IA048 – Aprendizado de Máquina

Exercícios de Fixação de Conceitos (EFC) 2 – 2s2020

Aluno: Rodrigo Santos Gonçalves **RA:** 265306

Parte 1 – Classificação binária

Problema: identificação do gênero do locutor a partir de trechos de voz
Base de dados: dados_voz_genero.csv
https://www.mldata.io/dataset-details/gender_voice/

Você dispõe de um conjunto de dados contendo 3168 amostras rotuladas. Cada amostra é descrita por 19 atributos acústicos extraídos de trechos gravados de voz, considerando a faixa de frequências de 0 a 280 Hz. A última coluna corresponde ao rótulo associado a cada padrão, sendo igual a '1' para o gênero masculino, e '0' para o gênero feminino.

a) Faça uma análise das características dos atributos de entrada considerando os respectivos histogramas e as medidas de correlação entre eles.

Resultados e Discussões:

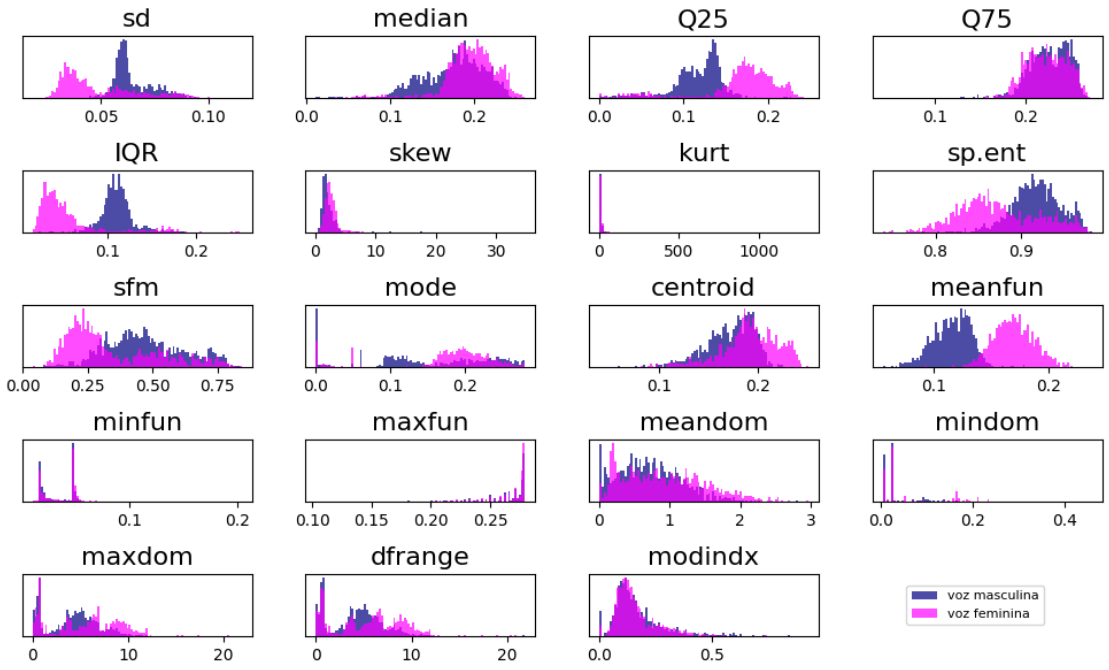

Figura 1 – Histogramas dos atributos de entrada, na cores azul se encontram os atributos relacionados aos rótulos de voz masculina e em rosa os rótulos atribuidos a voz feminina, em roxo temos a intersecção entre ambos histogramas.

Na figura 1 temos os respectivos histogramas das variáveis de entrada, observe que as variáveis onde a distinção entre as duas classes pelos histogramas é razoavelmente mais acentuada, são: sd, Q25, IQR, sp.ent e meanfun, nos demais histogramas é mais difícil

distinguir entre as duas classes, uma primeira observação no histograma dos atributos de entrada pode empiricamente fornecer uma noção da dependência entre os atributos de entrada e as classes do modelo de classificação. Os histogramas aqui representados graficamente foram obtidos utilizando a função numpy.histogram() com bins=100, onde bins é o número de "bandejas" com largura igual no histograma.

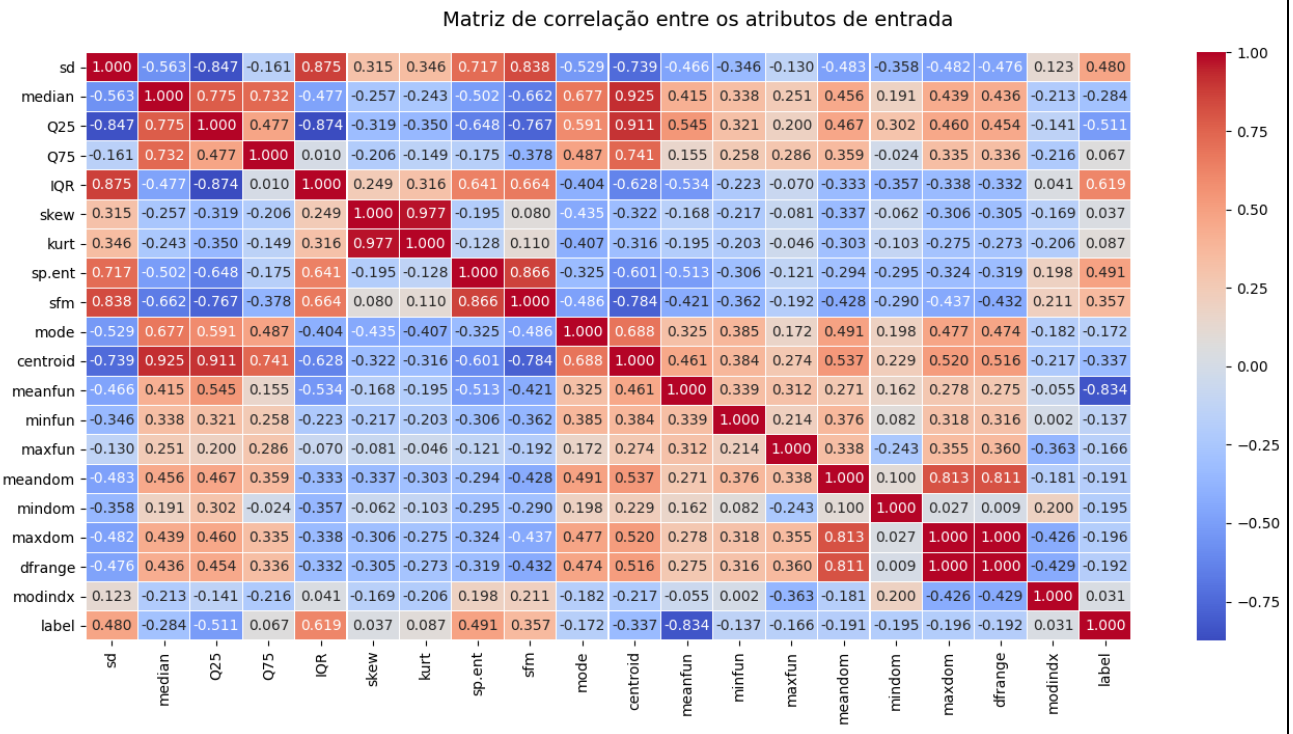

Figura 2 – Matriz de correlação dos atributos de entrada e saída, representada por meio de uma mapa de calor, quanto mais próximo do vermelho maior a correlação positiva entre os atributos, e quanto mais próximo do azul escuro maior a correlação negativa entre os atributos.

Na matriz de correlação da figura 2, pode se observar o grau de correlação entre dois atributos de entrada da base de dados ou seja o grau de dependência entre estes dois atributos, a partir da matriz se observa um grau de dependência muito alto entre as variáveis:

- IQR e sd;
- sfm e sd;
- centroid e median;
- sd e Q25 (-);
- IQR e Q25 (-);
- centroid e Q25;
- kurt e skew;
- sfm e sp.ent;
- maxdom e meandom;
- dfrange e meandom;

Aqui por simplificação, foram considerados os pares de atributos de entrada com modulo de correlação superior a 0,8, a matriz de correlação foi obtida utilizando o coeficiente de correlação de Pearson, que representa o grau de correlação entre valores no intervalo 1 e -1, a correlação negativa implica o quanto duas variáveis são inversamente proporcionais, ou seja quanto mais uma aumenta a outra diminui. Outro aspecto relevante da matriz de correlação é observar quais são as variáveis de entrada com maior valor absoluto de correlação com o atributo de saída (gênero da voz), observamos que:

- $|corr(label,\ meanfun)| = 0.834$
- $|corr(label,\ IQR)| = 0.619$
- $|corr(label,\ Q25)| = 0.511$
- $|corr(label,\ sp.ent)| = 0.491$
- $|corr(label,\ sd)| = 0.480$

Logo, é possível verificar uma correspondência entre os histogramas dos atributos de entrada que apresentavam visualmente um maior grau de separação entre as classe do atributo de saída e os maiores valores de correlação entre determinados atributos de entrada e o atributo de saída, observe que as cinco maiores correlações dos atributos de entrada são também os cinco histogramas que visualmente tem uma maior distinção entre a distribuição das classes de saída.

b) Construa, então, o modelo de regressão logística para realizar a classificação dos padrões. Para isso, reserve uma parte dos dados (e.g., 20%) para validação, usando todas as demais amostras para o treinamento do modelo. Pensem na pertinência e na possibilidade de realizar algum pré-processamento nos dados (e.g., normalização).
   Apresente e discuta os seguintes resultados com relação ao conjunto de validação:
   - ✓ A curva ROC;
   - ✓ A curva de evolução da F_1-medida em função do *threshold* de decisão.

Resultados e Discussões:

No conjunto de dados estudo, foi avaliado o quanto poderia contribuir em melhores resultados o pré-processamento dos atributos de entrada, através da biblioteca de pré processamento do sklearn, foram consideradas as funções de pré-processamento: Normalizer(), MinMaxScaler(), StandardScaler(), RobustScaler(), QuantileTransformer() e PowerTransformer(). Na análise as funções que apresentaram os melhores desempenhos

foram: StandardScaler(), RobustScaler(), QuantileTransformer() e PowerTransformer(), todas com desempenho no *f1-score* de 98%, e com pior desempenho foi a Normalizer() com *f1-score* de 96%, em todos os diferentes métodos de pré-processamento a área da curva ROC não apresentou alterações com relação ao modelo sem pré-processamento, em geral até nos melhores resultados os modelos com pré-processamento apresentaram resultados aproximadamente iguais ou inferiores ao modelo sem sua utilização, logo se adotou não utilizar pré-processamento por não proporcionar melhores desempenhos neste modelo.

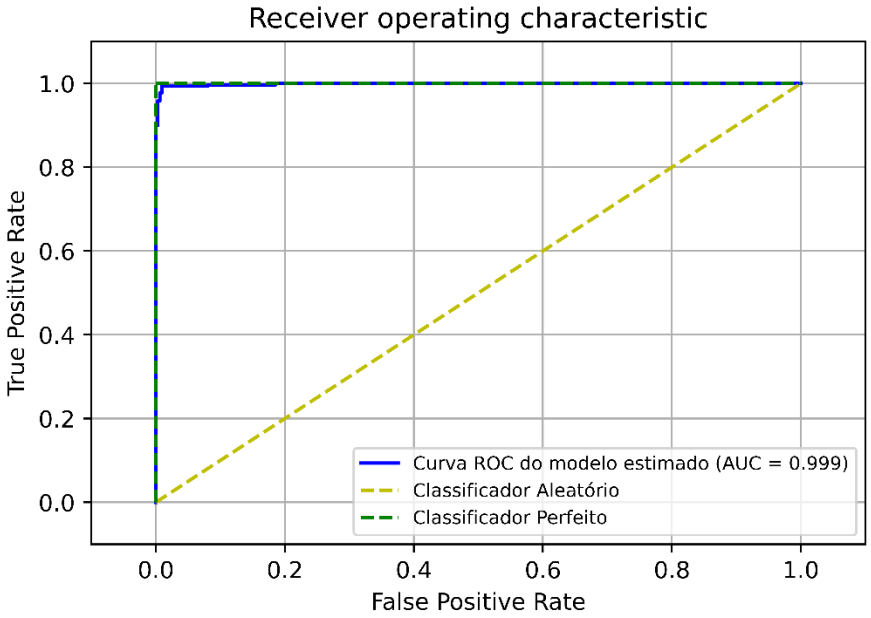

Figura 3 – Curva ROC do classificador.

Na figura 3 é apresentada a curva ROC do classificador desenvolvido somente com o conjunto de dados para treinamento, sendo testado com o conjunto reservado para validação que representa 20% do total dos dados originais. Através do método `train_test_split()` obtido de `sklearn.model_selection` todo o conjunto de dados é separado em instâncias para treinamento (80% dos dados totais) e instâncias para validação. O modelo de regressão logística é criado através da classe `LogisticRegressionCV()` e método `fit()`, disponível em `sklearn.linear_model`, adotando como o algoritmo de otimização o de Newton com gradiente conjugado, e um número máximo de iterações igual a 10000, outro ponto importante é que este método estima o modelo através de uma validação cruzada, no caso foram utilizadas 10 pastas.

A curva ROC foi estimada através da utilização do método `predict_proba()`

que foi utilizado para estimar as classificações dos atributos de saída do conjunto de validação em termos de probabilidades, e utilizando a função `roc_curve()` com os valores das instâncias de validação com as predições é plotada a curva ROC, a área é calculada com a função `roc_auc_score()`, e atingiu a valor de 99,9% da área unitária, o que representa um classificar bem próximo do ideal.

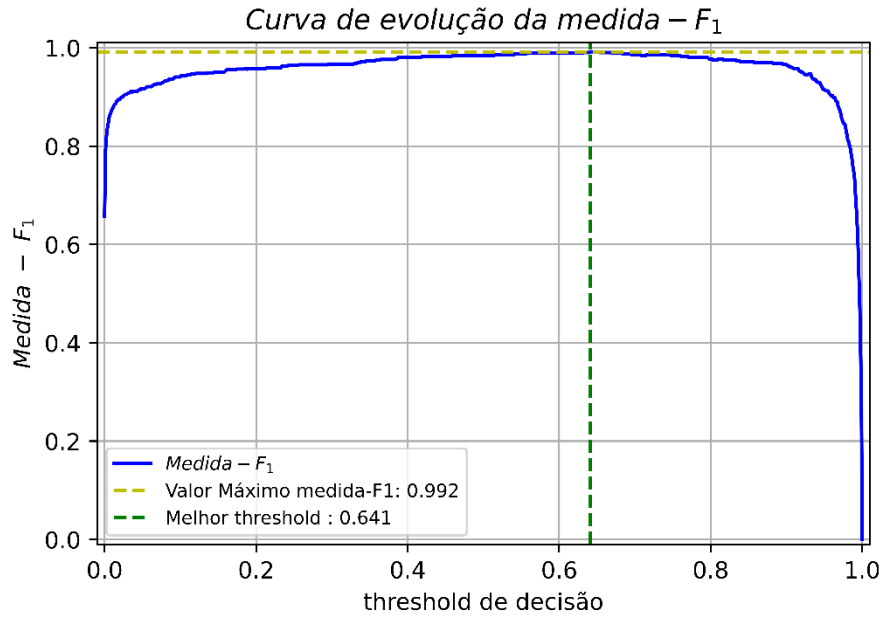

Figura 4 – Curva de evolução da medida - F_1 do classificador em função do *threshold*.

O gráfico da figura 4 foi estimado a partir da variação do valor de *threshold* no conjunto de dados de predições do modelo representados em termos de probabilidades, o conjunto de predições em termos de probabilidades, é possível ajustar o *threshold* e obter o conjunto de predições novamente em termos de classes, e calcular o *f1-score* para esse valor de *threshold*. O trecho de código a seguir foi utilizado para obter esse gráfico:

```
def funcao_degrau(v, threshold):
    if v >= threshold:
        return 1
    else:
        return 0

from sklearn.metrics import f1_score

medida_f1 = []

thresholds = np.arange(0.0, 1.001, 0.001)
for threshold in thresholds:
    ythreshold = np.array([funcao_degrau(x, threshold) for x in yprob])
    medida_f1.append(f1_score(y_validation, ythreshold))
```

No gráfico é perceptível uma estabilidade no *f1-score* para valores de *threshold* entre 0,2 e 0,8, enquanto que variações mais discrepantes ocorrem nos intervalos de valores abaixo de 0,2 e acima de 0,8, isto implica que as alterações no valor de *threshold*, influenciam na classificação de poucas instâncias, logo os valores de predições em termos de probabilidade para as duas classes se mantém para boa parte das instâncias próximos de 1,0 (voz masculina) ou 0,0 (voz feminina).

c) Indique qual seria o valor mais adequado para o *threshold* de decisão e por quê. Empregando, então, esse *threshold*, obtenha a matriz de confusão e a acurácia do classificador para o conjunto de validação. Comente os resultados obtidos.

Resultados e Discussões:

O valor mais adequado para o *threshold* foi adotado como o valor que torna *f1-score* máximo na curva do gráfico da figura 4, nele está indicado como 0,641 o valor de *threshold*, que maximiza o *f1-score* em 0,992 na figura 5 é exibida a matriz de confusão para este caso, e a acurácia obtida foi de 0,992, na tabela 1 estão listados os demais métricas para avaliação do classificador, observa-se que a precisão, sensibilidade e *f1-score* em termos de média macro ou ponderada, atingiram valores aproximadamente iguais a 0,992.

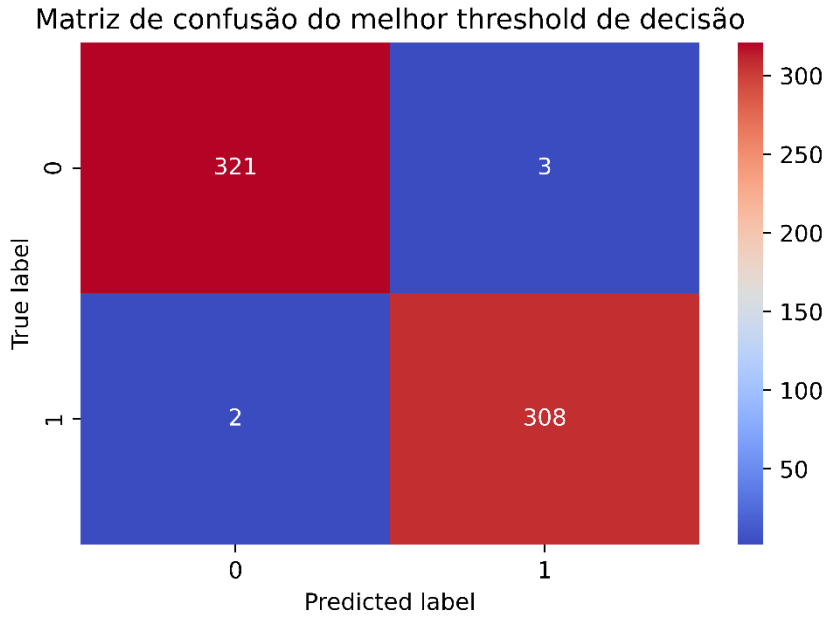

Figura 5 – Matriz de confusão do classificador com *threshold* de decisão mais adequado.

Além disso outro comentário pertinente é que o valor do *threshold* que maximiza o *f1-score* pode variar em função da forma como as instâncias de treino e teste são

sorteadas em ordens diferentes, por isso para manter a estabilidade entre uma simulação e outra é necessário fixar o valor da semente de `random_state` em `train_test_split()` no momento em que são distribuídas as instâncias, para sempre serem sorteadas na mesma ordem. Por último se fosse desenvolvido um modelo utilizando apenas os atributos de entrada meanfun, IQR, Q25, sp.ent e sd que são respectivamente os que possuem maior módulo de correlação com o atributo de saída, séria obtido um classificador com acurácia de 0,98 e *f1-score* de 0,98, o que implica que boa parte das variáveis de entrada são redundantes, inclusive sendo viável um classificador apenas com o atributo de entrada meanfun com acuraria e *f1-score* de 0,97, pois as variáveis IQR, Q25, sp.ent e sd apresentam valores de correlação entre si consideráveis.

Tabela 1 – Métricas do classificador com *threshold* ajustado em 0,641

	Precisão	Sensibilidade	*f1-score*	instâncias
gênero feminino	0,994	0,991	0,992	324
gênero masculino	0,990	0,994	0,992	310
acurácia			0,992	634
média macro	0,992	0,992	0,992	634
média ponderada	0,992	0,992	0,992	634

Parte 2 – Classificação multi-classe

Problema: identificação de atividade humana usando dados de *smartphones*
Base de dados: har_smartphone.zip
https://archive.ics.uci.edu/ml/datasets/human+activity+recognition+using+smartphones

Nesta atividade, o conjunto de dados contém atributos nos domínios do tempo e da frequência extraídos de sinais de acelerômetro e giroscópio de um *smartphone*. Os rótulos das amostras indicam qual a atividade realizada por um voluntário humano durante a aquisição dos sinais:

Rótulo	Atividade
0	Caminhada
1	Subindo escadas
2	Descendo escadas
3	Sentado
4	Em pé
5	Deitado

O conjunto de dados já está separado em uma parte para treinamento e outra para teste. Ao todo, temos 7352 amostras de treinamento e 2947 amostras de teste; cada amostra é descrita por 561 atributos temporais ou espectrais.
Dois métodos de classificação serão explorados nesta aplicação: regressão logística e

k-nearest neighbors.

a) Construa uma solução para este problema baseada no modelo de regressão logística. Descreva a abordagem escolhida para resolvê-lo (*softmax*, classificadores binários combinados em um esquema um-contra-um ou um-contra-todos). Obtenha, então, a matriz de confusão para o classificador considerando os dados do conjunto de teste.
Além disso, adote uma métrica global para a avaliação do desempenho (médio) deste classificador. Como sugestão, consulte a referência M. SOKOLOVA & G. LAPALME, "A Systematic Analysis of Performance Measures for Classification Tasks". *Information Processing & Management*, vol. 45, no. 4, pp. 427-437, 2009.
Discuta os resultados obtidos.

Resultados e Discussões:

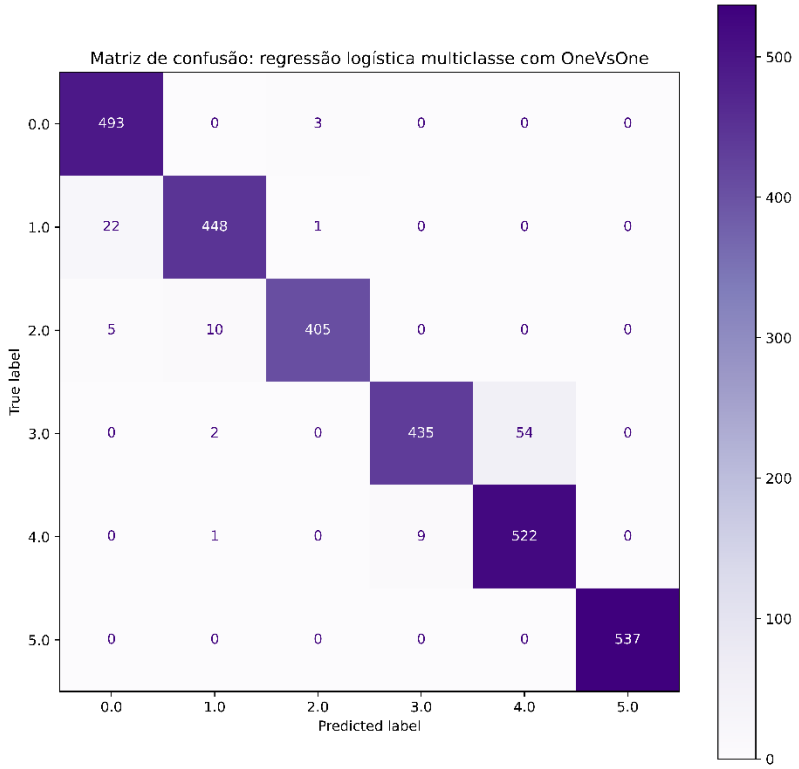

Figura 6 – Matriz de confusão do classificador utilizando a estratégia Um-contra-um nos dados de teste.

Na solução desse problema de classificação foram adotadas as duas estratégias de classificação multiclasse: Um-contra-um e Um-contra-todos, a aplicação das estratégias de classificação são feitas utilizando a biblioteca `sklearn.multiclass` e utilizando os métodos `OneVsOneClassifier()` e `OneVsRestClassifier()`, na implementação deve-se passar como argumento nos métodos o tipo de modelo que vai ser utilizado na classificação, como o modelo adotado é a regressão logística, portanto é passado como argumento para ambos os métodos a `LogisticRegressionCV()`, já mencionado na

parte 1 do trabalho, com os mesmo parâmetros utilizados naquela parte. Como é conhecido da estratégia Um-contra-um como temos 6 classes distintas teremos $6.(6-1)/2 = 15$ classificadores que irão distinguir entre apenas duas classes, ou seja classificador 0-1, classificador 0-2, classificador 0-3, ... e assim por diante, onde a classificação final dos atributos de entrada é dada pelo voto majoritário entre os classificadores. No caso da estratégia Um-contra-todos são utilizados 6 classificadores onde cada classificador adota apenas uma classe como positiva e as demais como negativas e através disto quanto o modelo é submetido a um conjunto de dados a classe de maior probabilidade é a classe selecionado pelo modelo. Note que em um primeiro momento intuitivamente o modelo Um-contra-um é aparentemente um classificador mais robusto pelo fato de a estratégia possuir uma certa "redundância" entretanto o classificador Um-contra-todos é computacionalmente menos custoso [professores].

Com relação ao artigo de Sokolov, o desafio é enquadrar as alterações entre as matrizes de confusão obtidas entre um modelo e outro em um dos oito casos particulares abordados no artigo, em um primeiro momento as alterações entre as matrizes de confusão 6x6 aqui obtidas, enquadrariam o problema entre os casos particulares I_2, I_3, I_4 e I_5, adotando uma heurística aqui elaborada para identificar a qual caso particular deve ser tratado o problema, representando as matrizes 6x6 de forma equivalente as matrizes 2x2 do artigo temos:

- Representação da matriz de Um-contra-um como uma matriz 2x2:

$$\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} 1346 & 58 \\ 49 & 1494 \end{bmatrix}$$

- Representação da matriz de Um-contra-todos como uma matriz 2x2:

$$\begin{bmatrix} tp & fn \\ fp & tn \end{bmatrix} \rightarrow \begin{bmatrix} 1351 & 57 \\ 51 & 1488 \end{bmatrix}$$

Obtendo a matriz de diferenças absolutas elemento a elemento das matrizes anteriores

$$\begin{bmatrix} |\Delta tp| & |\Delta fn| \\ |\Delta fp| & |\Delta tn| \end{bmatrix} \rightarrow \begin{bmatrix} 5 & 1 \\ 2 & 6 \end{bmatrix}$$

Logo a partir do resultado obtido onde a maior variação ocorre em Δtp e Δtn fica evidente que as matrizes aqui estudas pode ser considerados como enquadradas nos casos particulares I_2(Change of true negative counts) e/ou I_3 (Change of true positive counts), como pela tabela 6 do artigo nenhuma das métricas de classificações multiclasse é invariante para o caso particular I_3, adotamos que o caso particular se trata do I_2, logo métricas como precisão, sensibilidade e *f1-score* apresentaram invariância, enquanto acurácia e taxa de erro não são métricas adequadas para esse caso. No nosso caso existe

um leve desbalanceamento entre as classes por exemplo a maior tem 537 instâncias enquanto a menor tem 420, logo aqui neste problema como a média macro trata todas as classes igualmente e a média micro favorece as classes maiores, será adotada uma média ponderada para compensar o desbalanceamento [2].

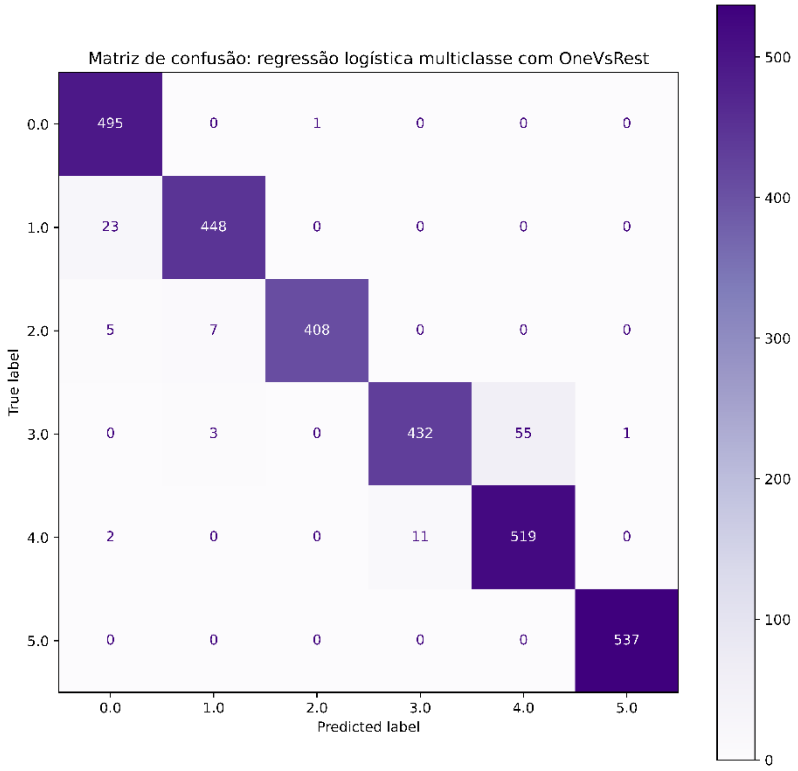

Figura 7 – Matriz de confusão do classificador utilizando a estratégia Um-contra-todos nos dados de teste.

Tabela 2 – Métricas do classificador Um-contra-um

	Precisão	Sensibilidade	*f1-score*	instâncias
média ponderada	0,965059	0,963692	0,963581	2947

Tabela 3 – Métricas do classificador Um-contra-todos

	Precisão	Sensibilidade	*f1-score*	instâncias
média ponderada	0.964771	0.963353	0.963222	2947

Levando em consideração valores até a 6ª casa decimal e adotando *f1-score* como a métrica global podemos ver que o classificador com melhor desempenho é o Um-contra-um, entretanto em uma situação prática devido à baixa diferença percentual de desempenho entre os dois modelos, algo em torno de 0,036%, é mais apropriado utilizar o classificador Um-contra-todos devido o menor custo computacional da implementação.

b) Considere, agora, a técnica *k-nearest neighbors* (*k*NN). Varie o parâmetro *k* e analise as matrizes de confusão obtidas junto aos dados de teste e o desempenho médio (computado com a mesma métrica adotada no item (a)). Comente os resultados obtidos, inclusive estabelecendo uma comparação com o desempenho da regressão logística.

Resultados e Discussões:

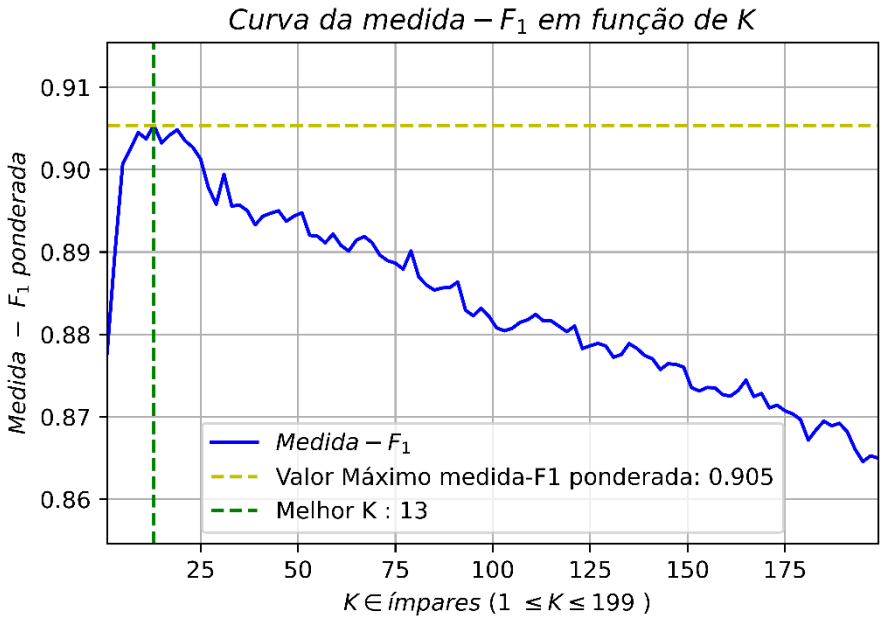

Figura 8 – Curva de medida - F_1

No figura 8 foi plotado um gráfica com o desempenho da métrica *f1-score* com o modelo de classificador KNN, ele foi implementado utilizando a biblioteca `sklearn.neighbors`, e a classe `KNeighborsClassifier()`, com o método `fit()`, o gráfico é obtido variando o valor do K e calculando o respectivo *f1-score*, como o problema de classificação aqui abordado possui um número de classes distintas par é uma boa prática adotar o valor de K como um número ímpar, pois a classificação é feita calculando o K vizinhos mais próximos da instância que se quer classificar e adotar um K ímpar evita um empate, no resultado do gráfico observa-se uma queda no desempenho a medida que se utiliza mais vizinhos para a classificação de maneira bem rápida, isto pode ser consequência da maldição da dimensionalidade ou *"Curse of dimensionality"*, pois temos uma base de dados com uma quantidade de atributos elevada: 561 atributos, e em geral bases de dados com muitos atributos de entrada, tem uma tendência a enganar a algoritmo KNN.

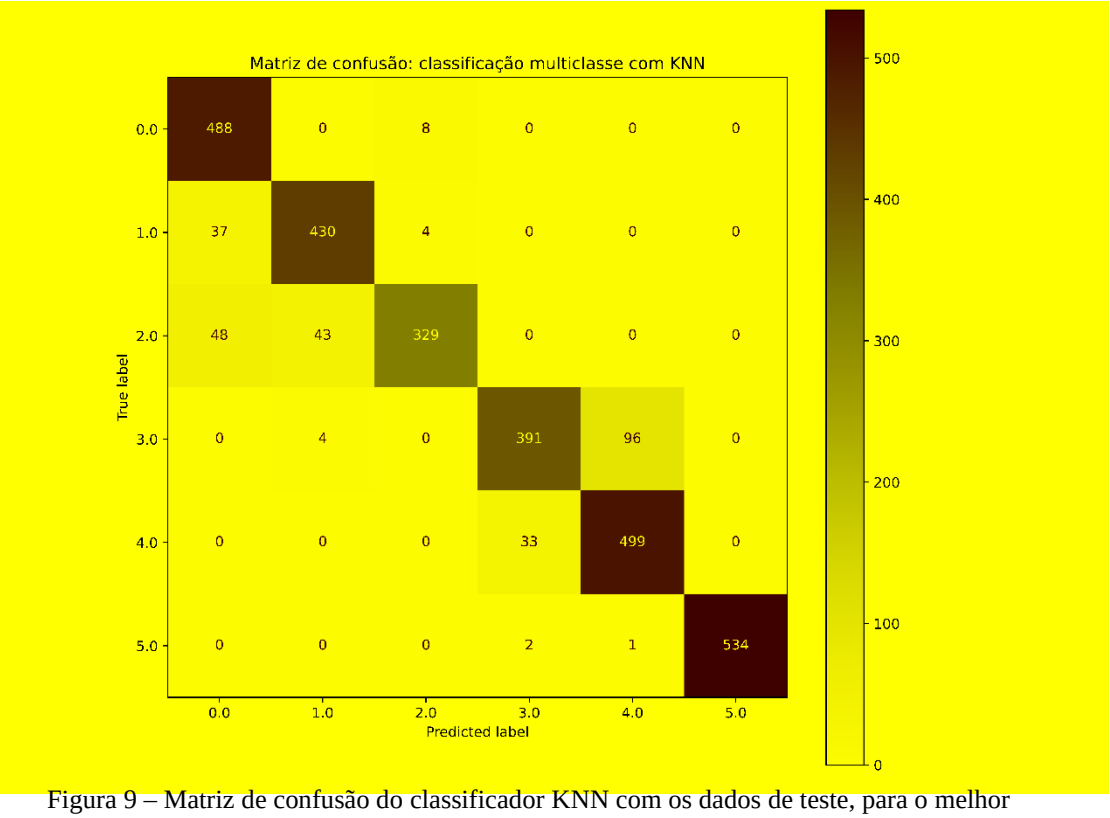

Figura 9 – Matriz de confusão do classificador KNN com os dados de teste, para o melhor classificador obtido, no caso com K=13.

Na figura 9 é exposta a matriz de confusão do classificador KNN, observa-se uma considerável queda no número de classificações corretas para cada classe, a seguir temos a tabela com os resultados considerando a mesma métrica global do item anterior.

Tabela 4 – Métricas do classificador KNN

	Precisão	Sensibilidade	*f1-score*	instâncias
média ponderada	0,911201	0,906345	0,905389	2947

Observa-se uma queda razoável no desempenho ao utilizar um classificador KNN, pois seu *f1-score* é de 0,905389 enquanto o melhor modelo em desempenho no item anterior com regressão logística com estratégia Um-contra-um tem *f1-score* de 0,963581, logo temos uma diferença percentual de desempenho de 5,82% entre um classificador multiclasse com regressão logística e outro com classificador KNN.

Considerações finais:

- No relatório, não é necessário descrever a teoria sobre os modelos de regressão logística e *k -nearest neighbors*. Não obstante, todas as escolhas feitas com respeito às características dos modelos, dos dados e dos experimentos devem ser apresentadas e justificadas, de modo a possibilitar a reprodução da metodologia.

Referências:

[1] Attux, Romis. Boccato, Levy. Notas de aula: Classificação Linear. Unicamp – FEEC – DCA, 2020.

[2] SOKOLOVA & G. LAPALME, "A Systematic Analysis of Performance Measures for Classification Tasks". Information Processing & Management, vol. 45, no. 4, pp. 427-437, 2009.

Links para os exercícios: no Google Colab:

- Parte 1 – Classificação binária: https://colab.research.google.com/drive/1oTrxmvFgGyE1lie7u3_whBRUyOmAhyJq?usp=sharing

- Parte 2 – Classificação multi-classe: https://colab.research.google.com/drive/1hT20BPMcJFF0jsq7vbcGUZ5DU6qYvGoS?usp=sharing

Anexos:

- Parte 1 – Classificação binária:

```
import pandas as pd

dataset_voz = pd.read_csv('/content/dados_voz_genero.csv', sep=';')
voz_masculina = dataset_voz.loc[dataset_voz['label']==1.0,:]
voz_feminina = dataset_voz.loc[dataset_voz['label']==0.0,:]

import numpy as np
import matplotlib.pyplot as plt

colunas_fonte = np.asarray(dataset_voz.columns)
figure_size=(10,6)
fig, axes = plt.subplots(5, 4, figsize=figure_size)
fig.canvas.set_window_title("Histograma dos atributos de entrada")
ax = axes.ravel()
plt.rcParams['legend.fontsize'] = 5
for i in range(19):
    _, bins = np.histogram(dataset_voz.loc[:,colunas_fonte[i]],
bins=100)
    ax[i].hist(voz_masculina.loc[:,colunas_fonte[i]], bins=bins,
color='navy', alpha=.7)
    ax[i].hist(voz_feminina.loc[:,colunas_fonte[i]], bins=bins,
color='magenta', alpha=.7)
    ax[i].set_title(colunas_fonte[i],size = 16)
    ax[i].set_yticks(())
    ax[i].legend(["voz masculina", "voz feminina"], loc="best")
fig.tight_layout()
plt.show()

import seaborn as sns

#correlacao_entradas = dataset_voz.drop(columns=['label'])
correlacao_entradas = dataset_voz
f, ax = plt.subplots(figsize=(15, 10))
f.canvas.set_window_title("Matriz de correlação entre os atributos de
entrada")
correlacao = correlacao_entradas.corr()
matriz_de_correlacao = sns.heatmap(round(correlacao, 3), annot=True,
ax=ax, cmap="coolwarm", fmt='.3f', linewidths=.05)
f.subplots_adjust(top=0.95)
t= f.suptitle('Matriz de correlação entre os atributos de entrada',
fontsize=14)
plt.show()

from sklearn.model_selection import train_test_split

x_train, x_validation, y_train, y_validation =
train_test_split(dataset_voz.drop('label', axis=1),
dataset_voz['label'], test_size=0.2, random_state=813)

from sklearn.linear_model import LogisticRegressionCV

logisticModel = LogisticRegressionCV(cv = 10, max_iter=10000,
solver='newton-cg' ,n_jobs=-1).fit(x_train, y_train)
predictions = logisticModel.predict(x_validation)
probabilities_predictions = logisticModel.predict_proba(x_validation)

from sklearn.metrics import  classification_report, confusion_matrix
```

```
print(classification_report(y_validation,predictions))
print(confusion_matrix(y_validation, predictions))

yprob = probabilities_predictions[:, 1]

from sklearn.metrics import roc_auc_score

auc = roc_auc_score(y_validation, yprob)

from sklearn.metrics import roc_curve

false_positives, true_positives, _ = roc_curve(y_validation, yprob)
f.canvas.set_window_title("Receiver operating characteristic")
plt.title("Receiver operating characteristic")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.xlim(-0.1, 1.1)
plt.ylim(-0.1, 1.1)
plt.grid(True)
plt.rcParams['legend.fontsize'] = 8
plt.plot(false_positives, true_positives, color='blue', label='Curva
ROC do modelo estimado (AUC = %0.3f)' % auc)
plt.plot([0, 1], [0, 1], color='y', linestyle='dashed',
label='Classificador Aleatório')
#plt.plot([0-0.006, 1], [1+0.006, 1+0.006], color='g',
linestyle='dotted', label='Classificador Perfeito')
#plt.plot([0-0.006, 0-0.006], [0-0.006, 1+0.006], linestyle='dotted',
color='g')
plt.plot([0, 1], [1, 1], color='g', linestyle='dashed',
label='Classificador Perfeito')
plt.plot([0, 0], [0, 1], linestyle='dashed', color='g')
plt.legend(loc='best')
plt.savefig('Receiver operating characteristic.png', dpi=1080,
facecolor='w', edgecolor='w', orientation='landscape', papertype=None,
format=None,transparent=False, bboxinches=None,
padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()


def funcao_degrau(v, threshold):
    if v >= threshold:
        return 1
    else:
        return 0

from sklearn.metrics import f1_score

medida_f1 = []

thresholds = np.arange(0.0, 1.001, 0.001)
for threshold in thresholds:
    ythreshold = np.array([funcao_degrau(x, threshold) for x in yprob])
    medida_f1.append(f1_score(y_validation, ythreshold))


f.canvas.set_window_title("Curva de evolução da medida-F1")
plt.title(r"$Curva \ de \ evolução \ da \ medida-F_{1}$")
plt.xlabel("threshold de decisão")
plt.ylabel(r"$Medida \ -\ F_{1}$")
plt.xlim(-0.01, 1.01)
plt.ylim(-0.01, 1.01)
plt.grid(True)
plt.plot(thresholds, medida_f1, color='b', label=r'$Medida-F_{1}$')
```

```
plt.plot([-0.01, 1.01], [max(medida_f1), max(medida_f1)], color='y',
linestyle='dashed', label='Valor Máximo medida-F1: %0.3f' %
max(medida_f1))
plt.plot([thresholds[medida_f1.index(max(medida_f1))],
thresholds[medida_f1.index(max(medida_f1))]], [-0.01, 1.01], color='g',
linestyle='dashed', label='Melhor threshold : %0.3f' %
thresholds[medida_f1.index(max(medida_f1))])
plt.legend(loc='best')
plt.savefig('Curva de evolução da medida-F1.png', dpi=1080,
facecolor='w', edgecolor='w', orientation='landscape', papertype=None,
format=None,transparent=False, bboxinches=None,
padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()

best_threshold = thresholds[medida_f1.index(max(medida_f1))]
ybest  = np.array([funcao_degrau(x, best_threshold) for x in yprob])

f.canvas.set_window_title("Matriz de confusão do melhor threshold de
decisão")
sns.heatmap(confusion_matrix(y_validation, ybest), annot=True, fmt="d",
cmap="coolwarm")
plt.title("Matriz de confusão do melhor threshold de decisão")
plt.xlabel("Predicted label")
plt.ylabel("True label")
plt.savefig('Matriz de confusão do melhor threshold de decisão.png',
dpi=1080, facecolor='w', edgecolor='w', orientation='landscape',
papertype=None, format=None,transparent=False, bboxinches=None,
padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()

print(classification_report(y_validation,ybest, digits=3))
print(confusion_matrix(y_validation, ybest))

print("Instâncias com rótulos de voz masculina: %d" %
len(list(filter(lambda i: i == 1.0, y_validation))))
print("Instâncias com rótulos de voz feminina: %d" %
len(list(filter(lambda i: i == 0.0, y_validation))))
```

- Parte 2 – Classificação multi-classe:

```
import pandas as pd
import numpy as np
import time

ini = time.time()

har_smartphone_Xtrain = pd.read_csv('/content/X_train.txt',
header=None)
X_train = np.float64(np.asarray([x[0].split() for x in
har_smartphone_Xtrain.values]))
har_smartphone_ytrain = pd.read_csv('/content/y_train.txt',
header=None)
y_train = np.float64(np.asarray(har_smartphone_ytrain))-1

har_smartphone_Xtest = pd.read_csv('/content/X_test.txt', header=None)
X_test = np.float64(np.asarray([x[0].split() for x in
har_smartphone_Xtest.values]))
har_smartphone_ytest = pd.read_csv('/content/y_test.txt', header=None)
y_test = np.float64(np.asarray(har_smartphone_ytest))-1

from sklearn.model_selection import train_test_split
```

```
xTrain, x_validation, yTrain, y_validation = train_test_split(X_train,
y_train, train_size=0.99, random_state=0)
X_train = np.append(xTrain, x_validation, axis=0)
y_train = np.append(yTrain, y_validation, axis=0)

y_train = y_train.ravel()
y_test = y_test.ravel()

from sklearn.linear_model import LogisticRegressionCV
from sklearn.multiclass import OneVsOneClassifier

logisticModel_OneVsOne = OneVsOneClassifier(LogisticRegressionCV(cv =
10, max_iter=10000, solver='newton-cg' ,n_jobs=-1)).fit(X_train,
y_train)

import matplotlib.pyplot as plt
from sklearn.metrics import plot_confusion_matrix

fig, ax = plt.subplots(figsize=(10, 10))
disp = plot_confusion_matrix(logisticModel_OneVsOne, X_test,
y_test.ravel(), cmap=plt.cm.Purples,values_format='d', ax=ax)
plt.title("Matriz de confusão: regressão logística multiclasse com
OneVsOne")
plt.savefig('Matriz de confusão: regressão logística multiclasse com
OneVsOne.png', dpi=1080, facecolor='w', edgecolor='w',
orientation='landscape', papertype=None, format=None,transparent=False,
bboxinches=None, padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()

from sklearn.metrics import classification_report, precision_score,
recall_score, f1_score

predictions_OneVsOne = logisticModel_OneVsOne.predict(X_test)

print(classification_report(y_test,predictions_OneVsOne, digits=6))

print("precisão micro: %0.16f" % precision_score(y_test,
predictions_OneVsOne, average='micro'))
print("precisão macro: %0.16f" % precision_score(y_test,
predictions_OneVsOne, average='macro'))
print("precisão weighted: %0.16f" % precision_score(y_test,
predictions_OneVsOne, average='weighted'))
print("recall micro: %0.16f" % recall_score(y_test,
predictions_OneVsOne, average='micro'))
print("recall macro: %0.16f" % recall_score(y_test,
predictions_OneVsOne, average='macro'))
print("recall weighted: %0.16f" % recall_score(y_test,
predictions_OneVsOne, average='weighted'))
print("f1-score micro: %0.16f" % f1_score(y_test, predictions_OneVsOne,
average='micro'))
print("f1-score macro: %0.16f" % f1_score(y_test, predictions_OneVsOne,
average='macro'))
print("f1-score weighted: %0.16f" % f1_score(y_test,
predictions_OneVsOne, average='weighted'))


from sklearn.multiclass import OneVsRestClassifier

logisticModel_OneVsRest = OneVsRestClassifier(LogisticRegressionCV(cv =
10, max_iter=10000, solver='newton-cg' ,n_jobs=-1)).fit(X_train,
y_train)

fig, ax = plt.subplots(figsize=(10, 10))
disp = plot_confusion_matrix(logisticModel_OneVsRest, X_test, y_test,
```

```
cmap=plt.cm.Purples,values_format='d', ax=ax)
plt.title("Matriz de confusão: regressão logística multiclasse com
OneVsRest")
plt.savefig('Matriz de confusão: regressão logística multiclasse com
OneVsRest.png', dpi=1080, facecolor='w', edgecolor='w',
orientation='landscape', papertype=None, format=None,transparent=False,
bboxinches=None, padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()

predictions_OneVsRest = logisticModel_OneVsRest.predict(X_test)

print(classification_report(y_test, predictions_OneVsRest, digits=6))

print("precisão micro: %0.16f" % precision_score(y_test,
predictions_OneVsRest, average='micro'))
print("precisão macro: %0.16f" % precision_score(y_test,
predictions_OneVsRest, average='macro'))
print("precisão weighted: %0.16f" % precision_score(y_test,
predictions_OneVsRest, average='weighted'))
print("recall micro: %0.16f" % recall_score(y_test,
predictions_OneVsRest, average='micro'))
print("recall macro: %0.16f" % recall_score(y_test,
predictions_OneVsRest, average='macro'))
print("recall weighted: %0.16f" % recall_score(y_test,
predictions_OneVsRest, average='weighted'))
print("f1-score micro: %0.16f" % f1_score(y_test,
predictions_OneVsRest, average='micro'))
print("f1-score macro: %0.16f" % f1_score(y_test,
predictions_OneVsRest, average='macro'))
print("f1-score weighted: %0.16f" % f1_score(y_test,
predictions_OneVsRest, average='weighted'))

from sklearn.neighbors import KNeighborsClassifier

array_f1_weighted = []
impares = np.arange(1, 200, 2)
for k in impares:
  KNN = KNeighborsClassifier(n_neighbors=k).fit(X_train, y_train)
  predictions_KNN = KNN.predict(X_test)
  array_f1_weighted.append(f1_score(y_test, predictions_KNN,
average='weighted'))

plt.title(r"$Curva \ da \ medida-F_{1} \ em \ função \ de \ K$")
plt.xlabel(r'$ \ K \in ímpares  \ (1\  \leq K \leq 199 \ ) $')
plt.ylabel(r"$Medida \ -\ F_{1}\ ponderada$")
plt.xlim(min(impares), max(impares))
plt.ylim(min(array_f1_weighted)-0.01, max(array_f1_weighted)+0.01)
plt.grid()
plt.plot(impares, array_f1_weighted, color='b', label=r'$Medida-
F_{1}$')
plt.plot([min(impares)-0.01, max(impares)+0.01],
[max(array_f1_weighted), max(array_f1_weighted)], color='y',
linestyle='dashed', label='Valor Máximo medida-F1 ponderada: %0.3f' %
max(array_f1_weighted))
plt.plot([impares[array_f1_weighted.index(max(array_f1_weighted))],
impares[array_f1_weighted.index(max(array_f1_weighted))]],
[min(array_f1_weighted)-0.01, max(array_f1_weighted)+0.01], color='g',
linestyle='dashed', label='Melhor K : %d' %
impares[array_f1_weighted.index(max(array_f1_weighted))])
plt.legend(loc='best')
plt.savefig('Curva da medida-F1 em função de K.png', dpi=1080,
facecolor='w', edgecolor='w', orientation='landscape', papertype=None,
format=None,transparent=False, bboxinches=None,
padinches=0.1,frameon=None, bbox_inches='tight')
```

```
plt.show()

Kbest = impares[array_f1_weighted.index(max(array_f1_weighted))]
KNN = KNeighborsClassifier(n_neighbors=Kbest).fit(X_train, y_train)

fig, ax = plt.subplots(figsize=(10, 10))
disp = plot_confusion_matrix(KNN, X_test, y_test,
cmap=plt.cm.Purples,values_format='d', ax=ax)
plt.title("Matriz de confusão: classificação multiclasse com KNN")
plt.savefig('Matriz de confusão: classificação multiclasse com
KNN.png', dpi=1080, facecolor='w', edgecolor='w',
orientation='landscape', papertype=None, format=None,transparent=False,
bboxinches=None, padinches=0.1,frameon=None, bbox_inches='tight')
plt.show()

predictions_KNN = KNN.predict(X_test)

print(classification_report(y_test, predictions_KNN, digits=6))

print("precisão micro: %0.16f" % precision_score(y_test,
predictions_KNN, average='micro'))
print("precisão macro: %0.16f" % precision_score(y_test,
predictions_KNN, average='macro'))
print("precisão weighted: %0.16f" % precision_score(y_test,
predictions_KNN, average='weighted'))
print("recall micro: %0.16f" % recall_score(y_test, predictions_KNN,
average='micro'))
print("recall macro: %0.16f" % recall_score(y_test, predictions_KNN,
average='macro'))
print("recall weighted: %0.16f" % recall_score(y_test, predictions_KNN,
average='weighted'))
print("f1-score micro: %0.16f" % f1_score(y_test, predictions_KNN,
average='micro'))
print("f1-score macro: %0.16f" % f1_score(y_test, predictions_KNN,
average='macro'))
print("f1-score weighted: %0.16f" % f1_score(y_test, predictions_KNN,
average='weighted'))

fim = time.time()

print("Tempo de execução de código: %0.18f segundos" % (fim-ini))
```