Universidade de São Paulo

Instituto de Física de São Carlos

Análise e Reconhecimento de Padrões

Docente: Prof. Luciano Fontoura da Costa

Projeto 3: PCA

Beatriz de Camargo Castex Ferreira
10728077
bcastex@usp.br

São Paulo - 22/05/2020

Quando trabalha-se com atributos muitas vezes é necessário normaliza-los ou transformai-los para conseguir mais informações sobre eles, ou para conseguir um conjunto de dados mais uniforma, diminuindo resíduos por exemplo, o que pode ser feito utilizando vários métodos. Um desses métodos é o chamado Análise de Componente Principal, ou PCA, que descorrelaciona um conjunto de atributos, transformando eles em atributos independentes.

Abaixo é apresentado um programa implementando o PCA e seus resultados:

Por motivos de visualização, os programas escritos no projeto serão apresentados de forma simplificada, ou em imagens (ou seja, não seria possível copiar o código e implementar em seu próprio computador), portanto todos estarão disponíveis em sua integridade, juntamente com quaisquer dados utilizados para gerar os resultados em uma pasta do google drive linkada abaixo.[1]

```python
import numpy as np
import matplotlib.pyplot as plt


# 1 - Gerar distribuição circular de pontos uniformemente distribuídos:

# Quantos pontos?
N = 500

# Podemos escolher pontos em cordenadas polares, escolhendo raios e ângulos
# dentro de r = 1 e theta = 2pi.
r = np.sqrt(np.random.uniform(0, 1, N)) # Temos que escolher com uma
# distribuição proporcional à raiz porque queremos uma distribuição 2D, ou seja,
# tem que ser proporcional à pi*r^2
theta = np.pi*np.random.uniform(0, 2, N)

# Fazemos a transformação para coordenadas cartesianas.
x = r * np.cos(theta)
y = r * np.sin(theta)

# 2 - Comprimir os dados verticalmente para 1/5:
y = y * 0.2

# Rotacionar pontos usando [cos(30) sin(30); sin(30) cos(30)]:
rotation = [[ np.cos(0.523599), np.sin(0.523599) ],
            [ np.sin(0.523599), np.cos(0.523599) ]]
[x, y] = np.dot(rotation, [x, y])
F = np.array([x, y]).T

# 4 - Visualizar os dados, para ver se está parecido com Figura 9, CDT-24
plt.scatter( x, y, color="orange", s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Antes do PCA')
plt.show()

# 5 - Obter matriz de covariância K dos dados gerados:
# Definir função de covariancia
def covariance(X, Y):

    # Obter média de x e y:
    mean_x = np.mean(X)
    mean_y = np.mean(Y)
    # Calcular a covariância:
    return np.sum( (X - mean_x)*(Y - mean_y) )/(len(X) - 1)
```

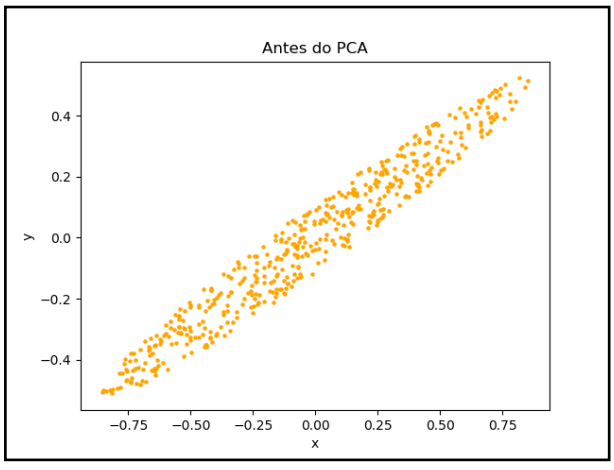
[1]Disponível em: <https://drive.google.com/drive/folders/11ldYReP0aaLtFBBj68nQ8VxenERCeNI7?usp=sharing>

```python
# Montar matriz:
K = np.zeros((N, N))
for i in range(N):
    for j in range(N):
        K[i][j] = covariance(F[i], F[j])

# 6 - Obter autovalores/autovetores de K
eig_values, eig_vectors = np.linalg.eig(K)

# 7 - Ordenar decrescentemente os autovalores juntamente com autovalores
eig_values_dec = np.sort(eig_values)[::-1]

# 8 - Obter matriz Q usando autovetores como linhas

Q = np.zeros((len(eig_vectors), len(eig_vectors)))

for o in range(len(eig_values)):
    for u in range(len(eig_values_dec)):
        if ( eig_values_dec[u] == eig_values[o] ):
            Q[u] = eig_vectors[o]


# 9 - Aplicar nos dados e mostrar novo resultado.
[x, y] = np.dot(Q, F).T

plt.scatter( x, y, color="orange", s=5)
plt.xlabel('x')
plt.ylabel('y')
plt.title('Depois do PCA')
plt.show()
```

pca.py [1] - Fonte: Elaborado pelo compilador.

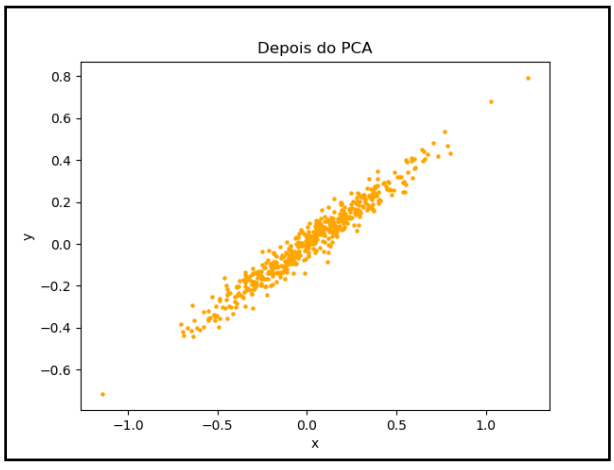
Resultados obtidos:

Figura 1 - Dados antes do PCA



Fonte: Elaborada pelo compilador.

Figura 2 - Dados depois do PCA



Fonte: Elaborada pelo compilador.

Referências

[1] COSTA. L. da F. **Features Transformation and Normalization: A Visual Approach**. Researchgate. Março, 2020 Disponível em: <https://www.researchgate.net/publication/340114268>