Assignment 1: PCA and Genetic Algorithms

Panagiotis Giagkoulas (s3423883)
Carlos Huerta (s3743071)
Saim Eser Comak (s3432548)
Emile Muller (s3787915)
Danny Rogaar (s2393344)
Daan Opheikens (s3038416)

Group 14

September 16, 2018

1.1.1: Eigen values and Eigen vectors

Given the mathematical identity that a matrix multiplied by an eigen vector($\vec{v}$) will be equal to same eigen vector multiplied by an eigen value(λ), we can deduce the following steps

$$A\vec{v} = (some\ scalar\ value(\lambda) * Identity\ matrix)\vec{v} \tag{1}$$

$$A\vec{v} - \lambda I\vec{v} = 0$$
$$\vec{v}\,(A - \lambda I) = 0$$

The only way to obtain zero in a vector by matrix multiplication in which the vector is non-zero, the determinant of the expression (A - λI) has to be zero. Which gives us the following expression:

$$det(A - \lambda I) = 0 \tag{2}$$

Next, the determinant is calculated. The first step is to subtract λ from the diagonal values of the matrix A:

$$\begin{bmatrix} 2-\lambda & 3 \\ 5 & 8-\lambda \end{bmatrix}$$

Followed by the subsequent algebraic steps:

$$(2-\lambda)(8-\lambda) - 15 = 0$$
$$\lambda^2 - 10\lambda + 1 = 0$$
$$\lambda_1 = 9.8989\ (5 + 2\sqrt(6))$$
$$\lambda_2 = 0.1010\ (5 - 2\sqrt(6))$$

Now that the eigen values are calculated, we can place them into the original equation 1 and find eigen vectors:

$$A\vec{v} = 9.8989\vec{v}\ (i) \quad \text{and} \quad A\vec{v} = 0.1010\vec{v}\ (ii)$$

Let's first solve equation (i)

$$\begin{aligned} 2x_1 + 3x_2 &= 9.8989x_1 \\ 3x_2 &= 7.8989x_1 \\ x_2 &= \frac{7.8989x_1}{3} \quad (a) \end{aligned} \qquad \begin{aligned} 5x_1 + 8x_2 &= 9.8989x_2 \\ 5x_1 &= 1.8989x_2 \\ x_1 &= \frac{1.8989x_2}{5} \quad (b) \end{aligned}$$

Equation (ii)

$$\begin{aligned} 2x_{11} + 3x_{22} &= 0.1010x_{11} \\ 3x_{22} &= -1.899x_{11} \\ x_{22} &= \frac{-1.899x_{11}}{3} \quad (c) \end{aligned} \qquad \begin{aligned} 5x_{11} + 8x_{22} &= 0.1010x_{22} \\ 5x_{11} &= -7.899x_{22} \\ x_{11} &= \frac{-7.899x_{22}}{5} \quad (d) \end{aligned}$$

We can use either of the equalities to calculate our eigen vectors.Currently we will use equatilies (b) and (d) to calculate eigen vectors for matrix A. Our vectors become then:

$$\vec{v}_1 = \begin{bmatrix} 1.8989 \\ 5 \end{bmatrix} \quad \vec{v}_2 = \begin{bmatrix} -7.899 \\ 5 \end{bmatrix}$$

Finally we transform the eigen vectors obtained for matrix A into unit length:

$$\begin{aligned} \vec{v}_1 &= \begin{bmatrix} \frac{1.8989}{\sqrt{1.8989^2+5^2}} \\ \frac{5}{\sqrt{1.8989^2+5^2}} \end{bmatrix} \quad & \vec{v}_2 &= \begin{bmatrix} \frac{-7.899}{\sqrt{-7.899^2+5^2}} \\ \frac{5}{\sqrt{-7.899^2+5^2}} \end{bmatrix} \\ &= \begin{bmatrix} 0.3550380 \\ 0.9348519 \end{bmatrix} & &= \begin{bmatrix} -0.8449496 \\ 0.5348459 \end{bmatrix} \end{aligned} \tag{3}$$

To perform same steps for the matrix B, we will use the equation 2 to calculate determinant of the following matrix:

$$\begin{bmatrix} 1-\lambda & 2 \\ 4 & 1-\lambda \end{bmatrix}$$

The algebraic steps are:

$$\begin{gathered} (1-\lambda)(1-\lambda) - 8 = 0 \\ \lambda^2 - 2\lambda - 7 = 0 \\ \lambda_1 = 3.8284 \ (1 + 2\sqrt{(}2)) \\ \lambda_2 = -1.8284 \ (1 - 2\sqrt{(}2)) \end{gathered}$$

Then we find our eigen vectors solving following equations:

$$B\vec{v} = 3.8284\vec{v} \ (iii) \quad \text{and} \quad B\vec{v} = -1.8284\vec{v} \ (iv)$$

Equation(iii)

$$\begin{aligned} 1x_1 + 2x_2 &= 3.8284x_1 \\ 2x_2 &= 2.8284x_1 \\ x_2 &= \frac{2.8284x_1}{2} \quad (a) \end{aligned} \qquad \begin{aligned} 4x_1 + 1x_2 &= 3.8284x_2 \\ 4x_1 &= 2.8284x_2 \\ x_1 &= \frac{2.8284x_2}{4} \quad (b) \end{aligned}$$

Equation(iv)

$$\begin{aligned} 1x_{11} + 2x_{22} &= -1.8284x_{11} \\ 2x_{22} &= -2.8284x_{11} \\ x_{22} &= \frac{-2.8284x_{11}}{2} \quad (c) \end{aligned} \qquad \begin{aligned} 4x_{11} + 1x_{22} &= -1.8284x_{22} \\ 4x_{11} &= 2.8284x_{22} \\ x_{11} &= \frac{-2.8284x_{22}}{4} \quad (d) \end{aligned}$$

We will use identity (b) and (d). Then we transform the eigen vectors obtained for matrix A into unit length:

$$\vec{v}_1 = \begin{bmatrix} \frac{2.8284}{\sqrt{2.8284^2+4^2}} \\ \frac{4}{\sqrt{2.8284^2+4^2}} \end{bmatrix} \quad \vec{v}_2 = \begin{bmatrix} \frac{-2.8284}{\sqrt{-2.8284^2+4^2}} \\ \frac{4}{\sqrt{-2.8284^2+4^2}} \end{bmatrix}$$
$$= \begin{bmatrix} 0.5773466 \\ 0.8164992 \end{bmatrix} \quad = \begin{bmatrix} -0.5773466 \\ 0.8164992 \end{bmatrix} \tag{4}$$

1.1.2: ANOVA

We want to compute an one-way ANOVA table, in order to estimate whether the means of the three categories are similar or not. Our hypothesis is: $H_0 : \bar{x}_h = \bar{x}_m = \bar{x}_l$. To fill in this ANOVA table we have to follow the steps that are described beneath.
First we summarize the data from our three categories ($k = 3$), to help us in our calculations:

Table 1: Representative statistic measurements of samples

Attendance	High	Medium	Low
Sample Size (n)	9	6	5
Mean ($\bar{x}$)	7.66667	7.08333	6.6
Standard Deviation (σ)	1.06066	0.73598	0.82158
Variance (σ^2)	1.125	0.54167	0.675

It should be noted that the standard deviation and variance have been calculated by dividing with $(n_i - 1)$, since we are working with samples and not the total population and want to calculate the non-biased variance of the samples. After computing our table, we can now calculate the grand mean of the sample means:

$$\bar{\bar{x}} = \frac{n_h * \bar{x}_h + n_m * \bar{x}_m + n_l * \bar{x}_l}{n_h + n_m + n_l} = 7.225 \tag{5}$$

Which we will use to calculate the sum of squares between groups :

$$SS_B = \sum_{n=1}^{k} n_i * (\bar{x}_i - \bar{\bar{x}})^2 = 3.82917 \tag{6}$$

The degrees of freedom between groups will be the number of categories minus one($k-1$), so we have $df_B = 2$. Moving on by dividing the sum of squares between groups with the degrees of freedom between groups, we get the mean of squares between groups:

$$MS_B = \frac{SS_B}{df_B} = 1.91458 \tag{7}$$

Next step is the calculation of the within group sum of squares:

$$SS_W = \sum_{n=1}^{k} (n_i - 1)\sigma_i^2 = 14.40833 \tag{8}$$

In this case the degrees of freedom are calculated as $N - k$, where N is the total sample size. Therefore $df_w = 17$. By dividing the sum of squares within group with the degrees of freedom within group, we calculate the mean of squares within group:

$$MS_W = \frac{SS_W}{df_W} = 0.84755 \tag{9}$$

Finally we can compute the f-statistic by dividing the mean of squares between group with the mean of squares within group:

$$MS_B = \frac{SS_B}{MS_W} = 2.25896 \tag{10}$$

Below is the complete ANOVA table:

Table 2: Completed ANOVA table for given samples

Source	SS	df	MS	F
Between	3.82917	2	1.91458	2.25896
Within	14.40833	17	0.84755	-
Total	18.2375	19	-	-

In order for us to reach a conclusion regarding our hypothesis, we use $\alpha = 0.05$, which corresponds to $F - critical = 3.59153$. For the selected significance level, our F-value is lower than the F-critical and thus we can not reject our null hypothesis and say that at least one of the three samples have significantly different means and thus belong to an entirely different population.

1.2.1: Implementation of the F-statistic

Here we are looking to write down a Matlab function that can compute the F-statistic of a given set of independent (IV) and dependant variables (DV). We first verify the error conditions, which are IV and DV must have the same size, and DV has more than one element. Then we create an array (DV2) that has the number of element of each category, and we recover the elements of each category of DV from IV (3 categories = 3 arrays).

Once that is done we calculate the grand mean of the three categories. With the grand mean, we can compute SS(B). We then calculate the variance of each categories. And then finally SS(W). With SS(B) and SS(W) we can determine the F-statistic, which, for the given example has a value of 2.2590. The code of this exercise is on the github repository. `https://github.com/RUG-IDS/team-14`

1.2.2: Principal Component Analysis

The principal components analysis allow us to find out patterns in high dimensional data where we assume a linear relationship between the variables and visualization is not an option. The code of this exercise is on the github. repository. `https://github.com/RUG-IDS/team-14`

1.2.3: Genetic Algorithm

The two functions that had to be implemented were getnewpopulation and getOffSpring. The get new population works by selecting the best percentage of the population (based on the nparentsratio variable). It will use these for the next generation, and replace all the other members by using these parents in the getnewpopulation function.

The getnewpopulation function looks at a random segment of scores from two random top-scoring parents (for instance, features 5 through 10) and compares these (as described in the Partially matched crossover (PMX) theory). If they have the same score, this score is also implemented in the offspring. All the other scores are random values of either 1 or 0. Finally, by taking into account the mutateprob variable, a certain percentage of the feature-scores are either inverted or not. The higher the mutation score, the more scores will be inverted.

The code of this exercise is on the github repository. `https://github.com/RUG-IDS/team-14`

1.2.4: Application: Face Recognition

The code for eigenfaces represents face images as a vector of pixel values. The principal components (PC) of the data are calculated such that linear combinations of pixel values form the axes in which the data varies most, i.e the principal components. The training data can then be reconstructed using only the most important PC's (whose corresponding eigenvalues are greatest and thereby explain the most variance). Clearly, as more principal components are used, reconstructed images use more pixel information and the loss of the images w.r.t the original images is reduced. Thus, the accuracy goes up with the number of principal components. The curve for this increasing relation goes up most steeply with the first PC's and stagnates relatively quickly.
The testing data in this exercise was projected onto the same principal components as derived from the training set which, given that we do not want to use information from the testset, is justified. Moreover, the mean subtracted from the testset was also calculated from the training set.

Using one-way ANOVA, the F-statistic is calculated and sorts the pixel values by how much they explain the variance in the data. The accuracy (according to nearest neighbour classification on the labels) is then calculated as a function of the number of sorted pixels used. Since the most important pixels are used first, a curve results that is similar to the one generated by using principal component method. Again, the accuracy increases quickly for the most explaining pixels and stagnates easily.

The genetic algorithm shows the fitness score instead of accuracy, given a certain chromosome. Here the fitness is based on the accuracy, but is also increased by using fewer pixels in the encoding. Plotting the fitness by chromosome shows that only few chromosomes are bad predictors of the images and, given the low variance over the chromosomes, seems to imply that a number of chromosomes share the same pixels.

The observed sharp increase in accuracy for the other methods would reflect in a few chromosomes with very good fitness (using the most important genes only) as well as a quick increase in accuracy for the first generations (discarding useless genes from parents with bad fitness). The first generation indeed has a lower fitness than any of the subsequent generations, although, this effect is much less pronounced given that a number of the pixels features seem to be shared between chromosomes.

The best chromosome reaches a final accuracy of 0.843. Instead, the principal component method as well as the method using one-way ANOVA, respectively, reach a top accuracy of 0.857 over the varying size feature sets. Indeed, the genetic algorithm is not guaranteed to find an optimal solution.

The code of this exercise is on the github repository. https://github.com/RUG-IDS/team-14