Relatório da Lista 1 de IC2

Aluno: Rennan de Lucena Gaio
DRE: 119122454

1. Introdução

Com base nos vídeos do Yaser, aprendemos diversas formas de trabalhar com algoritmos de classificação e regressão utilizando de forma apropriada a teoria por trás dessas técnicas. Com intuito de utilizá-los não só como ferramentas, mas de entender os pontos positivos e negativos de cada um deles, foi-se proposto a implementação sem a utilização de bibliotecas prontas para cada um desses principais algoritmos. Dessa forma conseguiremos adquirir uma maior intuição dos melhores casos para utilizá-los, seus respectivos ganhos computacionais e principalmente o ganho intelectual de traduzir toda a teoria observada nas aulas em código prático.

Neste relatório iremos abordar rapidamente as técnicas utilizadas, como foram inicialmente implementadas, algumas dificuldades encontradas e como foram solucionadas. Dentre as técnicas, foram propostas as implementações dos algoritmos: Perceptron, regressão linear, regressão não linear, gradiente descendente e a regressão logística. Para alguns dos desafios foram feitos alguns gráficos para a visualização na medida do possível.

2. Considerações iniciais

Todo o código foi implementado utilizando python 3. Foi-se escolhido tal linguagem pela facilidade técnica que o aluno já possui com a linguagem, além das bibliotecas matemáticas que o mesmo oferece.

Para a implementação dos módulos dos algoritmos a seguir foram utilizadas como apoio à implementação as bibliotecas do python: numpy, random e matplotlib. Caso algum erro aconteça por conta da falta dessas bibliotecas, as mesmas podem ser instaladas da seguinte forma:

```
# pip install numpy random matplotlib
```

Caso o python 3 não seja o seu python default (no caso de estar utilizando o python 2), repita o comando anterior substituindo “pip” por “pip3”.

Para executar códigos em anexo, simplesmente execute-os da seguinte maneira:

```
$ python <arquivo.py>
```

Caso o python 3 não seja seu python default, utilize o comando “python3” no lugar do “python” acima. Para checar sua versão default do python, pode-se executar o seguinte comando:

```
$ python --version
```

As respostas referentes a cada exercício estarão explicitadas em cada respectiva seção neste próprio documento.

Cada algoritmo foi implementado em um arquivo python diferente, para melhor entendimento e modularização. Dessa forma não teremos arquivos muito grandes a serem analisados de uma só vez. Algumas funções para criação de pontos randômicos, criação da target function, dentre outros, são comuns a todos os arquivos, e estarão presentes no início de cada um dos algoritmos (elas estarão devidamente sinalizadas, e não possui diferença entre elas ao longo dos exercícios propostos).

Todos os gráficos e imagens deste arquivo são geradas pelo código em questão, e podem ser reproduzidas alterando a flag “graphical” para “True” na main de cada um dos arquivos que possuem a possibilidade de gerar os gráficos. Não foi feito a parte gráfica para todos os exercícios, ela tinha como objetivo ser mais didática em alguns casos sem comprometer o escopo do relatório.

Qualquer dúvida adicional podem me contactar pelo seguinte email: “rennan.gaio12@gmail.com”

3. Exercícios propostos

Nessa seção serão descritas as implementações de cada um dos algoritmos, uma breve discussão sobre as perguntas feitas e uma análise dos resultados obtidos.

3.0. Funções auxiliares

Como um muitos dos exercícios propostos são necessários a criação de pontos aleatórios, a definição de funções target a priori e a rotulação de pontos, foram implementadas funções comuns a todos os arquivos, que estarão definidas no início de cada um, para realizarem esta tarefa.

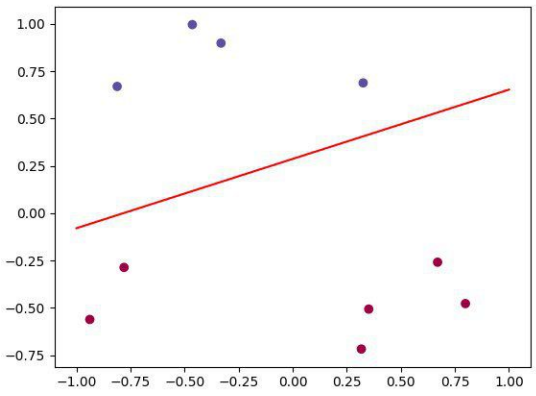
Dentre elas temos as funções:

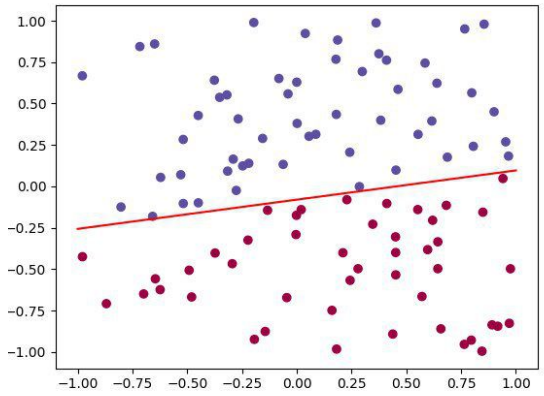
- “create_target()”: Inicializa 2 pontos aleatórios e cria os pesos correspondentes à reta entre eles.
- “transform_w_to_equation()”: Transforma nosso vetor de pesos para que possa ser utilizada pela biblioteca gráfica do matplotlib.
- “pre_classify()”: Rotula os dados gerados randomicamente com base nos pesos da função target.
- “load_data()”: Cria os N pontos requisitados em cada exercício e os rotula com base nos pesos da target function, retornando um vetor de dados e um vetor de labels.

Estas funções não terão comentários adicionais no código pela simplicidade das mesmas, e por já terem seu funcionamento descrito nesta seção.

Para testar a corretude de seus resultados, foram produzidos alguns exemplos gráficos utilizando a biblioteca do matplotlib. em que em azul são os pontos classificados

como +1, em vermelho os pontos classificados como -1, e a reta separando os pontos é nossa target function gerada. A primeira imagem foi gerada com N=10, e a segunda com N=100.





3.1. Algoritmo de aprendizagem Perceptron

3.1.1. Sobre seu funcionamento

Como visto com base nas aulas do Yaser, o algoritmo de aprendizagem do Perceptron é um classificador linear, que é atualizado a cada iteração selecionando um ponto classificado de forma incorreta pela sua atual linha. Para fazer essa atualização ele faz uma soma dos pesos do ponto classificado incorretamente (podendo ter um fator multiplicador entre [0,1] também chamado de Learning rate) com os pesos atuais da reta separadora. Esse procedimento até que os dados estejam totalmente separados. Note que esse algoritmo converge de forma natural se, e somente se, os dados forem linearmente separáveis. Caso este não seja o caso, podem ser feitas algumas alterações em que ele salva o estado com menor erro dentro da amostra, e seja definido um número máximo de iterações. Caso essas medidas não sejam tomadas para dados não linearmente separáveis o algoritmo não irá convergir.

3.1.2 Sobre sua implementação

O arquivo correspondente as implementações dos exercícios 1, 2, 3 e 4 da lista (exercícios relacionados ao Perceptron) estará em anexo com o nome: “perceptron.py”.

Foi-se criado para a realização das atividades uma classe “Perceptron”, e para seguir minimamente os padrões de classes de machine learning os nomes dos métodos foram mantidos na medida do possível. A intenção foi criar uma classe geral, que funcione independentemente do tipo de entrada de dados que possuir. Na classe estarão sendo implementadas as funções de:

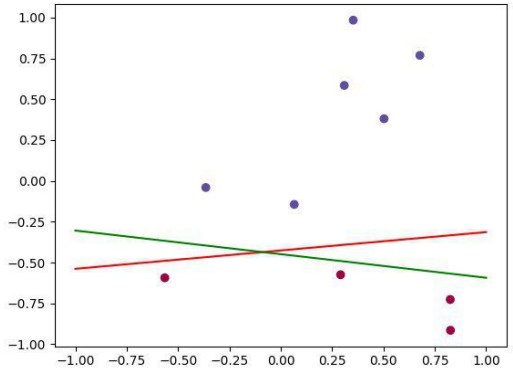
- “fit”: Cria o vetor de pesos do perceptron ajustado aos dados de entrada. Pode ser considerado como a fase de treino do nosso algoritmo.
- “predict”: Retorna o vetor de labels como resultado da predição dos pontos passados para esta função com relação aos pesos encontrados pelo perceptron na sua fase de treino (função g).

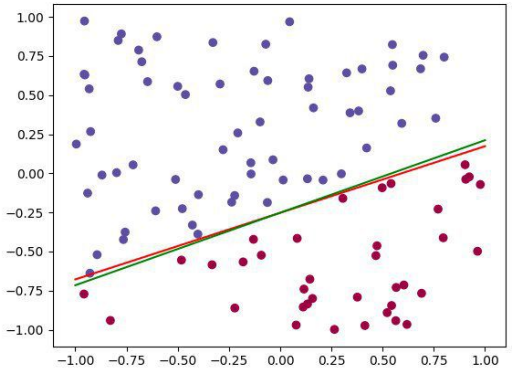
Para a execução dos exercícios 1 à 4, foi criada uma função “exercises”. Foi-se feito dessa forma pois a diferença entre os exercícios 1 e 3, 2 e 4, eram apenas a quantidade do parâmetro N. Nesta função ele estará criando as amostras, executando o classificador do perceptron, e a cada rodada de teste (total de 1000 rodadas conforme o enunciado, para a melhor colheita de estatísticas) ele irá guardar o número de iterações e a divergência de classificação de amostras entre as funções. Após o término das rodadas é utilizada a função de média de numpy para calcular a média dentre as amostras, que serão os resultados dos exercícios 1 à 4.

O funcionamento mais específico de cada passo das funções será melhor descrito em forma de comentários no código, desta forma ficará mais organizado e especificado.

3.1.3 Resultados encontrados

Para mostrar de forma gráfica o funcionamento dos algoritmo do perceptron, foi-se criado 2 gráficos representando os dados com suas respectivas labels e a target function (assim como no gráfico mostrado na seção de funções auxiliares), com a adição de uma reta verde, representando nossa função g criada pelo algoritmo do perceptron.





A primeira gerada com 10 pontos e a segunda gerada com 100 pontos. Podemos perceber que a quantidade de pontos influencia diretamente no quão perto nossa target e nossa g function estão similares, e poderemos perceber isso nos resultados encontrados após a execução sucessiva dos experimentos.

Sem mais delongas, os resultados encontrados executando 1000 rodadas para a média de iterações e a divergência média entre as funções:

- Para N=10
  - média de iterações ~= 10.77
  - divergência média ~= 0.1162
- Para N=100

    - média de iterações ~= 107.57
    - divergência média ~= 0.01393

  Temos então como resultados as respostas:

- 1 = B
- 2 = C
- 3 = B
- 4 = B

3.2. Regressão Linear

3.2.1. Sobre seu funcionamento

Como visto com base nas aulas do Yaser, o algoritmo de regressão linear é um regressor linear, que busca a melhor reta que representa seus pontos da amostra. Para fazer a tarefa de classificação do mesmo foi necessário criar um threshold na variável y (das labels), pois com esse threshold definido é possível fazer sua projeção de separação dos dados no plano das coordenadas X. Sem essa alteração o algoritmo não consegue fazer a divisão dos dados de maneira esperada. Para se criar a linha de regressão foi utilizada a técnica de mínimos quadrados, que pode ser descrita com as seguintes fórmulas:

$$Y = X\beta + \varepsilon, \qquad (X'X)\widehat{\beta} = X'Y.$$

Sendo a primeira a fórmula geral, e a segunda a fórmula para a criação dos pesos, descrito por Beta, que é a aplicação dos mínimos quadrados.

Para a criação do g, após a obtenção dos pesos basta aplicar os mesmos no ponto (lembrando que existe um threshold na variável y), caso seja menor que o threshold, classifique como -1, caso seja maior, classifique como +1, ou vise e versa.

3.2.2 Sobre sua implementação

O arquivo correspondente as implementações dos exercícios 5, 6 e 7 da lista (exercícios relacionados à regressão linear) estará em anexo com o nome: “regressao_linear.py”.

Foi-se criado para a realização das atividades uma classe “LinearRegression”, e para seguir minimamente os padrões de classes de machine learning os nomes dos métodos foram mantidos na medida do possível. A intenção foi criar uma classe geral, que funcione independentemente do tipo de entrada de dados que possuir. Na classe estarão sendo implementadas as funções de:

- “fit”: Cria o vetor de pesos da regressão ajustado aos dados de entrada. Pode ser considerado como a fase de treino do nosso algoritmo.
- “predict”: Retorna o vetor de labels como resultado da predição dos pontos passados para esta função com relação aos pesos encontrados pela regressão linear na sua fase de treino (função g).
- “fit2d”: Possui o mesmo funcionamento da função fit, porém limitada a 2 dimensões. Ela foi criada com um intuito da minha maior compreensão do problema em seu

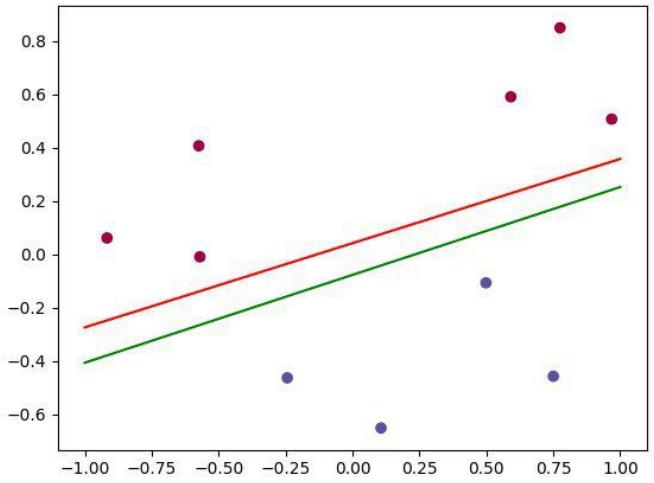
desenvolvimento, e também para alguns plots de estudo. Ela de fato não estará sendo utilizada nos exercícios propostos.

Para a execução dos exercícios 5 à 7, foram criadas funções respectivas para cada um dos problemas, seus nomes seguem um padrão de “exercise_<numero do exercício correspondente>”. Note que a utilização do threshold no código foi feito de maneira implícita, pois como as nossas labels eram de -1 até 1, foi escolhido 0 como threshold. E para representar isso no nosso algoritmo de aprendizado, como a nossa variável é zerada, eu apenas não considero as variáveis y na nossa classificação. Logo os pesos já saem no formato de reta diretamente (pois estamos lidando com dados de dimensionalidade 2).

O funcionamento mais específico de cada passo das funções será melhor descrito em forma de comentários no código, desta forma ficará mais organizado e especificado.

3.2.3 Resultados encontrados

Para uma visualização do funcionamento do nosso algoritmo de regressão linear para a classificação, foi-se gerado um gráfico com o matplotlib. De tal forma que os pontos em azul possuem label +1, os pontos em vermelho possuem label -1, a nossa função target é simbolizada pela reta em vermelho, e nossa função g criada pela regressão linear classificando os dados é simbolizada pela reta em verde.



Para a resposta dos exercícios propostos:

Primeiramente era pedido a média do erro dentro da amostra pela regressão linear, executando 1000 rodadas de teste para a maior confiabilidade. Para tal tarefa foi criada a função “exercise_5”, que teve como retorno o valor para a média Ein de 0.0383.

No exercício 6 foi pedido a estimativa do erro fora da amostra. Para isso após a criação a nossa função g, foram criados novos N pontos que não foram utilizados para o treino do nosso regressor, seguindo a mesma distribuição e depois foram aplicados a função de predict do nosso classificador. Novamente foram executadas 1000 rodadas para maior confiabilidade nos resultados, e o resultado obtido para Eout foi de 0.0387.

No exercício 7 foi pedido a quantidade média de iterações que o perceptron precisaria executar após os pesos já serem inicializados com o nosso algoritmo de regressão linear. Para isso foi utilizado a classe de perceptron já pronta, com a alteração de

que os pesos não seriam mais inicializados zerados, mas sim, com os pesos obtidos no nosso regressor. Após essa modificação, foram executadas 1000 rodadas e foi obtido para o valor de interações médias 2.21 interações.

Em resumo os valores obtidos foram:

- exercício 5 = 0.0383 = C
- exercício 6 = 0.0387 = C
- exercício 7 = 2.21 = A

3.3. Regressão não linear

3.3.1. Sobre seu funcionamento

A função de regressão não linear se assemelha muito com a função de regressão linear clássica, porém nas entrada de suas coordenadas, ao invés de utilizarmos variáveis de primeira ordem, estaremos utilizando variáveis de maior ordem, como no exemplo, estaremos utilizando as variáveis X1 e X2 ao quadrado. Conforme os dados forem ajustados no vetor de entrada dos pontos, o algoritmo de regressão irá funcionar de maneira similar ao exercício anterior.

3.3.2 Sobre sua implementação

O arquivo correspondente as implementações dos exercícios 8, 9 e 10 da lista (exercícios relacionados à regressão não linear) estará em anexo com o nome: “regressao_nao_linear.py”.

Como dito anteriormente, o seu funcionamento se assemelha muito com a classe de regressão linear, logo a solução tomada foi utilizar a mesma classe antes já produzida fazendo as devidas alterações apenas nos dados de entrada que ela irá receber.

Como a função target é definida no problema, a função auxiliar “create_target()” também foi alterada, para que ela deixe de criar uma reta aleatória, e utilizar os pesos definidos para as variáveis quadráticas em nosso problema.

Para a execução dos exercícios 8 à 10, foram criadas funções respectivas para cada um dos problemas, seus nomes seguem um padrão de “exercise_<numero do exercício correspondente>”. Note que a utilização do threshold no código foi feito de maneira implícita, assim como na regressão linear. Pois como as nossas labels eram de -1 até 1, foi escolhido 0 como threshold. E para representar isso no nosso algoritmo de aprendizado, como a nossa variável é zerada, eu apenas não considero as variáveis y na nossa classificação.

Para a alteração dos dados de entrada foi gerado um outro vetor com entradas respectivamente iguais a: X1*X2, X1^2 e X2^2, e nessas entradas foram recebidas com os valores dessas contas dados os pontos X1 e X2 dos dados gerados. Logo após isso os vetores foram juntados com a função append, e utilizados como dados primários de 6 dimensões.

Um detalhe adicional é que é pedido a criação de ruído na classificação dos dados. Para isso foi gerado um loop que é executado com o valor de 10% da quantidade dos dados

que de forma aleatória atribui a label daquele dado ou 1 ou -1. Dessa forma a label sorteada tem 50% de chance de permanecer inalterada e 50% de chance de ser modificada.

O funcionamento mais específico de cada passo das funções será melhor descrito em forma de comentários no código, desta forma ficará mais organizado e especificado.

3.3.3 Resultados encontrados

As visualizações criadas não foram muito satisfatórias, então elas não constarão neste relatório no atual momento.

Com relação aos exercícios propostos, primeiramente, no exercício 8 é pedido para utilizarmos dados sem transformação no nosso algoritmo de aprendizado da regressão linear. Sabemos que os dados foram gerados de maneira não linear, então o esperado é que a taxa de acerto aplicando essa abordagem seja muito baixa, até porque pela equação utilizada como target (X1^2 + X2^2 - 0.6) sabemos que se trata de uma circunferência, não de uma reta que separa. Para tal proposta, foi criado a função “exercise_8” que reproduz os passos citados acima, e para 100 rodadas de teste, obteve como o erro dentro da amostra médio (Ein) o valor de 0.497.

Com relação ao exercício 9 é pedido para que se faça a alteração devida nos dados de entrada, e retornamos os pesos encontrados pela nossa função g do classificador. Ao executar a função “exercise_9” foram obtidos os seguintes pesos para a nossa nova função g:

g(x1,x2) = sign(−1.13 + 0.032*x1 - 0.021*x2 + 0.117*x1x2 + 1.74*x1^2 + 1.84*x2^2)

Com relação ao exercício 10 é pedido para avaliarmos nossa função g encontrada e medir o erro fora a amostra fazendo a média de 1000 rodadas de execução. O valor encontrado para a média do erro fora da amostra (Eout) foi de 0.06.

Em resumo os valores obtidos foram:

- exercício 8 =0.497 = D
- exercício 9 = sign(−1.13 + 0.032*x1 - 0.021*x2 + 0.117*x1x2 + 1.74*x1^2 + 1.84*x2^2) = A
- exercício 10 = 0.06 = B

3.4. Gradiente Descendente

3.4.1. Sobre seu funcionamento

O algoritmo de Gradiente descendente tem por objetivo minimizar uma função a partir de um ponto inicial. Para conseguir alcançar a esse objetivo são realizadas as derivadas parciais da função a ser minimizada para cada variável do dado de entrada. Após esta etapa, é necessário caminhar sobre a direção contrária do seu gradiente (pois estamos buscando por minimização) com uma velocidade associada (que pode ser comparada com o nosso learning rate). E nós repetimos essa iteração até que a diferença do passo anterior

para o atual seja muito pequena, ou até que o nosso erro seja menor do que um valor mínimo.

3.4.2 Sobre sua implementação

O arquivo correspondente as implementações dos exercícios 11 e 12 da lista (exercícios relacionados ao gradiente descendente) estará em anexo com o nome: “gradiente_descendente.py”.

Para a realização das tarefas foram criadas 2 funções que correspondem às derivadas parciais em direção a “u” e “v” respectivamente: gradiente_u, gradiente_v. A função de erro também é expressa na função “error_function”, e ela será utilizada como condição de parada de nosso loop quando atingir determinado valor. Esta função é dada pelo enunciado, e as funções de gradiente foram calculadas com a ajuda do software “wolfram alpha”, acessível em: https://www.wolframalpha.com/.

Para a realização dos exercícios foram criadas 2 funções “descendent_gradient” e “descendent_coordenate” que irão resolver respectivamente os exercícios 11 e 12. A diferença em suas implementações é como eu atualizo o valor do meu ponto com os valores do gradiente encontrado. Na primeira eu calculo os 2 gradientes direcionais e depois eu atualizo o ponto, na segundo eu faço o primeiro gradiente, atualizo o ponto e depois eu realizo o segundo gradiente.

Para fazer essas atualizações foram criadas as funções “walk_through_gradient” e “walk_through_coordenate_gradient”, que realizam as 2 operações de atualização acima.

O funcionamento mais específico de cada passo das funções será melhor descrito em forma de comentários no código, desta forma ficará mais organizado e especificado.

3.4.3 Resultados encontrados

Com relação aos exercícios propostos, na questão 11, ele pede a quantidade de iterações necessárias para que o valor da função de erro no ponto fique abaixo de 10^(-14). E como resultado foi obtido o valor de 10 iterações para que este valor fosse obtido.

Com relação ao exercício 12, o ponto obtido no final das iterações foi o ponto = [0.044, 0.023]. Na segunda parte dessa questão ele pede para que utilizemos nossa outra rotina implementada, o “descendent_coordenate” para avaliarmos seu resultado com 15 iterações. Porém o resultado encontrado é completamente contra-intuitivo, pois como estamos atualizando o ponto a cada gradiente parcial, intuitivamente parecia que o algoritmo iria convergir mais rapidamente, porém ao fazermos isso com 5 iterações a mais que no exercício anterior foi obtido um resultado de 0.139, que é um erro muito maior que o de 10^(-14).

Em resumo os valores obtidos foram:

- exercício 11 = 10 = D
- exercício 12 = [0.044, 0.023] e 0.139 = E e A

3.5. Regressão Logística

3.5.1. Sobre seu funcionamento

Ela é uma técnica estatística que tem por objetivo classificar de forma categórica os dados de entrada. Seu resultado final é uma probabilidade (entre 0 e 1) de um dado pertencer a uma classe. Para criar essa probabilidade ela utiliza a função: $y = 1 \div (1 + e^{-f(x)})$ em que f(x) é nossa função de pesos e variáveis.

3.5.2 Sobre sua implementação

O arquivo correspondente as implementações dos exercícios 13 e 14 da lista (exercícios relacionados à regressão logistica) estará em anexo com o nome: “regressao_logistica.py”.

Foi-se criado para a realização das atividades uma classe “LogisticRegression”, nela estarão presentes as funções de cálculo do gradiente, para se achar o erro mínimo, a função de fit, para ajustar os pesos aos dados de treinamento da função g e as funções de “loss” para calcular o erro fora da amostra que será utilizado para responder à uma das questões propostas. O vetor de pesos da função é inicializado totalmente zerado.

A partir da explicação acima, teremos nossa função de erro definida como:

$$loss = 1/n * (\sum_{1}^{n} ln(1 + e^{-yn*(w.Xn)}))$$

E nossa função de gradiente definida como:

$$grad = -(y * x) \div (1 + e^{-y*(w.X)})$$

O gradiente será aplicado ao ponto até que a norma da diferença entre os pesos seja menor do que 0.01.

O funcionamento mais específico de cada passo das funções será melhor descrito em forma de comentários no código, desta forma ficará mais organizado e especificado.

3.5.3 Resultados encontrados

Com relação aos exercícios propostos, as duas perguntas poderiam ser respondidas simultaneamente dentro da função main, pois ambas eram respostas sobre a mesma execução. Para estimar o Eout foi criado um novo conjunto de dados após a etapa de treino da função g, e então esses foram avaliados pela função “loss_mean”, que avaliava a perda para cada ponto e fazia a média entre eles (assim como descrito no capítulo acima na nossa função de loss que possui um 1/n, a média, como multiplicador). Fazendo a média de 100 rodadas, o resultado obtido para o Eout médio foi de 0.101.

Para estimar a quantidade de épocas das etapas de aprendizado, uma variável foi criada para guardar a quantidade de iterações que o algoritmo levou para definir a sua função g. Fazendo uma média dessas épocas por 100 rounds foi obtido um resultado de 344.24 épocas por média de rodada.

Em resumo os valores obtidos foram:

- exercício 13 = 0.101 = D
- exercício 14 = 344.24 = A