dim032-classification

August 19, 2020

1 0. Introdução

Trabalho Clustering:

Aluno: Gabriel Luiz

Disciplina: Tópico em Aprendizado de Máquina

Objetivos :

- Escolha dois datasets rotulados.
- Realize a análise estatística, visualização e pré-processamento dos dados.
- Realize os experimentos criando duas bases de teste distintas:
- – considerando todos os atributos do dataset;
- – selecionando alguns atributos e descartando outros.
- Aplique três métodos de classificação distintos nas duas bases acima referentes a cada dataset.
- Para cada dataset, em cada uma das bases, analise os resultados segundo medidas de qualidade de classificação, usando índices de validação externa (acurácia, recall, precisão, F-measure, índice Kappa) e cruva ROC.
- Proponha uma maneira adicional de comparar os resultados obtidos além das medidas acima.
- Compare e interprete os resultados dos dois experimentos em cada dataset. Faça tabela com as medidas de validação

1.1 0.1 Dependências

Para realização da tarefa foram utilizados as seguintes bibliotecas:

[202]:
```
from datetime import datetime
import numpy as np
import pandas as pd
from sklearn.cluster import *
import seaborn as sns
from sklearn.preprocessing import StandardScaler
import matplotlib.pyplot as plt
from sklearn.feature_selection import SelectKBest
```

```python
from sklearn.feature_selection import chi2
from sklearn.model_selection import train_test_split

# KFold
from sklearn.model_selection import KFold
import random

# Classificadores
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.tree import DecisionTreeClassifier
from sklearn.svm import SVC

#Metricas
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import precision_score
from sklearn.metrics import f1_score
from sklearn.metrics import cohen_kappa_score
from  sklearn.metrics  import  roc_auc_score
from sklearn.metrics import balanced_accuracy_score
```

2 1. Dados

Para realização das tarefas envolvidas neste relatório utilizou-se o arquivo **dim032.csv** que contém dados não descritos, onde foram feitos para a realização de clustering que se encontram no site: http://cs.uef.fi/sipu/datasets/

2.1 1.1 Carregamento do arquivo

[203]:
```python
from clustering.labelMatch import rotulos, labelmatch
dataset = './dataset/dim032/dim032.csv'
clusters = './dataset/dim032/dim032-pa.csv'
```

[204]:
```python
data = pd.read_csv(
    dataset,
    header = None
    )

label = pd.read_csv(
    clusters,
    header = None
    )
```

[205]:
```
data.head()
```

[205]:
```
    0    1    2   3    4    5    6    7   8    9  ...   22  23   24   25  26  \
0  84  152  100  52   95  186  169  106  37  186  ...  190  65  214  116  75
1  86  149  101  56   93  181  171  116  37  192  ...  191  79  215  116  76
2  83  149   99  51   96  187  169  108  34  191  ...  190  65  213  118  73
3  86  142  101  64  105  183  172  116  49  180  ...  186  69  209  120  68
4  89  145  108  54   91  180  175  107  35  192  ...  188  67  212  118  91

   27   28  29   30   31
0  55  123  65  154  177
1  60  130  71  151  181
2  55  125  63  155  178
3  56  123  67  144  181
4  50  135  58  147  165

[5 rows x 32 columns]
```

[206]:
```
data.describe()
```

[206]:
```
                 0            1            2            3            4  \
count  1024.000000  1024.000000  1024.000000  1024.000000  1024.000000
mean     95.626953   109.116211   112.750000   127.612305   139.097656
std      33.615901    56.908917    51.135914    48.141948    59.470162
min      30.000000    40.000000    40.000000    41.000000    28.000000
25%      73.000000    56.000000    72.000000    81.750000    88.000000
50%      88.500000    97.000000    97.000000   142.000000   169.000000
75%     121.000000   145.000000   168.000000   162.000000   186.000000
max     162.000000   219.000000   217.000000   217.000000   218.000000

                 5            6            7            8            9  ...  \
count  1024.000000  1024.000000  1024.000000  1024.000000  1024.000000  ...
mean    130.491211   142.145508   134.344727    97.023438   135.126953  ...
std      39.287918    45.671907    59.378414    42.142075    66.366363  ...
min      48.000000    48.000000    25.000000    24.000000    29.000000  ...
25%     104.000000   106.000000    79.000000    63.000000    58.500000  ...
50%     129.000000   159.000000   145.000000    85.000000   169.500000  ...
75%     150.000000   171.000000   188.750000   134.750000   187.000000  ...
max     225.000000   220.000000   229.000000   174.000000   222.000000  ...

                22           23           24           25           26  \
count  1024.000000  1024.000000  1024.000000  1024.000000  1024.000000
mean    120.544922   154.849609   123.900391   123.157227   105.608398
std      67.089616    60.070835    58.308579    55.723743    48.049909
min      29.000000    39.000000    28.000000    25.000000    24.000000
25%      53.000000   118.750000    69.000000    87.500000    61.000000
50%     111.500000   176.000000   117.500000   116.000000   113.000000
```

```
75%      192.000000   207.000000   181.000000   179.750000   143.250000
max      223.000000   235.000000   222.000000   218.000000   208.000000

                 27           28           29           30           31
count   1024.000000  1024.000000  1024.000000  1024.000000  1024.000000
mean     122.179688   130.062500   130.897461   106.218750   116.990234
std       58.800397    61.676195    55.330114    47.630102    55.882102
min       28.000000    40.000000    51.000000    41.000000    34.000000
25%       56.000000    64.000000    88.000000    67.000000    74.000000
50%      138.000000   143.000000   118.500000   102.000000    97.000000
75%      169.750000   189.000000   182.250000   136.750000   162.750000
max      219.000000   226.000000   227.000000   218.000000   223.000000

[8 rows x 32 columns]
```

[ ]:
```
# 2. Pré-processamento
```

Validações efetivadas:

- 1. Dados faltantes representados por "NaN"
- 2. Dados que não possuem valores númericos

[207]:
```
# data.isna().sum()
```

[208]:
```
# for col in data:
    # print(col, data[col].unique())
```

2.1 Conclusão:

- Os dados não possuem a necessidade de pré-processamento visto que já estão todos com valores validos

2.1.1 2.3 Análise estatística

[209]:
```
data.corr()
```

[209]:
```
          0         1         2         3         4         5         6  \
0  1.000000  0.268198 -0.051122 -0.068849  0.599398 -0.438830  0.041834
1  0.268198  1.000000 -0.434193  0.065247  0.147223 -0.087154  0.052960
2 -0.051122 -0.434193  1.000000 -0.212930  0.077845  0.065985 -0.022429
3 -0.068849  0.065247 -0.212930  1.000000 -0.049977 -0.004621  0.348210
4  0.599398  0.147223  0.077845 -0.049977  1.000000 -0.512794 -0.189263
5 -0.438830 -0.087154  0.065985 -0.004621 -0.512794  1.000000  0.549921
6  0.041834  0.052960 -0.022429  0.348210 -0.189263  0.549921  1.000000
7  0.122806  0.112432 -0.216804  0.042810  0.334837 -0.635187 -0.410581
```

8	0.140755	0.227755	-0.180707	0.093820	0.483320	-0.236149	-0.194464
9	0.017996	0.442574	-0.143382	0.186358	0.257335	-0.517697	-0.334774
10	-0.416168	-0.247372	-0.078198	-0.256024	-0.421847	0.422388	0.085398
11	-0.325456	0.049006	-0.005908	-0.023452	-0.093708	0.000218	-0.346374
12	0.149802	-0.101557	-0.327047	0.109485	0.133371	-0.173939	0.251279
13	0.494533	0.135398	-0.275698	0.165248	0.393054	-0.037131	0.037670
14	0.054404	0.253658	0.027675	0.226054	0.026300	0.018823	0.180490
15	-0.029940	-0.281726	-0.376820	-0.159598	-0.033689	-0.125013	0.108979
16	0.076611	-0.338553	0.435520	-0.105110	0.021825	0.339884	0.230447
17	-0.302741	-0.274373	0.298525	-0.188220	-0.401813	0.141096	-0.093745
18	0.527674	0.629009	-0.384632	0.489437	0.561829	-0.389271	0.212298
19	0.320353	-0.094752	0.120599	0.444076	0.131633	-0.174375	0.417336
20	-0.479669	-0.452581	0.081423	0.053097	-0.336928	0.405705	0.502963
21	0.148475	0.161413	-0.386965	-0.059567	0.288114	-0.083627	-0.269380
22	0.298212	0.218035	0.142550	0.056818	0.427911	0.125967	0.240729
23	0.022004	-0.323157	-0.167976	0.125137	-0.102514	-0.170035	-0.315245
24	0.193500	0.388152	-0.479971	0.050242	0.071684	0.165715	0.261003
25	-0.007820	-0.007483	-0.183073	-0.082015	0.042287	-0.463263	-0.589061
26	-0.391960	-0.222688	0.094760	-0.042200	-0.246345	-0.138948	-0.626816
27	0.478101	0.224718	-0.298575	-0.079141	0.260857	0.029176	0.142038
28	-0.658871	0.126030	0.080989	0.141606	-0.326250	0.013689	-0.162874
29	0.275498	0.268201	-0.458842	0.276326	0.085673	-0.169076	0.393038
30	-0.166015	-0.014487	0.172647	0.313186	0.004851	0.132724	0.493069
31	0.234555	0.132197	-0.408891	-0.219960	0.082626	0.028978	0.155258

	7	8	9	…	22	23	24	25	\
0	0.122806	0.140755	0.017996	…	0.298212	0.022004	0.193500	-0.007820	
1	0.112432	0.227755	0.442574	…	0.218035	-0.323157	0.388152	-0.007483	
2	-0.216804	-0.180707	-0.143382	…	0.142550	-0.167976	-0.479971	-0.183073	
3	0.042810	0.093820	0.186358	…	0.056818	0.125137	0.050242	-0.082015	
4	0.334837	0.483320	0.257335	…	0.427911	-0.102514	0.071684	0.042287	
5	-0.635187	-0.236149	-0.517697	…	0.125967	-0.170035	0.165715	-0.463263	
6	-0.410581	-0.194464	-0.334774	…	0.240729	-0.315245	0.261003	-0.589061	
7	1.000000	0.546772	0.702223	…	-0.178068	0.247391	0.277945	0.474793	
8	0.546772	1.000000	0.271853	…	-0.024533	0.151648	0.257148	0.219656	
9	0.702223	0.271853	1.000000	…	0.087693	-0.236956	0.198970	0.278513	
10	-0.092287	-0.240591	-0.328448	…	-0.033714	0.088643	0.031608	-0.376352	
11	0.120182	0.052129	0.429568	…	0.142397	-0.407174	-0.075290	0.217259	
12	0.409645	0.260425	0.078683	…	-0.473271	0.256056	0.430193	0.171557	
13	0.161548	0.404304	0.132367	…	0.173041	0.090210	0.280425	-0.214557	
14	-0.027761	0.383065	-0.032232	…	-0.243979	0.092168	0.168610	0.326534	
15	0.221851	-0.183596	-0.171017	…	-0.164908	0.150529	0.192498	-0.069413	
16	-0.337374	-0.054222	-0.653484	…	-0.010527	0.264512	-0.244634	-0.393127	
17	-0.366459	-0.809519	-0.130695	…	0.146363	0.046578	-0.353528	-0.293029	
18	0.233610	0.380397	0.387718	…	0.400243	-0.182152	0.287166	-0.171046	
19	0.085567	-0.114620	0.213432	…	0.440008	-0.035431	0.041517	-0.244667	
20	-0.158576	-0.371150	-0.320810	…	-0.065699	-0.132771	0.109687	-0.225979	

21	-0.064146	0.156344	-0.158748	…	-0.110877	0.414004	0.017234	0.157122
22	-0.178068	-0.024533	0.087693	…	1.000000	-0.369970	-0.059085	-0.624774
23	0.247391	0.151648	-0.236956	…	-0.369970	1.000000	0.069852	0.399422
24	0.277945	0.257148	0.198970	…	-0.059085	0.069852	1.000000	0.221850
25	0.474793	0.219656	0.278513	…	-0.624774	0.399422	0.221850	1.000000
26	0.290873	0.184472	0.186412	…	-0.701118	0.442418	-0.092619	0.698714
27	-0.147543	0.315283	-0.334449	…	-0.193228	0.013597	0.388245	-0.020877
28	-0.145856	-0.126491	0.165812	…	-0.079521	-0.058917	-0.264124	0.123476
29	0.229722	0.228800	-0.066543	…	-0.320609	0.253746	0.468917	0.079350
30	0.120050	-0.127096	0.323513	…	0.285268	-0.297381	0.211759	-0.357671
31	-0.121036	-0.001655	-0.220399	…	-0.163059	0.283402	0.306188	0.149484

	26	27	28	29	30	31
0	-0.391960	0.478101	-0.658871	0.275498	-0.166015	0.234555
1	-0.222688	0.224718	0.126030	0.268201	-0.014487	0.132197
2	0.094760	-0.298575	0.080989	-0.458842	0.172647	-0.408891
3	-0.042200	-0.079141	0.141606	0.276326	0.313186	-0.219960
4	-0.246345	0.260857	-0.326250	0.085673	0.004851	0.082626
5	-0.138948	0.029176	0.013689	-0.169076	0.132724	0.028978
6	-0.626816	0.142038	-0.162874	0.393038	0.493069	0.155258
7	0.290873	-0.147543	-0.145856	0.229722	0.120050	-0.121036
8	0.184472	0.315283	-0.126491	0.228800	-0.127096	-0.001655
9	0.186412	-0.334449	0.165812	-0.066543	0.323513	-0.220399
10	-0.179221	-0.058145	-0.051353	0.065142	-0.060821	-0.216716
11	0.137810	-0.437226	0.259213	-0.526618	-0.172597	-0.439803
12	0.094848	0.344143	-0.423074	0.692584	0.200866	0.511520
13	-0.067142	0.508689	-0.684955	0.144636	-0.021558	0.228797
14	0.289699	0.241383	0.249653	0.154281	0.139658	0.214654
15	-0.244122	0.184142	-0.206692	0.347151	0.203000	0.091169
16	-0.020745	0.313308	-0.291156	0.051421	0.019573	-0.032069
17	-0.079840	-0.493866	0.325377	-0.354793	0.128913	-0.114960
18	-0.470166	0.358691	-0.076450	0.453114	0.222688	0.016510
19	-0.368946	-0.257975	-0.060471	-0.023146	0.636369	-0.017978
20	-0.251045	-0.177907	0.147623	0.103378	0.483521	-0.185867
21	0.138043	0.250837	-0.137726	0.282940	-0.534742	0.646400
22	-0.701118	-0.193228	-0.079521	-0.320609	0.285268	-0.163059
23	0.442418	0.013597	-0.058917	0.253746	-0.297381	0.283402
24	-0.092619	0.388245	-0.264124	0.468917	0.211759	0.306188
25	0.698714	-0.020877	0.123476	0.079350	-0.357671	0.149484
26	1.000000	-0.114451	0.229124	-0.243960	-0.226023	-0.060403
27	-0.114451	1.000000	-0.466836	0.431696	-0.227496	0.233082
28	0.229124	-0.466836	1.000000	-0.220608	0.153451	-0.178395
29	-0.243960	0.431696	-0.220608	1.000000	-0.038159	0.414073
30	-0.226023	-0.227496	0.153451	-0.038159	1.000000	-0.191372
31	-0.060403	0.233082	-0.178395	0.414073	-0.191372	1.000000

[32 rows x 32 columns]

[210]:
```
df = data
df = df.assign(label = label)
test = df[[0, 1, 2, 3, 'label']]
test
```

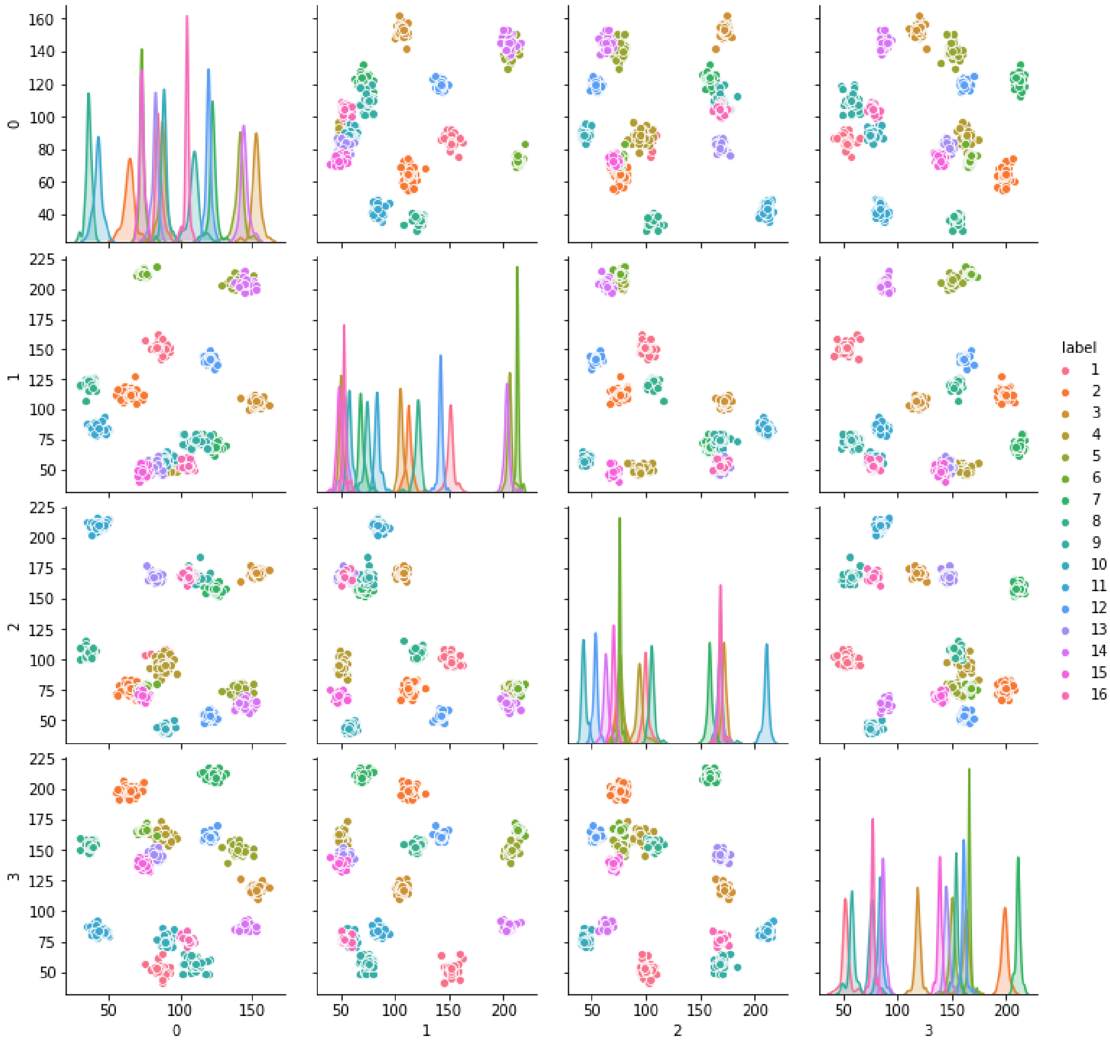
[210]:
```
        0    1    2   3  label
0      84  152  100  52      1
1      86  149  101  56      1
2      83  149   99  51      1
3      86  142  101  64      1
4      89  145  108  54      1
…    …   …   …   ..    …
1019  105   53  168  77     16
1020  104   53  169  77     16
1021  101   52  171  78     16
1022  106   59  165  74     16
1023  105   53  168  77     16

[1024 rows x 5 columns]
```

[ ]:
```
sns.pairplot(df, diag_kind="kde",hue='label')
```

[ ]:
```
<seaborn.axisgrid.PairGrid at 0x7fdd8a9bc710>
```

[211]:
```
sns.pairplot(test, diag_kind="kde",hue='label')
```

[211]:
```
<seaborn.axisgrid.PairGrid at 0x7fdd90af52e8>
```



2.1.2 2.4 Escalonando

Para aplicação dos algoritmos escalona-se os dados afim de parametriza-los num certo intervalor (-1 a 1)

[212]:
```
data = data.to_numpy()
scaler = StandardScaler().fit(data)
data_scaler = scaler.transform(data)
```

[213]:
```
# data_scaled = pd.DataFrame(data_scaler)
# data_scaled.head()
```

[214]:
```
data_results = np.array(label[0].tolist())
for idx, value in np.ndenumerate(data_results):
```

```
    data_results[idx] = value - 1
```

2.1.3 2.5 Plotando boxsplot

Pelo boxsplot é possivel visualizar que há alguns outliers.

[215]:
```
# data_scaled.plot(kind = 'box', figsize=(30,10), rot=90, )
```

2.2 3.2 Selecionando atributos do dataset

[216]:
```
data_reduzida = pd.DataFrame(SelectKBest(chi2, k=30).fit_transform(data, label))
data_reduzida.shape
data_reduzida = data_reduzida.to_numpy()

data_scaler2 = scaler.fit_transform(X = data_reduzida)
```

[217]:
```
# data_scaled2 = pd.DataFrame(data_scaler2)
# data_scaled2.head()
```

2.3 Classificando

2.4 Funções necessárias

[218]:
```
def calcula_metricas(metricas, y_test, y_predict):
    metricas['acc'] += (accuracy_score(y_test, y_predict))
    metricas['recall'] += (recall_score(y_test, y_predict, average='micro'))
    metricas['precision'] += (precision_score(y_test, y_predict,␣
 ↪average='macro'))
    metricas['f1'] += f1_score(y_test, y_predict, average='weighted')
    # metricas['roc'] += roc_auc_score(y_test, y_predict)
    metricas['kappa'] += cohen_kappa_score(y_test, y_predict)
    metricas['balanced_acc'] += balanced_accuracy_score(y_test, y_predict)
```

[219]:
```
def save_metricas(name, metricas):
    f = open(name, 'w')
    f.write('Acuária:' + str(metricas['acc']) + '\n')
    f.write('Recall:' + str(metricas['recall']) + '\n')
    f.write('Precisão:' + str(metricas['precision']) + '\n')
    f.write('F-Measure:' + str(metricas['f1']) + '\n')
    # f.write('Curva Roc:' + str(metricas['roc']) + '\n')
    f.write('Indice Kappa:' + str(metricas['kappa']) + '\n')
    f.write('Acuária Balanceada:' + str(metricas['balanced_acc']) + '\n')
    f.close()
```

[220]:
```
def show_metricas(metricas):
    print('Acuária:', metricas['acc'])
    print('Recall:', metricas['recall'])
    print('Precisão:', metricas['precision'])
    print('F-Measure:', metricas['f1'])
    # print('Curva Roc:', metricas['roc'])
    print('Indice Kappa:', metricas['kappa'])
    print('Acuária Balanceada:', metricas['balanced_acc'])
```

[221]:
```
def write_metricas(name_file, metricas, metodo):
    f = open(name_file, "a")
    f.write(metodo + ',')
    f.write(str(round(metricas['acc'],4)) + ',')
    f.write(str(round(metricas['recall'],4)) + ',')
    f.write(str(round(metricas['precision'],4)) + ',')
    f.write(str(round(metricas['f1'],4)) + ',')
    # f.write(str(round(metricas['roc'],4)) + ';')
    f.write(str(round(metricas['kappa'],4)) + ',')
    f.write(str(round(metricas['balanced_acc'],4)) + '\n')
    f.close()
```

2.5 Aplicando KNN com K-fold

2.6 DataFrame Cru

[222]:
```
formato = 'Cru'

folds_value = 16
```

[223]:
```
# TODO change split function
kf = KFold(n_splits=2, shuffle=True, random_state=random.randint(0, 10))
data_kfold = kf.split(data)

train = []
test = []

for train_index, test_index in data_kfold:
    train.append(train_index)
    test.append(test_index)
```

[224]:
```
name_file = 'metricas.csv'

# Roc;
f = open(name_file, "w")
f.write('Acurácia,Recall,Precisão,F1,Kappa,Acurácia Balanceada\n')
f.close()
```

2.7 Aplicando KNN com K-fold

[225]:
```
# 'roc': 0,
metodo = 'KNN'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'kappa': 0,␣
 ↪'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

neigh = KNeighborsClassifier(n_neighbors=10)
neigh.fit(x_train, y_train)

y_predict = neigh.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[226]:
```
# # 'roc': 0,
# metodo = 'KNN'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'kappa': 0,␣
 ↪'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler[train_index], data_scaler[test_index]
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     neigh = KNeighborsClassifier(n_neighbors=100)
#     neigh.fit(x_train, y_train)
#
#     y_predict = neigh.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.8 Aplicando GaussianNB com K-fold

[227]:
```
metodo = 'Gauss'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
↪0, 'balanced_acc': 0}


x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

gauss = GaussianNB()
gauss.fit(x_train, y_train)

y_predict = gauss.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[228]:
```
# metodo = 'Gauss'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler[train_index], data_scaler[test_index]
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     gauss = GaussianNB()
#     gauss.fit(x_train, y_train)
#
#     y_predict = gauss.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.9 Aplicando DecisionTreeClassifier com K-fold

[229]:
```
metodo = 'Tree'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train)

y_predict = tree.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.9921875
Recall: 0.9921875
Precisão: 0.9930555555555556
F-Measure: 0.9922002655228759
Indice Kappa: 0.9916421808684296
Acuária Balanceada: 0.993421052631579
```

[230]:
```
# metodo = 'Tree'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler[train_index], data_scaler[test_index]
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     tree = DecisionTreeClassifier()
#     tree.fit(x_train, y_train)
#
#     y_predict = tree.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.10 Aplicando SVM com K-fold

[231]:
```
metodo = 'SVM'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

svm = SVC()
svm.fit(x_train, y_train)

y_predict = svm.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[232]:
```
# metodo = 'SVM'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler[train_index], data_scaler[test_index]
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     svm = SVC()
#     svm.fit(x_train, y_train)
#
#     y_predict = svm.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.11 DataFrame Selecionado

2.12 Aplicando

[233]:
```
kf = KFold(n_splits=2, shuffle=True, random_state=random.randint(0, 10))
data_kfold = kf.split(data_scaler2)

train = []
test = []

for train_index, test_index in data_kfold:
    train.append(train_index)
    test.append(test_index)
```

2.13 Aplicando KNN com K-fold

[234]:
```
metodo = 'KNNSELECT'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
 ↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

neigh = KNeighborsClassifier(n_neighbors=10)
neigh.fit(x_train, y_train)

y_predict = neigh.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[235]:
```
# metodo = 'KNNSELECT'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
 ↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler2[train_index], data_scaler2[test_index]
```

```
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     neigh = KNeighborsClassifier(n_neighbors=20)
#     neigh.fit(x_train, y_train)
#
#     y_predict = neigh.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.14 Aplicando GaussianNB com K-fold

[236]:
```
metodo = 'GaussSELECT'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
 ↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

gauss = GaussianNB()
gauss.fit(x_train, y_train)

y_predict = gauss.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[237]:
```
# metodo = 'GaussSELECT'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
 ↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler2[train_index], data_scaler2[test_index]
```

```
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     gauss = GaussianNB()
#     gauss.fit(x_train, y_train)
#
#     y_predict = gauss.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.15 Aplicando DecisionTreeClassifier com K-fold

[238]:
```
metodo = 'TreeSELECT'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

tree = DecisionTreeClassifier()
tree.fit(x_train, y_train)

y_predict = tree.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 0.984375
Recall: 0.984375
Precisão: 0.9844909750337382
F-Measure: 0.9845306385931386
Indice Kappa: 0.9832720738381115
Acuária Balanceada: 0.984046743697479
```

[239]:
```
# metodo = 'TreeSELECT'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler2[train_index], data_scaler2[test_index]
```

```
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     tree = DecisionTreeClassifier()
#     tree.fit(x_train, y_train)
#
#     y_predict = tree.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

2.16 Aplicando SVM com K-fold

[240]:
```
metodo = 'SVMSELECT'
metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':␣
↪0, 'balanced_acc': 0}

x_train, x_test, y_train, y_test = train_test_split(data_scaler, data_results)

svm = SVC()
svm.fit(x_train, y_train)

y_predict = svm.predict(x_test)

calcula_metricas(metricas, y_test, y_predict)

show_metricas(metricas)
write_metricas(name_file, metricas, metodo)
```

```
Acuária: 1.0
Recall: 1.0
Precisão: 1.0
F-Measure: 1.0
Indice Kappa: 1.0
Acuária Balanceada: 1.0
```

[241]:
```
# metodo = 'SVMSELECT'
# metricas = {'acc': 0, 'recall': 0, 'precision': 0, 'f1': 0, 'roc': 0, 'kappa':
↪ 0, 'balanced_acc': 0}
#
# for train_index, test_index in zip(train, test):
#     x_train, x_test = data_scaler2[train_index], data_scaler2[test_index]
```

```
#     y_train, y_test = data_results[train_index], data_results[test_index]
#
#     svm = SVC()
#     svm.fit(x_train, y_train)
#
#     y_predict = svm.predict(x_test)
#
#     calcula_metricas(metricas, y_test, y_predict)
#
# for metrica, value in metricas.items():
#     metricas[metrica] = value/10
#
# show_metricas(metricas)
# write_metricas(name_file, metricas, metodo)
```

[242]:

```
analise = './metricas.csv'
metricas = pd.read_csv(
    analise,
    )
metricas
```

[242]:

	Acurácia	Recall	Precisão	F1	Kappa	Acurácia Balanceada
KNN	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Gauss	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
Tree	0.9922	0.9922	0.9931	0.9922	0.9916	0.9934
SVM	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
KNNSELECT	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
GaussSELECT	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000
TreeSELECT	0.9844	0.9844	0.9845	0.9845	0.9833	0.9840
SVMSELECT	1.0000	1.0000	1.0000	1.0000	1.0000	1.0000