Fast and Robust Archetypal Analysis for Representation Learning

Yuansi Chen, Julien Mairal, Zaid Harchaoui

Fast and Robust Archetypal Analysis for Representation Learning

Yuansi Chen[1,2], Julien Mairal[2], Zaid Harchaoui[2]
[1]EECS Department, University of California, Berkeley, [2]Inria*
[1]yuansi.chen@berkeley.edu, [2]firstname.lastname@inria.fr

Abstract

We revisit a pioneer unsupervised learning technique called archetypal analysis [5], which is related to successful data analysis methods such as sparse coding [18] and non-negative matrix factorization [19]. Since it was proposed, archetypal analysis did not gain a lot of popularity even though it produces more interpretable models than other alternatives. Because no efficient implementation has ever been made publicly available, its application to important scientific problems may have been severely limited.

Our goal is to bring back into favour archetypal analysis. We propose a fast optimization scheme using an active-set strategy, and provide an efficient open-source implementation interfaced with Matlab, R, and Python. Then, we demonstrate the usefulness of archetypal analysis for computer vision tasks, such as codebook learning, signal classification, and large image collection visualization.

1. Introduction

Unsupervised learning techniques have been widely used to automatically discover the underlying structure of data. This may serve several purposes, depending on the task considered. In experimental sciences, one may be looking for data representations that automatically exhibit interpretable patterns, for example groups of neurons with similar activation in a population, clusters of genes manifesting similar expression [7], or topics learned from text collections [3].

In image processing and computer vision, unsupervised learning is often used as a data modeling step for a subsequent prediction task. For example, natural image patches have been modeled with sparse coding [18] or mixture of Gaussians [25], yielding powerful representations for image restoration. Similarly, local image descriptors have been encoded with unsupervised learning methods [4, 12, 24], producing successful codebooks for visual recognition pipelines. Interpretation is probably not crucial for these prediction tasks. However, it can be important for other purposes, *e.g.*, for data visualization.

Our main objective is to rehabilitate a pioneer unsupervised learning technique called archetypal analysis [5], which is easy to interpret while providing good results in prediction tasks. It was proposed as an alternative to principal component analysis (PCA) for discovering latent factors from high-dimensional data. Unlike principal components, each factor learned by archetypal analysis, called *archetype*, is forced to be a convex combination of a few data points. Such associations between archetypes and data points are useful for interpretation. For example, clustering techniques provide such associations between data and centroids. It is indeed common in genomics to cluster gene expression data from several individuals, and to interpret each centroid by looking for some common physiological traits among individuals of the same cluster [7].

Interestingly, archetypal analysis is related to popular approaches such as sparse coding [18] and non-negative matrix factorization (NMF) [19], even though all these formulations were independently invented around the same time. Archetypal analysis indeed produces sparse representations of the data points, by approximating them with convex combinations of archetypes; it also provides a non-negative factorization when the data matrix is non-negative.

A natural question is why archetypal analysis did not gain a lot of success, unlike NMF or sparse coding. We believe that the lack of efficient available software has limited the deployment of archetypal analysis to promising applications; our goal is to address this issue. First, we develop an efficient optimization technique based on an active-set strategy [17]. Then, we demonstrate that our approach is scalable and orders of magnitude faster than existing publicly available implementations. Finally, we show that archetypal analysis can be useful for computer vision, and we believe that it could have many applications in other fields, such as neurosciences, bioinformatics, or natural language processing. We first show that it performs as well as sparse coding for learning codebooks of features in visual recognition tasks [24] and for signal classification [14, 21, 23]. Second, we show that archetypal analysis provides a simple and effective way for visualizing large databases of images.

*LEAR team, Inria Grenoble Rhône-Alpes, Laboratoire Jean Kuntzmann, CNRS, Univ. Grenoble Alpes.

This paper is organized as follows: in Section 2, we present the archetypal analysis formulation; Section 3 is devoted to optimization techniques; Section 4 presents successful applications of archetypal analysis to computer vision tasks, and Section 5 concludes the paper.

2. Formulation

Let us consider a matrix $\mathbf{X} = [\mathbf{x}_1, \ldots, \mathbf{x}_n]$ in $\mathbb{R}^{m \times n}$, where each column $\mathbf{x}_i$ is a vector in $\mathbb{R}^m$ representing some data point. Archetypal analysis learns a factorial representation of $\mathbf{X}$; it looks for a set of p archetypes $\mathbf{Z} = [\mathbf{z}_1, \ldots, \mathbf{z}_p]$ in $\mathbb{R}^{m \times p}$ under two geometrical constraints: each data vector $\mathbf{x}_i$ should be well approximated by a convex combination of archetypes, and each archetype $\mathbf{z}_j$ should be a convex combination of data points $\mathbf{x}_i$. Therefore, given a set of archetypes $\mathbf{Z}$, each vector $\mathbf{x}_i$ should be close to a product $\mathbf{Z}\boldsymbol{\alpha}_i$, where $\boldsymbol{\alpha}_i$ is a coefficient vector in the simplex Δ_p:

$$\Delta_p \triangleq \left\{ \boldsymbol{\alpha} \in \mathbb{R}^p \;\text{ s.t. }\; \boldsymbol{\alpha} \geq 0 \;\text{ and }\; \textstyle\sum_{j=1}^{p} \boldsymbol{\alpha}[j] = 1 \right\}. \quad (1)$$

Similarly, for every archetype $\mathbf{z}_j$, there exists a vector $\boldsymbol{\beta}_j$ in Δ_n such that $\mathbf{z}_j = \mathbf{X}\boldsymbol{\beta}_j$, where Δ_n is defined as in (1) by replacing p by n. Then, archetypal analysis is defined as a matrix factorization problem:

$$\min_{\substack{\boldsymbol{\alpha}_i \in \Delta_p \text{ for } 1 \leq i \leq n \\ \boldsymbol{\beta}_j \in \Delta_n \text{ for } 1 \leq j \leq p}} \|\mathbf{X} - \mathbf{X}\mathbf{B}\mathbf{A}\|_{\mathrm{F}}^2, \quad (2)$$

where $\mathbf{A} = [\boldsymbol{\alpha}_1, \ldots, \boldsymbol{\alpha}_n]$, $\mathbf{B} = [\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_p]$, and $\|.\|_{\mathrm{F}}$ denotes the Frobenius norm; the archetypes $\mathbf{Z}$ are represented by the product $\mathbf{Z} = \mathbf{X}\mathbf{B}$. Solving (2) is challenging since the optimization problem is non-convex; this issue will be addressed in Section 3. Interestingly, the formulation (2) is related to other approaches, which we briefly review here.

Non-negative matrix factorization (NMF) [19]. Assume that the data $\mathbf{X}$ is non-negative. NMF seeks for a factorization of $\mathbf{X}$ into two non-negative components:

$$\min_{\mathbf{Z} \in \mathbb{R}_+^{m \times p}, \mathbf{A} \in \mathbb{R}_+^{p \times n}} \|\mathbf{X} - \mathbf{Z}\mathbf{A}\|_{\mathrm{F}}^2. \quad (3)$$

Similarly, the matrices $\mathbf{Z}$ and $\mathbf{A}$ in archetypal analysis are also non-negative when $\mathbf{X}$ is itself non-negative. The difference between NMF and archetypal analysis is that the latter involves simplicial constraints.

Sparse Coding [18]. Given a fixed set of archetypes $\mathbf{Z}$ in $\mathbb{R}^{m \times p}$, each data point $\mathbf{x}_i$ is approximated by $\mathbf{Z}\boldsymbol{\alpha}_i$, under the constraint that $\boldsymbol{\alpha}_i$ is non-negative and sums to one. In other words, the ℓ_1-norm of $\boldsymbol{\alpha}_i$ is constrained to be one, which has a sparsity-inducing effect [1]. Thus, archetypal analysis produces sparse approximations of the input data, and archetypes play the same role as the "dictionary elements" in the following sparse coding formulation of [18]:

$$\min_{\substack{\mathbf{Z} \in \mathbb{R}^{m \times p}, \\ \mathbf{A} \in \mathbb{R}^{p \times n}}} \frac{1}{2}\|\mathbf{X} - \mathbf{Z}\mathbf{A}\|_{\mathrm{F}}^2 + \lambda \|\mathbf{A}\|_1 \;\text{ s.t. }\; \|\mathbf{z}_j\|_2 \leq 1 \;\; \forall j. \quad (4)$$

Since the ℓ_1-norm is related to the simplicial constraints $\boldsymbol{\alpha}_i \in \Delta_p$—the non-negativity constraints being put aside—the main difference between sparse coding and archetypal analysis is the fact that archetypes should be convex combinations of the data points $\mathbf{X}$. As a result, the vectors $\boldsymbol{\beta}_j$ are constrained to be in the simplex Δ_n, which encourages them to be sparse. Then, each archetype $\mathbf{z}_j$ becomes a linear combination of a few data points only, which is useful for interpreting $\mathbf{z}_j$. Moreover, the non-zero entries in $\boldsymbol{\beta}_j$ indicate in which proportions the input data points $\mathbf{x}_i$ are related to each archetype $\mathbf{z}_j$.

Another variant of sparse coding called "local coordinate coding" [26] is also related to archetypal analysis. In this variant, dictionary elements are encouraged to be close to the data points that uses them in their decompositions. Then, dictionary elements can be interpreted as anchor points on a manifold representing the data distribution.

2.1. Robust Archetypal Analysis

In some applications, it is desirable to automatically handle outliers—that is, data points $\mathbf{x}_i$ that significantly differ from the rest of the data. In order to make archetypal analysis robust, we propose the following variant:

$$\min_{\substack{\boldsymbol{\alpha}_i \in \Delta_p \text{ for } 1 \leq i \leq n \\ \boldsymbol{\beta}_j \in \Delta_n \text{ for } 1 \leq j \leq p}} \sum_{i=1}^{n} h\left(\|\mathbf{x}_i - \mathbf{X}\mathbf{B}\boldsymbol{\alpha}_i\|_2\right), \quad (5)$$

where $h : \mathbb{R} \mapsto \mathbb{R}$ is the Huber loss function, which is often used as a robust replacement of the squared loss in robust statistics [11]. It is defined here for any scalar u in $\mathbb{R}$ as

$$h(u) = \begin{cases} \frac{u^2}{2\varepsilon} + \frac{\varepsilon}{2} & \text{if } |u| \leq \varepsilon \\ |u| & \text{otherwise} \end{cases}, \quad (6)$$

and ε is a positive constant. Whereas the cost associated to outliers in the original formulation (2) can be large since it grows quadratically, the Huber cost only grows linearly. In Section 3, we present an effective iterative reweighted least-square strategy to deal with the Huber loss.

3. Optimization

The formulation (2) is non-convex, but it is convex with respect to one of the variables $\mathbf{A}$ or $\mathbf{B}$ when the other one is fixed. It is thus natural to use a block-coordinate descent scheme, which is guaranteed to asymptotically provide a stationary point of the problem [2]. We present such a strategy in Algorithm 1. As noticed in [5], when fixing all variables but a column $\boldsymbol{\alpha}_i$ of $\mathbf{A}$ and minimizing with respect

Algorithm 1 Archetypal Analysis

1: **Input:** Data $\mathbf{X}$ in $\mathbb{R}^{m\times n}$; p (number of archetypes); T (number of iterations);
2: Initialize $\mathbf{Z}$ in $\mathbb{R}^{m\times p}$ with random columns from $\mathbf{X}$;
3: Initialize $\mathbf{B}$ such that $\mathbf{Z}=\mathbf{XB}$;
4: **for** $t=1\ldots,T$ **do**
5: **for** $i=1\ldots,n$ **do**
6: $\boldsymbol{\alpha}_i \in \arg\min_{\boldsymbol{\alpha}\in\Delta_p} \|\mathbf{x}_i-\mathbf{Z}\boldsymbol{\alpha}\|_2^2$;
7: **end for**
8: $\mathbf{R}\leftarrow\mathbf{X}-\mathbf{ZA}$;
9: **for** $j=1\ldots,p$ **do**
10: $\boldsymbol{\beta}_j \in \arg\min_{\boldsymbol{\beta}\in\Delta_n} \left\|\frac{1}{\|\boldsymbol{\alpha}^j\|_2^2}\mathbf{R}\boldsymbol{\alpha}^{j\top}+\mathbf{z}_j-\mathbf{X}\boldsymbol{\beta}\right\|_2^2$;
11: $\mathbf{R}\leftarrow\mathbf{R}+(\mathbf{z}_j-\mathbf{X}\boldsymbol{\beta}_j)\boldsymbol{\alpha}^j$;
12: $\mathbf{z}_j\leftarrow\mathbf{X}\boldsymbol{\beta}_j$;
13: **end for**
14: **end for**
15: **Return** $\mathbf{A}$, $\mathbf{B}$ (decomposition matrices).

to $\boldsymbol{\alpha}_i$, the problem to solve is a quadratic program (QP):

$$\min_{\boldsymbol{\alpha}_i\in\Delta_p} \|\mathbf{x}_i-\mathbf{Z}\boldsymbol{\alpha}_i\|_2^2. \tag{7}$$

These updates are carried out on Line 6 of Algorithm 1. Similarly, when fixing all variables but one column $\boldsymbol{\beta}_j$ of $\mathbf{B}$, we also obtain a quadratic program:

$$\min_{\boldsymbol{\beta}_j\in\Delta_n} \|\mathbf{X}-\mathbf{XB}_{\text{old}}\mathbf{A}+\mathbf{X}(\boldsymbol{\beta}_{j,\text{old}}-\boldsymbol{\beta}_j)\boldsymbol{\alpha}^j\|_{\text{F}}^2, \tag{8}$$

where $\boldsymbol{\beta}_{j,\text{old}}$ is the current value of $\boldsymbol{\beta}_j$ before the update, and $\boldsymbol{\alpha}^j$ in $\mathbb{R}^{1\times n}$ is the j-th row of $\mathbf{A}$. After a short calculation, problem (8) can be equivalently rewritten

$$\min_{\boldsymbol{\beta}_j\in\Delta_n} \left\|\frac{1}{\|\boldsymbol{\alpha}^j\|_2^2}\left(\mathbf{X}-\mathbf{XB}_{\text{old}}\mathbf{A}\right)\boldsymbol{\alpha}^{j\top}+\mathbf{X}\boldsymbol{\beta}_{j,\text{old}}-\mathbf{X}\boldsymbol{\beta}_j\right\|_2^2, \tag{9}$$

which has a similar form as (7). This update is carried out on Line 10 of Algorithm 1. Lines 12 and 11 respectively update the archetypes and the residual $\mathbf{R}=\mathbf{X}-\mathbf{XBA}$. Thus, Algorithm 1 is a cyclic block-coordinate algorithm, which is guaranteed to converge to a stationary point of the optimization problem (2), see, e.g., [2]. The main difficulty to implement such a strategy is to find a way to efficiently solve quadratic programs with simplex constraints such as (7) and (9). We discuss this issue in the next section.

3.1. Efficient Solver for QP with Simplex Constraint

Both problems (7) and (9) have the same form, and thus we will focus on finding an algorithm for solving:

$$\min_{\boldsymbol{\alpha}\in\Delta_p}\left[f(\boldsymbol{\alpha})\triangleq\|\mathbf{x}-\mathbf{Z}\boldsymbol{\alpha}\|_2^2\right], \tag{10}$$

Algorithm 2 Active-Set Method

1: **Input:** matrix $\mathbf{Z}\in\mathbb{R}^{m\times p}$, vector $\mathbf{x}\in\mathbb{R}^m$;
2: Initialize $\boldsymbol{\alpha}_0\in\Delta_p$ with a feasible starting point;
3: Define $\mathcal{A}_0\leftarrow\{j \text{ s.t. } \boldsymbol{\alpha}_0[j]>0\}$;
4: **for** $k=0,1,2\ldots$ **do**
5: Solve (11) with $\mathcal{A}_k$ to find a step $\mathbf{q}_k$;
6: **if** $\mathbf{q}_k$ is 0 **then**
7: Compute $\nabla f(\boldsymbol{\alpha}_k)=-2\mathbf{Z}^\top(\mathbf{x}-\mathbf{Z}\boldsymbol{\alpha}_k)$;
8: **if** $\nabla f(\boldsymbol{\alpha}_k)[j]>0$ for all $j\notin\mathcal{A}_k$ **then**
9: **Return** $\boldsymbol{\alpha}^*=\boldsymbol{\alpha}_k$ (solution is optimal).
10: **else**
11: $j^\star\leftarrow\min_{j\notin\mathcal{A}_k}\nabla f(\boldsymbol{\alpha}_k)[j]$;
12: $\mathcal{A}_{k+1}\leftarrow\mathcal{A}_k\cup\{j^\star\}$;
13: **end if**
14: **else**
15: $\gamma_k\leftarrow\max_{\gamma\in[0,1]}[\gamma \text{ s.t. } \boldsymbol{\alpha}_k+\gamma\mathbf{q}_k\in\Delta_p]$;
16: **if** $\gamma_k<1$ **then**
17: Find $j^\star$ such that $\boldsymbol{\alpha}_k[j]+\gamma_k\mathbf{q}_k[j]=0$;
18: $\mathcal{A}_{k+1}\leftarrow\mathcal{A}_k\setminus\{j^\star\}$;
19: **else**
20: $\mathcal{A}_{k+1}\leftarrow\mathcal{A}_k$;
21: **end if**
22: $\boldsymbol{\alpha}_{k+1}\leftarrow\boldsymbol{\alpha}_k+\gamma_k\mathbf{q}_k$;
23: **end if**
24: **end for**

which is a smooth (least-squares) optimization problem with a simplicial constraint. Even though generic QP solvers could be used, significantly faster convergence can be obtained by designing a dedicated algorithm that can leverage the underlying "sparsity" of the solution [1].

We propose to use an active-set algorithm [17] that can benefit from the solution sparsity, when carefully implemented. Indeed, at the optimum, most often only a small subset $\mathcal{A}$ of the variables will be non-zero. Active-set algorithms [17] can be seen as an aggressive strategy that can leverage this property. Given a current estimate $\boldsymbol{\alpha}$ in Δ_p at some iteration, they define a subset $\mathcal{A}=\{j \text{ s.t. } \boldsymbol{\alpha}[j]>0\}$, and find a direction $\mathbf{q}$ in $\mathbb{R}^p$ by solving the reduced problem

$$\min_{\mathbf{q}\in\mathbb{R}^p}\|\mathbf{x}-\mathbf{Z}(\boldsymbol{\alpha}+\mathbf{q})\|_2^2 \text{ s.t. } \sum_{j=1}^p\mathbf{q}[j]=0 \text{ and } \mathbf{q}_{\mathcal{A}^C}=0, \tag{11}$$

where $\mathcal{A}^C$ denotes the complement of $\mathcal{A}$ in the index set $\{1\ldots,p\}$. Then, a new estimate $\boldsymbol{\alpha}'=\boldsymbol{\alpha}+\gamma\mathbf{q}$ is obtained by moving $\boldsymbol{\alpha}$ onto the direction $\mathbf{q}$—that is, choosing γ in $[0,1]$, such that $\boldsymbol{\alpha}'$ remains in Δ_p. The algorithm modifies the set $\mathcal{A}$ until the algorithm finds an optimal solution in Δ_p. This strategy is detailed in Algorithm 2. Open-source active-set solvers for generic QP exist, *e.g.*, quadprog in Matlab, but we have found them too slow for our purpose. Instead, a dedicated implementation has proven to

be much more efficient. More precisely, we use some tricks inspired from the Lasso solver of the SPAMS toolbox [15]: (i) initialize $\mathcal{A}$ with a single variable; (ii) update at each iteration the quantity $(\mathbf{Z}_{\mathcal{A}}^\top \mathbf{Z}_{\mathcal{A}})^{-1}$ by using Woodbury formula; (iii) implicitly working with the matrix $\mathbf{Q} = \mathbf{Z}^\top \mathbf{Z}$ without computing it when updating $\boldsymbol{\beta}$.

As a resutl, each iteration of the active-set algorithm has a computational complexity of $O(mp + a^2)$ operations where a is the size of the set $\mathcal{A}$ at that iteration. Like the simplex algorithm for solving linear programs [17], the maximum number of iterations of the active-set algorithm can be exponential in theory, even though it is much smaller than $\min(m, p)$ in practice. Other approaches than the active-set algorithm could be considered, such as the fast iterative shrinkage-thresholding algorithm (FISTA) and the penalty approach of [5]. However, in our experiments, we have observed significantly better performance of the active-set algorithm, both in terms of speed and accuracy.

3.2. Optimization for Robust Archetypal Analysis

To deal with the Huber loss, we use the following variational representation of the Huber loss (see, [11]):

$$h(u) = \frac{1}{2} \min_{w \geq \varepsilon} \left[\frac{u^2}{w} + w \right]. \tag{12}$$

which is equivalent to (6). Then, robust archetypal analysis from Eq. (5) can be reformulated as

$$\min_{\substack{\boldsymbol{\alpha}_i \in \Delta_p \text{ for } 1 \leq i \leq n \\ \boldsymbol{\beta}_j \in \Delta_n \text{ for } 1 \leq j \leq p \\ w_i \geq \varepsilon \text{ for } 1 \leq i \leq n}} \frac{1}{2} \sum_{i=1}^n \frac{1}{w_i} \|\mathbf{x}_i - \mathbf{X}\mathbf{B}\boldsymbol{\alpha}_i\|_2^2 + w_i. \tag{13}$$

We have introduced here one weight w_i per data point. Typically, $(1/w_i)$ becomes small for outliers, reducing their importance in the objective function. Denoting by $\mathbf{w} = [w_1, \ldots, w_n]$ the weight vector in $\mathbb{R}^n$, the formulation (13) has the following properties: (i) when fixing all variables $\mathbf{A}, \mathbf{B}, \mathbf{w}$ but one vector $\boldsymbol{\alpha}_i$, and optimizing with respect to $\boldsymbol{\alpha}_i$ we still obtain a quadratic program with simplicial constraints; (ii) the same is true for the vectors $\boldsymbol{\beta}_j$; (iii) when fixing $\mathbf{A}$ and $\mathbf{B}$, optimizing with respect to $\mathbf{w}$ has a closed form solution. It is thus natural to use the block-coordinate descent scheme, which is presented in Algorithm 3, and which is guaranteed to converge to a stationary point.

The differences between Algorithms 1 and 3 are the following: each time a vector $\boldsymbol{\alpha}_i$ is updated, the corresponding weight w_i is updated as well, where w_i is the solution of Eq. (12) with $u = \|\mathbf{x}_i - \mathbf{Z}\boldsymbol{\alpha}\|_2$. The solution is actually $w_i = \max(\|\mathbf{x}_i - \mathbf{Z}\boldsymbol{\alpha}\|_2, \varepsilon)$. Then, the update of the vectors $\boldsymbol{\beta}_j$ is slightly more involved. Updating $\boldsymbol{\beta}_j$ yields the following optimization problem:

$$\min_{\boldsymbol{\beta}_j \in \Delta_n} \left\| \left(\mathbf{X} - \mathbf{X}\mathbf{B}_{\text{old}}\mathbf{A} + \mathbf{X}(\boldsymbol{\beta}_{j,\text{old}} - \boldsymbol{\beta}_j)\boldsymbol{\alpha}^j \right) \boldsymbol{\Gamma}^{1/2} \right\|_{\text{F}}^2, \tag{14}$$

Algorithm 3 Robust Archetypal Analysis

1: **Input:** Data $\mathbf{X}$ in $\mathbb{R}^{m \times n}$; p (number of archetypes); T (number of iterations);
2: Initialize $\mathbf{Z}$ in $\mathbb{R}^{m \times p}$ with random columns from $\mathbf{X}$;
3: Initialize $\mathbf{B}$ such that $\mathbf{Z} = \mathbf{X}\mathbf{B}$;
4: Initialize $\mathbf{w}$ in $\mathbb{R}^n$ with $w_i = 1$ for all $1 \leq i \leq n$;
5: **for** $t = 1 \ldots, T$ **do**
6: **for** $i = 1 \ldots, n$ **do**
7: $\boldsymbol{\alpha}_i \in \arg\min_{\boldsymbol{\alpha} \in \Delta_p} \|\mathbf{x}_i - \mathbf{Z}\boldsymbol{\alpha}\|_2^2$;
8: $w_i \leftarrow \max(\|\mathbf{x}_i - \mathbf{Z}\boldsymbol{\alpha}\|_2, \varepsilon)$;
9: **end for**
10: $\boldsymbol{\Gamma} \leftarrow \text{diag}(\mathbf{w})^{-1}$ (scaling matrix);
11: $\mathbf{R} \leftarrow \mathbf{X} - \mathbf{Z}\mathbf{A}$;
12: **for** $j = 1 \ldots, p$ **do**
13: $\boldsymbol{\beta}_j \in \arg\min_{\boldsymbol{\beta} \in \Delta_n} \left\| \frac{\mathbf{R}\boldsymbol{\Gamma}\boldsymbol{\alpha}^{j\top}}{\boldsymbol{\alpha}^j \boldsymbol{\Gamma} \boldsymbol{\alpha}^{j\top}} + \mathbf{z}_j - \mathbf{X}\boldsymbol{\beta} \right\|_2^2$;
14: $\mathbf{R} \leftarrow \mathbf{R} + (\mathbf{z}_j - \mathbf{X}\boldsymbol{\beta}_j)\boldsymbol{\alpha}^j$;
15: $\mathbf{z}_j \leftarrow \mathbf{X}\boldsymbol{\beta}_j$;
16: **end for**
17: **end for**
18: **Return** $\mathbf{A}$, $\mathbf{B}$ (decomposition matrices).

where the diagonal matrix $\boldsymbol{\Gamma}$ in $\mathbb{R}^{n \times n}$ carries the inverse of the weights $\mathbf{w}$ on its diagonal, thus rescaling the residual of each data point $\mathbf{x}_i$ by $1/w_i$ as in (13). Then, it is possible to show that (14) is equivalent to

$$\min_{\boldsymbol{\beta}_j \in \Delta_n} \left\| \frac{1}{\boldsymbol{\alpha}^{jT} \boldsymbol{\Gamma} \boldsymbol{\alpha}^j} \left(\mathbf{X} - \mathbf{X}\mathbf{B}_{\text{old}}\mathbf{A} \right) \boldsymbol{\Gamma} \boldsymbol{\alpha}^{j\top} + \mathbf{X}\boldsymbol{\beta}_{j,\text{old}} - \mathbf{X}\boldsymbol{\beta}_j \right\|_2^2,$$

which is carried out on Line 13 of Algorithm 3.

4. Experiments

We now study the efficiency of archetypal analysis in various applications. Our implementation is coded in C++ and interfaced with R, Python, and Matlab. It has been included in the toolbox SPAMS v2.5 [15]. The number of iterations for archetypal analysis was set to $T = 100$, which leads to a good performance in all experiments.

4.1. Comparison with Other Implementations

To the best of our knowledge, two software packages implementing archetypal analysis are publicly available:

- The Python Matrix Factorization toolbox (PyMF)[1] is an open-source library that tackles several matrix factorization problems including archetypal analysis. It performs an alternate minimization scheme between the $\boldsymbol{\alpha}_i$'s and $\boldsymbol{\beta}_j$'s, but relies on a generic QP solver from CVX.[2]
- The R package archetypes[3] is the reference implemen-

[1] http://code.google.com/p/pymf/.
[2] http://cvxr.com/cvx/.
[3] http://archetypes.r-forge.r-project.org/.

tation of archetypal analysis for R, which is one of the most widely used high-level programming language in statistics. Note that the algorithm implemented in this package deviates from the original archetypal analysis described in [5].

We originally intended to try all methods on matrices $\mathbf{X}$ in $\mathbb{R}^{m \times n}$ with different sizes m and n and different numbers p of archetypes. Unfortunately, the above software packages suffer from severe limitations and we were only able to try them on small datasets. We report such a comparison in Figure 1, where the computational times are measured on a single core of an Intel Xeon CPU E5-1620. We only report results for the R package on the smallest dataset since it diverged on larger ones, while PyMF was several orders of magnitudes slower than our implementation. We also conducted an experiment following the optimization scheme of PyMF but replacing the QP solver by other alternatives such as Mosek or quadprog, and obtained similar conclusions.

Then, we study the scalability of our implementation in regimes where the R package and PyMF are unusable. We report in Figure 2 the computational cost per iteration of our method when varying n or p on the MNIST dataset [13], where $m = 784$. We observe that the empirical complexity is approximately linear in n, allowing us to potentially learn large datasets with more than $100\,000$ samples, and above linear in p, which is thus the main limitation of our approach. However, such a limitation is also shared by classical sparse coding techniques, where the empirical complexity regarding p is also greater than $O(p)$ [1].

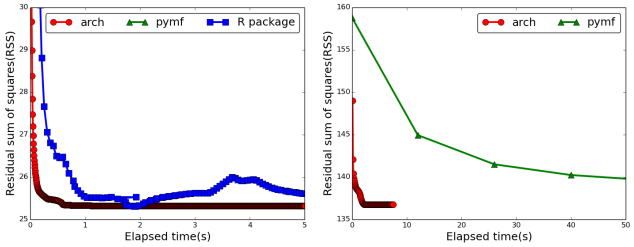


Figure 1: Experimental comparison with other implementations. Left: value of the objective function vs computational time for a dataset with $m = 10$, $n = 507$ and $p = 5$ archetypes. Our method is denoted by arch. PyMF was too slow to appear in the graph and the R-package exhibits a non-converging behavior. Right: same experiment with $n = 600$ images from MNIST [13], of size $m = 784$, with $p = 10$ archetypes. The R package diverged while PyMF was between 100 and 1000 times slower than our approach.

4.2. Codebook Learning with Archetypal Analysis

Many computer vision approaches have represented so far images under the form of a "bag of visual words", or by using some variants of it. In a nutshell, each local patch

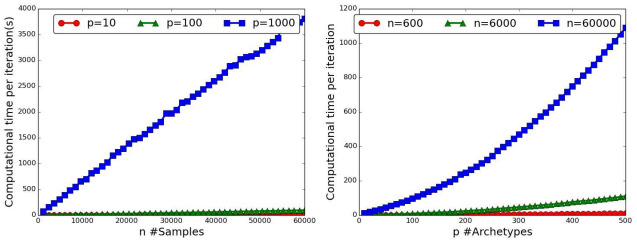


Figure 2: Scalability Study. Left: the computational time per iteration when varying the sample size n for different numbers of archetypes p. The complexity of our implementation is empirically linear in n. Right: the same experiment when varying p and with fixed sample sizes n. The complexity is more than linear in p.

(small regions of typically 16×16 pixels) of an image is encoded by a descriptor which is invariant to small deformations, such as SIFT [12]. Then, an unsupervised learning technique is used for defining a codebook of visual patterns called "visual words". The image is finally described by computing a histogram of word occurrences, yielding a powerful representation for discriminative tasks [4, 12].

More precisely, typical methods for learning the codebook are K-means and sparse coding [12, 24]. SIFT descriptors are sparsely encoded in [24] by using the formulation (4), and the image representation is obtained by "max-pooling" the sparse codes, as explained in [24]. Spatial pyramid matching (SPM) [12] is also used, which includes some spatial information, yielding better accuracy than simple bags of words on many benchmark datasets. Ultimately, the classification task is performed with a support vector machine (SVM) with a linear kernel.

It is thus natural to wonder whether a similar performance could be achieved by using archetypal analysis instead of sparse coding. We thus conducted an image classification experiment by using the software package of [24], and simply replacing the sparse coding component with our implementation of archetypal analysis. We use as many archetypes as dictionary elements in [24]—that is, $p = 1\,024$, and $n = 200\,000$ training samples, and we call the resulting method "archetypal-SPM". We use the same datasets as [24]—that is, Caltech-101 [9] and 15 Scenes Categorization [10, 12]. The purpose of this experiment is to demonstrate that archetypal analysis is able to learn a codebook that is as good as sparse coding and better than K-means. Thus, only results of similar methods are represented here such as [12, 24]. The state of the art on these data sets may be slightly better nowadays, but involves a different recognition pipeline. We report the results in Tables 1 and 2, where archetypal analysis seems to perform as well as sparse coding. Note that KMeans-SPM-χ^2 uses a χ^2-kernel for the SVM [12].

Algorithms	15 training	30 training
KMeans-SPM-χ^2 [12]	56.44 ± 0.78	63.99 ± 0.88
KMeans-SPM [24]	53.23 ± 0.65	58.81 ± 1.51
SC-SPM [24]	67.00 ± 0.45	73.20 ± 0.54
archetypal-SPM	64.96 ± 1.04	72.00 ± 0.88

Table 1: Classification accuracy (%) on Caltech-101 dataset. Following the same experiment in [24], 15 or 30 images per class are randomly chosen for training and the rest for testing. The standard deviation is obtained with 10 randomized experiments.

Algorithms	Classification Accuracy
KMeans-SPM-χ^2 [12]	81.40 ± 0.50
KMeans-SPM [24]	65.32 ± 1.02
SC-SPM [24]	80.28 ± 0.93
archetypal-SPM	81.57 ± 0.81

Table 2: Classification accuracy (%) on Scene-15 dataset. Following the same experiment in [24], 100 images are randomly chosen for training and the rest for testing. The standard deviation is obtained with 10 randomized experiments.

4.3. Digit Classification with Archetypal Analysis

Even though sparse coding is an unsupervised learning technique, it has been used directly for classification tasks [14, 21, 23]. For digit recognition, it has been observed in [21] that simple classification rules based on sparse coding yield impressive results on classical datasets such as MNIST [13] and USPS. Suppose that one has learned on training data a dictionary $\mathbf{Z}_k$ in $\mathbb{R}^{m \times p}$ for every digit class $k = 0, \ldots, 9$ by using (4), and that a new test digit $\mathbf{x}$ in $\mathbb{R}^m$ is observed. Then, $\mathbf{x}$ can be classified by finding the class $k^\star$ that best represents it with $\mathbf{Z}_k$:

$$k^\star = \underset{k \in \{0,\ldots,9\}}{\arg\min} \left[\min_{\boldsymbol{\alpha} \in \mathbb{R}^p} \frac{1}{2} \|\mathbf{x} - \mathbf{Z}_k \boldsymbol{\alpha}\|_2^2 + \lambda \|\boldsymbol{\alpha}\|_1 \right], \quad (15)$$

where λ is set to 0.1 in [21] and the vectors $\mathbf{x}$ are normalized. Since we want to compare archetypal analysis with sparse coding, it is thus natural to also consider the corresponding "archetype" classification rule:

$$k^\star = \underset{k \in \{0,\ldots,9\}}{\arg\min} \left[\min_{\boldsymbol{\alpha} \in \Delta_p} \|\mathbf{x} - \mathbf{Z}_k \boldsymbol{\alpha}\|_2^2 \right], \quad (16)$$

where the $\mathbf{Z}_k$ are archetypes learned for every digit class. Note that when archetypes are made of all available training data, the convex dual of (16) is equivalent to the nearest convex hull classifier of [16]. We report the results when all the training data is used as archetypes in Table 3, and when varying the number of archetypes per class in Figure 3. We include in this comparison the performance of SVM with a Gaussian kernel, and the K-nearest neighbor classifier (K-NN). Even though the state of the art on MNIST achieves less than 1% test error [14, 22], the results reported in Table 3 are remarkable for several reasons: (i) the method AA-All has no hyper-parameter and performs almost as well as sparse coding, which require choosing λ; (ii) AA-All and SC-All significantly outperform K-NN and perform similarly as a non-linear SVM, even though they use a simple Euclidean norm for comparing two digits; (iii) none of the methods in Table 3 exploit the fact that the $\mathbf{x}_i$'s are in fact images, unlike more sophisticated techniques such as convolutional neural networks [22]. In Figure 3, neither SC nor AA are helpful for prediction, but archetypal analysis can be useful for reducing the computational cost at test time. The choice of dictionary size K should be driven by this trade-off. For example, on USPS, using 200 archetypes per class yields similar results as AA-All.

Dataset	AA-All	SC-All	SVM	K-NN
MNIST	1.51	1.35	1.4	3.9
USPS	4.33	4.14	4.2	4.93

Table 3: Classification error rates (%) on the test set for the MNIST and USPS datasets. AA-All and SC-All respectively mean that all data points are used as archetypes and dictionary elements. SVM uses a Gaussian kernel.

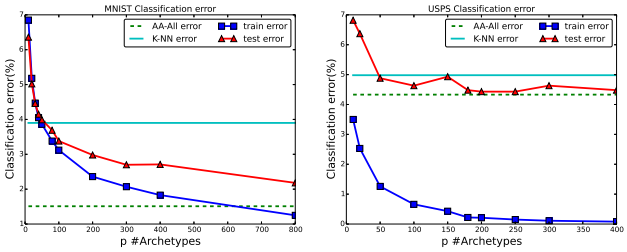


Figure 3: Train and Test error on MNIST (left) and USPS (right) when varying the number p of archetypes. All-AA means that all data points are used as archetypes.

4.4. Archetyping Flickr Requests

Data visualization has now become an important topic, especially regarding image databases from the Internet [6], or videos [8]. We focus in this section on public images downloaded from the Flickr website, and present a methodology for visualizing the content of different requests using *robust archetypal analysis* presented in Section 2.1.

For example, we present in this section a way to visualize the request "Paris" when downloading 36 600 images uploaded in 2012 and 2013, and sorted by "relevance" according to Flickr. We first compute dense SIFT descriptors [12] for all images, and represent each image by using

a Fisher vector [20], which have shown good discriminative power in classification tasks. Then, we learn $p = 256$ archetypes. Interestingly, we observe that the Flickr request has a large number of outliers, meaning that some images tagged as "Paris" are actually unrelated to the city. Thus, we choose to use the robust version of archetypal analysis in order to reduce the influence of such outliers. We use similar heuristics for choosing ϵ as in the robust statistics literature, resulting in $\epsilon = 0.01$ for data points that are ℓ_2-normalized.

Even though archetypes are learned in the space of Fisher vectors, which are not displayable, each archetype can be interpreted as a sparse convex combination of data points. In Figure 4 we represent some of the archetypes learned by our approach; each one is represented by a few training images with some proportions indicated in red (the value of the $\boldsymbol{\beta}_j$'s). Classical landmarks appear in Figure 4a, which is not surprising since Flickr contains a large number of vacation pictures. In Figure 4b, we display several archetypes that we did not expect, including ones about soccer, graffitis, food, flowers, and social gatherings. In Figure 4c, some archetypes do not seem to have some semantic meaning, but they capture some scene composition or texture that are common in the dataset. We present the rest of the archetypes in the supplementary material, and results obtained for other requests, such as London or Berlin.

In Figure 5, we exploit the symmetrical relation between data and archetypes. We show for four images how they decompose onto archetypes, indicating the values of the $\boldsymbol{\alpha}_i$'s. Some decompositions are trivial (Figure 5a); some others with high mean squared error are badly represented by the archetypes (Figure 5c); some others exhibit interesting relations between some "texture" and "architecture" archetypes (Figure 5b).

5. Discussions

In this paper, we present an efficient active-set strategy for archetypal analysis. By providing the first scalable open-source implementation of this powerful unsupervised learning technique, we hope that our work will be useful for applying archetypal analysis to various scientific problems. In particular, we have shown that it has promising applications in computer vision, where it performs as well as sparse coding for prediction tasks, and provides an intuitive visualization technique for large databases of natural images. We also propose a robust version, which is useful for processing datasets containing noise, or outliers, or both.

Acknowledgements

This work was supported by the INRIA-UC Berkeley Associated Team "Hyperion", a grant from the France-Berkeley fund, the project "Gargantua" funded by the program Mastodons-CNRS, and the Microsoft Research-Inria joint centre.

References

[1] F. Bach, R. Jenatton, J. Mairal, and G. Obozinski. Optimization with sparsity-inducing penalties. *Found. Trends Mach. Learn.*, 4:1–106, 2012. 2, 3, 5

[2] D. P. Bertsekas. *Nonlinear programming*. Athena Scientific, 1999. 2, 3

[3] D. Blei, A. Ng, and M. Jordan. Latent dirichlet allocation. *"J. Mach. Learn. Res."*, 3:993–1022, 2003. 1

[4] G. Csurka, C. Dance, L. Fan, J. Willamowski, and C. Bray. Visual categorization with bags of keypoints. In *Workshop on statistical learning in computer vision, ECCV*, 2004. 1, 5

[5] A. Cutler and L. Breiman. Archetypal analysis. *Technometrics*, 36(4):338347, 1994. 1, 2, 4, 5

[6] C. Doersch, S. Singh, A. Gupta, J. Sivic, and A. Efros. What makes paris look like paris? *SIGGRAPH*, 31(4), 2012. 6

[7] M. Eisen, P. Spellman, P. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *P. Natl. Acad. Sci. USA*, 95(25):14863–14868, 1998. 1

[8] E. Elhamifar, G. Sapiro, and R. Vidal. See all by looking at a few: Sparse modeling for finding representative objects. In *CVPR*, 2012. 6

[9] L. Fei-Fei, R. Fergus, and P. Perona. Learning generative visual models from few training examples: An incremental Bayesian approach tested on 101 object categories. *Comput. Vis. Image Und.*, 106(1):59–70, 2007. 5

[10] L. Fei-Fei and P. Perona. A Bayesian hierarchical model for learning natural scene categories. In *CVPR*, 2005. 5

[11] K. Lange, D. R. Hunter, and I. Yang. Optimization transfer using surrogate objective functions. *J. Comput. Graph. Stat.*, 9(1):1–20, 2000. 2, 4

[12] S. Lazebnik, C. Schmid, and J. Ponce. Beyond bags of features: Spatial pyramid matching for recognizing natural scene categories. In *CVPR*, 2006. 1, 5, 6

[13] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition. *Proc. IEEE*, 86(11):2278–2324, 1998. 5, 6

[14] J. Mairal, F. Bach, and J. Ponce. Task-driven dictionary learning. *PAMI*, 34(4):791–804, 2012. 1, 6

[15] J. Mairal, F. Bach, J. Ponce, and G. Sapiro. Online learning for matrix factorization and sparse coding. *J. Mach. Learn. Res.*, 11:19–60, 2010. 4

[16] G. I. Nalbantov, P. J. Groenen, and J. C. Bioch. Nearest convex hull classification. Technical report, Erasmus School of Economics (ESE), 2006. 6

[17] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, 2006. 1, 3, 4

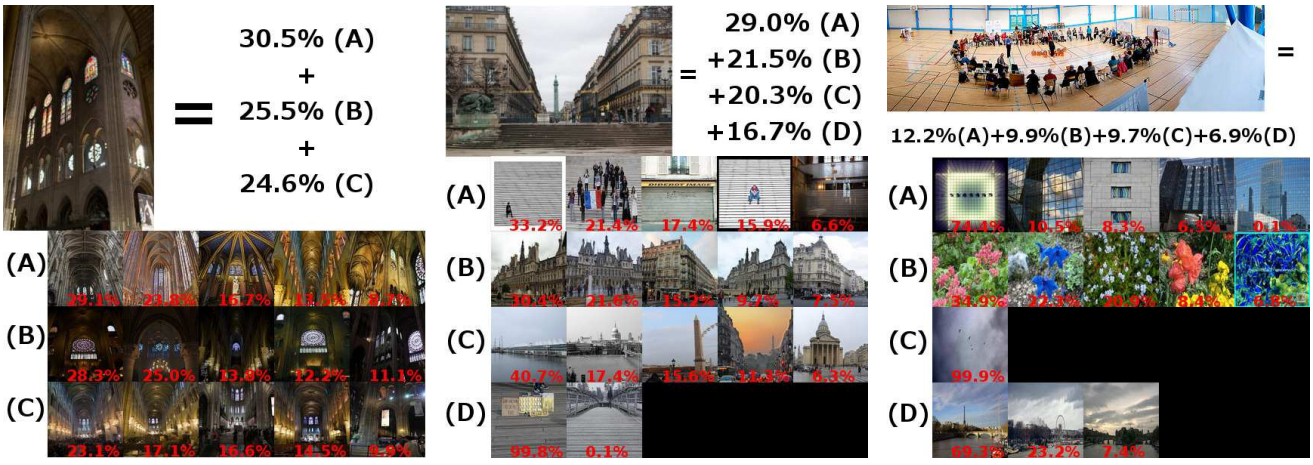
[18] B. A. Olshausen and D. J. Field. Emergence of simple-cell receptive field properties by learning a sparse code for natural images. *Nature*, 381(6583):607–609, 1996. 1, 2

[19] P. Paatero and U. Tapper. Positive matrix factorization: A non-negative factor model with optimal utilization of error estimates of data values. *Environmetrics*, 5(2):111–126, 1994. 1, 2

[20] F. Perronnin, J. Sánchez, and T. Mensink. Improving the Fisher kernel for large-scale image classification. In *ECCV*. 2010. 7

(a) Archetypes representing (expected) landmarks. (b) Less expected archetypes. (c) Archetypes representing scene composition.

Figure 4: Some archetypes learned from 36 600 pictures corresponding to the request "Paris" downloaded from Flickr.



(a) Image with trivial interpretation. (b) Image with interesting interpretation. (c) Image with bad (non-sparse) reconstruction.

Figure 5: Decomposition of some images onto the learned archetypal set.

[21] I. Ramirez, P. Sprechmann, and G. Sapiro. Classification and clustering via dictionary learning with structured incoherence and shared features. In *CVPR*, 2010. 1, 6

[22] M. Ranzato, F. J. Huang, Y.-L. Boureau, and Y. Lecun. Unsupervised learning of invariant feature hierarchies with applications to object recognition. In *CVPR*, 2007. 6

[23] J. Wright, A. Y. Yang, A. Ganesh, S. S. Sastry, and Y. Ma. Robust face recognition via sparse representation. *PAMI*, 31(2):210–227, 2009. 1, 6

[24] J. Yang, K. Yu, Y. Gong, and T. Huang. Linear spatial pyramid matching using sparse coding for image classification. In *CVPR*, 2009. 1, 5, 6

[25] G. Yu, G. Sapiro, and S. Mallat. Solving inverse problems with piecewise linear estimators: from Gaussian mixture models to structured sparsity. *IEEE T. Image Process.*, 21(5):2481–2499, 2012. 1

[26] K. Yu, T. Zhang, and Y. Gong. Nonlinear learning using local coordinate coding. In *NIPS*, 2009. 2