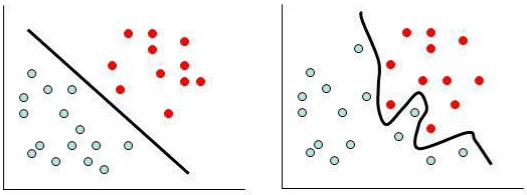
ISABEL SALES DE CASTRO ALMEIDA
DRE:118016977

Perceptron Learning Algorithm (PLA)

Resumo: Este documento corresponde ao Trabalho Prático I do curso de Inteligência Computacional II (CPS849) do Programa de Engenharia de Sistemas e Computação da Coppe - UFRJ. O mesmo visa a implementação do Algoritmo de Aprendizagem Perceptron (PLA). No primeiro capítulo é feita uma breve introdução ao assunto. No segundo capítulo é apresentado a implementação deste algoritmo na linguagem Python e por fim os resultados são expostos no último capítulo.

I. INTRODUÇÃO

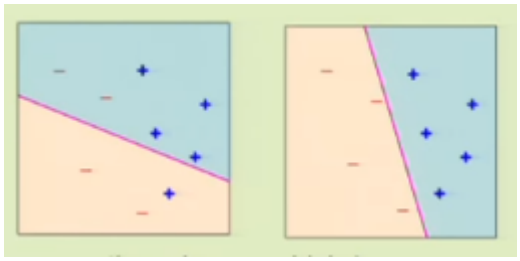
O Perceptron é o tipo mais básico de rede neural. Criado em 1957 por Frank Rosenblatt [1] é um classificador linear, isso significa que ele só irá lidar com problemas onde o conjunto de dados seja linearmente separável. [6]



Linearmente separável

Não linearmente separável

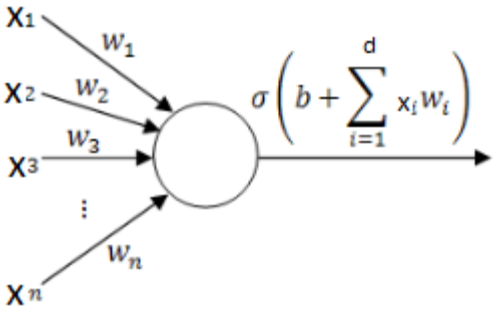
Dado um conjunto de pontos, o modelo de Perceptron é utilizado para fazer a correta separação destes, como pode ser observado na figura:



A fórmula da hipótese do Perceptron é dada por:

$$h(\mathbf{x}) = \text{sign}(\mathbf{w}^T\mathbf{x})$$

Com as entradas e saídas esperadas e os valores iniciais dos pesos (geralmente zero), multiplica-se as entradas e os pesos e em seguida, tudo é somado e avaliado com um valor limite.



Acima, temos o algoritmo representado de forma esquemática onde de **x1**, **x2**, **x3** e **xn** são os inputs e **W1**, **W2**, **W3** e **Wn** são os respectivos pesos de cada input.

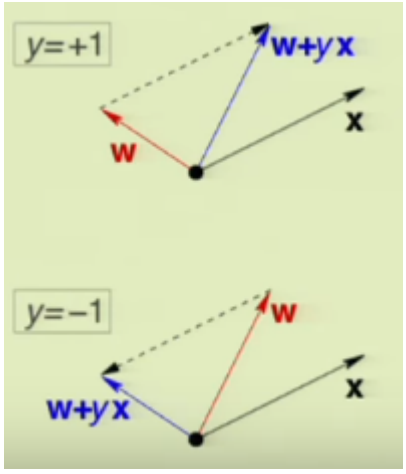
Após a aplicação da fórmula, caso a soma seja positiva, o resultado é 1, caso contrário, -1. Quando o resultado não é o esperado, ou seja, há pontos com classificação incorreta:

$$\text{sign}(\mathbf{w}^T\mathbf{x}_n) \neq y_n$$

Faz se necessário a alteração dos pesos, o que altera a hipótese, de modo que ela se comporte melhor naquele ponto específico:

$$\mathbf{w} \leftarrow \mathbf{w} + y_n\mathbf{x}_n$$

Assim, o ajuste é feito de acordo com a fórmula:



II. IMPLEMENTAÇÃO

O algoritmo foi desenvolvido em Python, por ser uma linguagem de alto nível e de desenvolvimento rápido (RAD - Rapid Application Development). Foi calculado o número médio de iterações de PLA e a média de erro das amostras (discordância entre f e g), para diferentes tamanhos de treinamento. Dessa forma, o programa torna possível comparar o funcionamento do algoritmo para diferentes tipos de dados.
Foram definidos dois tamanhos de treinamento: [10, 100], 1000 foi o número de execuções utilizadas para calcular a média de cada tamanho de treinamento e 1000 pontos de amostra foram utilizados para calcular a discordância entre f e g.

Para visualizar o problema, foi utilizado d = 2 e X = [-1,1] × [-1,1] com probabilidade uniforme de seleção x ∈ X. Em cada execução, uma reta aleatória no plano é escolhida como função *target f* na qual de um lado a reta mapeia +1 e do outro -1. A função *target* é avaliada para cada entrada e uma saída correspondente é obtida. O PLA é inicializado com o vetor de pesos zerado e a cada iteração o algoritmo escolhe um ponto aleatório a partir de um conjunto de pontos classificados incorretamente. Abaixo segue o código do algoritmo comentado:

Algoritmo de Aprendizagem Perceptron (PLA):

```python
import numpy as np
import matplotlib.pyplot as plt


def rnd(n):
    return np.random.uniform(-1, 1, size=n)


def get_E_out_iteracoes(N_pontos):
    execucoes = 1000
    total_iteracoes = 0
    incompatibilidade_total = 0
    N_teste = 1000
    N = N_pontos
    d = 2

    # ------------------------------------------------------------------------------------------------------------------

    for exe in range(execucoes):

        #Escolha de dois pontos aleatórios A, B em [-1,1] x [-1,1]/ para d=2
        A = rnd(d)
        B = rnd(d)

        #A reta pode ser descrita por y = m * x + b onde m é a inclinação
        m = (B[1] - A[1]) / (B[0] - A[0])
        b = B[1] - m * B[0]
        w_f = np.array([b, m, -1])

        # ------------------------------------------------------------------------------------------------------------------

        #Criação de N pontos de dados (x, y) da função de destino
        X = np.transpose(np.array([np.ones(N), rnd(N), rnd(N)]))  #input
        y_f = np.sign(np.dot(X, w_f))  #output

        # ------------------------------------------------------------------------------------------------------------------

        #Escolha da a hipótese h
        w_h = np.zeros(3)  #Inicializa o vetor de peso para a hipótese h
        t = 0  #Conta o número de iterações no PLA


        while True:
            #Inicia o PLA
            y_h = np.sign(np.dot(X, w_h))  #Classificação por hipótese
            comp = (y_h != y_f)  #Comparação da classificação com dados reais da função alvo
            erros = np.where(comp)[0]  #Indices de pontos com classificação errada pela hipótese h

            if erros.size == 0:
                break

            ponto_aleatorio = np.random.choice(erros)  #Escolha de um ponto aleatório mal classificado

            #Atualização do vetor de peso (nova hipótese):
            w_h = w_h + y_f[ponto_aleatorio] * np.transpose(X[ponto_aleatorio])
            t += 1

        total_iteracoes += t
```

```
        #Cálculo do erro
        #Criação de dados "fora" de dados de treinamento
        test_x0 = np.random.uniform(-1, 1, N_teste)
        test_x1 = np.random.uniform(-1, 1, N_teste)

        X_teste = np.array([np.ones(N_teste), test_x0, test_x1]).T

        y_target = np.sign(np.dot(X_teste, w_f))
        y_hipotese = np.sign(np.dot(X_teste, w_h))

        relacao_incompatibilidade = ((y_target != y_hipotese).sum()) / N_teste
        incompatibilidade_total += relacao_incompatibilidade

    # ---------------------------------------------------------------------------------------------------------------

    print("\n---------------------------------------- Relatório Detalhado ----------------------------------------\n")

    print("Tamanho dos dados de treinamento: N = ", N, "pontos")

    iteracoes = total_iteracoes / execucoes
    print("\nNúmero médio de iterações de PLA durante", execucoes, "Iterações: t_avg = ", iteracoes)

    incompatibilidade = incompatibilidade_total / execucoes
    print("\nRazão média para a incompatibilidade entre f (x) e (x) fora dos dados de treinamento:")
    print("P(f(x)!=h(x)) = E_out = ", incompatibilidade)



dados = [10,100]
E_out, iteracoes_dados = [], []

for tamanho in dados:
    incompatibilidade, iteracoes = get_E_out_iteracoes(tamanho)
    E_out.append(incompatibilidade)
    iteracoes_dados.append(iteracoes)


plt.figure(1)
plt.plot(dados, E_out, 'ro')
plt.ylabel("E_out")
plt.xlabel("Dados de treinamento")
plt.savefig('E_out_vs_dados_treinamento.png')

plt.figure(2)
plt.plot(dados, iteracoes_dados, 'bo')
plt.ylabel("Iterações PLA")
plt.xlabel("Dados de treinamento")
plt.savefig('iteracoes_PLA_vs_dados_treinamento.png')

plt.show()

#Relatório
print("\n---------------------------------------- Relatório Final ----------------------------------------\n")
print("Dados de treinamento ", dados)
print("Iterações do PLA: ", iteracoes_dados)
print("P(f(x)!=h(x)) = E_out = ", E_out)
```

Código completo disponível em: https://github.com/isabelsales/PLA

III . RESULTADOS

Foram apresentados quatro problemas e para cada, o algoritmo deveria apresentar qual das soluções informadas é a correta ou a mais próxima do resultado obtido. Foi gerada uma aproximação por meio de uma quantidade de conjuntos separados de pontos e a fim de se obter uma estimativa confiável o experimento foi repetido por 1000 execuções retornando a média destas. Na teoria da probabilidade, a desigualdade de Hoeffding fornece um limite superior na probabilidade de que a soma de variáveis aleatórias independentes limitadas se desvie de seu valor esperado, este conceito fez-se necessário para a análise da discordância entre f e g (P[f(x) != g(x)]).

Primeiro experimento:

1) Para N = 10. Em média quantas iterações são necessárias para que o PLA convirja para N = 10 pontos treinados? Apresente o valor aproximado de seu resultado (resultado próximo à média | sua resposta - opção é próxima de 0).
a) 1
b) 15
c) 300
d) 5000
e) 10000

2) Qual a opção mais se aproxima de P[f(x) != g(x)] para N = 10:
a) 0.001
b) 0.01
c) 0.1
d) 0.5
e)0.8

Resultado obtido pelo algoritmo:

```
---------------------------------------- Relatório Detalhado ----------------------------------------------

Tamanho dos dados de treinamento: N =  10 pontos

Número médio de iterações de PLA durante 1000 Iterações: t_avg =  13.18

Razão média para a incompatibilidade entre f (x) e (x) fora dos dados de treinamento:
P(f(x)!=h(x)) = E_out =  0.107674
```

Para N = 10, o algoritmo retorna **13.18**, como média das iterações do PLA durante as **1000** execuções. Deste modo, é possível concluir que a letra **b**, cuja alternativa apresenta o número **15**, é a opção mais próxima do valor alcançado pelo algoritmo, sendo esta, a alternativa correta dentre as opções apresentadas para a questão **1**. Para o cálculo da razão média da discordância entre f e g (E_out), o algoritmo apresentou o valor **0.107674**, sendo a alternativa **c**, a mais próxima dentre as opções apresentadas pela para a questão **2**.

Segundo experimento:

3) Agora, teste N= 100. Em média quantas iterações são necessárias para que o PLA convirja para N = 100 pontos de treinamento? Informe o valor mais próximo ao seu resultado.
a) 50
b) 100
c) 500
d) 1000
e)5000

4) Qual a opção mais se aproxima de P[f(x) != g(x)] para N = 100:
a) 0.001
b) 0.01
c) 0.1
d) 0.5
e)0.8

Resultado obtido pelo algoritmo:

```
---------------------------------------- Relatório Detalhado ----------------------------------------------

Tamanho dos dados de treinamento: N =  100 pontos

Número médio de iterações de PLA durante 1000 Iterações: t_avg =  110.376

Razão média para a incompatibilidade entre f (x) e (x) fora dos dados de treinamento:
P(f(x)!=h(x)) = E_out =  0.012403
```

Neste novo cenário, com N = 100, o algoritmo retorna **110.376**, como média das iterações do PLA durante as **1000** execuções. Deste modo, é possível concluir que a letra **b**, cuja alternativa apresenta o número **100**, é a opção mais próxima do valor alcançado pelo algoritmo, sendo esta, a alternativa correta dentre as opções apresentadas para a questão **3**. Para o cálculo da razão média da discordância entre f e g (E_out), o algoritmo apresentou o valor **0.012403**, sendo a alternativa **b**, a mais próxima dentre as opções apresentadas pela para a questão **2**.

Relatório Final:

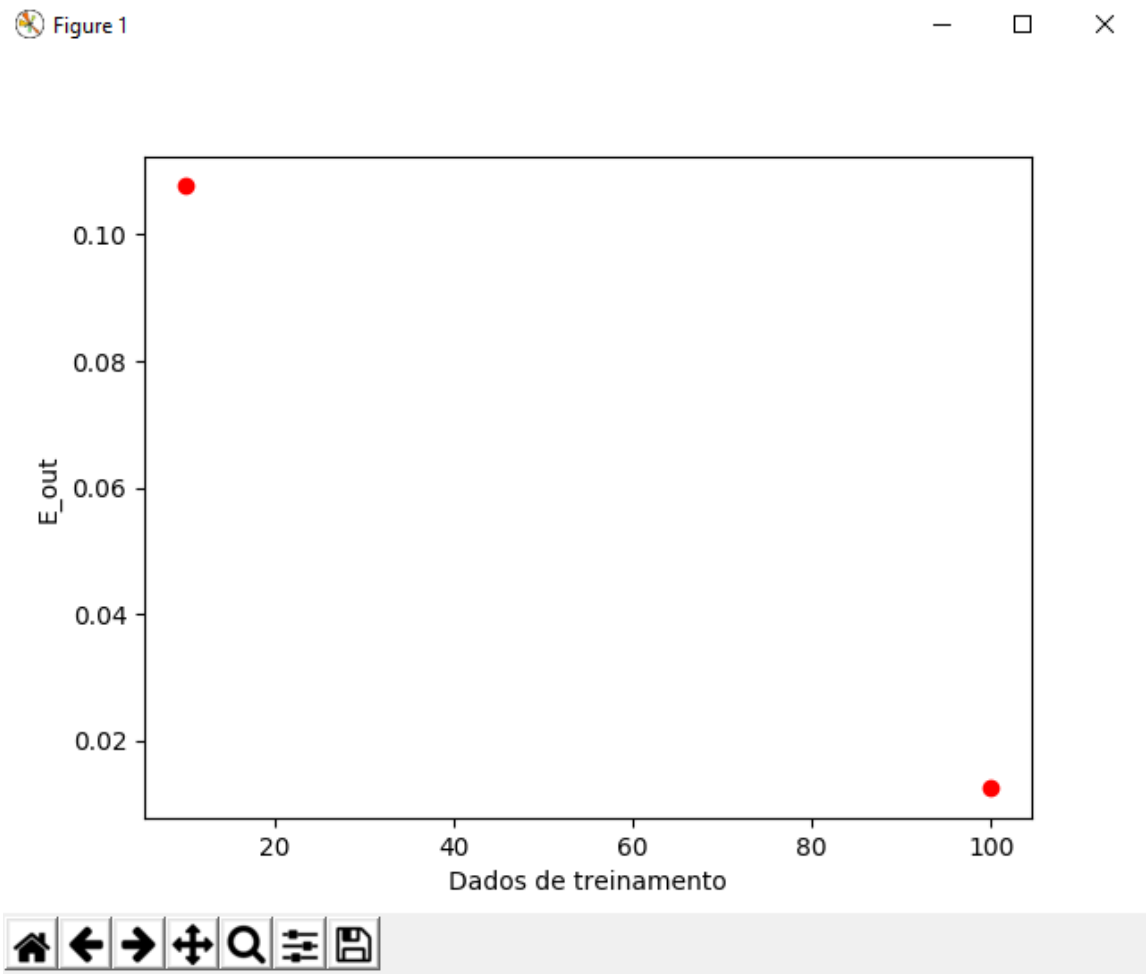
De acordo com o os resultados obtidos para os conjuntos de treinamento [10,100], foi possível obter a média de iterações do PLA durante 1000 execuções e a razão média da discordância entre f e g (E_out), conforme resumido na figura abaixo:

Resultado obtido pelo algoritmo:

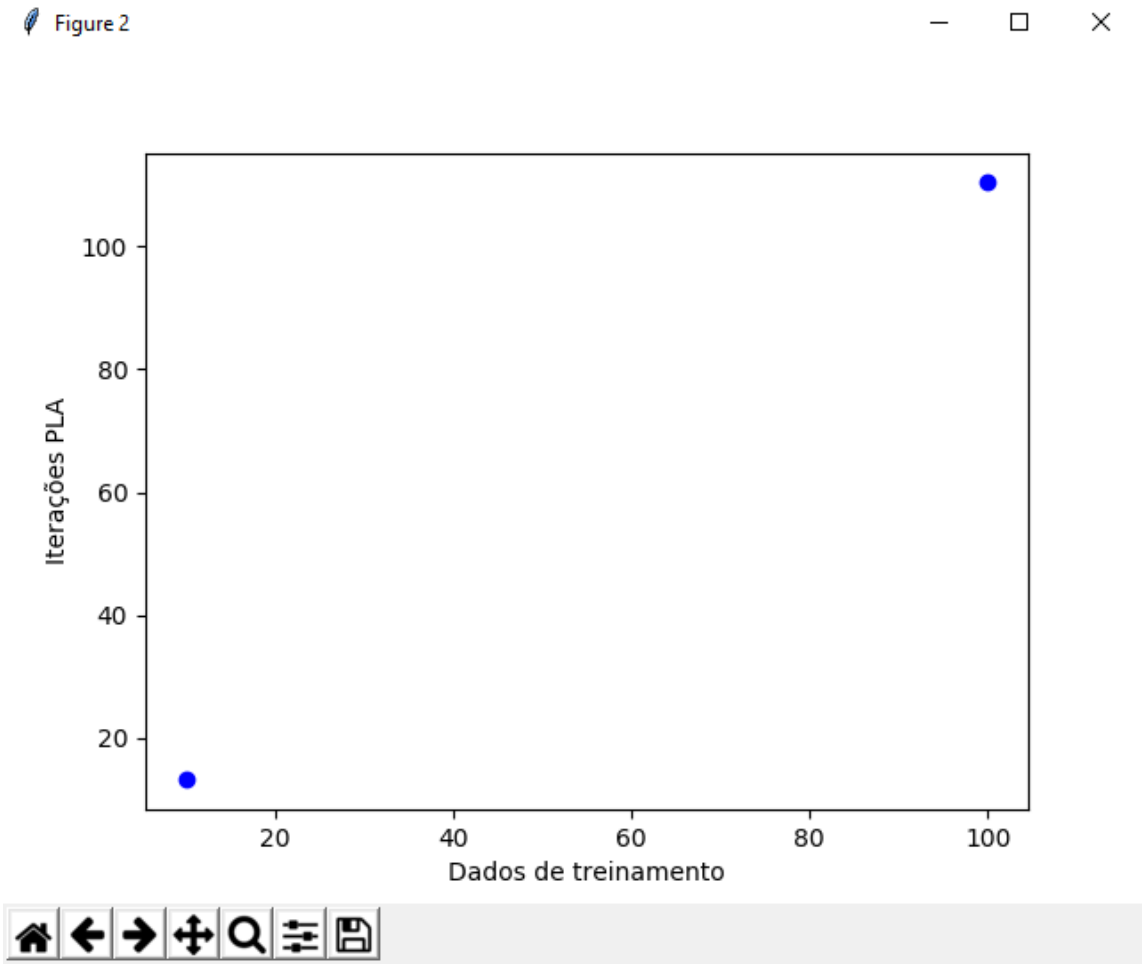
```
---------------------------------------------- Relatório Final ----------------------------------------------

Dados de treinamento  [10, 100]
Iterações do PLA:  [13.18, 110.376]
P(f(x)!=h(x)) = E_out =  [0.10767400000000006, 0.012402999999999982]

Process finished with exit code 0
```

Abaixo segue as representações dos dados de treinamento versus o E_out e as iterações do PLA.

Representação E_out vs dados de treinamento:



Representação das Iterações do PLA vs dados de treinamento:



IV. REFERÊNCIAS

[1] Perceptron - Uma breve introdução, Disponível em: <http://redesneuraisartificiais.blogspot.com/2011/06/perceptron-uma-breve-explicacao.html>. Acesso em 25 de julho de 2018.

[2] Single Layer Perceptron as Linear Classifier, Disponível em: <https://www.codeproject.com/Articles/125346/Single-Layer-Perceptron-as-Linear-Classifier>. Acesso em 25 de julho de 2018.

[3] Lecture 01 - The Learning Problem, Disponível em: <https://www.youtube.com/watch?v=mbyG85GZ0PI&hd=1>. Acesso em 25 de julho de 2018.

[4] Lecture 02 - Is Learning Feasible?, Disponível em: <https://www.youtube.com/watch?v=MEG35RDD7RA>. Acesso em 25 de julho de 2018.

[5] Pyplot tutorial, Disponível em: <https://matplotlib.org/users/pyplot_tutorial.html>. Acesso em 26 de julho de 2018.

[6] Introdução ao Perceptron - Passo a passo, Disponível em: <https://juliocprocha.wordpress.com/2017/07/27/perceptron-para-classificacao-passo-a-passo/>. Acesso em 37 de julho de 2018.