UNIVERSIDAD CATOLICA SAN PABLO - AREQUIPA
FACULTAD DE CIENCIAS DE LA COMPUTACIÓN

Estructura de Datos Avanzados

Informe

PCA

Advisor: Erick Mauricio Gomes Nieto
Carlos Eduardo Atencio Torres

Student: Loayza Huarachi Angel Josue

AREQUIPA, JUNE 2023

Contents

1 Introducción

El PCA, o mejor conocido como el Análisis de Componentes Principales, es una técnica muy utilizada en el análisis de datos. Este se convierte en una estrategia estadística muy conveniente, pues nos ayuda en la reducción de la dimensionalidad, es decir, transforma un conjunto de datos en uno nuevo llamado componentes principales. Dichas componentes son combinaciones lineales estrictamente ordenadas con base en la varianza de todos sus ejes; y su propia estructura nos permite escoger entre estos, el principal. Seleccionando la primera componente, es posible representar todos los datos a una dimensión menor con la seguridad de no perder una cantidad significativa de información.

En este trabajo, se presentará el resultado tras aplicar el Análisis de Componentes Principales (PCA), utilizando el lenguaje de programación C++ junto a la librería Eigen, para extraer información como la matriz centrada, la matriz de covarianza, la matriz de proyección, los *eigenvectors* y los *eigenvalues*; y para su visualización usaremos el lenguaje de programación Python junto a la librería numpy. El trabajo estará limitado a solo mostrar los resultados y alguna u otra explicación de ciertas partes del código donde sea necesario. Aun así, se compartirá el link del repositorio de GitHub para futuras pruebas y mejoras: https://github.com/angel452/PCA

2 Estructura

Como se comentó anteriormente, tenemos 2 archivos: pca_test.cpp y grapher.py. A continuación se mencionarán algunas consideraciones para la comprensión de dicho código.

pca_test.cpp Es en este archivo donde se harán todos los cálculos basándonos en la matriz de entrada que le demos. Dicho algoritmo, me retornará toda la información necesaria para poder graficarlo.

1. Archivos de exportacion:
   Primero creamos los archivos (txt) donde se almacenaran los resultados obtenidos:

```
// ------- Export data to .txt file --------
ofstream puntosXYZ_Originales("puntosXYZ_Originales.txt");
ofstream puntosXYZ("puntosXYZ.txt");
ofstream rectaPCA("rectaPCA.txt");
```

```
ofstream proyeccion("proyeccion.txt");
ofstream reProyeccion("reProyeccion.txt");
ofstream planoOrtogonal("planoOrtogonal.txt");
```

2. Creación de puntos:
   Luego asignamos puntos aleatorios dentro de la matriz *pca_data_matrix* con la función *generarPuntos*. Esta función me retornará una matriz de *nx*3, donde "n" es la cantidad de puntos que queremos generar, y cada punto tendrá 3 dimensiones.

```
    Eigen::Matrix<float, Eigen::Dynamic, Eigen::Dynamic>
generarPuntos(int npuntos){...}

  const int ndim = 3;
  const int npuntos = 500;

  Eigen::Matrix<float, npuntos, ndim> pca_data_matrix;

  // Ejemplo 1...
  pca_data_matrix = generarPuntos(npuntos);
```

3. Cálculo del PCA:
   Luego le pedimos que con los datos ingresados me calcule la matriz centrada, la matriz de covarianza, la matriz de proyección, los eigenvectors y los eigenvalues:

```
pca_t<float> pca;
pca.set_input(pca_data_matrix);
pca.compute();
```

4. Exportamos información:
   A continuación, exportaremos la información obtenida en los archivos txt creados anteriormente.

   - Los puntos originales dentro del archivo puntosXYZ_Originales.txt
   - Los puntos centrados dentro del archivo puntosXYZ.txt
   - La recta/vector del primer componente principal dentro del archivo rectaPCA.txt
   - Las proyecciones de los puntos centrados hacia la recta en el archivo proyeccion.txt

- Las proyecciones de los puntos originales hacia la recta en el archivo reProyeccion.txt

grapher.py En este archivo, lo único que ser hará será leer los archivos exportados y graficarlos usando la librería numpy. Por ello, es que desde la línea 1 a la 34 son puras funciones que leen los txt. Pasamos a explicar el MAIN.

1. Configuraciones iniciales:
   Solo creamos el entorno para poder graficar los puntos, proyecciones y recta, además de ponerle su respectivo nombre a cada eje dimensional para una mejor visualización.

```
1 fig = plt.figure()
2 ax = plt.axes(projection='3d')
3 ax.set_xlabel('X')
4 ax.set_ylabel('Y')
5 ax.set_zlabel('Z')
```

2. Graficamos los puntos centrados:
   Para esto, usamos la función scatter, propia de numpy, y le pasamos los puntos tanto para el eje "X", "Y" y "Z"

```
1 x,y,z = leerPuntosXYZ()
2 ax.scatter(x,y,z)
```

3. Graficamos la recta del primer componente principal:
   Para esto, usamos la función plot3D; donde le pasamos el inicio y el fin donde la recta se extenderá en los 3 ejes. En este ejemplo, si creamos 500 puntos, decimos que la recta se extienda desde el -500 hasta el +500 en todos los ejes.

```
1 recta = leerRectaPCA()
2 ax.plot3D(  [-500*recta[0], 500*recta[0]],
3             [-500*recta[1], 500*recta[1]],
4             [-500*recta[2], 500*recta[2]], 'black' )
```

4. Graficamos las proyecciones:
   La lógica de este gráfico es similar a la anterior, usamos la librería plot3D, pero ahora le pasamos como punto de inicio, el punto centrado; y como final, el punto proyectado del archivo proyeccion.txt. Esto lo hacemos para cada punto previamente guardado.

```
xP,yP,zP = leerProyecciones()
#ax.scatter(xP,yP,zP)
for i in range (len(xP)):
    xAux = x[i]
    yAux = y[i]
    zAux = z[i]
    ax.plot3D( [xAux, xP[i]],
               [yAux, yP[i]],
               [zAux, zP[i]], 'red')
```

5. Mostramos gráfico:
   Por último, mostramos todos los puntos y gráficas creados.
   `plt.show()`

3 Resultados

A continuación, presentaremos 2 casos/ejemplos, donde se usó dicho algoritmo para crear el principal componente basándose en una serie de puntos

3.1 Ejemplo 1:

Ingresaremos manualmente estos puntos dentro de la matriz llamada pca_data_matrix:

```
pca_data_matrix <<
  126.0, 78.0, -1.0,
  128.0, 80.0, -1.0,
  128.0, 82.0, 0.0,
  130.0, 82.0, 0.0,
  130.0, 84.0, 1.0,
```

```
7     132.0, 86.0, 1.0;
```

Además, tendremos que cambiar la cantidad de puntos a 6, para evitar errores:

```
const int npuntos = 6; // linea 60
```

Luego de compilar y ejecutar el proyecto, nuestros archivos estarán actualizados y listos para ser graficados. Así que nos dirigimos al archivo grapher.py y modificaremos la parte donde graficamos la recta PCA, pues ya que son puntos que no se distancian mucho, no es necesario extender la recta demasiado; la adecuaremos a una longitud aceptable para visualizarlo mejor.

```
1  # Graficar recta PCA
2  recta = leerRectaPCA()
3  # --> Para el grafico de la presentacion
4  ax.plot3D(  [-7*recta[0], 7*recta[0]],
5              [-7*recta[1], 7*recta[1]],
6              [-7*recta[2], 7*recta[2]], 'black' )
```

Tras ejecutar el archivo grapher.py, tendremos el siguiente resultado:

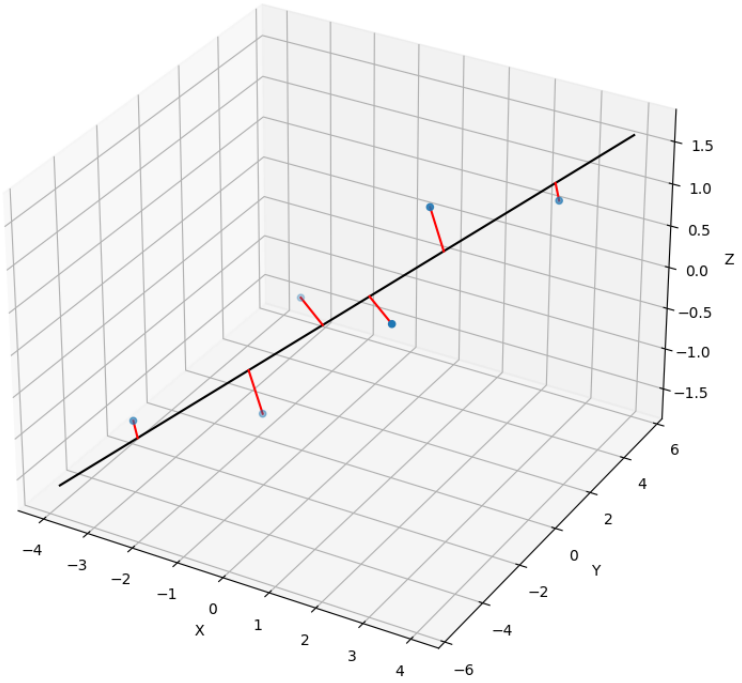

Figure 3.1: Ejemplo 1

3.2 Ejemplo 2:

En este ejemplo propondremos un modelo más complicado; por eso, usaremos la función *generarPuntos* para crear 500 puntos y de igual manera veremos su resultado.
Antes de compilar, debemos ajustarlo para que acepte 500 puntos; es decir, modificamos la línea 60 de la siguiente manera:

```
const int npuntos = 500;
```

Y también la forma en que ingresamos los puntos a la matriz:

```
pca_data_matrix = generarPuntos(npuntos);
```

Compilando y ejecutando dicho proyecto, y con nuestros archivos txt actualizados, procedemos a ejecutar el grapher.py, pero también ajustando la recta PCA para que se pueda visualizar mejor:

```
1 # Graficar recta PCA
2 recta = leerRectaPCA()
3 # --> Para el grafico oficial
4 ax.plot3D(  [-500*recta[0], 500*recta[0]],
5             [-500*recta[1], 500*recta[1]],
6             [-500*recta[2], 500*recta[2]], 'black' )
```

El resultado es el siguiente:

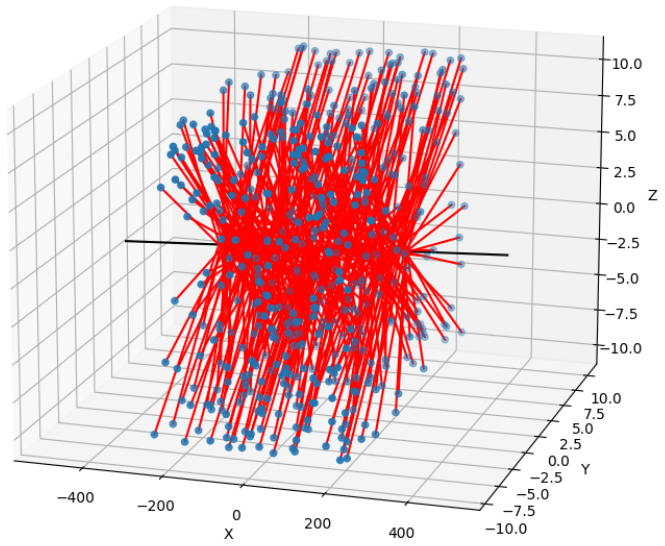

Figure 3.2: Ejemplo 1