Algorithms for Processing Massive Data Sets

Purushottam Kar

Department of Computer Science and Engineering,
Indian Institute of Technology, Kanpur

March 11, 2010

Overview

Introduction

- An example from Massive Databases
- An example from Network Monitoring

Overview

Introduction

An example from Massive Databases

An example from Network Monitoring

A Useful Tool

The Random Projection Method

Overview

Overview

Overview

The story so far ...

Algorithm Design 101

- Goal : Solve a computational problem $\mathcal{P}$ using an algorithm $\mathcal{A}$.

Algorithm Design 101

- Goal : Solve a computational problem $\mathcal{P}$ using an algorithm $\mathcal{A}$.
- Assume $\mathcal{P}$ is formalized as a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$.

Algorithm Design 101

- Goal : Solve a computational problem $\mathcal{P}$ using an algorithm $\mathcal{A}$.
- Assume $\mathcal{P}$ is formalized as a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$.
- Given input $x \in \{0,1\}^*$, $\mathcal{A}$ has to output $f(x)$.

Algorithm Design 101

- Goal : Solve a computational problem $\mathcal{P}$ using an algorithm $\mathcal{A}$.
- Assume $\mathcal{P}$ is formalized as a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$.
- Given input $x \in \{0,1\}^*$, $\mathcal{A}$ has to output $f(x)$.
- $\mathcal{A}$ is called an *efficient* algorithm if given input x, $\mathcal{A}$ uses
  - at most $|x|^{c_1}$ space for some absolute constant c_1
  - at most $|x|^{c_2}$ time for some absolute constant c_2

  to give the correct answer.

Algorithm Design 101

- Goal : Solve a computational problem $\mathcal{P}$ using an algorithm $\mathcal{A}$.
- Assume $\mathcal{P}$ is formalized as a function $f : \{0,1\}^* \longrightarrow \{0,1\}^*$.
- Given input $x \in \{0,1\}^*$, $\mathcal{A}$ has to output $f(x)$.
- $\mathcal{A}$ is called an *efficient* algorithm if given input x, $\mathcal{A}$ uses
  - at most $|x|^{c_1}$ space for some absolute constant c_1
  - at most $|x|^{c_2}$ time for some absolute constant c_2

  to give the correct answer.
- Here $|x|$ denotes the size of the input and can be variously defined.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go.
- The input is presented in a RAM-like data structure so that $\mathcal{A}$ can revisit parts of the input again and again.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go.
- The input is presented in a RAM-like data structure so that $\mathcal{A}$ can revisit parts of the input again and again.
- Small integral values of the constants c_1 and c_2 usually translate to algorithms that are efficient *in practice*.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go. (Assumption 1)
- The input is presented in a RAM-like data structure so that $\mathcal{A}$ can revisit parts of the input again and again.
- Small integral values of the constants c_1 and c_2 usually translate to algorithms that are efficient *in practice*.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go. (Assumption 1)
- The input is presented in a RAM-like data structure so that $\mathcal{A}$ can revisit parts of the input again and again. (Assumption 2)
- Small integral values of the constants c_1 and c_2 usually translate to algorithms that are efficient *in practice*.

Some finer points

- $\mathcal{A}$ is presented the complete input in one go. (Assumption 1)
- The input is presented in a RAM-like data structure so that $\mathcal{A}$ can revisit parts of the input again and again. (Assumption 2)
- Small integral values of the constants c_1 and c_2 usually translate to algorithms that are efficient *in practice*. (Assumption 3)

A twist in the tale ...

Working with High Dimensional Databases

- Find the nearest neighbor of a query point in the database.

Working with High Dimensional Databases

- Find the nearest neighbor of a query point in the database.
- Even for moderately high dimensions ($d = 20$), any nearest neighbor search algorithm essentially reduces to sequential search

Working with High Dimensional Databases

- Find the nearest neighbor of a query point in the database.
- Even for moderately high dimensions ($d = 20$), any nearest neighbor search algorithm essentially reduces to sequential search

Example (taken from [Bhattacharya])

Even to obtain a selectivity of 0.0001 on a database having points from the unit cube in 20 dimensions, the range required to be queried in each dimension is 0.63.

Working with High Dimensional Databases

- Find the nearest neighbor of a query point in the database.
- Even for moderately high dimensions ($d = 20$), any nearest neighbor search algorithm essentially reduces to sequential search

Example (taken from [Bhattacharya])

Even to obtain a selectivity of 0.0001 on a database having points from the unit cube in 20 dimensions, the range required to be queried in each dimension is 0.63.

Example (taken from [Dasgupta and Freund2008])

Sizes of indexing structures like k-d trees grow exponentially with dimension (depth grows linearly).

Working with High Dimensional Databases

- Find the nearest neighbor of a query point in the database.
- Even for moderately high dimensions ($d = 20$), any nearest neighbor search algorithm essentially reduces to sequential search

Example (taken from [Bhattacharya])

Even to obtain a selectivity of 0.0001 on a database having points from the unit cube in 20 dimensions, the range required to be queried in each dimension is 0.63.

Example (taken from [Dasgupta and Freund2008])

Sizes of indexing structures like k-d trees grow exponentially with dimension (depth grows linearly).

The Curse of Dimensionality

Working with High Dimensional Databases

- Typical database sizes are huge $> 1,000,000$ entries.

Working with High Dimensional Databases

- Typical database sizes are huge $> 1,000,000$ entries.
- If each entry is very high dimensional then huge amounts of storage required.

Working with High Dimensional Databases

- Typical database sizes are huge $> 1,000,000$ entries.
- If each entry is very high dimensional then huge amounts of storage required.
- More importantly, a linear scan through such a database would be very slow - leading to slow response times.

Working with High Dimensional Databases

- Typical database sizes are huge $> 1,000,000$ entries.
- If each entry is very high dimensional then huge amounts of storage required.
- More importantly, a linear scan through such a database would be very slow - leading to slow response times.
- Hence Assumption 3 does not hold !

Another example ...

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])
  - Wish to detect malicious activity in the network (eg. a Distributed Denial of Service attack)

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])
  - Wish to detect malicious activity in the network (eg. a Distributed Denial of Service attack)
  - Wish to keep track of most visited Internet websites, users with highest usage

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])
  - Wish to detect malicious activity in the network (eg. a Distributed Denial of Service attack)
  - Wish to keep track of most visited Internet websites, users with highest usage
- All these aggregate statistics can be computed if we have the Frequency Vector for the network stream.

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])
  - Wish to detect malicious activity in the network (eg. a Distributed Denial of Service attack)
  - Wish to keep track of most visited Internet websites, users with highest usage
- All these aggregate statistics can be computed if we have the Frequency Vector for the network stream.

Figure: A network stream and the corresponding frequency vector

Working with Streaming Data

- Problem of network monitoring - data being received at a network switch at high (Gigabit) rates (Example taken from [Ganguly])
  - Wish to detect malicious activity in the network (eg. a Distributed Denial of Service attack)
  - Wish to keep track of most visited Internet websites, users with highest usage
- All these aggregate statistics can be computed if we have the Frequency Vector for the network stream.

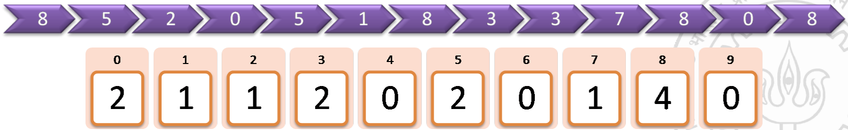

Figure: A network stream and the corresponding frequency vector

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates
  - The vector is too large to be stored explicitly and updated

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates
  - The vector is too large to be stored explicitly and updated

Example

The number of IP addresses - which is the size of the frequency vector in the network monitoring examples is 2^{32} (this will become 2^{128} with the introduction of IPv6).

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates
  - The vector is too large to be stored explicitly and updated

Example

The number of IP addresses - which is the size of the frequency vector in the network monitoring examples is 2^{32} (this will become 2^{128} with the introduction of IPv6).

- The network stream is too large to be stored as well.

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates
  - The vector is too large to be stored explicitly and updated

Example

The number of IP addresses - which is the size of the frequency vector in the network monitoring examples is 2^{32} (this will become 2^{128} with the introduction of IPv6).

- The network stream is too large to be stored as well.
- This means random access into the input is not possible.

Working with Streaming Data

- The most obvious solution of working directly with the frequency vector does not work because
  - The frequency vector is only available as a series of updates
  - The vector is too large to be stored explicitly and updated

Example

The number of IP addresses - which is the size of the frequency vector in the network monitoring examples is 2^{32} (this will become 2^{128} with the introduction of IPv6).

- The network stream is too large to be stored as well.
- This means random access into the input is not possible.
- Hence Assumption 1 and 2 do not hold ! Assumption 3 does not hold either.

Beating this curse ...

The Main Idea

- Working with the actual input is not possible because of high dimensionality.

The Main Idea

- Working with the actual input is not possible because of high dimensionality.
- Why not try getting a lower dimensional "compressed" version of the input ?

The Main Idea

- Working with the actual input is not possible because of high dimensionality.
- Why not try getting a lower dimensional "compressed" version of the input ?
- This version should retain all the *interesting* properties of the original input.

The Main Idea

- Working with the actual input is not possible because of high dimensionality.
- Why not try getting a lower dimensional "compressed" version of the input ?
- This version should retain all the *interesting* properties of the original input.
- Note : This usually means working with approximately correct answers.

The Main Idea

- Working with the actual input is not possible because of high dimensionality.
- Why not try getting a lower dimensional "compressed" version of the input ?
- This version should retain all the *interesting* properties of the original input.
- Note : This usually means working with approximately correct answers.
- If the correct answer is L then our algorithms return a value $\hat{L}$ such that $|L - \hat{L}| < \epsilon L$ for small $\epsilon > 0$.

The Random Projection Method

- Think of it as a very high dimensional camera

The Random Projection Method

- Think of it as a very high dimensional camera
- Key observation : most of our photographs (unless taken from a very weird angle) resemble us

The Random Projection Method

- Think of it as a very high dimensional camera
- Key observation : most of our photographs (unless taken from a very weird angle) resemble us
- Hence a random photograph preserves all the features of our faces approximately

The Random Projection Method

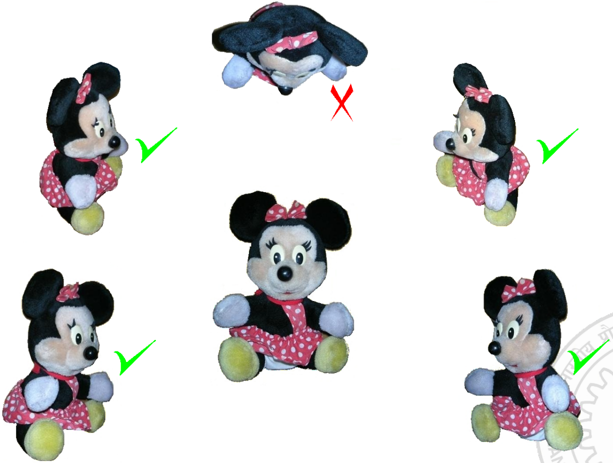

Figure: A Random Photograph is good enough !

The Random Projection Method

- Apply this to high dimensional point sets - a random low-dimensional *photograph* should approximately preserve most interesting properties of any point set.

The Random Projection Method

- Apply this to high dimensional point sets - a random low-dimensional *photograph* should approximately preserve most interesting properties of any point set.

Example

In a database where every entry is a 1000-dimensional vector, if I randomly map every entry to a 150-dimensional vector then

The Random Projection Method

- Apply this to high dimensional point sets - a random low-dimensional *photograph* should approximately preserve most interesting properties of any point set.

Example

In a database where every entry is a 1000-dimensional vector, if I randomly map every entry to a 150-dimensional vector then

- I should not lose too many interesting properties of the database

The Random Projection Method

- Apply this to high dimensional point sets - a random low-dimensional *photograph* should approximately preserve most interesting properties of any point set.

Example

In a database where every entry is a 1000-dimensional vector, if I randomly map every entry to a 150-dimensional vector then

- I should not lose too many interesting properties of the database
- i.e. my query routines should return almost the same answers

The Random Projection Method

- Apply this to high dimensional point sets - a random low-dimensional *photograph* should approximately preserve most interesting properties of any point set.

Example

In a database where every entry is a 1000-dimensional vector, if I randomly map every entry to a 150-dimensional vector then

- I should not lose too many interesting properties of the database
- i.e. my query routines should return almost the same answers
- only that the routines would be faster since dimensionality has reduced

The Random Projection Method

- Several key results in the fields of dimensionality reduction and data streaming look at various classes of interesting properties and demonstrate how can random projections preserve them.

The Random Projection Method

- Several key results in the fields of dimensionality reduction and data streaming look at various classes of interesting properties and demonstrate how can random projections preserve them.
- The most celebrated of these results is the Johnson-Lindenstrauss Lemma that deals with data sets in Euclidean spaces - the interesting property of a set of points in this result is the pairwise inter-point Euclidean distances.

The Random Projection Method

- Several key results in the fields of dimensionality reduction and data streaming look at various classes of interesting properties and demonstrate how can random projections preserve them.
- The most celebrated of these results is the Johnson-Lindenstrauss Lemma that deals with data sets in Euclidean spaces - the interesting property of a set of points in this result is the pairwise inter-point Euclidean distances.

Definition (Low-distortion embeddings)

Given two metric spaces (X, ρ) and (Y, σ), a mapping $f : X \longrightarrow Y$ is called a D-embedding where $D \geq 1$, if there exists a number $r > 0$ such that for all $x, y \in X$,

$$r \cdot \rho(x, y) \leq \sigma(f(x), f(y)) \leq D \cdot r \cdot \rho(x, y)$$

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset \left(\mathbb{R}^d, \ell_2\right)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1+\epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset \left(\mathbb{R}^d, \ell_2\right)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1+\epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

- How to implement a random mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$?

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset \left(\mathbb{R}^d, \ell_2\right)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1+\epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

- How to implement a random mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$?
- A simple linear mapping - multiply each vector in $\mathbb{R}^d$ with a random $k \times d$ matrix P

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset \left(\mathbb{R}^d, \ell_2\right)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1 + \epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

- How to implement a random mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$?
- A simple linear mapping - multiply each vector in $\mathbb{R}^d$ with a random $k \times d$ matrix P
- The d-dimensional vector x is mapped to the k-dimensional vector Px.

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset (\mathbb{R}^d, \ell_2)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1+\epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

- How to implement a random mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$?
- A simple linear mapping - multiply each vector in $\mathbb{R}^d$ with a random $k \times d$ matrix P
- The d-dimensional vector x is mapped to the k-dimensional vector Px.
- It suffices to choose each entry from the set $\{-1, 1\}$ randomly.

The Johnson Lindenstrauss Lemma

Theorem ([Johnson and Lindenstrauss1984])

Let X be an n-point set in a d-dimensional Euclidean space (i.e. $(X, \ell_2) \subset (\mathbb{R}^d, \ell_2)$), and let $\epsilon \in (0, 1]$ be given. Then there exists a $(1+\epsilon)$-embedding of X into $(\mathbb{R}^k, \ell_2)$ where $k = \mathcal{O}\left(\epsilon^{-2} \log n\right)$. Furthermore, this embedding can be found out in randomized polynomial time.

- How to implement a random mapping from $\mathbb{R}^d$ to $\mathbb{R}^k$?
- A simple linear mapping - multiply each vector in $\mathbb{R}^d$ with a random $k \times d$ matrix P
- The d-dimensional vector x is mapped to the k-dimensional vector Px.
- It suffices to choose each entry from the set $\{-1, 1\}$ randomly.
- Linear mappings have other benefits - more on this later

Random Projections

- Why cant I just choose k of the d dimensions and get a deterministic projection ?

Random Projections

- Why cant I just choose k of the d dimensions and get a deterministic projection ?
- Alignment problems - most of the interesting information in the vector may lie in the dimensions we have chosen to throw away

Random Projections

- Why cant I just choose k of the d dimensions and get a deterministic projection ?
- Alignment problems - most of the interesting information in the vector may lie in the dimensions we have chosen to throw away

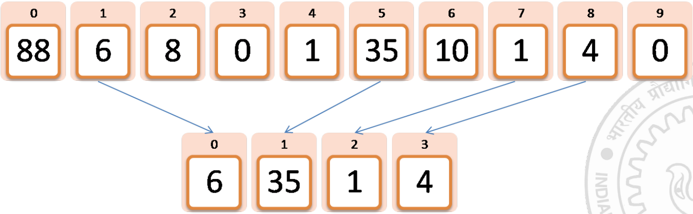

Figure: Shortcomings of a deterministic projection

Random Projections

- Why cant I just choose k of the d dimensions and get a deterministic projection ?
- Alignment problems - most of the interesting information in the vector may lie in the dimensions we have chosen to throw away
- A random matrix undoes any such alignments - referred to as incoherence in Compressed Sensing literature

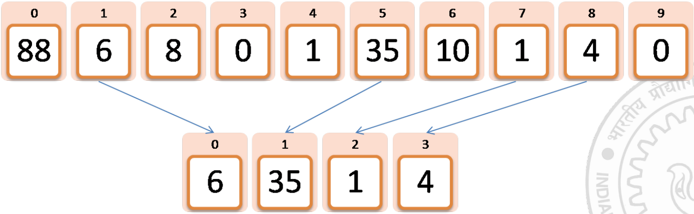

Figure: Shortcomings of a deterministic projection

Random Projection at work ...

Other notions of interesting properties

- What if the interesting properties of a data set are the inter-point distance for some distance measure other than the Euclidean ?

Other notions of interesting properties

- What if the interesting properties of a data set are the inter-point distance for some distance measure other than the Euclidean ?
- For example statistical distance measures (Mahalanobis, Kullback-Leibler, Bhattacharyya) that are useful in image retrieval, bio-informatics etc.

Other notions of interesting properties

- What if the interesting properties of a data set are the inter-point distance for some distance measure other than the Euclidean ?
- For example statistical distance measures (Mahalanobis, Kullback-Leibler, Bhattacharyya) that are useful in image retrieval, bio-informatics etc.
- Two possible ways of handling high-dimensional databases that use these measures
  - Find ways to project (randomly) to lower dimensions directly so that inter-point distances are preserved.
  - Embed these distances in Euclidean spaces and then use Johnson-Lindenstrauss Lemma to reduce dimensionality.

Some Positive Results

Definition (Bhattacharyya Distance)

For two vectors $P = (p_1, p_2, \ldots, p_d)$ and $Q = (q_1, q_2, \ldots q_d)$ with $\sum_{i=1}^{d} p_i = \sum_{i=1}^{n} q_i = 1$ and each $p_i, q_i \geq 0$, the *Bhattacharyya distance* between them is defined to be
$BD(P, Q) = -\ln\left(\sum_{i=1}^{n} \sqrt{p_i q_i}\right)$.

Some Positive Results

Definition (Bhattacharyya Distance)

For two vectors $P = (p_1, p_2, \ldots, p_d)$ and $Q = (q_1, q_2, \ldots q_d)$ with $\sum_{i=1}^{d} p_i = \sum_{i=1}^{n} q_i = 1$ and each $p_i, q_i \geq 0$, the *Bhattacharyya distance* between them is defined to be
$BD(P, Q) = -\ln\left(\sum_{i=1}^{n} \sqrt{p_i q_i}\right)$.

Definition (Mahalanobis Distance Measure)

A $d \times d$ positive definite matrix A defines a Mahalanobis Distance measure over $\mathbb{R}^d$ given by $M_A(x, y) = \sqrt{(x-y)^T A(x-y)}$.

Some Positive Results

Definition (Bhattacharyya Distance)

For two vectors $P = (p_1, p_2, \ldots, p_d)$ and $Q = (q_1, q_2, \ldots q_d)$ with $\sum_{i=1}^{d} p_i = \sum_{i=1}^{n} q_i = 1$ and each $p_i, q_i \geq 0$, the *Bhattacharyya distance* between them is defined to be
$BD(P, Q) = -\ln\left(\sum_{i=1}^{n} \sqrt{p_i q_i}\right)$.

Definition (Mahalanobis Distance Measure)

A $d \times d$ positive definite matrix A defines a Mahalanobis Distance measure over $\mathbb{R}^d$ given by $M_A(x, y) = \sqrt{(x-y)^T A(x-y)}$.

Theorem ([Bhattacharya et al.2009])

One can project data sets using the Bhattacharyya and the Mahalanobis distance measures to low dimensional spaces using linear random projections.

A (partial) Negative Result

Definition (Kullback Leibler Divergence)

Given two vectors $P = \{p_1, p_2, \ldots, p_d\}$ and $Q = \{q_2, q_2 \ldots q_d\}$, the Kullback-Leibler divergence between the two vectors is defined as $KL(P, Q) = \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i}$.

A (partial) Negative Result

Definition (Kullback Leibler Divergence)

Given two vectors $P = \{p_1, p_2, \ldots, p_d\}$ and $Q = \{q_2, q_2 \ldots q_d\}$, the Kullback-Leibler divergence between the two vectors is defined as
$KL(P, Q) = \sum_{i=1}^{d} p_i \ln \frac{p_i}{q_i}$.

Theorem ([Bhattacharya et al.2009])

Point sets using the Kullback-Leibler divergence cannot be embedded into any metric space (in particular the Euclidean space) without distorting the inter-point distances by large amounts.

Processing Massive Data Streams using Random Projections ...

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.
- Useful in database query optimization and network traffic anomaly detection.

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.
- Useful in database query optimization and network traffic anomaly detection.
- The L_p norm of a vector $(f_1, f_2, \ldots, f_d)$ for $p > 0$ is $\sum_{i=1}^{d} f_i^p$.

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.
- Useful in database query optimization and network traffic anomaly detection.
- The L_p norm of a vector $(f_1, f_2, \ldots, f_d)$ for $p > 0$ is $\sum_{i=1}^{d} f_i^p$.
- Note that the L_2 norm of a vector is just the squared Euclidean length of the vector.

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.
- Useful in database query optimization and network traffic anomaly detection.
- The L_p norm of a vector $(f_1, f_2, \ldots, f_d)$ for $p > 0$ is $\sum_{i=1}^{d} f_i^p$.
- Note that the L_2 norm of a vector is just the squared Euclidean length of the vector.
- In a series of seminal papers [Alon et al.1999, Indyk2006], the random projection technique was extended to the problem of estimating F_p for $0 < p \leq 2$.

Norm Estimation in Streams

- A very important problem in Data Streaming is estimating the L_2 norm of the frequency vector.
- Useful in database query optimization and network traffic anomaly detection.
- The L_p norm of a vector $(f_1, f_2, \ldots, f_d)$ for $p > 0$ is $\sum_{i=1}^{d} f_i^p$.
- Note that the L_2 norm of a vector is just the squared Euclidean length of the vector.
- In a series of seminal papers [Alon et al.1999, Indyk2006], the random projection technique was extended to the problem of estimating F_p for $0 < p \leq 2$.
- Using random projections that are linear mappings is crucial here since the frequency vector is never available to us.

L_2-estimation in Streams

- The trick is to identify j^{th} stream element say 5 with a frequency vector update s_j

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0

L_2-estimation in Streams

- The trick is to identify j^{th} stream element say 5 with a frequency vector update s_j

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0

- The frequency vector $f = \sum_{j=1}^{m} s_j$ where m is the length of the stream. Thus, for any linear mapping P, $Pf = \sum_{j=1}^{m} Ps_j$.

L_2-estimation in Streams

- The trick is to identify j^{th} stream element say 5 with a frequency vector update s_j

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0

- The frequency vector $f = \sum_{j=1}^{m} s_j$ where m is the length of the stream. Thus, for any linear mapping P, $Pf = \sum_{j=1}^{m} Ps_j$.
- This allows us to incrementally update low-dimensional sketches.

L_2-estimation in Streams

- The trick is to identify j^{th} stream element say 5 with a frequency vector update s_j

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0

- The frequency vector $f = \sum_{j=1}^{m} s_j$ where m is the length of the stream. Thus, for any linear mapping P, $Pf = \sum_{j=1}^{m} Ps_j$.
- This allows us to incrementally update low-dimensional sketches.
- Construct P by choosing every element randomly from $\{-1, 1\}$.

L_2-estimation in Streams

- The trick is to identify j^{th} stream element say 5 with a frequency vector update s_j

0	1	2	3	4	5	6	7	8	9
0	0	0	0	0	1	0	0	0	0

- The frequency vector $f = \sum_{j=1}^{m} s_j$ where m is the length of the stream. Thus, for any linear mapping P, $Pf = \sum_{j=1}^{m} Ps_j$.
- This allows us to incrementally update low-dimensional sketches.
- Construct P by choosing every element randomly from $\{-1, 1\}$.
- [Alon et al.1999] Reducing a d-dimensional frequency vector to $k = \mathcal{O}\left(\frac{\log d}{\epsilon^2}\right)$ dimensions does not change the L_2 norm by more than an ϵ fraction.

Processing an Update

$$
\begin{array}{c}
\begin{array}{ccccccc} \phantom{p_{11}} & \phantom{p_{12}} & \phantom{p_{13}} & & \downarrow & & \phantom{p_{1d}} \end{array} \\
\begin{bmatrix}
p_{11} & p_{12} & p_{13} & \cdots & p_{1j} & \cdots & p_{1d} \\
p_{21} & p_{22} & p_{23} & \cdots & p_{2j} & \cdots & p_{2d} \\
p_{31} & p_{32} & p_{33} & \cdots & p_{3j} & \cdots & p_{3d} \\
\vdots & \vdots & \vdots & \ddots & \vdots & \ddots & \vdots \\
p_{k1} & p_{k2} & p_{k3} & \cdots & p_{kj} & \cdots & p_{kd}
\end{bmatrix}
\end{array}
\begin{bmatrix} 0 \\ 0 \\ 0 \\ \vdots \\ 1 \\ \vdots \\ 0 \end{bmatrix}
\begin{array}{l} \\ \\ \\ \\ \leftarrow j^{th} \text{ element} \\ \\ \\ \end{array}
= \begin{bmatrix} p_{1j} \\ p_{2j} \\ p_{3j} \\ \vdots \\ p_{kj} \end{bmatrix}
$$

$$
\text{Sketch}_{\text{new}} = \text{Sketch}_{\text{old}} + \begin{bmatrix} p_{1j} \\ p_{2j} \\ p_{3j} \\ \vdots \\ p_{kj} \end{bmatrix}
$$

A small problem ...

A small problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.

A small problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.
- We agreed that the d-dimensional vector f is too big to be stored.

A BIG problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.
- We agreed that the d-dimensional vector f is too big to be stored.
- ??

A BIG problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.
- We agreed that the d-dimensional vector f is too big to be stored.
- ??
- The key is to use tools from Computational Complexity called Pseudo-random Generators [Nisan1992].

A BIG problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.
- We agreed that the d-dimensional vector f is too big to be stored.
- ??
- The key is to use tools from Computational Complexity called Pseudo-random Generators [Nisan1992].
- These allow us to generate parts of the matrix as and when needed and do not require us to store the entire matrix explicitly

A BIG problem ...

- How do we store the projection matrix P ?? Recall that P has dimensions $k \times d$.
- We agreed that the d-dimensional vector f is too big to be stored.
- ??
- The key is to use tools from Computational Complexity called Pseudo-random Generators [Nisan1992].
- These allow us to generate parts of the matrix as and when needed and do not require us to store the entire matrix explicitly
- Thus, in order to get and ϵ-approximation to the L_2 norm of a data stream frequency vector, we need only $\tilde{\mathcal{O}}\left(\frac{1}{\epsilon^2}\log d\right)$ space.

More Applications in Data Streams

- By constructing the matrix P differently we can estimate L_p for any $0 < p \leq 2$.

More Applications in Data Streams

- By constructing the matrix P differently we can estimate L_p for any $0 < p \leq 2$.
- Other random projection techniques allow us to maintain short sketches of the frequency vector that allow us to
  - estimate the number of non-zero coordinates in the frequency vector (F_0 estimation)
  - return the coordinates that have the highest values (Heavy Hitter estimation)
  - ...

Some other techniques ...

Locality Sensitive Hashing

- Introduced by [Indyk and Motwani1998] as a solution to the approximate Nearest Neighbor Problem in high dimensions.

Locality Sensitive Hashing

- Introduced by [Indyk and Motwani1998] as a solution to the approximate Nearest Neighbor Problem in high dimensions.
- The idea is to come up with a family of hash functions such that nearby points attain the same value under the hash functions and far away points hash to different values.

Locality Sensitive Hashing

- Introduced by [Indyk and Motwani1998] as a solution to the approximate Nearest Neighbor Problem in high dimensions.
- The idea is to come up with a family of hash functions such that nearby points attain the same value under the hash functions and far away points hash to different values.
- More formally, a *Locality Sensitive Hash Family* for a distance measure d on a set X is a set of functions $\mathcal{H}$ that map points in X to some small universe U from such that for any two points $x, y \in X$,
  - if $d(x, y) < r$, then at least 90% of the hash functions in the hash family hash them to the same value i.e. $h(x) = h(y)$.
  - if $d(x, y) > R$, then at least 90% of the hash functions in the hash family hash them to different values i.e. $h(x) \neq h(y)$.

Locality Sensitive Hashing

- Introduced by [Indyk and Motwani1998] as a solution to the approximate Nearest Neighbor Problem in high dimensions.
- The idea is to come up with a family of hash functions such that nearby points attain the same value under the hash functions and far away points hash to different values.
- More formally, a *Locality Sensitive Hash Family* for a distance measure d on a set X is a set of functions $\mathcal{H}$ that map points in X to some small universe U from such that for any two points $x, y \in X$,
  - if $d(x, y) < r$, then at least 90% of the hash functions in the hash family hash them to the same value i.e. $h(x) = h(y)$.
  - if $d(x, y) > R$, then at least 90% of the hash functions in the hash family hash them to different values i.e. $h(x) \neq h(y)$.
- We now know efficient constructions of such hash families. See [Andoni and Indyk2008] for a survey.

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself
- Often one applies compression techniques to raw data that is very high dimensional (eg. JPEG-2000 compression to raw images)

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself
- Often one applies compression techniques to raw data that is very high dimensional (eg. JPEG-2000 compression to raw images)
- Thus the high-resolution acquisition is wasteful since the compressed data usually has a very sparse representation (eg. JPEG images have very sparse Fourier representations)

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself
- Often one applies compression techniques to raw data that is very high dimensional (eg. JPEG-2000 compression to raw images)
- Thus the high-resolution acquisition is wasteful since the compressed data usually has a very sparse representation (eg. JPEG images have very sparse Fourier representations)
- Compressed Sensing seeks to address this problem by acquiring the sparse representations directly and using algorithms for data reconstruction from these sparse representations.

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself
- Often one applies compression techniques to raw data that is very high dimensional (eg. JPEG-2000 compression to raw images)
- Thus the high-resolution acquisition is wasteful since the compressed data usually has a very sparse representation (eg. JPEG images have very sparse Fourier representations)
- Compressed Sensing seeks to address this problem by acquiring the sparse representations directly and using algorithms for data reconstruction from these sparse representations.
- The method was introduced in two seminal papers by Candes-Romberg-Tao and Donoho [Candes]

Compressed Sensing

- Addresses the problem of dimensionality at the data acquisition phase itself
- Often one applies compression techniques to raw data that is very high dimensional (eg. JPEG-2000 compression to raw images)
- Thus the high-resolution acquisition is wasteful since the compressed data usually has a very sparse representation (eg. JPEG images have very sparse Fourier representations)
- Compressed Sensing seeks to address this problem by acquiring the sparse representations directly and using algorithms for data reconstruction from these sparse representations.
- The method was introduced in two seminal papers by Candes-Romberg-Tao and Donoho [Candes]
- Has led to the development of compressed sensing hardware - eg. Single pixel camera

Single Pixel Camera (Rice University)

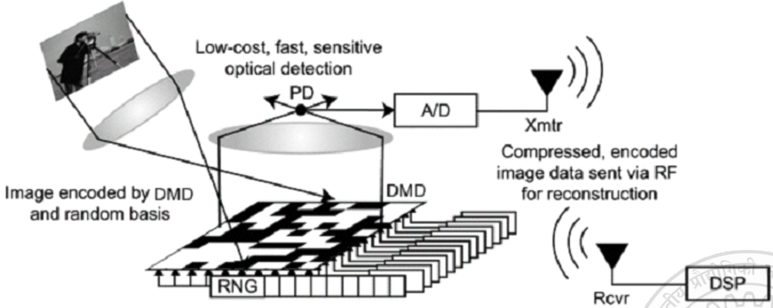

Figure: Compressed Sensing in Practice

Manifold Identification Techniques

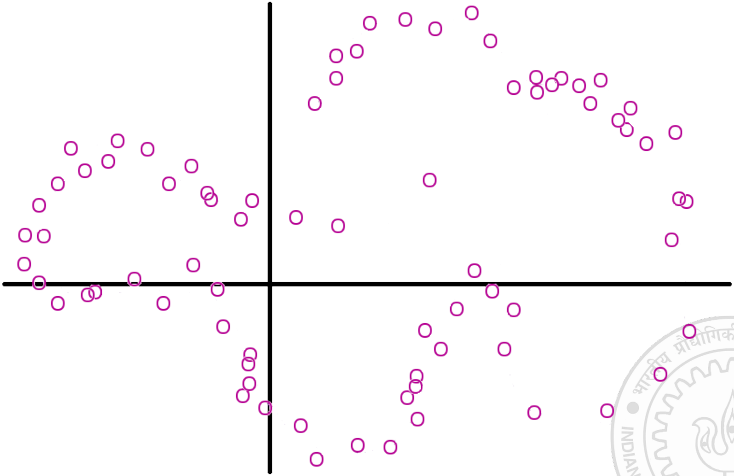

Figure: A 2-dimensional dataset

Manifold Identification Techniques

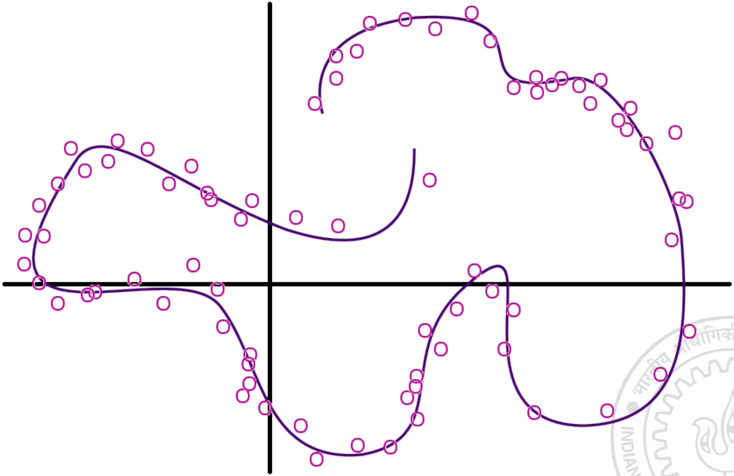

Figure: The dataset is intrinsically 1-dimensional

Manifold Identification Techniques

- Applies to situations when the high dimensional data has support only on some low dimensional surface (manifold) i.e. the data is intrinsically low-dimensional but is embedded in a high dimensional space.

Manifold Identification Techniques

- Applies to situations when the high dimensional data has support only on some low dimensional surface (manifold) i.e. the data is intrinsically low-dimensional but is embedded in a high dimensional space.
- For example : datasets of images of handwritten numerals - the images may by high dimensional (number of pixels) but the intrinsic dimensionality is low - eg. there are only a few degrees of freedom for images of the numeral zero which is essentially an ellipse

Manifold Identification Techniques

- Applies to situations when the high dimensional data has support only on some low dimensional surface (manifold) i.e. the data is intrinsically low-dimensional but is embedded in a high dimensional space.
- For example : datasets of images of handwritten numerals - the images may by high dimensional (number of pixels) but the intrinsic dimensionality is low - eg. there are only a few degrees of freedom for images of the numeral zero which is essentially an ellipse
- Various approaches used to exploit the low intrinsic dimensionality

Manifold Identification Techniques

- Applies to situations when the high dimensional data has support only on some low dimensional surface (manifold) i.e. the data is intrinsically low-dimensional but is embedded in a high dimensional space.
- For example : datasets of images of handwritten numerals - the images may by high dimensional (number of pixels) but the intrinsic dimensionality is low - eg. there are only a few degrees of freedom for images of the numeral zero which is essentially an ellipse
- Various approaches used to exploit the low intrinsic dimensionality
  - Discover embeddings of the data into spaces of dimensionality equal to the intrinsic dimensionality - ISOMAP algorithm [Tenenbaum et al.2009, Clarkson2008]

Manifold Identification Techniques

- Applies to situations when the high dimensional data has support only on some low dimensional surface (manifold) i.e. the data is intrinsically low-dimensional but is embedded in a high dimensional space.
- For example : datasets of images of handwritten numerals - the images may by high dimensional (number of pixels) but the intrinsic dimensionality is low - eg. there are only a few degrees of freedom for images of the numeral zero which is essentially an ellipse
- Various approaches used to exploit the low intrinsic dimensionality
  - Discover embeddings of the data into spaces of dimensionality equal to the intrinsic dimensionality - ISOMAP algorithm [Tenenbaum et al.2009, Clarkson2008]
  - Use the low intrinsic dimensionality implicitly to speed up data structures like k-d Trees [Dasgupta and Freund2008].

Concluding Remarks

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality
- The method can be adapted to various algorithms -

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality
- The method can be adapted to various algorithms -
  - Dimensionality Reduction, Sketching and Compressed Sensing

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality
- The method can be adapted to various algorithms -
    - Dimensionality Reduction, Sketching and Compressed Sensing
    - Locality Sensitive Hashing

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality
- The method can be adapted to various algorithms -
  - Dimensionality Reduction, Sketching and Compressed Sensing
  - Locality Sensitive Hashing
  - Manifold Identification techniques

Summarizing

- Improvements in our capability to acquire high-dimensional, continuous and massive data have posed the challenges of processing/storing this data in a resource efficient manner.
- High dimensional data poses peculiar challenges to algorithms in terms of space utilization and processing time (Curse of Dimensionality)
- The Random Projection Method gives us an elegant technique to overcome the curse of dimensionality
- The method can be adapted to various algorithms -
  - Dimensionality Reduction, Sketching and Compressed Sensing
  - Locality Sensitive Hashing
  - Manifold Identification techniques
- The areas of sketching, dimensionality reduction and manifold identification techniques continue to pose challenges and require a deeper understanding of the Random Projection Method.

References (1)

Alon, N., Matias, Y., & Szegedy, M. (1999).
The Space Complexity of Approximating the Frequency Moments.
Journal of Computer Systems and Sciences, 58(1):137–147.

Andoni, A. & Indyk, P. (2008).
Near-Optimal Hashing Algorithms for Approximate Nearest Neighbor in High Dimensions.
Communications of the ACM, 51(1):117–122.

Bhattacharya, A.
CS618 : Indexing and Searching Techniques in Databases.
Course Notes for the Fall'09 offering at Dept of CSE, IIT Kanpur.

References (2)

Bhattacharya, A., Kar, P., & Pal, M. (2009).
On Low Distortion Embeddings of Statistical Distance Measures into Low Dimensional Spaces.
In: *20th International Conference on Database and Expert Systems Applications (DEXA)*, pages 164–172.

Candes, E.
Compressive Sensing.
Tutorial given at Neural Information Processing Systems Conference, 2008.

Clarkson, K. L. (2008).
Tighter Bounds for Random Projections of Manifolds.
In: *ACM Symposium on Computational Geometry*.

References (3)

Dasgupta, S. & Freund, Y. (2008).
Random Projection Trees and Low Dimensional Manifolds.
In: *40th Annual ACM Symposium on Theory of Computing*, pages 537–546.

Ganguly, S.
CS719 : Data Stream Algorithms.
Course Notes for the Spring'10 offering at Dept of CSE, IIT Kanpur.

Indyk, P. (2006).
Stable Distributions, Pseudorandom Generators, Embeddings and Data Stream Computations.
Journal of the ACM, 53(3):307–323.

References (4)

Indyk, P. & Motwani, R. (1998).
Approximate Nearest Neighbors : Towards Removing the Curse of Dimensionality.
In: *30th Annual ACM Symposium on Theory of Computing*, pages 604–613.

Johnson, W. B. & Lindenstrauss, J. (1984).
Extensions of Lipschitz maps into a Hilbert Space.
Contemporary Mathematics, 26:189–206.

Nisan, N. (1992).
Pseudorandom Generators for Space Bounded Computations.
Combinatorica, 12(4):449–461.

References (5)

Tenenbaum, J. B., de Silva, V., & Langford, J. C. (2009).
A Global Geometric Framework for Nonlinear Dimensionality Reduction.
Science.