SCC-252 – Visualização Computacional
Aula de Laboratório 3:

Implementação do PCA e kPCA

Moussa Reda Mansour e
Maria Cristina Ferreira de Oliveira

Antes de Começar

Fazer download da página da disciplina: arquivo kitLab.zip (link disciplina)

Neste arquivo você irá enconrar:

Pasta data: contêm os dados que usaremos nas simulações.
Sphere.dat:
Dados não lineares, dimensão 798x4 sendo a última coluna a classe a qual cada amostra pertence.
wines.dat:
Dados de vinhos, dimensão 178x14, sendo a primeira coluna a classe a qual cada amostra percente.

Pasta src: contêm um código fonte para leitura dos dados
DataReader.pde
Leitor de Dados

Pasta lib: contêm as bibliotecas necessárias para a implementação das técnicas PCA e kPCA.

Biblioteca papaya (http://adilapapaya.com/papayastatistics/)
Operações em matrizes
Operações estatítiscas
Cálculo de autovalores

Biblioteca graphica (https://github.com/jagracar/grafica)
Geração de gráficos em 2D.

Para instalar as bibliotecas:

Copiar a pasta referente a **cada** biblioteca na pasta PROCESSING_PATH\modes\java\libraries

Implementação do PCA

Criar no processing um scketch chamado **AppPCA**, e anexar a ele o arquivo DataReader e os arquivos de dados (sphere.dat e wines.dat).

Abrindo e analisando os dados:

```
import papaya.*;

String [] textlines;

DataReader data; //leitor de dados

float X[][]; //matriz de dados
int   L[];//matriz de classes
```

```
void setup()
{
  data = new DataReader("wines.dat", ",", 0);//nome do arquivo, separador e índice do
                                              //label
  X = data.getData();
  L = data.getLabels();

  Mat.print(X,2);//mostre a matriz de dados com duas casas decimas apos o vírgula
  Mat.print(L,1);//mostrar os labels
}
```

Iniciar a implementação da classe PCA:

```
String [] textlines;

DataReader data;

PCA        pca;//classe do PCA a ser implementado ainda

float X[][];
int   L[];

void setup()
{
  //PCA part
  data = new DataReader("vinhos.dat", ",", 0);

  X = data.getData();
  L = data.getLabels();

  pca = new PCA(X, false);//recebe os dados (matriz X) e um parâmetro indicando se
                          //vai ser necessária a normalização dos dados
  float Y[][] = pca.solve();//calcula as componentes principais e retorna a projeção
}
```

Classe PCA:

```
//Importar a biblioteca papaya via sketch->add library.
import papaya.*;

class PCA
{
   float[][] X;//matriz de dados
   float[][] D;//matriz com diagonal composta de autovalores
   float[][] V;//matriz dos autovetores correspondentes aos autovalores em D

   PCA(float[][] X, boolean normalize)
   {

       this.X = X;

       //Exercício 1: elaborar um algoritmo para normalizar os dados,
      //caso normalize seja true, caso contrário, manter os dados como estão.

   }
```

Para o PCA vamos precisar implementar alguns métodos privados para auxiliar na implementação:

```
   /*****************************************************************/
   /* Auxiliar Methods                                              */
   /*****************************************************************/
   //Dada uma matriz X e a posição de k de uma coluna, este método retorna o vetor
   //referente a coluna k da matriz X
   private float[] getArray(float X[][], int k)
   {
       float array[] = new float[X.length];
```

```
    for(int i=0; i < X.length; i++)
        array[i] = X[i][k];

    return array;
}

//Dado um número de linhas e colunas, este método retorna uma matriz n x m de 1's.
private float[][] ones(int n, int m)
{
    float mat[][] = new float[n][m];

    for(int i=0; i<n; i++)
      for(int j=0; j<m; j++)
        mat[i][j] = 1.0;

    return mat;
}

//Dado um vetor v, este método cria uma matriz mat cuja diagonal principal é o
//próprio vetor v.
private float[][] matdiag(float[] v)
{
  float mat[][] = new float[v.length][v.length];

  for(int i=0; i<v.length; i++)
    for(int j=0; j<v.length; j++)
      if( i == j )
        mat[i][j] = v[i];
      else
        mat[i][j] = 0;

  return mat;
}

//Dado uma matriz, este método retorna a diagonal principal da mesma.
private float[] diag(float mat[][])
{
    float D[] = new float[mat.length];

    for(int i=0; i<mat.length; i++)
      D[i] = mat[i][i];

    return D;
}
```

Centrando os dados:

$$X_c = X - \bar{X}$$

Que é a mesma coisa que:

$$X_c = X - O\bar{D}$$

Sendo O matriz de 1's e $\bar{D}$ matriz com diagonal igual a $\bar{X}$.

```
//Centra os dados da matrix X, retornando a matriz centrada.
private float[][] centralize()
{
    float matMean[][];//matriz média, cuja diagonal é a própria média dos dados
    float O[][];//matriz de 1's
    float Xc[][];//matriz centrada
    float mean[] = new float[X[0].length];//vetor média dos dados

    //calculando a média para cada coluna de variáveis i, para tal usamos a função
    //getArray.
    for(int i=0; i<X[0].length; i++)
```

```
        mean[i]=Descriptive.mean(getArray(X,i));

    //obtemos a matriz com diagonal média
    matMean = matdiag(mean);

    //obtemos a matriz de 1's
    O = ones(X.length, matMean.length);

    //calculamos a matriz centrada via Xc = X - O*D
    Xc = Mat.subtract(X,Mat.multiply(O, matMean));


    //verificando se a matriz está centrada. A matriz é dita centrada
    //se ela possuir média nula. No nosso caso, média muito próxima a
    //zero
    println("Mean:");
    for(int i=0; i<X[0].length; i++)
      print(" "+Descriptive.mean(getArray(Xc,i)));
    println();

    return Xc;
}

public float[][]  solve()
{
    float Xc[][] = centralize();

    return null;
}
```

Executar o **AppPCA** para verificar o resultado, **CUIDADO** com o retorno do **solve**, pode dar **null exception**.

Agora vamos implementar o **solve** do PCA. O método **solve** centra os dados em **X**, calcula de matriz de covariância e os autovalores e autovetores da matriz de covariância. A projeção é dada pela equação:

$$Y = X_c V$$

Sendo V os autovetores (componentes principais) e Y a projeção. Não considerei o **k** ainda (fica como **exercício**).

```
public float[][]  solve()
{
    float Xc[][] = centralize();
    //calculando a matriz de covarância de Xc, lembrar do porque do TRUE
    float Sx[][] =  Correlation.cov(Xc, true);

    //Calculando os autovalores e autovetores de Sx
    Eigenvalue eigen = new Eigenvalue(Sx);
    V = Cast.doubleToFloat(eigen.getV());
    D = Cast.doubleToFloat(eigen.getD());

    //Analisando os autovetores de Sx, para verificar a variância
    for(int i=0; i < D.length;i++)
      println(D[i][i]);

    //Calculando a projeção usando Y = Xc*V
    return Mat.multiply(Xc, V);
}
```

A classe PCA foi implementada, agora vamos terminar de implementar a **AppPCA**. Mas primeiro, vamos verificar se a classe **PCA** está retornando algum valor em **Y**.

```
String [] textlines;

DataReader data;
```

```
PCA        pca;
float X[][];
int   L[];

void setup()
{
  //PCA part
  data = new DataReader("wines.dat", ",", 0);

  X = data.getData();
  L = data.getLabels();

  pca = new PCA(X, true);
  float Y[][] = pca.solve();

  Mat.print(Y,2);
}
```

Implementando a interface gráfica

```
import grafica.*;//inserir a biblioteca gráfica

String [] textlines;

DataReader data;
PCA        pca;
float X[][];
int   L[];

int nPoints;//número de pontos a serem plotados
GPointsArray points;//array de pontos
GPlot plot;//classe para plotar os dados

void setup()
{
  //PCA part
  data = new DataReader("wines.dat", ",", 0);

  X = data.getData();
  L = data.getLabels();

  pca = new PCA(X, true);
  float Y[][] = pca.solve();

  //Plotting part
  nPoints = Y.length;
  size(500, 350);
  points = new GPointsArray(nPoints);

  //pegando os 2 primeiros componentes em Y.
  for (int i = 0; i < nPoints; i++) {
      points.add(Y[i][0],Y[i][1]);
  }

  //Cria um novo plot, cuja posição inicia na coordenada (25,25)
  //do layout principal
  plot = new GPlot(this);
  plot.setPos(25, 25);

  //Setando o título do gráfico e dos eixos
  plot.setTitleText("Principal Component Analysis");
  plot.getXAxis().setAxisLabelText("PC-1");
  plot.getYAxis().setAxisLabelText("PC-2");

  //Adicionando os pontos
  plot.setPoints(points);
```

```
}

void draw() {
  background(150);

  plot.beginDraw();
    plot.drawBackground();
    plot.drawBox();
    plot.drawXAxis();
    plot.drawYAxis();
    plot.drawTopAxis();
    plot.drawRightAxis();
    plot.drawTitle();

    //Quando L é igual a 1, significa que aquele ponto pertence a classe 1
    //Quando L é igual a 2, significa que aquele ponto pertence a classe 2
    //Quando L é igual a 3, significa que aquele ponto pertence a classe 3
    //Logo, os trechos a seguir indicam uma cor para cada ponto, de acordo com
    //a classe de cada u,.
    for (int i = 0; i < nPoints; i++ )
    {
     if( L[i] == 1)
                        //ponto, cor (R,G,B) e tamanho do ponto
          plot.drawPoint(points.get(i),color(255, 0, 0), 5);
      else
      if( L[i] == 2)
          plot.drawPoint(points.get(i),color(0, 255, 0), 5);
       else
          plot.drawPoint(points.get(i),color(0, 0, 255), 5);

  }
  plot.endDraw();
}
```

Após projetar os pontos, verificaremos que o gráfico ficou bagunçada (Figura 1).

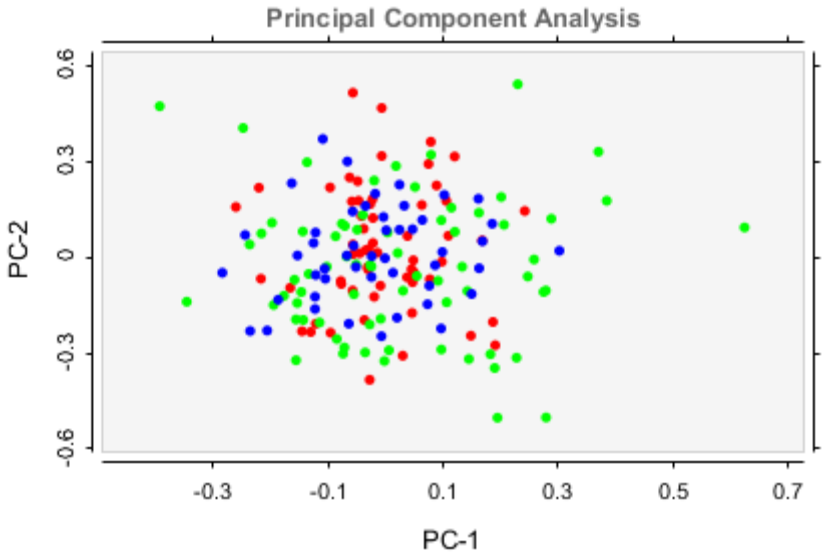

Figura 1 - Projeção do PCA pegando os k autovetores, sem considerar a ordenação dos autovalores.

Isso ocorre pelo fato do método de cálculo de autovalores do **papaya** não ordenar adequadamente os autovalores. Para resolver este problema, basta olharmos os autovalores que verificaremos que os mesmo estão ordenados de forma crescente. Então vamos substituir a linha

```
      points.add(Y[i][0],Y[i][1]);
```

por

```
      points.add(Y[i][Y[i].length-1],Y[i][Y[i].length-2]);
```

pois os autovalores com a maior variância são os dois últimos. A figura 2 ilustra a projeção obtida com esta modificação.

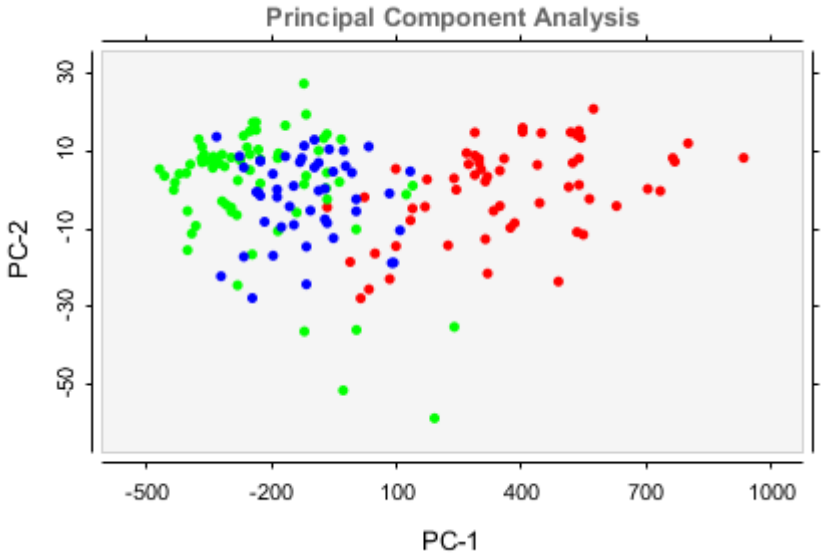

Figura 2 - Projeção do PCA pegando os k autovetores, considerando a ordenação dos autovalores.

Exercício: implemente a normalização dos dados, projete os dados e compare com o resutlado obtido sem a normalização.

Implementação do kPCA

Vamos aproveitar bastante coisa do que fizemos até agora para implementar o kPCA. Então vamos criar no processing o projeto **AppkPCA**, e anexar a ele o arquivo **DataReader** e os arquivos de dados (spheres.dat e wines.dat).

Abrindo e analisando os dados:

```
String [] textlines;

DataReader data;
float X[][];
int   L[];

void setup()
{
  data = new DataReader("spheres.dat", ",", 3);

  X = data.getData();
  L = data.getLabels();

  Mat.print(X,2);//mostre a matriz de dados com duas casas decimais após o vírgula
  Mat.print(L,1);//mostrar os labels
}
```

Iniciar a implementação da classe kPCA:

```
String [] textlines;

DataReader data;
kPCA       kpca;
float X[][];
int   L[];

void setup()
```

```
{
  //kPCA part
  data = new DataReader("spheres.dat", ",", 3);

  X = data.getData();
  L = data.getLabels();

  kpca = new kPCA(X);
  float Y[][] = kpca.solve(0, 20);//passamos como parâmetros do tipo de kernel a ser
                                  //aplicado e o parâmetro deste kernel
}
```

Classe kPCA:

```
//Importar a biblioteca papaya via sketch->add library.

import papaya.*;

class kPCA
{
   float[][] X;//matriz de dados
   float[][] D;//matriz com diagonal composta pelos autovalores
   float[][] V;//matriz de autovetores correspondents a D

   kPCA(float[][] X)
   {
       this.X = X;
   }
```

Calculando a matriz **K**, para tal criamos um método privado que recebe como parâmetros o tipo de kernel que vai ser usado e o parâmetro do mesmo.

```
   private float[][] kernelMatrix(int type, float parameter)
   {
       //vamos aplicar apenas o kernel gaussiano
       if(type == 0)//gaussian kernel
       {
           return gaussianMatrix(parameter);//recebe como parâmetro o sigma e retorna a
                                            //matriz K calculada.
       }

       return null;
   }
```

Calculando o kernel gaussiano:

$$K(x,y) = e^{-|x-y|^2/sigma^2}$$

```
   //este método recebe como parâmetro o sigma e retorna a matriz K.
   private float[][] gaussianMatrix( float sigma )
   {
       float K[][];

       K = new float[X.length][];
       for( int i = 0; i < X.length; i++ )
           K[i] = new float[X.length];

       float aux, sub[];
       for (int i = 0; i < X.length; i++)
       {
         for (int j = 0; j < X.length; j++)
         {
             sub = Mat.subtract(X[i], X[j]);//esta parte é o |x-y|
             aux = Mat.sum(Mat.multiply(sub,sub));//esta parte  é o |x-y|^2
             K[i][j] = exp(-aux/pow(sigma,2));//esta é a parte e^-|x-y|^2/sigma^2
         }
       }
       return K;
```

```
}
```

Para o kPCA vamos precisar implementar alguns métodos privados auxiliares, que são os mesmos que foram implementados no PCA:

```
private float[] getArray(float X[][], int k);
private float[][] ones(int n, int m);
private float[][] Matdiag(float[] v);
private float[] diag(float mat[][]);
```

Para centrar os dados, vamos precisar da seguinte expressão:

$$K_c = K - OK - KO + OKO$$

Que é a mesma coisa que:

$$A = K - OK$$
$$B = KO$$
$$C = OKO$$
$$D = A - B$$
$$K_c = D + C$$

Sendo O matriz de 1's.

```
//centrando a matriz K, recebe K como parâmetro e retorna a matriz centrada Kc.
private float[][] centralize(float K[][])
{
    float matMean[][];
    float O[][];
    float Kc[][], A[][], B[][], C[][];
    O = ones(K.length, K.length);

    // A = K - O*K
    A = Mat.subtract(K,Mat.multiply(O, K));

    // B = K*O
    B = Mat.multiply(K,O);

    // C = O*K*O
    C = Mat.multiply(O,Mat.multiply(K,O));

    // D = A - B
    D = Mat.subtract(A,B);

    // Kc = D + C
    Kc = Mat.sum(D,C);

    //verificando se a matriz está centrada
    println("Mean:");
    for(int i=0; i<X[0].length; i++)
      print(" "+Descriptive.mean(getArray(Kc,i)));
    println();

    return Kc;
}
```

Lembrando que a média da matriz Kc tem que ser muito próxima a zero.

```
public float[][]  solve(int type, float parameter)
{
    float K[][] = kernelMatrix(type, parameter);
    float Kc[][] = centralize( K );

    return null;
}
```

Executar o **AppKPCA** para verificar o resultado, **CUIDADO** com o retorno do **solve**, pode dar **null exception**.

Agora vamos implementar o **solve** do kPCA. O método **solve** centra a matriz **K** e calcula os autovalores e autovetores de **Kc**. E a projeção é realizada usando a equação:

$$Y = K_c V$$

sendo V os autovetores (componentes principais) e Y a projeção. Não considerei o **k** ainda (fica como **exercício**).

```
public float[][]  solve(int type, float parameter)
{
    float K[][] = kernelMatrix(type, parameter);
    float Kc[][] = centralize( K );


    //calculando os autovalores e autovetores de Kc
    Eigenvalue eigen = new Eigenvalue(Kc);
    V = Cast.doubleToFloat(eigen.getV());
    D = Cast.doubleToFloat(eigen.getD());

    //analisando os autovalores de Kc
    for(int i=0; i < D.length;i++)
      println(D[i][i]);

    //calculando a projeção Y = Kc*V
    return Mat.multiply(Kc, V);
}
```

Verificando a projeção Numericamente.

```
String [] textlines;

DataReader data;
kPCA        kpca;
float X[][];
int   L[];

void setup()
{
  //kPCA part
  data = new DataReader("spheres.dat", ",", 3);

  X = data.getData();
  L = data.getLabels();

  kpca = new kPCA(X, false);
  float Y[][] = kpca.solve(0, 20);//passamos como parâmetros o typo de kernel a ser
                                  //aplicado e o parâmetro deste kernel

  Mat.print(Y,2);
}
```

Implementando a interface gráfica. A parte referente a interface gráfica é a mesma que implementamos no PCA.

```
import grafica.*;

String [] textlines;

DataReader data;
kPCA        kpca;
float X[][];
int   L[];
```

```
int nPoints;
GPointsArray points;
GPlot plot;

void setup()
{
  //kPCA part
  data = new DataReader("spheres.dat", ",", 3);

  X = data.getData();
  L = data.getLabels();

  kpca = new kPCA(X);

  //retorna a projeção
  float Y[][] = kpca.solve(0, 20);

  nPoints = Y.length;
  size(500, 350);
  points = new GPointsArray(nPoints);

  for (int i = 0; i < nPoints; i++) {
      points.add(Y[i][0],Y[i][1]);
  }

  plot = new GPlot(this);
  plot.setPos(25, 25);

  plot.setTitleText("Non Linear Principal Component Analysis");
  plot.getXAxis().setAxisLabelText("PC-1");
  plot.getYAxis().setAxisLabelText("PC-2");

  plot.setPoints(points);
}

void draw() {
  background(150);

  plot.beginDraw();
    plot.drawBackground();
    plot.drawBox();
    plot.drawXAxis();
    plot.drawYAxis();
    plot.drawTopAxis();
    plot.drawRightAxis();
    plot.drawTitle();

    for (int i = 0; i < nPoints; i++ )
    {
     if( L[i] == 1)
      plot.drawPoint(points.get(i),color(255, 0, 0),5);
    else
      if( L[i] == 2)
          plot.drawPoint(points.get(i),color(0, 255, 0),5);
       else
          plot.drawPoint(points.get(i),color(0, 0, 255),5);

  }
  plot.endDraw();
}
```

Após projetar os pontos, verificaremos não saiu o resultado que esperávamos (Figura 3).

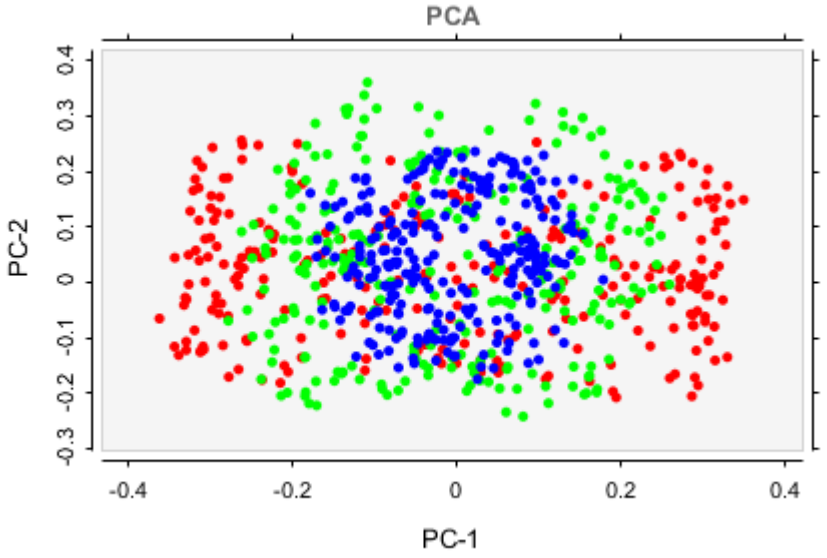

Figura 3 - Projeção sem efetuar a análise dos autovalores.

Isso ocorre pelo fato do método de cálculo de autovalores do **papaya** não ordenar adequadamente os autovalores. Para resolver este problema, basta analisar os autovalores que verificaremos que os mesmos estão ordenados de forma crescente. Então vamos substituir a linha

```
points.add(Y[i][0],Y[i][1]);
```

por

```
points.add(Y[i][Y[i].length-1],Y[i][Y[i].length-2]);
```

pois os autovalores com a maior variância são os dois últimos. A figura 3 ilustra a projeção obtida com esta modificação.

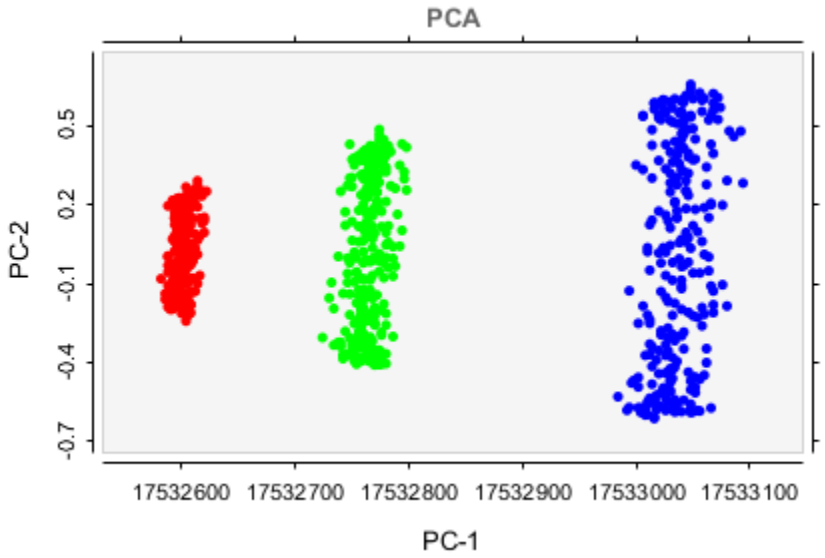

Figura 4 - Projeção realizada sobre os 2 componentes principais com as maiores variâncias.

OBS: Se o método não fornece os autovalores ordenados, o programador deverá ordena-los. Para esta ordenação tem que ser realizada sobre os autovalores e refletir no respectivos autovetores.

Exercício: implementar o kernel polynomial.

$$K(x,y) = (x^T y)^p$$

sendo p o parâmetro.

Trabalho: Implementar para o PCA e o kPCA a ordenação dos autovalores e autovetores e a escolha do k. Selecionar um conjunto de dados e aplicar as duas técnicas, fazer relatório e enviar o algoritmo.