PCA: Principal Component Analysis

PCA é muito usado para diminuir a dimensionalidade das características (features).

Na internet há muitos textos bons sobre PCA:

[1] Um exemplo numérico muito bom está em (Brian Russell):
www.cgg.com/technicalDocuments/cggv_0000014063.pdf

[2] Wikipedia:
https://en.wikipedia.org/wiki/Principal_component_analysis

[3] NIST:
http://www.itl.nist.gov/div898/handbook/pmc/section5/pmc55.htm

Aqui escrevemos os programas em Cekeikon/OpenCV para PCA. Escreveremos usando a classe PCA pronta do OpenCV e também usando a definição de PCA (através do cálculo da matriz de covariância e autovetores).

Nota 1) PCA analisa somente os atributos de entrada, sem levar em conta a saída. Assim, PCA não consegue distinguir atributos "úteis" dos "inúteis" ou "ruídos". Para PCA, um atributo com valores aleatórios terá tanta importância quanto um outro atributo que determina a classificação. Se calcular PCA da figura 1a, será obtida algo como a figura 1b. PCA considera o atributo 1 da figura 1b mais importante do que o atrabuto f2.

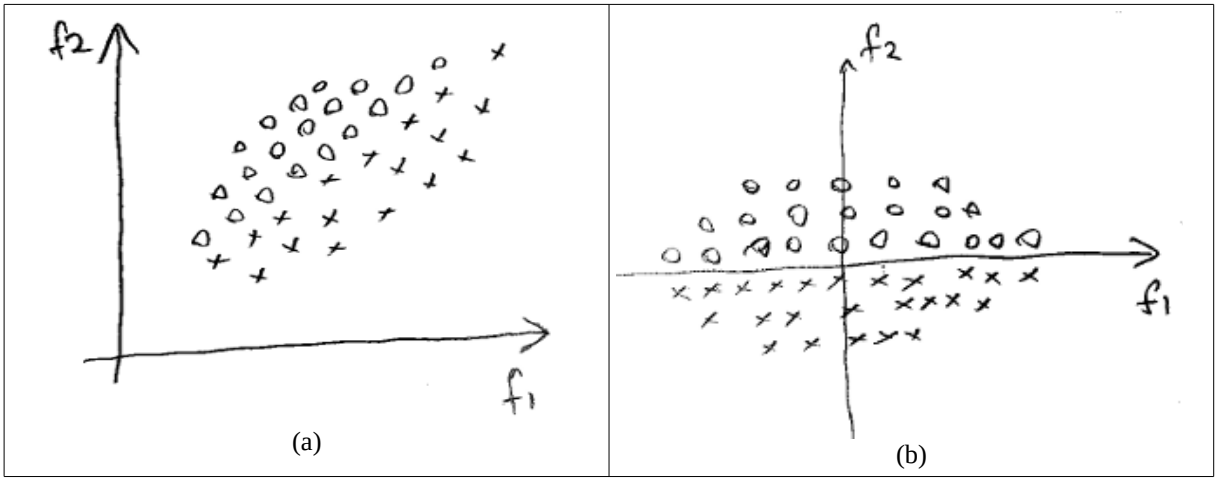


Figura 1: PCA não distingue atributos úteis dos inúteis. PCA considera f1 mais importante do que f2.

Nota 2) PCA analisa o desvio-padrão em torno da média. Assim, todos os atributos devem estar na mesma "unidade" para que PCA faça sentido. Assim, não podem coexistir atributos em metros e outros em quilômetros, assim como não podem coexistir atributos em quilogramas e outros em volts.

Nota 3) Para poder usar PCA, cada atributo deve ter média zero. As implementações de PCA subtraem média antes de calcular PCA.

Programa que implementa o exemplo numérico de [1]:

```
//russell2.cpp
#include <cekeikon.h>

int main() {
  xdebug1("Simulando o artigo do Brian Russell...")
  Mat_<FLT> S=(Mat_<FLT>(3,2) << 1, -1,
                                 0,  1,
                                -1,  0);
  xprint(S);

  { //<<<<<<<<<<< PCA pela definicao <<<<<<<<<<<<<<<<<<<<<<<<
    xdebug1("Calculando PCA pela definicao...")
    Mat_<FLT> C; //matriz de covariancia
    Mat_<FLT> mean; //media coluna a coluna - da zero neste caso
    calcCovarMatrix(S, C, mean, CV_COVAR_NORMAL | CV_COVAR_ROWS, CV_32F);
    xprint(C);

    Mat_<FLT> autoval; //autovalores
    Mat_<FLT> U; //matriz de autovetores U - coluna a coluna
    eigen(C,autoval,U);
    xprint(autoval);
    xprint(U);

    Mat_<FLT> P=S*U; //matriz de componentes principais - coluna a coluna
    xprint(P);

    //reconstruir coeficientes (1,0) e (0,1)
    int ncoeffs=2;
    for (int i=0; i<ncoeffs; i++) {
      Mat_<FLT> coeffs(1,ncoeffs,0.0); coeffs(i)=1.0;
      Mat_<FLT> reconstructed=coeffs*U;
      xprint(reconstructed);
    }
  }

  { //<<<<<<<<<< usando classe PCA <<<<<<<<<<<<<<<<<<<<<<<<
    xdebug1("Usando classe PCA...");
    int ncoeffs=2;
    PCA pca(S, // pass the data
            Mat(), // there is no pre-computed mean vector,
                   // so let the PCA engine to compute it
            CV_PCA_DATA_AS_ROW, // indicate that the vectors are stored
                                // as matrix rows
            ncoeffs // specify how many principal components to retain
    );

    //Obtendo matriz de autovetores U - coluna a coluna
    Mat_<FLT> U(ncoeffs,ncoeffs);
    for (int i=0; i<ncoeffs; i++) {
      Mat_<FLT> coeffs(1,ncoeffs,0.0); coeffs(i)=1.0;
      Mat_<FLT> reconstructed;
      pca.backProject(coeffs, reconstructed);
      reconstructed.copyTo(U.row(i));
    }
    xprint(U);
  }
}
```

```
c:\haepi\algpi\pca\russell>russell2
File=russell2.cpp line=5 Simulando o artigo do Brian Russell...
S =
[1, -1;
 0, 1;
 -1, 0]
File=russell2.cpp line=12 Calculando PCA pela definicao...
C =
[2, -1;
 -1, 2]
autoval =
[3;
 1]
U =
[-0.70710677, 0.70710677;
 0.70710677, 0.70710677]
P =
[-1.4142135, 0;
 0.70710677, 0.70710677;
 0.70710677, -0.70710677]
reconstructed =
[-0.70710677, 0.70710677]
reconstructed =
[0.70710677, 0.70710677]
File=russell2.cpp line=37 Usando classe PCA...
U =
[-0.70710683, 0.70710683;
 0.70710683, 0.70710683]
```

Simulando o exemplo numérico do artigo [2]:

```
//nist4.cpp
#include <cekeikon.h>

int main() {
  xdebug1("Simulando artigo de NIST...")
  Mat_<FLT> S=(Mat_<FLT>(10,3) <<
                7, 4, 3,
                4, 1, 8,
                6, 3, 5,
                8, 6, 1,
                8, 5, 7,
                7, 2, 9,
                5, 3, 3,
                9, 5, 8,
                7, 4, 5,
                8, 2, 2
              );
  xprint(S);
  Mat_<FLT> media=dcRejectCol(S);
  xprint(S);
  xprint(media);

  {
    //<<<<<<<<<<<<<< PCA pela definicao <<<<<<<<<<<<<<<<<<<<
    Mat_<FLT> C,mean;
    calcCovarMatrix(S, C, mean, CV_COVAR_NORMAL | CV_COVAR_ROWS, CV_32F);
    xprint(C);

    Mat_<FLT> autoval,U; //autovalores e autovetores
    eigen(C,autoval,U);
    xprint(autoval);
    xprint(U);

    Mat_<FLT> P=S*U; //componentes principais
    xprint(P);

    //reconstruir coeficientes (1,0,0); (0,1,0); (0,0,1)
    //Multiplicar coeficientes por U equivale a backprojection
    int ncoeffs=3;
    for (int i=0; i<ncoeffs; i++) {
      Mat_<FLT> coeffs(1,ncoeffs,0.0); coeffs(i)=1.0;
      Mat_<FLT> reconstructed=coeffs*U;
      xprint(reconstructed);
    }
  }

  {
    //<<<<<<<<<<<<<< usando classe PCA <<<<<<<<<<<<<<<<<<<<<<<<
    xdebug1("Usando classe PCA...");
    int ncoeffs=3;
    PCA pca(S,
            Mat(),
            CV_PCA_DATA_AS_ROW,
            ncoeffs
    );

    //Obtendo matriz de autovetores U - coluna a coluna pelo backprojection
    Mat_<FLT> U(ncoeffs,ncoeffs);
    for (int i=0; i<ncoeffs; i++) {
      Mat_<FLT> coeffs(1,ncoeffs,0.0); coeffs(i)=1.0;
      Mat_<FLT> reconstructed;
      pca.backProject(coeffs, reconstructed);
      reconstructed.copyTo(U.row(i));
    }
    xprint(U);
  }
}
```

```
c:\haepi\algpi\pca\nist>nist4
File=nist4.cpp line=5 Simulando artigo de NIST...
S =
[7, 4, 3;
 4, 1, 8;
 6, 3, 5;
 8, 6, 1;
 8, 5, 7;
 7, 2, 9;
 5, 3, 3;
 9, 5, 8;
 7, 4, 5;
 8, 2, 2]
S =
[0.099999905, 0.5, -2.0999999;
 -2.9000001, -2.5, 2.9000001;
 -0.9000001, -0.5, -0.099999905;
 1.0999999, 2.5, -4.0999999;
 1.0999999, 1.5, 1.9000001;
 0.099999905, -1.5, 3.9000001;
 -1.9000001, -0.5, -2.0999999;
 2.0999999, 1.5, 2.9000001;
 0.099999905, 0.5, -0.099999905;
 1.0999999, -1.5, -3.0999999]
media =
[6.9000001, 3.5, 5.0999999]
C =
[20.9, 14.5, -3.9000008;
 14.5, 22.5, -11.5;
 -3.9000008, -11.5, 70.900002]
autoval =
[74.465485;
 33.085167;
 6.7493539]
U =
[-0.13757084, -0.25045973, 0.95830292;
 0.69903731, 0.66088945, 0.27307993;
 0.70172751, -0.70745713, -0.084161557]
P =
[-1.1378661, 1.7910585, 0.40910941;
 0.68637204, -2.9775162, -3.7058468;
 -0.29587758, -0.034285292, -0.99059653;
 -1.2808175, 4.2772923, 2.0818954;
 2.2305105, -0.62834013, 1.303846;
 1.6744242, -3.7754631, -0.64201975;
 -1.5617617, 1.6310886, -1.7805763;
 2.794667, -1.586257, 2.1779871;
 0.26558888, 0.37614441, 0.24078631;
 -3.3752391, 0.92627704, 0.90541399]
reconstructed =
[-0.13757084, -0.25045973, 0.95830292]
reconstructed =
[0.69903731, 0.66088945, 0.27307993]
reconstructed =
[0.70172751, -0.70745713, -0.084161557]
File=nist4.cpp line=49 Usando classe PCA...
U =
[-0.13757093, -0.25045979, 0.95830309;
 0.69903713, 0.66088939, 0.27308002;
 0.70172739, -0.70745718, -0.084161483]
```

mnist1.cpp: Comprime dígitos MNIST usando classe PCA e 20 coeficientes. Depois, descomprime usando classe PCA e definição.

```
//mnist1.cpp
#include <cekeikon.h>

Mat_<FLT> dcRejectCol(Mat_<FLT>& S) {
  // Calcula media coluna a coluna.
  // Subtrai media de cada coluna.
  // Retorna a media
  Mat_<FLT> media(1,S.cols);
  for (int c=0; c<S.cols; c++) {
    media(c)=mean(S.col(c))[0];
    S.col(c) -= media(c);
  }
  return media;
}

int main() {
  xdebug1("Lendo...");
  int nlado=16;
  int ncoeffs=20;
  MNIST mnist(nlado, true, true);
  mnist.le("c:/haebase/mnist");

  xdebug1("Calculando PCA...");
  Mat_<FLT> media=dcRejectCol(mnist.ax);
  PCA pca(mnist.ax, Mat(), CV_PCA_DATA_AS_ROW, ncoeffs);

  xdebug1("Calculando autovetores...");
  Mat_<FLT> U(ncoeffs,nlado*nlado);
  for (int i=0; i<ncoeffs; i++) {
    Mat_<FLT> coeffs(1,ncoeffs,0.0); coeffs(i)=1.0;
    Mat_<FLT> reconstructed;
    pca.backProject(coeffs, reconstructed);
    reconstructed.copyTo(U.row(i));
  }

  xdebug1("Imprimindo autovetores...");
  for (int i=0; i<ncoeffs; i++) {
    Mat_<FLT> reconstructed;
    U.row(i).copyTo(reconstructed);
    reconstructed += media;
    Mat_<FLT> reshaped=reconstructed.reshape(1,nlado);
    imp(reshaped,format("U%02d.png",i));
  }

  xdebug1("Projeta e retroprojeta 10 imagens...");
  for (int i=0; i<9; i++) {
    imp(mnist.QX[i],format("qx%02d-ori.png",i));
    Mat_<FLT> coeffs;
    Mat_<FLT> teste=mnist.qx.row(i)-media;
    pca.project(teste, coeffs);

    //usando classe PCA
    Mat_<FLT> rec1;
    pca.backProject(coeffs,rec1);
    rec1 += media;
    Mat_<FLT> reshaped=rec1.reshape(1,nlado);
    imp(reshaped,format("qx%02d-re1.png",i));

    //usando definicao
    Mat_<FLT> rec2(1,nlado*nlado,0.0);
    for (int j=0; j<ncoeffs; j++)
      rec2 = rec2 + coeffs(j)*U.row(j);
    rec2 += media;
    Mat_<FLT> reshaped2=rec2.reshape(1,nlado);
    imp(reshaped2,format("qx%02d-re2.png",i));
  }
}
```

 U00	 U01	 U02		
				
				
				 U19

Base ortogonal

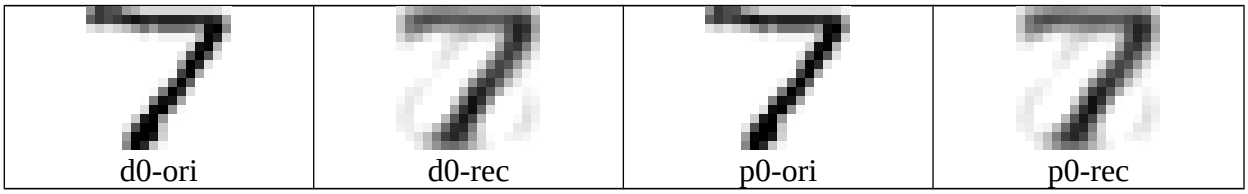
ori	re1 - reconstruído pela classe PCA	re2 - reconstruído pela definição	ori	re1 - reconstruído pela classe PCA	re2 - reconstruído pela definição
 qx00	 qx00	 qx00	 qx01	 qx01	 qx01
					
					
					

Imagens reconstruídas usando 20 primeiros coeficientes

mnist4.cpp: Comprime dígitos MNIST usando definição da PCA e 20 coeficientes e descomprime. Efetua a mesma operação usando classe PCA.

```
//mnist4.cpp
#include <cekeikon.h>

void imprimir(Mat_<FLT> a, Mat_<FLT> media, int nlado, string nome) {
  assert(a.rows==1 && media.rows==1 && a.cols==media.cols);
  Mat_<FLT> b=a+media;
  Mat_<FLT> reshaped=b.reshape(1,nlado);
  imp(reshaped,nome);
}

int main() {
  xdebug1("Lendo...");
  int nlado=16;
  int ncoeffs=20;
  MNIST mnist(nlado, true, true);
  mnist.le("c:/haebase/mnist");

  xdebug1("Eliminando DC...");
  Mat_<FLT> media=dcRejectCol(mnist.ax);

  {
    xdebug1("Calculando PCA pela definicao...");
    Mat_<FLT> C,mean;
    calcCovarMatrix(mnist.ax, C, mean, CV_COVAR_NORMAL | CV_COVAR_ROWS, CV_32F);
    Mat_<FLT> autoval,U; //autovalores e autovetores
    eigen(C,autoval,U);

    xdebug1("Projeta e retroprojeta imagem 0...");
    Mat_<FLT> ori=mnist.qx.row(0)-media;
    //xprint(ori);
    imprimir(ori,media,nlado,"d0-ori.png");

    Mat_<FLT> t=ori*U.t(); //note que tem que usar a transposta
    Mat_<FLT> coeffs(t,Rect(0,0,ncoeffs,1));
    xprint(coeffs);

    Mat_<FLT> Uroi(U,Rect(0,0,U.cols,coeffs.cols));
    Mat_<FLT> rec=coeffs*Uroi;
    //xprint(rec);
    imprimir(rec,media,nlado,"d0-rec.png");
  }

  {
    xdebug1("Usando classe PCA...");
    PCA pca(mnist.ax,
            Mat(),
            CV_PCA_DATA_AS_ROW,
            ncoeffs
    );

    xdebug1("Projeta e retroprojeta imagem 0...");
    Mat_<FLT> ori=mnist.qx.row(0)-media;
    //xprint(ori);
    imprimir(ori,media,nlado,"p0-ori.png");

    Mat_<FLT> coeffs;
    pca.project(ori, coeffs);
    xprint(coeffs);

    Mat_<FLT> rec;
    pca.backProject(coeffs,rec);
    //xprint(rec);
    imprimir(rec,media,nlado,"p0-rec.png");
  }
}
```

```
c:\haepi\algpi\pca\mnist>mnist4
File=mnist4.cpp line=12 Lendo...
File=mnist4.cpp line=18 Eliminando DC...
File=mnist4.cpp line=22 Calculando PCA pela definicao...
File=mnist4.cpp line=28 Projeta e retroprojeta imagem 0...
coeffs =
[-0.98118114, 2.3771076, 0.93955755, -1.2493175, 0.18989184, 2.4279239, -0.01627
4298, 1.0335188, 0.70490503, -0.78367448, -0.664038, -1.3154854, 0.79543525, 0.7
9455191, 0.645419, 0.26229927, -0.56779844, 0.34890088, 0.32819107, 0.37063813]
File=mnist4.cpp line=44 Usando classe PCA...
File=mnist4.cpp line=51 Projeta e retroprojeta imagem 0...
coeffs =
[-0.98117954, 2.377115, 0.93955672, -1.2493217, 0.18988875, 2.4279282, -0.016270
775, 1.0335186, 0.70490432, -0.78366947, -0.66404879, -1.3154905, 0.79542893, 0.
79455191, 0.64541781, 0.26229912, -0.56780094, 0.34889847, 0.32818699, 0.3706376
3]
```



recmnist1.cpp: Comprime os dígitos de MNIST para 144 features usando PCA. Treina e faz predição usando flann.

```
//recmnist1.cpp - flann - opencv2 ou opencv3
#include <cekeikon.h>

int main() {
  xdebug1("Lendo...");
  //int nlado=16;
  int nlado=28;
  int ncoeffs=144;
  MNIST mnist(nlado, true, true);
  mnist.le("c:/haebase/mnist");

  xdebug1("Eliminando DC...");
  Mat_<FLT> media=dcRejectCol(mnist.ax);
  for (int l=0; l<mnist.nq; l++)
    mnist.qx.row(l) -= media;

  xdebug1("Usando classe PCA...");
  PCA pca(mnist.ax,
          Mat(),
          CV_PCA_DATA_AS_ROW,
          ncoeffs
  );

  xdebug1("Projeta mnist.ax e mnist.qx cria fax e fqx...");
  Mat_<FLT> fax(mnist.na,ncoeffs);
  for (int l=0; l<mnist.na; l++)
    pca.project(mnist.ax.row(l), fax.row(l));
  Mat_<FLT> fqx(mnist.nq,ncoeffs);
  for (int l=0; l<mnist.nq; l++)
    pca.project(mnist.qx.row(l), fqx.row(l));

  xdebug1("treinando");
  flann::Index ind(fax,flann::KDTreeIndexParams(4));

  xdebug1("fazendo predicao");
  //#pragma omp parallel for
  for (int l=0; l<mnist.nq; l++) {
    vector<int> indices(1);
    vector<float> dists(1);
    ind.knnSearch(fqx.row(l),indices,dists,1);
    mnist.qp(l)=mnist.ay(indices[0]);
  }

  printf("Erros=%10.2f\%\n",100.0*mnist.contaErros()/mnist.nq);
}
```

Taxa de erro obtida usando flann: 3,06%.
Taxa de erro obtida usando 144 features pelo redimensionamento (sem PCA) e flann: 3,09%.

Conclusão: Neste exemplo, fazer redução de dimensão usando PCA não foi melhor do que fazer redução de imagem.

Testar se o resultado é melhor usando SVM, árvore de decisão, boosting, etc.