Aprendizagem de Máquina: Atividade 10 – Naive Bayes

Carlos Emmanuel Pereira Alves
Curso de Bacharelado em Ciência da Computação
Universidade Federal do Agreste de Pernambuco (UFAPE)
Garanhuns, Brasil
carlos.emmanuel.236@gmail.com

1) Utilizando o classificador 1-NN com distância euclidiana.

```python
import numpy as np
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler, StandardScaler

# breast-cancer-wisconsin

cols = ["Sample code number", "Clump Thickness", "Uniformity of Cell Size", "Uniformity of Cell Shape",
        "Marginal Adhesion", "Single Epithelial Cell Size", "Bare Nuclei", "Bland Chromatin","Normal Nucleoli",
        "Mitoses", "Class"]

data = pd.read_csv('breast-cancer-wisconsin.data', header=None, names=cols)
data = data.replace('?', np.nan)

imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
data_imputer = pd.DataFrame(imputer.fit_transform(data), columns=data.columns)
data = data_imputer

data_X = data.drop('Class', axis=1)
data_y = data['Class']
data_y = data['Class'].astype(int)

s_scaler = StandardScaler()
data_X = pd.DataFrame(s_scaler.fit_transform(data_X), columns=data_X.columns)
```

```python
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder

# wdbc

cols2 = [
    'ID', 'Diagnosis', 'Radius_mean', 'Texture_mean', 'Perimeter_mean', 'Area_mean', 'Smoothness_mean',
    'Compactness_mean', 'Concavity_mean', 'Concave_points_mean', 'Symmetry_mean', 'Fractal_dimension_mean',
    'Radius_se', 'Texture_se', 'Perimeter_se', 'Area_se', 'Smoothness_se', 'Compactness_se', 'Concavity_se',
    'Concave_points_se', 'Symmetry_se', 'Fractal_dimension_se', 'Radius_worst', 'Texture_worst', 'Perimeter_worst',
    'Area_worst', 'Smoothness_worst', 'Compactness_worst', 'Concavity_worst', 'Concave_points_worst', 'Symmetry_worst', 'Fractal_dimension_worst'
]

data2 = pd.read_csv('wdbc.data', header=None, names=cols2)
data2 = data2.drop('ID', axis=1)

le = LabelEncoder()
data2['Diagnosis'] = le.fit_transform(data2['Diagnosis'])

data2_X = data2.drop('Diagnosis', axis=1)
data2_y = data2['Diagnosis']

s_scaler = StandardScaler()
data2_X = pd.DataFrame(s_scaler.fit_transform(data2_X), columns=data2_X.columns)
```

```
import pandas as pd
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder

# wpbc

data3 = pd.read_csv('wpbc.data', header=None)
data3 = data3.replace('?', np.nan)
data3 = data3.drop(data3.columns[0], axis=1)

imputer = SimpleImputer(strategy='most_frequent', missing_values=np.nan)
data_imputer = pd.DataFrame(imputer.fit_transform(data3), columns=data3.columns)
data3 = data_imputer

le = LabelEncoder()
data3[1] = le.fit_transform(data3[1])

data3_X = data3.drop(data3.columns[1], axis=1)
data3_y = data3[1]

s_scaler = StandardScaler()
data3_X = pd.DataFrame(s_scaler.fit_transform(data3_X), columns=data3_X.columns)
```

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# breast-cancer-wisconsin

loo = LeaveOneOut()
accuracies = []

for train_index, test_index in loo.split(data_X):
  X_train, X_test = data_X.iloc[train_index], data_X.iloc[test_index]
  y_train, y_test = data_y.iloc[train_index], data_y.iloc[test_index]

  knn = KNeighborsClassifier(n_neighbors=1)
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  accuracy = accuracy_score(y_test, predict)
  accuracies.append(accuracy)

mean = np.mean(accuracies)
std = np.std(accuracies)
confidence_interval_n = round(mean - (1.96 * std), 5)
confidence_interval_p = round(mean + (1.96 * std), 5)

print("Média:", mean)
print("Desvio padrão:", std)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.9542203147353362
Desvio padrão: 0.209006951276105
Intervalo de confiança: 0.54457   1.36387
```

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# wdbc

loo = LeaveOneOut()
accuracies2 = []

for train_index, test_index in loo.split(data2_X):
  X_train, X_test = data2_X.iloc[train_index], data2_X.iloc[test_index]
  y_train, y_test = data2_y.iloc[train_index], data2_y.iloc[test_index]

  knn = KNeighborsClassifier(n_neighbors=1)
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  accuracy = accuracy_score(y_test, predict)
  accuracies2.append(accuracy)

mean = np.mean(accuracies2)
std = np.std(accuracies2)
confidence_interval_n = round(mean - (1.96 * std), 5)
confidence_interval_p = round(mean + (1.96 * std), 5)

print("Média:", mean)
print("Desvio padrão:", std)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.9507908611599297
Desvio padrão: 0.2163044139510079
Intervalo de confiança: 0.52683   1.37475
```

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.neighbors import KNeighborsClassifier
from sklearn.metrics import accuracy_score, confusion_matrix

# wpbc

loo = LeaveOneOut()
accuracies3 = []

for train_index, test_index in loo.split(data3_X):
  X_train, X_test = data3_X.iloc[train_index], data3_X.iloc[test_index]
  y_train, y_test = data3_y.iloc[train_index], data3_y.iloc[test_index]

  knn = KNeighborsClassifier(n_neighbors=1)
  knn.fit(X_train, y_train)
  predict = knn.predict(X_test)

  accuracy = accuracy_score(y_test, predict)
  accuracies3.append(accuracy)

mean = np.mean(accuracies3)
std = np.std(accuracies3)
confidence_interval_n = round(mean - (1.96 * std), 5)
confidence_interval_p = round(mean + (1.96 * std), 5)

print("Média:", mean)
print("Desvio padrão:", std)
print(f"Intervalo de confiança: {confidence_interval_n}   {confidence_interval_p}")
```

```
Média: 0.9191919191919192
Desvio padrão: 0.2725401527925665
Intervalo de confiança: 0.38501   1.45337
```

2) Utilizando o classificador Naive Bayes e discretização de variáveis. Tratar as variáveis como categóricas e estimar as probabilidades por contagem. Teste a discretização para diferentes números de intervalos: 2, 4, 8, 16, 32, 64, 128 e 256.

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split

# breast-cancer-wisconsin

num_intervals = [2, 4, 8, 16, 32, 64, 128, 256]
accuracies = []

for num_interval in num_intervals:
  X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, test_size=0.3)

  discretizer = KBinsDiscretizer(n_bins=num_interval, encode='ordinal', strategy='uniform')
  X_train_discrete = discretizer.fit_transform(X_train)
  X_test_discrete = discretizer.transform(X_test)

  clf = CategoricalNB()
  clf.fit(X_train_discrete, y_train)
  predict = clf.predict(X_test_discrete)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Intervalos: {num_interval}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Intervalos: 2
Média: 0.9524
Matriz de Confusão:
[[133   4]
 [  6  67]]

Intervalos: 4
Média: 0.9571
Matriz de Confusão:
[[137   5]
 [  4  64]]

Intervalos: 8
Média: 0.9762
Matriz de Confusão:
[[138   2]
 [  3  67]]

Intervalos: 16
Média: 0.9714
Matriz de Confusão:
[[129   4]
 [  2  75]]

Intervalos: 32
Média: 0.9762
Matriz de Confusão:
[[139   3]
 [  2  66]]

Intervalos: 64
Média: 0.9762
Matriz de Confusão:
[[140   2]
 [  3  65]]

Intervalos: 128
Média: 0.9714
Matriz de Confusão:
[[137   3]
 [  3  67]]

Intervalos: 256
Média: 0.9714
Matriz de Confusão:
[[136   5]
 [  1  68]]
```

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split

# wdbc

num_intervals = [2, 4, 8, 16, 32, 64, 128, 256]
accuracies = []

for num_interval in num_intervals:
  X_train, X_test, y_train, y_test = train_test_split(data2_X, data2_y, test_size=0.3)

  discretizer = KBinsDiscretizer(n_bins=num_interval, encode='ordinal', strategy='uniform')
  X_train_discrete = discretizer.fit_transform(X_train)
  X_test_discrete = discretizer.transform(X_test)

  clf = CategoricalNB()
  clf.fit(X_train_discrete, y_train)
  predict = clf.predict(X_test_discrete)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Intervalos: {num_interval}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Intervalos: 2
Média: 0.8947
Matriz de Confusão:
[[113   4]
 [ 14  40]]

Intervalos: 4
Média: 0.9298
Matriz de Confusão:
[[95  7]
 [ 5 64]]

Intervalos: 8
Média: 0.9649
Matriz de Confusão:
[[97  2]
 [ 4 68]]

Intervalos: 16
Média: 0.9708
Matriz de Confusão:
[[110   4]
 [  1  56]]

Intervalos: 32
Média: 0.9123
Matriz de Confusão:
[[105   7]
 [  8  51]]

Intervalos: 64
Média: 0.9357
Matriz de Confusão:
[[98  3]
 [ 8 62]]

Intervalos: 128
Média: 0.9649
Matriz de Confusão:
[[113   1]
 [  5  52]]

Intervalos: 256
Média: 0.8830
Matriz de Confusão:
[[106   2]
 [ 18  45]]
```

```python
from sklearn.model_selection import LeaveOneOut
from sklearn.preprocessing import KBinsDiscretizer
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.naive_bayes import CategoricalNB
from sklearn.model_selection import train_test_split

# wpbc

num_intervals = [2, 4, 8, 16, 32, 64, 128, 256]
accuracies = []

for num_interval in num_intervals:
  X_train, X_test, y_train, y_test = train_test_split(data2_X, data2_y, test_size=0.3)

  discretizer = KBinsDiscretizer(n_bins=num_interval, encode='ordinal', strategy='uniform')
  X_train_discrete = discretizer.fit_transform(X_train)
  X_test_discrete = discretizer.transform(X_test)

  clf = CategoricalNB()
  clf.fit(X_train_discrete, y_train)
  predict = clf.predict(X_test_discrete)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Intervalos: {num_interval}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Intervalos: 2
Média: 0.8830
Matriz de Confusão:
[[97  2]
 [18 54]]

Intervalos: 4
Média: 0.9415
Matriz de Confusão:
[[103   1]
 [  9  58]]

Intervalos: 8
Média: 0.9298
Matriz de Confusão:
[[106   5]
 [  7  53]]

Intervalos: 16
Média: 0.9181
Matriz de Confusão:
[[100   8]
 [  6  57]]

Intervalos: 32
Média: 0.9474
Matriz de Confusão:
[[101   4]
 [  5  61]]

Intervalos: 64
Média: 0.9415
Matriz de Confusão:
[[103   0]
 [ 10  58]]

Intervalos: 128
Média: 0.9591
Matriz de Confusão:
[[100   1]
 [  6  64]]

Intervalos: 256
Média: 0.9298
Matriz de Confusão:
[[105   4]
 [  8  54]]
```

3) Utilizando o classificador Naive Bayes e janela de Pazern retangular para as bases que atributos numérico. Teste a discretização para diferentes larguras de janelas: 0,001; 0,01; 0,1; 1; 10; 100; 1000.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

# breast-cancer-wisconsin

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
  X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, test_size=0.3)

  kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
  X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
  X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

  clf = GaussianNB()
  clf.fit(X_train_discretized, y_train)
  predict = clf.predict(X_test_discretized)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Largura de Janela: {bandwidth}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.7095
Matriz de Confusão:
[[149   0]
 [ 61   0]]

Largura de Janela: 0.01
Média: 0.4429
Matriz de Confusão:
[[53 88]
 [29 40]]

Largura de Janela: 0.1
Média: 0.6143
Matriz de Confusão:
[[125   9]
 [ 72   4]]

Largura de Janela: 1
Média: 0.6190
Matriz de Confusão:
[[119  33]
 [ 47  11]]

Largura de Janela: 10
Média: 0.6524
Matriz de Confusão:
[[132  13]
 [ 60   5]]

Largura de Janela: 100
Média: 0.6667
Matriz de Confusão:
[[136   6]
 [ 64   4]]

Largura de Janela: 1000
Média: 0.6381
Matriz de Confusão:
[[133   2]
 [ 74   1]]
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

# wdbc

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
  X_train, X_test, y_train, y_test = train_test_split(data2_X, data2_y, test_size=0.3)

  kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
  X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
  X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

  clf = GaussianNB()
  clf.fit(X_train_discretized, y_train)
  predict = clf.predict(X_test_discretized)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Largura de Janela: {bandwidth}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.5497
Matriz de Confusão:
[[71 38]
 [39 23]]

Largura de Janela: 0.01
Média: 0.4094
Matriz de Confusão:
[[29 81]
 [20 41]]

Largura de Janela: 0.1
Média: 0.5789
Matriz de Confusão:
[[87 33]
 [39 12]]

Largura de Janela: 1
Média: 0.4971
Matriz de Confusão:
[[50 64]
 [22 35]]

Largura de Janela: 10
Média: 0.5906
Matriz de Confusão:
[[84 20]
 [50 17]]

Largura de Janela: 100
Média: 0.5614
Matriz de Confusão:
[[82 30]
 [45 14]]

Largura de Janela: 1000
Média: 0.5789
Matriz de Confusão:
[[85 27]
 [45 14]]
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score, confusion_matrix

# wpbc

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
  X_train, X_test, y_train, y_test = train_test_split(data3_X, data3_y, test_size=0.3)

  kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
  X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
  X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

  clf = GaussianNB()
  clf.fit(X_train_discretized, y_train)
  predict = clf.predict(X_test_discretized)

  accuracy = accuracy_score(y_test, predict)
  matrix = confusion_matrix(y_test, predict)

  print(f"Largura de Janela: {bandwidth}")
  print(f"Média: {accuracy:.4f}")
  print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.6333
Matriz de Confusão:
[[30 12]
 [10  8]]

Largura de Janela: 0.01
Média: 0.6167
Matriz de Confusão:
[[34 11]
 [12  3]]

Largura de Janela: 0.1
Média: 0.6833
Matriz de Confusão:
[[40  7]
 [12  1]]

Largura de Janela: 1
Média: 0.6500
Matriz de Confusão:
[[36 10]
 [11  3]]

Largura de Janela: 10
Média: 0.7500
Matriz de Confusão:
[[45  4]
 [11  0]]

Largura de Janela: 100
Média: 0.7500
Matriz de Confusão:
[[43  7]
 [ 8  2]]

Largura de Janela: 1000
Média: 0.7000
Matriz de Confusão:
[[41  6]
 [12  1]]
```

4) Utilizando o classificador Naive Bayes e janela de Pazern gaussiana para as bases que atributos numérico. Teste a discretização para diferentes larguras de janelas: 0,001; 0,01; 0,1; 1; 10; 100; 1000.

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# breast-cancer-wisconsin

X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, test_size=0.3)

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
    kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
    X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
    X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

    clf = GaussianNB()
    clf.fit(X_train_discretized, y_train)

    predict = clf.predict(X_test_discretized)
    accuracy = accuracy_score(y_test, predict)

    print(f"Largura de Janela: {bandwidth}")
    print(f"Média: {accuracy:.4f}")
    print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.6238
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.01
Média: 0.6286
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.1
Média: 0.6286
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1
Média: 0.6619
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 10
Média: 0.6286
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 100
Média: 0.6429
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1000
Média: 0.6714
Matriz de Confusão:
[[49  0]
 [ 0 11]]
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# wdbc

X_train, X_test, y_train, y_test = train_test_split(data2_X, data2_y, test_size=0.3)

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
    kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
    X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
    X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

    clf = GaussianNB()
    clf.fit(X_train_discretized, y_train)

    predict = clf.predict(X_test_discretized)
    accuracy = accuracy_score(y_test, predict)

    print(f"Largura de Janela: {bandwidth}")
    print(f"Média: {accuracy:.4f}")
    print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.4327
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.01
Média: 0.6140
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.1
Média: 0.5965
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1
Média: 0.5439
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 10
Média: 0.4854
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 100
Média: 0.5673
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1000
Média: 0.6082
Matriz de Confusão:
[[49  0]
 [ 0 11]]
```

```python
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KernelDensity
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import accuracy_score

# wpbc

X_train, X_test, y_train, y_test = train_test_split(data3_X, data3_y, test_size=0.3)

bandwidths = [0.001, 0.01, 0.1, 1, 10, 100, 1000]

for bandwidth in bandwidths:
    kde = KernelDensity(kernel='gaussian', bandwidth=bandwidth)
    X_train_discretized = kde.fit(X_train).sample(X_train.shape[0])
    X_test_discretized = kde.fit(X_train).sample(X_test.shape[0])

    clf = GaussianNB()
    clf.fit(X_train_discretized, y_train)

    predict = clf.predict(X_test_discretized)
    accuracy = accuracy_score(y_test, predict)

    print(f"Largura de Janela: {bandwidth}")
    print(f"Média: {accuracy:.4f}")
    print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Largura de Janela: 0.001
Média: 0.6333
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.01
Média: 0.7167
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 0.1
Média: 0.6333
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1
Média: 0.7333
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 10
Média: 0.7500
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 100
Média: 0.6667
Matriz de Confusão:
[[49  0]
 [ 0 11]]

Largura de Janela: 1000
Média: 0.6833
Matriz de Confusão:
[[49  0]
 [ 0 11]]
```

5) Utilizando o Naive Bayes assumindo que cada atributo segue uma distribuição normal (univariada).

6) Utilizando o classificador Baeysiano com estimação de probabilidade via distribuição normal multivariada.

```python
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from scipy.stats import sem, t

# breast-cancer-wisconsin

X_train, X_test, y_train, y_test = train_test_split(data_X, data_y, test_size=0.3)

clf = QuadraticDiscriminantAnalysis()
clf.fit(X_train, y_train)

predict = clf.predict(X_test)
accuracy = accuracy_score(y_test, predict)
matrix = confusion_matrix(y_test, predict)

n = len(y_test)
mean = accuracy
std = sem([1 if p == t else 0 for p, t in zip(predict, y_test)])
interval = std * t.ppf((1 + 0.95) / 2, n - 1)

print(f"Acurácia: {accuracy:.4f}")
print(f"Média: {mean:.4f}")
print(f"Desvio Padrão: {std:.4f}")
print(f"Intervalo de Confiança: {mean - interval:.4f}  {mean + interval:.4f}")
print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Acurácia: 0.9238
Média: 0.9238
Desvio Padrão: 0.0184
Intervalo de Confiança: 0.8876  0.9600
Matriz de Confusão:
[[128  12]
 [  4  66]]
```

```python
from sys import stdout
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from scipy.stats import sem, t

# wdbc

X_train, X_test, y_train, y_test = train_test_split(data2_X, data2_y, test_size=0.3)

clf = QuadraticDiscriminantAnalysis()
clf.fit(X_train, y_train)

predict = clf.predict(X_test)
accuracy = accuracy_score(y_test, predict)
matrix = confusion_matrix(y_test, predict)

n = len(y_test)
mean = accuracy
std = sem([1 if p == t else 0 for p, t in zip(predict, y_test)])
interval = std * t.ppf((1 + 0.95) / 2, n - 1)

print(f"Acurácia: {accuracy:.4f}")
print(f"Média: {mean:.4f}")
print(f"Desvio Padrão: {std:.4f}")
print(f"Intervalo de Confiança: {mean - interval:.4f}  {mean + interval:.4f}")
print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Acurácia: 0.9298
Média: 0.9298
Desvio Padrão: 0.0196
Intervalo de Confiança: 0.8912  0.9685
Matriz de Confusão:
[[100  10]
 [  2  59]]
```

```python
from sys import stdout
from sklearn.discriminant_analysis import QuadraticDiscriminantAnalysis
from scipy.stats import sem, t

# wpbc

X_train, X_test, y_train, y_test = train_test_split(data3_X, data3_y, test_size=0.3)

clf = QuadraticDiscriminantAnalysis()
clf.fit(X_train, y_train)

predict = clf.predict(X_test)
accuracy = accuracy_score(y_test, predict)
matrix = confusion_matrix(y_test, predict)

n = len(y_test)
mean = accuracy
std = sem([1 if p == t else 0 for p, t in zip(predict, y_test)])
interval = std * t.ppf((1 + 0.95) / 2, n - 1)

print(f"Acurácia: {accuracy:.4f}")
print(f"Média: {mean:.4f}")
print(f"Desvio Padrão: {std:.4f}")
print(f"Intervalo de Confiança: {mean - interval:.4f}  {mean + interval:.4f}")
print(f"Matriz de Confusão:\n{matrix}\n")
```

```
Acurácia: 0.82
Média: 0.82
Desvio Padrão: 0.06
Intervalo de Confiança: 0.69 a 0.95
Matriz de Confusão:
[[31  0]
 [ 7  1]]
```