COMPSCI 514: Algorithms for Data Science

Cameron Musco

University of Massachusetts Amherst. Fall 2024.
Lecture 11

Logistics

- Problem Set 2 is due Friday at 11:59pm.
- My office hours today are in LGRC A104A.
- The midterm exam is next Thursday 7-9pm.
- I will hold review sessions on Tuesday, Wednesday, and Thursday in class. See Piazza for details on times and on midterm review material.
- If you need extended time on the exam, you should have received an email from me. Reach out if you have not.

Summary

Last Class: Similarity Search and LSH

- Fast similarity search via locality sensitive hashing.
- Jaccard similarity and MinHashing for Jaccard LSH.

Summary

Last Class: Similarity Search and LSH

- Fast similarity search via locality sensitive hashing.
- Jaccard similarity and MinHashing for Jaccard LSH.

This Class:

- Finish up LSH – SimHash for cosine similarity.
- Start on randomized methods for compressing high dimensional data.
- Low-distortion embeddings and the Johnson-Lindenstraus (JL) Lemma.

Hashing for Duplicate Detection

	Hash Table	**Bloom Filters**	**MinHash Similarity Search**	**Distinct Elements**
Goal	Check if x is a duplicate of any y in database and return y.	Check if x is a duplicate of y in database.	Check if x is a duplicate of any y in database and return y.	Count # of items, excluding duplicates.
Space	$O(n)$ items	$O(n)$ bits	$O(n \cdot t)$ items (when t tables used)	$O\left(\frac{\log\log n}{\epsilon^2}\right)$
Query Time	$O(1)$	$O(1)$	Potentially o(n)	NA
Approximate Duplicates?	✗	✗	✓	✗

All different variants of detecting duplicates/finding matches in large datasets. An important problem in many contexts.

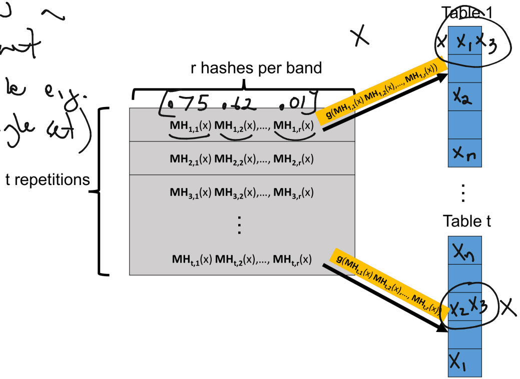
Balancing LSH Hit Rate and Query Time

In similarity search with LSH, we use repetition to balance a small probability of false negatives (a high hit rate) with a small probability of false positives (a small query time.)

x_1 – each is a document (could be e.g. a single set)

⋮

x_n



x_1, x_2, x_3

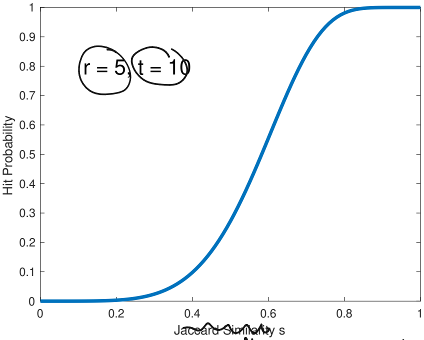
potential neighbors to x

Create t hash tables. Each is indexed into not with a single MinHash value, but with a length-r signature of values, appended together.

The s-curve

Using t repetitions each with a signature of r hash values, the probability that x and y with collision probability $\Pr[h(x) = h(y)] = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.

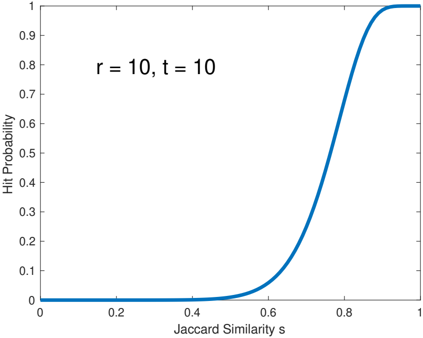
$J(x,y) = s$



hash collision probability

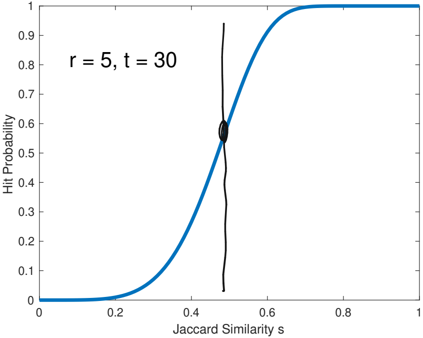
The s-curve

Using t repetitions each with a signature of r hash values, the probability that x and y with collision probability $\Pr[h(x) = h(y)] = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.



The s-curve

Using t repetitions each with a signature of r hash values, the probability that x and y with collision probability $\Pr[h(x) = h(y)] = s$ match in at least one repetition is: $1 - (1 - s^r)^t$.



Generalizing Locality Sensitive Hashing

Repetition and s-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

Generalizing Locality Sensitive Hashing

Repetition and *s*-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

Generalizing Locality Sensitive Hashing

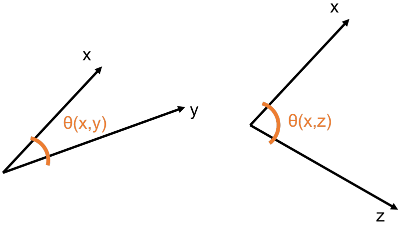
Repetition and s-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

Generalizing Locality Sensitive Hashing

Repetition and *s*-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

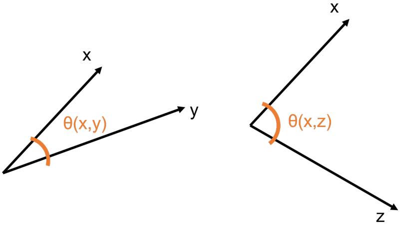
- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.



Generalizing Locality Sensitive Hashing

Repetition and *s*-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.

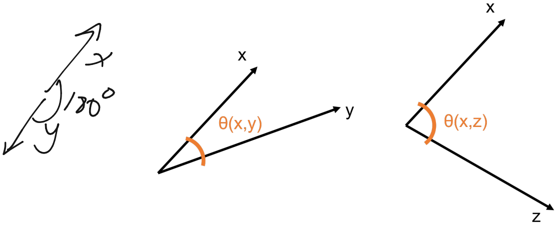


Cosine Similarity: $\cos(\theta(x,y))$

Generalizing Locality Sensitive Hashing

Repetition and *s*-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.



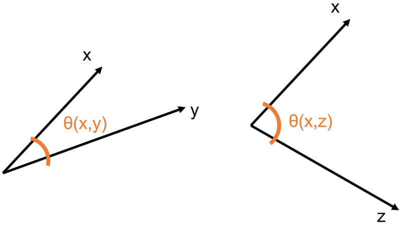
Cosine Similarity: $\cos(\theta(x,y))$

- $\underline{\cos(\theta(x,y)) = 1}$ when $\underline{\theta(x,y) = 0^\circ}$ and $\underline{\cos(\theta(x,y)) = 0}$ when $\underline{\theta(x,y) = 90^\circ}$, and $\underline{\cos(\theta(x,y)) =}\ -1$ when $\theta(x,y) = 180^\circ$

Generalizing Locality Sensitive Hashing

Repetition and *s*-curve tuning can be used for fast similarity search with any similarity metric, given a locality sensitive hash function for that metric.

- LSH schemes exist for many similarity/distance measures: hamming distance, cosine similarity, etc.



Cosine Similarity: $\boxed{\cos(\theta(x,y)) = \frac{\langle x,y\rangle}{\|x\|_2 \cdot \|y\|_2}}$.

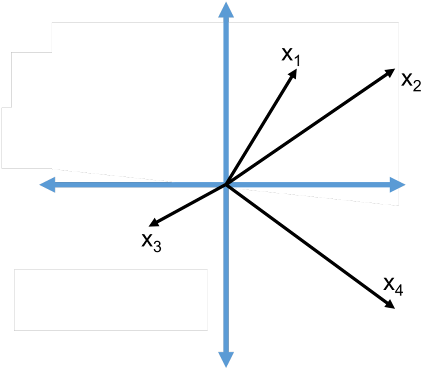
- $\cos(\theta(x,y)) = 1$ when $\theta(x,y) = 0°$ and $\cos(\theta(x,y)) = 0$ when $\theta(x,y) = 90°$, and $\cos(\theta(x,y)) = -1$ when $\theta(x,y) = 180°$

SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.

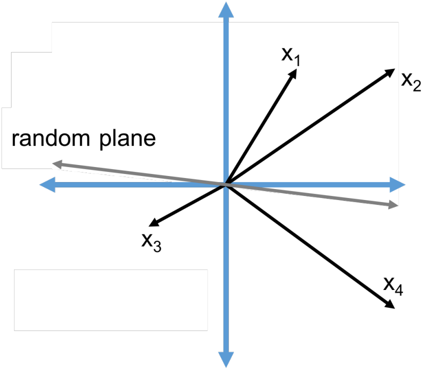
SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.



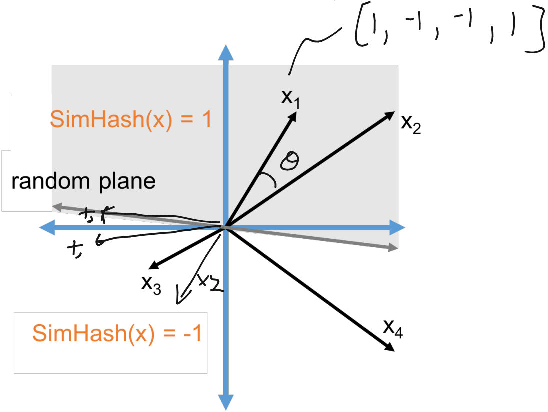
SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.



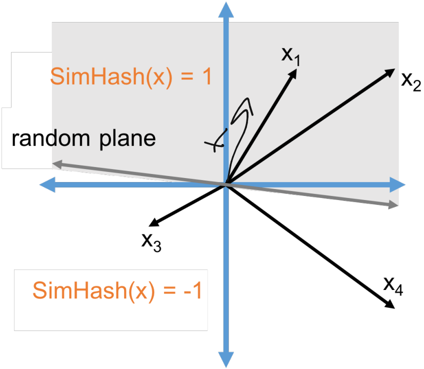
SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.



SimHash for Cosine Similarity

SimHash Algorithm: LSH for cosine similarity.



$SimHash(x) = \mathrm{sign}(\langle x, t\rangle)$ for a random vector t.

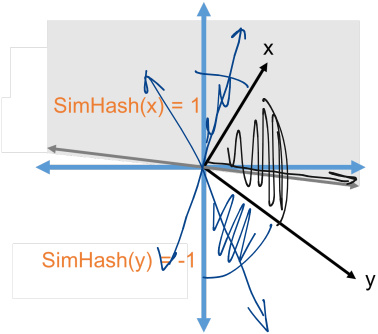
SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.

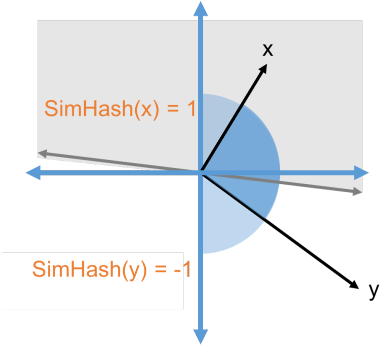


$$\frac{\theta(x,y)}{\pi}$$

SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

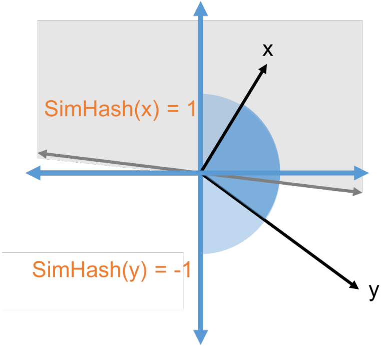
$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.



SimHash for Cosine Similarity

What is $\Pr\left[SimHash(x) = SimHash(y)\right]$?

$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.

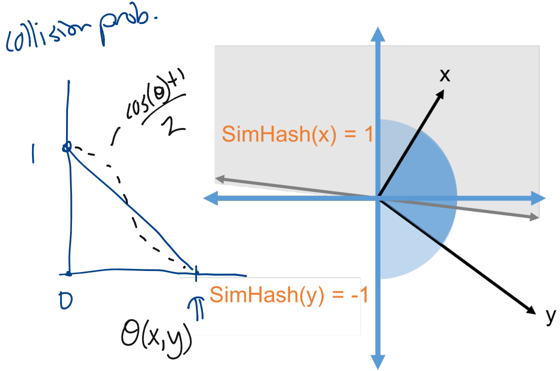


- $\Pr\left[SimHash(x) \neq SimHash(y)\right] = \frac{\theta(x,y)}{\pi}$

SimHash for Cosine Similarity

What is $\Pr[SimHash(x) = SimHash(y)]$?

$SimHash(x) \neq SimHash(y)$ when the plane separates x from y.



- $\Pr[SimHash(x) \neq SimHash(y)] = \frac{\theta(x,y)}{\pi}$
- $\Pr[SimHash(x) = SimHash(y)] = 1 - \underbrace{\frac{\theta(x,y)}{\pi}} \approx \underbrace{\frac{\cos(\theta(x,y))+1}{2}}$

Questions on MinHash and Locality Sensitive Hashing?

High Dimensional Data

‘Big Data’ means not just many data points, but many measurements per data point. I.e., very high dimensional data.

High Dimensional Data

‘Big Data’ means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.

High Dimensional Data

'Big Data' means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.
- A 3 minute Youtube clip with a resolution of 500×500 pixels at 15 frames/second with 3 color channels is a recording of ≥ 2 billion pixel values. Even a 500×500 pixel color image has $750,000$ pixel values.

High Dimensional Data

'Big Data' means not just many data points, but many measurements per data point. I.e., very high dimensional data.

- Twitter has 321 million active monthly users. Records (tens of) thousands of measurements per user: who they follow, who follows them, when they last visited the site, timestamps for specific interactions, how many tweets they have sent, the text of those tweets, etc.
- A 3 minute Youtube clip with a resolution of 500×500 pixels at 15 frames/second with 3 color channels is a recording of ≥ 2 billion pixel values. Even a 500×500 pixel color image has $750,000$ pixel values.
- The human genome contains 3 billion+ base pairs. Genetic datasets often contain information on 100s of thousands+ mutations and genetic markers.

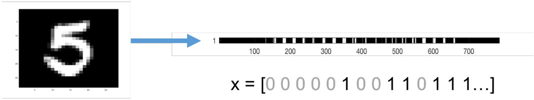
Data as Vectors and Matrices

In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.

Data as Vectors and Matrices

In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.

ATAGCCGTAGT → x = [1 2 1 3 4 4 3 2 1 3 4]

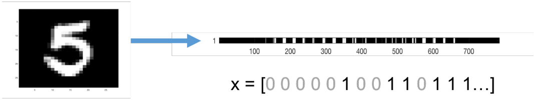


x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...]

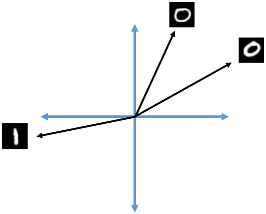
Data as Vectors and Matrices

In data analysis and machine learning, data points with many attributes are often stored, processed, and interpreted as high dimensional vectors, with real valued entries.

ATAGCCGTAGT → x = [1 2 1 3 4 4 3 2 1 3 4]



x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...]



Similarities/distances between vectors (e.g., $\langle x, y\rangle$, $\|x - y\|_2$) have meaning for underlying data points.

Datasets as Vectors and Matrices

Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

Data Points: $\underline{\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n} \in \underline{\mathbb{R}^d}$.

$$\begin{array}{cc} & 1 \\ d & \begin{bmatrix} x_i \end{bmatrix} \end{array}$$

Data Set: $\underline{X \in \mathbb{R}^{n \times d}}$ with i^{th} row equal to $\vec{x}_i^T$.

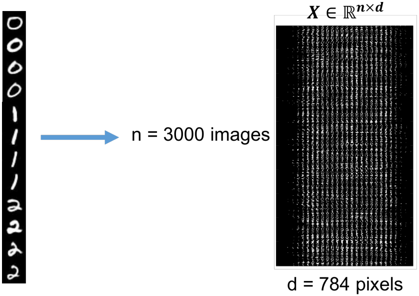
$$\begin{array}{cc} & d \\ n & \begin{bmatrix} - \; x_1^T \; - \\ - \; x_2^T \; - \\ \\ \\ \\ \end{bmatrix} \end{array}$$

Datasets as Vectors and Matrices

Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

Data Points: $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d$.

Data Set: $X \in \mathbb{R}^{n \times d}$ with i^{th} row equal to $\vec{x}_i^T$.

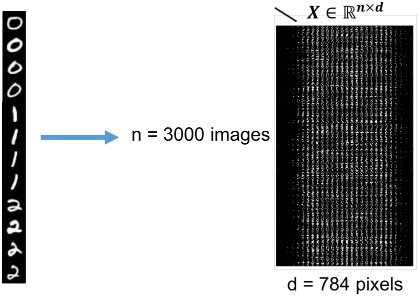


Datasets as Vectors and Matrices

Data points are interpreted as high dimensional vectors, with real valued entries. Data set is interpreted as a matrix.

Data Points: $\vec{x}_1, \vec{x}_2, \ldots, \vec{x}_n \in \mathbb{R}^d$.

Data Set: $X \in \mathbb{R}^{n \times d}$ with i^{th} row equal to $\vec{x}_i^T$.



Many data points $n \implies$ tall. Many dimensions $d \implies$ wide.

Dimensionality Reduction

Dimensionality Reduction: Compress data points so that they lie in many fewer dimensions.

Dimensionality Reduction

Dimensionality Reduction: Compress data points so that they lie in many fewer dimensions.

$$\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d.$$

x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...]

$\tilde{x}$ = [-5.5 4 3.2 -1]

Dimensionality Reduction

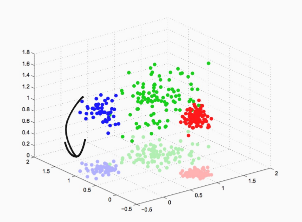
Dimensionality Reduction: Compress data points so that they lie in many fewer dimensions.

$$\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d.$$

$\mathbb{R}^{784}$ $\mathbb{R}^{4}$

$\longrightarrow$ x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...] $\longrightarrow$ $\tilde{x}$ = [-5.5 4 3.2 -1]

'Lossy compression' that still preserves important information about the relationships between $\vec{x}_1, \ldots, \vec{x}_n$.



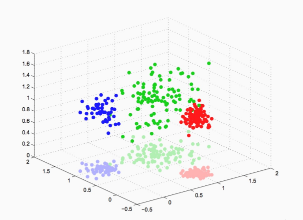
Dimensionality Reduction

Dimensionality Reduction: Compress data points so that they lie in many fewer dimensions.

$$\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d \rightarrow \tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m \text{ for } m \ll d.$$

$\rightarrow$ x = [0 0 0 0 0 1 0 0 1 1 0 1 1 1...] $\rightarrow$ $\tilde{x}$ = [-5.5 4 3.2 -1]

‘Lossy compression’ that still preserves important information about the relationships between $\vec{x}_1, \ldots, \vec{x}_n$.



Generally will not consider directly how well $\tilde{x}_i$ approximates $\vec{x}_i$.

Dimensionality Reduction

Dimensionality reduction is one of the most important techniques in data science. What methods have you heard of?

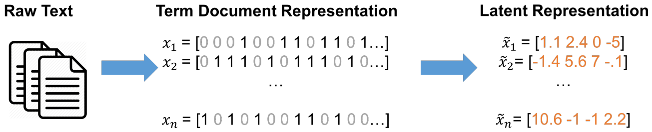
- project to low dimensions (PCA)
- Spectral clustering ← SVD TSNE
- VAE (autoencoders)

Dimensionality Reduction

Dimensionality reduction is one of the most important techniques in data science. What methods have you heard of?

- Principal component analysis
- Latent semantic analysis (LSA)

PCA



Raw Text

Term Document Representation

x_1 = [0 0 0 1 0 0 1 1 0 1 1 0 1…]
x_2 = [0 1 1 1 0 1 0 1 1 1 0 1 0…]
…
x_n = [1 0 1 0 1 0 0 1 1 0 1 0 0…]

Latent Representation

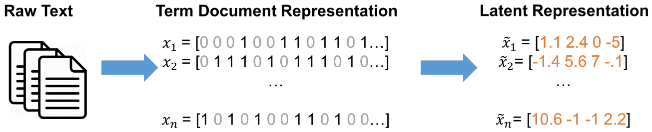
$\tilde{x}_1$ = [1.1 2.4 0 -5]
$\tilde{x}_2$= [-1.4 5.6 7 -.1]
…
$\tilde{x}_n$= [10.6 -1 -1 2.2]

- Linear discriminant analysis
- Autoencoders

Dimensionality Reduction

Dimensionality reduction is one of the most important techniques in data science. What methods have you heard of?

- Principal component analysis
- Latent semantic analysis (LSA)



Raw Text

Term Document Representation

x_1 = [0 0 0 1 0 0 1 1 0 1 1 0 1...]
x_2 = [0 1 1 1 0 1 0 1 1 1 0 1 0...]
...
x_n = [1 0 1 0 1 0 0 1 1 0 1 0 0...]

Latent Representation

$\tilde{x}_1$ = [1.1 2.4 0 -5]
$\tilde{x}_2$= [-1.4 5.6 7 -.1]
...
$\tilde{x}_n$= [10.6 -1 -1 2.2]

- Linear discriminant analysis
- Autoencoders

Compressing data makes it more efficient to work with. May also remove extraneous information/noise.

Embeddings for Euclidean Space

Euclidean Low Distortion Embedding: Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and error parameter $\underline{\epsilon \geq 0}$, find $\underline{\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m}$ (where $\underline{m \ll d}$) such that for all $i, j \in [n]$:

$$(1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

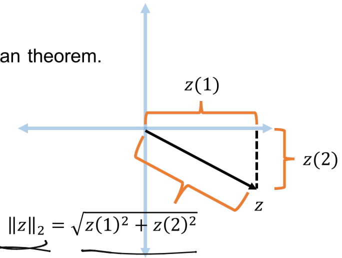
Embeddings for Euclidean Space

Euclidean Low Distortion Embedding: Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and error parameter $\epsilon \geq 0$, find $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m$ (where $m \ll d$) such that for all $i, j \in [n]$:

$$(1-\epsilon)\underbrace{\|\vec{x}_i - \vec{x}_j\|_2} \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Recall that for $\vec{z} \in \mathbb{R}^n$, $\|\vec{z}\|_2 = \sqrt{\sum_{i=1}^n \vec{z}(i)^2}$.

Pythagorean theorem.



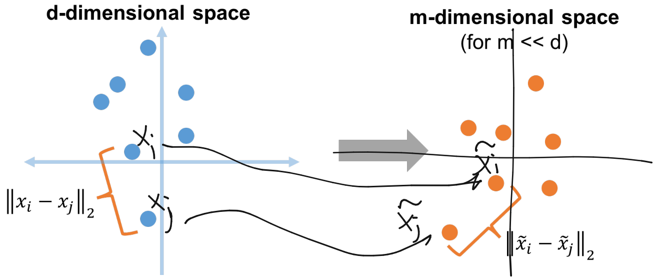
$$\underline{\|z\|_2} = \underline{\sqrt{z(1)^2 + z(2)^2}}$$

Embeddings for Euclidean Space

Euclidean Low Distortion Embedding: Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and error parameter $\epsilon \geq 0$, find $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m$ (where $m \ll d$) such that for all $i, j \in [n]$:

$\varepsilon = .1$ $\|x_i - x_j\|_2 = 2$

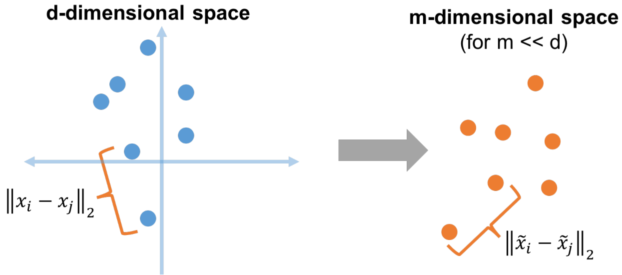
$$\underset{1.8}{(1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2} \leq \|\underline{\tilde{x}_i - \tilde{x}_j}\|_2 \leq \underset{2.2}{(1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2}.$$



Embeddings for Euclidean Space

Euclidean Low Distortion Embedding: Given $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and error parameter $\epsilon \geq 0$, find $\tilde{x}_1, \ldots, \tilde{x}_n \in \mathbb{R}^m$ (where $m \ll d$) such that for all $i, j \in [n]$:

$$(1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$



Can use $\tilde{x}_1, \ldots, \tilde{x}_n$ in place of $\vec{x}_1, \ldots, \vec{x}_n$ in clustering, SVM, linear classification, near neighbor search, etc.

The Johnson-Lindenstrauss Lemma

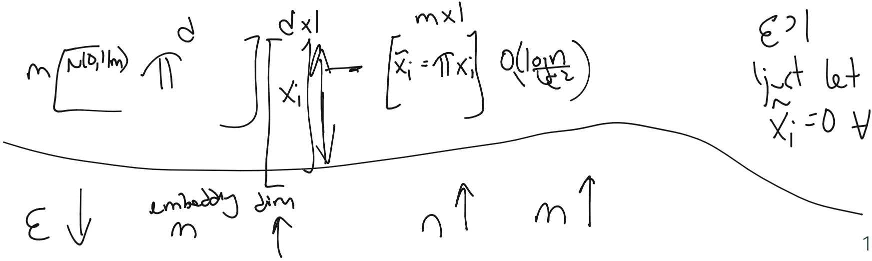
The Johnson-Lindenstrauss Lemma tells us that for any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and any $\epsilon > 0$, we can find an ϵ-distortion embedding into m dimensions, where m depends only on the error parameter ϵ and the number of points n, but not the original dimension d.

The Johnson-Lindenstrauss Lemma

Johnson-Lindenstrauss Lemma: For any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $\mathbf{\Pi} : \mathbb{R}^d \rightarrow \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i, j: \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Further, if $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ has each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, it satisfies the guarantee with high probability.



$\epsilon > 1$

just let

$\tilde{x}_i = 0 \ \forall i$

The Johnson-Lindenstrauss Lemma

Johnson-Lindenstrauss Lemma: For any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $\mathbf{\Pi} : \mathbb{R}^d \to \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i,j: \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Further, if $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ has each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, it satisfies the guarantee with high probability.

For $d = 1$ trillion, $\epsilon = .05$, and $n = 100,000$, $m \approx 6600$.

The Johnson-Lindenstrauss Lemma

Johnson-Lindenstrauss Lemma: For any set of points $\vec{x}_1, \ldots, \vec{x}_n \in \mathbb{R}^d$ and $\epsilon > 0$ there exists a linear map $\mathbf{\Pi} : \mathbb{R}^d \to \mathbb{R}^m$ such that $m = O\left(\frac{\log n}{\epsilon^2}\right)$ and letting $\tilde{x}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i, j : \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{x}_i - \tilde{x}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$

Further, if $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ has each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, it satisfies the guarantee with high probability.

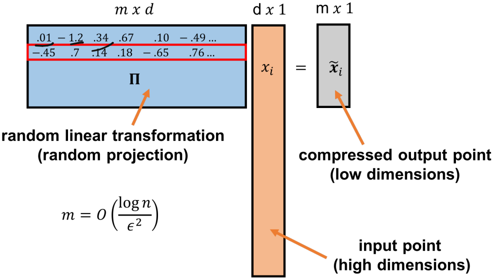
For $d = 1$ trillion, $\epsilon = .05$, and $n = 100,000$, $m \approx 6600$.

Very surprising! Powerful result with a simple construction: applying a random linear transformation to a set of points preserves distances between all those points with high probability.

Random Projection

For any $\vec{x}_1, \ldots, \vec{x}_n$ and $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ with each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i, j: \; (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$



random linear transformation (random projection)

compressed output point (low dimensions)

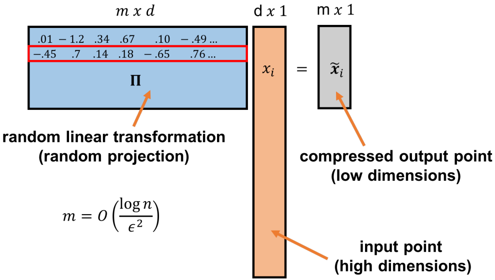
input point (high dimensions)

$$m = O\left(\frac{\log n}{\epsilon^2}\right)$$

Random Projection

For any $\vec{x}_1, \ldots, \vec{x}_n$ and $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ with each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i,j: \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$



random linear transformation (random projection)

compressed output point (low dimensions)

input point (high dimensions)

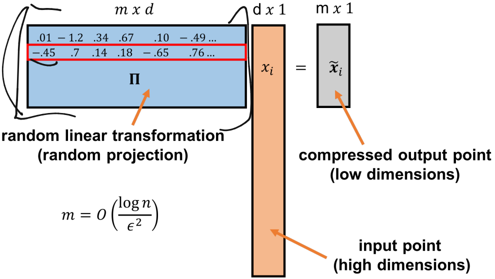
$$m = O\left(\frac{\log n}{\epsilon^2}\right)$$

- $\mathbf{\Pi}$ is known as a random projection. It is a random linear function, mapping length d vectors to length m vectors.

Random Projection

For any $\vec{x}_1, \ldots, \vec{x}_n$ and $\mathbf{\Pi} \in \mathbb{R}^{m \times d}$ with each entry chosen i.i.d. from $\mathcal{N}(0, 1/m)$, with high probability, letting $\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i$:

$$\text{For all } i,j: \ (1-\epsilon)\|\vec{x}_i - \vec{x}_j\|_2 \leq \|\tilde{\mathbf{x}}_i - \tilde{\mathbf{x}}_j\|_2 \leq (1+\epsilon)\|\vec{x}_i - \vec{x}_j\|_2.$$



random linear transformation (random projection)

compressed output point (low dimensions)

$$m = O\left(\frac{\log n}{\epsilon^2}\right)$$

input point (high dimensions)

- $\mathbf{\Pi}$ is known as a random projection. It is a random linear function, mapping length d vectors to length m vectors.
- $\mathbf{\Pi}$ is data oblivious. Stark contrast to methods like PCA.

Algorithmic Considerations

- Many alternative constructions: ±1 entries, sparse (most entries 0), Fourier structured, etc. $\implies$ more efficient computation of $\underline{\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i.}$

Algorithmic Considerations

- Many alternative constructions: ± 1 entries, sparse (most entries 0), Fourier structured, etc. $\implies$ more efficient computation of $\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i$.
- Data oblivious property means that once $\mathbf{\Pi}$ is chosen, $\underline{\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n}$ can be computed in a stream with little memory.
- Memory needed is just $O(d + \underline{nm})$ vs. $\underline{O(nd)}$ to store the full data set.

x_i $\hat{x}_1 \ldots \tilde{x}_n$

$$m = O\left(\frac{\log n}{\varepsilon^2}\right)$$

independent of d

Algorithmic Considerations

- Many alternative constructions: ± 1 entries, sparse (most entries 0), Fourier structured, etc. $\implies$ more efficient computation of $\tilde{\mathbf{x}}_i = \mathbf{\Pi} \vec{x}_i$.
- Data oblivious property means that once $\mathbf{\Pi}$ is chosen, $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$ can be computed in a stream with little memory.
- Memory needed is just $O(d + nm)$ vs. $O(nd)$ to store the full data set.
- Compression can also be easily performed in parallel on different servers.

Algorithmic Considerations

- Many alternative constructions: ± 1 entries, sparse (most entries 0), Fourier structured, etc. $\implies$ more efficient computation of $\tilde{\mathbf{x}}_i = \mathbf{\Pi}\vec{x}_i$.
- Data oblivious property means that once $\mathbf{\Pi}$ is chosen, $\tilde{\mathbf{x}}_1, \ldots, \tilde{\mathbf{x}}_n$ can be computed in a stream with little memory.
- Memory needed is just $O(d + nm)$ vs. $O(nd)$ to store the full data set.
- Compression can also be easily performed in parallel on different servers.
- When new data points are added, can be easily compressed, without updating existing points.

$\tilde{X}_1 \ldots \tilde{X}_n$ $\Pi X_{n+1} = \tilde{X}_{n+1}$