# Redução de Dimensionalidade, utilizando PCA, para iniciantes

Uma abordagem de análise de componentes principais (PCA) em dados de câncer de mama (Breast Data Cancer [1] ).

## Introdução

Esse é meu primeiro post, e fiquei me perguntando qual seria o primeiro tópico que gostaria de compartilhar com vocês. Um assunto que achei bastante interessante de ser exposto na minha visão e com minhas palavras seria redução de dimensionalidade, utilizando a técnica de análise de componentes principais (PCA). A abordagem que decidi seguir neste tópico será bem introdutória, com uma aplicação prática. E espero que todos gostem.

Primeiro irei expor o que seria redução de dimensionalidade e suas vantagens. Para em seguida explicar basicamente o que seria PCA, e por fim, mostrar a técnica em um dataset real (dados de câncer de mama).

## Mas o que seria redução de dimensionalidade?

Redução de dimensionalidade é um processo de transformar os dados de uma dimensão para outra de dimensionalidade menor. Há um trade-off de perda de informação, com ganho computacional. Segundo Shalev-Shwartz (2014, p.323), este processo está intimamente relacionado ao conceito de compressão (com perdas) na teoria da informação.

## Quais seriam as vantagens de reduzir a dimensão dos dados?

A. A alta dimensionalidade requer muitas vezes poder computacional;
B. Muitas dimensões aumentam a complexidade dos modelos. O aumento do número de features, aumenta as chances de overfitting;
C. Reduzir o número de features com pequenas perdas nos dados pode melhorar a performance (tanto da acuracidade do modelo, quanto no tempo e nos recursos de memória), nos modelos de predição;
D. Redução de dimensionalidade pode ser usada para melhorar a interpretação dos dados, sua visualização, e assim, conseguir extrair informações.

Iremos representar o problema de escrever os dados em uma base de dimensão menor, matematicamente.

## O que seria a Análise de Componentes Principais?

*Principal Components Analysis* (PCA) é uma transformação linear ortogonal que transforma o dado para um novo sistema de coordenadas de modo que a menor variância de alguma projeção escalar dos dados esteja na primeira coordenada (chamada de primeiro componente principal), a segunda maior variância na segunda coordenada e assim por diante [3].

Assim, os primeiros componentes principais descrevem melhor os dados. E a utilização de PCA na redução de dimensionalidade recai em escolher um número de componentes menor que o número de features dos dados originais, retendo o máximo possível da informação.

Iremos descrever o problema de redução de dimensionalidade por transformações lineares.

As $m$ observações de nossos dados podem ser representadas por $x \in \mathbb{R}^{d,m}$. Dessa forma cada linha de $x$ é uma observação de dimensão $d$. Seja a matriz de projeção na nova base $W \in \mathbb{R}^{n,d}$, tal que $x$ seja mapeado em $Wx$. Cada linha de $Wx$ tem dimensão $n$, em que $d > n$. Portanto, cada observação $x_i$ , em que antes representamos por $d$ features, após a transformação será representada por $n$ features, havendo uma redução de dimensionalidade.

Iremos descrever também o problema de recuperação da informação que foi compactada por transformação linear. Ou seja, dado a nova representação de $x_i$ na nova base, qual seria a transformação linear que transformaria essa nova representação na representação original? Queremos na verdade o problema inverso!

Considere uma matriz secundária $U \in \mathbb{R}^{d,n}$ de modo que $UWx$ se aproxime de $x$, fazendo o mapeamento inverso (o problema inverso). Essa transformação inversa tem que se aproximar o máximo possível de $x$. Dessa forma, avaliaremos essa recuperação por uma determinada função de erro.

PCA é uma técnica em que tanto a compactação da informação quanto a recuperação é feita por transformações lineares, como já descrevemos, e o erro desse mapeamento inverso é o erro quadrático.

Assim, gostaríamos de encontrar $U$ e $W$ de modo que o erro quadrático desse mapeamento inverso seja o menor possível:

$$\underset{W,U}{\operatorname{argmin}} \sum_{i=1}^{m} \|x_i - UWx_i\|_2^2 \qquad \text{[Problema de Minimização]}$$

A solução do problema de minimização descrito acima tem como solução $U$ uma matriz ortogonal e $W = U^T$. Assim, PCA é uma técnica transforma os dados originais em uma nova base ortogonal, como havíamos exposto.

Iremos deixar para um próximo post a solução matemática para o problema de minimização. Além disso, há diversas técnicas de redução de dimensionalidade , como *Non-negative matrix factorization* (NMF), *Linear discriminant analysis* (LDA), *Autoencoders*, entre outras.

# Abordagem Prática em Python

## 1) Dados de Câncer de Mama

Os dados de câncer de mama definidos em [1] são dados de 569 pacientes que possuem tumores benignos ou malignos (câncer).

Há 30 informações (features) de cada paciente. Essas informações são dados métricos do seio da mulher, levando em conta não somente tamanho, mas também rigidez e aspereza, por exemplo. Desses 569 pacientes, 357 tem tumor benigno e 212 tem tumor maligno. Obtivemos esse dataset público por meio do *package sklearn.*

Extraindo os dados do dataset:

```
#import and preparing the dataset
from sklearn.datasets import load_breast_cancer
#loading de data
breast_cancer = load_breast_cancer()
```

O dataset de câncer de mama é um dicionário com uma chave *data,* que contém os dados, e uma chave *target* que contém a categoria do tumor (maligno ou benigno).

Criando um *dataframe* com os dados importados:

```
#creating dataframe with the data
df_breast = pd.DataFrame(breast_cancer.data,
columns=breast_cancer.feature_names)
```

Extraindo e criando as labels de classificação:

```
#filling with the target values
df_breast["target"] = breast_cancer.target
#mapping categorical values
df_breast["target_label"] = df_breast.map({0:"maligno",1:"benigno"})
```

Iremos normalizar os dados utilizando a média e a variância dos dados.

Para isso , utilizaremos a classe *StandardScaler* do *package sklearn.* Passamos os valores para o método *fit_transform,* que irá normalizar os dados, excluindo a coluna *target.*

```
# Standardize the feature matrix
features_names = set(df_breast.columns)-set(["target"])
standart_data =
StandardScaler().fit_transform(df_breast[features_names].values)
```

Neste ponto, *standart_data* são os valores do *dataframe* normalizados, excluindo a coluna target.

Nestes dados, iremos aplicar a técnica PCA. São dois pontos principais a atentar: número de componentes mínimo e porcentagem da variância dos dados retido. Vamos utilizar a classe PCA do package do *sklearn*.

A classe PCA, é inicializada recebendo *n_components* (número de componentes principais) como parâmetro. Porém, se esse número for entre 0 e 1, representa não o número de componentes, mas sim a variância acumulada retida.

A função criada abaixo (*pca_model*) recebe como parâmetro os dados (*data*) e a porcentagem da variância retida (*retained_variance*), e em seguida inicializa a classe PCA e transforma os dados para a nova base.

Retornamos o objeto pca criado e exibimos o número de features nessa nova representação dos dados.

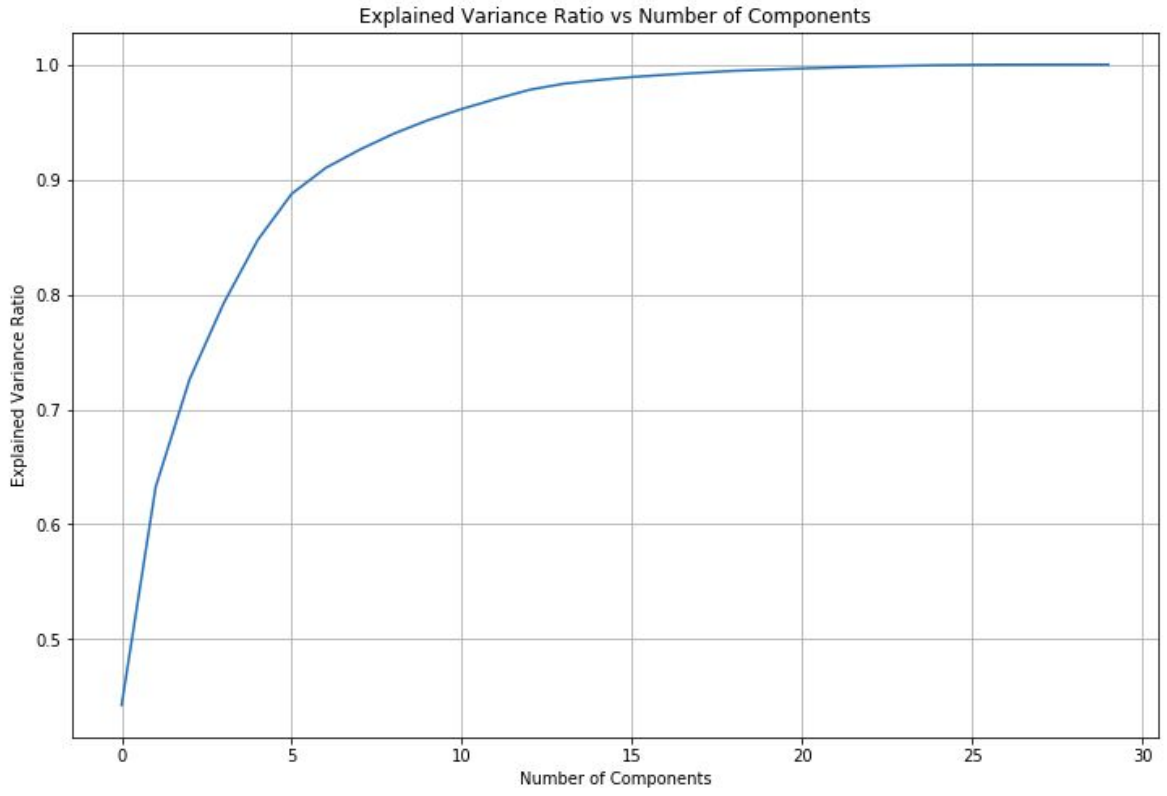
```
def pca_model(data, retained_variance):
    # Create a PCA that will retain some % of variance
    pca = PCA(n_components=retained_variance, whiten=True)

    # Conduct PCA
    data_pca = pca.fit_transform(data)

    # Show results
    print("Original number of features:", data.shape[1])
    print("Reduced number of features:", data_pca.shape[1])

    return data_pca
```

Para a 99% da variância retida, são necessários 17 componentes! Para 95%, somente 10 componentes e para 90%, 7 componentes, conforme a figura a seguir:

```
pca_model(data_standarzed,0.99)
pca_model(data_standarzed,0.95)
pca_model(data_standarzed,0.90)
```

```
Original number of features: 30
Reduced number of features: 17
Original number of features: 30
Reduced number of features: 10
Original number of features: 30
Reduced number of features: 7
```

O gráfico a seguir é a variância retido pelo número de componentes principais. Por esse gráfico, é possível perceber que com 20 componentes somente, conseguimos descrever praticamente toda a informação, com poucas perdas.




Para construir esse gráfico, utilizamos o atributo *explained_variance_ratio_*, que contém a variância retida dos principais componentes. Então, basicamente se somarmos cumulativamente componente por componente, conseguimos identificar a variância retida por um dado número de componentes. Utilizando esse raciocínio, plotamos o gráfico exibido. O código está disponível a seguir:

```
def plot_explained_varience(pca):
    '''
    This function received the transformed data and plots it in scree
plots
    '''
    plt.plot(np.cumsum(pca.explained_variance_ratio_))
    plt.xlabel('Number of Components')
    plt.ylabel('Explained Variance Ratio')
    plt.title('Explained Variance Ratio vs Number of Components')
    plt.grid(b=True)
    plot = plt.show()
```

Vamos então escolher 3 componentes principais . Ou seja, representaremos os dados com somente 3 componentes.

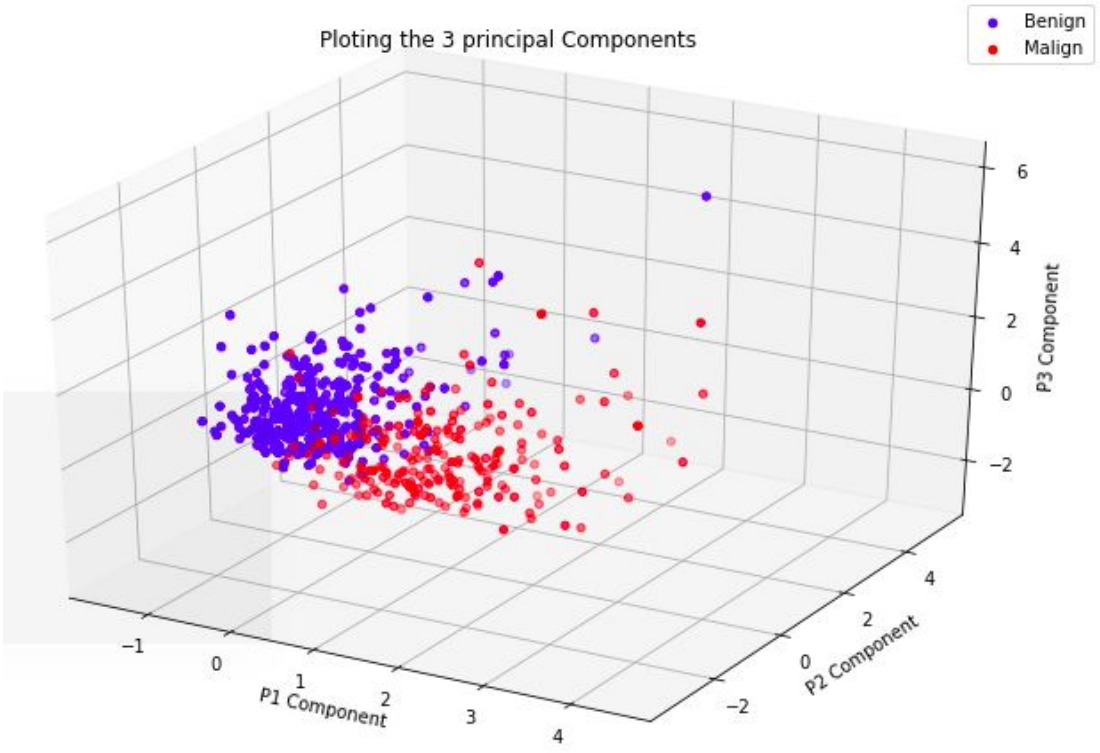
```
#Using 3 components
pca = PCA(n_components=3, whiten=True)
data_pca = pca.fit_transform(data_standarzed)

#Constructing DataFrame
df_pca_3components = pd.DataFrame(data=data_pca)
df_pca_3components["target"] = df_breast["target"].values
```

Com 3 componentes retemos 72.6% da informação. O primeiro componente 44.27%, enquanto o terceiro 9% somente.

```
print(pca.explained_variance_ratio_)
sum(pca.explained_variance_ratio_)
```

```
[0.44272026 0.18971182 0.09393163]

0.7263637090860156
```

Iremos plotar como os dados se dividem somente utilizando 3 componentes principais:




Há uma divisão bem aparente entre os dois tipos de tumor quando plotamos utilizando as 3 primeiras componentes principais.

O código para plotar o gráfico está explicado abaixo.

```
from mpl_toolkits.mplot3d import Axes3D

fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

#selecting Benign Labels
df_pca_3components_benign =
df_pca_3components[df_pca_3components["target"]==1]
#selecting Malign Labels
df_pca_3components_malign =
df_pca_3components[df_pca_3components["target"]==0]

#selecting x,y,z values from benign labels
x_benign = df_pca_3components_benign[0].values
y_benign = df_pca_3components_benign[1].values
z_benign = df_pca_3components_benign[2].values

#Color of benign will be blue
color_benign = "blue"
```

```
#selecting x,y,z values from malign labels

x_malign = df_pca_3components_malign[0].values
y_malign = df_pca_3components_malign[1].values
z_malign = df_pca_3components_malign[2].values

#Color of malign will be red
color_malign = "red"

#Ploting Both Benign and Malign
ax.scatter(x_benign,y_benign,z_benign, c=color_benign,label="Benign")
ax.scatter(x_malign,y_malign,z_malign, c=color_malign,label="Malign")
ax.legend()

ax.set_xlabel('P1 Component')
ax.set_ylabel('P2 Component')
ax.set_zlabel('P3 Component')
ax.set_title("Ploting the 3 principal Components")
plt.show()
```

Uma outra questão bem interessante é quais features dos dados originais tem maior peso nos componentes principais? Ou seja, dado o primeiro componente principal, que retém 44.27% da informação, quais features possuem mais peso?

Para responder isso é bem simples. O atributo *components_* da classe PCA, mostra os pesos de cada feature original na transformação linear para a nova base. Por exemplo, usamos 3 componentes principais na análise acima, retornará um array de formato (3,30), sendo 30 o número de features originais.

Dessa forma, na posição 0 (primeiro componente principal), há um array de tamanho 30, que representa o peso de cada uma das features!

```
pca.components_.shape
```

```
(3, 30)
```

Sendo assim, vamos verificar então quais features tem maior peso no primeiro componente principal. O código abaixo mostra as features com maior peso nessa primeira componente principal.

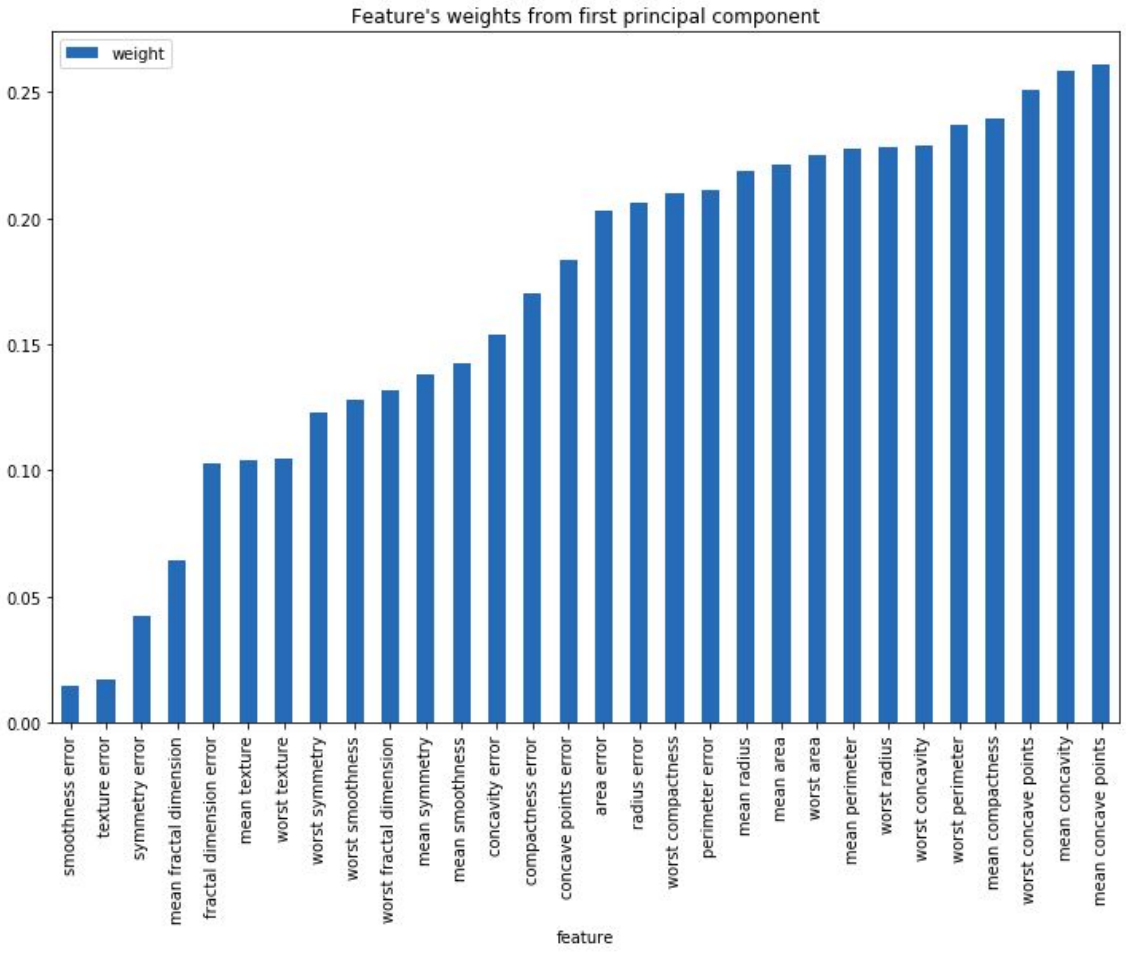
```
#Getting components
components = pca.components_
```

```
#constructing a dict with feature names and first principal components
feature_weights = dict(zip(features_names, components[0]))
#Sorting
sorted_weights = sorted(feature_weights.items(), key = lambda kv: kv[1])
#Lowest Weights
print('Lowest Weights: ')
for feature, weight, in sorted_weights[:4]:
    print('\t{:20} {:.3f}'.format(feature, weight))
#Highest Weights
print('Highest Weights: ')
for feature, weight in sorted_weights[-4:]:
    print('\t{:20} {:.3f}'.format(feature, weight))
```

Como resultado obtemos que as features que receberam peso maior são *mean compactness*, *worst concave points* .

```
Lowest Weigths:
        smoothness error     0.015
        texture error        0.017
        symmetry error       0.042
        mean fractal dimension 0.064
Highest Weights:
        mean compactness     0.239
        worst concave points 0.251
        mean concavity       0.258
        mean concave points  0.261
```

O gráfico abaixo mostra o gráfico construído dos pesos das features no primeiro componente principal:




O código é bem simples, já selecionamos os pesos das features na variável *sorted_weights*. Basta plotarmos esses valores:

```
sorted_weightsdata = []
for feature, weight, in sorted_weights:
    data.append([feature,weight])

df =  pd.DataFrame(data,columns=["feature","weight"])
df.set_index("feature",inplace=True)

df.plot(kind="bar",title="Feature's weights from first principal
component")
```

Por aqui terminamos a exposição do assunto. Nos próximos posts, iremos entrar em mais detalhes sobre redução de dimensionalidade. Você pode conferir o código na íntegra no repositório : [https://github.com/vgp314/4CodingBlog](https://github.com/vgp314/4CodingBlog)

# Referências

[1] sklearn.datasets.load_breast_cancer. (n.d.). Retrieved 7, 2020, from https://scikit-learn.org/stable/modules/generated/sklearn.datasets.load_breast_cancer.html

[2] Shalev-Shwartz, S., & Ben-David, S. (2014). Understanding Machine Learning: From Theory to Algorithms. Cambridge: Cambridge University Press, p. 323. doi:10.1017/CBO9781107298019

[3] Jolliffe I.T. Principal Component Analysis, Series: Springer Series in Statistics, 2nd ed., Springer, NY, 2002, XXIX, 487 p. 28 illus. ISBN 978-0-387-95442-4