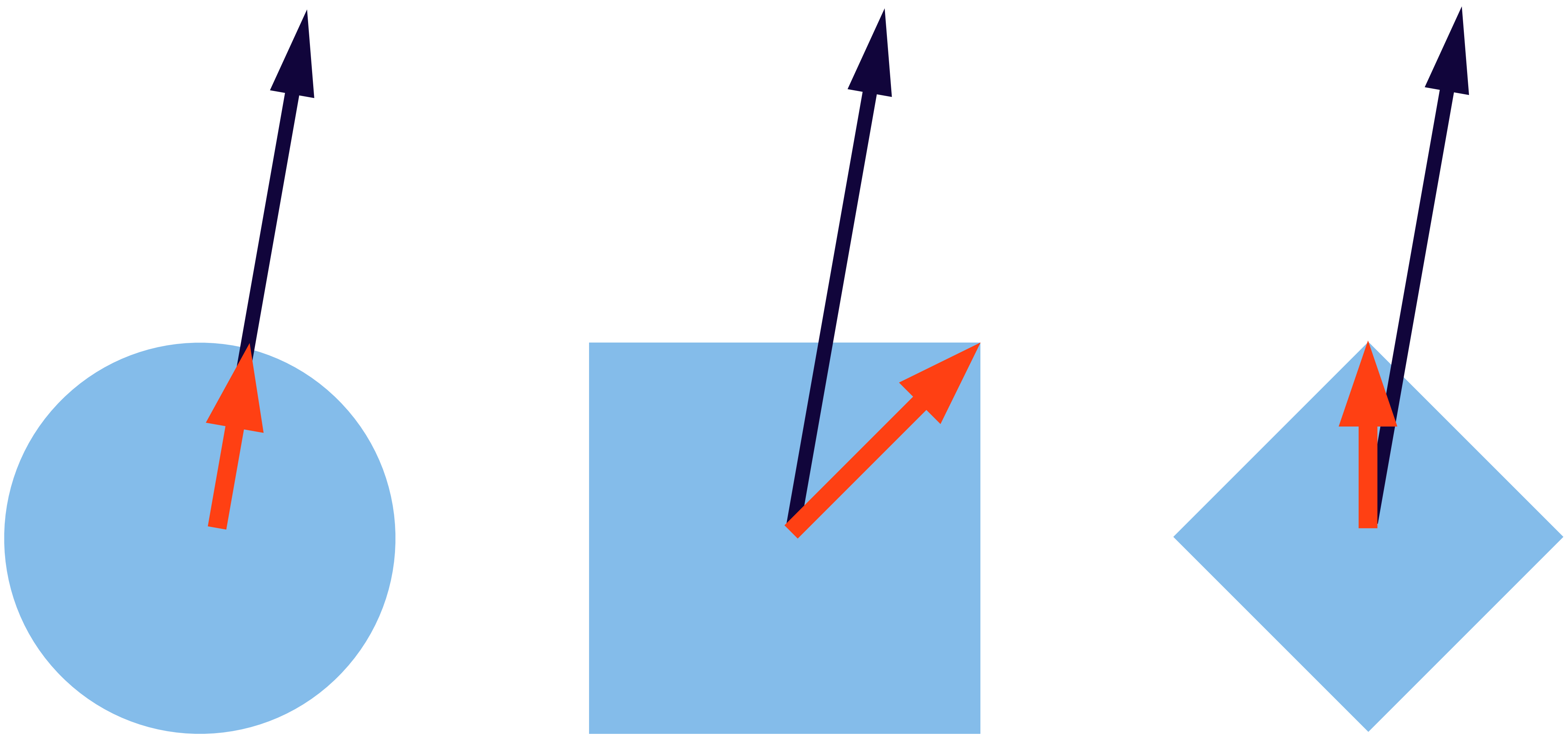
Functional Analysis 101

- Banach space B
  - Normed vector space
  - Linear functions on B induce bilinear forms
    $f(ax + b) = af(x) + f(b)$ and $[af + g](x) = af(x) + g(x)$
    Express as inner products
    $$f(x) =: \langle f, x \rangle$$
- Examples
  - l_1 (absolutely summable series)
  - l_∞ (bounded series)
  - l_2 (square summable series)

Functional Analysis 101

- Dual Norm

$$\|v\| := \sup_{u:\|u\|\leq 1} \langle u, v \rangle$$



Functional Analysis 101

- Operator norm

$$A : \mathcal{B} \to \mathcal{B}' \text{ hence } \|A\| = \sup_{u \in \mathcal{B}, v \in \mathcal{B}'} \langle v, Au \rangle$$

- For Euclidean space this is the largest singular value of the matrix.
- Other norms
  - Trace norm - sum over singular values
  - Frobenius norm - sum over squared singular values

$$\|M\|_{\text{Trace}} = \operatorname{tr} M \text{ for } M \succeq 0 \text{ and } \|M\|_{\text{Frob}} = \left[\operatorname{tr} M M^\top\right]^{\frac{1}{2}}$$

Duality 101

- Fenchel-Legendre dual

$$f^*(v) = \sup_u \langle u, v \rangle - f(u)$$

- Connection to dual norm via indicator function

$$\|v\| = \sup_{u:\|u\| \leq 1} \langle u, v \rangle = \sup_u \langle u, v \rangle - \xi_{U_1}(u)$$

  - Dual norm via dual of characteristic function on unit ball
- Convexity follows via sup over linear functions
- Useful, e.g. for general SVM problems

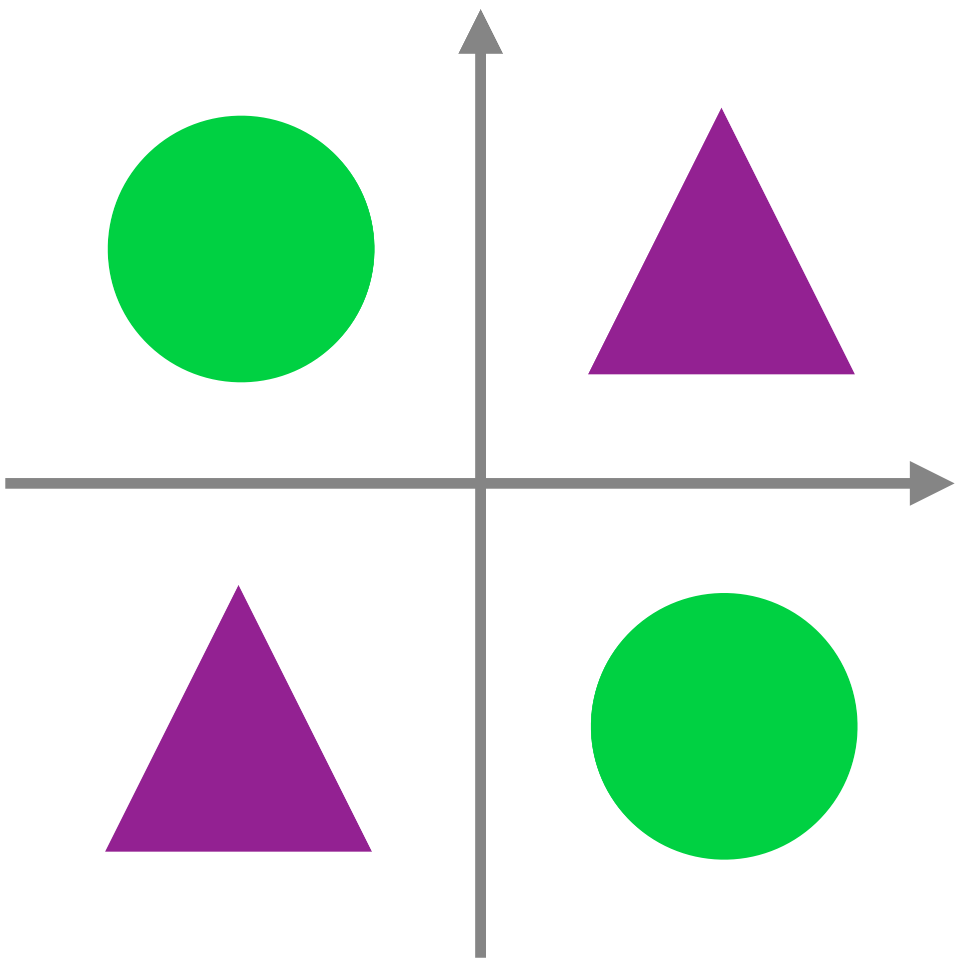
Translation table

vector	function
matrix	operator
vector space	Banach Space (or Hilbert Space)
norm	norm
eigenvalue	eigenvalue
eigenvector	eigenfunction
transpose	adjoint
symmetric matrix	self-adjoint operator
finite dimensional	infinite dimensional

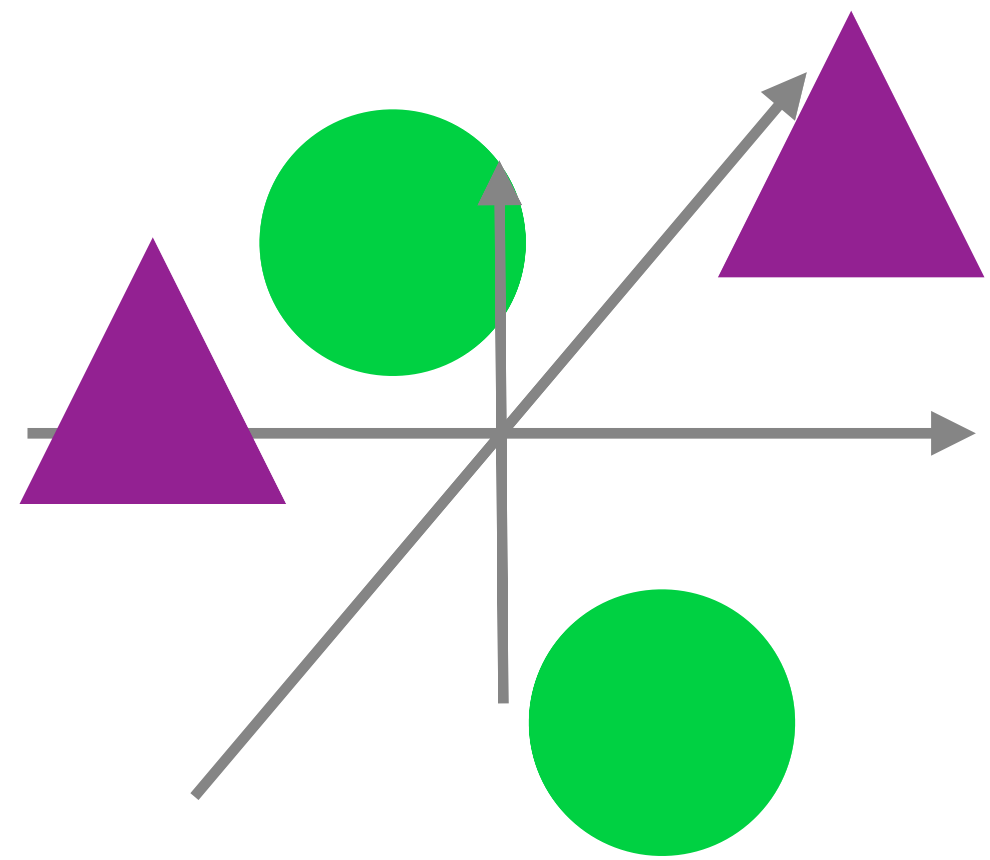
*Terms and conditions apply. Check the theorems.

Kernels

Solving XOR



(x_1, x_2)



$(x_1, x_2, x_1 x_2)$

- XOR not linearly separable
- Mapping into 3 dimensions makes it easily solvable

Kernels vs. Features

Problems

- Need to be an expert in the domain (e.g. Chinese characters).
- Features may not be robust (e.g. postman drops letter in dirt).
- Can be expensive to compute.

Solution

- Use shotgun approach.
- Compute many features and hope a good one is among them.
- Do this efficiently.

Feature Space Mapping

- Naive Nonlinearization Strategy
  - Express data x in terms of features $\phi(x)$
  - Solve problem in feature space
  - Requires explicit feature computation
- Kernel trick
  - Write algorithm in terms of inner products
  - Replace $\langle x, x' \rangle$ by $k(x, x') := \langle \phi(x), \phi(x') \rangle$
  - Works well for dimension-insensitive methods
  - Kernel matrix K is positive semidefinite

Quadratic Kernel

Quadratic Features in $\mathbb{R}^2$

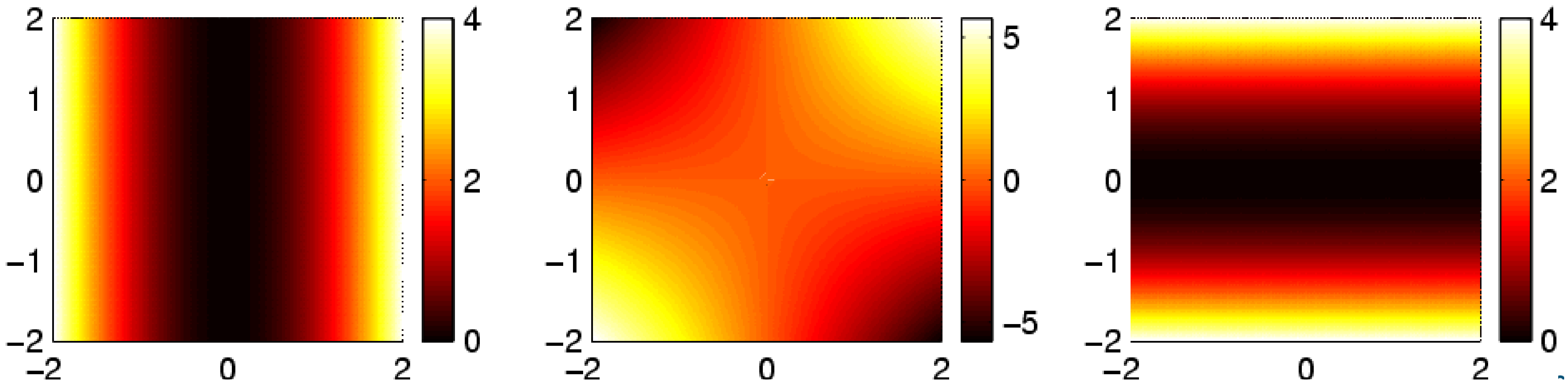
$$\Phi(x) := \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right)$$

Dot Product

$$\begin{aligned}\langle \Phi(x), \Phi(x') \rangle &= \left\langle \left(x_1^2, \sqrt{2}x_1x_2, x_2^2\right), \left({x'_1}^2, \sqrt{2}x'_1x'_2, {x'_2}^2\right) \right\rangle \\ &= \langle x, x' \rangle^2.\end{aligned}$$

Insight

Trick works for any polynomials of order d via $\langle x, x' \rangle^d$.



Computational Efficiency

Problem

- Extracting features can sometimes be very costly.
- Example: second order features in 1000 dimensions. This leads to 5005 numbers. For higher order polynomial features much worse.

Solution

Don't compute the features, try to compute dot products implicitly. For some features this works . . .

Definition

A kernel function $k : \mathfrak{X} \times \mathfrak{X} \to \mathbb{R}$ is a symmetric function in its arguments for which the following property holds

$$k(x, x') = \langle \Phi(x), \Phi(x') \rangle \text{ for some feature map } \Phi.$$

If $k(x, x')$ is much cheaper to compute than $\Phi(x)$ . . .

Polynomial Kernels

Idea

- We want to extend $k(x,x') = \langle x, x' \rangle^2$ to

$$k(x,x') = (\langle x, x' \rangle + c)^d \text{ where } c > 0 \text{ and } d \in \mathbb{N}.$$

- Prove that such a kernel corresponds to a dot product.

Proof strategy

Simple and straightforward: compute the explicit sum given by the kernel, i.e.

$$k(x,x') = (\langle x, x' \rangle + c)^d = \sum_{i=0}^{m} \binom{d}{i} (\langle x, x' \rangle)^i c^{d-i}$$

Individual terms $(\langle x, x' \rangle)^i$ are dot products for some $\Phi_i(x)$.

Kernel Conditions

Computability

We have to be able to compute $k(x, x')$ efficiently (much cheaper than dot products themselves).

"Nice and Useful" Functions

The features themselves have to be useful for the learning problem at hand. Quite often this means smooth functions.

Symmetry

Obviously $k(x, x') = k(x', x)$ due to the symmetry of the dot product $\langle \Phi(x), \Phi(x') \rangle = \langle \Phi(x'), \Phi(x) \rangle$.

Dot Product in Feature Space

Is there always a Φ such that k really is a dot product?

Mercer's Theorem

The Theorem

For any symmetric function $k : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ which is square integrable in $\mathcal{X} \times \mathcal{X}$ and which satisfies

$$\int_{\mathcal{X} \times \mathcal{X}} k(x, x') f(x) f(x') dx dx' \geq 0 \text{ for all } f \in L_2(\mathcal{X})$$

there exist $\phi_i : \mathcal{X} \to \mathbb{R}$ and numbers $\lambda_i \geq 0$ where

$$k(x, x') = \sum_i \lambda_i \phi_i(x) \phi_i(x') \text{ for all } x, x' \in \mathcal{X}.$$

Interpretation

Double integral is the continuous version of a vector-matrix-vector multiplication. For positive semidefinite matrices we have

$$\sum \sum k(x_i, x_j) \alpha_i \alpha_j \geq 0$$

Properties

Distance in Feature Space

Distance between points in feature space via

$$\begin{aligned} d(x,x')^2 :=& \|\Phi(x) - \Phi(x')\|^2 \\ =& \langle \Phi(x), \Phi(x) \rangle - 2\langle \Phi(x), \Phi(x') \rangle + \langle \Phi(x'), \Phi(x') \rangle \\ =& k(x,x) + k(x',x') - 2k(x,x) \end{aligned}$$

Kernel Matrix

To compare observations we compute dot products, so we study the matrix K given by

$$K_{ij} = \langle \Phi(x_i), \Phi(x_j) \rangle = k(x_i, x_j)$$

where x_i are the training patterns.

Similarity Measure

The entries K_{ij} tell us the overlap between $\Phi(x_i)$ and $\Phi(x_j)$, so $k(x_i, x_j)$ is a similarity measure.

Properties

K is Positive Semidefinite

Claim: $\alpha^\top K \alpha \geq 0$ for all $\alpha \in \mathbb{R}^m$ and all kernel matrices $K \in \mathbb{R}^{m \times m}$. Proof:

$$\begin{aligned}\sum_{i,j}^{m} \alpha_i \alpha_j K_{ij} &= \sum_{i,j}^{m} \alpha_i \alpha_j \langle \Phi(x_i), \Phi(x_j) \rangle \\ &= \left\langle \sum_{i}^{m} \alpha_i \Phi(x_i), \sum_{j}^{m} \alpha_j \Phi(x_j) \right\rangle = \left\| \sum_{i=1}^{m} \alpha_i \Phi(x_i) \right\|^2\end{aligned}$$

Kernel Expansion

If w is given by a linear combination of $\Phi(x_i)$ we get

$$\langle w, \Phi(x) \rangle = \left\langle \sum_{i=1}^{m} \alpha_i \Phi(x_i), \Phi(x) \right\rangle = \sum_{i=1}^{m} \alpha_i k(x_i, x).$$

A Counterexample

A Candidate for a Kernel

$$k(x, x') = \begin{cases} 1 & \text{if } \|x - x'\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

This is symmetric and gives us some information about the proximity of points, yet it is not a proper kernel . . .

Kernel Matrix

We use three points, $x_1 = 1, x_2 = 2, x_3 = 3$ and compute the resulting "kernelmatrix" K. This yields

$$K = \begin{bmatrix} 1 & 1 & 0 \\ 1 & 1 & 1 \\ 0 & 1 & 1 \end{bmatrix}$$ and eigenvalues $(\sqrt{2}-1)^{-1}, 1$ and $(1-\sqrt{2})$.

as eigensystem. Hence k is not a kernel.

Examples

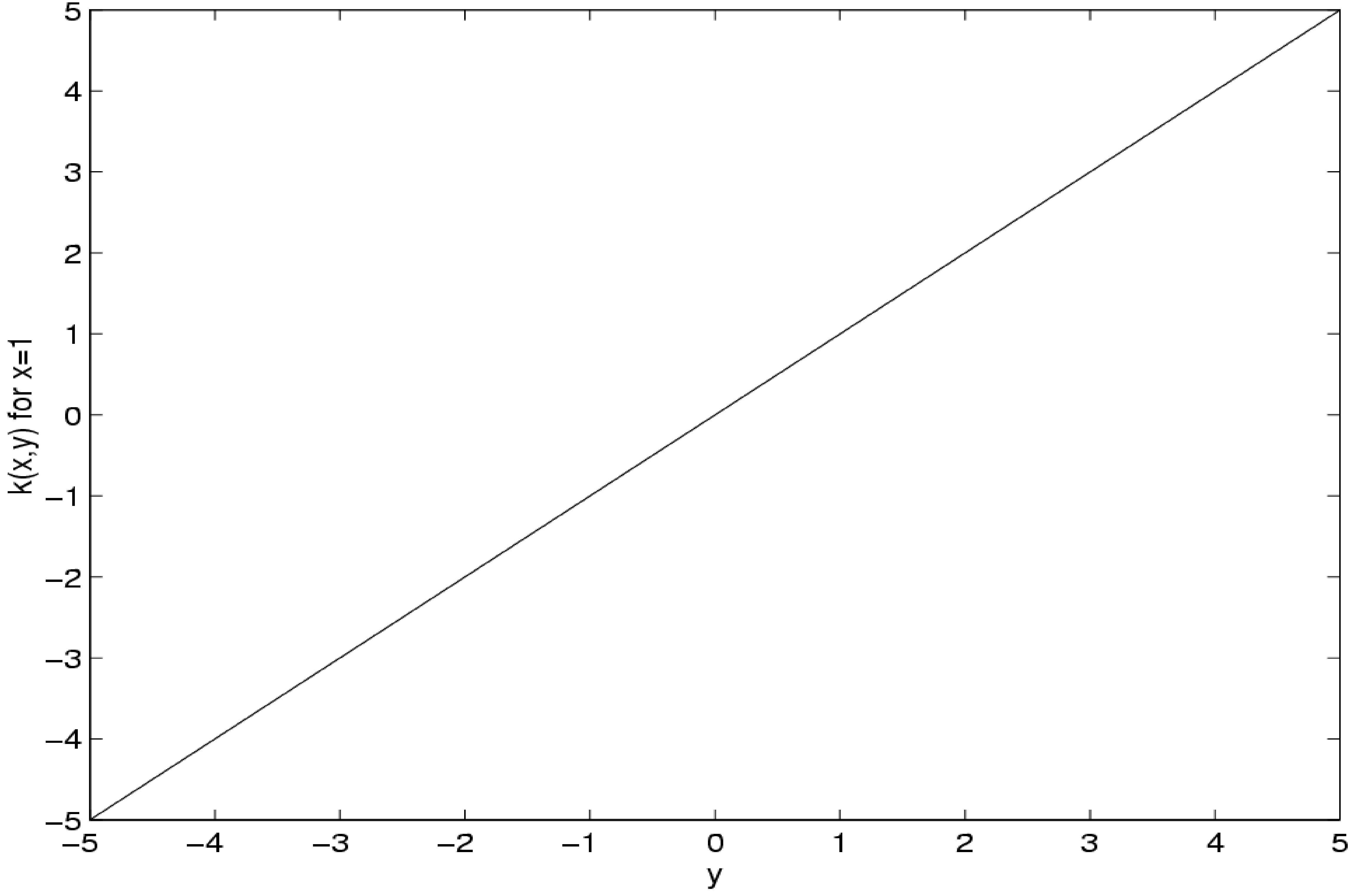
Examples of kernels $k(x, x')$

Linear	$\langle x, x' \rangle$
Laplacian RBF	$\exp\left(-\lambda \|x - x'\|\right)$
Gaussian RBF	$\exp\left(-\lambda \|x - x'\|^2\right)$
Polynomial	$\left(\langle x, x' \rangle + c \rangle\right)^d, c \geq 0, \; d \in \mathbb{N}$
B-Spline	$B_{2n+1}(x - x')$
Cond. Expectation	$\mathbf{E}_c[p(x\|c)p(x'\|c)]$

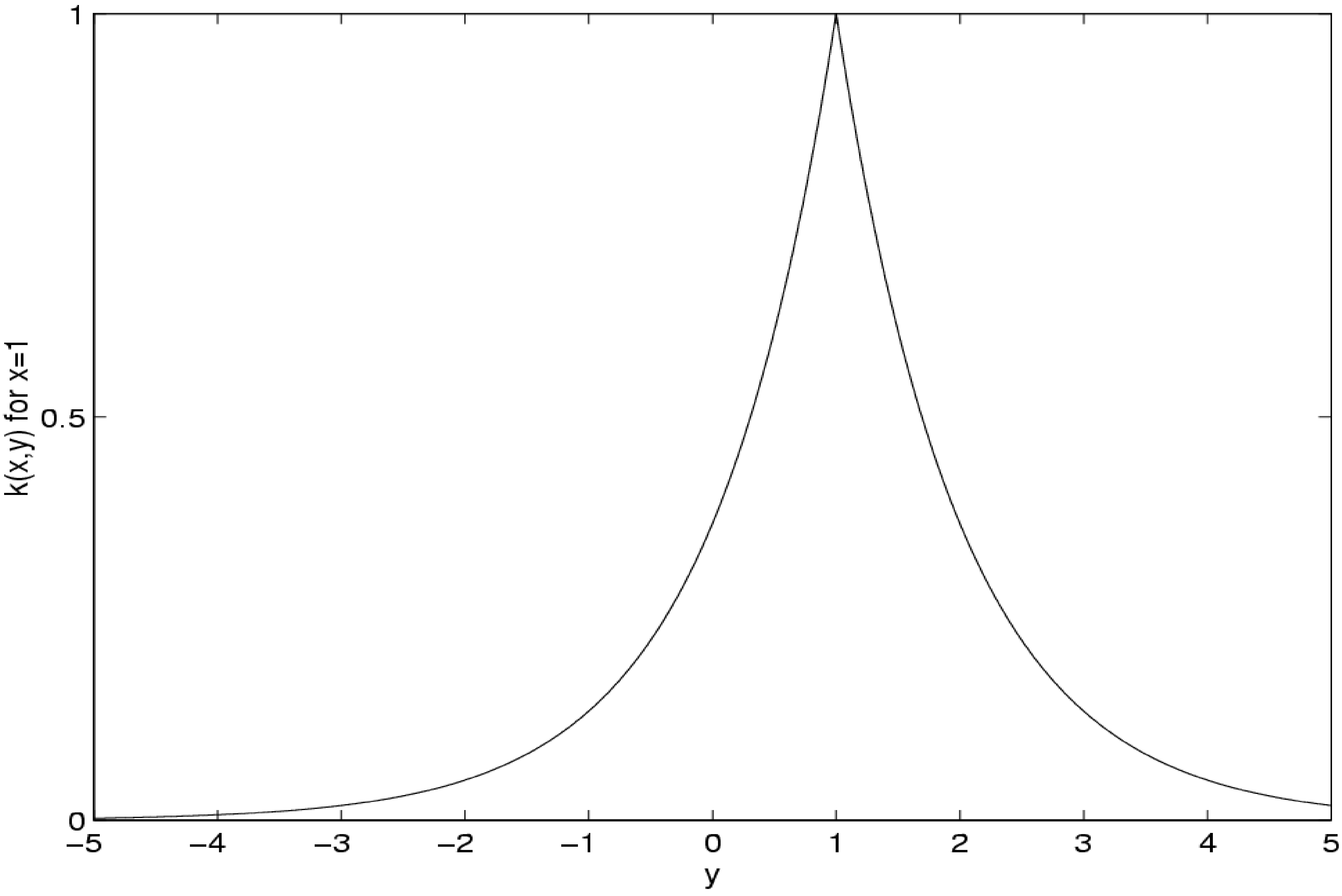
Simple trick for checking Mercer's condition

Compute the Fourier transform of the kernel and check that it is nonnegative.

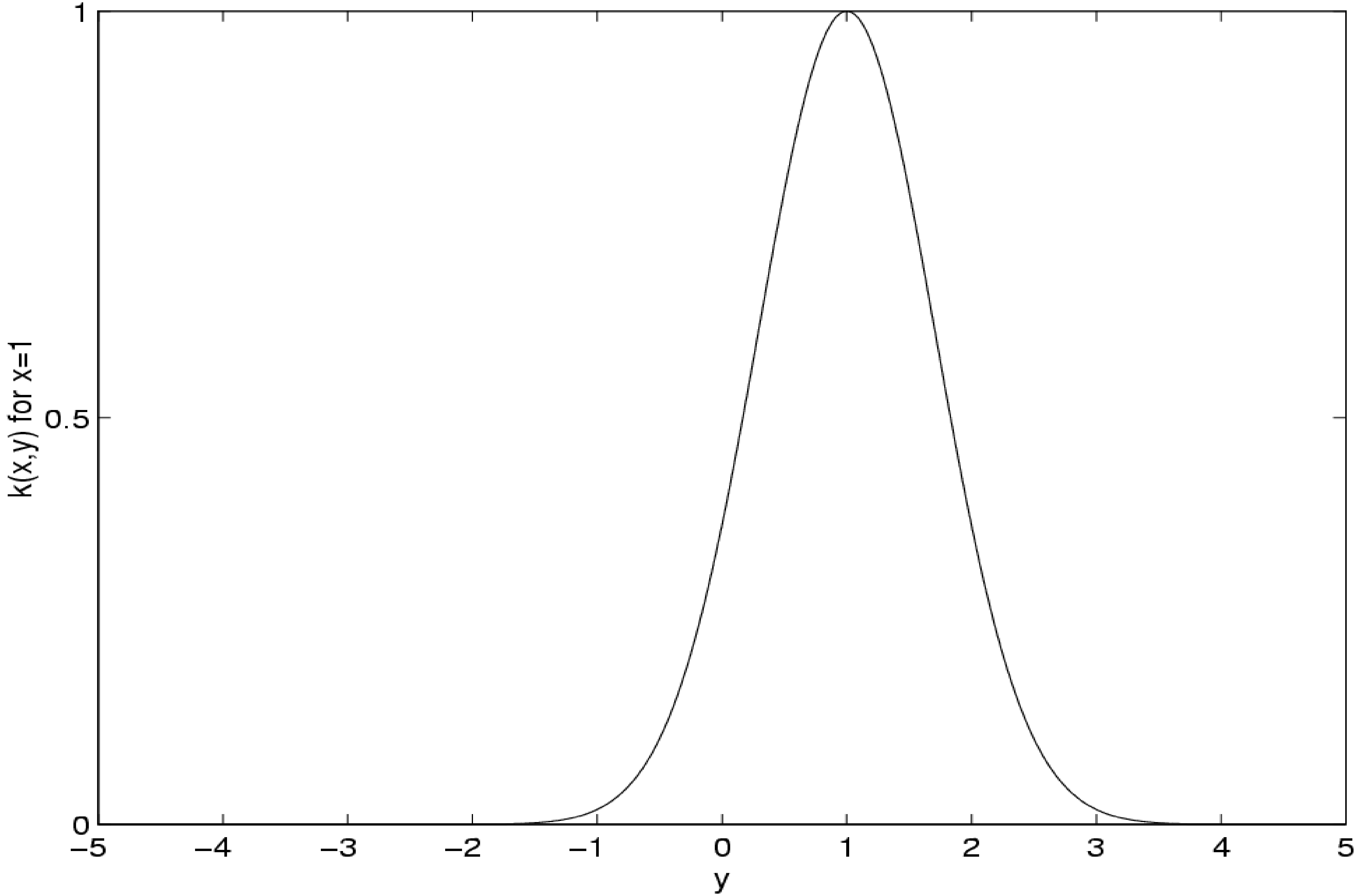
Linear Kernel



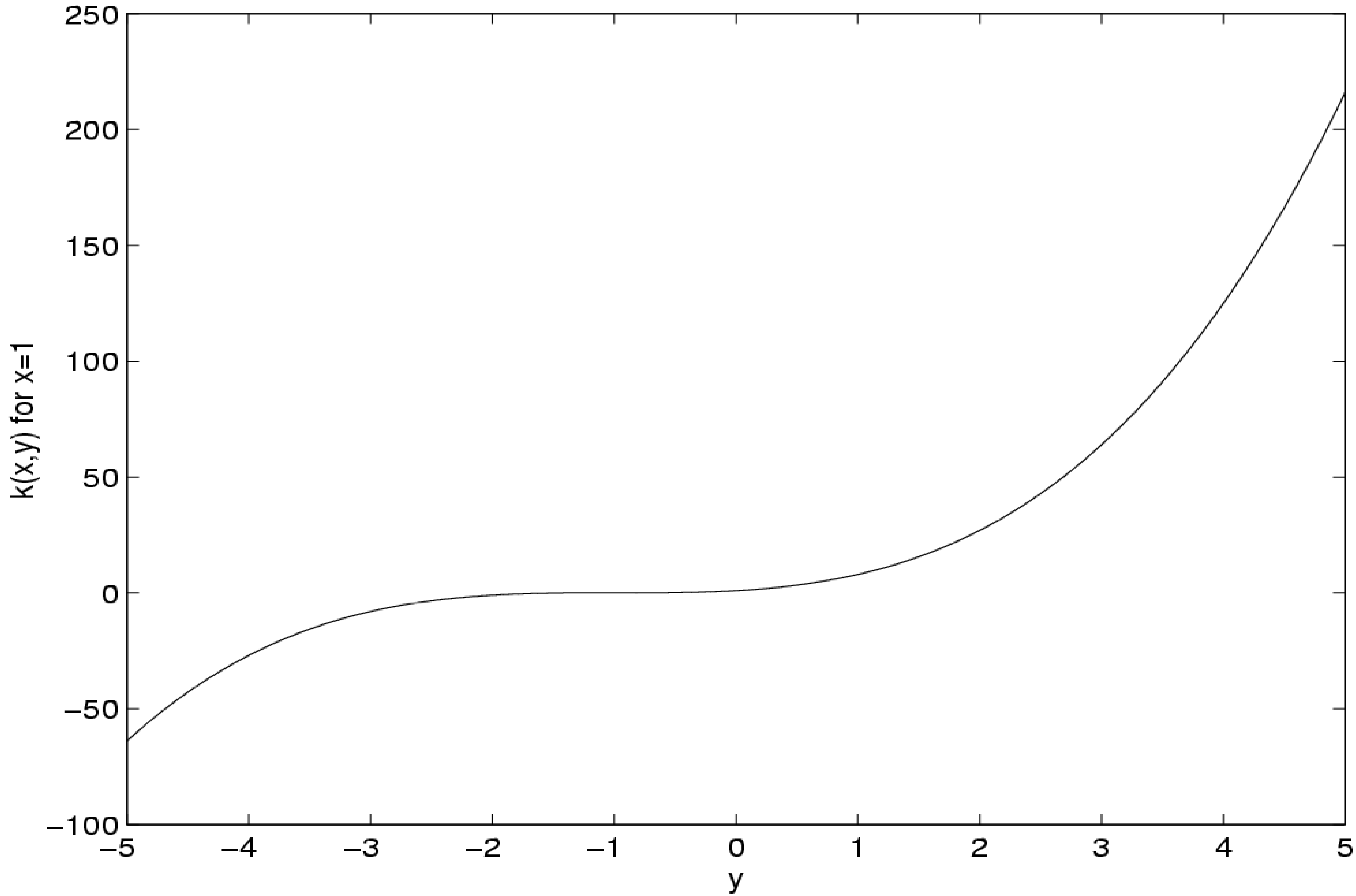
Laplacian Kernel



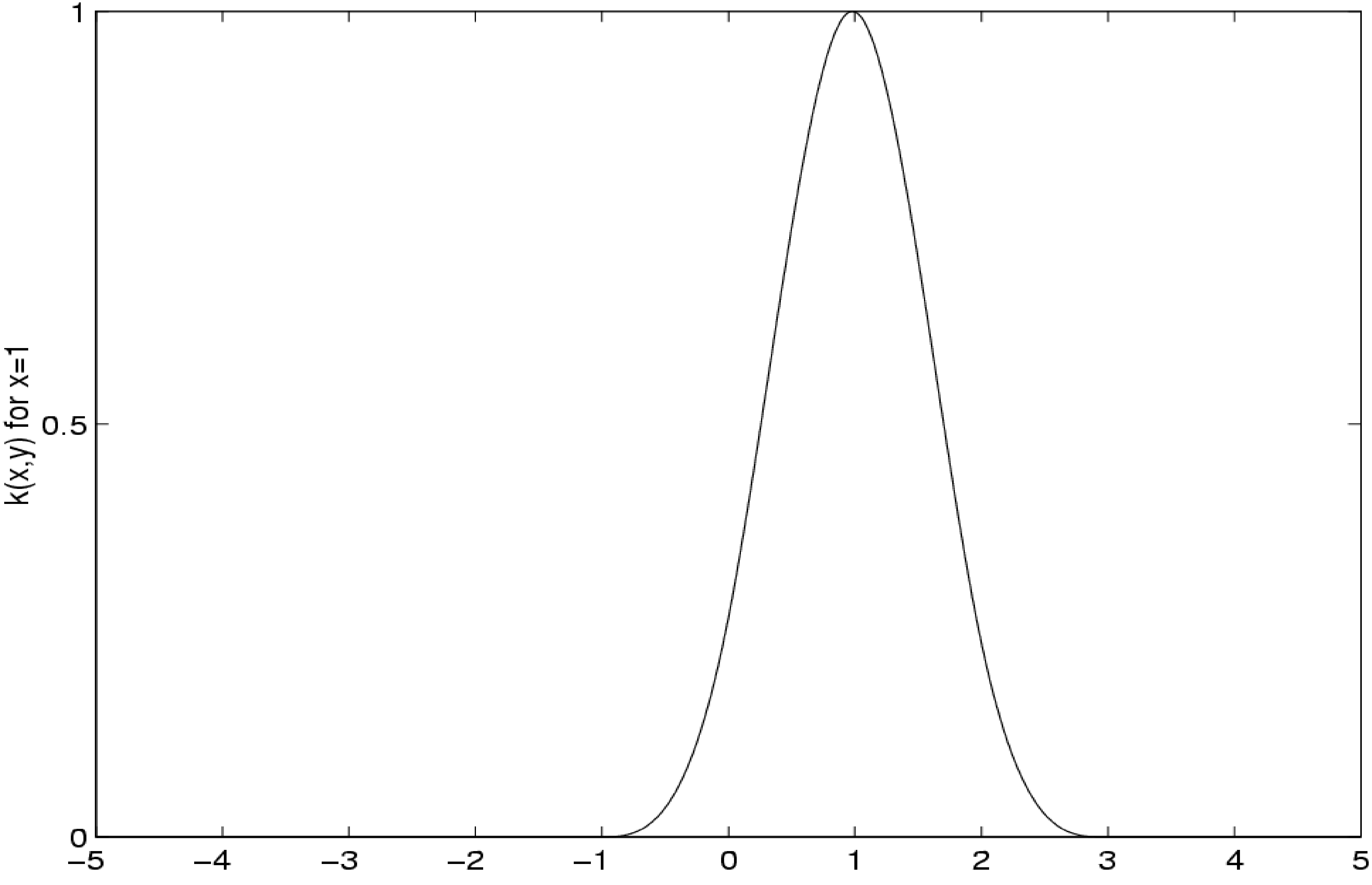
Gaussian Kernel



Polynomial of order 3



B_3 Spline Kernel



Mini Summary

Features

- Prior knowledge, expert knowledge
- Shotgun approach (polynomial features)
- Kernel trick $k(x, x') = \langle \phi(x), \phi(x') \rangle$
- Mercer's theorem

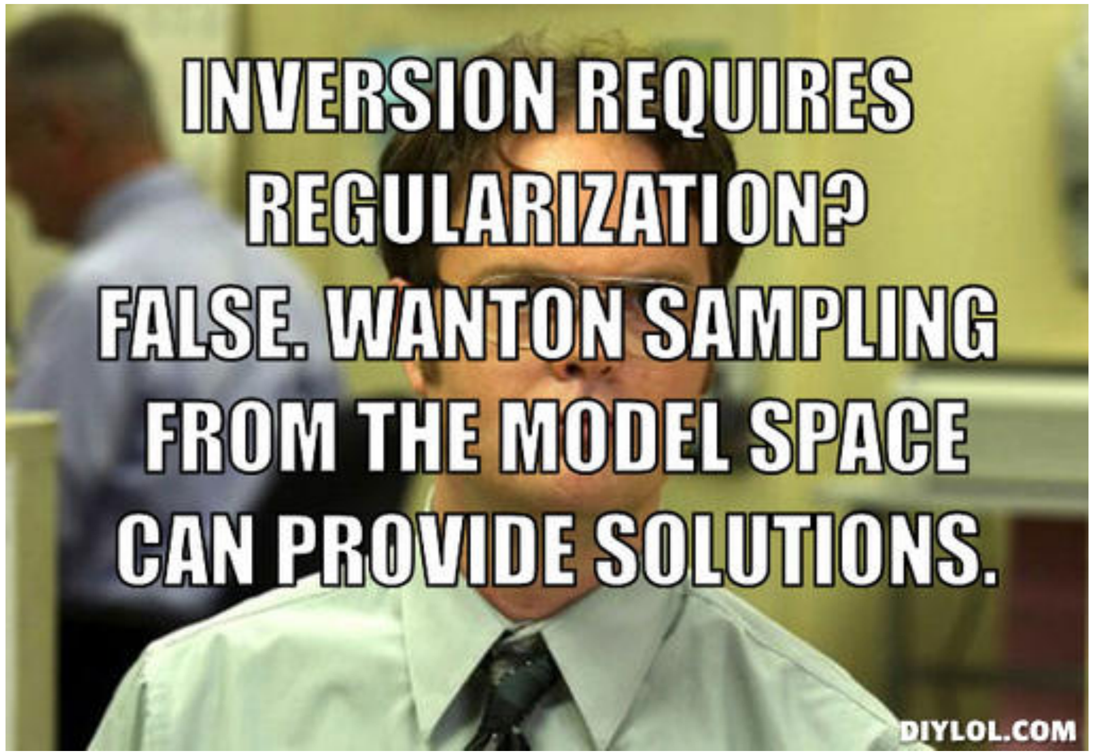
Applications

- Kernel Perceptron
- Nonlinear algorithm automatically by query-replace

Examples of Kernels

- Gaussian RBF
- Polynomial kernels

Regularization



Problems with Kernels

Myth

Support Vectors work because they map data into a high-dimensional feature space.

And your statistician (Bellmann) told you ...

The higher the dimensionality, the more data you need

Example: Density Estimation

Assuming data in $[0, 1]^m$, 1000 observations in $[0, 1]$ give you on average 100 instances per bin (using binsize 0.1^m) but only $\frac{1}{100}$ instances in $[0, 1]^5$.

Worrying Fact

Some kernels map into an **infinite**-dimensional space, e.g., $k(x, x') = \exp(-\frac{1}{2\sigma^2}\|x - x'\|^2)$

Encouraging Fact

SVMs work well in practice ...

Solving the Mystery

The Truth is in the Margins

Maybe the maximum margin requirement is what saves us when finding a classifier, i.e., we minimize $\|w\|^2$.

Risk Functional

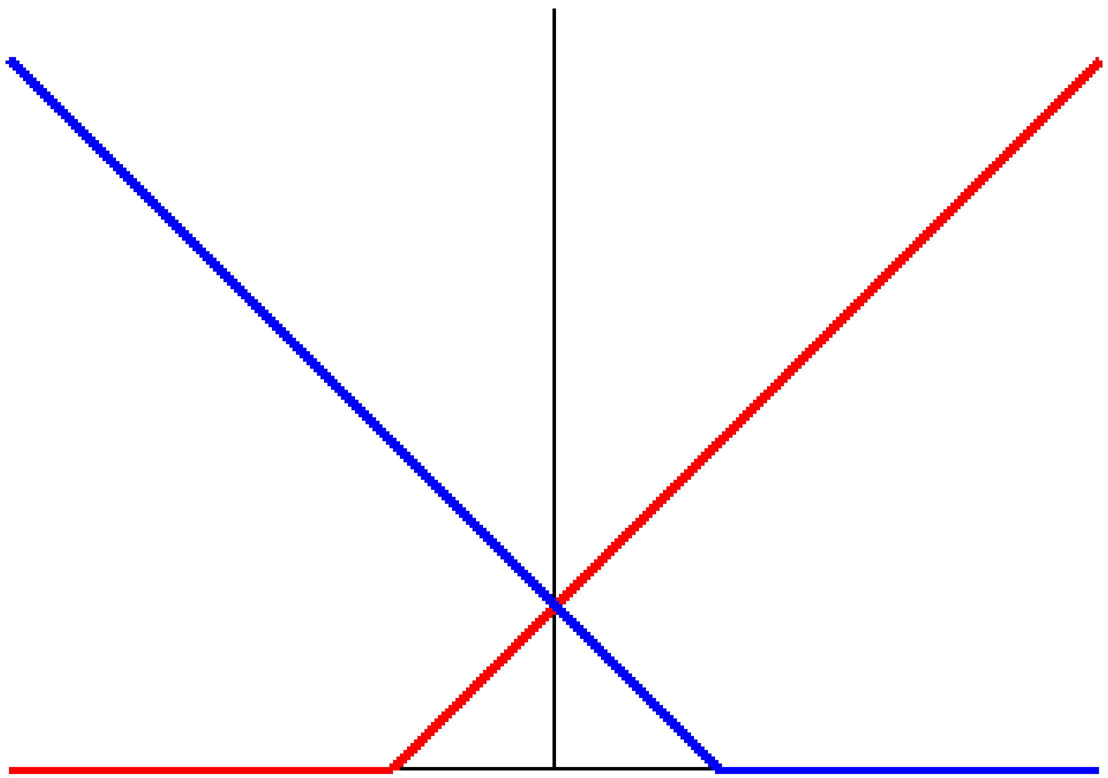
Rewrite the optimization problems in a unified form

$$R_{\mathrm{reg}}[f] = \sum_{i=1}^{m} c(x_i, y_i, f(x_i)) + \Omega[f]$$

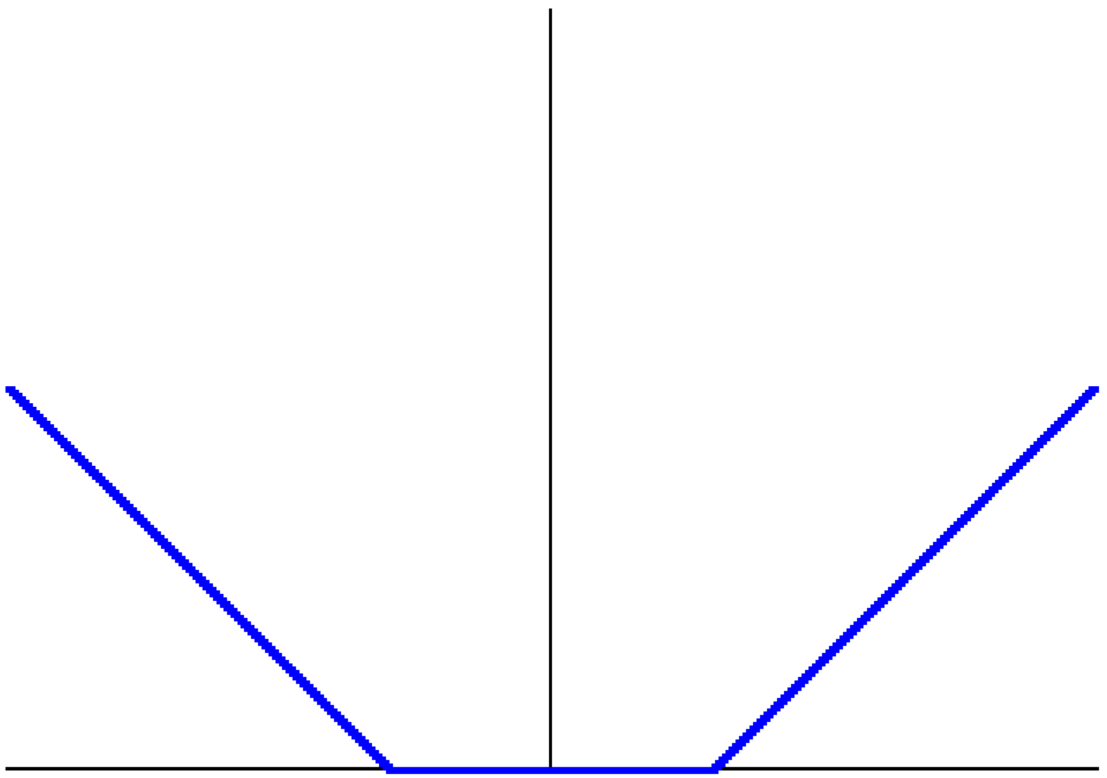
$c(x, y, f(x))$ is a loss function and $\Omega[f]$ is a regularizer.

- $\Omega[f] = \frac{\lambda}{2}\|w\|^2$ for linear functions.
- For classification $c(x, y, f(x)) = \max(0, 1 - yf(x))$.
- For regression $c(x, y, f(x)) = \max(0, |y - f(x)| - \epsilon)$.

Typical SVM loss



Soft Margin Loss



ε-insensitive Loss

Soft Margin Loss

Original Optimization Problem

$$\underset{w,\xi}{\text{minimize}} \quad \frac{1}{2}\|w\|^2 + C\sum_{i=1}^{m}\xi_i$$

subject to $y_i f(x_i) \geq 1 - \xi_i$ and $\xi_i \geq 0$ for all $1 \leq i \leq m$

Regularization Functional

$$\underset{w}{\text{minimize}} \quad \frac{\lambda}{2}\|w\|^2 + \sum_{i=1}^{m}\max(0, 1 - y_i f(x_i))$$

- For fixed f, clearly $\xi_i \geq \max(0, 1 - y_i f(x_i))$.
- For $\xi > \max(0, 1 - y_i f(x_i))$ we can decrease it such that the bound is matched and improve the objective function.
- Both methods are equivalent.

Why Regularization?

What we really wanted ...

Find some $f(x)$ such that the expected loss $\mathbf{E}[c(x, y, f(x))]$ is small.

What we ended up doing ...

Find some $f(x)$ such that the empirical average of the expected loss $\mathbf{E}_{\mathrm{emp}}[c(x, y, f(x))]$ is small.

$$\mathbf{E}_{\mathrm{emp}}[c(x, y, f(x))] = \frac{1}{m} \sum_{i=1}^{m} c(x_i, y_i, f(x_i))$$

However, just minimizing the empirical average does not guarantee anything for the expected loss (overfitting).

Safeguard against overfitting

We need to constrain the class of functions $f \in \mathfrak{F}$ somehow. Adding $\Omega[f]$ as a penalty does exactly that.

Some regularization ideas

Small Derivatives

We want to have a function f which is smooth on the entire domain. In this case we could use

$$\Omega[f] = \int_X \|\partial_x f(x)\|^2 \, dx = \langle \partial_x f, \partial_x f \rangle.$$

Small Function Values

If we have no further knowledge about the domain X, minimizing $\|f\|^2$ might be sensible, i.e.,

$$\Omega[f] = \|f\|^2 = \langle f, f \rangle.$$

Splines

Here we want to find f such that both $\|f\|^2$ and $\|\partial_x^2 f\|^2$ are small. Hence we can minimize

$$\Omega[f] = \|f\|^2 + \|\partial_x^2 f\|^2 = \langle (f, \partial_x^2 f), (f, \partial_x^2 f) \rangle$$

Regularization

Regularization Operators

We map f into some Pf, which is small for desirable f and large otherwise, and minimize

$$\Omega[f] = \|Pf\|^2 = \langle Pf, Pf \rangle.$$

For all previous examples we can find such a P.

Function Expansion for Regularization Operator

Using a linear function expansion of f in terms of some f_i, that is for $f(x) = \sum_i \alpha_i f_i(x)$ we can compute

$$\Omega[f] = \left\langle P \sum_i \alpha_i f_i(x), P \sum_j \alpha_j f_i(x) \right\rangle = \sum_{i,j} \alpha_i \alpha_j \langle P f_i, P f_j \rangle.$$

Regularization and Kernels

Regularization for $\Omega[f] = \frac{1}{2}\|w\|^2$

$$w = \sum_i \alpha_i \Phi(x_i) \Longrightarrow \|w\|^2 = \sum_{i,j} \alpha_i \alpha_j k(x_i, x_j)$$

This looks very similar to $\langle Pf_i, Pf_j \rangle$.

Key Idea

So if we could find a P and k such that

$$k(x, x') = \langle Pk(x, \cdot), Pk(x', \cdot) \rangle$$

we could show that using a kernel means that we are minimizing the empirical risk plus a regularization term.

Solution: Greens Functions

A sufficient condition is that k is the Greens Function of P^*P, that is $\langle P^*Pk(x, \cdot), f(\cdot) \rangle = f(x)$.

One can show that this is necessary and sufficient.

Building Kernels

Kernels from Regularization Operators:

Given an operator P^*P, we can find k by solving the self consistency equation

$$\langle Pk(x,\cdot), Pk(x',\cdot)\rangle = k^\top(x,\cdot)(P^*P)k(x',\cdot) = k(x,x')$$

and take f to be the span of all $k(x,\cdot)$.

So we can find k for a given measure of smoothness.

Regularization Operators from Kernels:

Given a kernel k, we can find some P^*P for which the self consistency equation is satisfied.

So we can find a measure of smoothness for a given k.

Spectrum and Kernels

Effective Function Class

Keeping $\Omega[f]$ small means that $f(x)$ cannot take on arbitrary function values. Hence we study the function class

$$\mathfrak{F}_C = \left\{ f \,\middle|\, \frac{1}{2}\langle Pf, Pf \rangle \leq C \right\}$$

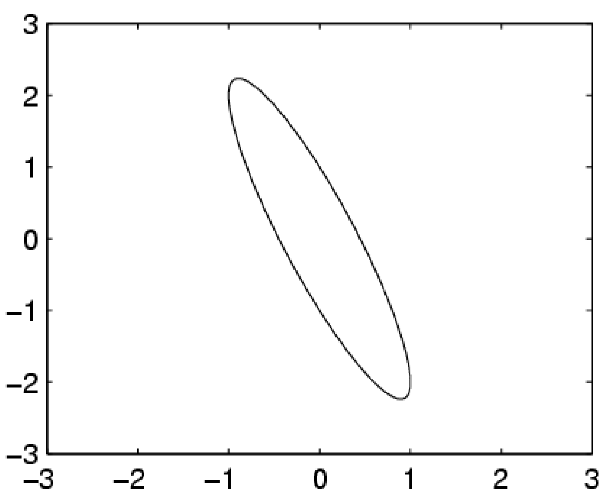
Example

For $f = \sum_i \alpha_i k(x_i, x)$ this implies $\frac{1}{2}\alpha^\top K \alpha \leq C$.

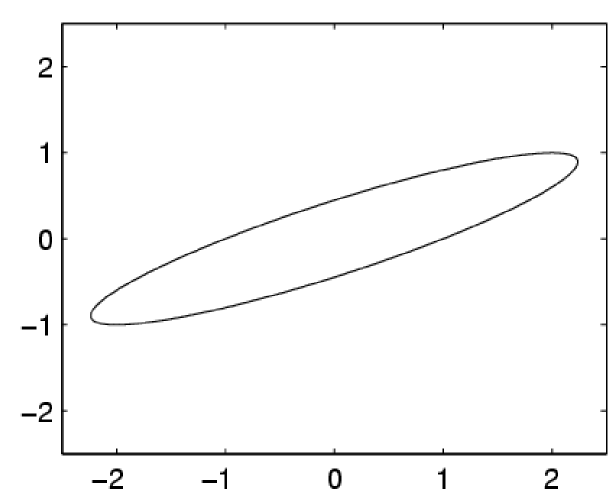
Kernel Matrix

$$K = \begin{bmatrix} 5 & 2 \\ 2 & 1 \end{bmatrix}$$

Coefficients



Function Values



Fourier Regularization

Goal

Find measure of smoothness that depends on the frequency properties of f and not on the position of f.

A Hint: Rewriting $\|f\|^2 + \|\partial_x f\|^2$

Notation: $\tilde{f}(\omega)$ is the Fourier transform of f.

$$\begin{aligned}
\|f\|^2 + \|\partial_x f\|^2 &= \int |f(x)|^2 + |\partial_x f(x)|^2 dx \\
&= \int |\tilde{f}(\omega)|^2 + \omega^2 |\tilde{f}(\omega)|^2 d\omega \\
&= \int \frac{|\tilde{f}(\omega)|^2}{p(\omega)} d\omega \text{ where } p(\omega) = \frac{1}{1+\omega^2}.
\end{aligned}$$

Idea

Generalize to arbitrary $p(\omega)$, i.e. $\Omega[f] := \frac{1}{2} \int \frac{|\hat{f}(\omega)|^2}{p(\omega)} d\omega$

Greens Function

Theorem

For regularization functionals $\Omega[f] := \frac{1}{2}\int \frac{|\hat{f}(\omega)|^2}{p(\omega)} d\omega$ the self-consistency condition

$$\langle Pk(x,\cdot), Pk(x',\cdot)\rangle = k^\top(x,\cdot)(P^*P)k(x',\cdot) = k(x,x')$$

is satisfied if k has $p(\omega)$ as its Fourier transform, i.e.,

$$k(x,x') = \int \exp(-i\langle\omega,(x-x')\rangle)p(\omega)d\omega$$

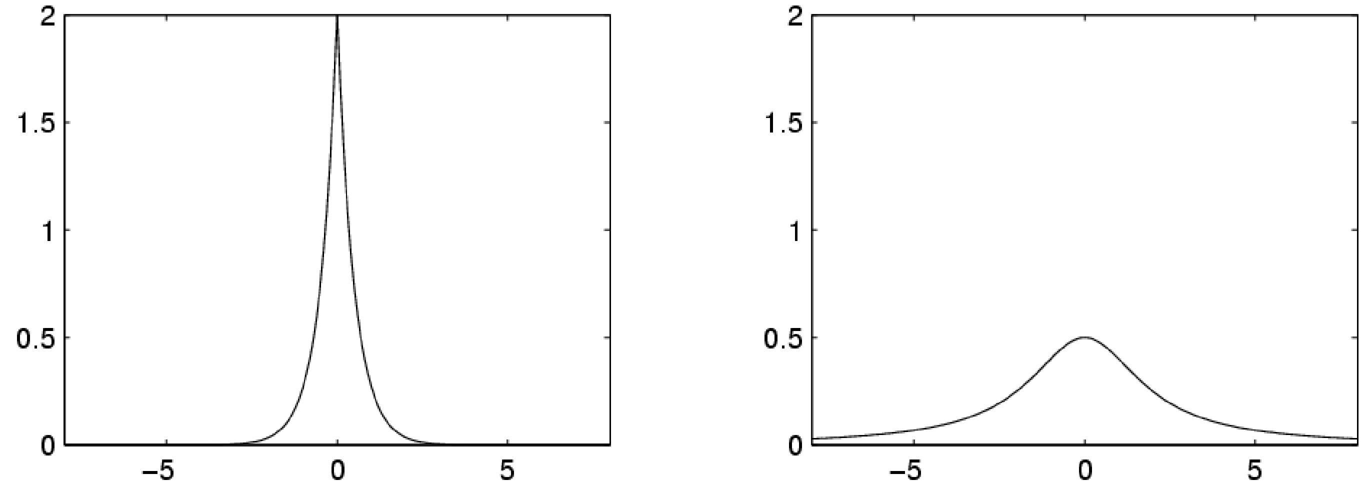
Consequences

- small $p(\omega)$ correspond to high penalty (regularization).
- $\Omega[f]$ is translation invariant, that is $\Omega[f(\cdot)] = \Omega[f(\cdot - x)]$.

Examples

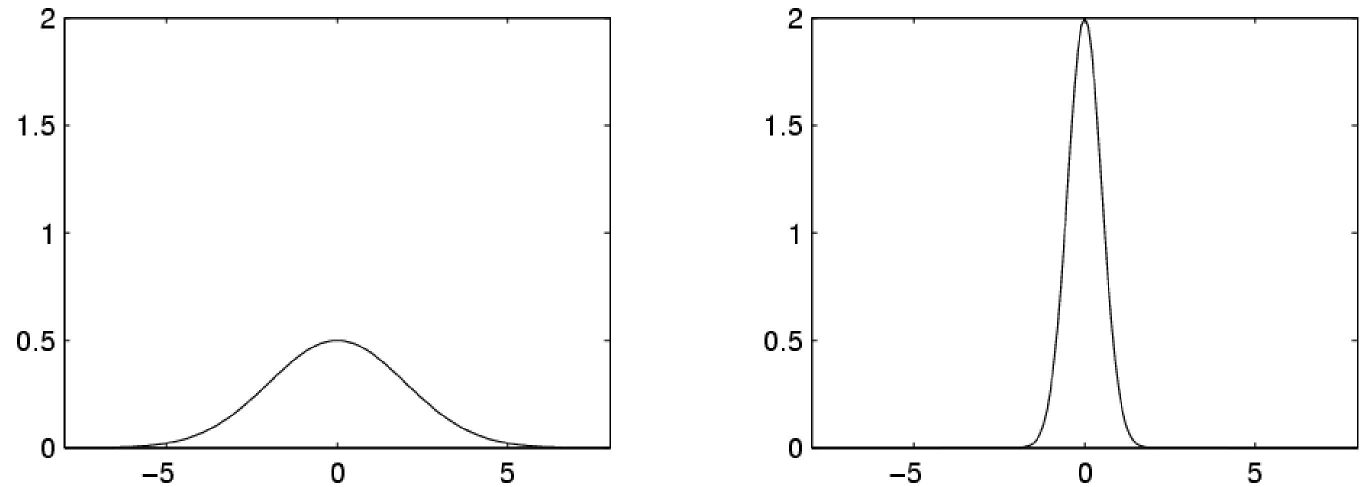
Laplacian Kernel

$$k(x, x') = \exp(-\|x - x'\|)$$
$$p(\omega) \propto (1 + \|\omega\|^2)^{-1}$$



Gaussian Kernel

$$k(x, x') = e^{-\frac{1}{2}\sigma^{-2}\|x - x'\|^2}$$
$$p(\omega) \propto e^{-\frac{1}{2}\sigma^{2}\|\omega\|^2}$$



Fourier transform of k shows regularization properties.
The more rapidly $p(\omega)$ decays, the more high frequencies are filtered out.

Rules of thumb

- Fourier transform is sufficient to check whether $k(x, x')$ satisfies Mercer's condition: only check if $\tilde{k}(\omega) \geq 0$.
- Example: $k(x, x') = \mathrm{sinc}(x - x')$.
  $\tilde{k}(\omega) = \chi_{[-\pi,\pi]}(\omega)$, hence k is a proper kernel.
- Width of kernel often more important than type of kernel (short range decay properties matter).
- Convenient way of incorporating prior knowledge, e.g.: for speech data we could use the autocorrelation function.
- Sum of derivatives becomes polynomial in Fourier space.

Polynomial Kernels

Functional Form

$$k(x, x') = \kappa(\langle x, x' \rangle)$$

Series Expansion

Polynomial kernels admit an expansion in terms of Legendre polynomials (L_n^N: order n in $\mathbb{R}^N$).

$$k(x, x') = \sum_{n=0}^{\infty} b_n L_n(\langle x, x' \rangle)$$

Consequence:

L_n (and their rotations) form an orthonormal basis on the unit sphere, P^*P is rotation invariant, and P^*P is diagonal with respect to L_n. In other words

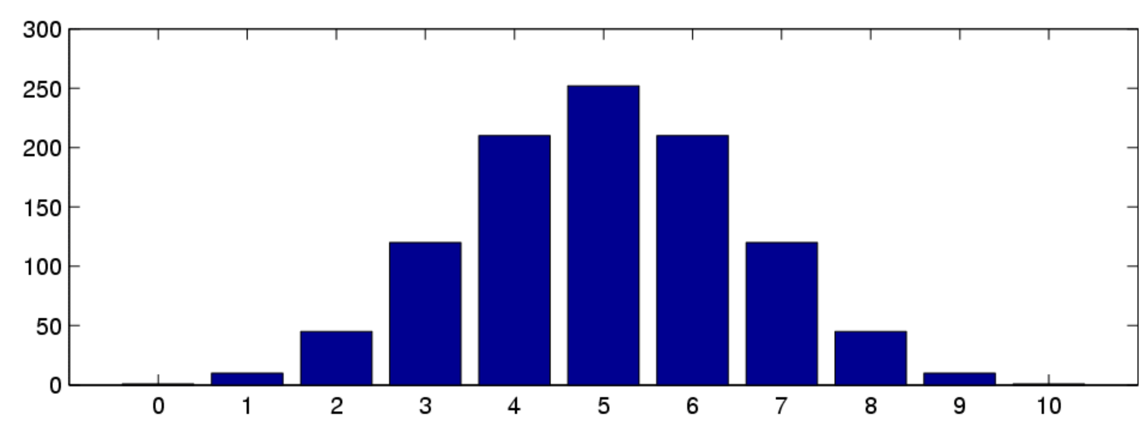
$$(P^*P)L_n(\langle x, \cdot \rangle) = b_n^{-1} L_n(\langle x, \cdot \rangle)$$

Polynomial Kernels

- Decay properties of b_n determine smoothness of functions specified by $k(\langle x, x' \rangle)$.
- For $N \to \infty$ all terms of L_n^N but x^n vanish, hence a Taylor series $k(x, x') = \sum_i a_i \langle x, x' \rangle^i$ gives a good guess.

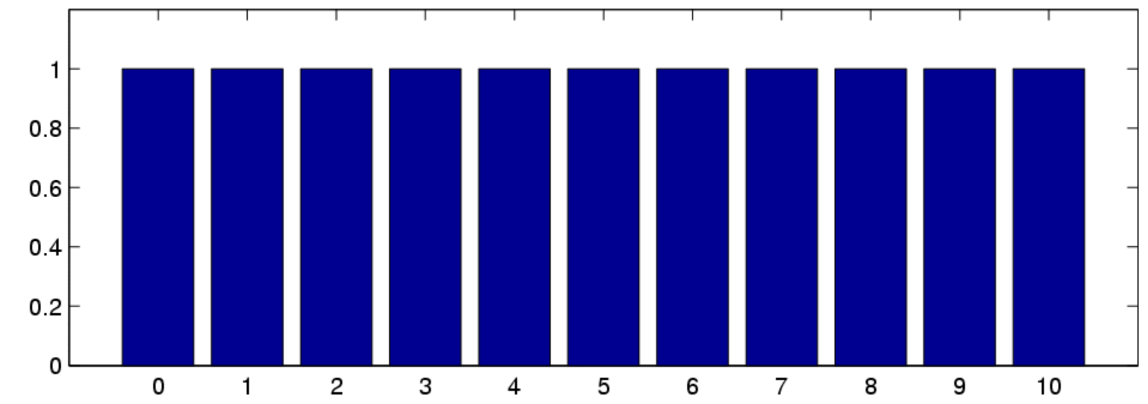
Inhomogeneous Polynomial

$$\begin{aligned} k(x, x') &= (\langle x, x' \rangle + 1)^p \\ a_n &= \binom{p}{n} \text{ if } n \leq p \end{aligned}$$



Vovk's Real Polynomial

$$\begin{aligned} k(x, x') &= \frac{1 - \langle x, x' \rangle^p}{1 - (\langle x, x' \rangle)} \\ a_n &= 1 \text{ if } n < p \end{aligned}$$



Mini Summary

Regularized Risk Functional

- From Optimization Problems to Loss Functions
- Regularization
- Safeguard against Overfitting

Regularization and Kernels

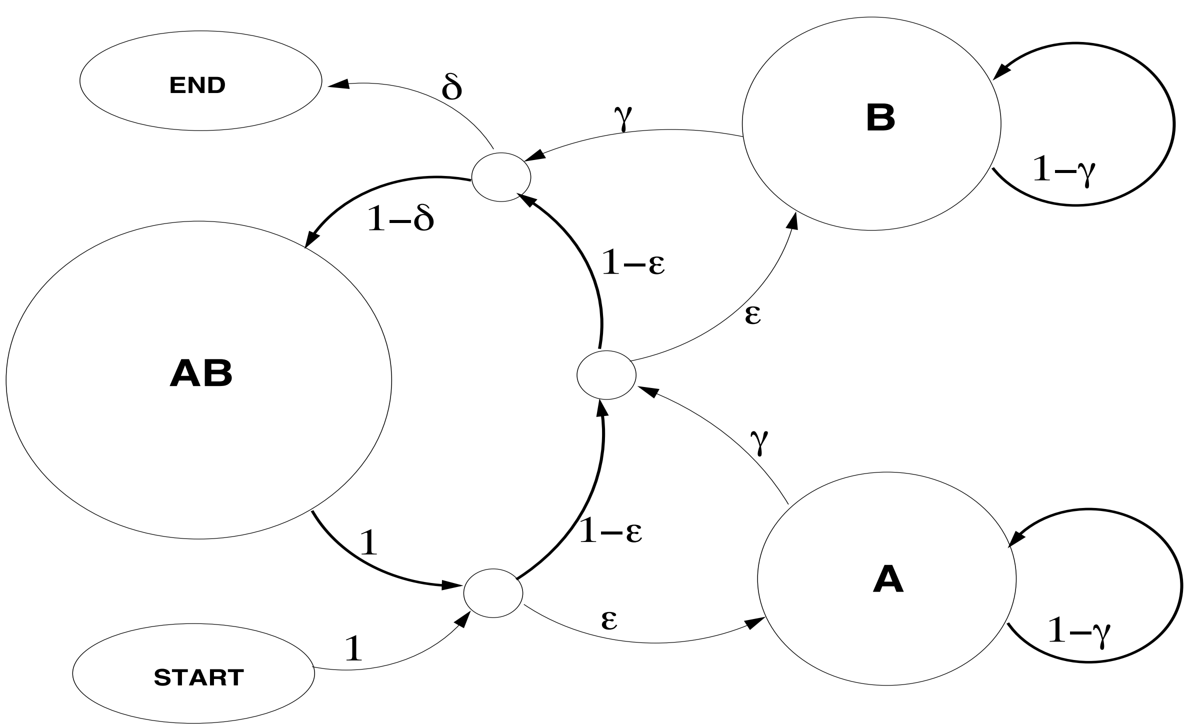
- Examples of Regularizers
- Regularization Operators
- Greens Functions and Self Consistency Condition

Fourier Regularization

- Translation Invariant Regularizers
- Regularization in Fourier Space
- Kernel is inverse Fourier Transformation of Weight

Polynomial Kernels and Series Expansions

String Kernel (pre)History



The Kernel Perspective

- Design a kernel implementing good features

$$k(x, x') = \langle \phi(x), \phi(x') \rangle \text{ and } f(x) = \langle \phi(x), w \rangle = \sum_i \alpha_i k(x_i, x)$$

- Many variants
  - Bag of words (AT&T labs 1995, e.g. Vapnik)
  - Matching substrings (Haussler, Watkins 1998)
  - Spectrum kernel (Leslie, Eskin, Noble, 2000)
  - Suffix tree (Vishwanathan, Smola, 2003)
  - Suffix array (Teo, Vishwanathan, 2006)
  - Rational kernels (Mohri, Cortes, Haffner, 2004 ...)

Bag of words

- At least since 1995 known in AT&T labs

$$k(x, x') = \sum_w n_w(x) n_w(x') \text{ and } f(x) = \sum_w \omega_w n_w(x')$$

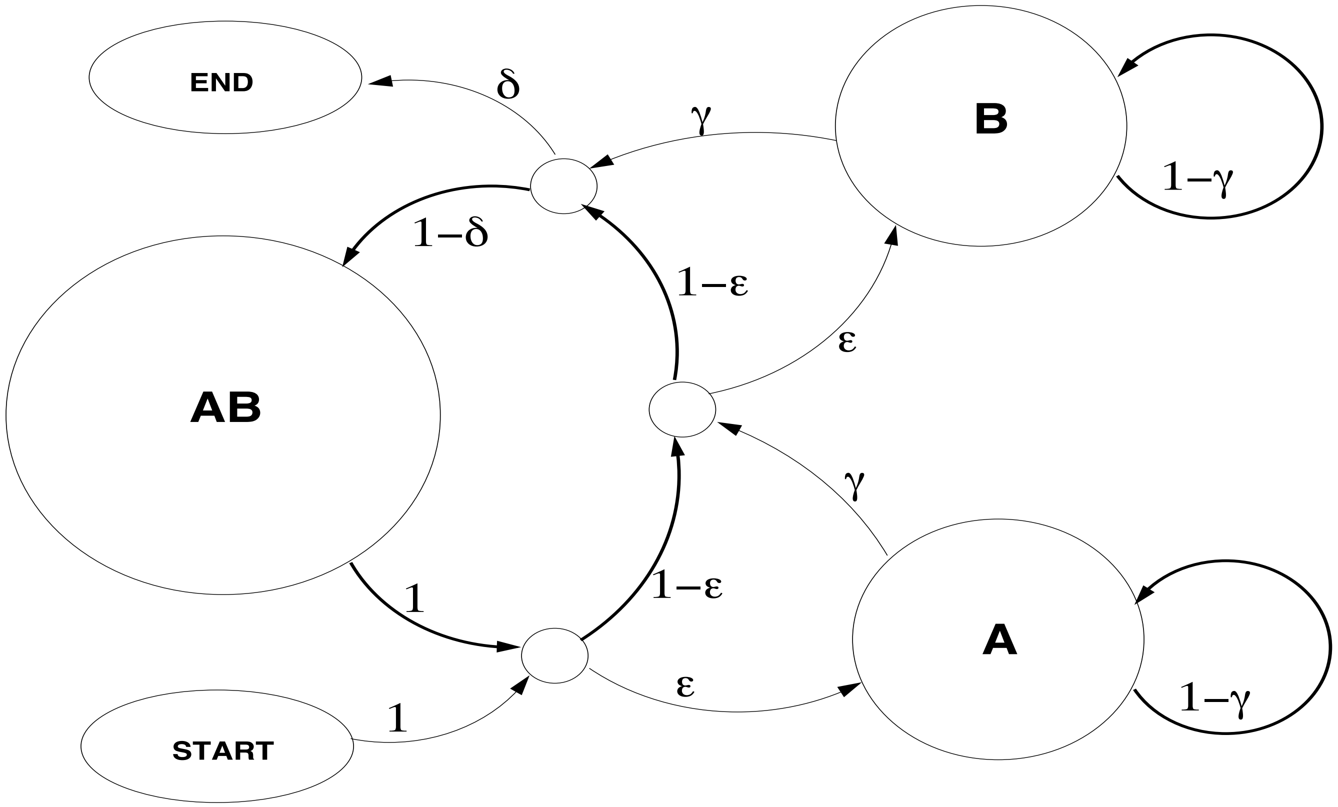
```
(to be or not to be) → (be:2, or:1, not:1, to:2)
```

- Joachims 1998: Use sparse vectors
- Haffner 2001: Inverted index for faster training
- Lots of work on feature weighting (TF/IDF)
- Variants of it deployed in many spam filters

Substring (mis)matching

- Watkins 1998+99 (dynamic alignment, etc)
- Haussler 1999 (convolution kernels)

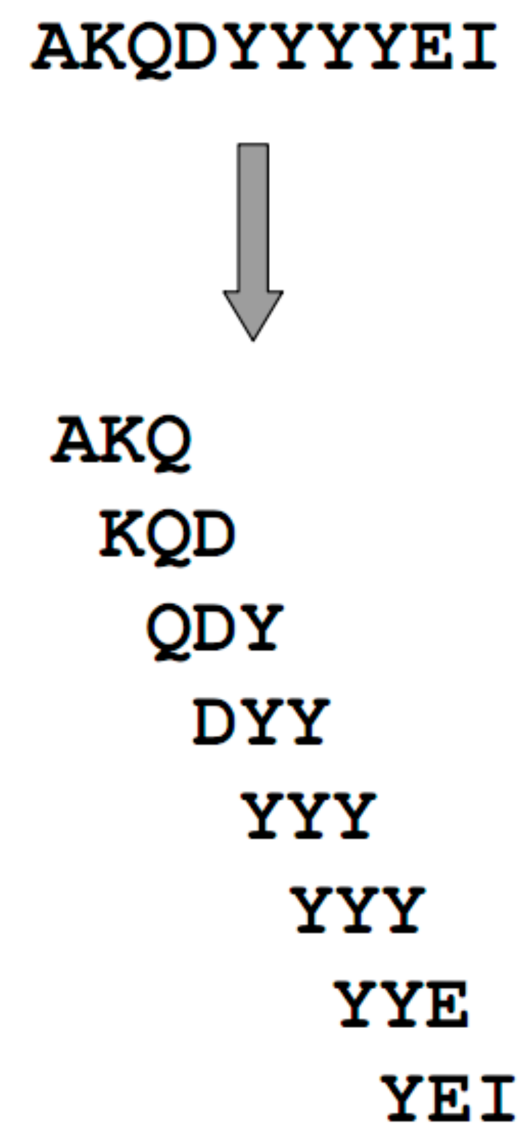
$$k(x, x') = \sum_{w \in x} \sum_{w' \in x'} \kappa(w, w')$$

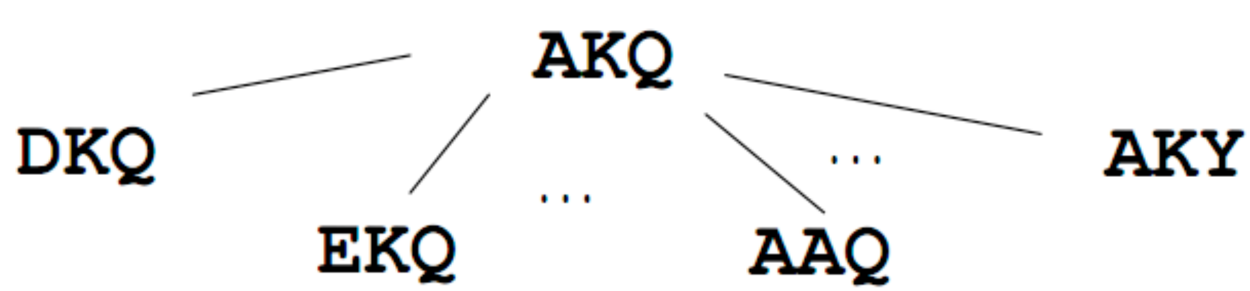


- In general O(x x′) runtime
  (e.g. Cristianini, Shawe-Taylor, Lodhi, 2001)
- Dynamic programming solution for pair-HMM

Spectrum Kernel

- Leslie, Eskin, Noble & coworkers, 2002
- Key idea is to focus on features directly
  - Linear time operation to get features
  - Limited amount of mismatch (exponential in number of missed chars)
  - Explicit feature construction (good & fast for DNA sequences)

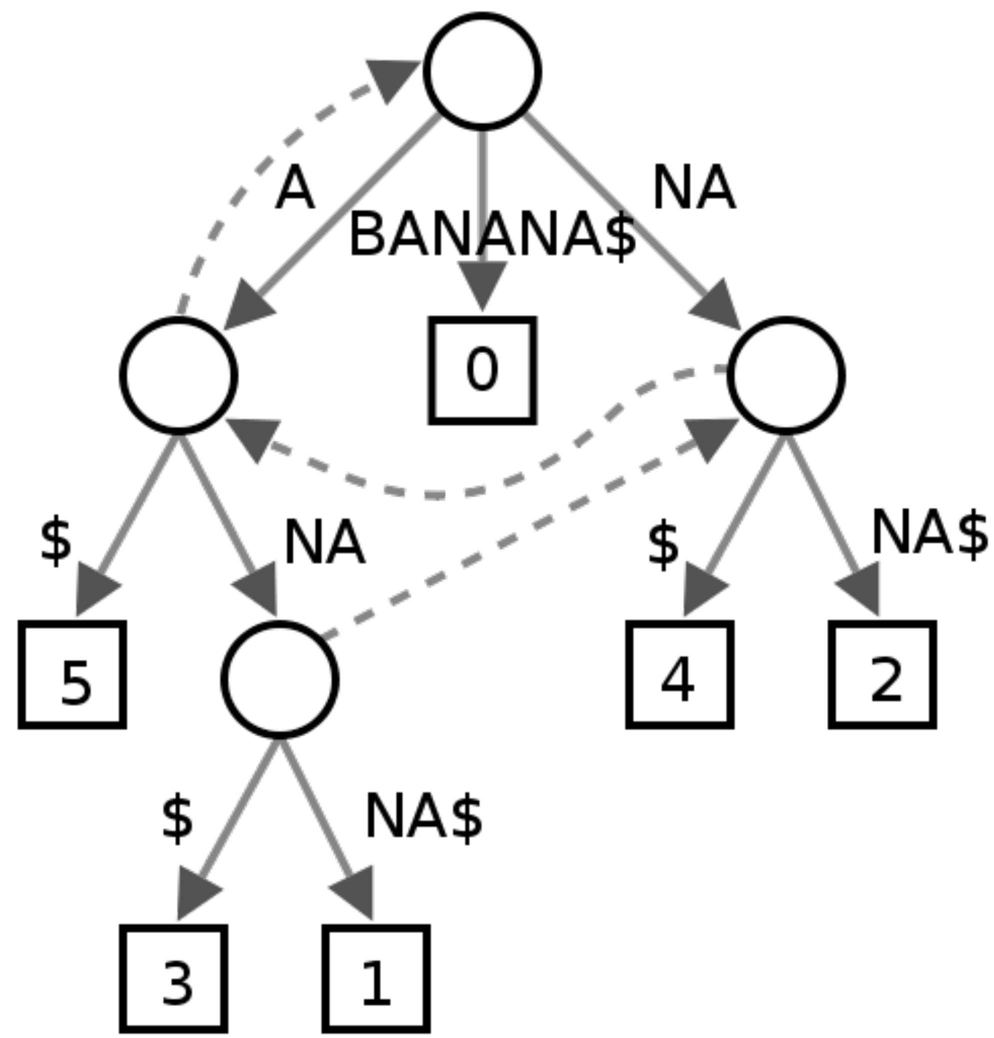




Suffix Tree Kernel

- Vishwanathan & Smola, 2003 (O(x + x′) time)
- Mismatch-free kernel + arbitrary weights

$$k(x, x') = \sum_w \omega_w n_w(x) n_w(x')$$



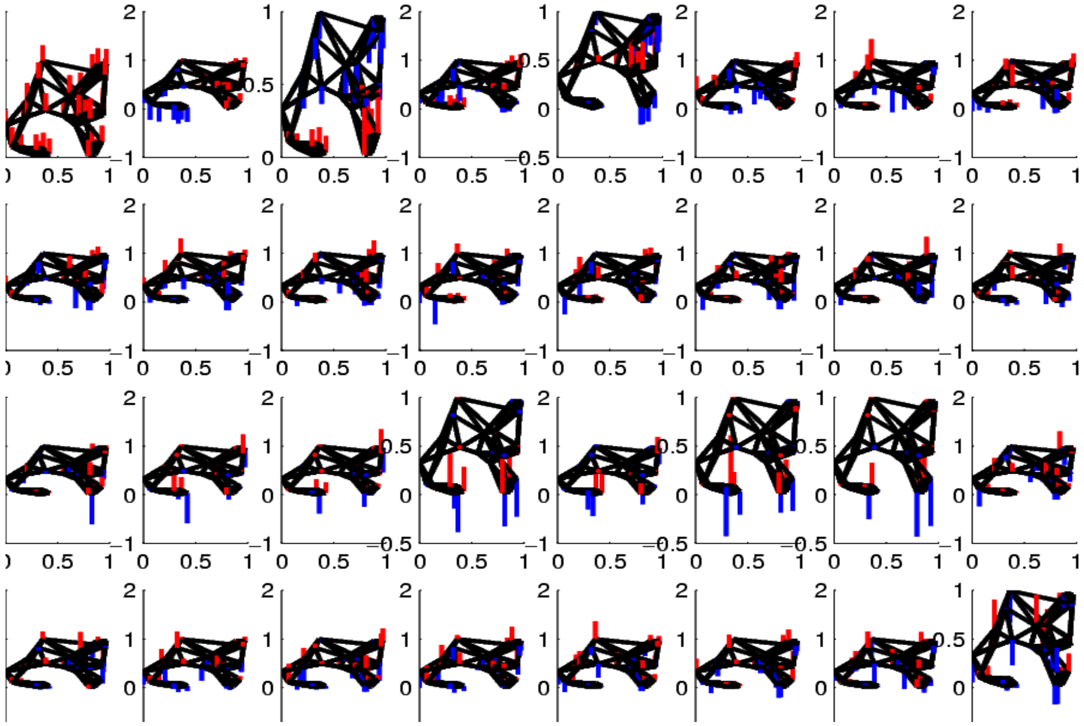
- Linear time construction (Ukkonen, 1995)
- Find matches for second string in linear time (Chang & Lawler, 1994)
- Precompute weights on path

Are we done?

- Large vocabulary size
- Need to build dictionary
- Approximate matches are still a problem
- Suffix tree/array is storage inefficient (40-60x)
- Realtime computation
- Memory constraints (keep in RAM)
- Difficult to implement

stay tuned

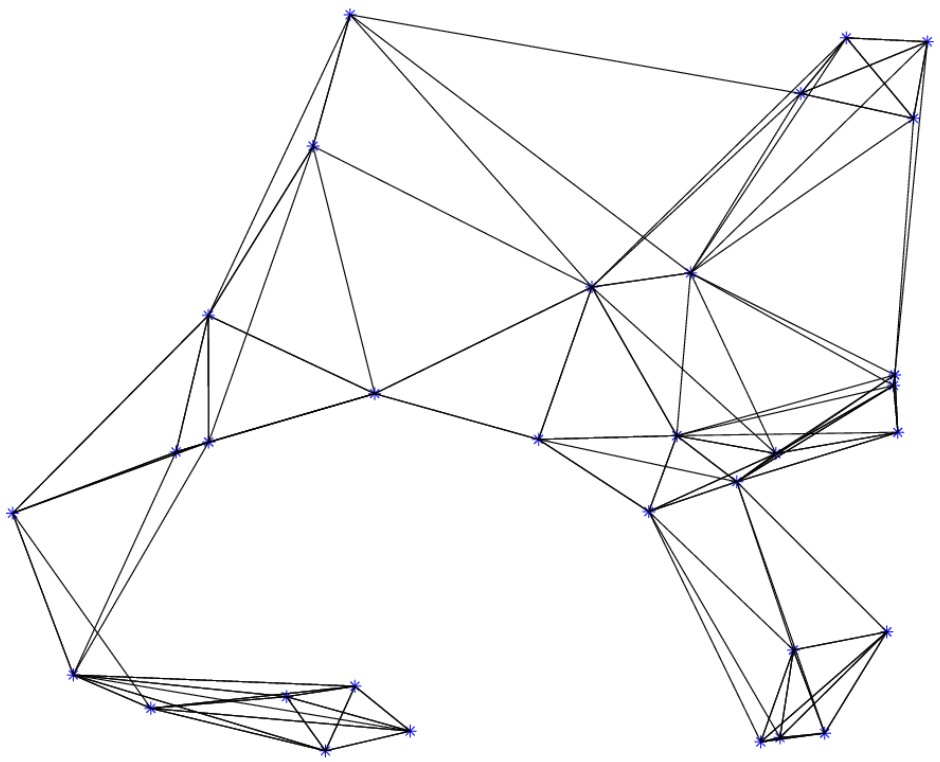
Graph Kernels



Graphs

Basic Definitions

- Connectivity matrix W where $W_{ij} = 1$ if there is an edge from vertex i to j ($W_{ij} = 0$ otherwise). For undirected graphs $W_{ii} = 0$.
- In this talk only undirected, unweighted graphs: $W_{ij} \in \{0, 1\}$ instead of $\mathbb{R}_0^+$.



Graph Laplacian

$$L := W - D \text{ and } \tilde{L} := D^{-\frac{1}{2}} L D^{-\frac{1}{2}} = D^{-\frac{1}{2}} W D^{-\frac{1}{2}} - \mathbf{1}$$

where $D = \mathrm{diag}(L\vec{1})$, i.e., $D_{ii} = \sum_j W_{ij}$. This talk only $\tilde{L}$

Graph Segmentation

Cuts and Associations

$$\operatorname{cut}(A, B) = \sum_{i \in A, j \in B} W_{ij}$$

$\operatorname{cut}(A, B)$ tells us how well A and B are connected.

Normalized Cut

$$\operatorname{Ncut}(A, B) = \frac{\operatorname{cut}(A, B)}{\operatorname{cut}(A, V)} + \frac{\operatorname{cut}(A, B)}{\operatorname{cut}(B, V)}$$

Connection to Normalized Graph Laplacian

$$\min_{A \cup B = V} \operatorname{Ncut}(A, B) = \min_{y \in \{\pm 1\}^m} \frac{y^\top (D - W) y}{y^\top D y}$$

- Proof idea: straightforward algebra
- Approximation: use eigenvectors / eigenvalues instead

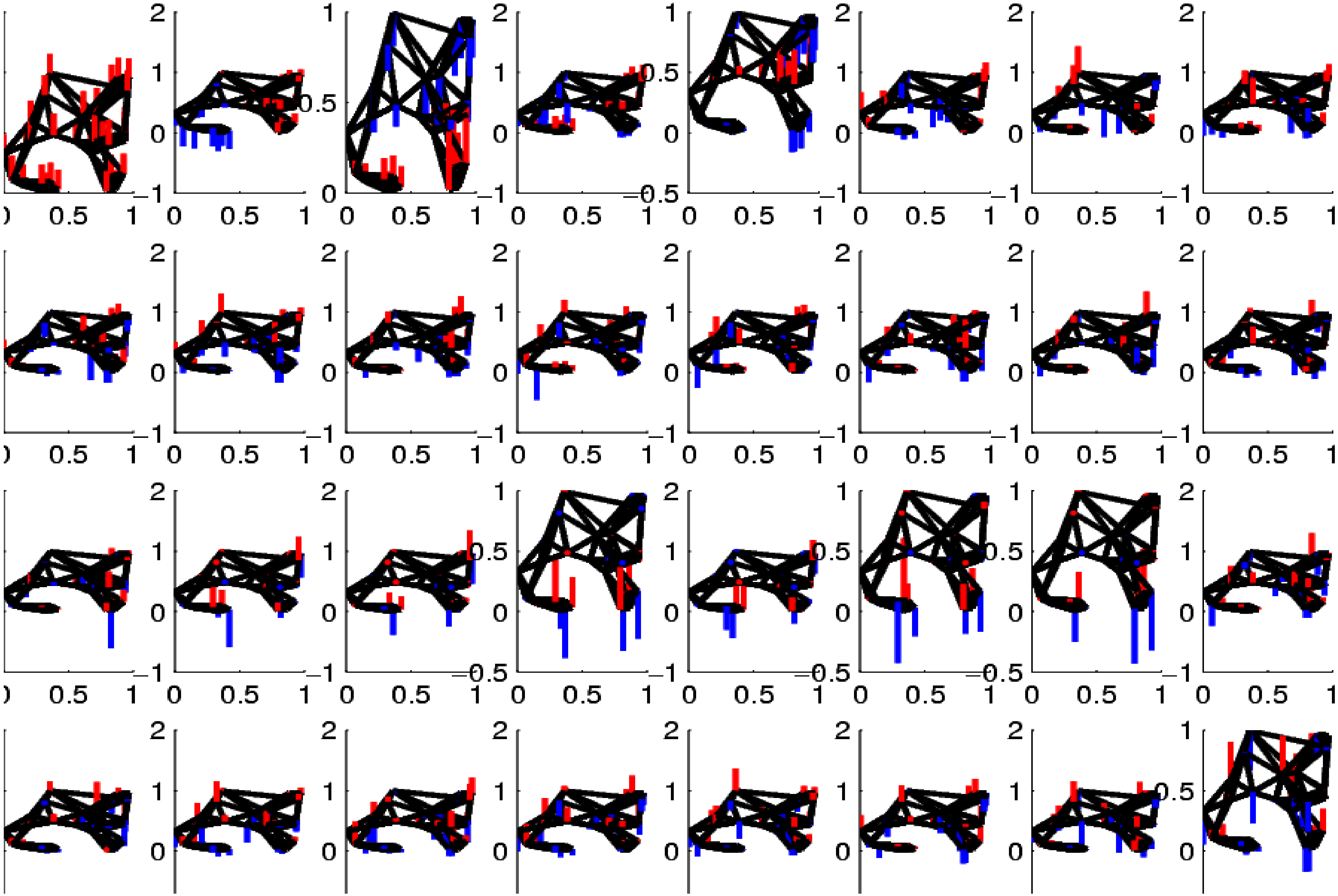
Eigensystem of the Graph Laplacian

- The spectrum of $\tilde{L}$ lies in $[0, 2]$ (via Gerschgorin's Theorem)
- Smallest eigenvalue/vector is $(\lambda_1, v_1) = (0, \vec{1})$
- Second smallest (λ_2, v_2) is Fiedler vector, which segments graph using approximate min-cut (cf. tutorials).
- Larger λ_i correspond to v_i which vary more clusters.
- For grids $\tilde{L}$ is the discretization of the conventional Laplace Operator

$$— \; x-2\delta \; — \; x-\delta \; — \; x \; — \; x+\delta \; —$$

Key Idea: use the v_i to build a hierarchy of increasingly complex functions on the graph.

Eigenvectors



Regularization operator on graph

Functions on the Graph

Since we have only exactly n vertices, all f are $f \in \mathbb{R}^n$.

Regularization Operator

$M := P^*P$ is therefore a matrix $M \in \mathbb{R}^{n \times n}$. Choosing the v_i as complexity hierarchy we set M

$$M = \sum_i r(\lambda_i) v_i v_i^\top \text{ and hence } M = r(\tilde{L})$$

Consequently, for $f = \sum_i \beta_i v_i$ we have $Mf = \sum_i r(\lambda_i) v_i$.

Some Choices for r

- $r(\lambda) = \lambda + \epsilon$ (Regularized Laplacian)
- $r(\lambda) = \exp(\sigma\lambda)$ (Diffusion on Graphs)
- $r(\lambda) = (a - \lambda)^{-p}$ (p-Step Random Walk)

Kernels

Self Consistency Equation

Matrix notation for $k^\top(x,\cdot)(P^*P)k(x',\cdot) = k(x,x')$:

$$KM^{-1}K = K \text{ and hence } K = M^{-1}$$

Here we take the pseudoinverse if M^{-1} does not exist.

Regularized Laplacian

$r(\lambda) = \lambda + \epsilon$, hence $M = \tilde{L} + \epsilon\mathbf{1}$ and $K = (\tilde{L} + \epsilon\mathbf{1})^{-1}$. Work with K^{-1}!

Diffusion on Graphs

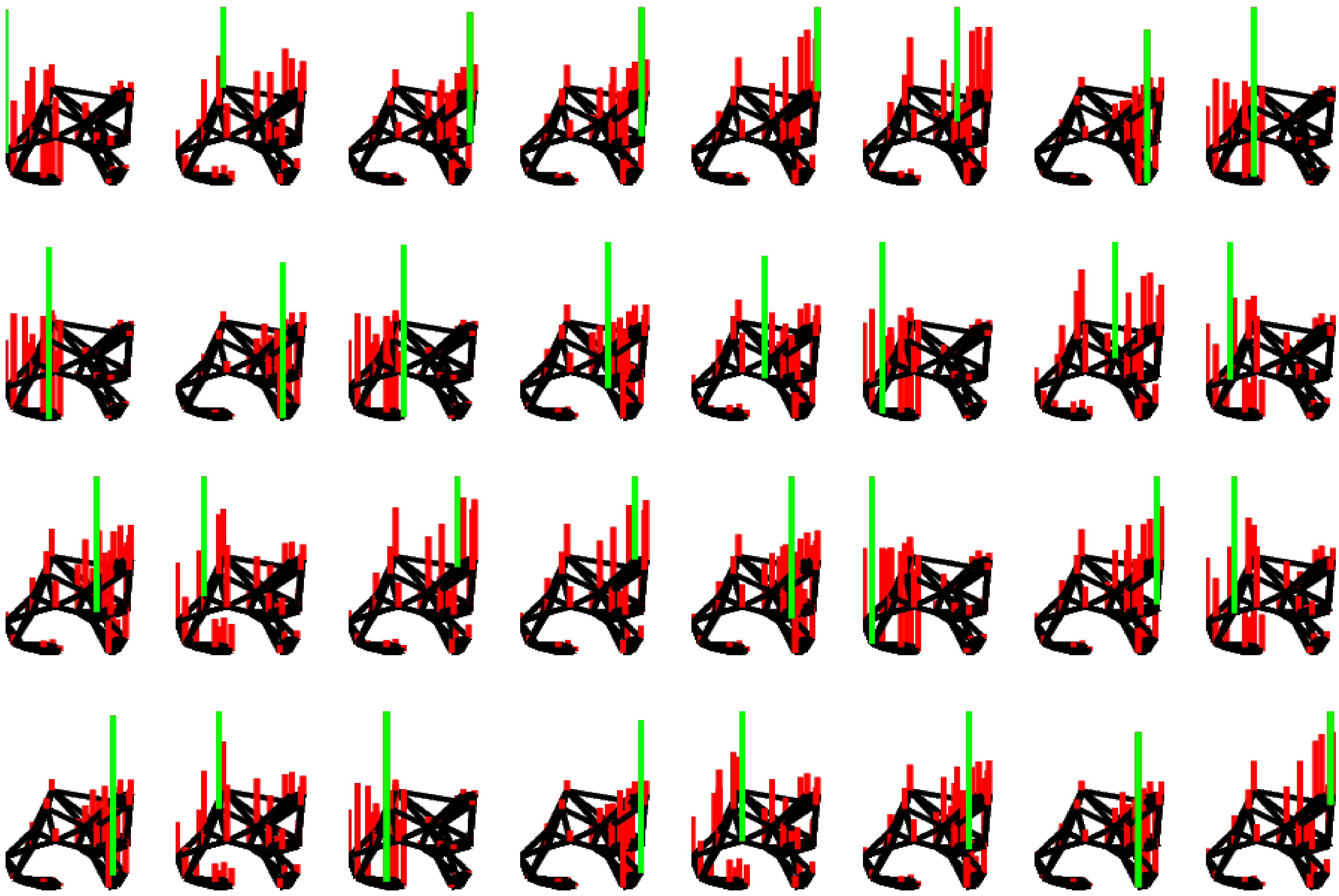
$r(\lambda) = \exp(\sigma\lambda)$, hence $M = \exp(\sigma\tilde{L})$ and $K = \exp(-\sigma\tilde{L})$. Here K_{ij} is the probability of reaching i from j.

p-Step Random Walk

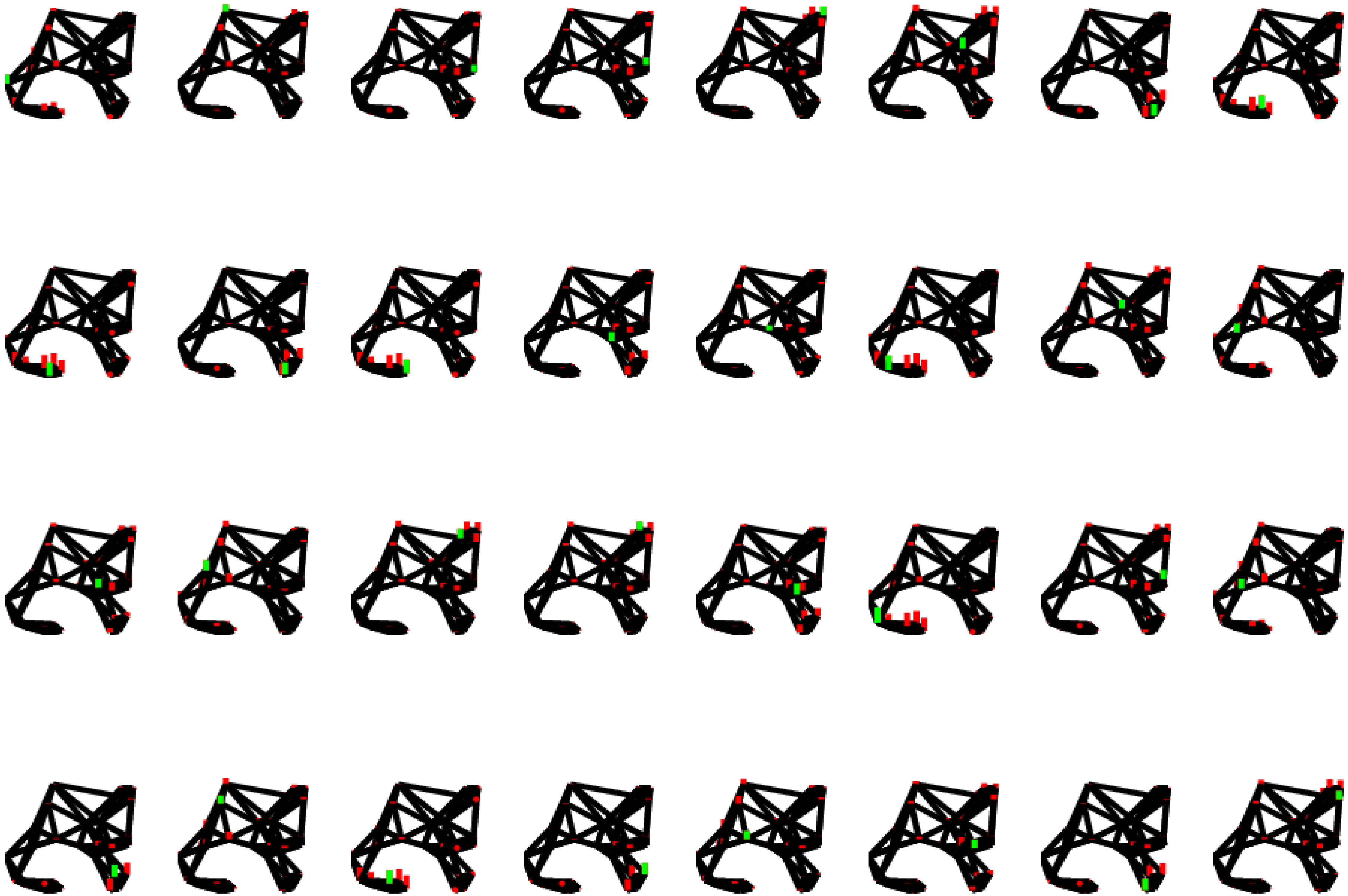
For $r(\lambda) = (a - \lambda)^{-p}$ we have $K = (a\mathbf{1} - \tilde{L})^p$.

Weighted combination over several random walk steps.

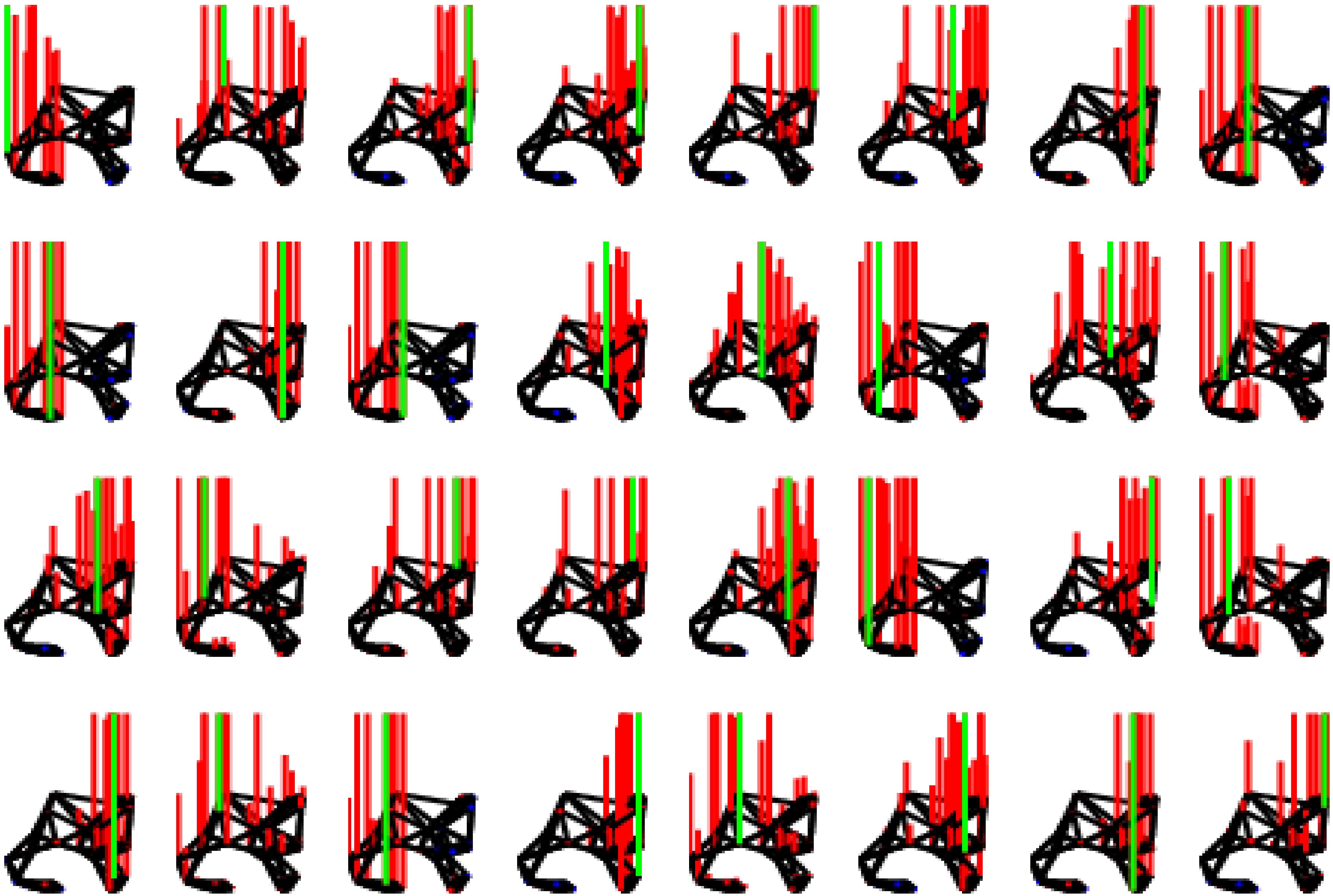
Graph Laplacian Kernel



Diffusion Kernel



4-Step Random Walk



Fast computation

- Primal space computation
- Weisfeiler-Lehman hash
- Heat equation

Watson, *Bessel Functions*

PREFACE TO THE SECOND EDITION

To incorporate in this work the discoveries of the last twenty years would necessitate the rewriting of at least Chapters XII—XIX; my interest in Bessel functions, however, has waned since 1922, and I am consequently not prepared to undertake such a task to the detriment of my other activities. In the preparation of this new edition I have therefore limited myself to the correction of minor errors and misprints and to the emendation of a few assertions (such as those about the unproven character of Bourget's hypothesis) which, though they may have been true in 1922, would have been definitely false had they been made in 1941.

My thanks are due to many friends for their kindness in informing me of errors which they had noticed; in particular, I cannot miss this opportunity of expressing my gratitude to Professor J. R. Wilton for the vigilance which he must have exercised in the compilation of his list of corrigenda.

G. N. W.

March 31, 1941.

Midterm Project Presentations

- Midterm project presentations
  - March 13, 4-7pm
  - Send the PDF (+supporting material) to Dapo by March 12, midnight
- Questions to answer
  - What (you will do, what you have already done)
  - Why (it matters)
  - How (you're going to achieve it)
- Rules
  - 10 minutes per team (6 slides maximum)
  - 10 pages supporting material (maximum)

Regularization Summary

Regularization

- **Feature space Expansion**

$$\underset{\beta}{\text{minimize}} \sum_i l\left(y_i, [X\beta]_i\right) + \frac{\lambda}{2} \|\beta\|^2$$

- **Kernel Expansion**

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, [XX^\top \alpha]_i\right) + \frac{\lambda}{2} \alpha^\top X X^\top \alpha$$

- **Function Expansion**

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, f_i\right) + \frac{\lambda}{2} f^\top (XX^\top)^{-1} f$$

$$f = X\beta = X^\top X\alpha$$

Feature Space Expansion

$$\underset{\beta}{\text{minimize}} \;\; \sum_i l\left(y_i, [X\beta]_i\right) + \frac{\lambda}{2}\left\|\beta\right\|^2$$

- Linear methods
  - Design feature space
  - Solve problem there
  - Fast ‘primal space’ methods for SVM solvers
  - Stochastic gradient descent solvers

Kernel Expansion

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, [XX^\top \alpha]_i\right) + \frac{\lambda}{2}\alpha^\top XX^\top \alpha$$

- Using the kernel trick

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, [K\alpha]_i\right) + \frac{\lambda}{2}\alpha^\top K\alpha$$

- Optimization via
  - Interior point solvers
  - Coefficient-wise updates (e.g. SMO)
  - Fast matrix vector products in K

Function Expansion

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, f_i\right) + \frac{\lambda}{2} f^\top (X X^\top)^{-1} f$$

- Using the kernel trick yields Gaussian Process

$$\underset{\alpha}{\text{minimize}} \sum_i l\left(y_i, f_i\right) + \frac{\lambda}{2} f^\top K^{-1} f$$

- Inference via
  - Fast inverse kernel matrix (e.g. graph kernel)
  - Low-rank approximation of K
  - Occasionally useful for distributed inference

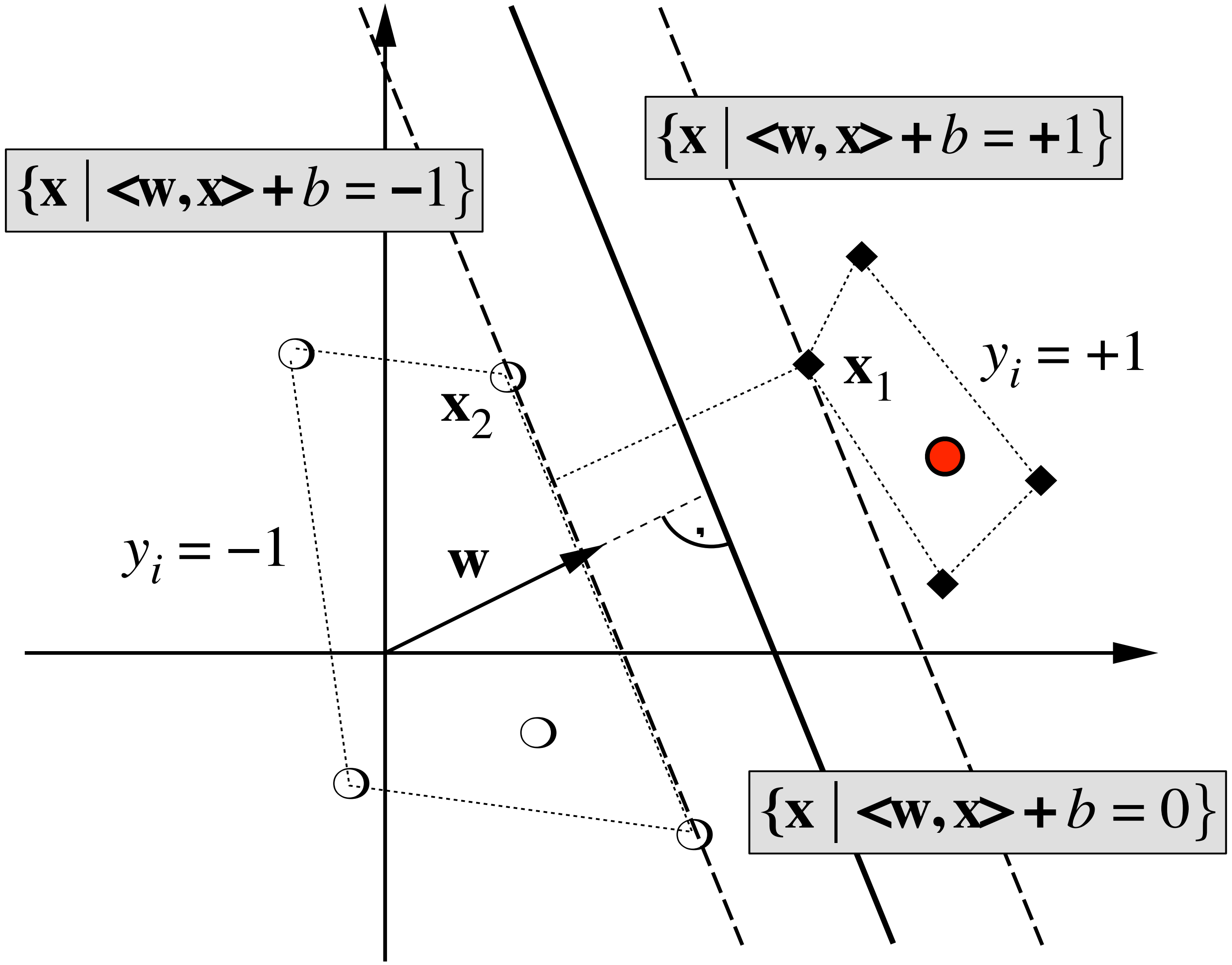
Optimization Algorithms

Efficient Optimization

- Dual Space
  Solve the original SVM dual problem efficiently
  (SMO, LibLinear, SVMLight, ...)
- Subspace
  Find a subspace that contains a good approximation to the solution
  (Nystrom, SGMA, Pivoting, Reduced Set)
- Function values
  Explicit expansion of regularization operator
  (graphs, strings, Weisfeiler-Lehman)
- Parameter space
  Efficient linear parametrization without projection
  (hashing, random kitchen sinks, multipole)

Dual Space

Support Vector Machine



dual problem

$$\underset{\alpha}{\text{minimize}} \ \ \frac{1}{2}\alpha^\top K \alpha - 1^\top \alpha$$

$$\text{subject to } \sum_i \alpha_i y_i = 0$$

$$\alpha_i \in [0, C]$$

$$K_{ij} = y_i y_j \langle x_i, x_j \rangle$$

$$w = \sum_i \alpha_i y_i x_i$$

$$\underset{w,b}{\text{minimize}} \ \ \frac{1}{2}\|w\|^2 + C\sum_i \xi_i$$

$$\text{subject to } y_i \left[\langle w, x_i \rangle + b\right] \geq 1 - \xi_i \text{ and } \xi_i \geq 0$$

Problems

- Kernel matrix may be huge
  - Cannot store it in memory
  - Expensive to compute
  - Expensive to evaluate linear functions
- Quadratic program is too large
  Cubic cost for naive Interior Point solution
- Only evaluate rows
- Cache values
- Cache linear function values
- Solve subsets of the problem and iterate

Subproblem

Full problem (using $\bar{K}_{ij} := y_i y_j k(x_i, x_j)$)

$$\text{minimize } \frac{1}{2}\sum_{i,j=1}^{m} \alpha_i \alpha_j \bar{K}_{ij} - \sum_{i=1}^{m} \alpha_i$$

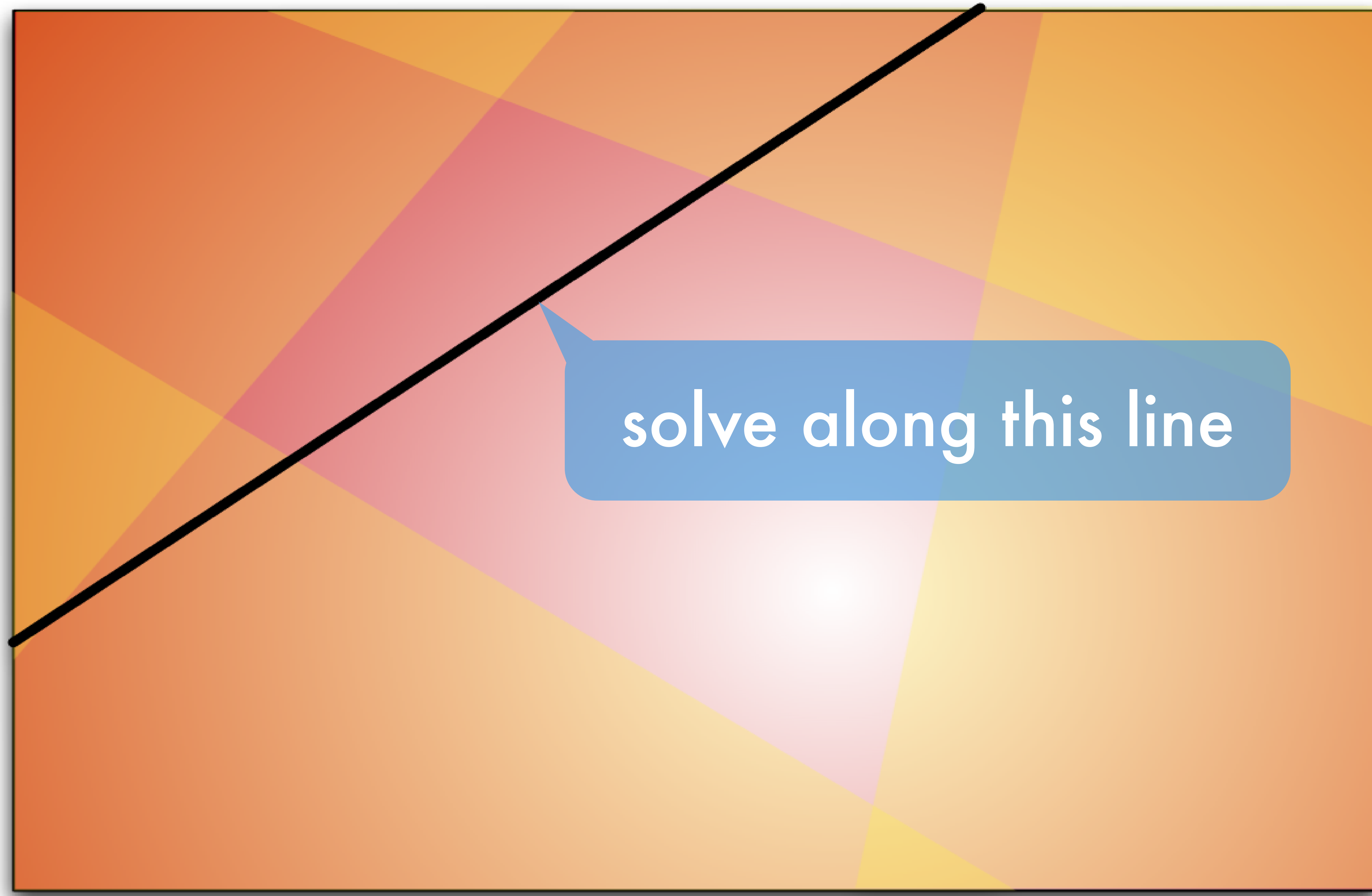
$$\text{subject to } \sum_{i=1}^{m} \alpha_i y_i = 0 \text{ and } \alpha_i \in [0, C] \text{ for all } 1 \leq i \leq m$$

Constrained problem: pick subset S

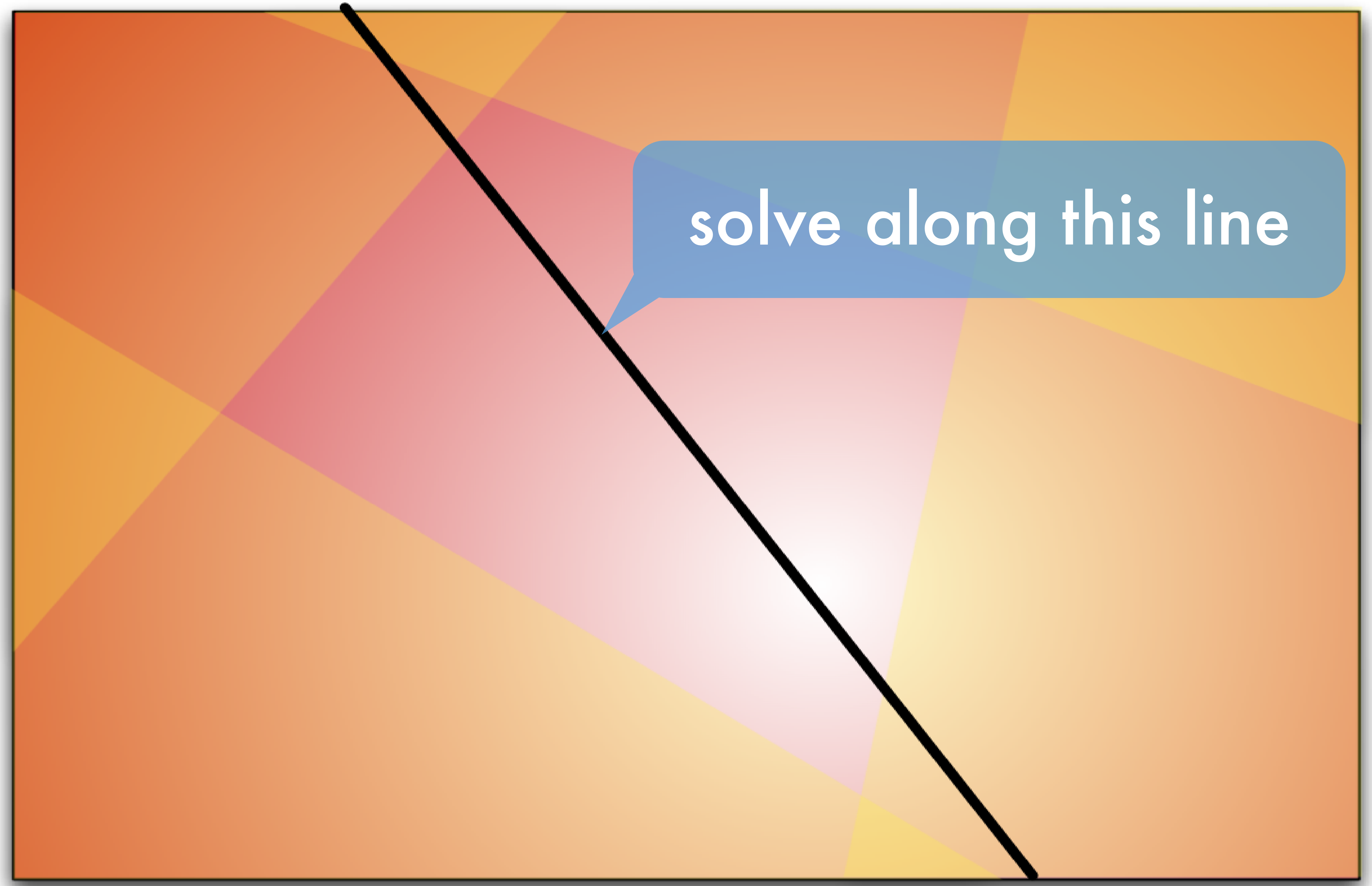
$$\text{minimize } \frac{1}{2}\sum_{i,j \in S} \alpha_i \alpha_j \bar{K}_{ij} - \sum_{i \in S} \alpha_i \left[1 - \sum_{j \notin S} K_{ij} \alpha_j\right] + \text{const.}$$

$$\text{subject to } \sum_{i \in S} \alpha_i y_i = -\sum_{i \notin S} \alpha_i y_i \text{ and } \alpha_i \in [0, C] \text{ for all } i \in S$$

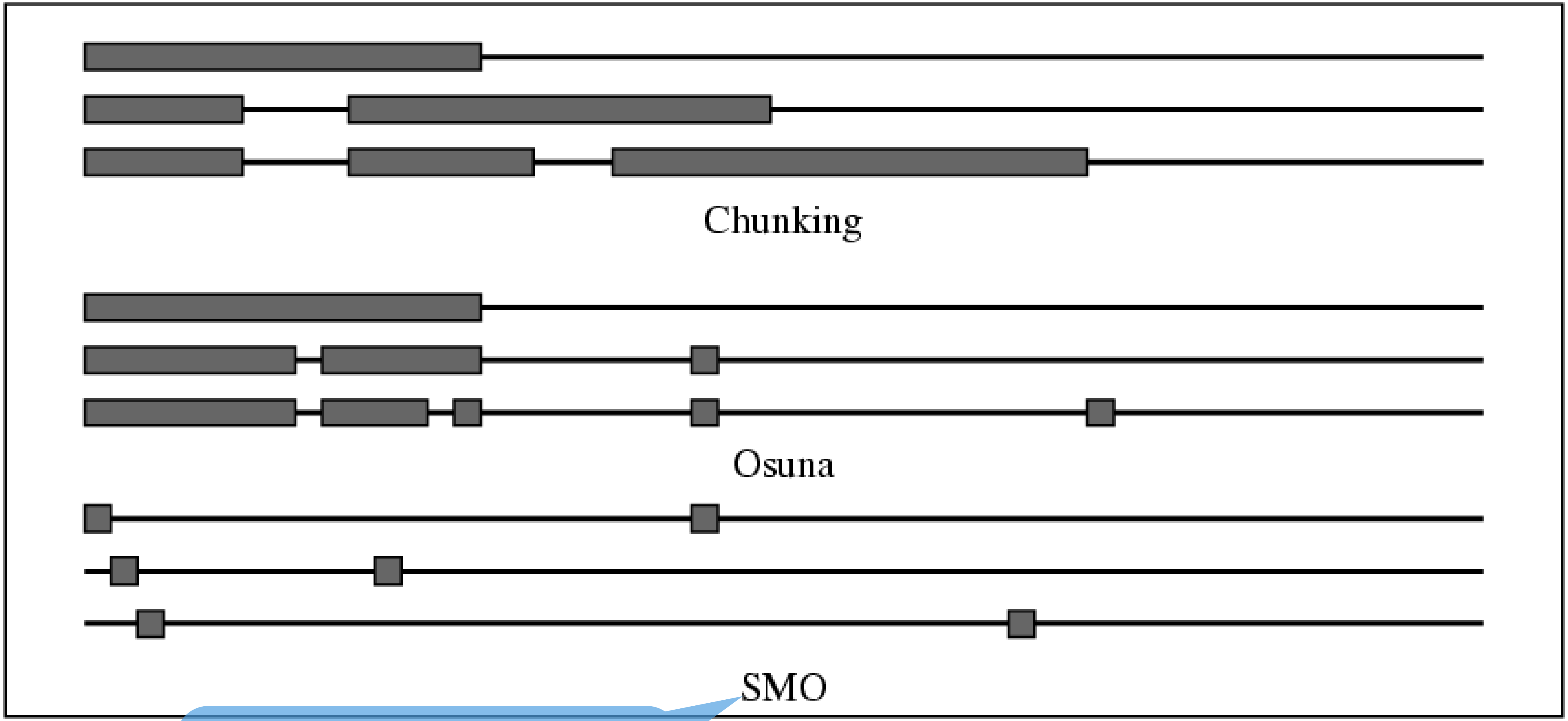
Active set strategy



Active set strategy



Subset Selection Strategies



often fastest

Improved Sequential Minimal Optimization
Dual Cached Loops

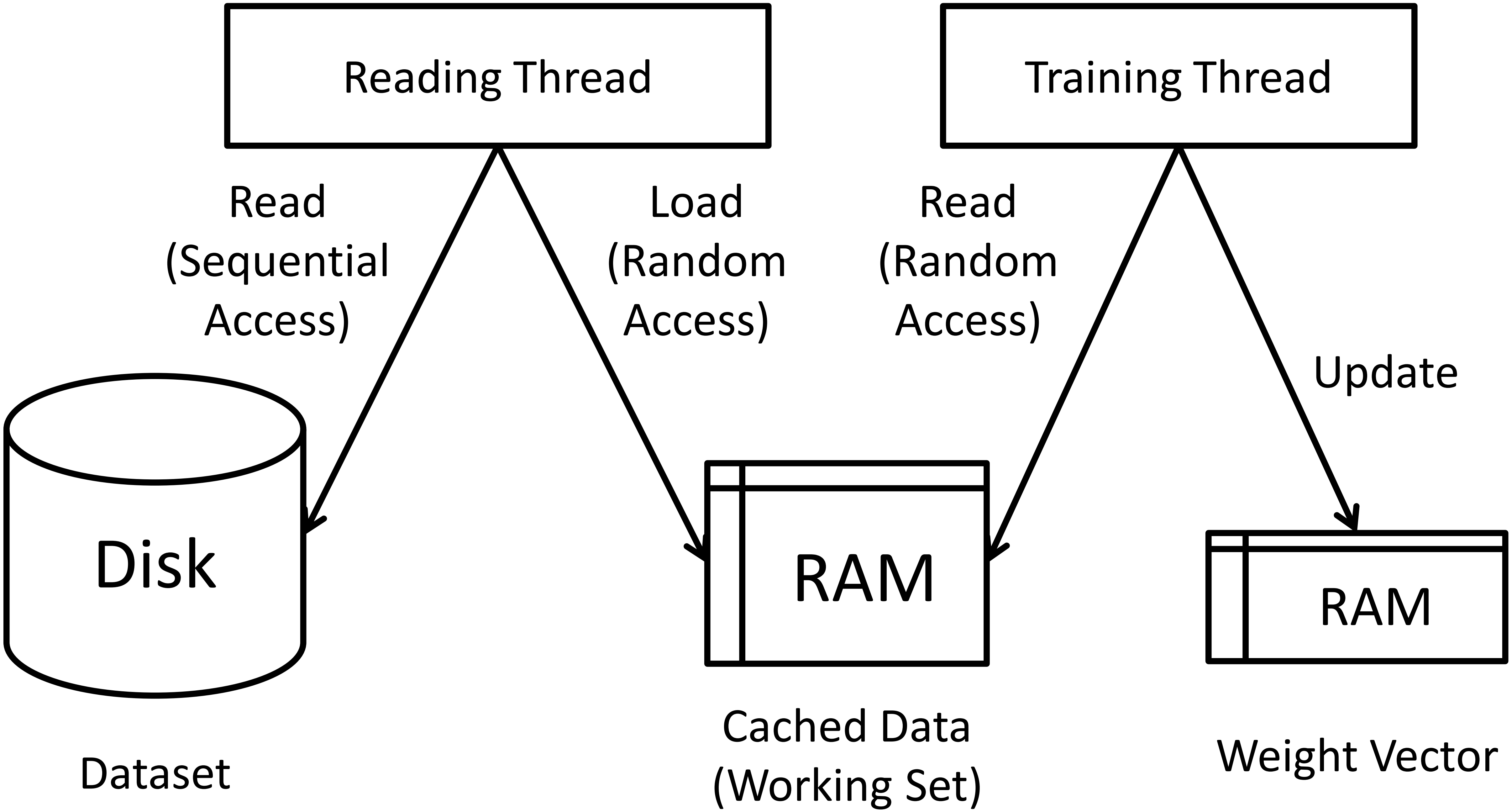
Storage Speeds

System	Capacity	Bandwidth	IOP/s
Disk	3TB	150MB/s	10^2
SSD	256GB	500MB/s	$5 \cdot 10^4$
RAM	16GB	30GB/s	10^8
Cache	16MB	100GB/s	10^9

- Algorithms iterating data from disk are disk bound
- Increasing number of cores makes this worse
- True for full memory hierarchy (10x per level)

Key Idea: recycle data once we load it in memory

Dataflow



Reading Thread
Training Thread
Read
(Sequential
Access)
Load
(Random
Access)
Read
(Random
Access)
Update
Disk
RAM
RAM
Dataset
Cached Data
(Working Set)
Weight Vector

Convex Optimization

- SVM optimization problem (without b)

$$\underset{w \in \mathbb{R}^d}{\text{minimize}} \frac{1}{2} \|w\|^2 + C \sum_{i=1}^{n} \max\{0, 1 - w^\top y_i x_i\}$$

- Dual problem

no equality constraint

$$\underset{\alpha}{\text{minimize}}\ D(\alpha) := \frac{1}{2}\alpha^\top Q \alpha - \alpha^\top \mathbf{1}$$
$$\text{subject to } \mathbf{0} \leq \alpha \leq C\mathbf{1}.$$

- Coordinate descent (SMO style - really simple)

$$\alpha_{i_t}^{t+1} = \underset{0 \leq \alpha_{i_t} \leq C}{\text{argmin}}\ D(\alpha^t + (\alpha_{i_t} - \alpha_{i_t}^t)\mathbf{e}_{i_t})$$

Algorithm - 2 loops

Reader
while not converged **do**
  read example (x, y) from disk
  if buffer full **then** evict random (x', y') from memory
  insert new (x, y) into ring buffer in memory
end while

at disk speed

Trainer
while not converged **do**
  randomly pick example (x, y) from memory
  update dual parameter α
  update weight vector w
  if deemed to be uninformative **then** evict (x, y) from memory
end while

at RAM speed

margin criterion

Advantages

- Extensible to general loss functions (simply use convex conjugate)
- Extensible to other regularizers (again using convex conjugate)

$$\underset{\alpha}{\text{minimize}} \sum_i l^*(z_i, y_i) + \lambda\Omega^*(\alpha) \text{ for } z = X\alpha$$

- Parallelization by oversample, distribute & average (Murata, Amari, Yoshizawa theorem)
- Convergence proof via Luo-Tseng

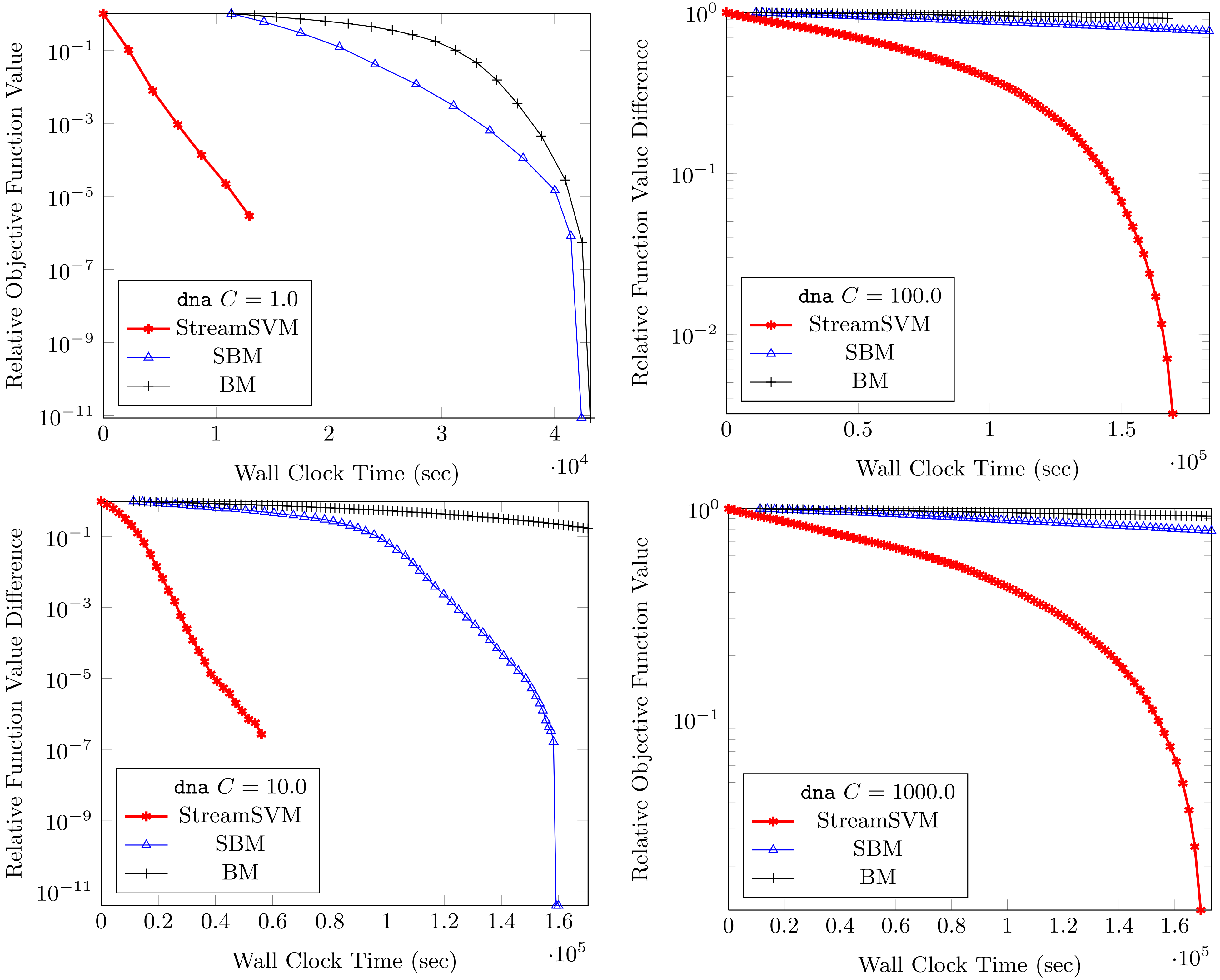
Results

- 12 core Opteron (currently not all cores used)
- Datasets

dataset	n	d	$s(\%)$	$n_+ : n_-$	Datasize	Ω	SBM Blocks	BM Blocks
`ocr`	3.5 M	1156	100	0.96	45.28 GB	150,000	40	20
`dna`	50 M	800	25	3e−3	63.04 GB	700,000	60	30
`webspam-t`	0.35 M	16.61 M	0.022	1.54	20.03 GB	15,000	20	10
`kddb`	20.01 M	29.89 M	1e-4	6.18	4.75 GB	2,000,000	6	3

- Variable amounts of cache
- Comparison to Chih-Jen Lin's KDD'11 prize winning LibLinear solver (SBM) and simple block minimization (BM)
- Kyoto cabinet for caching (suboptimal)

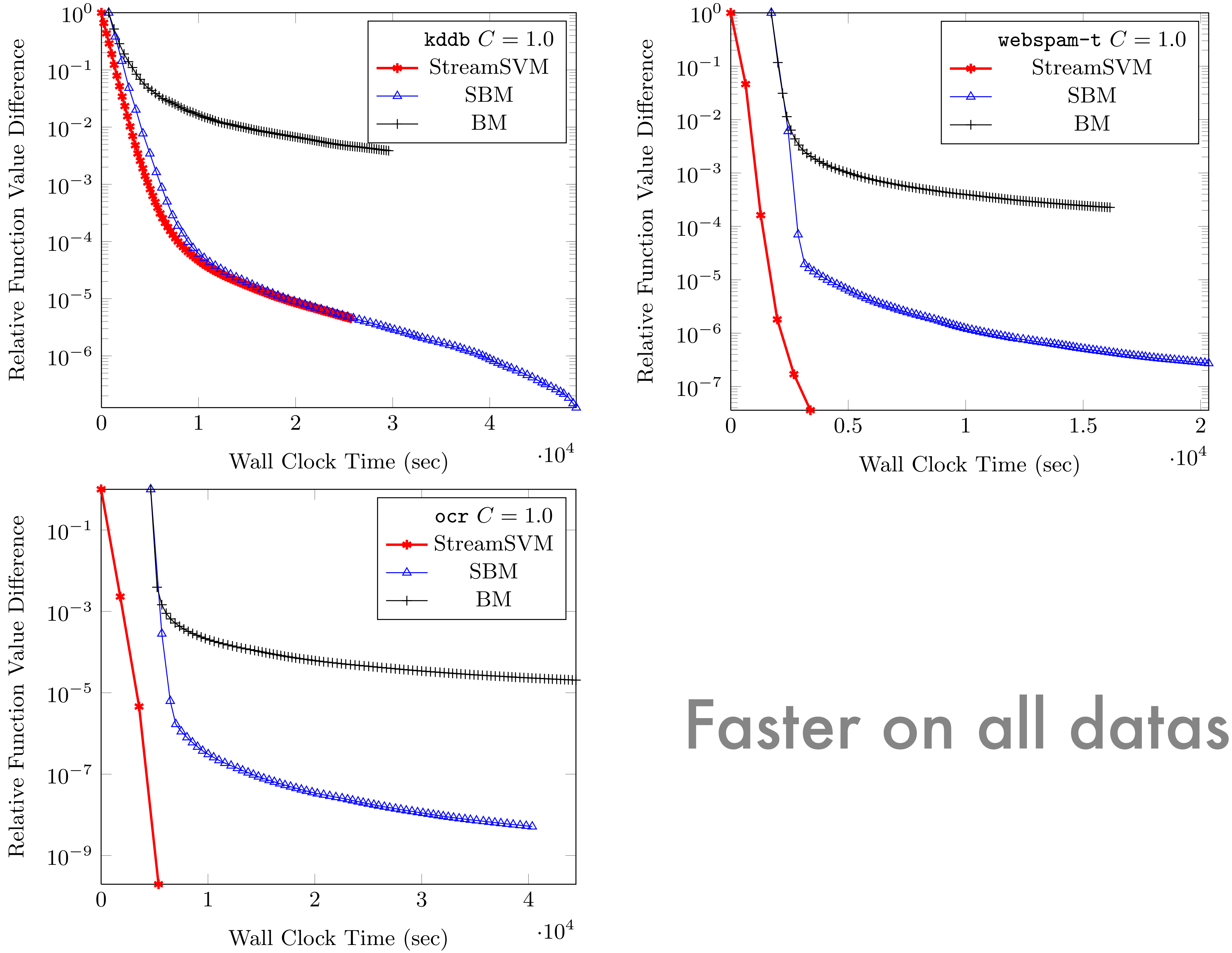
Convergence (DNA, different C)



much better for large C

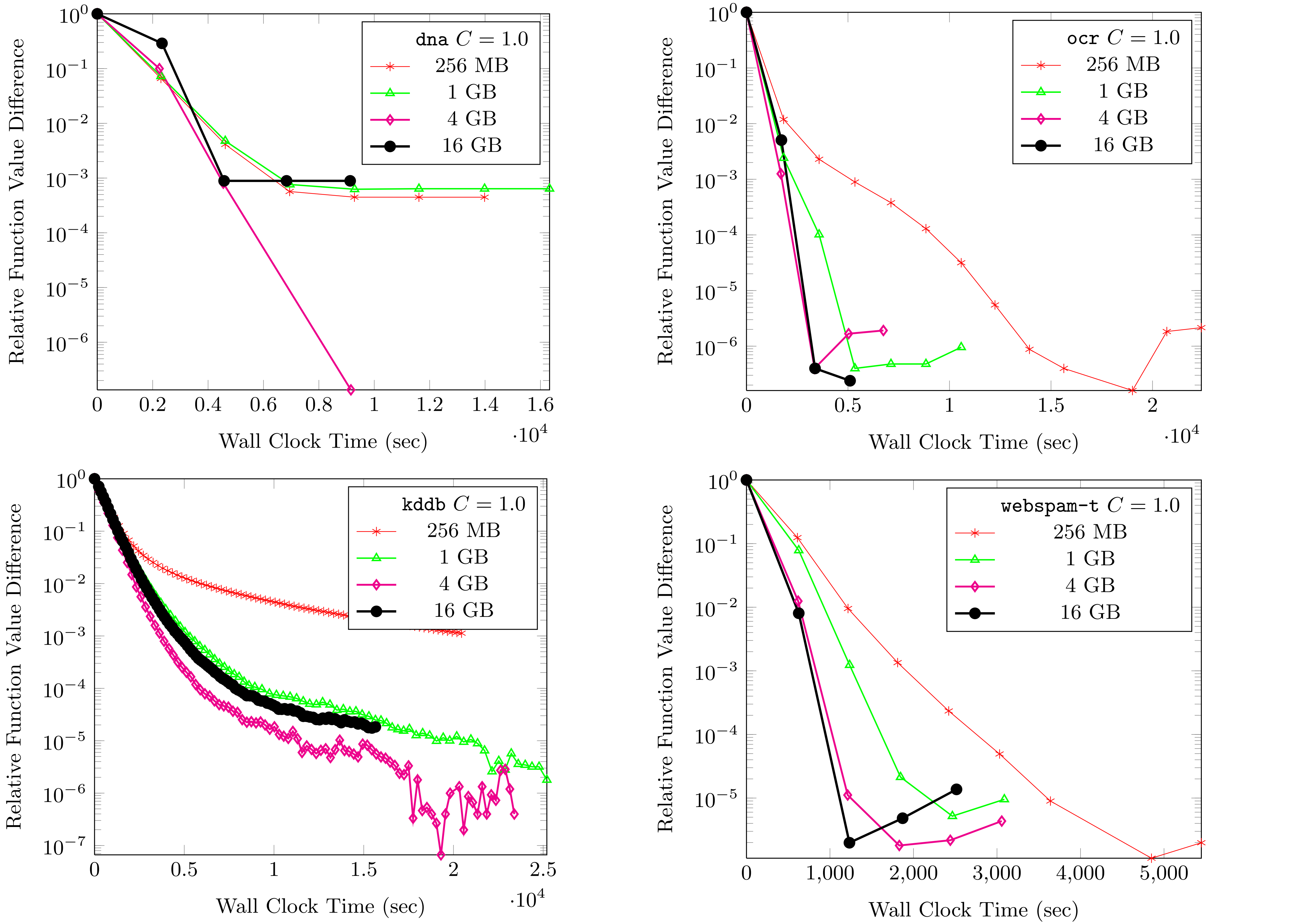
70h on 1machine

Convergence (C=1, different datasets)



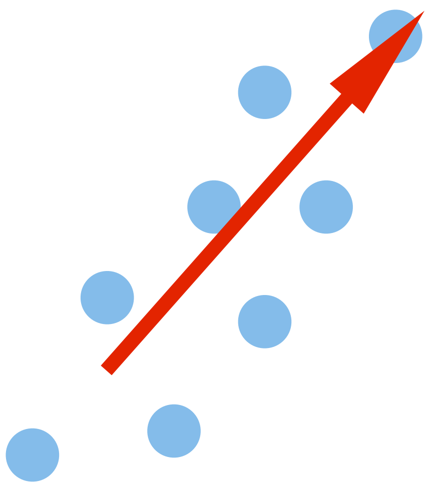
Faster on all datasets

Effect of caching



Subspace

Basic Idea



- Solution lies is in low-dimensional subspace of data (approximately)
- Find a sparse linear expansion
  - Before solving the problem
    (Sparse greedy, pivoting)
    Find solution in low-dimensional subspace
  - After solving the problem
    (Reduced set)
    Need to sparsify existing solution

Linear Approximation

- Project data into lower-dimensional space
  - Data in feature space $x \to \phi(x)$
  - Set of basis functions $\{\phi(x_1), \ldots \phi(x_n)\}$
  - Projection problem

$$\underset{\beta}{\text{minimize}} \left\| \phi(x) - \sum_{i=1}^{n} \phi(x_i) \right\|^2$$

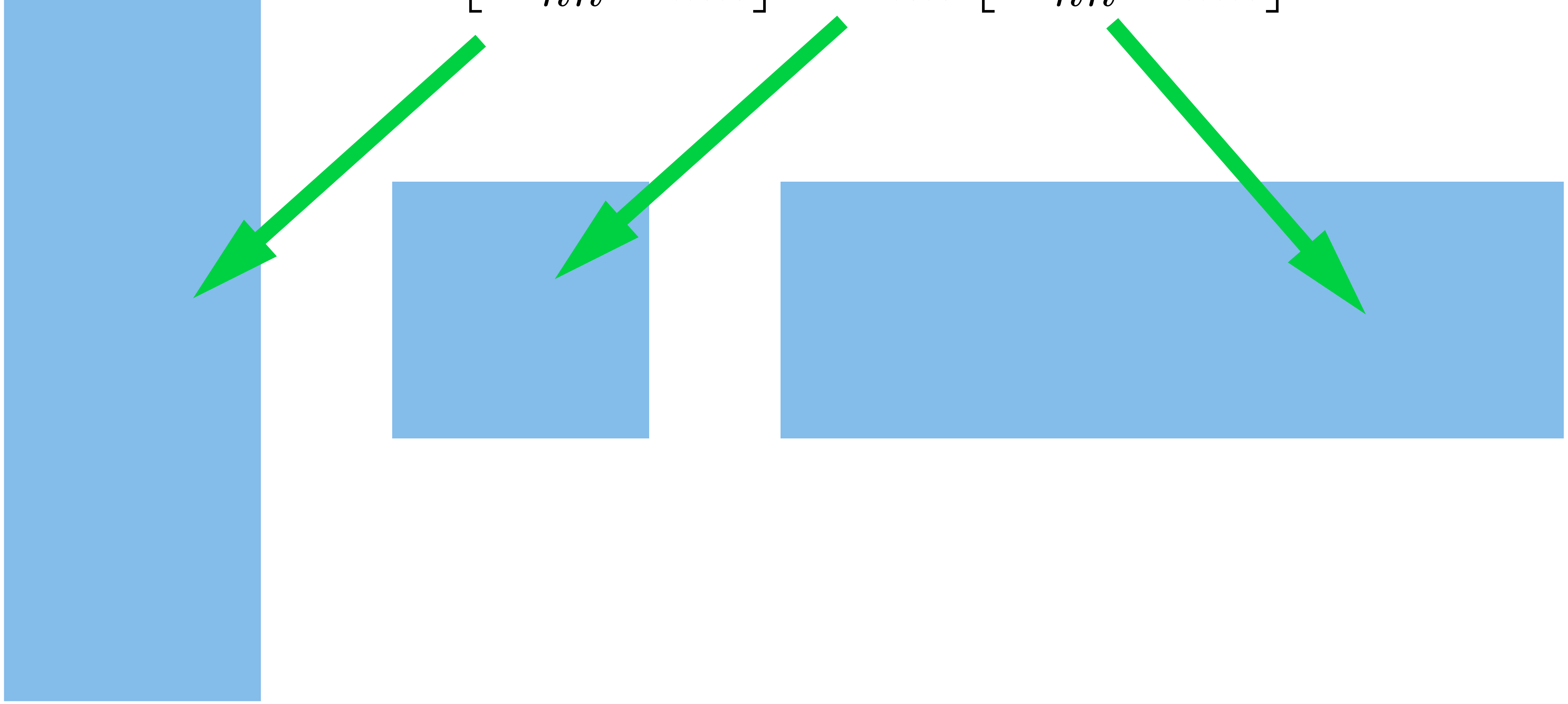
  - Solution $\beta = K(X,X)^{-1} K(X,x)$
  - Residual $\left\| \phi(x) - \sum_{i=1}^{n} \phi(x_i) \right\|^2 = \|\phi(x)\|^2 - \left\| \sum_{i=1}^{n} \phi(x_i) \right\|^2$

$$= k(x,x) - K(x,X)K(X,X)^{-1}K(X,x)$$

Subspace Finding

- Incomplete Cholesky factorization

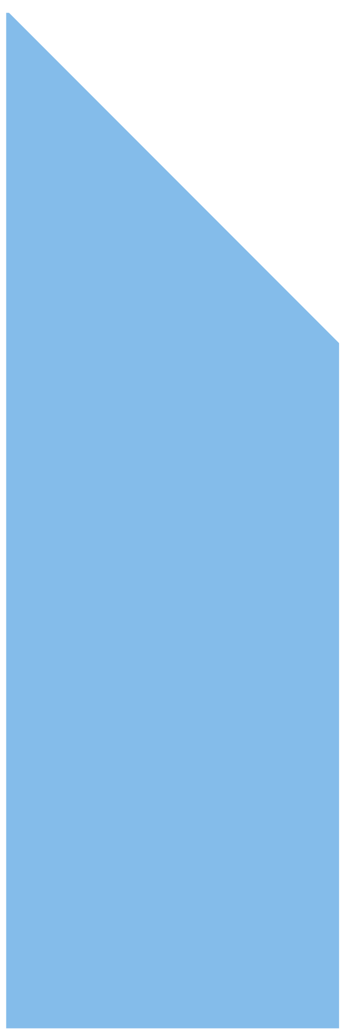
$$
\begin{aligned}
K &= \left[\phi(x_1), \ldots, \phi(x_m)\right]^\top \left[\phi(x_1), \ldots, \phi(x_m)\right] \\
&\approx K_{mn}^\top K_{nn}^{-1} K_{mn} \\
&= \left[K_{nn}^{-1} K_{mn}\right]^\top K_{nn} \left[K_{nn}^{-1} K_{mn}\right]
\end{aligned}
$$

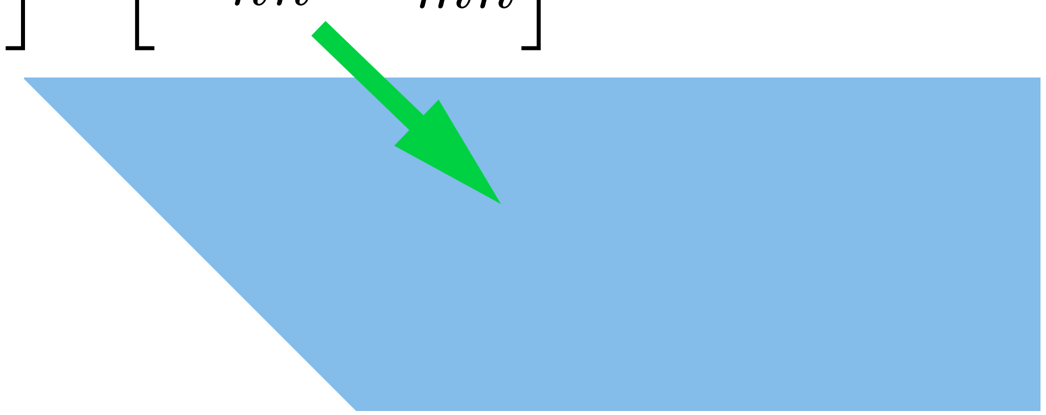


Subspace Finding

- Incomplete Cholesky factorization

$$
\begin{aligned}
K &= \left[\phi(x_1), \ldots, \phi(x_m)\right]^\top \left[\phi(x_1), \ldots, \phi(x_m)\right] \\
&\approx K_{mn}^\top K_{nn}^{-1} K_{mn} \\
&= \left[K_{nn}^{-1} K_{mn}\right]^\top K_{nn} \left[K_{nn}^{-1} K_{mn}\right] \\
&= \left[K_{nn}^{-\frac{1}{2}} K_{mn}\right]^\top \left[K_{nn}^{-\frac{1}{2}} K_{mn}\right]
\end{aligned}
$$





Picking the Subset

- Variant 1 (‘Nystrom’ Method)
  Pick random directions (not so great accuracy)
- Variant 2 (Brute force)
  Try out all directions (very expensive)
- Variant 3 (Tails)
  Pick 59 random candidates. Keep best (better)
- Variant 4 (Positive diagonal pivoting)
  Pick term with largest residual. As good (or better) than 59 random terms, much chaper

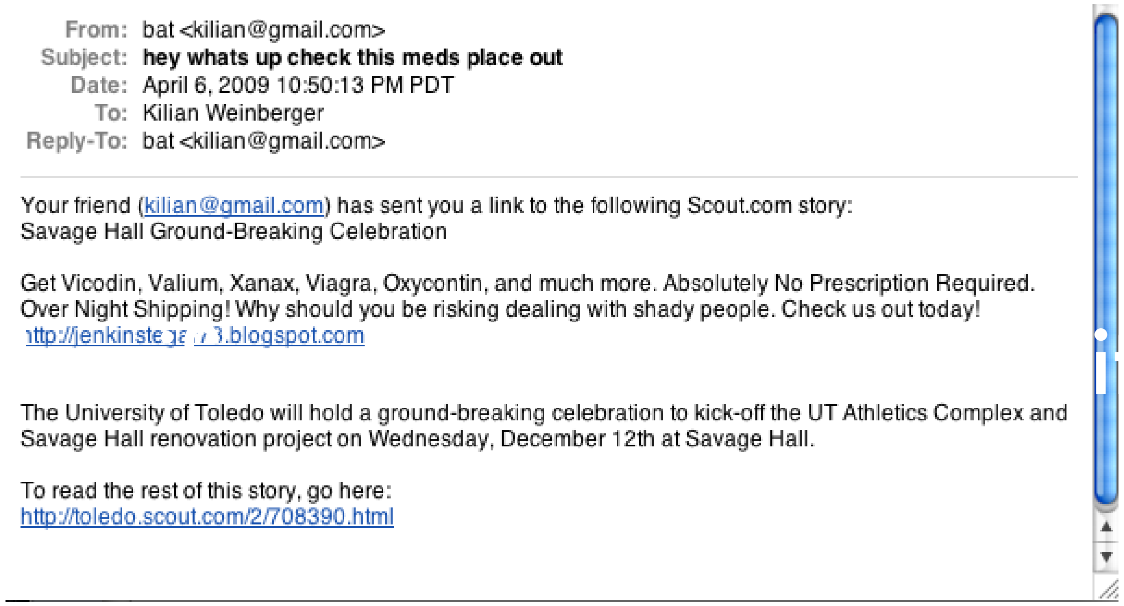
Function values

Basic Idea

- Exploit matrix vector operations
  - In some kernels $K\alpha$ is cheap
  - In others kernel inverse is easy to compute (e.g. inverse graph Laplacian) $K^{-1}y$
- Variable substitution in terms of y
- Solve decomposing optimization problem (this can be orders of magnitude faster)
- Example - spam filtering on webgraph. Assume that linked sites have related spam scores.

Motivation: Multitask Learning

Spam Classification



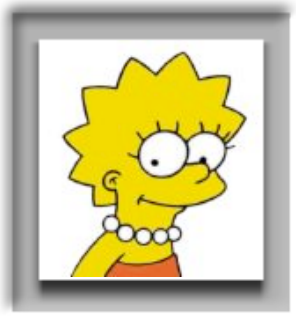
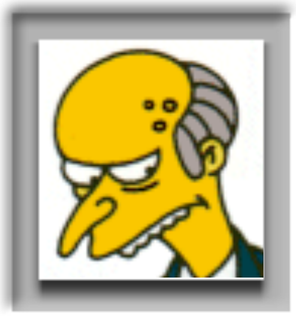
From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

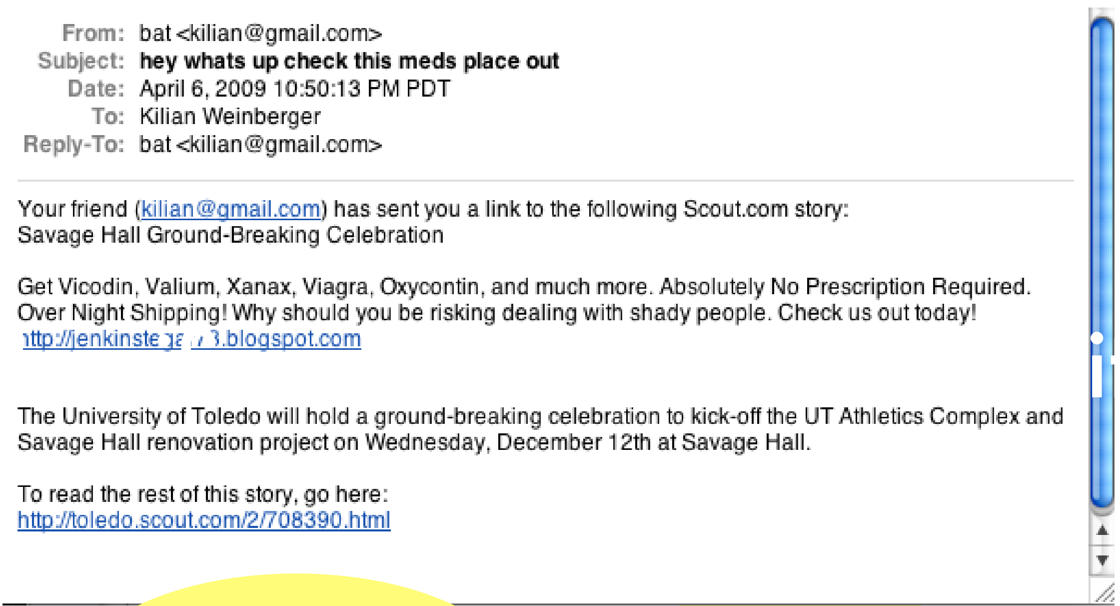
Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required. Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
ttp://jenkinste[illegible]3.blogspot.com

The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html

Spam Classification

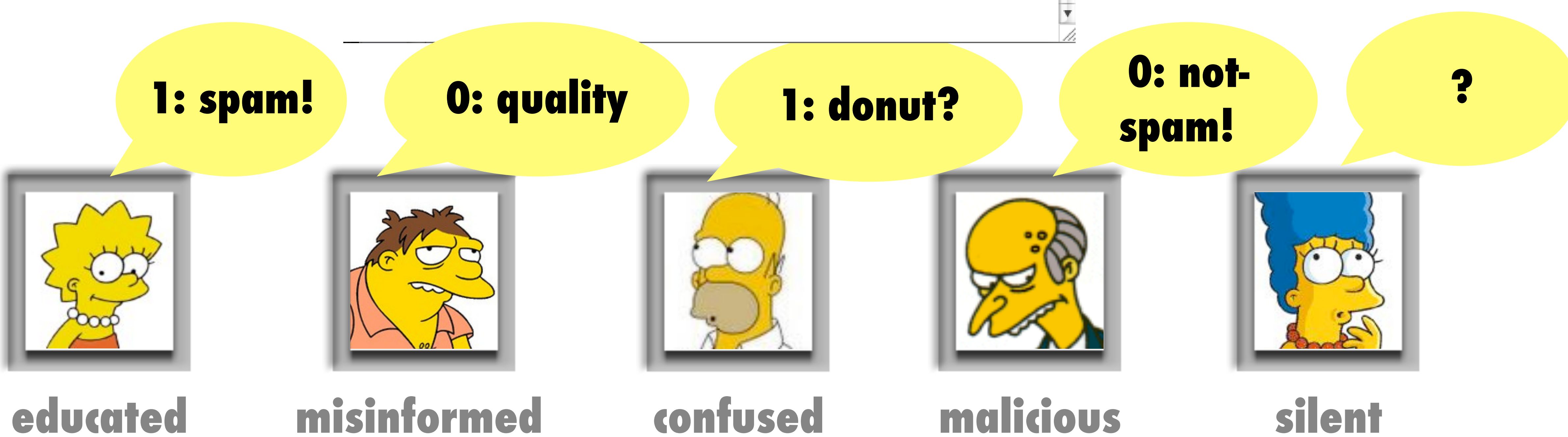

From: bat <kilian@gmail.com>
Subject: **hey whats up check this meds place out**
Date: April 6, 2009 10:50:13 PM PDT
To: Kilian Weinberger
Reply-To: bat <kilian@gmail.com>

Your friend (kilian@gmail.com) has sent you a link to the following Scout.com story:
Savage Hall Ground-Breaking Celebration

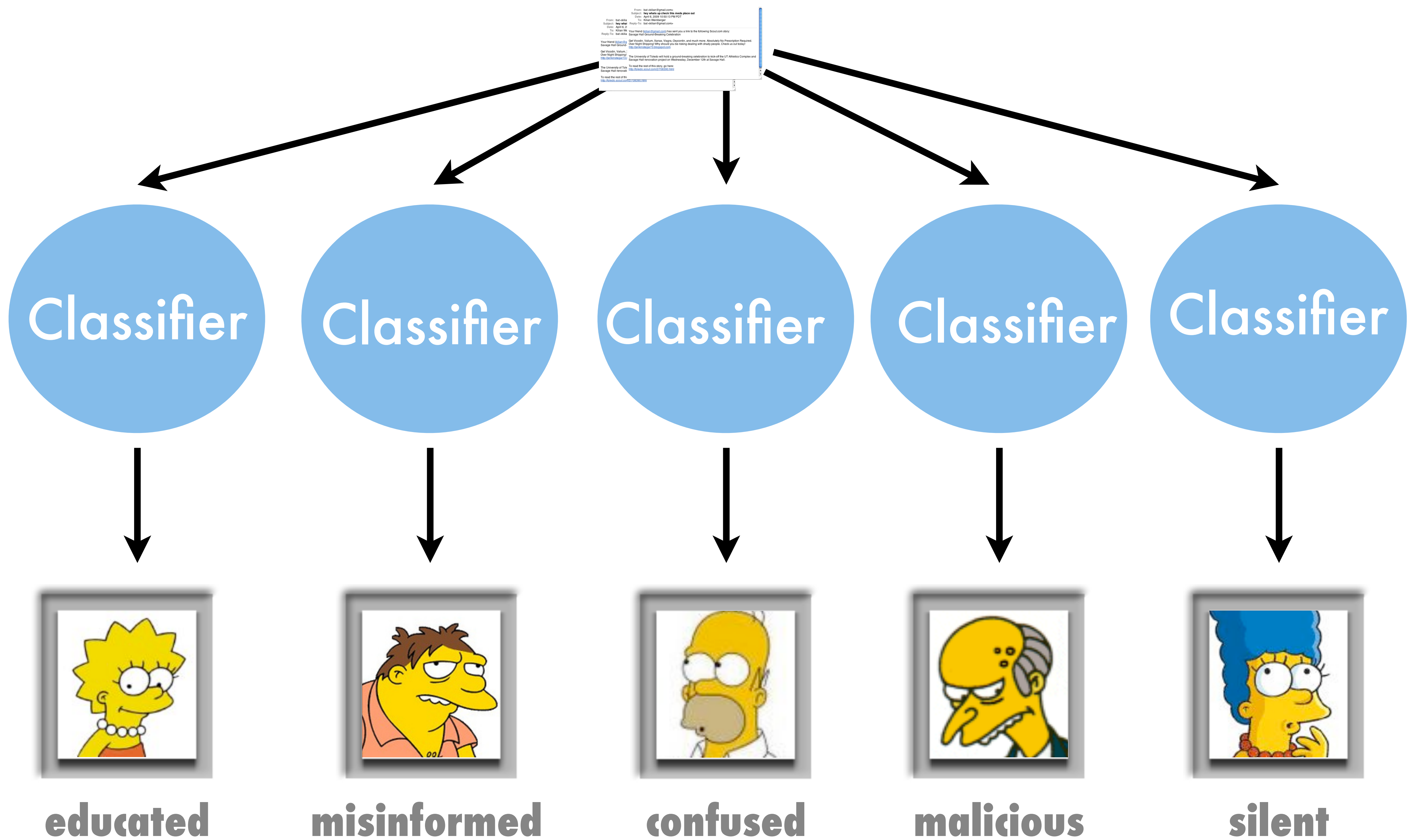
Get Vicodin, Valium, Xanax, Viagra, Oxycontin, and much more. Absolutely No Prescription Required. Over Night Shipping! Why should you be risking dealing with shady people. Check us out today!
http://jenkinste[illegible]3.blogspot.com

The University of Toledo will hold a ground-breaking celebration to kick-off the UT Athletics Complex and Savage Hall renovation project on Wednesday, December 12th at Savage Hall.

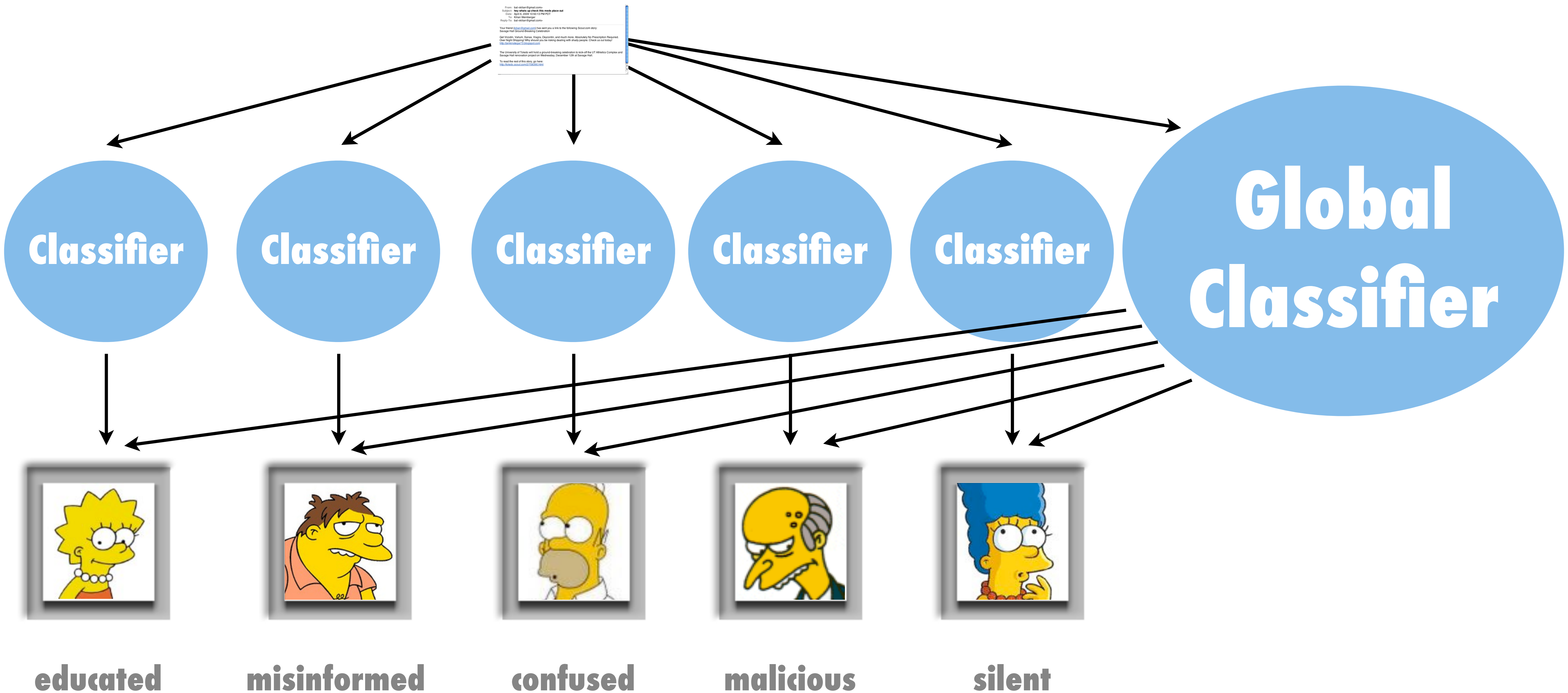
To read the rest of this story, go here:
http://toledo.scout.com/2/708390.html



Spam Classification



Multitask Learning



Collaborative Classification

- **Primal representation**

$$f(x, u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

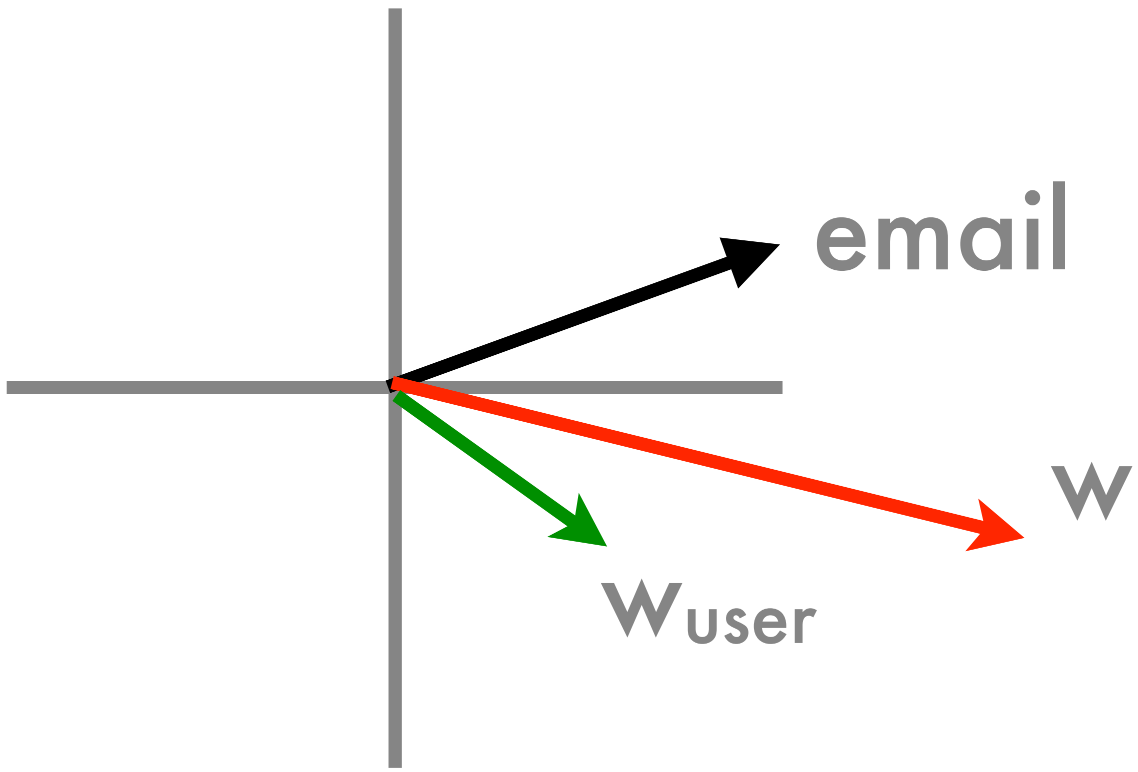
  Kernel representation

$$k((x, u), (x', u')) = k(x, x')[1 + \delta_{u,u'}]$$

  Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal representation**

$$f(x, u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

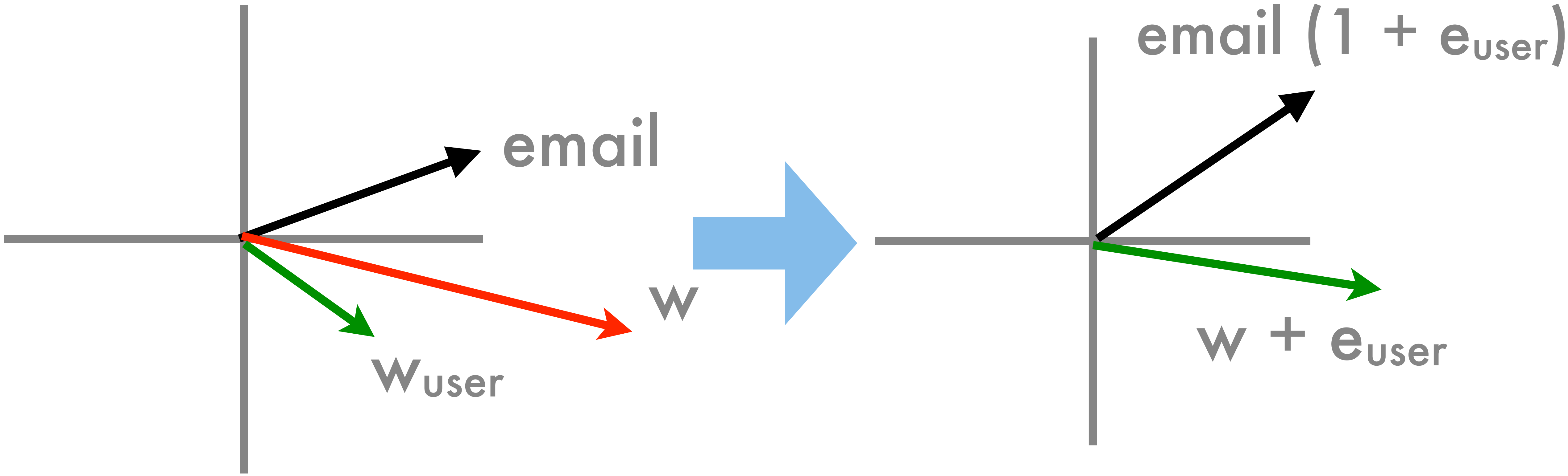
Kernel representation

$$k((x, u), (x', u')) = k(x, x')[1 + \delta_{u,u'}]$$

Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

Collaborative Classification



- **Primal representation**

$$f(x, u) = \langle \phi(x), w \rangle + \langle \phi(x), w_u \rangle = \langle \phi(x) \otimes (1 \oplus e_u), w \rangle$$

Kernel representation

$$k((x, u), (x', u')) = k(x, x')[1 + \delta_{u,u'}]$$

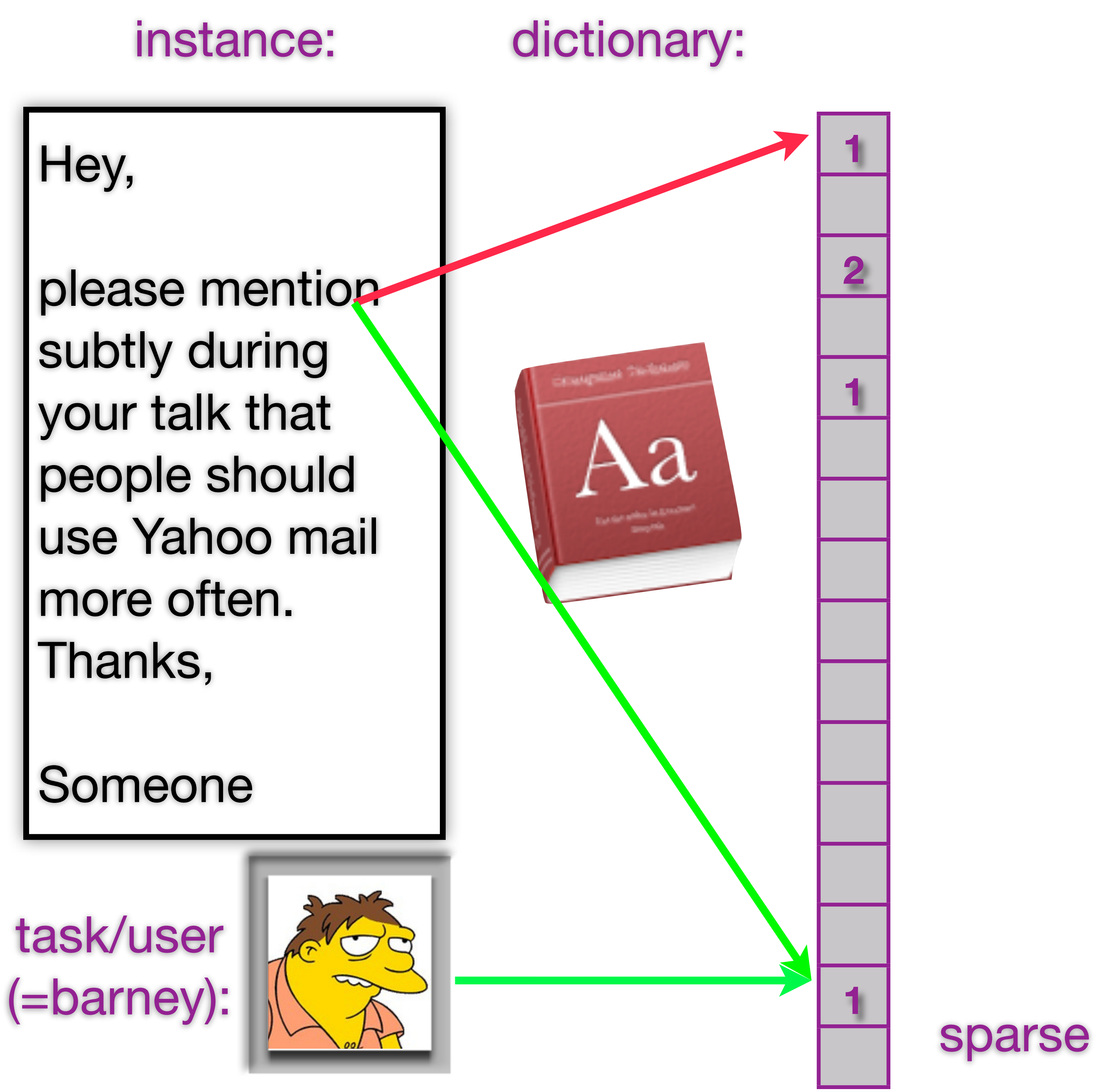
Multitask kernel (e.g. Pontil & Michelli, Daume). Usually does not scale well ...

- **Problem -** dimensionality is 10^{13}. That is 40TB of space

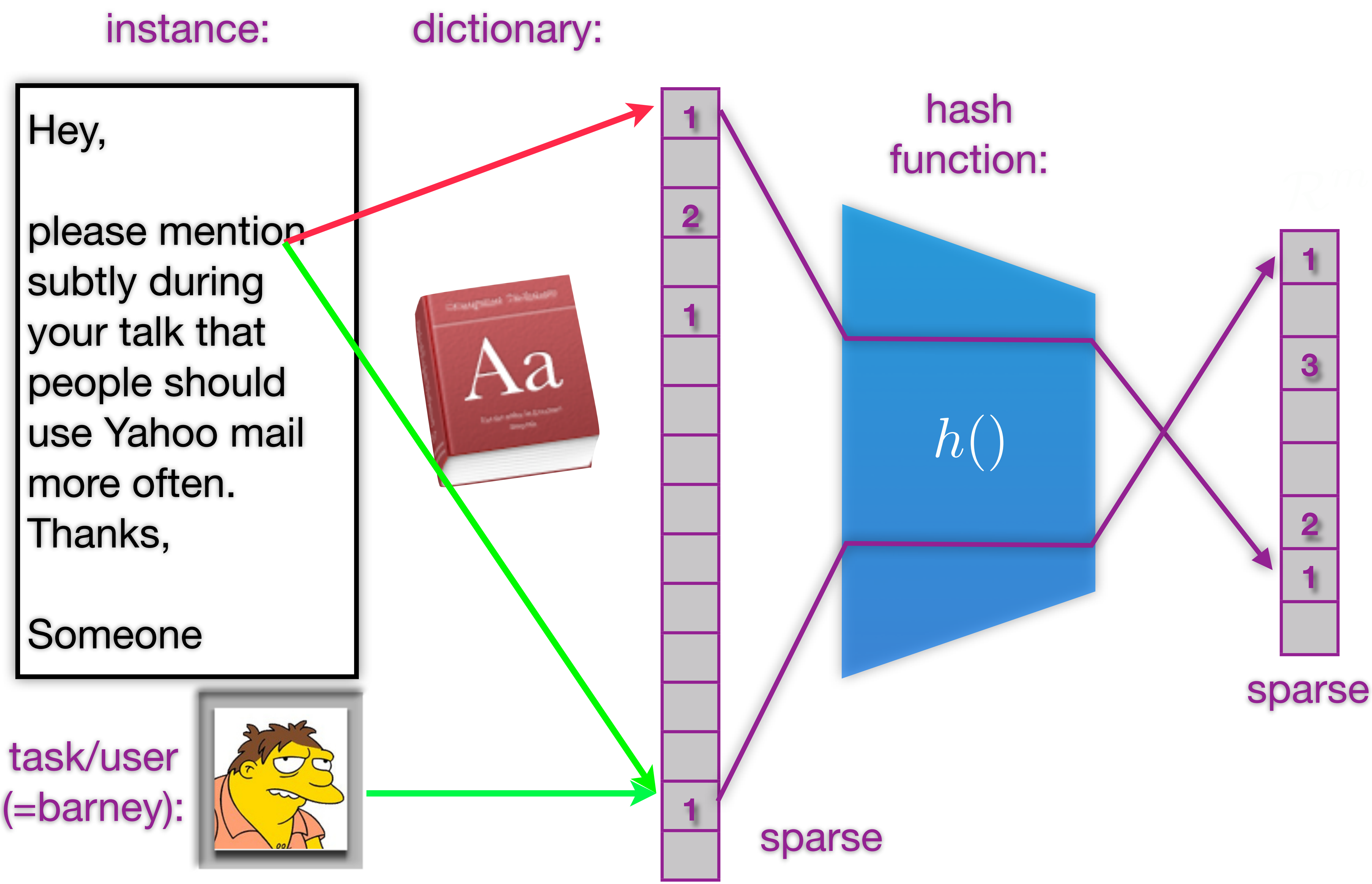
Hashing

Hash Kernels

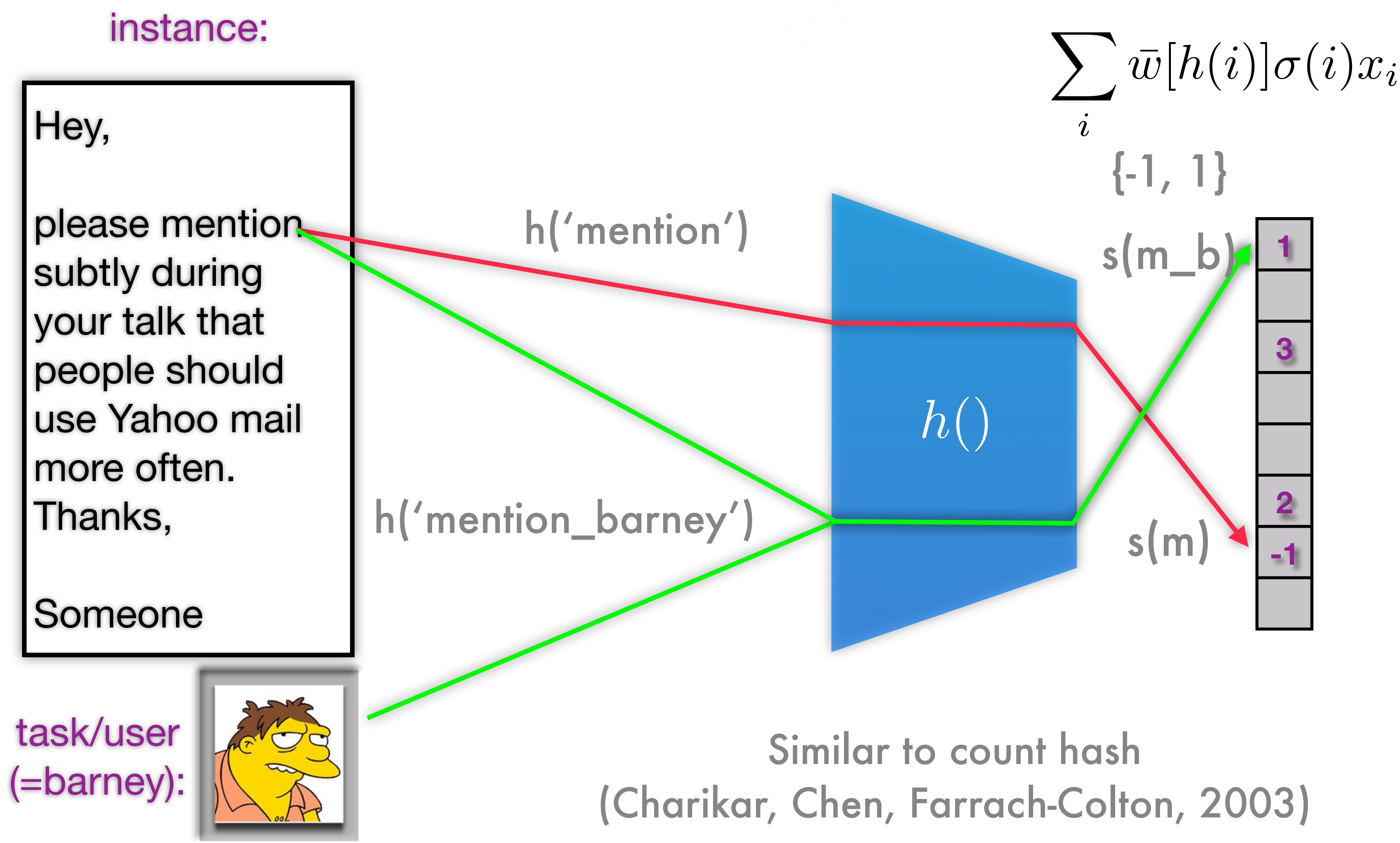
Hash Kernels



Hash Kernels



Hash Kernels



Similar to count hash
(Charikar, Chen, Farrach-Colton, 2003)

Advantages of hashing

Advantages of hashing

- **No dictionary**!
  - Content drift is no problem
  - All memory used for classification
  - Finite memory guarantee (with online learning)

Advantages of hashing

- **No dictionary**!
  - Content drift is no problem
  - All memory used for classification
  - Finite memory guarantee (with online learning)
- **No Memory** needed for projection. (vs LSH)

Advantages of hashing

- **No dictionary**!
  - Content drift is no problem
  - All memory used for classification
  - Finite memory guarantee (with online learning)
- **No Memory** needed for projection. (vs LSH)
- **Implicit** mapping into high dimensional space!

Advantages of hashing

- **No dictionary**!
  - Content drift is no problem
  - All memory used for classification
  - Finite memory guarantee (with online learning)

- **No Memory** needed for projection. (vs LSH)
- **Implicit** mapping into high dimensional space!
- It is **sparsity preserving**! (vs LSH)

Inner product preserving

- Unhashed inner product

$$\langle w, x \rangle = \sum_i w_i x_i$$

- Hashed inner product

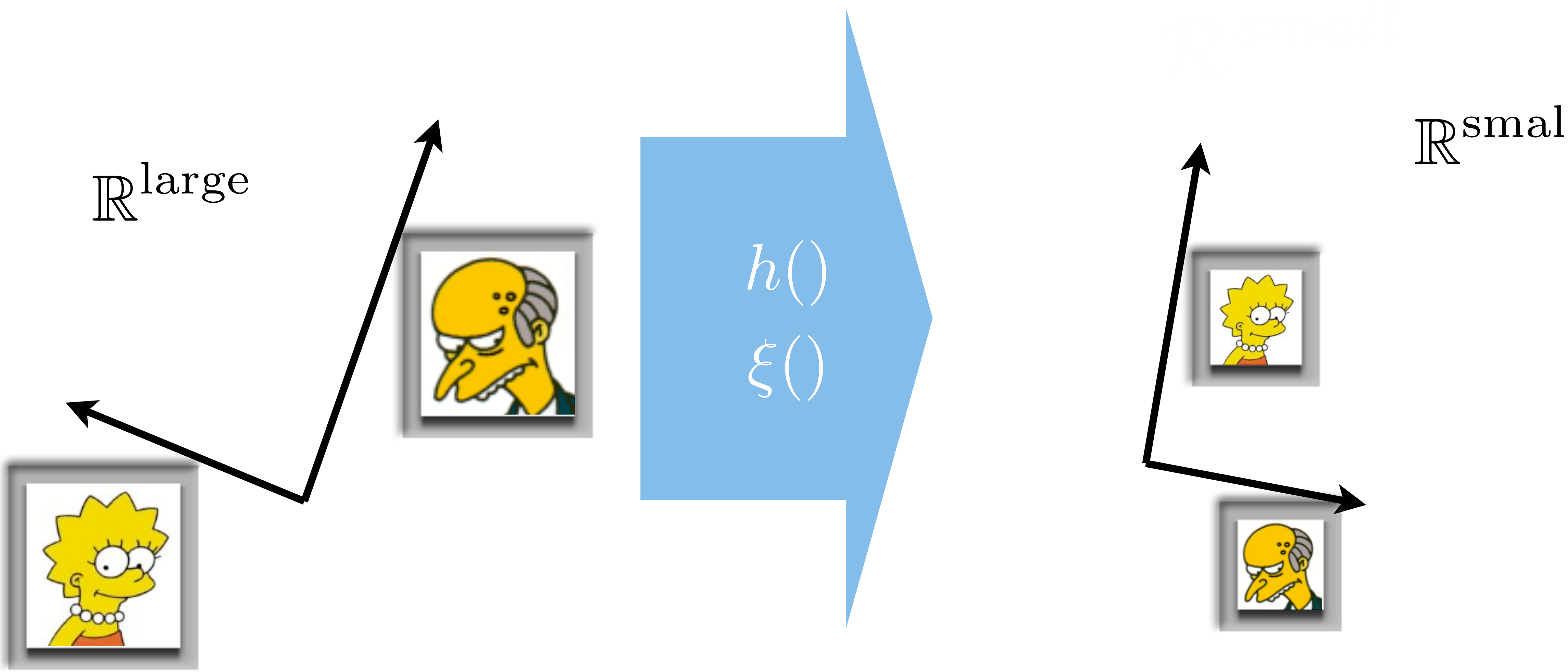
$$\langle \bar{w}, \bar{x} \rangle = \sum_j \left[\sum_{i:h(i)=j} w_i \sigma(i) \right] \left[\sum_{i:h(i)=j} x_i \sigma(i) \right]$$

- Taking expectations

$$\mathbf{E}_\sigma [\sigma(i) \sigma(i')] = \delta_{ii'}$$

hence inner product is preserved in expectation

Approximate Orthogonality



We can do multi-task learning!

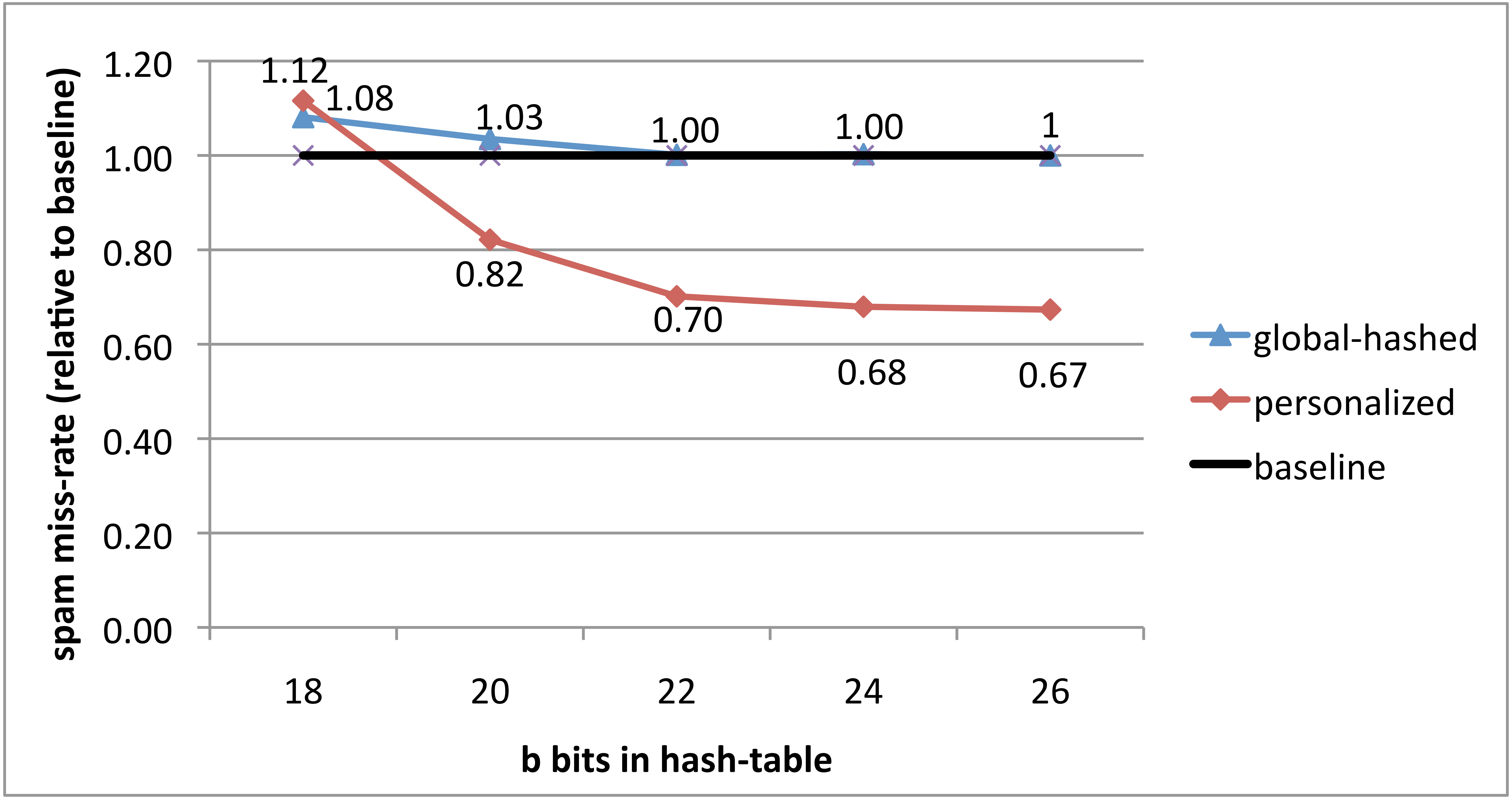
Guarantees

- For a random hash function the inner product vanishes with high probability via
$$\Pr\{|\langle w_v, h_u(x)\rangle| > \epsilon\} \leq 2e^{-C\epsilon^2 m}$$
- We can use this for multitask learning

Direct sum in Hilbert Space → **Sum** in Hash Space

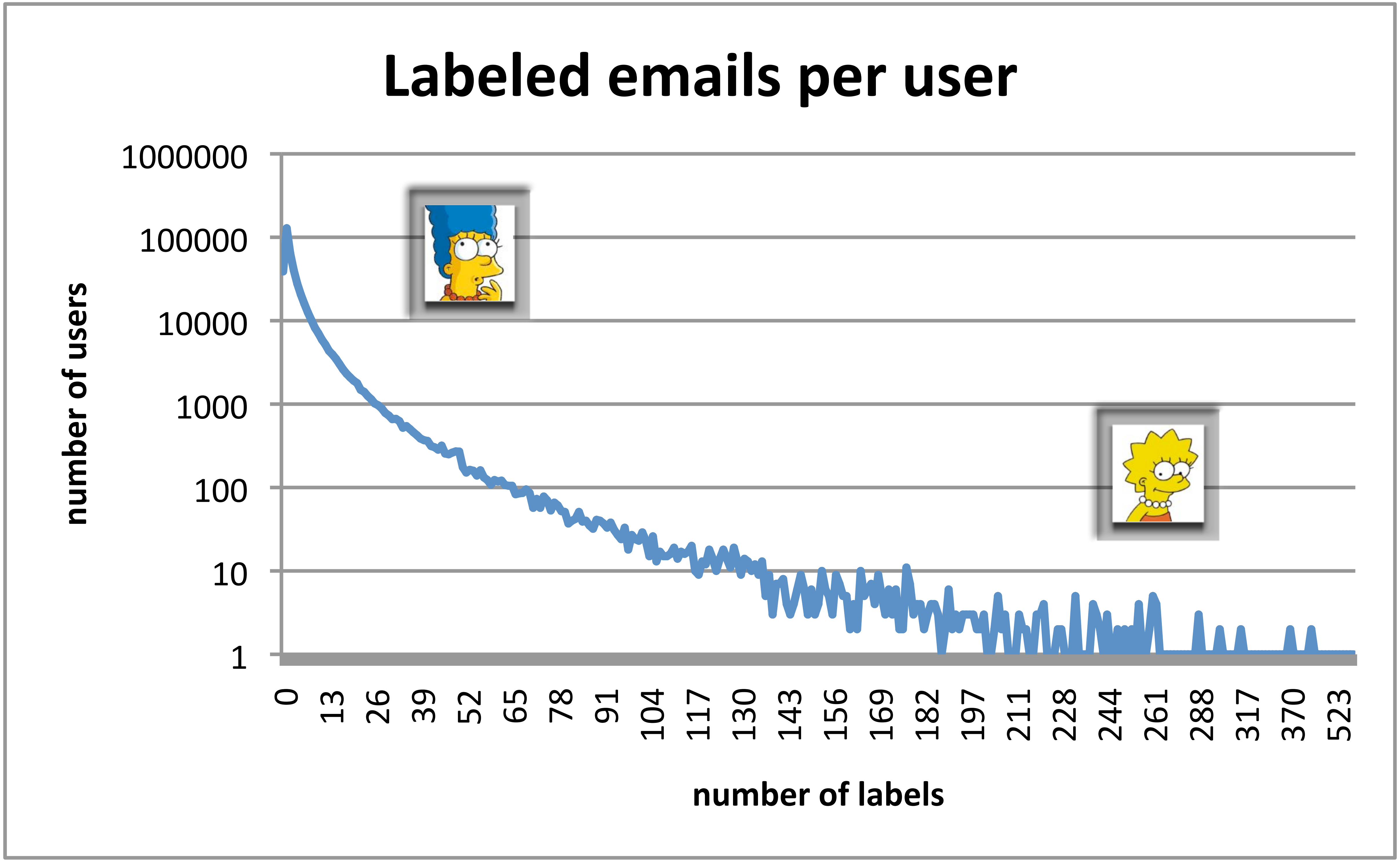
- The hashed inner product is unbiased
Proof: take expectation over random signs
- The variance is O(1/n)
Proof: brute force expansion
- Restricted isometry property (Kumar, Sarlos, Dasgupta 2010)

Spam classification results



N=20M, U=400K

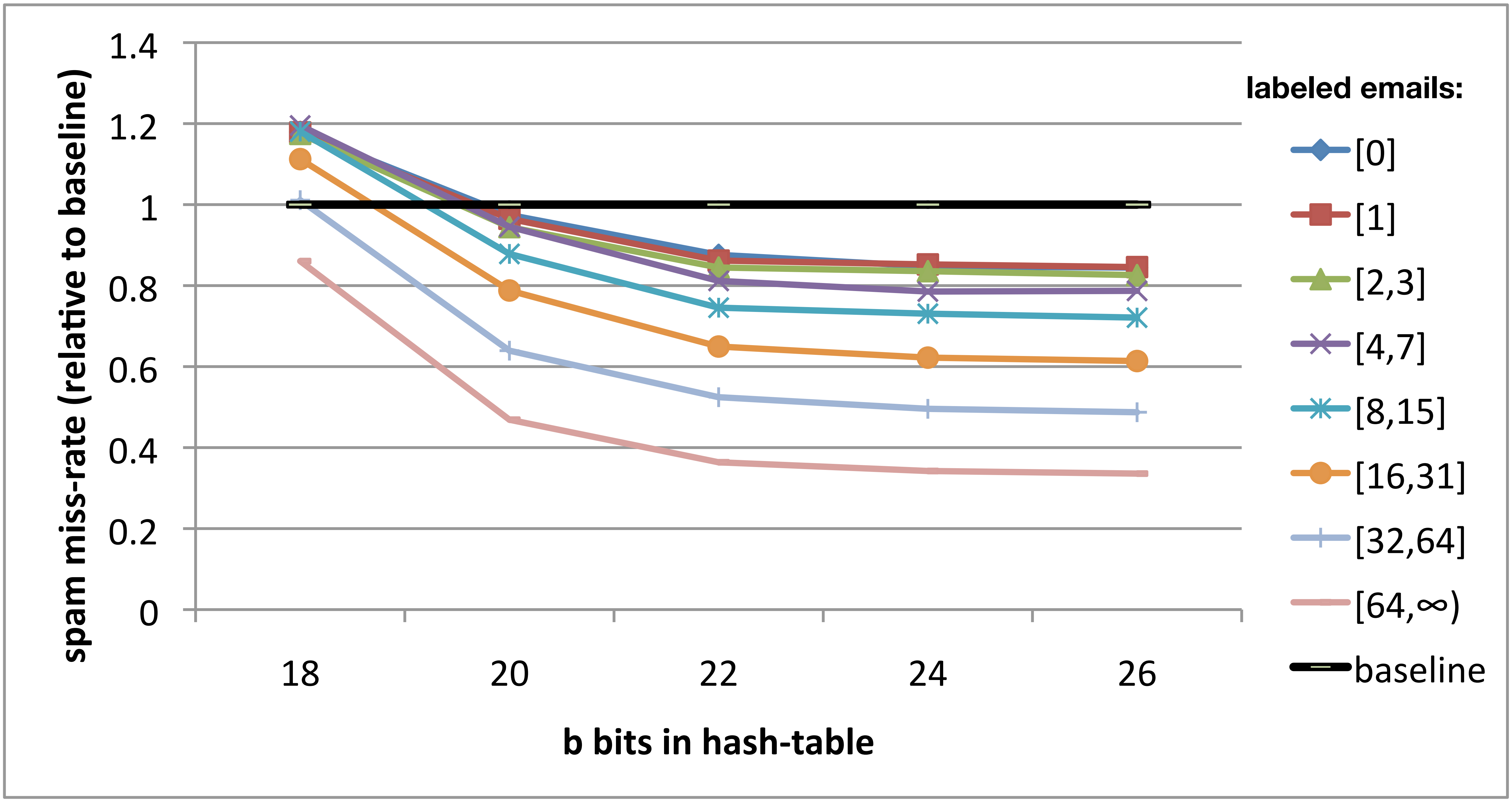
Lazy users ...



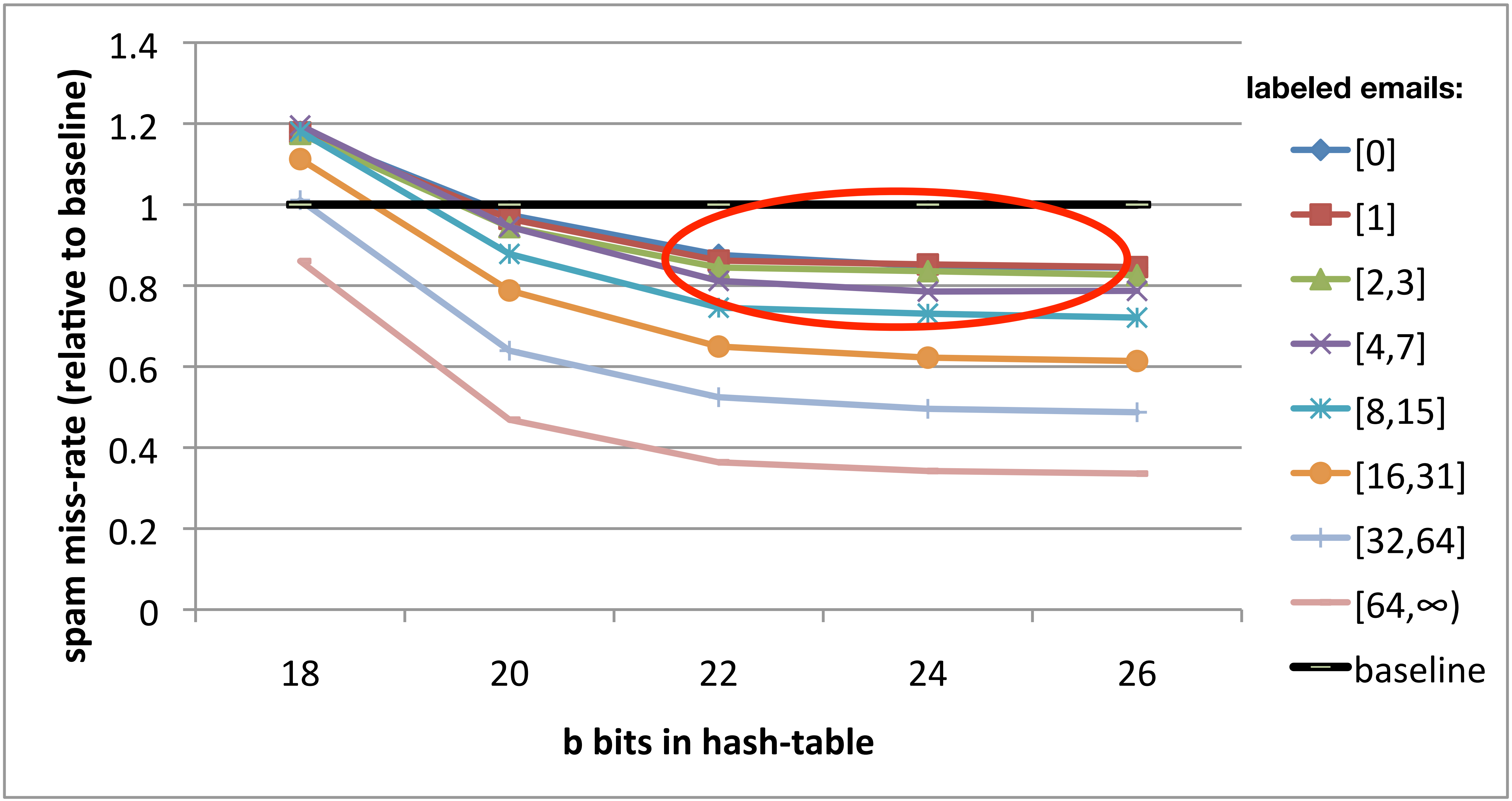
Labeled emails per user
1000000
100000
10000
1000
100
10
1
number of users
0
13
26
39
52
65
78
91
104
117
130
143
156
169
182
197
211
228
244
261
288
317
370
523
number of labels

Results by user group

Results by user group



Results by user group



Approximate String Matches

- General idea

$$k(x, x') = \sum_{w \in x} \sum_{w' \in x'} \kappa(w, w') \text{ for } |w - w'| \leq \delta$$

Berkeley
B3rkeley
Berkely
8erkeley
Berkley
Berke1ey

gotta catch them all

Approximate String Matches

- General idea

$$k(x, x') = \sum_{w \in x} \sum_{w' \in x'} \kappa(w, w') \text{ for } |w - w'| \leq \delta$$

- Simplification
  - Weigh by mismatch amount |w-w'|
  - Map into fragments: `dog -> (*og, d*g, do*)`
  - Hash fragments and weigh them based on mismatch amount
  - Exponential in amount of mismatch
    But not in alphabet size

Approximate String Matches

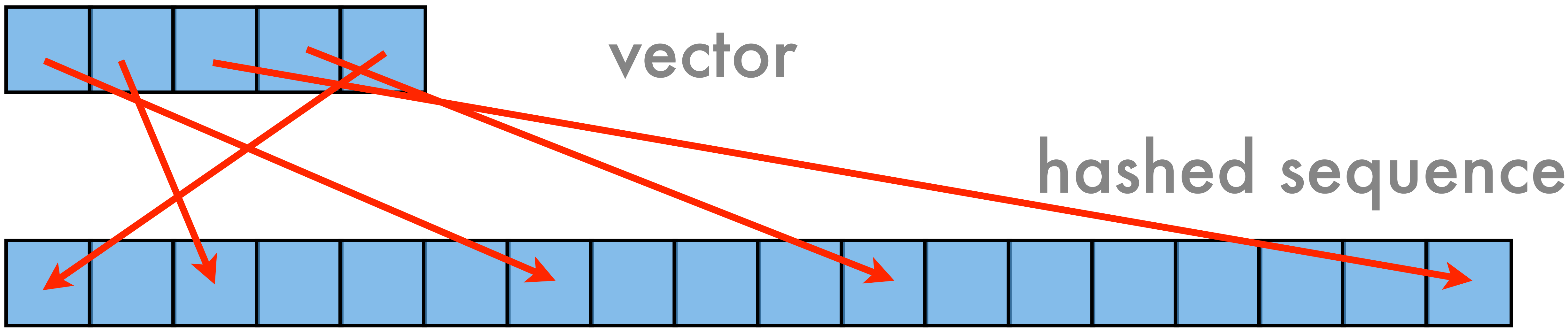
- General idea

$$k(x, x') = \sum_{w \in x} \sum_{w' \in x'} \kappa(w, w') \text{ for } |w - w'| \leq \delta$$

Berkeley	
B3rkeley	B*rkeley
Berkely	Berkel*y
8erkeley	*erkeley
Berkley	Berk*ley
Berke1ey	Berke*ey

Memory access patterns

- Cache size is a few MBs
  Very fast random memory access
- RAM (DDR3 or better) is GBs
  - Fast sequential memory access (burst read)
  - CPU caches memory read from RAM
  - Random memory access is very slow
  - CPU caches memory read from RAM



Speeding up access

- Key idea - bound the range of h(i,j)
- Linear offset
  bad collisions in i
- Sum of hash functions
  bad collisions in j
- Optimal Golomb Ruler (Langford)
  NP hard in general
- Feistel Network / Cryptography (new)

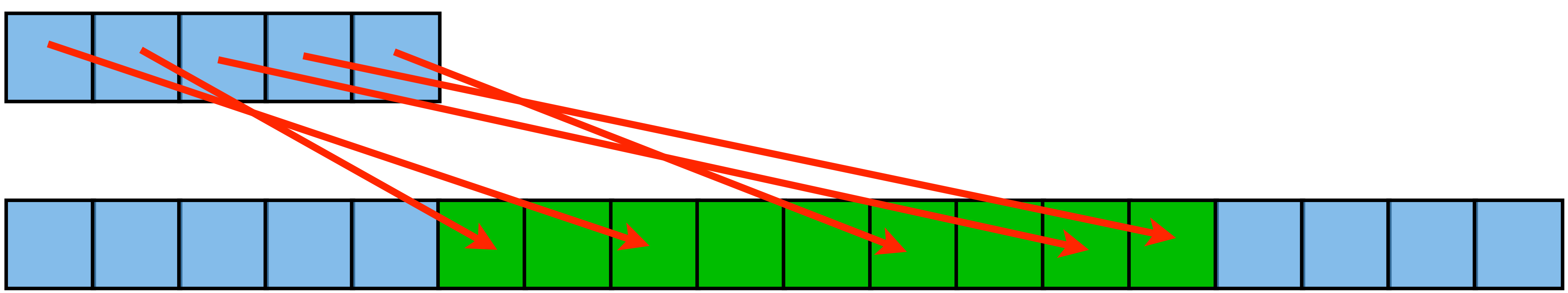
for j=1 to n access h(i,j)

$$h(i,j) = h(i) + j$$

$$h(i,j) = h(i) + h'(j)$$

$$h(i,j) = h(i) + \mathrm{OGR}(j)$$

$$h(i,j) = h(i) + \mathrm{crypt}(j|i)$$



Structured Estimation

Large Margin Classifiers

- Large Margin without rescaling (convex) (Guestrin, Taskar, Koller)

$$l(x, y, f) = \sup_{y' \in \mathcal{Y}} \left[f(x, y') - f(x, y) + \Delta(y, y') \right]$$

- Large Margin with rescaling (convex) (Tsochantaridis, Hofmann, Joachims, Altun)

$$l(x, y, f) = \sup_{y' \in \mathcal{Y}} \left[f(x, y') - f(x, y) + 1 \right] \Delta(y, y')$$

- Both losses majorize misclassification loss

$$\Delta\left(y, \operatorname*{argmax}_{y'} f(x, y')\right)$$

- Proof by plugging argmax into the definition

Recipe

1. Identify estimation problem with structured y
2. Design function f(x,y) efficiently maximized in y
3. Design linear function space for f
4. Design tractable loss Δ(y,y')

5. Solve optimization problem

$$\operatorname*{argmax}_{y'} f(x, y') + \Delta(y, y')$$

6. Write a paper ...

Graph Matching

Graph Matching

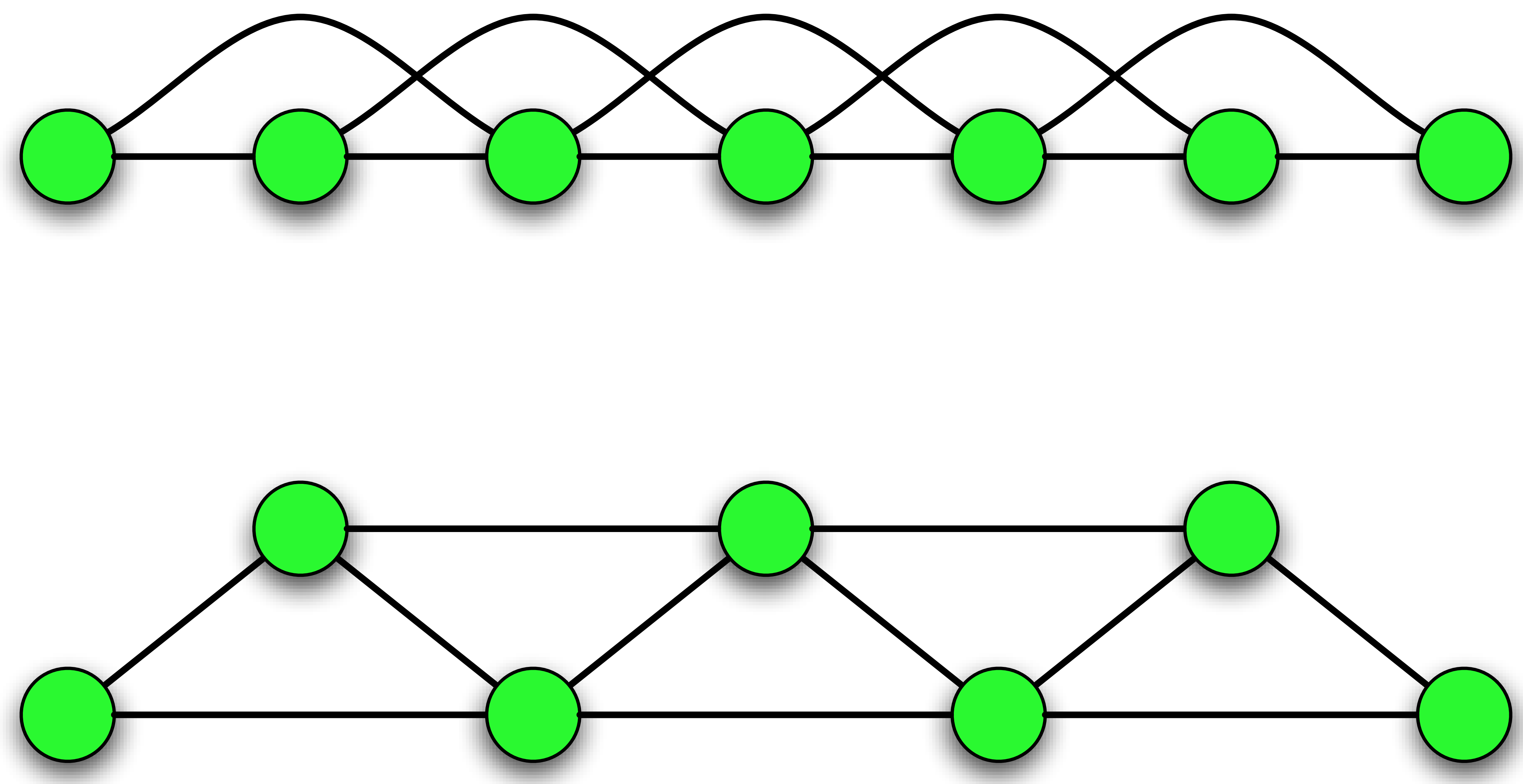
Chemistry and Biology

- Molecules stored in database
- Regulatory networks
- Function estimation for proteins

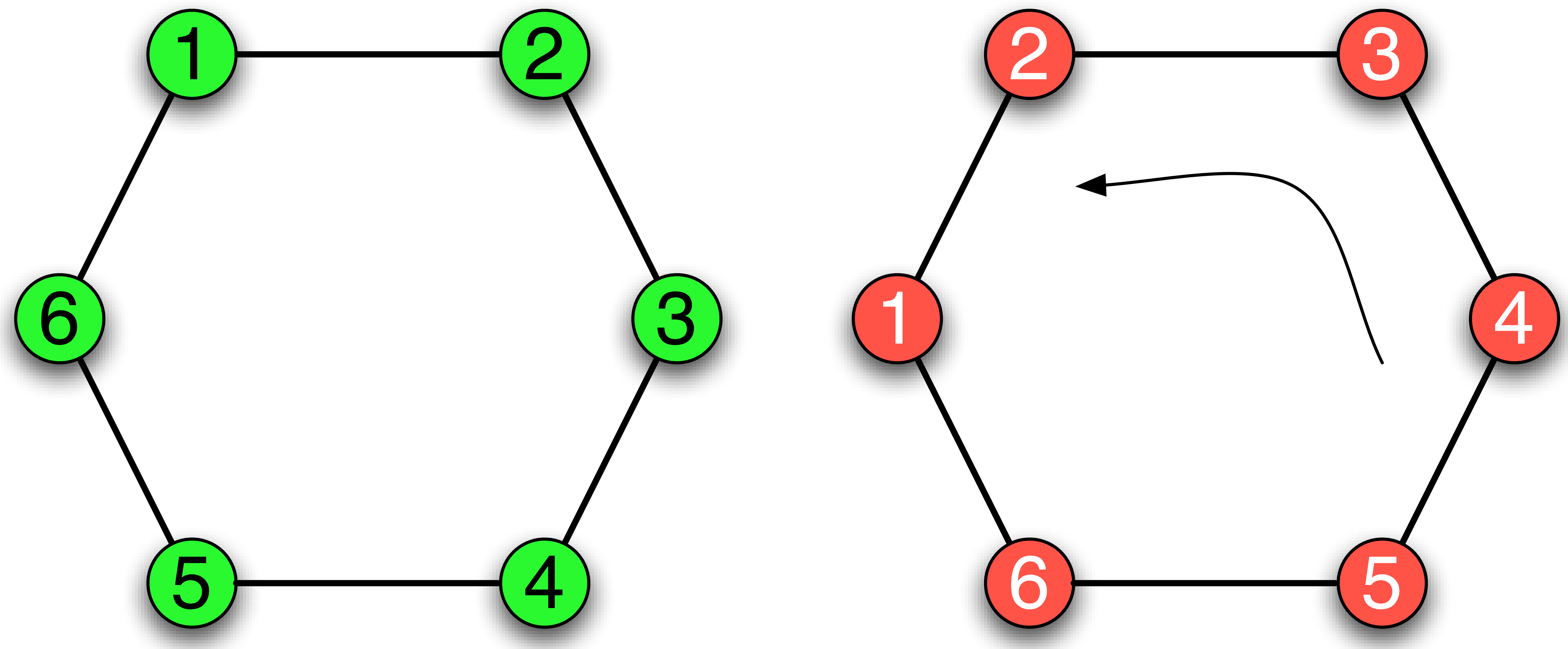
Computer Vision

- Object matching (e.g. wide baseline match)
- Preprocessing for camera calibration
- 3D reconstruction
- Match maps to aerial photographs (automatic map updates)

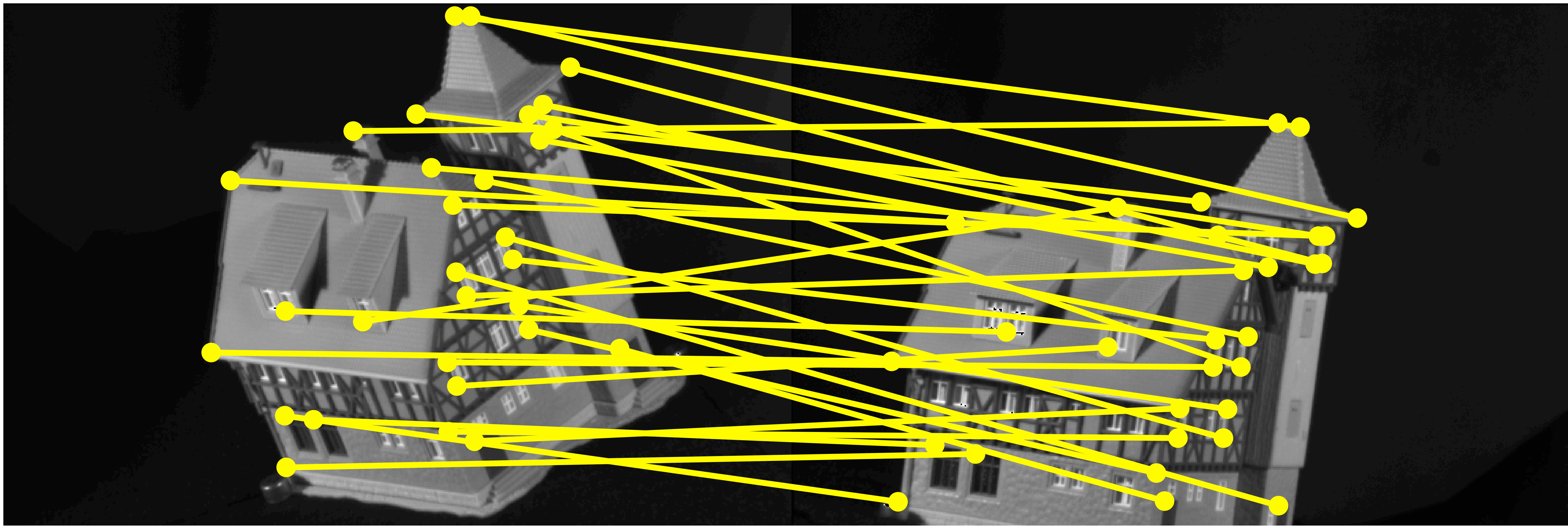
Identical Graphs



Ambiguities

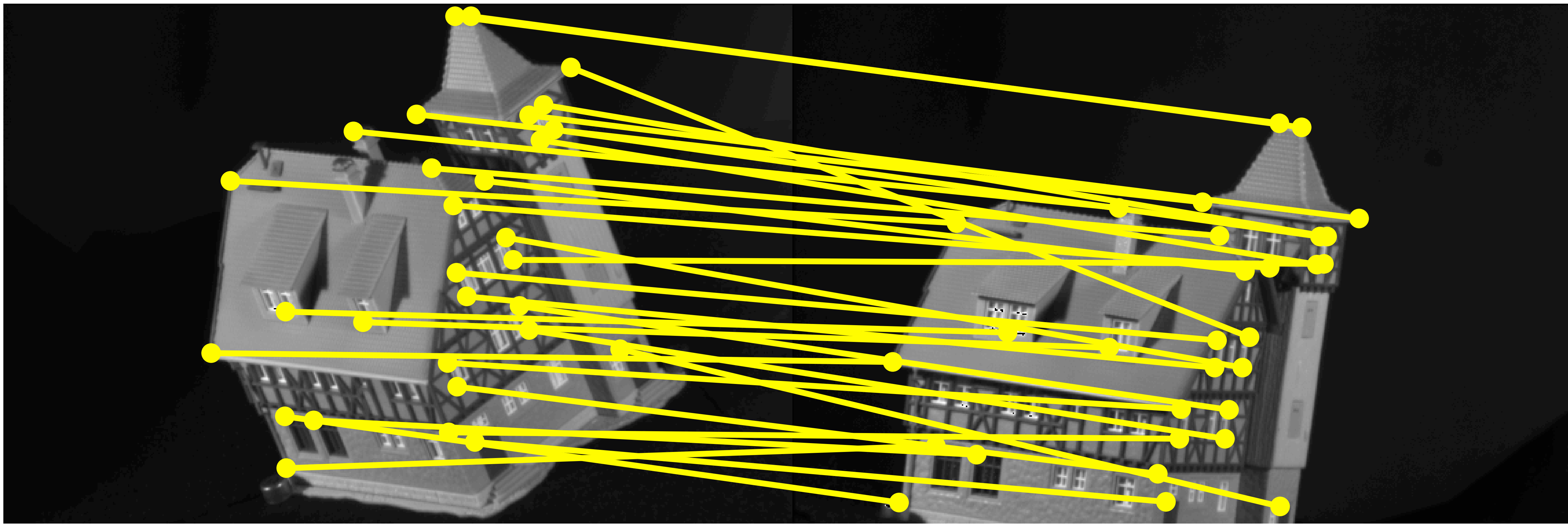


Computer vision



- Graph matching via quadratic assignment is NP hard
- Can we learn a linear assignment function?

Computer vision



- Graph matching via quadratic assignment is NP hard
- Can we learn a linear assignment function?

Recipe

1. Identify estimation problem with structured y

   Graph Matching

Problems

Hardness

No currently known polynomial time algorithm for matching.
Checking is linear in the number of edges.

Completeness

- The graphs may not be identical
- We just may want to find a "best match"
- Problem often ill-defined (e.g. largest common subgraph, best matches overall, etc.)

Attributes

- SIFT features — unlikely to be identical at all
- Different image resolutions (e.g. different cameras)
- Different image content (e.g. black and white vs. color)
- Different representation (e.g. pixels vs. symbolic)

Size

For very large graphs heuristics are popular.

Good News

Key observation

Graph matching often needed only for a **restricted domain**.

Idea

- Graph matching on restricted subset of graphs is often **much** easier.
- Attributes in graphs can help a lot (e.g. Bunke's work for uniquely attributed vertices — matching becomes trivial)
- Local neighborhood may be sufficient for matching.

Strategy

- Use examples of matched graphs. Trivial if both graphs are of the same type: only need collection of graphs, no labeling needed.
- For corresponding objects of different representations training data is needed. Also if we want system to have a robust attribute matching function.

Linear Assignment

Notation

- Graphs G and G' with vertices V, V' and edges E, E'.
- We use $G_{ij} = 1$ to denote presence of an edge between i and j (and $G_{ij} = 0$ to denote its absence).
- V_i denotes vertex i (and its attributes)
- Permutation matrix Π describing match between G and G' with $\Pi_{ij} \in \{0; 1\}$ and $\Pi 1 = \Pi^\top 1 = 1$.

Objective Function

- Score C_{ij} for match between vertex V_i and V'_j.
- Best assignment by solving

$$\underset{\Pi}{\text{minimize}} \sum_{i,j} \Pi_{ij} C_{ij}$$

- For uniquely attributed graphs (trivial) we set $C_{ij} = \delta_{V_i, V'_i}$.

Linear Assignment

Integer Program

$$\underset{\Pi}{\text{minimize}} \sum_{i,j} \Pi_{ij} C_{ij} \text{ subject to } \Pi_{ij} \in \{0; 1\} \text{ and } \Pi 1 = \Pi^\top 1 = 1$$

Linear Programming Relaxation

$$\underset{\Pi}{\text{minimize}} \sum_{i,j} \Pi_{ij} C_{ij} \text{ subject to } \Pi_{ij} \in [0, 1] \text{ and } \Pi 1 = \Pi^\top 1 = 1$$

Properties

- Can be solved in polynomial time (e.g. interior point)
- All vertices are **integral**, hence the two problems are **equivalent**.
- Fast shortest path solvers available.
- Adding prior knowledge is easy — clamp Π_{ij} to 0 or 1.

Recipe

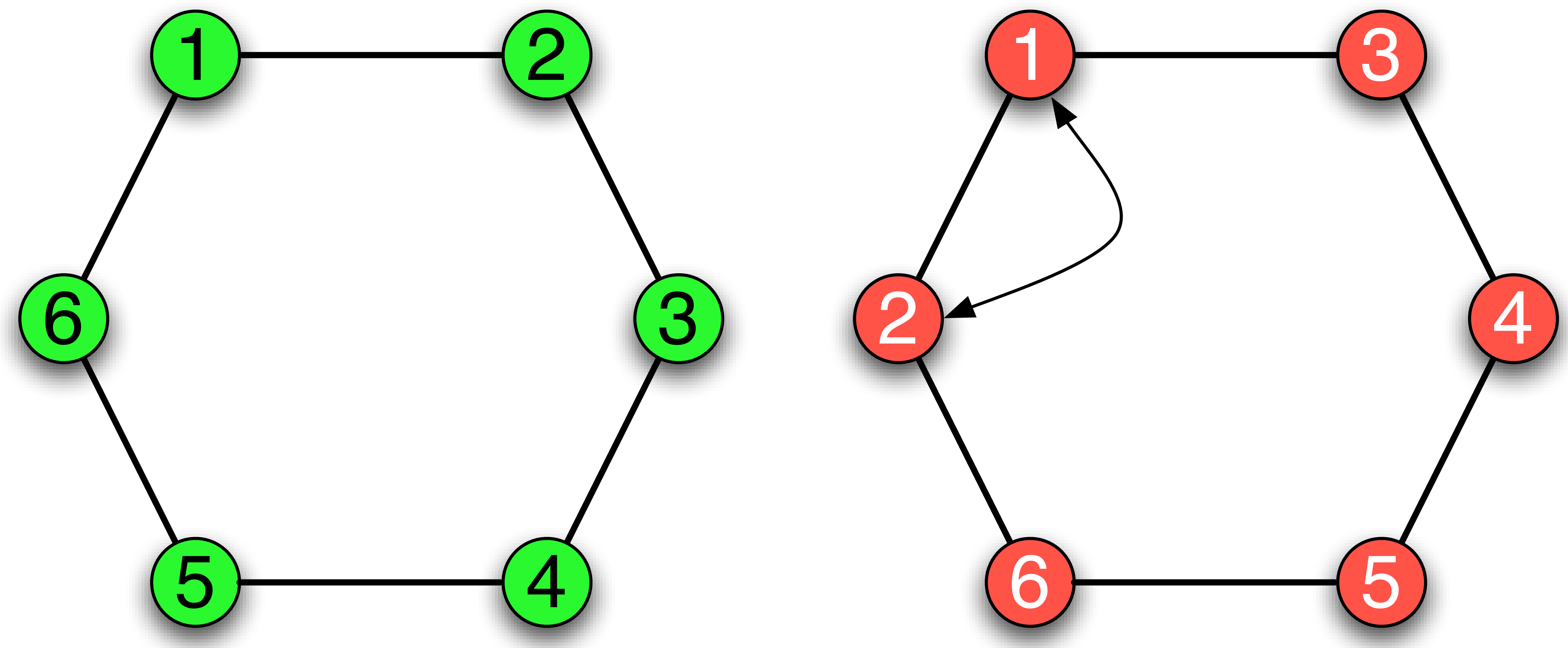
1. Identify estimation problem with structured y
2. Design function f(x,y) efficiently maximized in y

$$\text{maximize}\, \mathrm{tr}\, C\pi$$

$$\text{subject to } \sum_i \pi_{ij} = \sum_j \pi_{ij} = 1 \text{ and } \pi_{ij} \geq 0$$

3. Linear function space is trivial (functions for entries of C)

Failure modes



Diagnosis

Why?

Graph matching is hard, so the Hungarian method (polynomial time algorithm) must fail.

What went wrong?

- Local features insufficient for matching.
- Symmetries create long range dependencies.
- Maybe we used the wrong matching score C_{ij}?

How bad is it really?

- Fails on degenerate problems with lots of symmetry.
- Works fine on graphs with enough characteristic features.
- **We should engineer C_{ij} for specific problems.**

Not a fix - Quadratic Assignment

Key Idea

Use edge features for match.

Optimization Problem

$$\underset{\Pi}{\text{minimize}} \sum_{i,j} C_{ij}\Pi_{ij} + \sum_{i,j,u,v} Q_{ij,uv}\Pi_{ij}\Pi_{uv}$$

Properties

- C_{ij} describes vertex feature match (as before)
- $Q_{ij,uv}$ describes agreement between (potential) edges (i, u) and (j, v).
- For $Q_{ij,uv} = 1 - \delta_{G_{iu}, G'_{jv}}$ we have exact matching.
- Problem is NP hard to solve.

Tools of the trade

Genetic algorithms

Tabu search

Ant colony systems

actual name of algorithm!

Any other really really desparate heuristic . . .

Graduated Assignment

- First order Taylor approximation of Quadratic Assignment problem is Linear Assignment problem.
- Take small steps.
- Iterative procedure (Sinkhorn, 1964) for small steps.

Semidefinite Relaxations

Not very scalable, $O(m^4)$ storage and $O(m^6)$ computation.

In practice . . .

Can only solve problems of size < 100.

Changing the question

Key Idea

- Exact graph matching is too expensive.
- Linear assignment works if matching scores are good.
- **Use data to learn matching scores C_{ij}.**

Bottom line

Work hard to ask the right question not to find the answer for the wrong question. Use structured estimation.
We get **problem dependent scores**.

Optimization Problem

Optimization Problem

$$\underset{C(\cdot,\cdot)}{\text{minimize}} \sum_{i=1}^{m} \Delta(\Pi^i, \mathbf{1}) \text{ where } \Pi^i = \underset{\Pi}{\text{argmin}} \sum_{uv} \Pi_{uv} C(V_u^i, V_v^i)$$

The goal is to find a compatibility function $C(\cdot,\cdot)$ such that graphs are perfectly matched. Obvious extensions for inexact matches — replace $\mathbf{1}$ by optimal match.

Loss Function

$$\Delta(\Pi, \Pi') = \|\Pi - \Pi'\|^2 = 2(n - \operatorname{tr} \Pi^\top \Pi')$$

Obviously other loss functions are possible.

Problem

The optimization is **nonconvex**. Even worse, it is **piecewise constant**. Risk of **overfitting**.

Recipe

1. Identify estimation problem with structured y
2. Design function f(x,y) efficiently maximized in y
3. Design linear function space for f
4. Design tractable loss Δ(y,y')

$$\Delta(\Pi, \Pi') = \|\Pi - \Pi'\|^2 = 2\left(n - \operatorname{tr}\Pi^\top \pi'\right)$$

Regularization

Parametric Model for C

$$C(V_u, V_j) = \langle \phi(V_u, V_j), w \rangle$$

Regularizer

Assume that small $\|w\|$ corresponds to smooth functions C.
Hence minimize regularized risk functional

$$\operatorname*{minimize}_{w} \sum_{i=1}^{m} \Delta(\Pi^i, \mathbf{1}) + \lambda \|w\|^2$$

Structured Estimation

Original Objective Function

$$\Delta(\Pi, 1) \text{ subject to } \Pi = \operatorname*{argmin}_{\Pi} \Pi^\top C$$

Convex Upper Bound

$$\xi \text{ where } \xi \geq \operatorname{tr}(\mathbf{1} - \Pi')^\top C + \Delta(\Pi', \mathbf{1}) \text{ for all } \Pi'.$$

To see that this is an upper bound, plug in $\Pi' = \Pi$. The problem is **convex** in ξ and C.

Optimization Problem

$$\operatorname*{minimize}_{w} \sum_{i=1}^{m} \xi_i + \lambda \|w\|^2$$

$$\text{subject to } \xi_i \geq \operatorname{tr}(\mathbf{1} - \Pi')^\top C(G^i, G^i) + 2(n - \operatorname{tr} \Pi'_i) \text{ for all } \Pi'.$$

Optimization

Issues

- Convex problem but ...
- Exponential number of constraints
- Need to find most violated constraints efficiently

Column Generation

- Maximizing the constraint is linear assignment problem

$$\underset{\Pi'}{\text{maximize}} - \operatorname{tr} {\Pi'}^\top [C(G^i, G^i) + 2 \cdot \mathbf{1}]$$

- Recall that $C(G^i, G^i)$ is a **compatibility** score.
- Problem made harder by adding $2 \cdot \mathbf{1}$ to enforce margin.

Algorithm

- Minimize w for given set of constraints
- Find next set of worst constraints

Recipe

1. Identify estimation problem with structured y
2. Design function f(x,y) efficiently maximized in y
3. Design linear function space for f
4. Design tractable loss Δ(y,y')

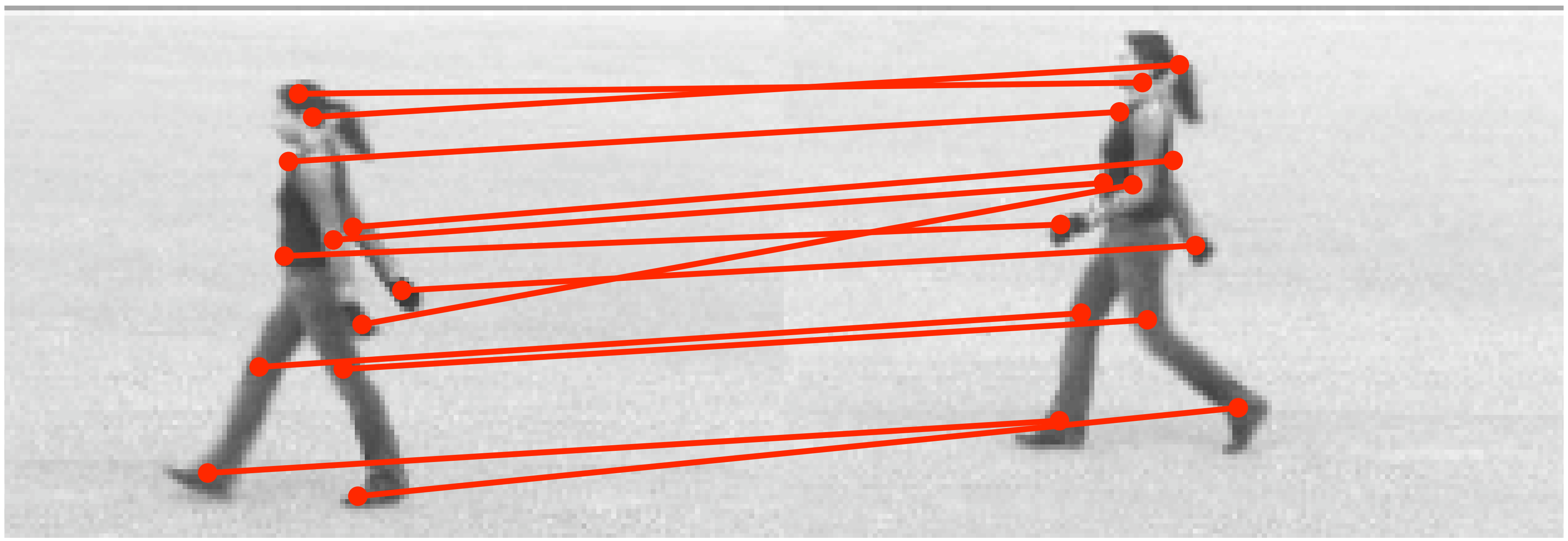
5. Solve optimization problem

$$\operatorname*{argmax}_{y'} f(x,y') + \Delta(y,y')$$

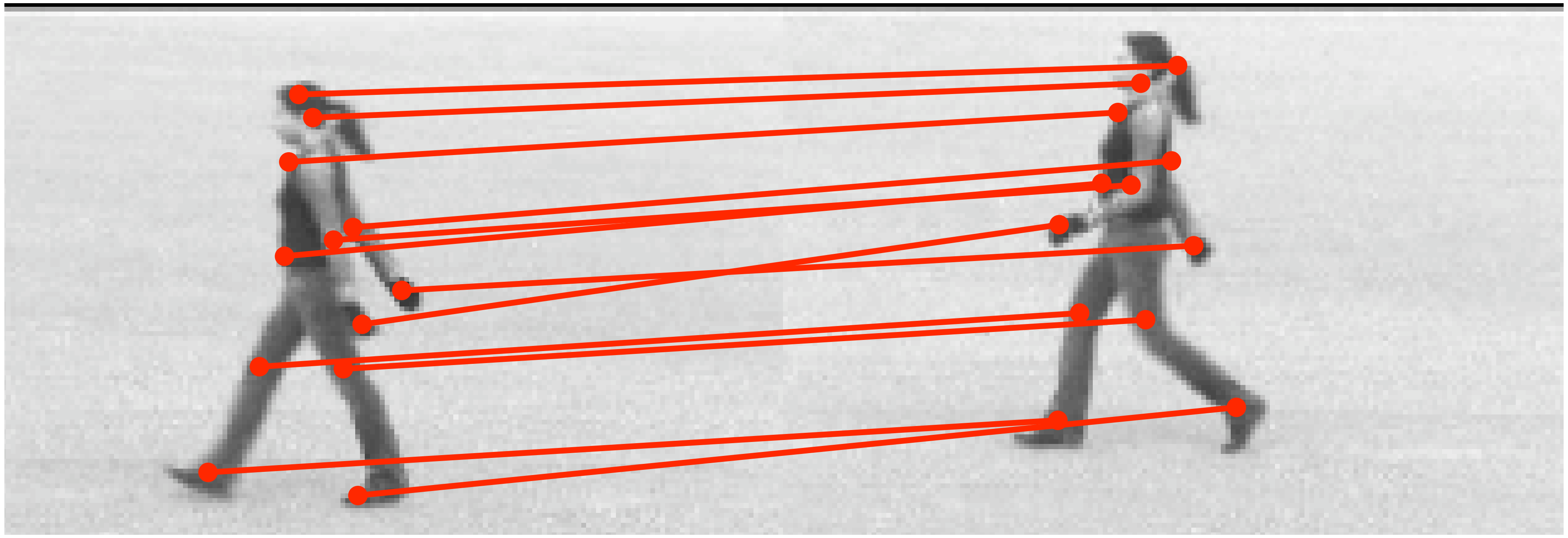
(this is a linear assignment problem again)

Experiments

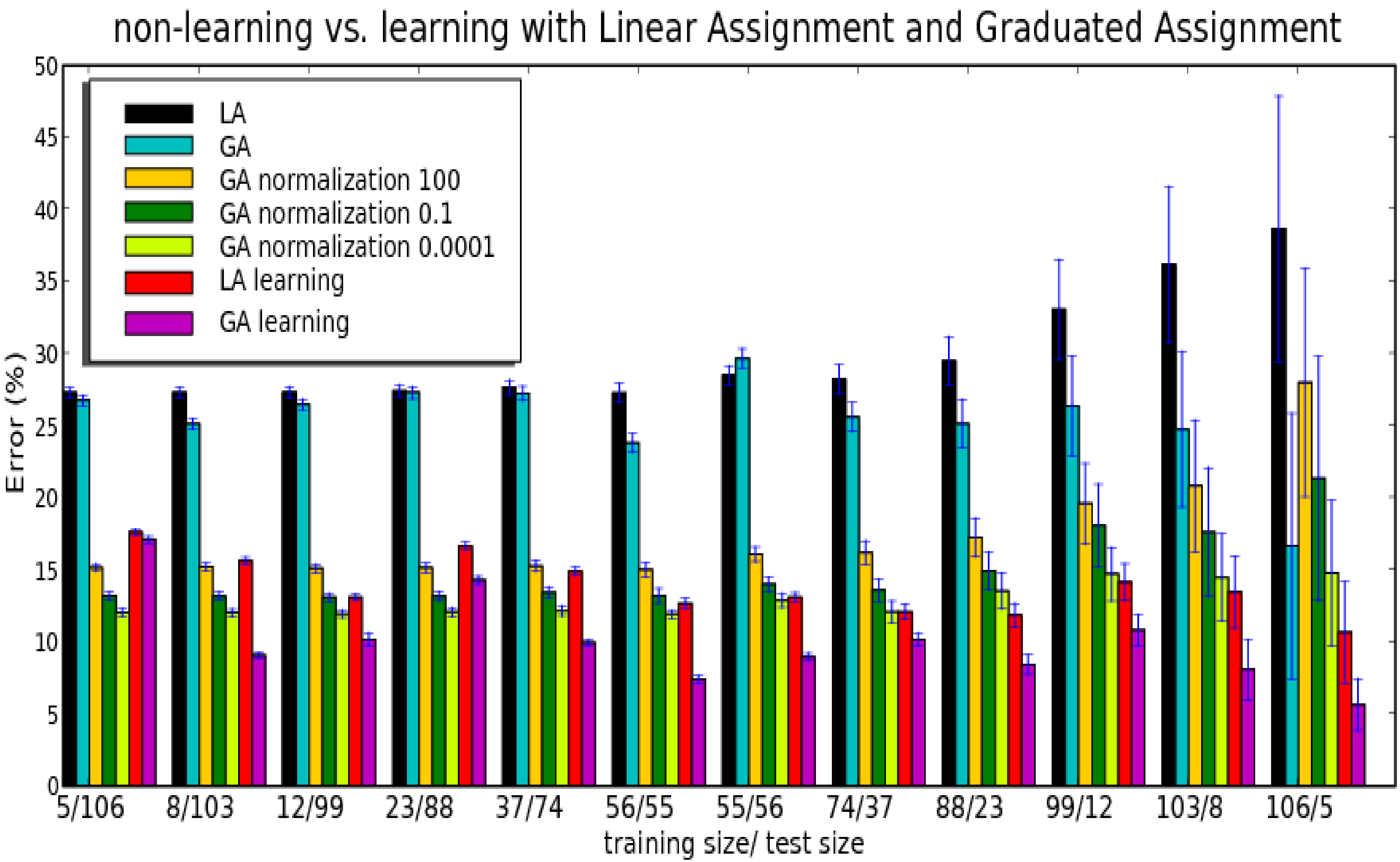
no learning



learning

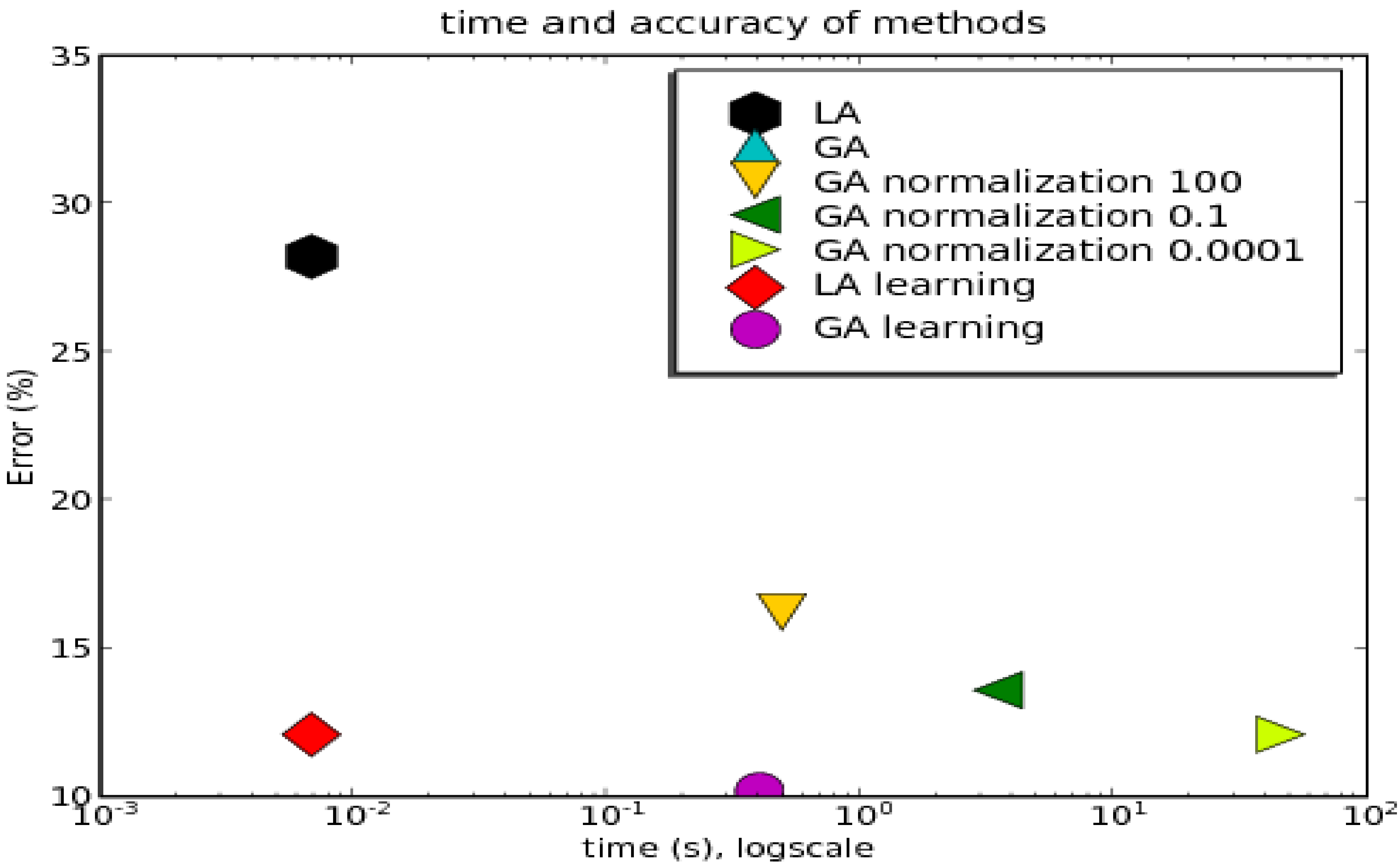


Accuracy



non-learning vs. learning with Linear Assignment and Graduated Assignment
LA
GA
GA normalization 100
GA normalization 0.1
GA normalization 0.0001
LA learning
GA learning
Error (%)
0
5
10
15
20
25
30
35
40
45
50
5/106
8/103
12/99
23/88
37/74
56/55
55/56
74/37
88/23
99/12
103/8
106/5
training size/ test size

Speed



time and accuracy of methods
35
30
25
20
15
10
Error (%)
10^{-3}
10^{-2}
10^{-1}
10^{0}
10^{1}
10^{2}
time (s), logscale
LA
GA
GA normalization 100
GA normalization 0.1
GA normalization 0.0001
LA learning
GA learning

Beyond

Setting

- Internet retailer (e.g. Netflix) sells movies *M* to users *U*.
- Users rate movies if they liked them.
- Retailer wants to suggest some more movies which might be interesting for users.

Goal

Suggest movies that user will *like*. Pointless to recommend movies that users do not like since they are unlikely to rent.

Problems with Netflix contest

- Error criterion is uniform over all movies.
- Can only recommend a small number of movies at a time (probably no more than 10).
- Need to do well only on top scoring movies.

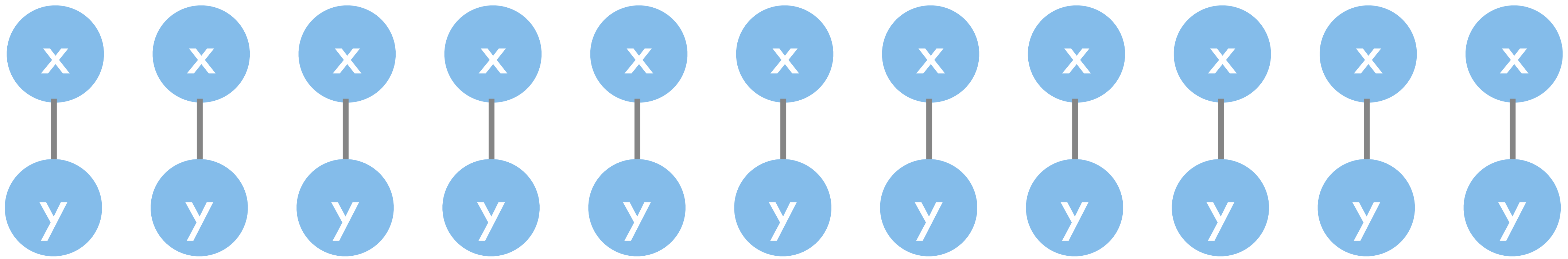
Insight

We can use linear assignment / sorting for ranking.

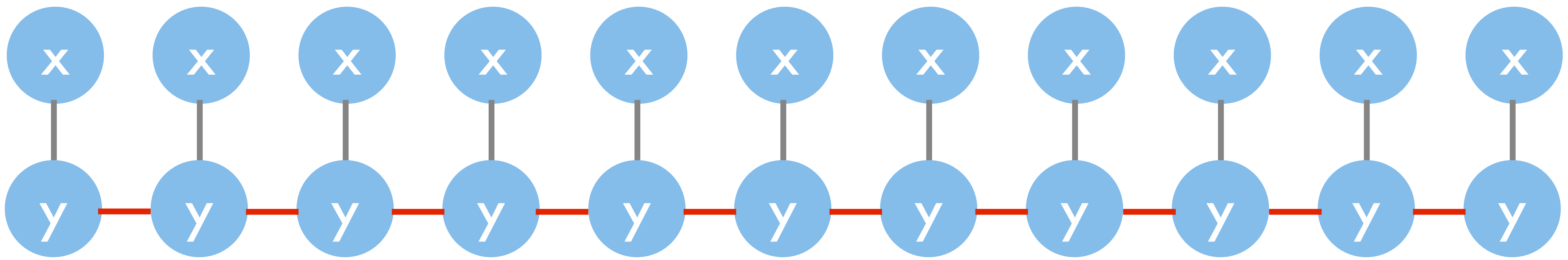
Sequence Annotation

Sequence Annotation

- Simple classification

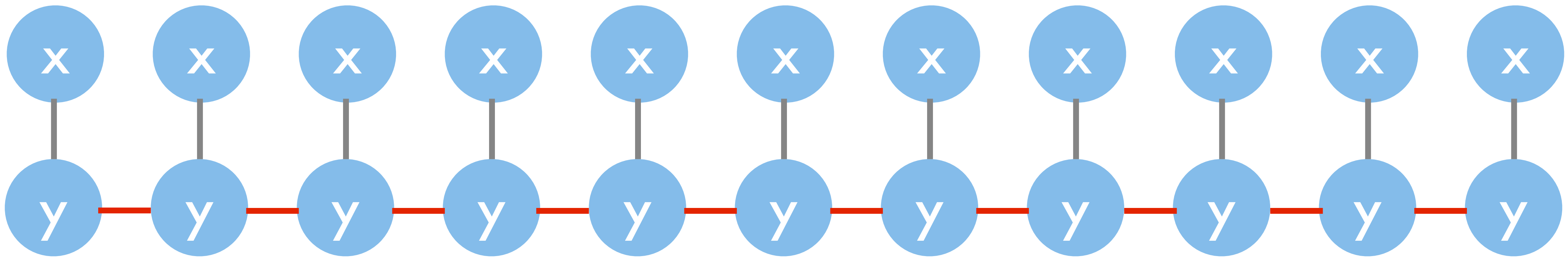


- What if adjacent labels are correlated?
- Can we exploit this for estimation?



Sequence Annotation

- Labeling problem



- Define f(x,y) on sequence

$$f(x,y) = \sum_{i=1}^{m} y_i f(x_i)$$ classification

$$f(x,y) = \sum_{i=1}^{m} y_i f(x_i) + f(y_i, y_{i+1})$$ sequence labeling

Dynamic Programming

- Clique Potential

$$f(x,y) = \sum_{i=1}^{m} \underbrace{y_i f(x_i) + f(y_i, y_{i+1})}_{:=g(y_i, y_{i+1})} = \sum_{i=1}^{m} g(y_i, y_{i+1})$$

- Forward pass (solve and backsubstitute)

$$\begin{aligned}
\max_y \sum_{i=1}^{m} g(y_i, y_{i+1}) &= \max_{y_2,\ldots,y_m} \Big[\underbrace{\max_{y_1} g(y_1, y_2)}_{:=h_2(y_2)} + \sum_{i=2}^{m} g(y_i, y_{i+1}) \Big] \\
&= \max_{y_3,\ldots,y_m} \Big[\underbrace{\max_{y_2} h_2(y_2) + g(y_2, y_3)}_{:=h_3(y_3)} + \sum_{i=3}^{m} g(y_i, y_{i+1}) \Big] \\
&= \ldots = \max_{y_m} h_m(y_m)
\end{aligned}$$

Dynamic Programming

- Backward pass
  (run same recursion from the end)
- Pairwise clique potential measures affinity between labels
- Loss function

$$\Delta(y, y') = \sum_{i=1}^{m} |y_i - y'_i|$$

- Computing loss gradient is dynamic program
- Solve by distributed subgradient procedure
  (we could also use kernels if we wanted to)

Loss function

- Structured large margin

$$
\begin{aligned}
l(x, y, f) &= \max_{y'} f(x, y') - f(x, y) + \Delta(y, y') \\
&= \max_{y'} \sum_{i=1}^{m} \left\{ y_i' f(x_i) + f(y_i', y_{i+1}') \right\} + \sum_{i=1}^{m} |y_i - y_i'| \\
&\quad - \sum_{i=1}^{m} \left\{ y_i' f(x_i) + f(y_i', y_{i+1}') \right\}
\end{aligned}
$$

- Need to solve argmax to compute gradient in f
- Iterate to solve convex program

Extensions

Structured Ramp Loss

- Binary ramp loss

$$l(x, y, f) = \operatorname{clip}\left\{[0,1], 1 - yf(x)\right\}$$

  - upper bound on error
  - solve by iterative Concave Convex Procedure
- Multiclass ramp loss

$$l(x, y, f) = \max_{y'}\left[f(x, y') + \Delta(y, y')\right] - \max_{y'} f(x, y')$$

  - upper bound bound on error
  - tighter bound than structured loss

Invariances

- Data
- Set of invariance transforms
(e.g. shift, slant, stroke, size, rotation for OCR)
- Not necessarily in group
- Not necessaritly absolute (with degradation)

$$l(x, y, f) = \sup_{y'} \left[f(x, y') - f(x, y) + \Delta(y, y') \right]$$

$$l(x, y, f) = \sup_{y', g} \left[f(g \circ x, g \circ y') - f(g \circ x, g \circ y) + \Delta(y, y', g) \right]$$

Pitching

- http://blogs.wsj.com/venturecapital/2010/01/11/how-to-pitch-a-venture-capitalist-on-a-napkin/
- http://en.wikipedia.org/wiki/George_H._Heilmeier#Heilmeier.27s_Catechism
- http://www.slideshare.net/dmc500hats/how-to-pitch-a-vc-aka-startup-viagra
- http://research.microsoft.com/en-us/um/people/simonpj/papers/proposal.html
- Practice, Practice, Practice

Further reading

- Girosi - Equivalence between sparse approximation and SVM
ftp://publications.ai.mit.edu/ai-publications/pdf/AIM-1606.pdf
- Smola, Schölkopf, Müller - Kernels and Regularization
http://alex.smola.org/teaching/berkeley2012/slides/Smola1998connection.pdf
- Aronszajn - RKHS paper (the one that started it all)
http://www.ams.org/journals/tran/1950-068-03/S0002-9947-1950-0051437-7/home.html
- Schölkopf, Herbrich, Smola - Generalized Representer Theorem
http://alex.smola.org/papers/2001/SchHerSmo01.pdf
- Hofmann, Scholkopf, Smola - Kernel Methods in Machine Learning
http://alex.smola.org/papers/2008/HofSchSmo08.pdf
- Teo, Globerson, Roweis and Smola - Convex learning with Invariances
http://books.nips.cc/papers/files/nips20/NIPS2007_1047.pdf
- Caetano, McAuley, Le, Smola - Learning Graph Matching
http://alex.smola.org/papers/2009/Caetanoetal09.pdf
- Keshet and McAllester - Tighter bounds for ramp loss
http://ttic.uchicago.edu/~jkeshet/papers/McAllesterKe11.pdf
- Chapelle, Do, Le, Smola, Teo - Ramp loss examples
http://alex.smola.org/papers/2009/Chapelleetal09.pdf
- Platt - Sequential Minimal Optimization
http://research.microsoft.com/en-us/um/people/jplatt/smoTR.pdf
- Joachims - Multivariate performance measures
http://www.cs.cornell.edu/people/tj/svm_light/svm_perf.html