LPC Based Analysis and Clustering on Humpback Whale Vocalizations

Carlos A. Rueda

(draft)

September 17, 2019

Abstract

This document provides some basic background relevant to the clustering experiments performed on whale vocalizations reported at `https://github.com/ecoz2/ecoz2-whale-cb`. The processing of the signal is based on linear predictive coding. In each experiment, LPC analysis is applied on given whale sound files to generate corresponding training vectors for clustering. The training vector space is then clustered using a non-Euclidian version of K-means where an LPC-based distance function is used. Codebooks of incremental size are generated while capturing some metrics for evaluation and visualization, including average distortion as a function of codebook size; cluster cardinality and distortion for specific codebook sizes; and minimum distortion for each training vector to support visualization of cluster shapes based on the reflection coefficients associated with the prediction coefficients.

1 Linear Predictive Coding

Linear predictive analysis is a widely used techique in speech processing [1, 2]. On humpback whale vocalizations, [3] is a recent and relevant reference in that not only includes LPC as part of their analysis, but also, more generally, investigates the sub-unit structure of song units, which is one of the core motivations for the experiments in our work.

In what follows we introduce some minimal LPC background but hopefully sufficient to articulate the main aspects explored in the reported experiments at `https://github.com/ecoz2/ecoz2-whale-cb`. Further details can be found in the references.

Let $\mathbf{s} = [s_1, s_2, \ldots, s_L]$ be the discrete signal representing a certain whale vocalization or series of vocalizations. To obtain a more compact representation of the signal, yet not loosing significant information, we use *linear predictive coding* (LPC) analysis. This analysis is applied on consecutive, overlapping time windows where the signal in each window is assumed to be approximately stationary (i.e., statistics within the window do not change as a function of time). Let T be the number of windows extracted from $\mathbf{s}$, and $\mathbf{x} = [x_1, x_2, \ldots, x_N]$ denote one of such windows. The core idea in LPC analysis is to estimate each sample value x_n as a linear combination of a number of previous sample values:

$$\hat{x}_n = \sum_{i=1}^{P} a_i x_{n-i}$$

where P is the *order of the prediction*. So, the error at the n-th sample is:

$$e_n = x_n - \hat{x}_n = x_n - \sum_{i=1}^{P} a_i x_{n-i}$$

and the goal is to find the a_i coefficients that minimize the squared error over the window:

$$E = \sum_n e_n^2 = \sum_n (x_n - \sum_{i=1}^{P} a_i x_{n-i})^2$$

It is possible to solve this problem with a variety of methods and depending on particular signal characteristics (e.g., voiced or unvoiced). The software[1] used in our experiments apply the *autocorrelation method* as described in [2]. The details are out of the scope of this document but we will note that this error can be expressed in matrix form as:

$$E = \mathbf{a}'\mathbf{R}\mathbf{a} \tag{1}$$

where $'$ denotes the transpose operation and $\mathbf{R}$ is the the autocorrelation matrix of window $\mathbf{x}$. This expression is used below as the basis to measure the distance between two prediction vectors.

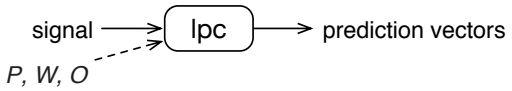

Figure 1: The LPC analysis step applied on a signal resulting in a number of prediction vectors. P is the order of prediction, and W and O are the window size and offset in ms.

So, by applying LPC analyis on each of the windows $\mathbf{x}$ from the input signal we obtain the corresponding prediction coefficients, $\mathbf{a} = [a_1, a_2, \ldots, a_P]$. Fig. 1 is a schematic of this step.

[1] `https://github.com/ecoz2/ecoz2`, developed as part of [4].

2 Clustering

The set of all LPC vectors extracted from the input signal will comprise the training set for clustering. We closely follow Juang et al [5] for the implementation of the clustering algorithm as well as in terms of evaluation based on distortion performance.

A core operation required for clustering is measuring the distance or distortion $d(\mathbf{a}, \mathbf{b})$ between two LPC vectors $\mathbf{a}$ and $\mathbf{b}$. Since the vectors are coming from linear predictive analysis, we use a measure based on the prediction error itself according to (1). More specifically, noting that (1) is the minimum prediction error found for a particular window $\mathbf{x}$ from which $\mathbf{a}$ was obtained, we can also perform a similar calculation, still using the same autocorrelation matrix $\mathbf{R}$ for $\mathbf{x}$, but involving any other prediction vector $\mathbf{b}$:

$$E_{\mathbf{b}} = \mathbf{b}'\mathbf{R}\mathbf{b} \tag{2}$$

Since $E_{\mathbf{b}} \geq E_{\mathbf{a}}$, we can define the distorion measure as:

$$d(\mathbf{a}, \mathbf{b}) \quad = \quad E_{\mathbf{b}}/E_{\mathbf{a}} - 1 \tag{3}$$

In part because the autocorrelation matrix $\mathbf{R}$ is symmetric and *Toeplitz*,[2] it can be shown that the prediction error given in (2) can be expressed as the scalar product:

$$E_{\mathbf{b}} = \mathbf{r}_{\mathbf{x}}[0]\mathbf{r}_{\mathbf{b}}[0] + 2\sum_{p=1}^{P} \mathbf{r}_{\mathbf{x}}[p]\mathbf{r}_{\mathbf{b}}[p] \tag{4}$$

where $\mathbf{r}_{\mathbf{x}}$ is the autocorrelation sequence of $\mathbf{x}$ and $\mathbf{r}_{\mathbf{b}}$ is the autocorrelation sequence of the prediction vector $\mathbf{b}$. Using this in (3) we get [5]:

$$d(\mathbf{a}, \mathbf{b}) = \frac{\mathbf{r}_{\mathbf{x}}[0]}{E_{\mathbf{a}}}\mathbf{r}_{\mathbf{b}}[0] + 2\sum_{p=1}^{P} \frac{\mathbf{r}_{\mathbf{x}}[p]}{E_{\mathbf{a}}}\mathbf{r}_{\mathbf{b}}[p] - 1 \tag{5}$$

With the distortion measure function now established, let us now proceed with the clustering step. Let $\mathbf{A} = [\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_T]$ be the set of prediction coefficient vectors that we want to cluster. The clustering method used here is a variant of the K-means algorithm [6, 2]. With $\mathbf{A}$ as input and a threshold parameter ϵ used as a convergence criterium (i.e., a change in average distortion –see below– less than this threshold stops the iterative refinement of the codebook),

[2] `https://en.wikipedia.org/wiki/Toeplitz_matrix`

the method generates codebooks of incremental size. For a given size K, the method finds the set of centroids $\mathbf{C} = [\mathbf{c}_1, \mathbf{c}_2, \ldots, \mathbf{c}_K]$ that best represent the vector space associated with the training set according to the distortion measure, that is, one that minimizes the average distortion:

$$D_K = \frac{1}{T} \sum_{\mathbf{a} \in \mathbf{A}} \min_{k=1}^{K} d(\mathbf{a}, \mathbf{c}_k) \tag{6}$$

The method first creates a codebook of size $K = 2$ using K-means. Once the average distortion is sufficiently small for this size according to the ϵ parameter, the codebook is doubled in size (via splitting each centroid into two) and the K-means algorithm is performed on the new size $K = 4$. This procedure is repeated until a maximum codebook size is reached. Fig. 2 illustrates the clustering step.

Figure 2: The clustering step applied on a set of prediction vectors resulting in a number of codebooks of incremental size.

Codebooks typically become the basis to subsequently perform *vector quantization*[3], which is used in a variety of applications. Here, we are mainly focused on the clustering process itself and the inspection and evaluation of the generated codebooks.

2.1 Codebook evaluation

The evaluation in our experiments is of an *internal*[4] nature. As a function of codebook size K, we capture and inspect the following values:

- Average (intra-cluster) distortion D_K, as defined in (6);
- σ-ratio, the ratio between the inter-cluster and intra-cluster average distortions [7]:

$$\sigma_K = \frac{\frac{1}{K} \sum_{i=1}^{K} \frac{1}{K-1} \sum_{j=1}^{K} d(\mathbf{c}_i, \mathbf{c}_j)}{D_K};$$

[3] Vector quantization basically consists in mapping any given P-order vector $\mathbf{a}$ (not necessarily from the training set) to the index $k_\mathbf{a}$ in a codebook $\mathbf{C}$ of size K such that $k_\mathbf{a} = \operatorname{argmin}_{k=1}^{K} d(\mathbf{a}, \mathbf{c}_k)$.

[4] `https://en.wikipedia.org/wiki/Cluster_analysis#Evaluation_and_assessment`

- *Inertia* ("within-cluster sum-of-squares criterion"), definition adapted from [8]:

$$\mathbf{inertia}_K = \sum_{\mathbf{a} \in \mathbf{A}} \min_{k=1}^{K} d(\mathbf{a}, \mathbf{c}_k)^2.$$

Fig. 3, taken from one of the reported experiments, shows a typical plot of these values as a function of codebook size.

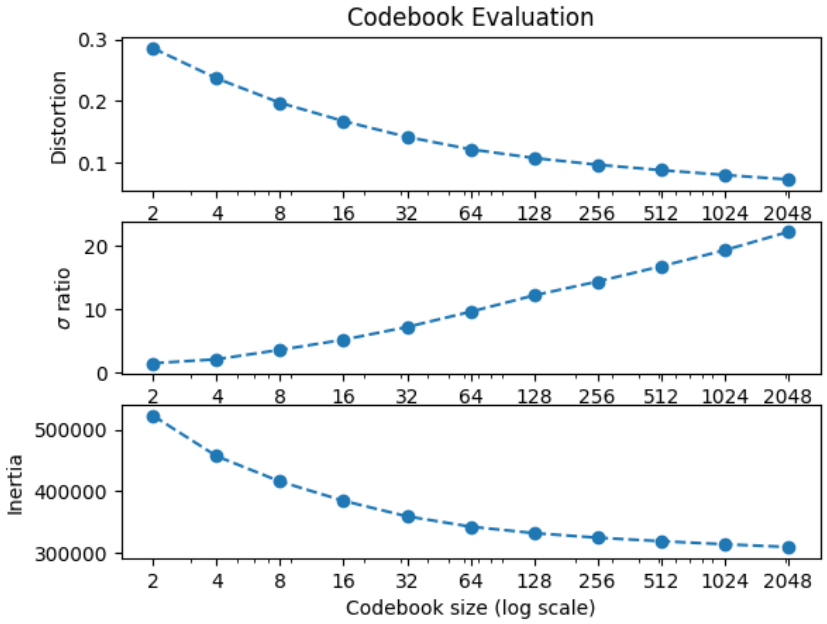

Figure 3: A plot of average distortion, σ-ratio, and inertia.

For each cluster we also track its cardinality (number of training vectors in each cluster) and average distortion [7]. Fig. 4 shows a resulting scatter plot from one of the experiments with a codebook of size $M = 1024$.

Cluster visualization

Instead of visualizing the space of the prediction coefficient vectors **a** directly, we follow [5] to look into the corresponding *reflection coefficient* vector space. Fig. 5 plots the first two reflection coefficients, k_1 and k_2, taken from the training vector set as well as from the resulting codebook ($M = 128$). A similar plot but with the first three reflection coefficients is shown in Fig. 6.

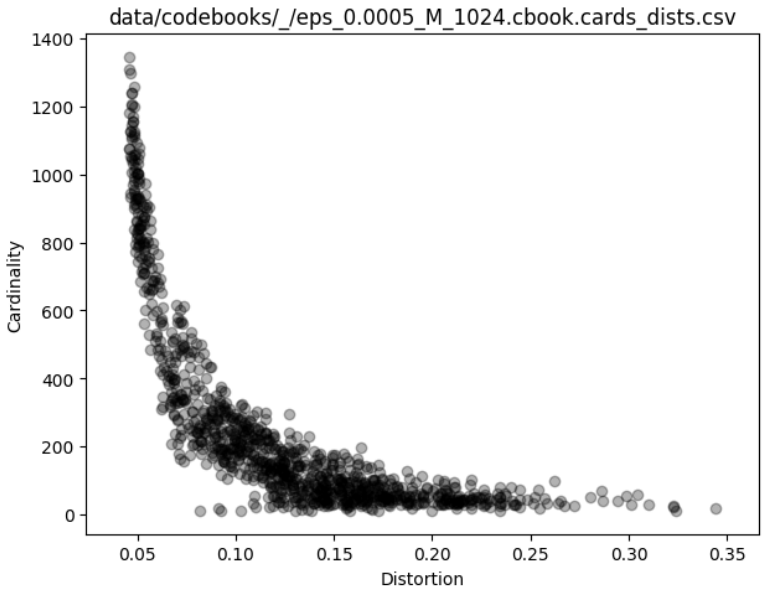

Figure 4: A typical distortion vs. cardinality scatter plot (the greater the cluster distortion, the lower its cardinality).

Codeword assignment visualization

The goal here is to color-map the assigned codewords over time while expecting to see some similarity when comparing regions with similar acoustic content. Intervals were selected by visually inspecting the spectrogram of the input signal using Audacity.[5] Each desired interval is defined by a start time (wrt to beginning of the file), e.g., `2h0.941s` (2 hours and 0.941 seconds), and a duration e.g., `4.057s`.

Each of the plots generated for this purpose consists of:

- Spectrogram of the selected interval.
- Corresponding codeword per time window displayed with an arbitrary color mapping.
- Corresponding distortion per time window.

With quantization against a $M = 512$ codebook, Fig. 7 is an example of one of these plots.

[5] `https://www.audacityteam.org`

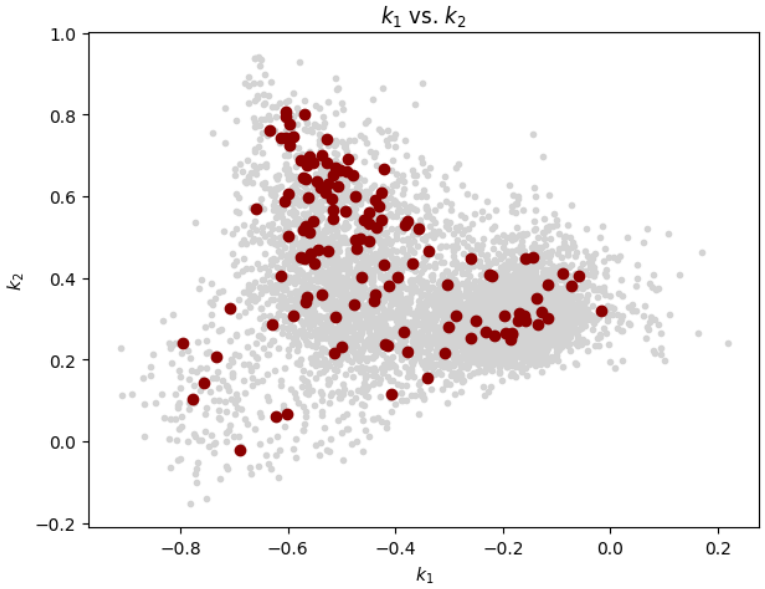

Figure 5: k_1-k_2 scatter plot showing values from training vectors (gray dots) and codebook centroids (red dots). Codebook size $M = 128$.

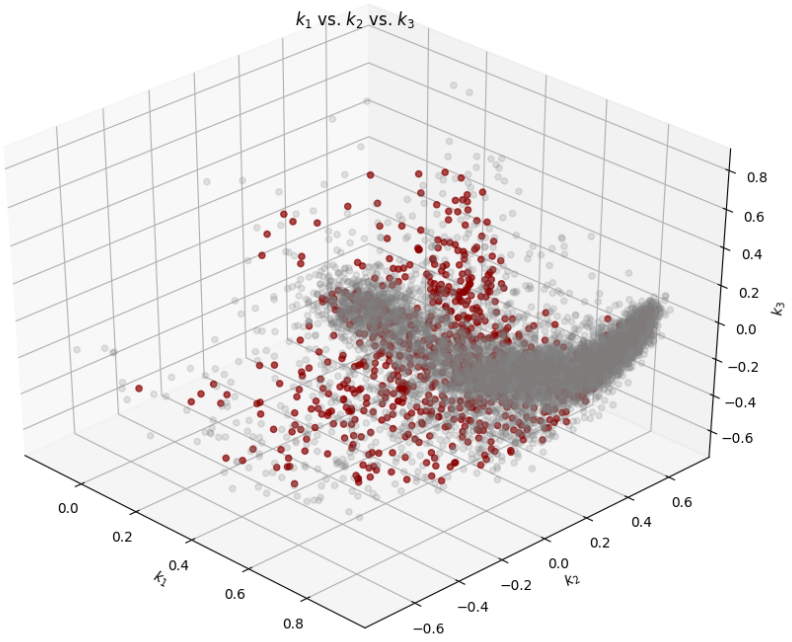

Figure 6: k_1-k_2-k_3 scatter plot showing values from training vectors (gray dots) and codebook centroids (red dots). Codebook size $M = 1024$.

.

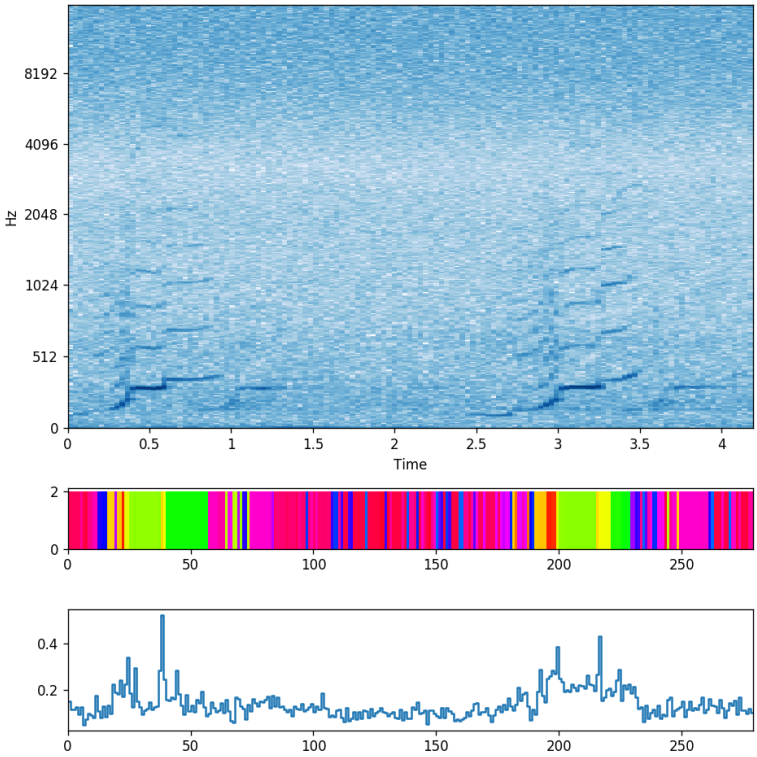

Figure 7: Example of codeword assignment visualization for a particular time interval. In general, similar coloring is expected to be observed for similar spectrogram features. (The axes of the codeword and distortion subplots are in terms of the window index, but aligned with the time axis of the spectrogram subplot.)

References

[1] Lawrence R. Rabiner and Ronald W. Schafer. Introduction to digital speech processing. *Found. Trends Signal Process.*, 1(1):1–194, January 2007.

[2] T. Parsons. *Voice and speech processing*. McGraw-Hill, 1987.

[3] Howard Pines. Mapping the phonetic structure of humpback whale song units: extraction, classification, and Shannon-Zipf confirmation of sixty sub-units. *Proceedings of Meetings on Acoustics*, 35(1):010003, 2018.

[4] C.A. Rueda. *Implementation and experimentation with speech recognition –isolated digits– using LPC vector quantization and hidden Markov models*. BS. Thesis, Systems Engineering Dept. Universidad Autónoma de Manizales, 1993.

[5] B-H. Juang, D.Y. Wong, and A.H. Gray. Distortion performance of vector quantization for LPC voice coding. *IEEE Trans. ASSP*, 30(2), 1982.

[6] Y. Linde, A. Buzo, and R. M. Gray. An algorithm for vector quantizer design. *IEEE Trans. Communications*, 28(1):84–95, 1980.

[7] L. R. Rabiner, S. E. Levinson, and M. M. Sondhi. On the application of vector quantization and hidden Markov models to speaker-independent, isolated word recognition, April 1983.

[8] scikit-learn section on K-means. `https://scikit-learn.org/stable/modules/clustering.html#k-means`. Accessed: 2019-09-01.