Lab 1: Eigenfaces

Gwen Lofman, October 24, 2018

Contents

1 Instructions

In order to run the package, the rust build toolchain must be installed on the computer. The preferred installation method is using the build script from rustup.rs:

```
curl https://sh.rustup.rs -sSf | sh
```

This will install the package manager, compiler, and other important tools. Once the rust toolchain has been installed, clone the project from GitHub using:

```
git clone https://github.com/glfmn/eigenfaces.git
```

Finally, enter the newly cloned project and use cargo to download the dependencies, compile the program, and run it using:

```
cargo run --release -- -o <output-path>
```

The program will create all of the output images at the specified `output-path`, creating any directories necessary.

Alternatively, if a copy of the code is already obtained, then simply run the above `cargo` command to run the program.

1.1 Dependencies

1. `gnuplot` to generate the error plot
2. `cairo` for eps conversion

2 Code Overview

The entire code is located in `src/main.rs`, with dependencies specified in `Cargo.toml`.

Using the `nalgebra` library to do the heavy lifting (which calls down to `BLAS`), I implemented the facial recognition code in Rust. Rust is a systems programming language that is pretty young but promises an interesting combination of features. I've used it for a few projects but never anything mathematical so I wanted to see how it compares to `Clojure`, `Python`, and `Matlab`.

First, I defined all of the operations as functions which take mutable and immutable references to the data they need, a mix between functional and c-style code. Then I wrote a main function which performs the entire routine and returns the error of the data-set for plotting. Finally, I called the "main routine" for each variation in the lab.

With more time the performance could have been improved by not needlessly recomputing the entire experiment for each step in part B. On my computer it still takes less than 3 seconds.

3 Results

(a) Test (b) Gallery (c) Test (d) Gallery

Figure 1: Randomly selected test set images and their detected closest matches from the gallery set

3.1 Error

Interestingly enough, the lowest error was at a PCA Basis size of 30, dropping nearly 5% at 40 and increasing again at 50.

Examining a few of the wrongly matched images in figure 3 reveals the limitations of eigenfaces when dealing with changes lighting: a poorly lit image for one white person in the data-set matches the value of the black person in the data set, but they are matched together likely because of the similarities in value of the skin in the lighting of each photograph.

Part C has an accuracy of 86%, which is higher than that found in Part A. This means that the alternate training set of 4 images per person may improve accuracy.

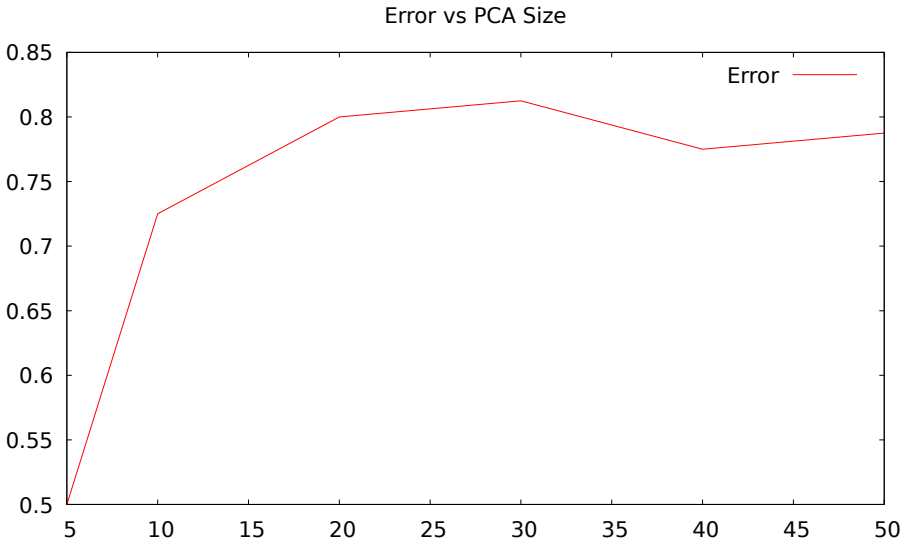

Figure 2: y-axis is the percent error, x-axis is the basis size of the PCA, or the number of selected eigenfaces.

Figure 3: Above is the test image and below is the mis-matched gallery image.

3.2 Most and Least Significant Eigenfaces

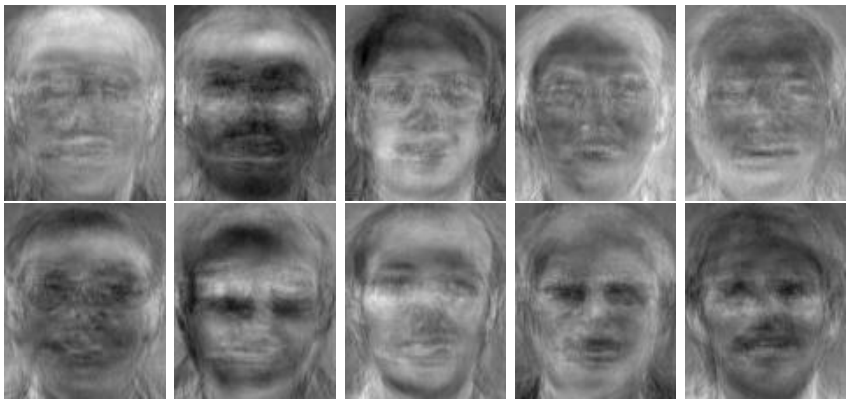

Figure 4: The ten most significant eigenfaces of the basis setfor part A, in order from most to least significant, left being most significant.

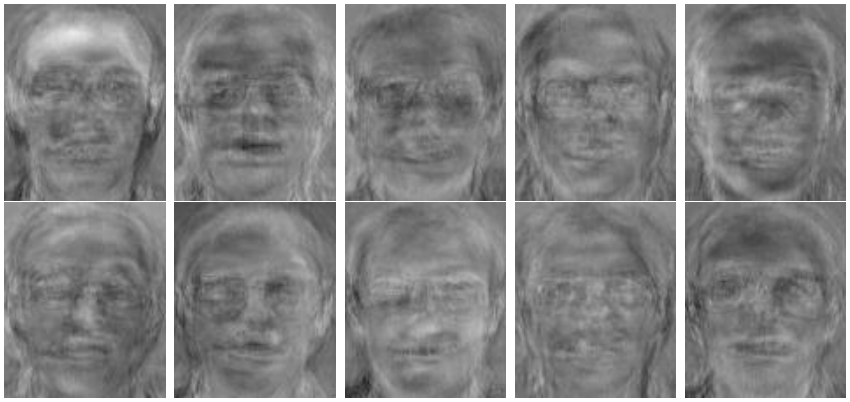

Figure 5: The ten *least* significant eigenfaces of the basis setfor part A, in order from most to least significant, left being most significant.

The most significant eigenfaces appear to resemble the "components" of a face, where the first 10 capture larger variations in face shape and eye, nose, mouth, position, and the less significant ones capture finer grain variation and changes in value that affect the outlines of the larger features.