PPGEE2249 Aprendizado de Máquina
Assignment 2
Prof. Daniel Guerreiro e Silva

Thiago Tomás de Paula
October 30, 2025

► Question 1

Consider the three-dimensional dataset in `data_pca.csv`. Apply PCA to study the data.

▷ Item a

What is the sample mean of the data?
Then, create a new dataset with null sample mean and unit variance.

▷ Item b

Calculate the sample covariance matrix of the **new** dataset.
Calculate its eigenvalues and eigenvectors.

▷ Item c

Based on the results of (a) and (b), analyze if it is possible to reduce the dataset to (i) one or (ii) two dimensions. Use numerical measures to justify your analysis.

▷ Item d

Plot, in the same 3D graph:

i) the original data,
ii) the **reconstructed** data from the 1D projection and
iii) the **reconstructed** data from the 2D projection.

Comment the results.

▷ Item e

Based on the previous results, is PCA a useful tool for this dataset?

We consider a dataset that is an $N \times d$ matrix of samples $\mathbf{X}$ such that each line represents a sample of the attributes, separating them column by column. Principal Component Analysis (PCA) aims to reduce the dimentionality of the sample matrix by projecting the samples onto the orthonormal eigenvectors $\mathbf{v}_i$, $1 \leq i \leq d$, of the dataset's correlation matrix $\mathbf{W}$[1].

More precisely, it is the dataset centered at $\mathbf{0}$ that is projected onto the eigenvectors, generating a set of vectors $\mathbf{Z}$. A reconstruction $\widehat{\mathbf{X}}$ of the original dataset can then be obtained by inverting the projections and summing the samples averages to these projections. Importantly, a "full" projection of $\mathbf{X}$ onto $\mathbf{W}$, would be associated with vectors that have an equal number of dimentions to those in the original dataset, and moreover $\widehat{\mathbf{X}} = \mathbf{X}$.

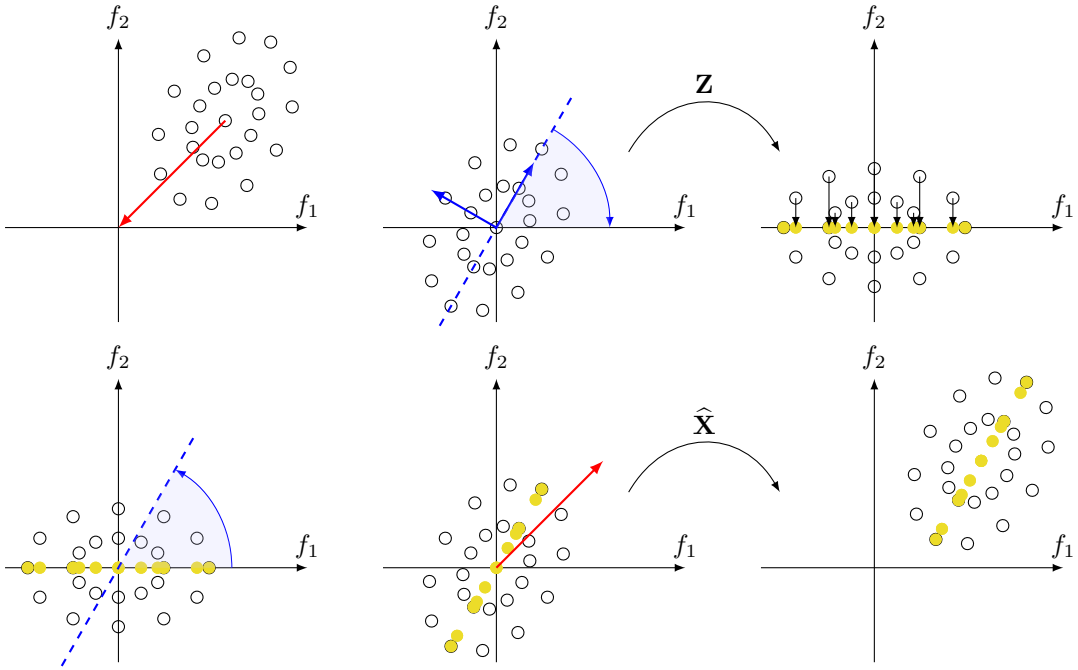
Therefore, for PCA to be effective, the samples are projected only onto the orthonormal eigenvectors associated with largest eigenvalues α_i, $1 \leq i \leq d$, in absolute size. In this text, eigenvalues are indexed such that $|\alpha_1| > |\alpha_2| > \cdots > |\alpha_d|$.

The correlation matrix of $\mathbf{X}$ is given by $\mathbf{W} = \mathbf{Y}^{\mathrm{T}}\mathbf{Y}/(N-1)$, where $Y_{ij} = (X_{ij} - \overline{X}_j)/S_j$, with $\overline{X}_i$ denoting the mean of the j-th attribute, and S_i, the root of its variance[2].

Note that by defining the $n \times d$ matrices $\overline{X}_{ij} = \overline{X}_j$ and $S_{ij} = S_j$, $\mathbf{Y}$ can be handily given as two pointwise operations, $\mathbf{Y} = (\mathbf{X} - \overline{\mathbf{X}})/\mathbf{S}$.

[1] It can be shown that this procedure maximizes sample dispersion, justifying the method.
[2] Since $\mathbf{Y}$ is zero-mean, $\mathbf{W}$ also represents its *covariance* matrix.

Figure 2: Illustration of PCA algorithm on a 2D dataset. Top row illustrates PCA projection, bottom row illustrates the reconstruction from those projections. The red vectors are tied to the samples' average, while blue vectors indicate the covariance matrix' orthonormal eigenvectors. Yellow dots are the projections of samples onto the most expressive eigenvector (arrows for some samples omitted for readability).



Now, for the dataset in question, the following values were observed[3]:

$$X_1 = 2.523, \quad X_2 = -2.813, \quad X_3 = 0.304 \tag{1}$$
$$S_1 = 1.408, \quad S_2 = 5.379, \quad S_3 = 9.365 \tag{2}$$

Which lead to the following correlation matrix:

$$\mathbf{W} = \begin{bmatrix} 1.000 & -0.534 & -0.607 \\ -0.534 & 1.000 & 0.941 \\ -0.607 & 0.941 & 1.000 \end{bmatrix} \tag{3}$$

The orthonormal eigenvectors associated with $\mathbf{W}$ and associated eigenvalues are presented below.

$$\mathbf{v}_1 = \begin{bmatrix} 0.498 \\ -0.605 \\ -0.621 \end{bmatrix}, \quad \mathbf{v}_2 = \begin{bmatrix} -0.863 \\ -0.416 \\ -0.287 \end{bmatrix}, \quad \mathbf{v}_3 = \begin{bmatrix} 0.085 \\ -0.678 \\ 0.730 \end{bmatrix}, \tag{4}$$

$$\alpha_1 = 2.406, \qquad \alpha_2 = 0.542, \qquad \alpha_3 = 0.056 \tag{5}$$
$$(80.116\ \%) \qquad (18.034\ \%) \qquad (1.859\ \%) \tag{6}$$

The values in parentheses represent the weight of the eigenvector in reconstructions of data from PCA, i.e., the value of the associated eigenvalue divided by the sum of all eigenvalues. In turn, the 1D,

[3] See the `.csv` files in folder `results/pca/` inside the code zip.

2D, and 3D reconstructions of $\mathbf{X}$ using the calculated eigenvectors are given by the following equations:

$$\widehat{\mathbf{X}}_{1D} = (\mathbf{y}_1 \mathbf{v}_1^T) \circ \mathbf{S} + \overline{\mathbf{X}} \tag{7}$$

$$\widehat{\mathbf{X}}_{2D} = \mathbf{X}_{1D} + (\mathbf{y}_2 \mathbf{v}_2^T) \circ \mathbf{S} \tag{8}$$

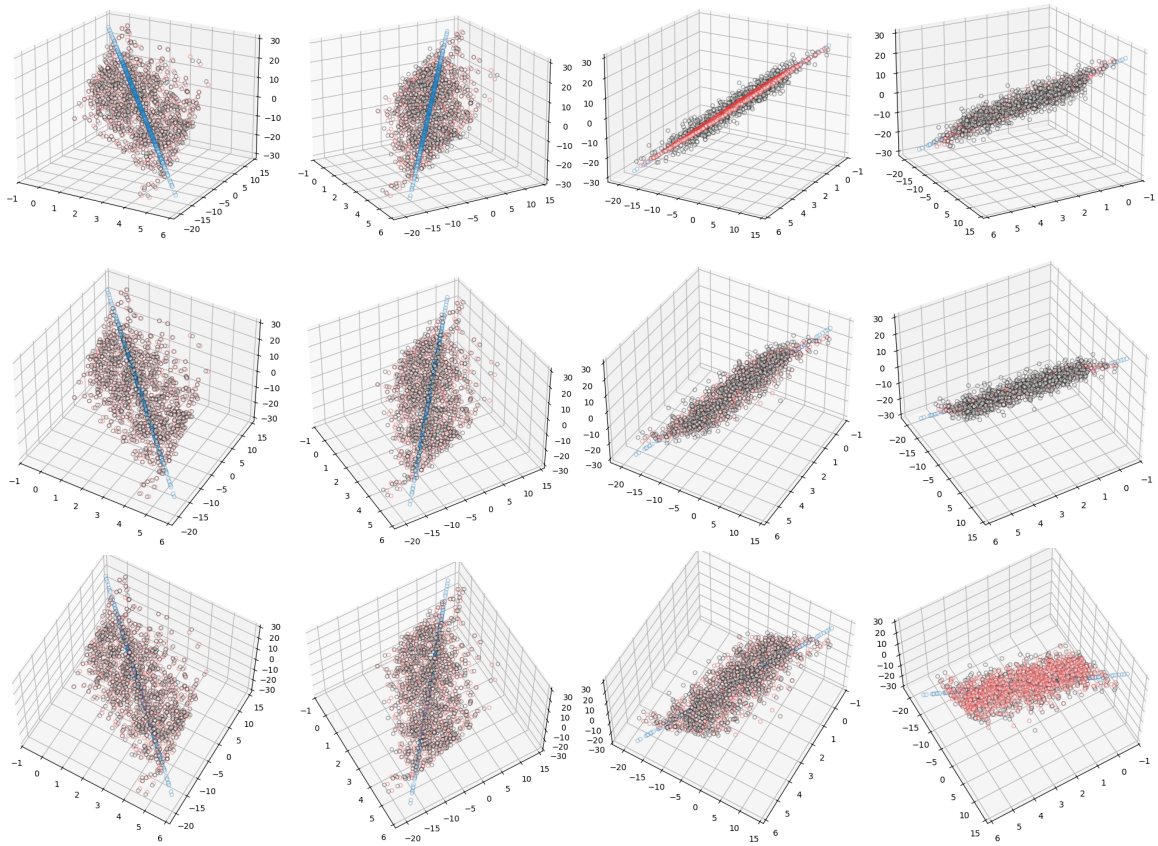
$$\widehat{\mathbf{X}}_{3D} = \mathbf{X}_{2D} + (\mathbf{y}_3 \mathbf{v}_3^T) \circ \mathbf{S} \tag{9}$$

where $\circ$ denotes the Hadamard (i.e., pointwise) product and $\widehat{\mathbf{X}}_{3D}$ is exactly equal to the original data.

These results indicate that $\mathbf{v}_1$ is, by far, the most representative direction of spread of data, although $\mathbf{v}_2$ also hold meaningful weight. Conversely, the smallest vector $\mathbf{v}_3$ contributes very little information into the reconstructions, and should be safely excluded in PCA.

In light of this brief analysis, it is concluded that an effective PCA will use only $\mathbf{v}_1$ and $\mathbf{v}_2$, dropping one dimention from the original dataset. The figure below shows the original data *versus* its reconstructions using (i) only $\mathbf{v}_1$; (ii) both $\mathbf{v}_1$ and $\mathbf{v}_2$. From inspection, it is safe to say that the 2D reconstruction is quite representative of data, while the 1D is not. As such, PCA, in particular its bidimensional version, should prove a useful tool for simplifying this dataset.

Figure 3: Original data (black) versus PCA 1D (blue) and 2D (red) reconstructions. Bottom row shows the same view angles as top row but at higher elevation.



► Question 2

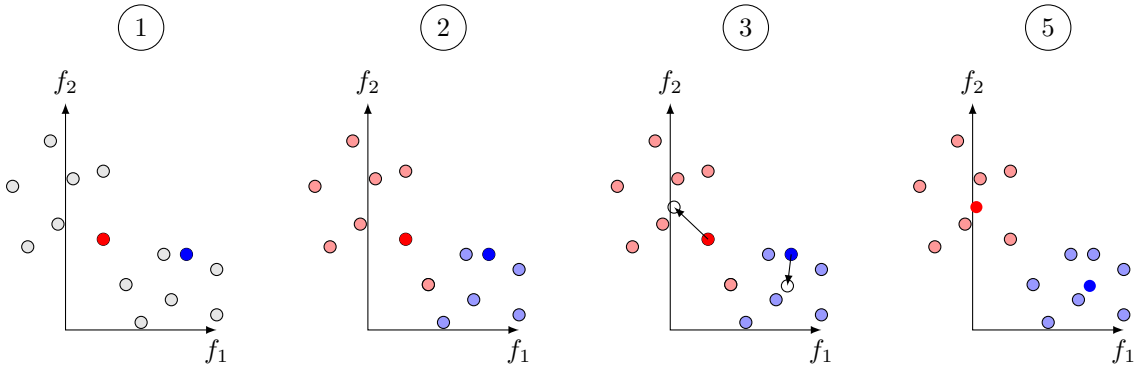
Implement the k-means clustering algorithm. Then, choose a 3D dataset to apply your tool and test it. Explain the steps of your solution and justify your choice of the number of clusters, using graphical or numerical evidence.

The k-means clustering algorithm is an expectation-maximization algorithm for unsupervised classification. Output-wise, it attempts to separate the original dataset in subsets (clusters) that are each associated to unique label, i.e., a vector in sample space. Once the algorithm is done, the samples in each group are assigned the values of the calculated labels; this results in a new dataset that is a rough approximation of the original, but that is built out of by much smaller dataset, the labels themselves. Put simply, k-means attempts to reduce sample variability with as little information loss as possible by taking advantage of underlying correlations across samples.

In further detail, the algorithm takes these steps, also illustrated afterwards:

1. Initialize k labels in sample space
2. Assign each sample in the dataset to the label closest to it
3. Calculate the centroids of the clusters found in the previous step
4. If the centroids are close enough to the labels, finish the algorithm
5. Otherwise, update the labels with the centroids values
6. Go back to step 2

Figure 4: Iteration of the k-means algorithm on a simple 2D dataset. Feature space. Labels are set at random from the given samples and $k = 2$, then updated according to the clusters' averages (centroids). Numbers in the bubbles indicate the relevant step in the numbered list.



I.e., k-means is template matching the data to its underlying distributions. This overview of the algorithm introduces a number of questions that must be addressed for unambiguous implementation.

First and foremost, the number k of clusters/labels must be given a priori, despite heavily affecting the performance of the algorithm. One of the possible ways of tackling this problem is running the algorithm for many values of k, which may be computationally expensive. Ideally, the dataset either makes a clear suggestion of the number of clusters, or the user has knowledge about the samples that help narrowing down the value.

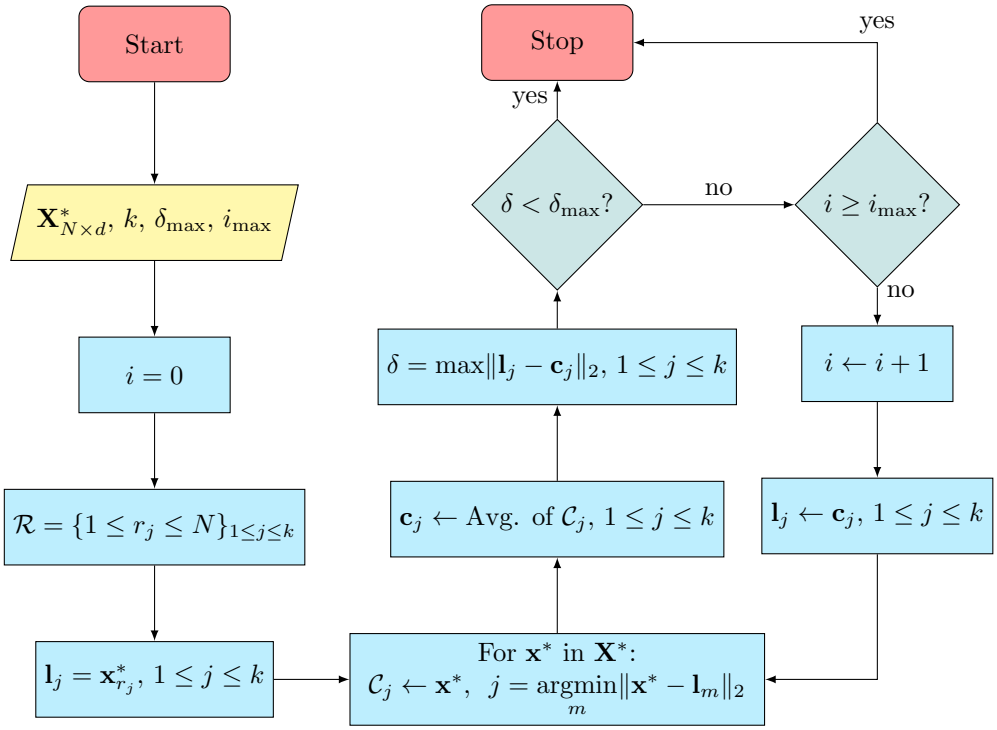
Secondly, initialization of labels is not unique. Although this detail usually impacts reconstruction results much less than the number of clusters, it still plays an important role in the algorithm. If a label is too far from data relative to the other labels, it risks being dropped altogether, i.e., no samples are assigned to it. A simple method that avoids this worst case scenario, and which is used in this implementation, is picking k samples at random from the dataset as initial labels.

Thirdly, the way which distances are calculated may vary. Usually, however, L_2-norm is applied, and it was preferred in this implementation. This was also the metric used to calculate the reconstruction error across iterations.

Lastly, what defines what is "close enough" when deciding wheter or not the algorithm should stop iterating is arbitrary. Generally, the distance metric between the new centroids and the previous labels is compared against a "small" positive value $\delta_{\max}$, and if it is less than this threshold, the algorithm ends. Since what is a "small" threshold depends on the range of the sample space (a margin of 0.01 is much tighter on attributes that range from 0-256 than 0-1), the original dataset was centered at zero and normalized before being fed to the algorithm. Additionally, it's good practice to limit the number of iterations to a set value $i_{\max}$ in case the centroids don't converge or take too long to do so.

With the above observations in mind, a complete flowchart of the implemented k-means algorithm can be drawn, presented below.

Figure 5: Implemented k-means algorithm. $\mathbf{X}^*$ denotes the zero-mean, normalized dataset, and $\mathbf{x}^*$ is a row from it. Values r_j are random integers. Omitted: the returns of the algorithm, namely the clusters $\mathcal{C}_j$, labels $\mathbf{l}_j$, and reconstruction errors.



To test the implementation, a small set of pictures was chosen as datasets. These pictures, depicting either certain animals or flowers, come from a much bigger dataset available openly at https://www.kaggle.com/datasets/pavansanagapati/images-dataset/data.

The sample space corresponds to the red, green and blue values of each pixel in the image, making it a subset of $\mathbb{R}^3$. The number k of clusters was decided by inspection of the figures, i.e., it was estimated to be equal to the number of main colors in the picture. All case studies adopted a threshold $\delta_{\max}$ of 10^{-3}, a maximum number of iterations $i_{\max}$ of 50, and had the same random seed for initialization (the integer 242104677).

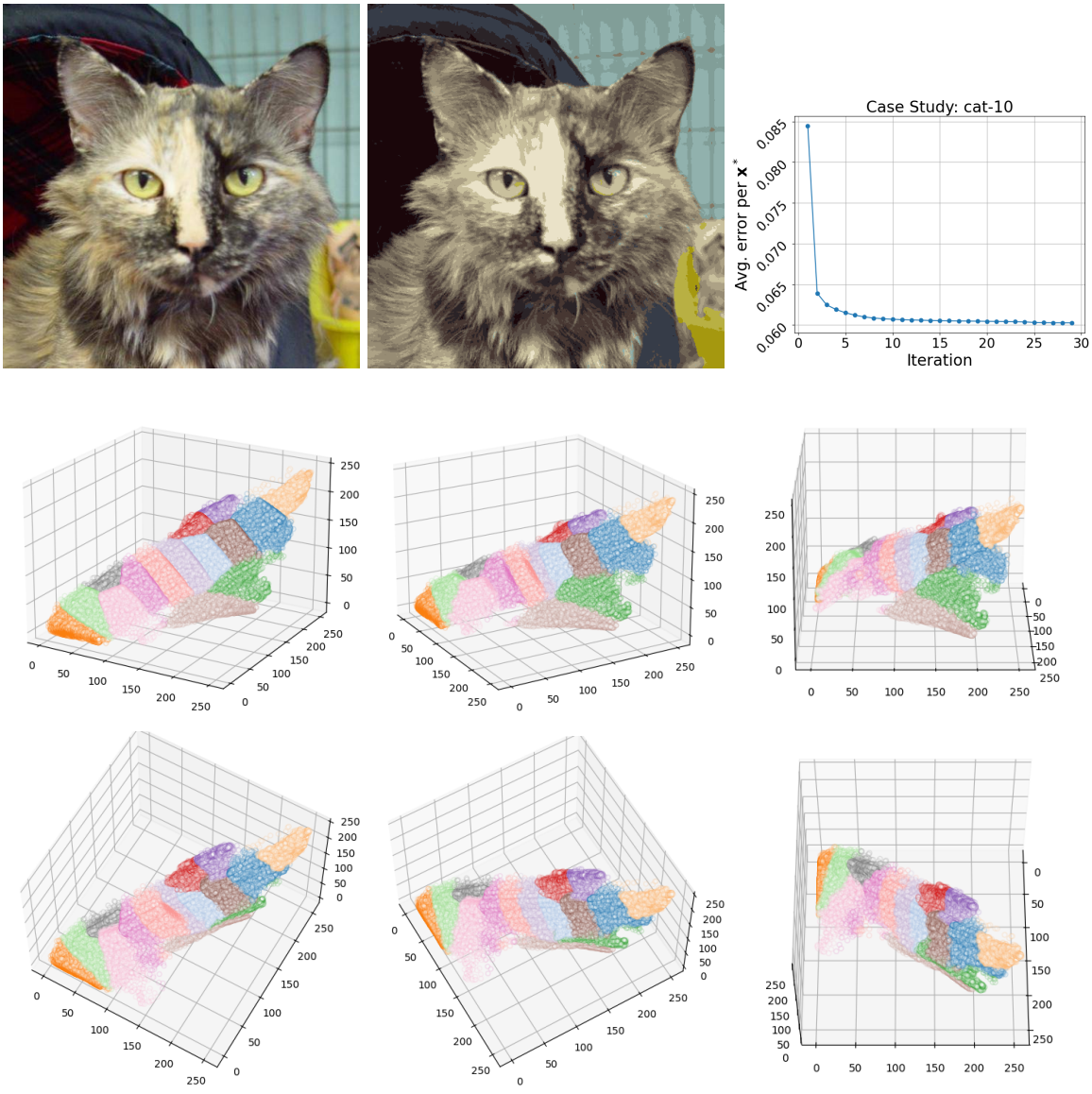
Each picture tested, the number of clusters given to it, the resulting reconstruction, the associated reconstruction error, and what the final clustering looked like in sample space[4], are presented

[4] A clustering always refer to the associated iteration labels *before* the update using the new centroids. E.g., if

below. From inspection, it is clear that the implemented k-means worked as intended, specially for the case studies `cat-10.jpg`, `flower-14.jpg`, `horse-139.jpg`, and `horse-170.jpg`. Interestingly, cases where the algorithm did not converge within the set number of iterations did not result in poor reconstructions, such as `cat-101.jpg` and `cat-110.jpg`.

Conversely, a small reconstruction error did not guarantee a good visual result, as in `horse-137.png` (note the white spots on the horse's hindleg), `flower-6.jpg` (note how yellow tones were desaturated), and `flower-23.jpg` (note how a slice of the sky was highlighted). Overall, however, the reconstructions are visually close to the originals, and the reconstruction errors are small.

Figure 6: Case study: `cat-10.jpg`, $k = 15$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.



convergence is achieved in iteration 40, the results presented are iteration's 39 labels and centroids. This is done to make sure centroids and clusters are synchronized.

Figure 7: Case study: `cat-101.jpg`, $k = 15$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

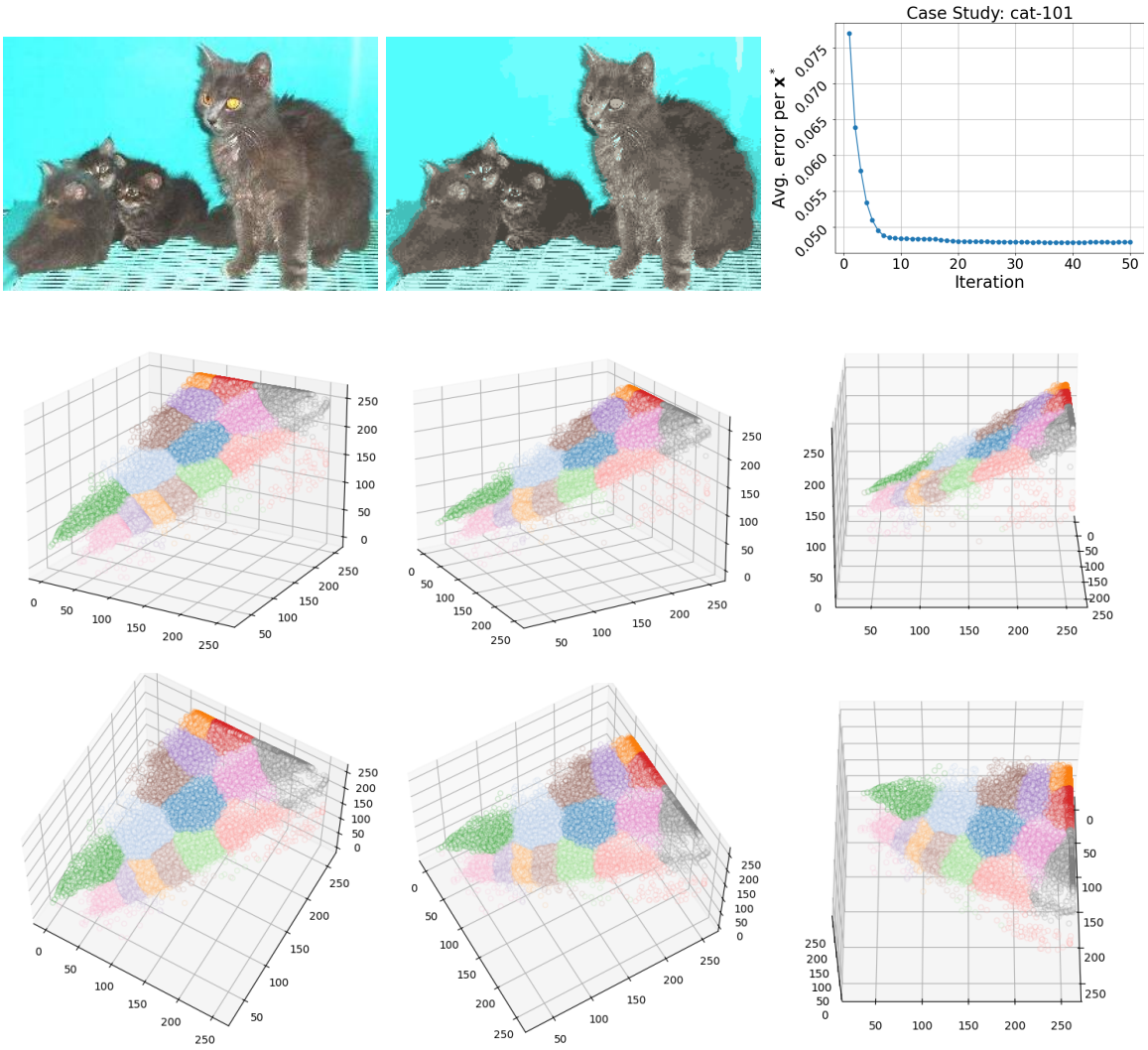


Figure 8: Case study: `cat-110.jpg`, $k = 15$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

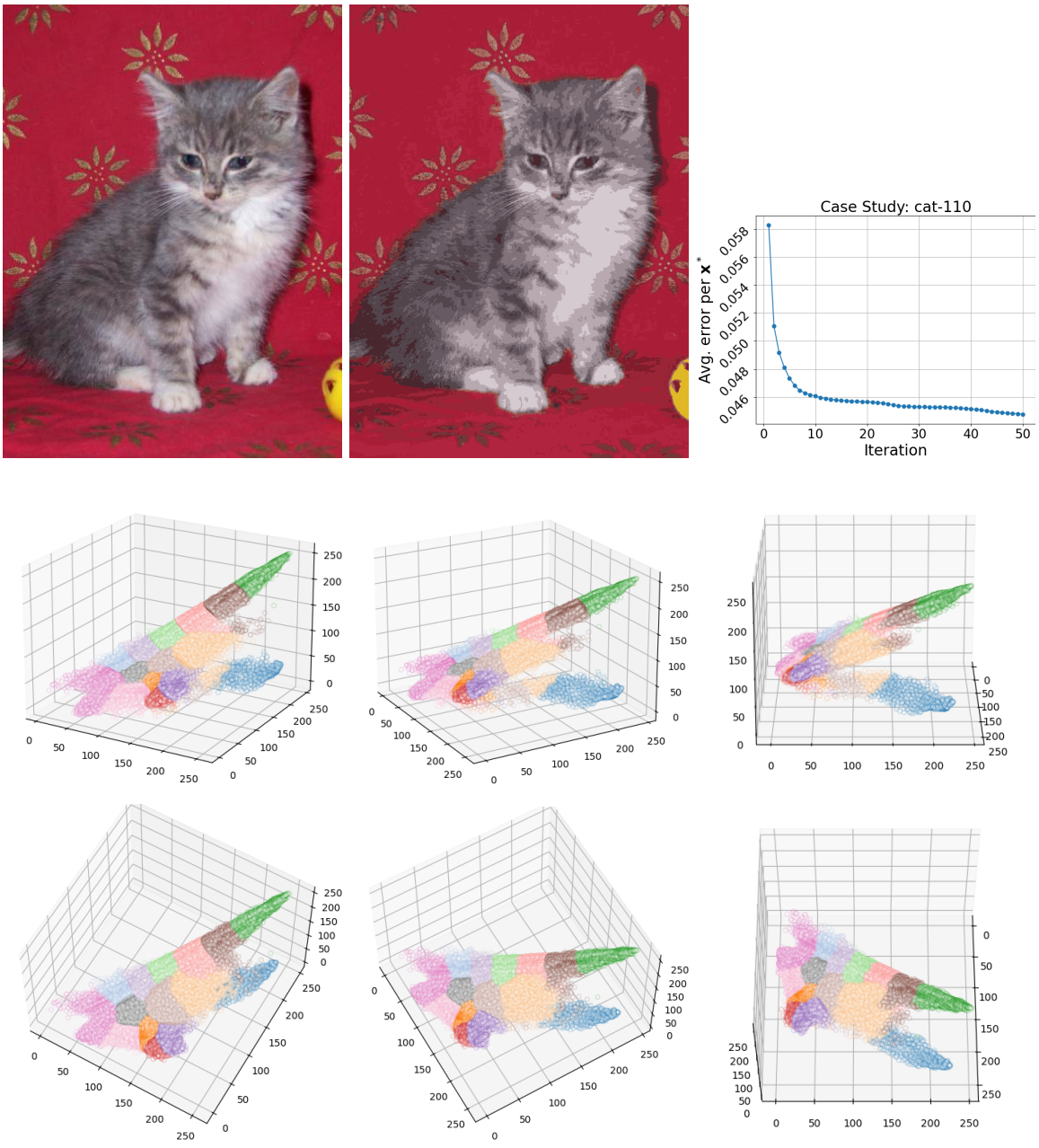


Figure 9: Case study: `flower-6.jpg`, $k = 10$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

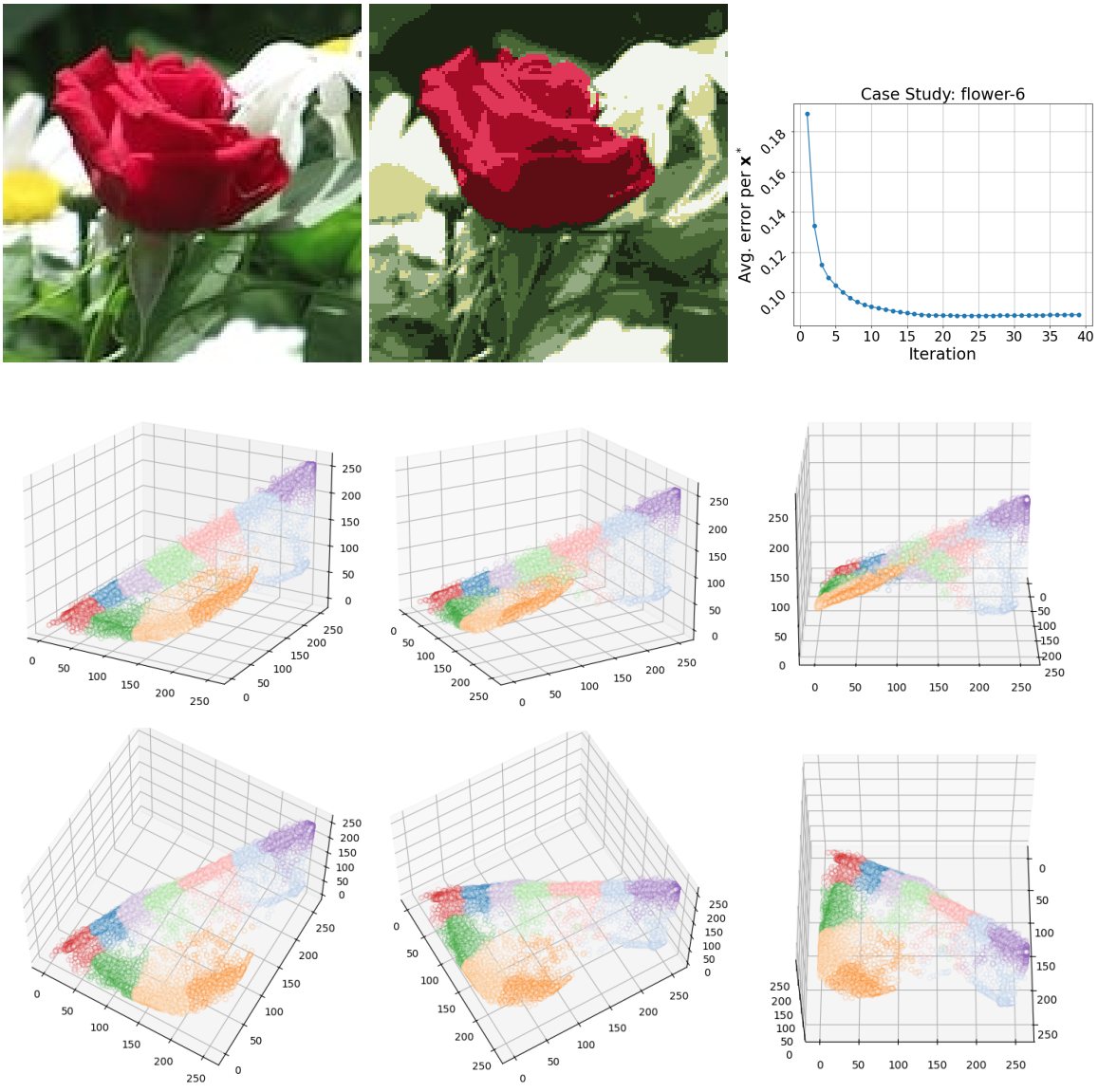


Figure 10: Case study: `flower-14.jpg`, $k = 10$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

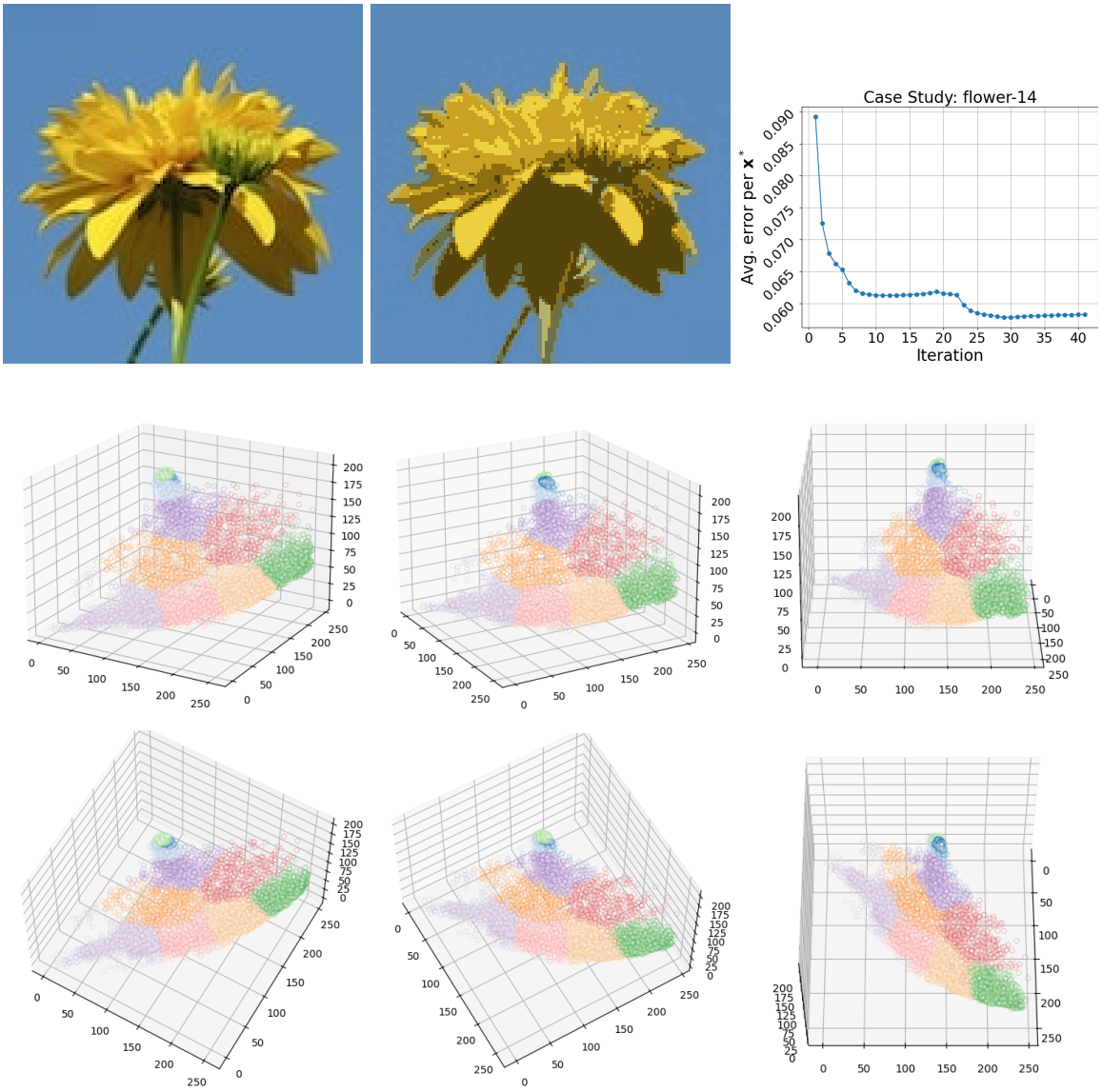


Figure 11: Case study: `flower-23.jpg`, $k = 10$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

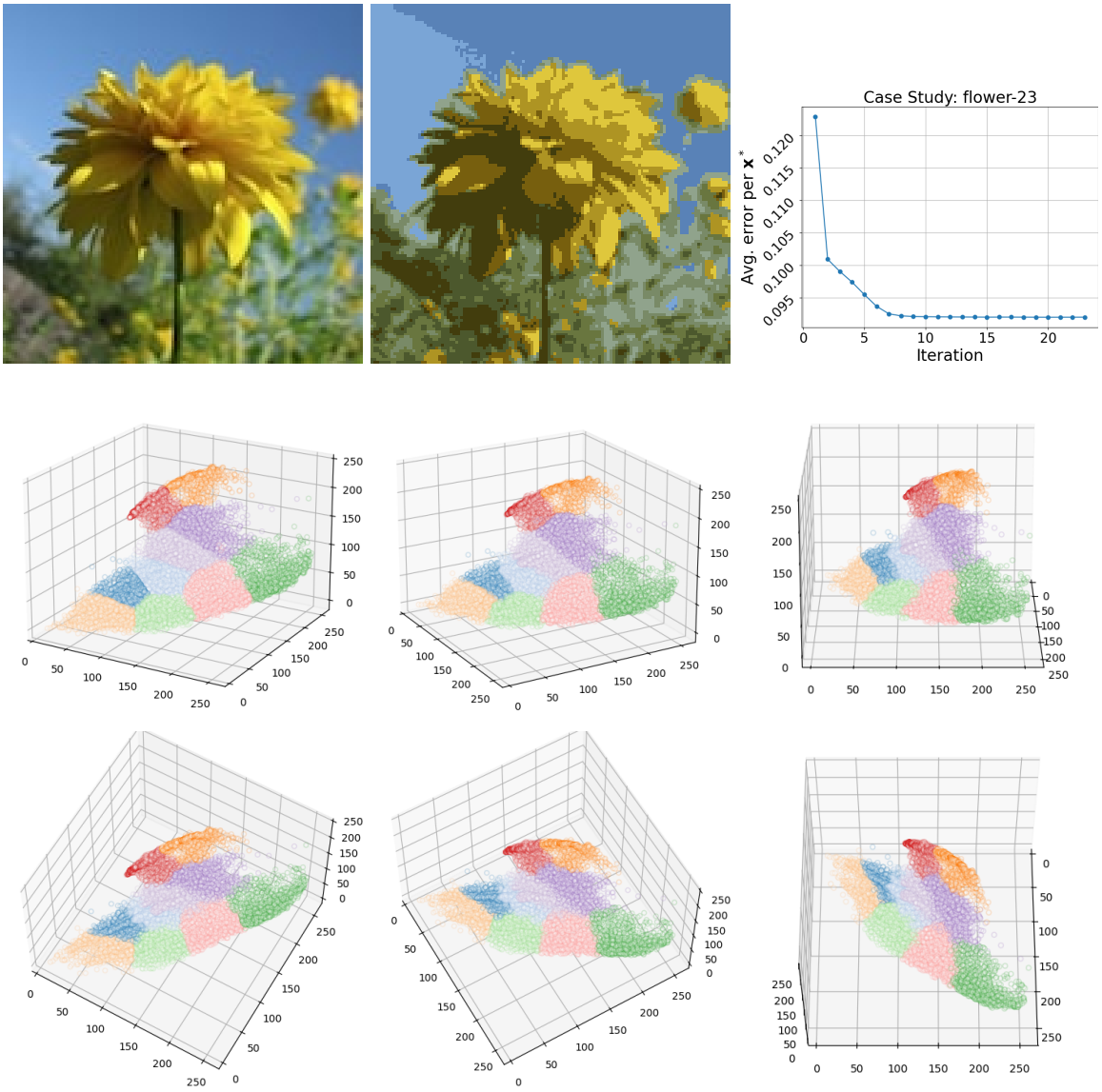


Figure 12: Case study: `horse-137.jpg`, $k = 10$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

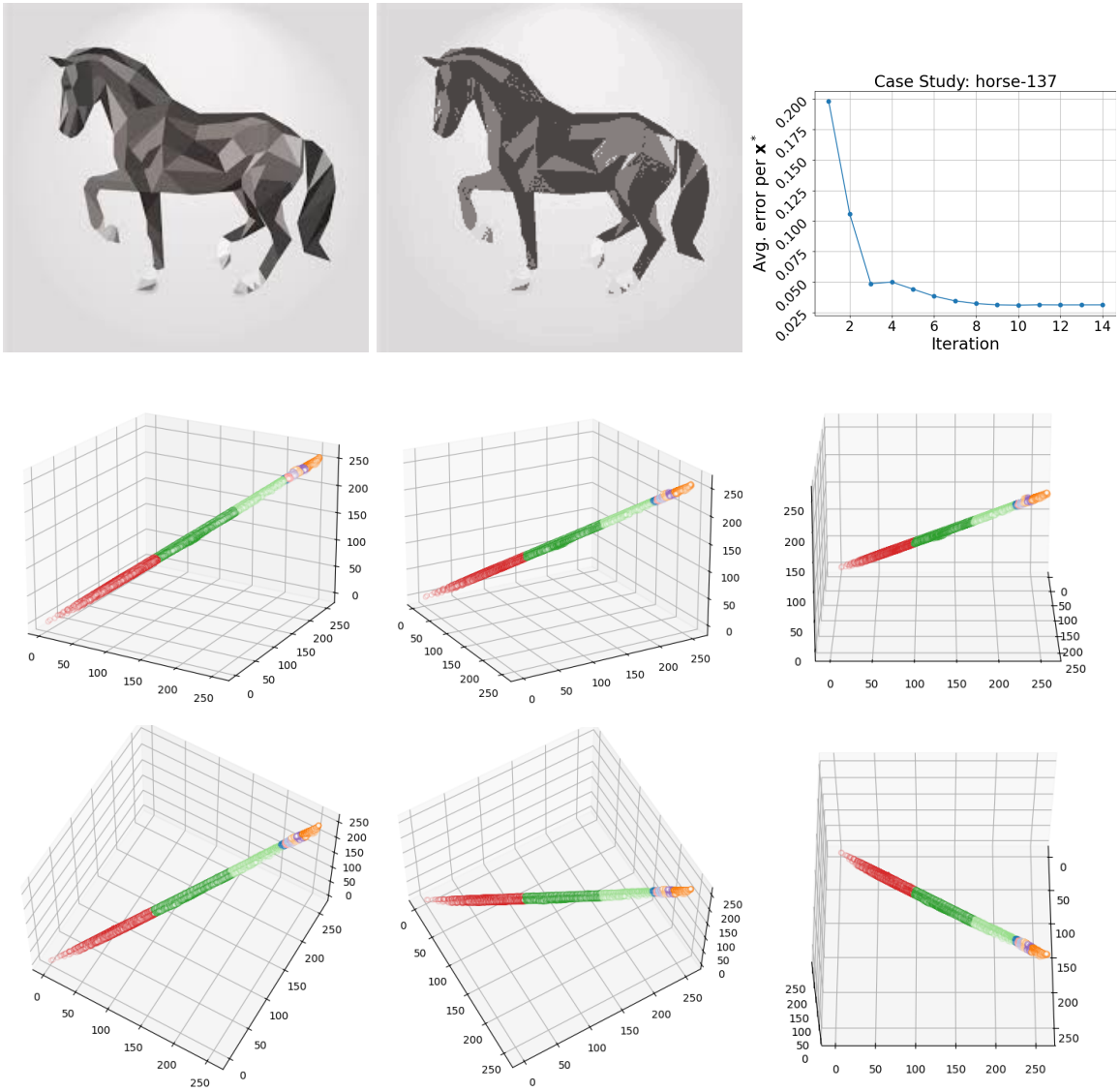


Figure 13: Case study: `horse-139.jpg`, $k = 15$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.

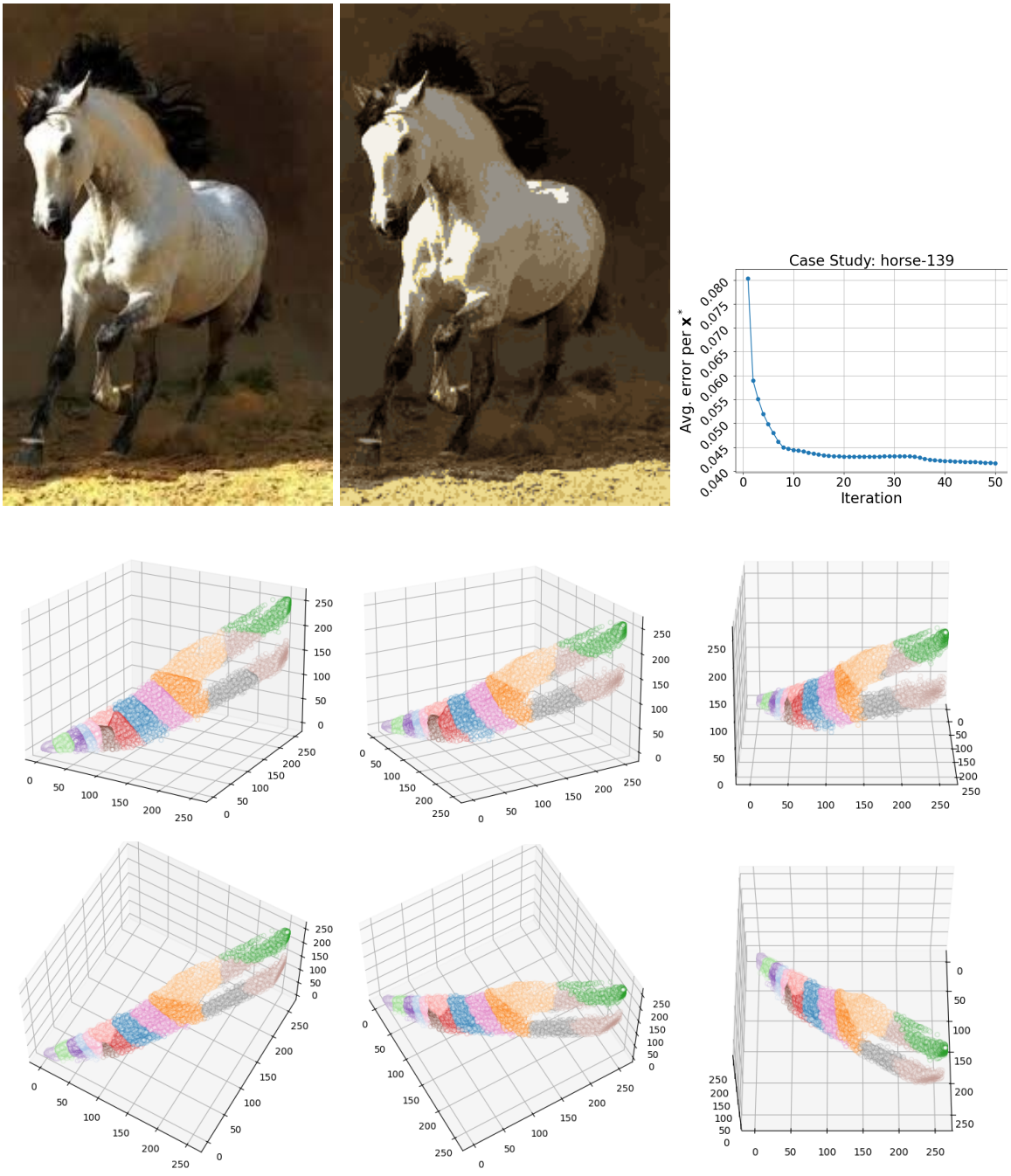
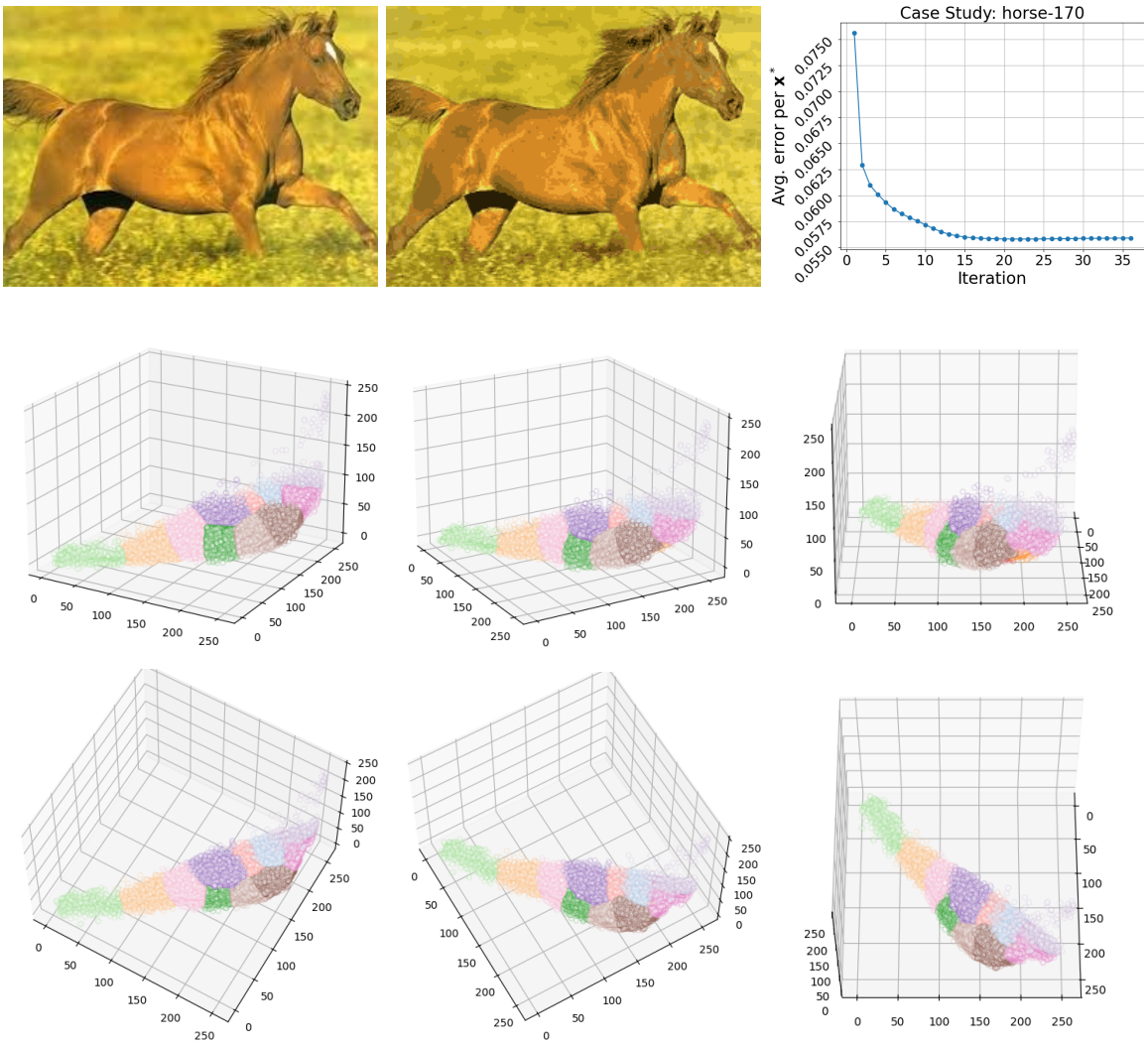


Figure 14: Case study: `horse-170.jpg`, $k = 15$. Original image, reconstructed image using k-means, reconstruction error, and clusterings in sample space.



► Question 3

You are given a dataset of 3168 labeled instances for voice-based gender classification: `data_gender_voice.csv`. Each instance contains 19 acoustic features extracted from voice recordings in the 0-280Hz frequency band. The associated label is in the last column ofthe dataset, where '1' indicates male and '0' indicates female.

▷ Item a

Perform a basic feature analysis: plot the histograms of the features and compute their correlations.

▷ Item b

Implement a logistic regression model to classify voice instances by gender. Use a 20% split for the test set, with the remaining instances for training. Keep in mind that the dataset is ordered, therefore, randomization is necessary before splitting. Consider whether preprocessing (e.g. normalization) is appropriate. Present and discuss the results on the test set, including:

- the ROC curve;
- the F1-score versus decision threshold curve.

See Chapter 20 of the textbook and this paper for more information about the ROC curve and F1-score.

▷ Item c

Which decision threshold is the most appropriate? Why? Using this threshold, compute and plot the confusion matrix and the classifier accuracy on the test set. Discuss your results.

Assuming that a collection of samples' classes follow gaussian distributions, a standard Bayesian approach would use a subset of the original dataset (i.e., the *training* dataset) to first estimate each of these underlying distributions' parameters (e.g., covariance matrix), as well as prior probabilities of each distribution, to tune a discriminant function. Then, with a subset from the original samples different than the training set (i.e., the *test* set), the discriminant is applied sample by sample and its results, the degree of belief that the sample belongs to a class, compared with the associated original classification.

The logistic regression, as in the Bayesian approach, assumes there are underlying gaussian distributions to the classifications, but the algorithm circumvents the parameter estimations by bringing the analysis straight to the data, weighting the attributes influences on the classifications. In particular, the logistic regression assumes that classes can be separated by lines in feature space, making the weighting linear.

This discussion merits mathematical detailing. Let:

- N be the number of samples;
- K be the number of different classes a sample can assume;
- d be the number of attributes;
- $\mathbf{x}_i$, a $1 \times d$ row vector, be the i-th sample;
- $\mathbf{X}_{N\times d}$ be the original dataset.

The discriminant associated with class k applied to sample $\mathbf{x}_i$ is denoted by b_{ki} and given by

$$b_{ki} = \mathbf{w}_k[1 \quad \mathbf{x}_i]^{\mathrm{T}}, \quad 1 \leq k \leq K \tag{10}$$

where $\mathbf{w}_k = [w_0\ w_1\ \ldots\ w_d]$ is a $1 \times (d+1)$ row vector whose entries correspond to the weight of each attribute in leading to class k (plus a term w_0 for the line's linear coefficient). Then, the probability that sample i belongs to class k is denoted by y_{ki} and given by[5]

$$y_{ki} = \frac{\exp(b_{ki})}{\exp(b_{1i}) + \exp(b_{2i}) + \cdots + \exp(b_{Ki})} = \frac{\exp(b_{ki})}{\sum_{k=1}^{K} \exp(b_{ki})}, \tag{11}$$

which can be shown to be equal to the posterior of class k given sample i. The associated original classifications are denoted by r_{ki} and given by

$$r_{ki} = \begin{cases} 1, & \mathbf{x}_i \text{ has class k} \\ 0, & \text{otherwise} \end{cases}, \tag{12}$$

to which the real-valued $\widehat{y}_{ki}$ is compared by "snapping" it to either 0 or 1, according to a decision threshold D:

$$y_{ki}^{s} = \begin{cases} 1, & \widehat{y}_{ki} > D \\ 0, & \text{otherwise} \end{cases}. \tag{13}$$

Now, the most important part of the algorithm is how to tune the weight vectors $\mathbf{w}_k$. In other words, there must be desgined a procedure to find $\mathbf{w}_k$ that maximizes the likelihood of y_{ki}. Assuming the likelihood $r_{ki}|\mathbf{x}_i$ is Bernoulli(y_{ki}), we have

$$\mathbf{w}_k = \underset{\mathbf{w}^*}{\text{argmax}} \prod_{i=1}^{N} y_{ki}(\mathbf{w}^*)^{r_{ki}} \tag{14}$$

$$= \underset{\mathbf{w}^*}{\text{argmin}} - \sum_{i=1}^{N} r_{ki} \log(y_{ki}(\mathbf{w}^*)) \tag{15}$$

$$= \underset{\mathbf{w}^*}{\text{argmin}}\, E_k(\mathbf{w}^*), \quad E_k(\mathbf{w}^*) = -\sum_{i=1}^{N} r_{ki} \log(y_{ki}(\mathbf{w}^*)). \tag{16}$$

One way of solving the minimization above is by using gradient-descent, i.e., iteratively updating $\mathbf{w}^*$ with the gradient of the scalar function[6]: $\mathbf{w}^* \leftarrow \mathbf{w}^* - \eta \nabla E_k$. Here, η is a positive constant, called the learning rate, that regulates how far along the gradient direction the update must go. Given the relationships between y_{ki}, b_{ki} and $\mathbf{w}_{ki}$ (equations (10)-(11)), one can show that

$$\nabla E_k = -\sum_{i=1}^{N} (r_{ki} - y_{ki})[1 \quad \mathbf{x}_i]. \tag{17}$$

The discussion above is nearly enough to derive an implementation of the logistic regression, but a few points must still be addressed:

- First, since b_{ki} is a weighted sum, attributes in greater ranges will have undue effect on the result. For that reason, it is important to normalize the data beforehand.
- Second, gradient descent is only applied during training, so N in eqs. (16) and (17) becomes $N_{\text{train}} < N$. Usually, $0.7 \leq N_{\text{train}}/N \leq 0.8$.
- Third, there is no one best way of initializing $\mathbf{w}^*$ in the minimization. A simple way of doing so is taking random small values, say between -0.01 and 0.01. Note that this range is reasonable only if data is normalized.

[5] This operation is also called softmax.
[6] Technically, this is the transpose of the gradient, which is given in column vector form.

- Fourth, the learning rate must be set. Unless more advanced methods are used, e.g. line search and moving average, trial and error is required.
- Finally, the algorithm must have a stop condition. An epoch is a top-to-bottom reading of training set within the context of gradient descent. A simple way of finishing the descent, and thus the logistic discrimination, is to simply impose a limit on the number of epochs, $e_{\max}$.

The implementation of the logistic regression made here has set $N_{\text{train}} = 0.8N$, $\mathbf{w}^*$ entries initialized in $[-0.01, 0.01]$ (with seed of value 242104677), $\eta = 0.1$ and $e_{\max} = 100$. The learning rate and number of epochs were obtained through trial and error, trying to make sure the algorithm finishes well into the elbow of the training error curve while also having low error rates.

The following is a pseudocode of the algorithm. Note that matrix multiplications have been used where convenient. It was defined

$$\mathbf{W} = \begin{bmatrix} \mathbf{w}_1 \\ \mathbf{w}_2 \\ \vdots \\ \mathbf{w}_K \end{bmatrix}, \quad \mathbf{R} = \begin{bmatrix} \mathbf{r}_1 \\ \mathbf{r}_2 \\ \vdots \\ \mathbf{r}_K \end{bmatrix}, \quad \mathbf{Y} = \begin{bmatrix} \mathbf{y}_1 \\ \mathbf{y}_2 \\ \vdots \\ \mathbf{y}_K \end{bmatrix} \tag{18}$$

where $\mathbf{r}_i = [r_{1i},\ r_{2i},\ \ldots,\ r_{Ki}]$ and $\mathbf{y}_i = [y_{1i},\ y_{2i},\ \ldots,\ y_{Ki}]$.

Algorithm 1: Implementation of the logistic regression (training). Samples are assumed to be zero-mean and normalized.

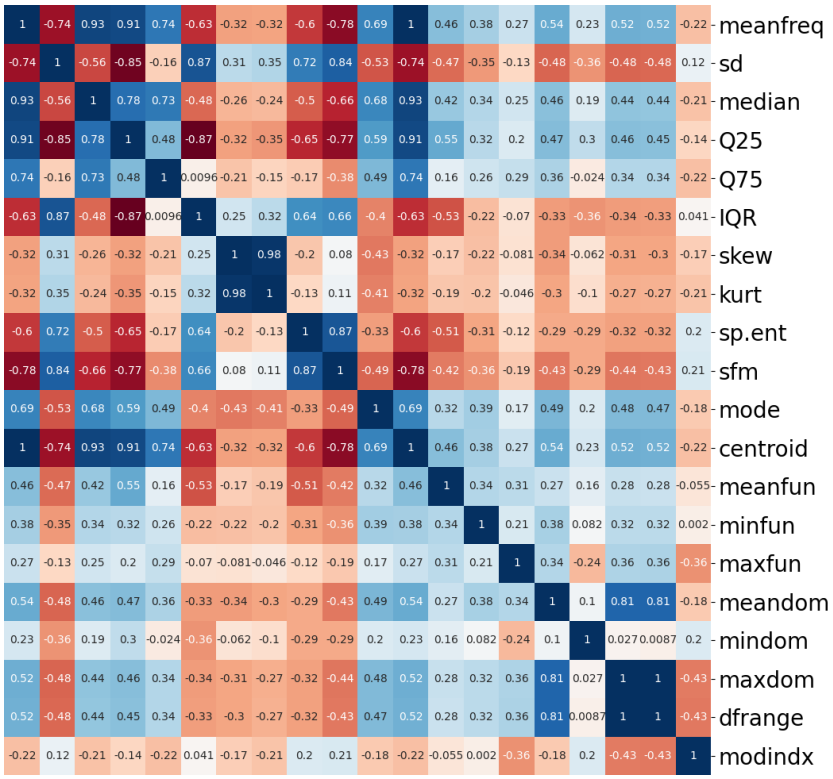
Data: $\mathbf{X}$, $\mathbf{R}$, η, $e_{\max}$
Result: $\mathbf{Y}$, $\mathbf{W}$
$N \leftarrow$ rows of N;
$d \leftarrow$ columns of N;
$k \leftarrow$ columns of R;
$\mathbf{W} \leftarrow [\ \text{random}(-0.01, 0.01)\ ]_{k \times d}$; ▷ `Weights initially random`
$epoch \leftarrow 0$;
while $epoch < e_{\max}$ **do**
  $\mathbf{Y} \leftarrow$ `calculate_predictions`$(\mathbf{W}, \mathbf{X})$; ▷ `Eqs.` (10), (11)
  $\boldsymbol{\Delta}\mathbf{W} = \sum_{i=1}^{N} (\mathbf{r}_i - \mathbf{y}_i)^T [1 \quad \mathbf{x}_i]$; ▷ `Opposite of gradient`
  $\mathbf{W} \leftarrow \mathbf{W} + \eta \boldsymbol{\Delta}\mathbf{W}$; ▷ `Gradient descent`
  $epoch \leftarrow epoch + 1$;
end

Algorithm 2: Class prediction calculation algorithm, with $\mathbf{u}_k$ the k-th row of $\mathbf{U}$.

Data: $\mathbf{W}$, $\mathbf{X}$
Result: $\mathbf{Y}$
$N \leftarrow$ rows of $\mathbf{X}$;
$K \leftarrow$ columns of $\mathbf{W}$;
$\mathbf{B} \leftarrow \mathbf{W}\ [\mathbf{1}_{N \times 1}\ \mathbf{X}]^{\mathrm{T}}$; ▷ `Matrix form of` (10)
$\mathbf{U} \leftarrow \exp.(\mathbf{B})$; ▷ `Pointwise exponentiation of` $\mathbf{B}$
$\mathbf{V} \leftarrow \mathbf{1}_{k \times 1} \sum_{k=1}^{K} \mathbf{u}_k$; ▷ `Broadcast of row sums`
$\mathbf{Y} \leftarrow \mathbf{U}./\mathbf{V}$; ▷ `Pointwise division,` (11)

Before presenting the results of the logistic regression, a discussion on the dataset is necessary. fig. 15 presents the correlations between attributes, while fig. 17 shows each attribute's histogram.

Figure 15: Correlation between attributes as a colormap. The original correlation matrix was reorganized as to cluster bigger correlations at the upper left corner.



In training and testing, attributes with correlation greater than 0.95 were considered to add too little information to the sample, and thus removed from the dataset in order to simplify the arithmethic. By inspecting fig. 15, one can see that, in all, three pairs of attributes are above the threshold. Of those six, those removed were `meanfreq`, `kurt` and `maxdom`. Indeed, fig. 17 shows that the removed three are nearly identical to their pairs.

Having made it clear that the algorithm is now working with only 17 of the original 20 attributes, the results can be presented and discussed. fig. 16 presents the weight matrix $\mathbf{W}$ obtained after training and used in testing. Then, fig. 18a and fig. 18b show the F1-score and ROC for varying decision thresholds D (see (13)) obtained in training and testing, respectively.

Figure 16: Weight matrix as a colormap. Bottom row classifies "male", top "female".

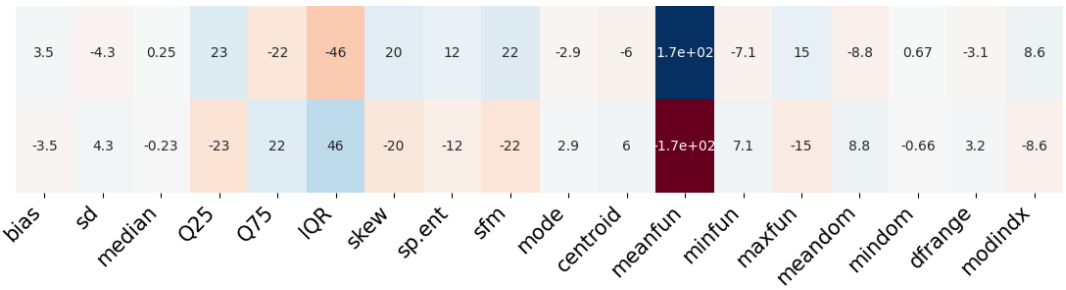
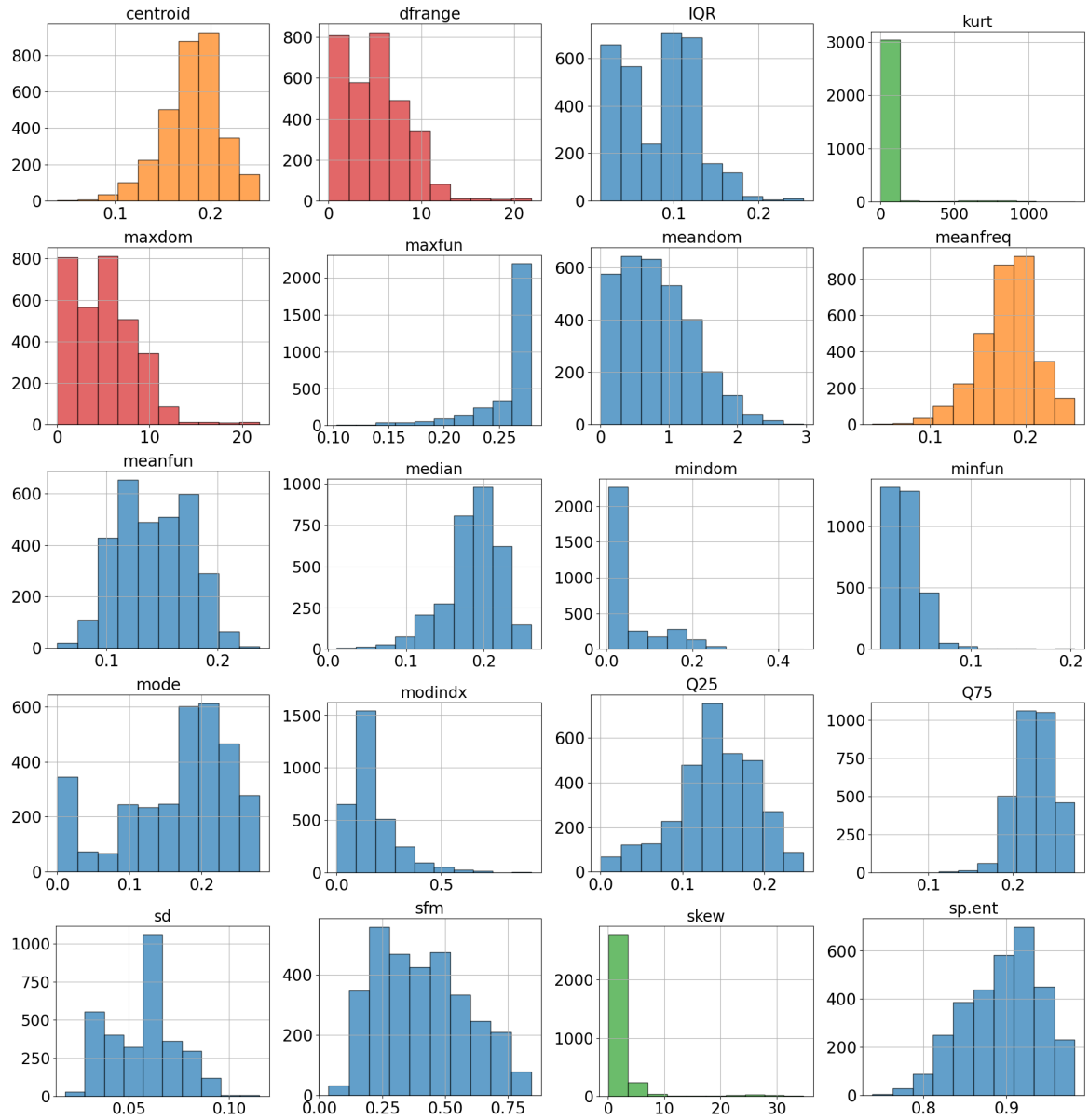


Figure 17: Histograms of attributes. Highly correlated (removed) attributes were colored either orange, green or red.



The F1-Score and ROC curves below show clearly that the regression produces reliable results regardless of threshold value, validating the model. A threshold of about 0.35, in particular, optimizes the F1-Score in testing while maintaining very high F1-Score in training. As such, this value is likely to be the most appropriate for this dataset.

For this threshold, the confusion matrix yelded the rates in fig. 19, and overall accuracy of 97.00%. This was expected because the F1-Score has its second, greatest plateau close to that threshold and the ROC's associated second point is also close to 97% (between 97% and 96.9%).

Figure 18: F1-Score and ROC for (a) training and (b) testing. All rates refer to "male" class.

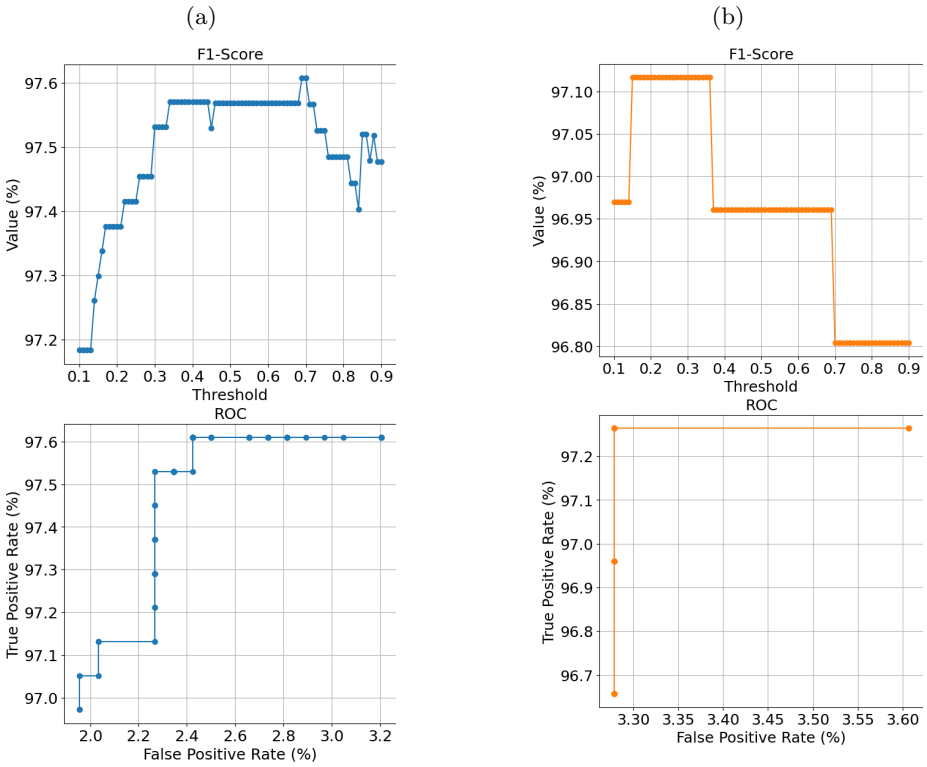
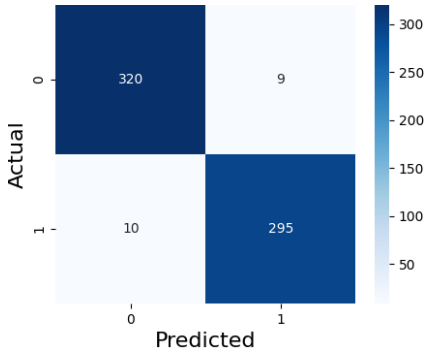


Figure 19: Confusion matrix of the regressor using $D = 0.35$ in test set.



Quick note on code

This report is the textual part of a greater bundle that also encompasses code. The algorithms discussed in this report (PCA, k-means, logistic regression) were implemented by the author in the Rust programming language, and stores the relevant data for later analysis as CSV files. Rust is known for its memmory safety and fast performance, even among compiled languages. However, given its focus on systems applications, there is a lack of robust, mature plotting libraries. Therefore, a Python script (`plots.py`) was written to process and extract from Rust's results any graph, plot or image that was deemed necessary to include in this report.

Both the Rust source code and well as the Python script were included in the bundle, and are available together with this report in an open Github repository made by the author:

https://github.com/Thiago-TP/Aprendizado-de-Maquina/tree/main/list_2.