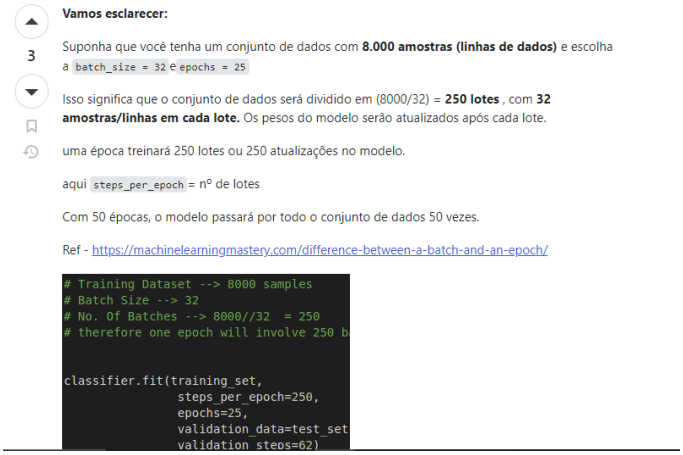
BASE DE DADOS BREAST CANCER

- Estudo de caso com redes neurais. Tarefa de classificação binária.
    - Tem ou não câncer maligno
- Passo a passo
    - Importar a biblioteca pandas
        - Import pandas as pd (nome dado por vc msm)
    - Criação de duas variáveis
        - Previsores = pd.read_csv('nomedoarquivo.csv')
            - Csv é o tipo de arquivo que está os dados
            - Read_csv é para ler esse arquivo
        - Classe = pd.read_csv('nomedoarquivo.csv')
    - Divisão dessa base de dados:
        - Treinamento:
            - No exemplo, é nessa base de dados treinamento que a rede neural vai encontrar o padrão nesses dados
            - Vai descobrir quais são os pesos utilizados para os atributos/conexões na rede neural
        - Teste:
            - Fazer a avaliação de qual é o percentual de acerto percentual de erro que a rede neural está tendo
        - Fazer uma importação
            - From sklearn.model_selection import train_test_split
                - É uma função automática que vai fazer a divisão da base de dados entre treinamento e teste
                - Sklearn é uma biblioteca
    - Criação de variáveis:
        - Previsores_treinamento
        - Previcores_classe
        - Classe_treinamento
        - Classe_teste
        - Todas as variáveis juntas = train_test_split(previsores, classe, test_size=0.25)
            - Size=0.25 quer dizer que vamos usar 25% de todos os registros para fazer um teste
                - 75% para treinar
                - 25% para testar
        - Faz a divisão entre a base de dados de treinamento e teste
    - Estrutura da rede neural
        - Importa o keras
            - Import keras
        - From keras.models import Sequential
            - Sequential é a classe utilizada para criação da rede neural
                - Basicamente para todos os exemplos do curso
                - Chamado de modelo sequencial, porque tem uma sequência entre a camada de entrada, camada oculta (ou várias) e a camada saída
        - Fazer a importação de mais uma camada

- from keras.layers import Dense
    - layers = camada
    - import Dense = porque vai ter camadas densas na rede neural
        - camada densa: cada um dos neurônios é ligado a cada um da camada subsequente
- Criação da rede neural
    - classificador = Sequential()
    - Adicionar a primeira camada oculta
        - Classificador.add(Dense(units = 16, activation = 'relu', kernel_initializer='random_uniform', input_dim = 30)
            - Dentro do Dense são os parâmetros
                - Units = quanto neurônios farão partes da camada oculta
                - Para saber quantos neurônios, precisa usar a formula: numeroDeEntradas + numerosDeNeuroniosNaCamadaSaida /2
                - Activation = função de ativação
                - Kernel_initializer = inicialização dos pesos
                - Input_dim = quantos elementos existe na camada de entrada
        - Obs: Não precisa criar a camada de entrada explicitamente antes, porque já tem input_dim para saber quantos neurônios vai ser preciso
    - Adicionando a camada de saída
        - Classificador.add(Dense(units = 1, activation = 'sigmoid'
            - Classificação binaria, então a função de ativação vai ser sigmoid
- Configuração e execução da rede neural
    - Agora precisamos compilar a rede neural, parâmetros/configurações:
        - Classificador.compile(optimizer='adam', loss=binary_crossentropy, metrics = [binary_accuracy]
            - Optimizer = qual é a função que vai utilizar para fazer o ajuste do peso
                - Adam = Descida do gradiente estocástico
                    - Adam é o mais indicado e mais adaptável em outros casos
                - Loss = função de perda
                    - É onde faz o cálculo do erro
                - Binary_crossentropy (usado com problema binário)
                - Metrics = fazer a avaliação
                    - É em formato de vetor, pode ser em lista

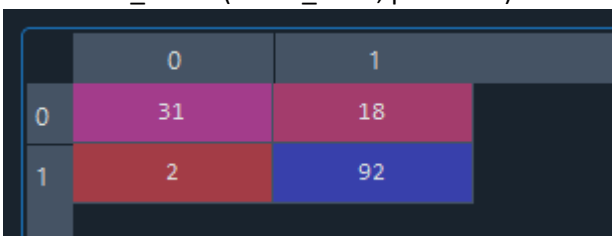
- Accucary = quantos registros foram classificados certos e quantos registros foram classificados errados
    - Exemplo: 100 registro. 90 classificados corretamente.
    - 90/100
    - Vai ter a precisão

- Agora fazer, efetivamente, o treinamento
    - Classificador.fit(previsores_treinamento, classe_treinamento, batch_size=10, epochs = 100
        - Fit = encaixar os previsores_treinamento com classe_treinamento
        - Batch_size = vai calcular o erro para 10 registros (número escolhido por você) depois que ele vai atualizar os pesos. Calcula o erro por mais 10 registro e depois faz o ajuste dos pesos
            - Ideia da descida do gradiente estocástico
            - Só que pega conjunto de 10 em 10 ao invés de 1 em 1
        - O ideal é usar steps_per_epoch= n° da quantidade de registro quanto se tem muito registros

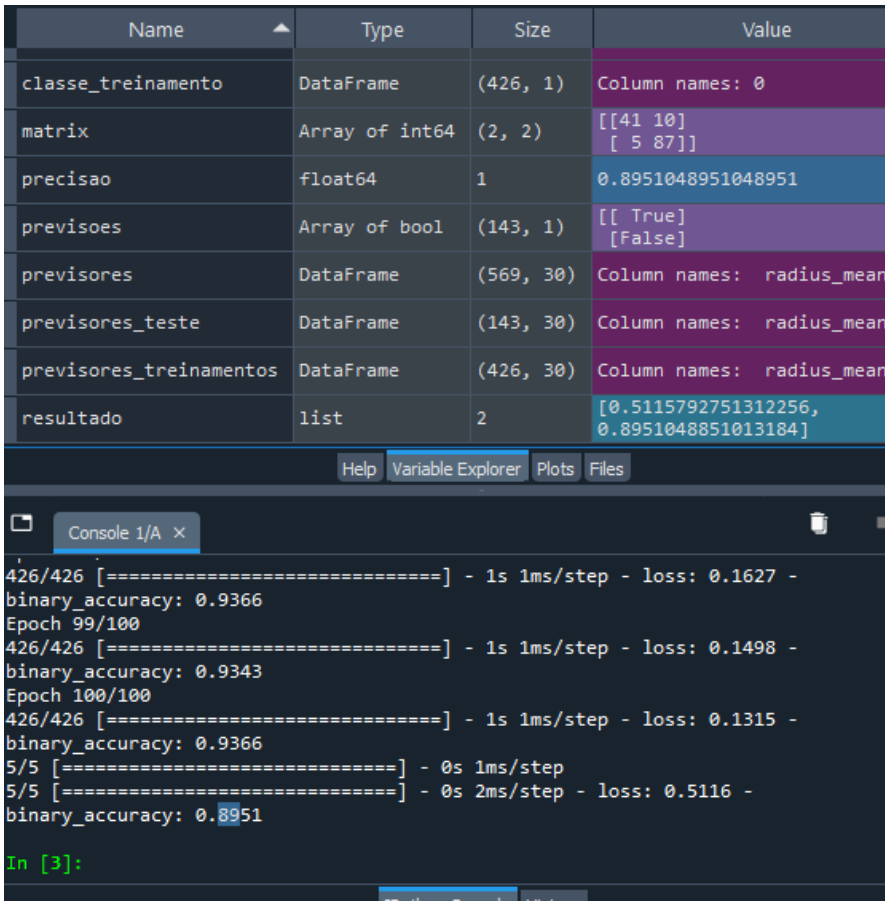


        - Epochs = quantas épocas eu quero executar o treinamento
            - Ou seja, quantas vezes eu vou fazer o ajuste dos pesos

- Lembrando que até o momento só fizemos o treinamento com 426 resgistro, não podemos levar em conta o 93% de acerto. Precisamos fazer com os previsores_teste de 143 registro
    - Previsões e avaliação da rede neural (com base de dados teste)
        - Criar uma variável
            - Previsores
        - Passar como parâmetro o previsores_teste
            - Previsões = classificar.predict(previsores_teste)
        - Agora que já mostra os valores de probabilidade (do lado, no explorador de variáveis)

    - Para saber se a rede neural está se comportando bem, abre os valores de previsoes e os valores da classe_teste e compara
- Esses valores da variável previsoes são os valores são o retorno da função sigmoid em probabilidade, então precisa fazer a conversão em valores 0 ou 1
    - Se previsoes for maior que 0.5, então vai retornar como true e se for menor vai retornar false
        - Previsoes = (previsoes >0.5
    - Agora é mais fácil fazer o comparativo dos resultados
- fazer a avaliação manual
    - From sklearn.metrics import confusion_matrix, accuracy_score
        - Matrix de confusão
    - Precisao = accuracy_score (classe_teste, previsoes)
        - Parâmetros: vetores que vão ser comparados
    - Matrix = confusion_matrix(classe_teste, previsoes)
        - 

        - 0 (linha dos tumores benignos) = Acertou 31 0 (tumor benigno) e errou 18 1 (tumor maligno)
        - 1 (linha dos tumores malignos) = errou 2 0 (tumor benigno) e acertou 92 1 (tumor maligno)
- resultado = classificador.evaluate(previsores_teste, classe_teste)
    - pega os previsores_testes vai submeter para rede neural, a rede neural vai fazer a respectiva previsão e já faz a avaliação (passando a classe_teste/ as respostas corretas)

- 

Name	Type	Size	Value
classe_treinamento	DataFrame	(426, 1)	Column names: 0
matrix	Array of int64	(2, 2)	[[41 10] [5 87]]
precisao	float64	1	0.8951048951048951
previsoes	Array of bool	(143, 1)	[[True] [False]
previsores	DataFrame	(569, 30)	Column names: radius_mean
previsores_teste	DataFrame	(143, 30)	Column names: radius_mean
previsores_treinamentos	DataFrame	(426, 30)	Column names: radius_mean
resultado	list	2	[0.5115792751312256, 0.8951048851013184]

Help | Variable Explorer | Plots | Files

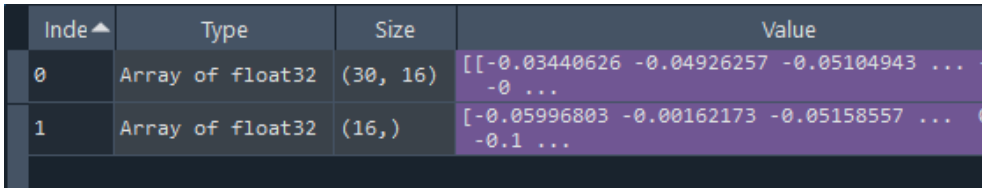
Console 1/A

```
426/426 [==============================] - 1s 1ms/step - loss: 0.1627 -
binary_accuracy: 0.9366
Epoch 99/100
426/426 [==============================] - 1s 1ms/step - loss: 0.1498 -
binary_accuracy: 0.9343
Epoch 100/100
426/426 [==============================] - 1s 1ms/step - loss: 0.1315 -
binary_accuracy: 0.9366
5/5 [==============================] - 0s 1ms/step
5/5 [==============================] - 0s 2ms/step - loss: 0.5116 -
binary_accuracy: 0.8951

In [3]:
```

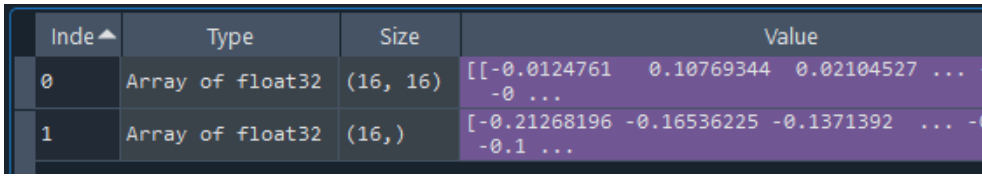
  - Adicionar mais uma camada oculta
    - Mais camadas e parâmetros do otimizador – aula
    - Só “copiar” a primeira camada oculta
    - classificador.add(Dense(units=16, activation='relu',
    - kernel_initializer='random_uniform'))
    - A única diferença é que somente na primeira camada que recebe o “input_dim=30
    - Na primeira camada oculta precisa ter o input_dim e na e na camada de saída (com a quantidade de neurônio necessário) tem que ter a função de ativação correta.
  - Compile, configuração dos parâmetros
    - Usar o optimizers e o adam com os parâmetros:
      - Otimizador (variável) = kera.optimizers.Adam(lr =
        - Parâmetros do adam: learning rate (taxa de aprendizagem) = lr=0.001
          - O ideal é chegar no mínimo global é melhor ter pesos menores, pois ele pode ultrapassar esse ponto
          - Learning rate grande com peso grande a tendência é que ele chegue no máximo global rápido, mas corre risco de
        - Decay = indicar quanto o learning rate vai ser decrementado a cada interação/atualização de peso
          - Começa com learning rate alto e vai diminuindo

        - Existe dois parâmetros comuns para todos os otimizadores
            - Clipvalue =0.5
                - É como se o valor fosse “preso”
                - Valor máximo de 0.5, valor mínimo de -0.5. vai congelar esses valores para que ele não passe muito do padrão
- Visualização dos pesos – aula 22
    - Criar variáveis:
    - peso0 = classificador.layers[0].get_weights()



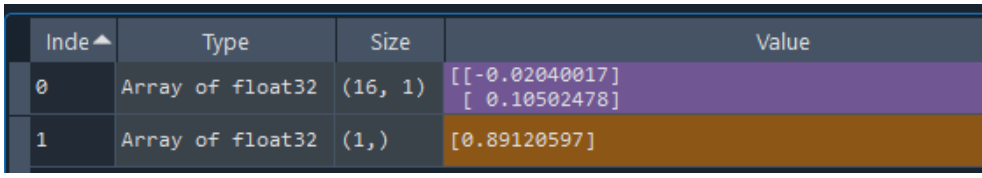
Inde	Type	Size	Value
0	Array of float32	(30, 16)	[[-0.03440626 -0.04926257 -0.05104943 ... -0 ...
1	Array of float32	(16,)	[-0.05996803 -0.00162173 -0.05158557 ... -0.1 ...

        - (30, 16)
            - 30 é a quantidade de entradas, 16 é a quantidade de neurônio que temos na camada oculta
        - (16,)
            - É a unidade de baias. Quando define uma rede neural, por padrão, ele já vem colocado com a unidade de baias marcada como true. Então quer dizer que, ele criou mais 1 neurônio e colocou a ligação desse neurônio para cada um dos 16 neurônios da minha camada oculta
        - Para visualizar: print(peso0)
        - Print(len(peso0))
            - Lê quantas posições tem
    - Pesos1=calssificador.layers[1].get_weights()



Inde	Type	Size	Value
0	Array of float32	(16, 16)	[[-0.0124761 0.10769344 0.02104527 ... -0 ...
1	Array of float32	(16,)	[-0.21268196 -0.16536225 -0.1371392 ... -0.1 ...

        - (16,16)
            - Significa ligação da primeira camada oculta com a segunda camada oculta
        - (16,)
            - Mais unidade de baias. Outro neurônio ligados a todos neurônios da segunda camada oculta
    - Pesos2=cl calssificador.layers[2].get_weights()



Inde	Type	Size	Value
0	Array of float32	(16, 1)	[[-0.02040017] [0.10502478]
1	Array of float32	(1,)	[0.89120597]

        - (16,1)
            - 16 neurônios na ultima camada oculta ligada na camada de saída

- (1,)
    - Camada de baias, mais um neurionio que faz a ligação com o da camada de saída