Streaming Algorithms

Objectives

- Seminar
- Streaming Algorithms for Pre-processing
  - Recap
    - Descriptive Statistics, Sampling, Sketches
  - Quantization
  - Transforms
  - Dimensionality Reduction

Principles

- Avoid multi-pass algorithms
  - Data available streaming
- Need efficiency in space-compute-complexity
  - Cannot keep large amounts in memory
- Approximate answers are acceptable
  - Bounds on approximation error desirable

Principles

- Support distributed implementations on SPS
  - E.g. split using data parallelism
  - Stateful versus stateless processing
- Tradeoff data versus process time
  - Understand interaction with sliding/tumbling windows

Principles: Approximation Error

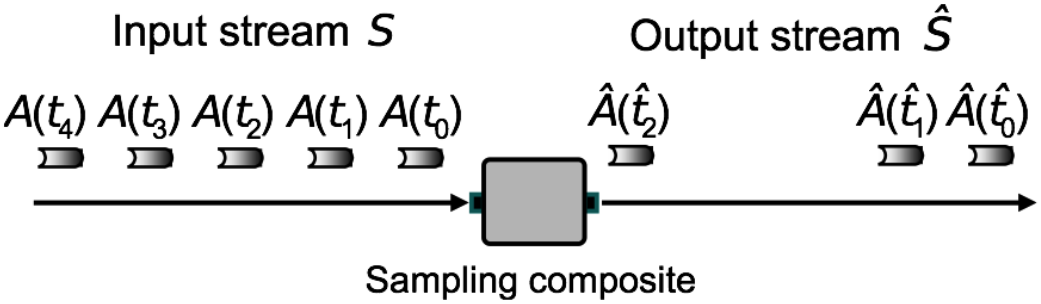
- Depend on the nature of the aggregation
  - Counts/Histogram
  - Additive Aggregations
  - Multi-input Data
- Bounds on Error
  - No Bounds
  - Deterministic Bounds
  - Probabilistic Bounds

BasicCounting Algorithm

- Question: Assume you have a stream that contains tuples that are either 0s or 1s, how can you maintain the number of 1s in the last W tuples
  - Sliding window of size W, slide 1
- Assume W is large
  - So $O(W)$ is too large. We want to store logarithmic space
- The *BasicCounting* algorithm
  - Uses space of the order $O\left(\frac{1}{\varepsilon}\log^2 W\right)$
  - With error tolerance ε, i.e. the counts are within $1\pm\varepsilon$ of its actual value. ε is specified by application objective
- Study: Data structure, update, query

Sampling

- Sampling is used to reduce the data stream by selecting some tuples based on one or more criteria



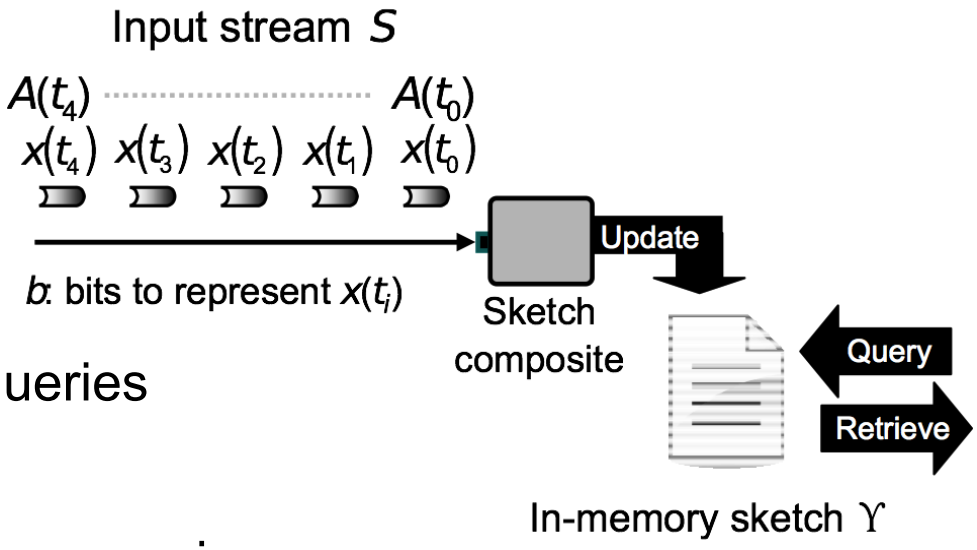
- Three types of sampling
  - Systematic or Uniform
  - Data Driven
  - Random

Summary

Techniques	Tuple Types	Windowing
Moments	Numeric Tuple, Univariate or Multivariate	Sliding or tumbling window
Counts and Histograms (BasicCounting)	Numeric or non-numeric data	Sliding or tumbling window
Systematic Sampling	Numeric or non-numeric data	Not windowed. Can be tumbling window
Data-driven Sampling	Numeric or non-numeric data (Interpolation driven is for numeric data)	Not windowed. Can be tumbling window
Random Sampling	Numeric or non-numeric data	Not windowed. Can be tumbling window

Sketches

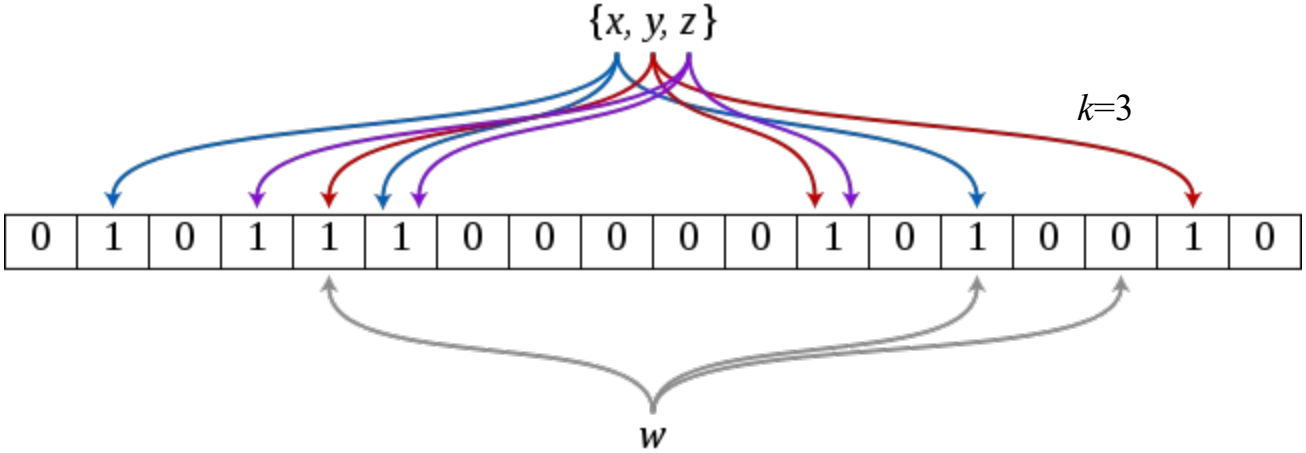
- In-memory data structures that contain compact, often lossy, synopses of streaming data
- They capture the key properties of the stream



- In order to answer specific queries
  - Error bounds and
  - probabilistic guarantees
- Used for approximate query processing

Bloom Filters

- Let us say we want to answer containment queries
- Whether a given item has been see so far or not
- Bloom filter
  - Keep a bit array of size m
  - Use k pairwise independent hash functions (more later)
  - Update: Hash item to k locations and set the bits to one
  - Query: Hash item to k locations
    - Return true if all of the k locations are set



Bloom Filters

- False negatives are not possible
  - If we say an item has not been seen, then it is not seen
- False positives are possible
  - If we say an item has been seen, we may be wrong
  - We want to set m, k to reach a desired possibility
- How to create pairwise independent hash functions?
  - In general difficult (we will discuss an example later)
  - Take a hash function with large range and slice into k parts
  - If a hash function takes a seed, give it k different seeds

Generating Independent Hash Functions

- Pick a prime p in the range $m < p < 2m$
  - such a prime exists due to Bertrand-Chebyshev theorem
- For each j in $[0..k)$
  - Pick two random numbers $a_j > 0$ and b_j in range $[0..p)$
  - Set

$$h_j : x \rightarrow ((a_j x + b_j) \bmod p) \bmod m)$$

$$Pr\,[(h_j(x_1) = y_1) \wedge (h_i(x_2) = y_2) = 1/m^2$$

Bloom Filter Analysis

- Probability of a bit not being set to 1 by a hash function (assuming hash function is uniform)

$$1-\frac{1}{m}$$

- Probability of a bit not being set by k independent hash functions

$$\left(1-\frac{1}{m}\right)^{k}$$

- Probability of a bit not being set after observing n values

$$\left(1-\frac{1}{m}\right)^{kn}$$

- Hence probability of a bit being 1 after n values

$$1-\left(1-\frac{1}{m}\right)^{kn}$$

Bloom Filter Analysis

- Now, when we test for new value probability that all k bits are set to 1 is

$$\left[1-\left(1-\frac{1}{m}\right)^{kn}\right]^k$$

- This false positive probability may be approximated as

$$\left(1-e^{-kn/m}\right)^k$$

- Optimal number of hash functions for a given m and n is

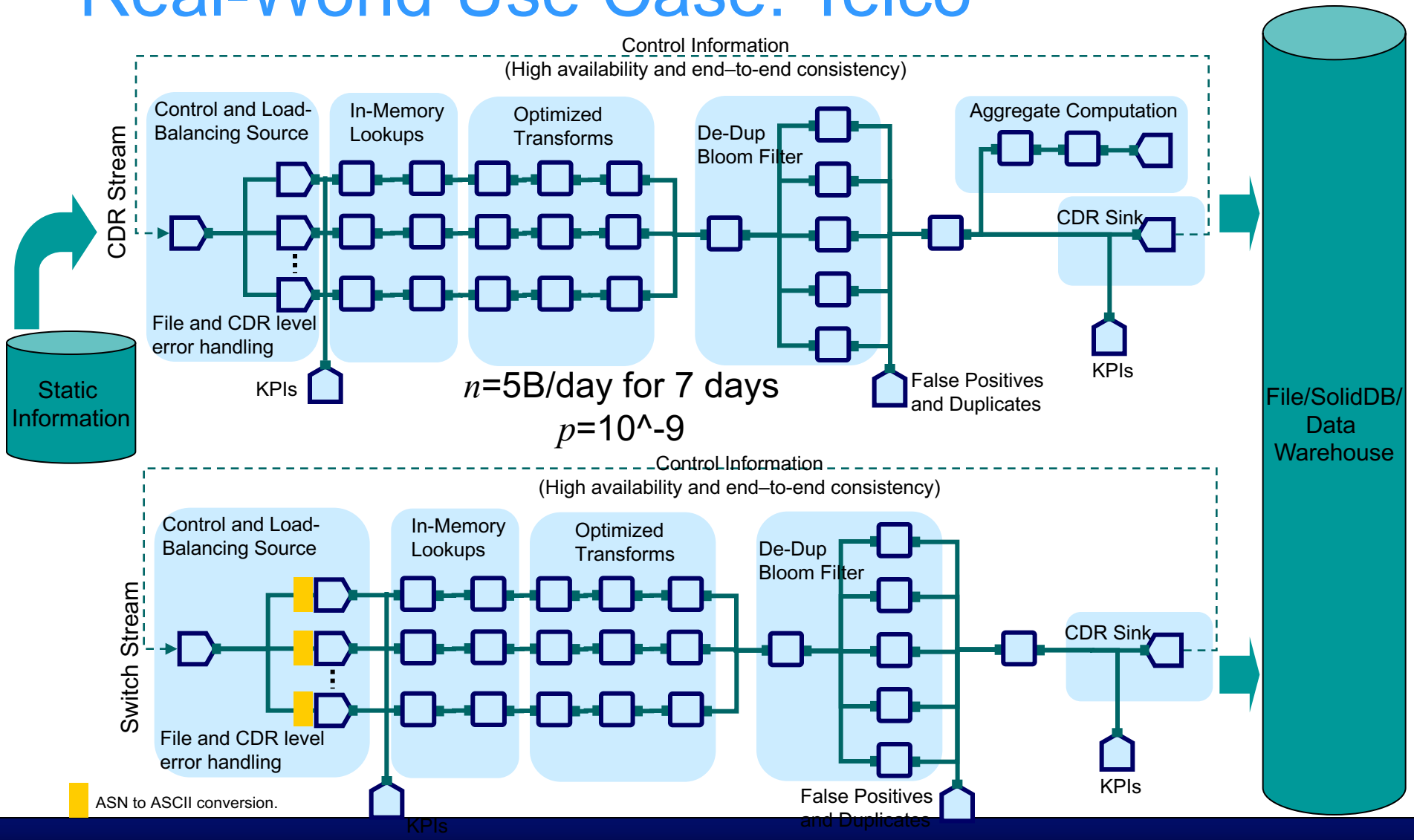
$$k=\frac{m}{n}\log 2$$

- Hence, given a n a desired false alarm rate p, we have

$$m=\frac{-n\ln p}{(\ln 2)^2}$$

Bloom filter calculator: http://hur.st/bloomfilter?n=4&p=1.0E-20

Real-World Use Case: Telco

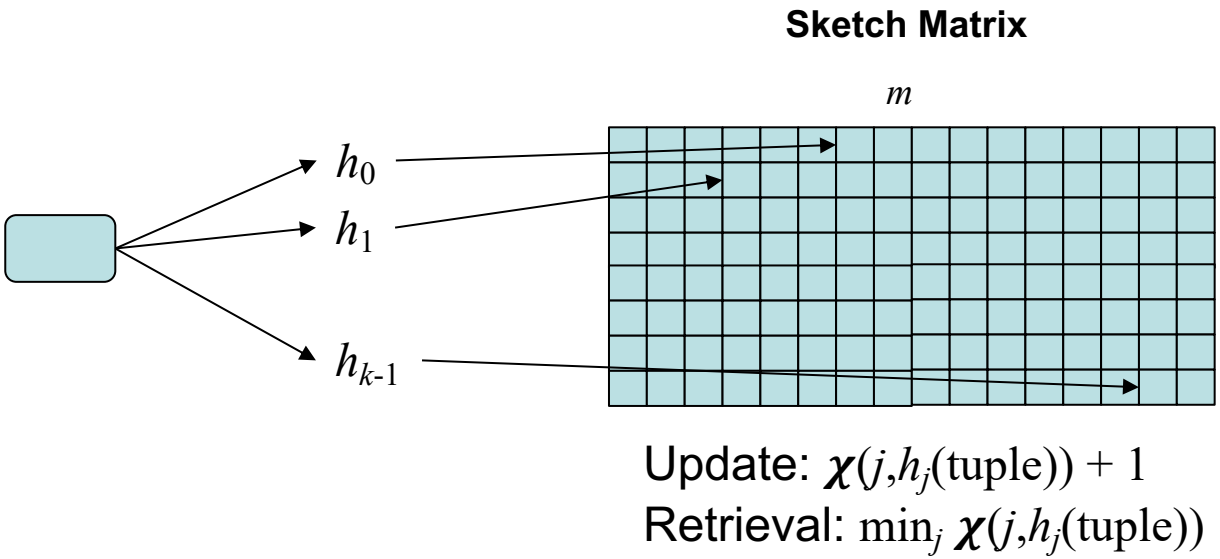


Count-Min Sketch

- Assume we have a numeric attribute with a very large domain
  - E.g. the domain of ip v6 addresses
- Assume we want to find, given a value, its frequency
- Let us use D to denote the domain size
- The Count-Min sketch keeps k rows and m columns
- Let H be a set of k pair-wise independent hash functions with range $[0..m)$

Count-Min Sketch: Construction and Retrieval

- Construction: Apply the k hash functions
  - Increment the cell in each row whose index is the hash value
- Retrieval: Apply the k hash functions
  - Pick the smallest count value

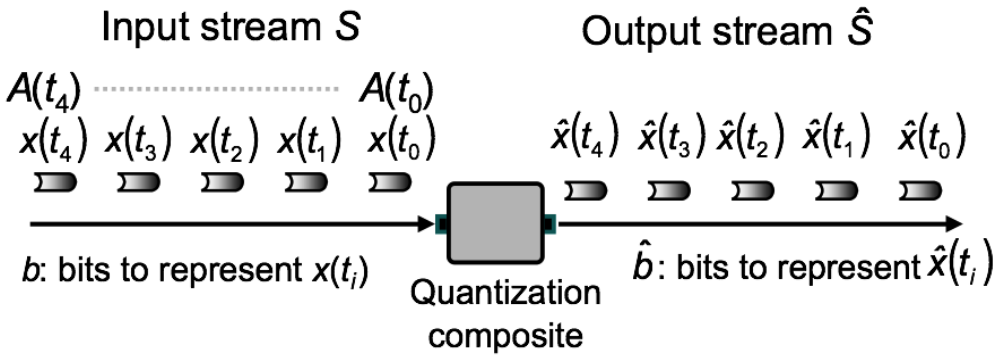


Count-Min Sketch Properties

- Set $m = ceil(2/\varepsilon)$ and $k = ceil(\log(1/\delta))$
- With probability 1- δ
  - The error in the estimate (|original-estimate|) is less than ε times the sum of all frequencies
- Say δ is 0.01 and ε is 10^{-6}
- So with probability 0.99, our estimate is going to be within $\pm 10^{-6}$ of the original count
- This will require 20MB state
- 5 hash functions (very quick update and query)

Quantization

- Simplest lossy data reduction mechanism
  - Perfect reconstruction of original signal not possible
- Maps Input Sample → Reconstruction Value from predefined codebook
- Example
  - Rounding Function
  - Floor Function
  - Truncation
- Original sample
  - Numeric Tuples with Continuous or Discrete Space



Quantization

- We go from b bits to b' bits, where $b' < b$
- The *quantization error*: A function on the difference between the original and the quantized value
- Advantages:
  - Reduces memory requirement
  - Reduces computation requirements
- Two main techniques
  - Scalar quantization
    - apply on a single attribute
  - Vector quantization
    - apply on a vector of attributes

Scalar Quantization

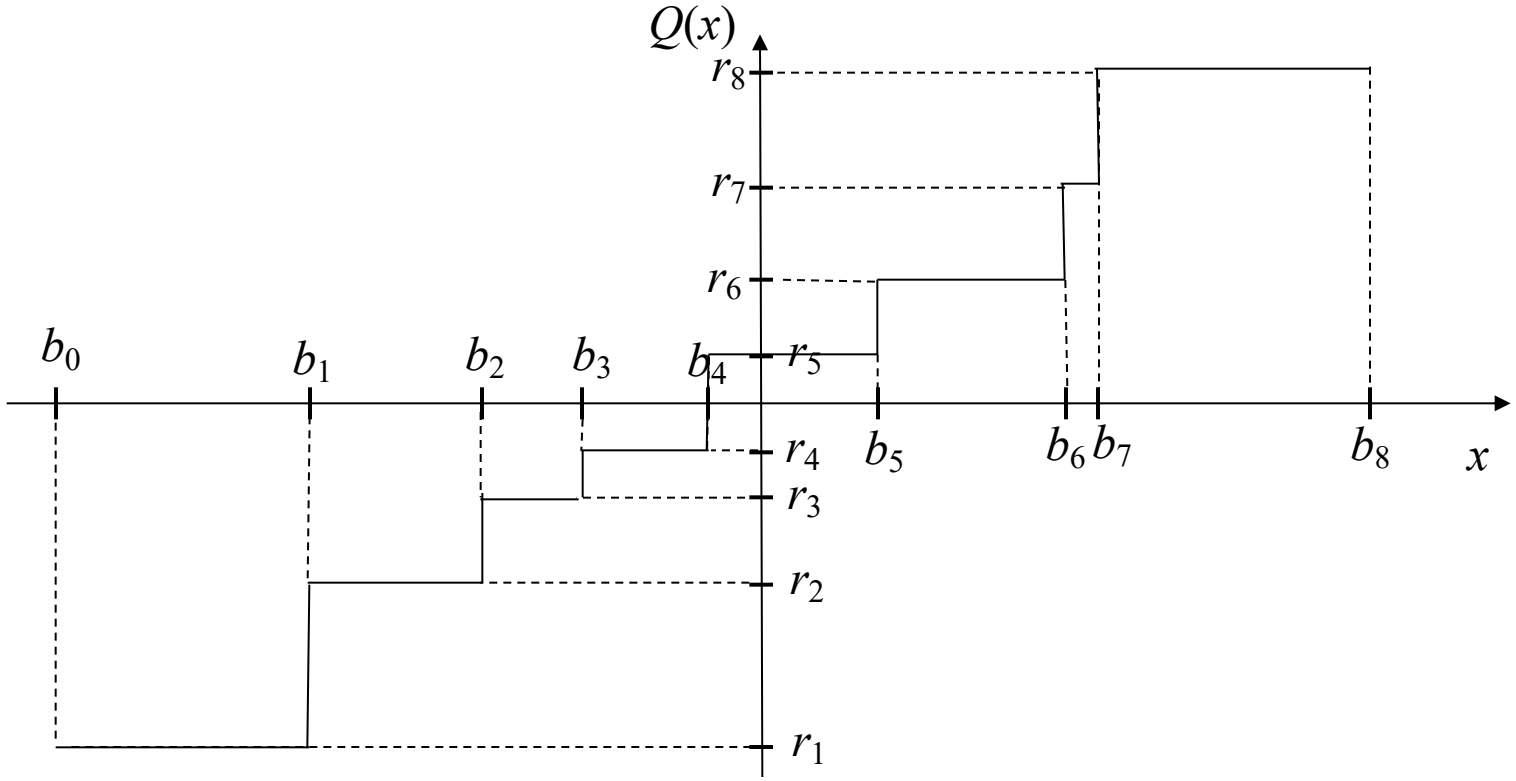
- Quantizer Q described by
  - L : the number of reconstruction levels
    - Set L=$\{1,2,..,L\}$ of reconstruction level indices
    - Requires at most $(\log_2 L)$ bits for representation
  - b_l : the set of boundary values
    - Boundary region $\mathrm{B}_l = [b_{l-1}, b_l)$
    - Support space $\mathrm{B} = \{\mathrm{B}_l\}$
  - r_l : the set of reconstruction levels

$$Q(x) = r_l \quad \text{if} \quad x \in \quad _l, \quad l \in$$

Partitions the support space of random variable x into L regions

Represents x with L discrete values

Scalar Quantization



Graphical Representation

Quantizing Streaming Data

- Applying a known quantizer (codebook)
  - Stateless operation
  - Very efficient
  - Does not matter whether data is windowed or not
- If codebook shared downstream
  - Can transmit just index – higher compression
  - Applied to batches and combined with entropy coding (e.g. Huffman Coding/Arithmetic Coding)

Quantizer Design: Scalar Quantization

- Select boundaries and reconstruction levels to minimize reconstruction error
  - Mean absolute error
  - Mean squared error
- Multiple Greedy Optimization Techniques
  - Lloyd-Max algorithm (not a streaming algorithm)

Scalar Quantization

$$D_Q = E\left[d\left(X, Q(X)\right)\right] = \int_{x\in} d\left(x, Q(x)\right) p_X(x)\, dx$$

Distance function

Probability density

$$d(x,y) = |x-y|$$

$$d(x,y) = |x-y|^2$$

Mean squared error (MSE)

$$p_X(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Gaussian

$$p_X(x) = \begin{cases} \frac{1}{b-a} & a \le x < b \\ 0 & \text{otherwise} \end{cases}$$

Uniform

$$p_X(x) = \frac{1}{2b} e^{-\frac{|x-\mu|}{b}}$$

Laplacian

Scalar Quantization

$$D_Q = \int_{x\in} d(x, Q(x)) p_X(x) dx$$

$$D_Q = \sum_{l=1}^{L} \int_{b_{l-1}}^{b_l} d(x, r_l) p_X(x) dx$$

$$D_Q = \sum_{l=1}^{L} \int_{b_{l-1}}^{b_l} (x - r_l)^2 p_X(x) dx$$

MSE distance function

To derive the optimal or Minimum Mean Squared Error Quantizer, take derivatives w.r.t. r_l and b_l and set to zero

Scalar Quantization

$$\frac{\partial D_Q}{\partial r_l} = 0 \Rightarrow \int_{b_{l-1}}^{b_l} 2(x - r_l) p_X(x) dx = 0$$

$$r_l = \frac{\int_{b_{l-1}}^{b_l} x p_X(x) dx}{\int_{b_{l-1}}^{b_l} p_X(x) dx} = E[X \mid X \in B_l]$$

Optimal reconstruction values lie at the centroid of each boundary region

$$\frac{\partial D_Q}{\partial b_l} = 0 \Rightarrow (b_l - r_l)^2 p_X(b_l) - (b_l - r_{l+1})^2 p_X(b_l) = 0$$

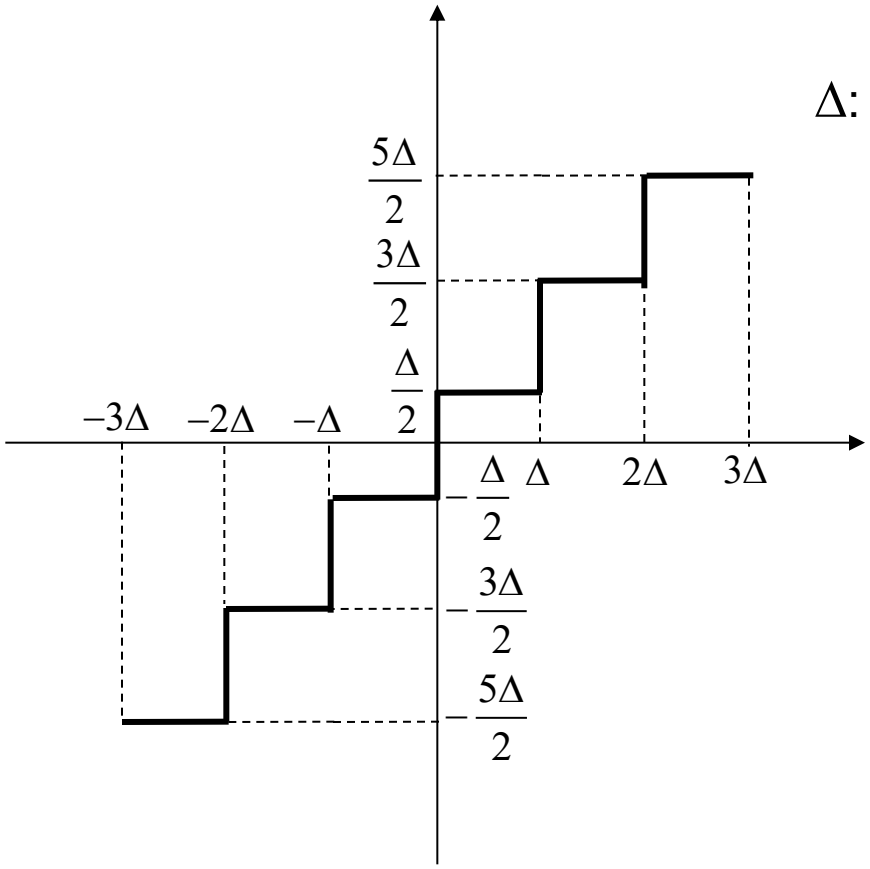
$$b_l = \frac{r_{l+1} + r_l}{2}$$

Lloyd-Max Algorithm

- Iterative Algorithm to determine r_l and b_l
  - Based on a set of training samples
- Step 1: Pick a set of reconstruction levels
- Step 2: Determine boundary partitions
  - Average of reconstruction levels (use nearest neighbor condition if discrete set of values)
- Step 3: Update reconstruction levels as centroids of each boundary region
- Step 4: Compute Quantization Distortion
- Iterate Steps 2 through 4 till Distortion converges

- Gradient Descent Based Algorithm
  - Convergence guaranteed to only Local Minimum (not necessarily Global Minimum)
- *Is this Streaming friendly?*

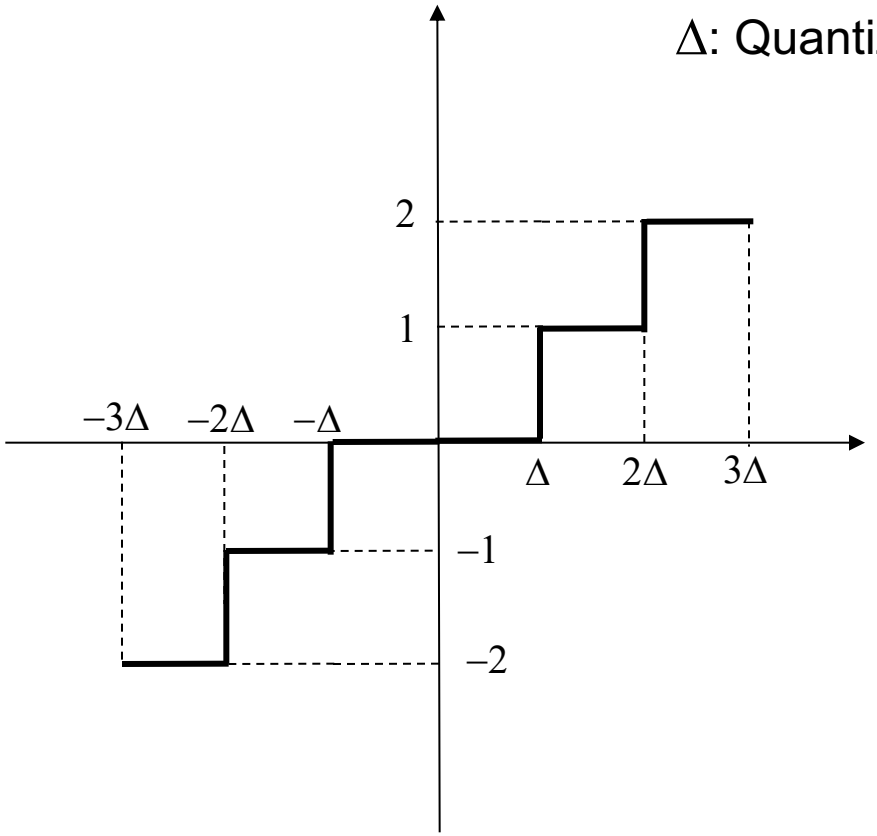
Uniform Quantizer



Δ: Quantization Step-size

Optimal MMSE Quantizer
for uniform distribution

Uniform Quantizer: Mid-tread



Uniform Mid-Tread Quantizer

Quantization with "Dead Zone"

"Dead Zone"

$$Q(x) = sign(x) \times \frac{|x + f(\Delta)|}{\Delta}$$

$$\hat{x} = \Delta Q(x)$$

Reconstruction of original

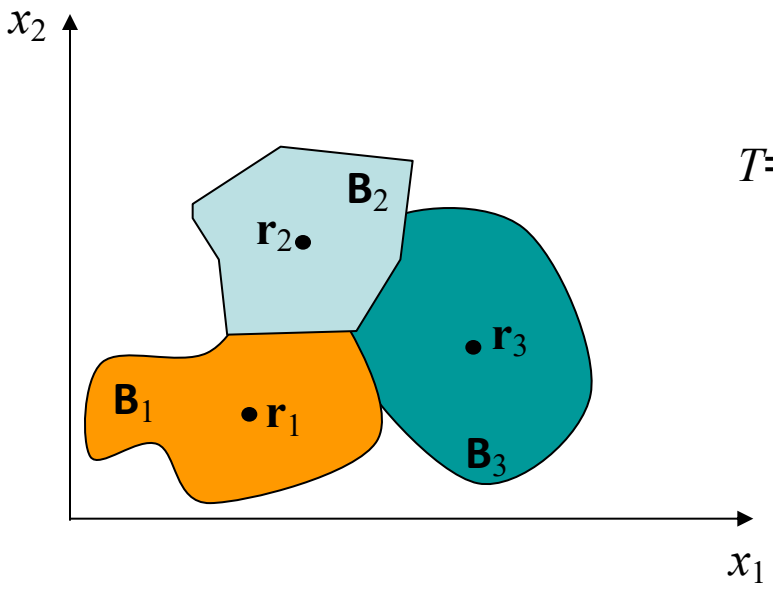
Vector Quantization

- Quantizer Q described by
  - T : the number of dimensions per vector
  - L : the number of reconstruction vectors
    - Set $\mathrm{L}=\{1,2,..,L\}$ of reconstruction vector indices
  - Boundary region B_l
    - Support space $\mathbf{B} = \{B_l\}$
  - $\mathbf{r}_l$: the set of reconstruction vectors (codewords)

$$Q(\mathbf{x}) = \mathbf{r}_l \text{ if } \mathbf{x} \text{ in } B_l$$

Partitions the support space of random vector $\mathbf{x}$ into L regions
Represents $\mathbf{x}$ with one of L discrete values

Illustration: 2D Vector Quantization



T=2, L=3

Represents vectors in this 2-D space with 3 reconstruction vectors

Bit-rate required after quantization ~ $\log_2 3$ per tuple ~ $1/T(\log_2 3)$ per dimension

Quantization – analogous to clustering: Similar goals and algorithms

Vector Quantization Algorithms

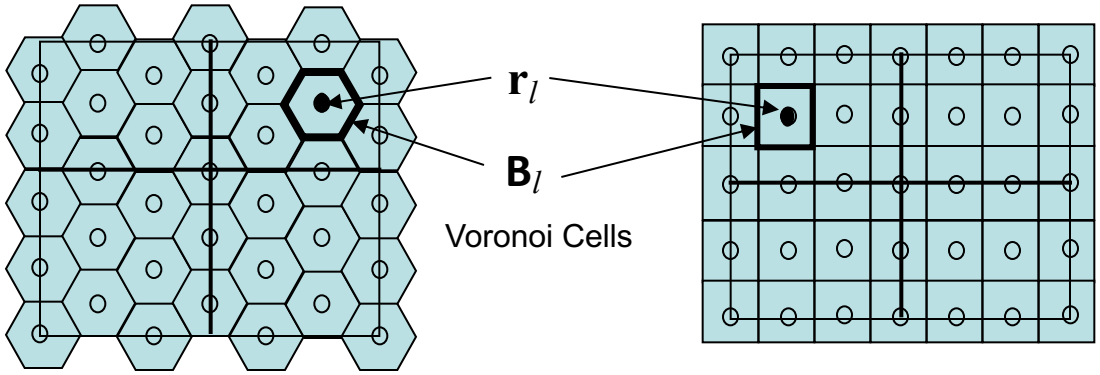
- Linde Buzo and Gray (LBG) Algorithm
  - Analogous to the Lloyd-Max algorithm
- Choice of initial codebook
  - A representative subset of the training vectors
    - e.g., some face blocks, some shirt blocks, some background blocks
  - Scalar quantization in each dimension
  - Splitting…

- Nearest Neighbor (NN) algorithm [Equitz, 1984]
  - Start with the entire training set
  - Merge the two vectors that are closest into one vector equal to their mean
  - Repeat until the desired number of vectors is reached, or the distortion exceeds a certain threshold

Uniform Vector Quantizer

Uniform vector quantizer is a lattice quantizer

Lattice in 2D



Hexagonal Lattice

Rectangular Lattice

Recall: Analogy to sampling

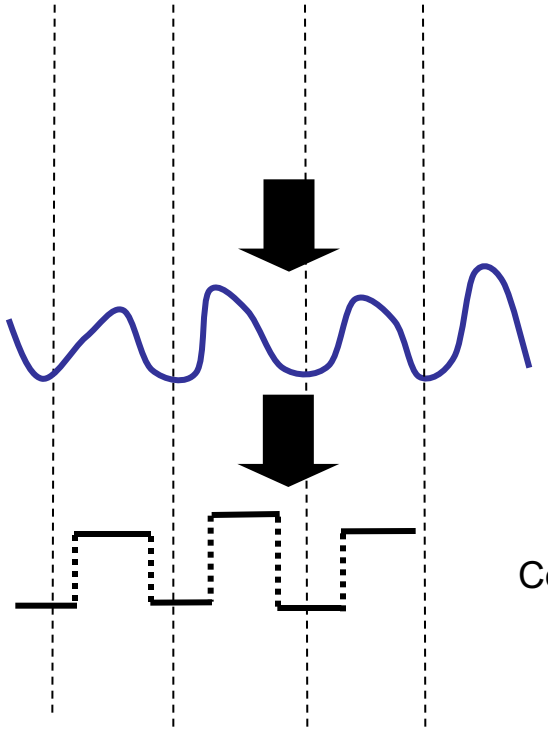
Codebook contains all lattice point (reconstruction vectors)
Nearest neighbor property used for quantization

Quantization for Stream Mining

- Given a quantizer design
  - Quantization is naturally incremental
- Quantizer design itself is not incremental
  - Can be performed within a tumbling window
- Goal may not be error in representation
  - Minimize mining specific error
    - Speaker Verification
  - ‘Shape’ preservation
    - Time series analysis and clustering

Shape Preserving Quantization

Properties of interest mostly limited to ‘shape’ of time series
Absolute values of time series samples not of interest



Original Time Series

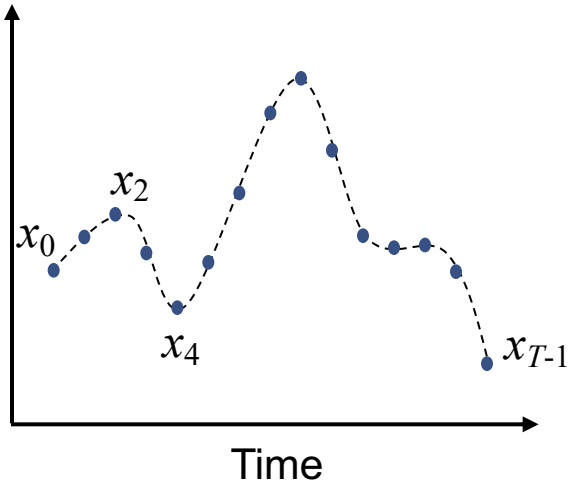
Shape Preserving Representation

Coarse representation (1-bit quantization)

All three representations capture the signal periodicity faithfully

Moment Preserving Quantization

Moments – capture the shape of the signal, e.g. Mean, Variance, Skew etc.



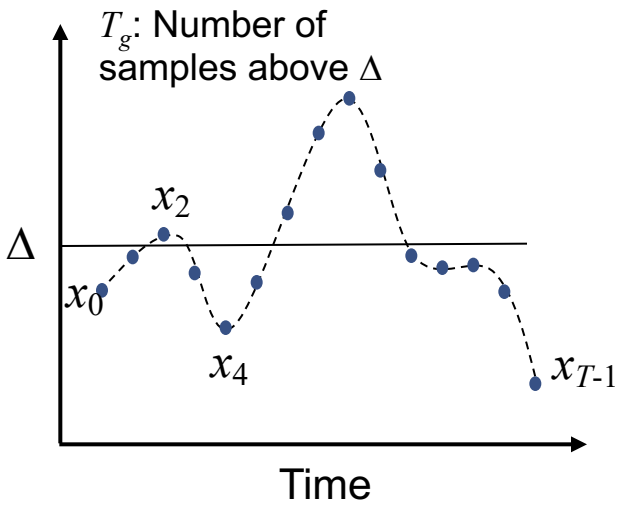
First sample moment $$E[x] = \mu = \frac{1}{T}\sum_{i=0}^{T-1} x_i$$

Second sample moment $$E[x^2] = \sigma^2 + \mu^2 = \frac{1}{T}\sum_{i=0}^{T-1} (x_i)^2$$

Consider a 1-bit quantizer $\Rightarrow$ uses only two values to represent signal

$$Q(x_i) = \begin{cases} a & x_i \geq \Delta \\ b & x_i < \Delta \end{cases}$$

Moment Preserving Quantization



$$Q(x_i) = \begin{cases} \overbrace{\mu + \sigma\sqrt{\frac{T - T_g}{T_g}}}^{a}; x_i \geq \Delta \\ \underbrace{\mu - \sigma\sqrt{\frac{T_g}{T - T_g}}}_{b}; x_i < \Delta \end{cases}$$

This choice of a and b guarantees preservation of first two moments

$$\frac{1}{T}\sum_{i=0}^{T-1} Q(x_i) = \frac{1}{T}\sum_{i=0}^{T-1} x_i$$

$$\frac{1}{T}\sum_{i=0}^{T-1} [Q(x_i)]^2 = \frac{1}{T}\sum_{i=0}^{T-1} (x_i)^2$$

Moment Preserving Quantization

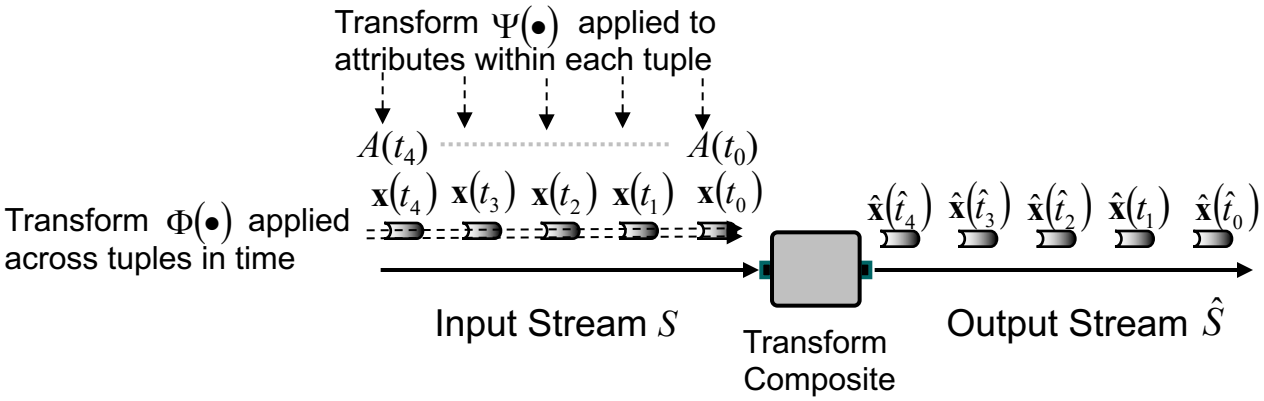
- Can select Δ to preserve third moment as well
  - Can set it to μ to avoid sending threshold
- Can be extended to B bit quantizer
  - Preserves 2^B+1 moments
- Ideal for preserving 'stationary' portions of signal
  - High compression rates with shape preservation
- Need to combine with change-point detection
  - To account for highly varying time series
- Windowed implementations of quantization
  - Including design of quantizer
  - Need to transmit a, b and Δ (may be implicit)
  - Different bit-allocation per time window

Quantization Summary

- Quantization used extensively in data compression
  - Rate-distortion optimization
  - Streaming: Rate-distortion-complexity optimization
- Quantizer design not incremental
  - Can be done window by window
  - *What is the impact on latency?*
- Link between vector quantization and clustering

Transforms

- Transform a tuple's attributes from one domain to another
  - Within tuple transform (for multivariate processing)
  - Across tuple transform (typically for univariate processing)
    - Temporal Transform



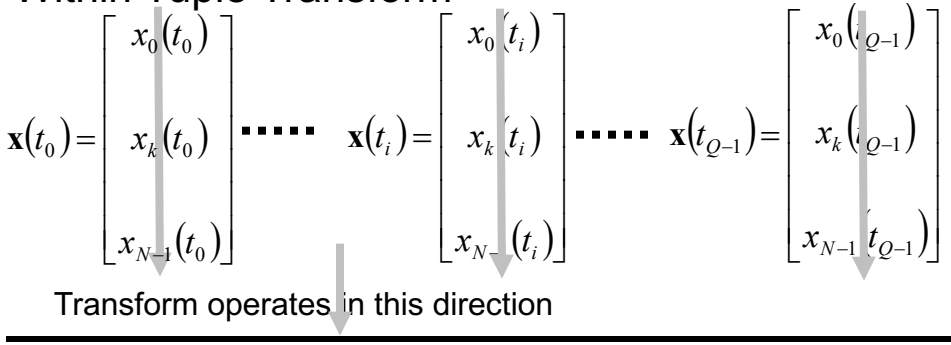
- Focus on Linear Transforms

1-D Transforms

$$\mathbf{x}(t_0)=\begin{bmatrix} x_0(t_0) \\ \vdots \\ x_k(t_0) \\ \vdots \\ x_{N-1}(t_0) \end{bmatrix} \cdots\cdots \mathbf{x}(t_i)=\begin{bmatrix} x_0(t_i) \\ \vdots \\ x_k(t_i) \\ \vdots \\ x_{N-1}(t_i) \end{bmatrix} \cdots\cdots \mathbf{x}(t_{Q-1})=\begin{bmatrix} x_0(t_{Q-1}) \\ \vdots \\ x_k(t_{Q-1}) \\ \vdots \\ x_{N-1}(t_{Q-1}) \end{bmatrix}$$

Q numeric input tuples with N attributes each

Within Tuple Transform

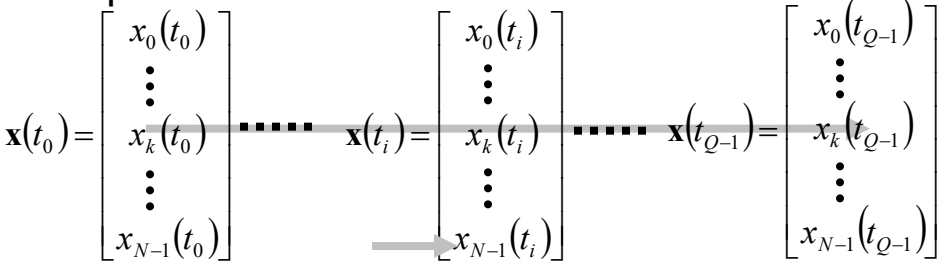


Transform operates in this direction

$$\mathbf{y}=\mathbf{x}(t_i)$$

$$\Psi(\mathbf{y})=\Lambda_{N\times N}\mathbf{x}(t_i)$$

Temporal Transform



Transform operates in this direction

$$y_i=x_k(t_i)$$

$$\Phi(\mathbf{y})=\Lambda_{Q\times Q}\mathbf{y}$$

Refresh on 1-D Linear Transforms

What makes a transform a linear transform?

Represent signal with N samples as vector

$$\mathbf{y} = [y_0, y_1, \ldots, y_{N-1}]^T$$

1-D transform represents signal as a linear combination of N basis vectors

In general, basis vector may be complex

$$\mathbf{u}_k \in C^N$$

This means $\mathbf{t} \in C^N$ even if $\mathbf{y} \in R^N$

$$\mathbf{y} = \sum_{k=0}^{N-1} t_k \mathbf{u}_k$$

$\mathbf{u}_k$: basis vector (linearly independent)

t_k: transformed coefficients

With $\mathbf{t} = [t_0, t_1, \ldots, t_{N-1}]^T$ and $\mathbf{U} = [\mathbf{u}_0, \mathbf{u}_1, \ldots, \mathbf{u}_{N-1}]$ we have

$$\mathbf{y} = \mathbf{U}\mathbf{t}$$ Inverse transform

and

$$\mathbf{t} = \mathbf{U}^{-1}\mathbf{y}$$ Forward transform

Unitary 1-D Linear Transforms

Basis vectors are orthonormal

$$\left\langle \mathbf{u}_j, \mathbf{u}_k \right\rangle = \mathbf{u}_j^H \mathbf{u}_k = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}$$

where $\mathbf{u}_j^H = \left(\mathbf{u}_j^*\right)^T$ transpose of complex conjugate

The matrix $\mathbf{U}$ is called unitary and we have $\mathbf{U}^H\mathbf{U} = \mathbf{U}\mathbf{U}^H = \mathbf{I}$

$\mathbf{y} = \mathbf{U}\mathbf{t}$ Inverse transform

and

$\mathbf{t} = \mathbf{U}^H\mathbf{y}$ Forward transform

1-D Fourier Transform

$$t_k = \frac{1}{\sqrt{N}} \sum_{n=0}^{N-1} y_n e^{-j\frac{2\pi kn}{N}}$$ Forward transform

$$y_n = \frac{1}{\sqrt{N}} \sum_{k=0}^{N-1} t_k e^{j\frac{2\pi kn}{N}}$$ Inverse transform

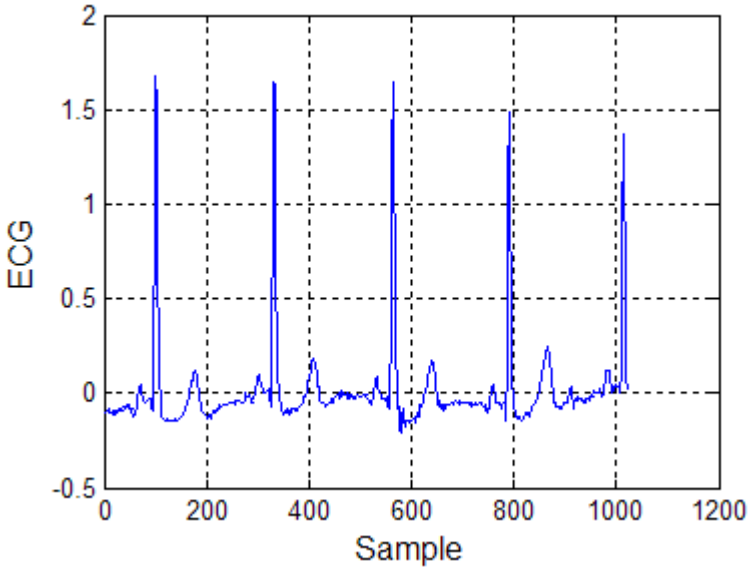
$$\mathbf{u}_k = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 \\ e^{j\frac{2\pi k}{N}} \\ \vdots \\ e^{j\frac{2\pi k(N-1)}{N}} \end{bmatrix}$$

$$\mathbf{U} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & 1 & \cdots & 1 \\ 1 & e^{j\frac{2\pi}{N}} & \cdots & e^{j\frac{2\pi(N-1)}{N}} \\ \vdots & \vdots & \cdots & \vdots \\ 1 & e^{j\frac{2\pi(N-1)}{N}} & \cdots & e^{j\frac{2\pi(N-1)(N-1)}{N}} \end{bmatrix}$$

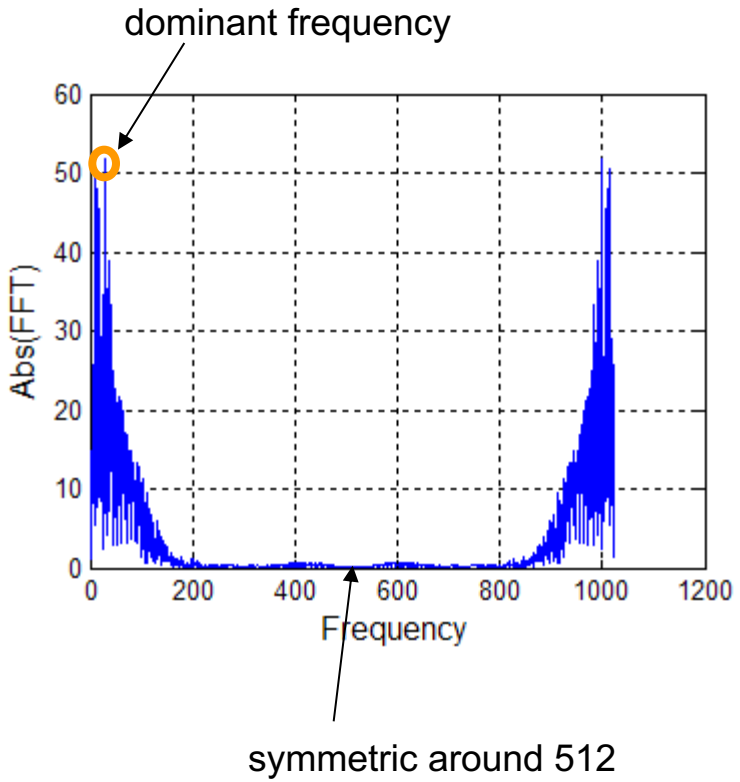
For real signal $\mathbf{y}$ we have $t_k = t^*_{N-k}$ or symmetry in Fourier Transform

1-D Fourier Transform

For real signal $\mathbf{y}$, n has interpretation of time, k has interpretation of frequency



1024 point sampled ECG signal



1-D Discrete Cosine Transform

Similar* to the Fourier transform but uses only real valued bases

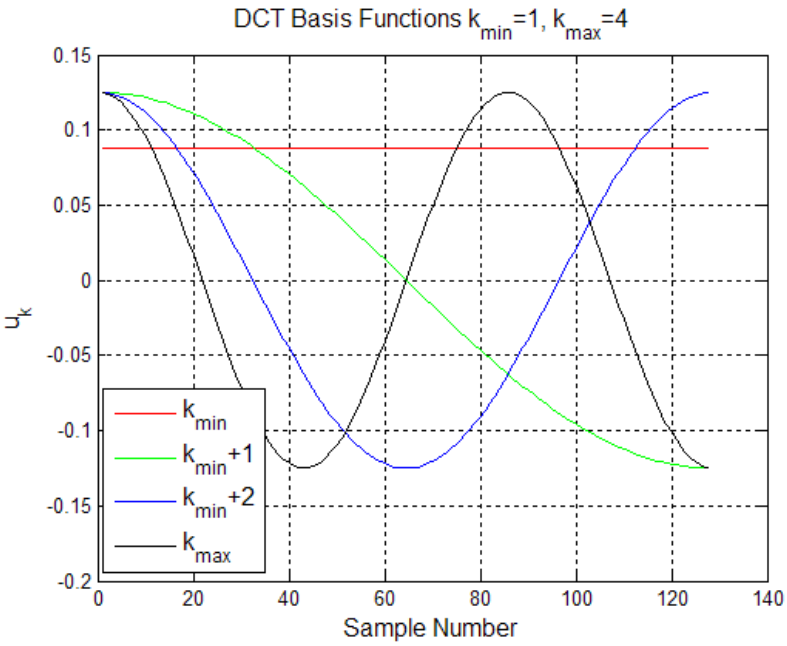
$$t_k = \sum_{n=0}^{N-1} y_n \alpha(k) \cos\left(\frac{(2n+1)k\pi}{2N}\right)$$ Forward transform

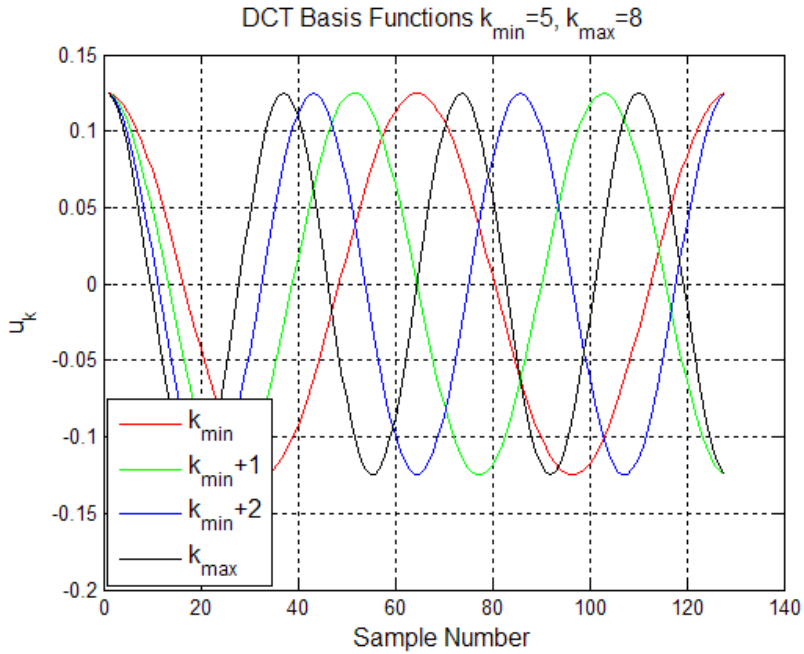
$$\alpha(k) = \begin{cases} \frac{1}{\sqrt{N}}; & k = 0 \\ \sqrt{\frac{2}{N}}; & \text{otherwise} \end{cases}$$

$$y_n = \sum_{k=0}^{N-1} t_k \alpha(k) \cos\left(\frac{(2n+1)k\pi}{2N}\right)$$ Inverse transform

$$\mathbf{U} = \frac{1}{\sqrt{N}} \begin{bmatrix} 1 & \sqrt{2}\cos\left(\frac{\pi}{2N}\right) & \cdots & \sqrt{2}\cos\left(\frac{(N-1)\pi}{2N}\right) \\ 1 & \sqrt{2}\cos\left(\frac{3\pi}{2N}\right) & \cdots & \sqrt{2}\cos\left(\frac{3(N-1)\pi}{2N}\right) \\ \vdots & \vdots & \cdots & \vdots \\ 1 & \sqrt{2}\cos\left(\frac{(2N-1)\pi}{2N}\right) & \cdots & \sqrt{2}\cos\left(\frac{(2N-1)(N-1)\pi}{2N}\right) \end{bmatrix}$$

1-D Discrete Cosine Transform

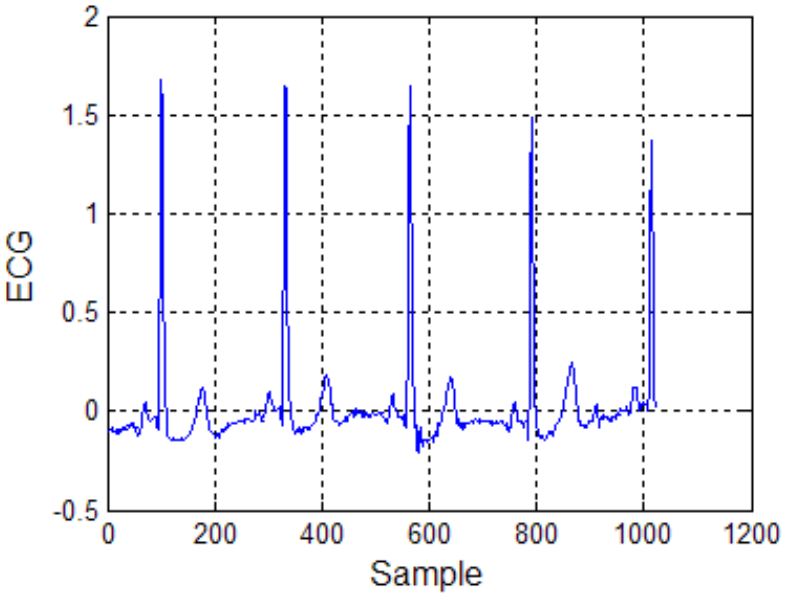




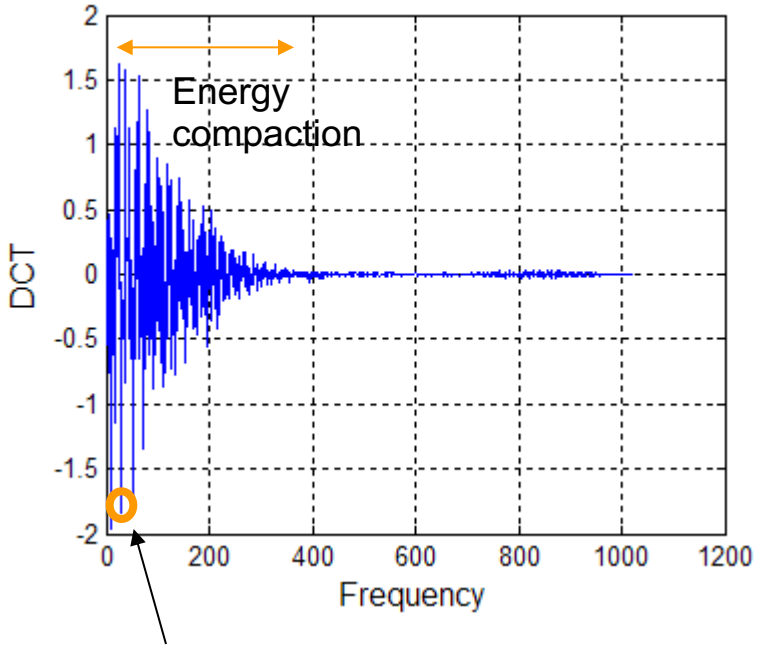
First 8 basis vectors for N=128

Basis functions are sinusoids with increasing frequency

1-D Discrete Cosine Transform

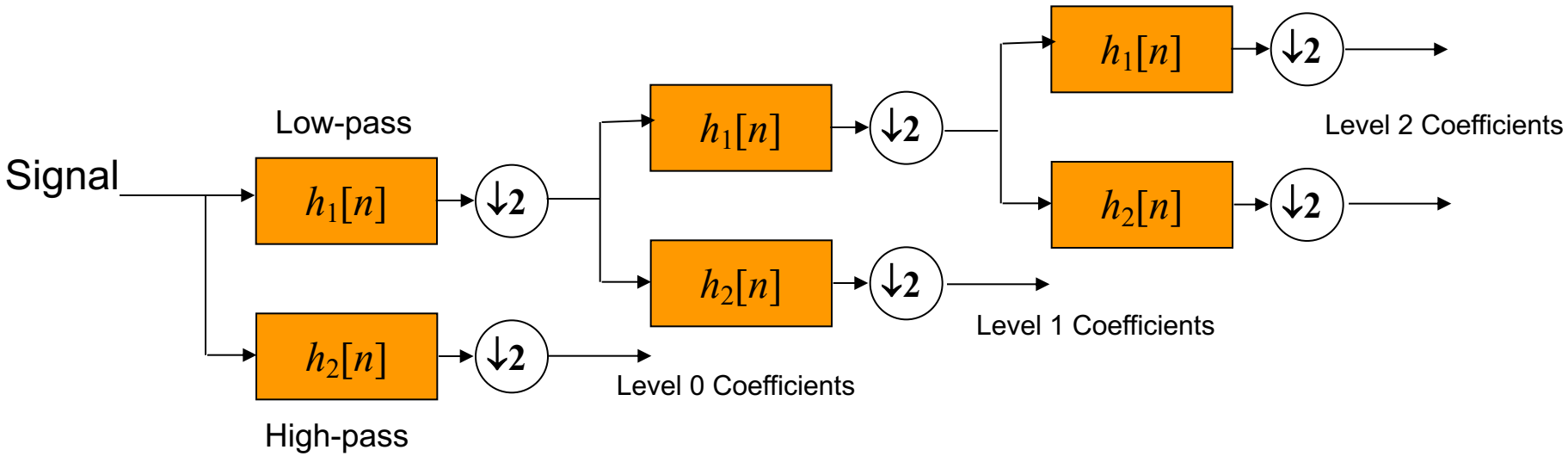


1024 point sampled ECG signal



dominant frequency

Subband Coding: Wavelet Transform

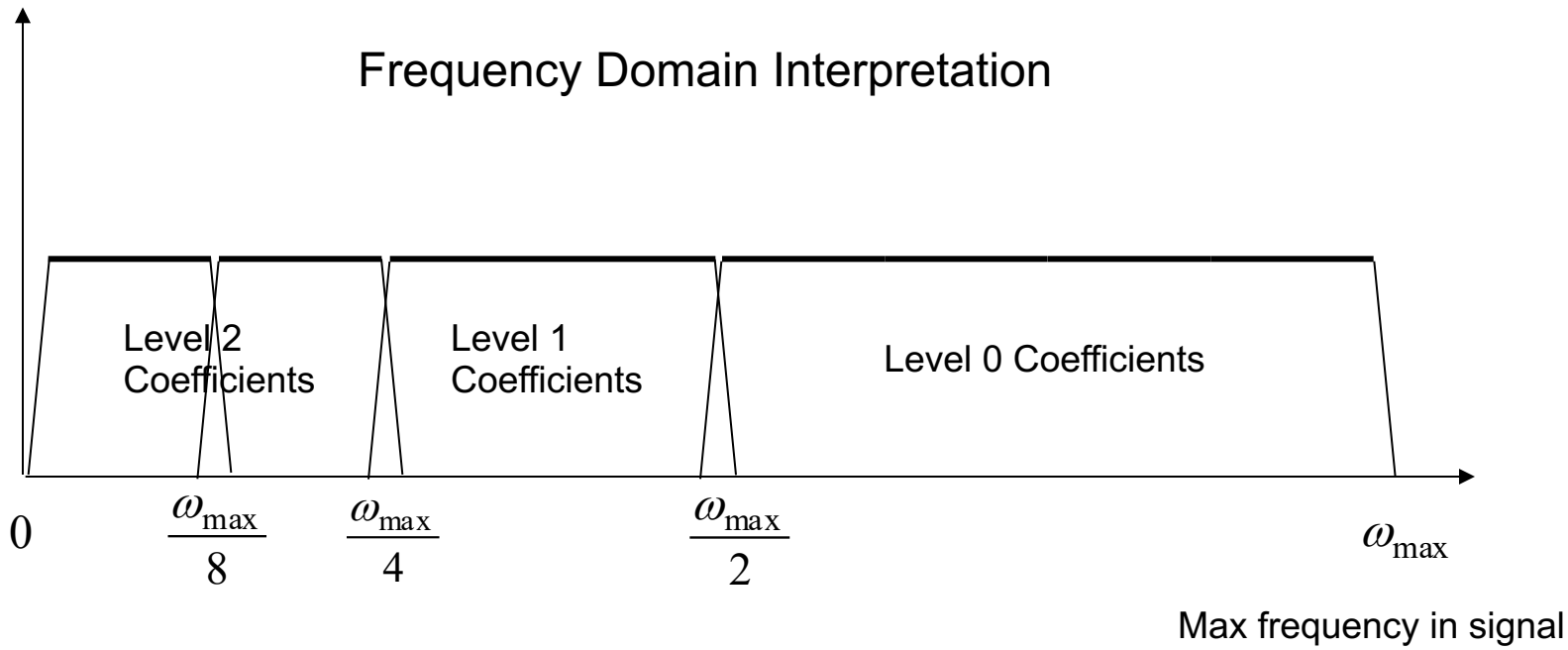


Recursively apply decomposition to low freq band $\Rightarrow$ increase frequency resolution

Set of filters collectively called – "Filter bank"

Multi-resolution representation of signal – **Scalable Coding**

Subband Coding: Wavelet Transform



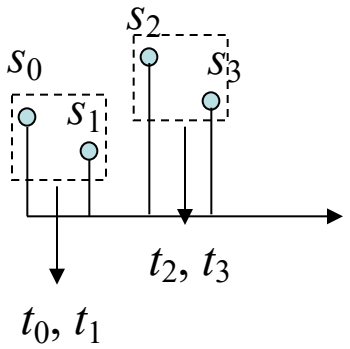
Wavelet Transform: Recursively partitions frequency space
Provides time-frequency localization

1-D Haar Wavelet Transform

$$\mathbf{U} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix} \qquad \mathbf{U}^H = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

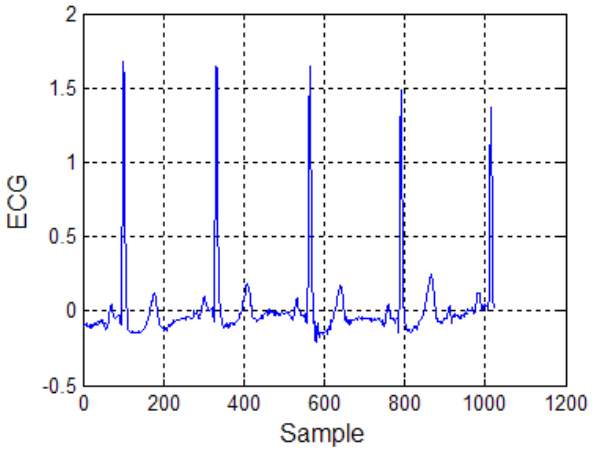
$$t_0 = \frac{s_0 + s_1}{\sqrt{2}} \qquad t_1 = \frac{s_0 - s_1}{\sqrt{2}}$$

Normalized sum and difference

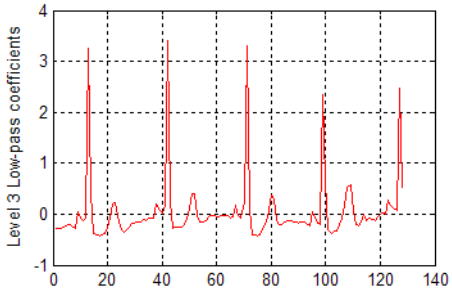


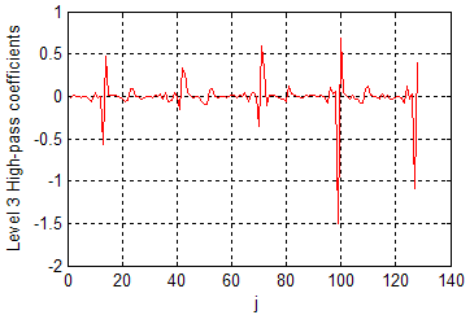
Windowing Based Implementation

1-D Haar Wavelet Transform

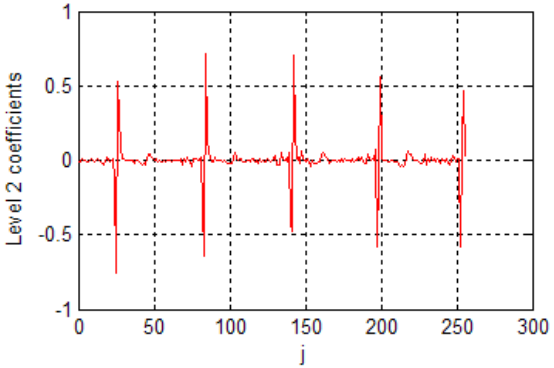


Original Signal

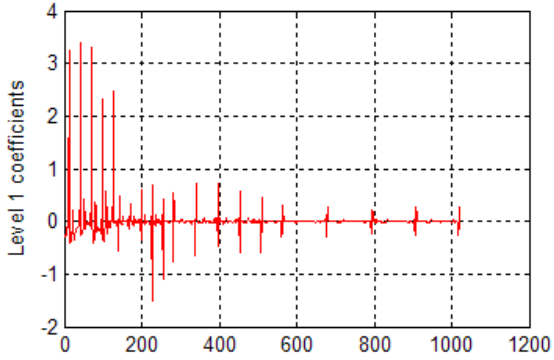




Level 3 Coefficients



Level 2 Coefficients

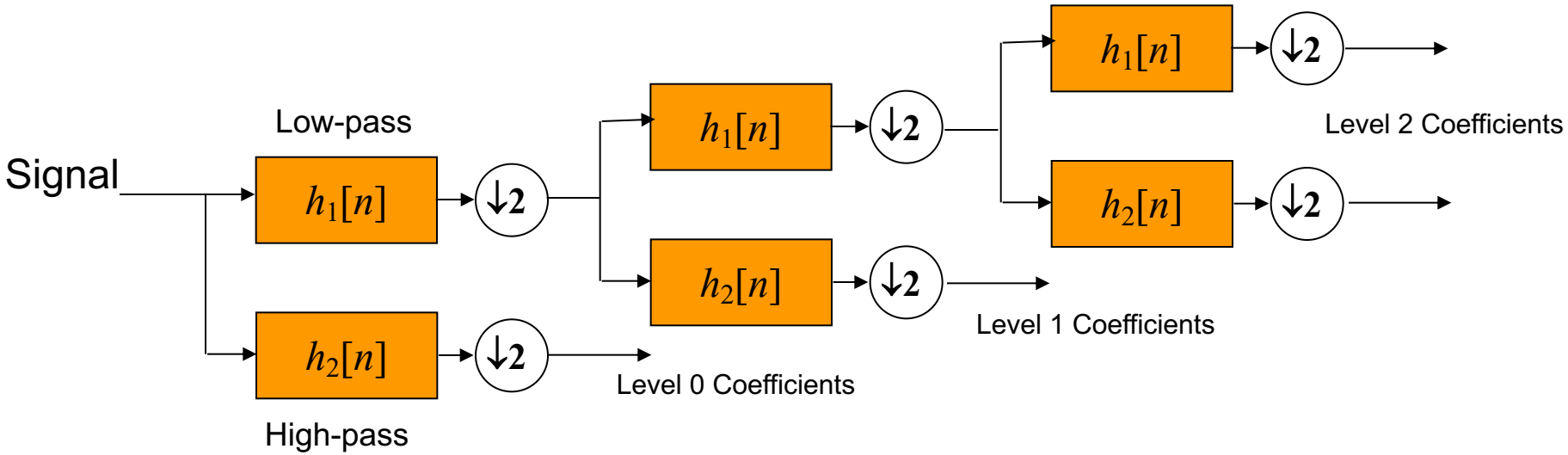


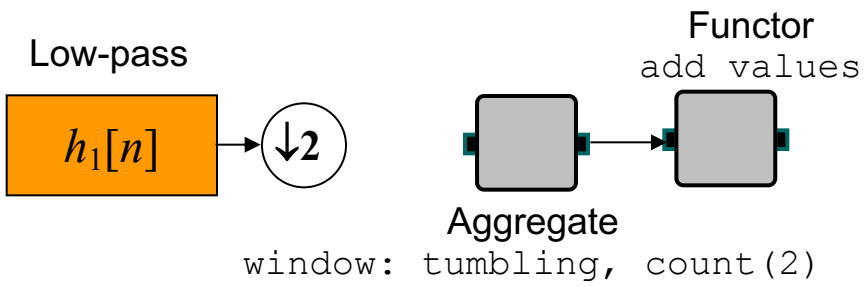
Haar Wavelet Transform

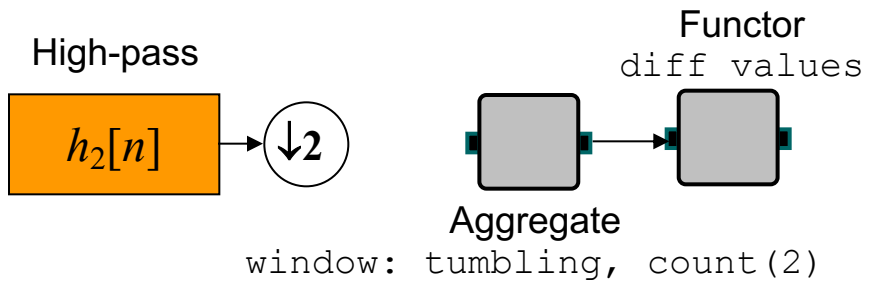
How does this relate to compressibility?

Implementing the Temporal Haar Transform

Applied window by window: For a given window, with a fixed size







Implementing the Temporal Haar Transform

- Fixed structure
  - if window size fixed, and known apriori can construct topology at compile time
- What can we do if window size unknown apriori?
  - Construct a transform matrix on the fly
  - Consider a 2 level Haar Transform. *Can we build a single transform matrix for this?*

$$\mathbf{U} = \frac{1}{\sqrt{2}}\begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

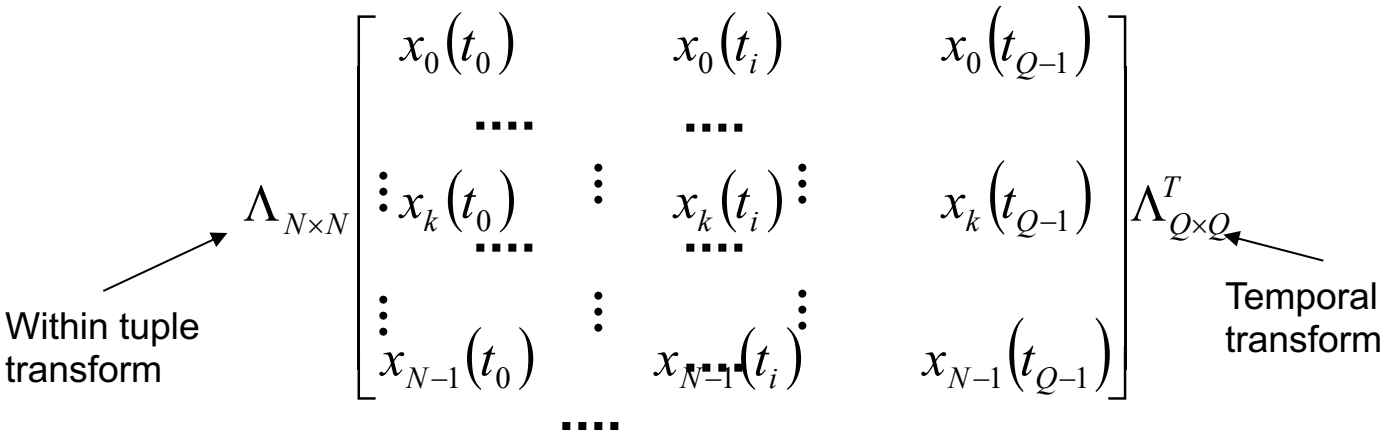
$$\Phi = \frac{1}{2}\begin{bmatrix} 1 & 1 & 1 & 1 \\ 1 & 1 & -1 & -1 \\ \sqrt{2} & \sqrt{2} & 0 & 0 \\ 0 & 0 & \sqrt{2} & -\sqrt{2} \end{bmatrix}$$

Implementing the Haar Transform

- Temporal transform with unknown window size
  - Aggregate tuples for the window
  - Construct the transform matrix
  - Apply across the tuples (or tuple attributes)
- Within tuple transform
  - Can be applied tuple by tuple
  - Purely streaming – with transform matrix approach
  - Do not need any aggregation

From 1-D to 2-D Transforms

2-D Transform: Across tuples and across attributes within tuple



Collect tuples into one matrix $\mathbf{X}_{N\times Q}$

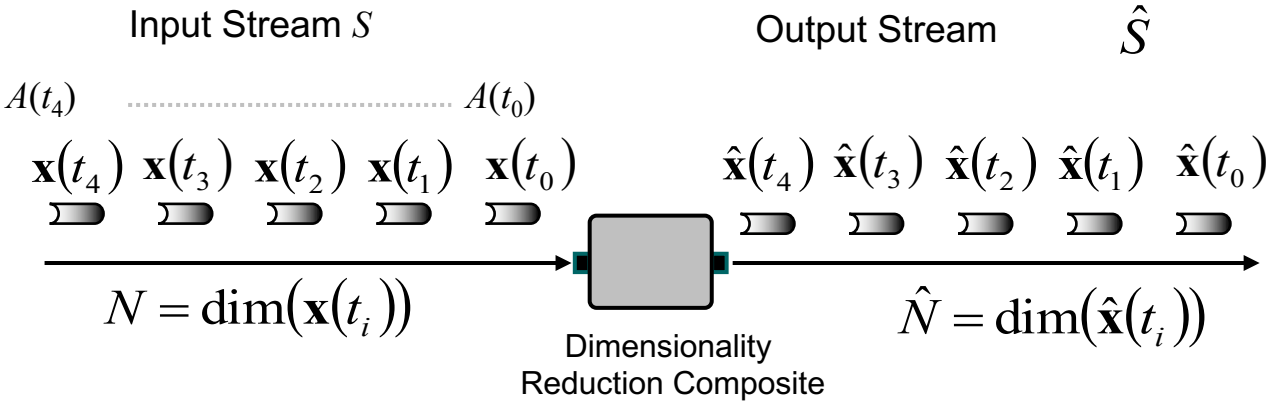
Assume Separability

Transforms: Summary

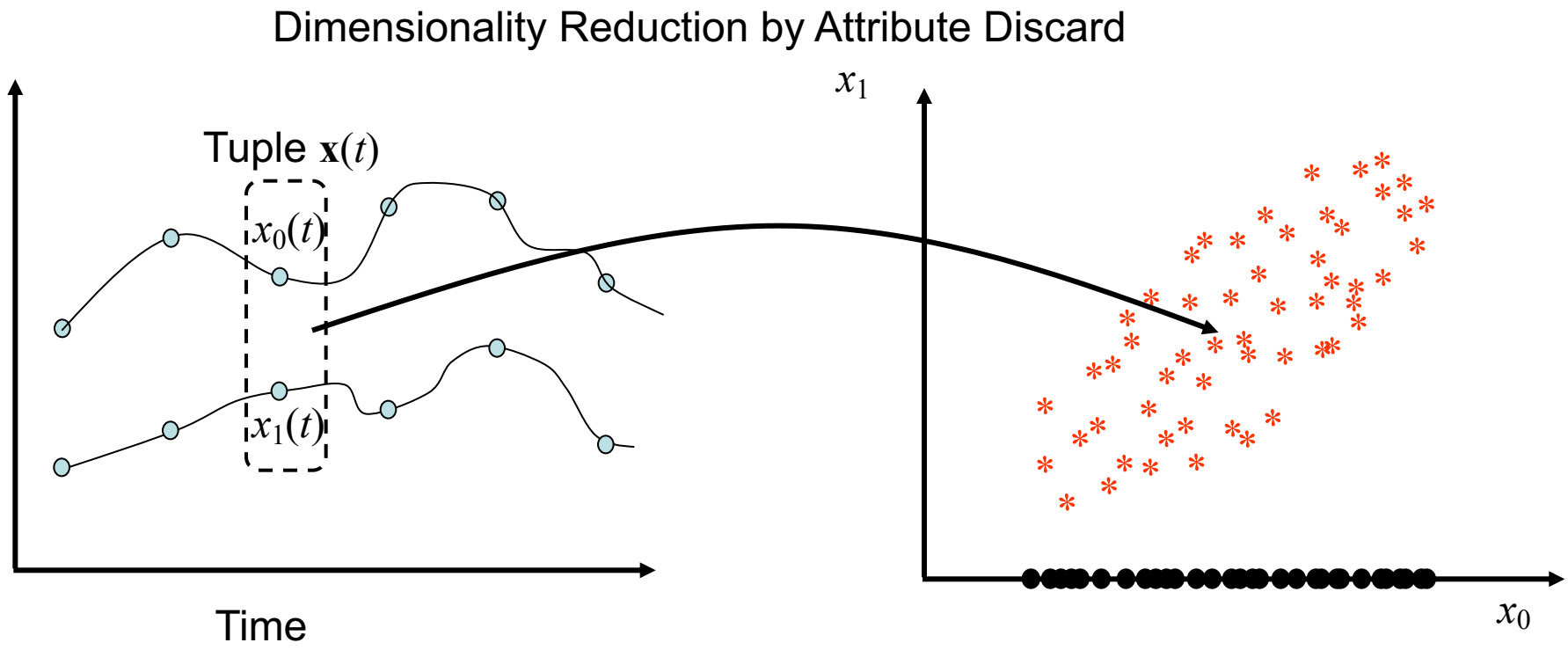
- Capture time-frequency properties of the signal
  - Allow different features to be extracted
- Allow approximating signal with smaller number of coefficients
  - Energy compaction
  - Link to dimensionality reduction and sketches
- Linear transforms
  - Provide windowed computation
  - Provide reversible representations

Dimensionality Reduction

- Applied within tuple
- Reduce number of attributes within tuple
  - From N to $\hat{N}$
- For numeric tuple
  - Appropriately transform the data into domain where attributes can be discarded
  - See link with energy compacting transforms

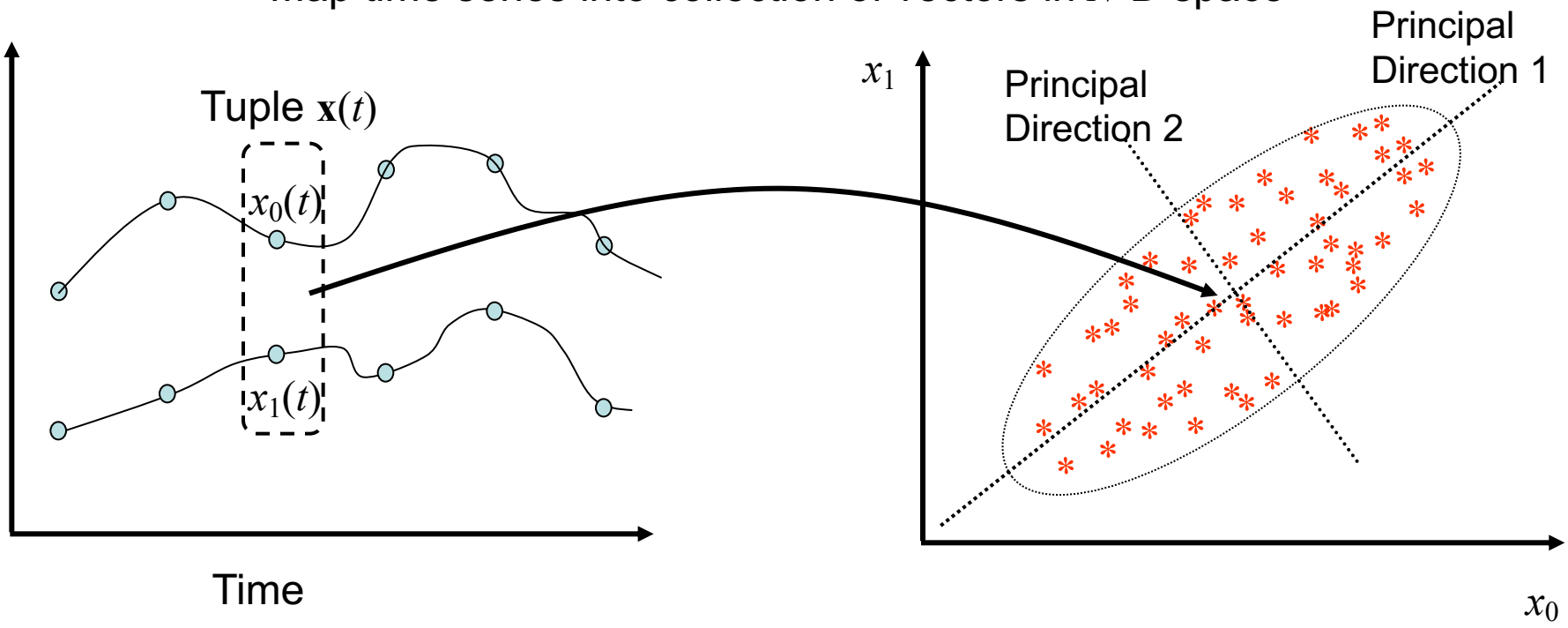


Dimensionality Reduction 2-D Example



Dimensionality Reduction: 2-D Example

Map time series into collection of vectors in N-D space



PCA identifies principal directions of variation in signal

Refresh: PCA and Karhunen-Loewe Transform

- Special kind of unitary transforms
  - Basis vectors are eigenvectors of covariance matrix
  - Decorrelates the dimensions
- Provides guaranteed smallest average squared reconstruction error
  - When only K ($<N$) coefficients are used to approximate signal
  - Consider that we look at Q time steps

$$\boldsymbol{\mu}_S = \frac{1}{Q}\sum_{j=0}^{Q-1}\mathbf{x}(t_j)$$

Sample Mean vector

$$\boldsymbol{\Sigma}_S = \frac{1}{Q}\sum_{j=0}^{Q-1}\left(\mathbf{x}(t_j)-\boldsymbol{\mu}\right)\left(\mathbf{x}(t_j)-\boldsymbol{\mu}\right)^T$$

Sample Covariance matrix

PCA and KLT

The basis vectors for KLT are eigenvectors of the covariance matrix of **s**

$$\boldsymbol{\Sigma}_S \mathbf{u}_k = \lambda_k \mathbf{u}_k$$

Using the relationship $\boldsymbol{\Sigma}_T = \mathbf{U}^H \boldsymbol{\Sigma}_S \mathbf{U}$ we have

$$\boldsymbol{\Sigma}_T = \begin{bmatrix} \mathbf{u}_0^H \\ \vdots \\ \mathbf{u}_{N-1}^H \end{bmatrix} \boldsymbol{\Sigma}_S \begin{bmatrix} \mathbf{u}_0 & \cdots & \mathbf{u}_{N-1} \end{bmatrix}$$

$$\boldsymbol{\Sigma} = \begin{bmatrix} \mathbf{u}_0^H \\ \vdots \\ \mathbf{u}_{N-1}^H \end{bmatrix} \begin{bmatrix} \lambda_0 \mathbf{u}_0 & \cdots & \lambda_{N-1} \mathbf{u}_{N-1} \end{bmatrix} = \begin{bmatrix} \lambda_0 & 0 & \cdots & 0 \\ 0 & \lambda_1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & \lambda_{N-1} \end{bmatrix}$$

Diagonal Matrix

Decorrelating transform

The eigenvalues are equal to the variance of transformed coefficients $\lambda_k = \sigma_{t_k}^2$

PCA and KLT

- Provides optimal squared error representation
  - Very efficient dimensionality reduction
  - Added de-correlating property
- Hard to compute in practice
  - Need all vectors available
  - Need to compute covariance matrix
  - Need to compute eigenvectors
  - Not directly suited for streaming data
- Incremental PCA algorithms
  - Approximate PCA

SPIRIT Algorithm

- SPIRIT: Streaming Pattern dIscoveRy in multIple Timeseries
- Transform based dimensionality reduction
  - Online algorithm: Compute the basis vectors (eigenvectors) incrementally instead of doing an eigen-decomposition
  - Can change amount of reduction incrementally
- Used in multiple applications
  - Forecasting
  - Pattern Detection
  - Anomaly Detection

SPIRIT Algorithm

Input dimensions N
Output dimensions $\hat{N}$

Initialization

Basis vectors

$$\mathbf{u}_k = \begin{bmatrix} \vdots \\ 1 \\ \vdots \end{bmatrix}$$

Energy (eigenvalue)

$$d_k = \varepsilon$$

$$0 \le k \le \hat{N} - 1$$

$$\mathbf{z} = \mathbf{x}(t)$$

New Tuple

for k=0: $\hat{N}$ -1

Iterate through the reduced number of dimensions

$$\lambda_k = (\mathbf{z})^T \mathbf{u}_k$$

Project onto k-th basis vector

Iterative Update

$$d_k = (\lambda_k)^2 + \alpha d_k$$

Update energy of k-th basis vector

SPIRIT Algorithm

$\mathbf{z} = \mathbf{x}(t)$ — New Tuple

for k=0: $\hat{N}$ -1 — Iterate through the reduced number of dimensions

Iterative Update

$\lambda_k = (\mathbf{z})^T \mathbf{u}_k$ — Project onto k-th basis vector

$d_k = (\lambda_k)^2 + \alpha d_k$ — Update energy of k-th basis vector

$\mathbf{e}_k = \mathbf{z} - \lambda_k \mathbf{u}_k$ — Find approximation error

$\mathbf{u}_k = \mathbf{u}_k + \frac{1}{d_k} \lambda_k \mathbf{e}_k$ — Update basis vector

$\mathbf{z} = \mathbf{z} - \lambda_k \mathbf{u}_k$ — Update basis vector

end

submit vector $\begin{bmatrix} \lambda_0 & \dots & \lambda_{\hat{N}-1} \end{bmatrix}$

Dimensionality Reduction

- SPIRIT
  - Can produce approximations to PCA
  - Number of vectors can be changed incrementally (based on error measures)
- Other techniques
  - MUSCLES
  - Distributed Dimensionality Reduction
  - SVD based dimensionality reduction

Summary

Techniques	Tuple Types	Windowing
Sketches	Numeric or non-numeric Tuples, Univariate or Multivariate	Not windowed. Can be tumbling window
Quantization	Numeric tuples. Univariate (scalar) or Multivariate (vector)	Tumbling window based
Transforms	Numeric tuples. Univariate (across tuples) or multivariate (within tuples) or both	Not windowed. Temporal transforms typically tumbling window
Dimensionality Reduction	Numeric tuples. Multivariate	Not windowed. Can be tumbling window

Wrapup

- Pre-processing Algorithms
  - Sketches
  - Quantization
  - Transforms
  - Dimensionality Reduction
- Next lecture
  - Stream mining and modeling algorithms
  - Classification
  - Regression

Reading

- Book – Chapter 10: Preprocessing Algorithms
- Other Reading
  - Sampling and Summarization: Data Streams: Models and Algorithms (Chapter 2, Chapter 9, Chapter 12)
    - Sampling (Uniform, Non-uniform, threshold based)
      - Signal Processing Text
    - Reservoir Sampling
      - http://www.cs.umd.edu/~samir/498/vitter.pdf
    - Linear Transforms
      - Signal Processing Text (Alan V. Oppenheim, Ronald W. Schafer, John R. Buck : Discrete-Time Signal Processing)
  - Quantization: Signal Processing Text
    - Task specific Quantization for Speaker Verification
      - H. Tseng et al, “Quantization for Adapted GMM-based speaker verification”, ICASSP 2006.
    - Moment Preserving Quantization
      - E. Delp, M. Saenz, and P. Salama. "Block Truncation Coding (BTC)," The Handbook of Image and Video Processing. Academic Press, 2000.