

Pattern Recognition

Part 5: Codebook Training

Gerhard Schmidt

Christian-Albrechts-Universität zu Kiel
Faculty of Engineering
Institute of Electrical and Information Engineering
Digital Signal Processing and System Theory



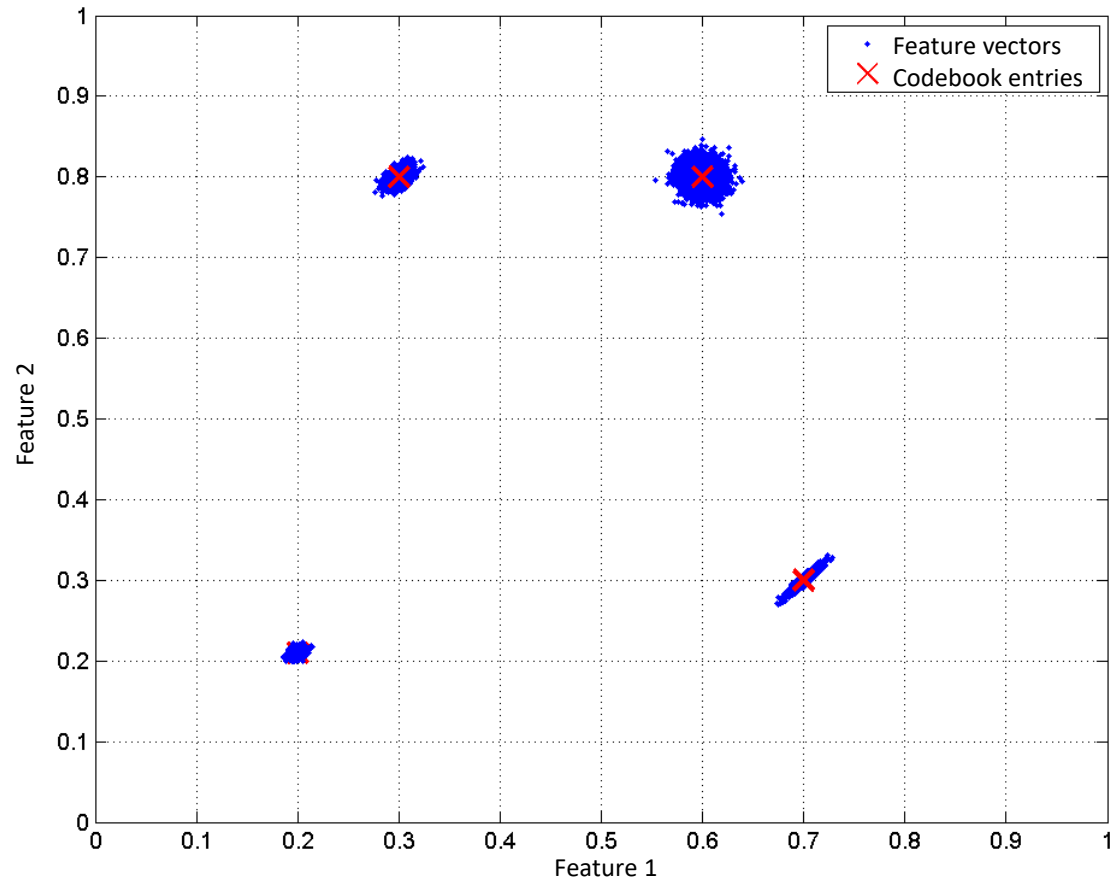
Contents

- ❑ Motivation
- ❑ Application examples
- ❑ Cost function for the training of a codebook
- ❑ LBG- and k-means algorithm
 - ❑ Basic schemes
 - ❑ Extensions
- ❑ Combination with additional mapping schemes



Introduction and Motivation – Part 1

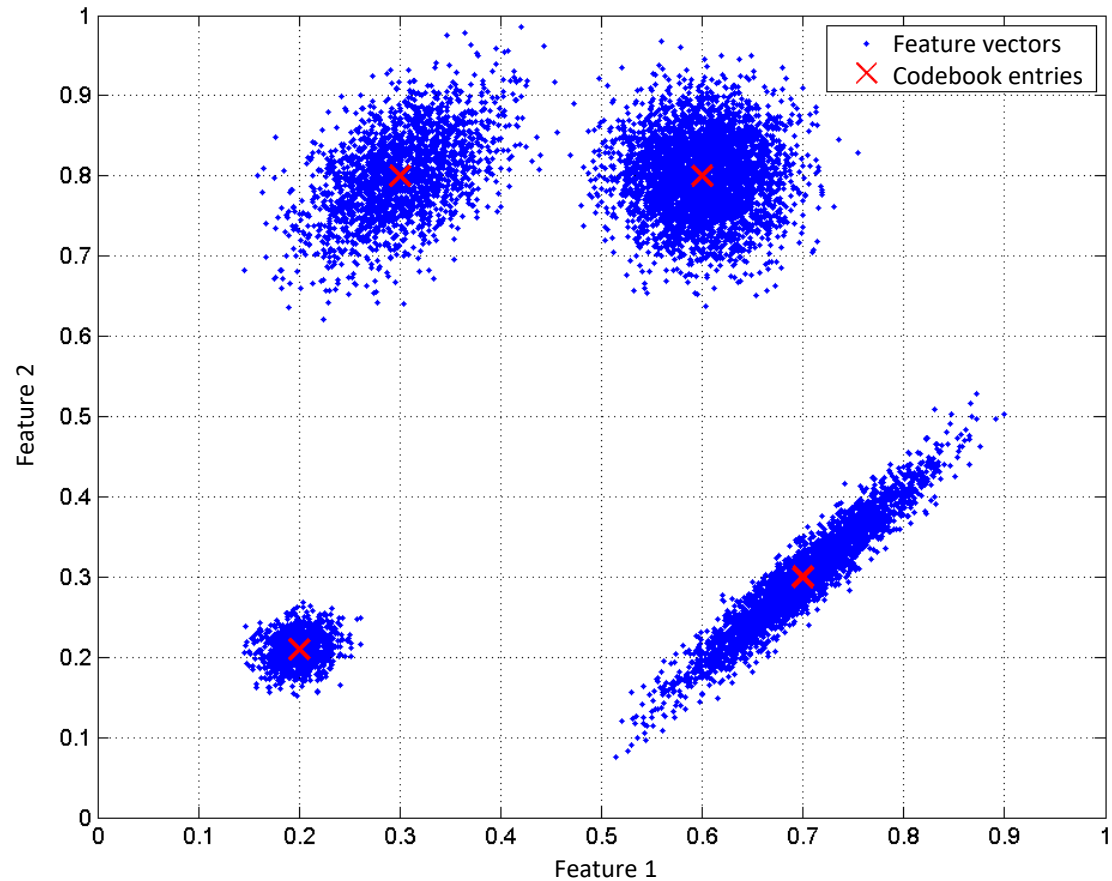
Feature vectors and corresponding codebook – feature room 1:



- Codebook with 4 entries
- 10.000 feature vectors are quantized

Introduction and Motivation – Part 2

Feature vectors and corresponding codebook – feature room 2:



- Codebook with 4 entries
- 10.000 feature vectors are quantized

Codebooks

Codebook definition:

$$\mathbf{C} = [\mathbf{c}_0, \mathbf{c}_1, \dots, \mathbf{c}_{K-1}] \quad \leftarrow \text{Codebook matrix}$$

$$\mathbf{c}_i = [c_{i,0}, c_{i,1}, \dots, c_{i,M-1}]^T \quad \leftarrow \text{Codebook vector}$$

- The codebook vectors should be chosen such that they represent a large number of so-called feature vectors with a small average distance.
- For feature vectors

$$\mathbf{x}(n)$$

and a **distance measure**

$$d(\mathbf{x}(n), \mathbf{c}_i),$$

it should follow for the **average distance**:

$$\frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0 \dots K-1} \{d(\mathbf{x}(n), \mathbf{c}_i)\} \longrightarrow \min.$$

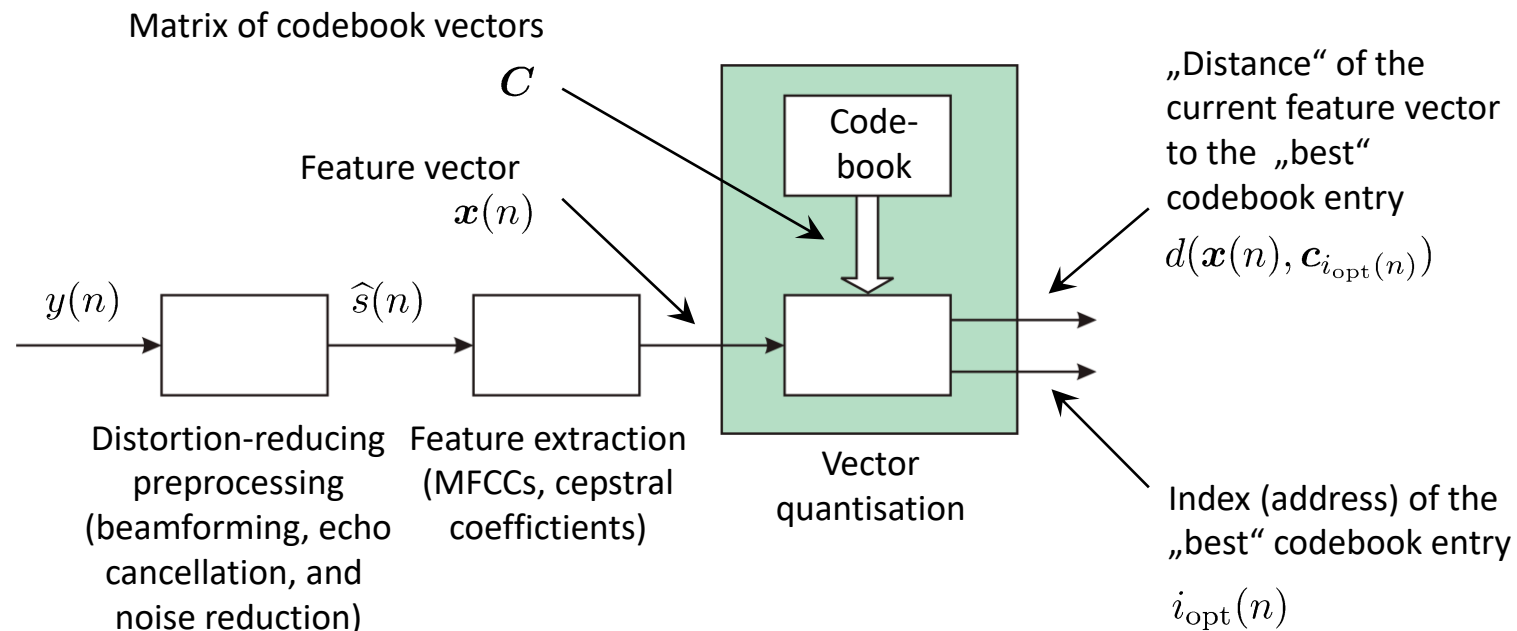
Literature

Codebook training:

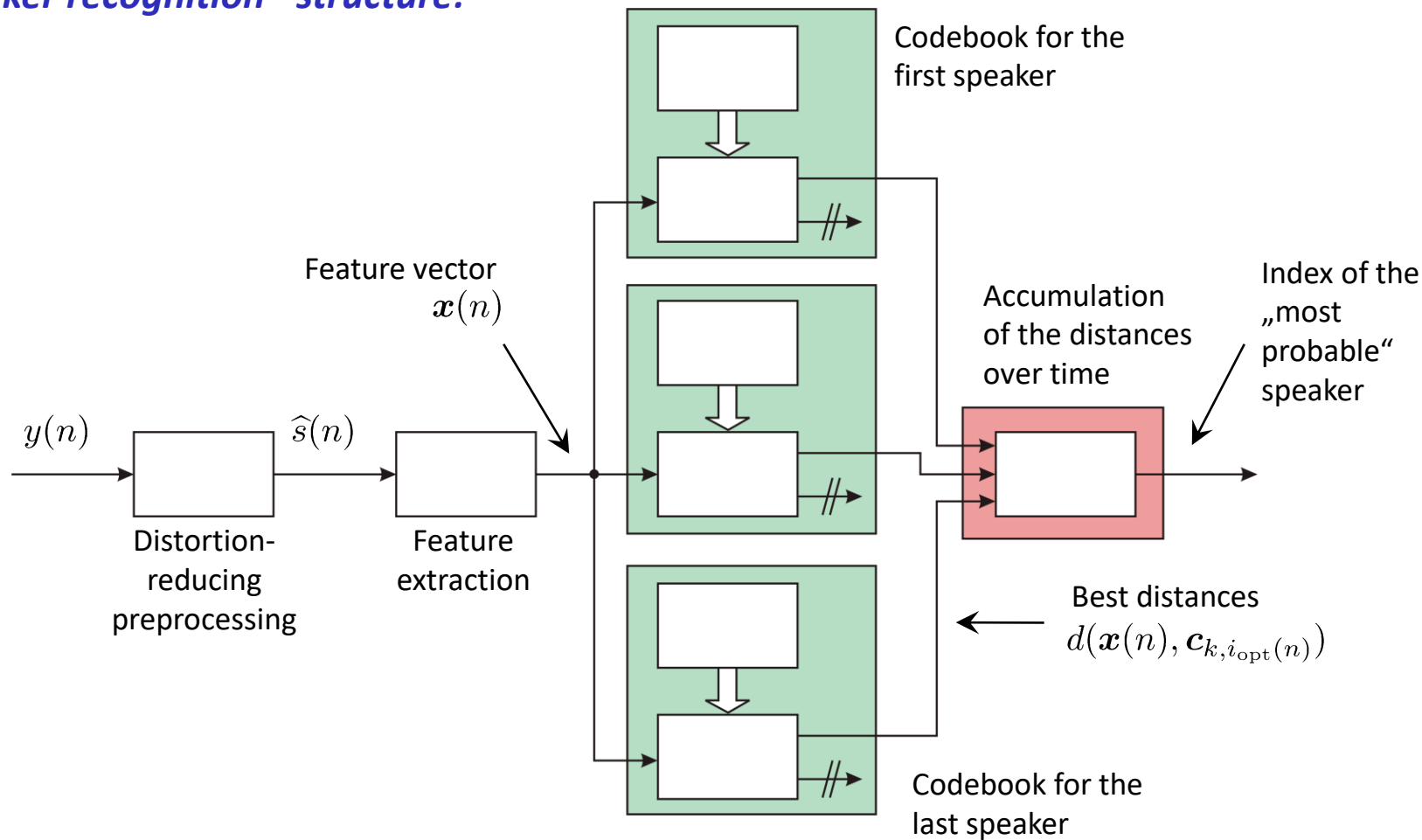
- ❑ L. R. Rabiner, R. W. Schafer: *Digital Processing of Speech Signals*, Prentice Hall, 1978
- ❑ C. Bishop: *Pattern Recognition and Machine Learning*, Springer, 2006
- ❑ B. Pfister, T. Kaufman: *Sprachverarbeitung*, Springer, 2008 (in German)

Current version of „Sprachsignalverarbeitung“ is free via the university library ...

Basic structure of a codebook search:



Speaker recognition - structure:

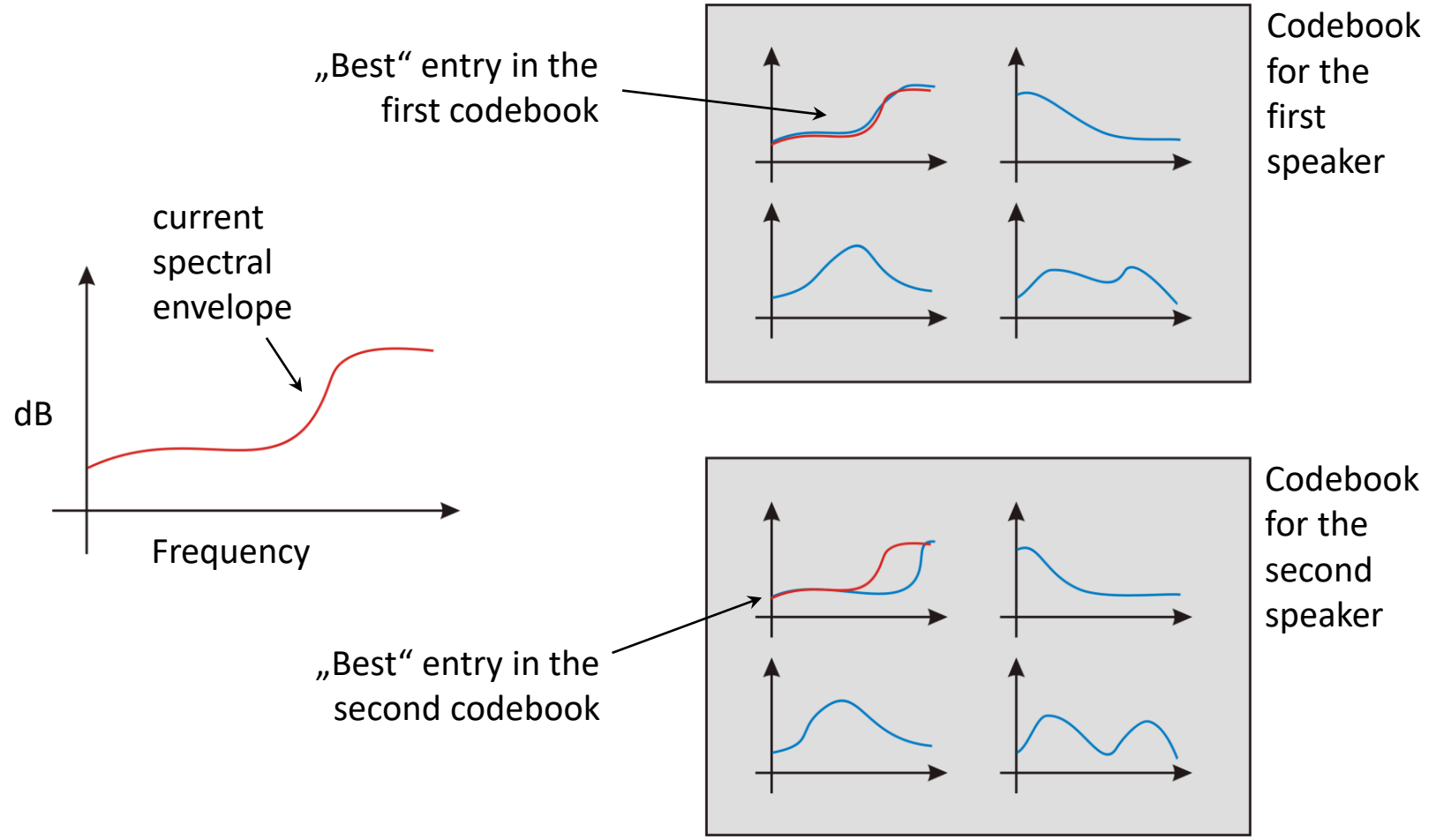


Application Examples – Part 3

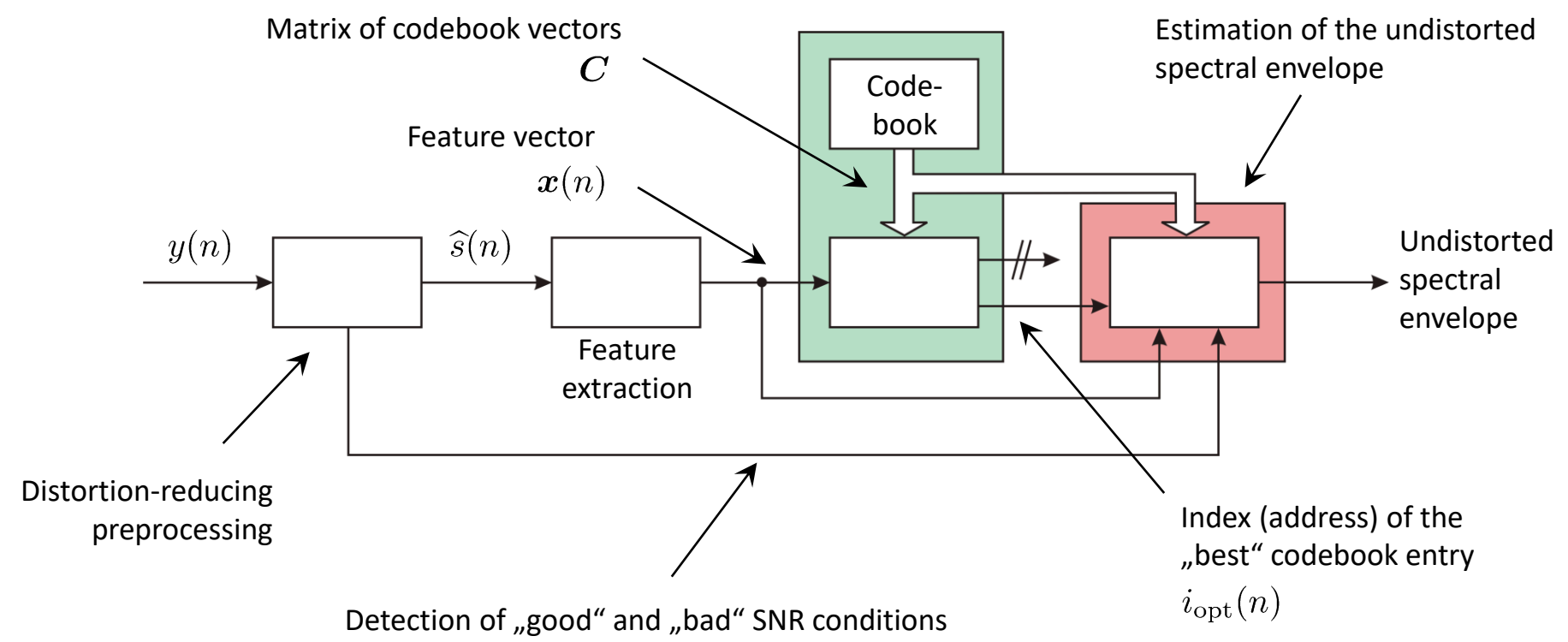
Speaker recognition – basic principle:

- ❑ The current spectral envelope of the signal is compared to the entries of several codebooks and the distance to the „**best match**“ is computed for each codebook.
- ❑ The codebooks belong to a speaker that is known in advance and have been trained to his data.
- ❑ The **smallest distances** of each codebook are **accumulated**.
- ❑ The **smallest accumulated distance** determines on **which speaker** it will be decided.
- ❑ Models of the speakers that are known in advance compete against one or more „**universal**“ **models**. A new speaker can be recognized if the „universal“ codebook is better than the best individual one. In this case, a **new codebook** for the new speaker can be initialized.
- ❑ An **update** of the „winner-codebook“ is usually done.

Speaker recognition – basic principle:



Signal reconstruction – structure:

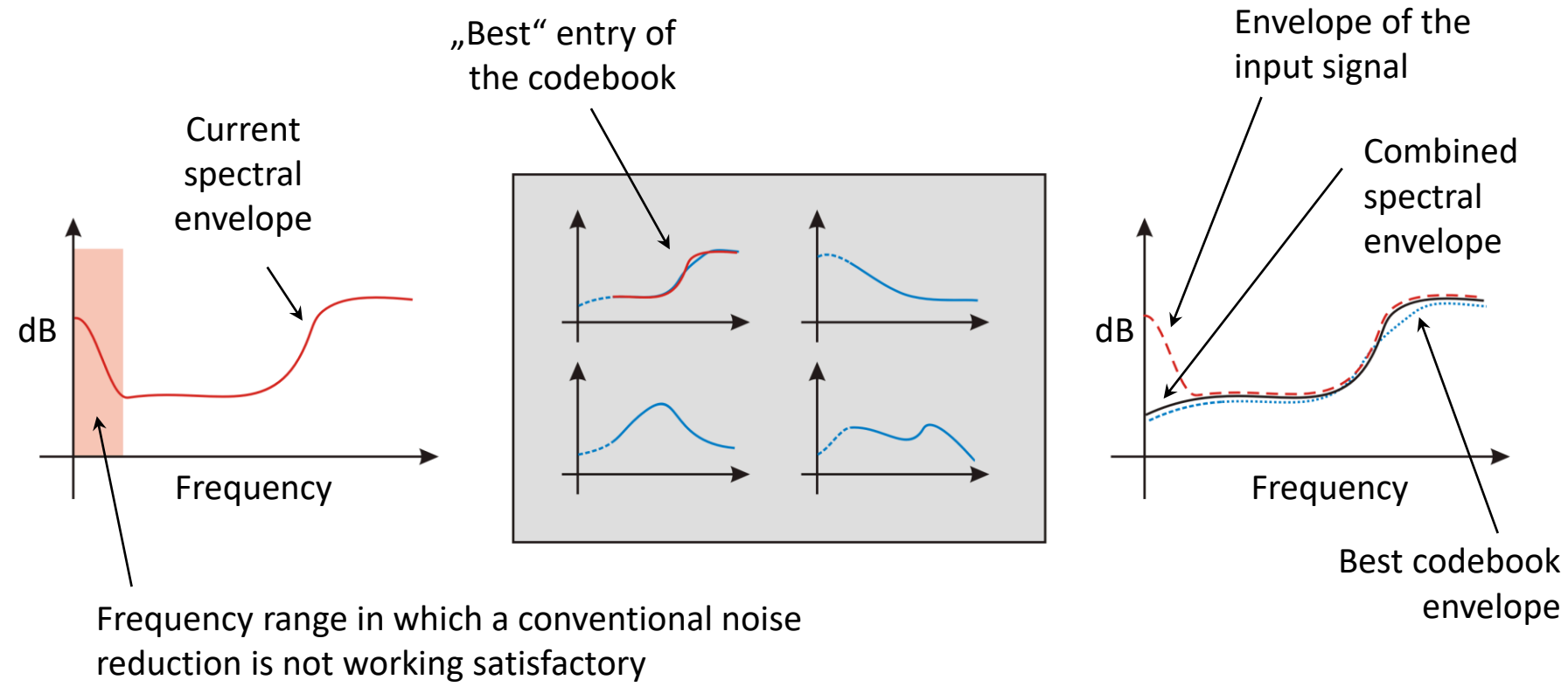


Application Examples – Part 6

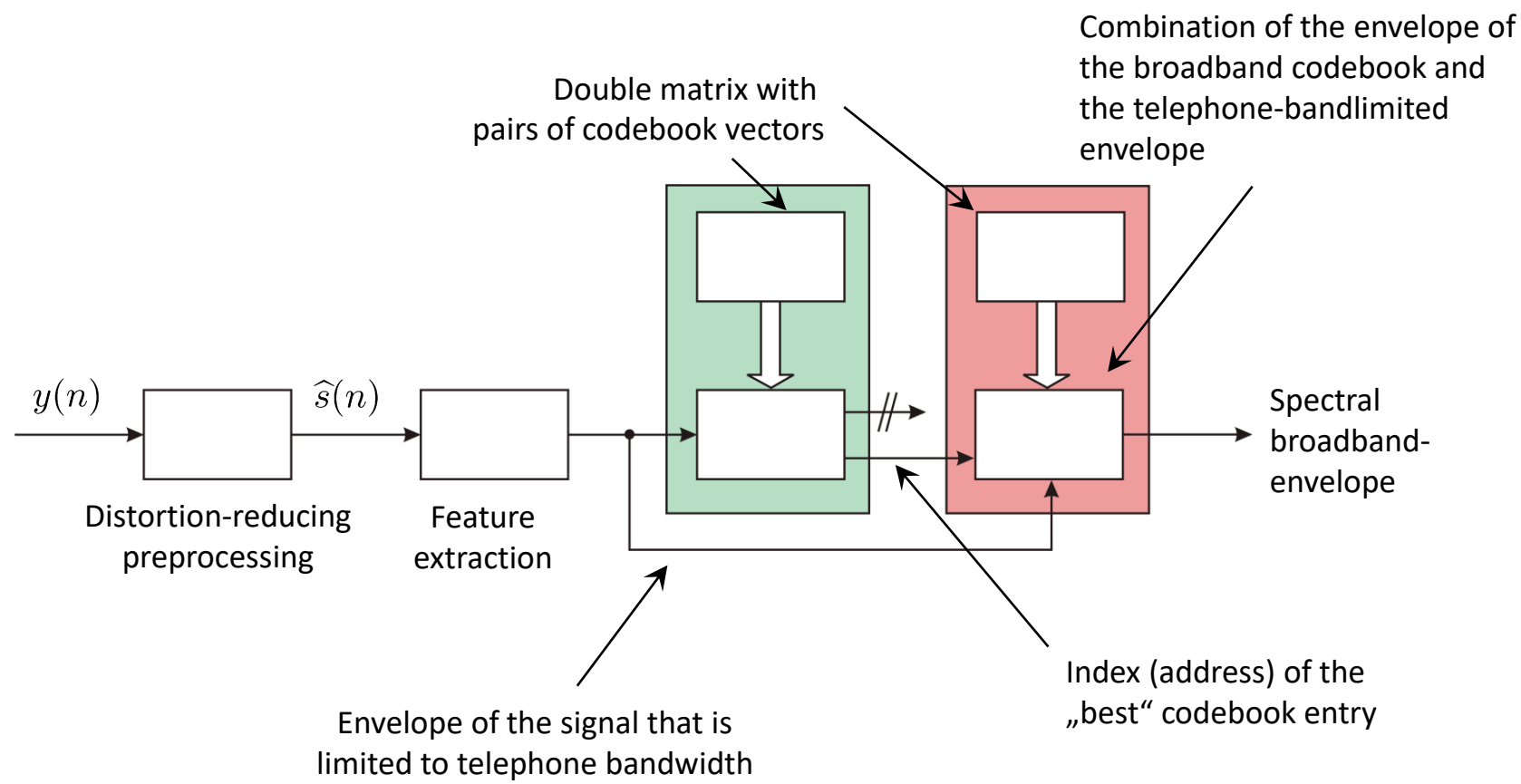
Signal reconstruction – basic principle:

- ❑ First, the ***input signal*** is „conventionally“ ***noise-reduced***. In this processing step it is also determined in which frequency ranges the conventional method does not “open” (attenuation is less than the maximum attenuation).
- ❑ Based on the conventionally improved signal, the ***spectral envelope is estimated*** by, e.g., the logarithmic melband-signal powers.
- ❑ In a codebook it is now searched for that envelope that has the ***smallest distance*** to the input signal’s envelope within the „***allowed***“ ***frequency range***.
- ❑ Because the „allowed“ frequency range is not known *a priori*, both, ***the features that are used and the cost function, have to be chosen appropriately***.
- ❑ Finally, the extracted ***spectral envelope of the input signal and the best codebook envelope are combined*** such that the codebook envelope is chosen in such areas where the conventional noise reduction fails and the original input envelope is chosen for the remaining frequency range.

Signal reconstruction – basic principle:



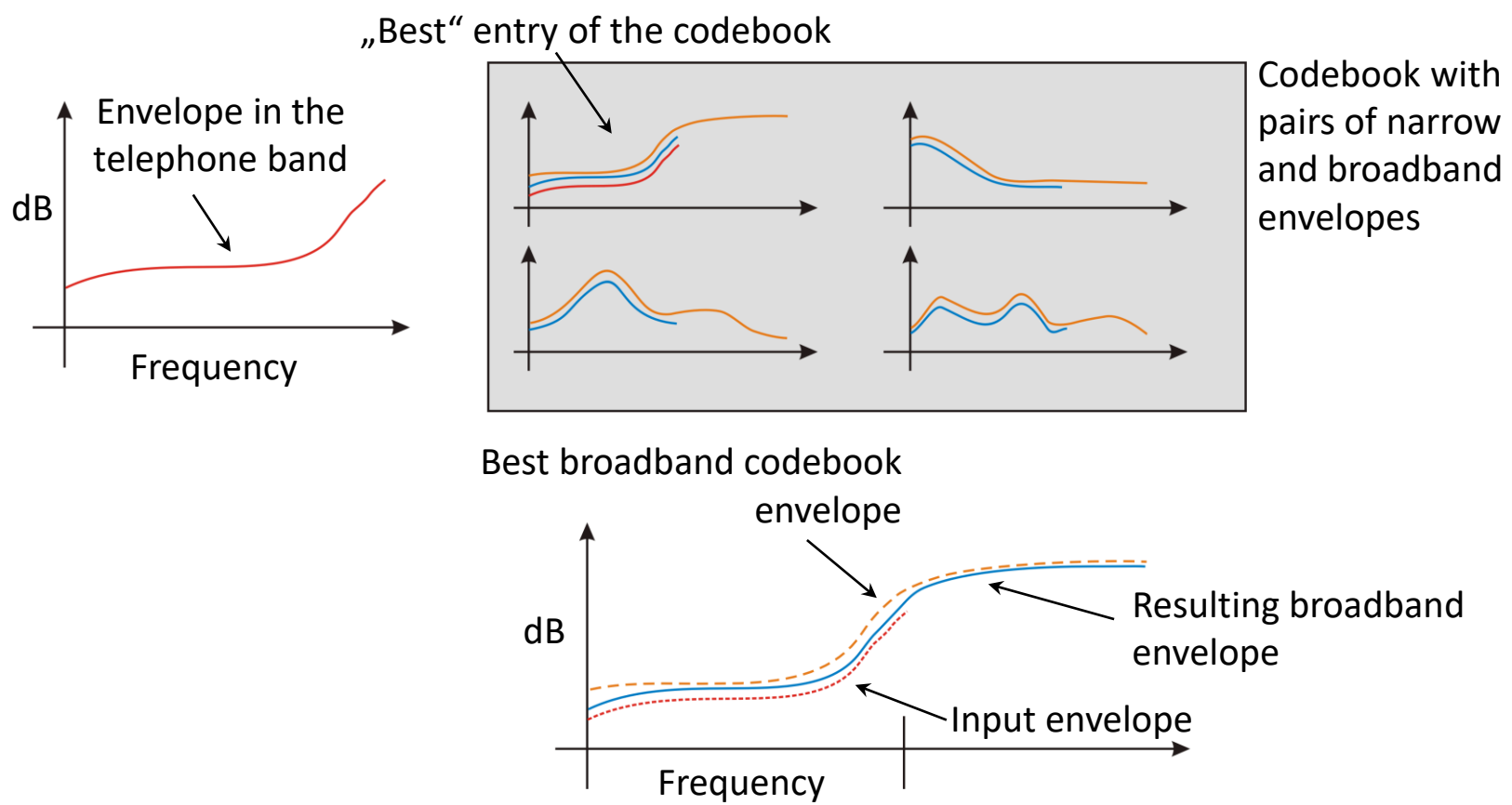
Bandwidth extension – structure:



Bandwidth extension – basic principle:

- ❑ For bandwidth extension, *two codebooks are trained in parallel*: one for the envelopes of the telephone-bandlimited signal and a second one for the envelopes of the broadband signal.
- ❑ During the training it is important that the input vectors (so-called double vectors) are available *synchronously*, such that identical feature data can be used for the clustering of the narrowband and broadband data.
- ❑ The *search* is done in the *narrowband-codebook*. The suitable broadband code vector is chosen and combined with the extracted narrowband entry in such a way, that the original envelope is chosen in the telephone band and the broadband codebook entry for the remaining frequency range.

Bandwidth extension – basic principle:



Application Examples – Part 11

More application examples

- ❑ **Speech coding**: The aim is basically to code the spectral envelope with as little bits as possible. Vector quantization is a good choice here.
- ❑ **Classification of speech sound** (sibilant sounds, vowels): Several codebook entries are trained for each speech sound. Afterwards, a good classification can be achieved. This can be used, e.g., to select between different preprocessing methods for reducing distortions or to parametrize a universal preprocessing method properly.
- ❑ (Noise-) **environment recognition**: Depending on the environment, different hard- and software components should be used (switching between cardioid and omnidirectional microphones, activation or deactivation of dereverberation, etc.). To realize such a classification, codebooks for different environments can be trained and compared during operation.

Cost Functions for the Codebook Training – Part 1

Considerations regarding complexity and convenience:

- ❑ For *creating* a codebook, the *same* or at least a similar *cost function* to that used during *operation* should be used.
- ❑ Usually all codebook entries have to be compared to the input feature vector for every processing frame. Thus, a computationally cheap cost function should be chosen: typically the squared distance or a magnitude distance.
- ❑ If not all elements of the feature space can be extracted with sufficient quality, a non-negative *weighting* for the elements can be introduced.
- ❑ If logarithmic feature elements are used, a zero before taking the logarithm can lead to very large distances. For that reason, a limitation can be introduced.

Cost Functions for the Codebook Training – Part 2

Ansatz:

- **Difference** between the feature elements:

$$\tilde{\Delta}_i(n) = x_i(n) - y_i(n)$$

- **Limitation:**

$$\Delta_i(n) = \min \left\{ \Delta_{\max}, \max \left\{ \Delta_{\min}, \tilde{\Delta}_i(n) \right\} \right\}$$

- **Squaring** and **weighting**:

$$d(\mathbf{x}(n), \mathbf{y}(n)) = \mathbf{\Delta}^T(n) \mathbf{G} \mathbf{\Delta}(n)$$

with:

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

$$\mathbf{\Delta}(n) = [\Delta_0(n), \dots, \Delta_{M-1}(n)]^T$$

Evaluation of Codebooks

Basic principle:

- Division of the feature data base in to **training data** and **evaluation data**. A 80/20-division could be used (80 % of the data are used for training, 20 % for evaluation)
- The codebook training is done **iteratively**, i.e., a initial start codebook is improved in each step:

$$\mathbf{C}^p = [\mathbf{c}_0^p, \mathbf{c}_1^p, \dots, \mathbf{c}_{K-1}^p] \longrightarrow \mathbf{C}^{p+1} = [\mathbf{c}_0^{p+1}, \mathbf{c}_1^{p+1}, \dots, \mathbf{c}_{K-1}^{p+1}].$$

- A commonly used **condition for termination** is:

$$\begin{aligned} \overline{d(p)} &= \frac{1}{N} \sum_{n=0}^{N-1} \min_{i=0 \dots K-1} \{d(\mathbf{x}(n), \mathbf{c}_i^p)\} < d_{\min}, \\ \frac{\overline{d(p)}}{\overline{d(p-1)}} &> 1 - \epsilon, \\ p &> p_{\max}. \end{aligned}$$

k-means Algorithm – Part 1

Given:

- A large number of training vectors $\mathbf{x}(n)$ and evaluation vectors $\tilde{\mathbf{x}}(n)$.

Wanted:

- A codebook \mathbf{C} of size K with the lowest possible mean distance

$$\bar{d} = \frac{1}{\tilde{N}} \sum_{n=0}^{\tilde{N}-1} \min_{i=0 \dots K-1} \left\{ d(\tilde{\mathbf{x}}(n), \mathbf{c}_i) \right\},$$

regarding the (training and) evaluation vectors.

Problem:

- Up to now, there is no method that solved the problem with reasonable computational complexity in an optimal way. Therefore, „suboptimal“ methods are used.

k-means Algorithm – Part 2

Initialization:

- Selection of K arbitrary, different vectors out of the training data as codebook vectors.

Iteration:

- **Classification:** Assign each training vector $x(n)$ to a codebook vector c_i^p with minimum distance.
- **Codebook correction:** A new codebook vector c_i^{p+1} is generated by averaging over all training vectors c_i^p that are assigned to the same codebook vector.
- **Termination condition:** By using the evaluation vectors it is checked, whether the termination condition (mentioned before) is met or not.

LBG-Algorithm – Part 1

Comparison with the k-means method:

- ❑ The principle of the LBG-algorithm is quite similar to the k-means approach. However, codebooks with increasing size are generated. Usually the size increases in powers of two but it could also change linearly.
- ❑ „LBG“ stands for the names of the inventors of the algorithm: Linde, Buzo, and Gray.

Initialization:

- ❑ The start codebook consists of only one entry which is chosen as them mean over all training vectors.

LBG-Algorithm – Part 2

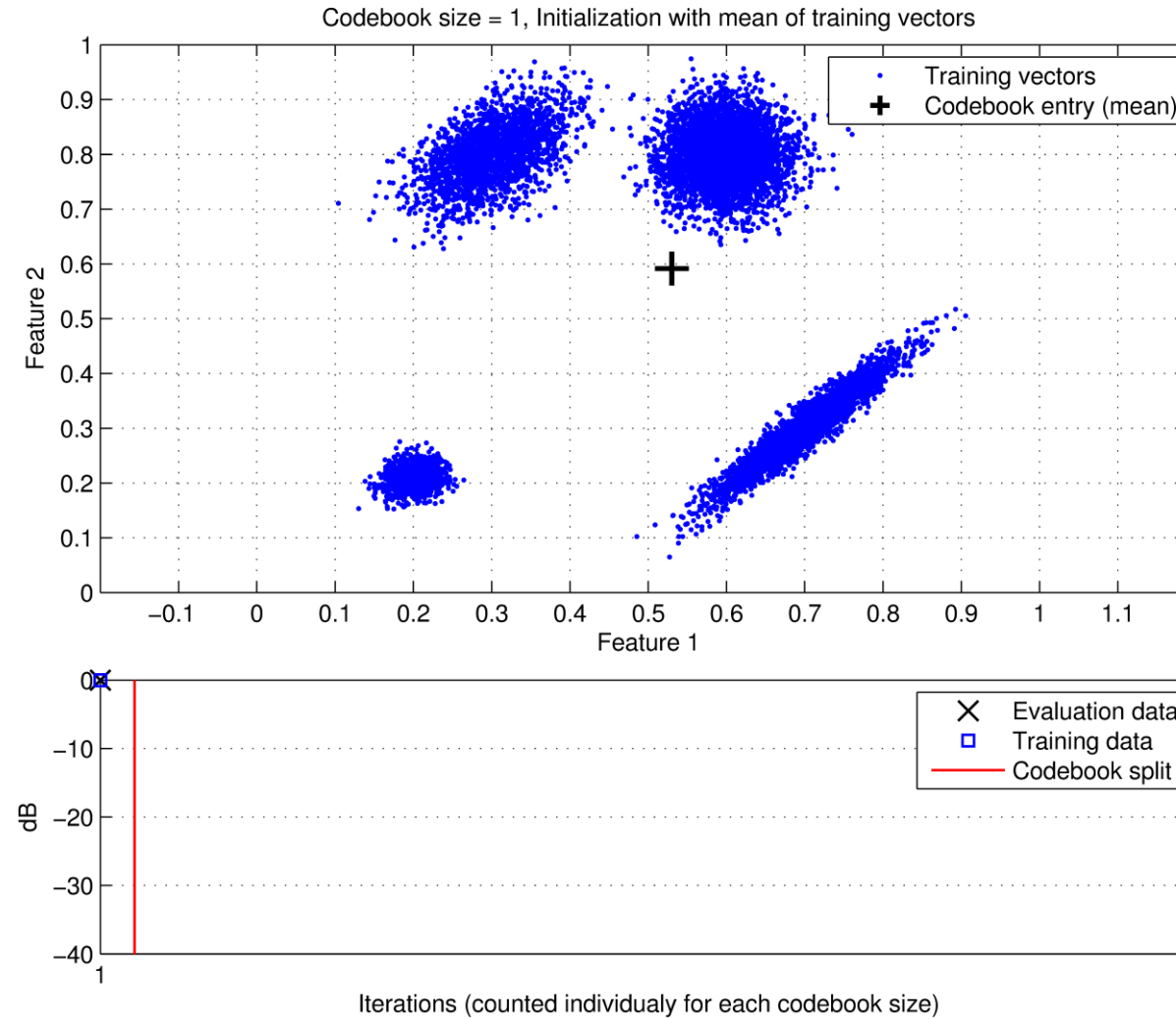
Iteration:

- ❑ **Increasing the codebook size:** The codebook size is doubled. The start vectors for the next codebook size are obtained by adding and subtracting vectors with small random entries to the code vectors of the previous code book.
- ❑ **Classification:** Assign each training vector $x(n)$ to a codebook vector c_i^p with minimum distance.
- ❑ **Codebook correction:** A new codebook vector c_i^{p+1} is generated by averaging over all training vectors c_i^p that are assigned to the same codebook vector.
- ❑ **Termination condition:** By using the evaluation vectors it is checked, whether the termination condition mentioned before is met or not.

Codebook Training

LBG-Algorithm – Example, Part 1

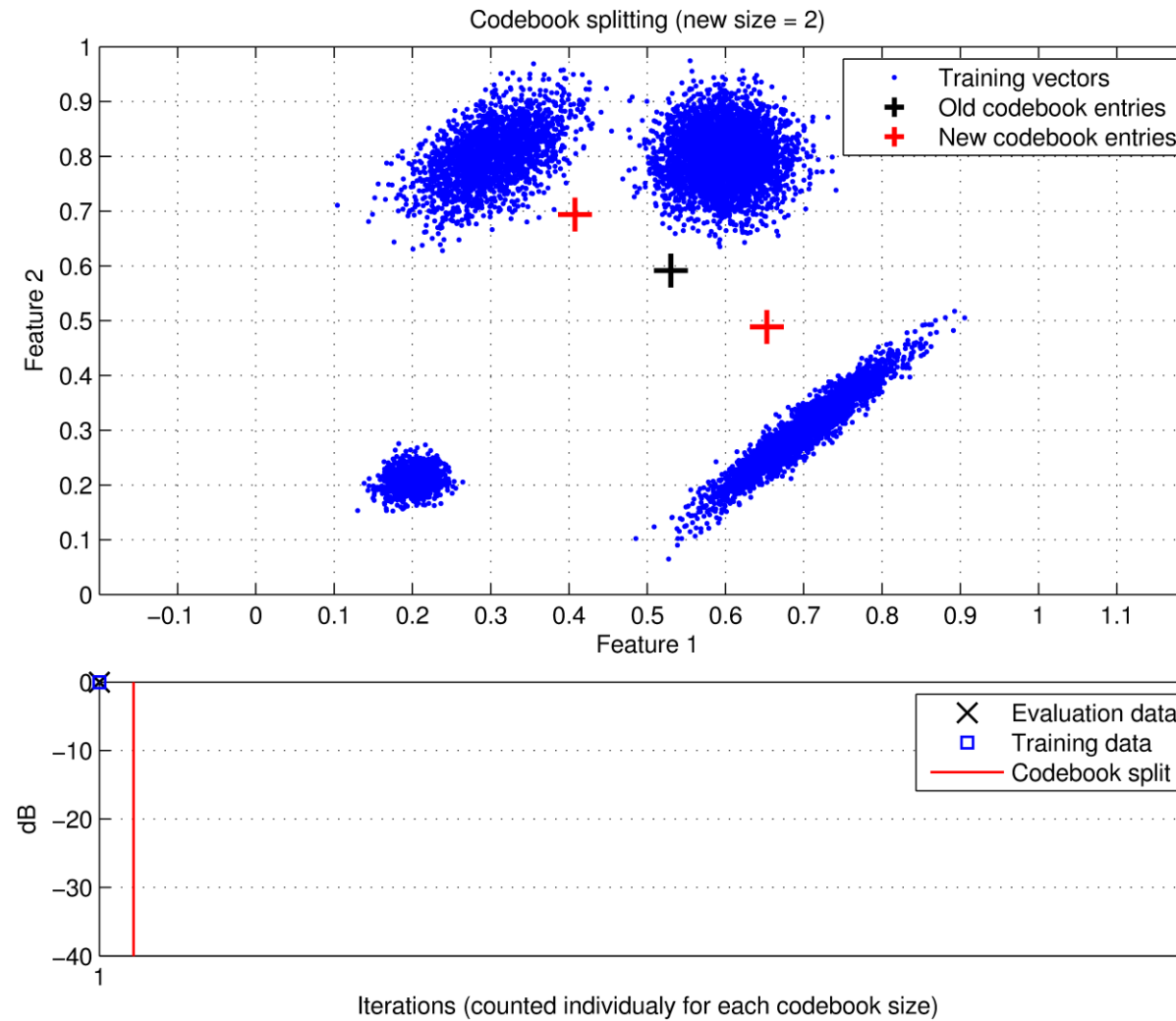
Initialization:



Codebook Training

LBG-Algorithm – Example, Part 2

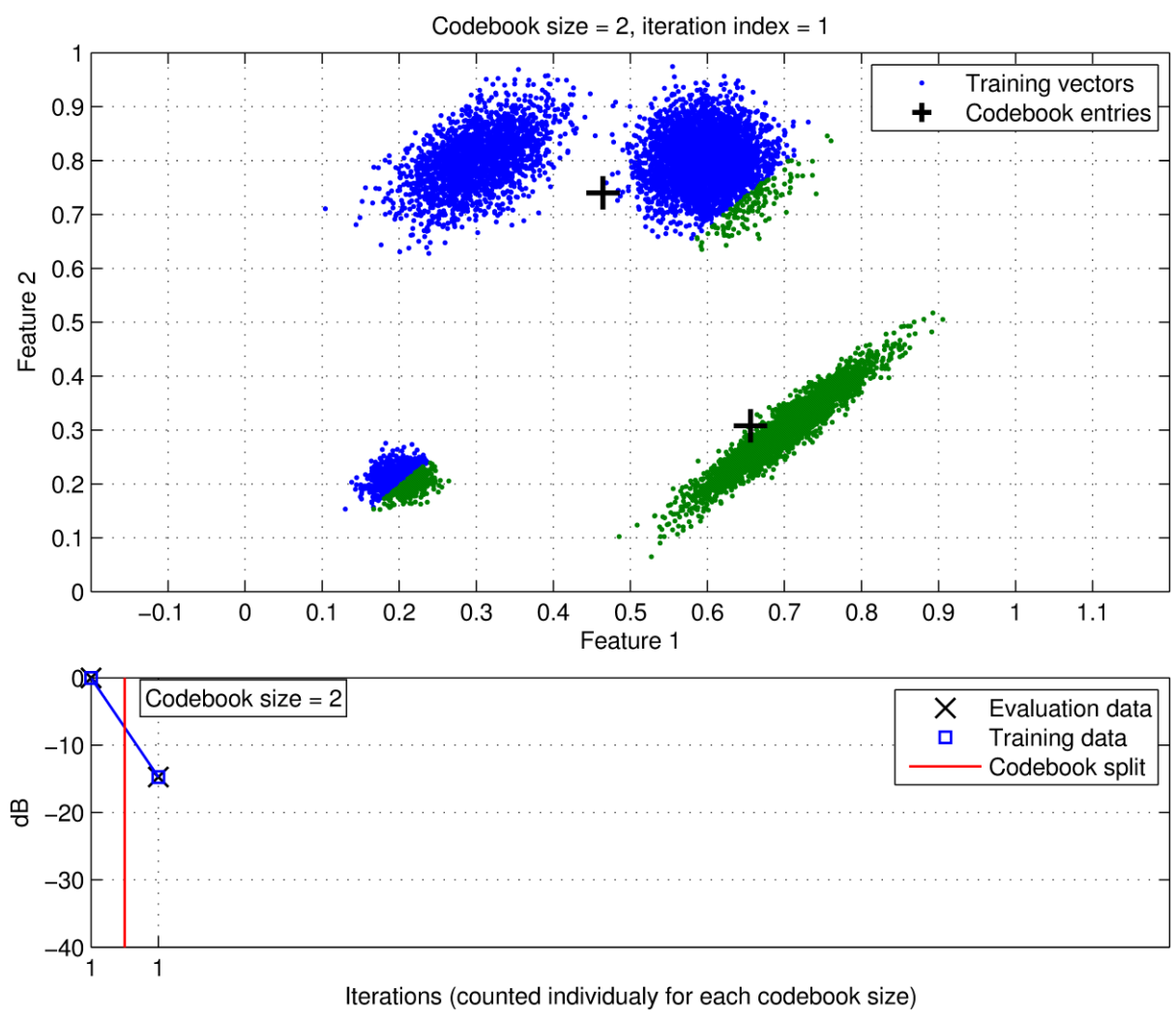
First codebook split:



Codebook Training

LBG-Algorithm – Example, Part 3

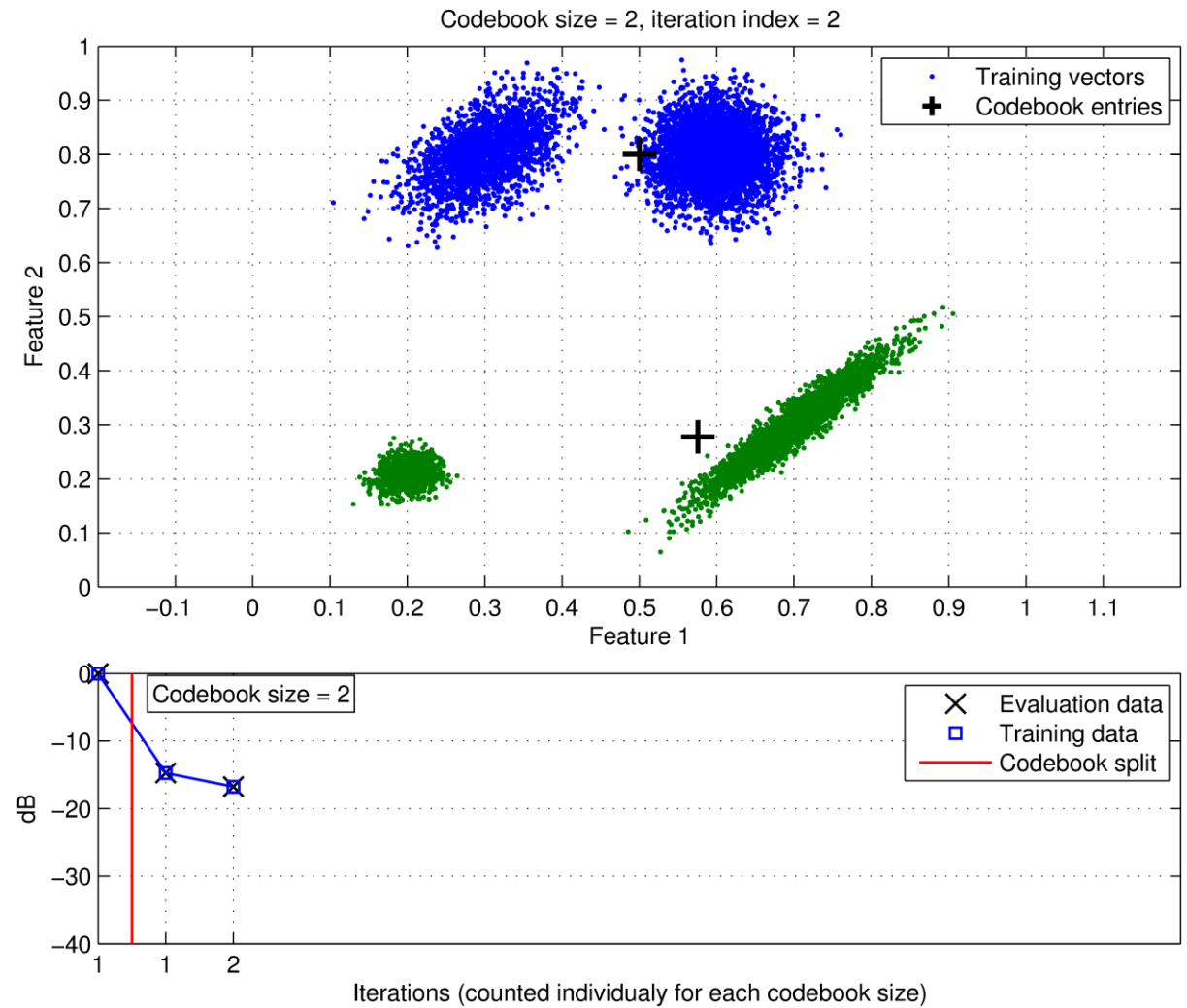
Codebook of size 2, after 1. iteration:



Codebook Training

LBG-Algorithm – Example, Part 4

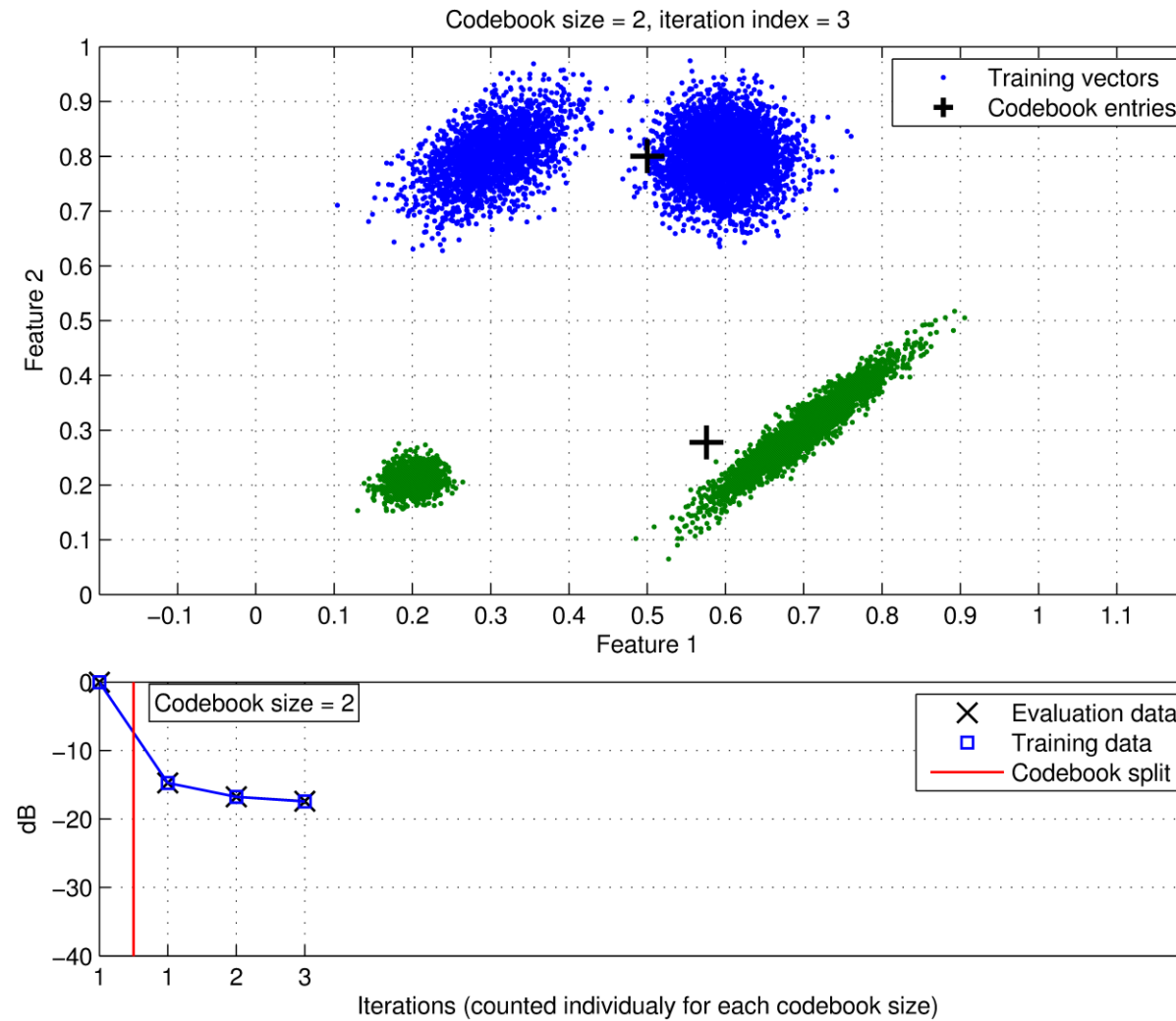
Codebook of size 2, after 2. iteration:



Codebook Training

LBG-Algorithm – Example, Part 5

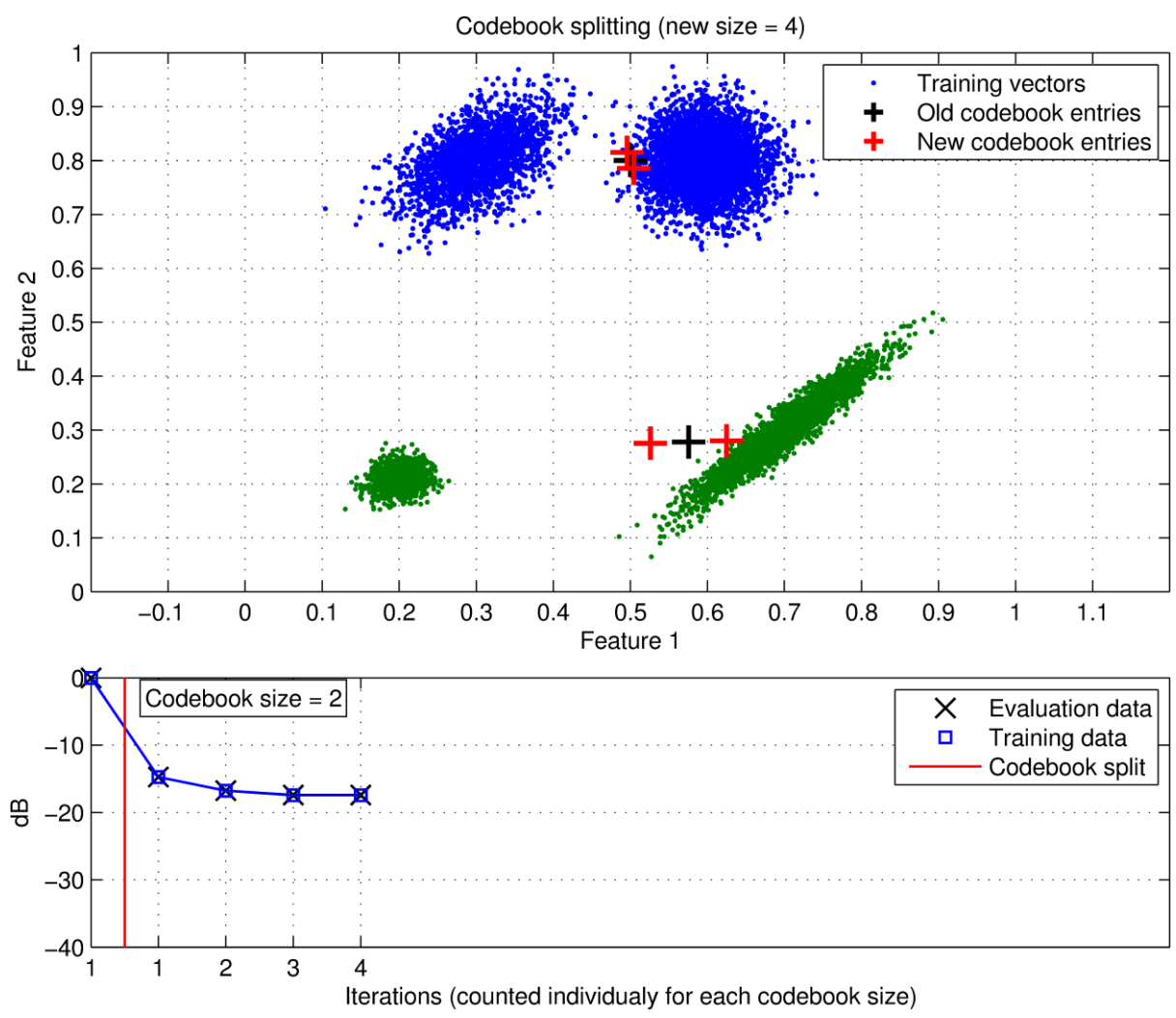
Codebook of size 2, after 3. iteration:



Codebook Training

LBG-Algorithm – Example, Part 6

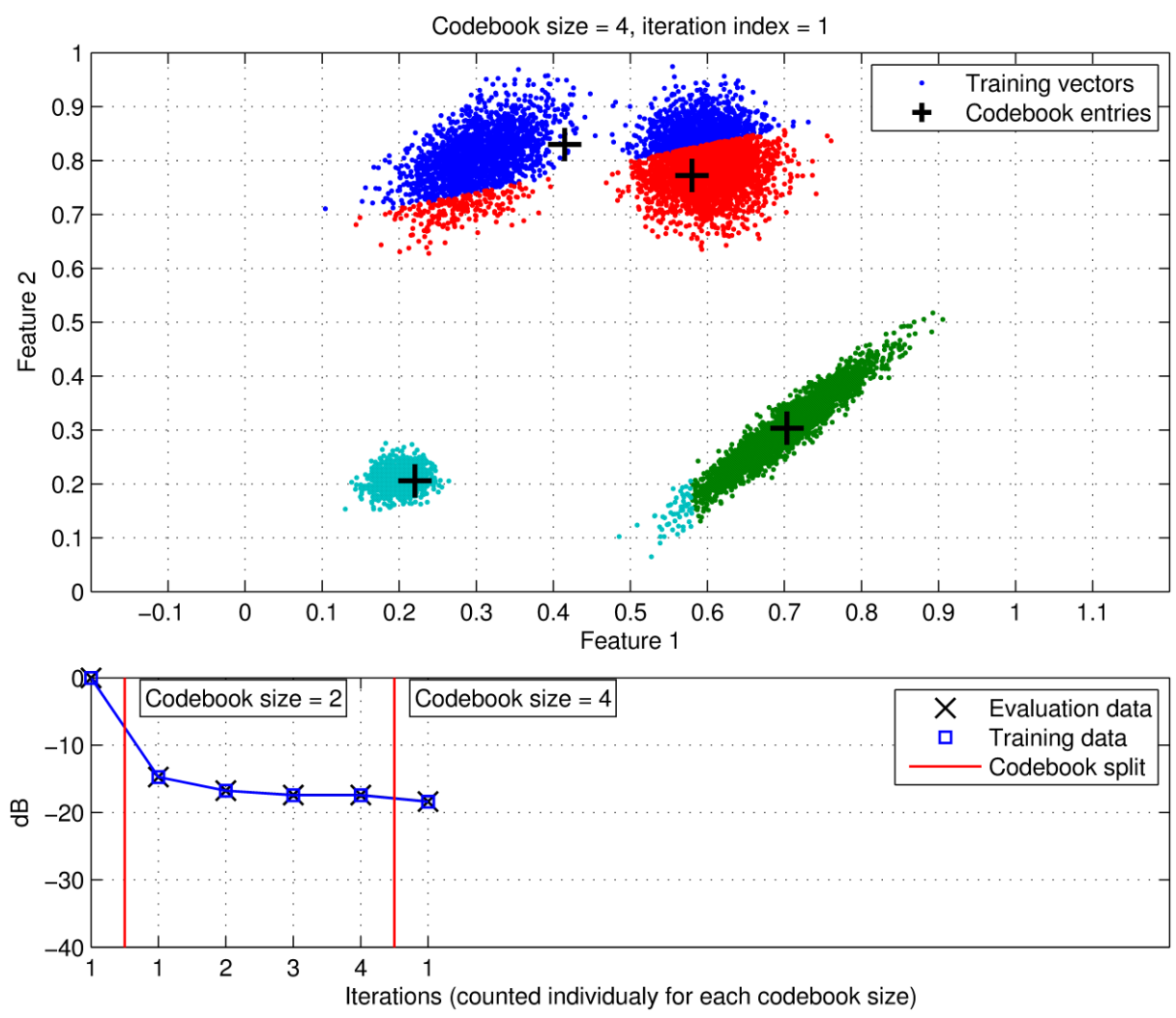
Codebook of size 4, after splitting:



Codebook Training

LBG-Algorithm – Example, Part 7

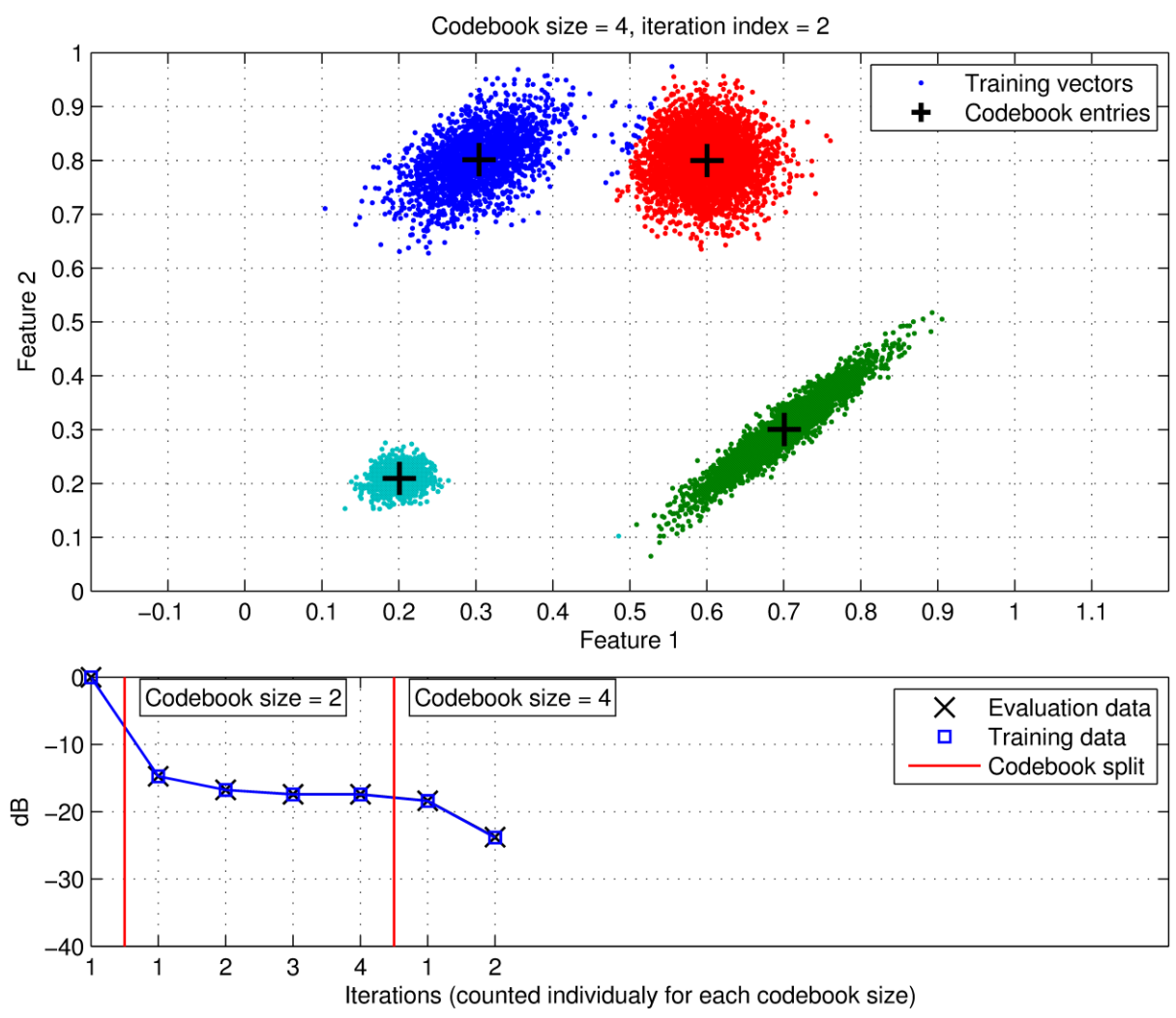
Codebook of size 4, after 1. iteration:



Codebook Training

LBG-Algorithm – Example, Part 8

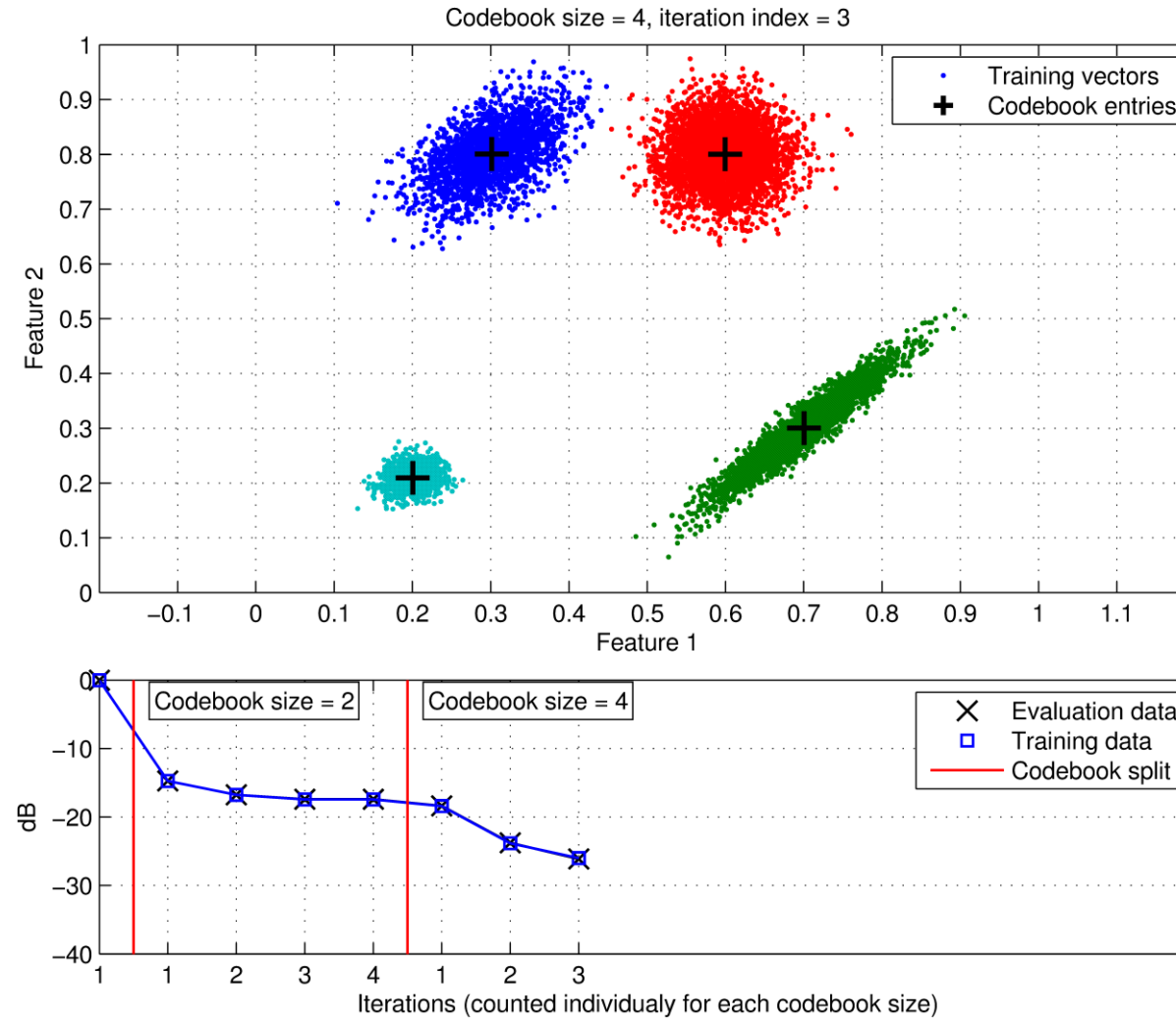
Codebook of size 4, after 2. iteration:



Codebook Training

LBG-Algorithm – Example, Part 9

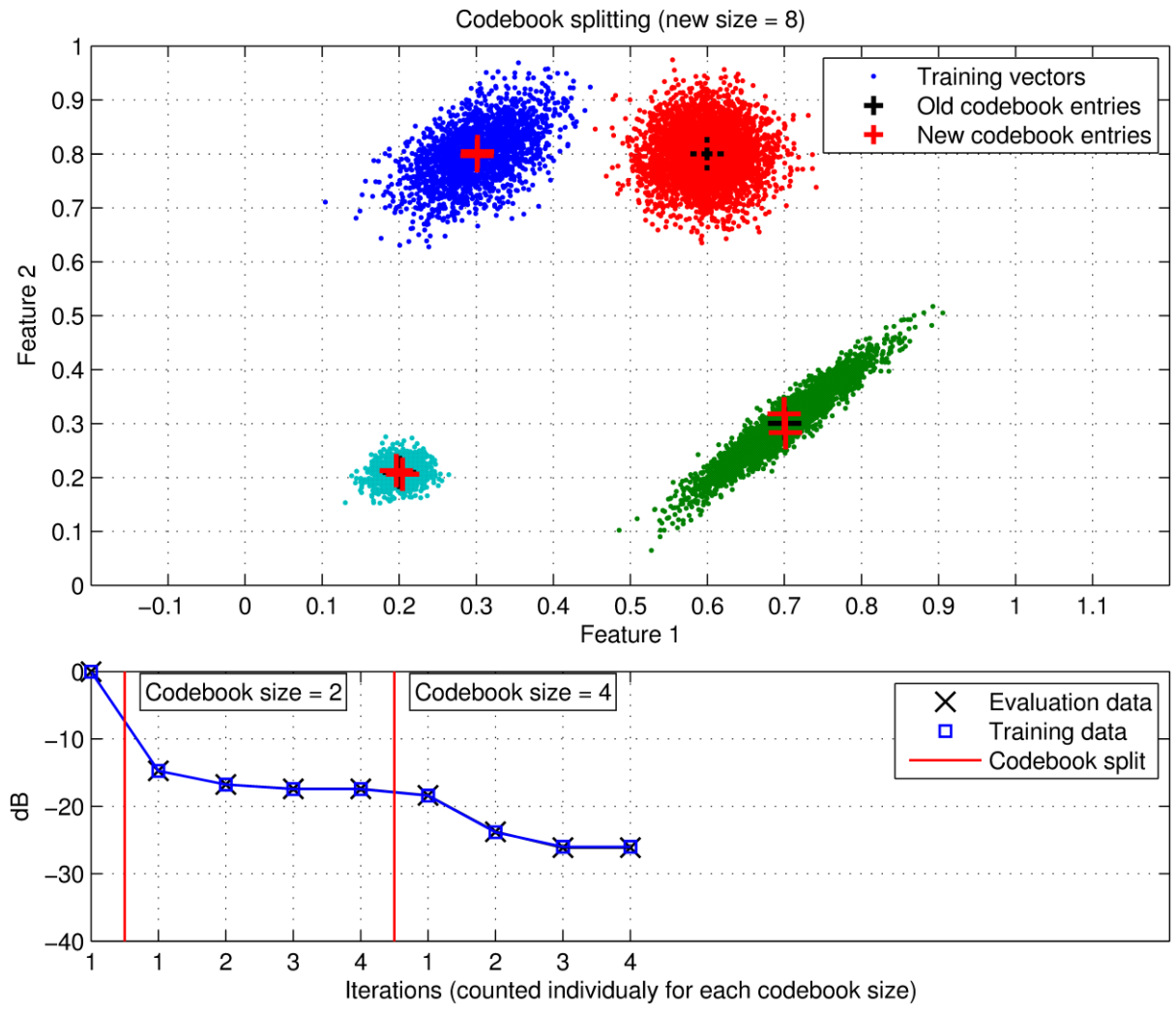
Codebook of size 4, after 3. iteration:



Codebook Training

LBG-Algorithm – Example, Part 10

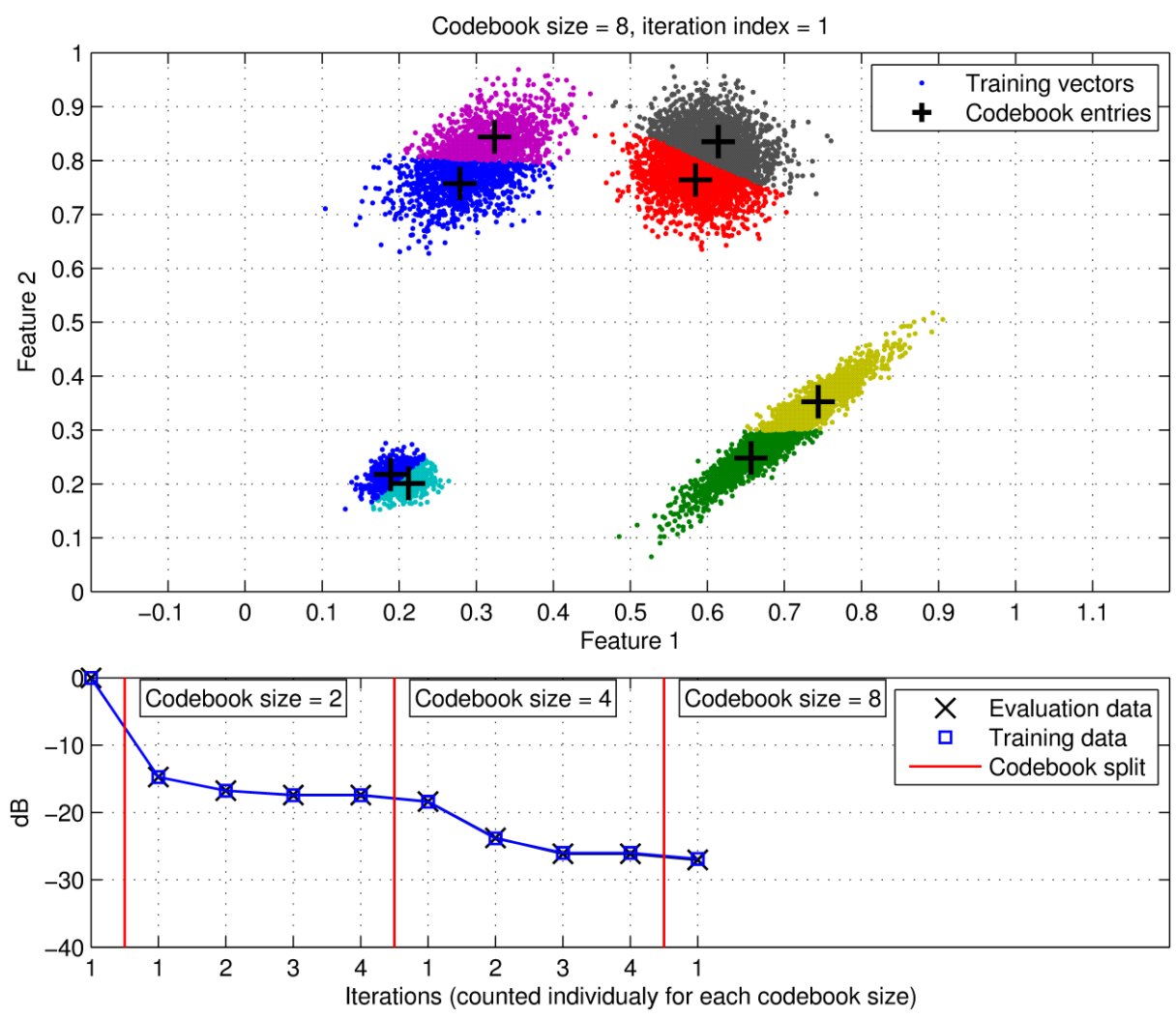
Codebook of size 8, after splitting:



Codebook Training

LBG-Algorithm – Example, Part 11

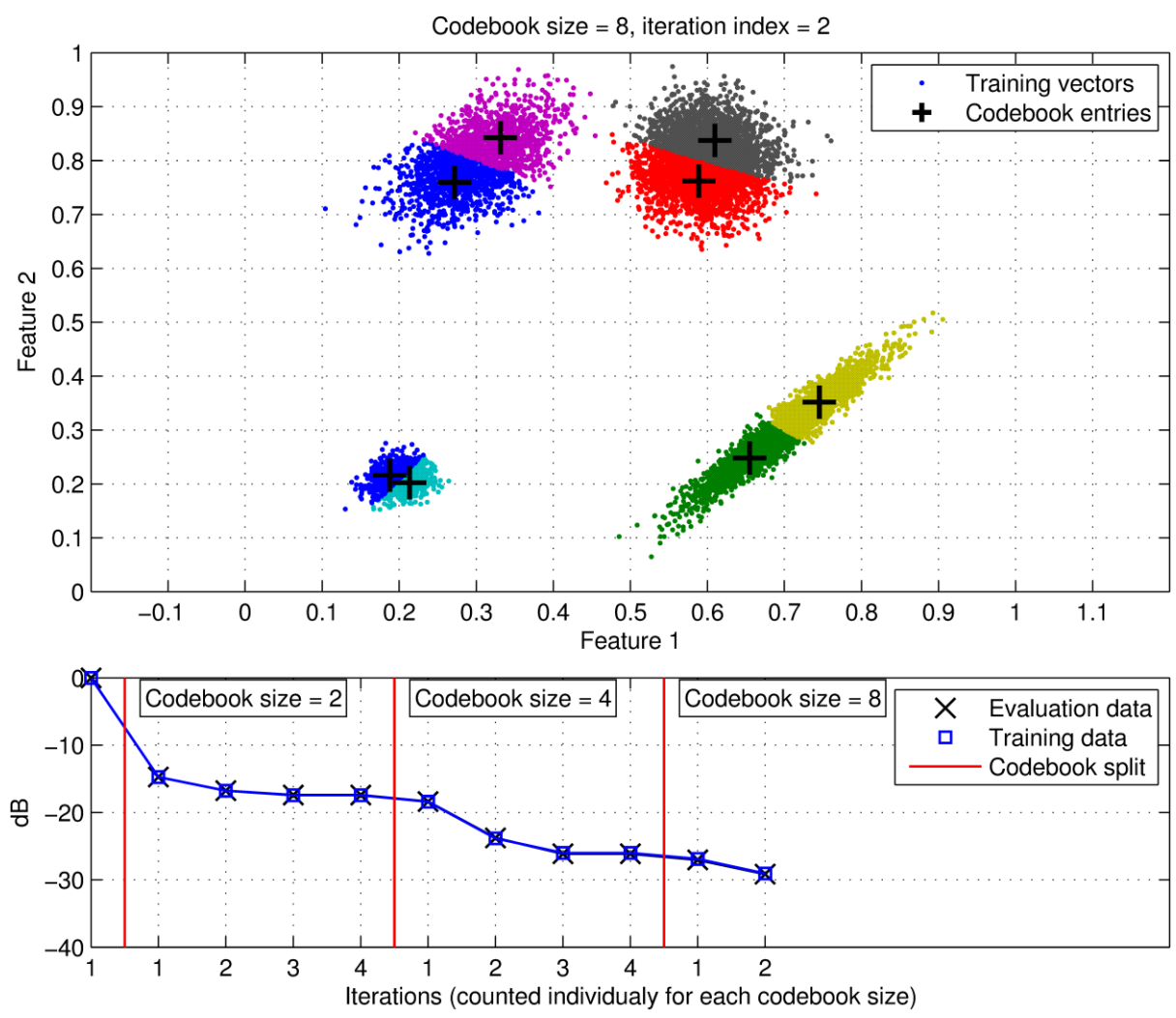
Codebook of size 8, after 1. iteration:



Codebook Training

LBG-Algorithm – Example, Part 12

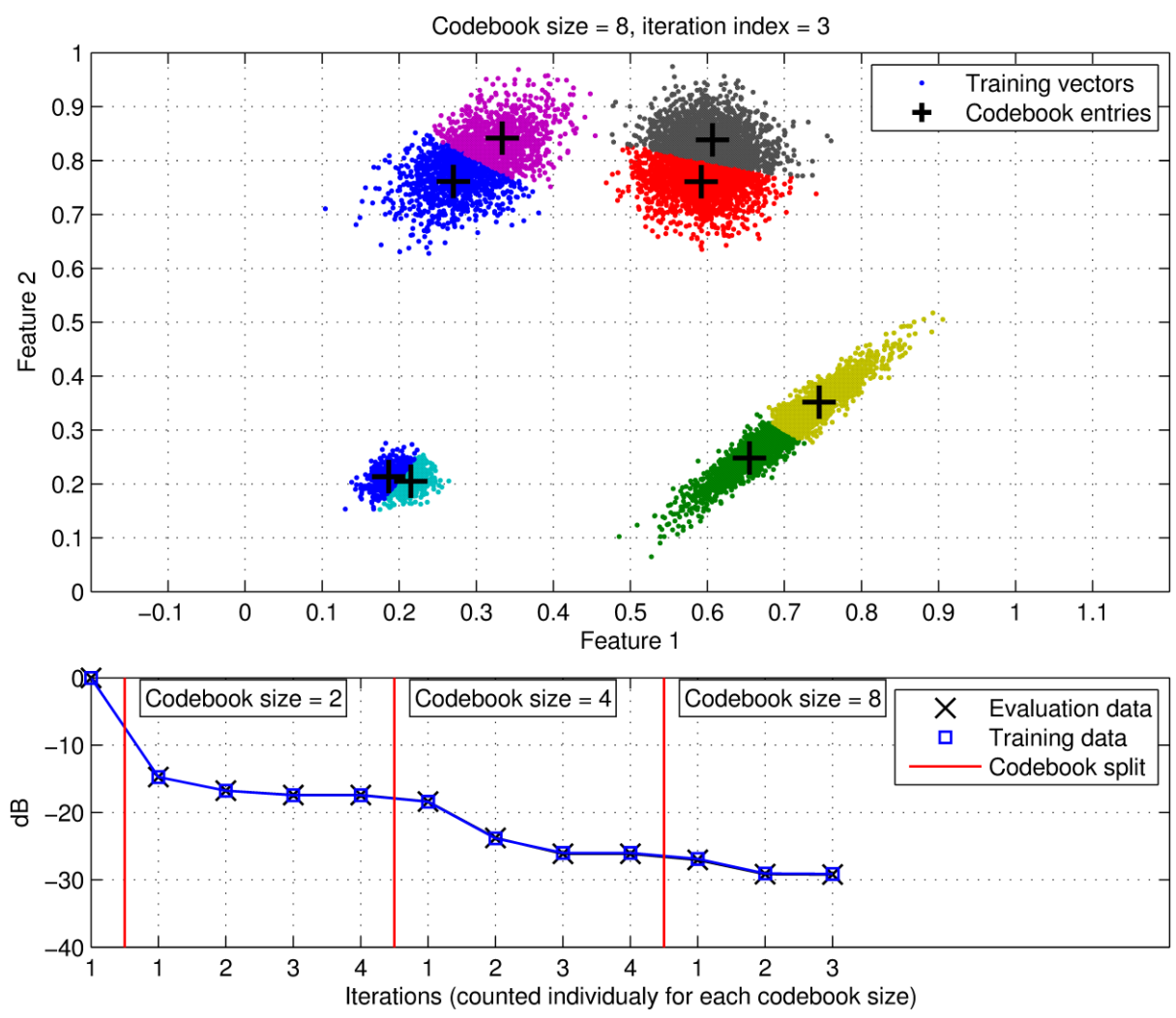
Codebook of size 8, after 2. iteration:



Codebook Training

LBG-Algorithm – Example, Part 13

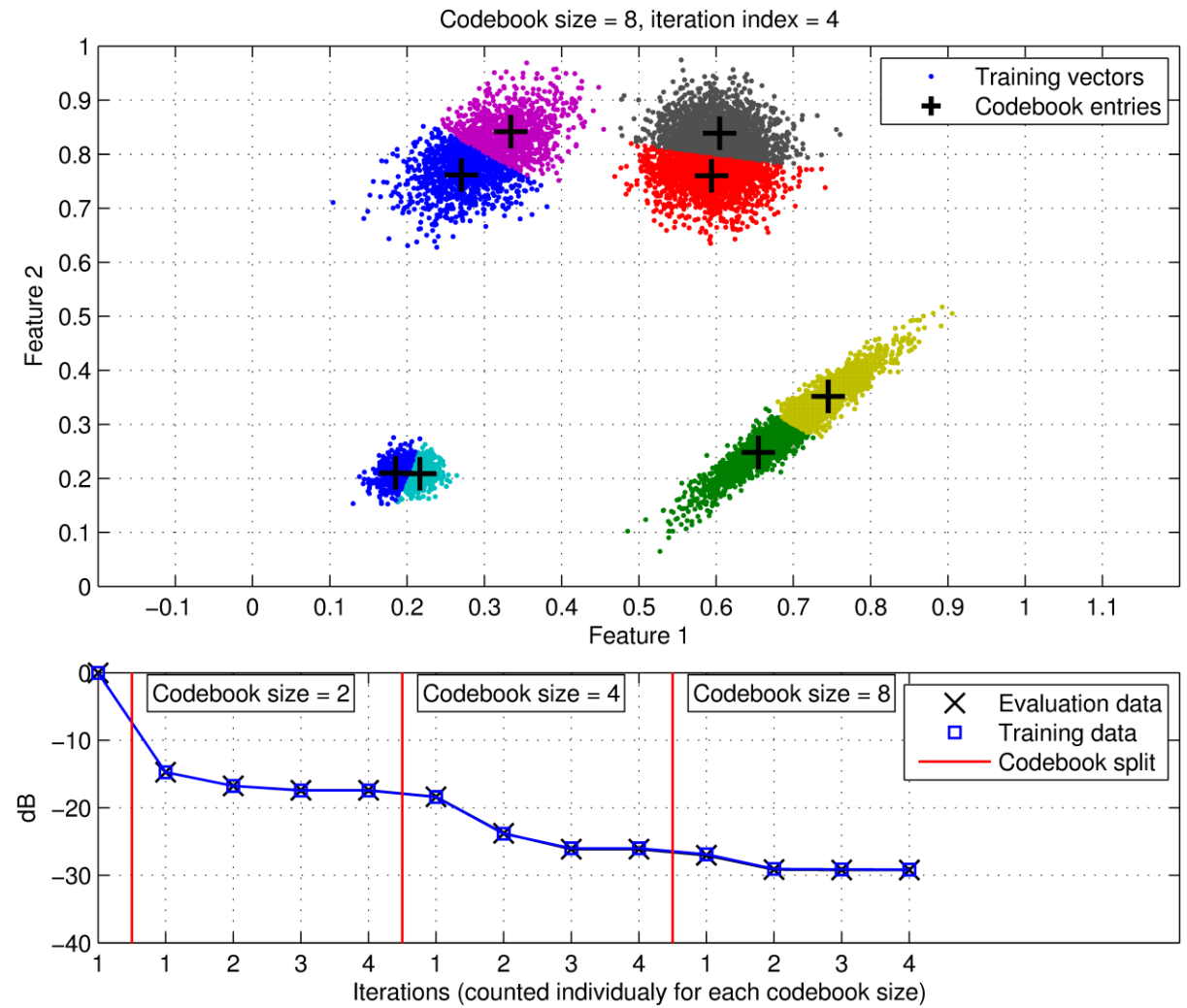
Codebook of size 8, after 3. iteration:



Codebook Training

LBG-Algorithm – Example, Part 14

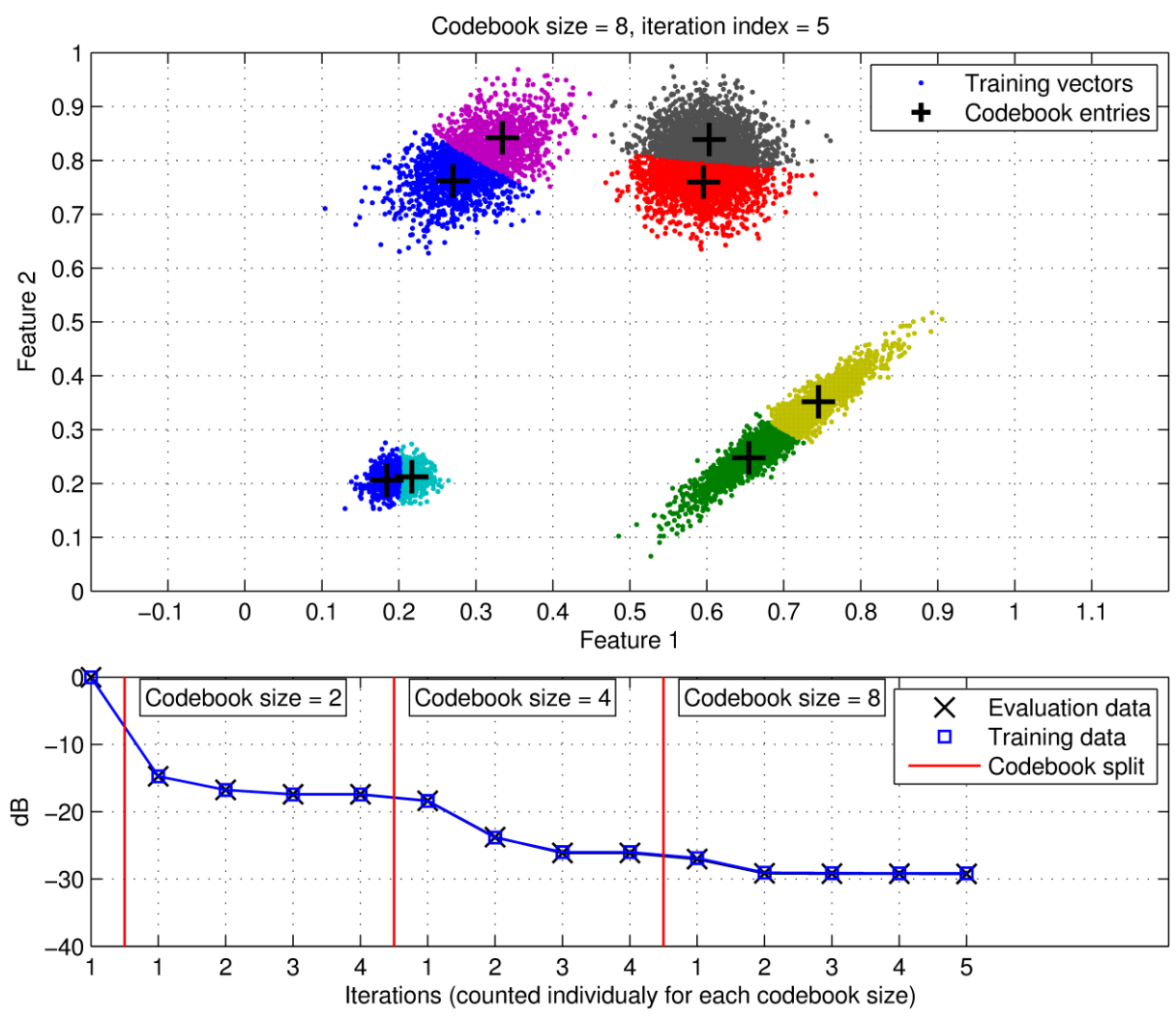
Codebook of size 8, after 4. iteration:



Codebook Training

LBG-Algorithm – Example, Part 15

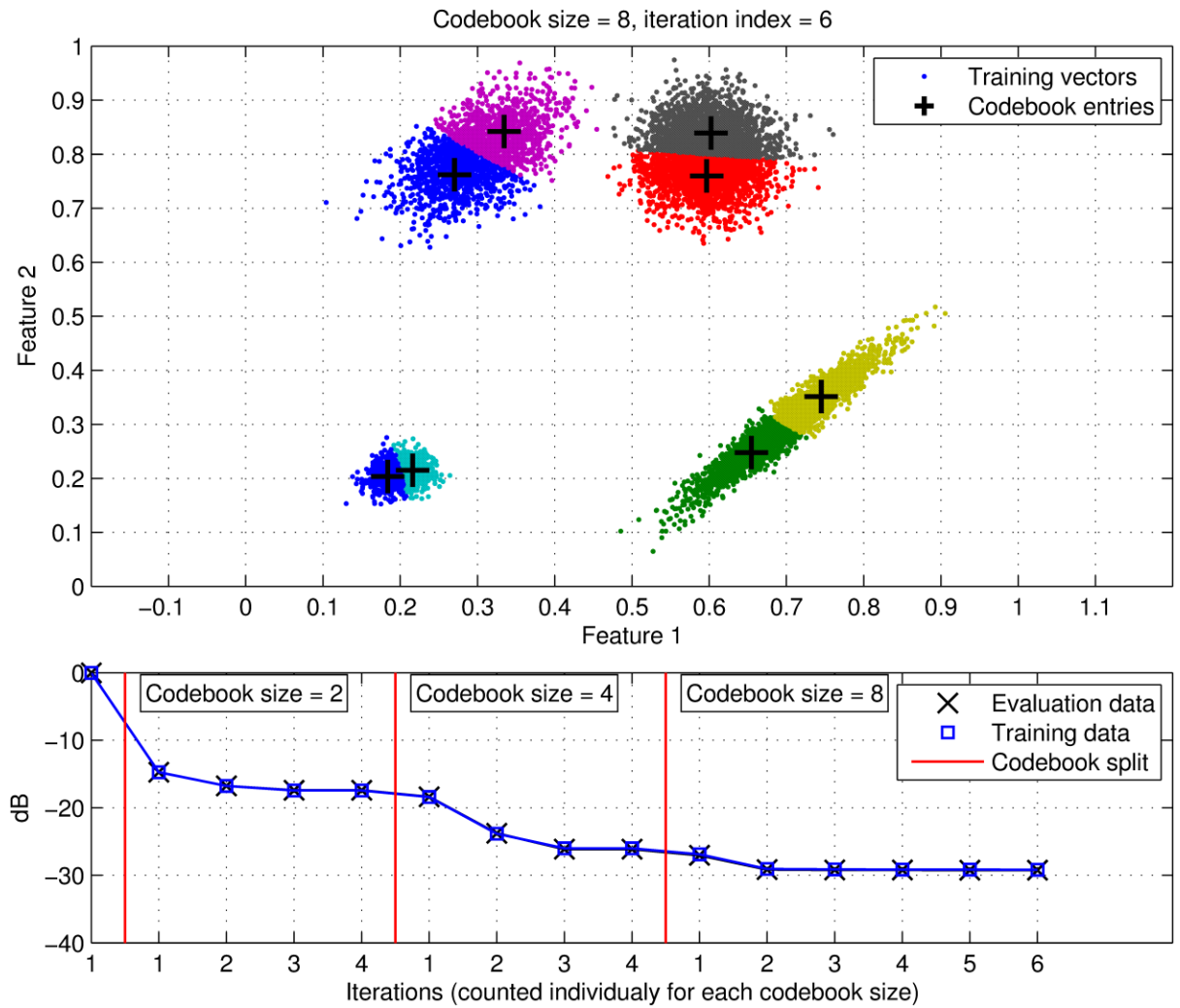
Codebook of size 8, after 5. iteration:



Codebook Training

LBG-Algorithm – Example, Part 16

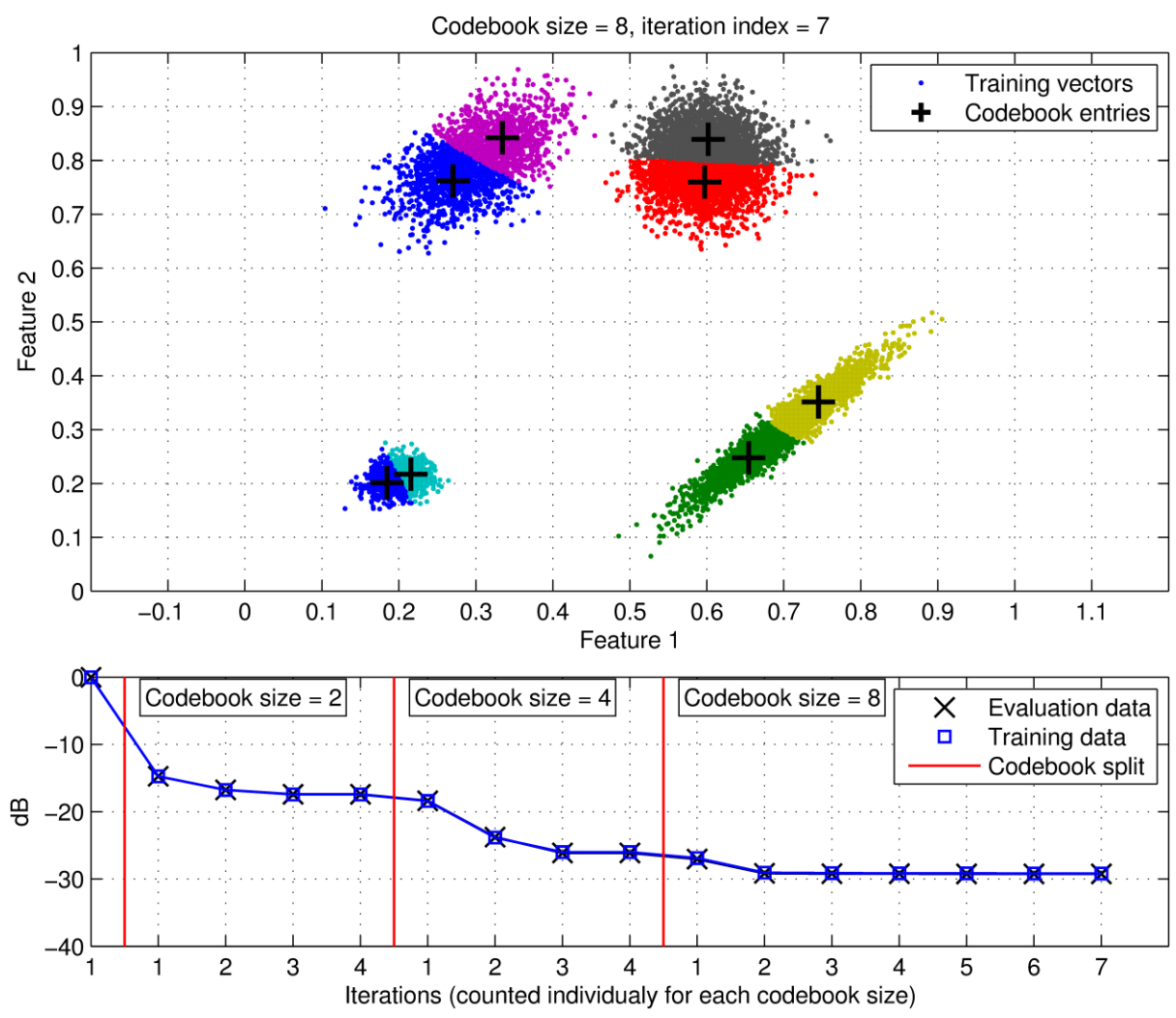
Codebook of size 8, after 6. iteration:



Codebook Training

LBG-Algorithm – Example, Part 17

Codebook of size 8, after 7. iteration:



Determining the Codebook Size

“Elbow” method:

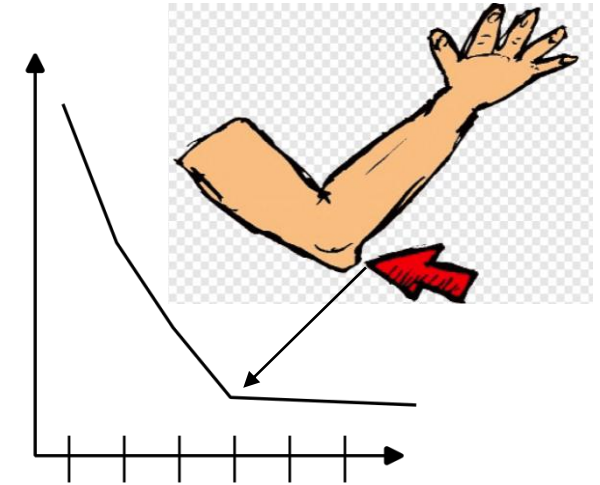
□ Principle idea

□ Find the size with maximal curvature

$$Cur(K) = \frac{\overline{d(p_\infty)}^{(K-1)} - \overline{d(p_\infty)}^{(K)}}{\overline{d(p_\infty)}^{(K)} - \overline{d(p_\infty)}^{(K+1)}}$$

with $\overline{d(p_\infty)}^{(K)}$ being the average distance at iteration p_∞ (after convergence) for a codebook of size K .


□ “Sharp elbow” for good detection needed.




Determining the Codebook Size

Alternative methods:

- ❑ Blog from Samuele Mazzanti (<https://medium.com/@mazzanti.sam>)
- ❑ Start was, that he asked Chat GPT



How should I chose k in k-means?



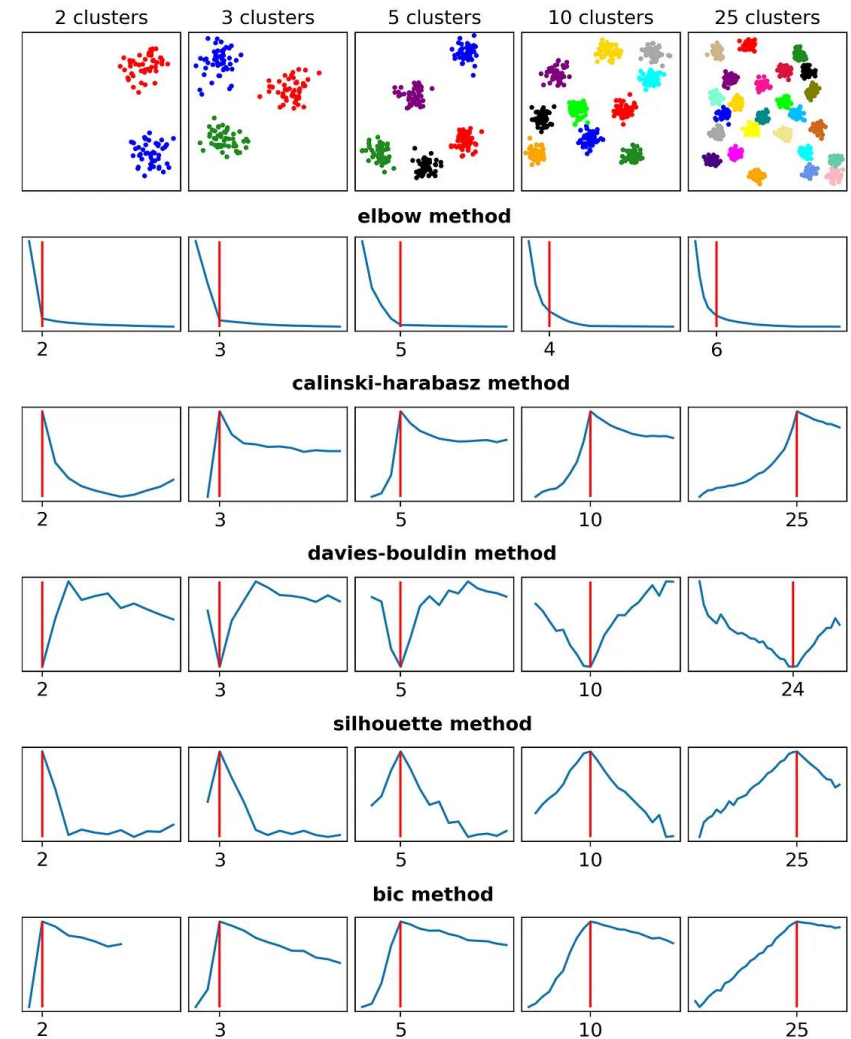
Choosing the right value of k (number of clusters) in k-means is a crucial step, as it significantly influences the results of the clustering algorithm. Here are several methods you can use to determine the optimal value of k:

- 1. Elbow Method:**
 - Run the k-means algorithm for a range of values of k (e.g., from 1 to a certain maximum value).
 - For each value of k, calculate the sum of squared distances from each point to its assigned center (inertia or distortion).
 - Plot the distortion for each k and look for the "elbow" point where the rate of decrease sharply changes. The point where the distortion starts to level off is often a good choice for k.

Determining the Codebook Size

Alternative methods:

- ❑ Blog from Samuele Mazzanti (<https://medium.com/@mazzanti.sam>)
- ❑ Start was, that he asked Chat GPT.
- ❑ Afterwards, he created artificial data and tested several other distance measures (see picture on the left [from Samuele Mazzanti]).
- ❑ Basic idea of the methods is that you estimate the average distance within each cluster and relate it to the difference to either the nearest neighboring cluster or to all other clusters. Such ratios should be maximized.



Extensions for the Codebook Generation

Extensions:

- ❑ In a next step, all codebook entries can be replaced by the *nearest training or evaluation vector*. This ensures that the codebook entries are „valid“ vectors (e.g., stable filter coefficients)
- ❑ An alternative for doubling all codebook entries in the LBG algorithm is to *split only that vector that contributes the „biggest“ part to the average distance*.
- ❑ If codebook pairs should be trained, the two feature vectors are appended first. By choosing the weighting matrix \mathbf{G} properly, either of the features can dominate the codebook generation. Alternatively, also a weighted sum can be used (both feature vectors contribute to the clustering)

Affine-linear mappings:

- Global approach:

$$\mathbf{y}(n) = \mathbf{M} [\mathbf{x}(n) - \mathbf{m}_x] + \mathbf{m}_y$$

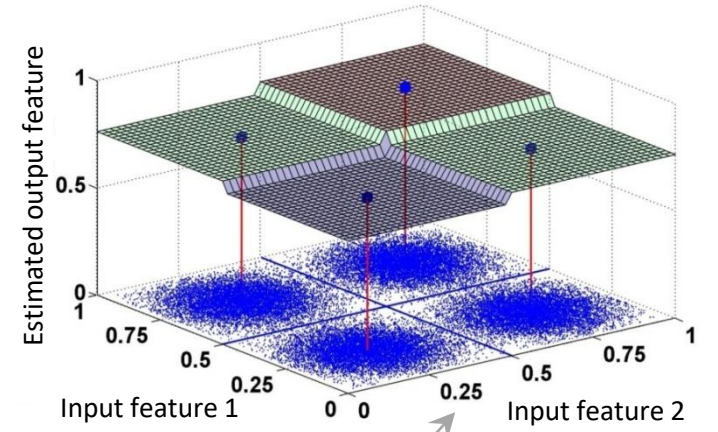
- Piecewise defined mapping:

$$\mathbf{y}(n) = \mathbf{M}_{i(n)} [\mathbf{x}(n) - \mathbf{m}_{x,i(n)}] + \mathbf{m}_{y,i(n)}$$

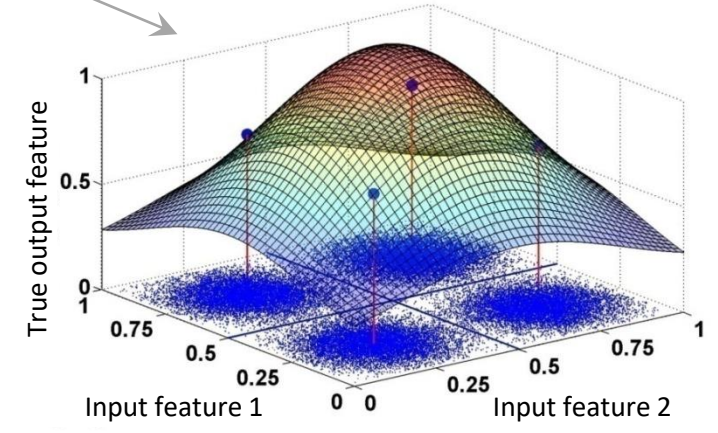
← *This can be seen as a generalized version of the codebook approach. The codebook approach would use a matrix of zeros for the matrix \mathbf{M} and the y -mean vector would be the best codebook entry.*

Combination of Codebook Approaches With Other Mappings – Part 2

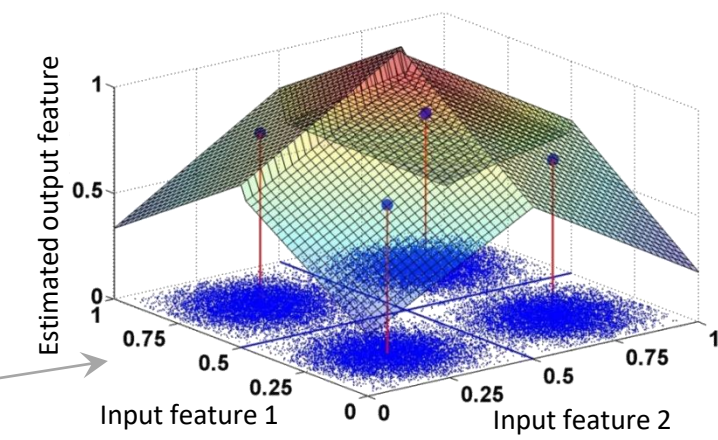
Example for the relation between input and output features



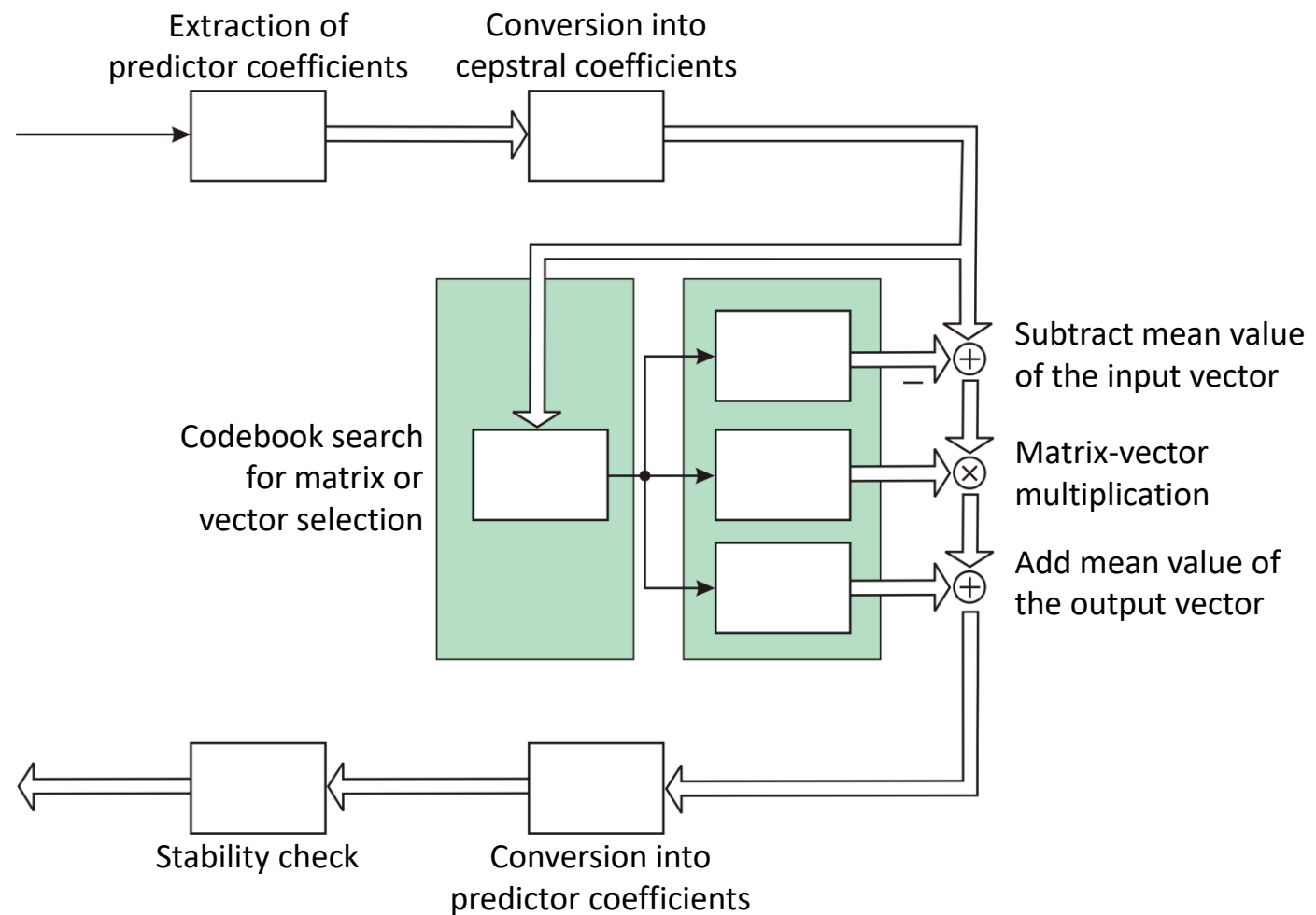
Approximation by codebook pairs



Example for a locally optimized linear mapping



Combination of Codebook Approaches With Other Mappings – Part 3



Summary and Outlook

Summary:

- ❑ Motivation
- ❑ Application examples
- ❑ Cost functions for the training of a codebook
- ❑ LBG- and k-means algorithm
 - ❑ Basic schemes
 - ❑ Extensions
- ❑ Combinations with additional mapping schemes

Next part:

- ❑ Bandwidth extension

