Intro. Machine Learning - Tarefa 3

UFRJ

Aluno: Matheus Oliveira Silva

DRE: 119180151

Sumário

Considerações do relatório

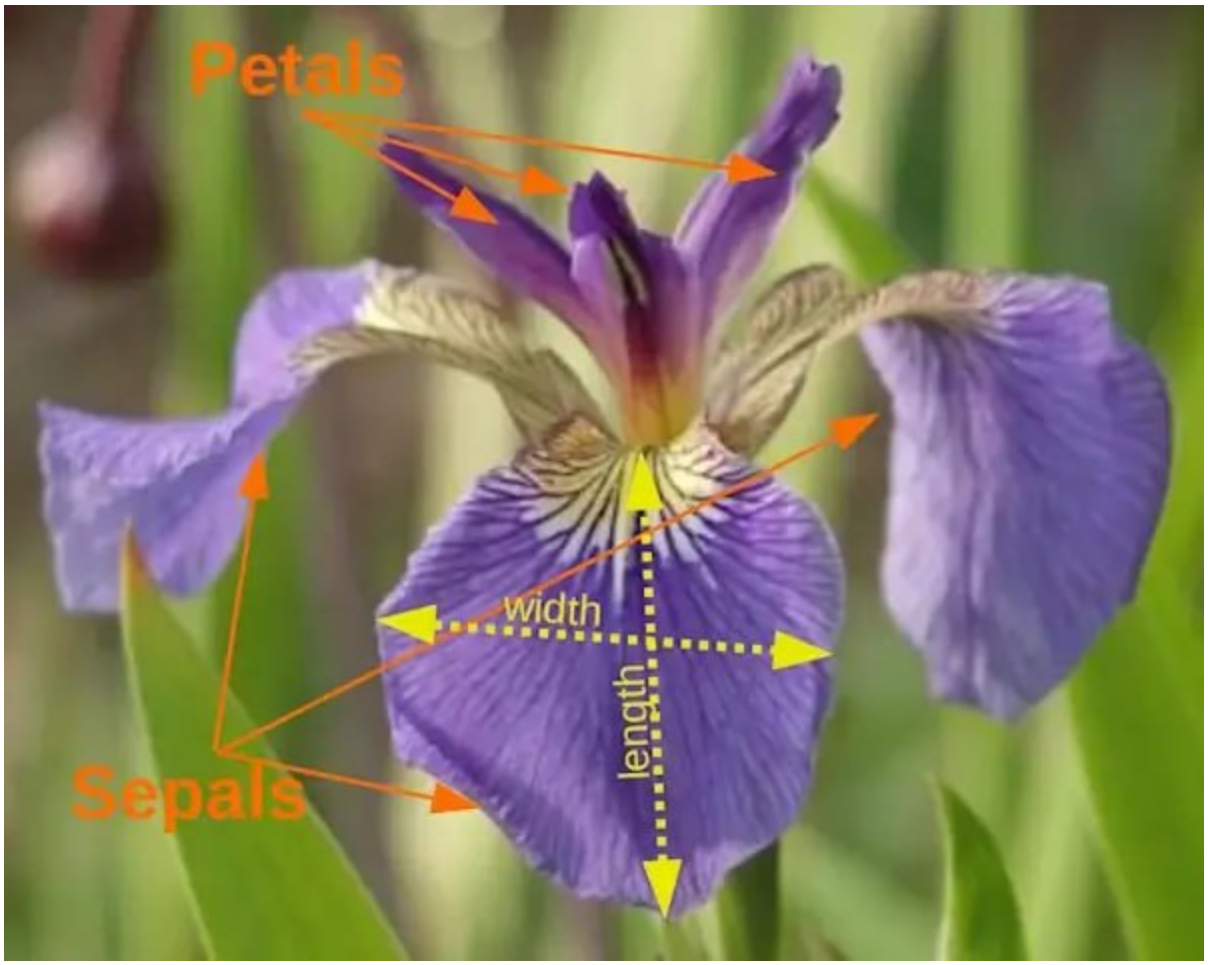
O código do notebook em que fiz esse trabalho se chama “tarefa3.ipynb” e está na pasta “código/scripts” do zip onde se próprio relatório se encontra.

Meu material de estudo para desenvolvimento do trabalho foi os slides do professor, e meus resumos especiais que fiz adicionando comentários extras em algumas ocasiões, porque slides não contém a informação completa, precisa unir com o que o professor explica.
Você pode encontrar o link para esse compilado de resumos sobre a matéria de introdução ao Machine Learning aqui .

Uma coisa importante sobre esse trabalho é que fiquei em duvida se eu fazia uma análise completa ou ia direto ao ponto e só fazia o que foi pedido de forma rápida, por isso fiz um sumário, então o leitor consegue ir direto ao ponto que interessa ele a qualquer momento.

Contextualização do problema e tratamento de outliers

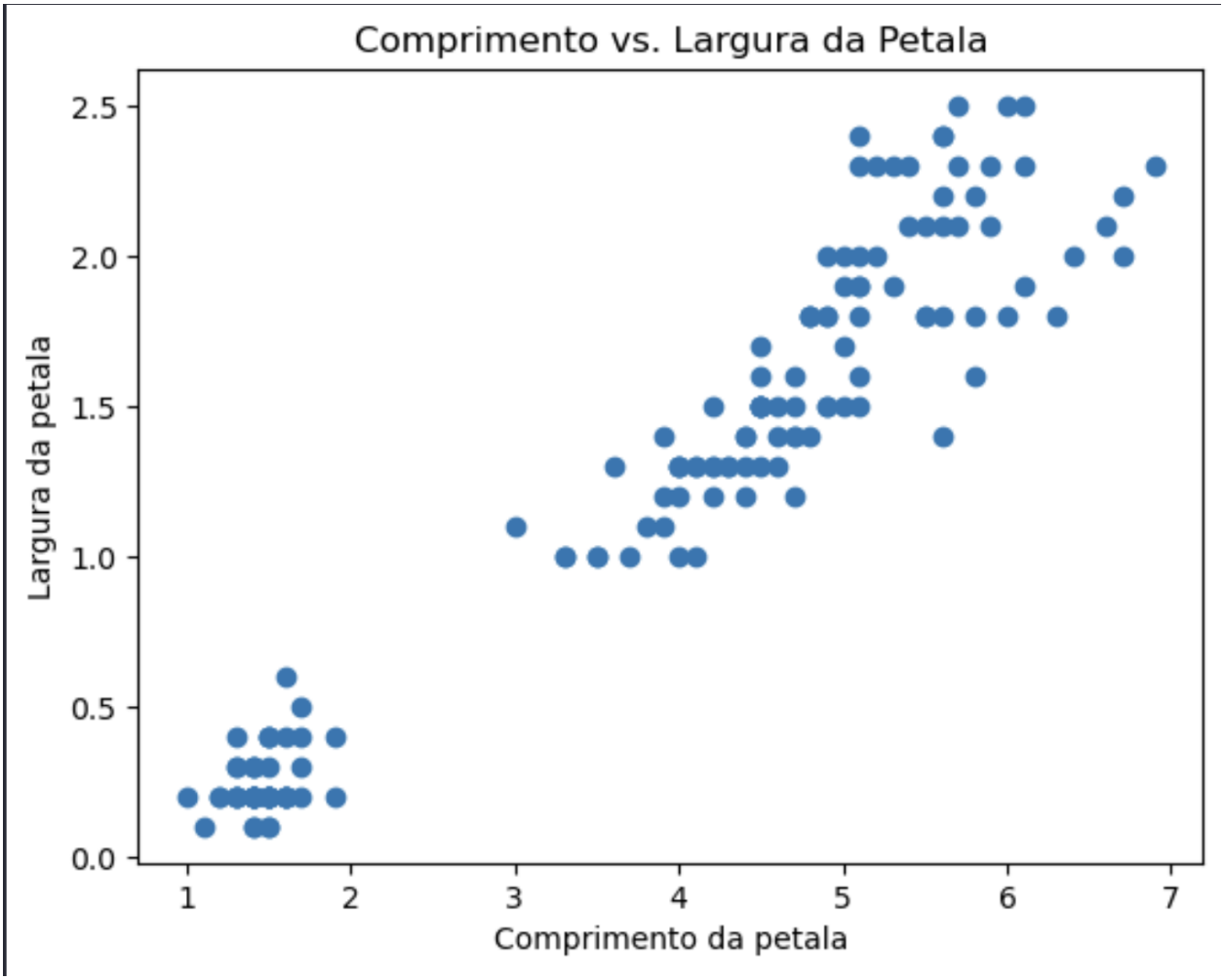
Abaixo está a imagem que resume bem o que estamos analisando no nosso dataset, imagem que o monitor Marcos botou no grupo de Intro. ao M. L. do telegram, a figura nos mostra o que são pétalas e sépalas, e também explica qual a referência de medida de comprimento e largura da Iris, e é em relação a “base” dela que vem do cabo dela.



Quanto aos outliers, foi estritamente pedido pelo professor para utilizar todas as 150 instâncias do conjunto de dados da Iris da biblioteca scikit-learn do Python, por isso não iremos fazer tratamento de outliers, justamente para manter todas as 150 instâncias.

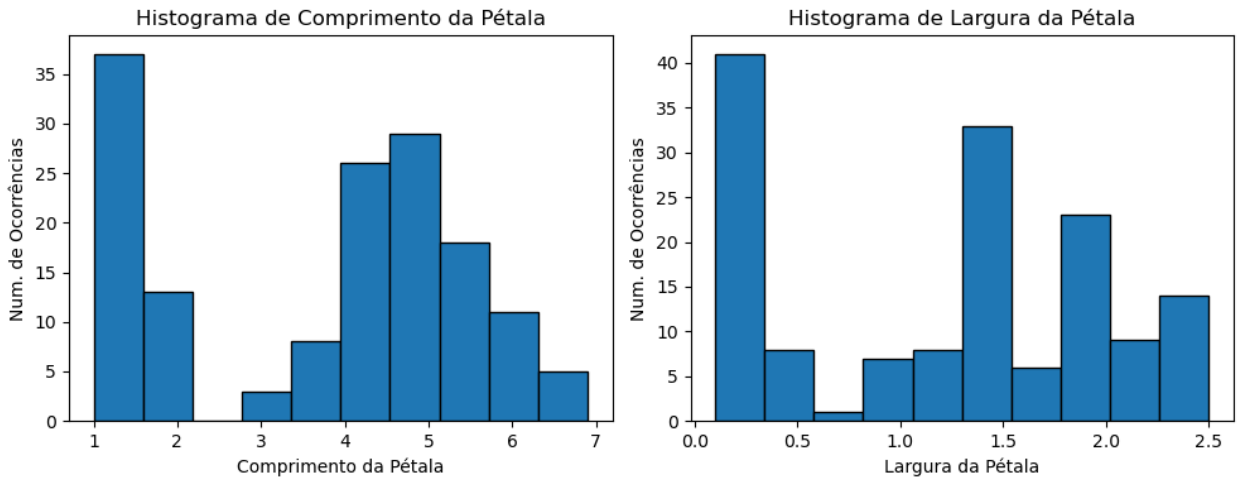
Plots iniciais para insights ingênuos e correlação

Primeiro eu procurei plotar um gráfico de comprimento vs. largura das pétalas, para ver se era possível observar padrões.

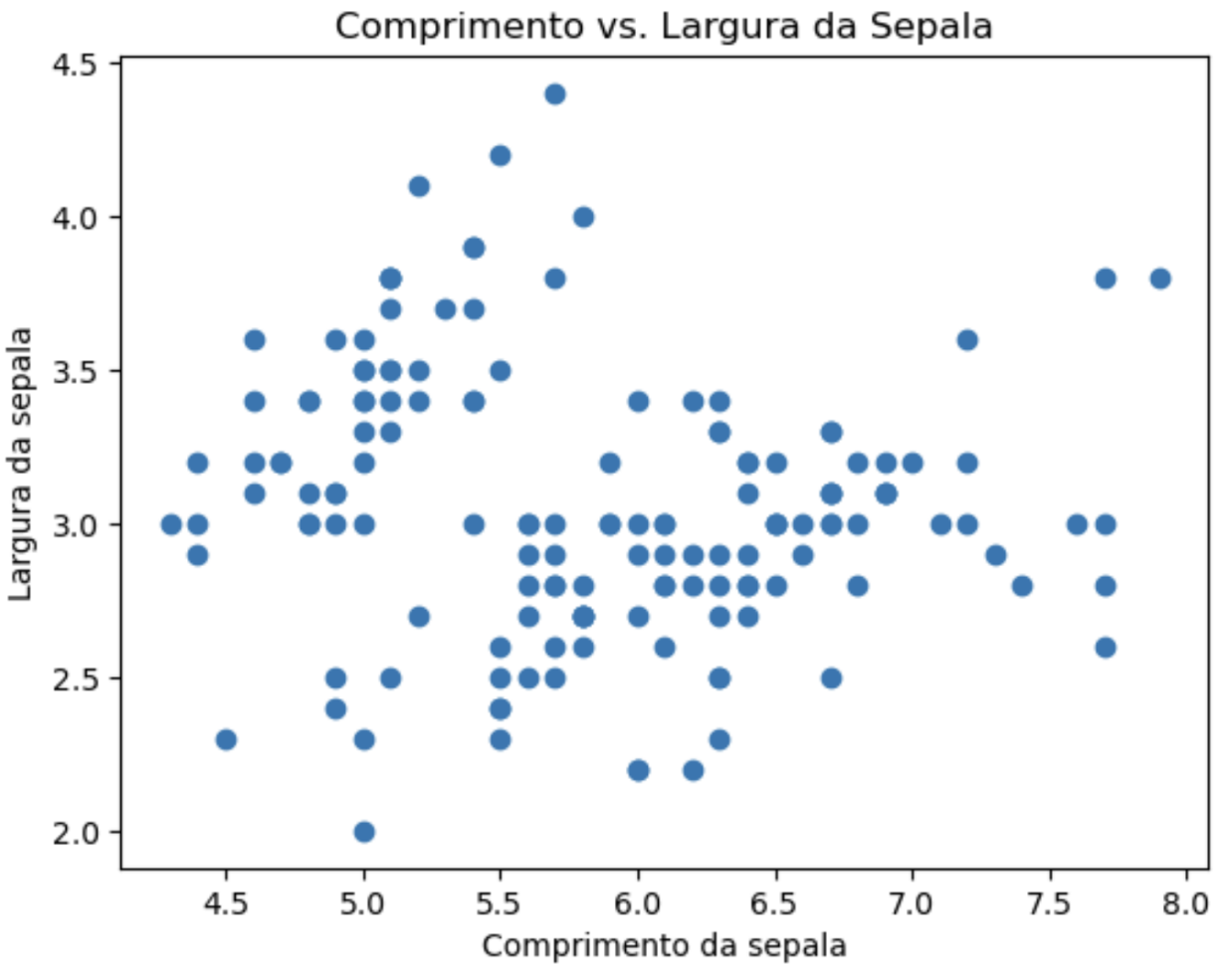


Isso honestamente me deixou bem calmo, por que da para ver uma crescimento linearmente positivo, ou seja, quando vemos crescendo o tamanho de uma variável, a outra acompanha, e isso se estende por todo conjunto de dados, sem outliers muito absurdos. Outra coisa que já é possível observar é um cluster bem pequeno nos valores de menor comprimento e largura, com certeza acharemos algo por ali, mas não podemos concluir nada ainda porque estamos apenas observando duas variáveis, dentre as quatro que temos, mas é um bom plot ingênuo.

Agora vou plotar o histograma de frequência de valores de tamanho de comprimento e largura das pétalas, onde não vemos nada muito diferente do que já visualizamos no plot anterior.

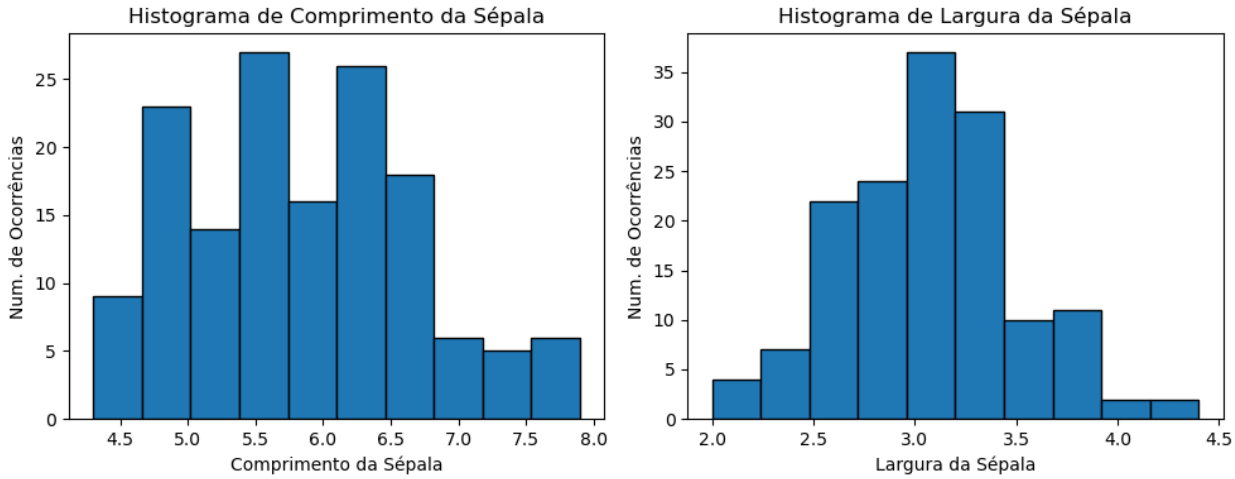


Podemos agora tentar plotar o comprimento vs. largura das sépalas:



Parece bem confuso, claramente não segue nenhuma distribuição conhecida, ou melhor, facilmente identificável. Conseguimos até reconhecer uns padrões, ver que temos agrupamentos por volta de alguns valores mas nesse caso não está claro com certeza nenhuma outra informação. Mas faz parte, por isso devemos testar.

Agora vou plotar o histograma de frequência de valores de tamanho de comprimento e largura de sépalas, onde não vemos nada muito diferente do que já visualizamos no plot anterior.



O único ponto engraçado é que no caso das pétalas, é mais fácil visualizar um padrão olhando um gráfico de dispersão, com as sépalas fica mais fácil observar o histograma de largura das sépalas, que apresenta uma distribuição quase que com formato de uma normal.

Correlação

	sepal length (cm)	sepal width (cm)	petal length (cm)	petal width (cm)	Target
sepal length (cm)	1.000000	-0.117570	0.871754	0.817941	0.782561
sepal width (cm)	-0.117570	1.000000	-0.428440	-0.366126	-0.426658
petal length (cm)	0.871754	-0.428440	1.000000	0.962865	0.949035
petal width (cm)	0.817941	-0.366126	0.962865	1.000000	0.956547
Target	0.782561	-0.426658	0.949035	0.956547	1.000000

Vendo nossa tabela de correlações, podemos ver que todos atributos exceto sepal width tem uma correlação boa com o nosso target, e isso talvez seja um indicativo de que devemos excluir esse atributo do nosso dataframe diminuindo a dimensão do problema. Mas se pensarmos bem, temos 3 tipos de targets, então usar correlação para tomar uma decisão desse tipo é meio precoce, pode ser que para alguma das classes de Iris, seja importante a informação que estamos tirando, então quanto a atributos, manteremos todos.

Preparando o problema

a- Separando 10% do dataset (15 instâncias, com 5 de cada classe)

Atendendo o que nos foi pedido, bastou utilizar o parâmetro `stratify=iris.target` que nos garante proporção das classes, como queremos justamente 5 de cada, com 15 instâncias e 3 classes, estamos pedindo justamente isso, que nosso conjunto de teste tenha a mesma proporção das classes.

Então esse primeiro pedido do professor conseguimos atender com uma linha de código:

```
X_train, X_test, y_train, y_test = train_test_split(
                                    iris.data,
                                    iris.target,
                                    test_size=0.1,
                                    stratify=iris.target
                                   )
```

b- Usando K-Fold e treinando o modelo e calculando acurácia de cada fold

No meu notebook, primeiro começo definindo as variáveis para atender o que foi pedido pelo professor

```
k = 5  # Número de folds
max_iterations = 500
hidden_layer_sizes = 2
solver = 'adam'
activation = 'identity'

kf = KFold(n_splits=k, shuffle=True, random_state=42)
```

Fiz isso para ficar bem fácil de modificar os valores depois para outros testes caso eu queira mudar.
Bem, para explicar como eu usei o K-Fold, basta explicar o que é K-Fold, ele se trata de uma técnica de validação cruzada, ou seja, dividiremos o conjunto de dados em K partes (folds) de tamanhos iguais. Em seguida, o algoritmo é treinado e avaliado K vezes, cada vez usando uma parte diferente como conjunto de testes e as demais

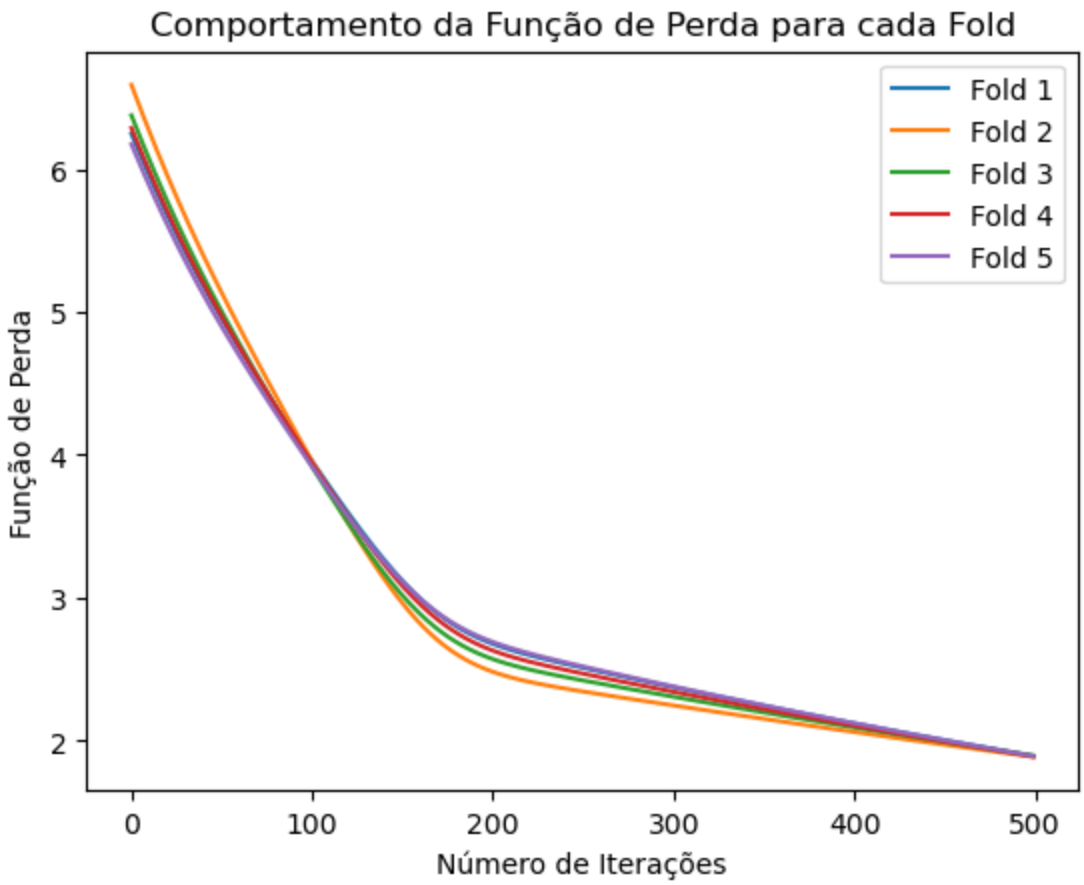
partes como conjunto de treinamento. O resultado final é a média das métricas de avaliação obtidas em cada iteração.

Dito isso, e já tendo criado o objeto `kf` com k=5 , eu vou usar o método`kf.split(X_train)`que vai criar 5 subconjuntos de index para treino e validação escolhido aleatoriamente (que no nosso caso sempre será igual quando rodarmos por que para motivo de testes simples eu setei a seed na última linha do trecho de código acima com `random_state=42`).

Dentro do código do for o que é feito basicamente é usar cada um desses 5 subconjuntos e criamos o objeto de Multi-Layer Perceptron Classifier usando a função `MLPClassifier( )` com os parâmetros que setamos lá encima conforme pedido pelo professor. Uma fez criado o objeto, treinamos o modelo usando as funções de `X_fold_train` e `y_fold_train` que serão usadas no método model.fit( ) e depois disso o que fazemos é calcular o score do treino e do teste, e esses valores são guardados para analisarmos nosso modelo.

c) Funções de perda e Análise da acurácia

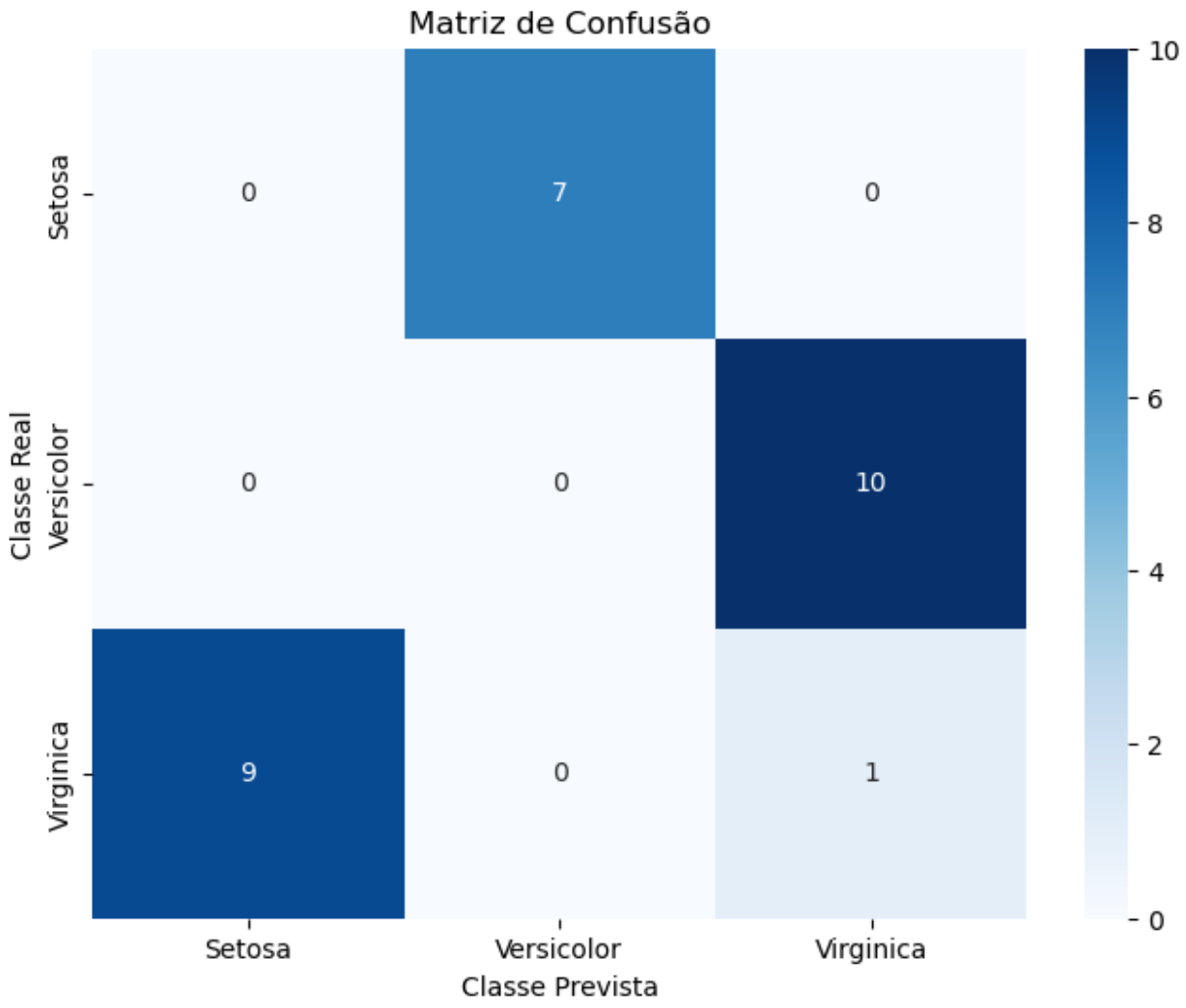
Eu fiquei bastante satisfeito olhando o plot das minhas funções de perda todas juntas convergindo para um número baixo, segue abaixo a imagem:



Mas depois de analisar o output dos scores não fiquei feliz. De início eu não sabia se era um erro do meu código mas eu fui observar os resultados dos scores dos meus modelos e obtive scores terrívels (considerando que é % de acerto e todos estavam mais próximos de 0 do que de 1).

```
Fold 1:
Score de Treinamento: 0.0555555555555555
Score de Teste: 0.0740740740740407

Fold 2:
Score de Treinamento: 0.03703703703703035
Score de Teste: 0.0

Fold 3:
Score de Treinamento: 0.19444444444444445
Score de Teste: 0.1111111111111111

Fold 4:
Score de Treinamento: 0.12037037037037036
Score de Teste: 0.1481481481481481

Fold 5:
Score de Treinamento: 0.03703703703703035
Score de Teste: 0.03703703703703035
```

Fora a matriz de confusão que eu pego sempe do último fold só para ter uma noção de como ela está e essa ficou horrível, igual a print que mandei, acerta 1 de 27, acurácia de 0.037 :



Mas tudo bem, claramente vimos que não está bom, mas é erro meu? Ou tomamos decisões erradas quanto escolha de parâmetros?

Eu fui ler o resto da proposta do trabalho e comecei a entender a ideia dele, nós começamos usando uns parâmetros que passam longe de serem bons. Fui procurar na documentação do sklearn e vi que :

- `solver = 'adam'` : esse parâmetro se refere ao Stochastic Gradient-based Optimizer , e o que sabemos de gradientes estocásticos é que eles funcionam melhor em grandes conjuntos de dados, podem não ser a escolha mais eficiente para nosso dataset de 150 instâncias.
- `activation = 'identity'` : esse parâmetro faz com que tenhamos uma função de ativação $f(x) = x$ ela é útil para tarefas de regressão, mas pode não ser

adequada para tarefas de classificação, como é o nosso caso. O que me fez pensar que poderiamos experimentar outras funções de ativação que estão na documentação como 'relu' ou 'tanh' ou ‘logistic’ que pode ser uma opção para melhorar o desempenho do modelo.

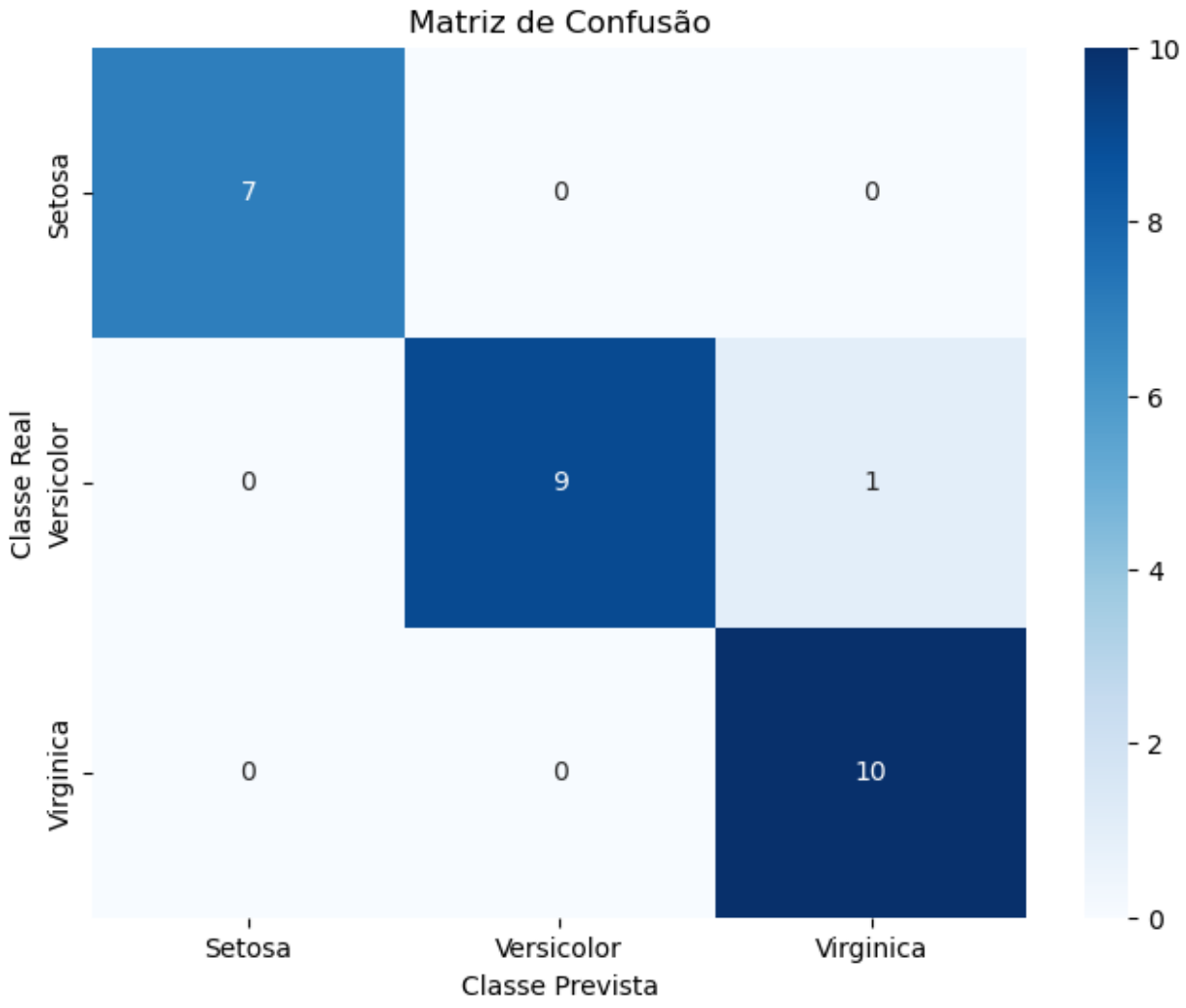
Esses foram os possíveis problemas que identifiquei de cara, porque são os mais gritantes, acredito que depois de modificar eles e ver o que acontece com o score do modelo podemos testar modificar outras coisas como o número de layers e número máximo de rodadas, mas minha intuição diz que devemos começar pelos dois, que teremos melhoras maiores.

Modificando os parâmetros

Observação inicial importante é que tive que mudar o meu código um pouco e quando faço uso do solver 'lbfgs' , não faço plot de função de perda porque o solver 'lbfgs' utiliza um método de otimização baseado em gradientes e aproximações da matriz Hessiana, que não possui a mesma interpretação de iterações e função de perda que os outros solvers. então por isso a gambiarra no meio do código aparace algumas vezes.

```
if solver in ['adam', 'sgd']:
  ... # do something
```

Outra coisa que tenho pra dizer é que eu não vou ficar repetindo print de matriz de confusão toda hora, matrizes boas de confusão vão se parecer com essa (LBFS + RELU) :



Quando as métricas de score vierem ruins, ai sim eu vou por a print da matriz de confusão para mostrar algum comportamento bizarro que esteja acontecendo.

LBFS:

LBFGS + ReLU

```
k = 5  # Número de folds
max_iterations = 500
hidden_layer_sizes = 2
solver = 'lbfgs'
activation = 'relu'

kf = KFold(n_splits=k, shuffle=True, random_state=42)

tarefa3(max_iterations, hidden_layer_sizes, solver, activation)
```

[119] ✓ 0.0s Python

```
Fold 1:
Score de Treinamento: 0.9907407407407407
Score de Teste: 0.9259259259259259

Fold 2:
Score de Treinamento: 0.9722222222222222
Score de Teste: 1.0

Fold 3:
Score de Treinamento: 0.9814814814814815
Score de Teste: 1.0

Fold 4:
Score de Treinamento: 1.0
Score de Teste: 0.9259259259259259

Fold 5:
Score de Treinamento: 0.9814814814814815
Score de Teste: 0.9629629629629629
```

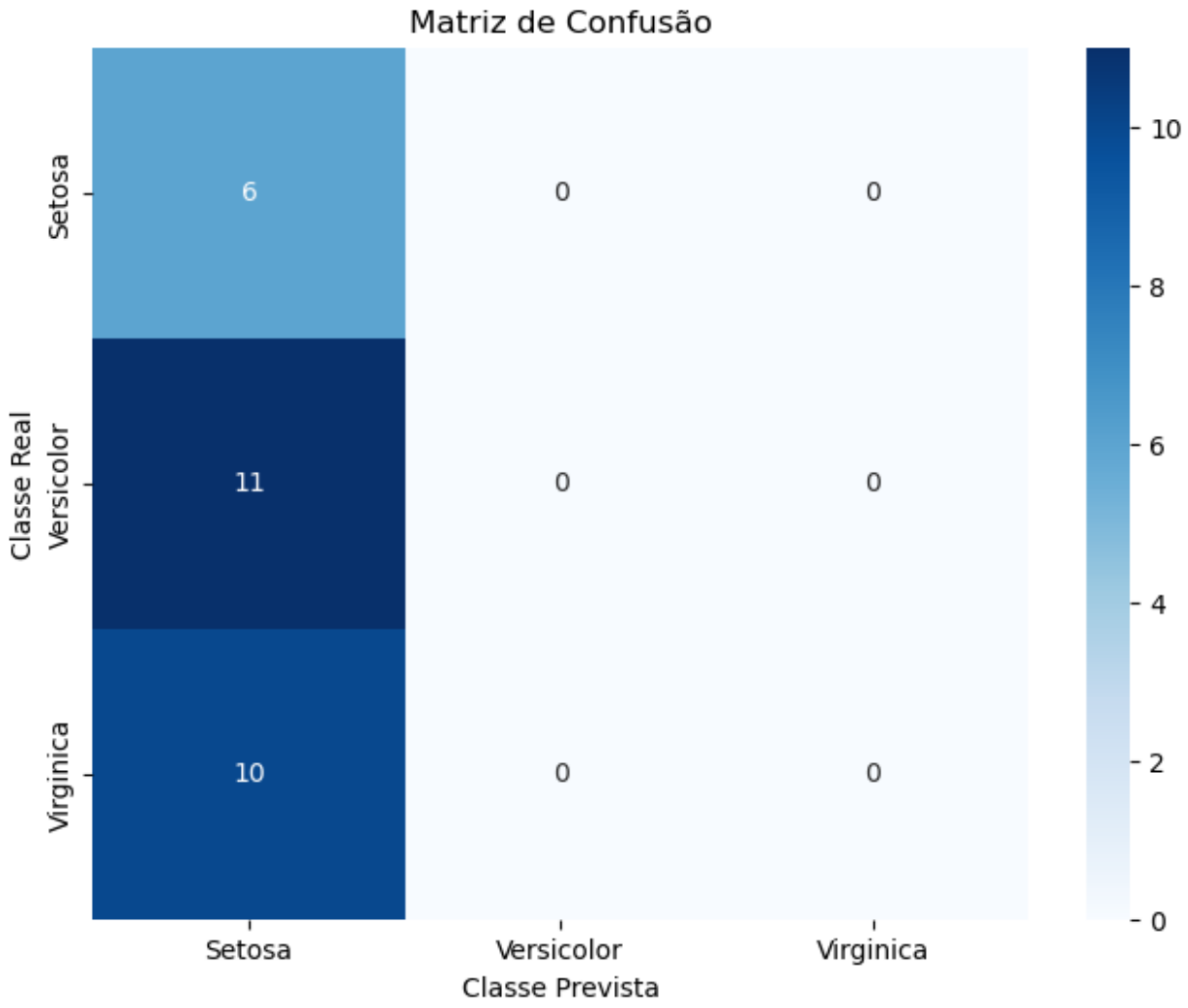
Melhora significativa de score, quando comparado com nossa primeira tentativa ingênua de modelo. E a sua matriz de confusão tem uma diagonal mais cheia, que é exatamente o que queriamos.

LBFS + Identity

```
Fold 1:
Score de Treinamento: 0.9814814814814815
Score de Teste: 0.8888888888888888

Fold 2:
Score de Treinamento: 0.9814814814814815
Score de Teste: 1.0

Fold 3:
Score de Treinamento: 1.0
Score de Teste: 0.9259259259259259

Fold 4:
Score de Treinamento: 0.9814814814814815
Score de Teste: 1.0

Fold 5:
Score de Treinamento: 0.9814814814814815
Score de Teste: 0.9629629629629629
```

E os scores são tão bons quanto do LBFGS + ReLU

LBFGS + TANH



Nos gerou uma matriz de confusão horrível. Onde nosso modelo só se viciou em Iris Setosa.

```
Fold 1:
Score de Treinamento: 0.3518518518518518
Score de Teste: 0.2592592592592924

Fold 2:
Score de Treinamento: 0.3518518518518518
Score de Teste: 0.2592592592592924

Fold 3:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2222222222222222

Fold 4:
Score de Treinamento: 0.3796296296296965
Score de Teste: 0.1481481481481481

Fold 5:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2222222222222222
```

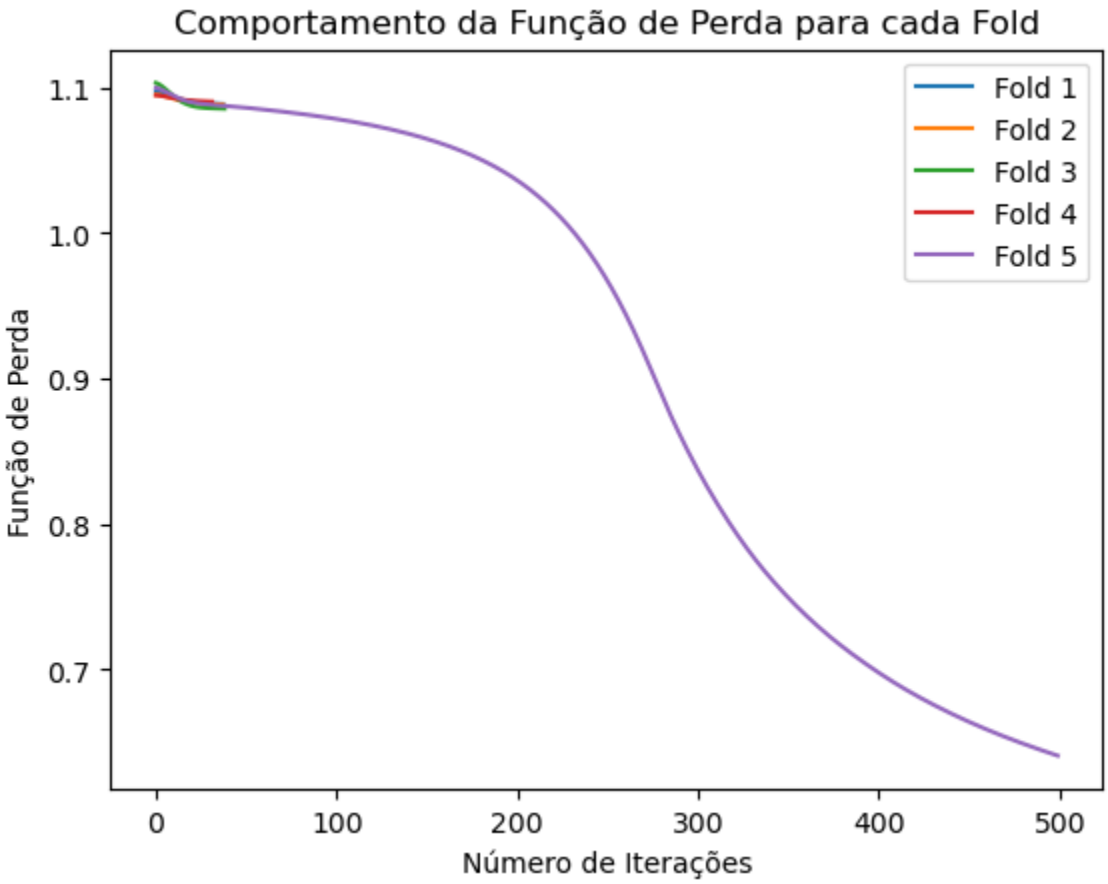
LBFS + Logistic

```
Fold 1:
Score de Treinamento: 0.9814814814814815
Score de Teste: 0.8888888888888888

Fold 2:
Score de Treinamento: 0.9814814814814815
Score de Teste: 1.0

Fold 3:
Score de Treinamento: 0.6759259259259259
Score de Teste: 0.6296296296296297

Fold 4:
Score de Treinamento: 0.6666666666666666
Score de Teste: 0.6666666666666666

Fold 5:
Score de Treinamento: 0.9814814814814815
Score de Teste: 0.9629629629629629
```

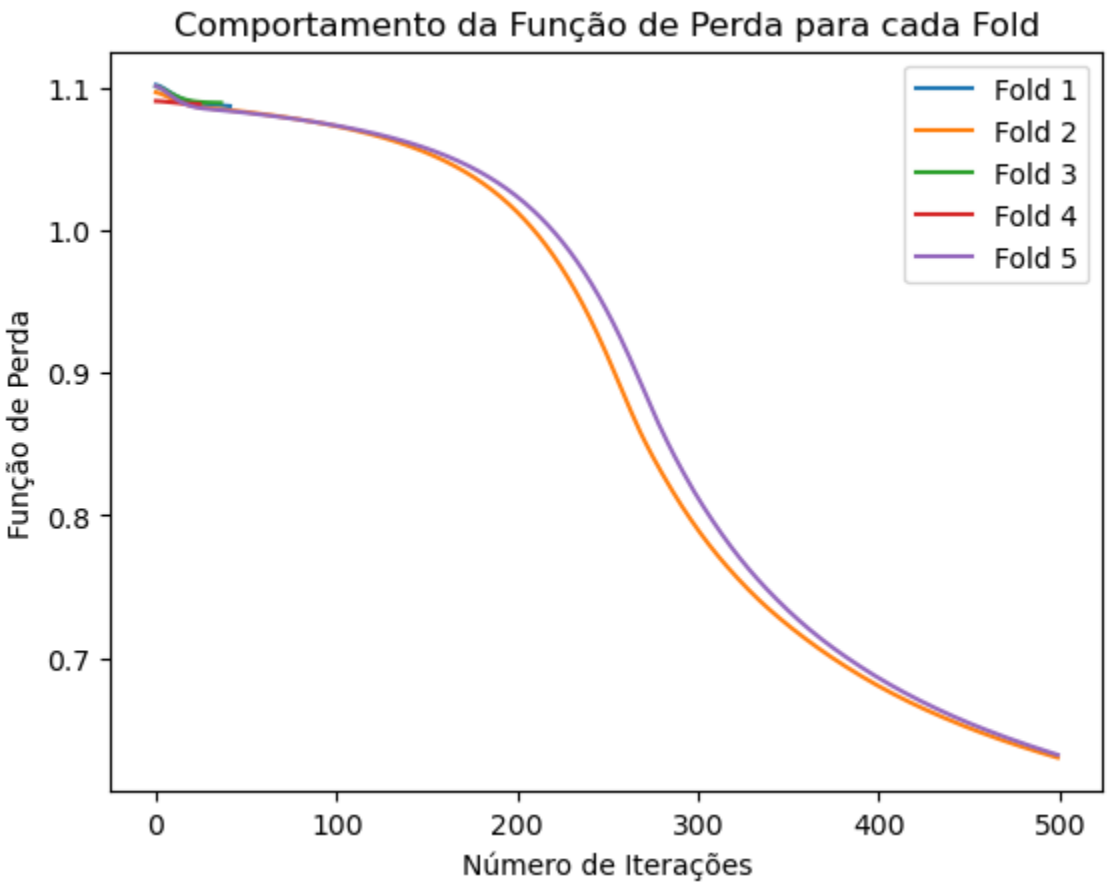
Eu achei meio estranho o comportamento dessa combinação por que depois de rodar várias vezes o código, eu percebi que sempre aparecem scores de testes proximos a 0.90 para cima mas alguns folds em todas rodadas que fiz são por volta de 0.6. Não sei o quanto vale a pena essa escolha de função de ativação tendo em vista que alguns folds sempre saem com menor desempenho.

SGD:

SGD + ReLU



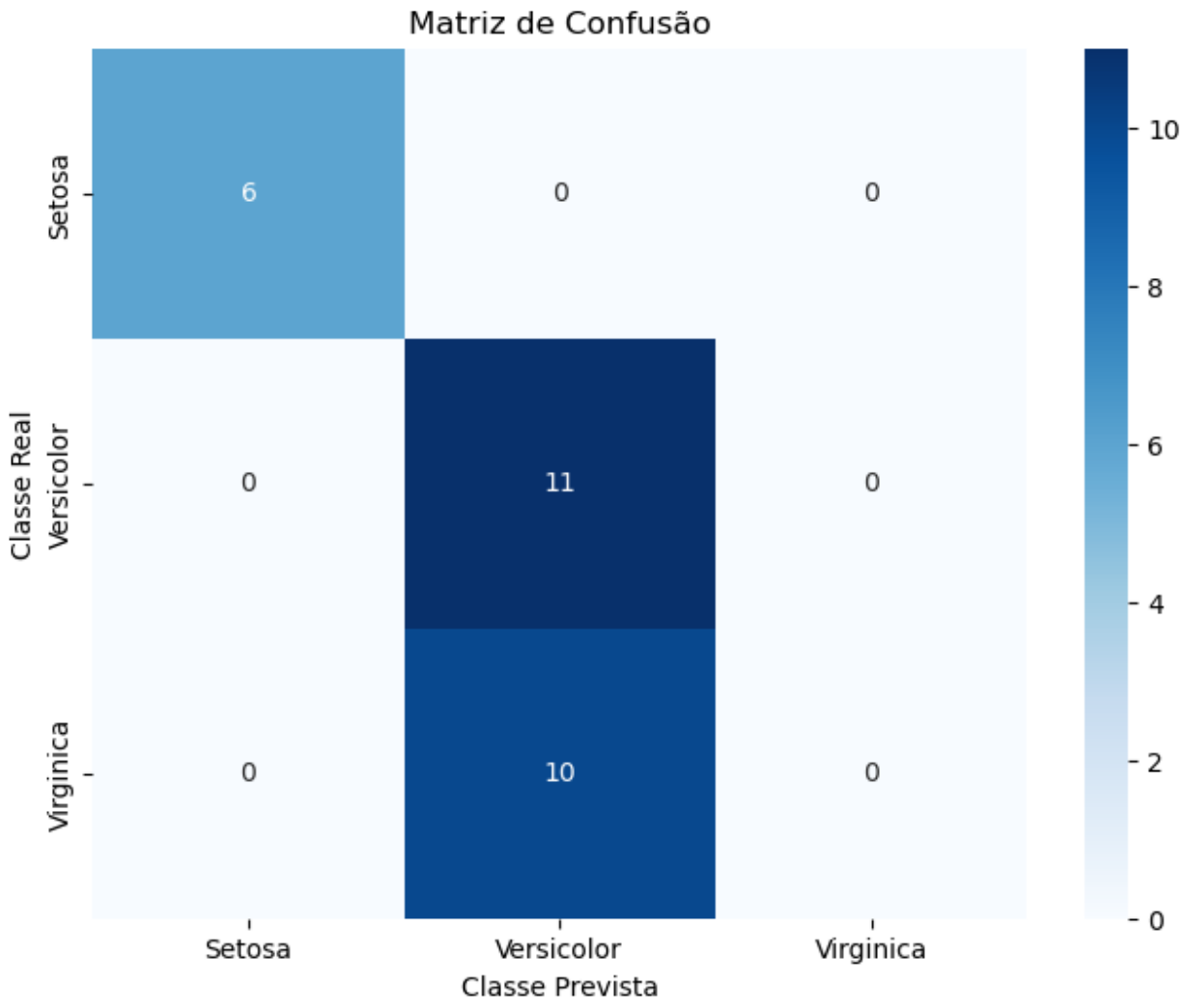
Achei estranho na primeira vez que eu olhei essas 4 primeiras curvas “morrendo” e a última demorando para convergir, suspeitei de isso ser algo relacionado à sensibilidade à inicialização dos pesos, por isso rodei outra vez com a `random_state = 30` e obtive algo parecido mas com uma curva a mais convergindo até o fim, acredito que o SGD não seja recomendado para nosso caso justamente por que estamos com um conjunto muito pequeno. Poderiamos claro mudar esse random state algumas vezes e testar se essas curvas melhoram mas acho que é um trabalho muito exaustivo fazer isso, ainda mais porque nosso método acima já encontrou scores melhores, não cheguei ainda a falar nos scores desse método, mas foram em geral bem ruins.



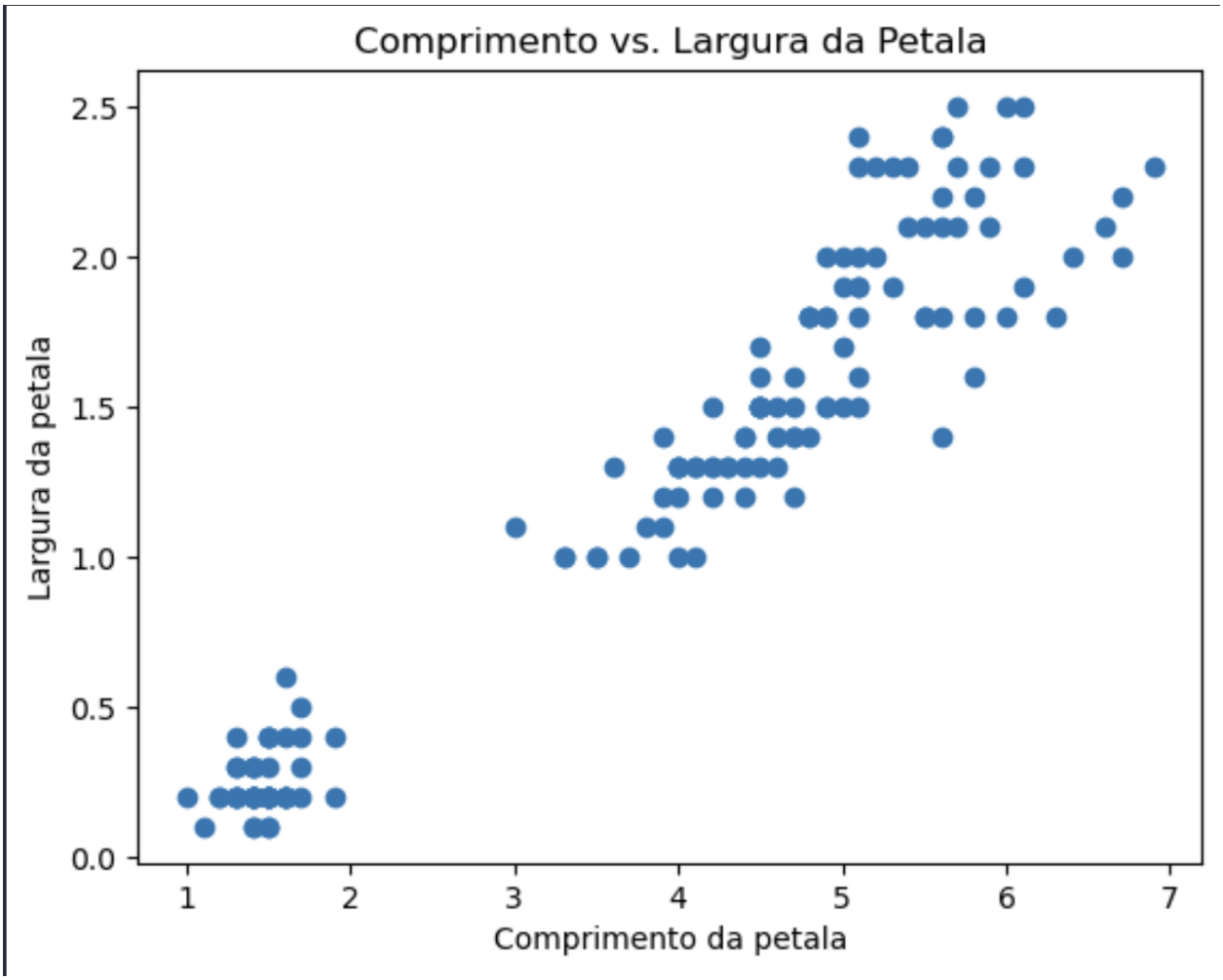
Além dos problemas que encontramos com as funções de perda, achei um score bem ruim:

```
Fold 1:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2592592592592924

Fold 2:
Score de Treinamento: 0.6851851851851852
Score de Teste: 0.5555555555555556

Fold 3:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2222222222222222

Fold 4:
Score de Treinamento: 0.3796296296296965
Score de Teste: 0.1481481481481481

Fold 5:
Score de Treinamento: 0.6666666666666666
Score de Teste: 0.6296296296296297
```

E isso é facilmente observável quando analisamos a matriz de confusão:



Houve algum problema no meio do caminho que fez nosso modelo entender bem o que é uma setosa, o que é uma versicolor mas não entender a virginica e confundiu todas elas com versicolor, pode ser inclusive um reflexo da nossa imagem lá do começo, lembra?

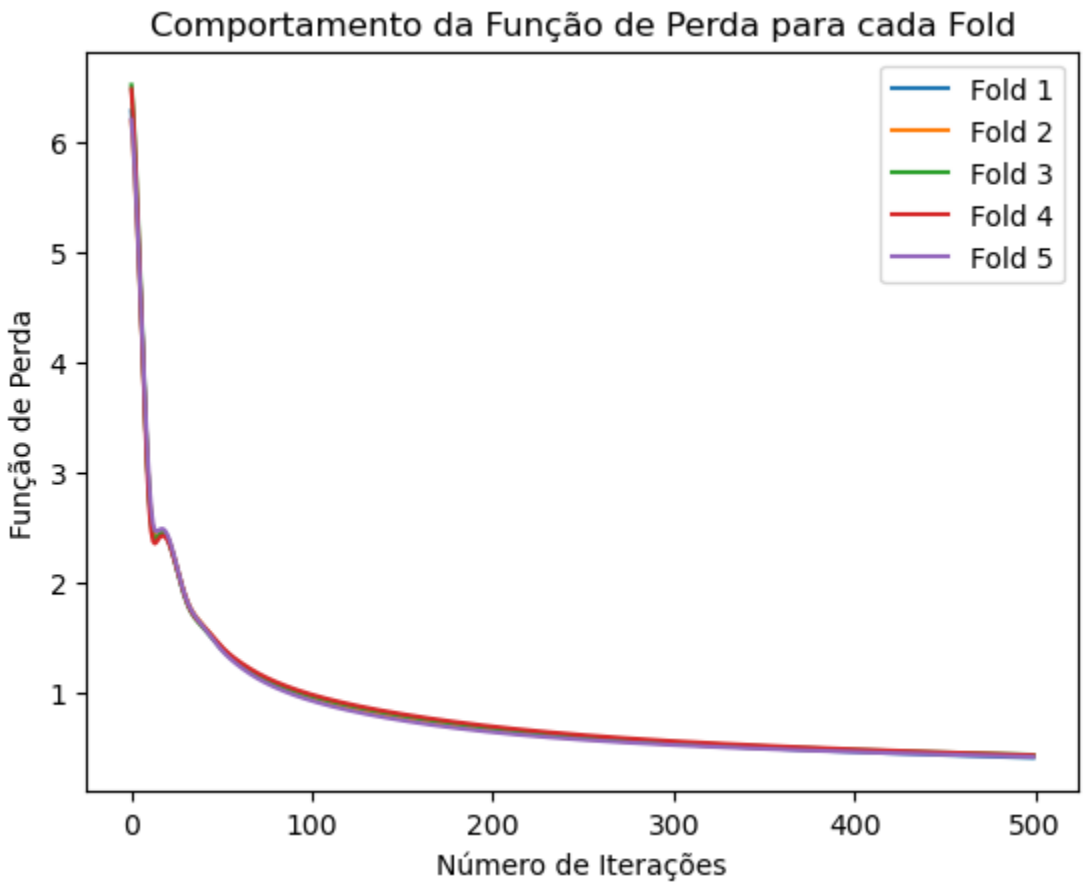


Ele pode ter agido igual uma pessoa ingênua que olha pra essa imagem e consegue ver claramente dois agrupamentos, ele decidiu que aquele aglomerado de comprimento e largura menor era a classe A e o resto era B , mas o modelo não reconhece que tem uma classe C misturada no agrupamente A e B.

Mas enfim, será o SGD de todo ruim? Vamos testar como ele funciona com as outras funções de ativação.

SDG + Identity

Achei importante mostrar que a nossa curva das funções de custo dessa vez tem um formato diferente, o custo aumenta mas depois volta a decrescer.



E a maior intensidade de decaimento são nas interações iniciais, o que nos leva a pensar em custo computacional vs. ganho .

```
Fold 1:
Score de Treinamento: 0.796296296296296
Score de Teste: 0.8148148148148148

Fold 2:
Score de Treinamento: 0.9166666666666666
Score de Teste: 0.8888888888888888

Fold 3:
Score de Treinamento: 0.9259259259259259
Score de Teste: 0.8518518518518519

Fold 4:
Score de Treinamento: 0.8333333333333334
Score de Teste: 0.7037037037037037

Fold 5:
Score de Treinamento: 0.8611111111111112
Score de Teste: 0.9629629629629629
```

Nossos scores são ok, mas não os melhores que já vimos por ai, e parecem sensíveis a mudanças, eu rodei com diferentes seeds e as vezes dava 0.9 em todos, algumas vezes apareciam uns 0.6.

SGD + tanh

```
Fold 1:
Score de Treinamento: 0.3703703703703035
Score de Teste: 0.1851851851851517

Fold 2:
Score de Treinamento: 0.3518518518518186
Score de Teste: 0.2592592592592924

Fold 3:
Score de Treinamento: 0.3796296296296965
Score de Teste: 0.1481481481481414

Fold 4:
Score de Treinamento: 0.3333333333333333
Score de Teste: 0.3333333333333333

Fold 5:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2222222222222222
```

Scores ruins e ele se vicia e diz que tudo é setosa.

SGD + logistic

```
Fold 1:
Score de Treinamento: 0.37037037037037035
Score de Teste: 0.1851851851851517

Fold 2:
Score de Treinamento: 0.04629629629629294
Score de Teste: 0.037037037037037035

Fold 3:
Score de Treinamento: 0.3796296296296965
Score de Teste: 0.1481481481481814

Fold 4:
Score de Treinamento: 0.3518518518518186
Score de Teste: 0.2592592592592924

Fold 5:
Score de Treinamento: 0.25
Score de Teste: 0.1851851851851517
```

Scores ruins

Do SGD então podemos concluir que comparado com IBFS nenhuma estratégia que adotaríamos seria competitiva.

Adam :

Adam + Relu

```
Fold 1:
Score de Treinamento: 0.7037037037037037
Score de Teste: 0.629629629629297

Fold 2:
Score de Treinamento: 0.9444444444444444
Score de Teste: 0.8888888888888888

Fold 3:
Score de Treinamento: 0.6944444444444444
Score de Teste: 0.5555555555555556

Fold 4:
Score de Treinamento: 0.75
Score de Teste: 0.629629629629297

Fold 5:
Score de Treinamento: 0.8518518518518519
Score de Teste: 0.8888888888888888
```

Possui uma melhor muito superior quanto ao primeiro uso que fizemos de Adam + identity, o que sugere que o problema na nossa abordagem inicial era realmente a ativação identity que somada ao solver Adam não era uma boa combinação. Mas ainda assim é um score ruim comparado a outros que já vimos.

Adam + tanh

```
Fold 1:
Score de Treinamento: 0.2685185185185185
Score de Teste: 0.3703703703703703

Fold 2:
Score de Treinamento: 0.3518518518518518
Score de Teste: 0.2592592592592592

Fold 3:
Score de Treinamento: 0.3055555555555556
Score de Teste: 0.3703703703703703

Fold 4:
Score de Treinamento: 0.3333333333333333
Score de Teste: 0.3333333333333333

Fold 5:
Score de Treinamento: 0.3611111111111111
Score de Teste: 0.2222222222222222
```

Score ruim

Adam + logistic

```
Fold 1:
Score de Treinamento: 0.6666666666666666
Score de Teste: 0.6666666666666666

Fold 2:
Score de Treinamento: 0.6851851851851852
Score de Teste: 0.5555555555555556

Fold 3:
Score de Treinamento: 0.6944444444444444
Score de Teste: 0.5555555555555556

Fold 4:
Score de Treinamento: 0.7870370370370371
Score de Teste: 0.7407407407407407

Fold 5:
Score de Treinamento: 0.6759259259259259
Score de Teste: 0.6296296296296297
```

O score não é muito bom e na matriz de confusão podemos ver que ele assim como no caso do SGD + Relu ele se confunde com os agrupamentos.

Terminado agora todas combinações possíveis para teste podemos concluir que claramente que os solvers que usam método do SGD sofrem em desempenhar bem porque se trata de um dataset pequeno, como já tinhamos citado anteriormente e conseguimos concluir agora depois de uma prova real. Com isso podemos virar nossos olhos para uma análise final de escolha do nosso método perfeito (ou quase isso) considerando apenas LFBS que não usa SGD.

O resultado obtido modificando os parâmetros foi o esperado?

Sim, já era de se esperar que métodos de solução envolvendo SGD seriam prejudicados pela nossa especificidade do problema ter poucas instâncias. Então era de se esperar uma superioridade do LFBS e foi o que conseguimos alcançar.

Conclusão

Para testar o modelo devemos rodar o treinamento mais de uma vez e tirar a média desses scores.

```
scores = []
for _ in range(n_repeats):
    model = MLPClassifier(hidden_layer_sizes=hidden_layer_sizes,
                          max_iter=max_iterations,
                          activation=activation,
                          solver=solver)

    # Treinando o modelo com 90% dos dados de treinamento
    model.fit(X_train, y_train)

    # Avaliando o modelo com as 15 instâncias não utilizadas
    test_score = model.score(X_test, y_test)
    scores.append(test_score)

# Calculando a média dos scores
mean_score = np.mean(scores)

print(f"Média do Score: {mean_score}")
```

Que é feito facilmente com o código acima, e ai rodei esse trecho de código com os melhores parâmetros da análise que fizemos acima.

LBFGS + ReLU

Quando fazemos:

```
max_iterations = 500
hidden_layer_sizes = 2
solver = 'lbfgs'
activation = 'relu'
n_repeats = 100
```

Já quando usamos mais iterações e mais layers isso vai subindo, até que com 1000 iterações e 8 layers temos:

Mas devemos nos atentar o tanto que tivemos que aumentar o número de iterações e layers para obter um resultado próximo de bom (0.95).

LBFGS + Identify

Com as mesmas configurações iniciais do ReLU, conseguimos de cara um score médio de 1 (repeti várias vezes o experimento porque não estava crendo.

Disso podemos concluir que nossa duvida entre a função de ativação ReLU ou identity para acompanhar nosso solver lbfgs devemos escolher a identity, porque apesar de o ReLU ir aumentando seu desempenho conforme movemos mais recursos

computacionais para ele, com identity fazemos isso de forma muito mais simples, logo mais otimizada, se usa menos recursos.

Então após tudo que fizemos, concluímos que a melhor escolha para esse problema é usar os parâmetros

```
max_iterations = 500
hidden_layer_sizes = 2
solver = 'lbfgs'
activation = 'identity'
n_repeats = 100
```

Que geram sempre uma matriz confusão perfeita: