EFC03

November 4, 2019

1 IA006–Exercícios de Fixação de Conceitos

1.1 EFC3 - 2s2019

1.1.1 Parte 1 - Revisitando algoritmo de retropropagação do erro

Dado o exposto no exercício e definindo as seguintes variáveis intermediárias:

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 por conseguinte 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:

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

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

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

Entãão para v_{12}:

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

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

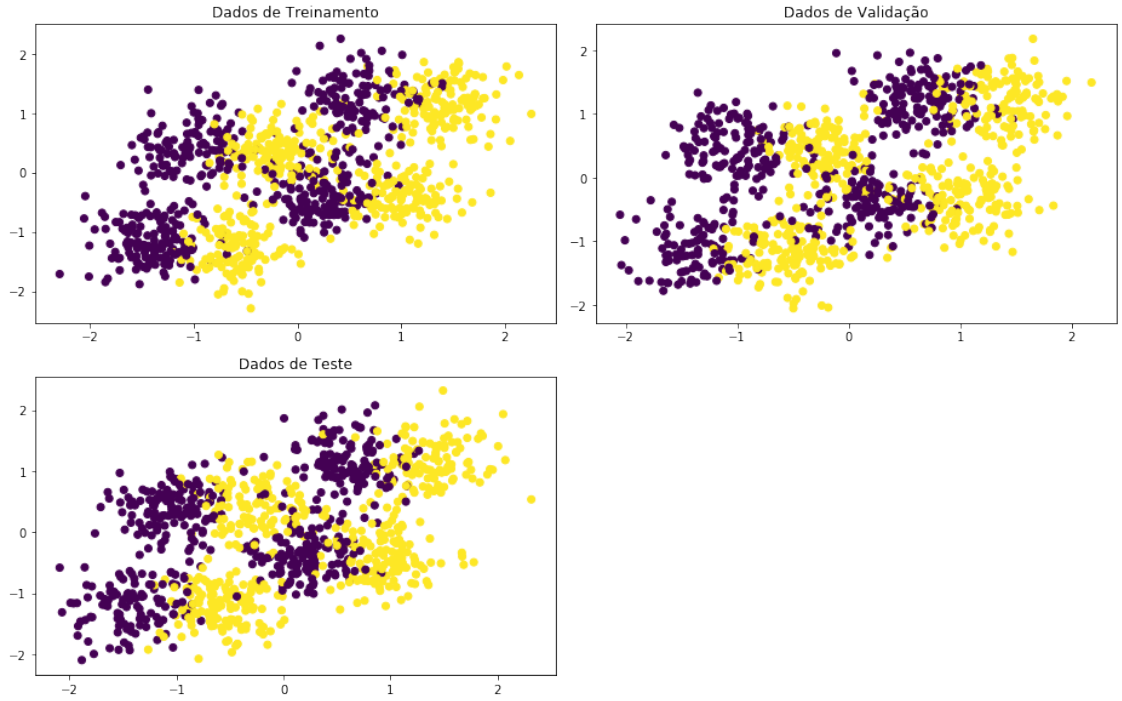
$\frac{\partial inpZ}{\partial v_1 2} = x_1$

Finalmente:

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

1.1.2 Parte 2 - Classificação binária com redes MLP e SVMs

Apresentação dos dados, tanto de treinamento, validação e testes.



a) Nesta questão fora utilizado duas implementações de uma rede MLP, tendo como base praticamente os mesmos parâmetros para comparação.

Na primeira implementação foi usada a MLP fornecida pelo pacote scikit-learn MLPClassifier, já na segunda, o pacote Keras foi selecionado e uma rede MLP com as mesmas características que a anterior foi criada.

Multi-Layer Perceptron (MLP) *1. MLPClassifier do scikit-learn*

Como mencionado a primeira implementação foi do scikit-learn. No caso a biblioteca jáá traz alguns hiperparâmetros ótimos já pré-preenchidos, os parâmetros usados:

- Função de ativação: **ReLU**
- Qtd. Unidades (Neurônios): **100**
- Qtd. Camadas: **1**
- Qtd. Máx. Épocas: **500**
- Learning rate (α): **0.001**
- Algoritmo Iterativo (optimizer): **Adam**
- Batch size: **32**

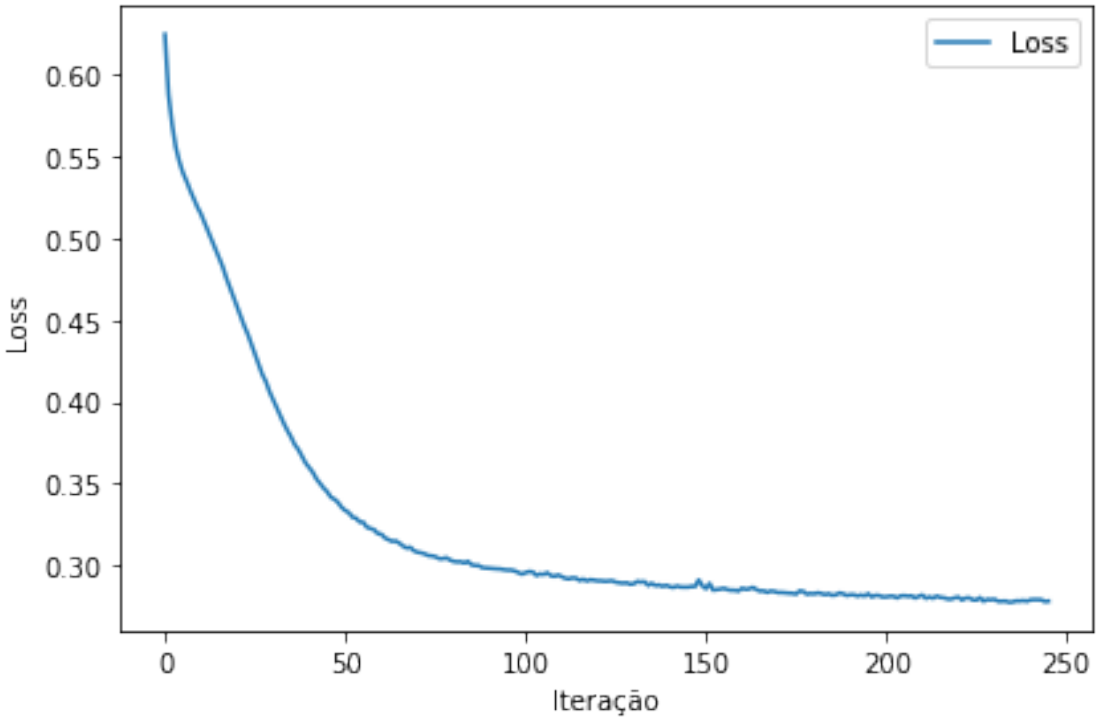
Todos os outros parâmetros não foram alterados.

Abaixo é apresentado todos os parâmetros que o classificador possui... claro, que conforme descrito na biblioteca alguns deles só são ativados caso da escolha de algum outro específico.

```
Out[7]: MLPClassifier(activation='relu', alpha=0.0001, batch_size=32, 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_state=1, shuffle=True, solver='adam', tol=0.0001,
validation_fraction=0.1, verbose=False, warm_start=False)
```

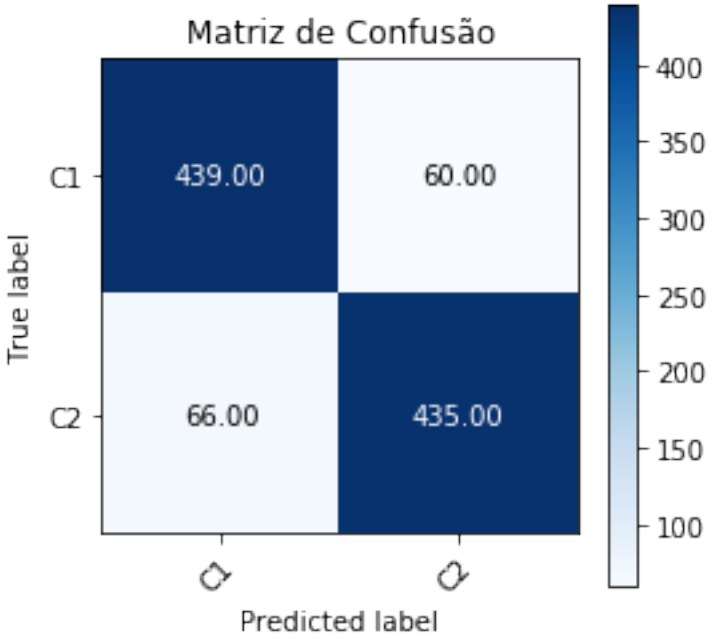
Ao realizar a execução, obteve-se a seguinte curva de erro:



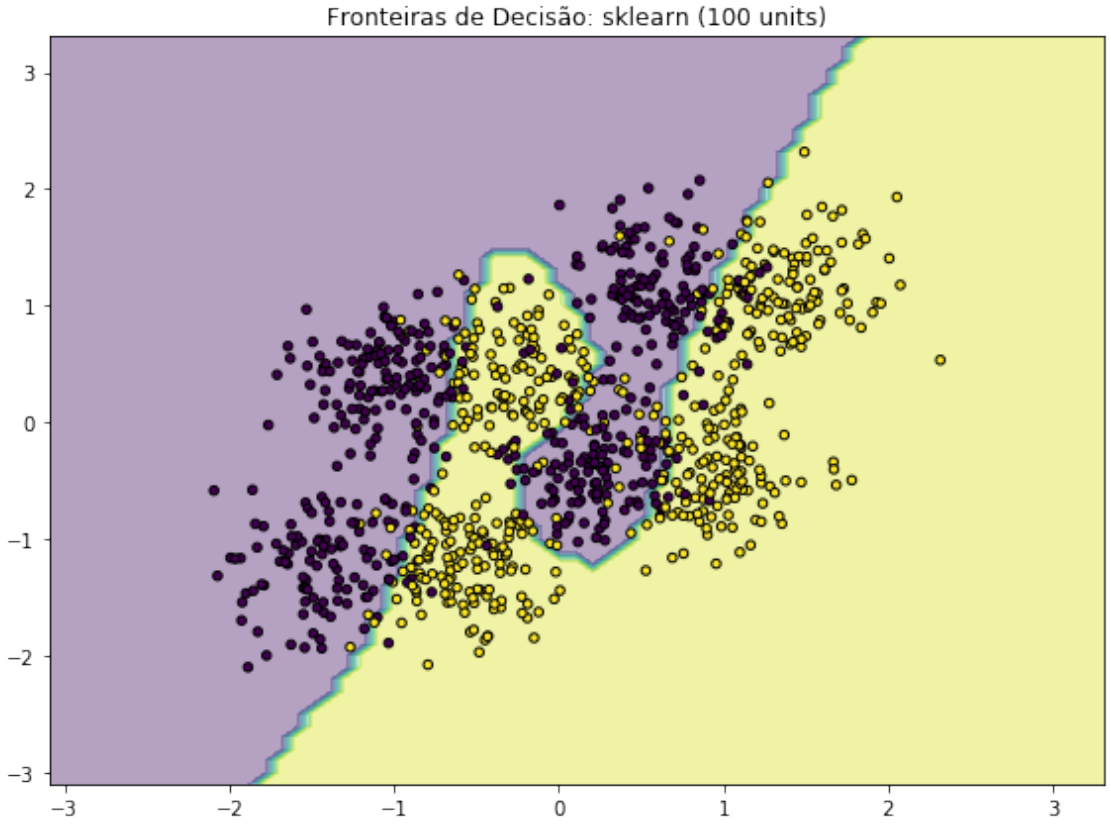
Consequente a isso, abaixo é apresentado o resultado da execução do classificador nos dados de teste.

```
Acurácia:
87.4%

Classification report:
              precision    recall  f1-score   support

          C1       0.87      0.88      0.87       499
          C2       0.88      0.87      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[9]: (1.5, -0.5)
```



As regiões de decisão do classificador MLPClassifier.



2. Rede Neural do Keras

Tendo em vista a real dificuldade de utilização do conjunto de validação (o qual fora disponibilizado) com o classificador da biblioteca do scikit-learn, optou-se também pelo uso (conforme mencionado) do framework Keras.

No caso da Rede Neural construída usando Keras, partiu-se do princípio de replicar a mesma estrutura com os mesmos parâmetos usados no classificador do scikit-learn.

Dessa maneira temos a seguinte estrutura da Rede:

```
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
```

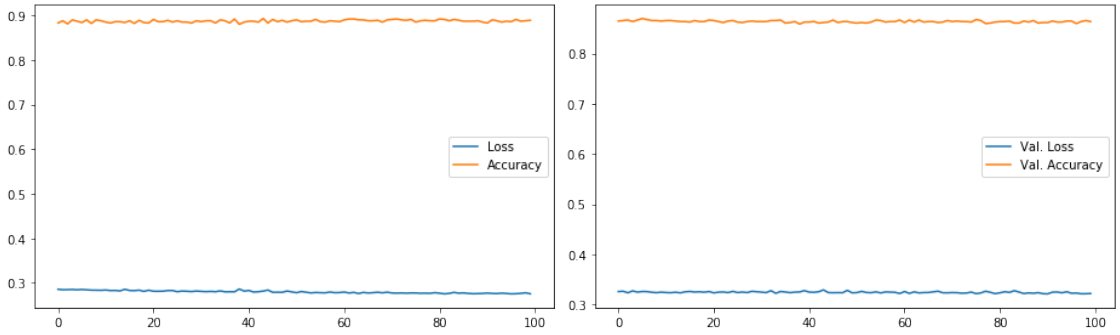
Conjuntamente com essa estrutura fora utilizado:

- Função de ativação: **ReLU**
- Qtd. Unidades (Neurônios): **100**
- Qtd. Camadas: **1**
- Qtd. Máx. Épocas: **100**
- Learning rate (α): **0.001**
- Algoritmo Iterativo (optimizer): **Adam**
- Dados embaralhados: **Sim**
- Batch size: **32**
- Camada de saída: **2 unidades e softmax**

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).

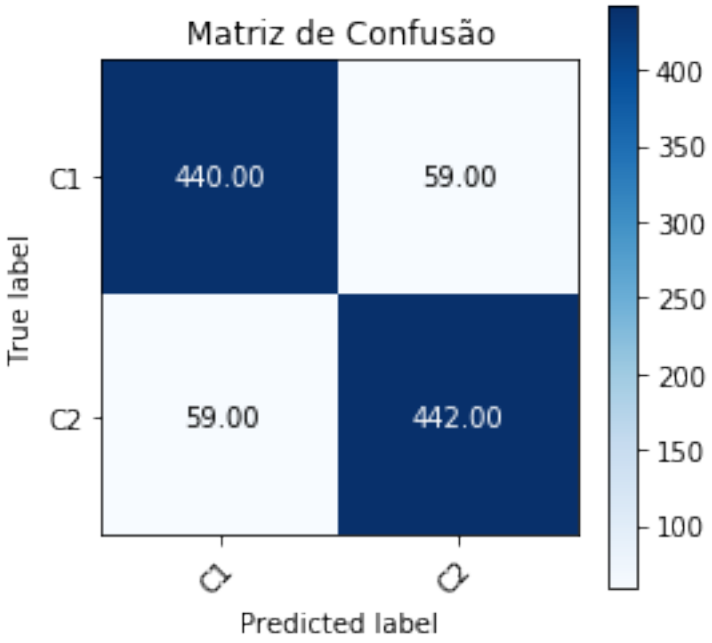
No caso do framework keras existe a possibilidade de avaliação do conjunto de validação conforme o andamento do aprendizado da Rede.

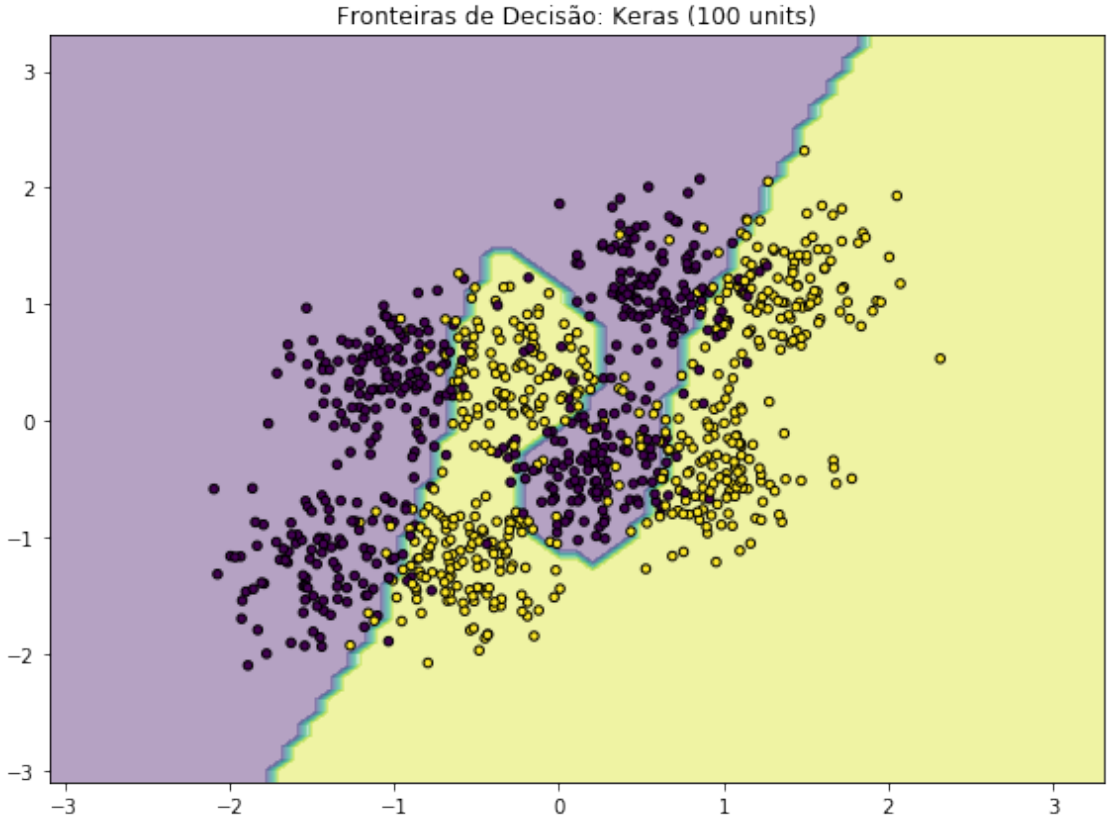
Os gráficos abaixo apresentam a Loss / Accuracy para os dados de treinamento e validação ao longos das épocas.



E como anteriormente os dados referêntes a execução do modelo nos dados de testes.

```
Acurácia:
88.2%

Classification report:
              precision    recall  f1-score   support

          C1       0.88      0.88      0.88       499
          C2       0.88      0.88      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[22]: (1.5, -0.5)
```


Matriz de Confusão
440.00
59.00
59.00
442.00
C1
C2
True label
Predicted label
400
350
300
250
200
150
100


Fronteiras de Decisão: Keras (100 units)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

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).

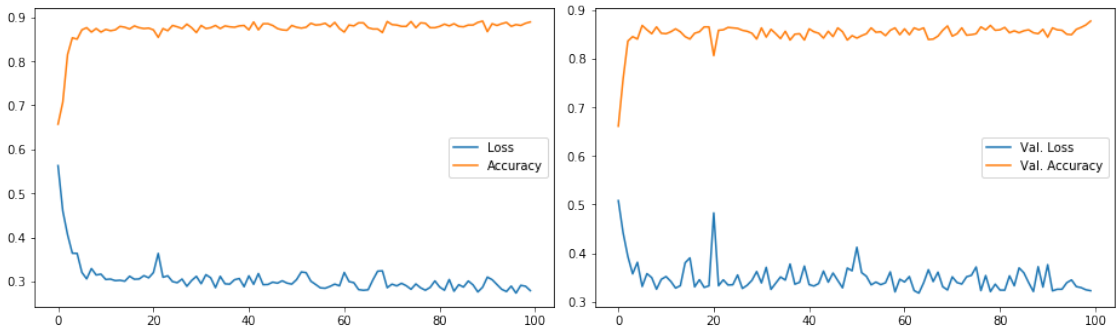
Será portanto utilizada a Rede do Keras para experimentar o uso de mais unidades (neurônios na camada intermediária). Para teste (e pensando na questão de uma Rede Neural ser um Aproximador Universal), aumentou-se de maneira relativamente expressiva a quantidade de unidades da camada intermediária para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

O valor 32768 fora escolhido por ser múltiplo de 2 (2^{15}) e representar uma alta quantidade de neurônios.

Todos os outros parâmetros se mantiveram os mesmos.

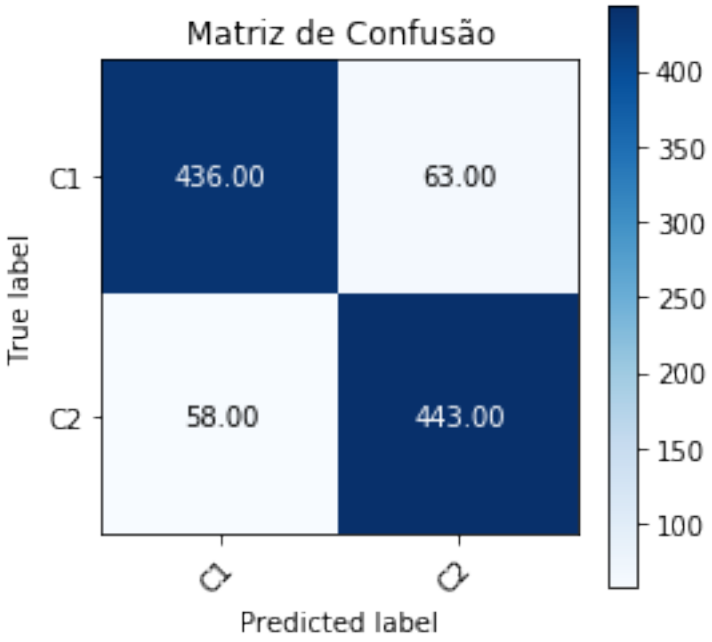
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer (Dense)          (None, 32768)             98304
_________________________________________________________________
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
```

Abaixo, os gráficos apresentando as curvas de Loss / Acurácia para os dados de treinamento e validação.

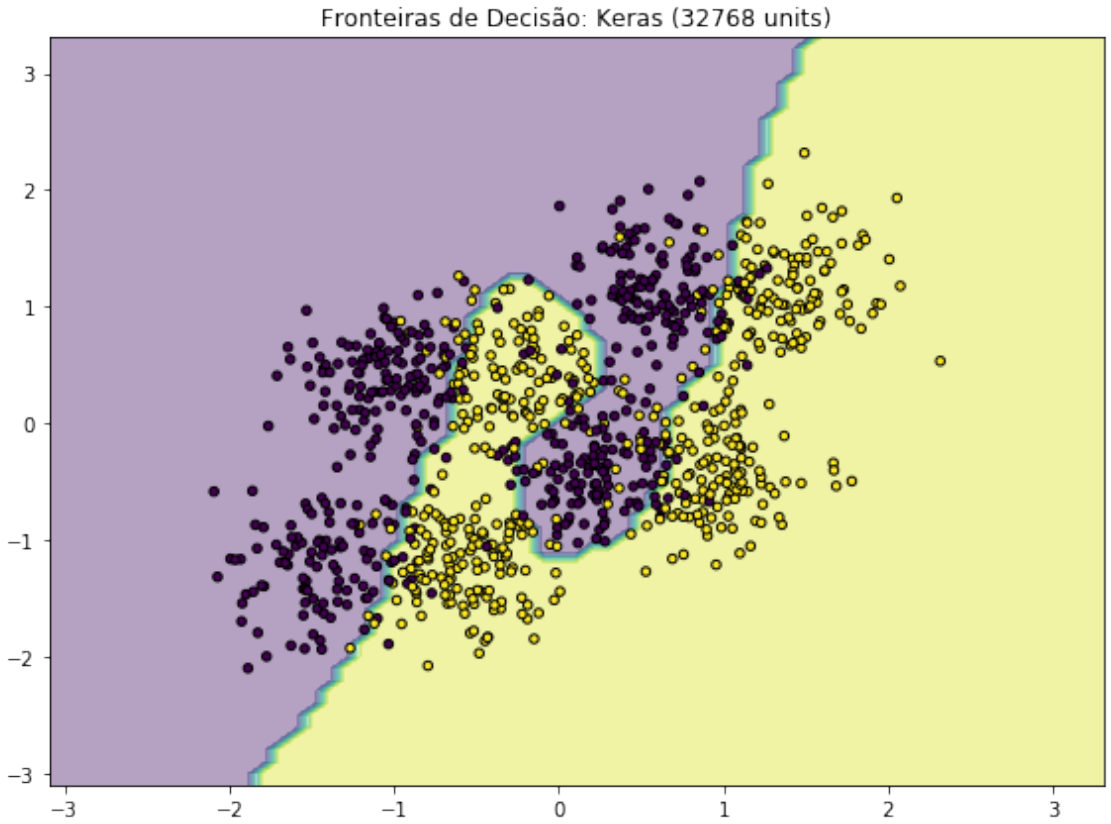


```
Acurácia:
87.9%

Classification report:
              precision    recall  f1-score   support

          C1       0.88      0.87      0.88       499
          C2       0.88      0.88      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[27]: (1.5, -0.5)
```



Por fim, as fronteiras de decisão geradas pela Rede com 32768 unidades.



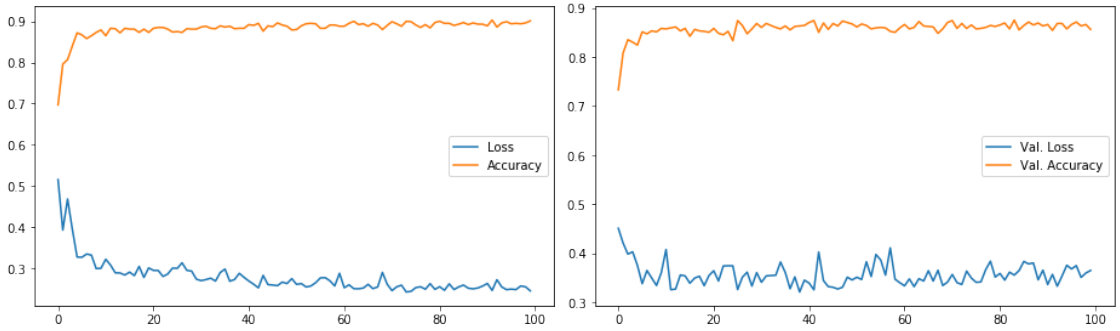
Nota-se que o aumento na quantidade de unidades (neurônios) para a mesma arquitetura de rede, na verdade adicionou mais instabilidade para o algoritmo tentar aproximar melhor a função.

Tendo em vista essa dificuldade da Rede, a mesma fora reformulada para a seguinte arquitetura:

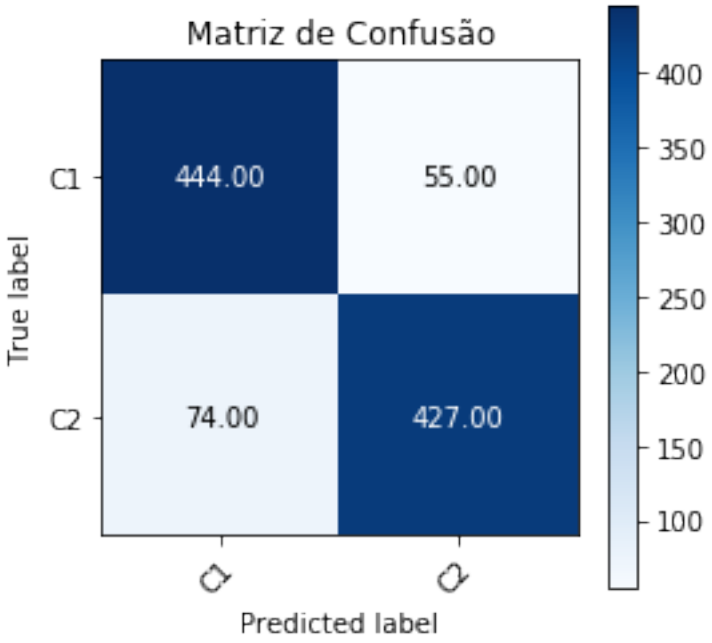
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer_1 (Dense)        (None, 1024)              3072
_________________________________________________________________
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
```

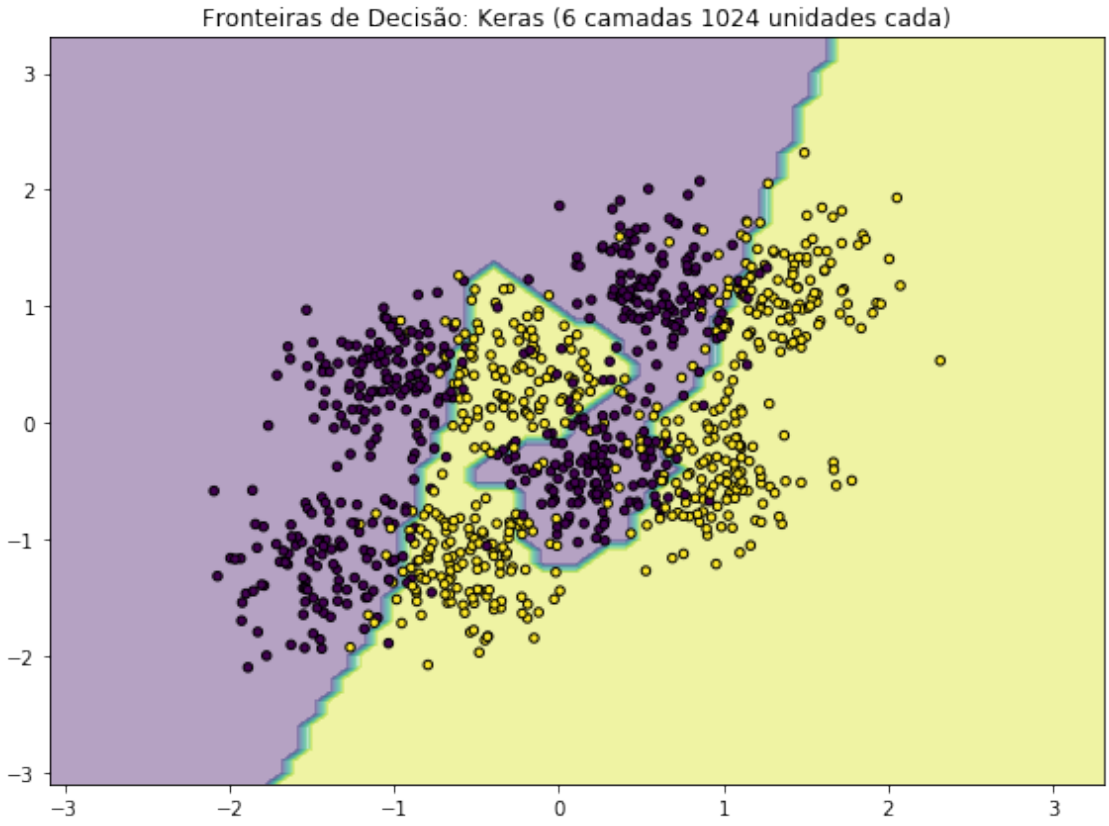
```
=================================================================
Total params: 5,253,122
Trainable params: 5,253,122
Non-trainable params: 0
_________________________________________________________________
None


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

Abaixo segue os resultados... apresentados na mesma ordem que os anteriores.




```
Acurácia:
87.1%

Classification report:
              precision    recall  f1-score   support

          C1       0.86      0.89      0.87       499
          C2       0.89      0.85      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[35]: (1.5, -0.5)
```

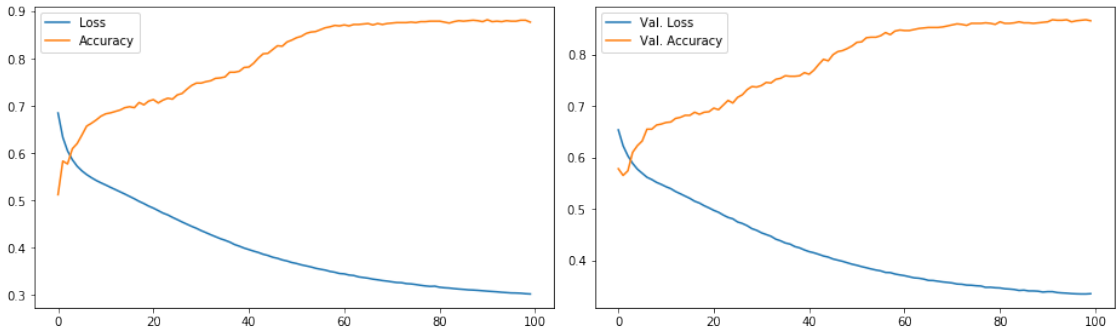

Matriz de Confusão
444.00
55.00
74.00
427.00
C1
C2
True label
Predicted label
400
350
300
250
200
150
100


Fronteiras de Decisão: Keras (6 camadas 1024 unidades cada)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

Visto que aumentar a quantidade de unidades (neurônios) e a quantidade de camadas intermediárias não ajuda a resolver o problema, fica aqui demonstrado a quantidade mínima de unidades (neurônios) necessários para a solução do problema.

Faz-se uso do framework Keras.

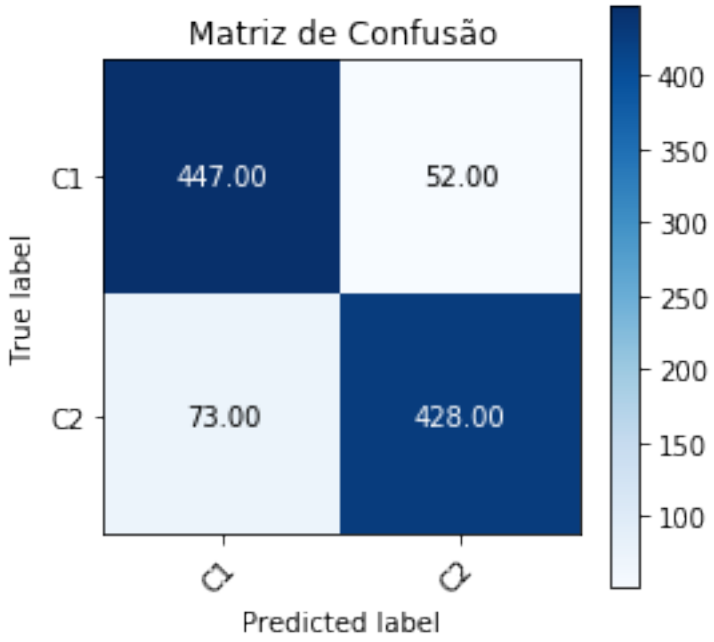
```
Model: "Multi Layer Perceptron"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
Input_Layer_1 (Dense)        (None, 30)                90
_________________________________________________________________
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
```

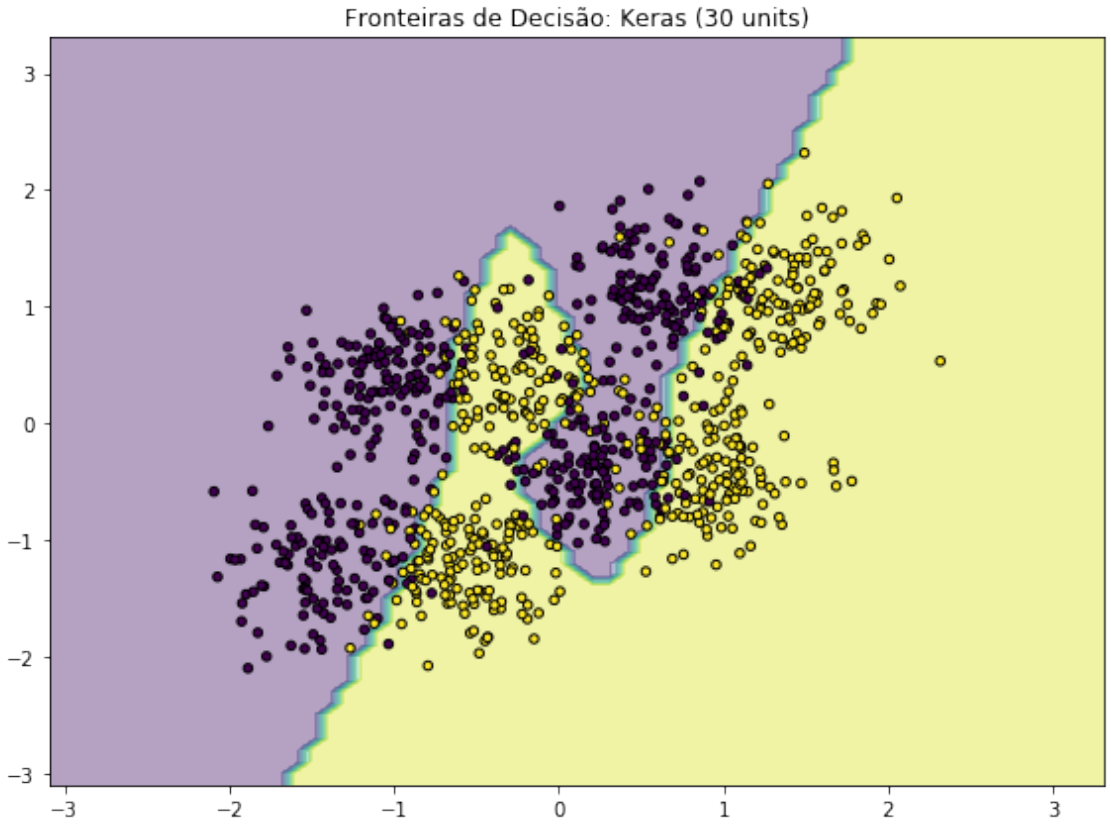



```
Acurácia:
88.3%

Classification report:
              precision    recall  f1-score   support
```

```
              C1       0.88      0.89      0.88       499
              C2       0.89      0.88      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[40]: (1.5, -0.5)





Portanto, como demonstrado acima, uma quantidade de apenas 30 unidades (neurônios) já é capaz de resolver o problema com acurácia de >= 87%.

Support Vector Machine (SVM) Para o caso do SVM, foi utilizado o kernel **rbf (Radial Basis Function)** o qual pode ser considerado uma aproximação de uma Rede Neural. Isso fica nítido no resultado apresentado.

Posteriormente, para fim de análise comparativa foi realizado o uso de um kernel polinomial de grau 3.

Além disso o valor hiperparâmetro *C* utilizado foi de 5. Nos testes realizados com o SVM esse valor foi médio, trazendo bons resultados e boa performance para treinar.

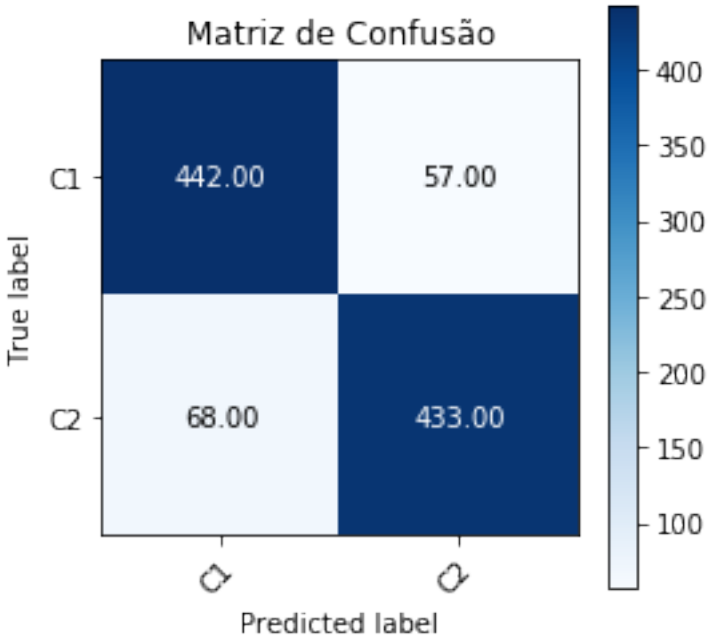
Abaixo segue as informações da execução do SVM usando *rbf* e *C=5*.

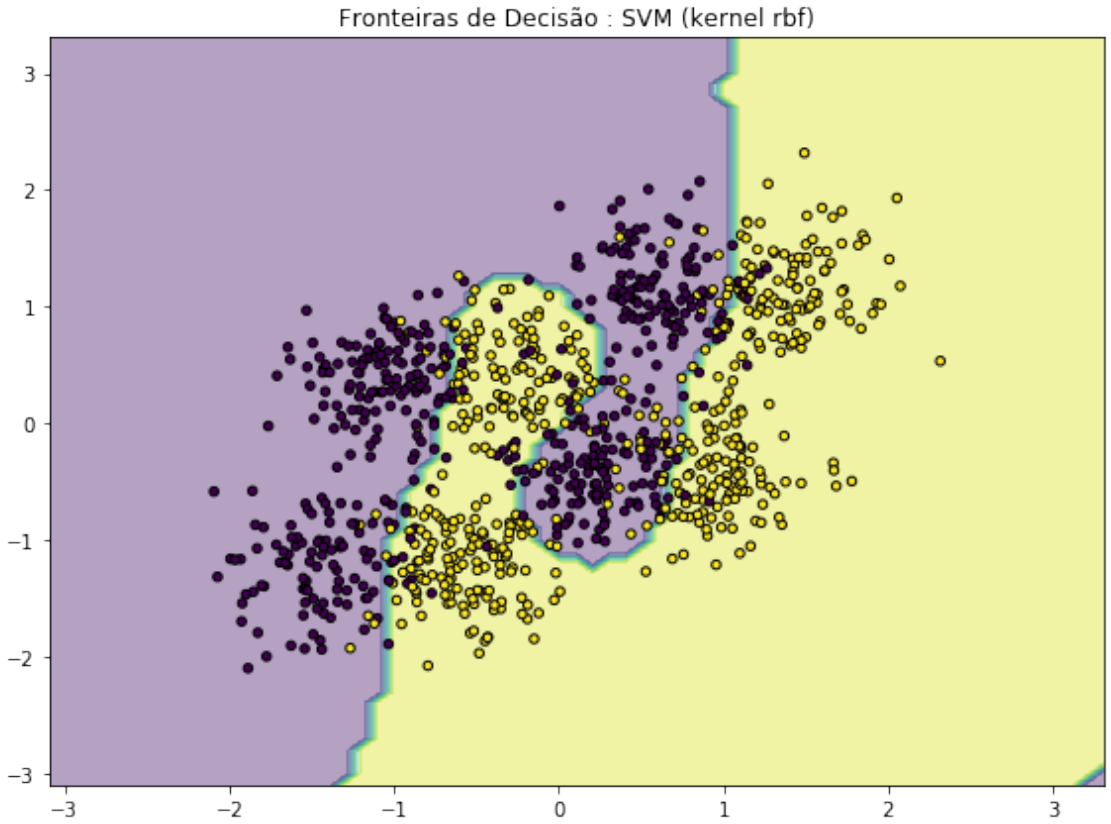
P.S.: Não foi alterado o parâmetro *gamma*.

```
Out[43]: 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[44]: (1.5, -0.5)





Conforme mencionado o kernel *rbf* com *C=5* trouxe uma acurácia próxima (se não, identica ao que uma MLP conseguiu).

Dessa forma é factível concluir que ambos os algoritmos tendem a resolver o problema de forma relativamente igual.

Uma diferença visível que não é apresentada, é o tempo de treinamento de cada algoritmo. No caso do SVM ele tem um tempo de treinamento MUITO inferior ao da MLP (tanto usando scikit-learn quanto Keras).

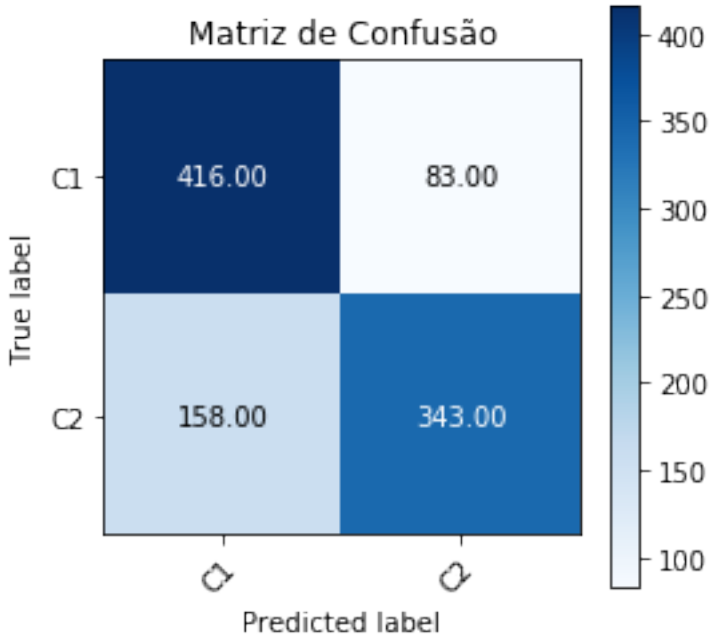
Além do kernel **rbf**, foi usado para fins de comparação um kernel polinomial com grau 3 (foi usado o mesmo valor de *C=5*).

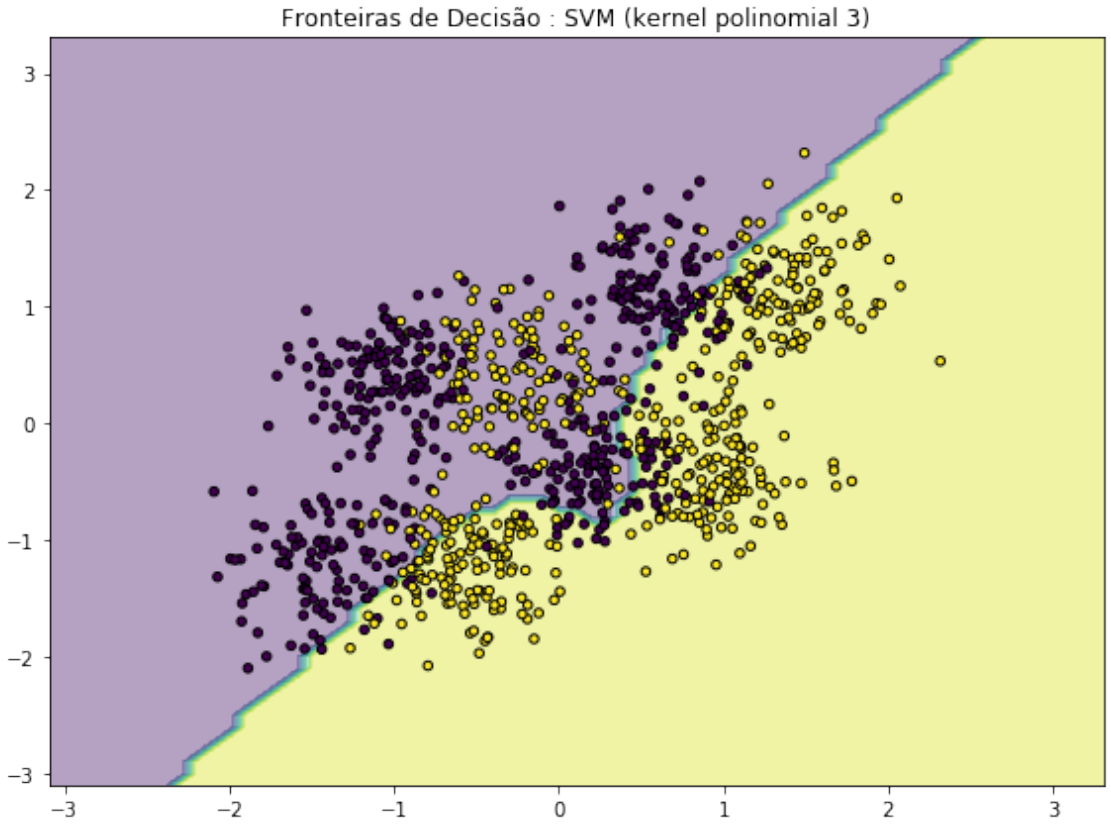
Abaixo, apresenta-se os resultados.

```
Out[46]: 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[47]: (1.5, -0.5)



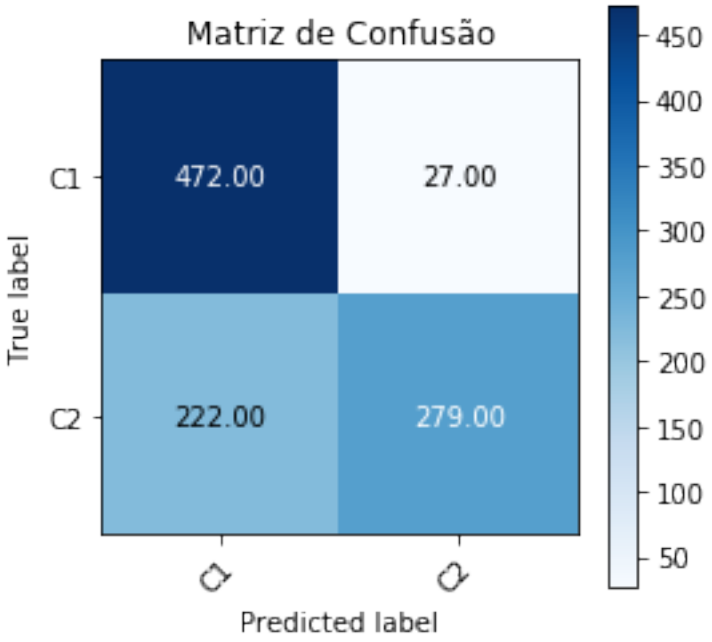


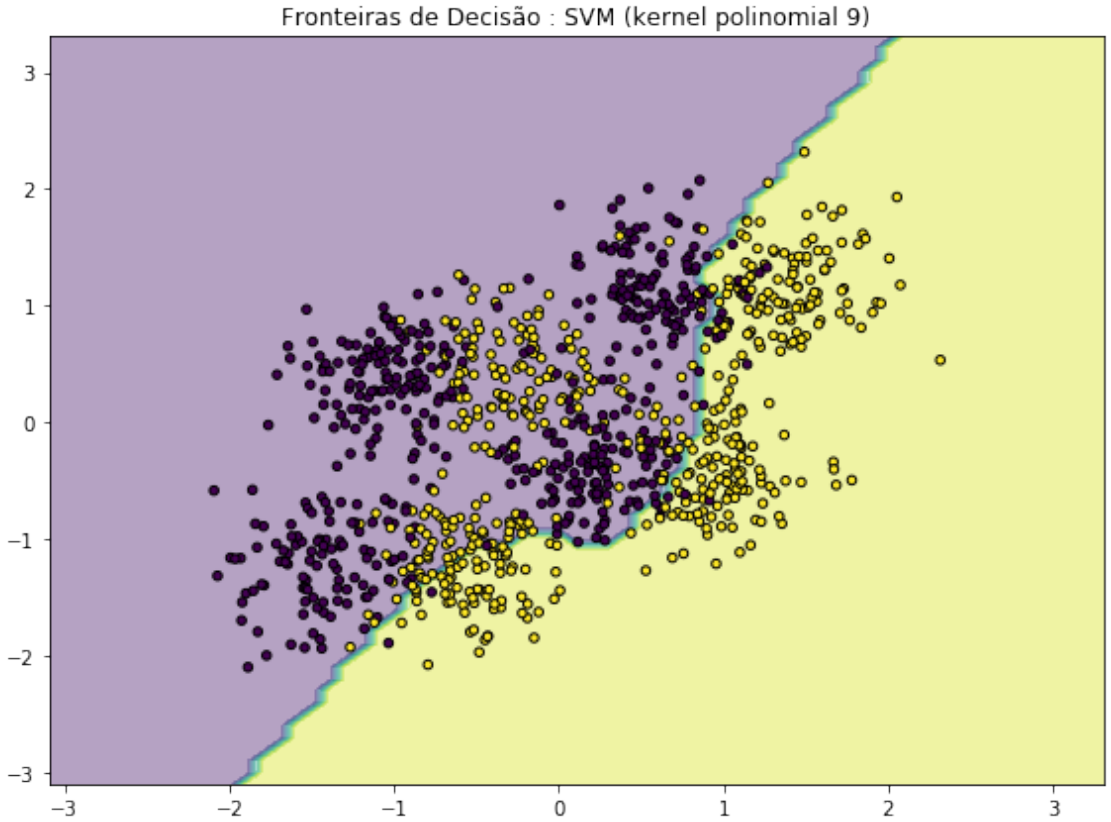
Adicionar mais graus ao kernel polinomial, não necessariamente aumenta a acurácia do modelo. Isso pode ser observado abaixo.

Foi utilizado o mesmo modelo SVM, com a execeção do uso de um polinômio de grau 9.

```
Out[49]: 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[50]: (1.5, -0.5)
```


Matriz de Confusão
C1
C2
472.00
27.00
222.00
279.00
True label
Predicted label
C1
C2
450
400
350
300
250
200
150
100
50


Fronteiras de Decisão : SVM (kernel polinomial 9)
3
2
1
0
−1
−2
−3
−3
−2
−1
0
1
2
3

Considerações gerais Percebemos que para o problema proposto, ambos algoritmos tiveram desempenho próximo (em torno de 87~88% de acurácia). A única diferença expressiva que percebemos foi a vantagem no tempo de treinamento do algoritmo SVM em relação a rede MLP.

Especificadamente falando da rede MLP, ficou claro que o aumento no número de camadas e o aumento no número de neurônios das camadas internas não reflete necessariamente na melhoria do resultado final. A rede possui um valor otimizado de neurônios e camadas internas para um determinado problema (no caso apresentado 1 camada com 30 neurônios) e quaisquer valores acima do ótimo representam inserção de ruído e desperdício computacional. Percebemos que a definição de outros hiperparâmetros (como otimizadores) são bem vindos pois colaboram para outros ganhos além da acurácia final, como por exemplo um tempo menor de convergência.

Em relação ao modelo SVM vimos que utilizar o truque do kernel de fato viabiliza o modelo para problemas não lineares, onde os resultados encontrados se assemelham muito com os da rede neural. Assim como na questão da rede neural, o algoritmo também apresenta valores otimizados para o problema proposto e o aumento destes hiperparametros de kernel e regularização além do ótimo não representa melhoria de desempenho.

NOTAS

- Uma maneira utilizada para encontrar alguns dos hiperparâmetros utilizados foi executar uma busca exaustiva avaliando diferentes valores. No caso específico a biblioteca scikit-learn possui o método GridSearchCV, o qual tem a seguinte explicação: ***Exhaustive search over specified parameter values for an estimator.***
- O algoritmo **Adam** foi usado como otimizador da Rede Neural devido a sua rápida convergência, apesar de ter sido demonstrado que o Gradiente descendente pode ter um menor erro.
- On the Convergence of Adam and Beyond
- The Marginal Value of Adaptive Gradient Methods in Machine Learning
- A escolha pelo parâmetro de *batch_size=32* vem de um paper indicado pelo **Yann LeCun** (Friends dont let friends use minibatches larger than 32). Entretanto é discutido em outro paper que o decay do batch_size (de algo grande para cada vez menor) pode também ajudar em uma melhor convergência.

Anthony Miranda Vieira - 229058
Rodolfo De Nadai - 208911
Todo o código deste relatório esta disponível em: https://github.com/rdenadai/ia006c