TRABALHO MACHINE LEARNING

GRUPO:

Matheus Garibaldi

Vitor Ruan

Leon Fagundes

Dataset utilizado: https://www.kaggle.com/datasets/adilshamim8/student-depression-dataset

Repositório com modelos: https://github.com/leonfagundes27/student_depression_prediction

(Cada modelo está separado em um arquivo .py com o nome referente a quantidade de neurônios da(s) sua(s) camada(s) oculta(s).)

Os algoritmos preditivos a seguir visam como atributo-alvo a predição de depressão em estudantes, utilizando as seguintes classes para os dados de treinamento:

Gênero, Idade, Cidade, Profissão, Pressão Acadêmica, Pressão no Trabalho, CRA, Satisfação com os Estudos, Satisfação no Trabalho, Duração do Sono, Hábitos Alimentares, Grau de Escolaridade, "Você já teve pensamentos suicidas?", Horas de Trabalho/Estudo, Estresse Financeiro, Histórico Familiar de Doença Mental.

São 27.901 objetos com essas características, cujo o atributo alvo (Depressão) é classificado como 0 para não-depressivo e 1 para depressivo.

Os modelos estão momeados com a quantidade de neurônios de sua(s) camada(s) oculta(s). Cada algoritmo carrega características distintas entre si variando a parametrização desde a razão de sepação de dados, função de ativação, taxa de aprendizado e, o mais importante da tarefa proposta, a quantidade de neurônios sua(s) camada(s) oculta(s). Contando com 10 modelos diferentes, visamos instanciar uma boa quantidade de modelos diferentes em parâmetro a fim de realizar uma análise sobre como cada parâmetro influencia na acurácia da rede neural.

ANÁLISE DOS MODELOS DESENVOLVIDOS

1. Ranking dos Modelos

O modelo 10.py obteve a melhor acurácia (84.95%), seguido por 20.py (82.87%). Modelos com mais neurônios nem sempre tiveram melhor desempenho.

2. Análise Comparativa

Neurônios: Modelos com menos neurônios (como 10) se saíram melhor.

Função de Ativação: Sigmoid/Logistic foi mais eficaz do que Relu.

Taxa de Aprendizado: A constante teve bom desempenho.

Épocas: Mais épocas nem sempre resultaram em melhor performance.

3. Parâmetros Importantes

O número de neurônios e a função de ativação influenciaram bastante. Modelos simples com Sigmoid/Logistic se destacaram.

4. Conclusão

O modelo mais simples (10.py) com 10 neurônios e Sigmoid teve o melhor desempenho, mostrando que menos complexidade pode ser mais eficiente.

Algoritmo	Acurácia (%)	Dados de Treinamento/Teste	Neurônios na Camada Oculta	Função de Ativação	Taxa de Aprendizado	Épocas (Previstas/Realizadas)
10.py	84.9534	70/30	10	Sigmoid/Logistic	Constante	500/22
20.py	82.8673	75/25	20	Sigmoid/Logistic	Constante	750/750
50+50.py	81.7443	85/15	50+50	Sigmoid/Logistic	Constante	1800/571
50.py	79.9283	65/35	50	Sigmoid/Logistic	Constante	1000/1000
200.py	79.251	80/20	200	Relu	Constante	1200/156
100.py	79.036	80/20	100	Relu	Adaptativa	2000/204
100+100.py	78.6379	90/10	100+100	Sigmoid/Logistic	Constante	1500/922
200+200.py	78.387	80/20	200+200	Sigmoid/Logistic	Constante	1500/570
10+10.py	78.144	90/10	10+10	Relu	Adaptativa	2300/726
20+20.py	76.7918	75/25	20+20	Relu	Adaptativa	2450/317

Abaixo, a análise individual de cada algoritmo.

Primeiro algoritmo (10.py):

Dados de treinamento e teste separados em 70/30

Tolerância: 10^-3

N° de neurônios da camada oculta: 10

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

Acurácia: 84.9534%

Épocas (Previstas / Realizadas): 500 / 22

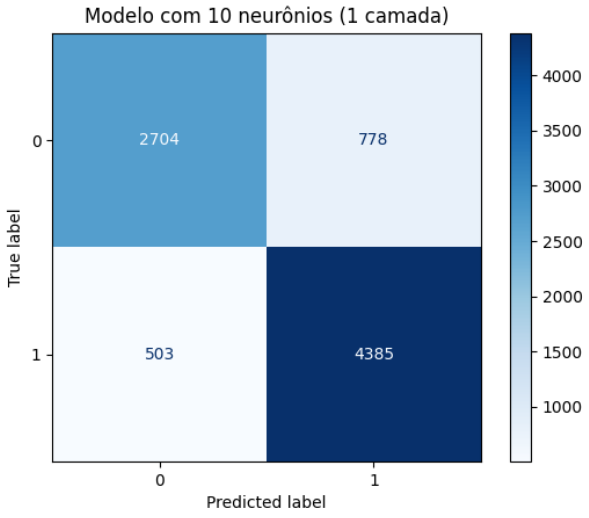
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.3, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

modelo = MLPClassifier(
    hidden_layer_sizes=(10,),
    max_iter=500,
    tol=1e-3,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.8469534050179212
Aperte enter para continuar! 😊

Matriz de Confusão:
[[2704  778]
 [ 503 4385]]

Classes =  [0 1]
Erro =  0.3499375299737051
Amostras visitadas =  429616
Atributos de entrada =  117
N ciclos =  22
N de camadas =  3
Tamanhos das camadas ocultas:  (10,)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```



Segundo algoritmo (20.py):

Dados de treinamento e teste separados em 75/25

Tolerância: 10^-5

N° de neurônios da camada oculta: 20

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

Acurácia: 82.8673%

Épocas (Previstas / Realizadas): 750 / 750

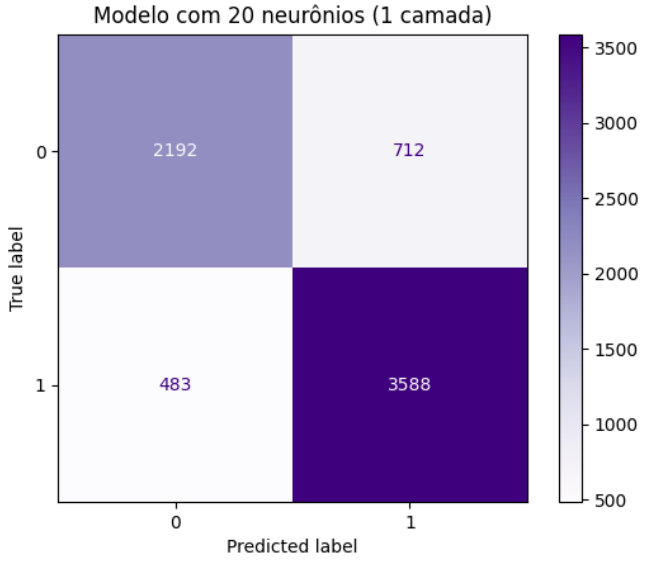
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

modelo = MLPClassifier(
    hidden_layer_sizes=(20,),
    max_iter=750,
    tol=1e-5,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.828673835125448
Aperte enter para continuar! 😊

Matriz de Confusão:
[[2192  712]
 [ 483 3588]]

Classes =  [0 1]
Erro =  0.2906120360625962
Amostras visitadas =  15692250
Atributos de entrada =  117
N ciclos =  750
N de camadas =  3
Tamanhos das camadas ocultas:  (20,)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```



Terceiro algoritmo (50.py):

Dados de treinamento e teste separados em 65/35

Tolerância: 10^-8

N° de neurônios da camada oculta: 50

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

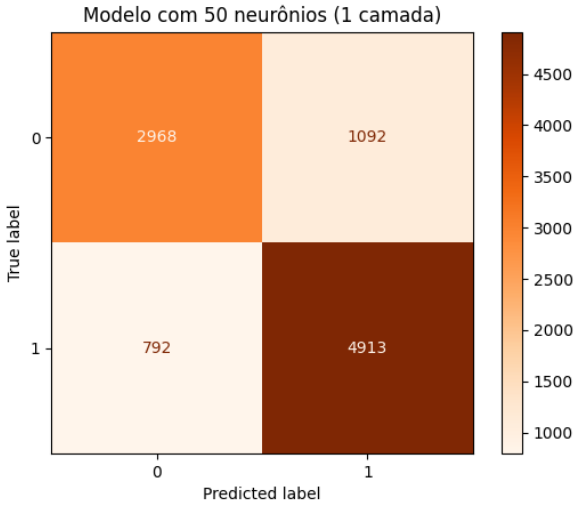
Acurácia: 79.9283%

Épocas (Previstas / Realizadas): 1000/ 1000

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.35, random_state=42)

modelo = MLPClassifier(
    hidden_layer_sizes=(50,),
    max_iter=1000,
    tol=1e-8,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.7992831541218638
Classes =  [0 1]
Erro =  0.16874026044550172
Amostras visitadas =  18133000
Atributos de entrada =  117
N ciclos =  1000
N de camadas =  3
Tamanhos das camadas ocultas:  (50,)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```



Quarto algoritmo (100.py):

Dados de treinamento e teste separados em 80/20

Tolerância: 10^-6

N° de neurônios da camada oculta: 100

Função de ativação: Relu

Taxa de aprendizado: Adaptativa

Acurácia: 79.0360%

Épocas (Previstas / Realizadas): 2000/ 204

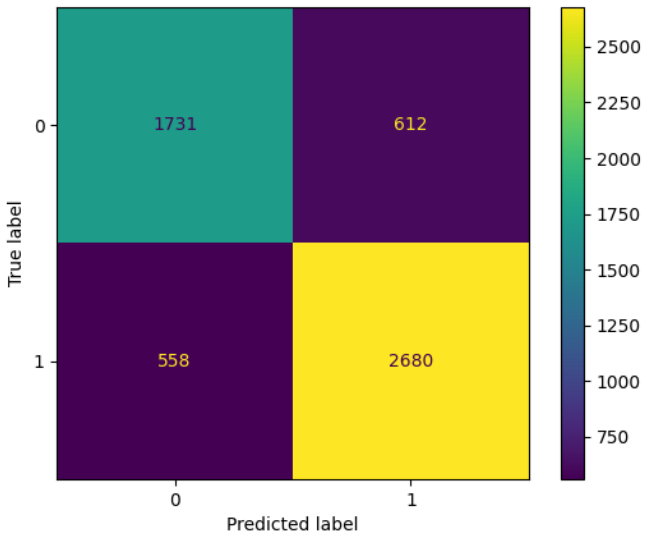
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

neural_network = MLPClassifier(
    verbose=True,
    hidden_layer_sizes=(100),
    max_iter=2000,
    tol=1e-6,
    activation='relu',
    learning_rate='adaptive'
)
```

```
Acurácia: 0.7903601505106612
Aperte enter para continuar! 😊
Aperte enter para continuar! 😊


Matriz de Confusão:
[[1731  612]
 [ 558 2680]]

Classes =  [0 1]
Erro =  0.0016062680815808765
Amostras visitadas =  4553280
Atributos de entrada =  514
N ciclos =  204
N de camadas =  3
Tamanhos das camadas ocultas:  100
N de neurons saida =  1
```



Quinto algoritmo (200.py):

Dados de treinamento e teste separados em 80/20

Tolerância: 10^-8

N° de neurônios da camada oculta: 200

Função de ativação: Relu

Taxa de aprendizado: Constante

Acurácia: 79.2510%

Épocas (Previstas / Realizadas): 1200/ 156

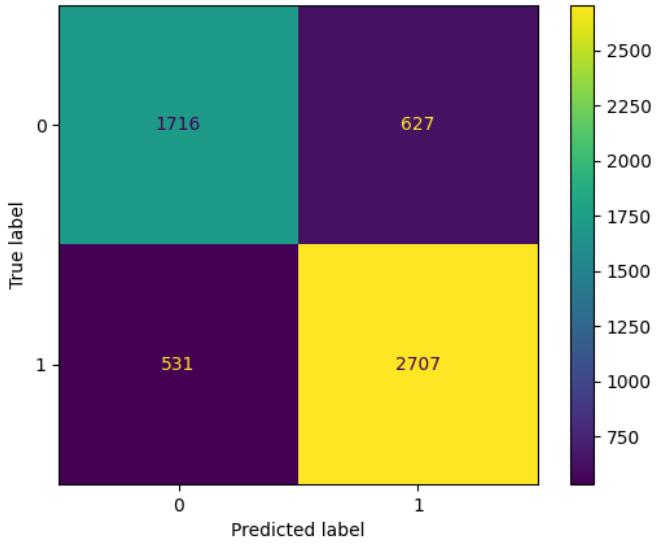
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2, random_state=42)

neural_network = MLPClassifier(
    verbose=True,
    hidden_layer_sizes=(200),
    max_iter=1200,
    tol=1e-8,
    activation='relu',
    learning_rate='constant'
)
```

```
Acurácia: 0.792510302813116
Aperte enter para continuar! 😊

Matriz de Confusão:
[[1716  627]
 [ 531 2707]]

Classes =  [0 1]
Erro =  0.0014165495528631776
Amostras visitadas =  3481920
Atributos de entrada =  514
N ciclos =  156
N de camadas =  3
Tamanhos das camadas ocultas:  200
N de neurons saida =  1
```



Sexto algoritmo (10+10.py):

Dados de treinamento e teste separados em 90/10

Tolerância: 10^-7

N° de neurônios das camadas ocultas: 10 e 10

Função de ativação: Relu

Taxa de aprendizado: Adaptativa

Acurácia: 78.1440%

Épocas (Previstas / Realizadas): 2300/ 726

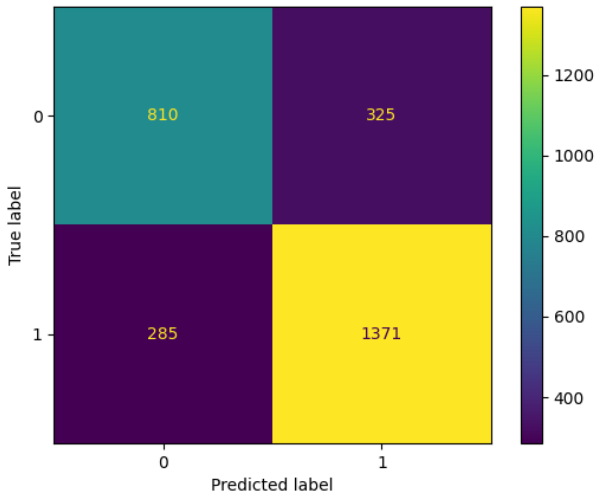
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.1, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

neural_network = MLPClassifier(
    verbose=True,
    hidden_layer_sizes=(10, 10),
    max_iter=2300,
    tol=1e-7,
    activation='relu',
    learning_rate='adaptive'
)
```

```
Acurácia: 0.7814403439627373
Aperte enter para continuar! 😊

Matriz de Confusão:
[[ 810  325]
 [ 285 1371]]

Classes =  [0 1]
Erro =  0.18800321725348873
Amostras visitadas =  18229860
Atributos de entrada =  514
N ciclos =  726
N de camadas =  4
Tamanhos das camadas ocultas:  (10, 10)
N de neurons saida =  1
```



Sétimo algoritmo (20+20.py):

Dados de treinamento e teste separados em 75/25

Tolerância: 10^-9

N° de neurônios das camadas ocultas: 20 e 20

Função de ativação: Relu

Taxa de aprendizado: Adaptativa

Acurácia: 76.7918%

Épocas (Previstas / Realizadas): 2450/ 317

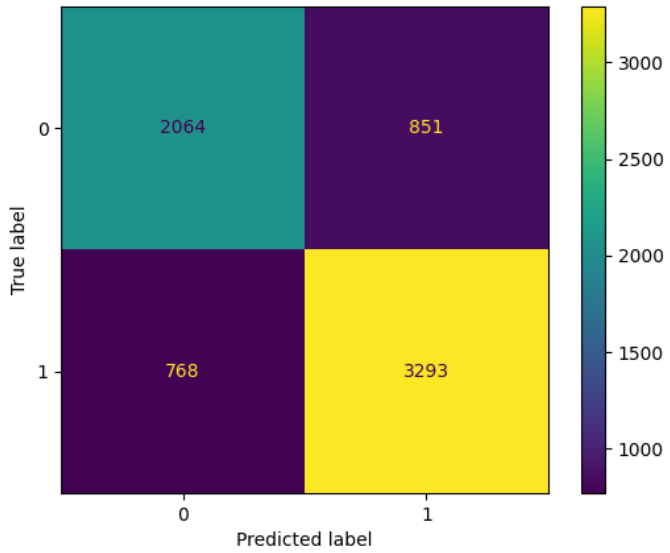
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.25, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

neural_network = MLPClassifier(
    verbose=True,
    hidden_layer_sizes=(20, 20),
    max_iter=2450,       Matheus, anteontem via  PR #1 • ✨ feat/new funccionalities …
    tol=1e-9,
    activation='relu',
    learning_rate='adaptive'
)
```

```
Acurácia: 0.7679185779816514
Aperte enter para continuar! 😊

Matriz de Confusão:
[[2064  851]
 [ 768 3293]]

Classes =  [0 1]
Erro =  0.015387714409167582
Amostras visitadas =  6633225
Atributos de entrada =  514
N ciclos =  317
N de camadas =  4
Tamanhos das camadas ocultas:  (20, 20)
N de neurons saida =  1
```



Oitavo algoritmo (50+50.py):

Dados de treinamento e teste separados em 85/15

Tolerância: 10^-5

N° de neurônios das camadas ocultas: 50 e 50

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

Acurácia: 81.7443%

Épocas (Previstas / Realizadas): 1800/ 571

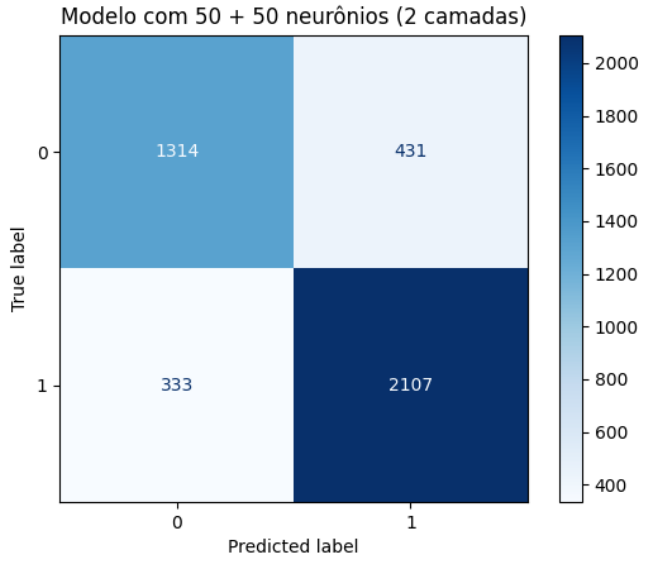
```
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.15, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

modelo = MLPClassifier(
    hidden_layer_sizes=(50, 50),
    max_iter=1800,
    tol=1e-5,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.8174432497013142
Aperte enter para continuar! 😊

Matriz de Confusão:
[[1314  431]
 [ 333 2107]]

Classes =  [0 1]
Erro =  0.2421886383436895
Amostras visitadas =  13540123
Atributos de entrada =  117
N ciclos =  571
N de camadas =  4
Tamanhos das camadas ocultas:  (50, 50)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```



Nono algoritmo (100+100.py):

Dados de treinamento e teste separados em 90/10

Tolerância: 10^-9

N° de neurônios das camadas ocultas: 100 e 100

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

Acurácia: 78.6379%

Épocas (Previstas / Realizadas): 1500/ 922

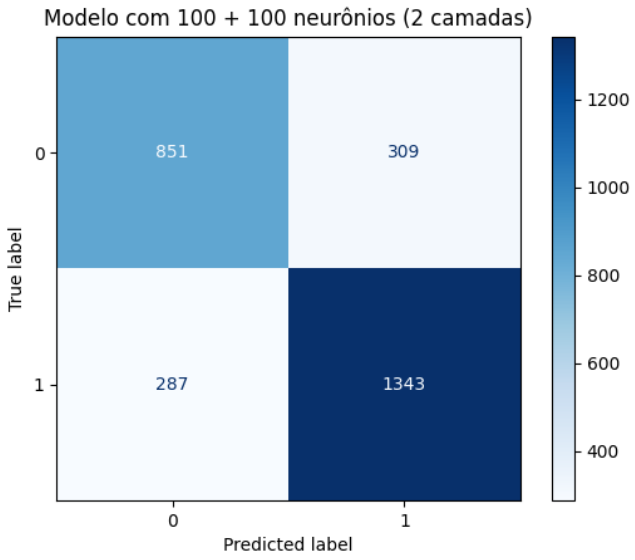
```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.10, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

modelo = MLPClassifier(
    hidden_layer_sizes=(100, 100),
    max_iter=1500,
    tol=1e-9,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.7863799283154121
Aperte enter para continuar! 😊

Matriz de Confusão:
[[ 851  309]
 [ 287 1343]]

Classes =  [0 1]
Erro =  0.05964348267943637
Amostras visitadas =  23149576
Atributos de entrada =  117
N ciclos =  922
N de camadas =  4
Tamanhos das camadas ocultas:  (100, 100)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```



Décimo algoritmo (200+200.py):

Dados de treinamento e teste separados em 80/20

Tolerância: 10^-5

N° de neurônios das camadas ocultas: 200 e 200

Função de ativação: Sigmoid/Logistic

Taxa de aprendizado: Constante

Acurácia: 78.3870%

Épocas (Previstas / Realizadas): 1500/ 570

```python
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.20, random_state=42)
print('\n Formato dados de treinamento: ', X_train.shape, Y_train.shape)
print('\n Formato dados de teste: ', X_test.shape, Y_test.shape)
input('Aperte enter para continuar! 😊\n')

modelo = MLPClassifier(
    hidden_layer_sizes=(200, 200),
    max_iter=1500,
    tol=1e-5,
    activation='logistic',
    learning_rate='constant',
    verbose=True
)
```

```
Acurácia: 0.7838709677419354
Aperte enter para continuar! 😊

Matriz de Confusão:
[[1731  617]
 [ 589 2643]]

Classes =  [0 1]
Erro =  0.017995491559322
Amostras visitadas =  12721260
Atributos de entrada =  117
N ciclos =  570
N de camadas =  4
Tamanhos das camadas ocultas:  (200, 200)
N de neurônios na saída =  1
Função de ativação da saída =  logistic
```