Company Bankruptcy

June 6, 2025

1 1. Prevendo Falência de Empresas

Esse projeto usa os algoritmos de classificação apresentados no livro **An Introduction to Statistical Learning** para prever falência de companhias usando a base Company Bankruptcy Prediction do Kaggle. Os seguintes algoritmos foram utilizados:

1. Regressão Logística
2. KNN
3. Naive Bayes
4. Árvores de Decisão
5. Bagging
6. Random Forest
7. Boosting
8. Support Vector Machines
9. Linear Discriminant Analysis
10. Quadratic Discriminant Analysis

A base de dados utilizada possui 96 variáveis que podem ser utilizadas como preditores, sendo que várias possuem pouca variação, ou seja, há um problema de multicolinearidade. Portanto, com tantas variáveis, alguns algoritmos apresentam uma performance ruim quando avaliadas a acurácia e a AUC da curva ROC pela comparação com o conjunto de teste, como o Naive Bayes e o QDA. Sendo assim, antes de que os algoritmos fossem treinados e comparados com um conjunto de teste, os dados foram ajustados para que os seis primeiros componentes principais fossem usados como preditores para o treinamento dos algoritmos. Esse ajuste melhora consideravelmente a performance de todos os algoritmos.

Links:

An Introduction to Statistical Learning

Dados

2 2. Importando os pacotes e baixando os dados

[1]:
```
# importando alguns pacotes que serão utilizados em todas as etapas

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
```

```python
import numpy as np
from sklearn.metrics import confusion_matrix
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import RocCurveDisplay
from sklearn.metrics import roc_auc_score
```

[2]:
```python
# baixando os dados

import kagglehub

# Download latest version
path = kagglehub.dataset_download("fedesoriano/company-bankruptcy-prediction")
```

[3]:
```python
df = pd.read_csv(path + '\\data.csv')
```

[4]:
```python
df.describe()
```

[4]:
```
         Bankrupt?  ROA(C) before interest and depreciation before interest  \
count  6819.000000                                        6819.000000
mean      0.032263                                           0.505180
std       0.176710                                           0.060686
min       0.000000                                           0.000000
25%       0.000000                                           0.476527
50%       0.000000                                           0.502706
75%       0.000000                                           0.535563
max       1.000000                                           1.000000

        ROA(A) before interest and % after tax  \
count                              6819.000000
mean                                  0.558625
std                                   0.065620
min                                   0.000000
25%                                   0.535543
50%                                   0.559802
75%                                   0.589157
max                                   1.000000

        ROA(B) before interest and depreciation after tax  \
count                                        6819.000000
mean                                            0.553589
std                                             0.061595
min                                             0.000000
25%                                             0.527277
50%                                             0.552278
75%                                             0.584105
max                                             1.000000
```

	Operating Gross Margin	Realized Sales Gross Margin	\
count	6819.000000	6819.000000	
mean	0.607948	0.607929	
std	0.016934	0.016916	
min	0.000000	0.000000	
25%	0.600445	0.600434	
50%	0.605997	0.605976	
75%	0.613914	0.613842	
max	1.000000	1.000000	

	Operating Profit Rate	Pre-tax net Interest Rate	\
count	6819.000000	6819.000000	
mean	0.998755	0.797190	
std	0.013010	0.012869	
min	0.000000	0.000000	
25%	0.998969	0.797386	
50%	0.999022	0.797464	
75%	0.999095	0.797579	
max	1.000000	1.000000	

	After-tax net Interest Rate	\
count	6819.000000	
mean	0.809084	
std	0.013601	
min	0.000000	
25%	0.809312	
50%	0.809375	
75%	0.809469	
max	1.000000	

	Non-industry income and expenditure/revenue	…	\
count	6819.000000	…	
mean	0.303623	…	
std	0.011163	…	
min	0.000000	…	
25%	0.303466	…	
50%	0.303525	…	
75%	0.303585	…	
max	1.000000	…	

	Net Income to Total Assets	Total assets to GNP price	\
count	6819.000000	6.819000e+03	
mean	0.807760	1.862942e+07	
std	0.040332	3.764501e+08	
min	0.000000	0.000000e+00	
25%	0.796750	9.036205e-04	

50%	0.810619	2.085213e-03
75%	0.826455	5.269777e-03
max	1.000000	9.820000e+09

	No-credit Interval	Gross Profit to Sales \
count	6819.000000	6819.000000
mean	0.623915	0.607946
std	0.012290	0.016934
min	0.000000	0.000000
25%	0.623636	0.600443
50%	0.623879	0.605998
75%	0.624168	0.613913
max	1.000000	1.000000

	Net Income to Stockholder's Equity	Liability to Equity \
count	6819.000000	6819.000000
mean	0.840402	0.280365
std	0.014523	0.014463
min	0.000000	0.000000
25%	0.840115	0.276944
50%	0.841179	0.278778
75%	0.842357	0.281449
max	1.000000	1.000000

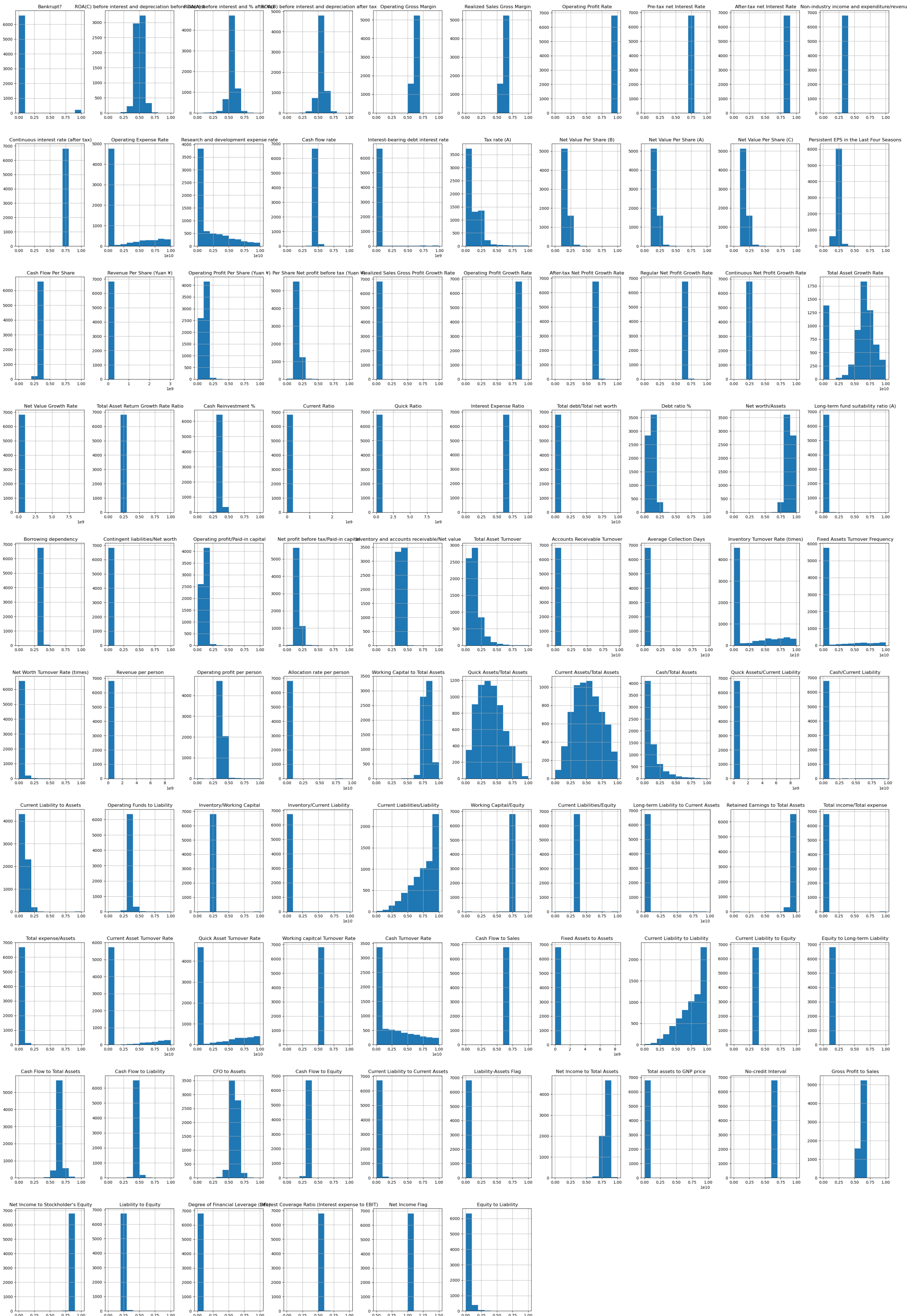
	Degree of Financial Leverage (DFL) \
count	6819.000000
mean	0.027541
std	0.015668
min	0.000000
25%	0.026791
50%	0.026808
75%	0.026913
max	1.000000

	Interest Coverage Ratio (Interest expense to EBIT)	Net Income Flag \
count	6819.000000	6819.0
mean	0.565358	1.0
std	0.013214	0.0
min	0.000000	1.0
25%	0.565158	1.0
50%	0.565252	1.0
75%	0.565725	1.0
max	1.000000	1.0

	Equity to Liability
count	6819.000000
mean	0.047578

```
std                 0.050014
min                 0.000000
25%                 0.024477
50%                 0.033798
75%                 0.052838
max                 1.000000

[8 rows x 96 columns]
```

[5]:
```
df.hist(figsize = (40,60))
plt.show()
```

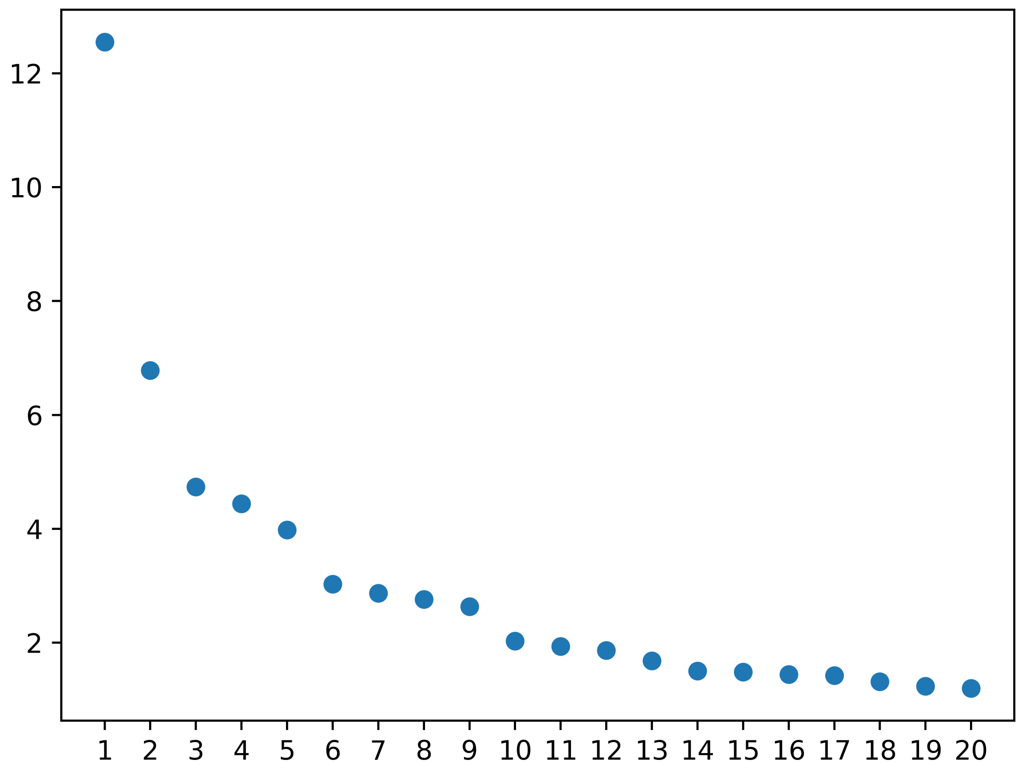

Bankrupt?
ROA(C) before interest and depreciation before interest
ROA(A) before interest and % after tax
ROA(B) before interest and depreciation after tax
Operating Gross Margin
Realized Sales Gross Margin
Operating Profit Rate
Pre-tax net Interest Rate
After-tax net Interest Rate
Non-industry income and expenditure/revenue
Continuous interest rate (after tax)
Operating Expense Rate
Research and development expense rate
Cash flow rate
Interest-bearing debt interest rate
Tax rate (A)
Net Value Per Share (B)
Net Value Per Share (A)
Net Value Per Share (C)
Persistent EPS in the Last Four Seasons
Cash Flow Per Share
Revenue Per Share (Yuan ¥)
Operating Profit Per Share (Yuan ¥)
Per Share Net profit before tax (Yuan ¥)
Realized Sales Gross Profit Growth Rate
Operating Profit Growth Rate
After-tax Net Profit Growth Rate
Regular Net Profit Growth Rate
Continuous Net Profit Growth Rate
Total Asset Growth Rate
Net Value Growth Rate
Total Asset Return Growth Rate Ratio
Cash Reinvestment %
Current Ratio
Quick Ratio
Interest Expense Ratio
Total debt/Total net worth
Debt ratio %
Net worth/Assets
Long-term fund suitability ratio (A)
Borrowing dependency
Contingent liabilities/Net worth
Operating profit/Paid-in capital
Net profit before tax/Paid-in capital
Inventory and accounts receivable/Net value
Total Asset Turnover
Accounts Receivable Turnover
Average Collection Days
Inventory Turnover Rate (times)
Fixed Assets Turnover Frequency
Net Worth Turnover Rate (times)
Revenue per person
Operating profit per person
Allocation rate per person
Working Capital to Total Assets
Quick Assets/Total Assets
Current Assets/Total Assets
Cash/Total Assets
Quick Assets/Current Liability
Cash/Current Liability
Current Liability to Assets
Operating Funds to Liability
Inventory/Working Capital
Inventory/Current Liability
Current Liabilities/Liability
Working Capital/Equity
Current Liabilities/Equity
Long-term Liability to Current Assets
Retained Earnings to Total Assets
Total income/Total expense
Total expense/Assets
Current Asset Turnover Rate
Quick Asset Turnover Rate
Working capitcal Turnover Rate
Cash Turnover Rate
Cash Flow to Sales
Fixed Assets to Assets
Current Liability to Liability
Current Liability to Equity
Equity to Long-term Liability
Cash Flow to Total Assets
Cash Flow to Liability
CFO to Assets
Cash Flow to Equity
Current Liability to Current Assets
Liability-Assets Flag
Net Income to Total Assets
Total assets to GNP price
No-credit Interval
Gross Profit to Sales
Net Income to Stockholder's Equity
Liability to Equity
Degree of Financial Leverage (DFL)
Interest Coverage Ratio (Interest expense to EBIT)
Net Income Flag
Equity to Liability

[6]:
```
# Separando entre obzervação e preditores

Y = df['Bankrupt?']
X = df[df.columns.tolist()[1:]]
```

3 3. Pegando os Componentes Principais

[7]:
```
# Pegando os componentes principais

from sklearn.decomposition import PCA
from sklearn.preprocessing import StandardScaler
```

[8]:
```
# normalizando os dados para pegar os PCAs

scaler = StandardScaler()

X_normal = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)

# pegando a variância explicada pelos componentes principais

pca = PCA(n_components = 20)

evs = pca.fit(X_normal).explained_variance_

fig, ax = plt.subplots(dpi = 720)

ax.scatter(range(1,21), evs)
ax.set_xticks(range(1,21))
ax.set_xticklabels(range(1,21))

plt.show()

print('A partir do sexto componente principal a queda na variância já é␣
↪extremamente marginal')
```



A partir do sexto componente principal a queda na variância já é extremamente marginal

[9]:
```
# pegando até o sexto componente

scaler = StandardScaler()

X_normal = pd.DataFrame(scaler.fit_transform(X), columns = X.columns)

pca = PCA(n_components = 6)

X_pca = pd.DataFrame(pca.fit_transform(X_normal), columns =␣
 ↪['pca_1','pca_2','pca_3',                                                    ␣
 ↪'pca_4','pca_5','pca_6'])

X_pca
```

[9]:

	pca_1	pca_2	pca_3	pca_4	pca_5	pca_6
0	-7.338294	0.373294	-0.309014	-1.021642	0.162798	1.448300
1	-2.703713	-0.986346	-2.155617	-2.473644	0.835692	0.607047
2	-4.307059	-0.404700	-0.309801	-0.729865	0.335723	-0.198898

```
3    -5.830736 -1.385669 -1.301105 -2.160071  0.731532  1.043664
4    -2.343197 -0.201748  0.806267 -0.894912  0.203652  0.266931
…          …         …         …         …         …         …
6814 -1.177758 -0.508512  0.189843  0.236673  0.063634 -0.119839
6815 -1.282275 -0.663230  0.247693 -0.445795  0.231082 -0.793491
6816  1.910308 -1.693786 -1.409778  0.955907 -0.078499 -2.136376
6817  1.854261 -0.397923 -0.488640  0.720515 -0.043994  0.648606
6818  1.235942 -2.329448 -0.466757  2.658750 -0.126020 -4.585579

[6819 rows x 6 columns]
```

4 4. Treinando os Algoritmos

[10]:
```
# criando um dataframe para salvar os resultados das avaliações

resultados = pd.DataFrame({'Algoritmo': ['Logística',
                                         'KNN',
                                         'Naive Bayes',
                                         'Árvores',
                                         'Bagging',
                                         'Random Forest',
                                         'Boosting',
                                         'Support Vector Machines',
                                         'LDA',
                                         'QDA'],
                           'Accuracy':np.zeros(10),
                           'AUC':np.zeros(10),
                           'True Positive':np.zeros(10)})
```

4.1 4.1 Regressão Logística

[11]:
```
from sklearn.linear_model import LogisticRegression
```

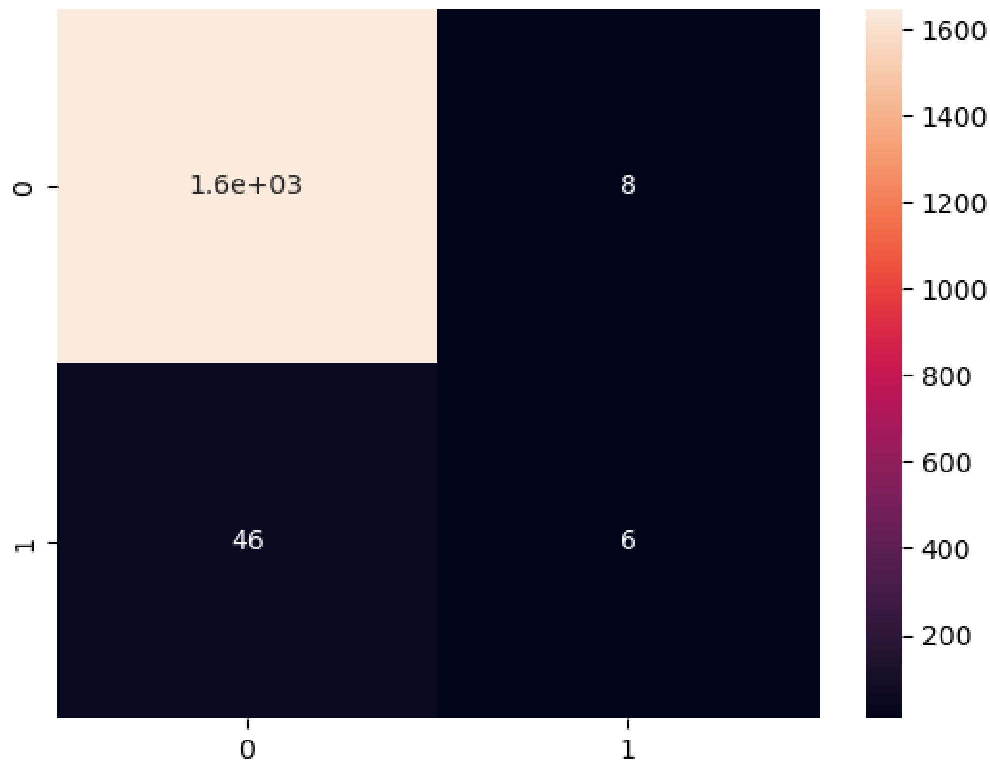
[12]:
```
# dividindo entre conjunto de treinamento e conjunto de teste

X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
 ↪random_state = 0)

# fit para o modelo com diferentes penalizações

model_none = LogisticRegression(penalty = None, max_iter = 7000).fit(X_train,␣
 ↪Y_train)
model_l1   = LogisticRegression(penalty = 'l1', max_iter = 7000, solver =␣
 ↪'saga', C = 0.1).fit(X_train, Y_train)
model_l2   = LogisticRegression(penalty = 'l2', max_iter = 7000, C = 0.1).
 ↪fit(X_train, Y_train)
```

```python
# olhando accuracia no conjunto de teste

Y_pred_none = model_none.predict(X_test)
Y_pred_l1   = model_l1.predict(X_test)
Y_pred_l2   = model_l2.predict(X_test)


print('A Acurácia não muda com a especificação da penalização, mesmo com um␣
 ↪parâmetro bem baixo')
print(f'none:  {accuracy_score(Y_test, Y_pred_none):.4f}')
print(f'l1:    {accuracy_score(Y_test, Y_pred_l1):.4f}')
print(f'l2:    {accuracy_score(Y_test, Y_pred_l2):.4f}')
```

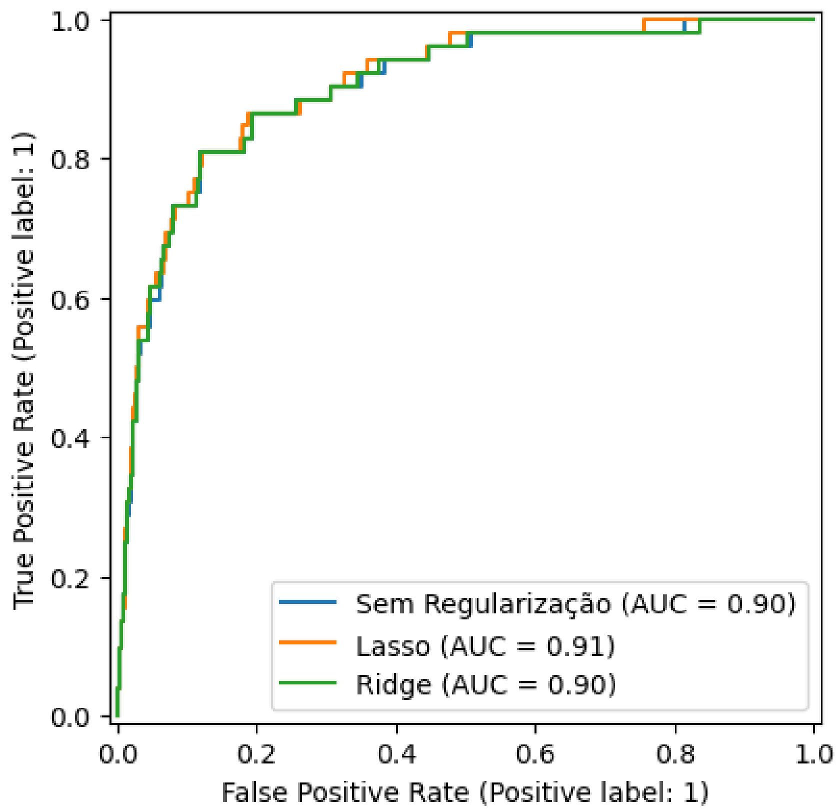
```
A Acurácia não muda com a especificação da penalização, mesmo com um parâmetro
bem baixo
none:  0.9683
l1:    0.9672
l2:    0.9683
```

[13]:

```python
# mostrando a matriz de confusão

from sklearn.metrics import confusion_matrix

cm_none = confusion_matrix(Y_test, Y_pred_none)
cm_l1   = confusion_matrix(Y_test, Y_pred_l1)
cm_l2   = confusion_matrix(Y_test, Y_pred_l2)

sns.heatmap(cm_none, annot = True)

plt.show()
```



[14]:
```
fig, ax = plt.subplots()

RocCurveDisplay.from_estimator(model_none, X_test, Y_test, ax = ax, name = 'Sem␣
↪Regularização')
RocCurveDisplay.from_estimator(model_l1,   X_test, Y_test, ax = ax, name =␣
↪'Lasso')
RocCurveDisplay.from_estimator(model_l2,   X_test, Y_test, ax = ax, name =␣
↪'Ridge')

plt.show()
```



[15]:

```
# salvando os resultados

acc = round(accuracy_score(Y_test, model_none.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model_none.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm_none.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Logística','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Logística','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Logística','True Positive'] =␣
 ↪round(tpr,3)
```

[ ]:

4.2 4.2 K-Nearest Neighbors

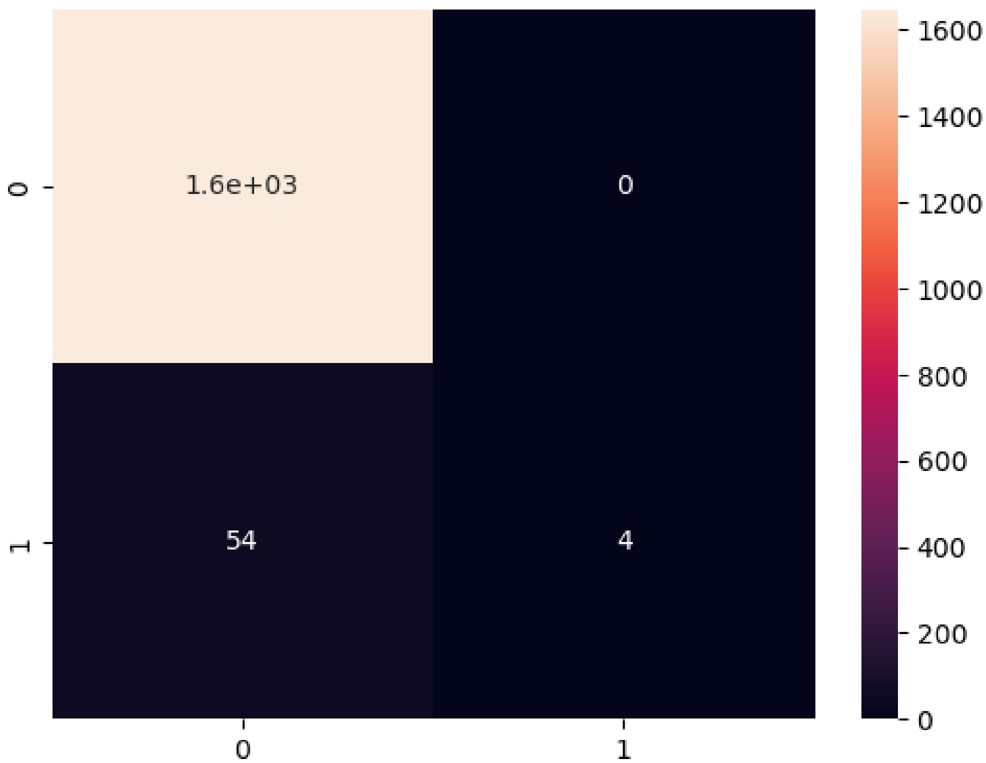
[16]:
```
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score
```

[17]:
```
# dividindo em grupo de teste e grupo de treinamento

X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25, 
 ↪random_state = 42)

# estimando

knn = KNeighborsClassifier(n_neighbors = 20).fit(X_train, Y_train)

Y_pred = knn.predict(X_test)

print(f'Acurácia:{accuracy_score(Y_test,Y_pred):.2f}')
```

```
Acurácia:0.97
```

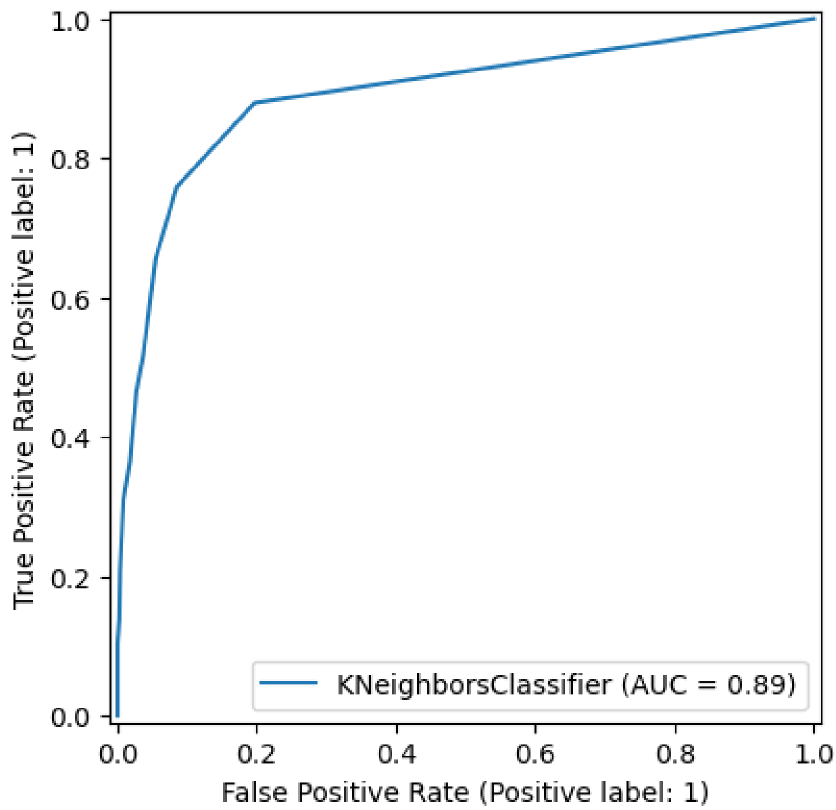
[18]:
```
# matriz de confusão

cm_knn = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm_knn, annot = True)
```

[18]:
```
<Axes: >
```




```
[19]: RocCurveDisplay.from_estimator(knn, X_test, Y_test)
```

```
[19]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd4944c050>
```



[20]:
```
acc = round(accuracy_score(Y_test, knn.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, knn.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm_knn.ravel()
tpr = tp/(tp+fn)


resultados.loc[resultados['Algoritmo'] == 'KNN','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'KNN','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'KNN','True Positive'] = round(tpr,3)
```

4.3 4.3 Naive Bayes

[21]:
```
from sklearn.naive_bayes import GaussianNB
```

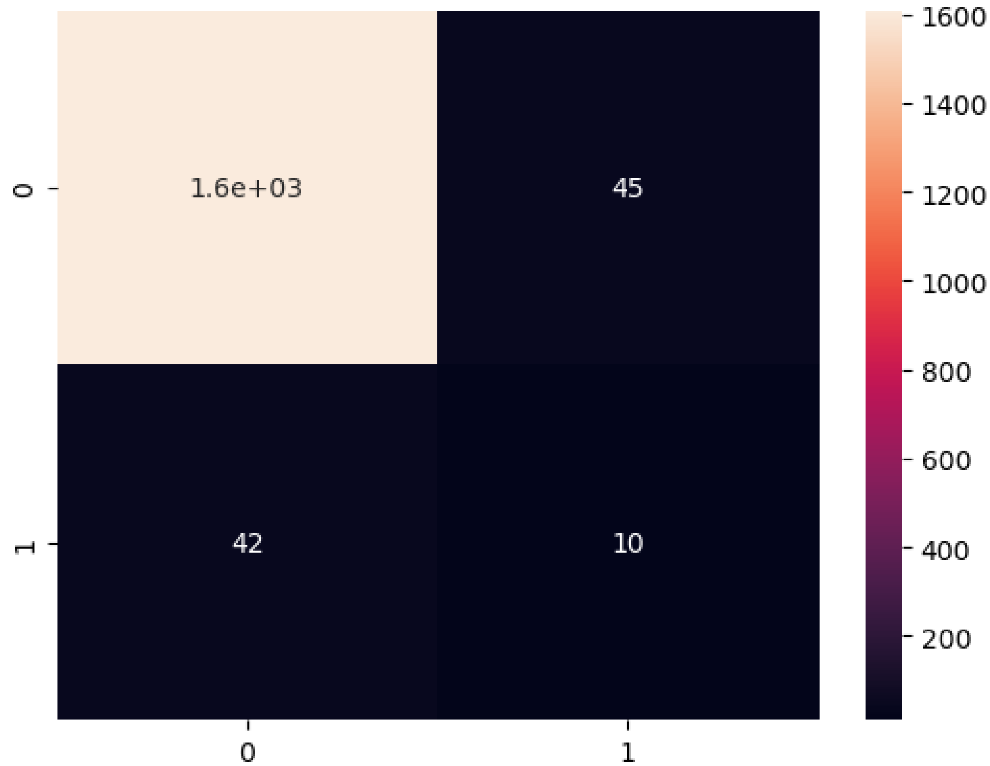
[22]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
↪random_state = 0)

model = GaussianNB()
```

```
model.fit(X_train, Y_train)

Y_pred = model.predict(X_test)

print(f'Acurácia: {accuracy_score(Y_test, Y_pred):.2f}')
```

```
Acurácia: 0.95
```

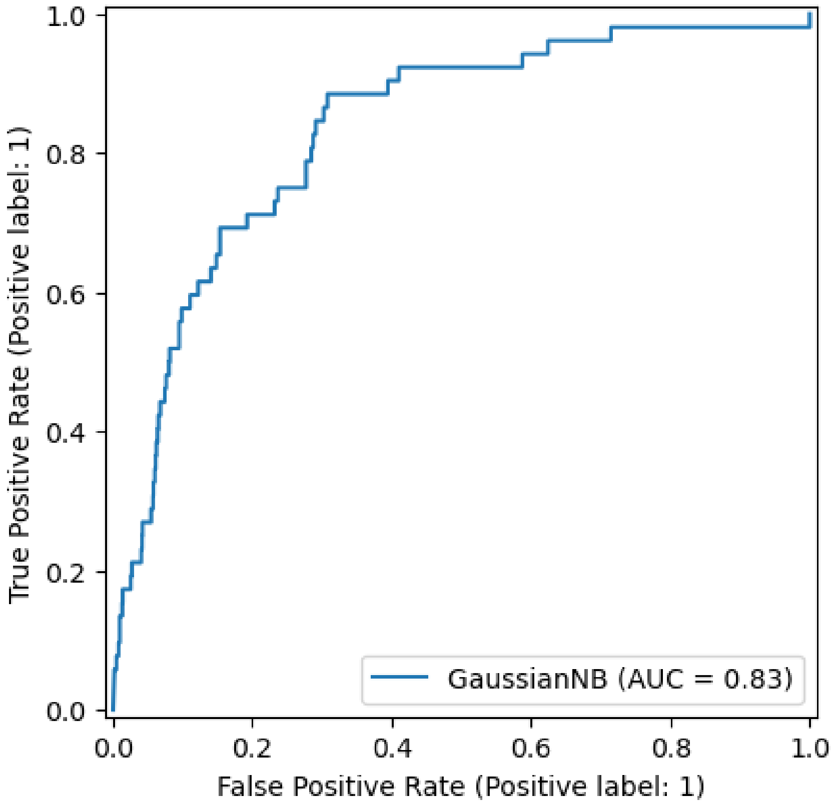
[23]:
```
# pelo naive bayes

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[23]: <Axes: >



[24]:
```
RocCurveDisplay.from_estimator(model, X_test, Y_test)
```

[24]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd4b70f590>



[25]:
```
acc = round(accuracy_score(Y_test, model.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Naive Bayes','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Naive Bayes','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Naive Bayes','True Positive'] =␣
 ↪round(tpr,3)
```

4.4 4.4 Árvores de Decisão

[26]:
```
from sklearn.tree import DecisionTreeClassifier, plot_tree
```

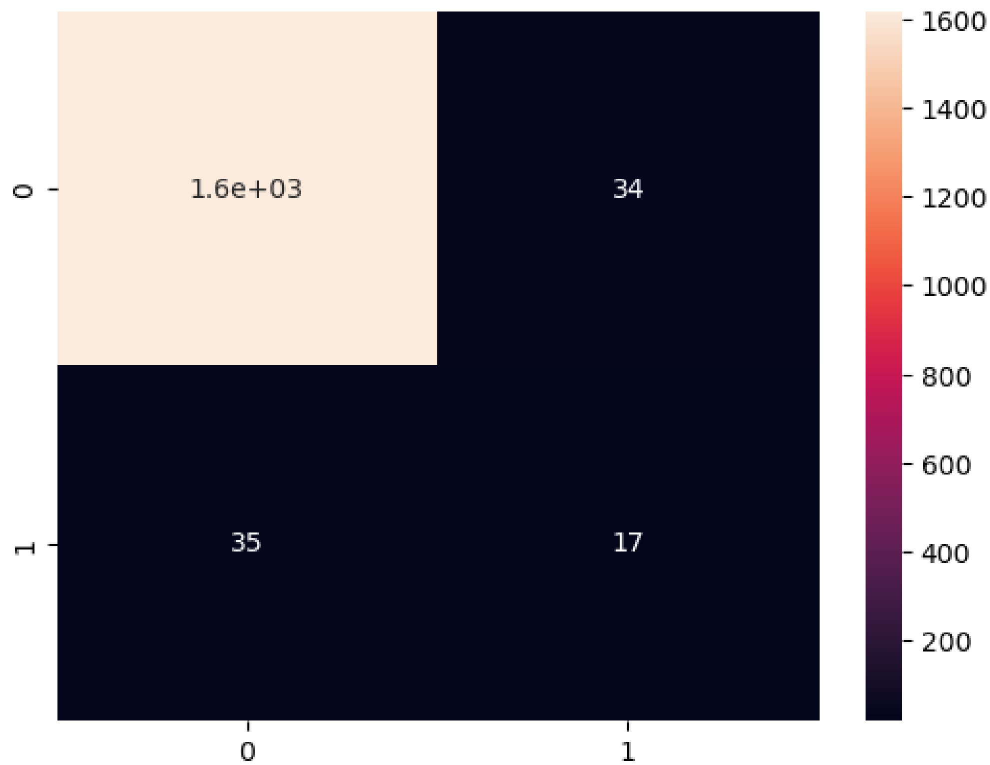
[27]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca,Y, test_size = 0.25,␣
 ↪random_state = 0)

model_entropy = DecisionTreeClassifier(criterion = 'entropy',
                                       random_state = 0,
```

```
                                               max_depth = 15)

model_gini    = DecisionTreeClassifier(criterion = 'gini',
                                               random_state = 0,
                                               max_depth = 15)

model_log_loss= DecisionTreeClassifier(criterion = 'log_loss',
                                               random_state = 0,
                                               max_depth = 15)

model_entropy.fit(X_train, Y_train)
model_gini.fit(X_train, Y_train)
model_log_loss.fit(X_train, Y_train)

print('O desempenho de uma árvore de decisão simples é bem próximo da regressão␣
 ↪logística')
print(f'Entropia: {accuracy_score(Y_test, model_entropy.predict(X_test)):.2f}')
print(f'Gini: {accuracy_score(Y_test, model_gini.predict(X_test)):.2f}')
print(f'Log Loss: {accuracy_score(Y_test, model_log_loss.predict(X_test)):.2f}')
```

```
O desempenho de uma árvore de decisão simples é bem próximo da regressão
logística
Entropia: 0.96
Gini: 0.95
Log Loss: 0.96
```

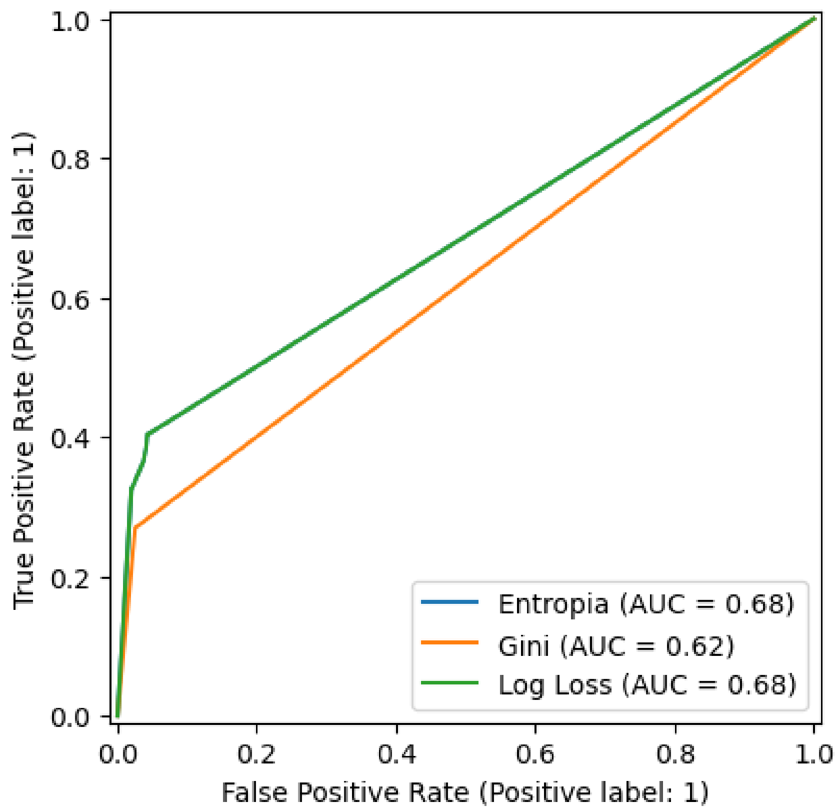
[28]:
```
Y_pred = model_entropy.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[28]:
```
<Axes: >
```



[29]:
```
fig, ax = plt.subplots()

RocCurveDisplay.from_estimator(model_entropy, X_test, Y_test, ax = ax, name =␣
  ↪'Entropia')
RocCurveDisplay.from_estimator(model_gini, X_test, Y_test, ax = ax, name =␣
  ↪'Gini')
RocCurveDisplay.from_estimator(model_log_loss, X_test, Y_test, ax = ax, name =␣
  ↪'Log Loss')
```

[29]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd520a2420>



[30]:
```
acc = round(accuracy_score(Y_test, model_entropy.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model_entropy.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Árvores','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Árvores','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Árvores','True Positive'] =␣
 ↪round(tpr,3)
```

4.5 4.5 Bagging

[31]:
```
from sklearn.ensemble import BaggingClassifier
```

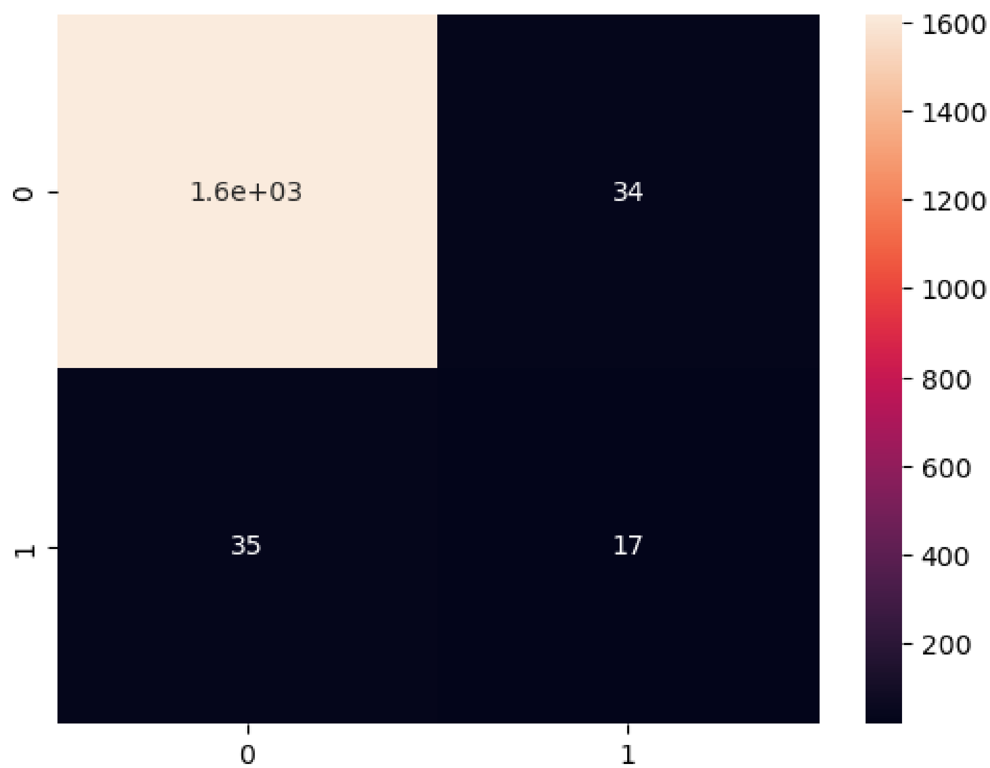
[32]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
 ↪random_state = 0)

model = BaggingClassifier(n_estimators = 20,
                          max_samples = 0.8)
```

```
model.fit(X_train, Y_train)

print('o bootstrap tem um desempenho marginalmente superior ao da árvore␣
 ↪normal')
print(f'bootstrap: {accuracy_score(Y_test, model.predict(X_test)):.2f}')
```

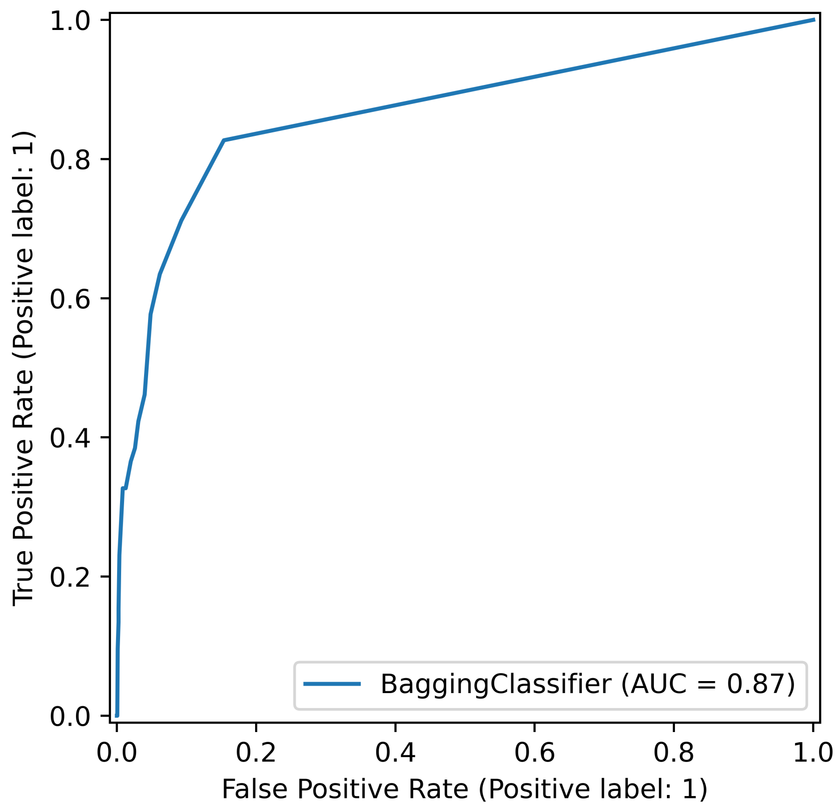
```
o bootstrap tem um desempenho marginalmente superior ao da árvore normal
bootstrap: 0.97
```

[33]:
```
cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[33]: <Axes: >



[34]:
```
fig, ax = plt.subplots(dpi = 720)

RocCurveDisplay.from_estimator(model, X_test, Y_test, ax = ax)

plt.show()
```



[35]:
```
acc = round(accuracy_score(Y_test, model.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Bagging','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Bagging','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Bagging','True Positive'] =
 ↪round(tpr,3)
```

4.6 4.6 Random Forest

[36]:
```
from sklearn.ensemble import RandomForestClassifier
```

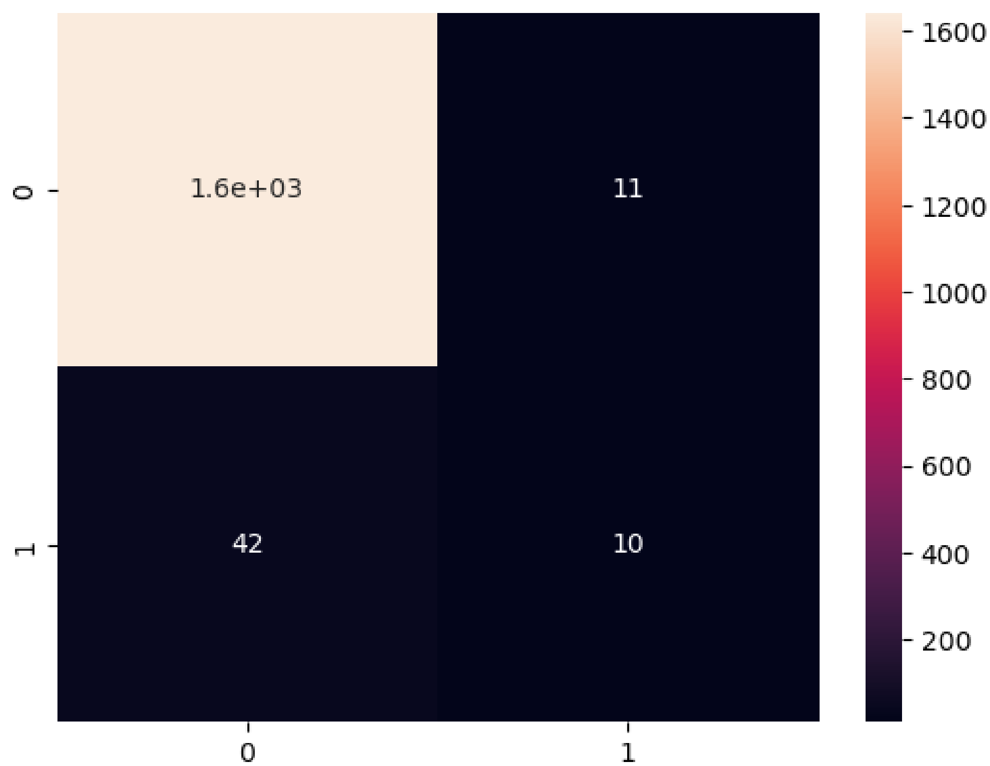
[37]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,
 ↪random_state = 0)

model_gini     = RandomForestClassifier(criterion = 'gini')
model_entropy  = RandomForestClassifier(criterion = 'entropy')
```

```
model_log_loss = RandomForestClassifier(criterion = 'log_loss')

model_gini.fit(X_train,Y_train)
model_entropy.fit(X_train,Y_train)
model_log_loss.fit(X_train,Y_train)

print('O desempenho não muda muito variando a métrica')
print(f'Gini: {accuracy_score(Y_test, model_gini.predict(X_test)):.2f}')
print(f'Entropy: {accuracy_score(Y_test, model_entropy.predict(X_test)):.2f}')
print(f'Log Loss: {accuracy_score(Y_test, model_log_loss.predict(X_test)):.2f}')
```

```
O desempenho não muda muito variando a métrica
Gini: 0.97
Entropy: 0.97
Log Loss: 0.97
```

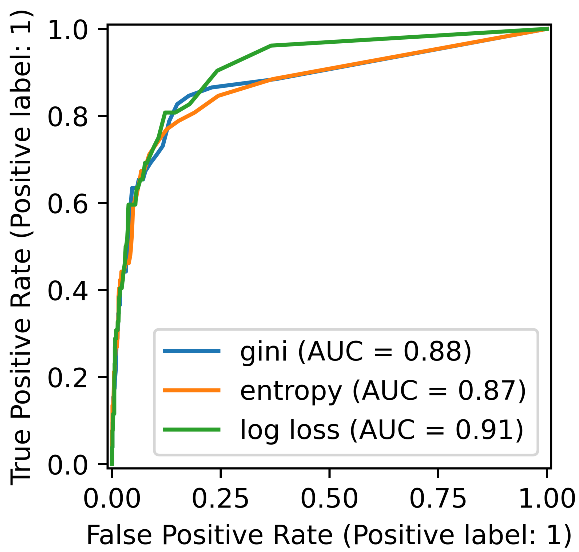
[38]:
```
Y_pred = model_gini.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[38]: <Axes: >



[39]:
```
fig, ax = plt.subplots(dpi = 720, figsize = (3,3))

RocCurveDisplay.from_estimator(model_gini, X_test, Y_test, ax = ax, name =
 ↪'gini')
RocCurveDisplay.from_estimator(model_entropy, X_test, Y_test, ax = ax, name =
 ↪'entropy')
RocCurveDisplay.from_estimator(model_log_loss, X_test, Y_test, ax = ax, name =
 ↪'log loss')
```

[39]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd51ef3890>
```



[40]:
```
acc = round(accuracy_score(Y_test, model_gini.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model_gini.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Random Forest','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Random Forest','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Random Forest','True Positive'] =
 ↪round(tpr,3)
```

4.7 4.7 Gradient Boosting

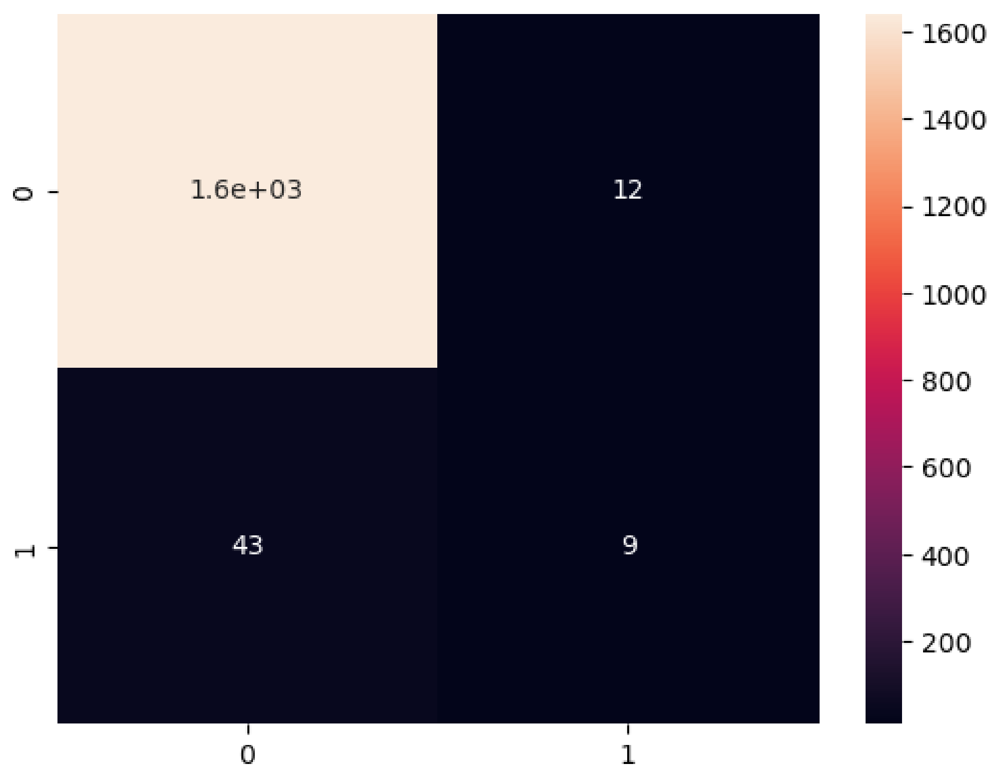
[41]:
```
from sklearn.ensemble import GradientBoostingClassifier
```

[42]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
 ↪random_state = 0)

model = GradientBoostingClassifier()

model.fit(X_train, Y_train)

print(f'boosting: {accuracy_score(Y_test, model.predict(X_test)):.2f}')
```

```
boosting: 0.97
```

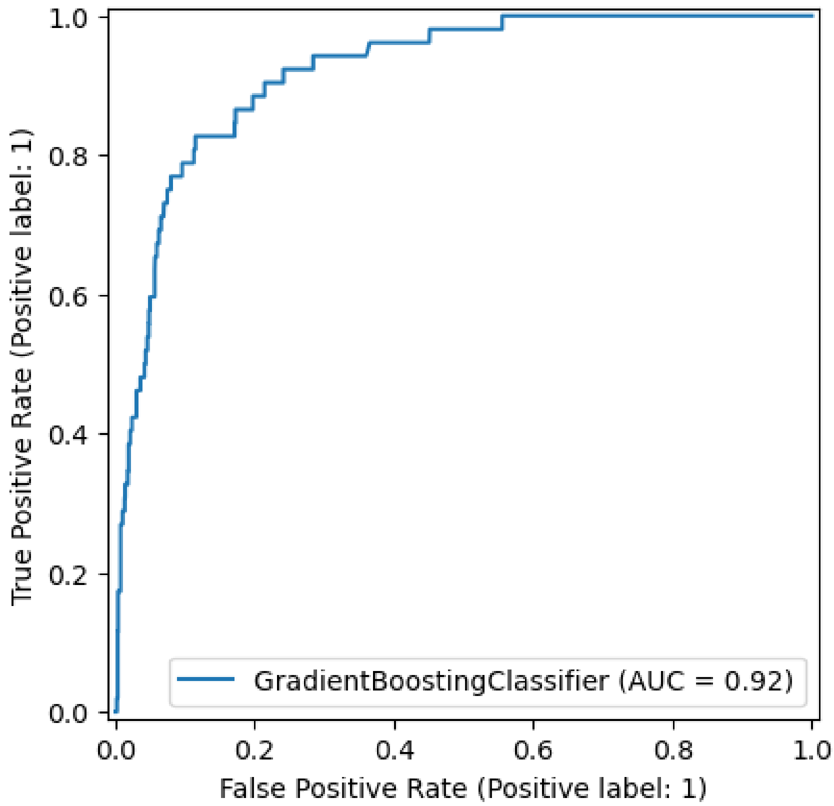
[43]:
```
Y_pred = model.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[43]:
```
<Axes: >
```



[44]:
```
RocCurveDisplay.from_estimator(model, X_test, Y_test)
```

[44]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd4bbde600>



[45]:
```
acc = round(accuracy_score(Y_test, model.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, model.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Boosting','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'Boosting','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'Boosting','True Positive'] =␣
↪round(tpr,3)
```

4.8 4.8 Support Vector Machine

[46]:
```
from sklearn.preprocessing import StandardScaler
from sklearn.svm import SVC
```

[47]:
```
# separando entre conjunto de treinamento e de teste

X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
↪random_state = 0)

# Normalizando os dados

cols = X_train.columns

scaler = StandardScaler()

X_train_scale = scaler.fit(X_train).transform(X_train)
X_train_scale = pd.DataFrame(X_train_scale, columns = cols)

X_test_scale = scaler.fit(X_test).transform(X_test)
X_test_scale = pd.DataFrame(X_test_scale, columns = cols)
```

[48]:
```
# Variando os kernels nos estimadores

# linear (colocar o 'linear' no parâmetro trava o python)
svc_linear  = SVC(kernel = 'poly',degree = 1).fit(X_train, Y_train)

# polinomial
svc_poly    = SVC(kernel = 'poly'   ).fit(X_train, Y_train)

# radial
svc_rbf     = SVC(kernel = 'rbf'    ).fit(X_train, Y_train)

# sigmoid
svc_sigmoid = SVC(kernel = 'sigmoid').fit(X_train, Y_train)
```

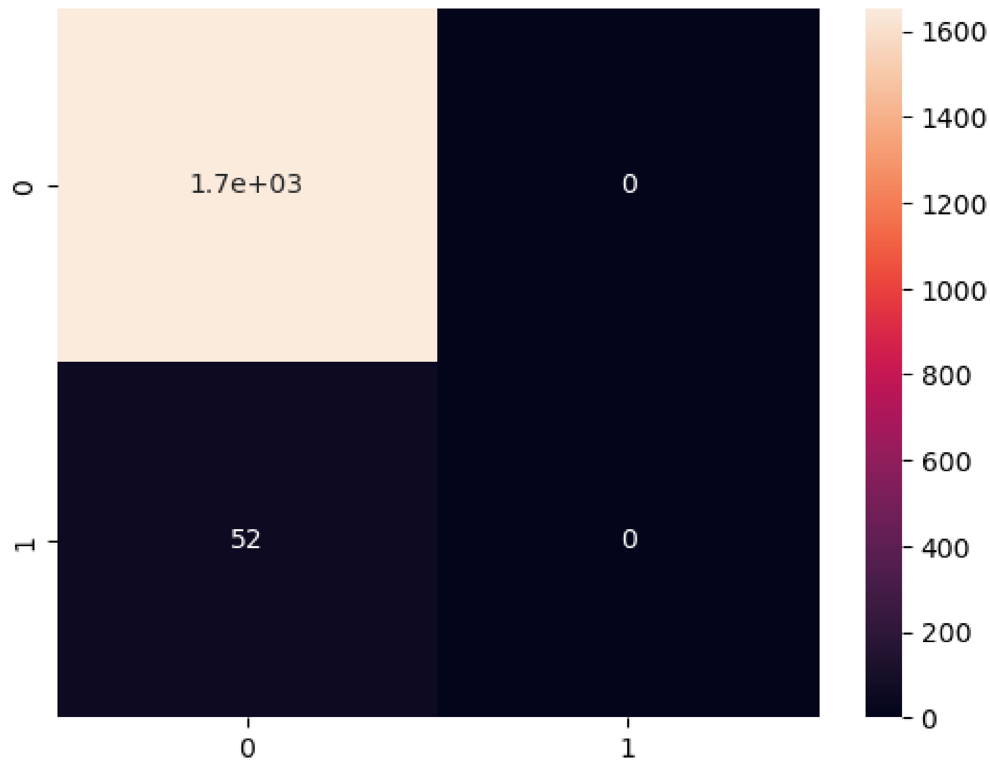
[49]:
```
# olhando os resultados das previsões
# apenas o kernel sigmoide tem um desempenho marginalmente pior
# o resto dos kerneis desempenha da mesma forma

print(f'linear: {accuracy_score(Y_test, svc_linear.predict(X_test)):.2f}')
print(f'polinomial: {accuracy_score(Y_test, svc_poly.predict(X_test)):.2f}')
print(f'rbf: {accuracy_score(Y_test, svc_rbf.predict(X_test)):.2f}')
print(f'svc_sigmoid: {accuracy_score(Y_test, svc_sigmoid.predict(X_test)):.2f}')
```

```
linear: 0.97
polinomial: 0.97
rbf: 0.97
```

```
svc_sigmoid: 0.96
```

[50]:
```python
# esse modelo tem um problema que ele nunca prevê nenhum negativo

Y_pred = svc_linear.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[50]:
```
<Axes: >
```



[51]:
```python
acc = round(accuracy_score(Y_test, svc_linear.predict(X_test)),3)

# auc = round(roc_auc_score(Y_test, knn.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'Support Vector Machines','Accuracy']␣
 ↪= acc
```

```
resultados.loc[resultados['Algoritmo'] == 'Support Vector Machines','AUC'] =␣
 ↪None
resultados.loc[resultados['Algoritmo'] == 'Support Vector Machines','True␣
 ↪Positive'] = round(tpr,3)
```

4.9 4.9 LDA

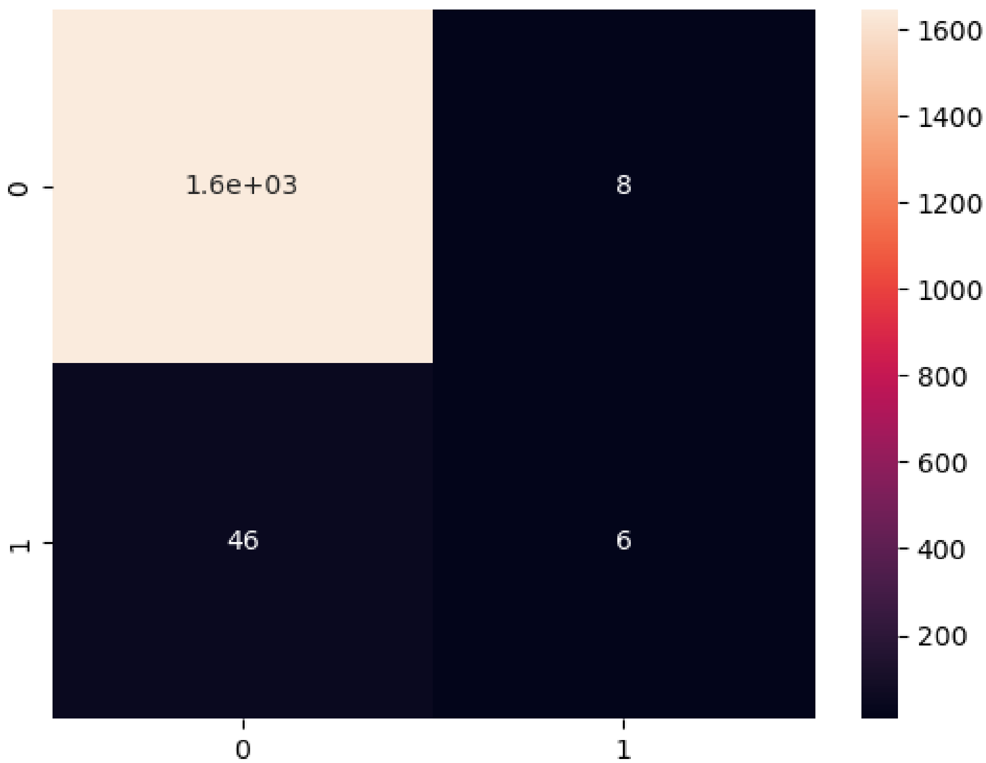
[52]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
```

[53]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
 ↪random_state = 0)

LDA = LinearDiscriminantAnalysis()

LDA.fit(X_train, Y_train)

print(f'LDA: {accuracy_score(Y_test, LDA.predict(X_test)):.2f}')
```

```
LDA: 0.97
```

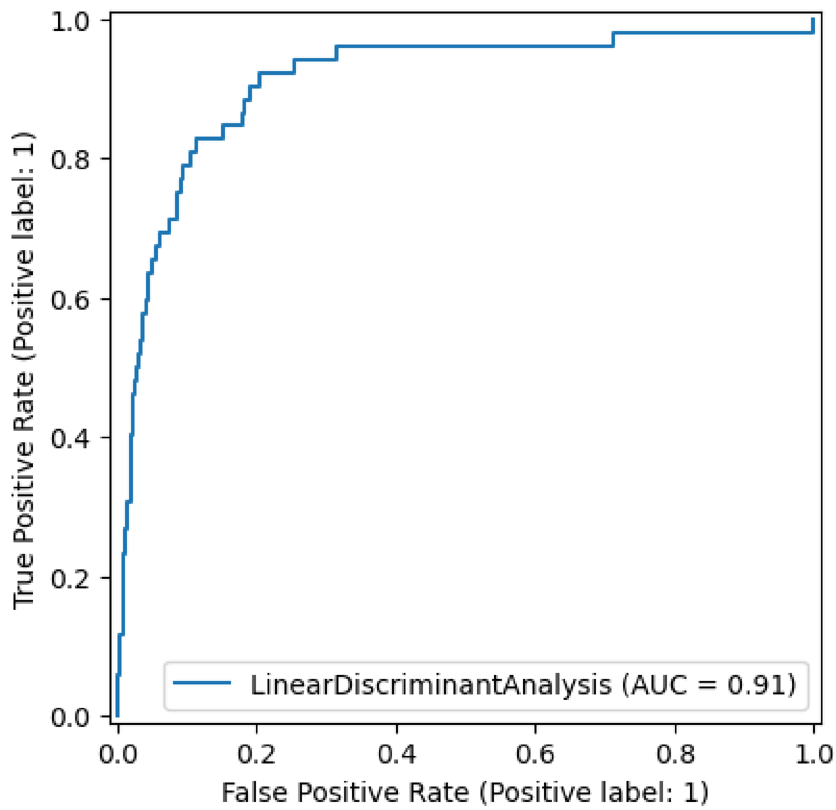
[54]:
```
Y_pred = LDA.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[54]:
```
<Axes: >
```



[55]:
```
RocCurveDisplay.from_estimator(LDA, X_test, Y_test)
```

[55]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd4bb09280>



[56]:
```
acc = round(accuracy_score(Y_test, LDA.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, LDA.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'LDA','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'LDA','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'LDA','True Positive'] = round(tpr,3)
```

4.10 4.10 QDA

[57]:
```
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
```

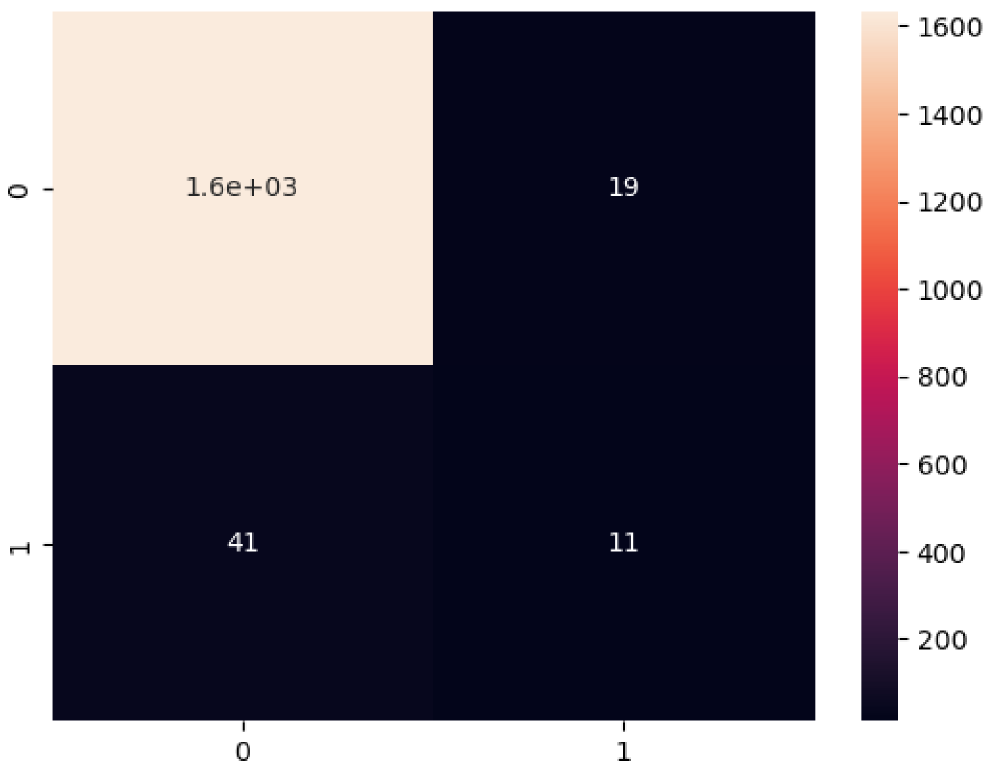
[58]:
```
X_train, X_test, Y_train, Y_test = train_test_split(X_pca, Y, test_size = 0.25,␣
↪random_state = 0)

QDA = QuadraticDiscriminantAnalysis()

QDA.fit(X_train, Y_train)
```

```python
print(f'QDA: {accuracy_score(Y_test, QDA.predict(X_test)):.2f}')
```

```
QDA: 0.96
```

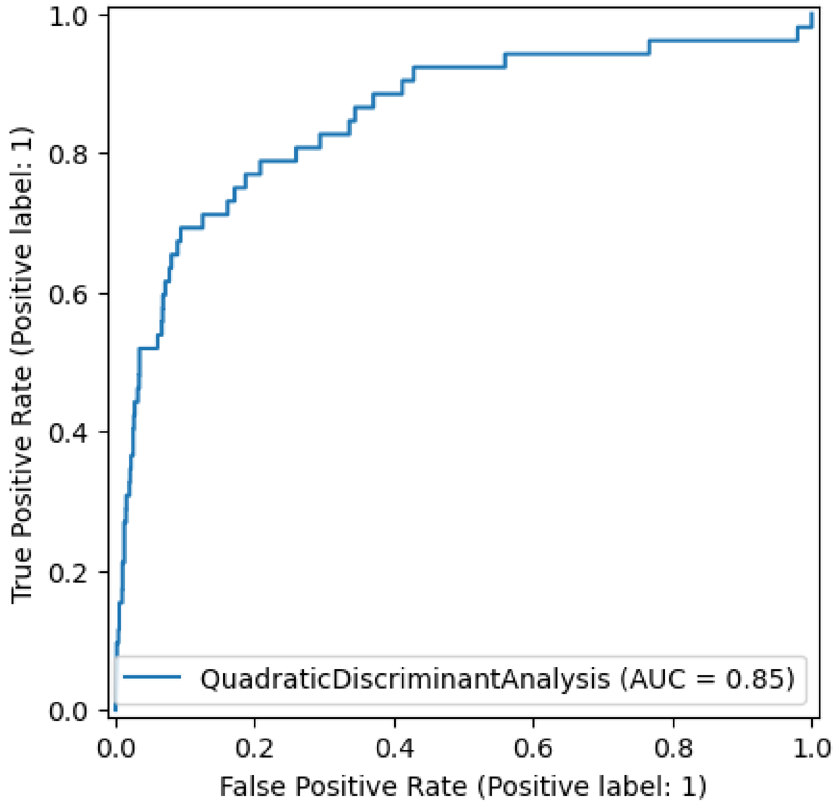
[59]:
```python
Y_pred = QDA.predict(X_test)

cm = confusion_matrix(Y_test, Y_pred)

sns.heatmap(cm, annot = True)
```

[59]: <Axes: >



[60]:
```python
RocCurveDisplay.from_estimator(QDA, X_test, Y_test)
```

[60]: <sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x1fd520a2cf0>



[61]:
```
acc = round(accuracy_score(Y_test, QDA.predict(X_test)),3)

auc = round(roc_auc_score(Y_test, QDA.predict_proba(X_test)[:,1]),3)

tn, fp, fn, tp = cm.ravel()
tpr = tp/(tp+fn)

resultados.loc[resultados['Algoritmo'] == 'QDA','Accuracy'] = acc
resultados.loc[resultados['Algoritmo'] == 'QDA','AUC'] = auc
resultados.loc[resultados['Algoritmo'] == 'QDA','True Positive'] = round(tpr,3)
```

5 5. Resultados

[62]:
```
resultados
```

[62]:

	Algoritmo	Accuracy	AUC	True Positive
0	Logística	0.968	0.901	0.115
1	KNN	0.968	0.892	0.069
2	Naive Bayes	0.949	0.834	0.192
3	Árvores	0.960	0.683	0.327

4	Bagging	0.973	0.867	0.327
5	Random Forest	0.969	0.875	0.192
6	Boosting	0.968	0.921	0.173
7	Support Vector Machines	0.970	NaN	0.000
8	LDA	0.968	0.912	0.115
9	QDA	0.965	0.852	0.212