Relatório - Projeto
Inteligência Artificial

Report - Project
Artificial Intelligence

Matheus H. Pimenta Z.
<https://github.com/omatheuspimenta/IA_project> *

2021

Resumo

Linguagem: Python 3.8.8

Bibliotecas:

```
pandas
seaborn
sklearn
matplotlib.pyplot
numpy
xgboost
```

URL: <https://github.com/omatheuspimenta/IA_project>

Todos os códigos estão no link acima. Neste relatório são apresentados algumas partes dos códigos apenas.

1 Dataset

O dataset utilizado é composto por 50 colunas que representam medidas topológicas de grafos extraídas através de *thresholds* e mais uma coluna representando os rótulos das classes. Foram consideradas 200 observações[1] ao todo, sendo 50 observações para cada uma das 4 classes de elementos considerados.

*matheus.pimenta@outlook.com

[1] A não consideração de um maior número de amostras não afetou o objetivo de apresentar o comportamento de diversos classificadores.

```
#file: preprocess.ipynb
#Leitura do dataset
df = pd.read_csv("dataframe.csv",
                 index_col = 0)
#Dimensão do dataset
df.shape
```

A escolha do dataset em análise é justificada por trabalhos já publicados utilizando datasets similares, isto é, utilizando medidas topológicas extraídas a partir de grafos (CHILDS et al., 2009; ITO et al., 2018).

As etapas de pré-processamento realizadas neste dataset são apresentadas a seguir, salienta-se que por tratar-se de um dataset contendo apenas valores numéricos muitas etapas não foram necessárias para a preparação e tratamento dos dados, contudo em outros tipos de dados etapas como remoção de valores faltantes, verificação de variáveis qualitativas, transformação de dados e etapas de processamento de texto, por exemplo são necessárias (GARCÍA; LUENGO; HERRERA, 2015).

```
#file: preprocess.ipynb
#Verificando a classe de cada coluna do dataset
df.dtypes
```

O primeiro tratamento realizado no dataset foi a verificação e remoção de colunas nulas, após a remoção das colunas nulas do dataset o tamanho final foi de 34 colunas de características e 1 coluna dos rótulos. Nenhuma observação foi desconsiderada.

```
#file: preprocess.ipynb
#Removendo colunas nulas
df = df.loc[:, (df != 0).any(axis=0)]
#Dimensão do dataset
df.shape
```

Após a remoção das colunas com valores nulos, uma normalização dos dados foi realizada com o objetivo de que todos os valores do dataset pertençam ao intervalo $[0, 1]$. A normalização realizada foi utilizando o valor máximo e mínimo de cada uma das observações em relação a característica, como apresentado pela equação 1:

$$x_{scale} = \frac{x - \min_C(x)}{\max_C(x) - \min_C(x)}, \tag{1}$$

onde x é o valor da observação, $\min_C(x)$ e $\max_C(x)$ são o valor mínimo e máximo da característica observada, respectivamente.

A normalização é utilizada para homogeneizar os dados, diminuir viéses existentes na geração dos dados e para reduzir efeitos de "valorização" de um dado em relação a outros devido a escala de cada característica. O pacote `preprocessing` da biblioteca `sklearn` foi utilizado para a normalização MinMax do dataset.

```
#file: preprocess.ipynb
```

```
#Normalização dos dados utilizando MinMax
min_max_scaler = preprocessing.MinMaxScaler()
df_minmax = min_max_scaler.fit_transform(df)
df = pd.DataFrame(df_minmax)
```

1.1 Caracterização do Dataset

Para as representações visuais do dataset foram consideradas apenas 10 colunas, sendo a primeira coluna de cada uma das 10 medidas topológicas consideradas.

Optou-se considerar apenas 10 colunas devido a limitação computacional da execução de um número maior de colunas, além da poluição visual caso sejam consideradas todas as 50 colunas.

A representação utilizando gráficos do tipo *scatter plot* são apresentadas nas figuras 1, 3, 2 e 4. Nas diagonais estão representados os histogramas das características e a distribuição de densidade, respectivamente. A distribuição de densidade das variáveis é uma maneira de visualizar o comportamento da variável contínua nas observações, o que é recomendado já que o dataset é composto por variáveis contínuas.

A dispersão dos pontos, indicam que entre algumas características existe correlação já que a disposição assemelha-se a uma reta, como é o caso das características MT4 e MT3, o que de fato é coerente visto que MT3 está relacionado ao número de motifs com 3 vértices e MT4 relaciona-se com motifs de tamanho 4. Além da análise de correlação, a caracterização das frequências no histograma pode levantar hipóteses em relação a natureza das características, como por exemplo assumir a normalidade de algumas variáveis, como é o caso da característica ASS.

```
#file: preprocess.ipynb
#Scatter plot com histogramas
pd.plotting.scatter_matrix(df[['ASS.1', 'BET.1', 'ASPL.1', 'CC.1', 'DEG.1',
'MIN.1', 'MAX.1',  'SD.1', 'MT3.1', 'MT4.1']], figsize=(10,10))

#Scatter plot com distribuição de densidade
pd.plotting.scatter_matrix(df[['ASS.1', 'BET.1', 'ASPL.1', 'CC.1', 'DEG.1',
'MIN.1', 'MAX.1',  'SD.1', 'MT3.1', 'MT4.1']], diagonal = 'kde',
figsize=(10,10))
```

Além das representações e gráficos *scatter plot*, foram extraídas informações estatísticas do dataset em análise. Os dados são apresentados nas tabelas 1 e 2.

```
#file: preprocess.ipynb
#Medidas de posição e dispersão do dataset
pd.options.display.float_format = "{:.3f}".format
df.describe().T
```

Após a normalização, um *boxplot* e um *violin plot* foram utilizados para a visualização da distribuição dos dados dentro do intervalo $[0, 1]$, como apresentado nas figuras 5 e 6. A escolha do *violin plot* é justificada pela alta frequência de valores considerados outliers nas amostras, como apresentado pelo *boxplot*, dessa maneira é possível verificar em qual região do intervalo está a maior concentração dos dados.

```
#file: preprocess.ipynb
#Boxplot
boxplot = df.boxplot(column=['ASS.1', 'BET.1', 'ASPL.1', 'CC.1', 'DEG.1',
'MIN.1', 'MAX.1',  'SD.1', 'MT3.1', 'MT4.1'], figsize=(10,8))

#Violin plot
fig, axes = plt.subplots(figsize=(10,8))
labels = ['ASS.1', 'BET.1', 'ASPL.1', 'CC.1', 'DEG.1', 'MIN.1',
'MAX.1',  'SD.1', 'MT3.1', 'MT4.1']

axes.violinplot(dataset=[df["ASS.1"],
                         df["BET.1"],
                         df["ASPL.1"],
                         df["CC.1"],
                         df["DEG.1"],
                         df["MIN.1"],
                         df["MIN.1"],
                         df["MAX.1"],
                         df["SD.1"],
                         df["MT3.1"],
                         df["MT4.1"]])
axes.set_title('Dataset Overview')
axes.yaxis.grid(True)
axes.set_ylabel('Values')
axes.xaxis.set_tick_params(direction='out')
axes.xaxis.set_ticks_position('bottom')
axes.set_xticks(np.arange(1, len(labels) + 1))
axes.set_xticklabels(labels)
axes.set_xlim(0.25, len(labels) + 0.75)
axes.set_xlabel('Features names')
plt.show()
```

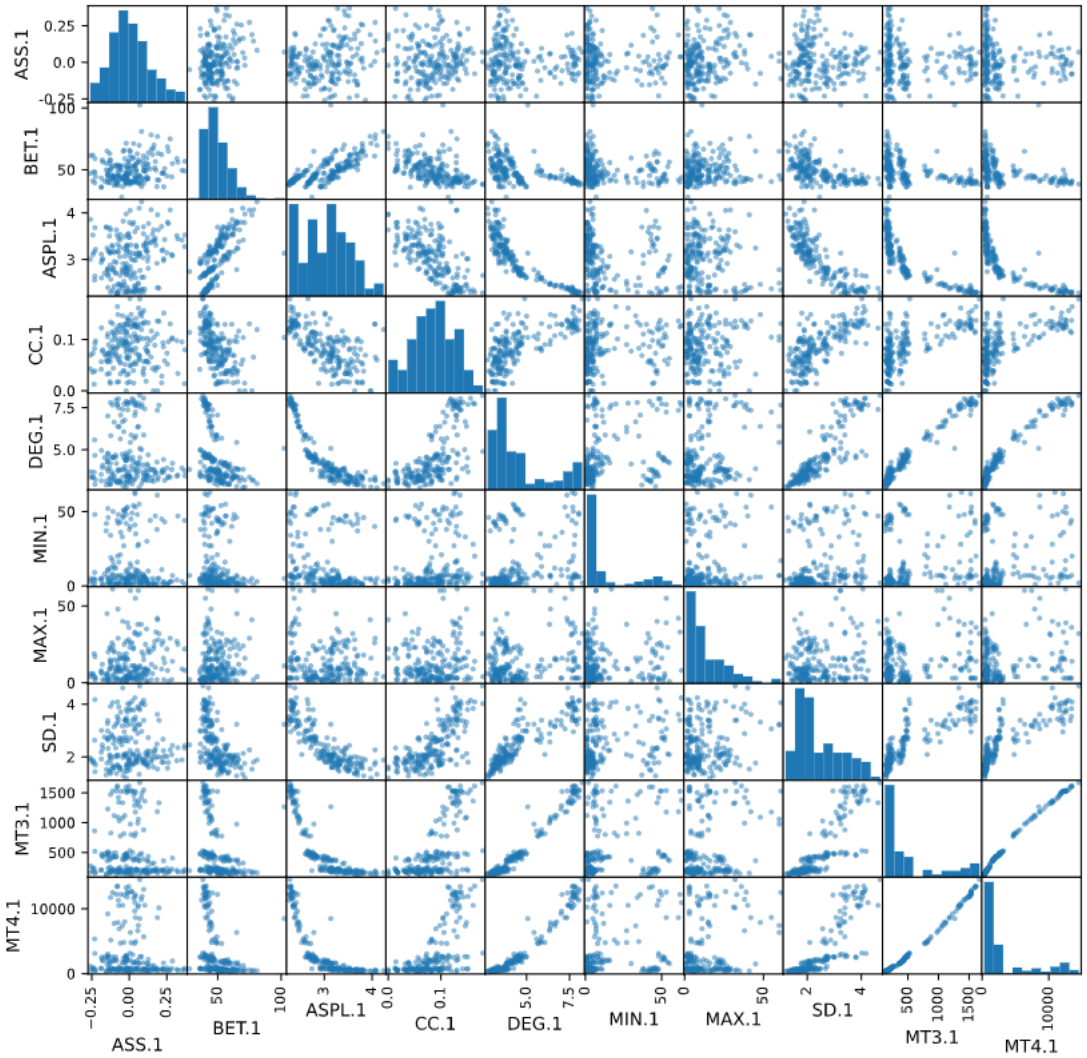


Figura 1 – Representação: *Scatter plot* com histograma - Dataset original.

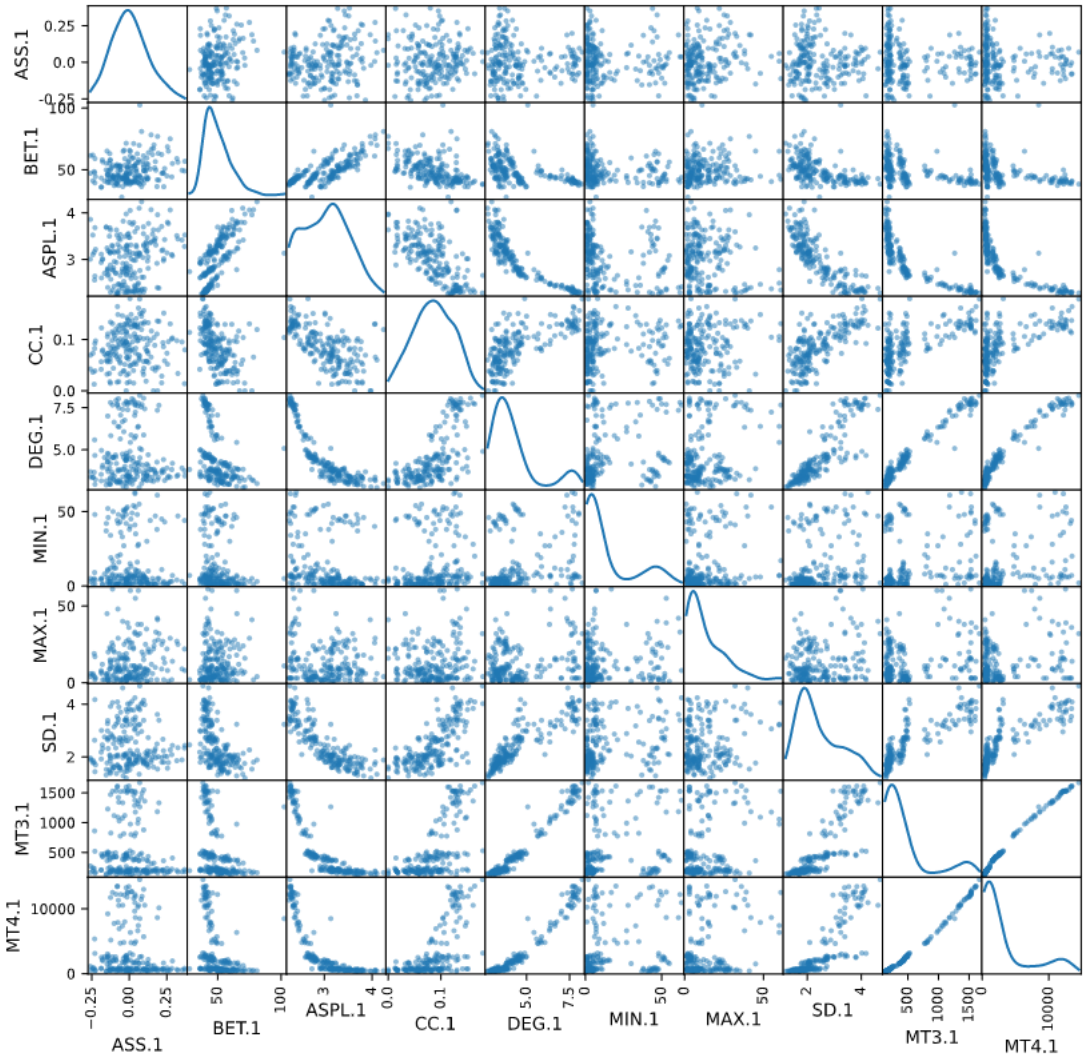


Figura 2 – Representação: *Scatter plot* com histograma - Dataset original.

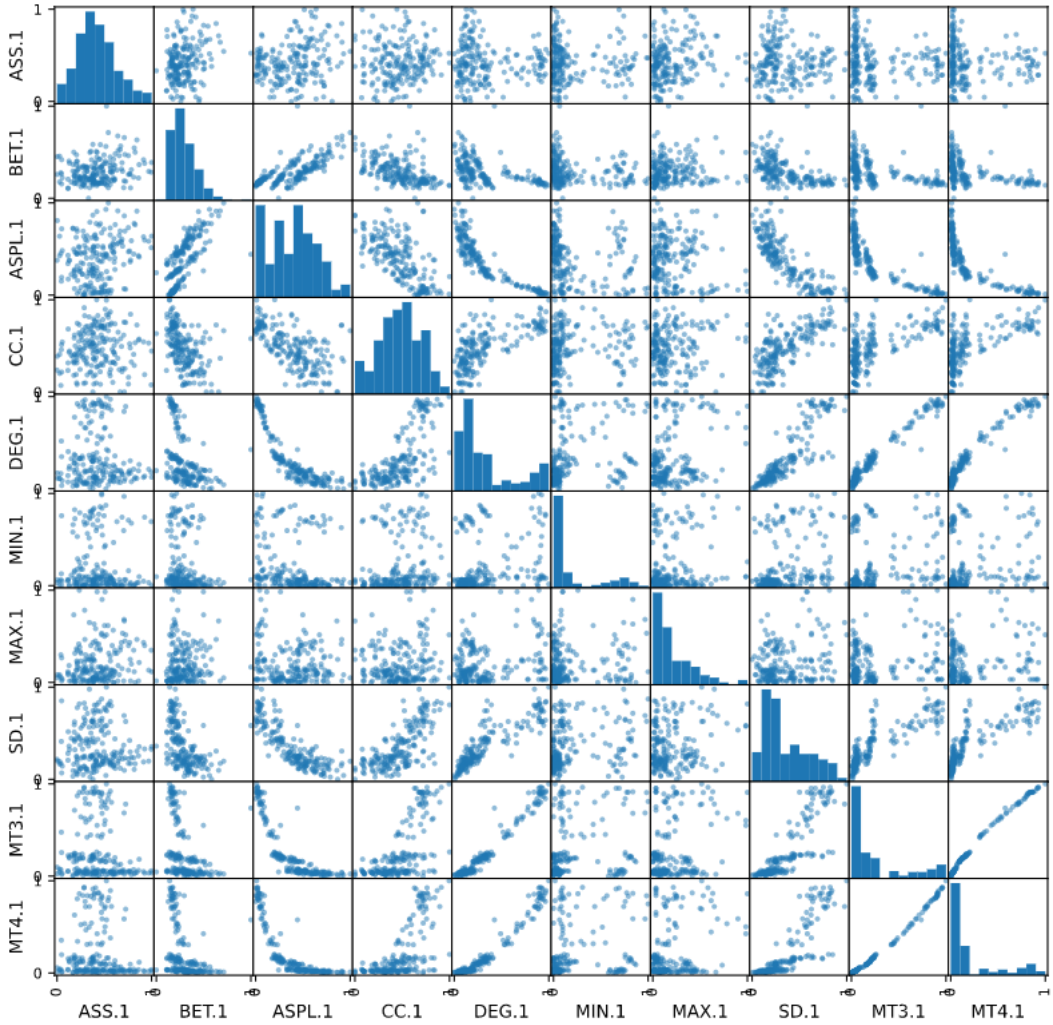


Figura 3 – Representação: *Scatter plot* com histograma - Dataset normalizado.

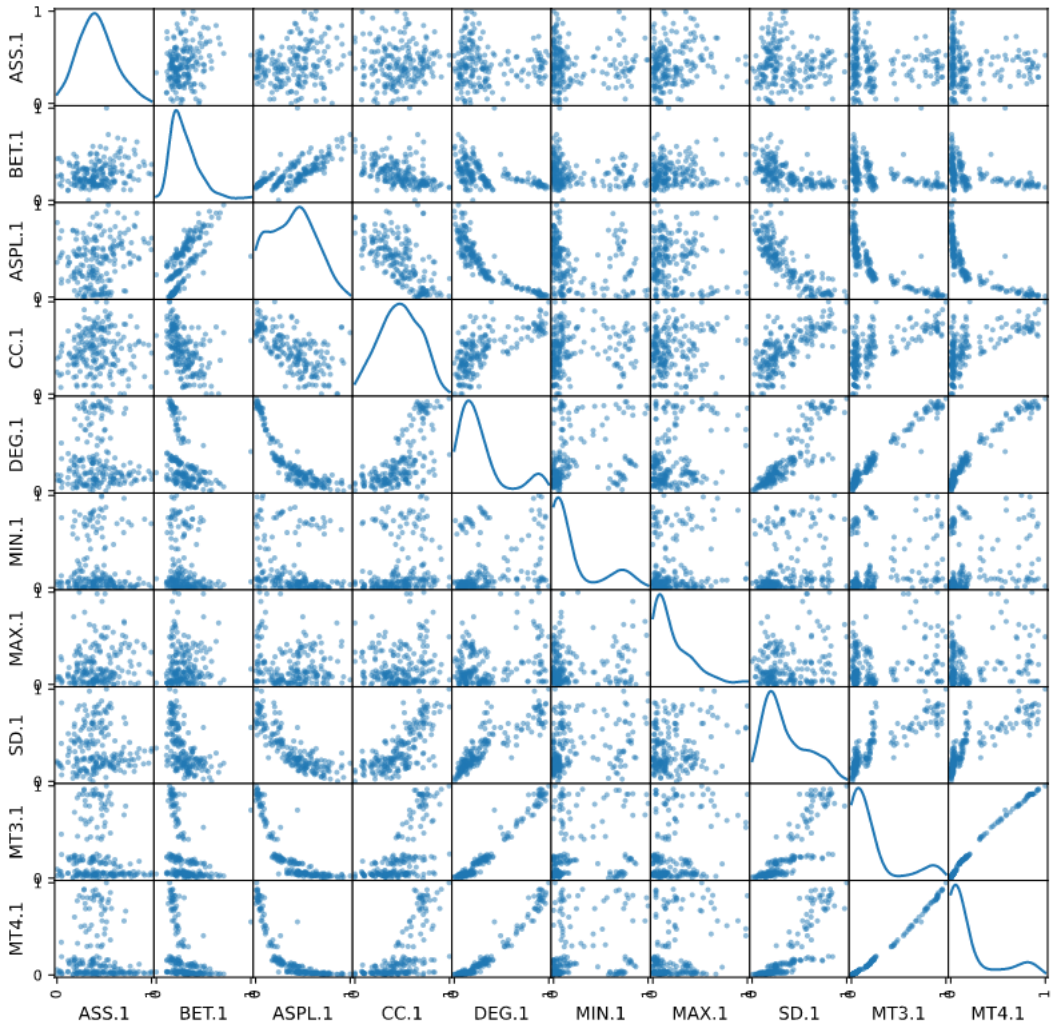


Figura 4 – Representação: *Scatter plot* com histograma - Dataset normalizado.

	count	mean	std	min	25%	50%	75%	max
ASS.1	200.000	0.008	0.131	-0.260	-0.081	-0.005	0.086	0.370
ASS.2	200.000	0.056	0.439	-1.000	-0.167	0.000	0.102	1.000
ASS.3	200.000	-0.008	0.153	-1.000	0.000	0.000	0.000	1.000
BET.1	200.000	49.619	10.103	27.818	42.495	47.379	55.014	102.443
BET.2	200.000	0.195	0.573	0.000	0.000	0.017	0.082	4.000
BET.3	200.000	0.001	0.008	0.000	0.000	0.000	0.000	0.078
ASPL.1	200.000	3.030	0.482	2.258	2.655	3.072	3.375	4.239
ASPL.2	200.000	46.516	17.194	0.000	43.448	48.695	58.519	96.856
ASPL.3	200.000	14.922	24.588	0.000	0.000	0.000	43.204	66.940
ASPL.4	200.000	1.124	7.939	0.000	0.000	0.000	0.000	62.968
CC.1	200.000	0.085	0.039	0.000	0.058	0.086	0.115	0.180
CC.2	200.000	0.002	0.016	0.000	0.000	0.000	0.000	0.176
DEG.1	200.000	4.560	1.603	2.739	3.383	3.861	5.018	8.230
DEG.2	200.000	0.486	0.373	0.000	0.218	0.400	0.693	1.651
DEG.3	200.000	0.047	0.090	0.000	0.000	0.000	0.097	0.408
DEG.4	200.000	0.004	0.026	0.000	0.000	0.000	0.000	0.258
MIN.1	200.000	14.090	17.678	1.000	3.000	5.000	16.250	63.000
MIN.2	200.000	1.370	1.261	0.000	1.000	1.000	1.000	12.000
MIN.3	200.000	0.280	0.461	0.000	0.000	0.000	1.000	2.000
MIN.4	200.000	0.020	0.140	0.000	0.000	0.000	0.000	1.000
MAX.1	200.000	13.280	12.463	1.000	4.000	9.000	20.000	61.000
MAX.2	200.000	14.210	14.217	0.000	2.000	10.500	22.000	64.000
MAX.3	200.000	4.675	9.666	0.000	0.000	0.000	5.000	61.000
MAX.4	200.000	0.315	2.685	0.000	0.000	0.000	0.000	27.000
SD.1	200.000	2.438	0.815	1.195	1.797	2.118	3.012	4.699
SD.2	200.000	0.969	0.526	0.000	0.668	0.977	1.312	2.627
SD.3	200.000	0.197	0.341	0.000	0.000	0.000	0.535	1.435
SD.4	200.000	0.016	0.116	0.000	0.000	0.000	0.000	0.991
MT3.1	200.000	523.565	472.070	125.000	192.000	291.500	590.500	1669.000
MT3.2	200.000	2.950	5.229	0.000	0.000	1.000	3.250	29.000
MT3.3	200.000	0.050	0.313	0.000	0.000	0.000	0.000	3.000
MT4.1	200.000	3318.465	4081.450	313.000	643.250	1180.500	3542.000	14500.000
MT4.2	200.000	2.490	7.193	0.000	0.000	0.000	1.000	53.000
MT4.3	200.000	0.015	0.122	0.000	0.000	0.000	0.000	1.000

Tabela 1 – Medidas de posição e dispersão - Dataset original.

	count	mean	std	min	25%	50%	75%	max
ASS.1	200.000	0.425	0.207	0.000	0.284	0.404	0.549	1.000
ASS.2	200.000	0.528	0.219	0.000	0.417	0.500	0.551	1.000
ASS.3	200.000	0.496	0.077	0.000	0.500	0.500	0.500	1.000
BET.1	200.000	0.292	0.135	0.000	0.197	0.262	0.364	1.000
BET.2	200.000	0.049	0.143	0.000	0.000	0.004	0.021	1.000
BET.3	200.000	0.016	0.105	0.000	0.000	0.000	0.000	1.000
ASPL.1	200.000	0.390	0.243	0.000	0.200	0.411	0.564	1.000
ASPL.2	200.000	0.480	0.178	0.000	0.449	0.503	0.604	1.000
ASPL.3	200.000	0.223	0.367	0.000	0.000	0.000	0.645	1.000
ASPL.4	200.000	0.018	0.126	0.000	0.000	0.000	0.000	1.000
CC.1	200.000	0.475	0.216	0.000	0.321	0.479	0.637	1.000
CC.2	200.000	0.011	0.093	0.000	0.000	0.000	0.000	1.000
DEG.1	200.000	0.332	0.292	0.000	0.117	0.204	0.415	1.000
DEG.2	200.000	0.294	0.226	0.000	0.132	0.242	0.420	1.000
DEG.3	200.000	0.116	0.221	0.000	0.000	0.000	0.238	1.000
DEG.4	200.000	0.014	0.100	0.000	0.000	0.000	0.000	1.000
MIN.1	200.000	0.211	0.285	0.000	0.032	0.065	0.246	1.000
MIN.2	200.000	0.114	0.105	0.000	0.083	0.083	0.083	1.000
MIN.3	200.000	0.140	0.231	0.000	0.000	0.000	0.500	1.000
MIN.4	200.000	0.020	0.140	0.000	0.000	0.000	0.000	1.000
MAX.1	200.000	0.205	0.208	0.000	0.050	0.133	0.317	1.000
MAX.2	200.000	0.222	0.222	0.000	0.031	0.164	0.344	1.000
MAX.3	200.000	0.077	0.158	0.000	0.000	0.000	0.082	1.000
MAX.4	200.000	0.012	0.099	0.000	0.000	0.000	0.000	1.000
SD.1	200.000	0.355	0.233	0.000	0.172	0.263	0.518	1.000
SD.2	200.000	0.369	0.200	0.000	0.254	0.372	0.499	1.000
SD.3	200.000	0.137	0.238	0.000	0.000	0.000	0.373	1.000
SD.4	200.000	0.017	0.117	0.000	0.000	0.000	0.000	1.000
MT3.1	200.000	0.258	0.306	0.000	0.043	0.108	0.301	1.000
MT3.2	200.000	0.102	0.180	0.000	0.000	0.034	0.112	1.000
MT3.3	200.000	0.017	0.104	0.000	0.000	0.000	0.000	1.000
MT4.1	200.000	0.212	0.288	0.000	0.023	0.061	0.228	1.000
MT4.2	200.000	0.047	0.136	0.000	0.000	0.000	0.019	1.000
MT4.3	200.000	0.015	0.122	0.000	0.000	0.000	0.000	1.000

Tabela 2 – Medidas de posição e dispersão - Dataset normalizado.

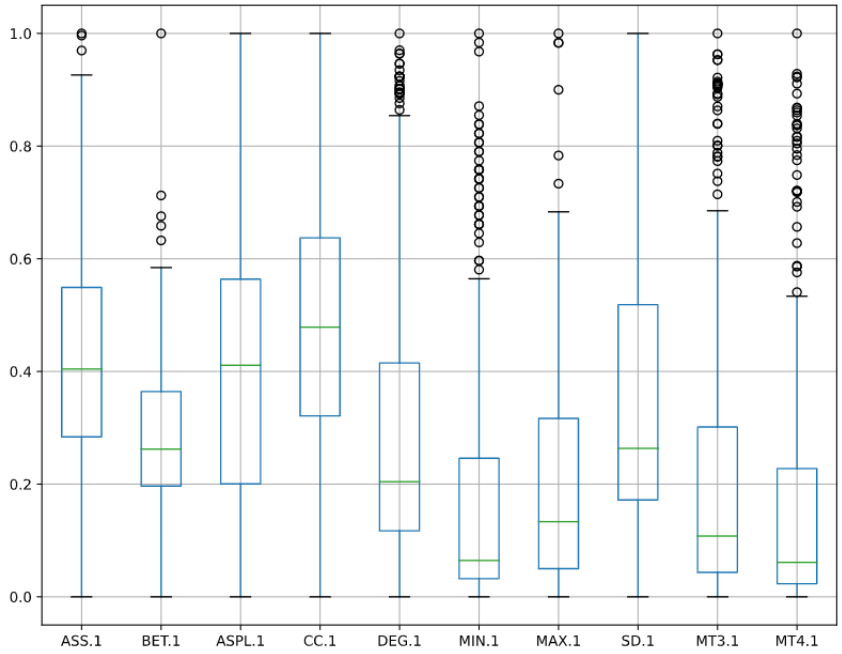


Figura 5 – *Boxplot* de 10 características - Dataset normalizado.

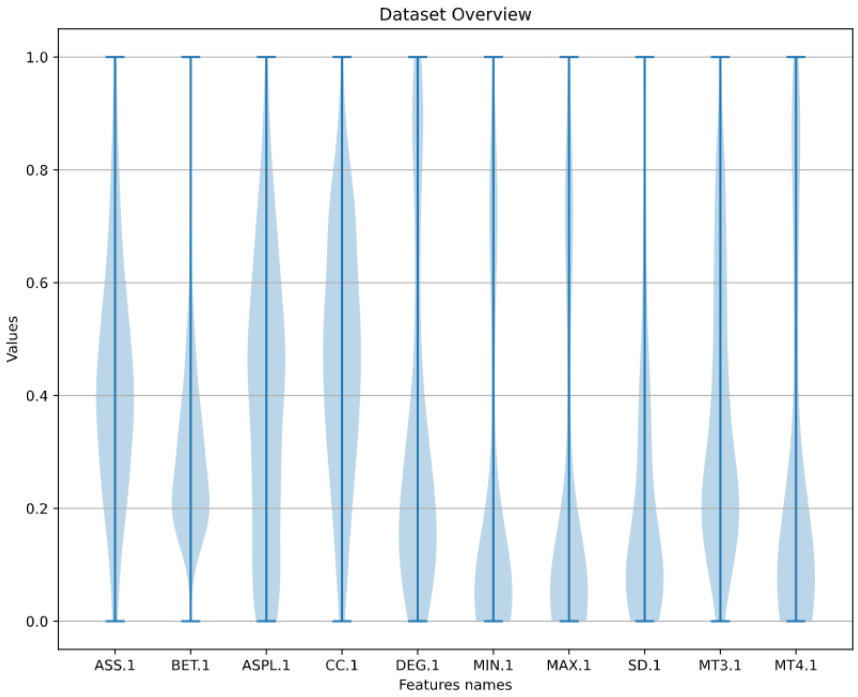


Figura 6 – *Violin plot* de 10 características - Dataset normalizado.

2 Classificadores

O objetivo deste trabalho é apresentar a execução de alguns classificadores, dessa forma optou-se por utilizar 6 classificadores e um dataset pequeno, devido as limitações de hardware para as execuções, logo não foram considerados classificadores do tipo *deep learning*. A escolha do dataset não afeta o objetivo ante exposto. Algumas observações em relação aos resultados obtidos são necessárias já que o dataset é pequeno para análises detalhadas sobre os resultados obtidos. Dessa maneira os resultados apresentados estão restritos a este dataset, para uma generalização é necessário um conjunto de dados maior.

O dataset foi binarizado e separado de maneira estratificada em conjuntos de treinamento e teste na seguinte proporção, 80% dos dados para treinamento e 20% para teste. A biblioteca `sklearn` foi utilizada nesta etapa.

```
#Binarizando os dados
Y_bin = label_binarize(df['CLASS'],
                       classes=class_label)

#Split dataset
X_train, X_test, y_train, y_test = train_test_split(X,
                                                    Y_bin,
                                                    test_size=0.2,
                                                    stratify=class_names)
y_true = np.argmax(y_test, axis = 1)
```

2.1 k-nearest neighbors $(k - NN)$

O classificador k-nearest neighbors foi proposto inicialmente na década de 60, e expandido na década de 90 (ALTMAN, 1992). A proposta do algoritmo é realizar a comparação da distância da i-ésima amostra com k vizinhos mais próximos e através de votação classificar a amostra como pertencente a classe de maior votos. Os parâmetros utilizados pelo algoritmo são os seguintes:

- A métrica para o cálculo da distância,
- O valor de k, isto é, quantos vizinhos serão considerados para a comparação.

As métricas mais utilizadas são a distância Euclidiana (eq. 2), de Minkowsky (eq. 3) e Chebyshev (eq. 4).

$$d_E(p,q) = \sqrt{\sum_{i=1}^{n}(q_i - p_i)^2} \tag{2}$$

$$D_M(p,q) = \left(\sum_{i=1}^{n}|p_i - q_i|^r\right)^{\frac{1}{r}} \tag{3}$$

$$D_C(p,q) = \max_i(|p_i, q_i|) \tag{4}$$

Já o valor de k é um valor a ser definido através de heurísticas para a seleção de melhores valores, neste projeto optou-se por realizar um refinamento do algoritmo utilizando a abordagem *grid*.

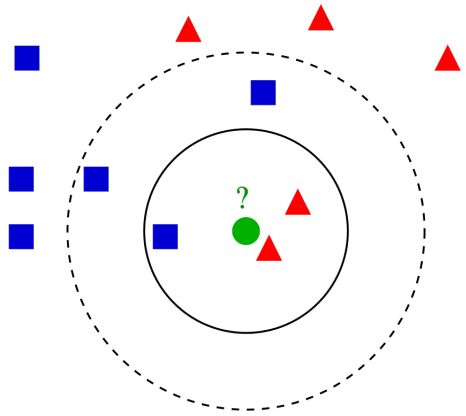


Figura 7 – Exemplo de classificação utilizando $k-NN$, o círculo verde é um novo elemento para classificação, caso seja considerado $k = 3$ (círculo preto contínuo) a classe do elemento verde é a mesma dos elementos vermelhos, por outro lado considerando $k = 5$ temos que a classe escolhido é a mesma dos elementos azuis, já que são maioria dentre este intervalo. Fonte: Antti Ajanki sob CC BY-SA 3.0

O algoritmo utilizado esta implementado no pacote `sklearn.neighbors`, com a métrica padrão Euclidiana. Inicialmente com o valor de $k = 3$ foi utilizado para a análise dos resultados.

```
#file: knn.ipynb
knn = KNeighborsClassifier(n_neighbors=3)
knn.fit(X_train, y_train)
knn3_res = knn.predict(X_test)
y_pred_knn3 = np.argmax(knn3_res, axis=1)
```

As métricas obtidas para o valor de $k = 3$ são apresentadas na tabela 3.

```
#file: knn.ipynb
#Matriz de confusão
KNN3_cm = confusion_matrix(y_true=y_true,
                           y_pred=y_pred_knn3)
#heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for
x-axis
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for
y-axis
sns.heatmap(KNN3_cm,
            vmin=0,
            vmax=10,
```

```
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for KNN Classification Model',pad=15)
```

	precision	**recall**	**f1-score**	**support**
class_1	0.71	1.00	0.83	10
class_2	1.00	1.00	1.00	10
class_3	0.75	0.30	0.43	10
class_4	0.75	0.90	0.82	10
accuracy			0.80	40
macro avg	0.80	0.80	0.77	40
weighted avg	0.80	0.80	0.77	40

Tabela 3 – Métricas - $k - NN$ - $k = 3$

E a seguinte matriz de confusão (Figura 8):

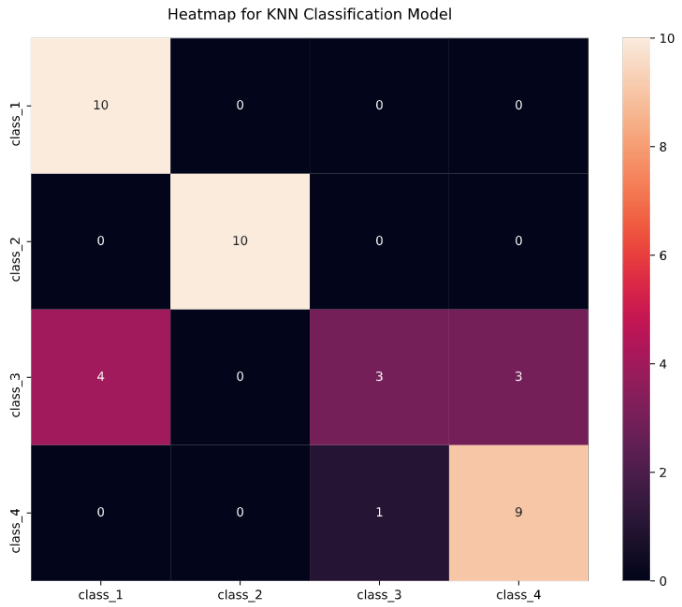


Figura 8 – Matriz de confusão e heatmap para o conjunto de teste utilizando o classificador $k - NN$ - $k = 3$.

Com o objetivo de obter um melhor valor da acurácia, foi realizada uma busca em *grid* para selecionar o melhor valor de k em um intervalo de $[1, 30]$ utilizando uma validação cruzada de $10 - fold$. Utilizou-se dessa abordagem como forma de mostrar que o classificador possui parâmetros que devem ser ajustados conforme o dataset utilizado.

```
#file: knn.ipynb
#Intervalo de busca
```

```
k_list = list(range(1,31))
knn_param = dict(n_neighbors=k_list)
#grid
grid = GridSearchCV(knn,
                    knn_param,
                    cv=10,
                    scoring='accuracy')
#Fit
grid.fit(X_train, y_train)
#Print
print("Melhores parametros {} com o valor de acurácia {}
".format(grid.best_params_,grid.best_score_))
```

Os seguintes valores de acurácias médias para cada uma das execuções das validações cruzados são exibidos na figura 9. Neste caso, em particular, a maior acurácia utilizando validação cruzada foi obtida com o valor de $k = 1$, isto significa que a classe a ser escolhida por um novo elemento será a classe do elemento mais próximo a este novo elemento.

```
#file: knn.ipynb
knn = KNeighborsClassifier(n_neighbors=1)
knn.fit(X_train, y_train)
knn1_res = knn.predict(X_test)
y_pred_knn1 = np.argmax(knn1_res, axis=1)
y_predp_knn1 = knn.predict_proba(X_test)
```

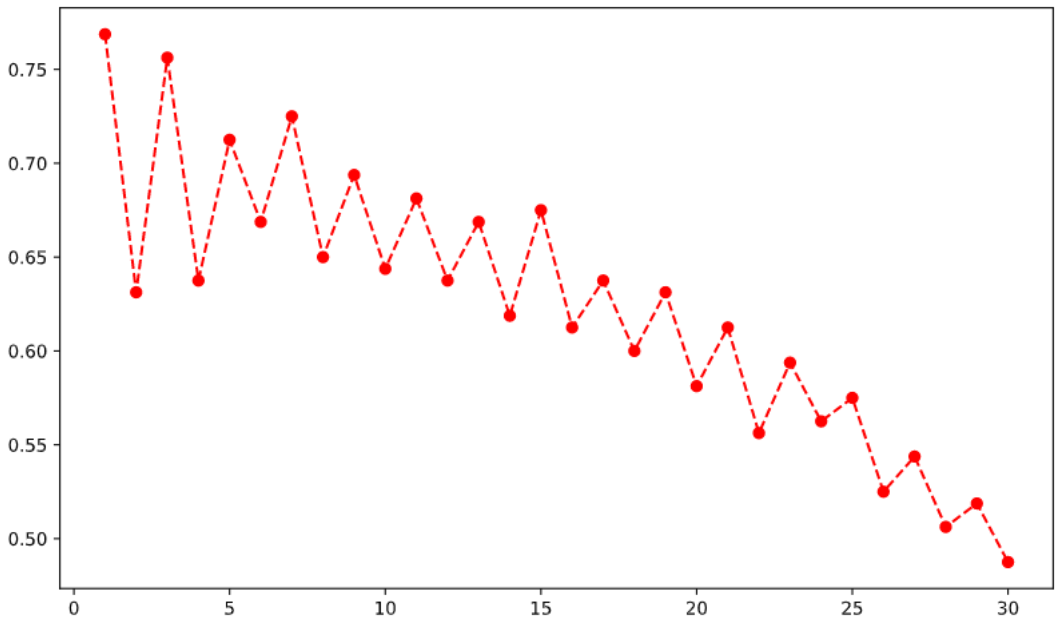


Figura 9 – Acurácias (eixo y) obtidas através da variação do valor de k (eixo x).

Utilizando o valor da maior acurácia obtido nas validações cruzadas ($k = 1$), as seguintes métricas (Tabela 4) e matriz de confusão (Figura 10) foram obtidas:

```
#file: knn.ipynb
KNN1_cm = confusion_matrix(y_true=y_true,
                           y_pred=y_pred_knn1)
print(classification_report(y_true, y_pred_knn1, target_names=class_label))
print(accuracy_score(y_true, y_pred_knn1))
```

	precision	**recall**	**f1-score**	**support**
class_1	0.75	0.90	0.82	10
class_2	0.91	1.00	0.95	10
class_3	0.80	0.40	0.53	10
class_4	0.75	0.90	0.82	10
accuracy			0.80	40
macro avg	0.80	0.80	0.78	40
weighted avg	0.80	0.80	0.78	40

Tabela 4 – Métricas - $k - NN$ - $k = 1$

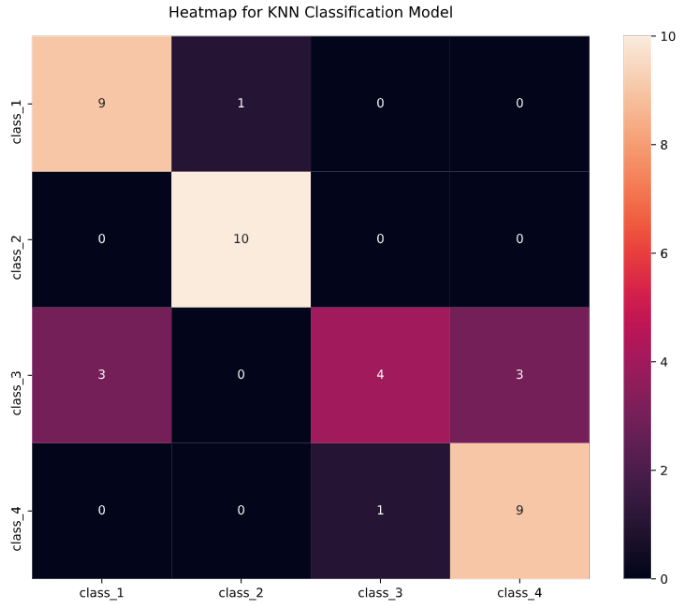
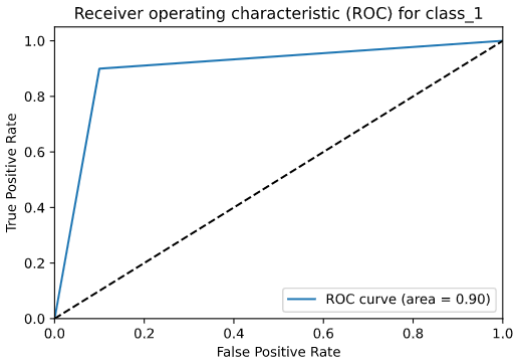


Figura 10 – Matriz de confusão e heatmap para o conjunto de teste - $k - NN$ - $k = 1$.

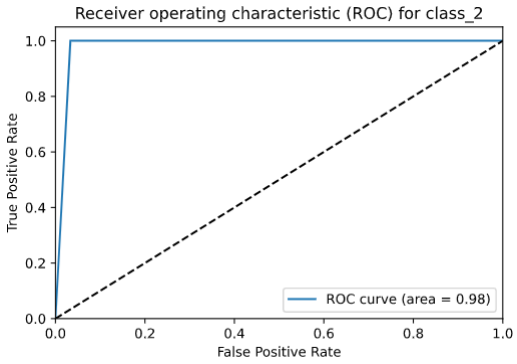
A análise da performance de classificadores pode ser feita analisando a área abaixo da curva ROC (Receiver Operating Characteristic). Quanto maior for a área, isto é, mais próxima a 1, melhor é a performance do classificador. Como o dataset é multi classe foi utilizada a abordagem "um contra todos" para a obtenção das curvas ROC para cada uma das classes, como mostra a figura 11.

```
#file: knn.ipynb
#OneVsRestClassifier
clf = OneVsRestClassifier(KNeighborsClassifier(n_neighbors=1))
y_pred_ovr_knn1 = clf.fit(X_train, y_train).predict_proba(X_test)
```

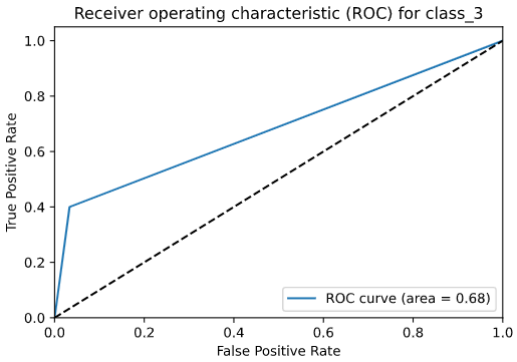
```
#Curva ROC para cada classe
fpr_KNN1 = dict()
tpr_KNN1 = dict()
roc_auc_KNN1 = dict()
for i in range(class_label.shape[0]):
   fpr_KNN1[i], tpr_KNN1[i], _ = roc_curve(y_test[:, i], y_pred_ovr_knn1[:,
   i])
   roc_auc_KNN1[i] = auc(fpr_KNN1[i], tpr_KNN1[i])
#Plot
for i in range(class_label.shape[0]):
   plt.figure()
   plt.plot(fpr_KNN1[i], tpr_KNN1[i], label='ROC curve (area = %0.2f)' %
   roc_auc_KNN1[i])
   plt.plot([0, 1], [0, 1], 'k--')
   plt.xlim([0.0, 1.0])
   plt.ylim([0.0, 1.05])
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.title('Receiver operating characteristic (ROC) for %s' %
   class_label[i])
   plt.legend(loc="lower right")
   plt.show()
```



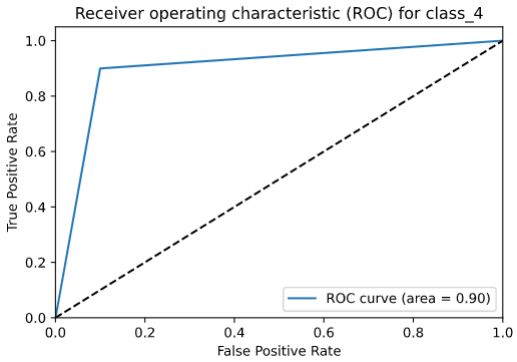
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 11 – Receiver operating characteristic (ROC) para o classificador $k - NN$ com $k = 1$.

O valor do coeficiente Kappa (COHEN, 1960) é uma métrica que busca avaliar concordância entre conjuntos, no caso de classificadores são analisados os conjuntos de teste e preditos, quanto mais próximo de 1, maior é o nível de concordância entre os conjuntos. Para $k - NN$ com $k = 1$ o valor do coeficiente de Kappa é 0.73 com as seguintes métricas (Tabela 5).

```
#file: knn.ipynb
#Kappa
knn_kappa = cohen_kappa_score(y_true, y_pred_knn1)
```

Métrica	Class1	Class2	Class3	Class4
Sensibilidade:	0.9	1	0.4	0.9
True Negative:	0.9	0.96	0.96	0.9
Precisão:	0.75	0.9	0.8	0.75
Pred. Negativa:	0.96	1	0.82	0.96
False Positive:	0.1	0.03	]0.03	0.1
False Negative:	0.1	0	0.6	0.1
False Discovery:	0.25	0.09	0.2	0.25
Acurácia:	0.9	0.97	0.82	0.9

Tabela 5 – Métricas - $k - NN$ - $k = 1$

2.2 Árvore de Decisão

O classificador árvore de decisão busca relacionar os atributos (características) dos elementos (observações) com seus respectivos rótulos (classes) utilizando uma estrutura de grafos em topologia de árvore (SAFAVIAN; LANDGREBE, 1991). O objetivo é identificar quais as melhores características para realizar a "divisão" dos dados nas classes corretas.

Para isso são considerados algumas métricas que buscam mensurar quão nítida pode ser essa divisão, como por exemplo a métrica de Gini ou ainda o quanto de informação é recebido com as divisões, através do cálculo de entropia.

É um método "simples", não exigindo conhecimentos prévios a respeito dos dados, contudo altamente propenso a *overfitting* dos dados de treinamento. Alternativas a este comportamento são propostas nos classificadores seguintes.

O algoritmo utilizado está implementado no pacote `tree` da biblioteca `sklearn` com o seguinte método `DecisionTreeClassifier`. Os parâmetros utilizados foram configurados com os valores *default* do pacote.

```
#file: dt.ipynb
DT = tree.DecisionTreeClassifier()
DT.fit(X_train,y_train)
DT_res = DT.predict(X_test)
y_pred_DT = np.argmax(DT_res, axis=1)
y_predp_DT = DT.predict_proba(X_test)
```

As métricas (Tabela 6), matriz de confusão (Figura 13) e curvas ROC (Figura 14) para cada uma das classes são apresentadas a seguir.

```
#file: dt.ipynb
#Matriz de confusão
DT_cm = confusion_matrix(y_true=y_true,
                         y_pred=y_pred_DT)
#Heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for
x-axis
```

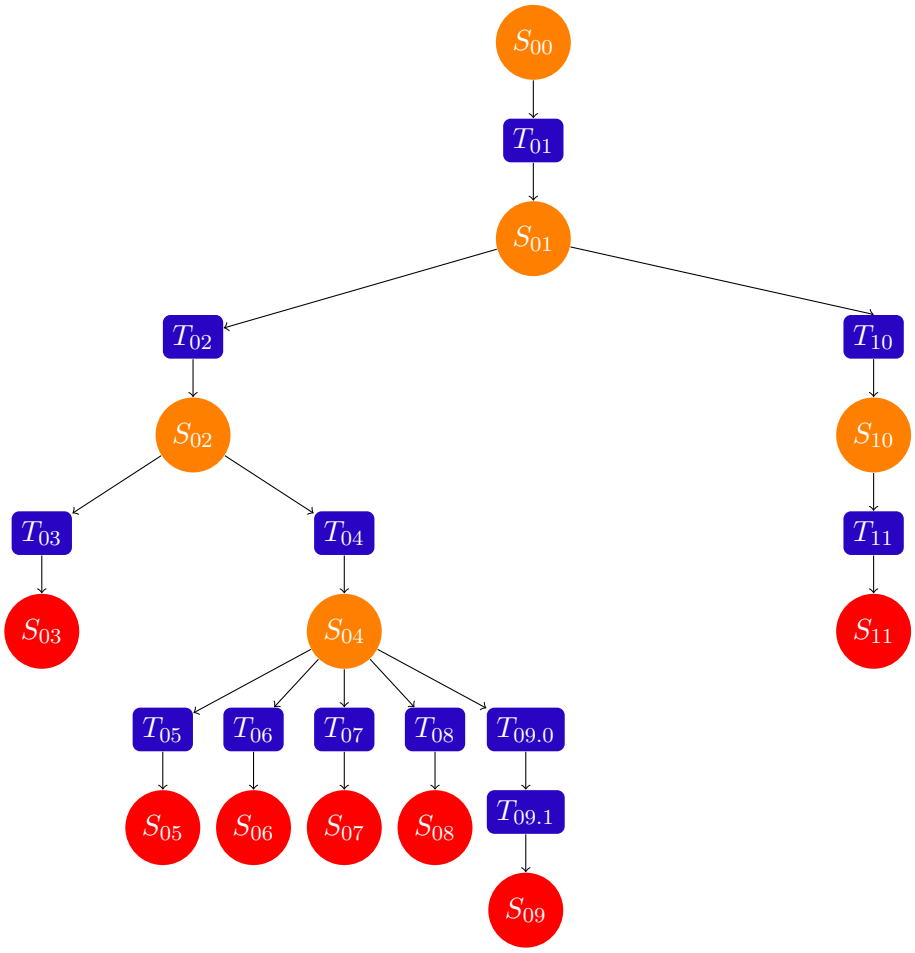


Figura 12 – Exemplo de árvore de decisão, com o nó raiz acima (S_{00}) e a partir dele os ramos (características) e folhas em vermelho (rótulos) das classificações.

```
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for
y-axis
sns.heatmap(DT_cm,
            vmin=0,
            vmax=10,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for Decision Tree Classification Model',pad=15)
#Métricas
print(classification_report(y_true, y_pred_DT, target_names=class_label))
print("Accuracy:",metrics.accuracy_score(y_true, y_pred_DT))
#Curva ROC
fpr_DT = dict()
tpr_DT = dict()
roc_auc_DT = dict()
```

```
for i in range(class_label.shape[0]):
    fpr_DT[i], tpr_DT[i], _ = roc_curve(y_test[:, i], y_pred_ovr_DT[:, i])
    roc_auc_DT[i] = auc(fpr_DT[i], tpr_DT[i])
#Plot
for i in range(class_label.shape[0]):
   plt.figure()
   plt.plot(fpr_DT[i], tpr_DT[i], label='ROC curve (area = %0.2f)' % roc_auc_DT[i])
   plt.plot([0, 1], [0, 1], 'k--')
   plt.xlim([0.0, 1.0])
   plt.ylim([0.0, 1.05])
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.title('Receiver operating characteristic (ROC) for %s' % class_label[i])
   plt.legend(loc="lower right")
   plt.show()
```

	precision	recall	f1-score	support
class_1	1.00	1.00	1.00	10
class_2	1.00	1.00	1.00	10
class_3	0.73	0.80	0.76	10
class_4	0.78	0.70	0.74	10
accuracy			0.88	40
macro avg	0.88	0.88	0.87	40
weighted avg	0.88	0.88	0.87	40

Tabela 6 – Métricas - Árvore de Decisão

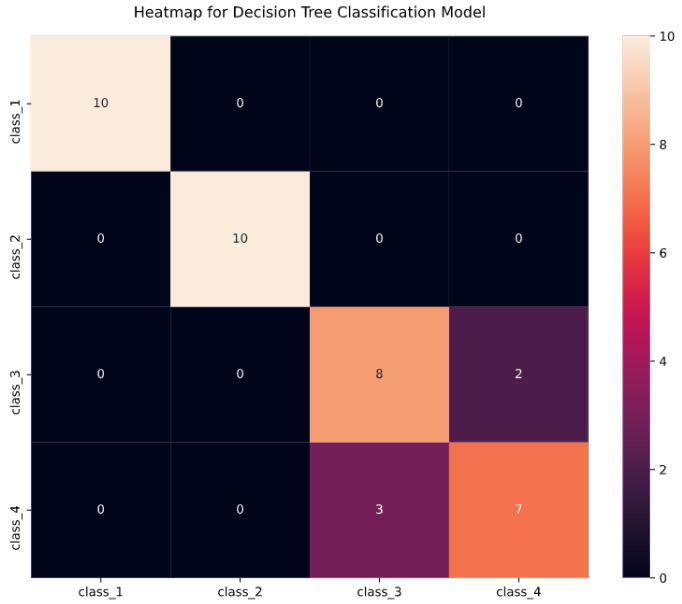
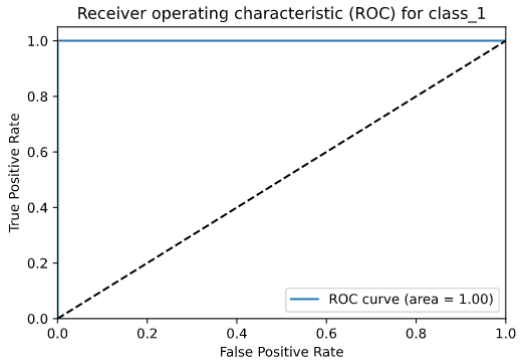


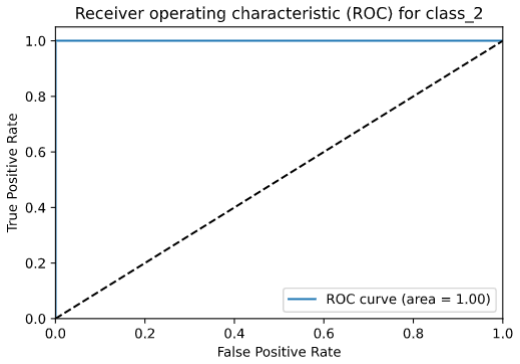
Figura 13 – Matriz de confusão e heatmap para o conjunto de teste - Árvore de Decisão.

O valor do coeficiente Kappa para a Árvore de Decisão foi de 0.83 com as seguintes métricas (Tabela 7).

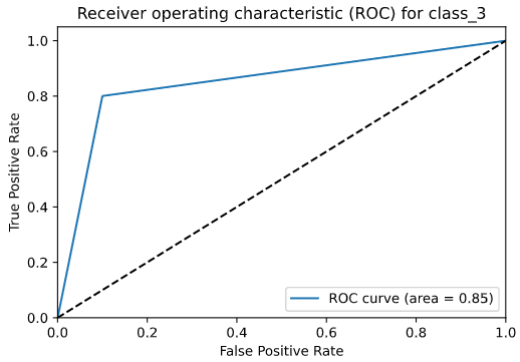
```
#file: dt.ipynb
#Kappa
DT_kappa = cohen_kappa_score(y_true, y_pred_DT)
```



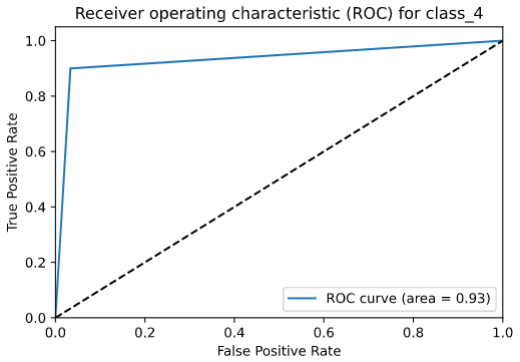
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 14 – Receiver operating characteristic (ROC) para o classificador Árvore de Decisão.

Métrica	**Class1**	**Class2**	**Class3**	**Class4**
Sensibilidade:	1	1	0.8	0.7
True Negative:	1	1	0.9	0.93
Precisão:	1	1	0.72	0.77
Pred. Negativa:	1	1	0.93	0.9
False Positive:	0	0	0.1	0.06
False Negative:	0	0.	0.2	0.3
False Discovery:	0	0	0.27]	0.22
Acurácia:	1	1	0.87	0.87

Tabela 7 – Métricas - Árvore de Decisão

2.3 Random Forest

O algoritmo Random Forest é um método *ensemble* que combina a execução de diversas árvores de decisão de maneira que relacione os atributos (características) dos elementos (observações) com seus respectivos rótulos (classes) com uma maior assertividade, diminuindo o efeito de *overfitting* do método árvore de decisão (IGUAL; SEGUÍ, 2017).

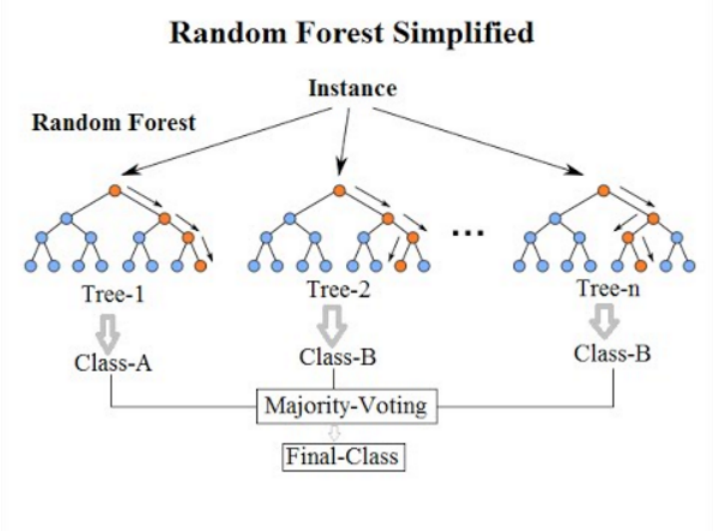


Figura 15 – Exemplo de classificação utilizando Random Forest. Diversas árvores de decisão são executadas simultaneamente, após a execução de todas as n árvores de decisão, através de votação é definida a classe do elemento em análise. Fonte: Venkata Jagannath sob CC BY-SA 3.0

O algoritmo utilizado está implementado no pacote `sklearn.ensemble` da biblioteca `sklearn` com o seguinte método `RandomForestClassifier`. Os parâmetros utilizados foram os valores default do método, isto é 100 árvores de decisão por execução e $max\_features = \sqrt{n\_features}$ como o número máximo de características selecionadas para cada árvore de decisão. Além da reamostragem do conjunto de treinamento ser realizada via *bootstrap* com o número máximo de amostras sendo o número de observações do conjunto de treinamento.

```
#file: rf.ipynb
#Random Forest
RF = RandomForestClassifier(n_estimators = 100)
RF.fit(X_train,y_train)
RF100_res = RF.predict(X_test)
y_pred_RF100 = np.argmax(RF100_res, axis=1)
y_predp_RF100 = RF.predict_proba(X_test)
```

As métricas (Tabela 8), matriz de confusão (Figura 16) e curvas ROC (Figura 17) para cada uma das classes são apresentadas abaixo.

```
#file: rf.ipynb
#Matriz de confusão
```

```
RF100_cm = confusion_matrix(y_true=y_true,
                            y_pred=y_pred_RF100)
#Heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for x-axis
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for y-axis
sns.heatmap(RF100_cm,
            vmin=0,
            vmax=10,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for RF100 Classification Model',pad=15)
#Métricas
print(classification_report(y_true, y_pred_RF100,
target_names=class_label))
print("Accuracy:",metrics.accuracy_score(y_true, y_pred_RF100))
```

	precision	**recall**	**f1-score**	**support**
class_1	0.91	1.00	0.95	10
class_2	1.00	0.90	0.95	10
class_3	0.77	1.00	0.87	10
class_4	1.00	0.70	0.82	10
accuracy			0.90	40
macro avg	0.92	0.90	0.90	40
weighted avg	0.92	0.90	0.90	40

Tabela 8 – Métricas - Random Forest

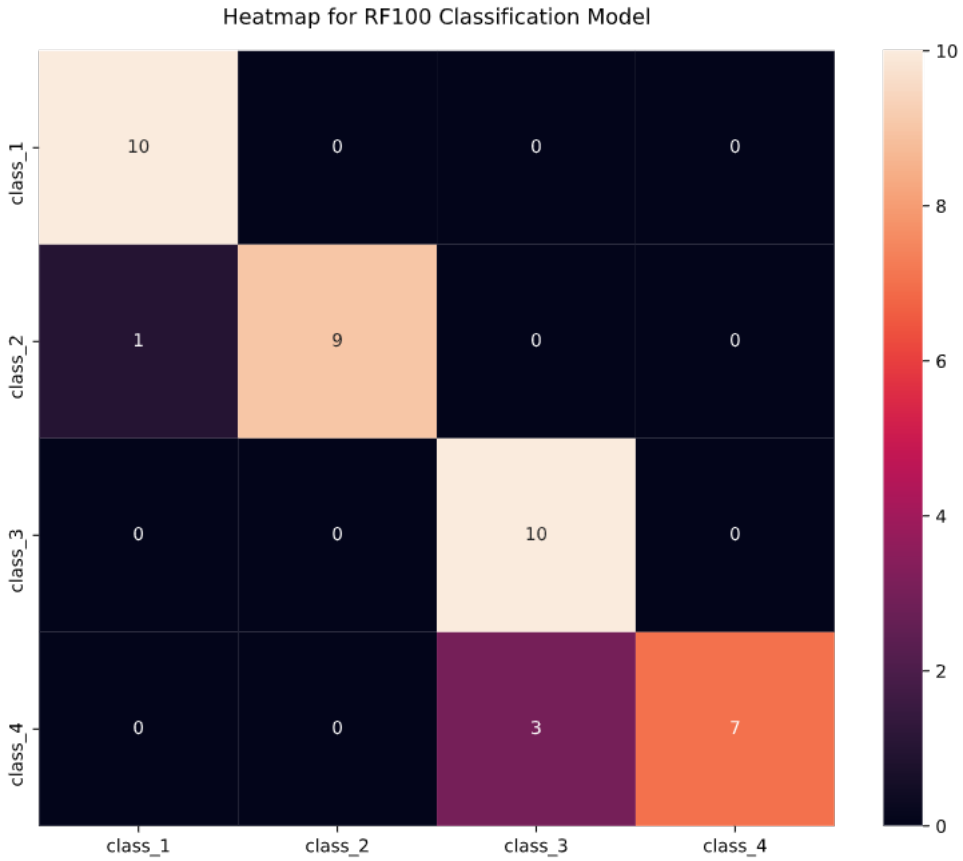
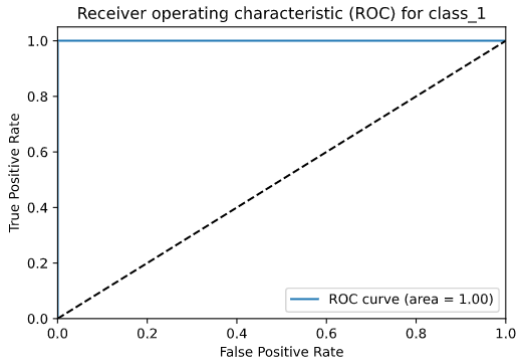
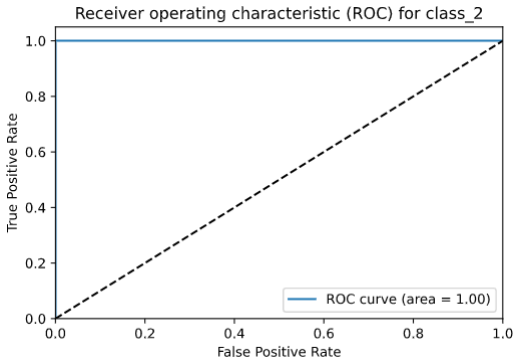


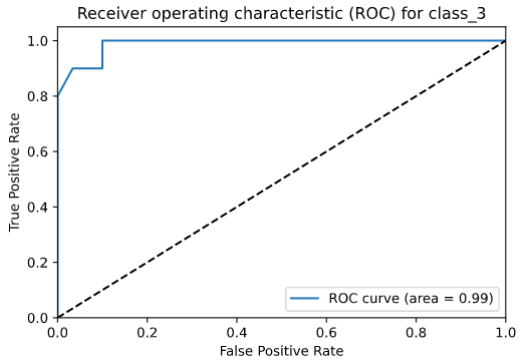
Figura 16 – Matriz de confusão e heatmap para o conjunto de teste - Random Forest.



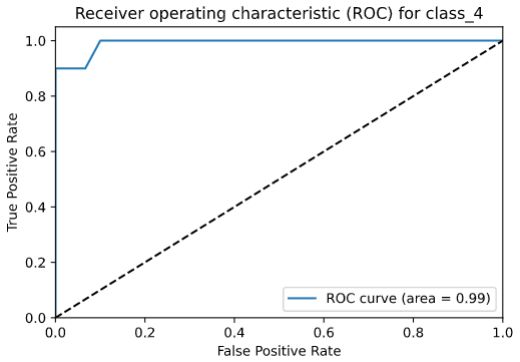
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 17 – Receiver operating characteristic (ROC) para o classificador Random Forest.

Além das análises apresentadas, foram analisados as características mais relevantes para a classificação, como apresentada na figura 18. Essa análise de quais características possuem maior relevância no dataset contribui fornecendo maiores informações a respeito do dataset utilizado.

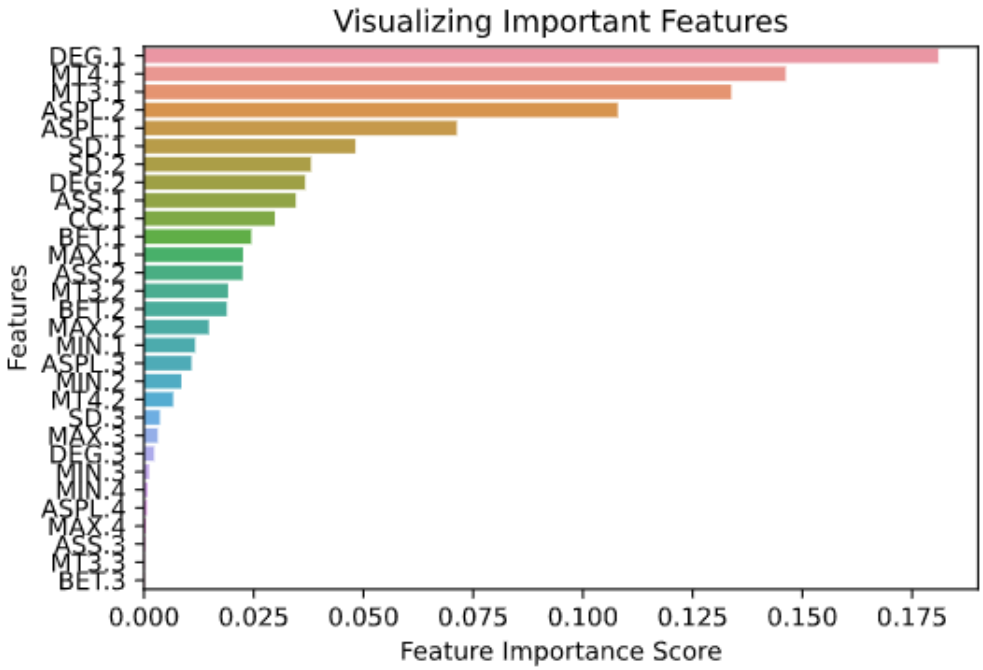


Figura 18 – Características de maior relevância para o classificador Random Forest. A análise pode ser utilizada para unir características semelhantes, selecionando subconjuntos apenas com as características de maior relevância e diminuindo o custo computacional do método.

O valor do coeficiente Kappa para o classificador Random Forest foi de 0.86 com as seguintes métricas (9):

Métrica	Class1	Class2	Class3	Class4
Sensibilidade:	1	0.9	1	0.7
True Negative:	0.96	1	0.9	1
Precisão:	0.90	1	0.76	1
Pred. Negativa:	1	0.96	1	0.9
False Positive:	0.03	0	0.1	0
False Negative:	0	0.1	0	0.3
F Discovery:	0.09	0	0.23	0
Acurácia:	0.97	0.97	0.92	0.92

Tabela 9 – Métricas - Random Forest

2.4 Gradient Boosting

Um outro método do tipo *ensemble* proposto é o Gradient Boosting inicialmente apresentaado por (FRIEDMAN; HASTIE; TIBSHIRANI, 2000). Utilizando a teoria *boosting* com ferramental matemático (otimização) e estatístico o método Gradient Boosting realiza uma busca do ponto mínimo da função *loss* associada ao modelo.

Em outras palavras, busca-se associar um funcional desconhecido utilizando um conjunto $(x, y)_{i=1}^{N}$, onde x são as características das observações e y os rótulos de cada observação (NATEKIN; KNOLL, 2013). Dessa forma busca-se definir um funcional $\hat{f}$ de

tal forma que $x \xrightarrow{f} y$ seja aproximado por $\hat{f}$ minimizando uma função $\Psi(y, f)$:

$$\hat{f}(x) = y \tag{5}$$

$$\hat{f}(x) = \underset{f(x)}{\operatorname{argmin}} \Psi(y, f(x)) \tag{6}$$

A equação 6 é reescrita de maneira a obter a função objetiva que sera diferenciada e utilizando a técnica de otimização do gradiente descendente busca-se definir o seu ponto de mínimo. O método pode ser detalhado em (NATEKIN; KNOLL, 2013).

O algoritmo utilizado está implementado no pacote `sklearn.ensemble` da biblioteca `sklearn` com o seguinte método `GradientBoostingClassifier`.

O método possuí diversos parâmetros e como o objetivo deste relatório não é refinar métodos de machine learning optou-se por analisar somente um parâmetro do método Gradient Boosting, contudo é válido reforçar que pode-se utilizar de diversas outras abordagens para obter melhores valores nas métricas de avaliação, como por exemplo uma busca em *grid* de diversos parâmetros e valores.

```
#file:grad_boost.ipynb
#Taxas de aprendizagem
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in lr_list:
   gb_clf = OneVsRestClassifier(GradientBoostingClassifier(n_estimators=100,
   gb_fit = gb_clf.fit(X_train, y_train)
   print("Learning rate: ", learning_rate)
   print("Accuracy score (training): {0:.3f}".format(gb_fit.score(X_train,
   y_train)))
   print("Accuracy score (teste): {0:.3f}".format(gb_fit.score(X_test,
   y_test)))
```

O parâmetro analisado foi a taxa de aprendizagem do algoritmo, com os seguintes valores: $0.05, 0.075, 0.1, 0.25, 0.5, 0.75$ e 1. Os demais parâmetros foram considerados os valores *default* do método. A execução das taxas de aprendizagem resultou em valores de acurácia diversos [2], a maior taxa de acurácia foi obtida com a taxa de 0.5, a qual foi utilizada ao decorrer das análises.

```
#file: grad_boost.ipynb
#L.R. 0.5
gb_clf = OneVsRestClassifier(GradientBoostingClassifier(n_estimators=100,
y_predp_ovr_gb = gb_clf.fit(X_train, y_train).predict_proba(X_test)
y_pred_ovr_gb = gb_clf.fit(X_train, y_train).predict(X_test)
y_pred_gb = np.argmax(y_pred_ovr_gb, axis=1)
```

As métricas (Tabela 10), matriz de confusão (Figura 19) e curvas ROC (Figura 20) para cada uma das classes foram as seguintes:

[2] Os demais valores estão no arquivo grad_boost.ipynb.

```
#file: grad_boost.ipynb
#Matriz de confusão
gb_cm = confusion_matrix(y_true=y_true,
                         y_pred=y_pred_gb)
#Heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for x-axis
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for y-axis
sns.heatmap(gb_cm,
            vmin=0,
            vmax=10,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for Gradient Boosting Classification Model',pad=15)
print(classification_report(y_true, y_pred_gb, target_names=class_label))
print("Accuracy:",metrics.accuracy_score(y_true, y_pred_gb))
```

	precision	recall	f1-score	support
class_1	1.00	0.80	0.89	10
class_2	1.00	1.00	1.00	10
class_3	0.77	1.00	0.87	10
class_4	1.00	0.90	0.95	10
accuracy			0.93	40
macro avg	0.94	0.92	0.93	40
weighted avg	0.94	0.93	0.93	40

Tabela 10 – Métricas - Gradient Boosting - L.R: 0.5

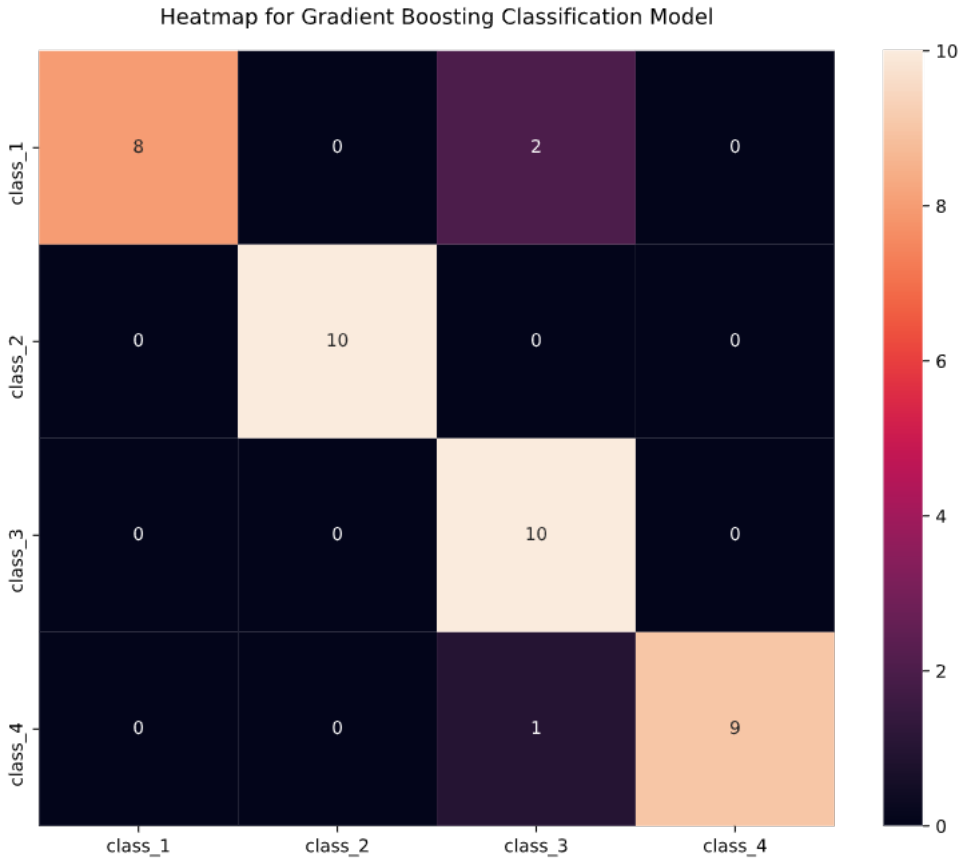
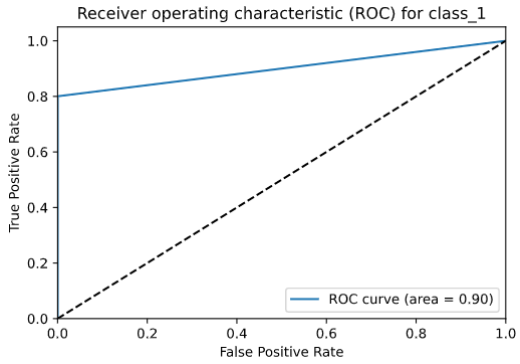
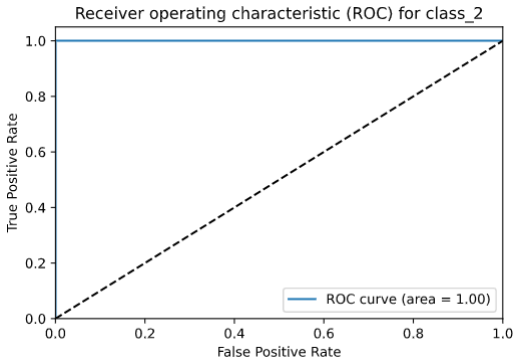


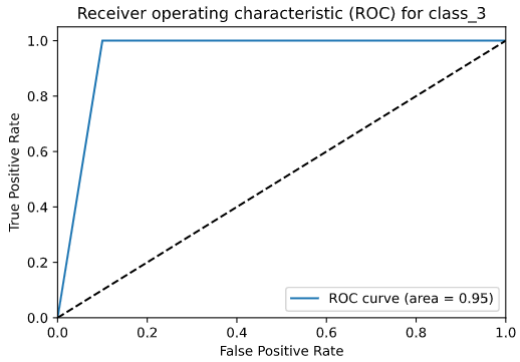
Figura 19 – Matriz de confusão e heatmap para o conjunto de teste - Gradient Boosting L.R. :0.5.



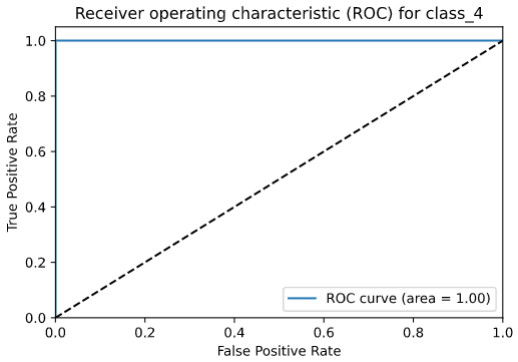
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 20 – Receiver operating characteristic (ROC) para o classificador Grandient Boosting.

O valor do coeficiente Kappa para o classificador Gradient Boosting foi de 0.9 com as seguintes métricas (11):

Métricas	**Class1**	**Class2**	**Class3**	**Class4**
Sensibilidade:	0.8	1	1	0.9
True Negative:	1	1	0.9	1
Precisão:	1	1	0.76	1
Pred. Negativa:	0.93	1	1	0.96
False Positive:	0	0	0.1	0
False Negative:	0.2	0	0	0.1
F Discovery:	0	0	0.23	0
Acurácia:	0.95	1	0.92	0.97

Tabela 11 – Métricas - Gradiente Boosting - L.R.:0.5

2.5 XGBoost

Proposto inicialmente por (CHEN; GUESTRIN, 2016) o método XGBoost também utiliza metodologia análoga aos métodos anteriores, isto é, é um método do tipo *boosting* que busca minimizar uma função *loss*. Quando comparado com outros métodos XGBoost possui recursos que aumentam sua acurácia dependendo do dataset utilizado. Maiores informações sobre o método e a abordagem matemática por ser acessada em (CHEN; GUESTRIN, 2016).

O algoritmo utilizado está implementado na biblioteca `xgboost` com o seguinte método `XGBClassifier`.

O método possuí diversos parâmetros, como o objetivo deste relatório não é refinar métodos de machine learning optou-se por analisar somente um parâmetro do método XGBoost, contudo é válido reforçar que pode-se utilizar de diversas outras abordagens para obter melhores valores nas métricas de avaliação, como por exemplo uma busca em *grid* de diversos parâmetros e valores.

O parâmetro analisado foi a taxa de aprendizagem do algoritmo, analisando os seguintes valores: $0.05, 0.075, 0.1, 0.25, 0.5, 0.75$ e 1. Os demais parâmetros foram considerados os valores default. A execução das taxas de aprendizagem resultou em valores de acurácia diversos [3], a maior taxa de acurácia foi obtida com a taxa de 0.075, a qual foi utilizada ao decorrer das análises.

```
#file:XGBoost.ipynb
#Analisando LR
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]
for learning_rate in lr_list:
   xgb_clf = OneVsRestClassifier(XGBClassifier(eval_metric='mlogloss',
                                               use_label_encoder=False,
                                               learning_rate=learning_rate)
                                               )

   xgb_fit = xgb_clf.fit(X_train, y_train)
   print("Learning rate: ", learning_rate)
   print("Accuracy score (training): {0:.3f}".format(xgb_fit.score(X_train,
   y_train)))
   print("Accuracy score (teste): {0:.3f}".format(xgb_fit.score(X_test,
   y_test)))
#LR
xgb_clf = OneVsRestClassifier(XGBClassifier(eval_metric='mlogloss',
                                            use_label_encoder=False,
                                            learning_rate=0.075))
y_predp_ovr_xgb = xgb_clf.fit(X_train, y_train).predict_proba(X_test)
y_pred_ovr_xgb = xgb_clf.fit(X_train, y_train).predict(X_test)
y_pred_xgb = np.argmax(y_pred_ovr_xgb, axis=1)
```

As métricas (Tabela 12), matriz de confusão (Figura 21) e curvas ROC (Figura 22) para cada uma das classes foram as seguintes:

[3] Os demais valores estão no arquivo XGBoost.ipynb.

```
#file: XGBoost.ipynb
#Matriz de confusão
xgb_cm = confusion_matrix(y_true=y_true,
                          y_pred=y_pred_xgb)
#Heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for x-axis
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4'] # labels for y-axis
sns.heatmap(xgb_cm,
            vmin=0,
            vmax=10,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for X Gradient Boosting Classification Model',pad=15)
#Métricas
print(classification_report(y_true, y_pred_xgb, target_names=class_label))
print("Accuracy:",metrics.accuracy_score(y_true, y_pred_xgb))
```

	precision	recall	f1-score	support
class_1	0.83	1.00	0.91	10
class_2	1.00	1.00	1.00	10
class_3	0.88	0.70	0.78	10
class_4	0.90	0.90	0.90	10
accuracy			0.90	40
macro avg	0.90	0.90	0.90	40
weighted avg	0.90	0.90	0.90	40

Tabela 12 – Métricas - XGBoost - L.R: 0.075

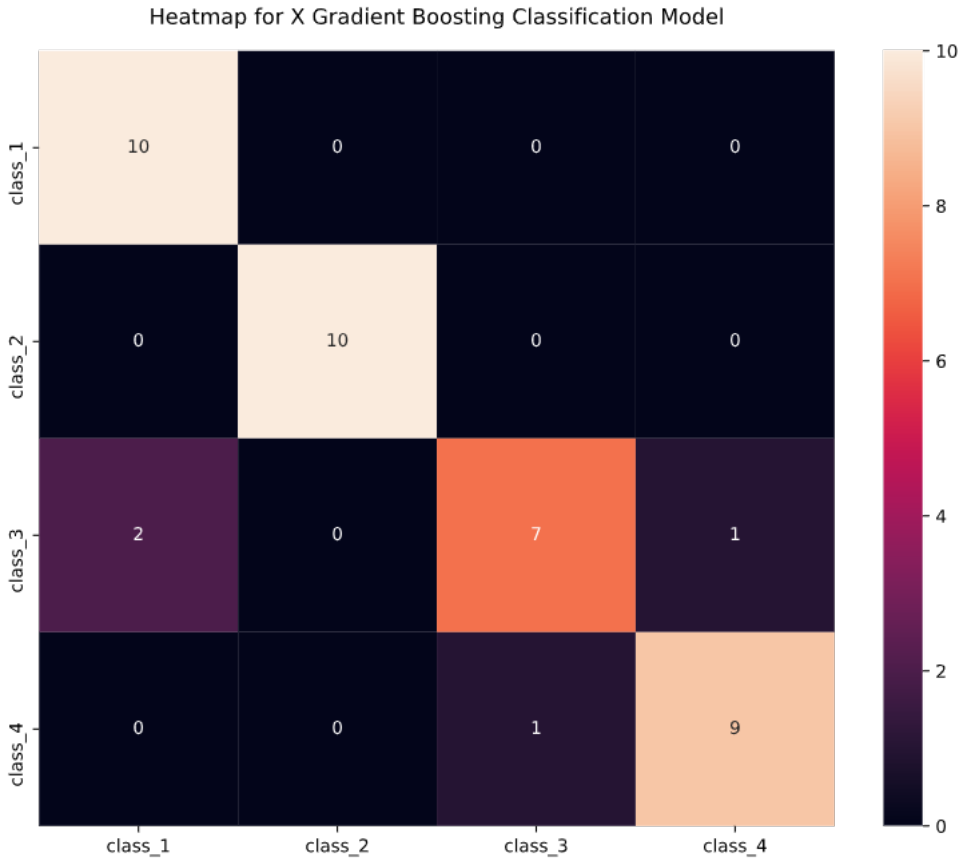
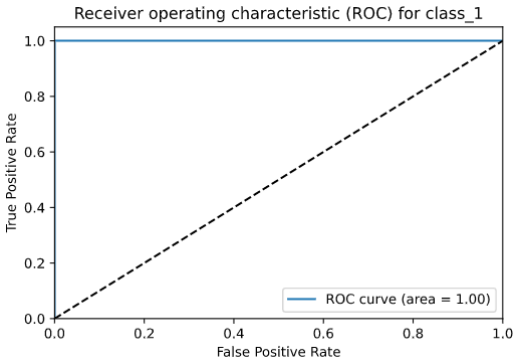
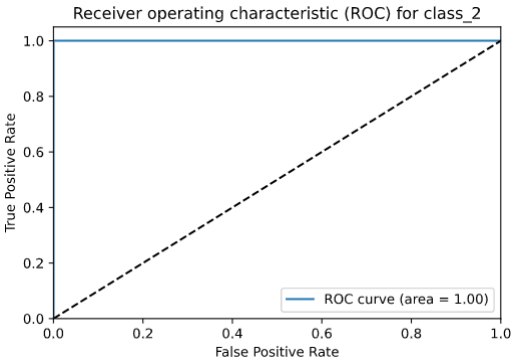


Figura 21 – Matriz de confusão e heatmap para o conjunto de teste - XGBoost L.R. :0.075.

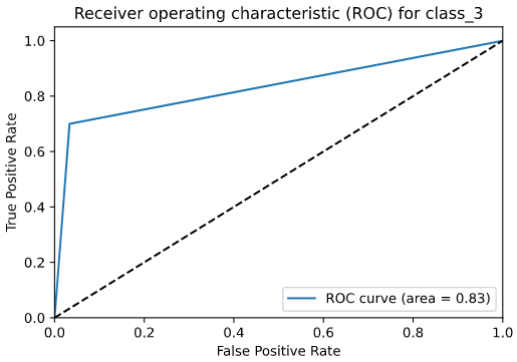
O valor do coeficiente Kappa para o classificador XGBoost foi de 0.86 com as seguintes métricas (13):



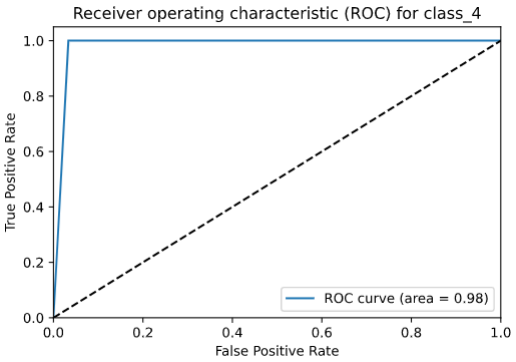
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 22 – Receiver operating characteristic (ROC) para o classificador XGBoost.

Métrica	**Class1**	**Class2**	**Class3**	**Class4**
Sensibilidade:	1	1	0.7	0.9
True Negative:	0.93	1	0.96	0.96
Precisão:	0.83	1	0.87	0.9
Pred. Negativa:	1	1	0.9	0.96
False Positive:	0.06	0	0.03	0.03
False Negative:	0	0	0.3	0.1
F Discovery:	0.16	0	0.125	0.1
Acurácia:	0.95	1	0.9	0.95

Tabela 13 – Métricas - XGBoost - L.R.:0.075

2.6 Support Vector Machine (SVM)

O objetivo de uma máquina de vetor suporte (do inglês Support Vector Machine (SVM)) é definir um hiperplano em um espaço $n-$dimensional de maneira que divida os pontos em classes, detalhes e demonstrações matemáticas para SVM podem ser encontradas em (BISHOP, 2006; SMOLA; SCHöLKOPF, 2004).

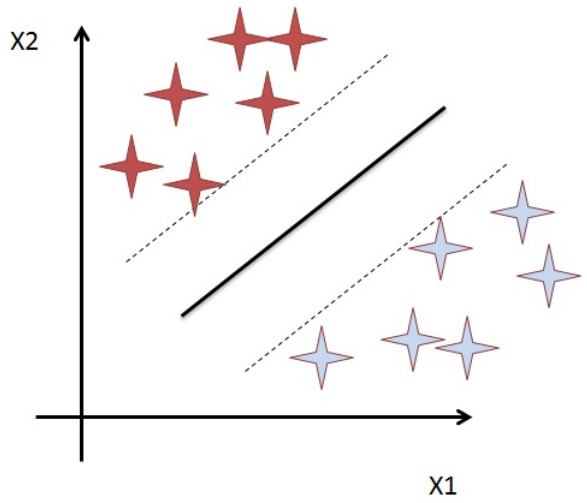


Figura 23 – Exemplo de classificação utilizando SVM. Fonte: Roni Shouval sob CC BY-SA 3.0

O algoritmo utilizado está implementado no pacote `svm` da biblioteca `sklearn` com o seguinte método `SVC`.

O método possui diversos parâmetros, como o objetivo deste relatório não é refinar métodos de machine learning optou-se por analisar somente um parâmetro do método SVM, contudo é válido reforçar que pode-se utilizar de diversas outras abordagens para obter melhores valores nas métricas de avaliação, como por exemplo uma busca em grid de diversos parâmetros e valores.

O parâmetro analisado foi a função kernel do algoritmo, analisando o comportamento do método com seguintes kernels: `'linear', 'poly', 'rbf', 'sigmoid'`. Os demais parâmetros foram considerados os valores default. A execução com diversas funções kernel resultou em valores de acurácia diversos [4], a maior taxa de acurácia foi obtida com o kernel linear, o qual foi utilizado ao decorrer das análises.

```
#file: svm.ipynb
krnl = ['linear', 'poly', 'rbf', 'sigmoid']
for kernel in krnl:
   svm_clf = OneVsRestClassifier(svm.SVC(kernel=kernel,
                                         probability=True))

   svm_fit = svm_clf.fit(X_train, y_train)

   print("Learning rate: ", kernel)
```

[4] Os demais valores estão no arquivo svm.ipynb.

```
print("Accuracy score (training): {0:.3f}".format(svm_fit.score(X_train,
y_train)))
print("Accuracy score (teste): {0:.3f}".format(svm_fit.score(X_test,
y_test)))
```

As métricas (Tabela 14), matriz de confusão (Figura 24) e curvas ROC (Figura 25) para cada uma das classes foram as seguintes:

```
#file: svm.ipynb
svm_clf = OneVsRestClassifier(svm.SVC(kernel='linear',
                                      probability=True))

y_predp_ovr_svm = svm_clf.fit(X_train, y_train).predict_proba(X_test)
y_pred_ovr_svm = svm_clf.fit(X_train, y_train).predict(X_test)
y_pred_svm = np.argmax(y_pred_ovr_svm, axis=1)
#Matriz de confusão
svm_cm = confusion_matrix(y_true=y_true,
                          y_pred=y_pred_svm)
#Heatmap
plt.figure(figsize = (10,8))
ax = plt.axes()
x_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for x-axis
y_axis_labels = ['class_1', 'class_2', 'class_3', 'class_4']
# labels for y-axis
sns.heatmap(svm_cm,
            vmin=0,
            vmax=10,
            annot=True,
            fmt="d",
            ax = ax,
            xticklabels=x_axis_labels,
            yticklabels=y_axis_labels)
ax.set_title('Heatmap for Gradient Boosting Classification Model',pad=15)
print(classification_report(y_true, y_pred_svm, target_names=class_label))
print("Accuracy:",metrics.accuracy_score(y_true, y_pred_svm))
```

	precision	**recall**	**f1-score**	**support**
class_1	0.59	1.00	0.74	10
class_2	1.00	1.00	1.00	10
class_3	0.67	0.20	0.31	10
class_4	0.90	0.90	0.90	10
accuracy			0.78	40
macro avg	0.79	0.78	0.74	40
weighted avg	0.79	0.78	0.74	40

Tabela 14 – Métricas - SVM Linear

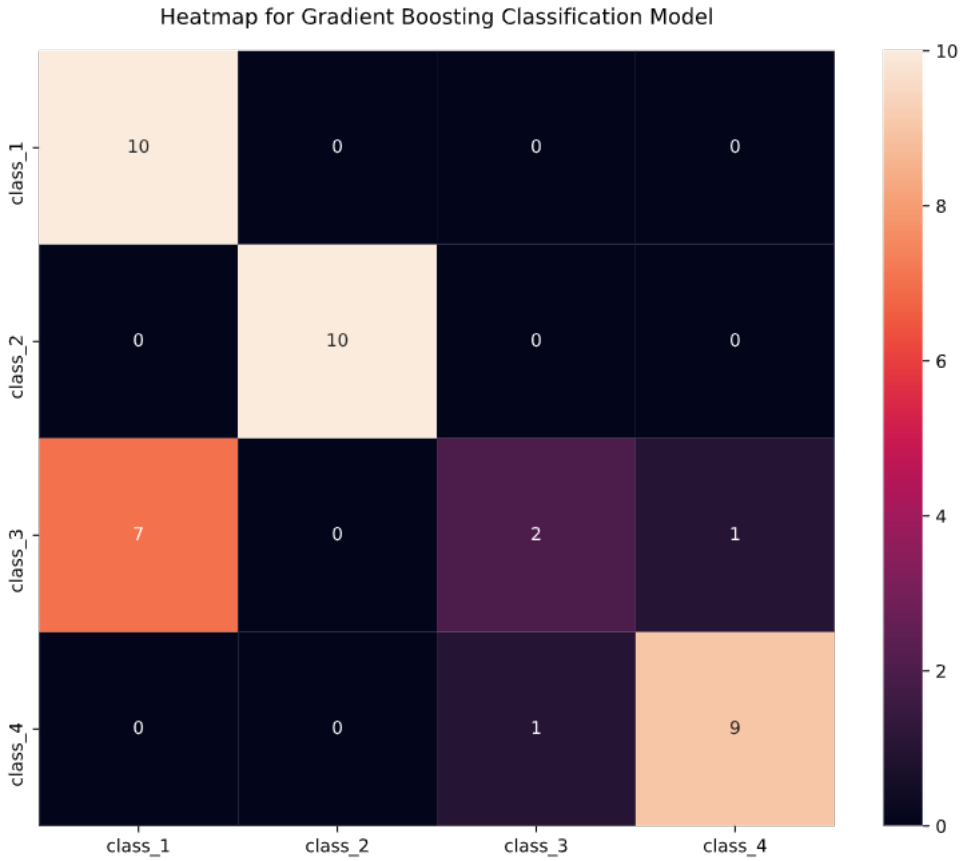
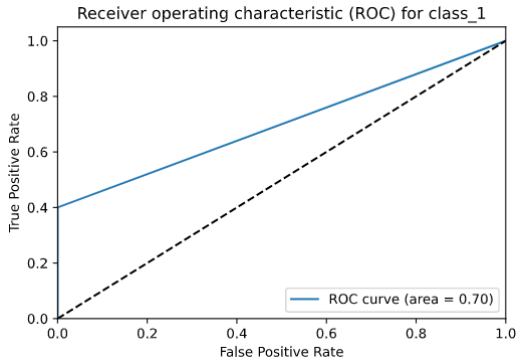
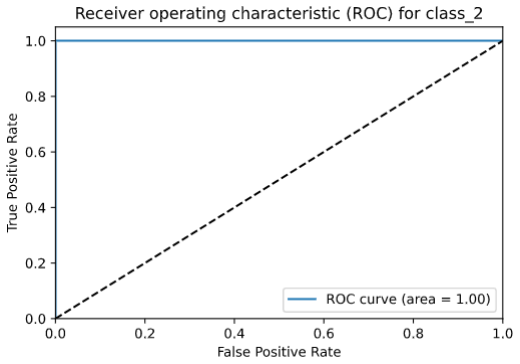


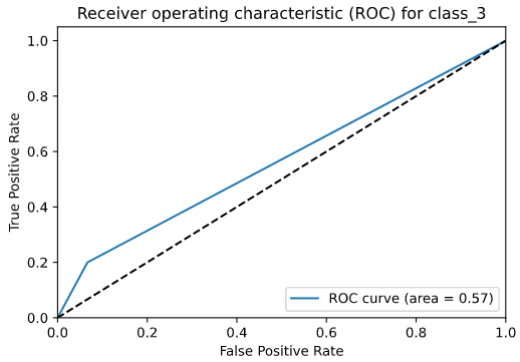
Figura 24 – Matriz de confusão e heatmap para o conjunto de teste - SVM Linear.



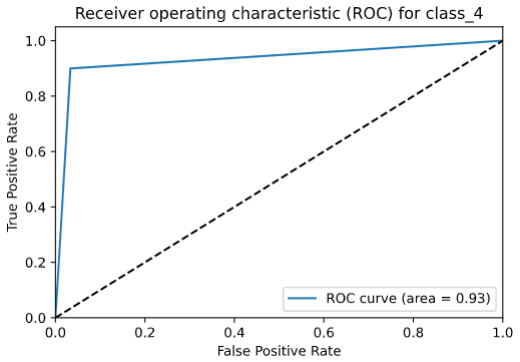
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 25 – Receiver operating characteristic (ROC) para o classificador SVM Linear.

O valor do coeficiente Kappa para o classificador Gradient Boosting foi de 0.7 com as seguintes métricas (15):

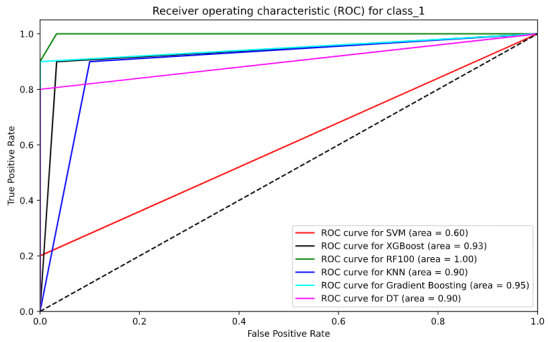
Métricas	**Class1**	**Class2**	**Class3**	**Class4**
Sensibilidade:	1	1	0.2	0.9
True Negative:	0.76	1	0.96	0.96
Precisão:	0.58	1	0.66	0.9
Pred. Negativa:	1	1	0.78	0.96
False Positive:	0.2	0	0.03	0.03
False Negative:	0	0	0.8	0.1
F Discovery:	0.41	0	0.33	0.1
Acurácia:	0.82	1	0.77	0.95

Tabela 15 – Métricas - SVM Linear

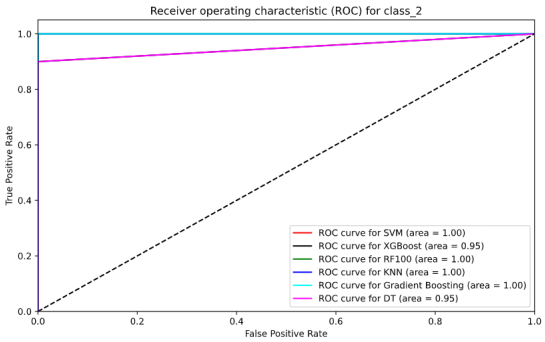
3 Conclusão

A execução de diversos métodos de aprendizagem de máquina proporcionou uma avaliação do comportamento ao dataset escolhido. Inicialmente foi realizado uma curva ROC para os métodos (Figura 26):

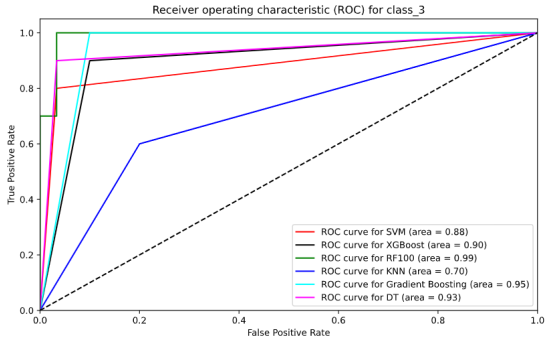
```
#file: comparison.ipynb
for i in range(class_label.shape[0]):
   plt.figure(figsize = (10,6))
   plt.plot(fpr_svm[i],
            tpr_svm[i],
            color='red',
            label='ROC curve for SVM (area = %0.2f)' % roc_auc_svm[i])
   plt.plot(fpr_xgb[i],
            tpr_xgb[i],
            color='black',
            label='ROC curve for XGBoost (area = %0.2f)' % roc_auc_xgb[i])
   plt.plot(fpr_RF100[i],
            tpr_RF100[i],
            color='green',
            label='ROC curve for RF100 (area = %0.2f)' % roc_auc_RF100[i])
   plt.plot(fpr_KNN1[i],
            tpr_KNN1[i],
            color='blue',
            label='ROC curve for KNN (area = %0.2f)' % roc_auc_KNN1[i])
   plt.plot(fpr_gb[i],
            tpr_gb[i],
            color='cyan',
            label='ROC curve for Gradient Boosting (area = %0.2f)' % roc_auc_gb[i])
   plt.plot(fpr_DT[i],
            tpr_DT[i],
            color='magenta',
            label='ROC curve for DT (area = %0.2f)' % roc_auc_DT[i])
   plt.plot([0, 1], [0, 1], 'k--')
   plt.xlim([0.0, 1.0])
   plt.ylim([0.0, 1.05])
   plt.xlabel('False Positive Rate')
   plt.ylabel('True Positive Rate')
   plt.title('Receiver operating characteristic (ROC) for %s' % class_label[i])
   plt.legend(loc="lower right")
   plt.show()
```



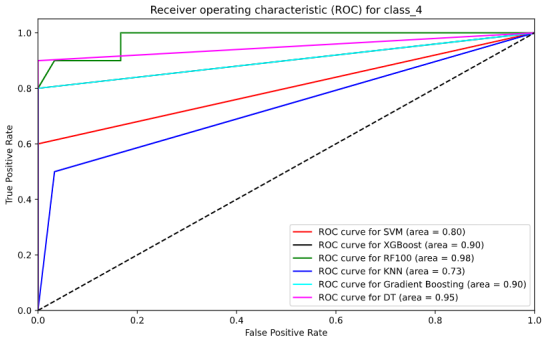
(a) Receiver operating characteristic (ROC) - $class_1$



(b) Receiver operating characteristic (ROC) - $class_2$



(c) Receiver operating characteristic (ROC) - $class_3$



(d) Receiver operating characteristic (ROC) - $class_4$

Figura 26 – Receiver operating characteristic (ROC) para os classificadores apresentados.

Após isso comparamos os resultados das acurácias através de um *radarplot* (Figura 27) e um *barplot* (Figura 28):

```
#file: comparison.ipynb
#Radarplot
df_rplot = pd.DataFrame(dict(ac=[ac_xgb, ac_svm, ac_rf, ac_knn, ac_gb,
ac_dt],
                             names_classif=['XGB','SVM', 'RF', 'KNN', 'GB',
                             'DT']))
fig = px.line_polar(df_rplot,
                    r='ac',
                    theta='names_classif',
                    line_close=True,
                    title='Radarplot - Accuracy')
fig.update_traces(fill='toself')
fig.show()
#Barplot
fig = plt.figure(figsize=(10,4))
ax = fig.add_axes([0,0,1,1])
name_classifier=['XG Boost','SVM', 'Random Forest', 'KNN', 'Gradient
Boost.', 'Decion Tree']
```

```
ac=[ac_xgb, ac_svm, ac_rf, ac_knn, ac_gb, ac_dt]
bar = ax.bar(name_class,ac, color=['black', 'red', 'green', 'blue', 'cyan',
'magenta'])
labels = name_classifier
opacity = 0.4
bar_width = 0.35
for rect, label in zip(bar, labels):
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2.0, height, label,
    ha='center', va='bottom')
ax.set_ylabel('Accuracy')
ax.set_title('Accuracy by Classifier')
plt.show()
```

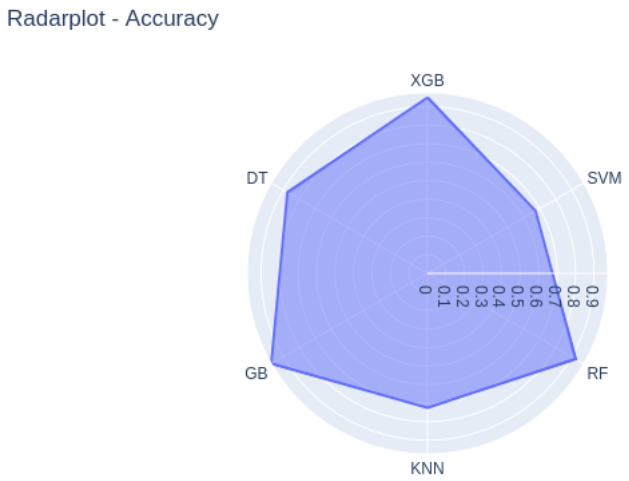


Figura 27 – Radarplot da acurácia dos métodos analisados.

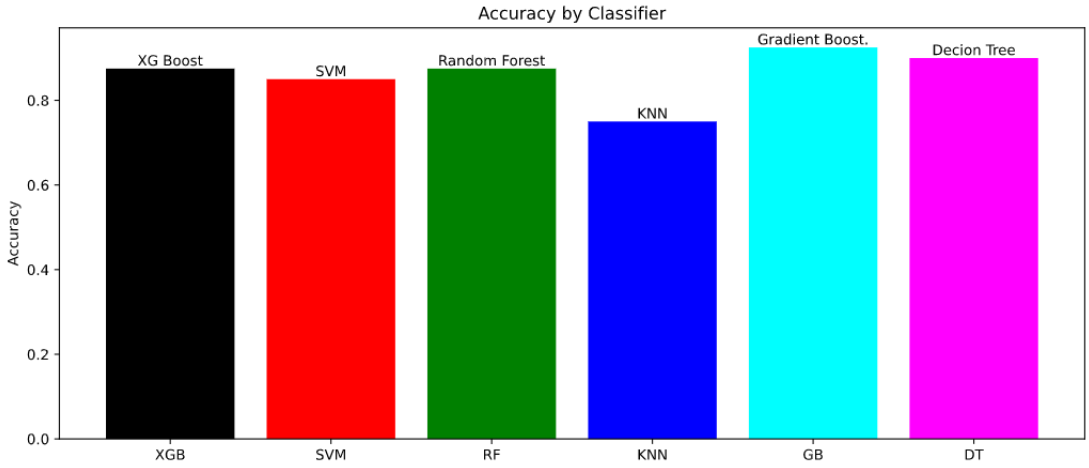


Figura 28 – Barplot da acurácia dos métodos analisados.

A advertência em relação a análise está na queantidade da dados utilizados para a geração das métricas, como já apresentado acima, estes valores estão restritos ao dataset e observações utilizadas, portanto a existência de valores ótimos em relação a métrica está diretamente relacionado a isso para cada uma das classes. Análises com um maior número de observações é necessária para a confirmação dos resultados, visto que em algumas classes o desempenho é exímio, não apresentando nenhum erro, o que na prática é uma situação muito difícil de ocorrer em todos os exemplares de teste. Diante disso, a comparação dos métodos foi capaz de mostrar que para esse dataset e metodologia adotada o método Gradient Boosting obteve uma maior acurácia quando comparado com os demais métodos, por outro lado métodos baseados em árvores sobressaíram aos outros métodos como $k-NN$ e SVM.

Referências

ALTMAN, N. S. An introduction to kernel and nearest-neighbor nonparametric regression. *The American Statistician*, Taylor & Francis, v. 46, n. 3, p. 175–185, 1992. Nenhuma citação no texto.

BISHOP, C. M. *Pattern Recognition and Machine Learning*. [S.l.]: Springer, 2006. Nenhuma citação no texto.

CHEN, T.; GUESTRIN, C. XGBoost. In: *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016. Disponível em: <https://doi.org/10.1145/2939672.2939785>. Nenhuma citação no texto.

CHILDS, L. et al. Identification and classification of ncRNA molecules using graph properties. *Nucleic Acids Research*, Oxford University Press (OUP), v. 37, n. 9, p. e66–e66, abr. 2009. Disponível em: <https://doi.org/10.1093/nar/gkp206>. Nenhuma citação no texto.

COHEN, J. A coefficient of agreement for nominal scales. *Educational and Psychological Measurement*, v. 20, n. 1, p. 37–46, 1960. Nenhuma citação no texto.

FRIEDMAN, J.; HASTIE, T.; TIBSHIRANI, R. Additive logistic regression: a statistical view of boosting (with discussion and a rejoinder by the authors). *The Annals of Statistics*, Institute of Mathematical Statistics, v. 28, n. 2, abr. 2000. Disponível em: <https://doi.org/10.1214/aos/1016218223>. Nenhuma citação no texto.

GARCÍA, S.; LUENGO, J.; HERRERA, F. *Data preprocessing in data mining*. [S.l.]: Springer, 2015. v. 72. Nenhuma citação no texto.

IGUAL, L.; SEGUÍ, S. *Introduction to Data Science*. Springer International Publishing, 2017. Disponível em: <https://doi.org/10.1007/978-3-319-50017-1>. Nenhuma citação no texto.

ITO, E. A. et al. BASiNET—BiologicAl sequences NETwork: a case study on coding and non-coding RNAs identification. *Nucleic Acids Research*, Oxford University Press (OUP), v. 46, n. 16, p. e96–e96, jun. 2018. Disponível em: <https://doi.org/10.1093/nar/gky462>. Nenhuma citação no texto.

NATEKIN, A.; KNOLL, A. Gradient boosting machines, a tutorial. *Frontiers in Neurorobotics*, Frontiers Media SA, v. 7, 2013. Disponível em: <https://doi.org/10.3389/fnbot.2013.00021>. Nenhuma citação no texto.

SAFAVIAN, S.; LANDGREBE, D. A survey of decision tree classifier methodology. *IEEE Transactions on Systems, Man, and Cybernetics*, Institute of Electrical and Electronics Engineers (IEEE), v. 21, n. 3, p. 660–674, 1991. Disponível em: <https://doi.org/10.1109/21.97458>. Nenhuma citação no texto.

SMOLA, A. J.; SCHöLKOPF, B. *A tutorial on support vector regression*. 2004. Nenhuma citação no texto.