Randomized Linear algebra for Machine Learning

Anirban Dasgupta

CSE, IIT Gandhinagar

Example Datasets and Problems @ web company!

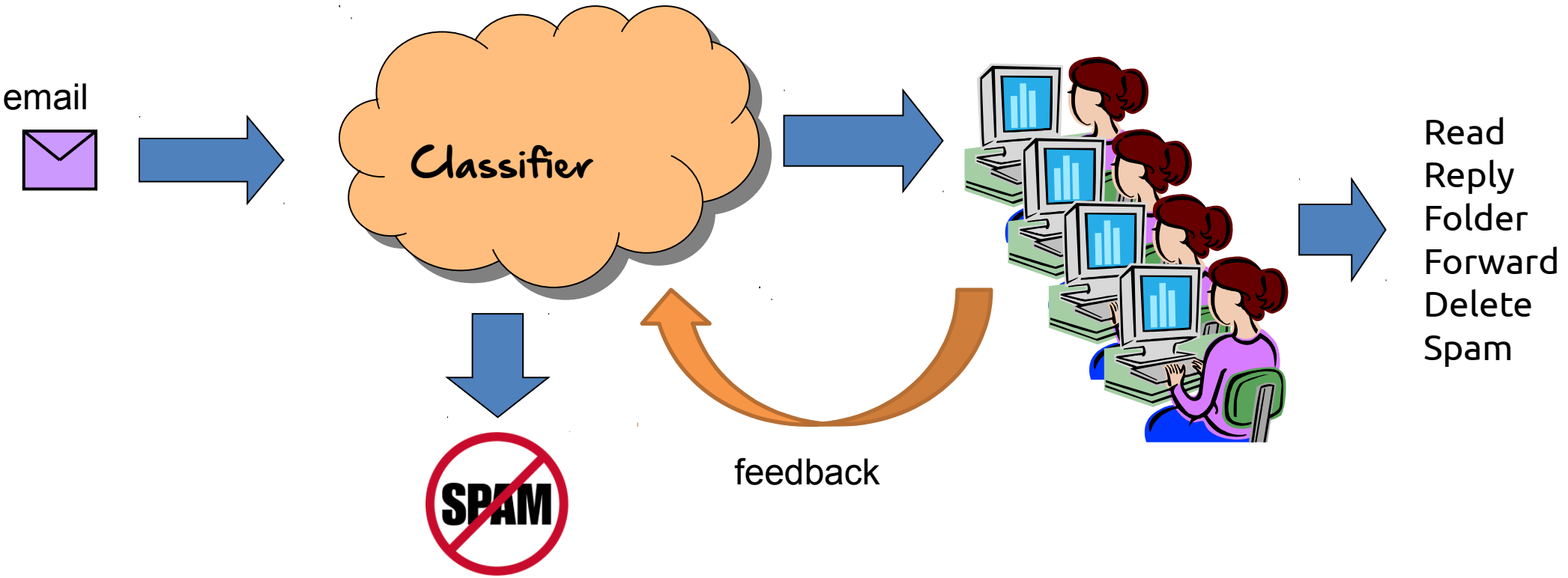
- Some Datasets
  - User Generated Content ~ 10-100 GB/day
  - Web ~ 10-100 PB
  - User behavioral data ~ 1 TB/day
  - Social networks ~ 10-100 GB
- Some Problems
  - Recommendation/matching - advt. to user, content to user, friends to users
  - Information extraction
  - Abuse filtering, anomaly detection
  - Modeling user behavior
- Some useful algorithmic techniques
  - scalable clustering, matrix factorization, regression, extracting different statistics
  - Sketching, sampling, dimension reduction

This talk

- Three problems:
  - Large scale personalization of email classifiers
  - Large regression problems
  - Finding nearest neighbors quickly

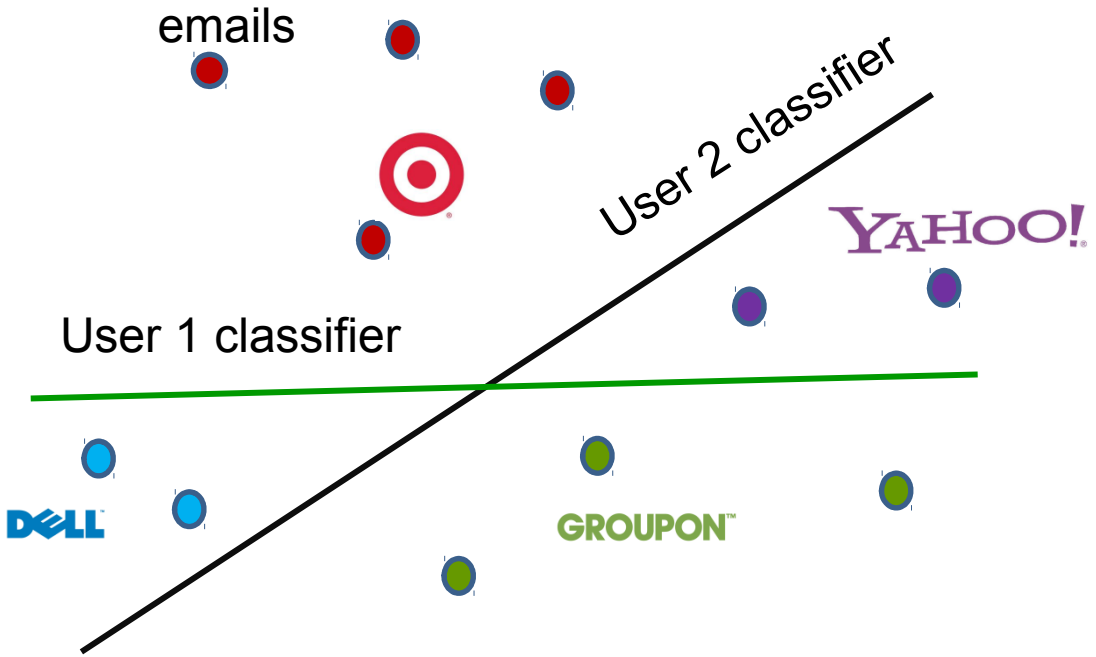
- Unified technique to handle all three

Mail Classification



Each email classified into spam/inbox
Classifier updated by user feedback
Users don’t always agree about what is spam

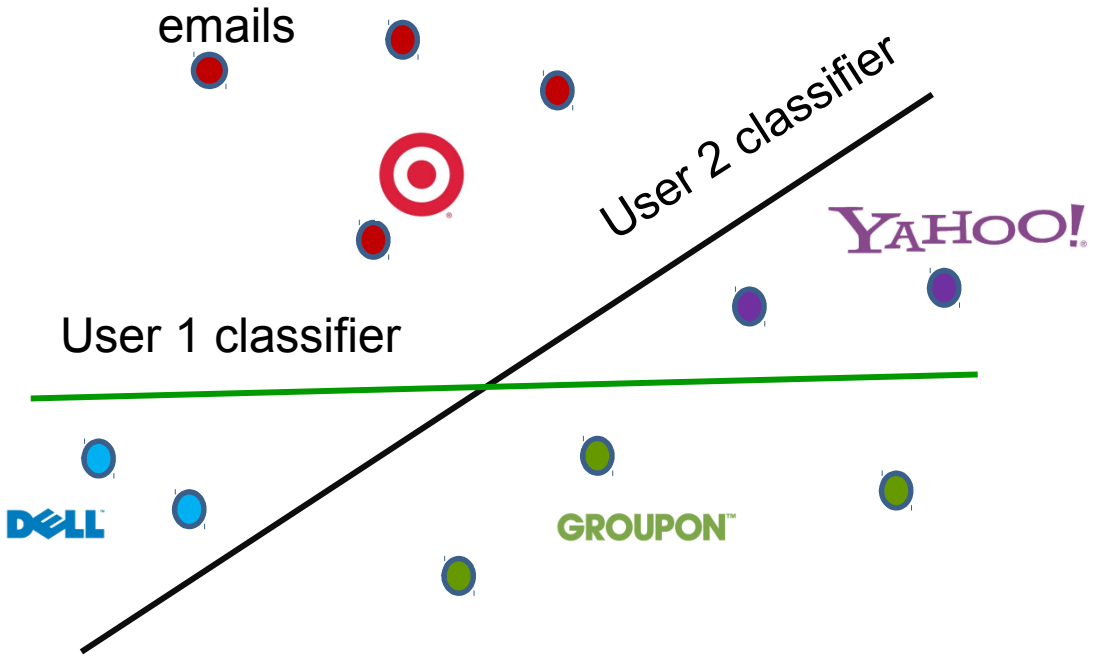
Efficient Personalization



Users have different spam preferences

Need to assign different classifiers

Efficient Personalization



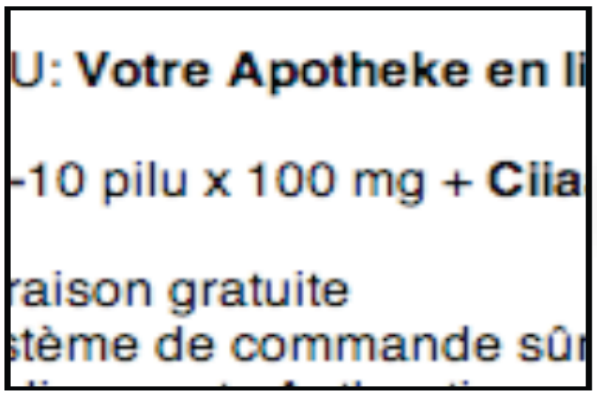
Users have different spam preferences

Need to assign different classifiers

Also should ensure new users get good classifier

Should be updated frequently using new feedback

Efficient Personalization: namespaces aka multitask learning



x

NEU
Votre
Apotheke
...

NEU
USER123_NEU
Votre
USER123_Votre
Apotheke
USER123_Apotheke
...

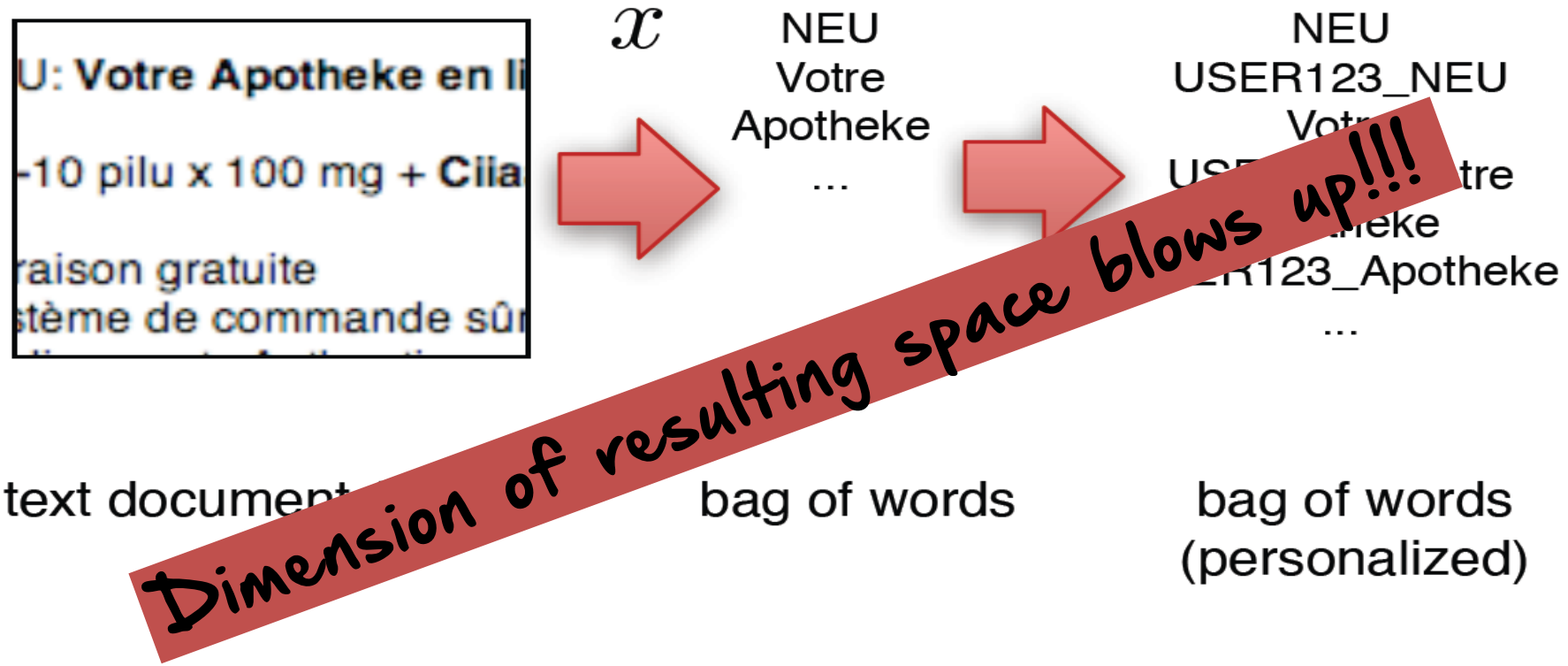
text document (email)

bag of words

bag of words (personalized)

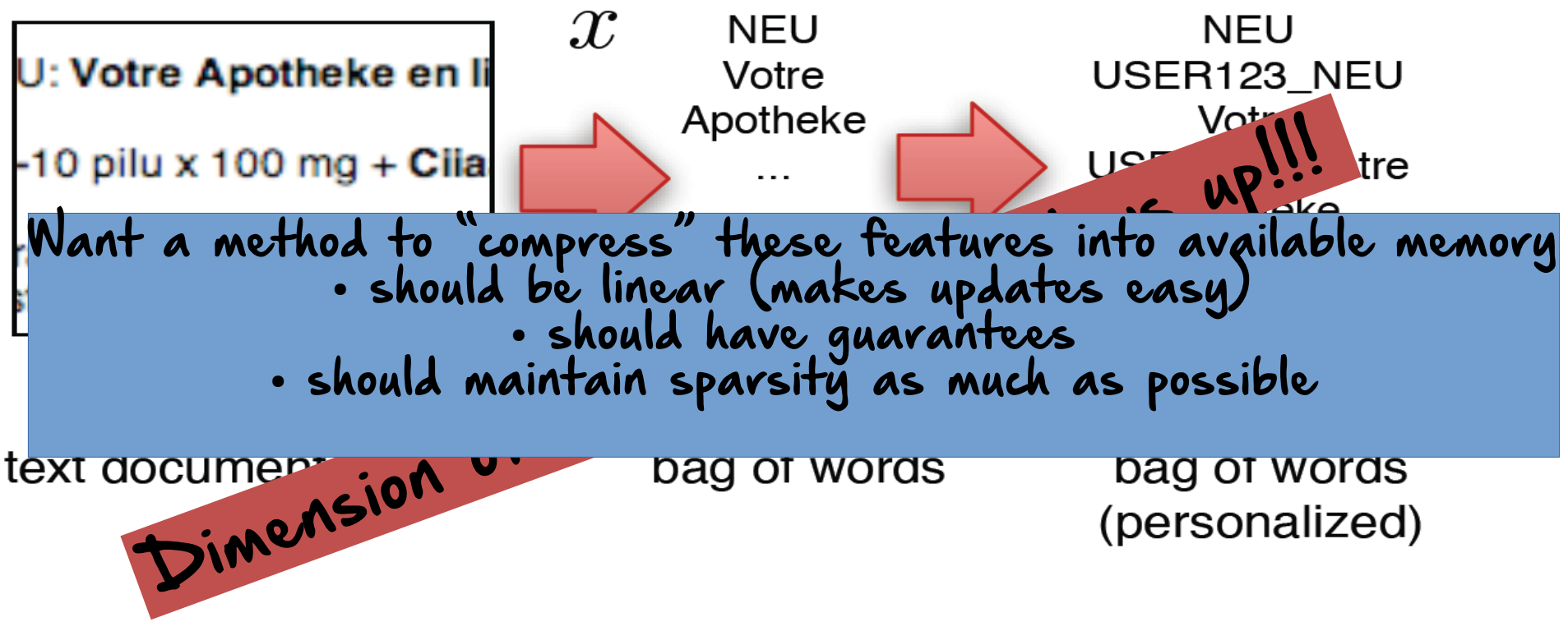
Classifier learnt is $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

Efficient Personalization: namespaces



Classifier learnt is $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

Efficient Personalization: namespaces



Classifier learnt is $w_{\text{global}}^T x_{\text{global}} + w_{\text{user}}^T x_{\text{user}}$

This talk

- Three problems:
  - Large feature set resulting from personalization
  - Regression with large number of examples
  - Finding nearest neighbors quickly

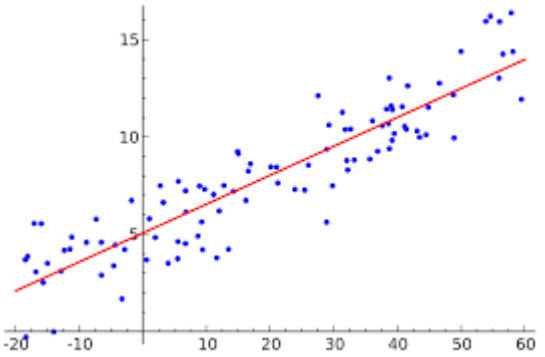
- Unified technique to handle all three

Linear Regression

$$\underset{n \times d}{\left(A \right)} \left(\hat{x} \right) \approx \left(b \right), \quad n \gg d$$

$$\begin{aligned} \mathcal{Z}_2 &= \min_{x \in \mathbb{R}^d} ||b - Ax||_2 \\ &= ||b - A\hat{x}||_2 \end{aligned}$$

- We are interested in the over-constrained case
  - Typically there is no x such that Ax = b
  - Find best x such that Ax ~ b
- Ubiquitous in statistics and ML
  - Statistical: best linear unbiased estimator
  - Geometrical: projection onto span(A)



[slides mostly borrowed from Michael Mahoney]

Solving Least Square

- Normal equations
  - $x = (A^T A)^{-1} A^T b$
- Cholesky decomposition
  - Create upper triangular R such that $A^T A = R^T R$ and then solve $R^T R\, x = A^T b$
  - Typically used when A is stable
- QR decomposition
  - Create orthogonal Q and upper triangular R such that $A = QR$
  - numerically more stable
- SVD decomposition
  - $A = USV^T$ and then solve $x = A^+ b = V S^{-1} U^T b$
- Theoretically, all of these take time $O(nd^2)$, the numerical stability controls the runtime in practice

Solving for large data

- Linear regression
  - Can we do better than $O(nd^2)$ in theory, as well as practice?
- Linear classifier with large feature set
  - Can we “compress” or “select” set of features to work on?
- Options:
  - Feature selection: not clear empirically or theoretically
  - Dimension reduction using PCA etc: too much time
  - Just uniform sampling examples/features?
- We want something with guaranteed approximations

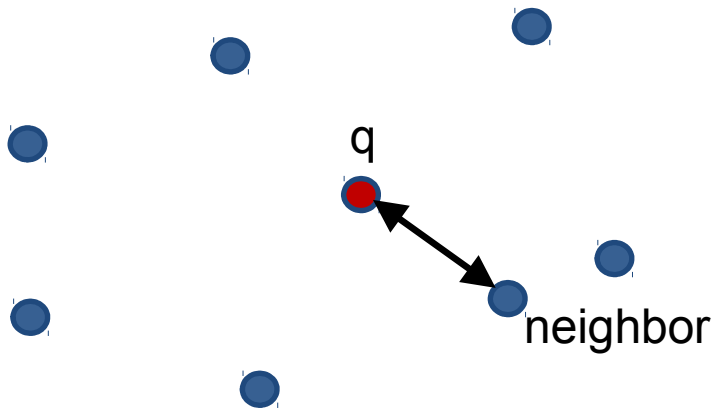
This talk

- Three problems:
  - Large feature set resulting from personalization
  - Regression with large number of examples
  - Finding nearest neighbors quickly

- Unified technique to handle all three

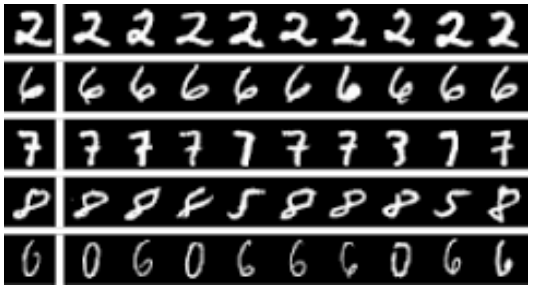
Finding Near Neighbors

- **Given**: a set of points S, a query point q
- **Find**: point in S ‘close’ enough to q
  - K-nearest neighbor
  - d(q, result) < r



Finding Near Neighbors: Applications

- Numerous
  - Finding similar images in search
  - Clustering news articles
  - Finding duplicate local listings
  - Nearest neighbor classifier
  - ….
- Other variations
  - Different metrics
  - Find all pairs near-duplicates

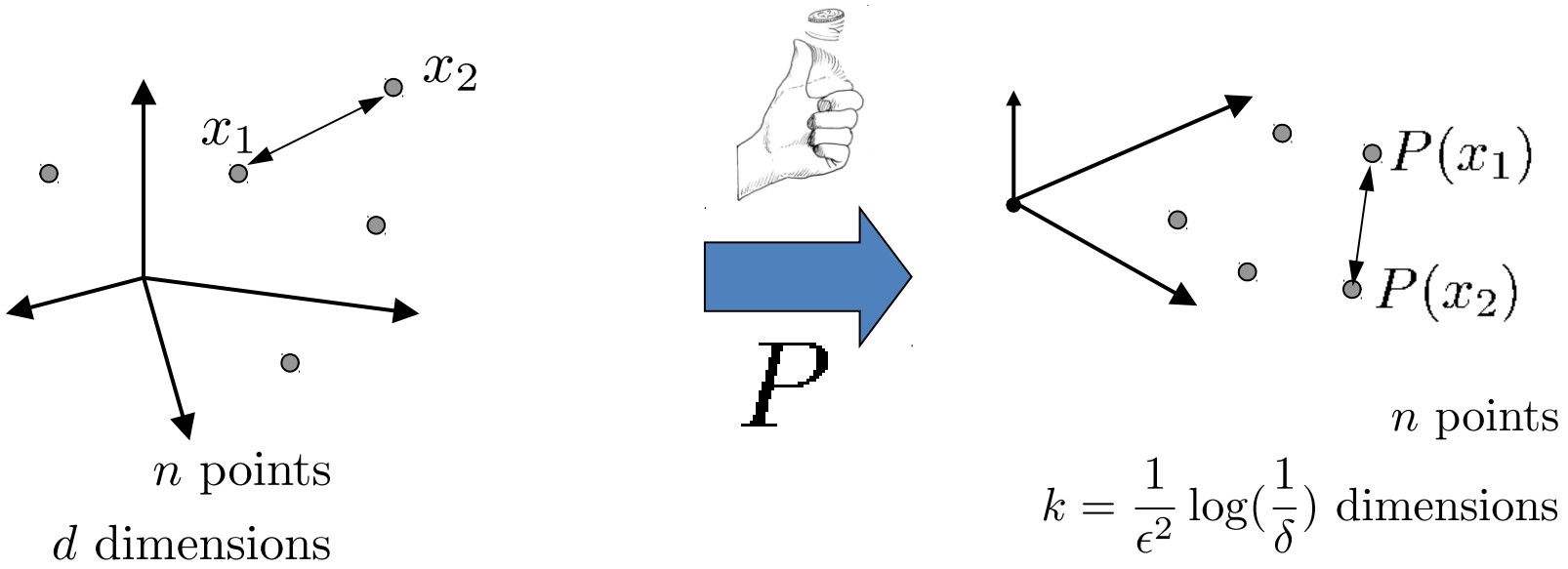


This talk

- Three problems:
  - Large feature set resulting from personalization
  - Regression with large number of examples
  - Finding nearest neighbors quickly

- Unified technique to handle all three
  - Random projections !

Random Projection
Johnson Lindenstrauss'84



$$(1-\epsilon)\|x_i - x_j\| \leq \|P(x_i - x_j)\| \leq (1+\epsilon)\|x_i - x_j\|$$

- Linear transformation – random matrix, oblivious of input
- Target dimension independent of source dimension
  - Quadratic dependence on error
  - Dependence on $\log(1/\delta)$
  - Optimal dependence on ϵ [Alon'03, Jayram-Woodruff'11]
- No such transform for L1-metric [Brinkman-Charikar'03]

Brief History of JL Lemma

- Recall that projection P is a k x d matrix

Publication	Transformation
Johnson, Lindenstrauss ’84	Random hyperplane
Frankl, Maehara ’88	Random hyperplane
Indyk, Motwani ’99	i.i.d. Gaussian
Dasgupta, Gupta ’99	i.i.d. Gaussian
Achlioptas ’03	i.i.d. ± 1
Indyk, Naor ’08	i.i.d. sub-Gaussian
Matousek ’08	i.i.d. sub-Gaussian

- All the above constructions take $O(dk)$ space to store
- Time to apply for dense vector $O(dk)$, time for sparse vector $O(k\|x\|_0)$
- Lower bound on sparsity $\Omega(k)$ for i.i.d. construction

Applications of JL

- Preserves pairwise distances, hence angles
- Helps us remove curse of dimensionality
- Many algorithmic applications
  - Nearest neighbor search [Indyk, Motwani ’98]
  - Data streams [Alon et al ’99]
  - Graph sparsification [Spielman, Srivastava ’08]
  - Machine learning [Arriaga, Vempala ’99]
  - …

- Intimate connection with Compressed Sensing, Baraniuk et al.’99, Krahmer-Ward’11.
- Randomized embeddings are a versatile and powerful tool in general
  - Linial et al. ’99, Bartal ’98

Basic idea of JL proof

- Suppose P is all iid gaussian k x d i.e. $P_{ij} \sim N(0,1)$
- Each row of Px satisfies

$$(Px)_i = \sum_j P_{ij} x_j \sim N(0, \|x\|_2^2)$$

$$E[(Px)_i^2] = \|x\|_2^2$$

Basic idea of JL proof

- Suppose P is all iid gaussian k x d i.e. $P_{ij} \sim N(0,1)$
- Each row of Px satisfies

$$(Px)_i = \sum_j P_{ij} x_j \sim N(0, \|x\|_2^2)$$

$$E[(Px)_i^2] = \|x\|_2^2$$

- Now show that $Z = \frac{1}{k} \sum_i (Px)_i^2$ is concentrated
  - Use the MGF of Z (chi-squared distbn)
- For non-Gaussian P, similar idea for proof
  - Advanced constructions need matrix concentration inequalities

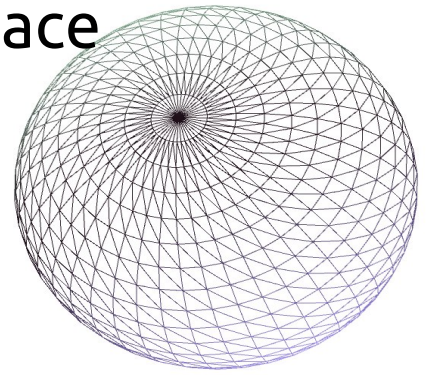
JL for Least Square Regression

- First “naive” approach
  - S is $k \times n$ JL matrix, from iid Gaussian
  - Solve $\text{argmin}_x || SA\,x - Sb ||_2$ and return x
  - This x will be a good approximation to original problem

JL for Least Square Regression

- First “naive” approach
  - S is $k \times n$ JL matrix, from iid Gaussian
  - Solve $\text{argmin}_x || SA\,x - Sb ||_2$ and return x
  - This x will be a good approximation to original problem

- Issues:
  - k needs to be chosen as $d/\varepsilon^{O(1)}$ for $1 \pm \varepsilon$ error
  - Projection runtime = $O(nd^2)$, so not very efficient over exact
  - Resulting SA dense even if A sparse → more space



Making JL fast: FJLT
(Ailon, Chazelle '06)

- The key observation is that the issue with sparse random matrices have is preserving norms of *sparse vectors only*
  - Make the projection matrix sparse
  - Preprocess the input vectors so that they are dense
    - recall time vs frequency duality in signal processing

Making JL fast: FJLT
(Ailon, Chazelle '06)

- The key observation is that the issue with sparse random matrices have is preserving norms of *sparse vectors only*
  - Make the projection matrix sparse
  - Preprocess the input vectors so that they are dense
    - recall time vs frequency duality in signal processing

$$H_d = \begin{pmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{pmatrix} \text{ and } H_1 = (1) \qquad P_{ij} = \begin{cases} 0, & \text{w.p. } 1-q, \\ N(0, 1/q), & \text{w.p. } q; \end{cases}$$

$$q = \Theta(\log^2(1/\delta)/d)$$

Making JL fast: FJLT
(Ailon, Chazelle '06)

- The key observation is that the issue with sparse random matrices have is preserving norms of *sparse vectors only*
  - Make the projection matrix sparse
  - Preprocess the input vectors so that they are dense
    - recall time vs frequency duality in signal processing

$$H_d = \begin{pmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{pmatrix} \text{ and } H_1 = (1) \qquad P_{ij} = \begin{cases} 0, & \text{w.p. } 1-q, \\ N(0, 1/q), & \text{w.p. } q; \end{cases}$$

$$q = \Theta(\log^2(1/\delta)/d)$$

$$x \mapsto \Phi x = PH_dDx \qquad D = \pm 1 \quad \text{in each diagonal entry}$$

Runtime = $O(d \ln d + \frac{1}{\epsilon^2} \log^2(1/\delta))$

To counter adversarial inputs

Making JL fast: FJLT
(Ailon, Chazelle '06)

- The key observation is that the issue with sparse random matrices have is preserving norms of *sparse vectors only*
  - Make the projection matrix sparse
  - Preprocess the input vectors so that they are dense
    - recall time vs frequency duality in signal processing

$$H_d = \begin{pmatrix} H_{d/2} & H_{d/2} \\ H_{d/2} & -H_{d/2} \end{pmatrix} \text{ and } H_1 = (1) \qquad P_{ij} = \begin{cases} 0, & \text{w.p. } 1-q, \\ N(0, 1/q), & \text{w.p. } q; \end{cases}$$

$$q = \Theta(\log^2(1/\delta)/d)$$

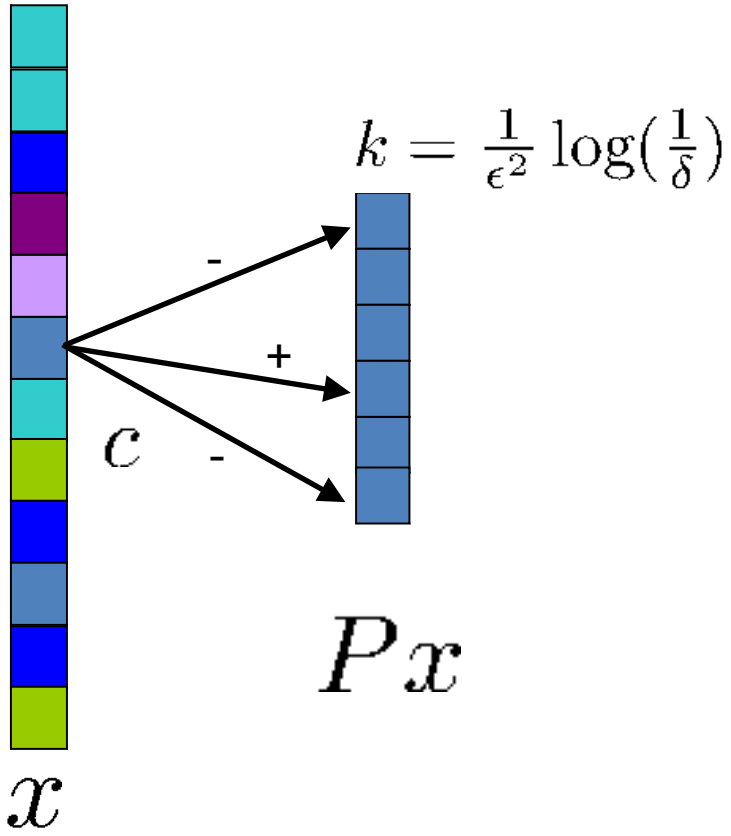
$$x \mapsto \Phi x = PH_dDx \qquad D = \pm 1 \quad \text{in each diagonal entry}$$

Runtime = $O(d \ln d + \frac{1}{\epsilon^2} \log^2(1/\delta))$

To counter adversarial inputs

However, the output is always a dense vector !

Making JL Sparse using hash functions
(D., Kumar, Sarlos '10)



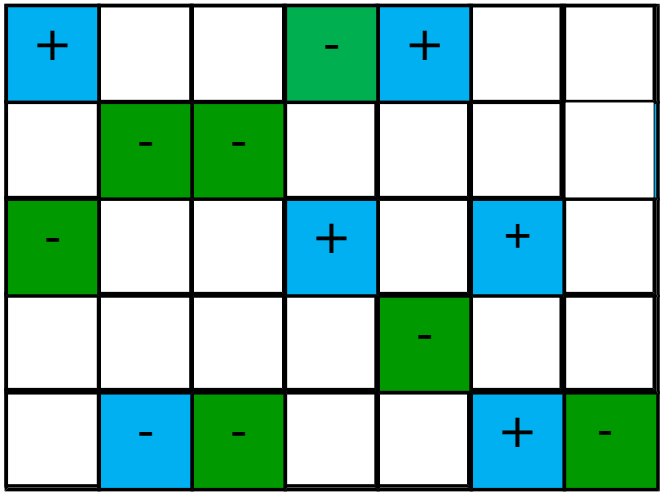
- c positions chosen uniformly at random with replacement for each column
- r_j independent ± 1
- $c \approx \frac{1}{\epsilon} = O(k^{1/2})$

- For any x
  with probability $1 - \delta$

$$(1-\epsilon) \leq \|Px\|_2^2 \leq (1+\epsilon)$$

- i.i.d. construction corresponds to G(n,p) with $p = \Omega(1)$

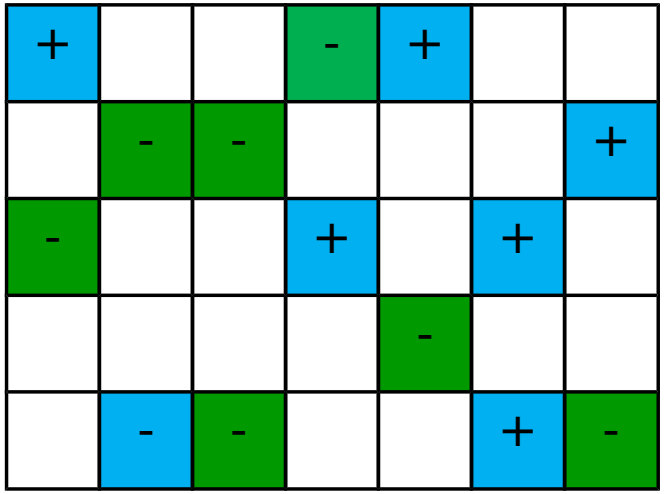
SparseJL – discussion



Last column fewer nnz due to collisions

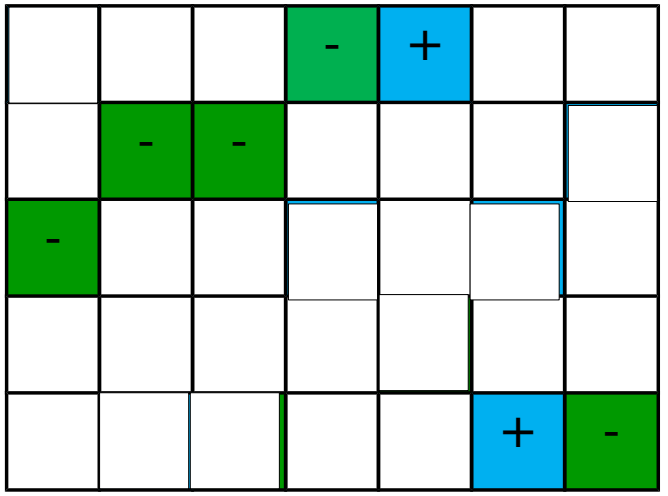
- Matrix is created column-wise, rows are dependent
- Has $\tilde{O}(\frac{1}{\epsilon})$ non-zeros per column, first $o(\frac{1}{\epsilon^2})$
- r_j being ± 1 is crucial. Gaussians do <u>not</u> work
- Sparser/faster than naïve JL only for small δ large
- Naïve construction runtime $O(\frac{\|x\|_0}{\epsilon}\log^3(\frac{1}{\epsilon\delta}))$

Sparser JL
(Kane, Nelson '10)



- Make sure the collisions do not happen
  - Select exactly $1/\varepsilon$ positions per column and then assign ± 1 uniformly at random
- Analysis requires sophisticated matrix concentration bounds, and path counting arguments
- Needs $1/\varepsilon$ nnz per columns, strictly faster than naïve JL
- Cannot really go sparser --- lower bound exists

Sparse Subspace Embeddings
(Clarkson Woodruff '12)



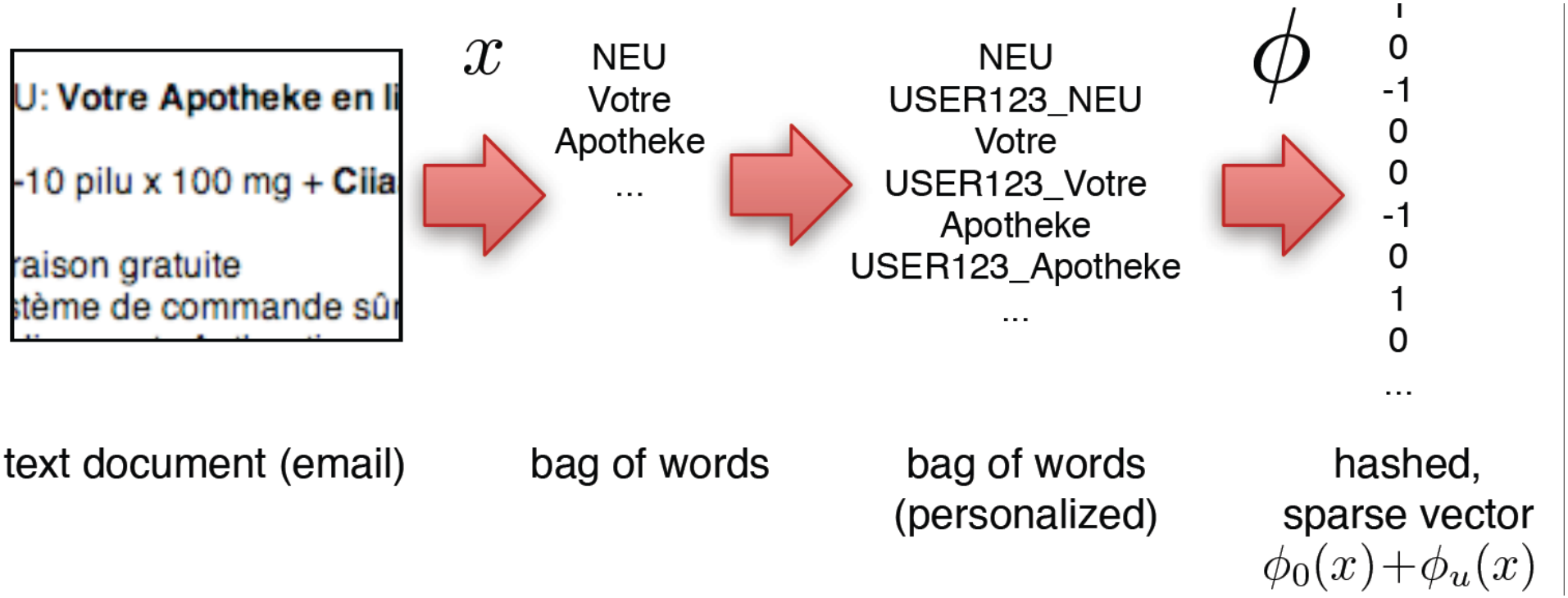
- Choose exactly one nnz position per column !!
  - Put ±1 uniformly at random
- Needs higher target dimension/hash buckets: $O(d^2)$
- Only works with when all points are from a subspace !!
  - But that's what we need for linear regression

This talk

- Three problems:
  - Large feature set resulting from personalization
  - Regression with large number of examples
  - Finding nearest neighbors quickly

- Unified technique to handle all three
- Application to each problem

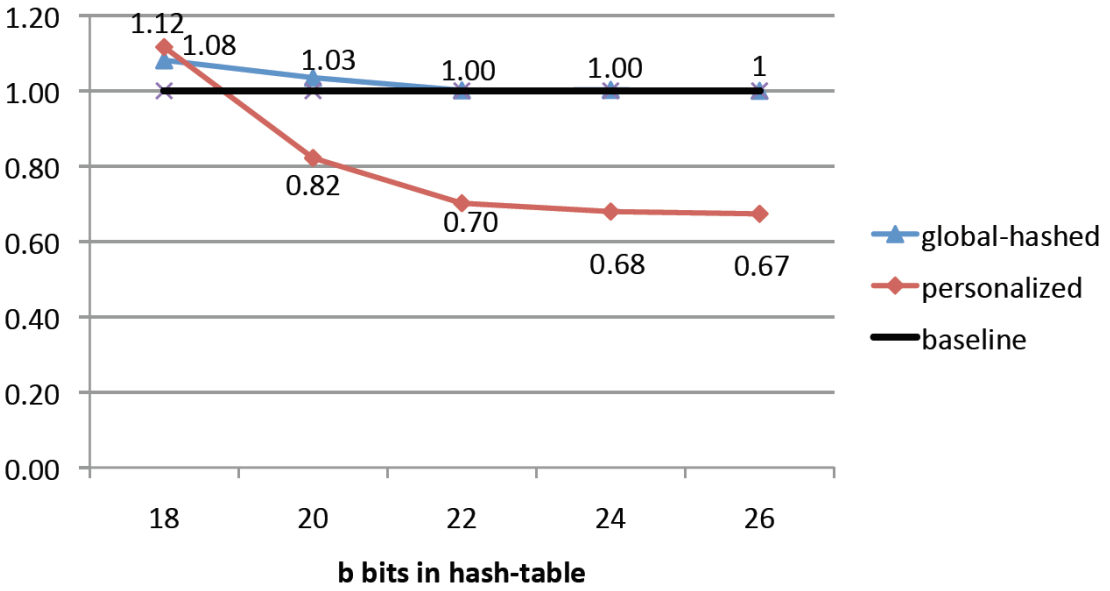
Efficient Personalization: namespaces aka multi-task learning



Use the sparse random projection Φ to map the large dimensional space into available memory

Weights are learnt in this projected space

Application in Feature hashing: Cheap personalization improves performance



- Use sparsity = 1
- 3.2 million emails, 433K users
- Used linear classifier in hashed space
- 2^{26} floats = 256 MB

[ICML'09, Weinberger, D., Smola, Attenberg, Langford]

Sparse JL Applications

Originally used in Vowpal Wabbit linear classifier by J. Langford

- http://hunch.net/~vw : "hashing trick"
- Shi, Petterson, Dror, Langford, Smola, Vishwanathan JMLR'09 give variance bounds
- In learning matrix factorizations for collaborative filtering: Karatzoglou, Smola, Weimer AISTATS'10
- Hashing implemented in feature construction in Apache Mahout

Used by Yahoo! mail Anti-spam classifiers

Empirical study in shows it to be 'best in class'

- Venkatsubramanian, Wang ALENEX'11

Hashing construct appears in

- Charikar et al. TCS'04, Thorup and Zhang, SODA'04, Cormode and Garofalakis, VLDB'05….

Caveat: median

Sparse JL for least square regression

- Use same approach as before, with S as sparse JL
  - S is k x n sparse JL matrix
  - Solve $\text{argmin}_x || SA\,x - Sb ||_2$ and return x
  - This x will be a good approximation to original problem

- Properties:
  - Can solve regression in $O(nnz(A) + (d/\varepsilon)^{O(1)})$
  - Works okay in practice too !
    - Typically numerical stability is worse than Gaussian projection

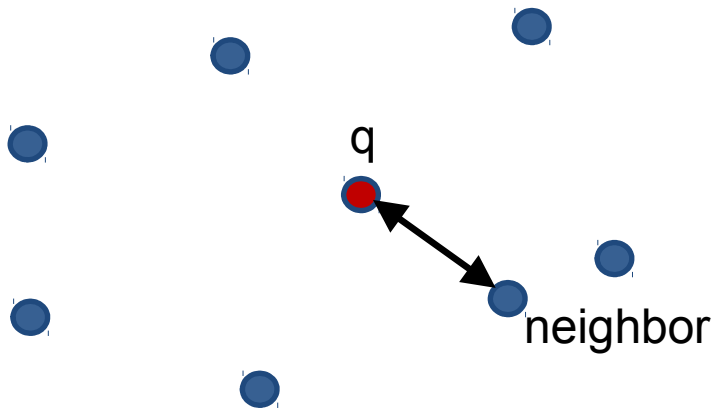
Making JL practical

- Avron, Maymounkov, and Toledo 2010:
  - Uses JL as a preconditioner to iterative methods
  - Blendenpik "beats Lapack's direct dense least-squares solver by a large margin on essentially any dense tall matrix”

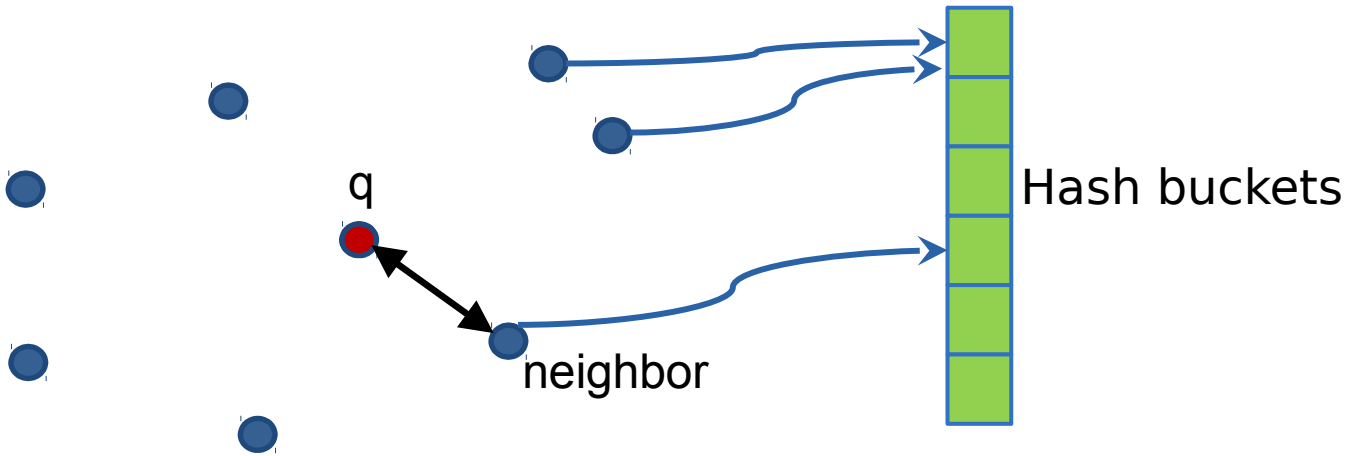
- Empirical results "show the potential of random sampling algorithms and suggest that random projection algorithms should be incorporated into future versions of Lapack."

Finding Near Neighbors

- **Given**: a set of points S, a query point q
- **Find**: point in S ‘close’ enough to q
  - K-nearest neighbor
  - d(q, result) < r



Locality Sensitive Hashing
(Indyk Motwani '98…)

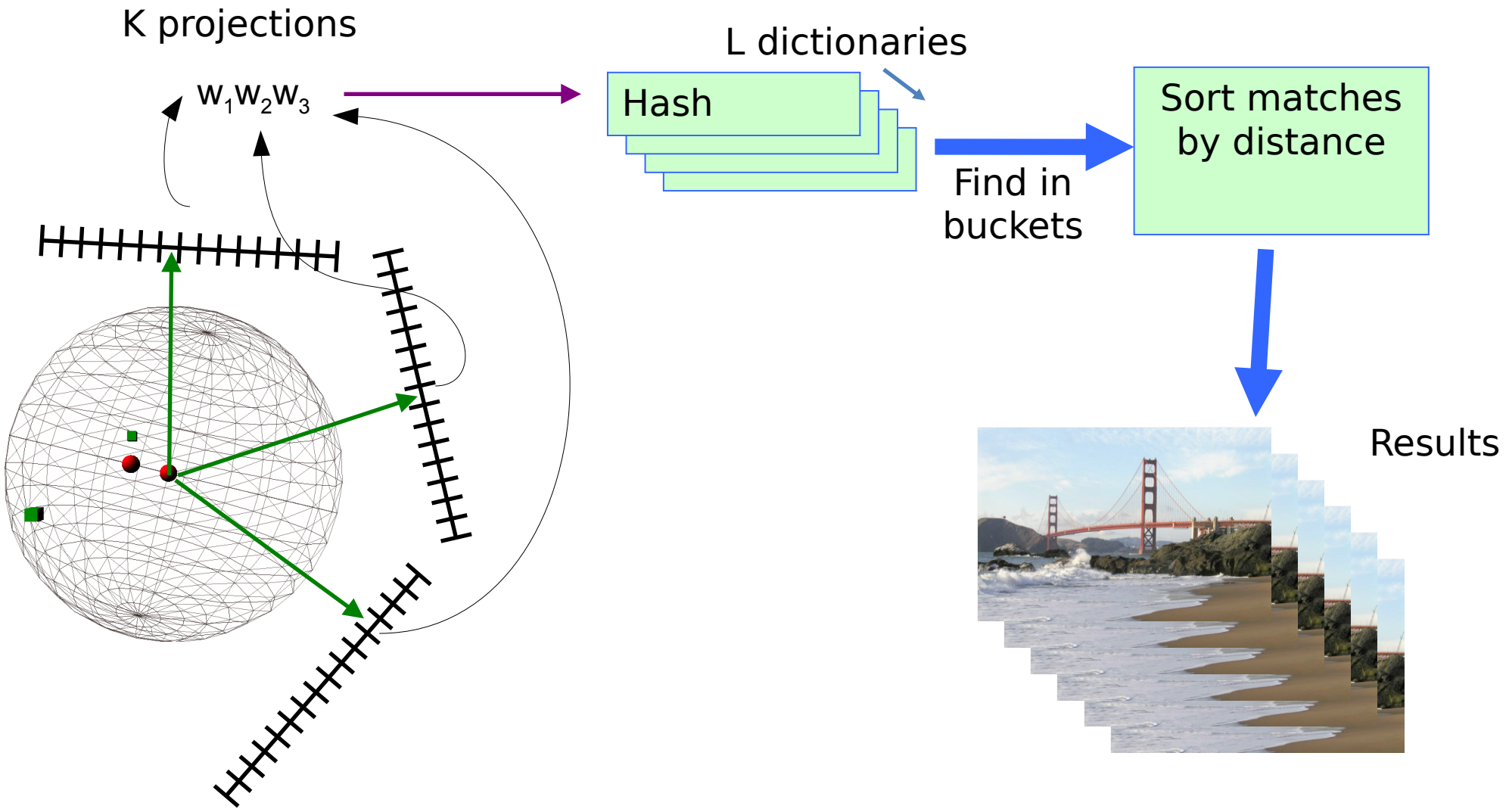


$$\Pr[h(u) = h(v)] \propto \mathrm{sim}(u, v)$$

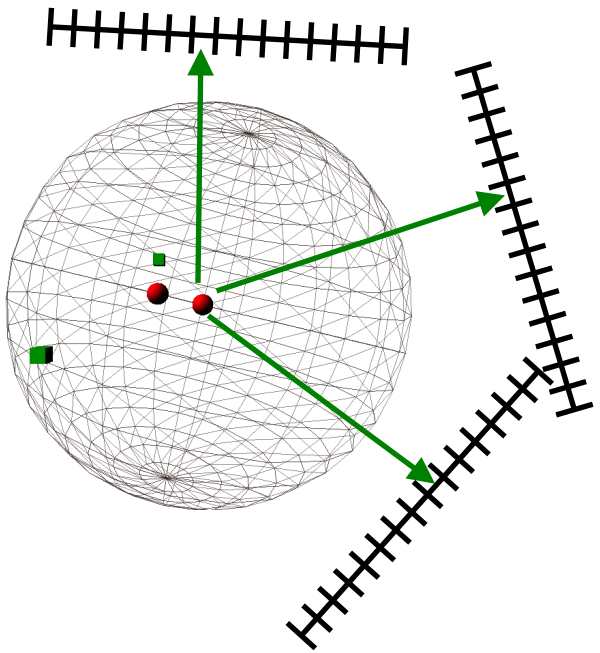
- Variants
  - E2LSH, SimHash, Minhash, Hamming
- Improvements:
  - Cutting down space: Entropy based LSH, MultiProbe LSH
  - Finding LSH parameters based on data distribution

SimHash: LSH for L2

(Charikar '02)



SimHash using random projections



$$A = \begin{pmatrix} & d & \\ & A_{ij} & \end{pmatrix} k$$

$$A_{ij} \sim N(0,1)$$

$$w_i = \lfloor \frac{A_i \cdot x + b_i}{c} \rfloor$$

Projection time $O(kdL)$

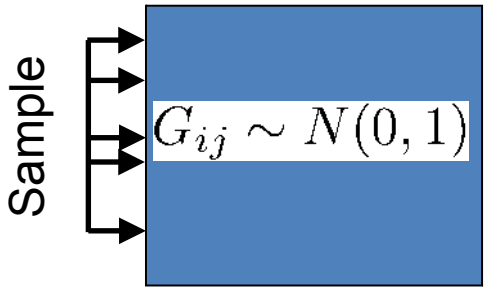
Space $O(nL)$

Can we improve the time taken to calculate the hash functions?

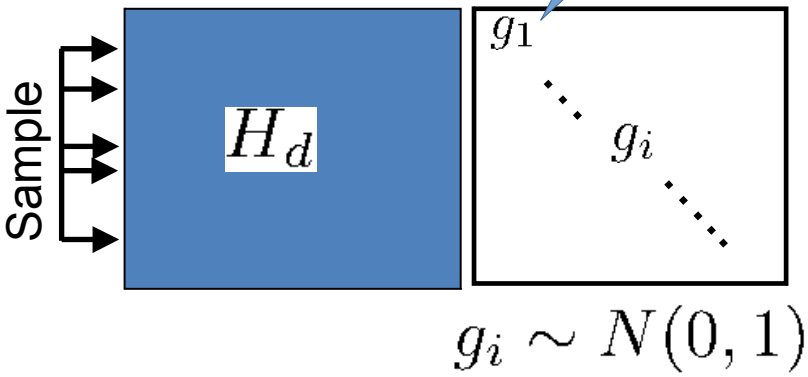
- would result in improved lookup time, which is critical
- We show such a construction using an idea similar to FJLT

FastLSH: Intuition

(D., Kumar, Sarlos '11)



VS.



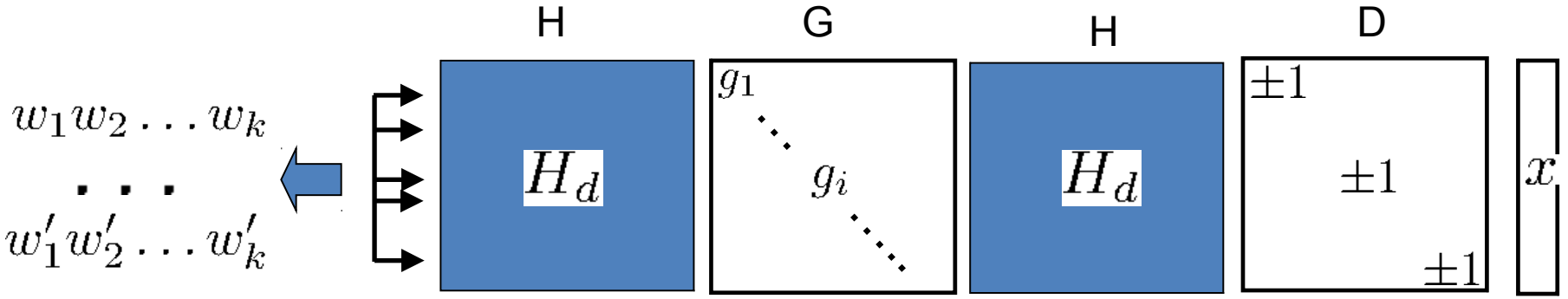
Gaussian diagonal unlike FULT

samples independent
randomness $O(ds)$
time to multiply $O(ds)$
"works" for all vectors

samples not independent
randomness $O(d)$
time to multiply $O(d \log d)$
'works' for 'smooth' vectors
i.e. mimics a random rotation

FastLSH

- Save time by making the k random projections not independent



Sample k*L entries w.r.

$$g_i \sim N(0,1)$$

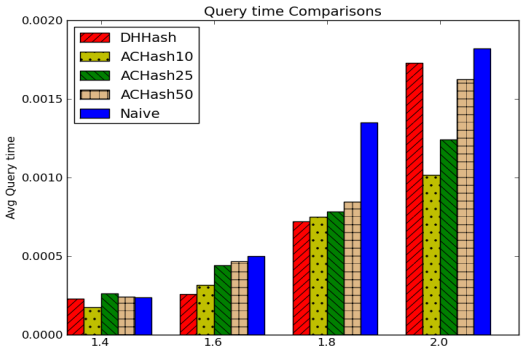
$$w_i = \left\lfloor \frac{g^T D_{H_i} \cdot (HDx) + b_i}{c} \right\rfloor$$

Projection Time $O(d \ln d + kL)$

(saving over O(dkL))

FastLSH summary

- Using ‘structured randomness’ gives improvements in LSH query time, with some space tradeoffs
  - Space usage can likely be tackled by using orthogonal techniques e.g. entropy-based LSH
- Easy to implement
  - Hadamard/FFT tools are highly optimized
- Comes with novel theoretical guarantees
  - Technique maybe applicable to related problems
- Possible applications in all-pairs similarity search, deduplication, similarity joins,…



Overall Summary

- Random projection is one of the versatile tools in randomized linear algebra
- Several motivating applications have yielded interesting new variants
  - FJLT, SparseJL, SubspaceJL...
- Chief boon/complaint is data obliviousness
  - Can we find a middle ground?
  - Upcoming work: learning projection matrix is “better” for multiclass multi-problem setting
- Handling non-L2 errors
  - structure beyond pairwise distances?
- This is a biased survey: host of other sampling/sketching techniques: see Woodruff'15, Mahoney'16

Questions?