Practice 1: Unsupervised Learning

Machine Learning

Grado en Ingeniería Informática y Tecnologías Virtuales

Ingeniería de Datos I

Grado en Ingeniería del Software

Academic year 2025/2026

Antonio M. Durán Rosal

Practice 1: Unsupervised Learning

The objective of this practice is not only to use the Python libraries for Machine Learning but also to know how to interpret the results obtained and the decisions taken in the process.

For this reason, the practical evaluation will consider both the code and the comments made in each requested section.

Each exercise has the same value.

Practical Analysis

1. Exercise 1

Without considering the class attribute, analyse the clusters created by the K-means algorithm for the Zoo database:

a) Test 5, 6, 7 and 8 clusters.

b) Run the algorithm using 3 seeds for each number of clusters. Obtain the mean values in terms of error.

c) Visualise the resulting clusters by representing two of the attributes.

d) Repeat the same process, including the class attribute, and analyse if there are differences between the clusters created.

2. Exercise 2

Using agglomerative clustering, perform a clustering of the Zoo database patterns by testing all the links available in scikit-learn:

a) Calculate the external validation metrics.

b) Decide which is the optimal number of clusters.

c) Plot the dendogram.

d) Analyse the results.

3. Exercise 3

Using Python, check that the solution to Problem 5 of Problems Bulletin 1 is correct.

Clustering for Image Compression

In this part, clustering algorithms will be used with the goal of reducing the number of colours in an image and its size.

In a standard image, each pixel is represented according to the RGB encoding. That is, each pixel consists of 3 sets of 8 bits with values from 0 to 255 that represent the red, green, and blue intensities of the pixel.

We will use K-Means clustering algorithm with the aim of finding reduced colour types, so that each pixel will be replaced by its prototype.

4. Implementation of helper functions

The following functions must be implemented using the `PIL` and `os` libraries:

```python
from PIL import Image
import numpy as np
import matplotlib.pyplot as plt
import os
```

- `load_image(path)`: loads a JPG or PNG image into memory and converts it into a numpy array.
- `show_image(image)`: displays the image contained in a numpy array in memory.
- `save_image(image, path)`: saves three images in PNG and JPG formats in the file indicated by `path`. **This is provided to you**.

```python
def save_image(image, path, k):
  img = Image.fromarray(image)
  img.save(path+".jpg", format="JPEG")
  img.save(path+".png", format="PNG")
  indexed_image = img.convert("P", palette=Image.ADAPTIVE, colors=k)
  indexed_image.save(path+"compressed.png", optimize=True)
```

- `get_size(path)`: returns the size of a file in KB.

5. Apply clustering to reduce the number of colours

Apply K-Means to reduce the colours of an image with $k = 3, 5, 10, 16, 20, 32, 50, 64$. For landscape files, only with $k = 5, 10$, and 20.

Images must be read from the provided `images` folder and the resulting reduced images must be saved in `reduced_images` following the pattern:
`originalname_k`, where:

- `originalname`: will be the original name of the image.
- `k`: indicates the number of colours obtained.

6. Relationship between KB and number of colours

At this point, for each image and number of colours, three images have been generated: one in JPG format and two in PNG format.

For each image, a graph must be shown that relates the size of the images as a function of the number of colours used by the K-Means method.

What happens in each image? Interpret the result, investigate, and justify why the reduction is not applied in the same way in all images and formats used.

Dimensionality Reduction

7. Reduction of the input space

The goal of this third part of the practical session is to use the `PCA` algorithm to reduce the size of the input space. We will work with the file `faces.mat` available on Moodle. The dataset in question contains 32x32 grayscale images. Each row of the matrix X corresponds to a face (a row vector of size 1024).

We must plot the original data and the data represented in the first 100 components of the PCA analysis.

For this purpose, the function `displayData(X)` is provided: this function allows us to plot the images. We are going to plot the first 25 original and reduced faces.

```python
def display_data(X):
  example_width = int(np.round(np.sqrt(X.shape[1])))
  example_height = X.shape[1] // example_width

  display_rows = int(np.floor(np.sqrt(X.shape[0])))
  display_cols = int(np.ceil(X.shape[0] / display_rows))

  fig, ax_array = plt.subplots(display_rows, display_cols, figsize=(8, 8))

  for i, ax in enumerate(ax_array.flat):
    if i >= X.shape[0]:
      break
    ax.imshow(X[i].reshape(example_height, example_width), cmap='gray')
    ax.axis('off')
  plt.show()
```

8. Percentage of variance explained by each component

Represent the percentage of variance explained by each variable in an elbow plot.

9. Testing our PCA as preprocessing

Use any dataset with a sufficient number of features, reduce its dimensionality, and check the performance of a classifier with and without applying PCA.