# Projeto Aprendizado de Máquina
# Comparação de Classificadores

Gabriel Rossetto Migliorini 11201721812
Gustavo Mantovani Fonseca 11201921803
Janusa Maia 11047211
Jaqueline Victal de Almeida 11034316

30 de novembro de 2021

# Sumário

https://drive.google.com/drive/folders/1pEsoOlAXmntlGgW0ZVfpqj_dcQvwnePh?usp=sharing

# 1 Introdução

Como apresentação final, esse trabalho se propõe a analisar métricas em dois diferentes algoritmos. O primeiro deles é um gerador de hipóteses gerais e estritas. O segundo, mais robusto, é o SVC (support vector classifier).

# 2 Metodologia

O comparativo será baseado nas métricas de acurácia e precisão (ainda que faça sentido utilizar demais métricas ex: F1).

# 3 Resultados

Como detalhado no relatório da Etapa 2, para utilizar o gerador de hipóteses era necessário discretizar variáveis contínuas. Para isso, uma vez criados os histogramas dessas variáveis, utilizamos a largura dos intervalos (bins) para gerar classes das mesmas. Com isso, foi possível converter um grande intervalo numérico em algumas classes. Por exemplo, no histograma relacionado ao percentual de Rap cantado na música temos 10 classes (A J), cada uma representando um intervalo de 10% - O algoritmo completo encontra-se disponível no Apêndice A.

Os resultados obtidos não são muito promissores. Isso porque as respostas encontradas foram:

Ouvinte 1:
**g** = []
**s** = ['?', '?', '?', '?', '?', '?', '?', **False**, '?', '?', '?', '?', '?', '?', '?']

No caso do ouvinte 1, a hipótese encontrada é de que qualquer música sem instrumentação de madeira seria recomendada.

Ouvinte 2:
**g** = [['?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '?', '**A**', '?', '?']]

**s = [’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ’?’, ‘A’, ’?’, ’?’]**

Ou seja, a hipótese encontrada é que qualquer música que possua (0 10%) de Rap seria recomendada para esse indivíduo.

Em ambos os casos é evidente que o ganho de informação é muito baixo em relação ao poder computacional utilizado. Quantitativamente, a precisão e acurácia para a classificação do Ouvinte1 foram, respectivamente: **0,24 e 0,44**.

## 3.1 SVC

Como dito anteriormente, o SVC é um algoritmo com um maior poder em relação ao primeiro. Nesse classificador, iremos identificar quais os melhores kernels e índice de penalidade ao erro (C) utilizando validação cruzada. Dito isso, a primeira etapa consiste em comparar os scores dos usuários em duas condições: ’crus’ e normalizados. A figura abaixo apresenta os resultados.

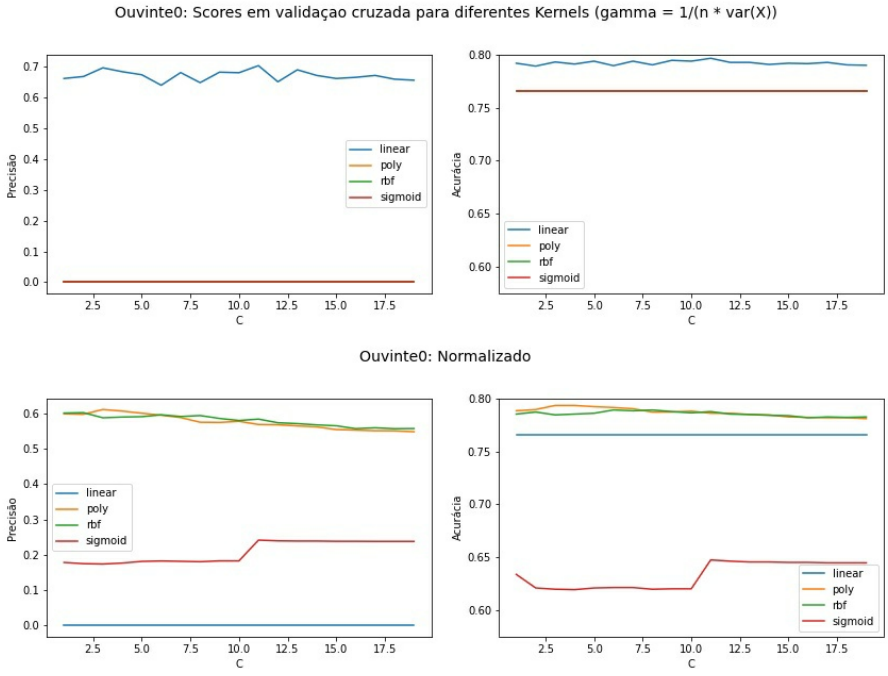


Figura 1: Scores para Ouvinte 0

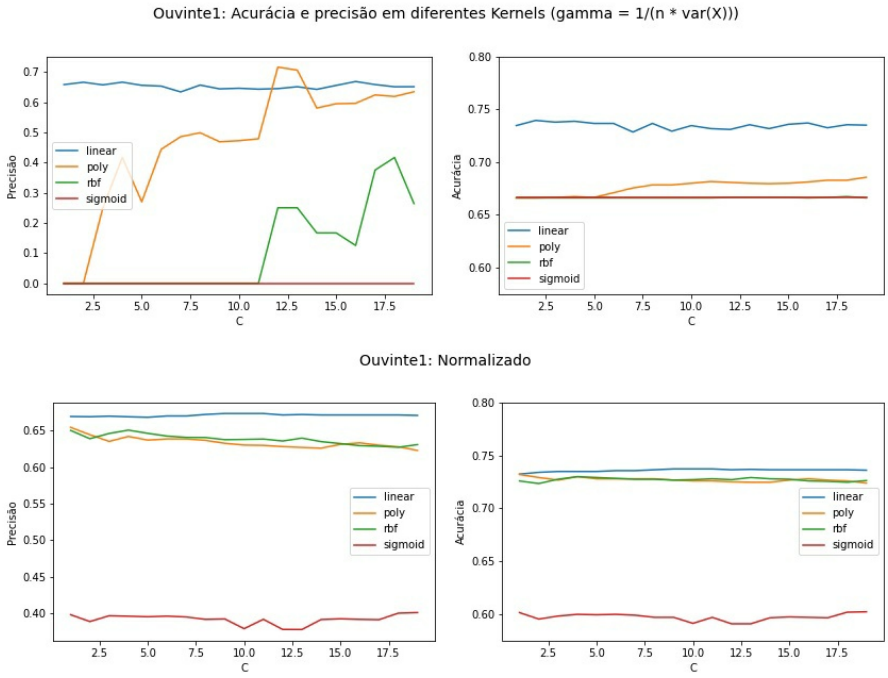


Figura 2: Scores para Ouvinte 1

| C | n_coef | Acuracia | Precisao |
|---|---|---|---|
| 3 | 3 | 0.793603 | 0.611735 |
| 4 | 3 | 0.793604 | 0.607115 |
| 5 | 3 | 0.792433 | 0.601091 |
| 8 | 2 | 0.790868 | 0.612528 |
| 9 | 2 | 0.790478 | 0.610194 |
| 12 | 2 | 0.790479 | 0.605601 |
| 13 | 2 | 0.790089 | 0.603794 |
| 14 | 2 | 0.790479 | 0.606041 |
| 15 | 2 | 0.790479 | 0.606415 |
| 16 | 2 | 0.791260 | 0.608766 |
| 17 | 2 | 0.790870 | 0.605568 |
| 18 | 2 | 0.790091 | 0.601345 |
| 19 | 2 | 0.790091 | 0.600525 |

Figura 3: Melhores classificadores polinomiais

Com base nesses dados temos duas alternativas. A primeira delas, utilizar um kernel linear em conjunto com dados não normalizados e, com isso, obter uma média aproximada de $(.7+.65)/2 =$ **.675** para precisão e $(.73+.79)/2 =$ **.76** para acurácia. A segunda opção seria utilizar o kernel polinomial, com os dados normalizados. A média aproximada dos melhores scores foram: $(.6+.65)/2 =$ **.625** para precisão e $(.73+.78)/2 =$ **.755** para acurácia.

Podemos notar embora a acurácia do modelo linear e polinomial sejam parecidas a precisão do modelo linear foi melhor. Contudo, é possível que exita um melhor classificador polinomial, uma vez que ainda não exploramos o parâmetrô referente aos graus de liberdade. Afim de verificar isso utilizamos o classificador no ouvinte 0 variando o grau de liberdade em conjunto com C. O resultado dos melhores classificadores encontra-se na figura 3.

Embora tenhamos encontrado distintos classificadores na mesma faixa do anterior, nenhum deles é significativamente maior em relação às metricas. Sendo assim, a melhor opção de algoritmo para recomendação analisada aqui é SVC linear com C = [2, 3 ou 11].

# 4 Apêndice A

```
1
2  def criaClasses(df_slice):
3
4      _ , edges = np.histogram(df_slice)
5
6     #cria nova lista
7     temp = df_slice.copy()
8
9     #cria classes
10    nbins   = len(edges)-1
11    classes = list(map(chr, range(65, 65+nbins)))
12
13    for i in range(nbins):
14
15        idx = df_slice.index[(df_slice >= edges[i]) & (df_slice <
      edges[i+1])]
16        temp[idx] = classes[i]
17
18
19    return temp
20
21
22 def Gpositivo(g, values, size):
23
24    #n hipoteses g
25    gHip  = len(g)
26    index = []
27
28    #percorre hipoteses
29    for i in range(gHip):
30
31        #percorre atributos
32        for j in range(size):
33
34            #valida
35            if g[i][j] != '?' and g[i][j] != values[j]:
36                index.append(i)
37                break
38
39    #remove inconsistencias
40    for idx in index:
41        g.pop(idx)
42
43    return g
```

```
44
45
46  def Spositivo(s, values):
47
48      temp = s.copy()
49      for i in range(len(s)):
50
51          #generaliza atributo
52          if s[i] == '-0-':
53              temp[i] = values[i]
54
55          elif s[i] != values[i]:
56              temp[i] = '?'
57
58      #verifica generalizade de h
59      counth = sum([1 for i in temp if i=='?'])
60
61      #verifica generalidade das g
62      for gHip in g:
63          countg = sum([1 for i in gHip if i=='?'])
64
65          #atualiza
66          if countg > counth:
67              s = temp
68              break
69
70      return s
71
72
73  def Snegativo(s, values):
74
75      #percorre hipoteses
76      for i in range(len(s)):
77
78          #elimina s
79          count=0
80          if s[i] == values[i] or s[i] == '?':
81                  count+=1
82
83      if count == len(s):
84          s = ["-0-" for i in range(size)]
85
86      return s
87
88
89  def Gnegativo(g, values, lista, size):
```

```
90
91      ##Elimina g##
92
93      #n hipoteses g
94      gHip = len(g)
95
96      #percorre hipoteses
97      for i in range(gHip):
98
99          #percorre atributos
100         for j in range(size):
101
102             #valida
103             if g[i][j] == '?' or g[i][j] != values[j]:
104                 g.pop(i)
105                 break
106
107
108     ##Adiciona g##
109
110     #gera minimos
111     size = len(lista)
112
113     #percorre atributos
114     for i in range(size):
115
116         #percorre elementos
117         for j in range(len(lista[i])):
118
119             #valida com h
120             if lista[i][j] != values[i]:
121
122                 if lista[i][j] == s[i]:
123                     #cria novas hipoteses g
124                     aux = ["?" for i in range(size)]
125                     aux[i] = lista[i][j]
126                     #atualiza g
127                     g.append(aux.copy())
128
129     return g
130
131 ########
132 # MAIN #
133 ########
134
135 #Discretiza vari veis cont nuas
```

```
136 colunasAjuste = ['BPM', 'VolMedio', 'PctCantada', 'PctRap', '
    ano_lancamento', 'n_reproducao']
137
138 for coluna in colunasAjuste:
139     ouvinte1[coluna] = criaClasses(ouvinte1[coluna])
140     ouvinte2[coluna] = criaClasses(ouvinte2[coluna])
141
142 ouvinte1.head(3)
143
144 #Cria lista total de atributos
145 colunas = ouvinte1.columns[:-1]
146
147 lista = []
148 for coluna in colunas:
149     lista.append(list(set(ouvinte1[coluna])))
150
151
152 #Ordena objetos positivos
153 ouvinte1['ordena'] = (ouvinte1['gostou'] == 0) + 1
154 ouvinte1 = ouvinte1.sort_values('ordena')
155 ouvinte1.drop(columns=['ordena'], inplace=True)
156
157 ouvinte2['ordena'] = (ouvinte2['gostou'] == 0) + 1
158 ouvinte2 = ouvinte2.sort_values('ordena')
159 ouvinte2.drop(columns=['ordena'], inplace=True)
160
161 #Main
162 dft   = ouvinte1
163 size = len(dft.iloc[0]) - 1
164
165 g = [["?" for i in range(size)]]
166 s = ["-0-" for i in range(size)]
167
168
169 for i in range(len(dft)):
170
171     #pega valores
172     values = dft.iloc[i].values
173
174
175     #Exemplo POSITIVO
176     #---------------
177     if (values[-1] == True):
178
179         values = values[:-1]
180
```

```
181         #Atualiza G
182         g = Gpositivo(g, values, size)
183 
184         #Atualiza S
185         s = Spositivo(s, values)
186 
187 
188     #Exemplo NEGATIVO
189     #----------------
190     else:
191 
192         values = values[:-1]
193 
194         #Atualiza S
195         s = Snegativo(s, values)
196 
197         #Atualiza G
198         g = Gnegativo(g, values, lista, size)
199 
200 print(g)
201 print(s)
202 
203 #############
204 # METRICAS #
205 #############
206 
207 #Predictions
208 TP = ouvinte1[(ouvinte1.Tem_Instr_Cordas == False) & (ouvinte1.
    gostou == True) ]
209 FP = ouvinte1[(ouvinte1.Tem_Instr_Cordas == False) & (ouvinte1.
    gostou == False)]
210 FN = ouvinte1[(ouvinte1.Tem_Instr_Cordas == True)  & (ouvinte1.
    gostou == True) ]
211 TN = ouvinte1[(ouvinte1.Tem_Instr_Cordas == True)  & (ouvinte1.
    gostou == False)]
212 
213 precisao = len(TP) / (len(TP) + len(FP))
214 accuracy = (len(TP) + len(TN)) / (len(TP) + len(FP) + len(FN) + len
    (TN))
215 print(precisao, accuracy)
```

# 5 Apêndice B

```
def pre_tratamento(df):

    #renomeia colunas
    colNames        = list(df.columns)
    colNames[-2]    = 'gostou'
    colNames[-3]    = 'n_reproducao'

    df = df.reindex(columns=colNames)


    #Cria Ouvintes
    ouvintes = []
    for idNum in set(df.id_cliente):
        ouvinte = df[df.id_cliente == idNum]
        ouvintes.append(ouvinte.drop(columns='id_cliente'))

    return df, ouvintes


def inconsistencia(df):

    #lista sets e/ou bordas de variaveis
    for i in range(len(df.columns)):

        if type(df[df.columns[i]][0]) != bool:
            print(df.columns[i], ':', max(df[df.columns[i]]), min(
    df[df.columns[i]]))
        else:
            print(df.columns[i], ':',  set(df[df.columns[i]][:5]))


def variavel_categorica(df):

    from unidecode import unidecode as decode

    #valores da variavel 'bateria'
    bateria = ['Eletr nica', 'Ac stica', 'Nenhuma']

    #discretiza
    for tipo in bateria:

        df[decode(tipo)] = (df['bateria'] == tipo)

    return df
```

```
44
45
46 def normaliza(df):
47
48     #normaliza df
49     df_min   = df.min().astype(np.float32)
50     df_max   = df.max().astype(np.float32)
51     df_range = df_max - df_min
52
53     df_scaled = (df - df_min) / df_range
54
55     return df_scaled
56
57
58 ########
59 # MAIN #
60 ########
61
62 #geral
63 import pandas as pd
64 import numpy  as np
65 import matplotlib.pyplot as plt
66
67 #aprendizado
68 from sklearn.model_selection import train_test_split
69 from sklearn.model_selection import cross_validate
70 from sklearn import svm
71
72 df = pd.read_csv('../JacquelineVictal_dados_treino.csv')
73
74 df, ouvintes = pre_tratamento(df)
75
76 df = inconsistencia(df)
77
78 #ajusta variavel
79 for i in range(len(ouvintes)):
80     ouvinte = variavel_categorica(ouvintes[i])
81     ouvintes[i].drop(['bateria'], axis=1, inplace=True)
82
83 #normaliza
84 ouvintes[0] = normaliza(ouvintes[0])
85 ouvintes[1] = normaliza(ouvintes[1])
86
87 #Separa dados de treino/teste
88 X = ouvintes[0].drop(['gostou'], axis=1)
89 y = ouvintes[0]['gostou']
```

```
90
91 #Lista modelos e m tricas
92 kernels  = ['linear', 'poly', 'rbf', 'sigmoid']
93 metricas = ['precision', 'accuracy']
94
95 #M tricas de cada modelo
96 modelos = [pd.DataFrame(columns=[metricas], index=list(range(1,20))
       ) for i in kernels]
97
98 #Roda SVC
99 for idx in range(len(kernels)):
100     for C in range(1,20):
101
102         #Cria classificador
103         clf = svm.SVC(C=C, kernel=kernels[idx], gamma='scale',
       random_state=42)
104
105         #Realiza validacao cruzada
106         scores = cross_validate(clf, X, y, cv=4, scoring=metricas,
       error_score=0)
107
108         #Armazena acur cia
109         modelos[idx].loc[C, 'accuracy'] = scores['test_accuracy'].
       mean()
110
111         #Armazena precis o
112         modelos[idx].loc[C, 'precision'] = scores['test_precision'
       ].mean()
113
114 ########
115 # POLY #
116 ########
117
118 poly_score = pd.DataFrame(columns=['C', 'n_coef', 'Acuracia', '
       Precisao'])
119
120 n_coef = range(1,10)
121 for C in range(1,20):
122     for coef in n_coef:
123         #Create a svm Classifier
124         clf = svm.SVC(C=C, kernel='poly', degree=coef, gamma='scale
       ', random_state=42)
125
126         #realiza cross-validation
127         sc = cross_validate(clf, X, y, cv=4, scoring=metricas,
       error_score=0)
```

```
128
129         result = pd.DataFrame([[C, coef, sc['test_accuracy'].mean()
    , sc['test_precision'].mean()]],
130                               columns=['C', 'n_coef', 'Acuracia', '
    Precisao'])
131
132         poly_score = poly_score.append(result, ignore_index=True)
133
134 print(poly_score[(poly_score.Acuracia > .79) & (poly_score.Precisao
     > .6)])
135
136
137 #Plot
138 plt.figure(1, figsize=(12,8))
139
140 for modelo in modelos:
141
142     plt.subplot(2,2,1)
143     modelo.plot(kind='line',y=metricas[0], use_index=True, ax=plt.
    gca())
144     plt.legend(kernels)
145     plt.xlabel('C')
146     plt.ylabel('Precis o')
147
148     plt.subplot(2,2,2)
149     modelo.plot(kind='line',y=metricas[1], use_index=True, ax=plt.
    gca())
150     plt.ylim((.575, .8))
151     plt.legend(kernels)
152     plt.xlabel('C')
153     plt.ylabel('Acur cia')
154
155
156 plt.suptitle('Scores em valida ao cruzada para diferentes Kernels
    (gamma = 1/(n * var(X))', fontsize=14)
157 plt.tight_layout(pad=2.0)
```