Lecture

Speech and Audio Signal Processing

TECHNISCHE UNIVERSITÄT DARMSTADT

Lecture 2: Prediction & Codebook processing

Content of the lecture

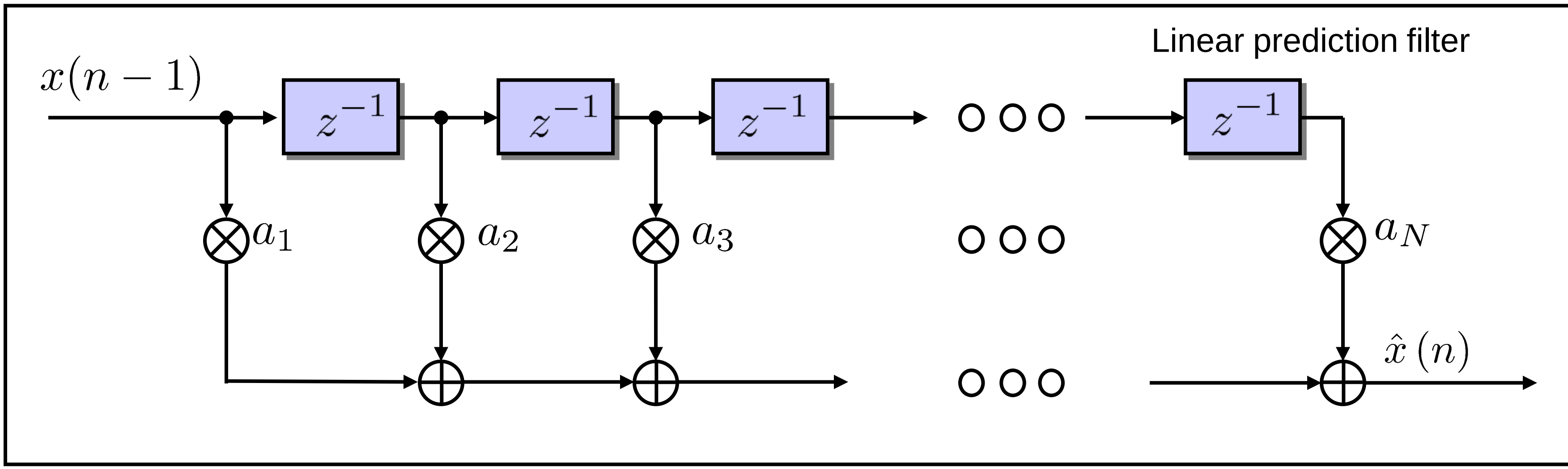
- Prediction
  - Derivation
  - Properties of the prediction error filter

- Codebook processing
  - Vector quantization
  - Principle of codebook processing
  - Applications of codebook processing
  - Training of codebooks
  - Efficient codebook search

Prediction

Prediction of the current signal sample based on the last N signal samples:

$$\hat{x}(n) = \sum_{i=1}^{N} a_i \, x(n-i)$$



With:

- $\hat{x}\,(n)$: Estimation for $x(n)$
- a_i : prediction coefficients
- N : Length / order of the prediction filter

Prediction

Prediction of the current signal sample based on the last N signal samples:

$$\hat{x}(n) = \sum_{i=1}^{N} a_i \, x(n-i) \qquad e(n) = x(n) - \hat{x}(n) = x(n) - \sum_{i=1}^{N} a_i \, x(n-i)$$

Minimization of the mean square error:

$$\boxed{\mathrm{E}\left\{e^2(n)\right\} \stackrel{!}{=} \min} \longrightarrow \frac{\partial}{\partial a_j} \mathrm{E}\left\{e^2(n)\right\} \stackrel{!}{=} 0$$

$$2\,\mathrm{E}\left\{e(n) \frac{\partial}{\partial a_j} e(n)\right\} \stackrel{!}{=} 0$$

$$\mathrm{E}\left\{e(n)\, x(n-j)\right\} \stackrel{!}{=} 0 \qquad \forall j$$

$$\mathrm{E}\left\{\left[x(n) - \sum_{i=1}^{N} a_i \, x(n-i)\right] x(n-j)\right\} \stackrel{!}{=} 0 \qquad \forall j$$

$$\sum_{i=1}^{N} a_i \, \mathrm{E}\left\{x(n-i)\, x(n-j)\right\} = \mathrm{E}\left\{x(n)\, x(n-j)\right\} \qquad \forall j$$

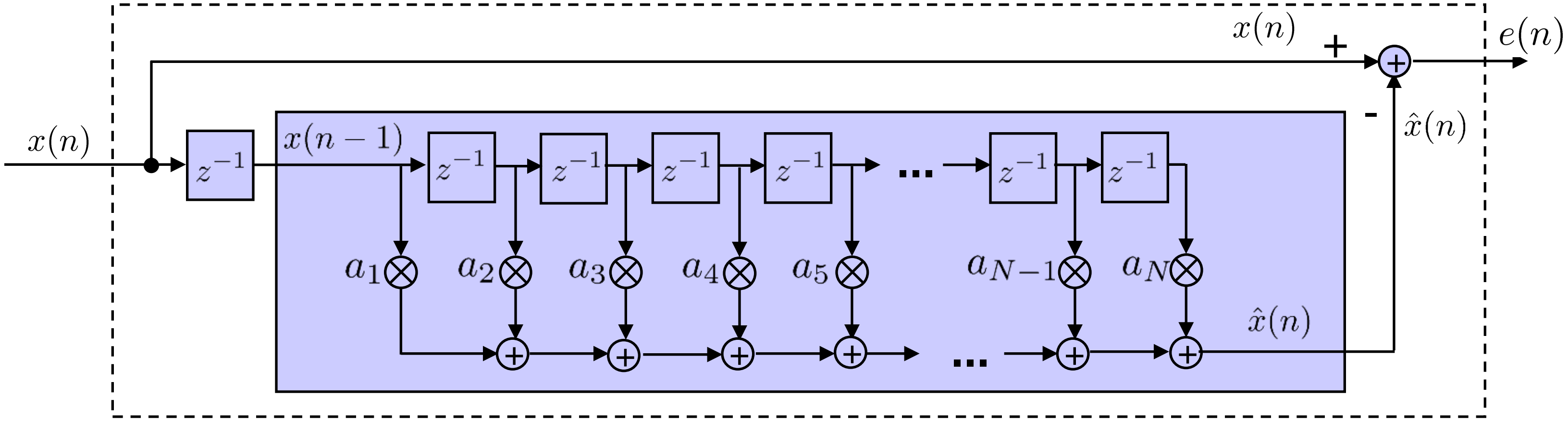
$$\boxed{\sum_{i=1}^{N} a_i \, r_{xx}(i-j) = r_{xx}(j) \qquad \forall j}$$

Prediction

$$\boldsymbol{a}_{\text{opt}} = \boldsymbol{R}_{xx}^{-1}\,\boldsymbol{r}_{xx}(1)$$

$$\begin{bmatrix} a_1 \\ a_2 \\ \vdots \\ a_N \end{bmatrix} = \begin{bmatrix} r_{xx}(0) & r_{xx}(1) & \cdots & r_{xx}(N-1) \\ r_{xx}(1) & r_{xx}(0) & \cdots & r_{xx}(N-2) \\ \vdots & \vdots & \ddots & \vdots \\ r_{xx}(N-1) & r_{xx}(N-2) & \cdots & r_{xx}(0) \end{bmatrix}^{-1} \begin{bmatrix} r_{xx}(1) \\ r_{xx}(2) \\ \vdots \\ r_{xx}(N) \end{bmatrix}$$

Prediction error filter



Properties of the **prediction error filter**

- Reduction of the output power:

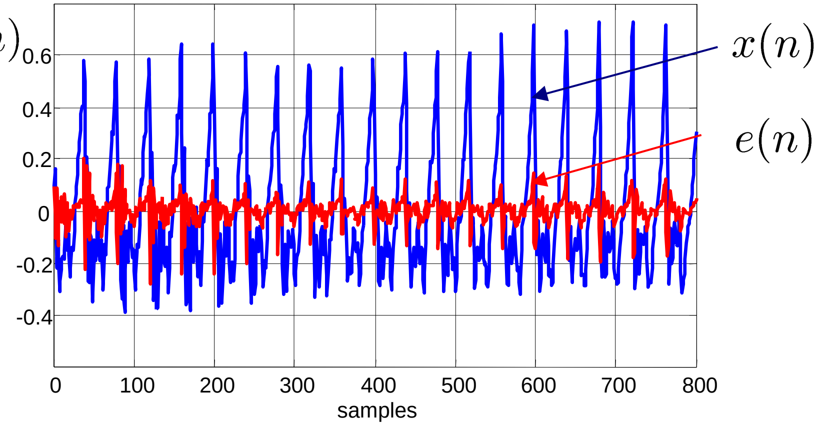
  Power of $e(n)$ equal or smaller than power of $x(n)$

  Prediction error gain (equal or larger than 1!):

$$\boxed{\frac{\mathrm{E}\{x^2(n)\}}{\mathrm{E}_{\min}} = \frac{r_{xx}(0)}{r_{xx}(0) - \boldsymbol{r}_{xx}^{\mathrm{T}}(1)\,\boldsymbol{R}_{xx}^{-1}\,\boldsymbol{r}_{xx}(1)}}$$

$$\mathrm{E}_{\min} = \mathrm{E}\{e^2(n)\}|_{\boldsymbol{a}\,=\,\boldsymbol{R}_{xx}^{-1}\,\boldsymbol{r}_{xx}(1)} = r_{ee}(0)$$

Error signal: prediction order 10; prediction gain: 13.6 dB

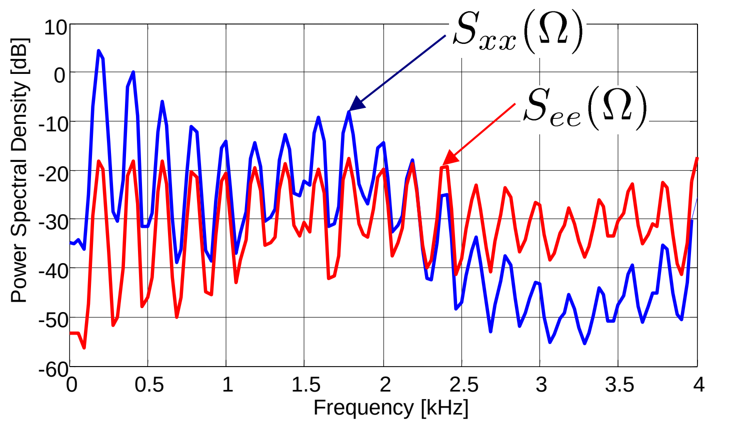


- Whitening of the output signal:

  The higher the prection order the closer e(n) is to a white signal

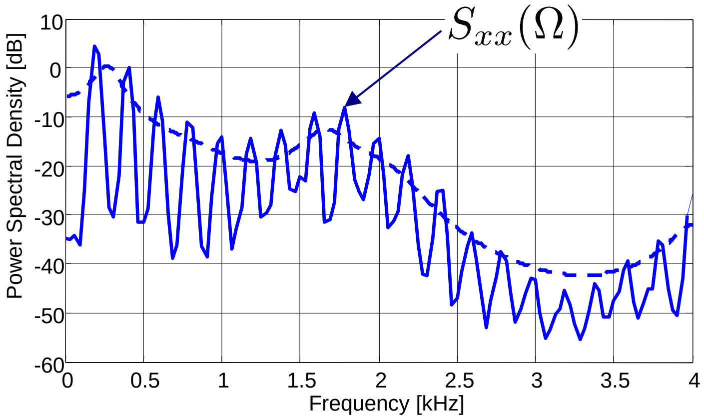
$$\boxed{r_{ee}(l) = 0 \quad \text{for} \;\; |l| > 0 \;\; \text{and} \;\; N \to \infty}$$

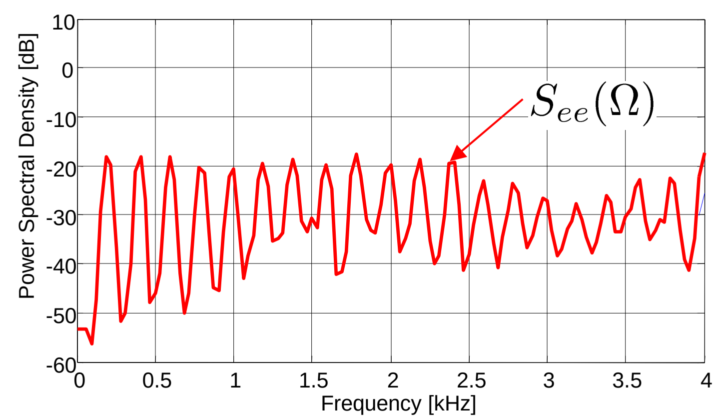
Error signal: prediction order 10; prediction gain: 13.6 dB

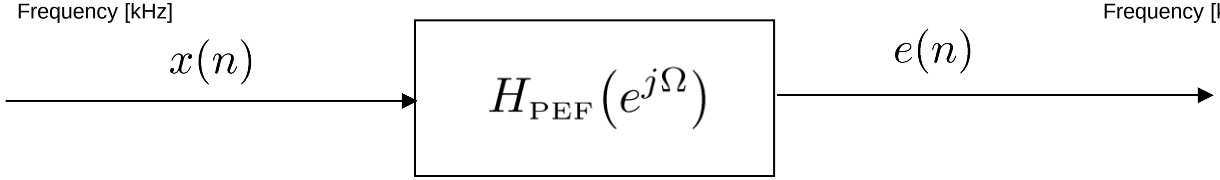


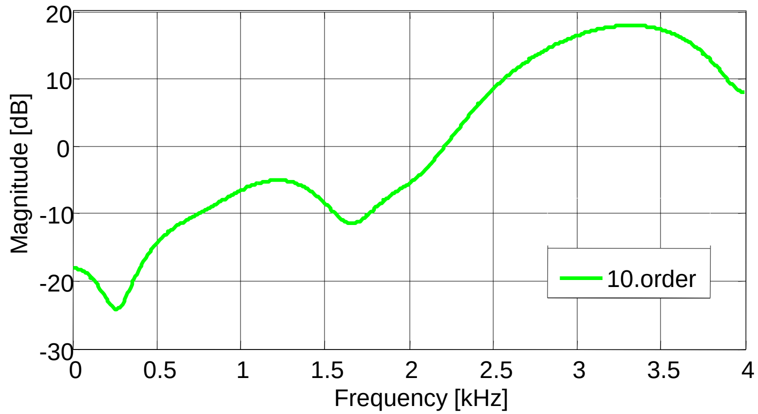
O sinal com menos redundancia é um sinal branco

Properties of the **prediction error filter**









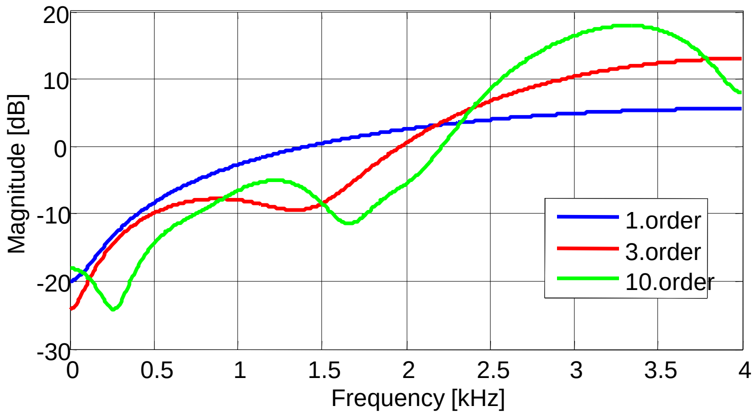
- Prediction error filter:

  $$H_{\mathrm{PEF}}\left(e^{j\Omega}\right)$$

  modelling the **inverse** spectral envelope

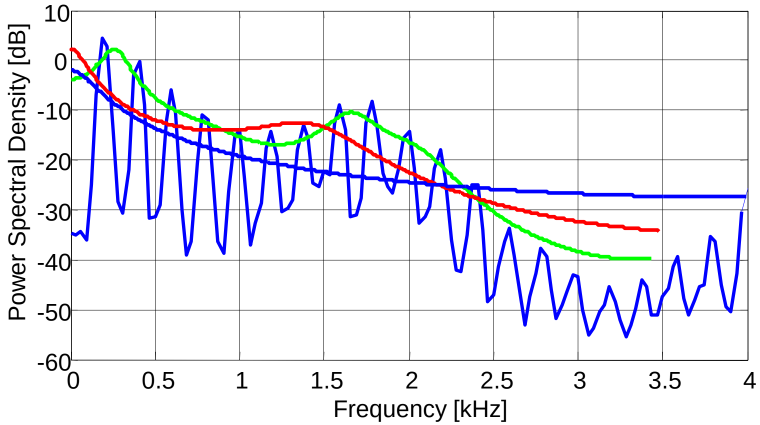
Properties of the **prediction error filter**

- Prediction error filter :



$$H_{\mathrm{PEF}}\left(e^{j\Omega}\right) = 1 - \sum_{i=1}^{N} a_i \, e^{-j\Omega i}$$

- Inverse prediction error filter
  Approximation of the input signal PSD:

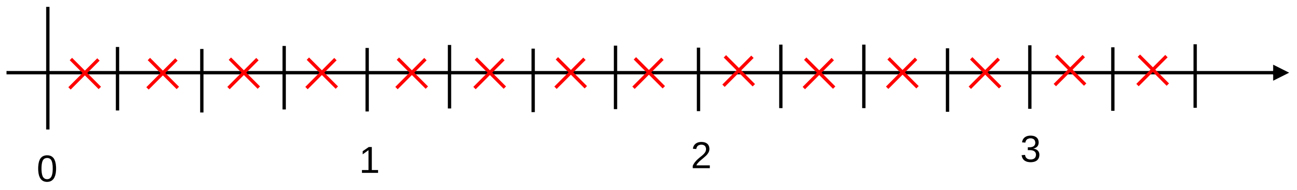


$$H_{\mathrm{inv.\,PEF}}\left(e^{j\Omega}\right) = \frac{1}{1 - \sum_{i=1}^{N} a_i \, e^{-j\Omega i}}$$

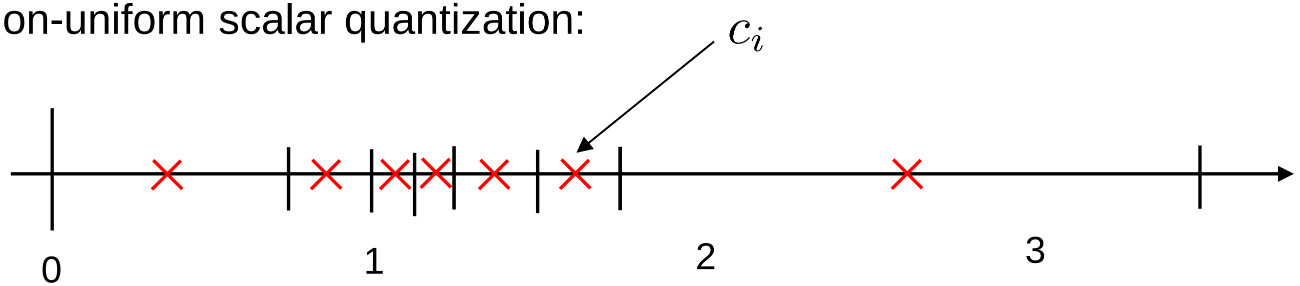
Parametric model of the spectral envelope
of the input signal.

Scalar quantization

- Uniform scalar quantization:



- Non-uniform scalar quantization:



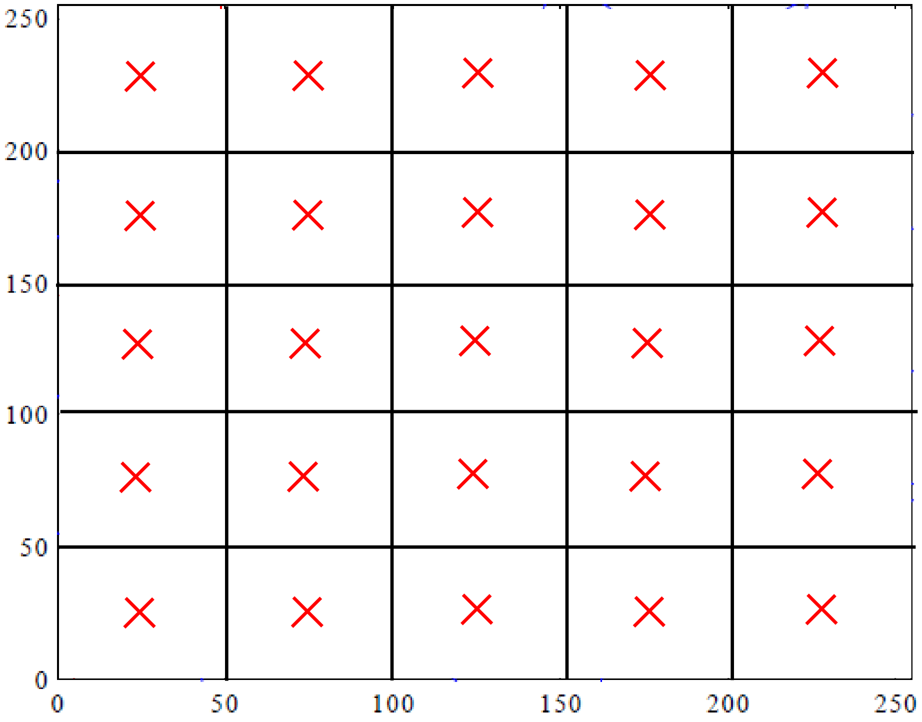
Non-uniform quantization is designed based on the distribution of the input data.

Quantization target: Minimize the mean distance of the K quantized values: $c_i \quad i \in [0 \ldots K-1]$
and the N (training) data values: $x(n) \quad n \in [0 \ldots N-1]$

$$\frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0 \ldots K-1} |x(n) - c_i|^2 \to \min$$

Vector quantization

- Uniform (2D) vector quantization:



Index i of 25 quantized values

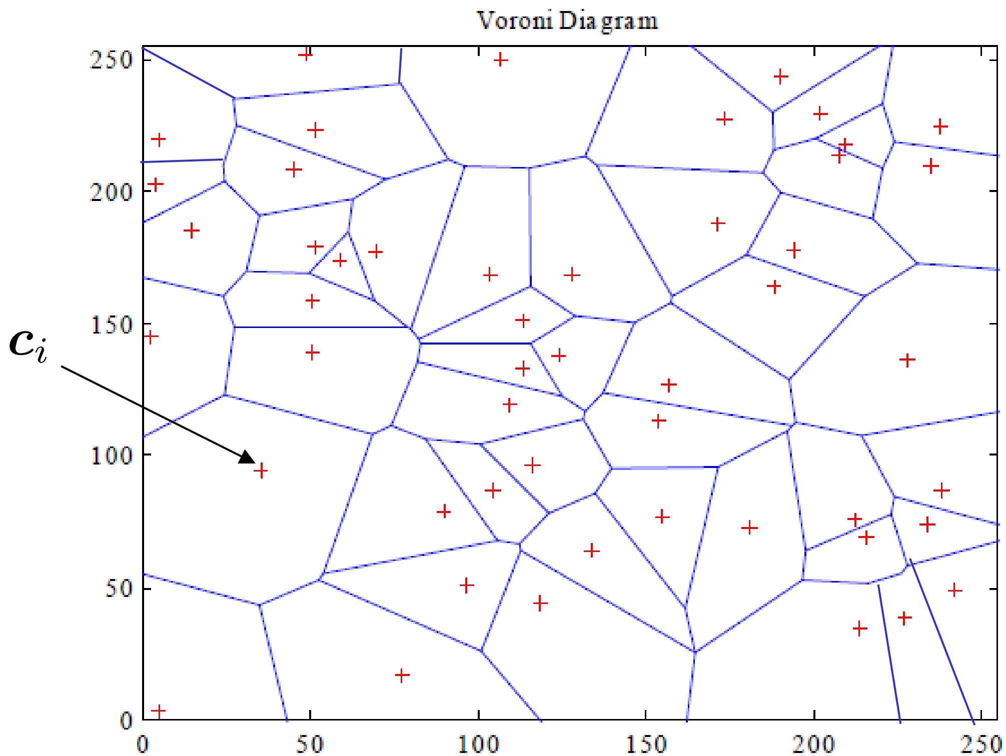
25 quantized values c_i

1: (25,25)
2: (25,75)
3: (25,125)
4: (25,175)
5: (25,225)
6: (75,25)
7: (75,75)
8: (75,125)
…
24: (225,175)
25: (225,225)

Vector quantization

- Non-uniform (2D) vector quantization:

Target for determining the quantized values: $$\frac{1}{N}\sum_{n=0}^{N-1}\min_{i=0\ldots K-1}||\boldsymbol{x}(n)-\boldsymbol{c}_i||^2 \rightarrow \min$$



All values within one cell are quantized to the red cross value within the cell.

The cells and the quantized values are determined based on the training data which should be representative for the data to quantize.

Vector quantization => Principle of codebooks

- Quantized vectors set up a "Codebook" which can efficiently quantize data, "feature vectors".

- For the setup of a codebook

Codebook matrix

$$\boldsymbol{C}_K = \left[\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{K-1}\right]$$

with:

Codebook vector

$$\boldsymbol{c}_i = \left[c_{i,0},\ c_{i,1},\ \ldots,\ c_{i,M-1}\right]^{\mathrm{T}}$$

M é a dimensão do vetor c / dos dados x

- The codebook vectors should be chosen such that they represent the feature vectors $\boldsymbol{x}(n)$ with a minimum mean distance.

- As distance measure $d(\boldsymbol{x}(n),\ \boldsymbol{c}_i)$, typically the quadratic distance is used:

$$||\boldsymbol{x}(n) - \boldsymbol{c}_i||^2$$

- and a codebook should be set up such as to fulfill:

$$\frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0 \ldots K-1} ||\boldsymbol{x}(n) - \boldsymbol{c}_i||^2 \rightarrow \min$$

Codebook training: LBG algorithm

Tentar aplicar para o Baja

- Target: Set up a codebook of size *K* where the training data of N values $\boldsymbol{x}(n)$ exhibits a minimum distance to the codebook vectors

$$d = \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0\cdots K-1} \{d(\boldsymbol{x}(n), \boldsymbol{c}_i)\}$$

usually: $d(\boldsymbol{x}(n), \boldsymbol{c}_i) = \|\boldsymbol{x}(n) - \boldsymbol{c}_i\|^2$

- Currently there is no method known which ensures to solve this problem optimally.

- The LBG algorithm is typically used which iteratively sets up a codebook of the desired size.

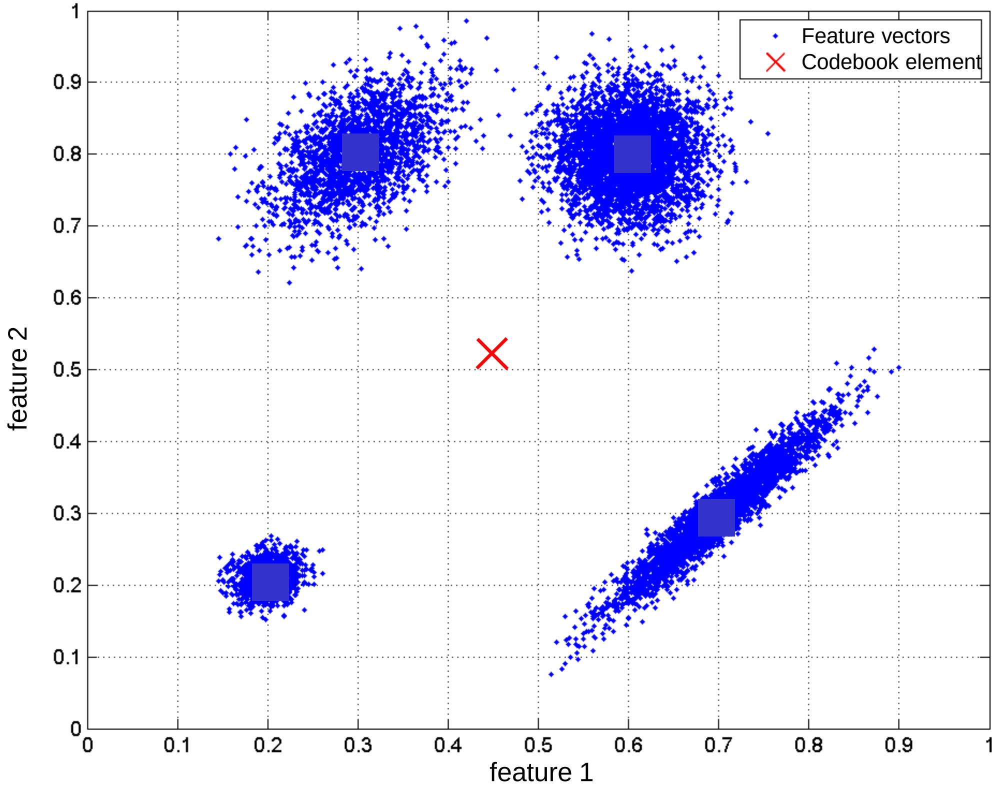
- LBG stands for the inventors of this algorithm: Linde, Buzo, and Gray.

- The LBG algorithm utilizes the k-means procedure.

Codebook training: LBG algorithm

- 1) Initialization: Start with a codebook with one element (mean of all training elements):

$$\boldsymbol{c}_0 = \frac{1}{N} \sum_{n=0}^{N-1} \boldsymbol{x}(n)$$



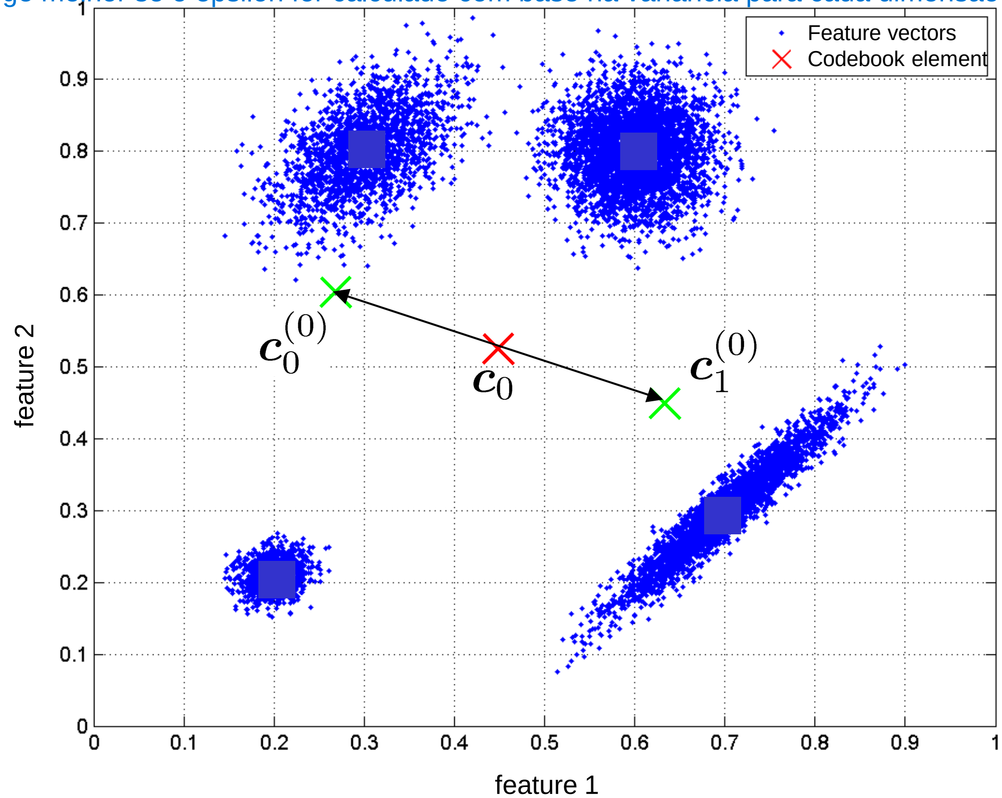
Codebook training: LBG algorithm

- 2) Splitting: Increase the codebook size (typically by a factor two):
  => Add and subtract a random vector to the codebook vector(s).

Converge melhor se o epsilon for calculado com base na variancia para cada dimensão

index of the k-means iteration

$$c_0 \rightarrow \begin{cases} c_0^{(0)} = c_0 + \epsilon_1 \\ c_1^{(0)} = c_0 - \epsilon_1 \end{cases}$$



Splitting in general:

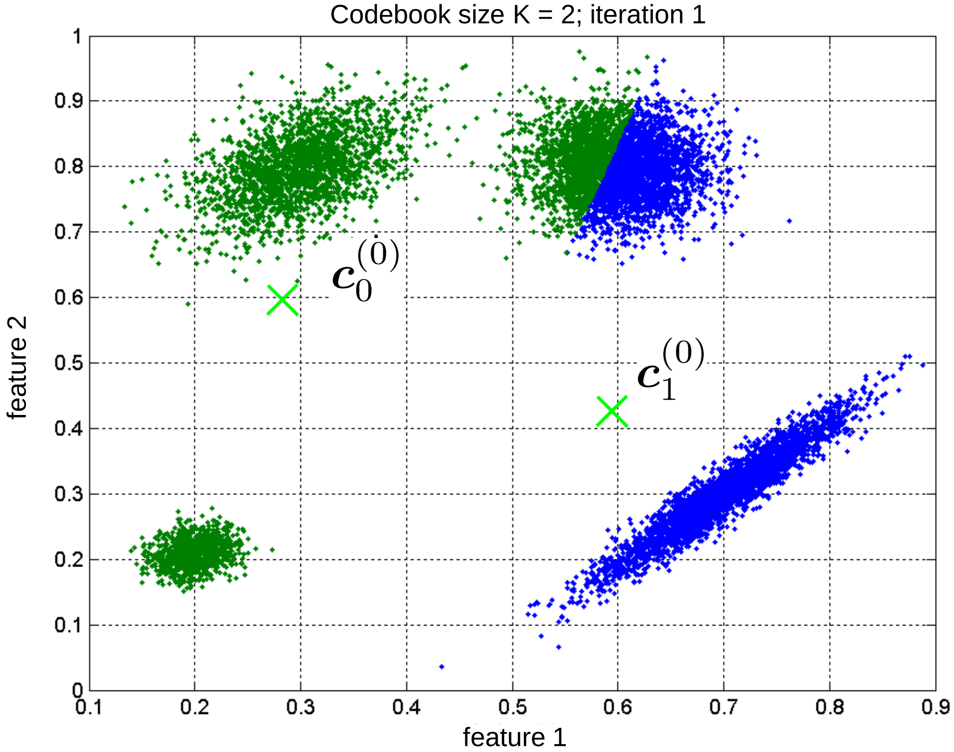
$$C_K = \left[c_0, c_1, \ldots, c_{K-1}\right]$$

$$C_{2K}^{(0)} = \left[(c_0, c_1, \ldots, c_{K-1}) + \epsilon_1, (c_0, c_1, \ldots, c_{K-1}) - \epsilon_1\right]$$

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K
  - a) **Assignment step:** Assign all elements to one codebook vector according to the smallest Euclidian distance

$$S_i^{(0)} : \left\{ \boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(0)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(0)}||^2 \ \forall j \in [0, K-1] \right\}$$



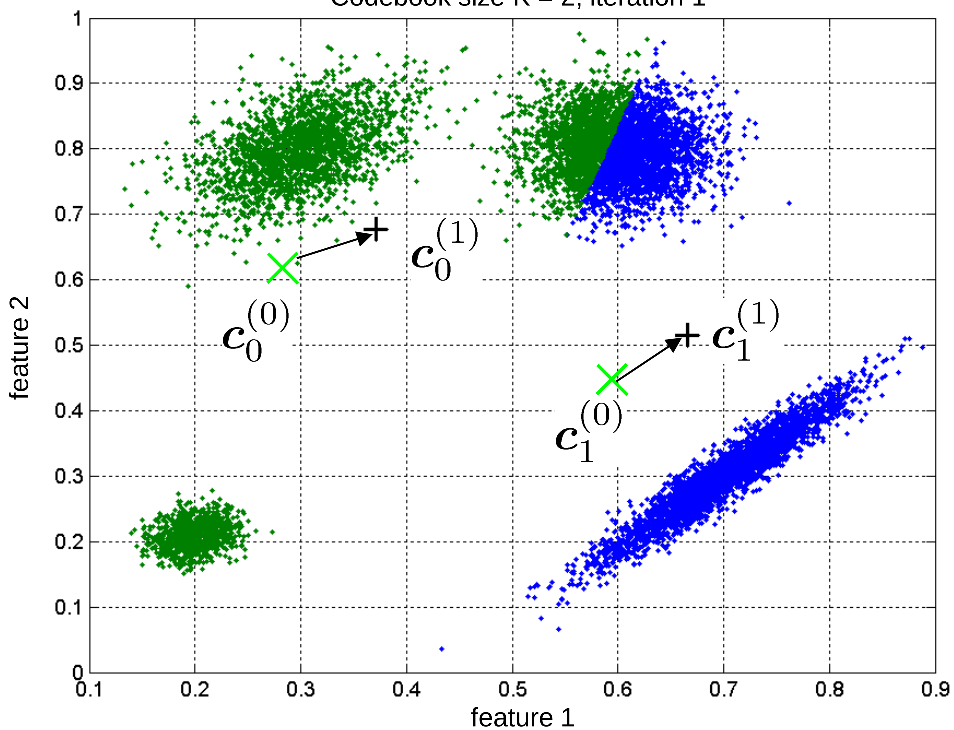
$S_0^{(0)}$: green elements

$S_1^{(0)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K
  - b) **Update step:** Compute new cluster centers (mean of the assigned values):

$$\boldsymbol{c}_i^{(1)} = \frac{1}{N_{S_i^{(0)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(0)}}}^{N-1} \boldsymbol{x}(n) \qquad \text{with:} \quad N_{S_i^{(0)}} = \sharp\{S_i^{(0)}\}$$



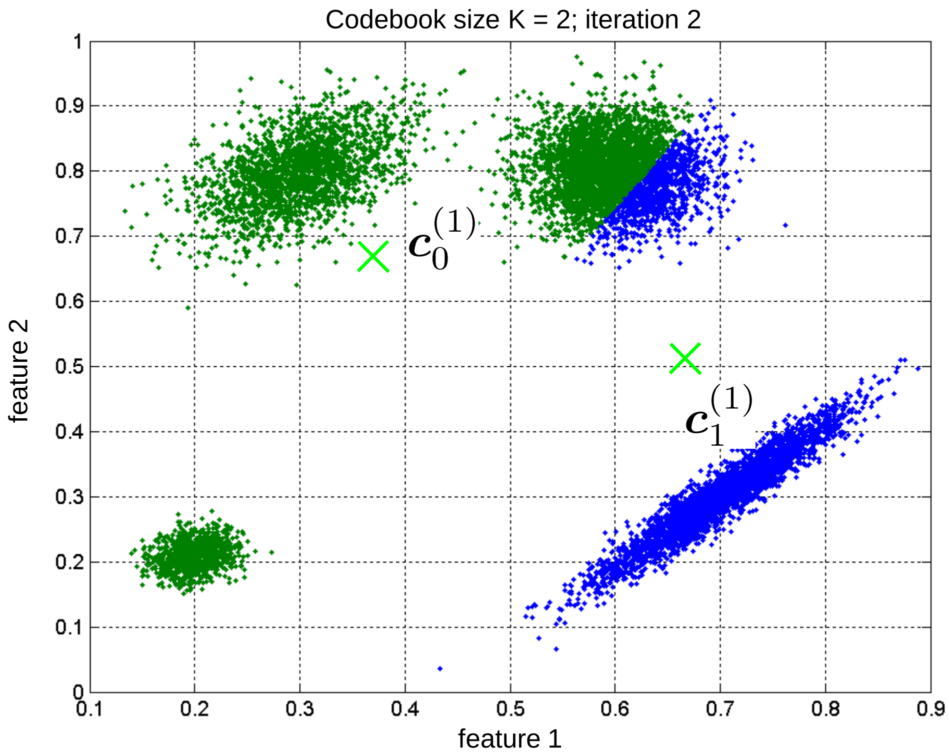
$S_0^{(0)}$: green elements

$S_1^{(0)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K
  - a) Assignment step: Assign all elements to one codebook vector according to the smallest Euclidian distance

$$S_i^{(1)} : \left\{ \boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(1)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(1)}||^2 \ \forall j \in [0, K-1] \right\}$$



$S_0^{(1)}$: green elements

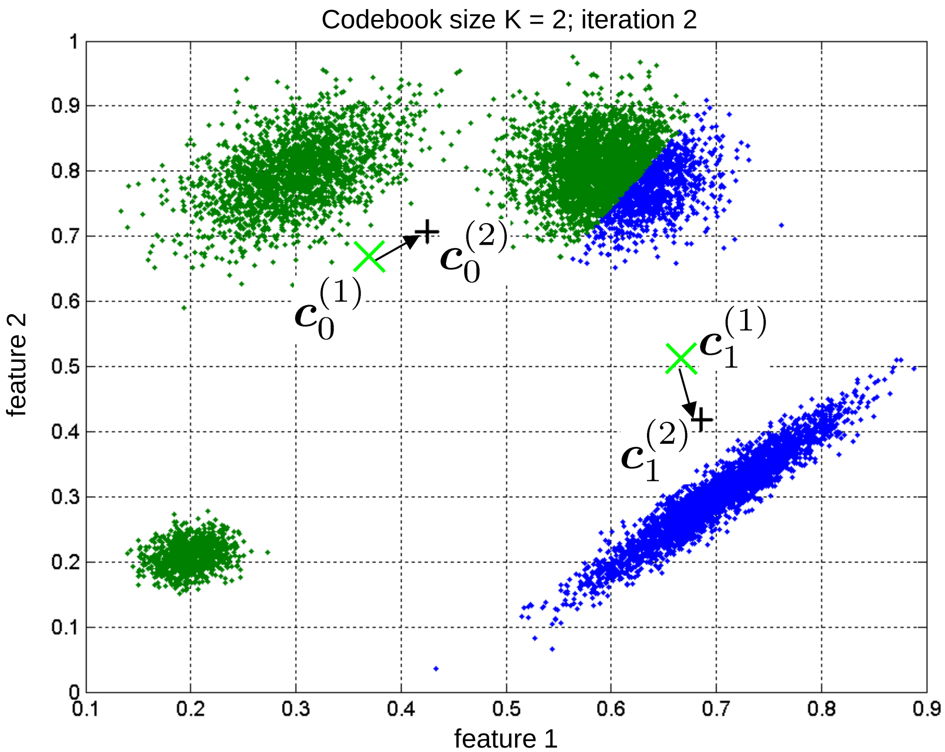
$S_1^{(1)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K
  - b) Update step: Compute new cluster centers:

$$\boldsymbol{c}_i^{(2)} = \frac{1}{N_{S_i^{(1)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(1)}}}^{N-1} \boldsymbol{x}(n)$$

with: $N_{S_i^{(1)}} = \sharp\{S_i^{(1)}\}$



$S_0^{(1)}$: green elements

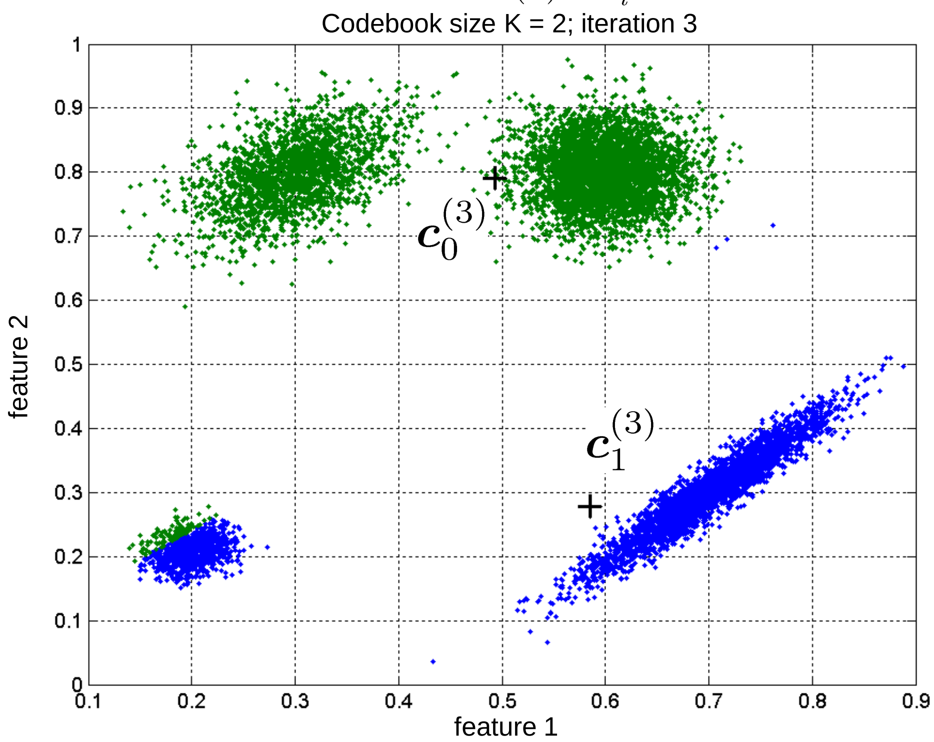
$S_1^{(1)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K

  a) Assignment step $$S_i^{(p)} : \left\{ \boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(p)}||^2 \ \forall j \in [0, K-1] \right\}$$

  b) Update step $$\boldsymbol{c}_i^{(p+1)} = \frac{1}{N_{S_i^{(p)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(p)}}}^{N-1} \boldsymbol{x}(n)$$ with: $$N_{S_i^{(p)}} = \sharp\{S_i^{(p)}\}$$



$S_0^{(2)}$: green elements

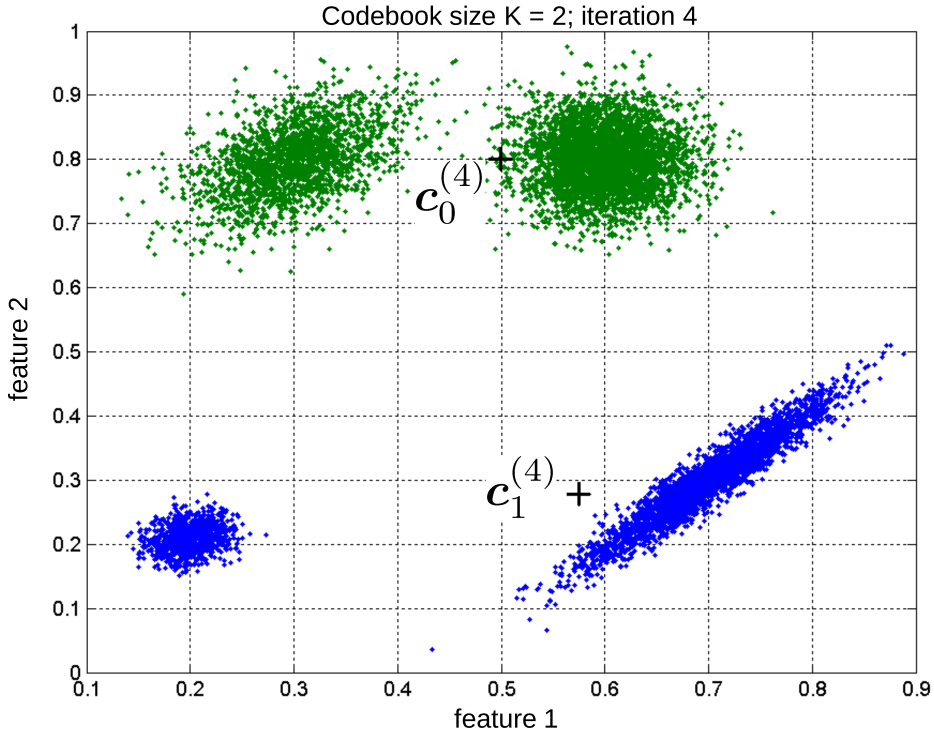
$S_1^{(2)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K

  a) Assignment step $$S_i^{(p)} : \left\{ \boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(p)}||^2 \; \forall j \in [0, K-1] \right\}$$

  b) Update step $$\boldsymbol{c}_i^{(p+1)} = \frac{1}{N_{S_i^{(p)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(p)}}}^{N-1} \boldsymbol{x}(n)$$ with: $$N_{S_i^{(p)}} = \sharp\{S_i^{(p)}\}$$



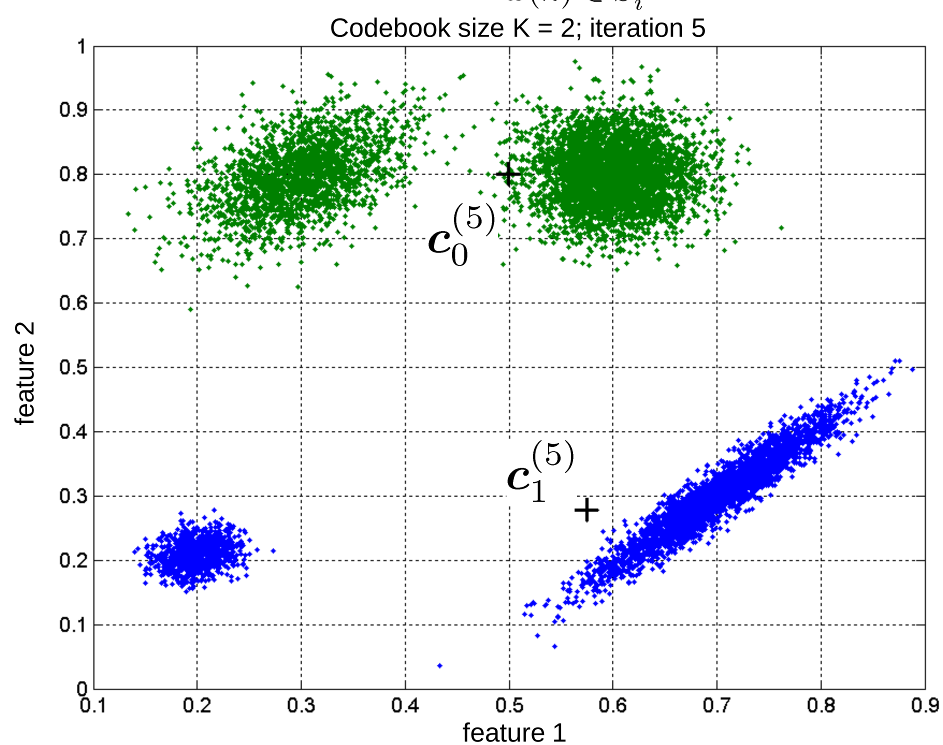
$S_0^{(3)}$: green elements

$S_1^{(3)}$: blue elements

Codebook training:
LBG algorithm using the k-means iteration

- 3) Iteration to optimize the codebook of size K
  - a) Assignment step $S_i^{(p)} : \left\{ \boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(p)}||^2 \; \forall j \in [0, K-1] \right\}$
  - b) Update step

$$\boldsymbol{c}_i^{(p+1)} = \frac{1}{N_{S_i^{(p)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(p)}}}^{N-1} \boldsymbol{x}(n) \qquad \text{with:} \quad N_{S_i^{(p)}} = \sharp\{S_i^{(p)}\}$$



$S_0^{(4)}$: green elements

$S_1^{(4)}$: blue elements

Codebook training:
LBG algorithm stop criterion for the k-means iteration

- Stop iteration when mean distortion changes only marginally any more

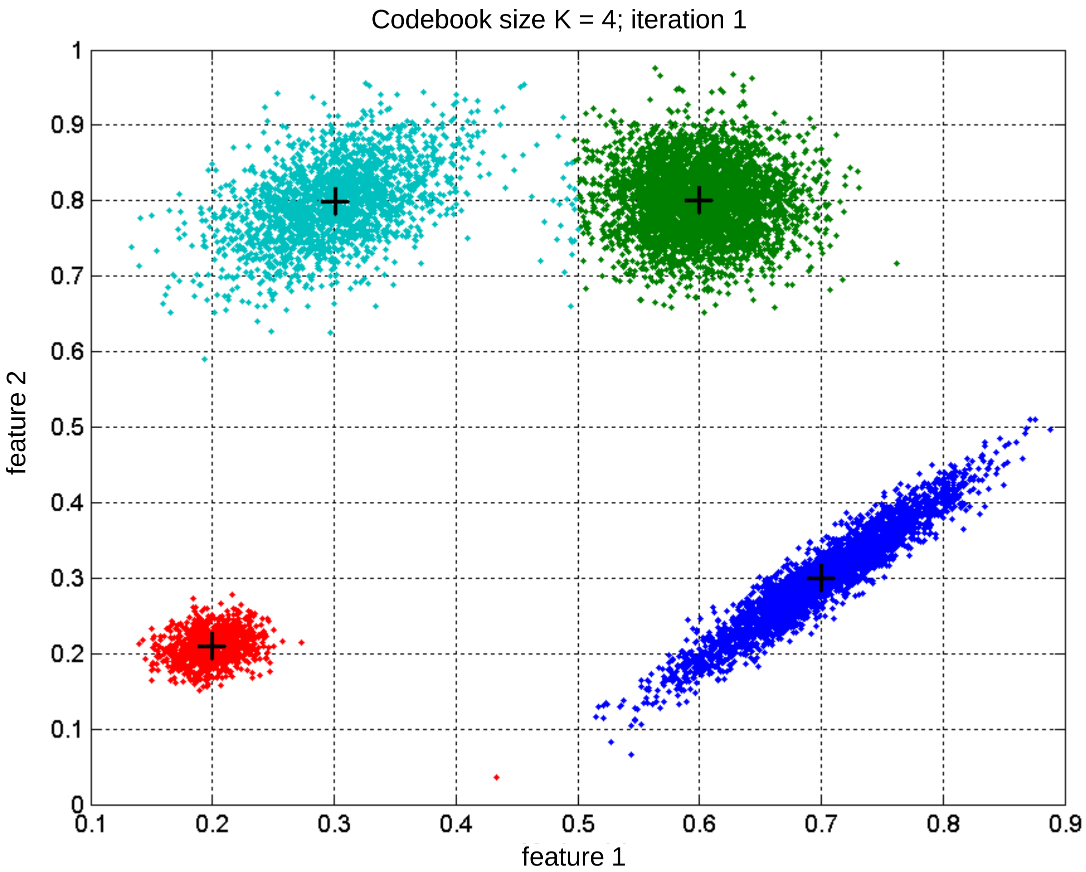
$$D^{(p)} = \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0\ldots K-1} ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2$$

$$\frac{D^{(p-1)} - D^{(p)}}{D^{(p-1)}} < d_{thr}$$

- Next step: Split the codebook vectors, i.e. goto 2) (page 9), double the size and proceed with the same steps.

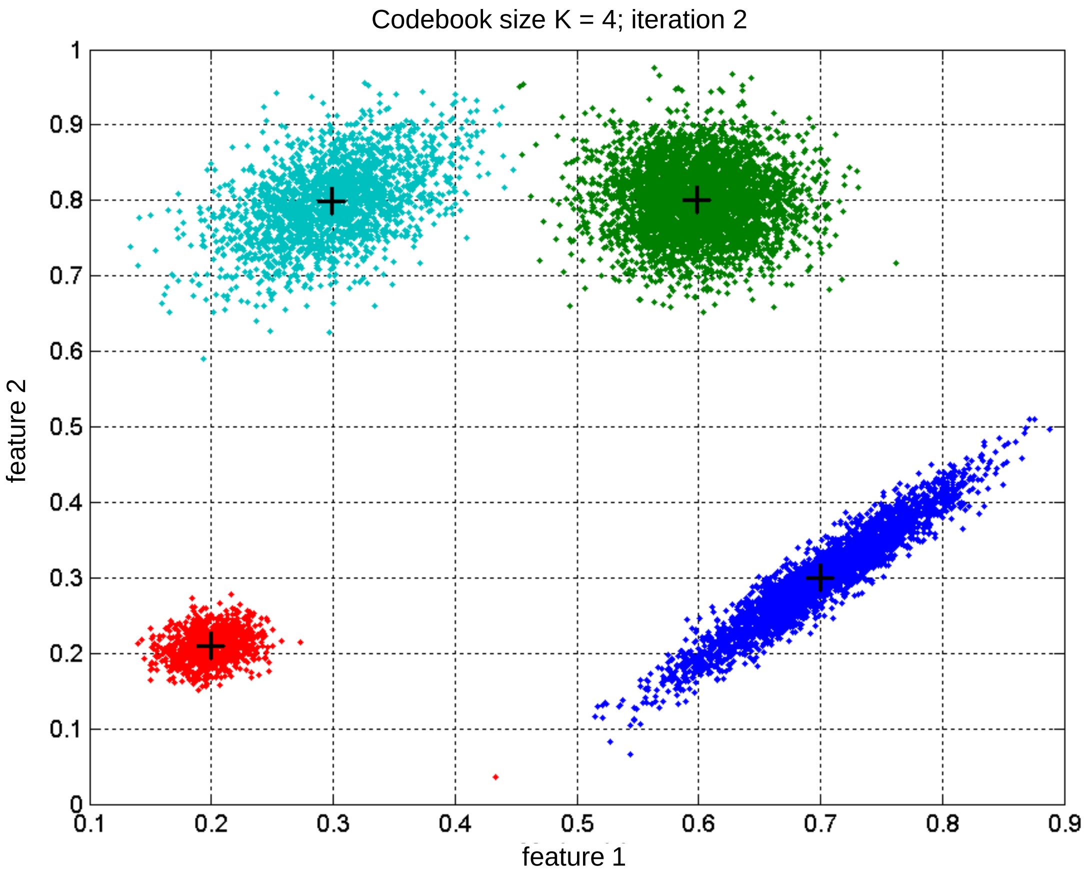
Codebook training:
LBG algorithm using the k-means iteration

- Codebook of size 4 after one iteration



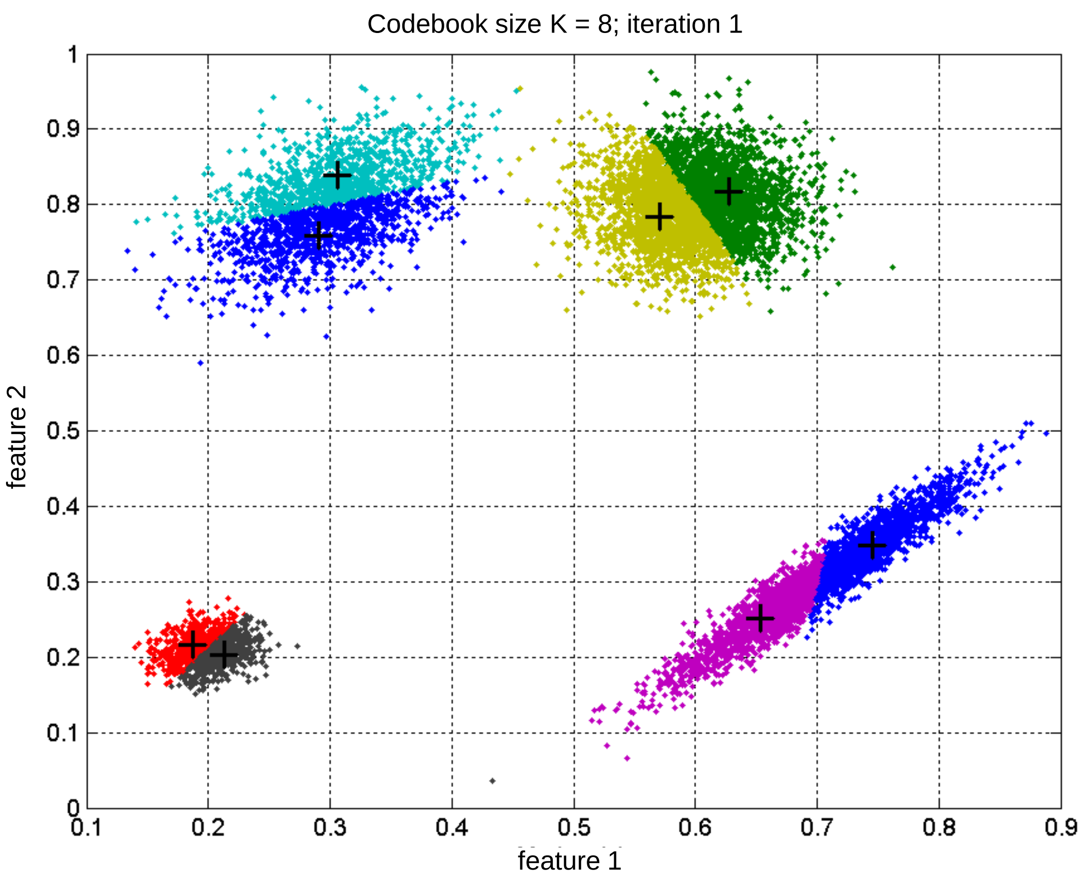
Codebook training:
LBG algorithm using the k-means iteration

- Codebook of size 4 after two iterations



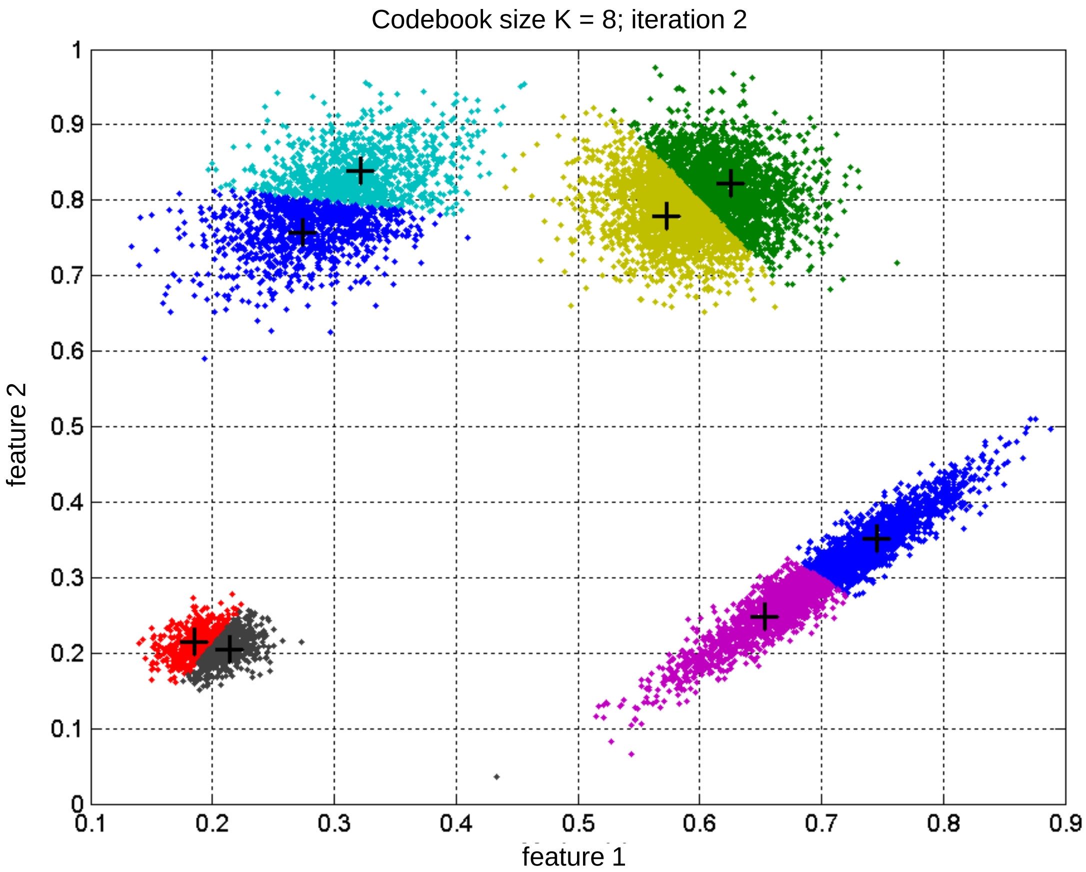
Codebook training:
LBG algorithm using the k-means iteration

- Codebook of size 8 after one iteration



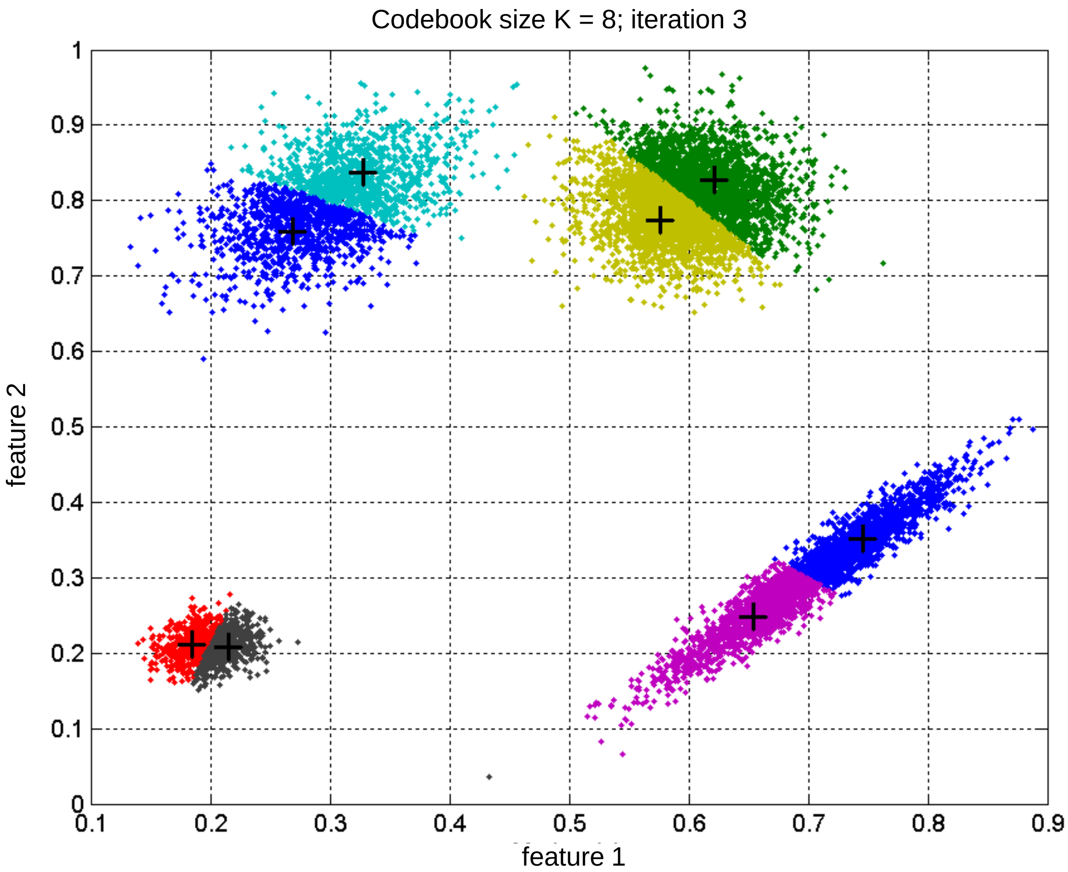
Codebook training: LBG algorithm using the k-means iteration

- Codebook of size 8 after two iterations



Codebook training:
LBG algorithm using the k-means iteration

- Codebook of size 8 after three iterations



Overview of the LBG algorithm

- 1) **Initialization**:

$$\boldsymbol{c}_0 = \frac{1}{N}\sum_{n=0}^{N-1} \boldsymbol{x}(n)$$

- 2) **Split the codebook**: $\boldsymbol{C}_K = \left[\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{K-1}\right]$

  to: $\boldsymbol{C}_{2K}^{(0)} = \left[(\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{K-1}) + \boldsymbol{\epsilon}_1, (\boldsymbol{c}_0, \boldsymbol{c}_1, \ldots, \boldsymbol{c}_{K-1}) - \boldsymbol{\epsilon}_1\right]$

  2K => K: $\boldsymbol{C}_K^{(0)} = \left[\boldsymbol{c}_0^{(0)}, \boldsymbol{c}_1^{(0)}, \ldots, \boldsymbol{c}_{K-1}^{(0)}\right]$ ← initial (0) estimate of the new enlarged codebook

- 3) **k-means iteration** (step index (p)) to optimize the codebook of size K

  a) Assignment step: $S_i^{(p)} : \left\{\boldsymbol{x}(n) : ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2 \leq ||\boldsymbol{x}(n) - \boldsymbol{c}_j^{(p)}||^2 \; \forall j \in [0, K-1]\right\}$

  b) Update step:

$$\boldsymbol{c}_i^{(p+1)} = \frac{1}{N_{S_i^{(p)}}} \sum_{\substack{n=0 \\ \boldsymbol{x}(n) \in S_i^{(p)}}}^{N-1} \boldsymbol{x}(n) \quad \text{with: } N_{S_i^{(p)}} = \sharp\{S_i^{(p)}\}$$

  c) Stop criterion:

$$\frac{D^{(p-1)} - D^{(p)}}{D^{(p-1)}} < d_{thr} \quad \text{with: } D^{(p)} = \frac{1}{N}\sum_{n=0}^{N-1} \min_{i=0\ldots K-1} ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2$$

- 4) **Increase codebook size with step 2)**

Training and evaluation steps

- Having a data base of vectors (of measurement data) it is important to separate data for codebook training and codebook evaluation.

- Typically an 80/20 split is performed: 80% of the data is used for training and 20% for the evaluation.

  avoid overfitting

- The same distance as for the training has to be used for the evaluation:

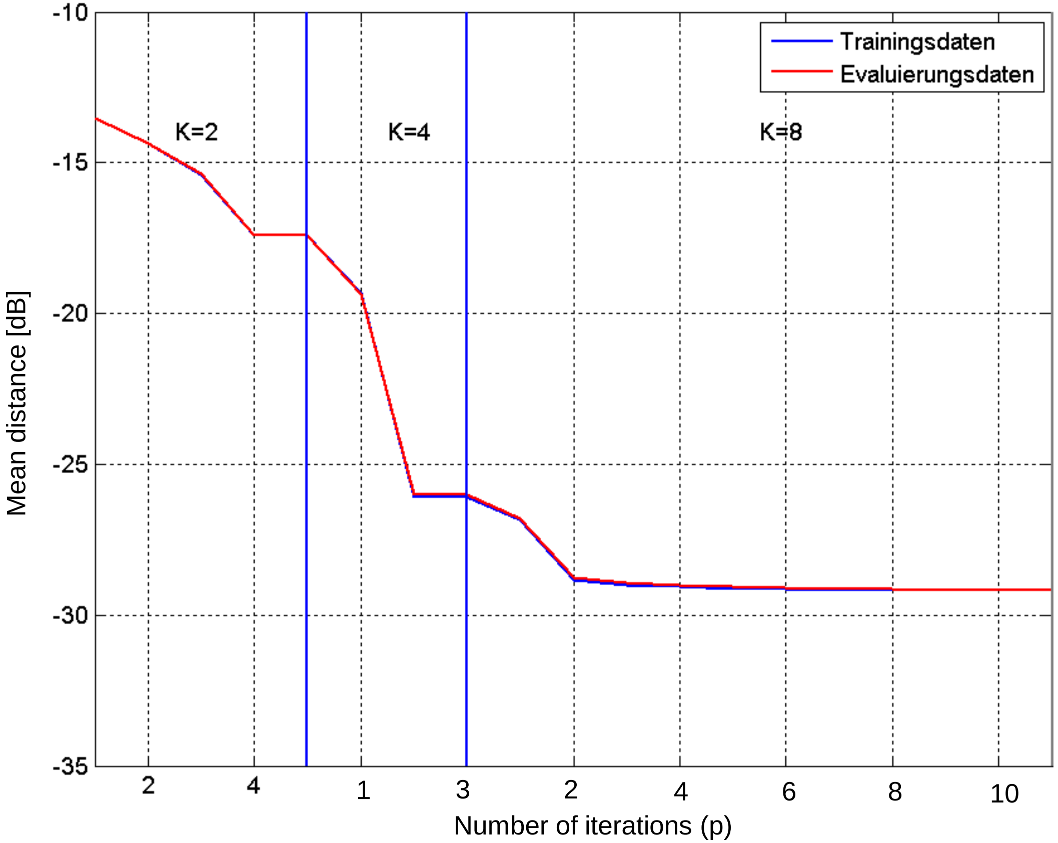
$$D_{\text{eval}}^{(p)} = \frac{1}{N_{\text{eval}}} \sum_{n=0}^{N_{\text{eval}}-1} \min_{i=0\ldots K-1} ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2$$

- In case of a well-trained codebook, no big difference should occur between the training distance $D^{(p)}$ and the evaluation distance $D_{\text{eval}}^{(p)}$

Final evaluation

Evolution over the mean distance over the iterations (p) and the codebook size (K) :

$$D^{(p)} = \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0 \dots K-1} ||\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}||^2$$



Cost functions for the codebook training

- For the codebook training and the evaluation, the same cost function should be utilized, as it is also used when applying the codebook.
- A computational efficient cost function should be utilized.
- Typically, squared distance or magnitude distance measures should be applied.
- A weighting of the different vector element can make sense dependent on the importance of the specific vector elements.
  - Square distance with weighting:

$$D_{\mathrm{w,quad}}^{(p)} = \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0\ldots K-1} \left[\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}\right]^{\mathrm{T}} \boldsymbol{G} \left[\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}\right]$$

  - Magnitude distance with weighting:

$$D_{\mathrm{w,lin}}^{(p)} = \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0\ldots K-1} \left\|\boldsymbol{G} \left[\boldsymbol{x}(n) - \boldsymbol{c}_i^{(p)}\right]\right\|$$

weights:

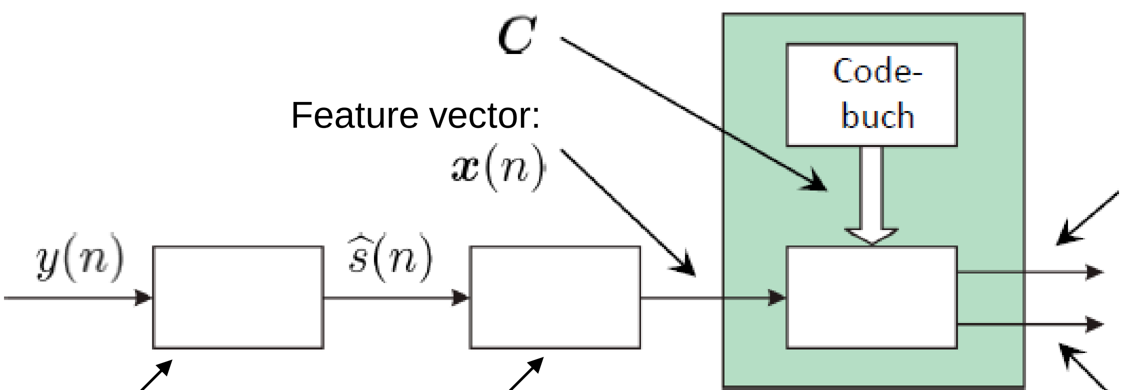
$$\boldsymbol{G} = \begin{bmatrix} g_0 & 0 & 0 & \dots & 0 \\ 0 & g_1 & 0 & \dots & 0 \\ 0 & 0 & g_2 & & 0 \\ \vdots & \vdots & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & g_{M-1} \end{bmatrix}^{\mathrm{T}}$$

Application examples (I): General codebook search

❑ Basic structure of a codebook search:

Matrix of codebook vectors:

Feature vector:



Minimum distance of the current feature vector and the „best“ codebook entry:

$$d_{\text{opt}}(n) = ||\boldsymbol{x}(n) - \boldsymbol{c}^{(p)}_{i,\text{opt}(n)}||^2$$

$i_{\text{opt}}(n)$

Index (address) of the „best“ codebook entry.

Pre-processing: for interference cancellation, e.g., noise reduction, beamforming, etc.

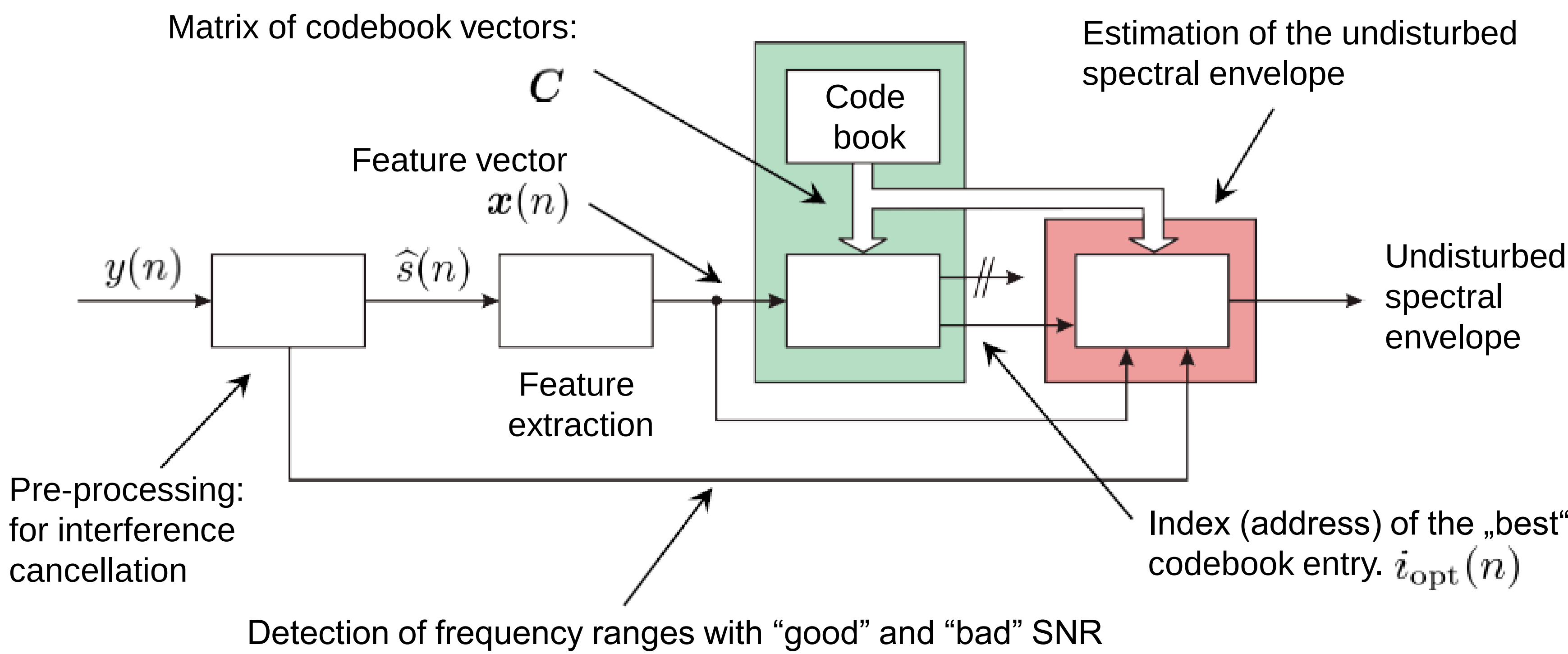
Feature extraction: e.g., prediction coefficients for a parametric spectral description.

Application examples (II): Speaker recognition

- **Topic of one separate lecture…**

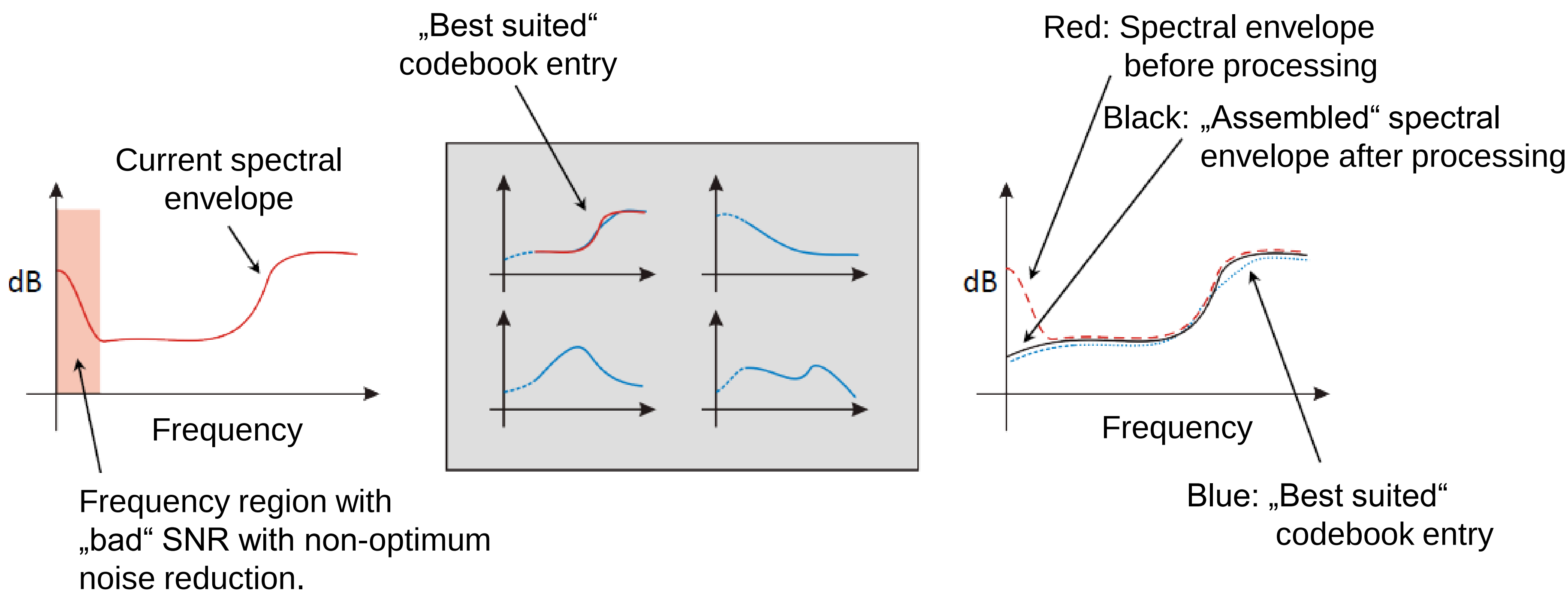
Application examples (III): Signal reconstruction

- **Reconstruction of target signal spectral envelopes with “bad” SNR:**



Application examples (III): Signal reconstruction

❑ Signal reconstruction - Basic principle:



Application examples (III): Signal reconstruction

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Signal reconstruction - Basic principle:**
  - First, a typical noise reduction is applied. Frequency regions with „bad“, i.e., very low SNR are determined. They are identified by checking each frequency component for a continuous attenuation of the noise reduction system. I.e., frequency components where no target signal component is detected. Que???????????
  - The spectral envelope of the signal after noise reduction is determined.
  - The corresponding codebook spectral envelope is determined with the lowest distance to the current signal spectral envelope – in the frequency regions with „good“ SNR
  - The frequency regions of „bad“ SNR have to be disregarded by setting the weights of the distance function to zero.
  - Finally, the extracted envelope replaces the frequency components with “bad” SNR. The other frequency components of the signal after noise reduction are not modified.

Other application examples

- **Other application examples:**

  - **Audio signal coding** (source coding)
    => will be a separate issue in the next lecture.

  - **Classification of speech sound** (vowels, fricatives, etc.): For each sound group, several codebook entries are trained which allows a nice classification. This allows a different processing of the speech sounds, e.g., for hearing aids where fricatives are typically more difficult to be perceived correctly by hearing impaired persons.

  - **Parametric speech quality assessment**:
    Comparing signal envelopes of processed signals with envelopes of clean reference signals.

Efficient codebook search

- **Target:**
  Find the best codebook entry (with smallest distance) for an input vector which has to be quantized.

- **Problem:**
  Computationally demanding since the distances to all codebook entries have to be calculated in order to find the codebook entry with the minimum distance.

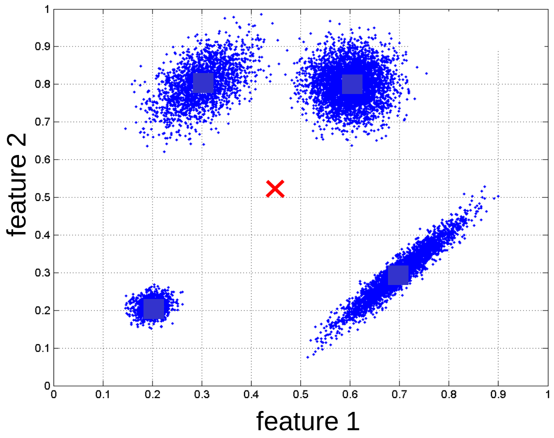
- => Analyze different procedures for an efficient search of the best codebook entry.

  => Tree-structured procedures where the tree is set-up by the different codebook levels.

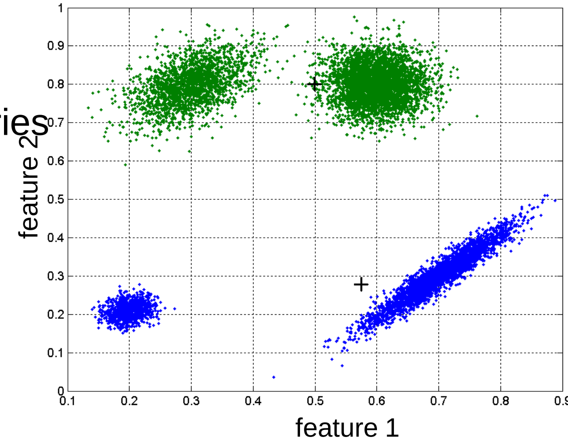
Efficient codebook search

- Review of the levels when generating the codebook:

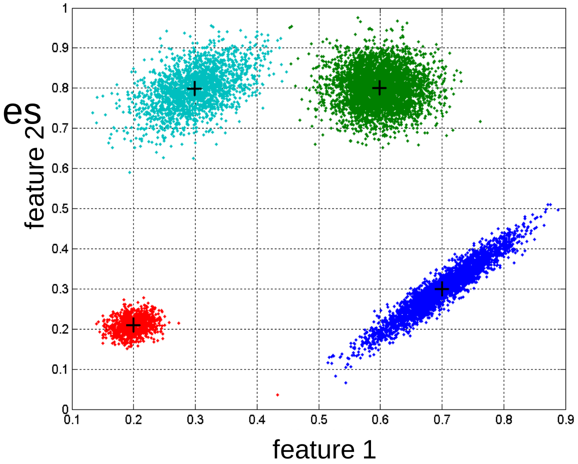
1. Level:
   one CB entry



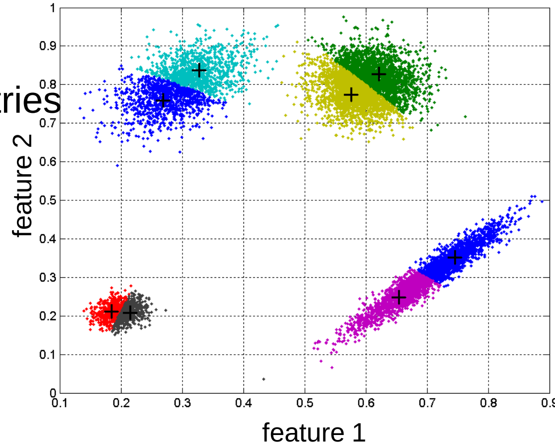
2. Level:
   two CB entries



3. Level:
   four CB entries

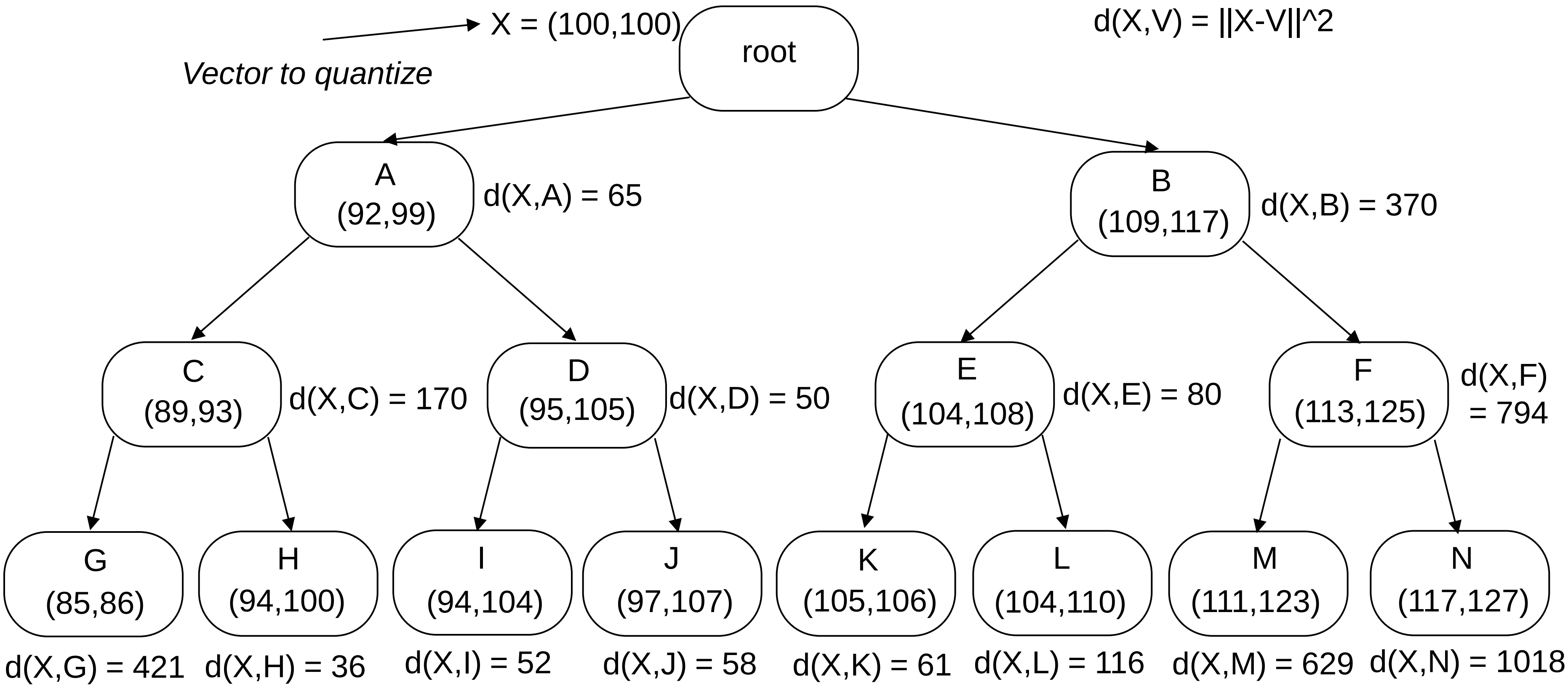


4. Level:
   eight CB entries



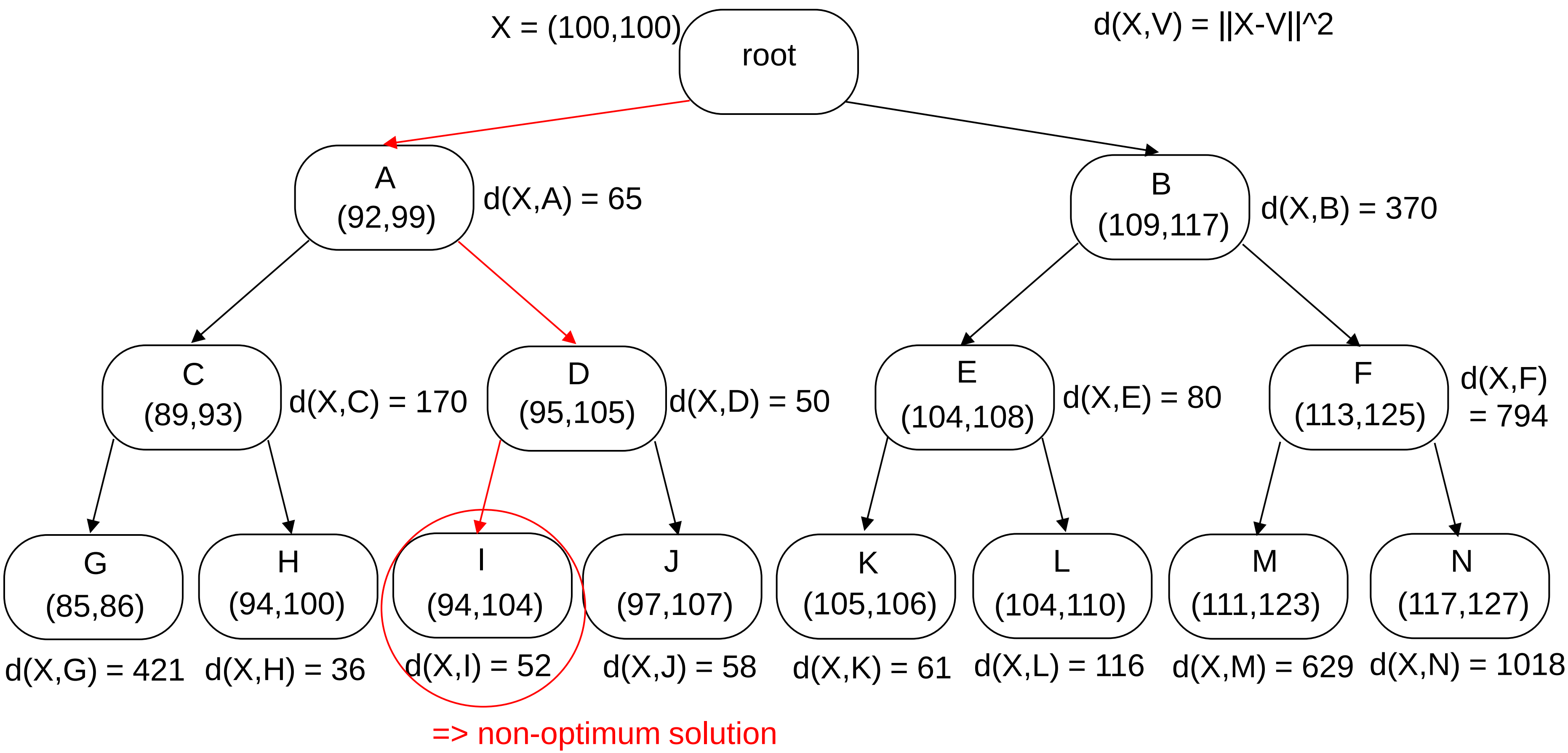
Tree-structured vector quantization (TSVQ) diagram

- The different levels can be designed in the “Tree-structured vector quantization” (TSVQ) diagram:



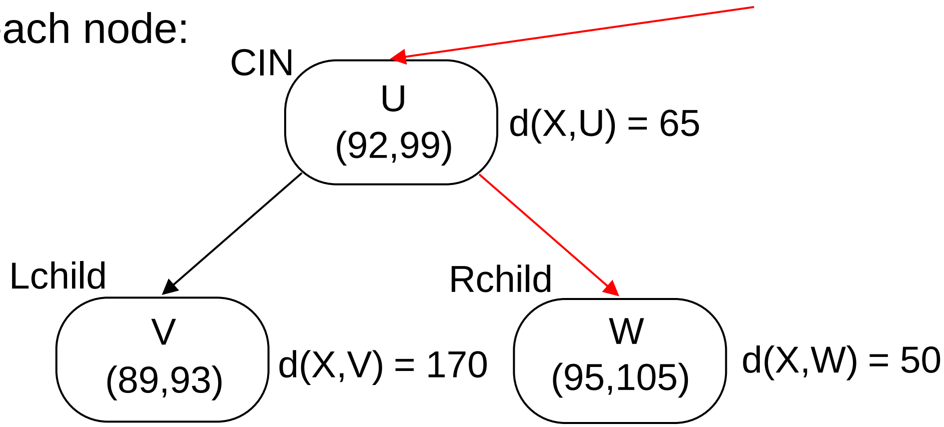
Single-path (SP) - TSVQ

- Look for the nearest child node:



Dynamic-path (DP) - TSVQ

- **Concept**: Decide at each node if one or two path will be followed.
- **Decision criterion**:
  Comparison of the mean square errors at each node:



Definitions:
- **CIN**: current initial node
- **Lchild(CIN)**: left child node of CIN
- **Rchild(CIN)**: right child node of CIN

- **Critical function:**

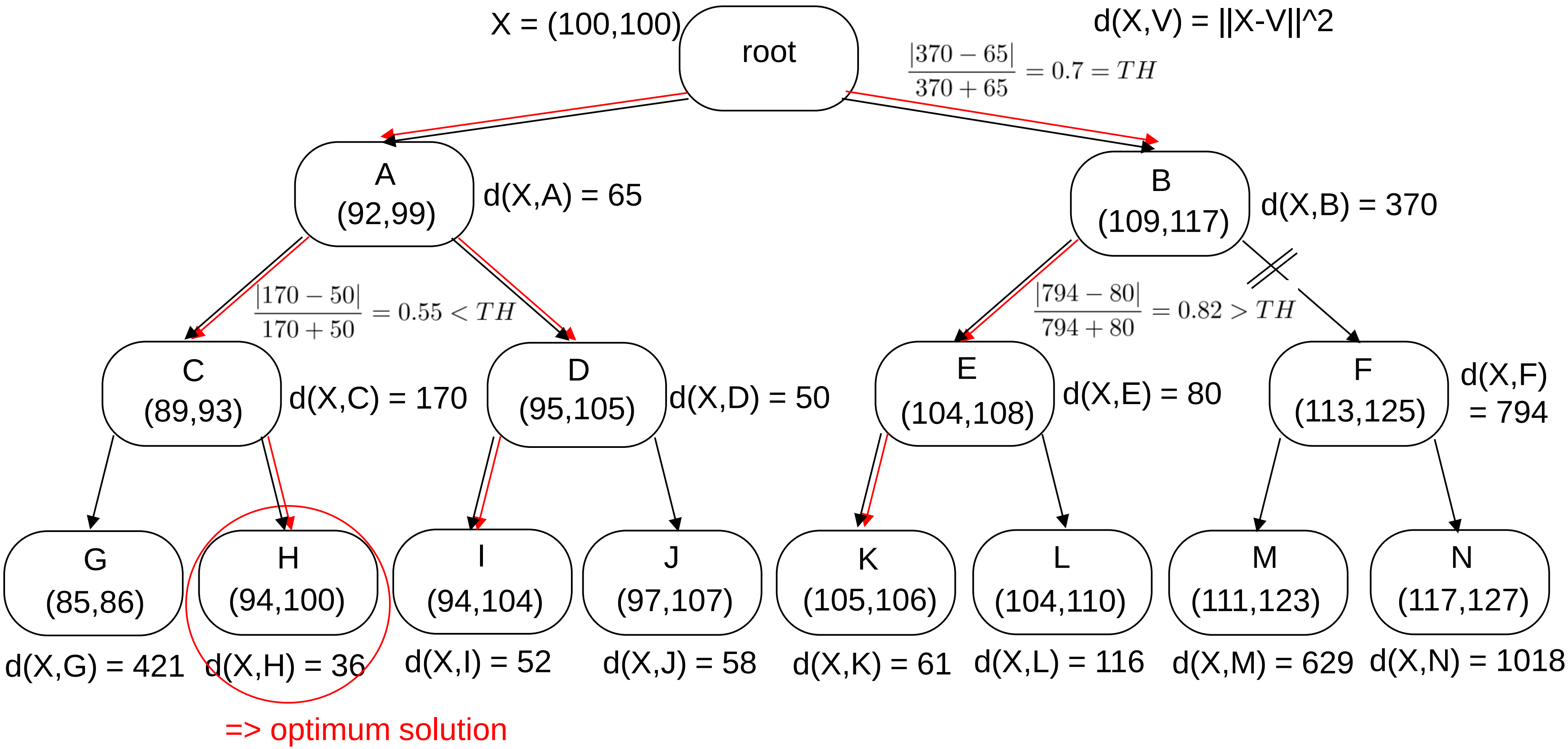
$$F(X, CIN) = \frac{|d(X, Lchild(CIN) - d(X, Rchild(CIN)|}{d(X, Lchild(CIN) + d(X, Rchild(CIN))}$$

- The larger the critical function is, the larger is the difference between the distances of X and the children. The probability that the node with the larger distance
  leads to the closest codeword is **rather low** when **the critical function is high** (close to 1).
  => only selection of the node with the lower distance:
- Current example:

$$F(X = (100, 100), CIN = U) = \frac{|170 - 50|}{170 + 50} = 0.55$$

Dynamic-path (DP) - TSVQ

- Comparison of the critical function with a threshold $F(X, CIN) < TH$ => select both children
  Typical threshold: *TH = 0.7*



Dynamic-path (DP) - TSVQ

TECHNISCHE UNIVERSITÄT DARMSTADT

- **Advantage of the DP-TSVQ:**

  Threshold allows balance between complexity and the probability to find the globally optimal quantization value.

- **However,**

  there is no guarantee to find the global optimum, unless the threshold is set to 1. => Then the complexity is higher than for a full search.

- **Target:**

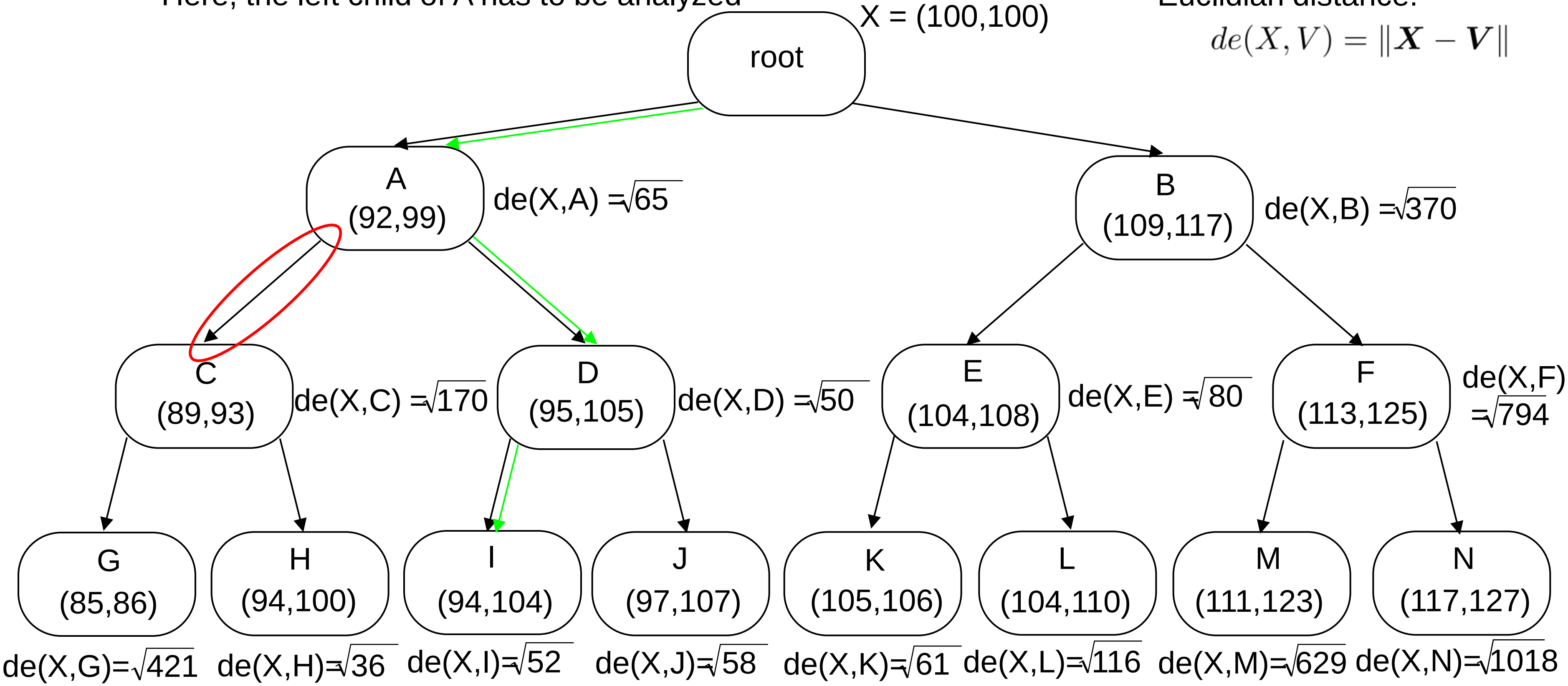
  Find an efficient way to find the global optimum =>
  "Full search equivalent" (FSE) – TSVQ.

Full search equivalent (FSE) - TSVQ

1. step: perform a single-path TSVQ (green path) => find a local optimum: LC
2. step: walk back the tree and analyse the branches
   Here, the left child of A has to be analyzed

Euclidian distance:

$$de(X,V) = \|\boldsymbol{X} - \boldsymbol{V}\|$$

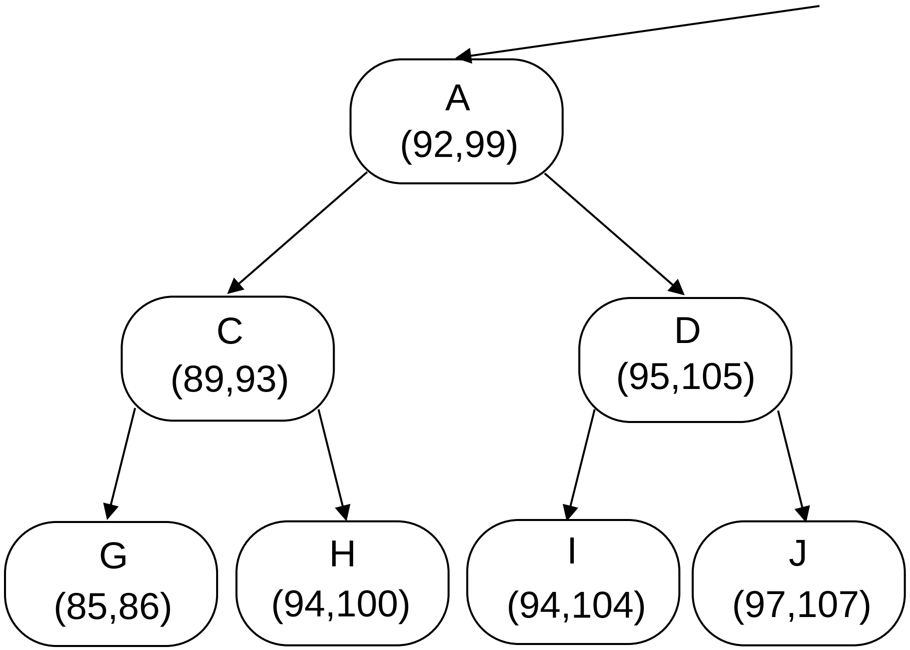


Full search equivalent (FSE) - TSVQ

- **FSE-TSVQ:**

  For the analyses the “radius” of each node has to be determined.
  This radius is independent of the data value to quantized
  => The “radius” can be determined once when the codebook is generated.

  The “radius” of a node is the max. euclidian distance of all nodes “below” the node under investigation and the current node:



- Examples on the left:

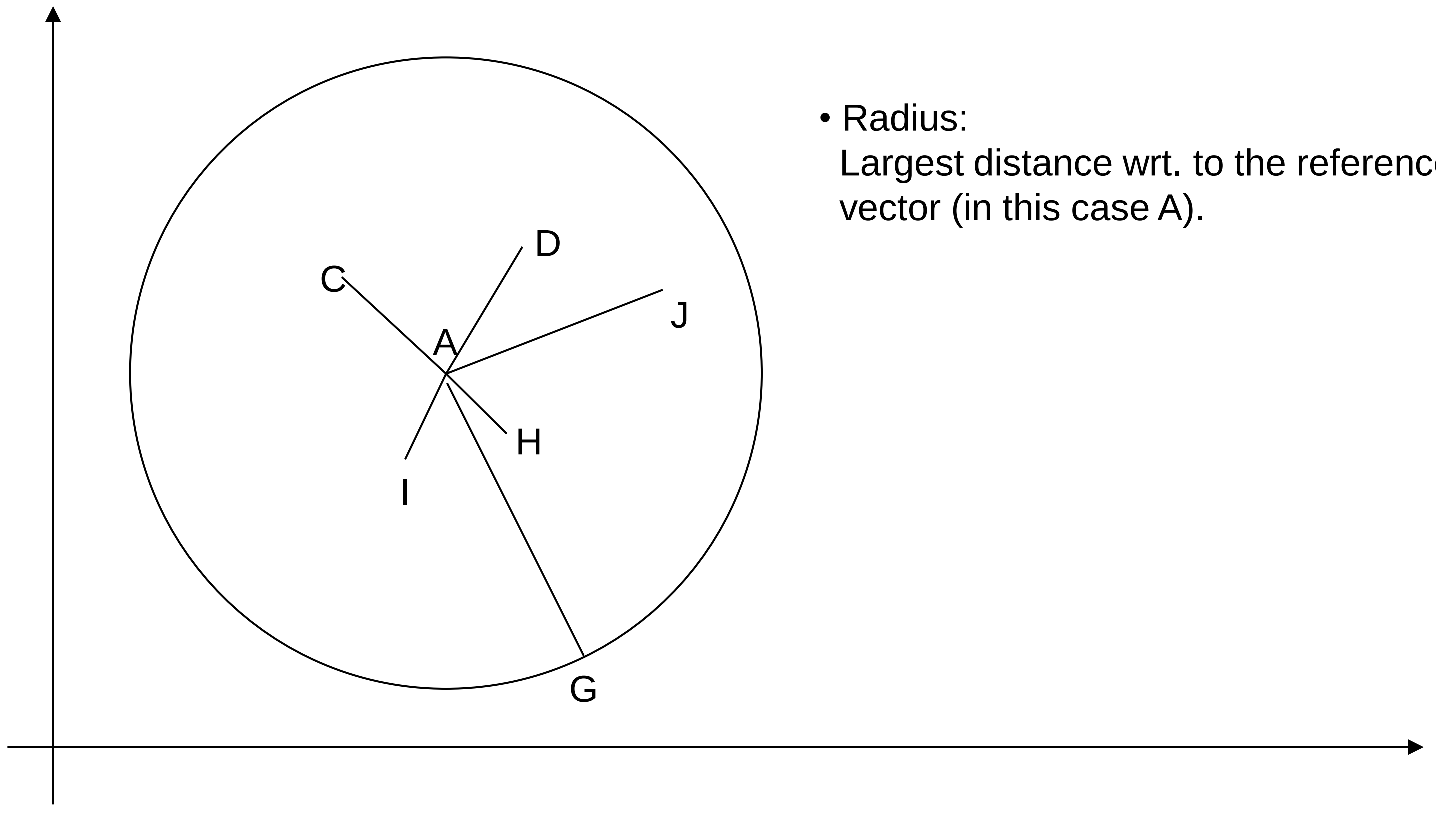
$$\begin{aligned}
de(A,C) &= \sqrt{3^2+6^2} &&= \sqrt{45}\\
de(A,D) &= \sqrt{3^2+6^2} &&= \sqrt{45}\\
de(A,G) &= \sqrt{7^2+13^2} &&= \sqrt{218}\\
de(A,H) &= \sqrt{2^2+1^2} &&= \sqrt{5}\\
de(A,I) &= \sqrt{2^2+5^2} &&= \sqrt{29}\\
de(A,J) &= \sqrt{5^2+8^2} &&= \sqrt{89}
\end{aligned}$$

radius of A

$$\begin{aligned}
de(C,G) &= \sqrt{4^2+7^2} &&= \sqrt{65}\\
de(C,H) &= \sqrt{5^2+7^2} &&= \sqrt{74}
\end{aligned}$$

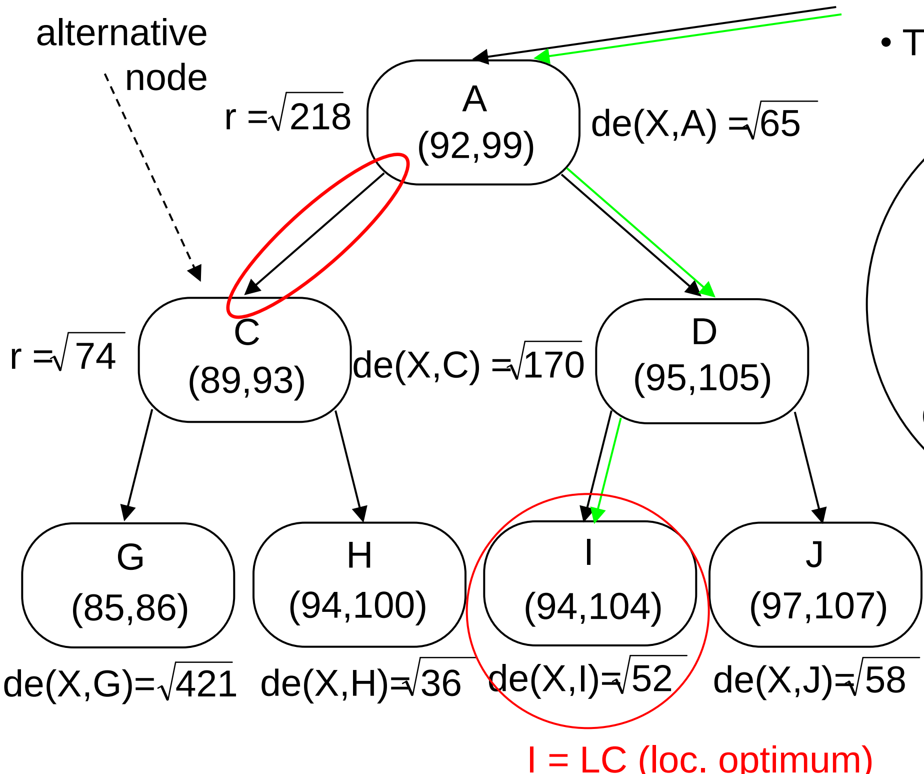
radius of C

Full search equivalent (FSE) – Definition of radius

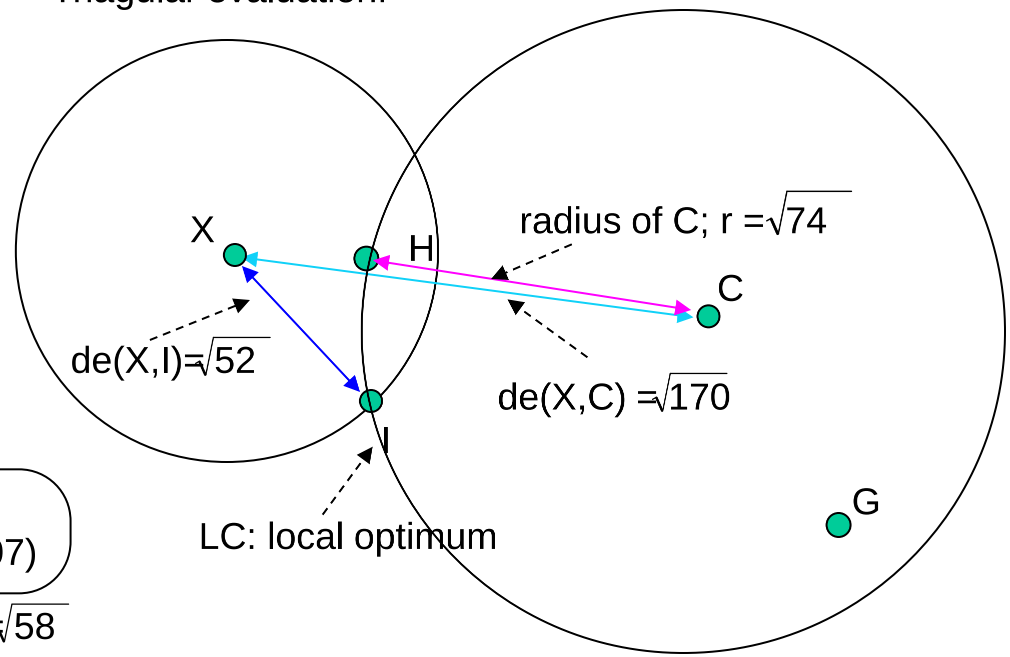


- Radius:
  Largest distance wrt. to the reference vector (in this case A).

Full search equivalent (FSE) - TSVQ



• Triagular evaluation:



- **Alternative 1:**

de(X,LC) + radius (alt. node) <= de(X, alt. node)

=> circles do not intersect => no better solution than the LC can be found under the alternative node

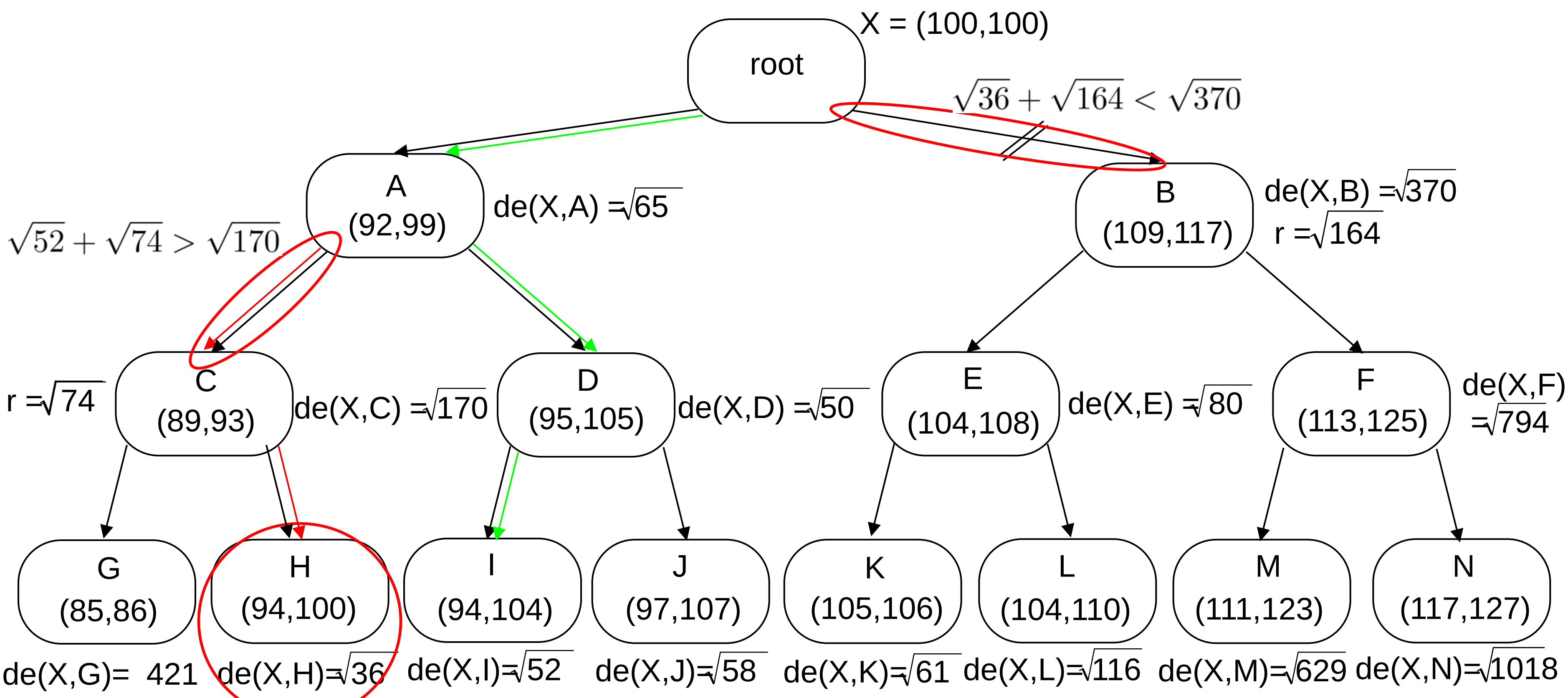
- **Alternative 2:**

de(X,LC) + radius (alt. node) > de(X, alt. node)

=> circles intersect => possible better solution than LC can be found under the alternative node

Full search equivalent (FSE) - TSVQ

1. step: perform a single-path TSVQ (green path) => find a local optimum: LC = I (green path)
2. step: walk back the tree and analyse the branches
   First, the left child of A has to be analyzed => new branch (red), new LC = H
   Then, the right child of the root has to be analyzed => no new branch



Enhanced Dynamic Path (EDP) - TSVQ

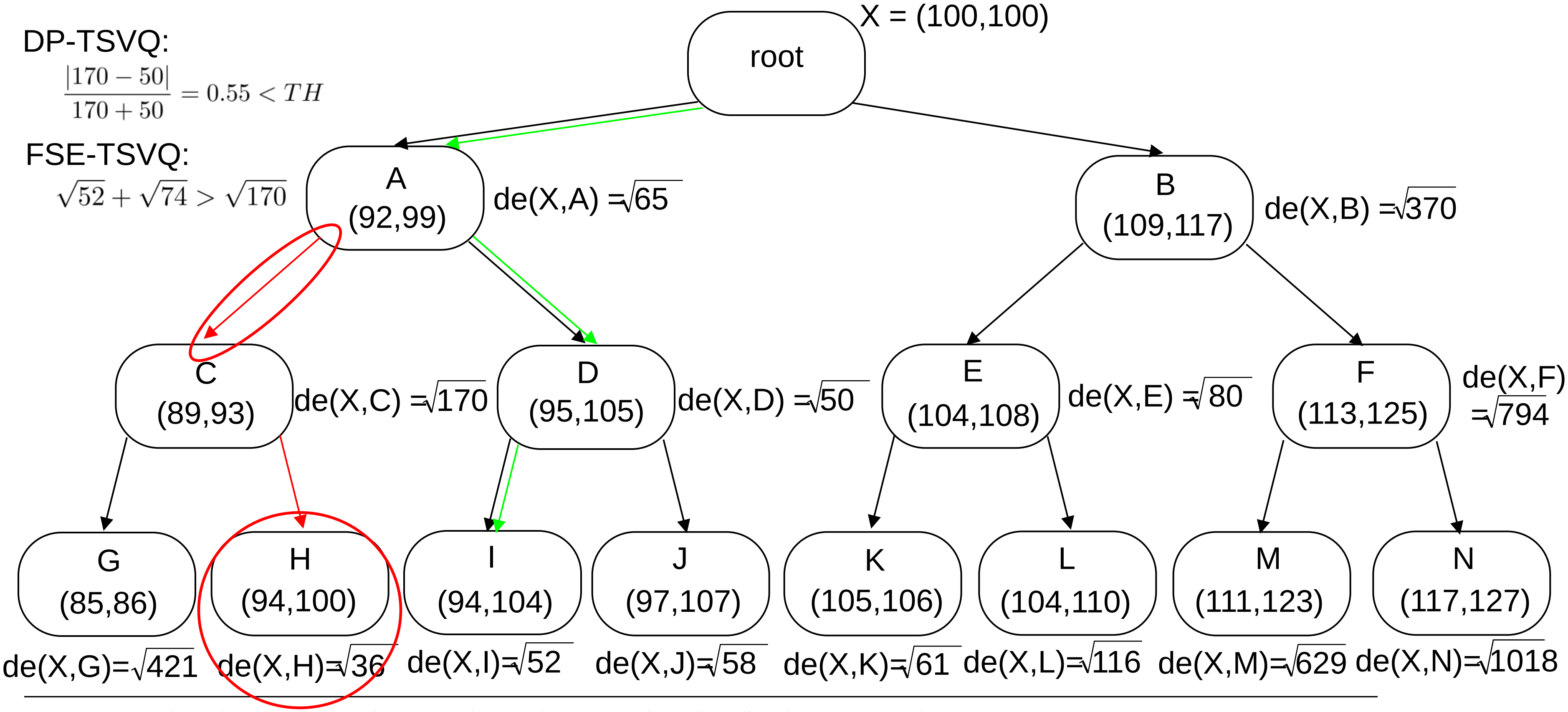
- Combination of DP-TSVQ and FSE-TSVQ:
  trade-off between computational complexity and quantization quality
- The conditions of both methods have to be fulfilled before opening another branch
  (no change in the current example, both conditions are fulfilled at node A)

DP-TSVQ:

$$\frac{|170-50|}{170+50} = 0.55 < TH$$

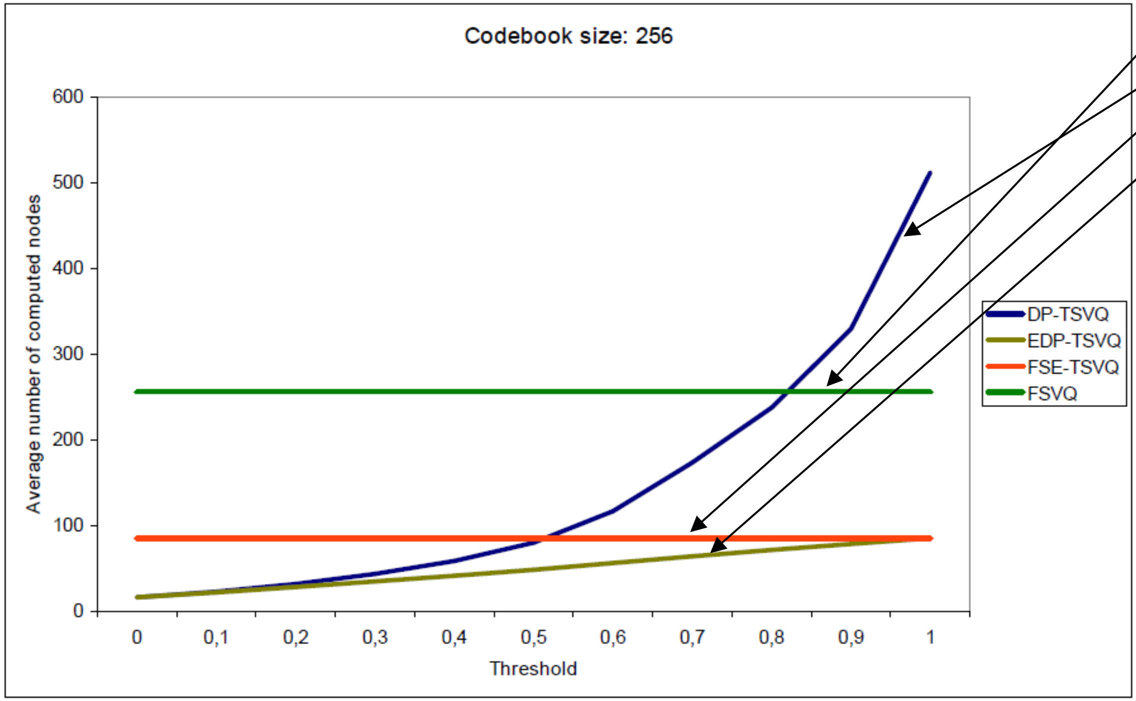
FSE-TSVQ:

$$\sqrt{52}+\sqrt{74} > \sqrt{170}$$



Comparing the complexity of the TSVQ methods

- Analysis with respect to the threshold TH which occurs in the DP-TSVQ and in the combined version (EDP-TSVQ).
- The FSVQ and the FSE-TSVQ are independent of the threshold
- The EDP-TSVQ guarantees the lowest complexity, however no guarantee to find to global optimum.



FSVQ: Full search vector codebook
DP-TSVQ: Dynamic path
FSE-TSVQ: Full search equivalent
EDP-TSVQ: Enhanced dynamic path

Different possible methods can be applied according to the individual optimization criterion.
=> Optimization between computational complexity and quality of the CB search!

Summary & Outlook

- Vector quantization for an efficient coding of vectors.
- Quantization according to the data distribution performed based on training data.
- The LBG algorithm (iteration) was introduced consisting of
  - An initialization
  - Step 1: A split of the codebook vectors
  - Step 2: An iterative better allocation procedure (k-means)
- Application examples were discussed
- We learned about different methods to find the best quantized codebook entry (quantized value) for a data vector sample.
- **Next week:** Audio coding, part I.