University of Groningen

Introduction to Data Science

Homework Assignment 5

Group 16:
Otte Tjepkema *(s3237184)*
José Rodrigues *(s4169328)*
Andrei Miculita *(s4161947)*
Robert Riesebos *(s3220672)*

October 8, 2019

Contents

1 Homework **2**
1.1 Exercises (Total 2 points) 2
1.1.1 Eigen values and Eigen vectors (1 point) 2
1.1.2 Matrix B 4
1.1.3 ANOVA (1 point) 6
1.2 Implementation (Total 8 points) 9
1.2.1 Implementation of the F-statistic (1 point) 9
1.2.2 Principal Component Analysis (2 points) 11
1.2.3 Genetic Algorithm (2 points) 11
1.2.4 Application: Face Recognition (3 points) 14

1 Homework

1.1 Exercises (Total 2 points)

1.1.1 Eigen values and Eigen vectors (1 point)

We will first determine the eigenvalues and eigenvectors of matrix A. To find the eigenvalues of the matrix we have to solve the equation

$$|A - \lambda I| = 0$$

filling in the values of A gives

$$\left| \begin{bmatrix} 4 & 1 \\ 2 & 6 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$
$$\left| \begin{bmatrix} 4-\lambda & 1 \\ 2 & 6-\lambda \end{bmatrix} \right| = (4-\lambda)(6-\lambda) - 2 * 1 = 0$$

writing out the polynomial gives the equation

$$\lambda^2 - 10\lambda + 22 = 0$$

this is an equation in the form $a\lambda^2 + b\lambda + c = 0$ which we can solve using the quadratic formula

$$\lambda_{1,2} = \frac{-b \pm \sqrt{b^2 - 4ac}}{a} \tag{1}$$

filling in our values $a = 1$, $b = -10$ and $c = 22$ gives

$$\lambda_1 = 5 + \sqrt{3}, \quad \lambda_2 = 5 - \sqrt{3}$$

Now that we have found our eigenvalues we can find the eigenvectors of A, the eigenvectors, by definition, have to satisify the following equation

$$A \cdot \boldsymbol{v}_i = \lambda_i \cdot \boldsymbol{v}_i$$

we start by finding the eigenvector $\boldsymbol{v}_1$, associated with the eigenvalue λ_1

$$A \cdot \boldsymbol{v}_1 = \lambda_1 \boldsymbol{v}_1$$
$$(A - \lambda_1 I) \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} 4-\lambda_1 & 1 \\ 2 & 6-\lambda_1 \end{bmatrix} \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} -1-\sqrt{3} & 1 \\ 2 & 1-\sqrt{3} \end{bmatrix} \cdot \boldsymbol{v}_1 = \begin{bmatrix} -1-\sqrt{3} & 1 \\ 2 & 1-\sqrt{3} \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$$

this is a system of equations that we can solve

$$\left[\begin{array}{cc|c} -1-\sqrt{3} & 1 & 0 \\ 2 & 1-\sqrt{3} & 0 \end{array}\right]$$

now we multiply row 1 by $1-\sqrt{3}$ and subtract it from row 2

$$\left[\begin{array}{cc|c} \left(-1-\sqrt{3}\right)\left(1-\sqrt{3}\right) & 1\left(1-\sqrt{3}\right) & 0 \\ 2 & 1-\sqrt{3} & 0 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 2 & 1-\sqrt{3} & 0 \\ 2 & 1-\sqrt{3} & 0 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 2 & 1-\sqrt{3} & 0 \\ 0 & 0 & 0 \end{array}\right]$$

from this we see that

$$2v_{1,1} + \left(1-\sqrt{3}\right)v_{1,2} = 0$$

$$v_{1,1} = \frac{\sqrt{3}-1}{2}v_{1,2}$$

any vector $\boldsymbol{v}_1$ that satisfies this equation will be an eigenvector of λ_1 therefore we will write $\boldsymbol{v}_1$ with an arbitrary constant c_1

$$\boldsymbol{v}_1 = c_1\begin{bmatrix} \frac{\sqrt{3}-1}{2} \\ 1 \end{bmatrix}$$

Now for the second eigenvector $\boldsymbol{v}_2$ associated with the eigenvalue λ_2 since this follows the exact same steps as before we will explain it more briefly

$$(A-\lambda_2 I)\cdot\boldsymbol{v}_2 = 0$$

$$\begin{bmatrix} -1+\sqrt{3} & 1 \\ 2 & 1+\sqrt{3} \end{bmatrix}\cdot\boldsymbol{v}_2 = \begin{bmatrix} -1+\sqrt{3} & 1 \\ 2 & 1+\sqrt{3} \end{bmatrix}\begin{bmatrix} v_{2,1} \\ v_{2,2} \end{bmatrix} = 0$$

again we set up and solve the system of equations

$$\left[\begin{array}{cc|c} -1+\sqrt{3} & 1 & 0 \\ 2 & 1+\sqrt{3} & 0 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 2 & 1+\sqrt{3} & 0 \\ 2 & 1+\sqrt{3} & 0 \end{array}\right]$$

$$\left[\begin{array}{cc|c} 2 & 1+\sqrt{3} & 0 \\ 0 & 0 & 0 \end{array}\right]$$

which gives

$$v_{2,1} = \frac{-1-\sqrt{3}}{2} v_{2,2}$$

again any vector $\boldsymbol{v}_2$ that satisfies this equation is an eigenvector, we write $\boldsymbol{v}_2$ with arbitrary constant c_2

$$\boldsymbol{v}_1 = c_2 \begin{bmatrix} \frac{-1-\sqrt{3}}{2} \\ 1 \end{bmatrix}$$

1.1.2 Matrix B

Now that we have calculated the eigenvalues and eigenvectors for A we can use the same methods to calculate them for matrix B

$$|B - \lambda I| = 0$$

filling in the values of B gives

$$\left| \begin{bmatrix} 3 & 5 \\ 8 & 3 \end{bmatrix} - \begin{bmatrix} \lambda & 0 \\ 0 & \lambda \end{bmatrix} \right| = 0$$

$$\left| \begin{bmatrix} 3-\lambda & 5 \\ 8 & 3-\lambda \end{bmatrix} \right| = (3-\lambda)(3-\lambda) - 5 * 8 = 0$$

writing out the polynomial gives the equation

$$\lambda^2 - 6\lambda - 31 = 0$$

this is an equation in the form $a\lambda^2 + b\lambda + c = 0$ which we can solve using equation (1), filling in our values $a = 1$, $b = -10$ and $c = 22$ gives

$$\lambda_1 = 3 + 2\sqrt{10}, \quad \lambda_2 = 3 - 2\sqrt{10}$$

Now that we have found our eigenvalues we can find the eigenvectors of B, the eigenvectors, by definition, have to satisify the following equation

$$B \cdot \boldsymbol{v}_i = \lambda_i \cdot \boldsymbol{v}_i$$

we start by finding the eigenvector $\boldsymbol{v}_1$, associated with the eigenvalue λ_1, since these are again the same steps as with matrix A we will explain the

steps briefly

$$(B - \lambda_1 I) \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} 3-\lambda_1 & 5 \\ 8 & 3-\lambda_1 \end{bmatrix} \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} -2\sqrt{10} & 5 \\ 8 & -2\sqrt{10} \end{bmatrix} \cdot \boldsymbol{v}_1 = \begin{bmatrix} -2\sqrt{10} & 5 \\ 8 & -2\sqrt{10} \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$$

this is a system of equations that we can solve

$$\left[\begin{array}{cc|c} -2\sqrt{10} & 5 & 0 \\ 8 & -2\sqrt{10} & 0 \end{array}\right]$$

now we multiply row 1 by $\frac{-2\sqrt{10}}{5}$ and subtract it from row 2

$$\left[\begin{array}{cc|c} -2\sqrt{10} & 5\frac{-2\sqrt{10}}{5} & 0 \\ 8 & -2\sqrt{10} & 0 \end{array}\right]$$
$$\left[\begin{array}{cc|c} 8 & -2\sqrt{10} & 0 \\ 8 & -2\sqrt{10} & 0 \end{array}\right]$$
$$\left[\begin{array}{cc|c} 8 & -2\sqrt{10} & 0 \\ 0 & 0 & 0 \end{array}\right]$$

from this we see that

$$8v_{1,1} - 2\sqrt{10}v_{1,2} = 0$$
$$v_{1,1} = \frac{2\sqrt{10}}{8}v_{1,2} = \frac{\sqrt{10}}{4}v_{1,2}$$

any vector $\boldsymbol{v}_1$ that satisfies this equation will be an eigenvector of λ_1 therefore we will write $\boldsymbol{v}_1$ with an arbitrary constant c_1

$$\boldsymbol{v}_1 = c_1 \begin{bmatrix} \frac{\sqrt{10}}{4} \\ 1 \end{bmatrix}$$

Now for the second eigenvector $\boldsymbol{v}_2$ associated with the eigenvalue λ_2 since this follows the exact same steps as before we will again explain it briefly

$$(B - \lambda_2 I) \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} 3-\lambda_2 & 5 \\ 8 & 3-\lambda_2 \end{bmatrix} \cdot \boldsymbol{v}_1 = 0$$
$$\begin{bmatrix} 2\sqrt{10} & 5 \\ 8 & 2\sqrt{10} \end{bmatrix} \cdot \boldsymbol{v}_1 = \begin{bmatrix} 2\sqrt{10} & 5 \\ 8 & 2\sqrt{10} \end{bmatrix} \begin{bmatrix} v_{1,1} \\ v_{1,2} \end{bmatrix} = 0$$

again we set up and solve the system of equations

$$\left[\begin{array}{cc|c} 2\sqrt{10} & 5 & 0 \\ 8 & 2\sqrt{10} & 0 \end{array}\right]$$
$$\left[\begin{array}{cc|c} 8 & 2\sqrt{10} & 0 \\ 8 & 2\sqrt{10} & 0 \end{array}\right]$$
$$\left[\begin{array}{cc|c} 8 & 2\sqrt{10} & 0 \\ 0 & 0 & 0 \end{array}\right]$$

which gives

$$v_{2,1} = -\frac{\sqrt{10}}{4} v_{2,2}$$

again any vector $\boldsymbol{v}_2$ that satisfies this equation is an eigenvector, we write $\boldsymbol{v}_2$ with arbitrary constant c_2

$$\boldsymbol{v}_1 = c_2 \begin{bmatrix} -\frac{\sqrt{10}}{4} \\ 1 \end{bmatrix}$$

1.1.3 ANOVA (1 point)

The first step to performing the anova analysis is to fill in the following table

Attendance	High	Med	Low
Sample size			
Mean			

We first fill in the sample size, this is simply the number of entries for each group

Attendance	High	Med	Low
Sample size	8	6	5
Mean			

next we calculate the means of the groups with the equation

$$\bar{x} = \frac{\sum_{i=1}^{n} x}{n} \tag{2}$$

where n is the sample size

$$\bar{x}_{High} = \frac{10 + 7.5 + 6.5 + 8 + 8.5 + 7.5 + 7.5 + 7 + 9}{8} = 8$$
$$\bar{x}_{Med} = \frac{7 + 6.5 + 5.5 + 7.5 + 6 + 6.5}{6} = 6.5$$
$$\bar{x}_{Low} = \frac{6 + 6 + 8 + 7 + 5.5}{5} = 6.5$$

Attendance	High	Med	Low
Sample size	8	6	5
Mean	8	6.5	6.5

Table 1: Table showing the sample size and mean for each group

Next we fill in the ANOVA table. We start out with the degrees of freedom (df). The degrees of freedom between groups, df(B), is the number of groups minus one. The degrees of freedom within groups, df(W), is the total sample size minus the number of groups. The total degrees of freedom is simply the sum of df(B) and df(W). In our case

$$df(B) = No_{groups} - 1 = 3 - 1 = 2$$
$$df(W) = No_{samples} - No_{groups} = (8 + 6 + 5) - 3 = 16$$
$$df(Total) = df(B) + df(W) = 2 + 16 = 18$$

Now that we have found all the values we can fill them in, in the ANOVA table

Source	SS	df	MS	F
Between		2		
Within		16		NA
Total		18	NA	NA

Next we compute the variance between groups SS(B), which is given by the following equation

$$SS(B) = \sum_{i=1}^{k} n_i \left(\bar{x}_i - \bar{\bar{x}}\right)^2 \tag{3}$$

where k is the number of groups, n_i is the sample size for that group, $\bar{x}_i$ is the mean of that group and $\bar{\bar{x}}$ is the weighted mean of the groups. We will

first calculate $\bar{\bar{x}}$ for our dataset, using the data from table 1

$$\bar{\bar{x}} = \frac{8 * 8 + 6 * 6.5 + 5 * 6.5}{8 + 6 + 5} = 7.13$$

with this we can fill in all the values for equation (3)

$$SS(B) = 8\,(8 - 7.13)^2 + 6\,(6.5 - 7.13)^2 + 5\,(6.5 - 7.13)^2 = 10.4$$

Next we calculate the within group sum of squares, we do this using the following formula

$$SS(W) = \sum_{j=1}^{k} \sum_{i=1}^{n} (x_{ij} - \bar{x_j})^2 \tag{4}$$

where n_j is the sample size of the group and $\bar{x_j}$ is the mean of that group, we can now fill in the values that we have

$$\begin{aligned} SS(W) = &\big[(10 - 8)^2 + (7.5 - 8)^2 + (6.5 - 8)^2 + (8 - 8)^2 + (8.5 - 8)^2 \\ &+ (7.5 - 8)^2 + (7 - 8)^2 + (9 - 8)^2\big] \\ &+ \big[(7 - 6.5)^2 + (6.5 - 6.5)^2 + (5.5 - 6.5)^2 + (7.5 - 6.5)^2 + (6 - 6.5)^2 + (6.5 - 6.5)^2\big] \\ &+ \big[(6 - 6.5)^2 + (6 - 6.5)^2 + (8 - 6.5)^2 + (7 - 6.5)^2 + (5.5 - 6.5)^2\big] \\ &= 15.5 \end{aligned}$$

The total sum of squares is simply SS(B) + SS(W), with this we can fill in the table further

Source	SS	df	MS	F
Between	10.4	2		
Within	15.5	16		NA
Total	25.9	18	NA	NA

next we calculate the mean of squares (MS) which is simply the sum of squares(SS) divided by the degrees of freedom (f) for each source

Source	SS	df	MS	F
Between	10.4	2	5.2	
Within	15.5	16	0.97	NA
Total	25.9	18	NA	NA

finally the F value is simply the means of squares between MS(between) divided by the mean of squares within MS(within)

Source	SS	df	MS	F
Between	10.4	2	5.2	5.38
Within	15.5	16	0.97	NA
Total	25.9	18	NA	NA

1.2 Implementation (Total 8 points)

1.2.1 Implementation of the F-statistic (1 point)

To perform ANOVA on the dataset we use a similar method to the manual calculation above. First we convert the two vectors containing the individual values and dependant variables into a single dataset containing seperate lists of all the values for each dependent variable. Then we create a table with the group name, sample size, mean, and variance (multiplied by sample size) of all the groups. This contains all the data that we need for the calculation. Next we calculate $\bar{\bar{x}}$ using weighted means. Then we calculate the sum of squares between groups by using eqation (3), which we then correct for the degrees of freedom. Finally we calculate the sum of squares within groups by summing all the variances and dividing it by the total sample size minus the number of groups. Then F is simply SS(B)/SS(W). The code is given below.

```
function F = myOneWayANOVA(IV,DV)
    %We leave in the values for IV and DV for testing purposes
    %IV =
    ↪  [10,7.5,6.5,8,8.5,7.5,7,9,7,6.5,5.5,7.5,6,6.5,6,6,8,7,5.5];
    %DV =
    ↪  ["H","H","H","H","H","H","H","H","M","M","M","M","M","M","L","L","L","L","L"];
    %Setup initial values
    ind_DV = unique(DV); %Find unique dependent variables
    no_DV = length(ind_DV); %Find the number of dependent variables
    if no_DV <2 %Throw error if only one unique DV value was given
        error("DV only has one unique value")
    end
    %Convert the two lists into a cell array which contains the
    ↪  values for each
    %dependent variable in a seperate vector
    dataset = transpose([IV; DV']);
    groups = cell(1,no_DV);
    for i = 1:length(ind_DV)
        groups{i} = str2double(dataset(dataset(:,2)==ind_DV(i)));
    end
```

```
    %Create table with necessary data (Sample size, Mean, St.Dev,
    ↪  Variance)
    stat_sum = cell(4,no_DV+1);
    %Start with naming the first column
    stat_sum{1,1} = "Group";
    stat_sum{2,1} = "Sample size";
    stat_sum{3,1} = "Mean";
    stat_sum{4,1} = "Variance";
    %Next fill in the values
    for i = 1:no_DV
        stat_sum{1,i+1} = ind_DV(i); %Groups
        stat_sum{2,i+1} = length(groups{i}); %Sample size
        stat_sum{3,i+1} = mean(groups{i}); %Mean
        stat_sum{4,i+1} = var(groups{i},1)*(stat_sum{2,i+1});
        ↪  %Variance
    end
    %Next we calculate the variance between groups SS(B)
    %First calculate x double bar
    xbb = 0;
    for i = 1:no_DV
        xbb = xbb+stat_sum{2,i+1}*stat_sum{3,i+1};  %Summation
        if i == no_DV
            xbb = xbb/sum(cell2mat(stat_sum(2,2:no_DV+1)));
            ↪  %Normalize
        end
    end
    %With this we can calculate SS(B)
    SS_B = 0;
    for i = 1:no_DV
        SS_B = SS_B+stat_sum{2,i+1}*(stat_sum{3,i+1}-xbb)^2;
        ↪  %Summation
        if i == no_DV
            SS_B = SS_B/(no_DV-1); %Divide by degrees of freedom
        end
    end
    %Next we find the mean of sum of squares SS(W), we use the
    ↪  formula from
    %Lecture 2 slide 48/75
    SS_W =
    ↪  sum(cell2mat(stat_sum(4,2:no_DV+1)))/(sum(cell2mat(stat_sum(2,2:no_DV+1)))-no_DV);
    %Finally we find F
    F = SS_B/SS_W;
end
```

1.2.2 Principal Component Analysis (2 points)

Note: in this section eigenvectors are synonymous to principal components.

First the data is centered and normalized. Then, in order to implement PCA in a scalable manner, we use A^TA instead of AA^T as the covariance matrix. Next the eigenvectors (pc) and eigenvalues are computed using the `eig` function. After finding the eigenvectors and eigenvalues, we use the relation between the eigenvectors of AA^T and A^TA to transform the eigenvectors back to the correct values. Finally the eigenvectors are scaled using the `normc` function (which uses L2-normalization) and the output variables are sorted in descending order. The code for the `myPCA` function is given below.

```
function [pc, eigenvalues] = myPCA(A)
    % Center and normalize data
    A = A - mean(A);
    A = normalize(A);

    % Compute pc and eigenvalues
    covariance_matrix = A' * A;
    [pc, eigenvalues] = eig(covariance_matrix);
    eigenvalues = diag(eigenvalues);

    % The eigenvectors of AA^T and A^TA are related as follows: u_i
    ↪  = Av_i
    pc = A * pc;

    % L2-normalization
    pc = normc(pc);

    % Sort pc and eigenvalues
    [~, srtidx] = sort(eigenvalues, 'descend');
    eigenvalues = eigenvalues(srtidx);
    pc = pc(:, srtidx);
end
```

1.2.3 Genetic Algorithm (2 points)

The genetic algorithm finds the best subset of features by repeatedly selecting the best performing chromosomes and combining them. An initial population is generated randomly. Then, each epoch, the `nparentsratio` of chromosomes which perform best in a 2-fold cross validation are selected.

These are then randomly paired and combined until a new population is obtained.

Below the implementation of `getnewpopulation` is given. We use the `sort` function to sort the passed `score` vector in descending order. The `sort` function returns the original indices of the, now sorted, `score` vector. Then we calculate how many parents have to be selected based on `nparentsratio`. We select the obtained amount from the `indices` vector and select the parents with the corresponding indices from the population.
Next we generate a new population using the previously selected, best parents. First we add the parents to the new population, then we generate and add offspring to the population until it is the same size as the original population. In the loop we make sure that parents don't reproduce with themselves.

```
function newpop =
↪  getnewpopulation(pop,score,nparentsratio,mutateprob)
    % Generate a new population by first selecting the best
↪  performing chromosomes from the given pop matrix, and
↪  subsequently generate new offspring chromosomes from randomly
↪  selected pairs of parent chromosomes.

    % Step 1. Write code to select the top performing chromosomes.
    ↪  Use nparentsratio to calculate how many parents you need. If
    ↪  pop has 100 rows and nparentsration is 0.2, then you have to
    ↪  select the top performing 20 chromosomes
    nr_of_chromosomes = size(pop, 1);
    nr_of_features = size(pop, 2);

    [~, indices] = sort(score, 'descend');
    nr_of_parents = ceil(nparentsratio * nr_of_chromosomes);
    selected_indices = indices(1:nr_of_parents);
    parents = pop(selected_indices, :);

    % Step 2. Iterate until a new population of the same size is
    ↪  generated.
    % In each iteration create a new offspring chromosome from two
    ↪  randomly selected  parent chromosomes. Use the function
    ↪  getOffSpring to generate a new offspring.
    new_population = zeros(nr_of_chromosomes, nr_of_features);
    new_population(1:nr_of_parents, :) = parents;

    for i = nr_of_parents:nr_of_chromosomes
       index_1 = randi([1 nr_of_parents], 1, 1);
```

```
        index_2 = randi([1 nr_of_parents], 1, 1);
        while (index_1 == index_2)
            index_2 = randi([1 nr_of_parents], 1, 1);
        end

        parent_1 = parents(index_1, :);
        parent_2 = parents(index_2, :);
        new_population(i, :) = getOffSpring(parent_1, parent_2,
        ↪  mutateprob);
    end

    newpop = new_population;
end
```

The crossover of each pair of chromosomes is done as demonstrated in the lecture, by randomly selecting a lower bound and an upper bound, and then copying the bits outside these bounds from one chromosome, and the ones inside these bounds from the other (2-point crossover). The code for this implementation is below. An alternative implementation (present in the code as a comment) would be randomly choosing for each bit which parent it will be taken from (uniform crossover with equal probability). Afterwards random mutations (bit flips) are applied with a chance at each bit of the offspring. It is important to note that offspring can't be an exact duplicate of it's parent because of the way the `lowerBound` (First crossover point) and `upperBound` (second crossover point) are chosen. Furthermore when `upperBound` equals `length(parent1)` then `parent1(upperBound+1:end)` results in an index out of bounds exception. This is not a problem since Matlab returns an empty vector in this case, effectively resulting in a single-point crossover.

```
function offspring = getOffSpring(parent1,parent2,mutateprob)
    % Generate an offspring from parent1 and parent2 and mutate the
↪   bits by using the probability mutateprob.

    %offspring = [];
    % Step 1. Write code that generates one offspring from the given
    ↪   two parents

    % This is done the same as what was shown in the lecture - using
    ↪   a random  interval, in this case we can often get
    ↪   consecutive bits from the same parent which may actually be
    ↪   desired. Another implementation would be randomly choosing a
    ↪   bit from parent 1 or 2 with a probability.
    lowerBound = randi(length(parent1) - 1);
```

```
    upperBound = randi([lowerBound + 1 length(parent1)], 1, 1);

    offspring = [parent1(1:lowerBound) parent2(lowerBound +
    ↪  1:upperBound) parent1(upperBound + 1:end)];

    % for i = 1:length(parent1)
    %     x = rand;
    %     if x < 0.5
    %         offspring(i) = parent1(i);
    %     else
    %         offspring(i) = parent2(i);
    %     end
    % end

    % Step 2. Write code to mutate some bits with given mutation
    ↪  probability mutateprob

    for i = 1:length(offspring)
        x = rand;
        if x < mutateprob
            offspring(i) = 1 - offspring(i);
        end
    end
end
```

1.2.4 Application: Face Recognition (3 points)

First we will give the code that was added to the myEigenFaces.m script. To find the first K principal components we can simply select them from the pc (principal component) variable. Then to project the the training data onto the first K principal components we simply transpose the principal components and multiply them with the training data. For the test data we use the same method however the test data still needs to be centered, therefore we subtract the mean training face from it before multiplying it by the principal components.

```
for K = KList
    % Project training data onto the first K principal components
    trainingVectors = pc(:, 1:K)' * trainingData;

    % Subtract the mean training face from the test data and project
    ↪  the
    % resulting matrix onto the first K prinicipal components
    testingVectors = pc(:, 1:K)' * (testingData - meanTrainingFace);
```

```
    % Compute accuracy
    accuracy(idx) = classify(testingVectors,trainingVectors,
    trainingFilenameList,testingFilenameList);

    idx = idx + 1;
end
```

To compare the different methods for face recognition we will plot the accuracy of the different methods below.

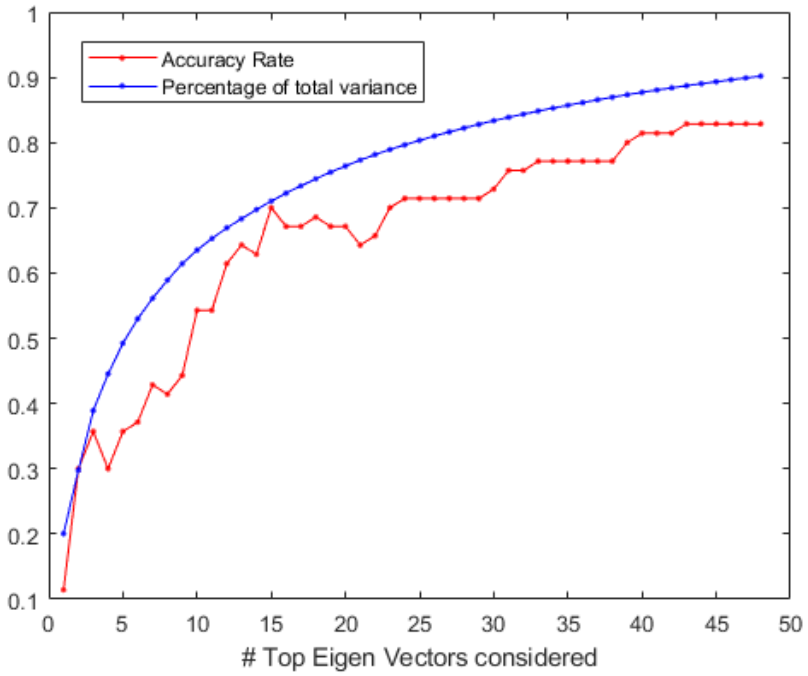

Figure 1: Principal component analysis

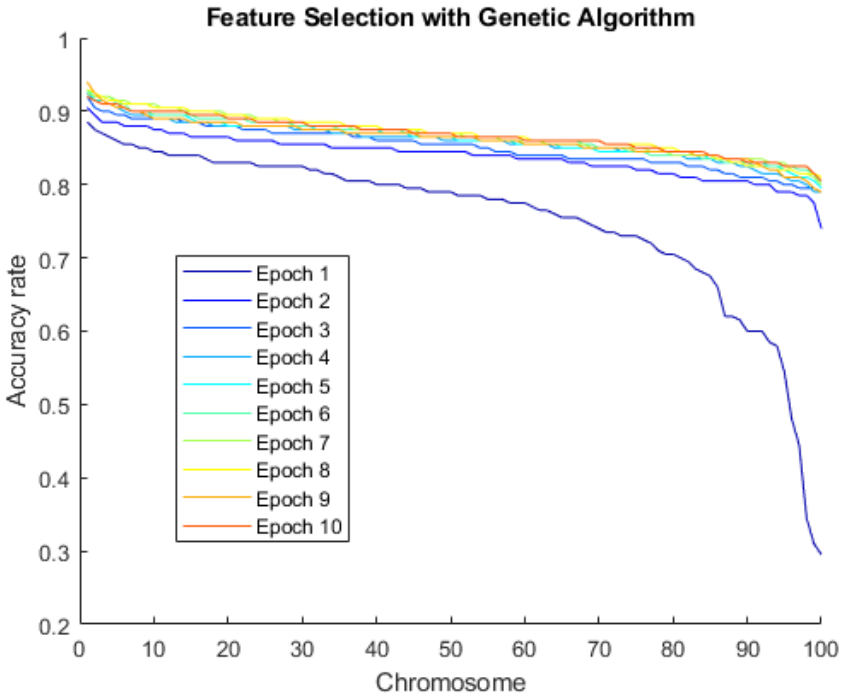

Figure 2: Feature selection with Genetic Algorithms

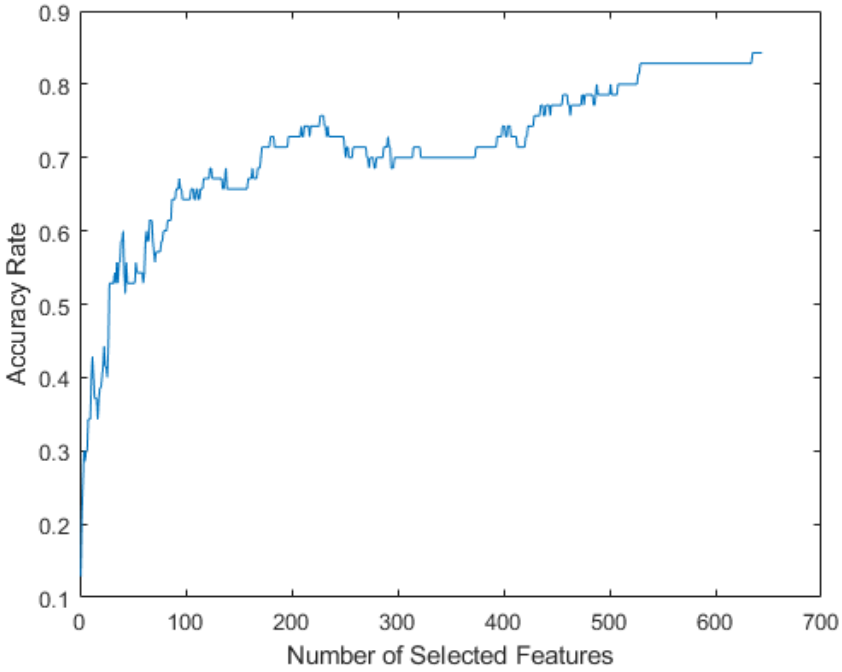

Figure 3: Feature selection with ANOVA

When it comes to absolute accuracy, the best performer out of the 3 methods was the Genetic algorithm (GA). These kind of algorithms often face the issue of not being able to find optimal parameters, but in this particular problem, the GA was particularly effective and achieved a near optimal solution relatively fast. The only drawback of this approach was time consumption (the ANOVA and PCA methods were faster), despite that, it pays off to use this approach given the additional 6-7% improvement in accuracy, at expense of 3 additional seconds.

Despite having worse results (in this case) than the GA algorithm (in terms of maximum accuracy), the ANOVA test is suitable to this and numerous kinds of problems since it's a robust method that's really applicable and has a straightforward analysis.

Like the ANOVA, the PCA-based method for facial recognition is quite straightforward and quick. Recognition and classification is made by projecting a new image in the "eigenface" subspace. This approach grants simplicity and robustness in the way that changes in angle or variable light gradients don't affect the recognition process too much. We can observe that a change in the number of eigenvectors considered doesn't cause as many oscillations in the accuracy rate as the change of nº of selected features for ANOVA, establishing PCA as a more stable method in this problem.

All in all, GA is definitely the best method if computational cost is not an issue (for this algorithm to be close to optimal it takes some iterations). However, if one wants simplicity and doesn't require too much precision, ANOVA and PCA methods are suitable as well.