

Classification of handwritten Greek letters using Logistic Regression and Principal Component Analysis

September 20, 2023

1 Introduction

In this report we are using machine learning classification methods on a dataset of handwritten Greek letters scanned as .jpg images. Our goal is to train a Logistic Regression -classifier on a PCA-reduced dataset which can correctly predict a presented letter given from the same dataset.

The structure of this report is roughly the following:

1. Introduction
2. Problem Formulation
3. Applying the Machine Learning methods
4. Results
5. Conclusions
6. References
7. Appendices

2 Problem Formulation

2.1 Datapoints

The dataset is from GCDB (Greek Character DataBase), which has a database of Greek unconstrained handwritten characters. It consists of 336 images of Greek letters. The same dataset can be also found on Kaggle.com. The sets have been split into a training set of 240 images (10 images/letter) and a testing set of 96 images (4 images/letter). The images are later transformed into 8-bit color format, commonly known as grayscale, where every pixel has a value representing the light intensity in range $[0, 255]$. Moreover, black has the weakest intensity of 0 and white the highest of 255.

2.2 Features

The feature of each image is presented in a vector $x \in \mathbf{R}^{784}$ of integers in the range $[0, 255]$. The original images are formatted in 28x28 matrices so the entries $[0, 27]$ correspond to the first row and so on.

2.3 Labels

There are 24 letters in the Greek alphabet so each image can be labeled from 0 to 23 where the index corresponds to a specific letter. (See appendices)

3 Used methods

3.1 Principal Component Analysis (PCA)

The original pictures were imported from Kaggle.com but they were varying-sized black and white-images, so we performed a conversion to 28x28 grayscale images using Lanczos interpolation. After that the pixel intensity values were stored in 1D numpy arrays.

Each image's data is stored in a \mathbf{R}^{284} -dimensional vector consisting of intensity ratings of the pixels as stated previously. By fitting PCA into the training data, our goal is to reduce the dimensionality but not lose too much data. The main purpose of using PCA is to make analyzing the data points easier and faster to process while preserving as much information as possible. PCA itself doesn't have a direct loss-function but rather performs various mathematical operations. First, it computes a covariance matrix to determine if there is some relationship between the variables. After that it computes the eigenvalues and eigenvectors of the covariance matrix to determine principal components. Those are new variables that are linear combinations of the initial variables in such way that new variables (principal components) are uncorrelated and most of the original features are squeezed into the first components.

4 Results

5 Conclusions

6 References

7 Appendices

7.1 Code Initialization

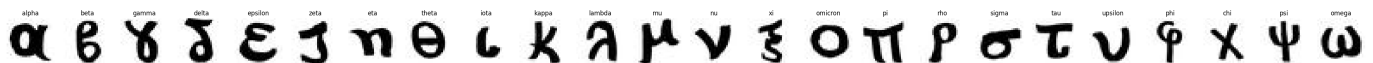
```
1 import pandas as pd
2 import matplotlib.pyplot as plt
3 from nose.tools import assert_equal
4 from numpy.testing import assert_array_equal
5 import os
6 import numpy as np
7 from PIL import Image
8 from sklearn.decomposition import PCA
9
10 %config Completer.use_jedi = False
11 from sklearn.linear_model import LinearRegression
12 from sklearn.metrics import mean_squared_error
13
14 # prepares an output directory named "grayscale_images" and ensures it exists,
15 # creating it if necessary.
16 # iterates through JPG image files located in the "greek_character_database" directory
17 # converting each image to grayscale using the 'L' mode with the convert('L') method
18 # and resizing it to a 28x28 pixel size using Lanczos interpolation with image.resize
19 # ()
20 # saves the converted and resized grayscale images in the "grayscale_images" directory
21 # with the same filenames as the original images.
22
23 input_directory = "greek_character_database"
24 output_directory = "grayscale_images"
25
26 os.makedirs(output_directory, exist_ok=True)
27
28 for filename in os.listdir(input_directory):
29     if filename.endswith(".jpg"):
30         image = Image.open(os.path.join(input_directory, filename)).convert('L')
```

```

17     image = image.resize((28, 28), Image.LANCZOS)
18
19     output_path = os.path.join(output_directory, filename)
20     image.save(output_path)

1  # reads grayscale images from a directory, categorizes them into classes based on the
    letters,
2  # then displays a representative image from each class in a row for visualization.
3
4  df = "grayscale_images"
5
6  class_images = {}
7
8  greek_letter_order = ["alpha", "beta", "gamma", "delta", "epsilon", "zeta", "eta", "
    theta", "iota", "kappa", "lambda", "mu", "nu", "xi", "omicron", "pi", "rho", "sigma
    ", "tau", "upsilon", "phi", "chi", "psi", "omega"]
9
10 for filename in os.listdir(df):
11     if filename.endswith(".jpg") and filename.startswith("class_"):
12         class_name = filename.split("_")[1].split(".")[0]
13         if class_name not in class_images:
14             class_images[class_name] = []
15
16         file_path = os.path.join(df, filename)
17         img = plt.imread(file_path)
18         class_images[class_name].append(img)
19
20 fig, axes = plt.subplots(1, len(class_images), figsize=(50, 20))
21
22 for i, class_name in enumerate(greek_letter_order):
23     ax = axes[i]
24     ax.imshow(class_images[class_name][0], cmap='gray')
25     ax.set_title(f'{class_name}')
26     ax.axis('off')
27
28 plt.show()

```



alpha beta gamma delta epsilon zeta eta theta iota kappa lambda mu nu xi omicron pi rho sigma tau upsilon phi chi psi omega

α β γ δ ε ζ η θ ι κ λ μ ν ξ ο π ρ σ τ υ φ χ ψ ω

α	0
β	1
γ	2
δ	3
ϵ	4
ζ	5
η	6
θ	7
ι	8
κ	9
λ	10
μ	11
ν	12
ξ	13
\omicron	14
π	15
ρ	16
σ	17
τ	18
υ	19
ϕ	20
χ	21
ψ	22
ω	23

Table 1: Caption

Conversion to vector format:

```

1 # reads a collection of JPG image files from a specified directory,
2 # converts each image into a 1D NumPy array of pixel values, and stores these arrays
  in image_data.
3 # Additionally, it keeps track of the filenames associated with each image in the
  file_names list.
4
5 image_data = []
6 file_names = []
7
8 for filename in os.listdir(df):
9     if filename.endswith(".jpg"):
10         file_path = os.path.join(df, filename)
11         file_names.append(filename)
12
13         img = Image.open(file_path)
14         img_array = np.array(img).flatten()
15         image_data.append(img_array)
16
17
18 image_data = np.array(image_data)
19
20
21 # performs Principal Component Analysis (PCA) on a dataset of generated features (
  image_data)
22 # with a specified number of components (n_components).
23 # visualizes the contribution of each generated feature (principal component) to the
  overall variance in the data
24 # and shows how much variance is captured as more components are included in the
  analysis.
25
26 n_components = 50
27 pca = PCA(n_components=n_components)
28 pca.fit(image_data)
29
30 fig, ax1 = plt.subplots(figsize=(12, 5))
31 color = 'tab:blue'

```

```

12 ax1.bar(1+np.arange(n_components), pca.explained_variance_ratio_, color=color)
13 ax1.set_xticks(1+np.arange(n_components, step=2))
14 ax1.tick_params(axis='y', labelcolor=color)
15 ax1.set_ylabel("Explained variance ratio", color=color)
16 ax1.set_xlabel("Generated feature")
17
18 ax2 = ax1.twinx()
19 color = 'tab:red'
20 ax2.tick_params(axis='y', labelcolor=color)
21 ax2.plot(1+np.arange(n_components), np.cumsum(pca.explained_variance_ratio_), color=
    color)
22 ax2.set_ylabel("Cumulative explained variance ratio", color=color)
23 fig.tight_layout()
24 plt.show()

```

