

# Trabalho IV - Aprendizado de Máquina com microarrays

Leonardo Azzi Martins

INF05018 - Biologia Computacional

---

## Setup

```
In [ ]: # importando as bibliotecas necessárias
import pandas as pd # manipulação de datasets
import numpy as np  # operações matemáticas

# sklearn é uma biblioteca de data science e machine learning
from sklearn.model_selection import train_test_split
from sklearn import svm
from sklearn.metrics import confusion_matrix
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
import scipy
from sklearn.metrics import accuracy_score, f1_score
from sklearn.metrics import silhouette_score

# biblioteca para visualização
import matplotlib.pyplot as plt
# unused but required import for doing 3d projections with matplotlib < 3
import mpl_toolkits.mplot3d # noqa: F401
```

Definir o nome do arquivo de dados que usaremos

```
In [5]: file_name = 'Prostate_GSE6919_U95B.csv' # https://sbcinf.ufrgs.br/cumid
```

```
In [6]: data = pd.read_csv(file_name, delimiter=',', header=0, index_col=0)
data
```

Out[6]:

		type	41880_at	41881_at	41882_at	41883_at	41884_a
<b>samples</b>							
<b>GSM152992.CEL</b>	primary_prostate_tumor		2.414076	4.113824	2.035911	3.102248	2.11557
<b>GSM152993.CEL</b>	primary_prostate_tumor		2.385157	4.078664	2.123064	3.087631	2.25419
<b>GSM152994.CEL</b>	primary_prostate_tumor		2.295522	4.085505	2.144344	3.071539	2.22942
<b>GSM152995.CEL</b>	primary_prostate_tumor		2.260478	4.466391	2.206410	3.505265	2.60501
<b>GSM152996.CEL</b>	primary_prostate_tumor		2.229731	4.291435	2.506255	3.220628	2.40467
...	...		...	...	...	...	.
<b>GSM153233.CEL</b>	normal		2.305050	4.095951	1.892349	3.097484	2.23130
<b>GSM153234.CEL</b>	normal		2.405650	4.083099	1.828318	2.987580	2.15925
<b>GSM153235.CEL</b>	normal		2.456738	4.121602	1.972852	3.057921	2.11511
<b>GSM153236.CEL</b>	normal		2.306891	3.990915	2.018053	3.038294	2.33877
<b>GSM153237.CEL</b>	normal		2.499361	4.129915	2.016119	3.121170	2.34436

124 rows × 12621 columns

## EDA

In [7]: `data.describe()`

Out[7]:

	41880_at	41881_at	41882_at	41883_at	41884_at	41885_at	41886_r_i
<b>count</b>	124.000000	124.000000	124.000000	124.000000	124.000000	124.000000	124.00000
<b>mean</b>	2.502464	4.166458	1.994689	3.163871	2.254477	1.995747	5.89160
<b>std</b>	0.143348	0.130313	0.159770	0.160648	0.162345	0.143794	0.21678
<b>min</b>	2.185090	3.889521	1.633173	2.722558	1.943469	1.767546	5.36103
<b>25%</b>	2.416656	4.087583	1.881999	3.075074	2.160879	1.887146	5.74264
<b>50%</b>	2.504266	4.146306	1.962583	3.149837	2.230363	1.979415	5.87483
<b>75%</b>	2.594960	4.250945	2.105921	3.249132	2.340170	2.090395	5.98780
<b>max</b>	2.861795	4.613252	2.506255	3.688947	2.878543	2.472451	6.65674

8 rows × 12620 columns

```
In [8]: # para obtermos uma lista com os identificadores das classes:
classes = data['type'].unique()
classes
```

Out[8]: `array(['primary_prostate_tumor', 'normal'], dtype=object)`

```
In [9]: data.dtypes
```

```
Out[9]: type          object
41880_at          float64
41881_at          float64
41882_at          float64
41883_at          float64
...
AFFX-TrpnX-M_at   float64
AFFX-YEL002c/WBP1_at float64
AFFX-YEL018w/_at float64
AFFX-YEL021w/URA3_at float64
AFFX-YEL024w/RIP1_at float64
Length: 12621, dtype: object
```

## Pré-processamento

### Transforma type (categórico) em code (numérico)

```
In [10]: data.type = pd.Categorical(data.type)
data['code'] = data.type.cat.codes
```

```
In [11]: data[['type', 'code']]
```

```
Out[11]:
```

	type	code
<b>samples</b>		
<b>GSM152992.CEL</b>	primary_prostate_tumor	1
<b>GSM152993.CEL</b>	primary_prostate_tumor	1
<b>GSM152994.CEL</b>	primary_prostate_tumor	1
<b>GSM152995.CEL</b>	primary_prostate_tumor	1
<b>GSM152996.CEL</b>	primary_prostate_tumor	1
...	...	...
<b>GSM153233.CEL</b>	normal	0
<b>GSM153234.CEL</b>	normal	0
<b>GSM153235.CEL</b>	normal	0
<b>GSM153236.CEL</b>	normal	0
<b>GSM153237.CEL</b>	normal	0

124 rows × 2 columns

```
In [12]: # atualiza classes
classes = data['code'].unique()
classes
```

```
Out[12]: array([1, 0], dtype=int8)
```

```
In [13]: data = data.drop('type', axis=1)
data
```

Out[13]:

	41880_at	41881_at	41882_at	41883_at	41884_at	41885_at	41886_r_at
samples							
<b>GSM152992.CEL</b>	2.414076	4.113824	2.035911	3.102248	2.115578	1.775455	6.107839
<b>GSM152993.CEL</b>	2.385157	4.078664	2.123064	3.087631	2.254190	1.815183	5.708878
<b>GSM152994.CEL</b>	2.295522	4.085505	2.144344	3.071539	2.229422	1.985899	5.679248
<b>GSM152995.CEL</b>	2.260478	4.466391	2.206410	3.505265	2.605014	1.887307	5.935039
<b>GSM152996.CEL</b>	2.229731	4.291435	2.506255	3.220628	2.404673	1.886664	5.965917
...	...	...	...	...	...	...	...
<b>GSM153233.CEL</b>	2.305050	4.095951	1.892349	3.097484	2.231303	1.971964	5.598093
<b>GSM153234.CEL</b>	2.405650	4.083099	1.828318	2.987580	2.159251	2.086542	5.566386
<b>GSM153235.CEL</b>	2.456738	4.121602	1.972852	3.057921	2.115119	1.863530	5.886834
<b>GSM153236.CEL</b>	2.306891	3.990915	2.018053	3.038294	2.338770	1.925691	5.361032
<b>GSM153237.CEL</b>	2.499361	4.129915	2.016119	3.121170	2.344369	1.915144	5.909312

124 rows × 12621 columns

```
In [14]: # para obtermos a média do valor de expressão gênica de cada gene (coluna
avg = data.mean()
print(avg)
```

```
41880_at      2.502464
41881_at      4.166458
41882_at      1.994689
41883_at      3.163871
41884_at      2.254477
```

...

```
AFFX-YEL002c/WBP1_at  2.119095
AFFX-YEL018w/_at     2.129572
AFFX-YEL021w/URA3_at 3.391223
AFFX-YEL024w/RIP1_at 2.586230
code                 0.516129
```

Length: 12621, dtype: float64

Separa atributos em X e rótulos/classes em Y

```
In [15]: Y = data['code']
X = data.drop('code', axis=1)
print(X)
print(Y)
```

	41880_at	41881_at	41882_at	41883_at	41884_at	41885_at
\ samples						
GSM152992.CEL	2.414076	4.113824	2.035911	3.102248	2.115578	1.775455
GSM152993.CEL	2.385157	4.078664	2.123064	3.087631	2.254190	1.815183
GSM152994.CEL	2.295522	4.085505	2.144344	3.071539	2.229422	1.985899
GSM152995.CEL	2.260478	4.466391	2.206410	3.505265	2.605014	1.887307
GSM152996.CEL	2.229731	4.291435	2.506255	3.220628	2.404673	1.886664
...	...	...	...	...	...	...
GSM153233.CEL	2.305050	4.095951	1.892349	3.097484	2.231303	1.971964
GSM153234.CEL	2.405650	4.083099	1.828318	2.987580	2.159251	2.086542
GSM153235.CEL	2.456738	4.121602	1.972852	3.057921	2.115119	1.863530
GSM153236.CEL	2.306891	3.990915	2.018053	3.038294	2.338770	1.925691
GSM153237.CEL	2.499361	4.129915	2.016119	3.121170	2.344369	1.915144

	41886_r_at	41887_at	41888_at	41889_at	...	AFFX-ThrX-3_
at \ samples						
GSM152992.CEL	6.107839	2.160168	3.363597	3.155905	...	2.9559
98						
GSM152993.CEL	5.708878	2.134447	3.336872	2.863654	...	3.1965
21						
GSM152994.CEL	5.679248	2.100443	3.356141	2.819615	...	2.9299
04						
GSM152995.CEL	5.935039	2.261295	3.754584	2.994412	...	3.5785
38						
GSM152996.CEL	5.965917	2.274317	3.450538	3.309058	...	3.5581
84						
...	...	...	...	...	...	.
..						
GSM153233.CEL	5.598093	2.092084	3.187975	3.038651	...	3.5212
92						
GSM153234.CEL	5.566386	1.968839	3.226227	2.904245	...	3.3750
90						
GSM153235.CEL	5.886834	2.105358	3.249482	2.964812	...	3.3389
30						
GSM153236.CEL	5.361032	2.245813	3.085293	2.903036	...	3.1872
67						
GSM153237.CEL	5.909312	1.967527	3.300792	3.110052	...	3.4805
23						

	AFFX-ThrX-5_at	AFFX-ThrX-M_at	AFFX-TrpnX-3_at	\
samples				
GSM152992.CEL	2.910953	2.095267	1.617076	
GSM152993.CEL	2.975412	2.249950	1.757867	
GSM152994.CEL	2.857025	2.047436	1.625339	
GSM152995.CEL	3.420946	2.736342	1.940826	
GSM152996.CEL	3.083316	2.562048	1.923414	
...	...	...	...	
GSM153233.CEL	3.091387	2.234156	1.774587	
GSM153234.CEL	3.122058	2.277573	1.922329	
GSM153235.CEL	3.205037	2.151657	1.891080	
GSM153236.CEL	2.886123	2.118569	1.883039	
GSM153237.CEL	3.279474	2.370650	2.051492	

	AFFX-TrpnX-5_at	AFFX-TrpnX-M_at	AFFX-YEL002c/WBP1_at	\
samples				
GSM152992.CEL	2.060144	1.962483	1.919590	
GSM152993.CEL	2.352185	2.054990	1.970140	
GSM152994.CEL	2.065674	1.955286	1.768858	

GSM152995.CEL	2.713500	2.363843	2.194837
GSM152996.CEL	2.775842	2.323899	2.288732
...	...	...	...
GSM153233.CEL	2.496925	2.340286	2.058535
GSM153234.CEL	2.475973	2.177743	1.813374
GSM153235.CEL	2.438726	2.292233	2.253969
GSM153236.CEL	2.223137	2.369814	2.282727
GSM153237.CEL	2.706354	2.651999	2.199191

	AFFX-YEL018w/_at	AFFX-YEL021w/URA3_at	AFFX-YEL024w/RIP1_a
t			
samples			
GSM152992.CEL	2.034229	3.013061	2.20842
1			
GSM152993.CEL	1.768459	4.701207	2.51356
0			
GSM152994.CEL	1.666836	3.115166	2.13676
0			
GSM152995.CEL	1.981020	3.141350	2.55004
1			
GSM152996.CEL	2.772796	2.848256	2.80326
4			
...	...	...	..
.			
GSM153233.CEL	2.146309	3.093957	2.44065
4			
GSM153234.CEL	1.917204	2.395584	2.10060
5			
GSM153235.CEL	2.145971	4.231840	2.80662
1			
GSM153236.CEL	2.127103	2.570136	2.46401
4			
GSM153237.CEL	2.127596	3.211866	2.69300
6			

[124 rows x 12620 columns]

samples

GSM152992.CEL	1
GSM152993.CEL	1
GSM152994.CEL	1
GSM152995.CEL	1
GSM152996.CEL	1

..

GSM153233.CEL	0
GSM153234.CEL	0
GSM153235.CEL	0
GSM153236.CEL	0
GSM153237.CEL	0

Name: code, Length: 124, dtype: int8

## a. Normalização com z-score

a. Dados brutos não são a melhor opção para descobertas biológicas devido à presença de ruído e variações entre as diferentes tecnologias envolvidas. Por conta disso, se faz necessária a aplicação de técnicas de Correção de background; Transformação de dados; Normalização de dados; e Validação estatística. A normalização dos dados consiste em transformar os valores obtidos de modo que as

variações de um experimento sejam reduzidas permitindo que duas amostras sejam apropriadamente comparadas. Para certas tarefas, como classificação e clustering, às vezes é necessário reescalar ou normalizar os dados para que todas as colunas tenham o mesmo "peso" ou certos intervalos de valores sejam obedecidos. Realize a normalização do dataset de estudo utilizando o z-score:  $z = (x - \mu) / \sigma$  ( $x$  = valor observado;  $\mu$  = valor médio da variável;  $\sigma$  = desvio padrão da variável). Para este item é necessário apresentar o código fonte utilizado para normalizar os dados;

```
In [16]: X = X.apply(scipy.stats.zscore)
X
```

Out[16]:

	41880_at	41881_at	41882_at	41883_at	41884_at	41885_at	41886_r_
samples							
<b>GSM152992.CEL</b>	-0.619096	-0.405540	0.259055	-0.385145	-0.859048	-1.538211	1.00151
<b>GSM152993.CEL</b>	-0.821660	-0.676450	0.806757	-0.476498	-0.001771	-1.260806	-0.8463
<b>GSM152994.CEL</b>	-1.449493	-0.623741	0.940487	-0.577074	-0.154954	-0.068768	-0.9835
<b>GSM152995.CEL</b>	-1.694949	2.310967	1.330536	2.133727	2.167972	-0.757192	0.2011
<b>GSM152996.CEL</b>	-1.910314	0.962945	3.214879	0.354737	0.928922	-0.761683	0.3441
...	...	...	...	...	...	...	...
<b>GSM153233.CEL</b>	-1.382754	-0.543249	-0.643144	-0.414918	-0.143320	-0.166066	-1.3594
<b>GSM153234.CEL</b>	-0.678116	-0.642279	-1.045538	-1.101820	-0.588946	0.633981	-1.5062
<b>GSM153235.CEL</b>	-0.320278	-0.345616	-0.137230	-0.662188	-0.861890	-0.923216	-0.0221
<b>GSM153236.CEL</b>	-1.369861	-1.352545	0.146827	-0.784859	0.521330	-0.489173	-2.4574
<b>GSM153237.CEL</b>	-0.021730	-0.281559	0.134672	-0.266883	0.555959	-0.562816	0.0820

124 rows × 12620 columns

## b. Divisão de dados

b. Para avaliar um classificador, é necessário dividir os dados em pelo menos 2 conjuntos: conjunto de treinamento e conjunto de teste. Uma divisão comum seria 70% treinamento, 30% teste. Os dados no conjunto de teste devem sempre estar ocultos do classificador durante o treinamento. O ideal nesta divisão é que ocorra uma amostragem estratificada, isto é, que a proporção entre classes seja a mesma em todos os subconjuntos. Realize a divisão do dataset de estudo (resultante da letra 'a') em conjunto de teste e conjunto de treinamento de forma a contemplar o conceito de amostragem estratificada. Para este item é necessário apresentar o código fonte utilizado para divisão dos dados em conjunto de treinamento e conjunto de teste.

```
In [17]: X_train, X_test, Y_train, Y_test = train_test_split(
        X, Y, test_size=0.3, random_state=42, stratify=Y
    )
```

```
print("Tamanho do conjunto de treino:", X_train.shape)
print("Tamanho do conjunto de teste:", X_test.shape)
```

Tamanho do conjunto de treino: (86, 12620)

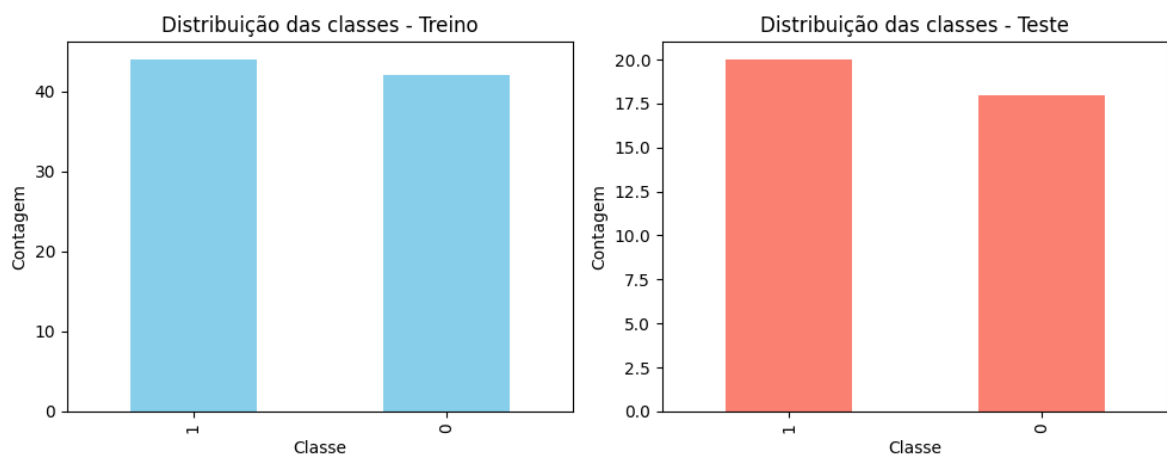
Tamanho do conjunto de teste: (38, 12620)

```
In [18]: fig, axes = plt.subplots(1, 2, figsize=(10, 4))

# treino
Y_train.value_counts().plot(kind='bar', ax=axes[0], color='skyblue')
axes[0].set_title('Distribuição das classes - Treino')
axes[0].set_xlabel('Classe')
axes[0].set_ylabel('Contagem')

# teste
Y_test.value_counts().plot(kind='bar', ax=axes[1], color='salmon')
axes[1].set_title('Distribuição das classes - Teste')
axes[1].set_xlabel('Classe')
axes[1].set_ylabel('Contagem')

plt.tight_layout()
plt.show()
```



## c. Classificação com SVM

c. Em tarefas de classificação, o objetivo é mapear um padrão de entrada para uma saída específica. Existem muitas opções de classificadores e cada uma tem suas particularidades. Support Vector Machine (SVM) é um dos algoritmos com melhor desempenho na classificação de dados de microarranjo segundo a literatura. Fazendo uso de bibliotecas (por exemplo, scikit-learn <https://scikit-learn.org/stable>) crie um classificador SVM para o dataset de estudo. O treinamento do classificador deve ser realizada com base no grupo de treinamento criado no item 'b'. Para este item é necessário apresentar o código fonte utilizado para a criação do classificador.

```
In [19]: # https://scikit-learn.org/stable/modules/svm.html
clf = svm.SVC()
clf.fit(X, Y)

prediction = clf.predict(X)
print(prediction) # mostra as classes preditas para X pelo modelo em clf
```

[illegible]

d. Métricas em Aprendizado de Máquina estão relacionadas aos aspectos de uma predição. Algumas métricas podem indicar propriedades discriminativas, enquanto outras se relacionam com habilidade preditiva. Nem todas são adequadas para diagnóstico devido ao desequilíbrio entre classes. A partir criado na letra 'c' e do conjunto de testes, avalie o classificador considerando as seguintes métricas: (i) a matriz de confusão; (ii) a acurácia; (iii) Sensitivity; (iv) Specificity; e (v) F1-score. O valor destas métricas deve ser reportado. Ao analisar as métricas você considera que o classificador teve um desempenho adequado?

```
In [20]: # (i) Matriz de confusão
fig, ax = plt.subplots()
im = ax.imshow(cm, cmap='Blues')

ax.set_xlabel('Classe Predita')
ax.set_ylabel('Classe Real')
ax.set_title('Matriz de Confusão')
ax.set_xticks(np.arange(len(classes)))
ax.set_yticks(np.arange(len(classes)))
ax.set_xticklabels(classes)
ax.set_yticklabels(classes)

for i in range(cm.shape[0]): # Adicionar valores nas células
    for j in range(cm.shape[1]):
        ax.text(j, i, cm[i, j], ha='center', va='center', color='black')

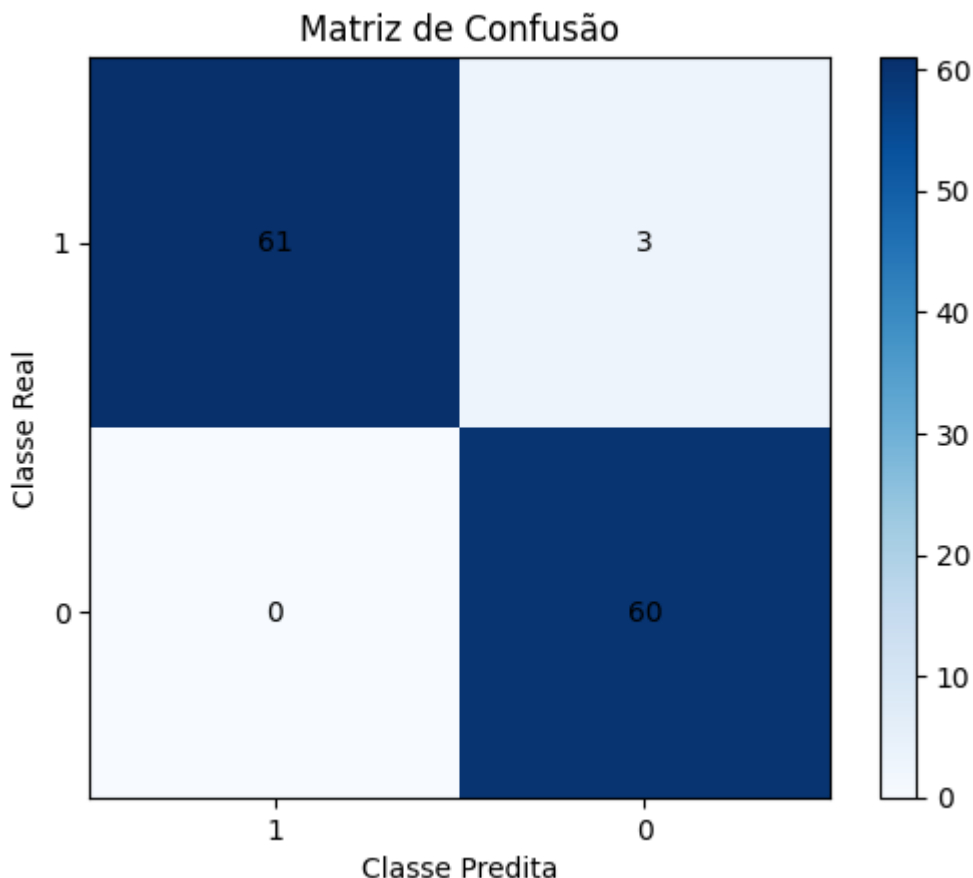
plt.colorbar(im)
plt.show()

# (ii) Acurácia
accuracy = accuracy_score(Y, prediction)
print("Acurácia:", accuracy)

# (iii) Sensitivity
TP = cm[1, 1]
FN = cm[1, 0]
sensitivity = TP / (TP + FN)
print("Sensitivity (Recall):", sensitivity)

# (iv) Specificity
TN = cm[0, 0]
FP = cm[0, 1]
specificity = TN / (TN + FP)
print("Specificity:", specificity)

# (v) F1-score (para classe positiva)
f1 = f1_score(Y, prediction, pos_label=1)
print("F1-score:", f1)
```



Acurácia: 0.9758064516129032

Sensitivity (Recall): 1.0

Specificity: 0.953125

F1-score: 0.976

**Ao analisar as métricas você considera que o classificador teve um desempenho adequado?**

R: Sim, pois as métricas são bastante robustas. O único ponto de atenção, dado o contexto de predição para câncer de próstata, é que falso-negativos podem ser "mais graves" do que falso-positivos, pois pode resultar em uma sub-notificação de uma doença que precisa de tratamento rápido e intensivo.

## e. Análise com K-means

e. Técnicas de clusterização podem ser utilizadas para descobrir novos grupos de amostras a partir do conjunto de dados. k-means é um método de clustering que objetiva particionar n observações dentre k grupos onde cada observação pertence ao grupo mais próximo da média. Fazendo uso de bibliotecas (por exemplo, scikit-learn <https://scikit-learn.org/stable>) utilize o método k-means para analisar o dataset de estudo (preparado no item 'a') considerando os seguintes cenários: existência de 2 grupos; 3 grupos e 4 grupos. Para cada um dos cenários reporte o número de amostras presentes de cada grupo.

```
In [29]: grupos = [2, 3, 4]

for n_grupos in grupos:
```

```

print(f"K-means com {n_grupos} grupos\n=====
kmeans = KMeans(n_clusters=n_grupos, random_state=0).fit(X)
clusters = kmeans.predict(X)
print(clusters)

# Silhouette Score (quanto maior, melhor a separação dos clusters; va
sil_score = silhouette_score(X, clusters)
print(f"Silhouette Score ({n_grupos} clusters): {sil_score:.3f}")

# Contagem de amostras por cluster
unique, counts = np.unique(clusters, return_counts=True)
for cluster_id, count in zip(unique, counts):
    print(f"Cluster {cluster_id}: {count} amostras")

# Visualização dos clusters usando PCA
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=clusters, cmap='vir
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title(f'Clusters encontrados pelo K-means ({2} clusters)')
plt.legend(*scatter.legend_elements(), title="Cluster")
plt.show()

```

K-means com 2 grupos

=====

```

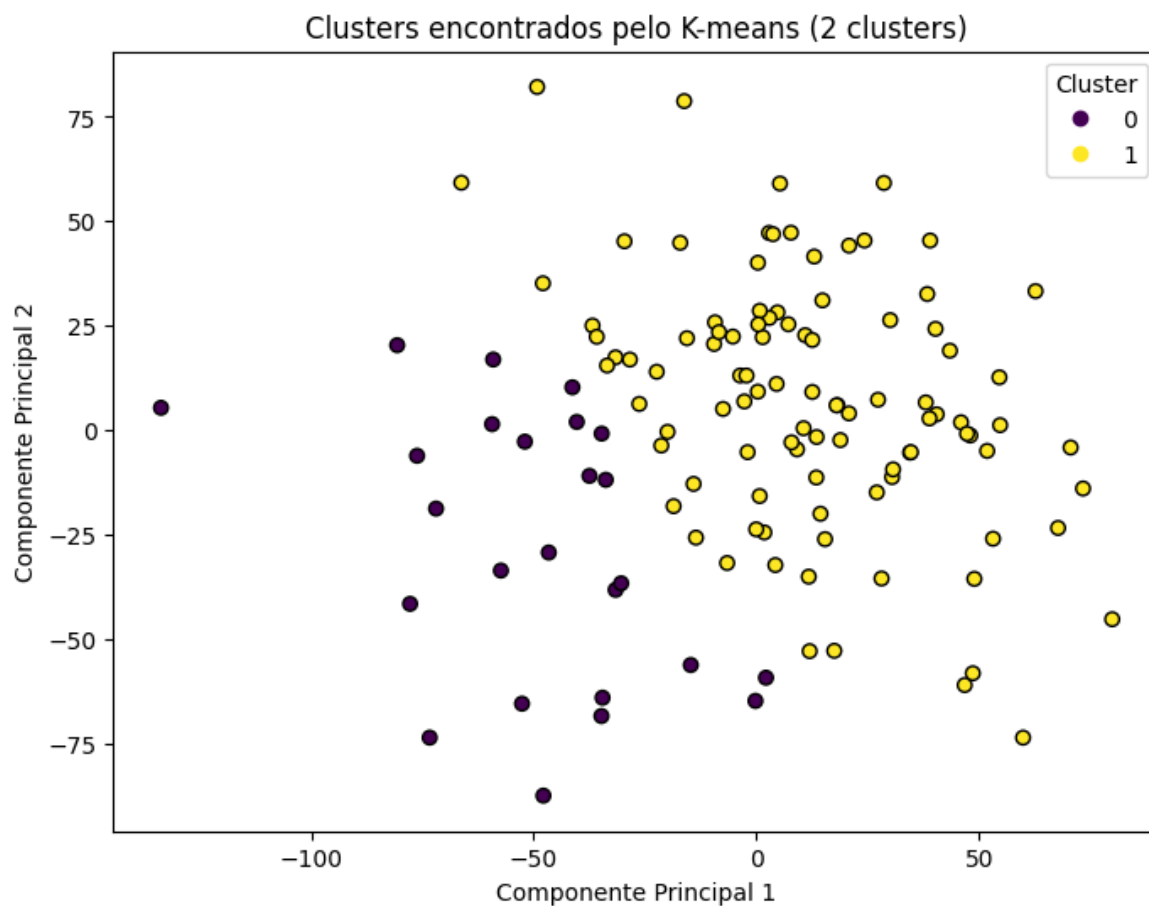
[1 1 1 0 0 0 0 0 0 1 0 1 1 0 1 1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 0 1 1 1 0 1 1 0 1 1 1 1 1 0 0 1 1 1 1 1 1 1 0 0 1 1 1 1
 1 1 1 0 1 1 1 1 1 1 1 1 0 0 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1]

```

Silhouette Score (2 clusters): 0.085

Cluster 0: 25 amostras

Cluster 1: 99 amostras



K-means com 3 grupos

=====

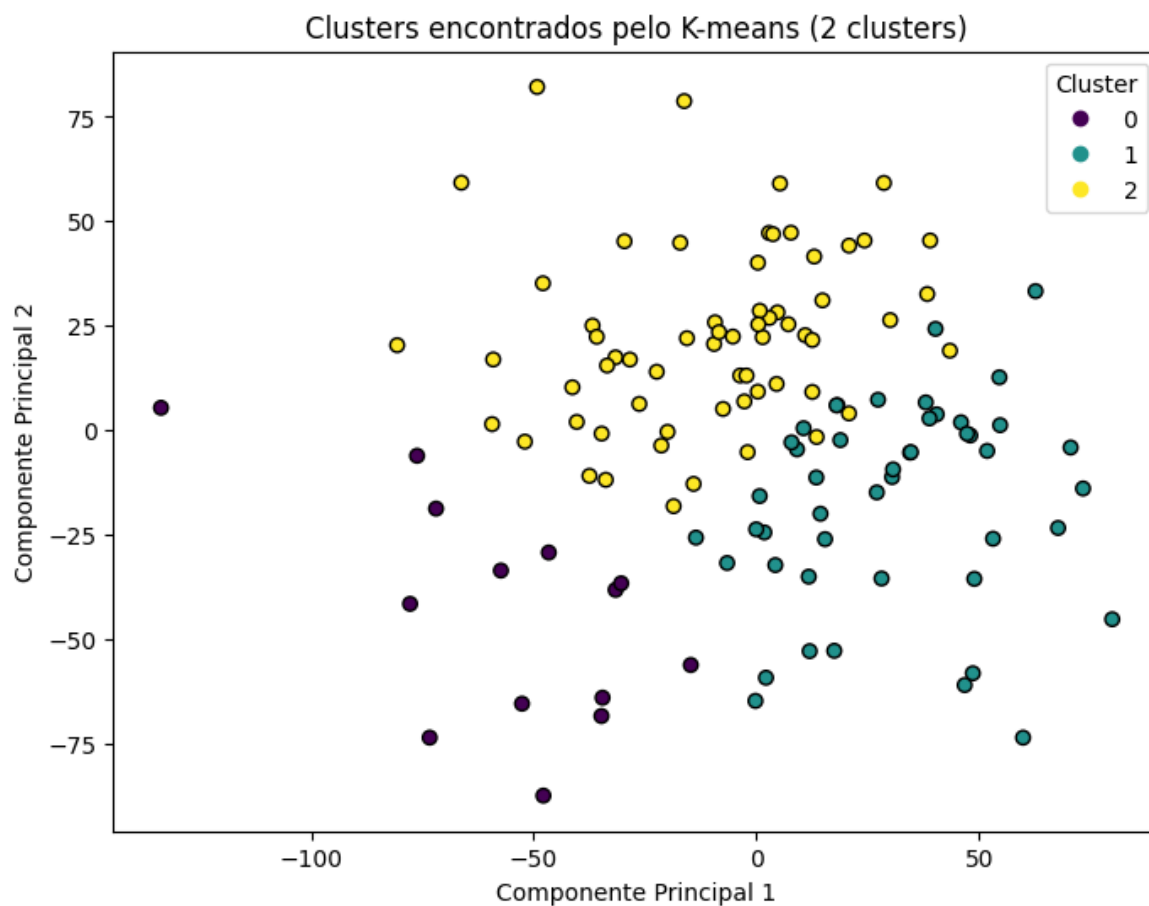
```
[1 1 1 0 0 0 0 0 0 1 2 0 2 1 0 1 1 1 0 1 2 2 2 2 0 2 2 1 1 2 2 2 2 1 1 2 2
 1 2 2 2 2 1 2 2 1 2 2 1 2 2 2 2 2 2 2 1 1 2 2 0 2 2 2 1 1 2 2 0 0 2 1 1 1
 2 2 2 0 1 1 2 2 1 1 2 2 2 2 1 1 2 1 1 1 1 2 2 2 2 2 1 1 1 2 2 2 2 2 2 1 2
 2 1 2 1 1 1 2 2 1 1 1 1 1]
```

Silhouette Score (3 clusters): 0.058

Cluster 0: 14 amostras

Cluster 1: 47 amostras

Cluster 2: 63 amostras



K-means com 4 grupos

=====

=====

```
[3 3 3 0 0 0 0 0 0 0 2 0 3 3 1 3 3 0 0 3 2 2 2 1 1 1 1 3 3 2 1 2 2 3 3 2 2
 3 2 2 2 2 3 2 2 3 1 1 3 1 1 1 1 1 1 3 3 1 1 1 1 1 1 3 3 1 1 0 0 1 3 3 3
 1 1 1 0 3 3 2 2 3 3 2 2 1 1 1 3 2 3 3 3 3 3 2 2 2 2 3 3 3 2 2 2 2 2 2 3 1
 2 3 1 3 3 3 1 1 3 3 3 3 3]
```

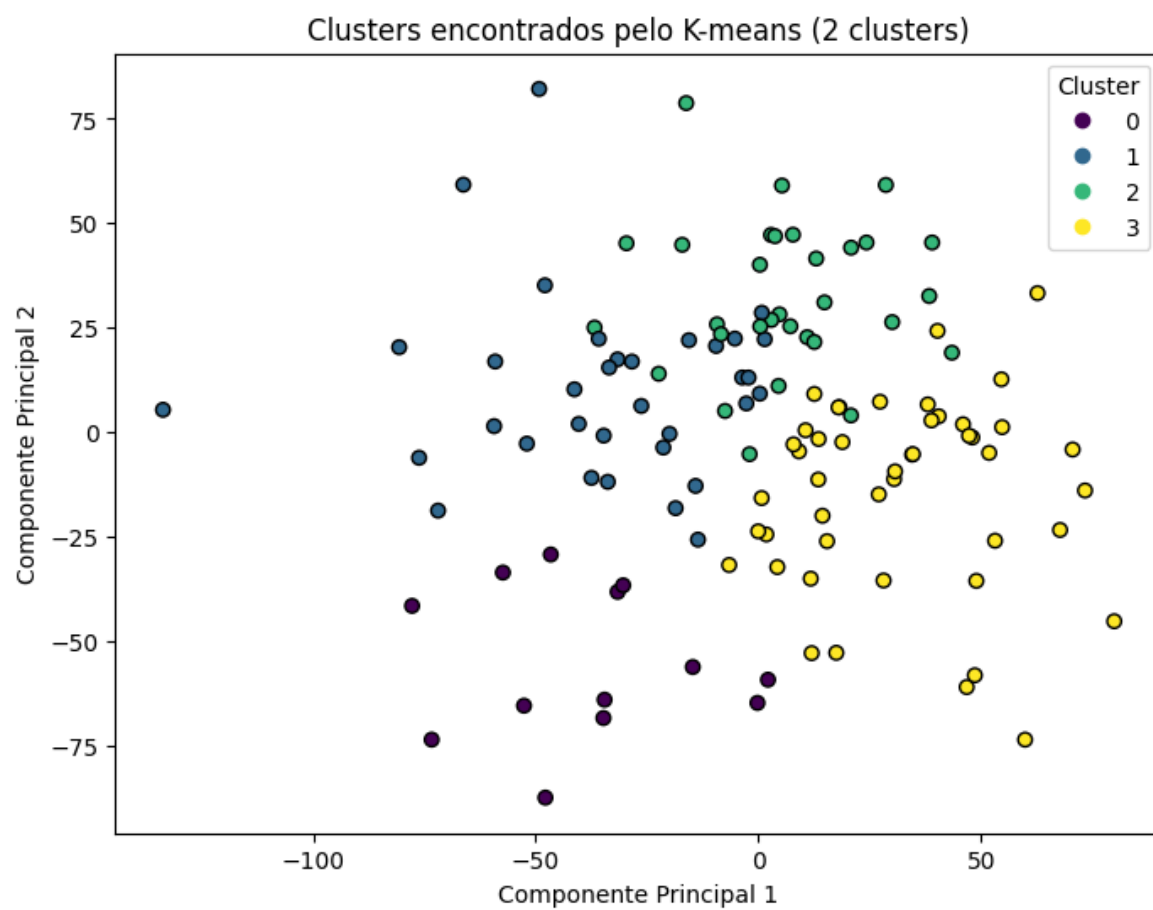
Silhouette Score (4 clusters): 0.061

Cluster 0: 13 amostras

Cluster 1: 34 amostras

Cluster 2: 31 amostras

Cluster 3: 46 amostras



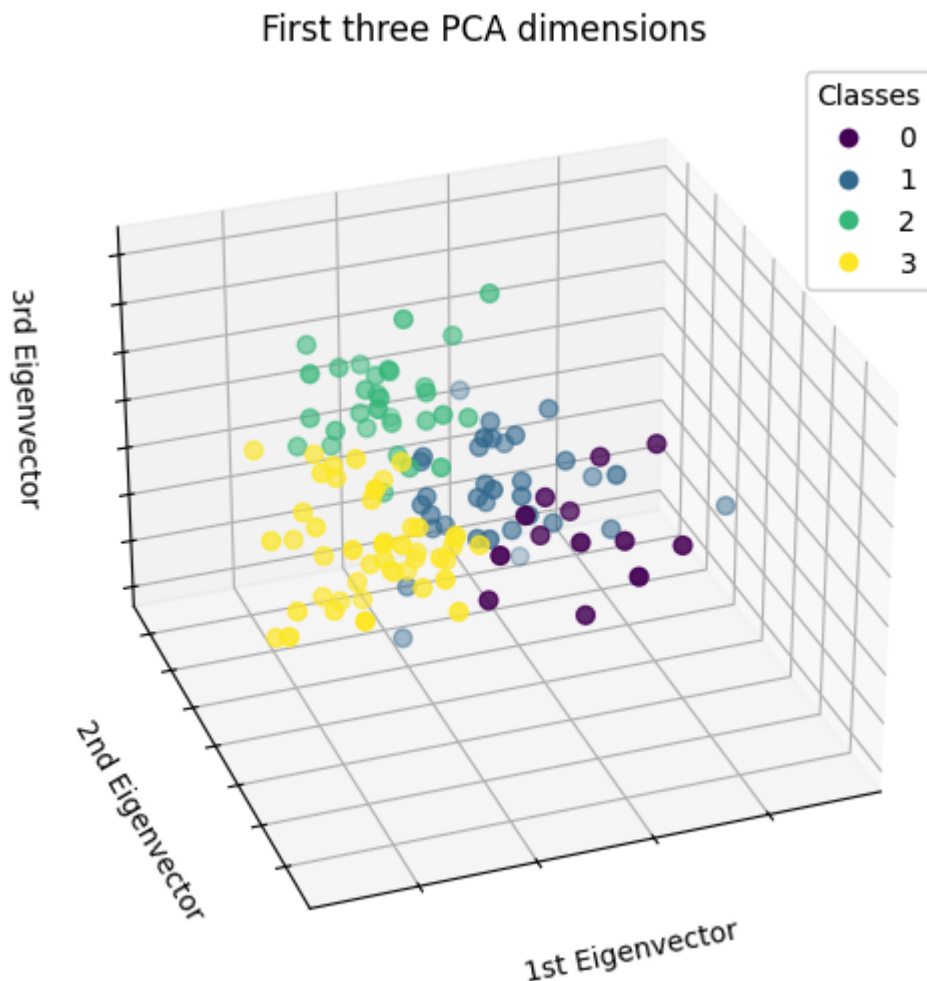
```
In [ ]: fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azimuth=110)

X_reduced = PCA(n_components=3).fit_transform(X)
scatter = ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=clusters,
    s=40,
)

ax.set(
    title="First three PCA dimensions",
    xlabel="1st Eigenvector",
    ylabel="2nd Eigenvector",
    zlabel="3rd Eigenvector",
)
ax.xaxis.set_ticklabels([])
ax.yaxis.set_ticklabels([])
ax.zaxis.set_ticklabels([])

# Add a legend
legend1 = ax.legend(
    scatter.legend_elements()[0],
    [0, 1, 2, 3],
    loc="upper right",
    title="Classes",
)
ax.add_artist(legend1)

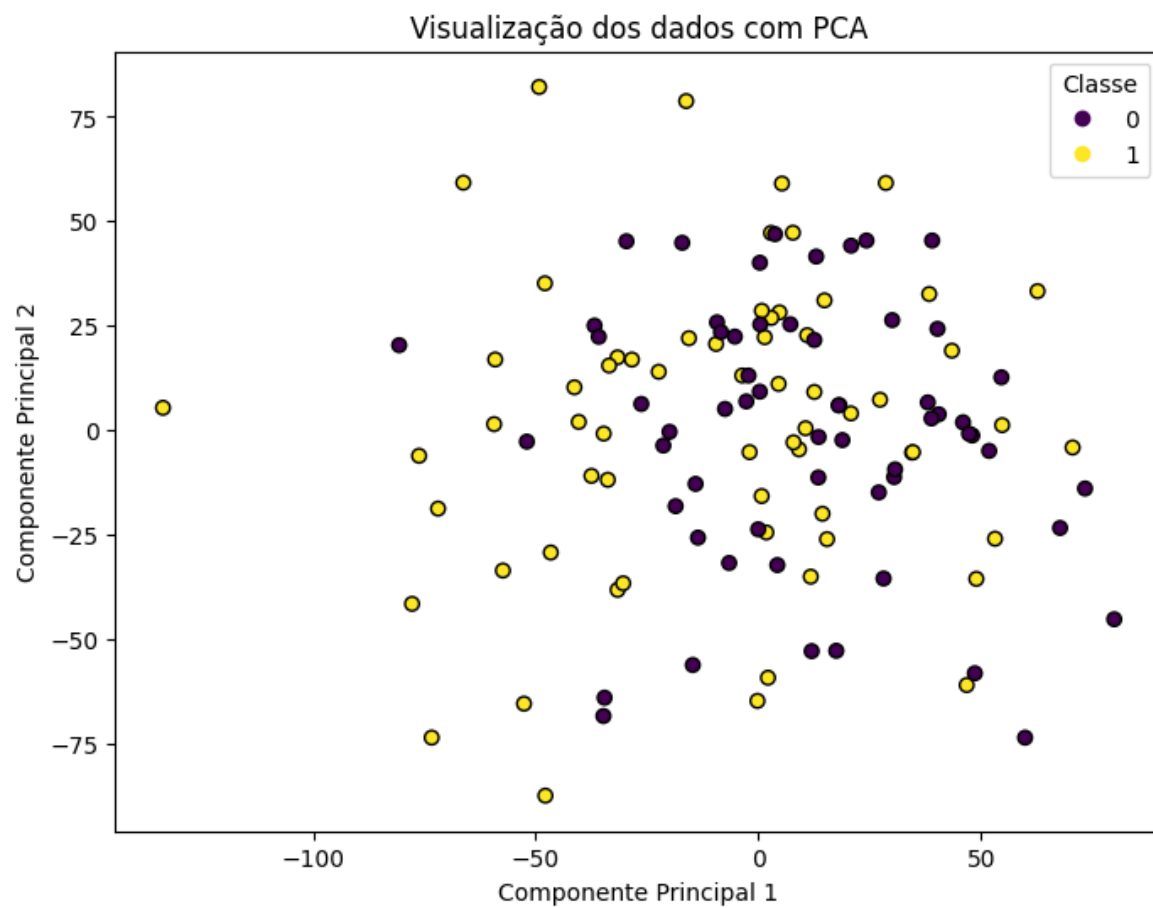
plt.show()
```



## Visualizando o PCA para os rótulos em Y

```
In [ ]: # Reduzindo para 2 componentes principais
pca = PCA(n_components=2)
X_pca = pca.fit_transform(X)

plt.figure(figsize=(8, 6))
scatter = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=Y, cmap='viridis', edge
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Visualização dos dados com PCA')
plt.legend(*scatter.legend_elements(), title="Classe")
plt.show()
```



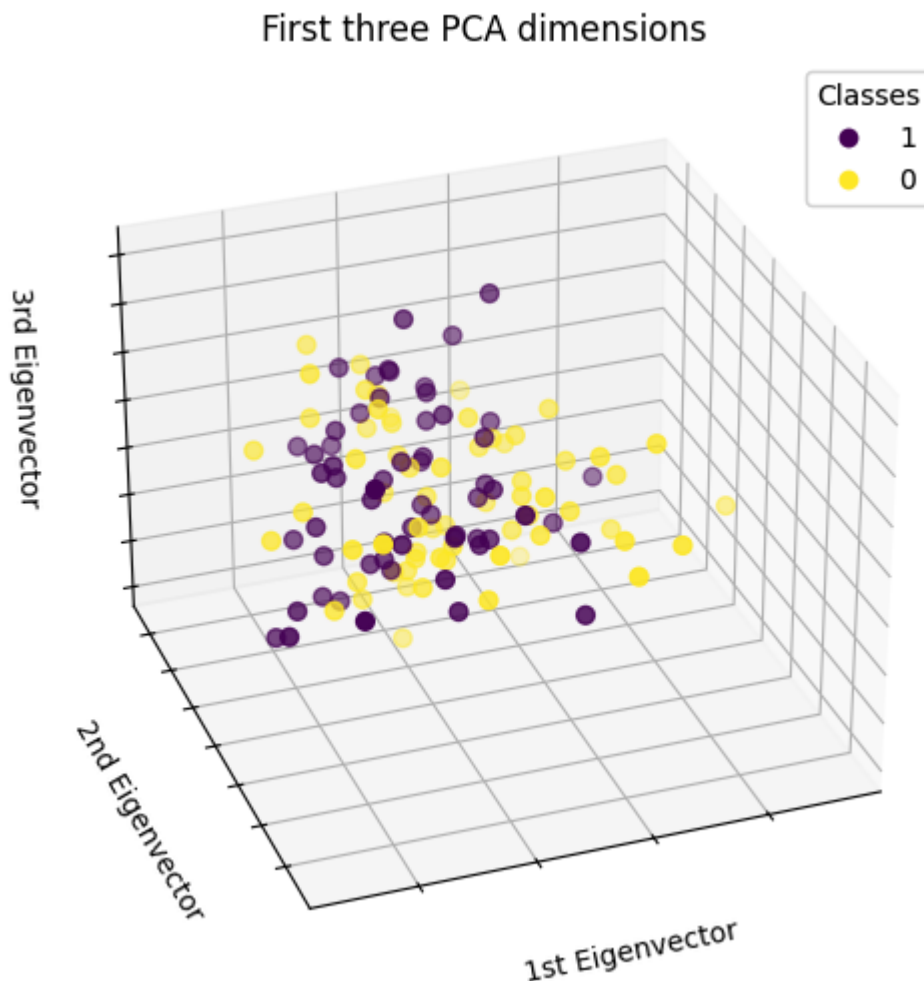
```
In [ ]: fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azimuth=110)

X_reduced = PCA(n_components=3).fit_transform(X)
scatter = ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=Y,
    s=40,
)

ax.set(
    title="First three PCA dimensions",
    xlabel="1st Eigenvector",
    ylabel="2nd Eigenvector",
    zlabel="3rd Eigenvector",
)
ax.xaxis.set_ticklabels([])
ax.yaxis.set_ticklabels([])
ax.zaxis.set_ticklabels([])

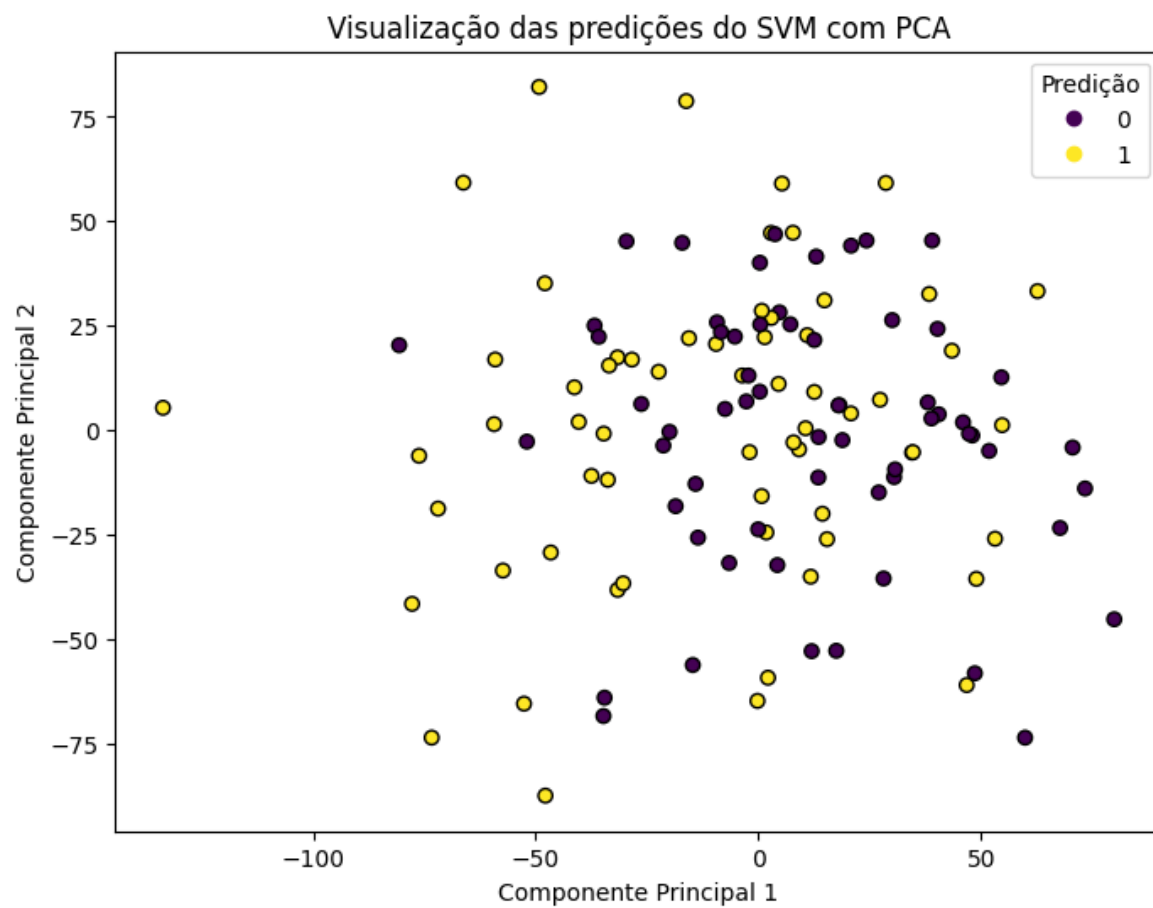
# Add a legend
legend1 = ax.legend(
    scatter.legend_elements()[0],
    classes,
    loc="upper right",
    title="Classes",
)
ax.add_artist(legend1)

plt.show()
```



## Visualizando o PCA para os rótulos Y\_pred

```
In [ ]: # Visualização das previsões do classificador SVM usando PCA
plt.figure(figsize=(8, 6))
scatter_pred = plt.scatter(X_pca[:, 0], X_pca[:, 1], c=prediction, cmap='
plt.xlabel('Componente Principal 1')
plt.ylabel('Componente Principal 2')
plt.title('Visualização das previsões do SVM com PCA')
plt.legend(*scatter_pred.legend_elements(), title="Predição")
plt.show()
```



```
In [ ]: fig = plt.figure(1, figsize=(8, 6))
ax = fig.add_subplot(111, projection="3d", elev=-150, azimuth=110)

X_reduced = PCA(n_components=3).fit_transform(X)
scatter = ax.scatter(
    X_reduced[:, 0],
    X_reduced[:, 1],
    X_reduced[:, 2],
    c=prediction,
    s=40,
)

ax.set(
    title="First three PCA dimensions",
    xlabel="1st Eigenvector",
    ylabel="2nd Eigenvector",
    zlabel="3rd Eigenvector",
)
ax.xaxis.set_ticklabels([])
ax.yaxis.set_ticklabels([])
ax.zaxis.set_ticklabels([])

# Add a legend
legend1 = ax.legend(
    scatter.legend_elements()[0],
    classes,
    loc="upper right",
    title="Classes",
)
ax.add_artist(legend1)

plt.show()
```

