Centralizando com HTML

EFC 3

html inicio

EFC 3

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

fim html

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}:

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

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

$$\frac{\partial inpZ}{\partial v_12} = x_1$$

Finalmente:

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

</center>

Parte II – Classificação binária com redes MLP e SVMs

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.

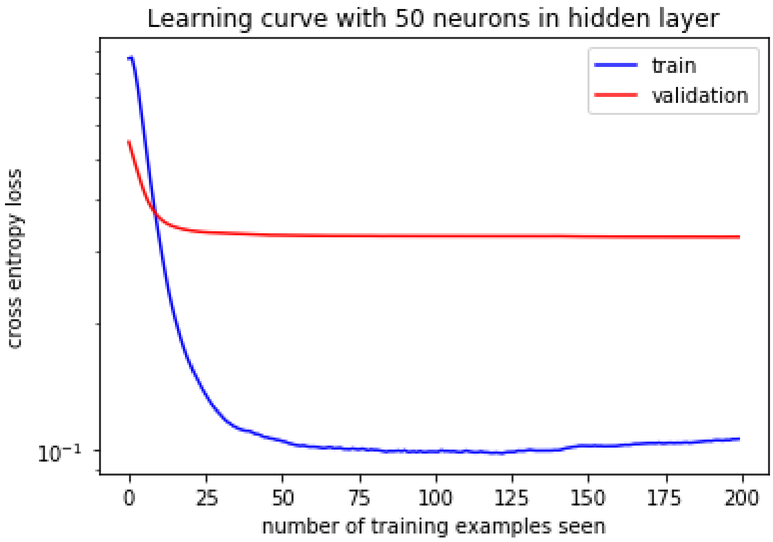


Figura 1: Curva de aprendizado.

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

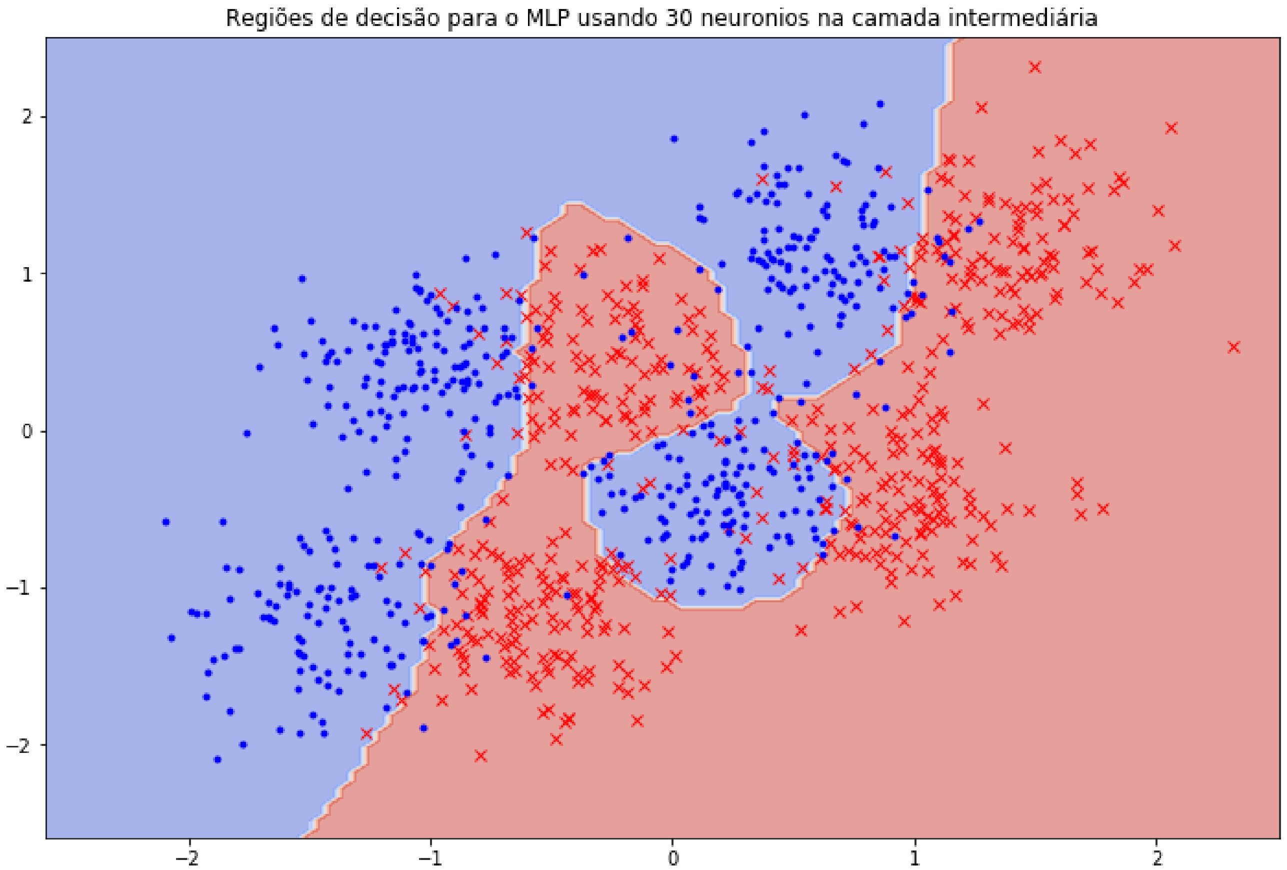
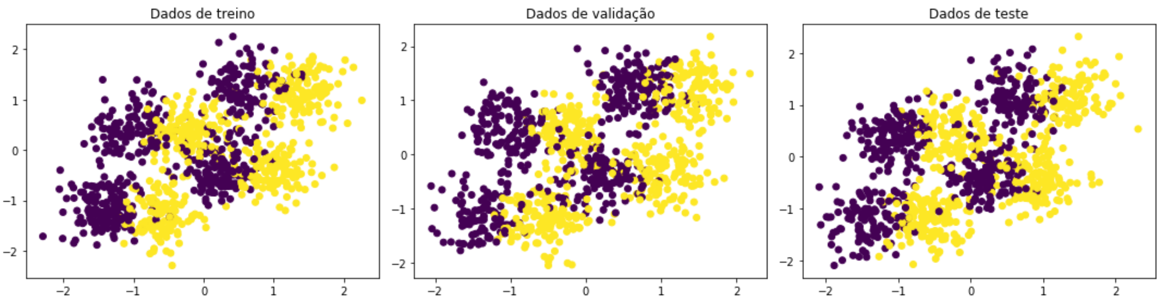


Figura 2: Regiões de decisão e classes

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



Aplicando a MLP

usando batch

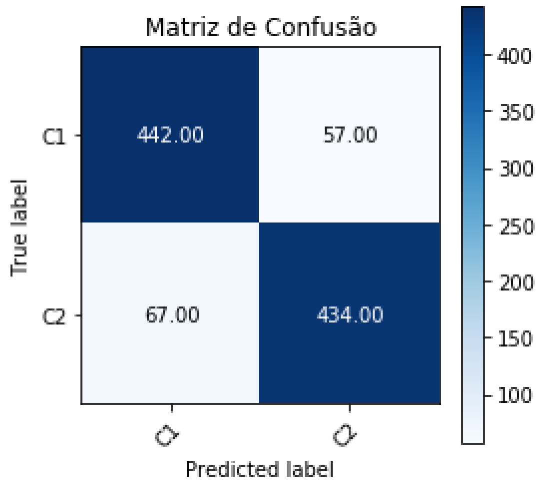
Out[197]:

```
MLPClassifier(activation='relu', alpha=0.0001, batch_size=36, 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)
```

```
Acurácia:
87.6%

Classification report:
              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
```

Out[198]:

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



Out[200]:

```
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)
```

Resultados usando

```
---------- Melhores parametros------------
Best parameters found:
 {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_sizes': (50, 50, 50), 'le
arning_rate': 'constant', 'solver': 'adam'}
---------- Melhores parametros------------

0.663 (+/-0.036) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.878 (+/-0.044) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.664 (+/-0.027) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.872 (+/-0.037) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.664 (+/-0.033) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.868 (+/-0.021) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.661 (+/-0.026) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.874 (+/-0.038) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.651 (+/-0.038) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.666 (+/-0.029) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.650 (+/-0.036) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.666 (+/-0.029) for {'activation': 'tanh', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.668 (+/-0.024) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.870 (+/-0.044) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.666 (+/-0.034) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.878 (+/-0.024) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.657 (+/-0.026) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.878 (+/-0.030) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.668 (+/-0.034) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.867 (+/-0.020) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.651 (+/-0.029) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}
```

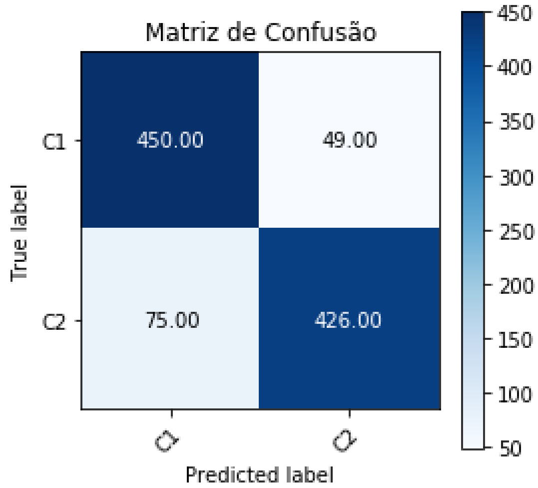
```
0.657 (+/-0.014) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.654 (+/-0.039) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.669 (+/-0.029) for {'activation': 'tanh', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.709 (+/-0.022) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.878 (+/-0.037) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.719 (+/-0.041) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.874 (+/-0.035) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.754 (+/-0.085) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.875 (+/-0.050) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.706 (+/-0.042) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.875 (+/-0.029) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.671 (+/-0.028) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.872 (+/-0.049) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.664 (+/-0.041) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.872 (+/-0.037) for {'activation': 'relu', 'alpha': 0.0001, 'hidden_layer_siz
es': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.711 (+/-0.018) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.882 (+/-0.034) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.712 (+/-0.031) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.879 (+/-0.034) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 50, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}

0.713 (+/-0.026) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'sgd'}

0.872 (+/-0.039) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'constant', 'solver': 'adam'}

0.719 (+/-0.036) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.880 (+/-0.039) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (50, 100, 50), 'learning_rate': 'adaptive', 'solver': 'adam'}
```

```
0.674 (+/-0.029) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'constant', 'solver': 'sgd'}

0.867 (+/-0.041) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'constant', 'solver': 'adam'}

0.670 (+/-0.036) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'adaptive', 'solver': 'sgd'}

0.871 (+/-0.040) for {'activation': 'relu', 'alpha': 0.05, 'hidden_layer_size
s': (100,), 'learning_rate': 'adaptive', 'solver': 'adam'}
```

```
Results on the test set:
              precision    recall  f1-score   support

        -1.0       0.86      0.90      0.88       499
         1.0       0.90      0.85      0.87       501

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

Out[203]:

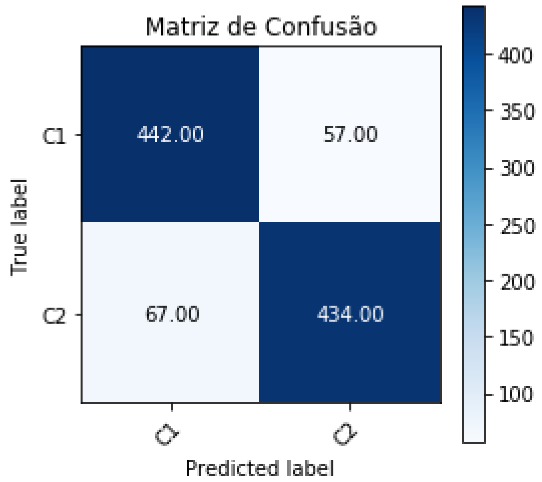
```
(1.5, -0.5)
```



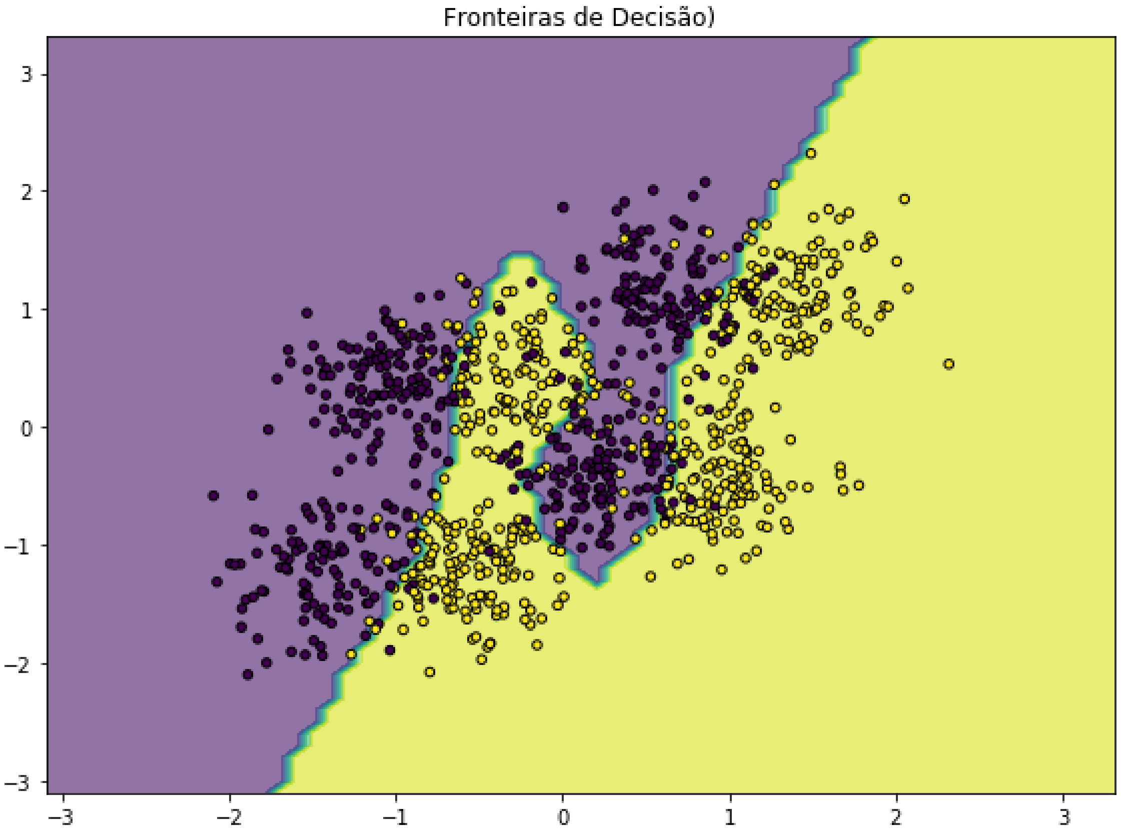
```
Acurácia:
87.6%

Classification report:
              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
```

Out[204]:

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



Fronteiras de decição

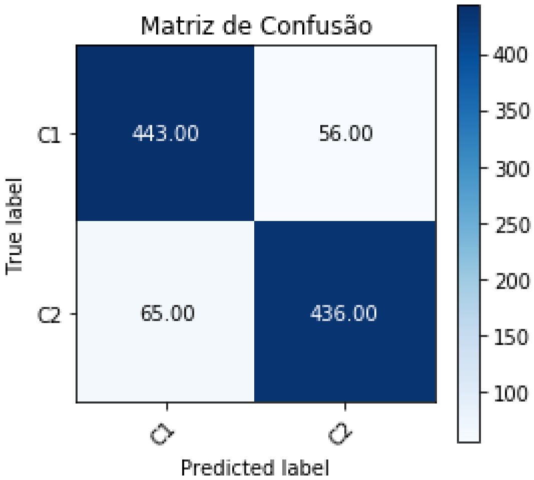

Fronteiras de Decisão)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

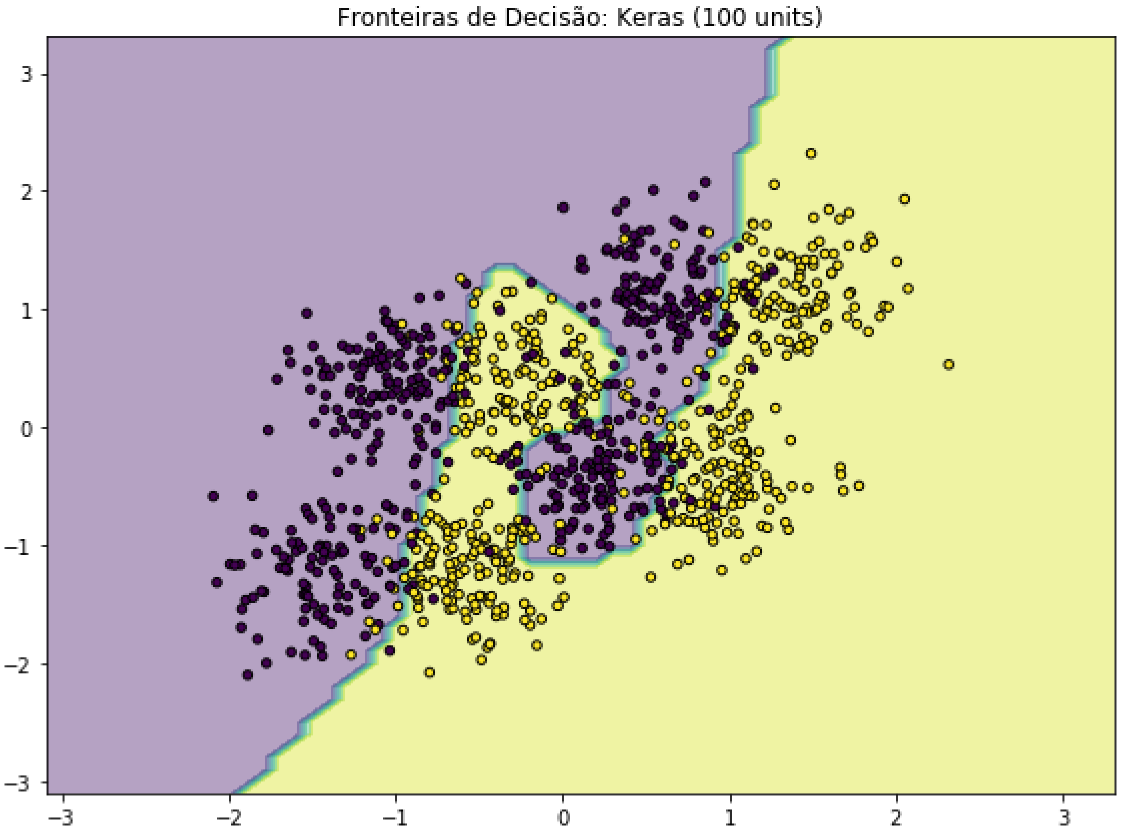
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
Input_Layer (Dense)          (None, 100)               300       
________________________________________________________________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


_________________________________________________________________
Optimizer:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: False



---------------------------------------------------------------------------
NameError                                 Traceback (most recent call last)
<ipython-input-10-684d061c2c1b> in <module>
----> 1 plt.figure(figsize=(13, 4))
      2 plt.subplot(1, 2, 1)
      3 plt.plot(history.history["loss"], label="Loss", color='red')
      4 plt.plot(history.history["accuracy"], label="Accuracy")
      5 plt.legend()

NameError: name 'plt' is not defined
```

```
Acurácia:
87.9%

Classification report:
              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
```

Out[217]:

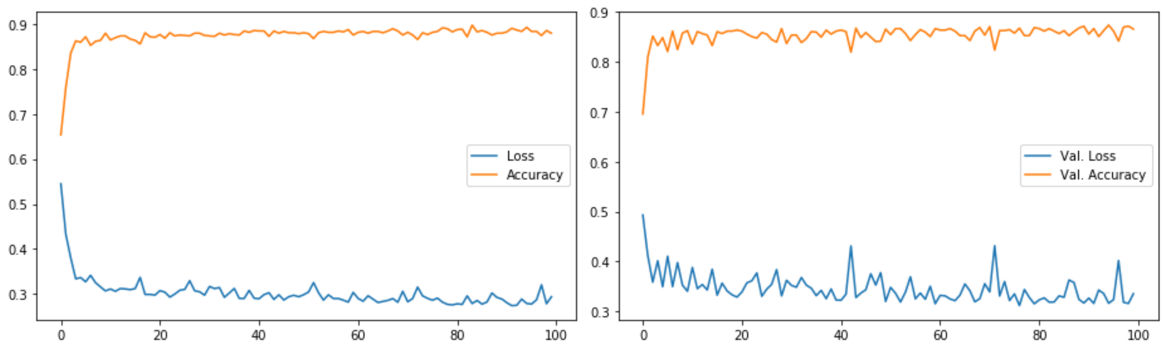
```
(1.5, -0.5)
```





com Keras, mais neuronios

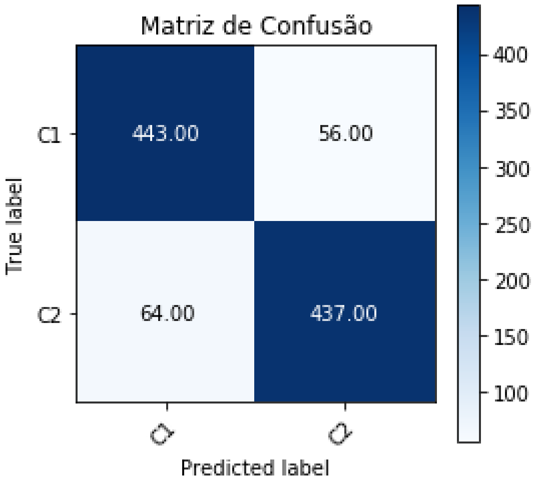
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer (Dense)          (None, 32768)             98304
_________________________________________________________________
Output_Layer (Dense)         (None, 2)                 65538
=================================================================
Total params: 163,842
Trainable params: 163,842
Non-trainable params: 0
_________________________________________________________________
None

_________________________________________________________________
Optimizer:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

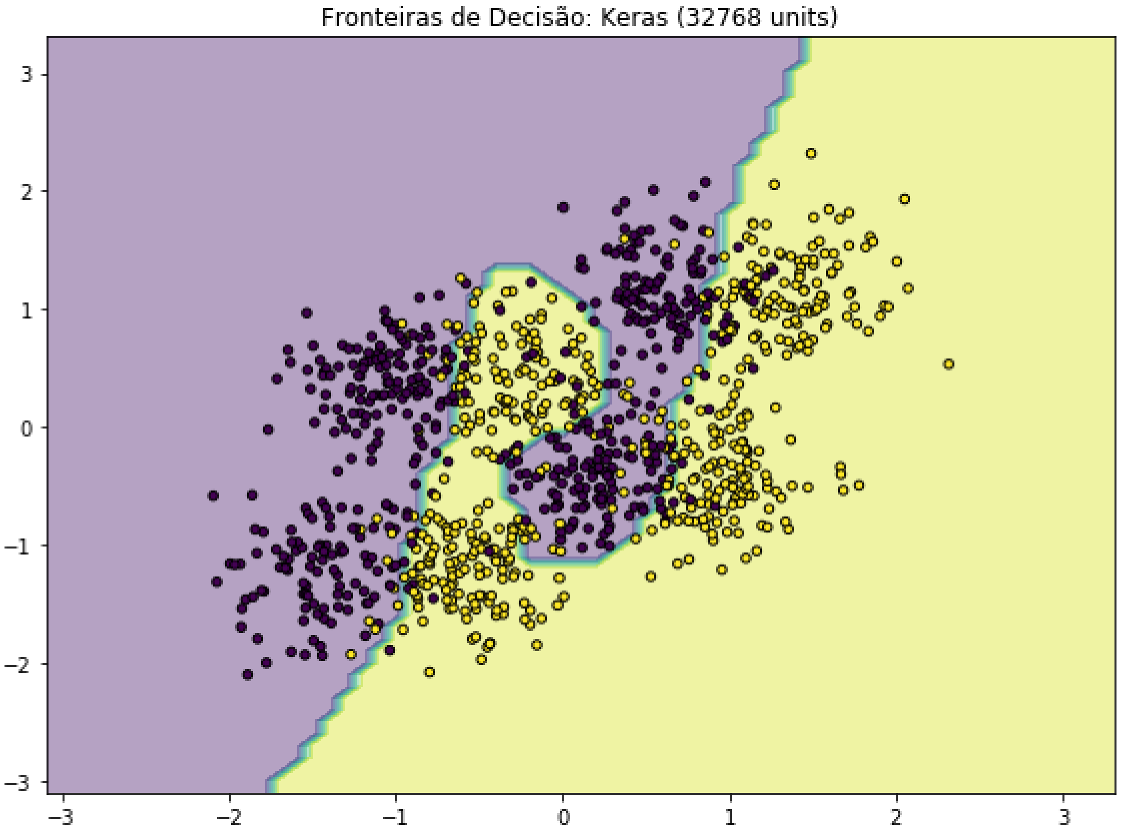


```
Acurácia:
88.0%

Classification report:
              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
```

Out[222]:

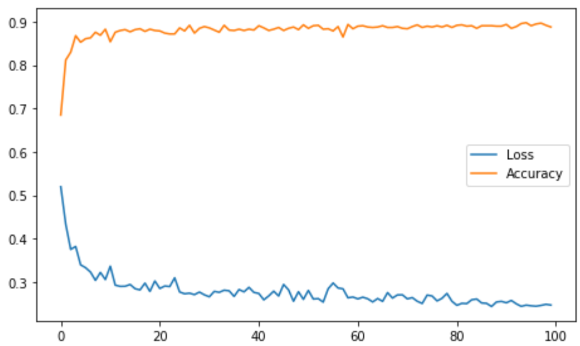
```
(1.5, -0.5)
```

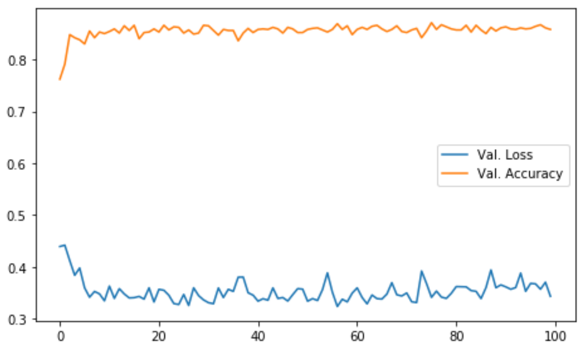




Mais camadas densas

```
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
_________________________________________________________________
Input_Layer_6 (Dense)        (None, 1024)              1049600
_________________________________________________________________
Output_Layer (Dense)         (None, 2)                 2050
=================================================================
Total params: 5,253,122
Trainable params: 5,253,122
Non-trainable params: 0
_________________________________________________________________
None

_________________________________________________________________
Optimizer:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

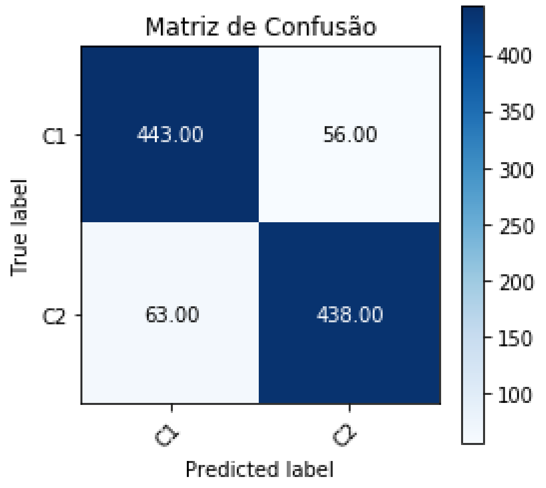


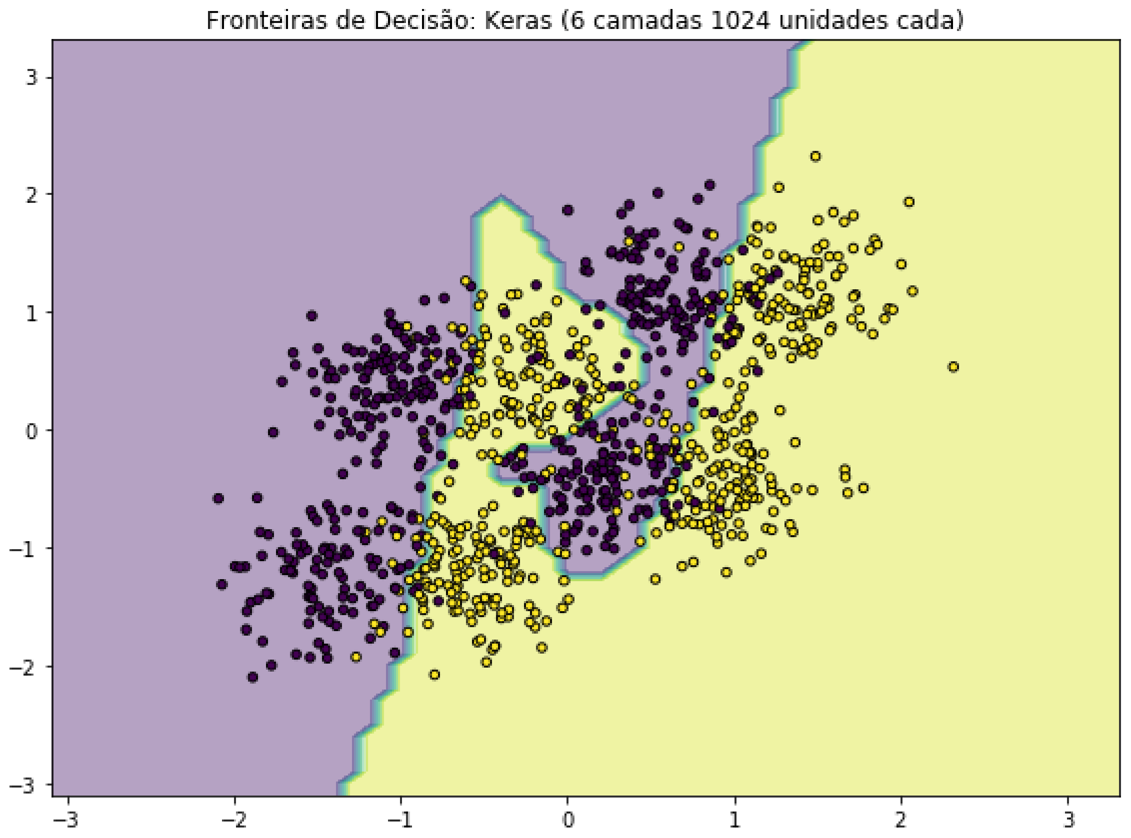


```
Acurácia:
88.1%

Classification report:
              precision    recall  f1-score   support

          C1       0.88      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
```

Out[227]:

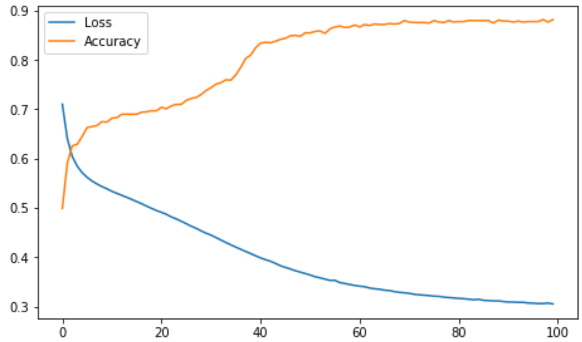
```
(1.5, -0.5)
```

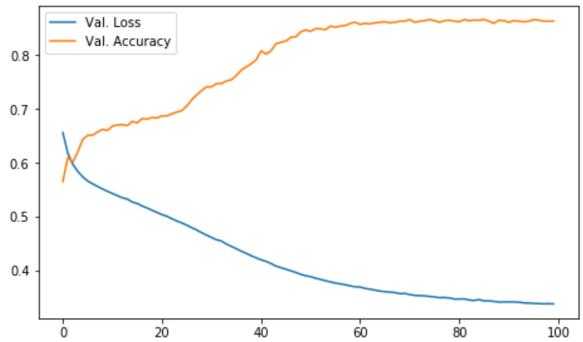




Minima quantidade de neuronios

```
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

_________________________________________________________________
Optimizer:
- learning_rate: 0.001
- beta_1: 0.9
- beta_2: 0.999
- decay: 0.0
- epsilon: 0.0
- amsgrad: True
```

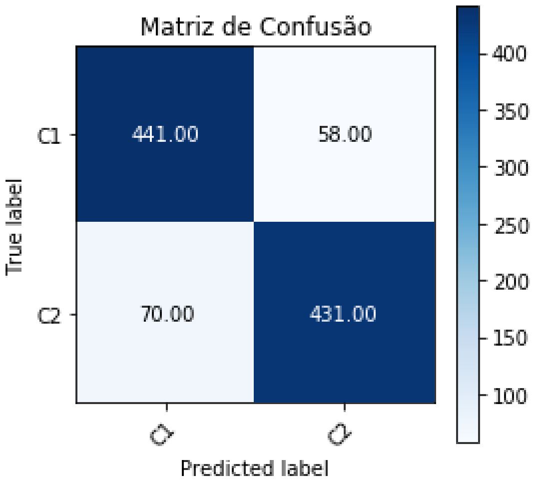


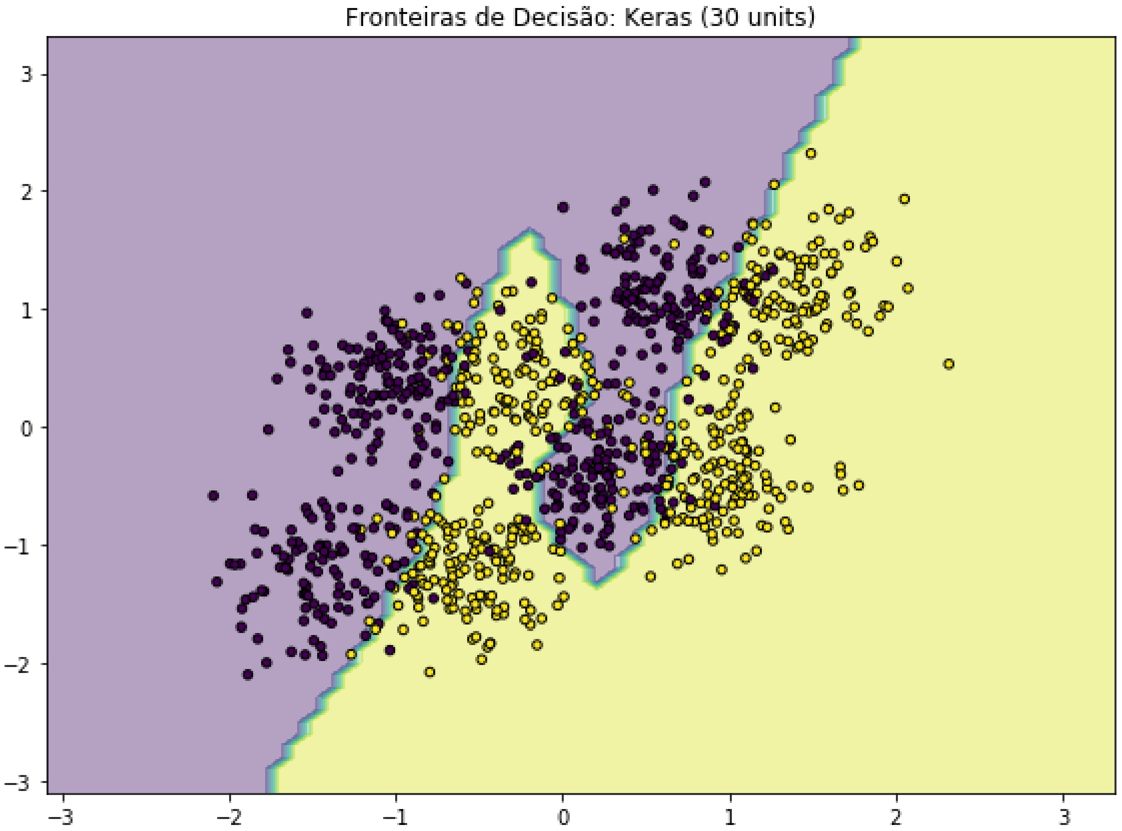


```
Acurácia:
87.2%

Classification report:
              precision    recall  f1-score   support

          C1       0.86      0.88      0.87       499
          C2       0.88      0.86      0.87       501

    accuracy                           0.87      1000
   macro avg       0.87      0.87      0.87      1000
weighted avg       0.87      0.87      0.87      1000
```

Out[231]:

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





SVM

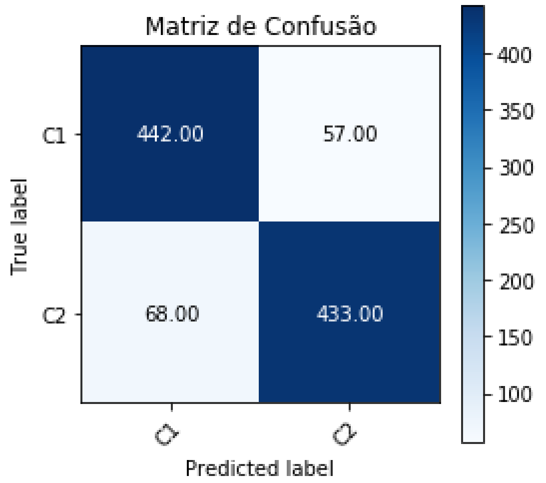
Out[233]:

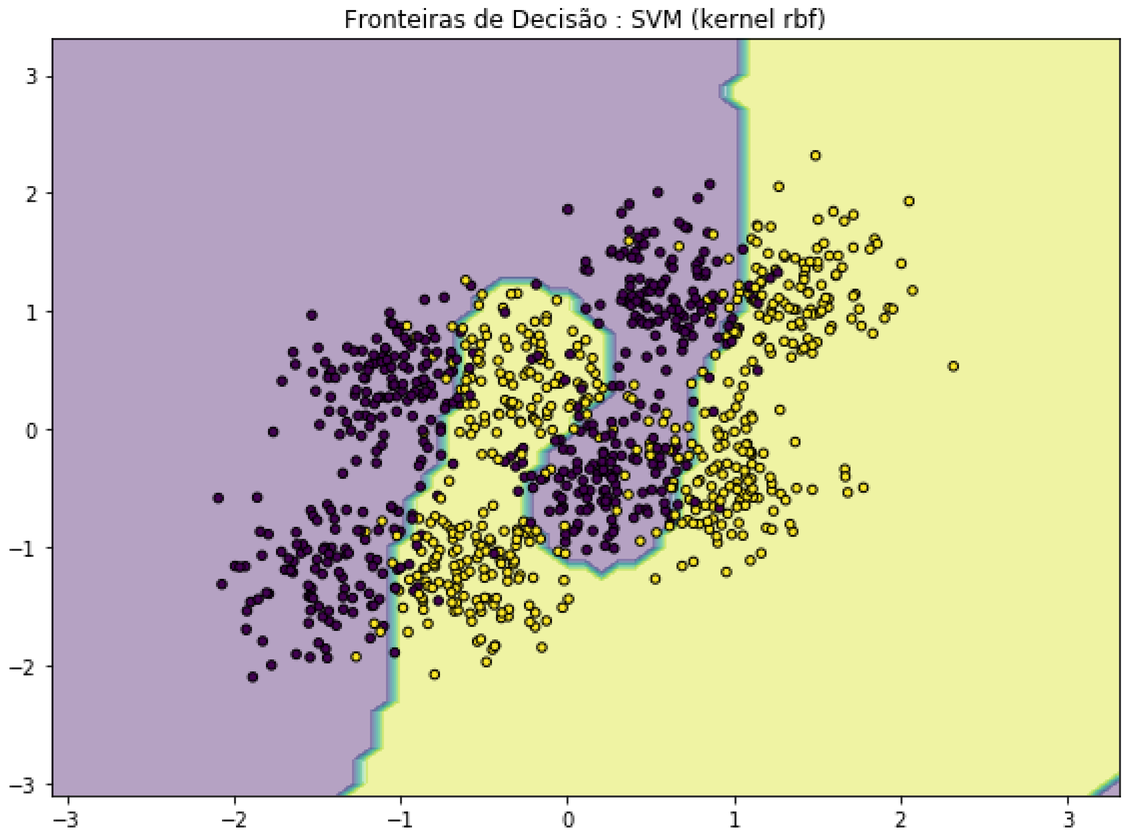
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='rbf',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
Acurácia:
87.5%

Classification report:
              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
```

Out[234]:

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





SVM diferente

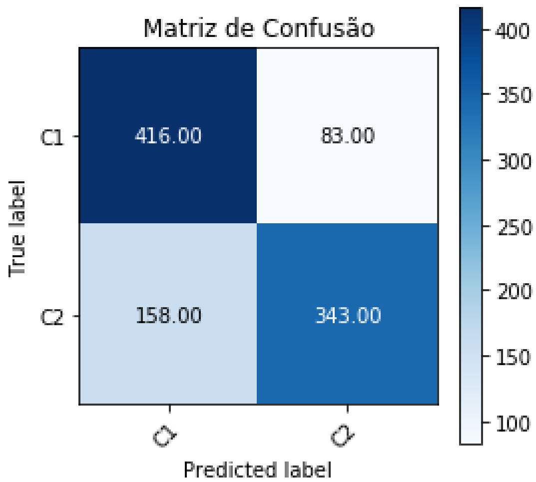
Out[236]:

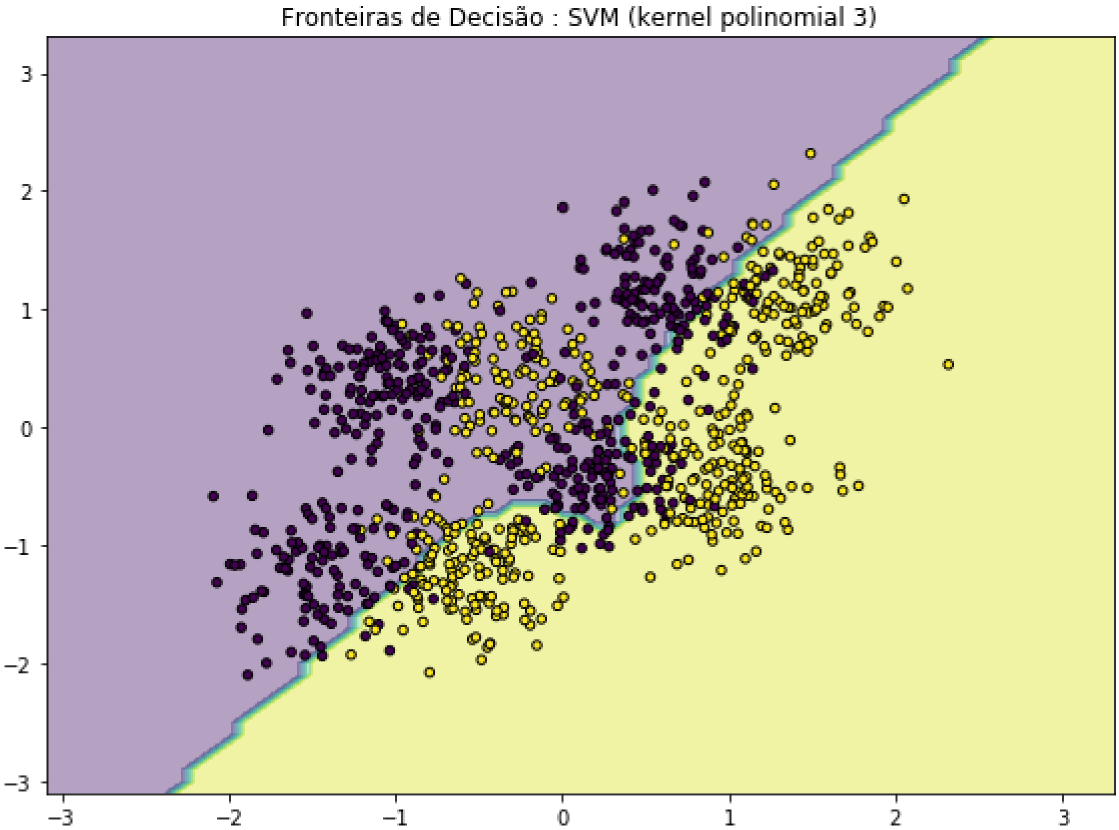
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=3, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
Acurácia:
75.9%

Classification report:
              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
```

Out[237]:

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





SVM 9

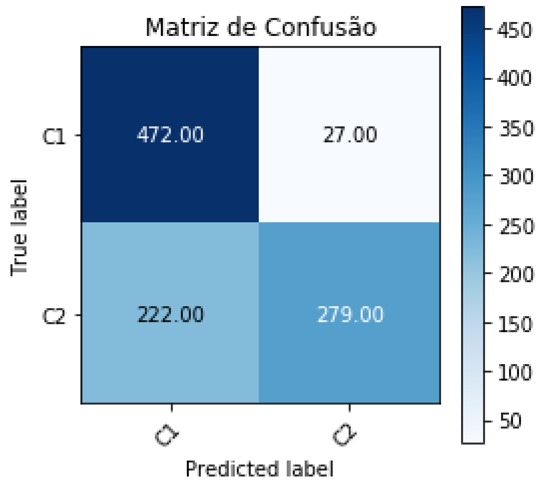
Out[239]:

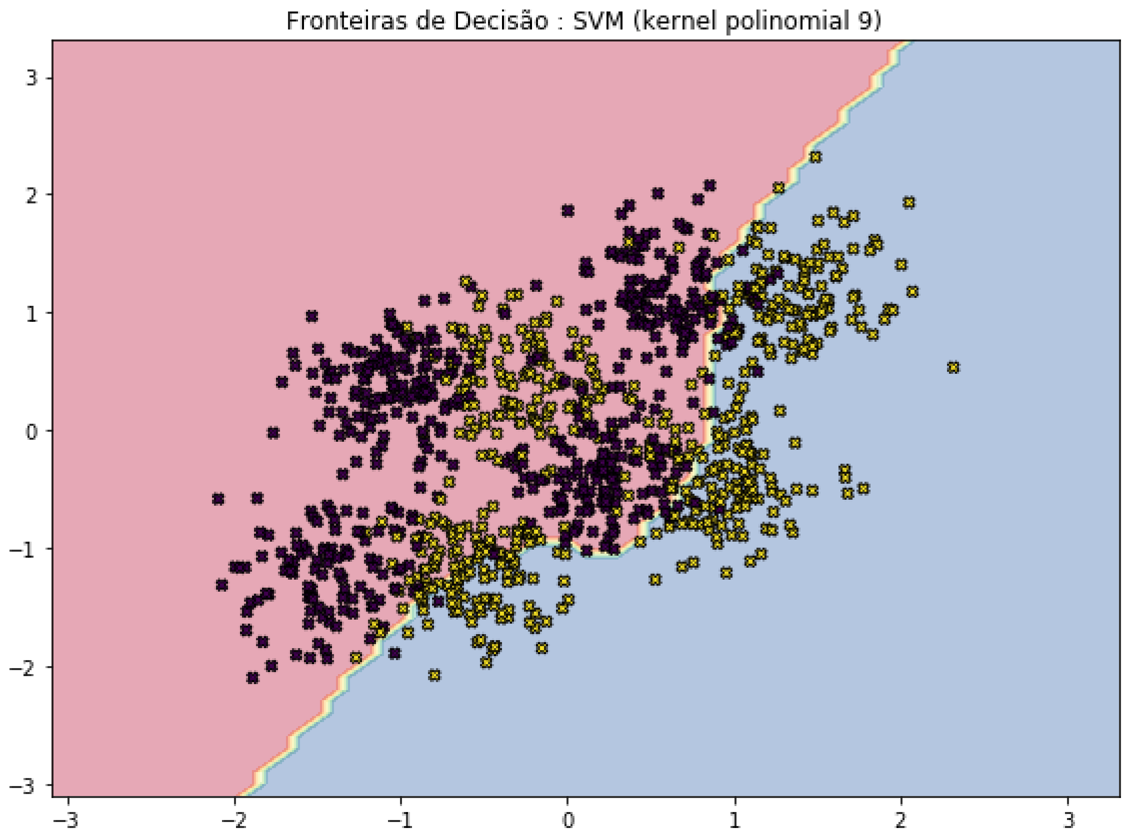
```
SVC(C=5, cache_size=200, class_weight=None, coef0=0.0,
    decision_function_shape='ovr', degree=9, gamma='scale', kernel='poly',
    max_iter=-1, probability=False, random_state=1, shrinking=True, tol=0.001,
    verbose=False)
```

```
Acurácia:
75.1%

Classification report:
              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
```

Out[240]:

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





```
H =  5

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:17: UserWarning: Us
ing a target size (torch.Size([1000, 2])) that is different to the input size (t
orch.Size([1000, 1])) is deprecated. Please ensure they have the same size.

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-74-823fb79200a5> in <module>
     15         otimizador.zero_grad()
     16         saida = modelo(X_treino_tmp)
---> 17         perda = F.binary_cross_entropy(saida, y_treino_tmp)
     18         perda.backward()
     19         otimizador.step()

~/.local/lib/python3.7/site-packages/torch/nn/functional.py in binary_cross_entr
opy(input, target, weight, size_average, reduce, reduction)
   2056     if input.numel() != target.numel():
   2057         raise ValueError("Target and input must have the same number of
 elements. target nelement ({}) "
-> 2058                          "!= input nelement ({})".format(target.numel(),
input.numel()))
   2059
   2060     if weight is not None:

ValueError: Target and input must have the same number of elements. target nelem
ent (2000) != input nelement (1000)
```

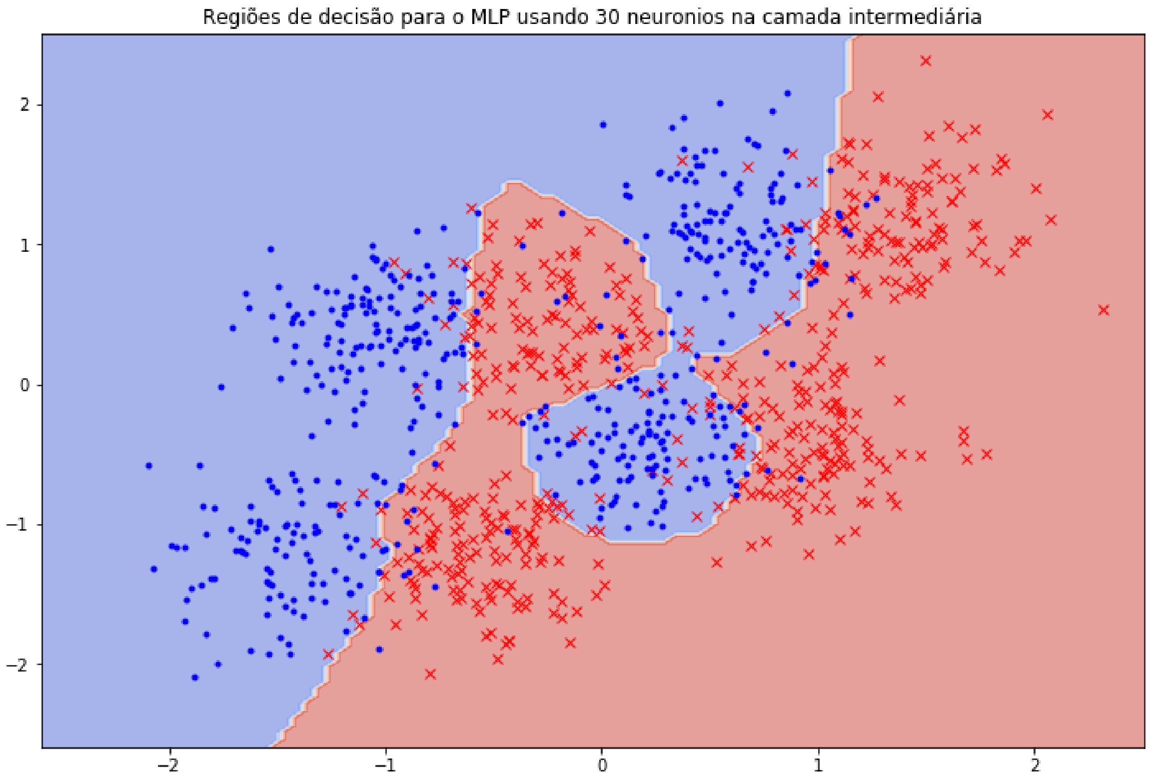
online (padrão-a-padrão)

```
H =  5

/usr/local/lib/python3.7/dist-packages/ipykernel_launcher.py:24: UserWarning: Us
ing a target size (torch.Size([2])) that is different to the input size (torch.S
ize([1])) is deprecated. Please ensure they have the same size.

---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-75-40a65456c008> in <module>
     22             otimizador.zero_grad()
     23             saida = modelo(data)
---> 24             perda = F.binary_cross_entropy(saida, target)
     25             perda.backward()
     26             otimizador.step()

~/.local/lib/python3.7/site-packages/torch/nn/functional.py in binary_cross_entr
opy(input, target, weight, size_average, reduce, reduction)
   2056     if input.numel() != target.numel():
   2057         raise ValueError("Target and input must have the same number of
 elements. target nelement ({}) "
-> 2058                          "!= input nelement ({})".format(target.numel(),
input.numel()))
   2059
   2060     if weight is not None:

ValueError: Target and input must have the same number of elements. target nelem
ent (2) != input nelement (1)

................................................................................
```

```
Dados de teste: Avg. loss: 0.3240, Accuracy: 881/1000 (88%)
```

```
/usr/local/lib/python3.7/dist-packages/torch/nn/_reduction.py:46: UserWarning: s
ize_average and reduce args will be deprecated, please use reduction='sum' inste
ad.
  warnings.warn(warning.format(ret))
```

```
Acurácia de teste 0.88
```



SVM

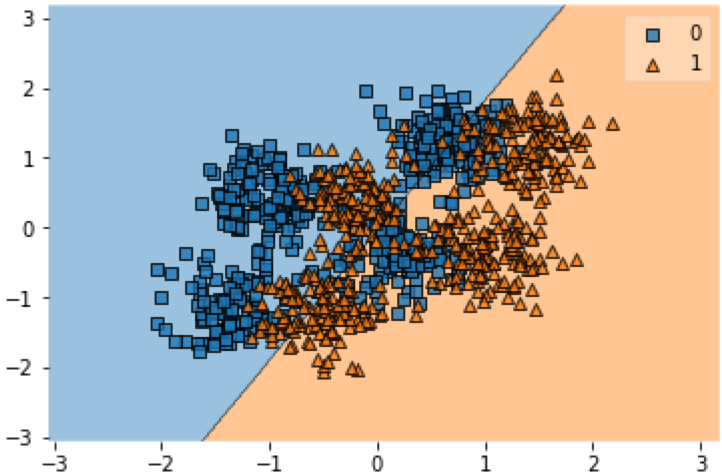
Out[61]:

```
SVC(C=1.0, 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=None, shrinking=True,
    tol=0.001, verbose=False)
```

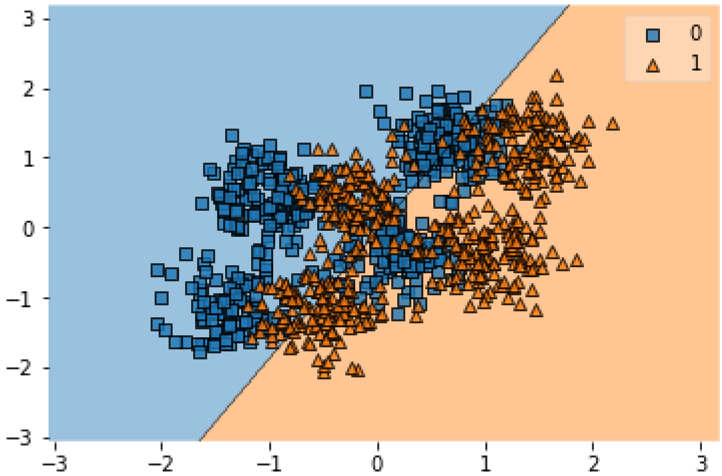
Out[62]:

```
0.853
```

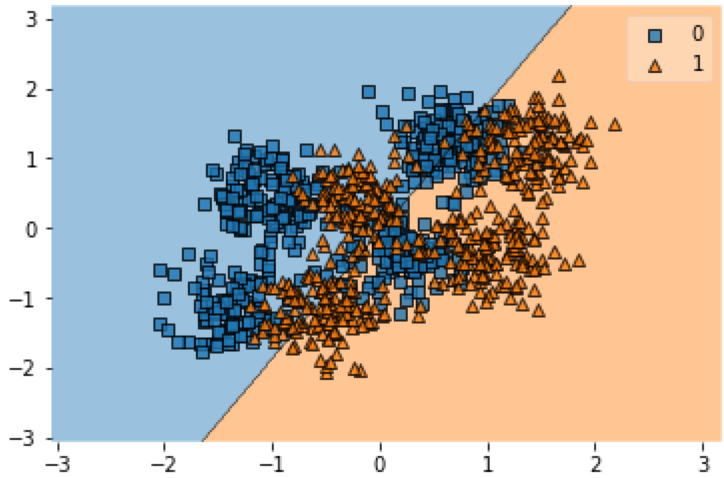
===
SVM com C = 1 e kernel = linear
Acurácia: 0.658, F1-score: 0.667, AUC: 0.658



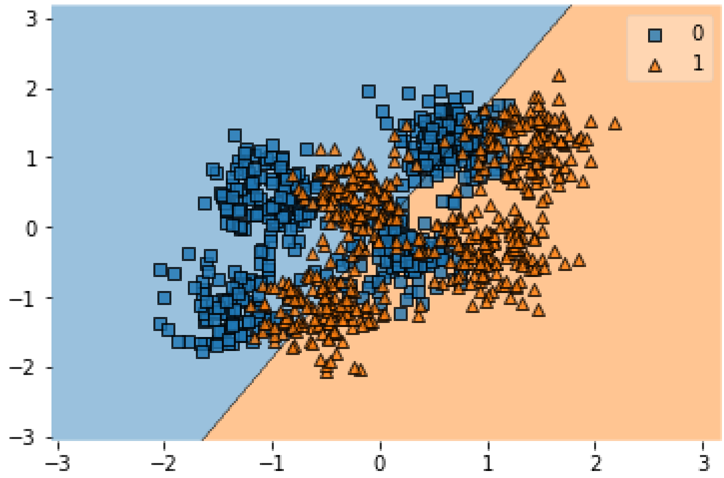
===
SVM com C = 10 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



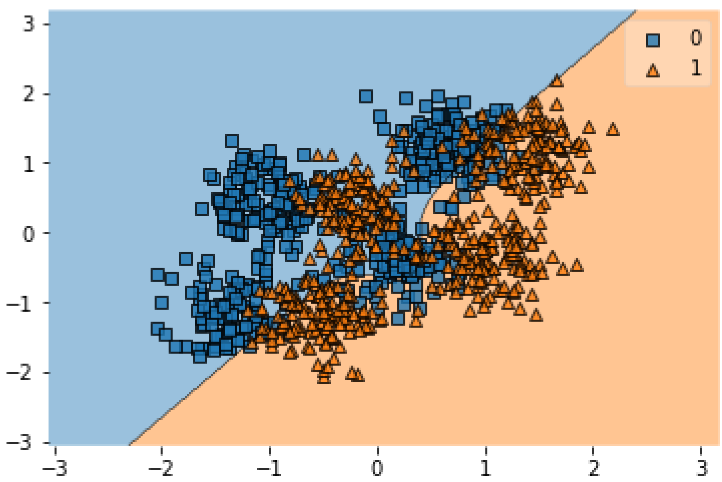
===
SVM com C = 50 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



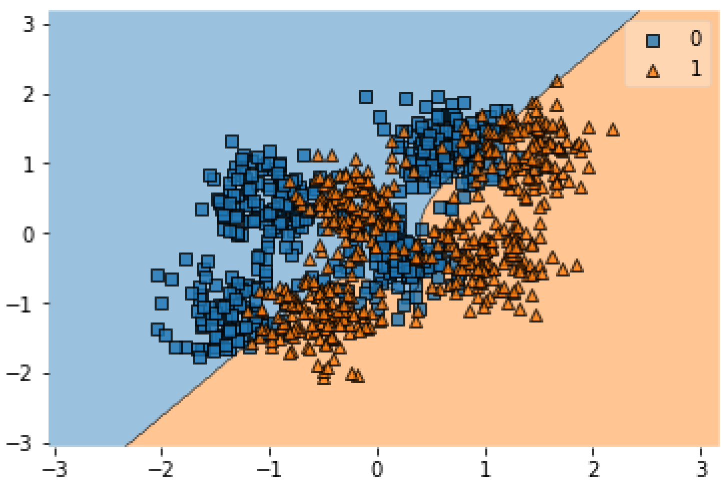
===
SVM com C = 100 e kernel = linear
Acurácia: 0.661, F1-score: 0.670, AUC: 0.661



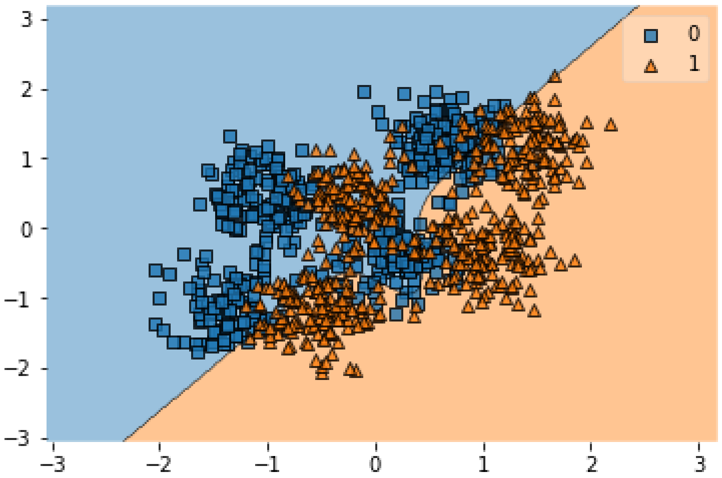
===
SVM com C = 1 e kernel = poly
Acurácia: 0.743, F1-score: 0.726, AUC: 0.746



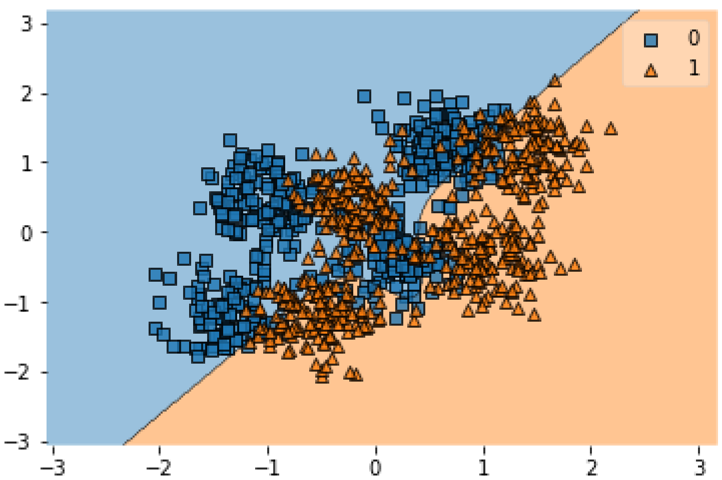
===
SVM com C = 10 e kernel = poly
Acurácia: 0.75, F1-score: 0.731, AUC: 0.754



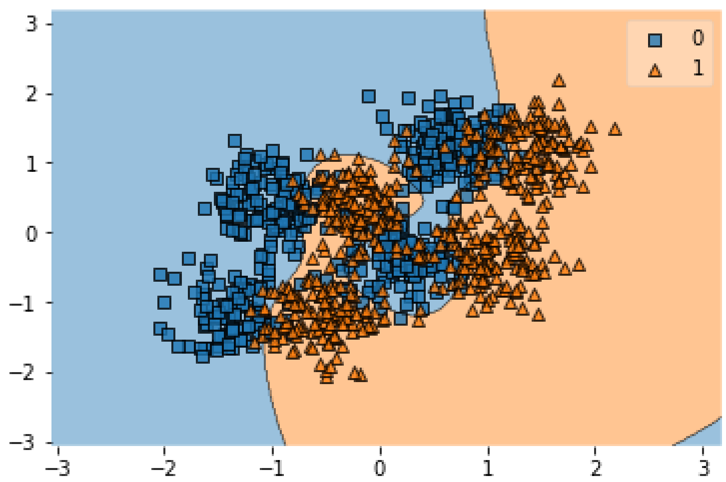
===
SVM com C = 50 e kernel = poly
Acurácia: 0.752, F1-score: 0.732, AUC: 0.756



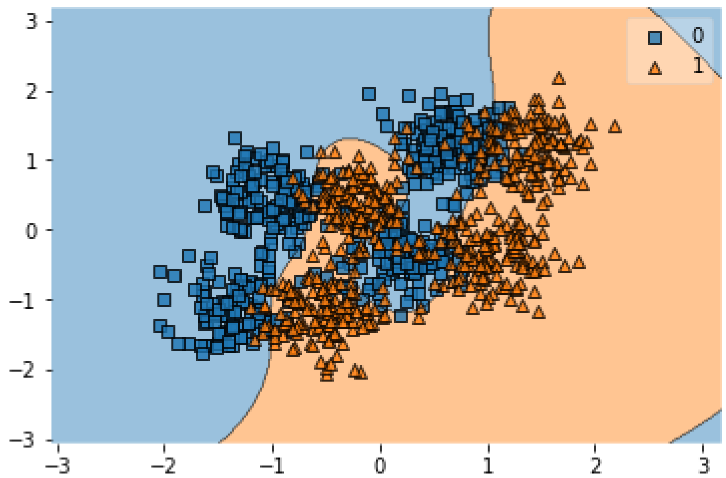
===
SVM com C = 100 e kernel = poly
Acurácia: 0.752, F1-score: 0.732, AUC: 0.756



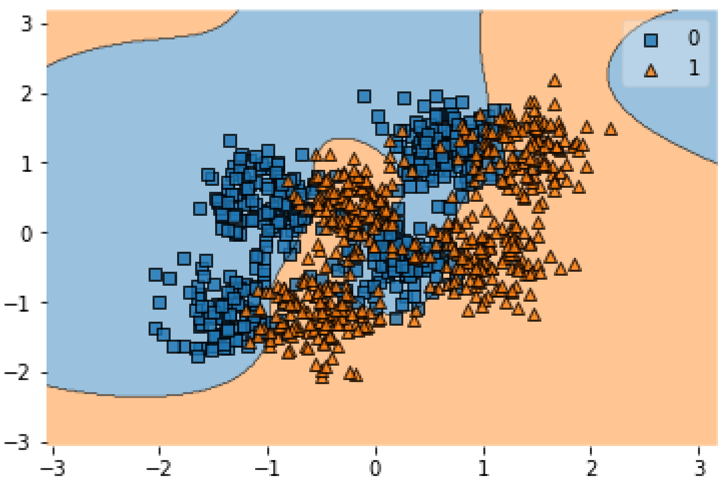
===
SVM com C = 1 e kernel = rbf
Acurácia: 0.853, F1-score: 0.858, AUC: 0.853



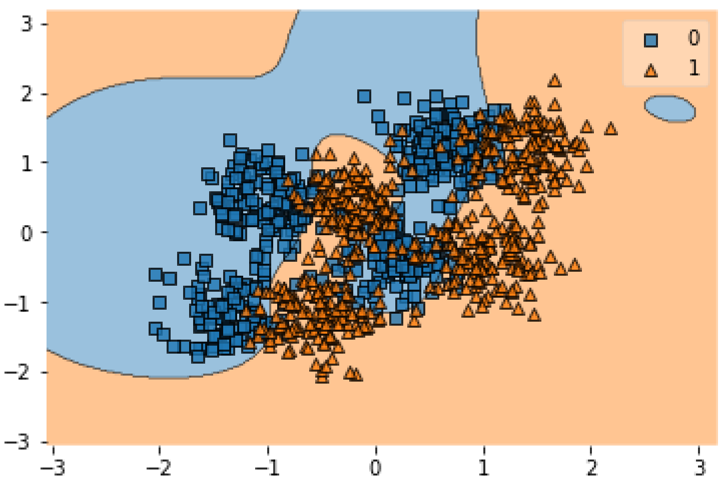
===
SVM com C = 10 e kernel = rbf
Acurácia: 0.864, F1-score: 0.868, AUC: 0.864



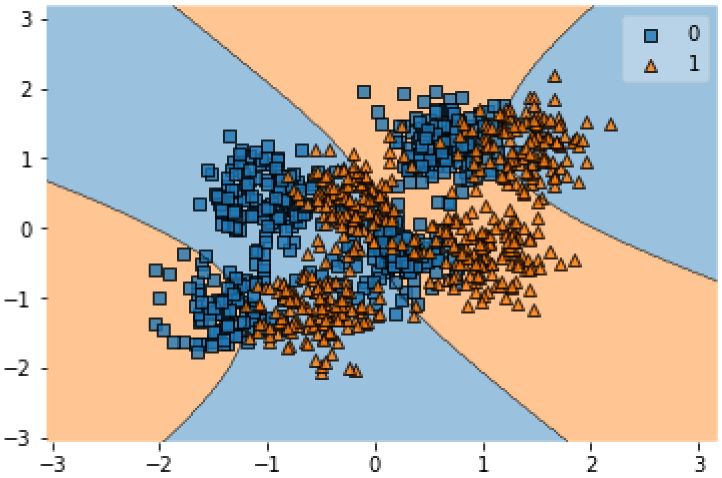
===
SVM com C = 50 e kernel = rbf
Acurácia: 0.867, F1-score: 0.871, AUC: 0.867



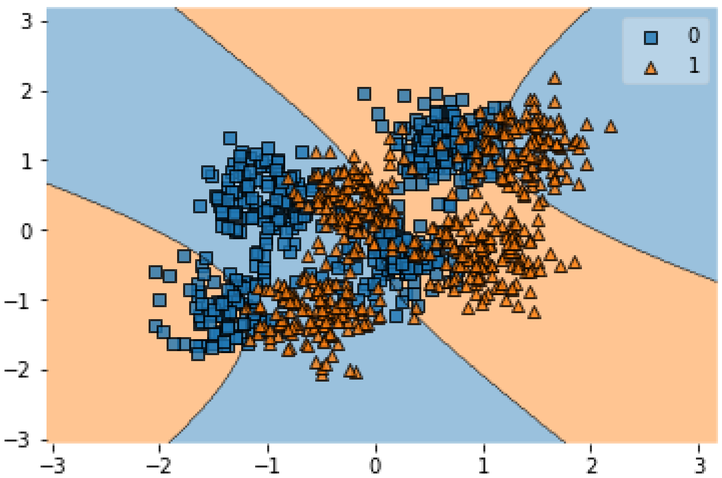
===
SVM com C = 100 e kernel = rbf
Acurácia: 0.866, F1-score: 0.870, AUC: 0.866



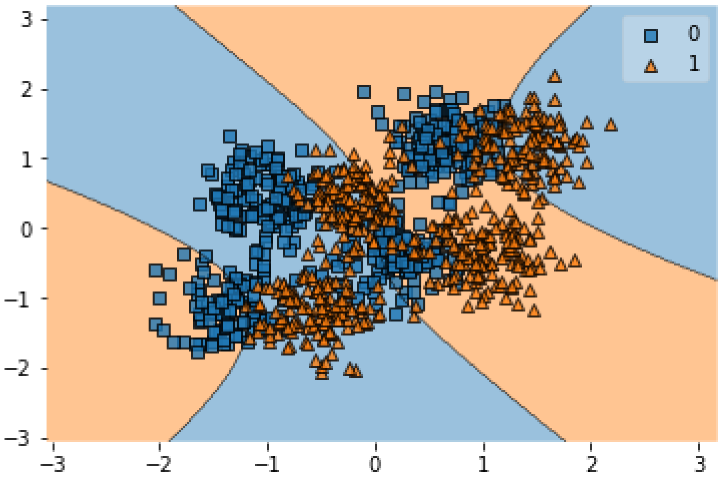
===
SVM com C = 1 e kernel = sigmoid
Acurácia: 0.415, F1-score: 0.421, AUC: 0.415



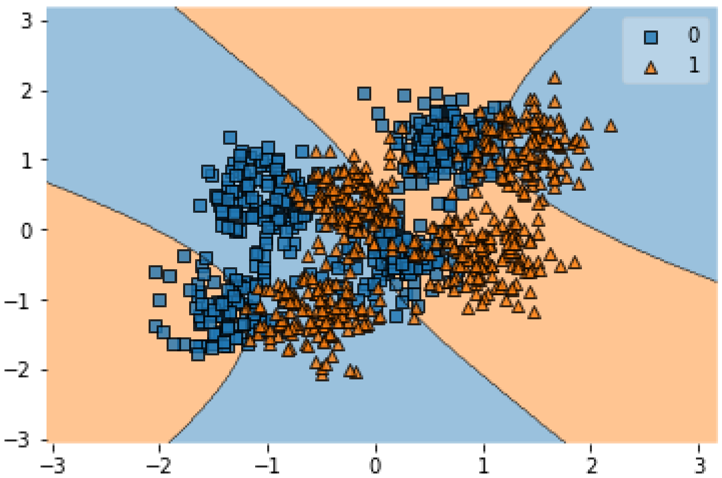

```
===
SVM com C = 10 e kernel = sigmoid
Acurácia: 0.413, F1-score: 0.421, AUC: 0.413
```




```
===
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
```