EFC 3

Jimi Togni - RA: 226359
Rodrigo de Freitas Pereira - RA: 192063

1. Parte I - Devivação

Z = Camada intermediária da rede.

$outZ$ = Saída da camada Z (de acordo com a função de ativação).

$inpZ$ = Entrada da camada Z (amostras de entrada).

$\hat{y}$ = Ground true

De forma geral temos a seguinte derivação para a retropopagação do erro para qualquer v_n.

$$\frac{\partial J}{\partial v_n} = \frac{\partial J}{\partial outZ}\frac{\partial outZ}{\partial inpZ}\frac{\partial inpZ}{\partial v_n}$$

No caso específico para v_{12} temos:

$$\frac{\partial J}{\partial v_{12}} = \frac{\partial J}{\partial outZ}\frac{\partial outZ}{\partial inpZ}\frac{\partial inpZ}{\partial v_{12}}$$

Realizando as derivadas expostas acima:

$$\frac{\partial J}{\partial outZ} = \sum_{n=1}^{N}(\hat{y} - y)w_n$$

$$\frac{\partial outZ}{\partial inpZ} = f(.)$$

$$\frac{\partial inpZ}{\partial v_n} = x_n$$

Então para v_{12}:

$$\frac{\partial J}{\partial outZ} = (\hat{y}_1 - y_1)w_{30} + (\hat{y}_2 - y_2)w_{31}$$

$$\frac{\partial outZ}{\partial inpZ} = f(.)$$

$$\frac{\partial inpZ}{\partial v_1 2} = x_1$$

Finalmente:

$$\frac{\partial J}{\partial v_{12}} = ((\hat{y}_1 - y_1)w_{30} + (\hat{y}_2 - y_2)w_{31}) \times f(.) \times x_1$$

Utilizando MLP, testou-se dois métodos de estimação: batch e online, dentre eles, pode-se observar que a melhor acurácia e também, convergiu mais rapidamente, em comparação ao batch, ocorreu quando usou-se o método de estimação batch, com as configurações:

- Épocas = 200.
- Camada oculta com 50 neurônios, com função de ativação ReLU.
- Entropia cruzada para a função custo.
- Os parâmetros foram calculadas utilizando o método Adam.
  Onde observou-se que o melhor resultado foi 86% de acurácia nos testes, utilizando a validação cruzada nos testes de validação, foram testados os valores 5, 10, 15, 30, 50 para a camada oculta, a que apresentou o melhor resultado foi a rede com 50 neurônios, resultado esse, pouco melhor do que quando utilizado o valor de 30 neurônios para a camada oculta, o resultado pode ser visto na figura 1.

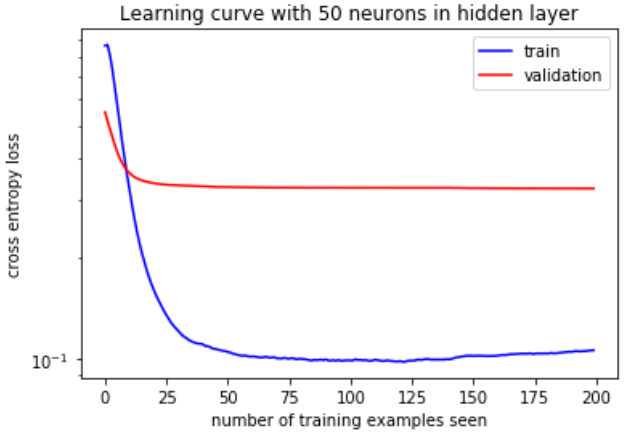


Figura 1: Curva de aprendizado.

Na figura 3, é possível analisar melhor as regiões de decisão e as classes de cada amostra, bastante parecida com a figura mostrada no enunciado utilizando o estimador MAP

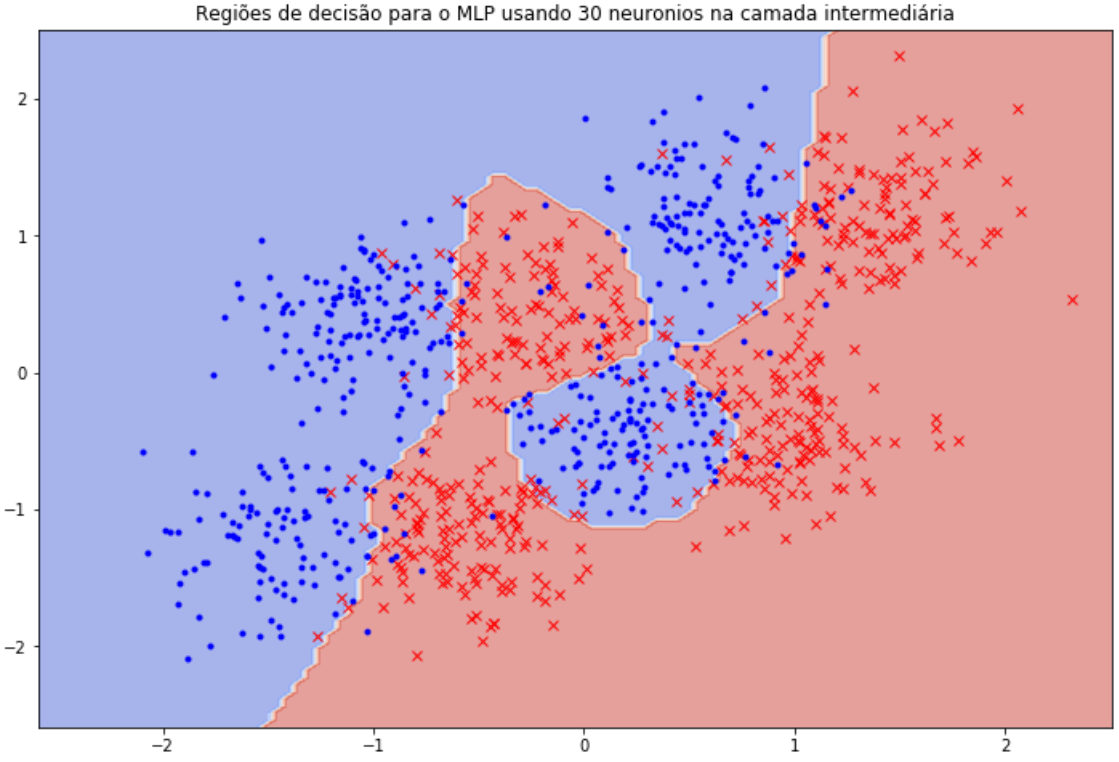


Figura 2: Regiões de decisão e classes

SVM - foi utilizada a biblioteca sklearn.svm para as máquinas de vetores de suporte, os hiperparâmetros foram escolhidos com validação

cruzada, igual feito no MLP. O melhor resultado obtido com nos testes foi com o kernel RBF e taxa de penalidade do erro = 50, a melhor acurácia foi de 0.867, o gráfico plotado pode ser visto na figura 3

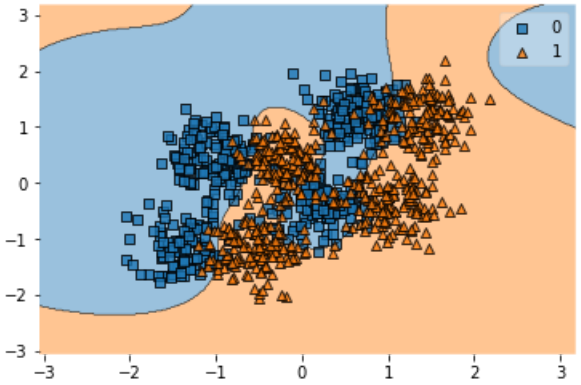


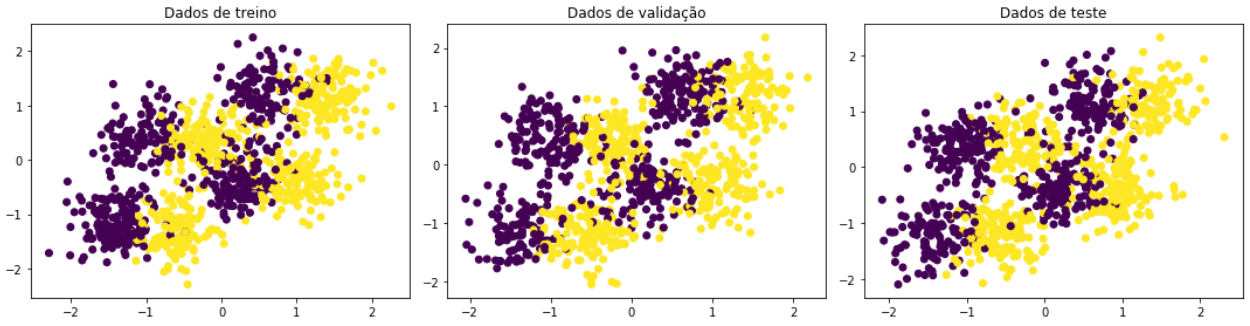
Figura 3: SVM com kernel rbf e penalização do erro =50

Para penalização (C) utilizou-se 1, 10, 50, 100 Os kernels testados foram 'linear', 'poly', 'rbf', 'sigmoid' No código, pode-se observar os resultados quando utilizado kernel linear, porém, o modelo não é capaz de classificar satisfatoriamente os dados

```
Using TensorFlow backend.
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:51
6: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:51
7: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:51
8: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:51
9: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:52
0: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorflow/python/framework/dtypes.py:52
5: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in a futu
re version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:541: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint8 = np.dtype([("qint8", np.int8, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:542: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint8 = np.dtype([("quint8", np.uint8, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:543: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint16 = np.dtype([("qint16", np.int16, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:544: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_quint16 = np.dtype([("quint16", np.uint16, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:545: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  _np_qint32 = np.dtype([("qint32", np.int32, 1)])
/home/jimitogni/.local/lib/python3.7/site-packages/tensorboard/compat/tensorflow_stub/dtype
s.py:550: FutureWarning: Passing (type, 1) or '1type' as a synonym of type is deprecated; in
a future version of numpy, it will be understood as (type, (1,)) / '(1,)type'.
  np_resource = np.dtype([("resource", np.ubyte, 1)])
```

2. Parte II – Classificação binária com redes MLP e SVMs

```
Visualizando os dados de treinamento, teste e validação
```



2.1 - Aplicando a MLP

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size=28, beta_1=0.9,
              beta_2=0.999, early_stopping=False, epsilon=1e-08,
              hidden_layer_sizes=(100,), learning_rate='constant',
              learning_rate_init=0.0001, max_iter=1000, momentum=0.9,
              n_iter_no_change=10, nesterovs_momentum=True, power_t=0.5,
              random_state=1, shuffle=True, solver='adam', tol=0.0001,
              validation_fraction=0.1, verbose=False, warm_start=False)
```

2.1.1 - MLP - Teste 1:

Hiperparametros Utilizados:
'activation': 'relu',
'hidden_layer_sizes': (100,),
'learning_rate': 'constant',
'learning_rate_init': 1e-4,
'batch_size': 28,
'max_iter': 1000,
'solver': 'adam',
'random_state': 1

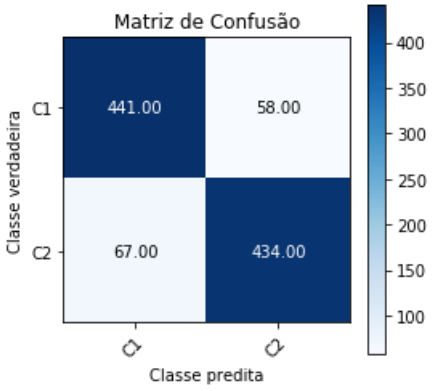
Utilizando a classe MLPClassifier da biblioteca sklearn

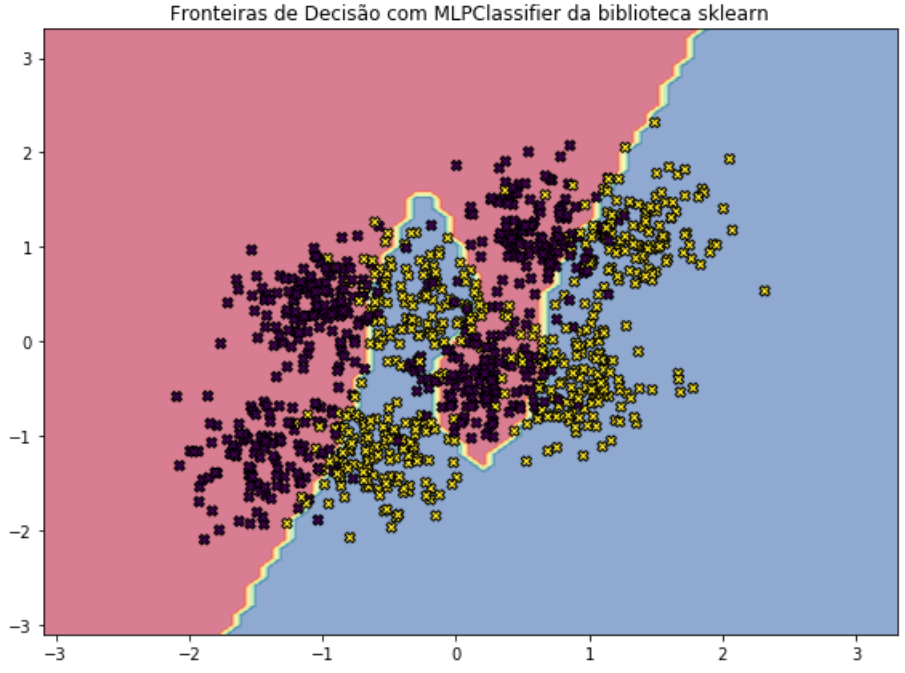
```
_____________________________________________________________
--- Alguns valores importantes a se destacar ---
_____________________________________________________________
Neurônios nas camadas ocultas: 100
Funçã de ativação: ReLu
Tamanho do batch: 28
Solver: Adam
Passo de aprendizado: 1e-4
Iterações máximas: 1000
_____________________________________________________________
```

2.1.1 - MLP - Teste 1

Melhor resultado

```
_____________________________________________________________
Acurácia:
87.5%
_____________________________________________________________
Relatório da classificação:

              precision    recall  f1-score   support

          C1       0.87      0.88      0.88       499
          C2       0.88      0.87      0.87       501

    accuracy                           0.88      1000
   macro avg       0.88      0.88      0.87      1000
weighted avg       0.88      0.88      0.87      1000

_____________________________________________________________
(1.5, -0.5)
```





2.1.2 - MLP - Teste 2

Principais configurações do teste 2

'hidden_layer_sizes': [(50,50,50), (50,100,50), (100,)],
'activation': ['tanh', 'relu'],
'solver': ['sgd', 'adam'],
'alpha': [0.0001, 0.05],
'learning_rate': ['constant','adaptive'],

Ainda utilizando MLPClassifier, porém com uma variação maior dos hiperparametros, que são:

Ainda utilizando MLPClassifier, porém com uma variação maior dos hiperparametros, que são:

```
GridSearchCV(cv=3, error_score='raise-deprecating',
             estimator=MLPClassifier(activation='relu', alpha=0.0001,
                                     batch_size='auto', beta_1=0.9,
                                     beta_2=0.999, early_stopping=False,
                                     epsilon=1e-08, hidden_layer_sizes=(100,),
                                     learning_rate='constant',
                                     learning_rate_init=0.001, max_iter=500,
                                     momentum=0.9, n_iter_no_change=10,
                                     nesterovs_momentum=True, power_t=0.5,
                                     random_sta...
                                     solver='adam', tol=0.0001,
                                     validation_fraction=0.1, verbose=False,
                                     warm_start=False),
             iid='warn', n_jobs=-1,
             param_grid={'activation': ['tanh', 'relu'],
                         'alpha': [0.0001, 0.05],
                         'hidden_layer_sizes': [(50, 50, 50), (50, 100, 50),
                                                (100,)],
                         'learning_rate': ['constant', 'adaptive'],
                         'solver': ['sgd', 'adam']},
             pre_dispatch='2*n_jobs', refit=True, return_train_score=False,
             scoring=None, verbose=0)
```

2.1.2.1 - MLP - Teste 2

Resultados gerais:

---------- Alguns resultados obtidos ------------

0.671 (+/-0.026) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.871 (+/-0.039) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.670 (+/-0.029) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.875 (+/-0.054) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.666 (+/-0.039) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.880 (+/-0.044) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.667 (+/-0.027) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.875 (+/-0.051) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.653 (+/-0.036) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.664 (+/-0.033) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.651 (+/-0.027) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.668 (+/-0.036) para -> {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.666 (+/-0.036) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.874 (+/-0.030) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.666 (+/-0.032) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.862 (+/-0.017) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.665 (+/-0.041) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.872 (+/-0.019) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.666 (+/-0.031) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.873 (+/-0.046) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.652 (+/-0.036) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.668 (+/-0.026) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.649 (+/-0.031) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

```
0.669 (+/-0.024) para -> {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_sizes': (100,),
'learning_rate': 'adaptive', 'solver': 'adam'}

0.707 (+/-0.032) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.876 (+/-0.030) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.729 (+/-0.064) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.878 (+/-0.032) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.712 (+/-0.014) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.879 (+/-0.039) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.710 (+/-0.011) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.874 (+/-0.030) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50,
100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.666 (+/-0.039) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10
0,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.866 (+/-0.053) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10
0,), 'learning_rate': 'constant', 'solver': 'adam'}

0.670 (+/-0.025) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10
0,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.868 (+/-0.043) para -> {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (10
0,), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.714 (+/-0.015) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 5
0, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.877 (+/-0.029) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 5
0, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.710 (+/-0.018) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 5
0, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.873 (+/-0.038) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 5
0, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.721 (+/-0.022) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 10
0, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.879 (+/-0.041) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 10
0, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.717 (+/-0.047) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 10
0, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.873 (+/-0.039) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 10
0, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.663 (+/-0.041) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,),
'learning_rate': 'constant', 'solver': 'sgd'}

0.867 (+/-0.051) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,),
'learning_rate': 'constant', 'solver': 'adam'}

0.667 (+/-0.025) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,),
'learning_rate': 'adaptive', 'solver': 'sgd'}
```

```
0.867 (+/-0.046) para -> {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (100,),
'learning_rate': 'adaptive', 'solver': 'adam'}
```

2.1.2.2 - MLP - Teste 2

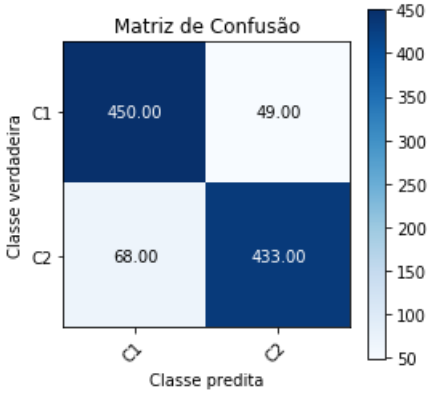
Melhor resultado

```
Hiperparametros:

 {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_sizes': (50, 50, 50), 'learning_rat
e': 'constant', 'solver': 'adam'}
```

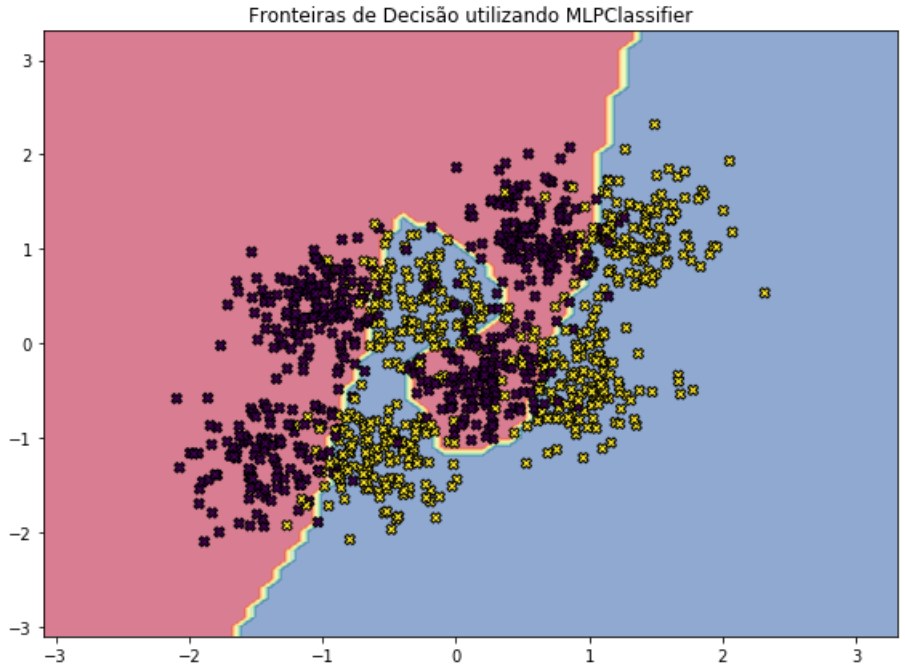
```
Resultados:
```

```
              precision    recall  f1-score   support

          C1       0.87      0.90      0.88       499
          C2       0.90      0.86      0.88       501

    accuracy                           0.88      1000
   macro avg       0.88      0.88      0.88      1000
weighted avg       0.88      0.88      0.88      1000
```

```
(1.5, -0.5)
```



Fronteiras de decição



2.2 - Testes utilizando Keras

2.2.1 - Teste 3:

- 2 camadas densamente conectadas:
- 1ª com 100 neuronios, função de ativação relu
- 2ª com 2 neuronios de saida, função de classificação softmax
- Otimizador Adam com passo de aprendizado 1e-3
- Loss: Entropia cruzada
- Métrica: Acurácia
- epochs: 300
- batch_size: 28

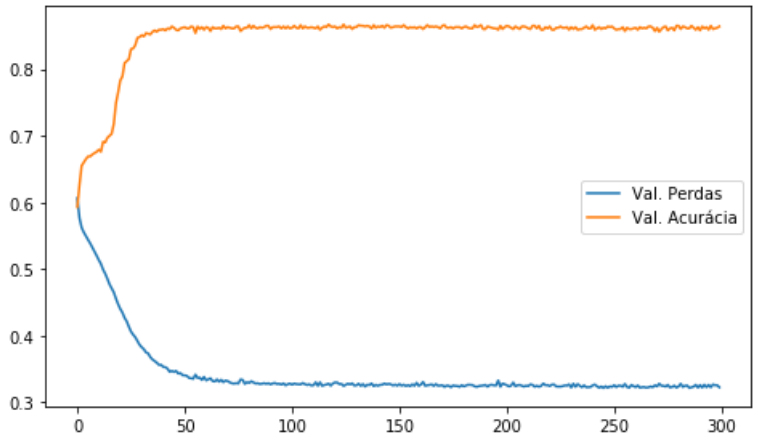
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Camada_de_entrada (Dense)    (None, 100)               300
_________________________________________________________________
Camada_de_saida (Dense)      (None, 2)                 202
=================================================================
Total params: 502
Trainable params: 502
Non-trainable params: 0
_________________________________________________________________
None

_________________________________________________________________

Alguns hiperparametros utilizados para otimização:

learning_rate: 0.001
beta_1: 0.9
beta_2: 0.999
decay: 0.0
epsilon: 0.0
amsgrad: False
```

```
WARNING:tensorflow:From /home/jimitogni/.local/lib/python3.7/site-packages/keras/backend/ten
sorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.
global_variables instead.
```

2.2.1 - Teste 3

Resultados

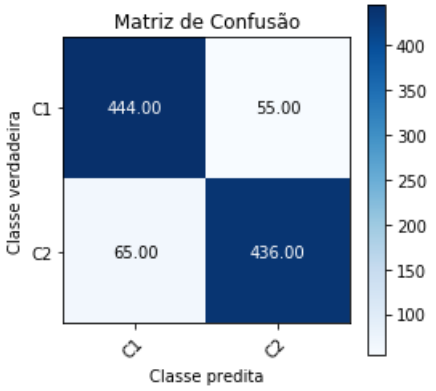
No caso do framework keras existe a possibilidade de avaliação do conjunto de validação conforme o andamento do aprendizado da Rede. Os gráficos abaixo apresentam a Loss / Accuracy para os dados de treinamento e validação ao longos das épocas. A escolha pelo softmax fora apenas para exercitar uma Rede Neural multiclasse padrão, no caso a camada de saída poderia ter como ativação a Função Logística (sigmoid).



2.2.1 - Teste 3

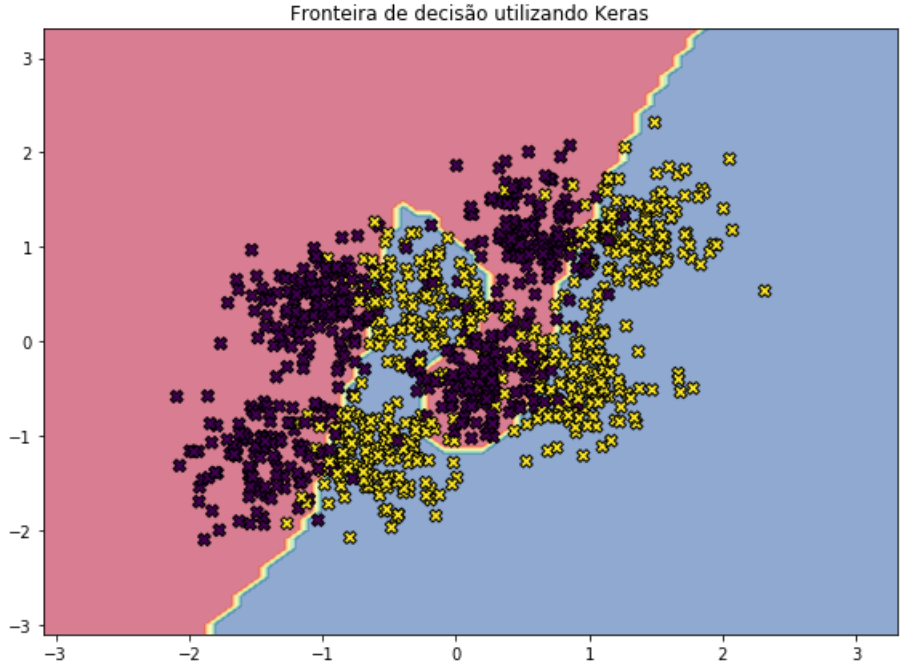
Resultados do modelo nos dados de testes.

```
_____________________________________________________________
Acurácia:
88.0%

_____________________________________________________________
Relatório da classificação:

              precision    recall  f1-score   support

          C1       0.87      0.89      0.88       499
          C2       0.89      0.87      0.88       501

    accuracy                           0.88      1000
   macro avg       0.88      0.88      0.88      1000
weighted avg       0.88      0.88      0.88      1000

(1.5, -0.5)
```



2.2.1 - Teste 3

Fronteiras de decisão



____

"Como é possível notar a Rede Neural construída via Keras teve uma pequena melhor performance, provavelmente devido ao fato menor hiperparamentrização (a rede do scikit-learn pré-configura diversos outros parâmetros como por exemplo regularização). Também pode ser visto que a Rede via Keras converge mais rápido (menos épocas).

Será portanto utilizada a Rede do Keras para experimentar o uso de mais unidades (neurônios na camada intermediária). Para teste (e pensando na questão de uma Rede Neural ser um Aproximador Universal), aumentou-se de maneira relativamente expressiva a quantidade de unidades da camada intermediária para 32768 ao invés de 100."

2.2.2 - Teste 4 - utilizando Keras com 30.000 neuônios

Para o teste 4, utilizou-se 30.000 neurônios na camada de entrada, numero este, escolhido de acordo com a documentação do framework Keras, sendo citado como um valor extramamente alto, para que possamos, posteriormente, fazer um comparação entre valores extremamente grande para neuronios e, outrora, valores minimos para os neurônios

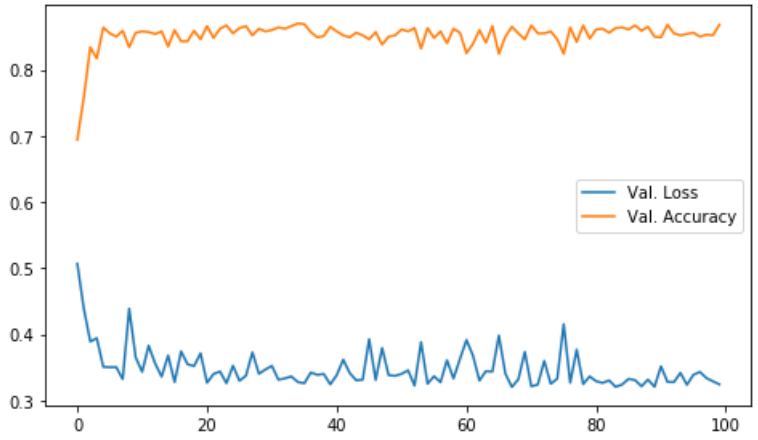
```
_________________________________________________________________
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Camada_Entrada (Dense)       (None, 30000)             90000
_________________________________________________________________
Camada_Saida (Dense)         (None, 2)                 60002
=================================================================
Total params: 150,002
Trainable params: 150,002
Non-trainable params: 0
_________________________________________________________________
None

_________________________________________________________________
Alguns hiperparametros utilizados para otimização:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

2.2.2 - Teste 4

Resultados obtidos em relação ao conjunto de validação

Aluns hiperparametros utilizados

epochs=100, batch_size=32, shuffle=True, verbose=False,

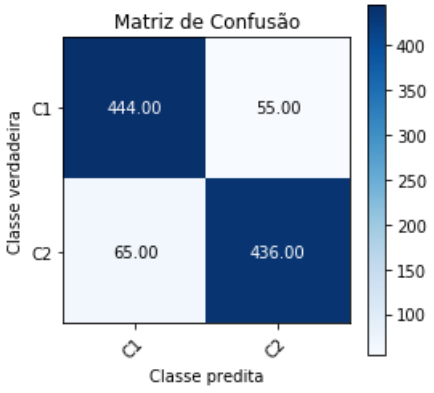


```
Acurácia:
88.0%
```

```
Relatório da classificação:
```

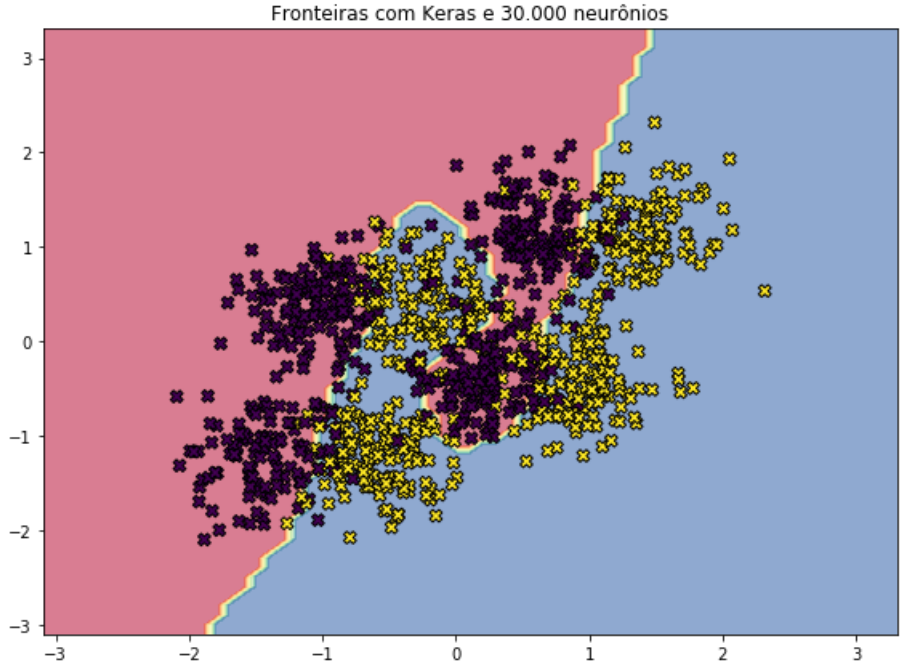
	precision	recall	f1-score	support
C1	0.87	0.89	0.88	499
C2	0.89	0.87	0.88	501
accuracy			0.88	1000
macro avg	0.88	0.88	0.88	1000
weighted avg	0.88	0.88	0.88	1000

```
(1.5, -0.5)
```



2.2.2 - Teste 4

Fronteiras de decisão



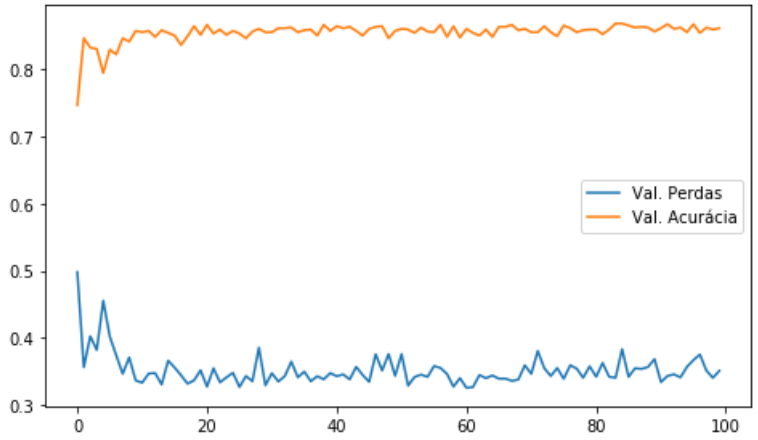
2.2.3 - Teste 5 - Mais camadas

Para este teste, utilizaremos 5 camadas intermediárias, com função de ativação ReLu e softmax para classificação

```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer_1 (Dense)        (None, 1024)              3072
_________________________________________________________________
Input_Layer_2 (Dense)        (None, 1024)              1049600
_________________________________________________________________
Input_Layer_3 (Dense)        (None, 1024)              1049600
_________________________________________________________________
Input_Layer_4 (Dense)        (None, 1024)              1049600
_________________________________________________________________
Input_Layer_5 (Dense)        (None, 1024)              1049600
_________________________________________________________________
Output_Layer (Dense)         (None, 2)                 2050
=================================================================
Total params: 4,203,522
Trainable params: 4,203,522
Non-trainable params: 0
_________________________________________________________________
None


_____________________________________________________________
Alguns hiperparametros utilizados para otimização:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

2.2.3 - Teste 5

Resultados obtidos

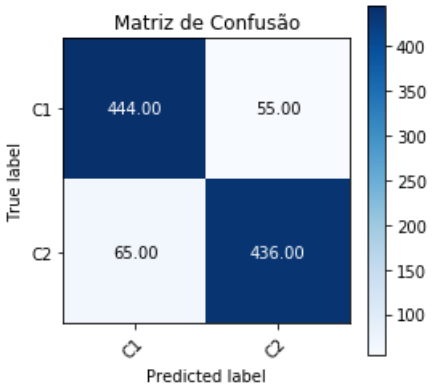


```
Acurácia:
88.0%
```

```
Relatório da classificação:
```

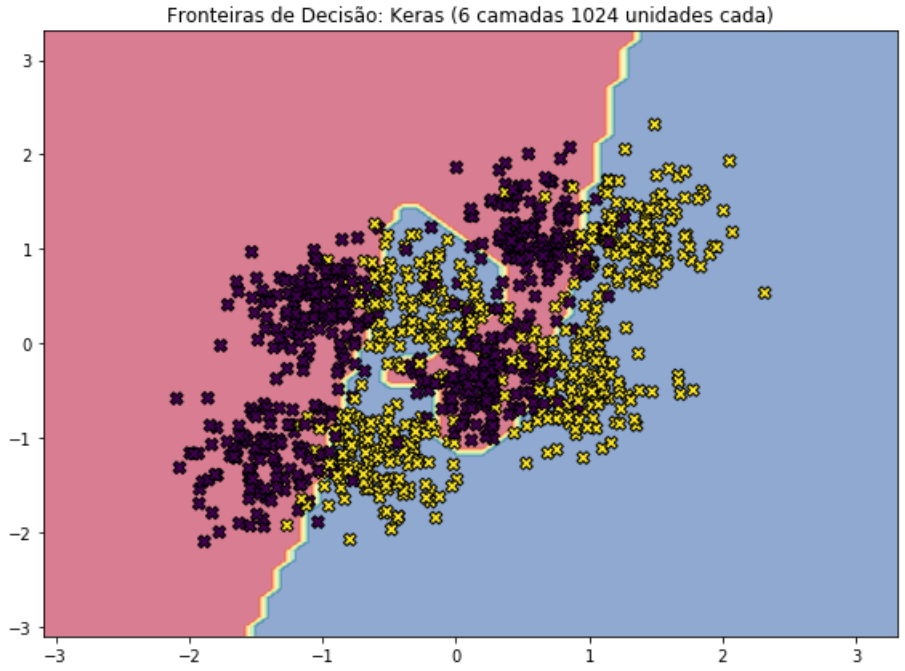
	precision	recall	f1-score	support
C1	0.87	0.89	0.88	499
C2	0.89	0.87	0.88	501
accuracy			0.88	1000
macro avg	0.88	0.88	0.88	1000
weighted avg	0.88	0.88	0.88	1000

```
(1.5, -0.5)
```



2.2.3 - Teste 5

Fronteiras de decisão



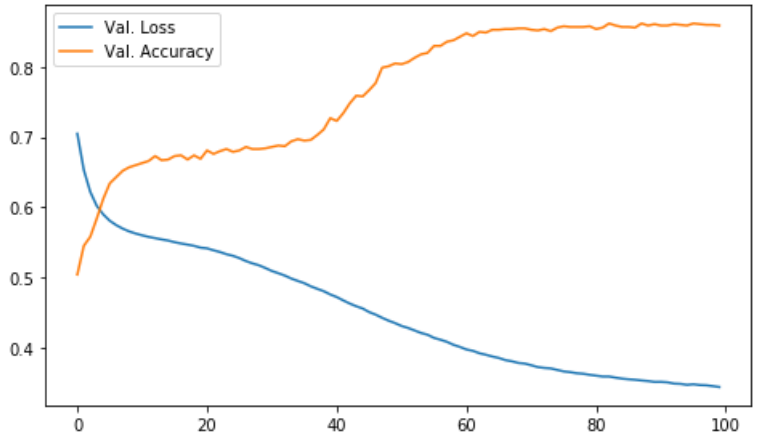
2.2.4 - Teste 6

Minima quantidade de neuronios

```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer_1 (Dense)        (None, 30)                90
_________________________________________________________________
Output_Layer (Dense)         (None, 2)                 62
=================================================================
Total params: 152
Trainable params: 152
Non-trainable params: 0
_________________________________________________________________
None

_____________________________________________________________
Alguns hiperparametros utilizados para otimização:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

2.2.4 - Teste 6

Resulados:

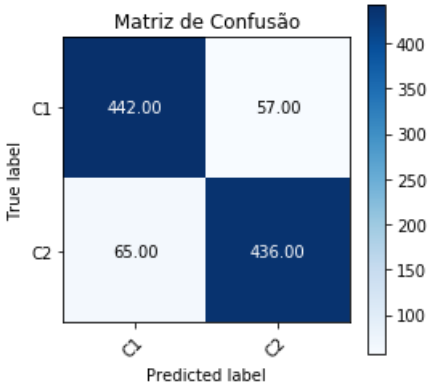


```
Acurácia:
87.8%
```

```
Relatório da classificação:
```

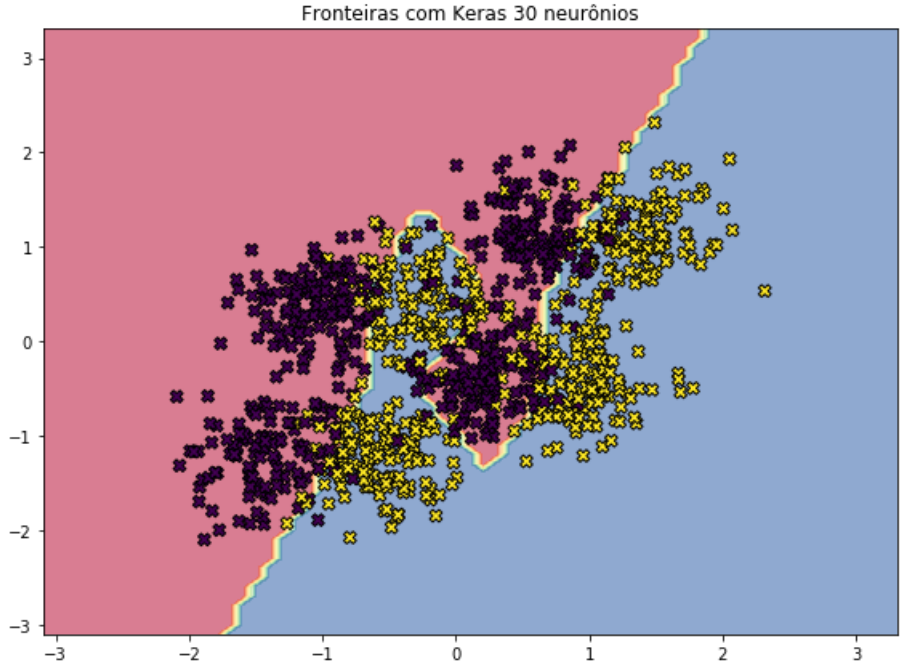
	precision	recall	f1-score	support
C1	0.87	0.89	0.88	499
C2	0.88	0.87	0.88	501
accuracy			0.88	1000
macro avg	0.88	0.88	0.88	1000
weighted avg	0.88	0.88	0.88	1000

```
(1.5, -0.5)
```



2.2.4 - Teste 6

Fronteiras de decisão:



2.3 - SVM

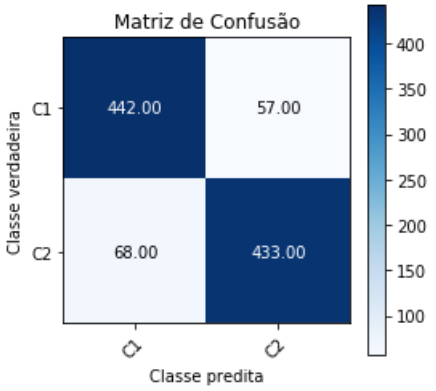
2.3.1 - SVM Teste 1

Alguns hiperparametros utilizados

'C': 5,
'gamma': 'scale',
'kernel': 'rbf',
'random_state': 1

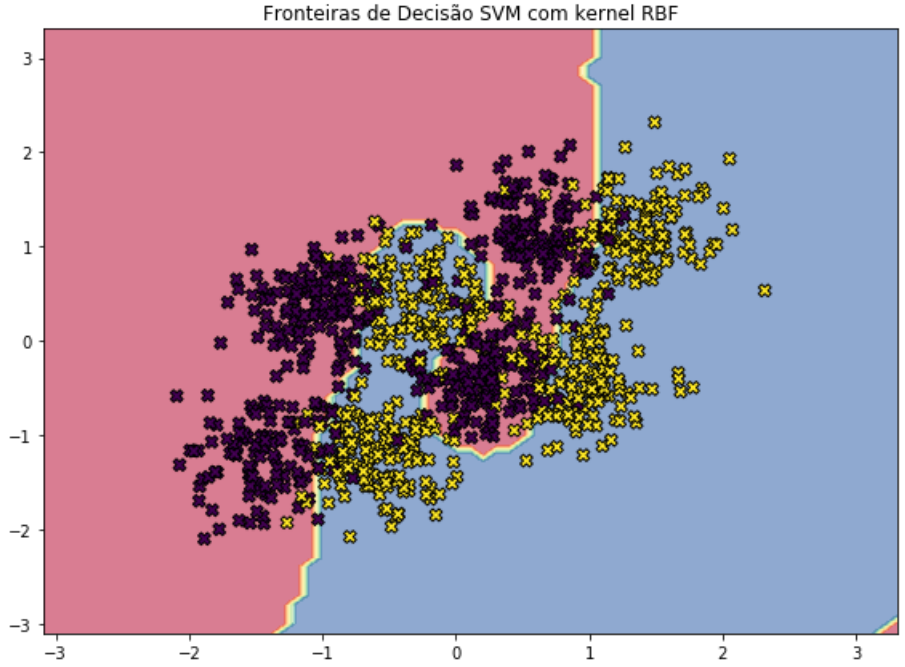
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
_____________________________________________________________

Acurácia:
87.5%
_____________________________________________________________
Relatório da classificação:

              precision    recall  f1-score   support

          C1       0.87      0.89      0.88       499
          C2       0.88      0.86      0.87       501

    accuracy                           0.88      1000
   macro avg       0.88      0.88      0.87      1000
weighted avg       0.88      0.88      0.87      1000

(1.5, -0.5)
```



2.3.1 - SVM Teste 1

Fronteiras de decisão



2.3.2 - SVM - Teste 2

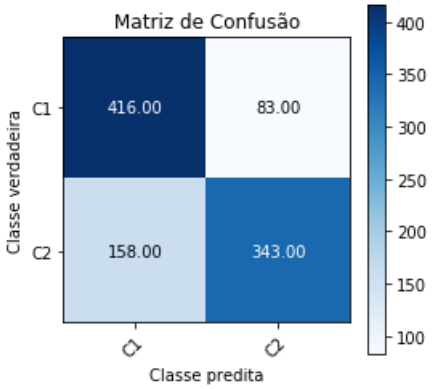
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
Acurácia:
75.9%
```

Relatório da classificação:

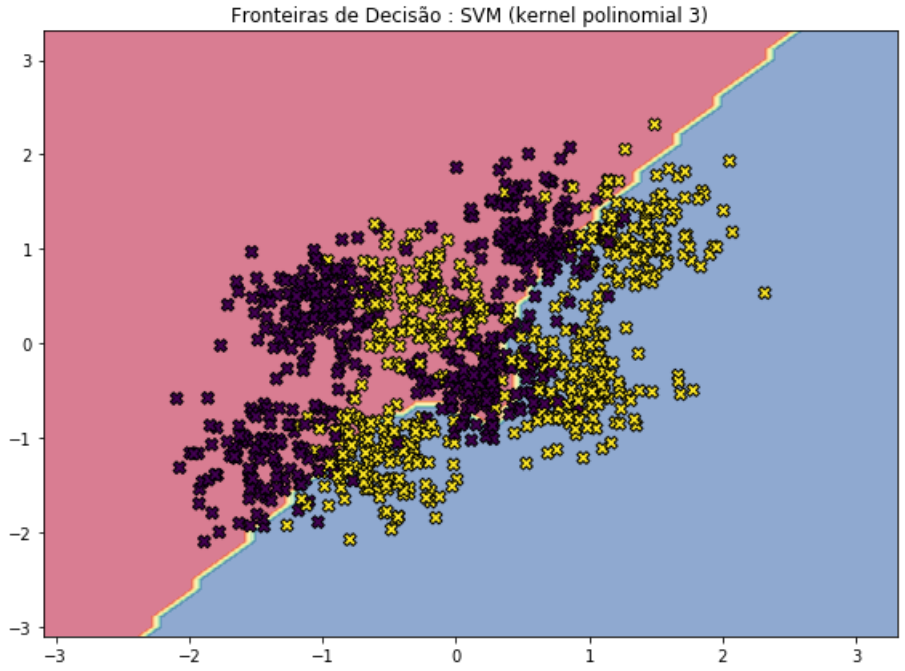
	precision	recall	f1-score	support
C1	0.72	0.83	0.78	499
C2	0.81	0.68	0.74	501
accuracy			0.76	1000
macro avg	0.76	0.76	0.76	1000
weighted avg	0.77	0.76	0.76	1000

```
(1.5, -0.5)
```



2.3.2 - SVM Teste 2

Fronteiras de decisão



2.3.3 - SVM - Teste 3

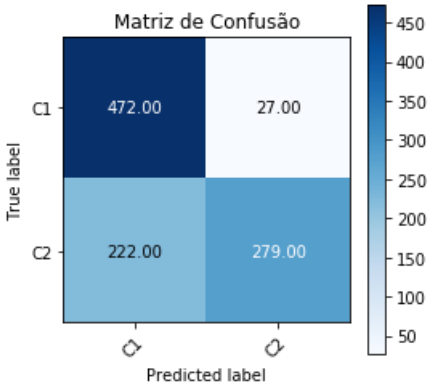
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=9, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
Acurácia:
75.1%
```

Relatório da classificação:

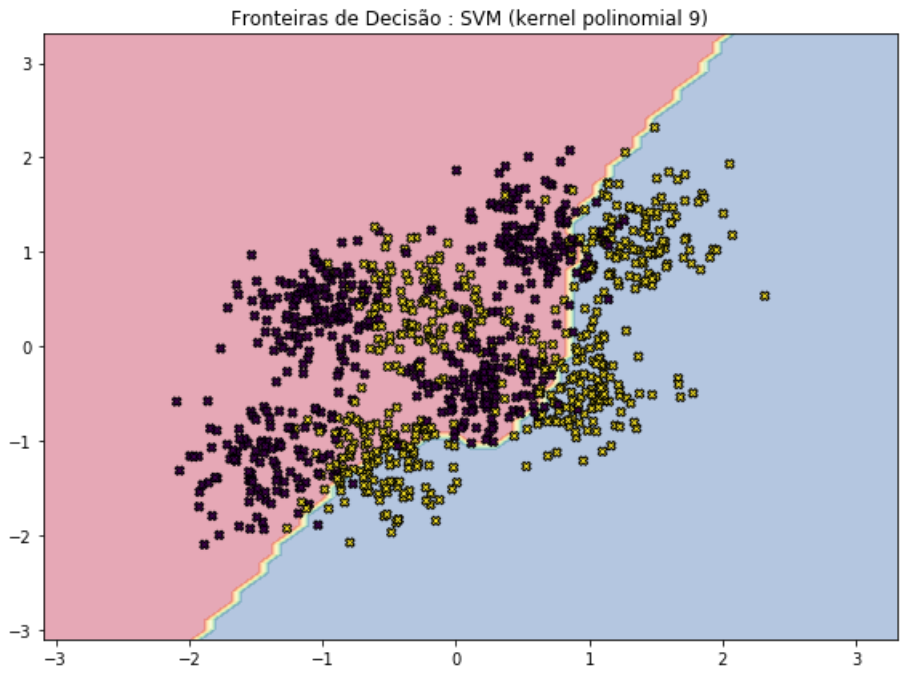
	precision	recall	f1-score	support
C1	0.68	0.95	0.79	499
C2	0.91	0.56	0.69	501
accuracy			0.75	1000
macro avg	0.80	0.75	0.74	1000
weighted avg	0.80	0.75	0.74	1000

```
(1.5, -0.5)
```



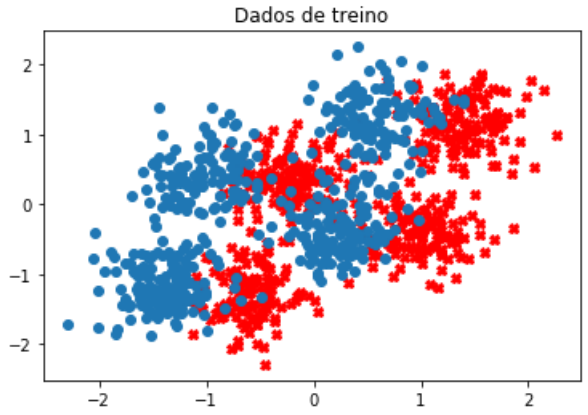
2.3.3 - SVM Teste 3

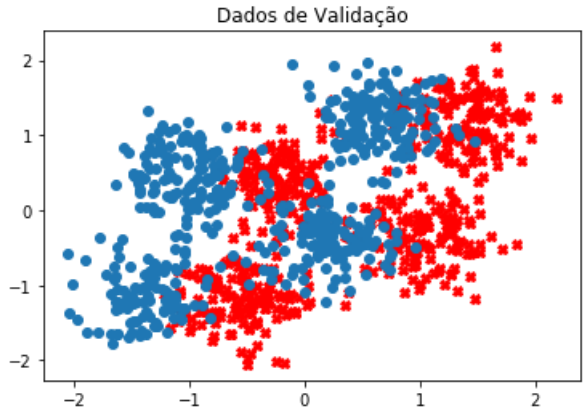
Fronteiras de decisão



Semestre passado

2.4 - MLP Utilizando torch para a criação dos modelos e testes com mini-batch e online






```
---------------------------------------------------------------------------
IndexError                                Traceback (most recent call last)
<ipython-input-66-9b4a6b9214be> in <module>
      9 plt.show()
     10
---> 11 plt.plot(X_teste[np.in1d(y_teste, 1), 0], y_teste[np.in1d(y_teste, 1), 1], 'rX')
     12 plt.plot(X_teste[np.in1d(y_teste, 0), 0], y_teste[np.in1d(y_teste, 0), 1], 'o')
     13 plt.title("Dados de Teste")

IndexError: index 1 is out of bounds for axis 1 with size 1
```

2.4 - MLP Utilizando torch e mini-batch

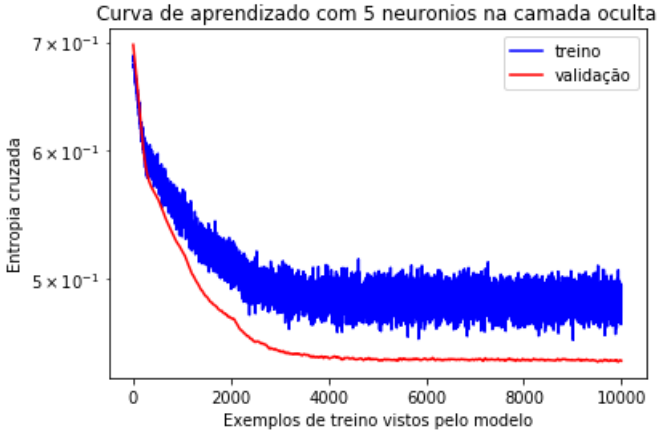
Camada oculta variando a quantidade de neurônios em: 5, 10, 15, 30, 50

Hiperparametros:

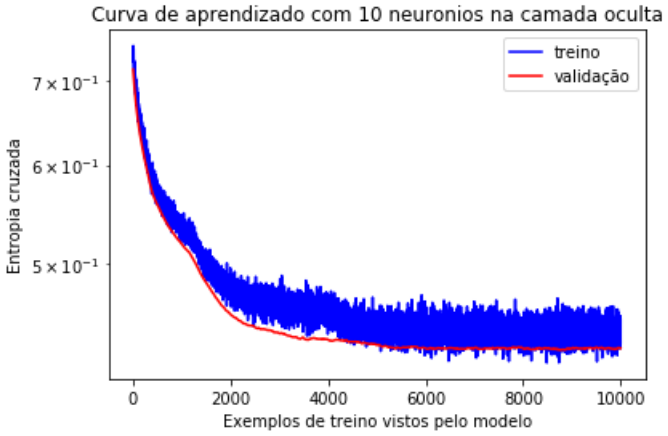
- classes = 1
- dim_entrada = 2
- epocas = 10000
- passagens = 1000
- otimizador = Adam
- função de ativação = sigmoid, relu
- dropout = 0,5

Resultados:

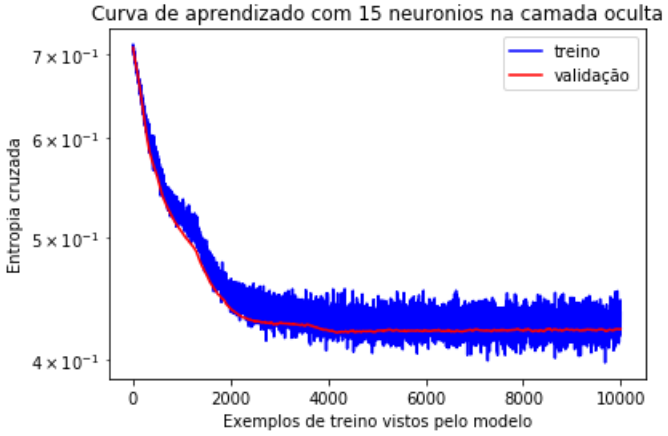
```
H =  5
Dados de validação: Avg. loss: 0.6974, Acurácia: 481/1000 (48%)
Dados de validação: Avg. loss: 0.5203, Acurácia: 684/1000 (68%)
Dados de validação: Avg. loss: 0.4731, Acurácia: 704/1000 (70%)
Dados de validação: Avg. loss: 0.4513, Acurácia: 747/1000 (75%)
Dados de validação: Avg. loss: 0.4469, Acurácia: 751/1000 (75%)
Dados de validação: Avg. loss: 0.4461, Acurácia: 746/1000 (75%)
Dados de validação: Avg. loss: 0.4463, Acurácia: 745/1000 (74%)
Dados de validação: Avg. loss: 0.4459, Acurácia: 746/1000 (75%)
Dados de validação: Avg. loss: 0.4458, Acurácia: 744/1000 (74%)
Dados de validação: Avg. loss: 0.4457, Acurácia: 746/1000 (75%)
```



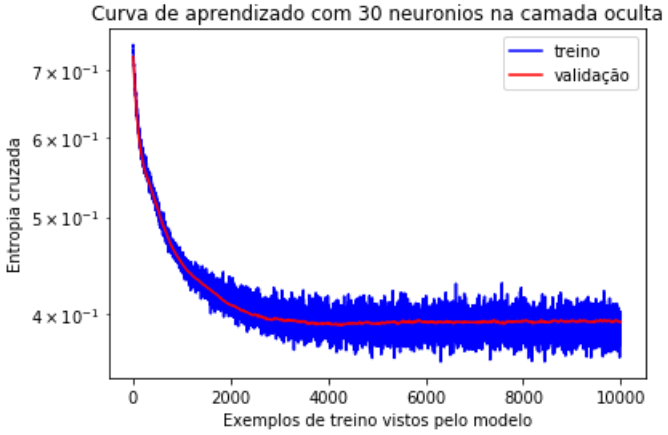

```
H =  10
Dados de validação: Avg. loss: 0.7147, Acurácia: 438/1000 (44%)
Dados de validação: Avg. loss: 0.5188, Acurácia: 676/1000 (68%)
Dados de validação: Avg. loss: 0.4560, Acurácia: 745/1000 (74%)
Dados de validação: Avg. loss: 0.4404, Acurácia: 753/1000 (75%)
Dados de validação: Avg. loss: 0.4353, Acurácia: 768/1000 (77%)
Dados de validação: Avg. loss: 0.4310, Acurácia: 771/1000 (77%)
Dados de validação: Avg. loss: 0.4285, Acurácia: 792/1000 (79%)
Dados de validação: Avg. loss: 0.4289, Acurácia: 786/1000 (79%)
Dados de validação: Avg. loss: 0.4284, Acurácia: 790/1000 (79%)
Dados de validação: Avg. loss: 0.4286, Acurácia: 788/1000 (79%)
```



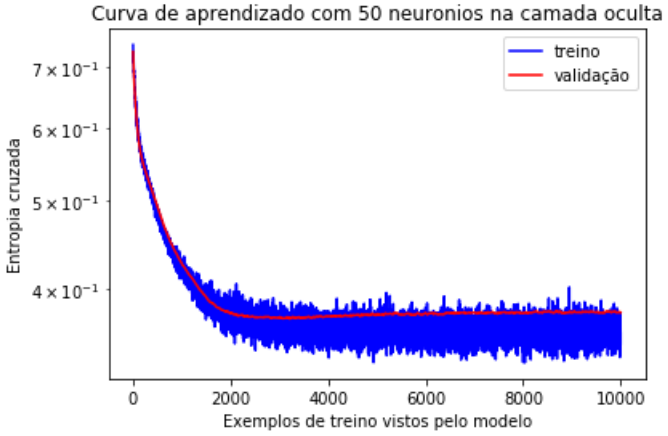

```
H =  15
Dados de validação: Avg. loss: 0.7078, Acurácia: 382/1000 (38%)
Dados de validação: Avg. loss: 0.5052, Acurácia: 687/1000 (69%)
Dados de validação: Avg. loss: 0.4392, Acurácia: 723/1000 (72%)
Dados de validação: Avg. loss: 0.4277, Acurácia: 765/1000 (76%)
Dados de validação: Avg. loss: 0.4223, Acurácia: 800/1000 (80%)
Dados de validação: Avg. loss: 0.4217, Acurácia: 814/1000 (81%)
Dados de validação: Avg. loss: 0.4218, Acurácia: 809/1000 (81%)
Dados de validação: Avg. loss: 0.4223, Acurácia: 806/1000 (81%)
Dados de validação: Avg. loss: 0.4217, Acurácia: 813/1000 (81%)
Dados de validação: Avg. loss: 0.4226, Acurácia: 807/1000 (81%)
```




```
H =  30
Dados de validação: Avg. loss: 0.7209, Acurácia: 421/1000 (42%)
Dados de validação: Avg. loss: 0.4497, Acurácia: 736/1000 (74%)
Dados de validação: Avg. loss: 0.4087, Acurácia: 832/1000 (83%)
Dados de validação: Avg. loss: 0.3957, Acurácia: 847/1000 (85%)
Dados de validação: Avg. loss: 0.3925, Acurácia: 852/1000 (85%)
Dados de validação: Avg. loss: 0.3916, Acurácia: 849/1000 (85%)
Dados de validação: Avg. loss: 0.3935, Acurácia: 850/1000 (85%)
Dados de validação: Avg. loss: 0.3929, Acurácia: 849/1000 (85%)
Dados de validação: Avg. loss: 0.3940, Acurácia: 848/1000 (85%)
Dados de validação: Avg. loss: 0.3945, Acurácia: 849/1000 (85%)
```

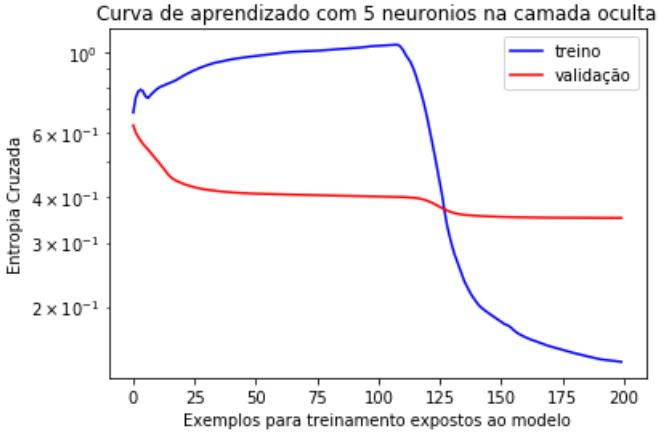



```
H =  50
Dados de validação: Avg. loss: 0.7267, Acurácia: 368/1000 (37%)
Dados de validação: Avg. loss: 0.4267, Acurácia: 803/1000 (80%)
Dados de validação: Avg. loss: 0.3758, Acurácia: 858/1000 (86%)
Dados de validação: Avg. loss: 0.3729, Acurácia: 864/1000 (86%)
Dados de validação: Avg. loss: 0.3738, Acurácia: 859/1000 (86%)
Dados de validação: Avg. loss: 0.3755, Acurácia: 859/1000 (86%)
Dados de validação: Avg. loss: 0.3768, Acurácia: 858/1000 (86%)
Dados de validação: Avg. loss: 0.3772, Acurácia: 859/1000 (86%)
Dados de validação: Avg. loss: 0.3764, Acurácia: 860/1000 (86%)
Dados de validação: Avg. loss: 0.3777, Acurácia: 859/1000 (86%)
```

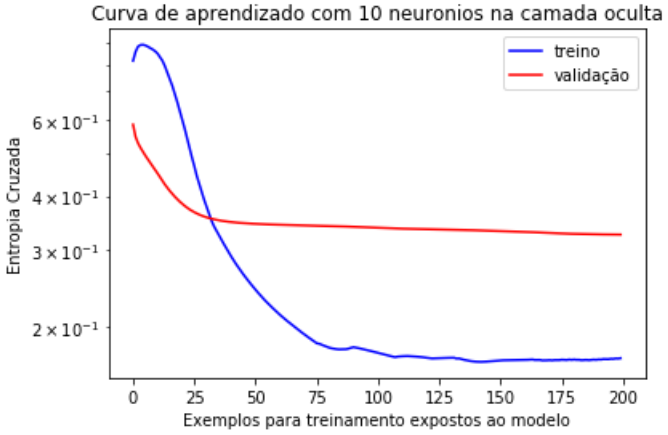

Curva de aprendizado com 50 neuronios na camada oculta
treino
validação
Entropia cruzada
7×10^{-1}
6×10^{-1}
5×10^{-1}
4×10^{-1}
0
2000
4000
6000
8000
10000
Exemplos de treino vistos pelo modelo

Online

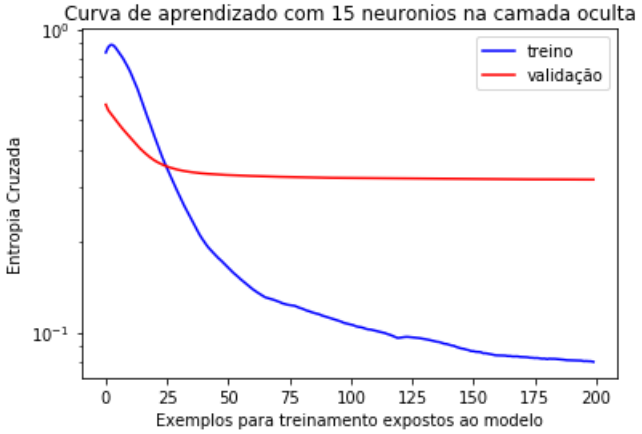
```
H =  5

/home/jimitogni/.local/lib/python3.7/site-packages/torch/nn/_reduction.py:43: UserWarning: s
ize_average and reduce args will be deprecated, please use reduction='sum' instead.
  warnings.warn(warning.format(ret))

Validação: Avg. loss: 0.6305, Acurácia: 622/1000 (62%)
Validação: Avg. loss: 0.4366, Acurácia: 772/1000 (77%)
Validação: Avg. loss: 0.4127, Acurácia: 799/1000 (80%)
Validação: Avg. loss: 0.4077, Acurácia: 799/1000 (80%)
Validação: Avg. loss: 0.4048, Acurácia: 801/1000 (80%)
Validação: Avg. loss: 0.4021, Acurácia: 807/1000 (81%)
Validação: Avg. loss: 0.3912, Acurácia: 821/1000 (82%)
Validação: Avg. loss: 0.3558, Acurácia: 848/1000 (85%)
Validação: Avg. loss: 0.3523, Acurácia: 843/1000 (84%)
Validação: Avg. loss: 0.3513, Acurácia: 847/1000 (85%)
```



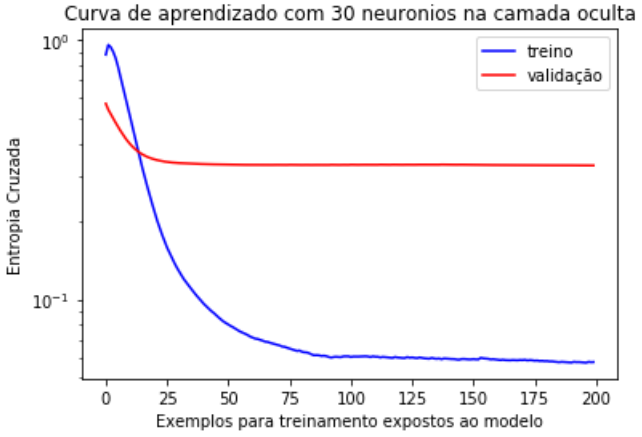

```
H =  10
Validação: Avg. loss: 0.5836, Acurácia: 588/1000 (59%)
Validação: Avg. loss: 0.3839, Acurácia: 831/1000 (83%)
Validação: Avg. loss: 0.3484, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3434, Acurácia: 858/1000 (86%)
Validação: Avg. loss: 0.3410, Acurácia: 857/1000 (86%)
Validação: Avg. loss: 0.3377, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3353, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3335, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3308, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3272, Acurácia: 865/1000 (86%)
```



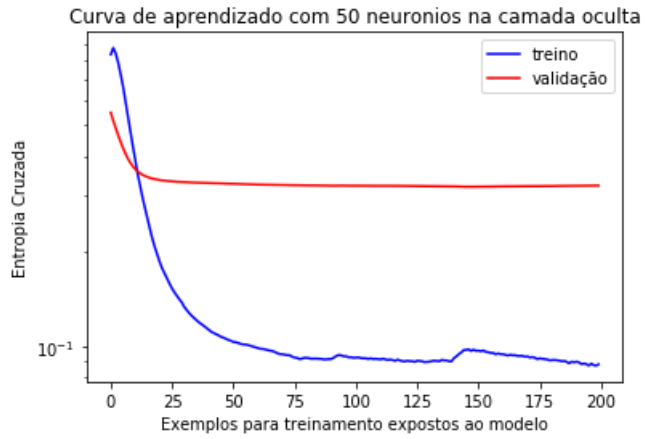
```
H =  15
Validação: Avg. loss: 0.5641, Acurácia: 632/1000 (63%)
Validação: Avg. loss: 0.3685, Acurácia: 857/1000 (86%)
Validação: Avg. loss: 0.3341, Acurácia: 863/1000 (86%)
Validação: Avg. loss: 0.3279, Acurácia: 863/1000 (86%)
Validação: Avg. loss: 0.3246, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3229, Acurácia: 864/1000 (86%)
Validação: Avg. loss: 0.3221, Acurácia: 861/1000 (86%)
Validação: Avg. loss: 0.3211, Acurácia: 862/1000 (86%)
Validação: Avg. loss: 0.3203, Acurácia: 863/1000 (86%)
Validação: Avg. loss: 0.3195, Acurácia: 864/1000 (86%)
```




```
H =  30
Validação: Avg. loss: 0.5697, Acurácia: 622/1000 (62%)
Validação: Avg. loss: 0.3471, Acurácia: 859/1000 (86%)
Validação: Avg. loss: 0.3333, Acurácia: 861/1000 (86%)
Validação: Avg. loss: 0.3308, Acurácia: 860/1000 (86%)
Validação: Avg. loss: 0.3306, Acurácia: 859/1000 (86%)
Validação: Avg. loss: 0.3314, Acurácia: 859/1000 (86%)
Validação: Avg. loss: 0.3312, Acurácia: 857/1000 (86%)
Validação: Avg. loss: 0.3315, Acurácia: 858/1000 (86%)
Validação: Avg. loss: 0.3304, Acurácia: 860/1000 (86%)
Validação: Avg. loss: 0.3297, Acurácia: 862/1000 (86%)
```




```
H =  50
Validação: Avg. loss: 0.5484, Acurácia: 647/1000 (65%)
Validação: Avg. loss: 0.3361, Acurácia: 859/1000 (86%)
Validação: Avg. loss: 0.3286, Acurácia: 863/1000 (86%)
Validação: Avg. loss: 0.3252, Acurácia: 861/1000 (86%)
Validação: Avg. loss: 0.3234, Acurácia: 862/1000 (86%)
Validação: Avg. loss: 0.3225, Acurácia: 867/1000 (87%)
Validação: Avg. loss: 0.3221, Acurácia: 868/1000 (87%)
Validação: Avg. loss: 0.3213, Acurácia: 867/1000 (87%)
Validação: Avg. loss: 0.3212, Acurácia: 869/1000 (87%)
Validação: Avg. loss: 0.3220, Acurácia: 868/1000 (87%)
```



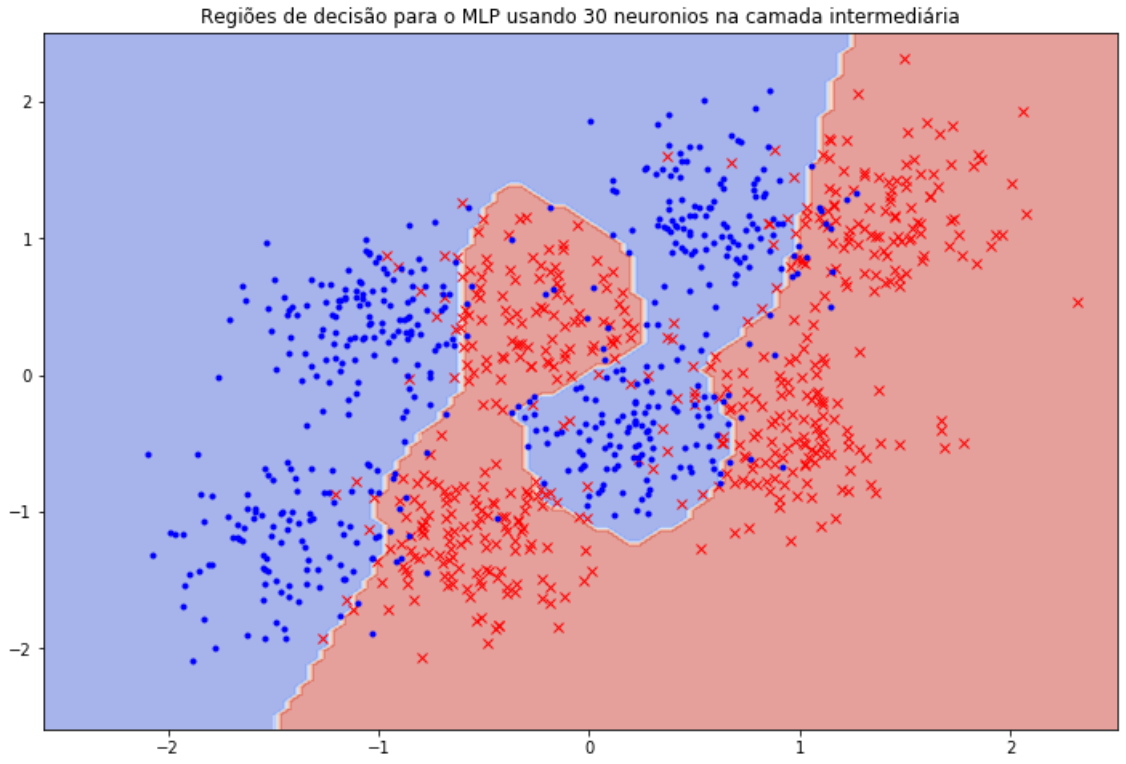
.....

Dados de teste: Avg. loss: 0.2873, Acurácia: 882/1000 (88%)

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-74-c6c483efb0a4> in <module>
      1 X_test_t = torch.FloatTensor(X_teste)
----> 2 y_hat_test = model(X_test_t)
      3 y_hat_test_class = np.where(y_hat_test.detach().numpy()<0.5, 0, 1)
      4 test_accuracy = np.sum(y_teste.reshape(-1,1)==y_hat_test_class) / len(y_teste)
      5 print("Acurácia de teste {:.2f}".format(test_accuracy))

~/.local/lib/python3.7/site-packages/keras/engine/base_layer.py in __call__(self, inputs, **
kwargs)
    439         if isinstance(inputs, list):
    440             inputs = inputs[:]
--> 441         with K.name_scope(self.name):
    442             # Handle laying building (weight creating, input spec locking).
    443             if not self.built:

~/.local/lib/python3.7/site-packages/tensorflow/python/framework/ops.py in __enter__(self)
   6511       try:
   6512         self._name_scope = g.name_scope(self._name)
-> 6513         return self._name_scope.__enter__()
   6514       except:
   6515         self._g_manager.__exit__(*sys.exc_info())

/usr/lib/python3.7/contextlib.py in __enter__(self)
    110         del self.args, self.kwds, self.func
    111         try:
--> 112             return next(self.gen)
    113         except StopIteration:
    114             raise RuntimeError("generator didn't yield") from None

~/.local/lib/python3.7/site-packages/tensorflow/python/framework/ops.py in name_scope(self,
 name)
   4310         # op name regex, which constrains the initial character.
   4311         if not _VALID_OP_NAME_REGEX.match(name):
-> 4312           raise ValueError("'%s' is not a valid scope name" % name)
   4313     old_stack = self._name_stack
   4314     if not name:  # Both for name=None and name="" we re-set to empty scope.

ValueError: 'Multi Layer Perceptron' is not a valid scope name
```

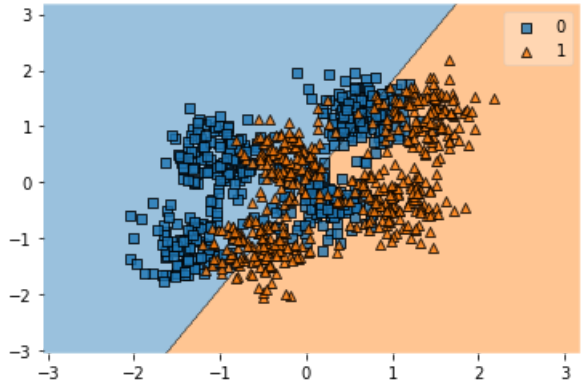
SVM

Acurácia de teste 0.88

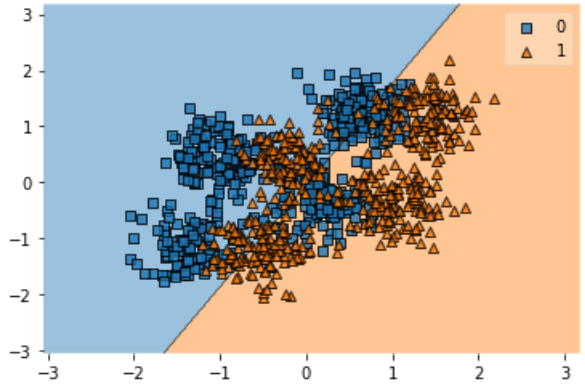


SVM

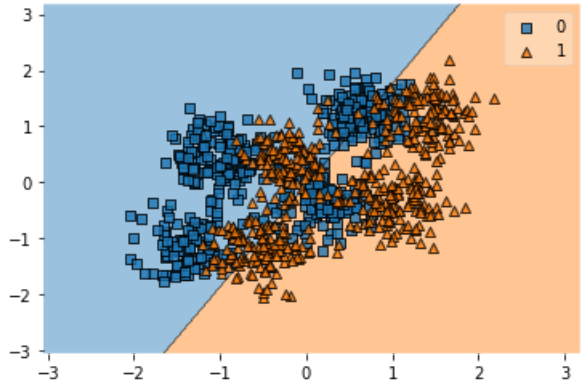
===
SVM com C = 1 e kernel = linear
Acurácia: 0.658, F1-score: 0.667, AUC: 0.658



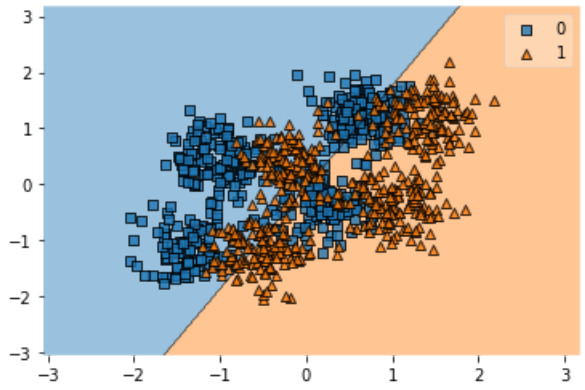
===
SVM com C = 10 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



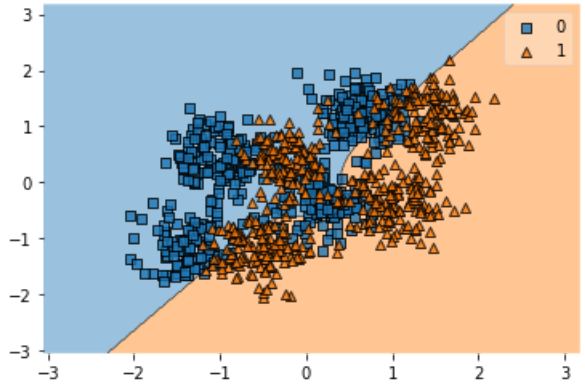
===
SVM com C = 50 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



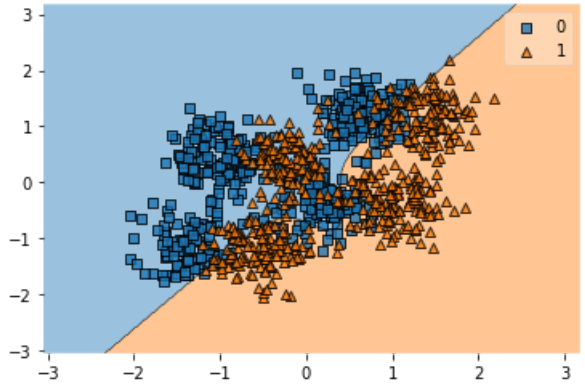
===
SVM com C = 100 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



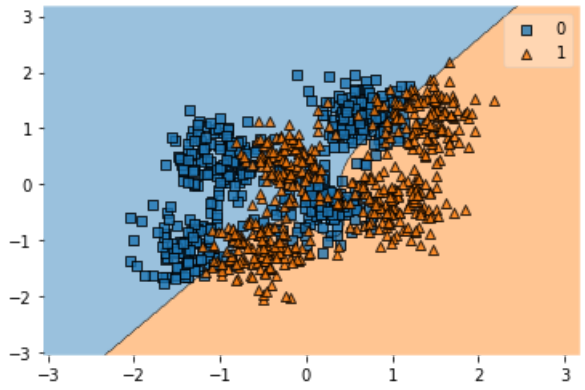
===
SVM com C = 1 e kernel = poly
Acurácia: 0.743, F1-score: 0.726, AUC: 0.746



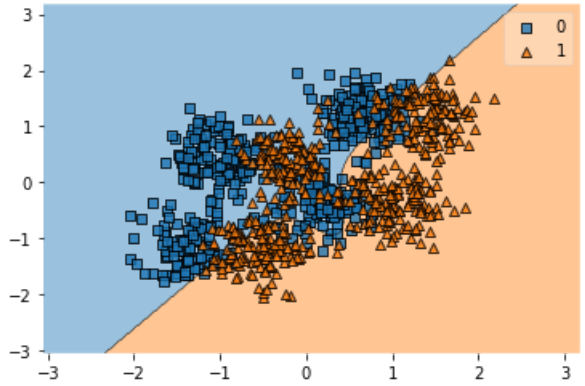
===
SVM com C = 10 e kernel = poly
Acurácia: 0.75, F1-score: 0.731, AUC: 0.754



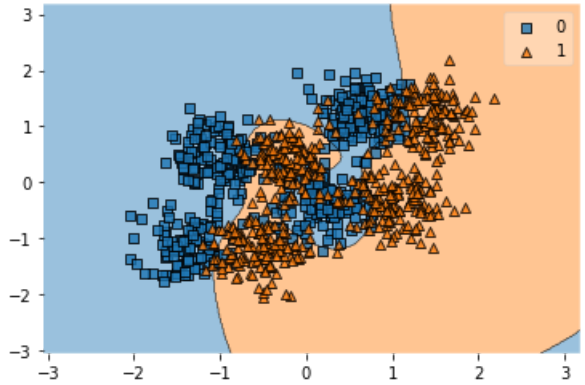
===
SVM com C = 50 e kernel = poly
Acurácia: 0.752, F1-score: 0.732, AUC: 0.756



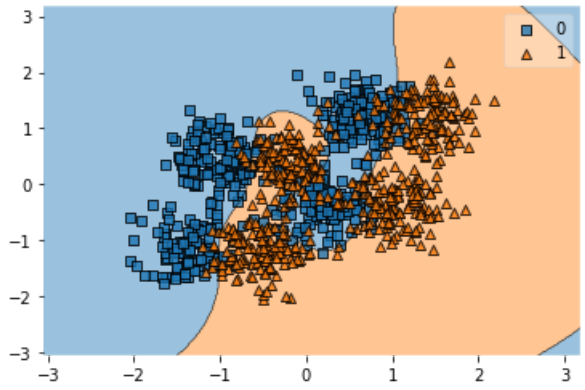
===
SVM com C = 100 e kernel = poly
Acurácia: 0.752, F1-score: 0.732, AUC: 0.756



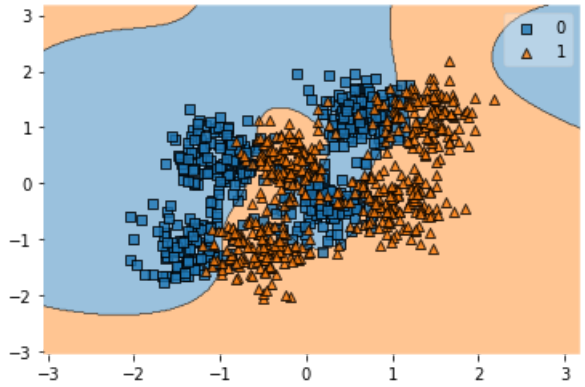
===
SVM com C = 1 e kernel = rbf
Acurácia: 0.853, F1-score: 0.858, AUC: 0.853



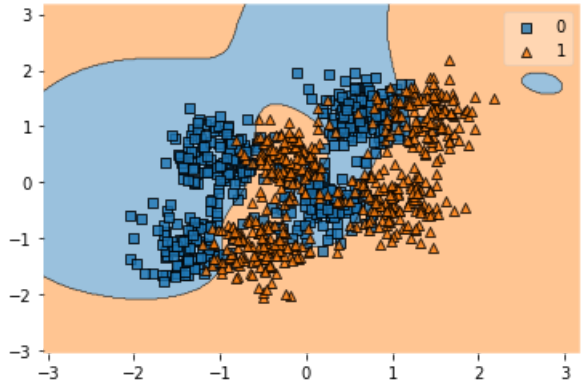
===
SVM com C = 10 e kernel = rbf
Acurácia: 0.864, F1-score: 0.868, AUC: 0.864



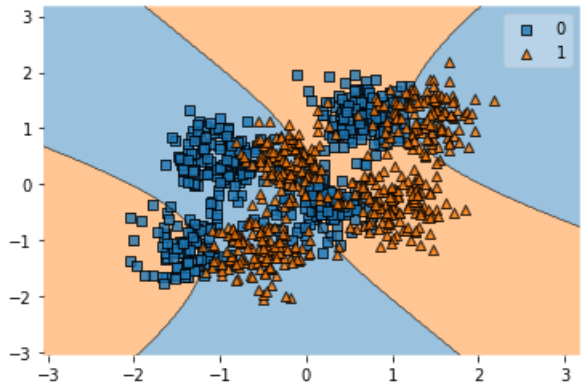
===
SVM com C = 50 e kernel = rbf
Acurácia: 0.867, F1-score: 0.871, AUC: 0.867



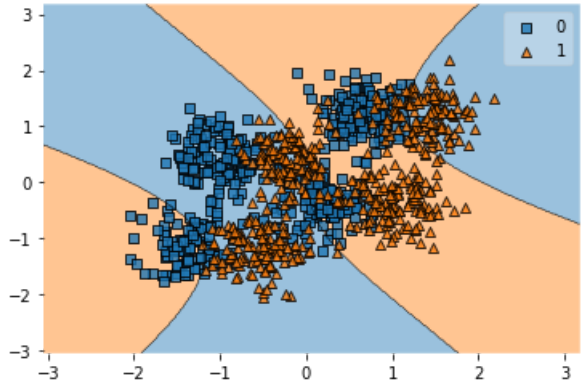
===
SVM com C = 100 e kernel = rbf
Acurácia: 0.866, F1-score: 0.870, AUC: 0.866



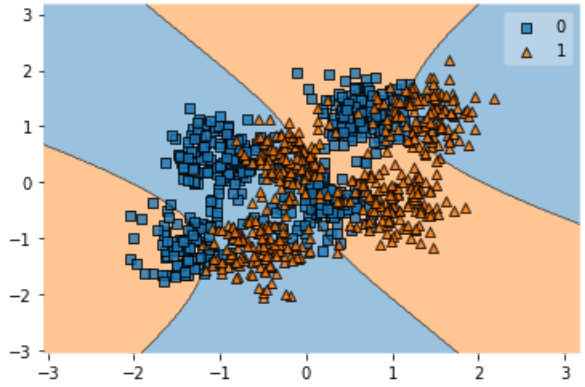
===
SVM com C = 1 e kernel = sigmoid
Acurácia: 0.415, F1-score: 0.421, AUC: 0.415



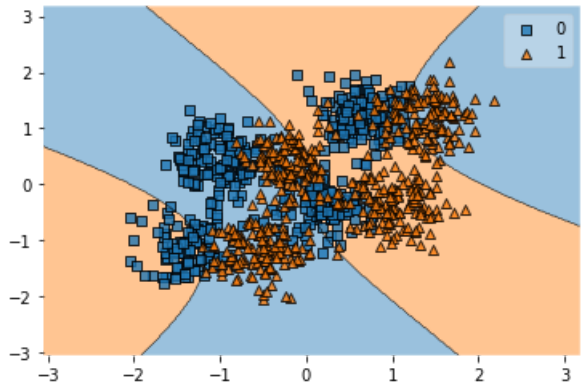
===
SVM com C = 10 e kernel = sigmoid
Acurácia: 0.413, F1-score: 0.421, AUC: 0.413



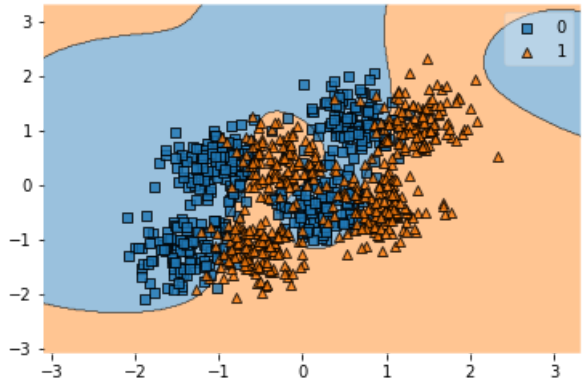
===
SVM com C = 50 e kernel = sigmoid
Acurácia: 0.414, F1-score: 0.422, AUC: 0.414



===
SVM com C = 100 e kernel = sigmoid
Acurácia: 0.414, F1-score: 0.422, AUC: 0.414



===
SVM com C = 50 e kernel = rbf
Acurácia: 0.874, F1-score: 0.873, AUC: 0.874



Fim fonte 2 html </fonte> </html>