Some Dimensionality Reduction Techniques

Julien Clancy

February 24, 2016

SVD

Dimensionality Reduction

- Situation: dataset is very high-dimensional, want to reduce dimensionality
  - Make computationally tractable, or
  - Make easier to understand
- “Compress” the data into fewer dimensions

SVD

- Classical method is SVD
- Data is m samples in $\mathbb{R}^n$, $X \in M_{m,n}(\mathbb{R})$, then

$$X = USV^{\intercal}$$

- S is diagonal with decreasing positive entries, U is "orthogonal", V is orthogonal

What is it Doing?

What does SVD "do"?

1. Finds orthogonal directions of maximum variance; U_i, S_i.
   - Maybe these are features
2. $X = USV^\intercal$. Interpretation:

$$X = \underbrace{\begin{bmatrix} | & & | \\ U_1 & \dots & U_n \\ | & & | \end{bmatrix}}_{\text{put in old coordinates}} \overbrace{\begin{bmatrix} \sigma_1 & & \\ & \ddots & \\ & & \sigma_n \end{bmatrix}}^{\text{scale}} \underbrace{\begin{bmatrix} — & V_1 & — \\ & \vdots & \\ — & V_n & — \end{bmatrix}}_{\text{put in new coordinates}}$$

3. Essentially, finds which hyperplane your data lies in

What is it Doing? Continued

- As a decomposition:

$$X_i = \sum_j \sigma_j \langle X_i, V_j \rangle U_j$$

- As a coordinate representation:

$$X_i \mapsto \Big(\sigma_j \langle X_i, V_j \rangle X_i \Big)_j$$

SVD Dimensionality Reduction

Dimensionality reduction: taking the first k singular values gives the k-dimensional *linear* embedding keeping the most variance

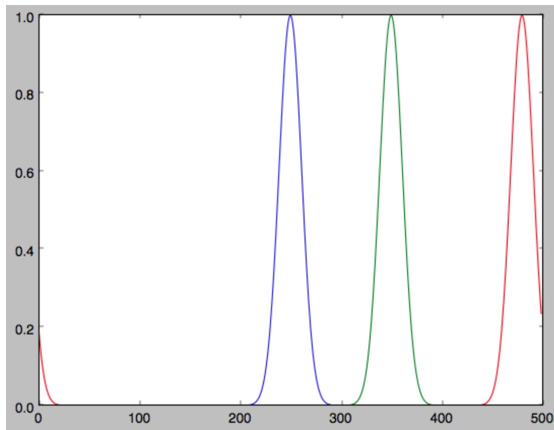
$$X = U \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \sigma_3 & & \\ & & & \sigma_4 & \\ & & & & \ddots \end{bmatrix} V^\intercal$$

$$\overline{X} = U \begin{bmatrix} \sigma_1 & & & & \\ & \sigma_2 & & & \\ & & \sigma_3 & & \\ & & & 0 & \\ & & & & \ddots \end{bmatrix} V^\intercal$$

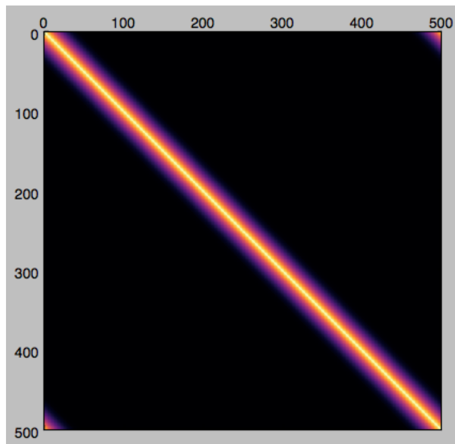
Just drop the lowest coordinates from the representation on the last slide

Very Simple Example

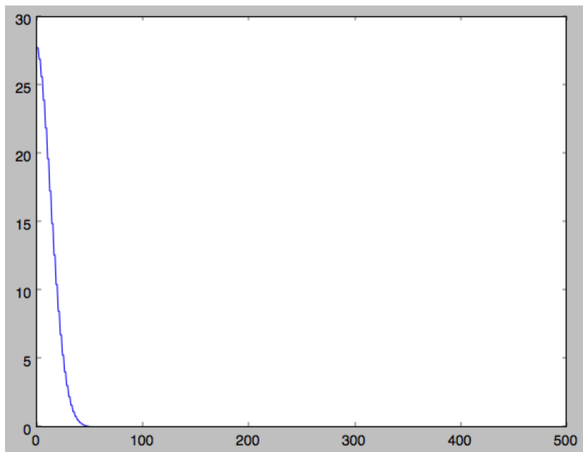
- Toy dataset: shifted kernel functions



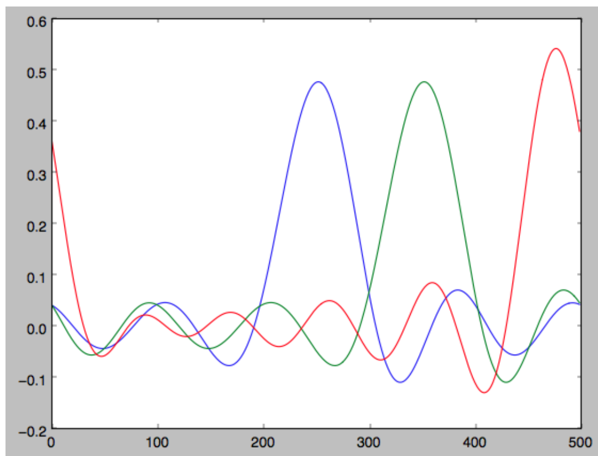
Pairwise Distances



Spectrum (Singular Values)

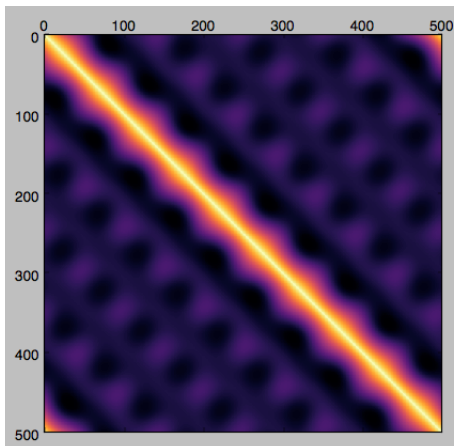


10D Embedding



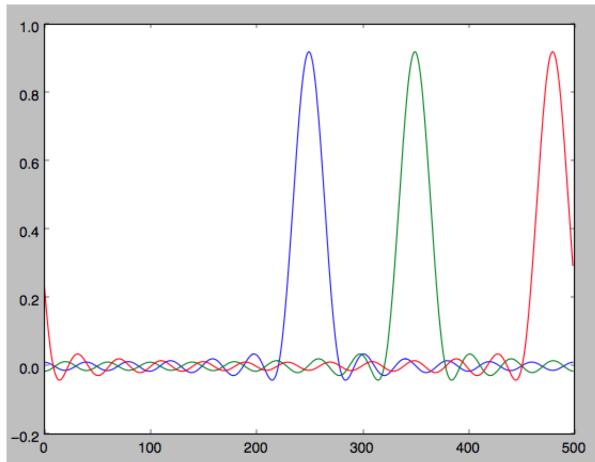
Accounts for 68% of the variance (but gets the most important part)

10D Embedding - Pairwise Distances



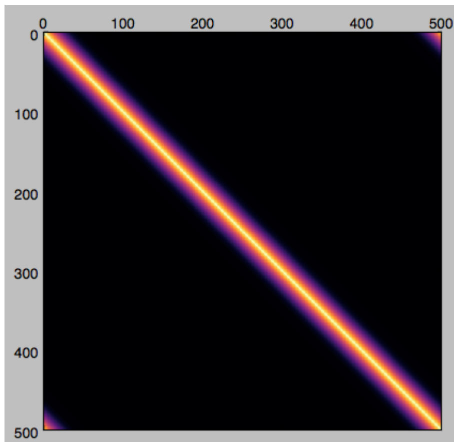
Still very good

25D Embedding



Accounts for 99% of the variance

25D Embedding - Pairwise Distances



Almost perfect

Pairwise Distances

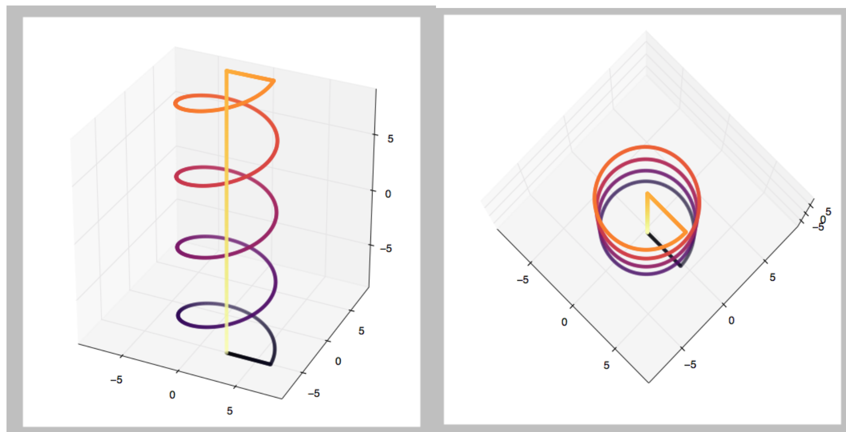
- These last slides: why care about pairwise distances?
- That's where the "geometry" of the data is
  - Ex: clustering.
  - Depends only on (local) distances
  - Theme: we define local distances that have meaning but global ones are murkier
- Driving force of the rest of the next section

Diffusion Maps

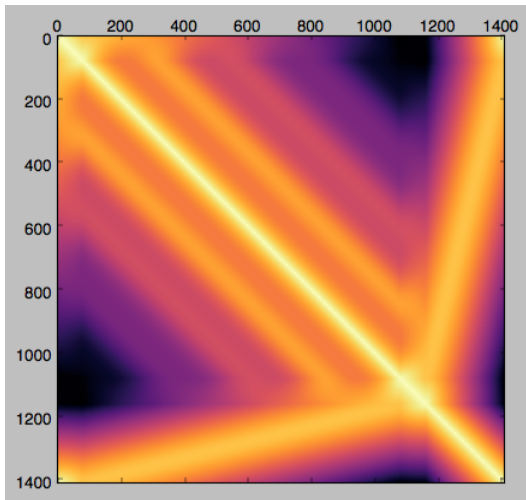
Nonlinear Dimensionality Reduction

- Have very high-dimensional data that (you suspect) has few "underlying parameters"
  - (Manifold embedded in $\mathbb{R}^n$)
- Another way of saying it: compressible, not as complex as the space it's in
- **Not** linear data, else SVD would work perfectly

Helix Data

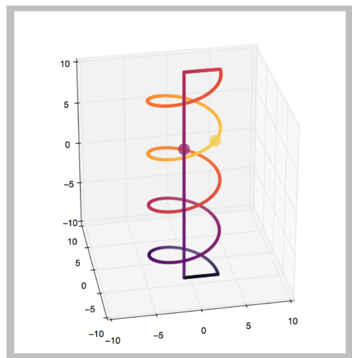


Helix Pairwise Distances



The Structure of the Data

What is this data? It's a circle. Moving "along" the helix gives you a circle. This "intrinsic distance" on the helix is very different from the Euclidean distance:



We want a represenation that better captures the intrinsic distance: get the circle back.

How to Get the Structure

- Tools which can get you the circle back are in the field of **geometry learning**
- Relevance to dimensionality reduction: if your data has low-dimensional geometry but is in a high-dimensional space, geometry learning should give you a low-dimensional representation
  - The intrinsic representation
  - Preserves local distances

Diffusion Maps

- **Diffusion maps** is a geometry learning tool
- Basic idea: aggregate local distances to get global distances
  - Ones that reflect the *intrinsic* geometry, not the embedded geometry
- Data is $\{x_i\} \subset \mathbb{R}^N$, N large/capital
- First, define a notion of "similarity" of data points, call it k
  - Usually $k(x, y) = e^{-\|x_i - x_j\|^2/\varepsilon^2}$
  - Or, replace $\|\cdot\|$ by your own metric
- Then define a random walk/Markov chain on the dataset
  - Probability of stepping from x_i to x_j is

$$P_{ij} = p(x_i, x_j) \propto_i k(x_i, x_j)$$

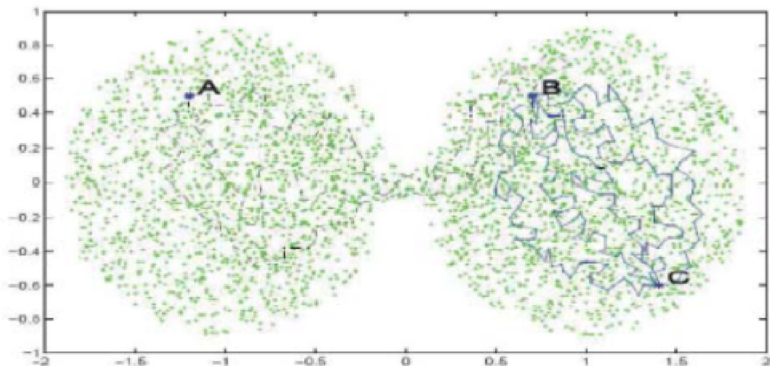
Diffusion Maps Continued

- Use a Markov chain to define the **diffusion distance**

$$D_t(x_i, x_j) = \sum_k \frac{1}{d_k} |P_{ik}^t - P_{kj}^t|^2 = \|p_t(x_i, \cdot) - p_t(x_j, \cdot)\|_{D^{-1}}$$

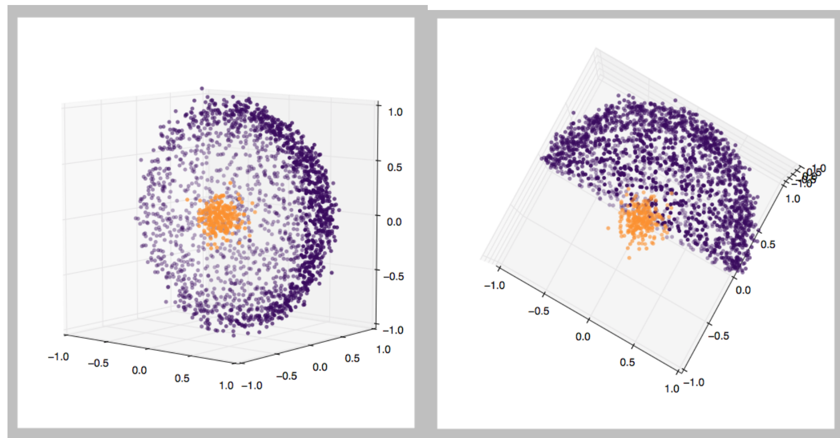
- What does this do? Counts all paths of length t between x_i, x_j, weighted by probability (scaled by the sampling density of each point)
- Hopefully local distances reflect the intrinsic geometry and we "rebuild" global geometry from those

Diffusion Distance Visualization



A is far from B and C, but B and C are very close. (Figure from Peter Jones)

Another Example



The points in the hemisphere are close together, as are the points in the pit, but they are mutually very far apart.

Why Random Walks, Why the Kernel?

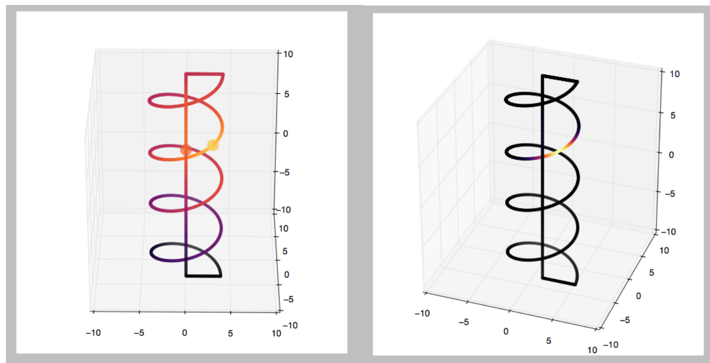


Figure : Left: Euclidean distance. Right: Kernel of Euclidean distance. Note how it approximates the intrinsic distance.

- The problem: every point on the helix is kind of close to the rod
- Kernel solves this, exponentially kills the distance

Actually Computing It

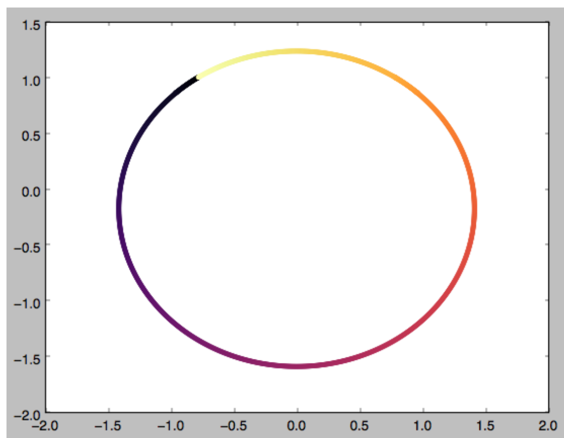
- Fancy distance, but how do you actually measure it? Summing is impractical
- P is the Markov matrix, $P = D^{-1}K$, similar to symmetric matrix
- (λ_i, ψ_i) its eigenvalues/vectors
- The **diffusion maps** are

$$x_i \mapsto (\lambda_1^t \psi_1(x_i), \ldots, \lambda_n^t \psi_n(x_i))$$

- Theorem: This embeds the dataset in Euclidean space with the diffusion distance
- Theorem: If you scale it right and sample a manifold well enough, this converges to the (intrinsic/Riemannian) distance on the manifold

Ex 1: Diffusion on the Helix

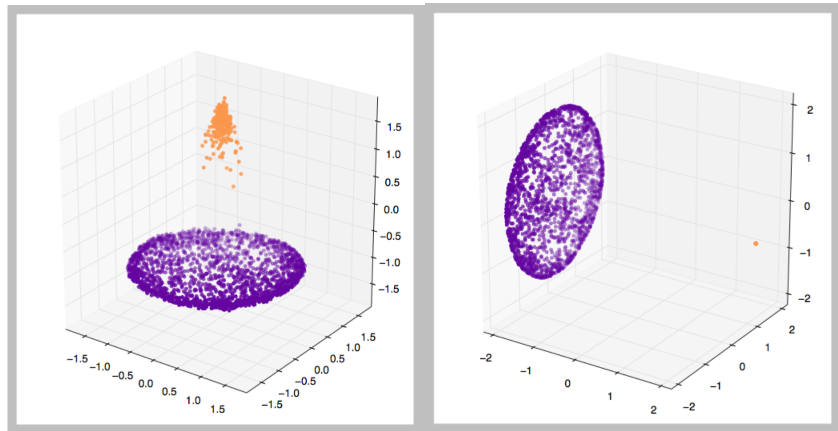
Diffusion actually run on the helix dataset: recovers the intrinsic coordinate



(Colored by the intrinsic coordinate)

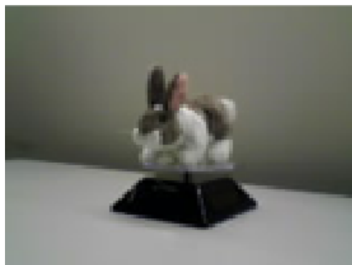
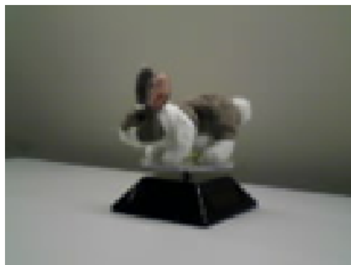
Ex 2: Diffusion on the Hemisphere

Diffusion run on the hemisphere set, with two different bandwidth kernels



Ex 3: Diffusion on Images

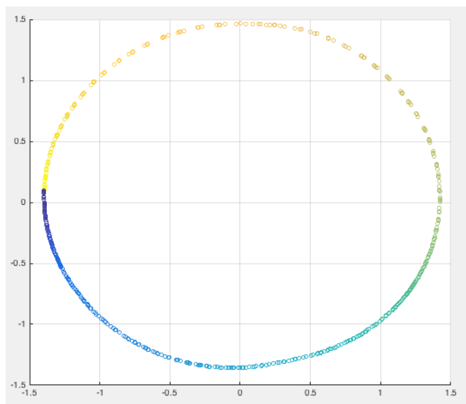
Dataset (thanks to Roy Lederman) consisting of images of a toy bunny on a rotating platform

It's $32,000$-dimensional, but the data "is" a circle: only one parameter, the angle.

Ex 3: Diffusion on Images, Continued

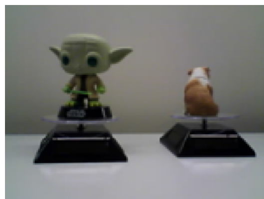
What does the 2D diffusion look like? Surprise,

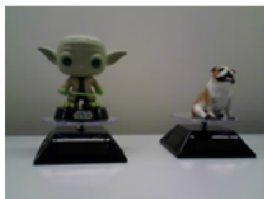


Again, colored by the intrinsic coordinate (known at the time of gathering the data)

Ex 4: Diffusion on Images, a Little More Complex

Similar dataset, but this time two toys rotating at different speeds

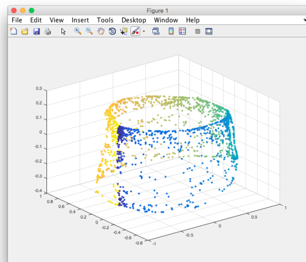


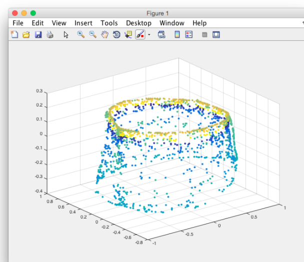


It “is” $S^1 \times S^1$ = a torus

Ex 4: Diffusion on Images, a Little More Complex, Continued

The diffusion bears this out:





These are colored by the intrinsic coordinate of the bulldog and Yoda, respectively (collected at experiment time)

How Do You Use This?

- Part of a pipeline
  - Visualization
  - Dimensionality reduction/feature creation
  - Denoising
  - Linearizing nonlinear problems
- Creates global features from local parameters
- Important point: you don't need to use the Euclidean distance in diffusion maps. Use any similarity you want (just make sure that very similar = small metric, dissimilar = large metric)
- Exploratory data analysis: do this first, get an idea of what your data is, start doing other things
- **Does not** replace SVD/PCA; use both when appropriate

Random Projections

Good Projections

- Projections are good for dimensionality reduction because they are easy to compute
- Good property of SVD: computes projections into k dimensions that preserve the most variance
- Bad property of SVD: very expensive: $O(dn^2 + d^2 n)$ (constants are apparently something like 4 and 22, respectively) for n data points in d dimensions
- Want a replacement that doesn't distort pairwise distances too much and is very fast

Random Projections

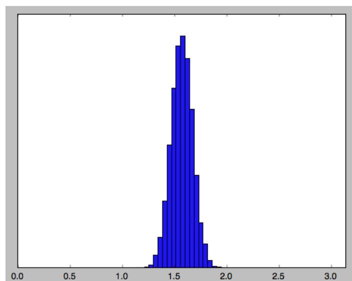
- Turns out, random projections do this just fine
- Theorem (Johnson-Lindenstrauss): If you have n data points $\{x_i\}$ in d-dimensional space and you project them as $\{y_i\}$ onto a random subspace of dimension $O(\log n)$, with high probability

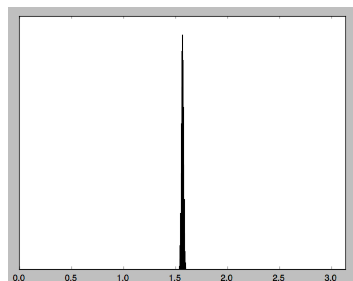
$$(1-\varepsilon)\|x_i - x_j\|^2 \leq \|y_i - y_j\|^2 \leq (1+\varepsilon)\|x_i - x_j\|^2$$

- How to do this in practice: Choose k Gaussian random vectors v_j and use $x_i \mapsto (\langle x_i, v_j \rangle)_j$. Time is kdn.

High-Dimensional Geometry

- "But don't we have to orthogonalize the vectors v_j so that the inner products are actually a projection onto a random subspace?"
- No, and actually that would amount to doing SVD. In high-dimensional space the expected angle between two vectors is 90 degrees.





(Histograms for angles in 100 and 10000 dimensions)

Sample Use Case

- Analyzing large/high-D datasets on your own computer: random projections are a lot faster
- Ex (I ran into this a few days ago): Want to minimize $\|Ax - b\|_1$ s.t. $x \geq 0$
  - Can be rewritten in a linear program by doubling the dimensions
  - Problem: simplex algorithm has complexity $O(d^3)$ on average
  - Randomly project A and b (same projection), solve the problem in 100 times fewer dimensions, average the results over a few runs
- Slight advantage over SVD: randomized, so no adversarial scenarios. More on that later.

Faster Random Projections

- Turns out, you don't even need to consider Gaussian random vectors
- (Li, Hastie, Church, see also Achlioptas) You can choose your vectors according to the distribution

$$\begin{aligned} \mathbb{P}(\sqrt{s}) &= \frac{1}{2s} \\ \mathbb{P}(-\sqrt{s}) &= \frac{1}{2s} \\ \mathbb{P}(0) &= 1 - \frac{1}{s} \end{aligned}$$

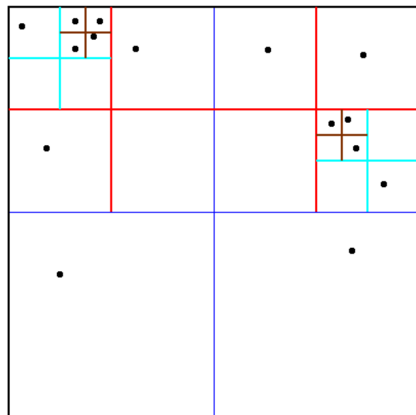
  for $s = \sqrt{d}$ (or even $\frac{d}{\log d}$ if you're feeling lucky)
- A lot faster because the inner products that contain zeros don't have to be calculated: speed up by a factor of $\sqrt{d}$
- $\sqrt{60000} = 245$

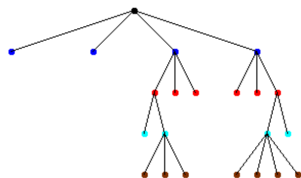
Random Projections Are a Powerful Idea

- Random projections are useful but the thematic idea is what's really powerful
- Two paradigms:
  1. Some high-dimensional problems are very hard to solve; randomly project and use lower-dimensional solutions for massive speedups
  2. Some problems are adversarial; randomly project onto a suitable subset of your space and try to solve the new problem, hoping to "project out" the adversary's tricks
- SVD defeats the purpose of the former case since computing it is so expensive
- It also is inapplicable in the latter case since it is only provides one output; can't get a consensus
  - Ensemble learning: random forests vs. decision trees

Ex. 1: Nearest Neighbors

- Given $X = \{x_i\} \in \mathbb{R}^N$ and a test point x want to find nearest neighbors of x in X.
- In the plane, this is easy: use a quadtree (space partition) for $\log n$ query time





Ex. 1: Nearest Neighbors Continued

- If $N > 2$ your quadtree is a 2^N-tree; if $N = 100$ this does not fit in a computer
- One solution (originally Kleinberg, version here is Jones, Osipov, Rokhlin):
    1. Randomly project[1] data into low dimensions
    2. Use a quadtree-style partition to find candidates for nearest neighbors, throw them in a candidate set S
    3. Do this several times; check the members of S to find approximate nearest neighbors
    4. (Some more tricks)
- Speed is $(n \log n)(d \log d)$

[1] They use random rotations for speed reasons, which are equivalent

Ex. 2: DNA Sequence Alignment

- Want to match many small substrings, with corruptions, to a large reference string S, and find best (Hamming-minimizing) mathes
- Naive method: scan through for each string, compare Hamming distances
- Better method (Roy Lederman):
  1. Generate a random permutation σ
  2. Permute all substrings $s_i \in S$ as $\sigma(s_i)$ and sort them lexigraphically into an array A
  3. Given a test string t find where $\sigma(t)$ fits in A using binary search/something faster
  4. If you got lucky σ moved all of t's corruptions to the end of the string, and then the "true" closest match for $\sigma(t)$ is within K steps in the sorted array
  5. Do this for linearly many permutations and you will get lucky at least once with very very high probability

Ex. 2: DNA Sequence Alignment Benchmarks

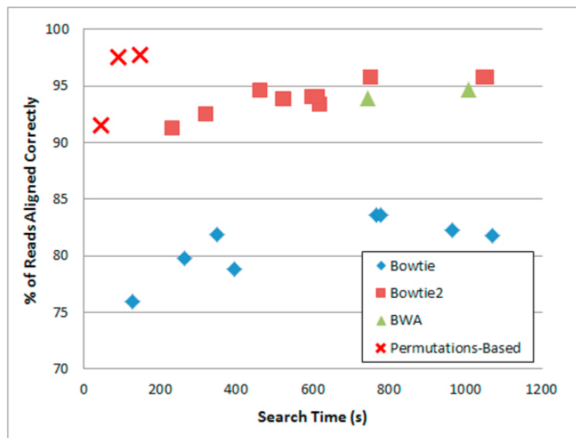
How well does it do?

Software	Search time (s)		
	SRR023337_1 78 bp	ERR009392_1 108 bp	ERR016249_1 160 bp
Bowtie -v 3	233	256	773
Bowtie -n 2	144	334	1560
Bowtie -n 2 -k 10	658	1142	2830
Bowtie2 -very-fast	179	285	440
Bowtie2 -sensitive	328	654	853
Bowtie2 -very-sensitive	812	1488	1855
Bowtie2 -very-sensitive -k 10	1121	2430	3869
BWA -o 0	548	860	2434
Permutations-based (mode 1)	65	68	111
Permutations-based (mode 2)	147	151	145
Permutations-based (fast)	**35**	**39**	**57**

Very fast (figure from Roy Lederman's website)

Ex. 2: DNA Sequence Alignment Benchmarks

How well does it do?



Matches many sequences (figure from Roy Lederman's website)

Ex. 3: Motif Detection in DNA Sequences

- Problem: Find "motifs" in nucleotide sequences which have been corrupted
- The (ℓ, d)-motif problem
  - Given t nucleotide sequences of length n each
  - Each contains a copy of M corrupted in d places ($|M| = \ell$)
  - Recover M
- Problem studied by Pevzner and Sze; paper by Buhler and Tompa solves very hard versions using random Hamming projections

Ex. 3: Their Approach

- Hamming projection is $P([x_1, \ldots, x_\ell]) = [x_{i_1}, \ldots, x_{i_k}]$ defined on length-ℓ substrings
- Randomly choose P, “throw away” the information not in P
- If k is small enough it’s likely that M and M' (a corruption) will have $P(M) = P(M')$
- Also if k is not too small it is unlikely that other substrings S will have $P(S) = P(M)$
- At the end choose the equivalence classes of size above some threshold, look for the motif

Ex. 3: How it Works

- Minimizing the probability that $P(S) = P(M)$ if S is random
  - $t(n - \ell + 1)$ length-ℓ substrings
  - If k is s.t. $4^k > t(n - \ell + 1)$ the average equivalence class contains < 1 substring
- Maximizing the probability that $P(M') = P(M)$
  - d corruptions, so $\ell - d$ non-corruptions
  - If $k < \ell - d$ then there's a reasonably good chance $P(M') = P(M)$
- Run lots of times, choose the equivalences classes larger than a threshold s
- s, k, and the number of times you have to run it are all small and can be estimated robustly
- To get the motif back from the large buckets, use maximum likelihood to find a distribution of corruptions