Universidade Federal do Rio de Janeiro
Instituto Alberto Luiz Coimbra de Pós-Graduação e Pesquisa de Engenharia

Programa de Engenharia de Sistemas e Computação

CPS844 - Inteligência Computacional I

Prof. Dr. Carlos Eduardo Pedreira

Trabalho prático

Luiz Henrique Souza Caldas
email: lhscaldas@cos.ufrj.br

20 de maio de 2024

1 Perceptron

Neste problema, você criará a sua própria função target f e uma base de dados D para que possa ver como o Algoritmo de Aprendizagem Perceptron funciona. Escolha $d = 2$ pra que você possa visualizar o problema, e assuma $\chi = [-1, 1] \times [-1, 1]$ com probabilidade uniforme de escolher cada $x \in \mathcal{X}$.

Em cada execução, escolha uma reta aleatória no plano como sua função target f (faça isso selecionando dois pontos aleatórios, uniformemente distribuídos em $\chi = [-1, 1] \times [-1, 1]$, e pegando a reta que passa entre eles), de modo que um lado da reta mapeia pra +1 e o outro pra -1. Escolha os inputs x_n da base de dados como um conjunto de pontos aleatórios (uniformemente em $\mathcal{X}$), e avalie a função target em cada x_n para pegar o output correspondente y_n.

Agora, pra cada execução, use o Algoritmo de Aprendizagem Perceptron (PLA) para encontrar g. Inicie o PLA com um vetor de pesos w zerado (considere que $sign(0) = 0$, de modo que todos os pontos estejam classificados erroneamente ao início), e a cada iteração faça com que o algoritmo escolha um ponto aleatório dentre os classificados erroneamente. Estamos interessados em duas quantidades: o número de iterações que o PLA demora para convergir pra g, e a divergência entre f e g que é $\mathbb{P}[f(x) \neq g(x)]$ (a probabilidade de que f e g vão divergir na classificação de um ponto aleatório). Você pode calcular essa probabilidade de maneira exata, ou então aproximá-la ao gerar uma quantidade suficientemente grande de novos pontos para estimá-la (por exemplo, 10.000).

A fim de obter uma estimativa confiável para essas duas quantias, você deverá realizar 1000 execuções do experimento (cada execução do jeito descrito acima), tomando a média destas execuções como seu resultado final.

Para ilustrar os resultados obtidos nos seus experimentos, acrescente ao seu relatório gráficos scatterplot com os pontos utilizados para calcular E_{out}, assim como as retas correspondentes à função target e à hipótese g encontrada.

Implementação:

Para responder os itens referentes a este problema, foi implementado em Python um Perceptron 2D utilizando as fórmulas e procedimentos apresentados na primeira aula do professor Yaser Abu-Mostafa. Foram criadas duas classes: uma para gerar a base de dados e a função target (código 1) e outra pra criar e treinar o perceptron (código 2). Ambos os códigos estão listados abaixo.

Código 1: Geração do da base de dados D

```
# Classe para criar o dataset e a função target
class Dataset:
    def __init__(self, N):
        self.N = N # tamanho do dataset
        self.a = 0 # coeficiente angular
        self.b = 0 # coeficiente linear

    # Método para gerar a linha da função target
    def generate_random_line(self):
        point1 = np.random.uniform(-1, 1, 2) # ponto aleatorio no domínio
        point2 = np.random.uniform(-1, 1, 2) # ponto aleatorio no domínio
        a = (point2[1] - point1[1]) / (point1[0] - point2[0]) # cálculo do
            coeficiente angular
        b = point1[1] - a*point1[0] # cálculo do coeficiente linear
        self.a = a
```

```
            self.b = b
            return a, b

        # Método para classificar pontos de acordo a função target
        def classify_point(self, point):
            a = self.a
            b = self.b
            y_reta = a*point[0] + b
            return np.sign(point[1] - y_reta) # verifica se a coordenada y do ponto está acima ou abaixo da reta

        # Método para gerar a base de dados D
        def generate_dataset(self):
            N = self.N
            data = np.random.uniform(-1, 1, (N, 2)) # gera N pontos no R2 com coordenadas entre [-1, 1]
            labels = np.array([self.classify_point(point) for point in data])
            return data, labels
```

Código 2: Perceptron

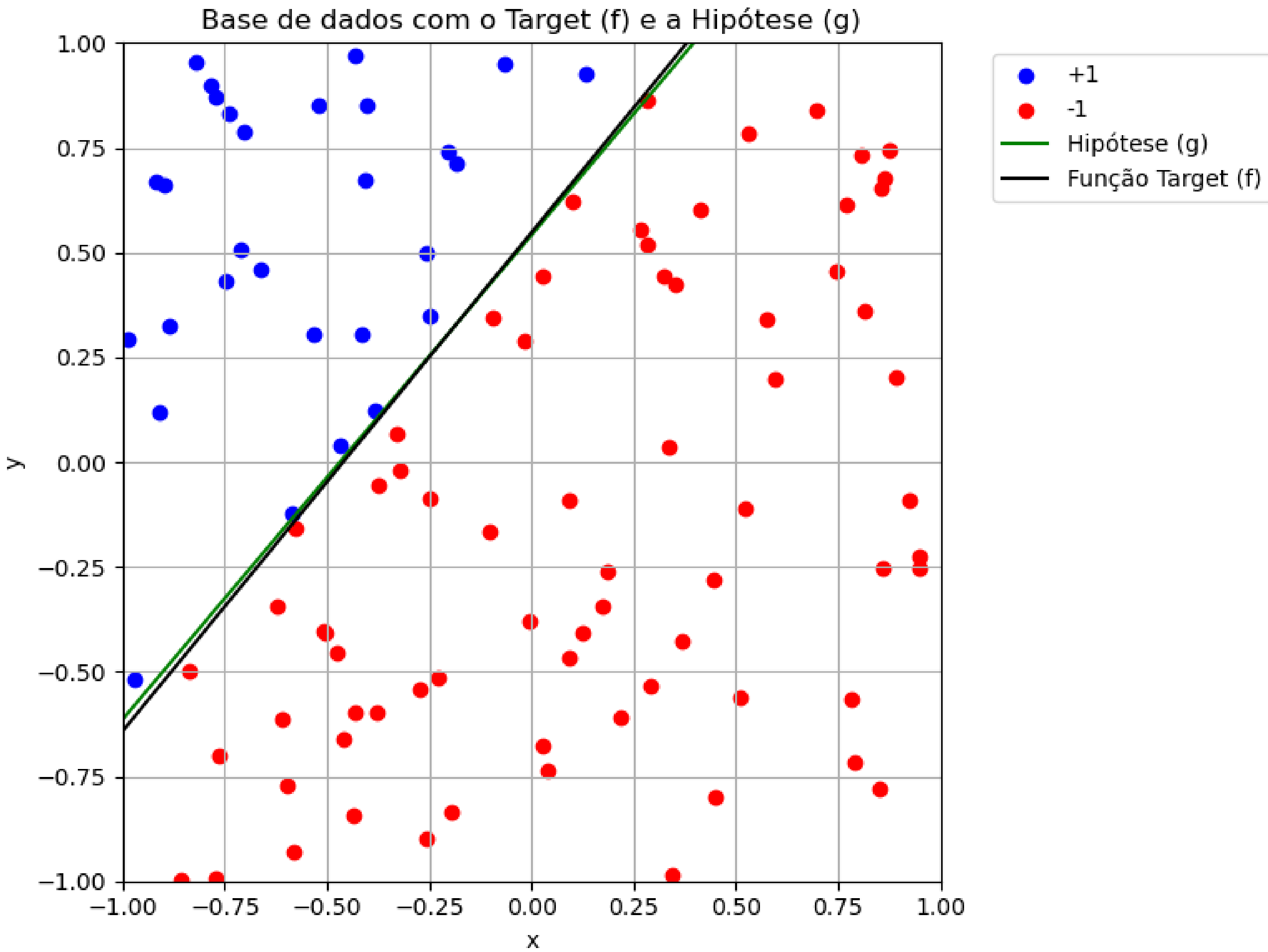
```
    # Classe para criar e treinar o perceptron 2D
    class Perceptron2D:
        def __init__(self, max_iter=10000):
            self.max_iter = max_iter
            self.w = np.zeros(3)  # inicializa os pesos (incluindo o w_0)

        # Método para treinar o perceptron usando o algoritmo de aprendizagem perceptron (PLA)
        def fit(self, data, labels):
            n_samples = len(data)
            X_bias = np.hstack([np.ones((n_samples, 1)), data]) # adiciona uma coluna de 1s para o X_0 (coordenada artificial)
            iterations = 0
            errors = 1
            while (errors > 0) and (iterations <= self.max_iter):
                errors = 0
                for i in range(n_samples):
                    if labels[i] * np.dot(self.w, X_bias[i]) <= 0:
                        self.w += labels[i] * X_bias[i] # atualiza os pesos
                        errors += 1
                iterations += 1
            return iterations, self.w

        # Método para classificar um dataset com base nos pesos aprendidos.
        def classificar(self, data):
            n_samples = len(data)
            X_bias = np.hstack([np.ones((n_samples, 1)), data]) # adiciona uma coluna de 1s para o bias X_0
            return np.sign(np.dot(X_bias, self.w)) # verifica o sinal do produto escalar entre x e w
```

Para testar as classes, foi feita uma função para plotar uma base de dados de 100 pontos com a função target f gerada e a hipótese g com a reta calculada pelo PLA (código 3). O resultado pode ser observado na figura 1.

Código 3: Teste das classes

```
def teste():
    # Criar o dataset e a função targe
    num_points = 100
    dataset = Dataset(num_points)
    a, b = dataset.generate_random_line()
    data, labels = dataset.generate_dataset()
    # Criar e treinar o perceptron
    perceptron = Perceptron2D()
    _, w = perceptron.fit(data,labels)
    # Plotar resultados
    plt.figure(figsize=(8, 6))
    x_pos = [data[i][0] for i in range(len(data)) if labels[i] == 1]
    y_pos = [data[i][1] for i in range(len(data)) if labels[i] == 1]
    x_neg = [data[i][0] for i in range(len(data)) if labels[i] == -1]
    y_neg = [data[i][1] for i in range(len(data)) if labels[i] == -1]
    plt.scatter(x_pos, y_pos, c='blue', label='+1')
    plt.scatter(x_neg, y_neg, c='red', label='-1')
    x = np.linspace(-1, 1, 100)
    y_target = a*x+b
    y_g = -(w[1] * x + w[0]) / w[2]
    plt.plot(x, y_g, 'g-', label='Hipótese (g)')
    plt.plot(x, y_target, 'k-', label='Função Target (f)')
    plt.xlim(-1, 1)
    plt.ylim(-1, 1)
    plt.xlabel('x')
    plt.ylabel('y')
    plt.title('Base de dados com o Target (f) e a Hipótese (g)')
    plt.legend(bbox_to_anchor=(1.05, 1), loc='upper left')
    plt.tight_layout(rect=[0, 0, 1, 1])
    plt.grid(True)
    plt.show()
```

Figura 1: Base de dados com o Target (f) e a Hipótese (g)



1. Considere $N = 10$. Quantas iterações demora, em média, para que o PLA convirja com $N = 10$ pontos de treinamento? Escolha o valor mais próximo do seu resultado.

   (a) 1
   (b) 15
   (c) 300
   (d) 5000
   (e) 10000

   Justificativa:

   Para responder a esse item foi implementada a seguinte função:

   Código 4: Cálculo do número de iterações

```
        def calc_num_iter(num_points):
```

```python
        num_iter = list()
        for _ in range(1000):
            dataset = Dataset(num_points)
            dataset.generate_random_line()
            data, labels = dataset.generate_dataset()
            perceptron = Perceptron2D()
            iter, _ = perceptron.fit(data,labels)
            num_iter.append(iter)

        print(f"{np.mean(num_iter)} iterações com desvio padrão {np.
            std(num_iter):.4f} (min:{np.min(num_iter)}, máx:{np.max(
            num_iter)})")
```

O resultado após 1000 execuções do experimento, com a variável *num_points* $= 10$, foi uma média de 5.0490(≈ 5) iterações, com desvio padrão de 7.7770(≈ 8) iterações, mínimo de 2 iterações e máximo de 99 iterações. Nota-se que o número de iterações pode variar bastante entre uma iteração e outra. Como 5 está mais próximo de 1 do que de 15, o **item a** foi selecionado.

2. Qual das alternativas seguintes é mais próxima de $\mathbb{P}[f(x) \neq g(x)]$ para $N = 10$?

   (a) 0.001
   (b) 0.01
   (c) 0.1
   (d) 0.5
   (e) 1

Justificativa:

$\mathbb{P}[f(x) \neq g(x)]$ pode ser estimada computacionalmente gerando-se uma quiantidade suficientemente grande de pontos novos e calculando o percentual de erro na classificação desses pontos. Para responder esse item foram realizadas 1000 execuções, nas quais foram gerados 10010 pontos, sendo 10 utilizados para o treinamento do perceptron e 10 mil utilizados para teste. A cada execução foi calculado o perceltual de erro, isto é, a quantidade de vezes que a classificação do perceptron foi diferente da função target divido pela quantidade de pontos. No final das execuções, $\mathbb{P}[f(x) \neq g(x)]$ foi estimada fazendo a média do percentual de erro em cada execução. A Implementação pode ser vista abaixo:

Código 5: Cálculo da probabilidade de erro

```python
    def calc_p_erro(num_points, train_size):
        lista_erro = list()
        for _ in range(1000):
            dataset = Dataset(num_points)
            data, labels = dataset.generate_dataset()
            x_train, x_test, y_train, y_test = train_test_split(
                data, labels, train_size=train_size, stratify = labels
                )
            perceptron = Perceptron2D()
            perceptron.fit(x_train,y_train)
```

```
                y_predicted = perceptron.classificar(x_test)
                erro = np.mean(y_test != y_predicted)
                lista_erro.append(erro)

            print(f"P[f(x)\u2260g(x)] = {np.mean(lista_erro):.4f}")
```

O resultado após 1000 execuções, com *num_points* = 10010 e *train_size* = 10, foi $\mathbb{P}[f(x) \neq g(x)] = 0.0671 = 6.71\%$. Como 0.0671 está mais próximo de 0.1 do que de 0.01, o **item c** foi selecionado.

3. Agora considere $N = 100$. Quantas iterações demora, em média, para que o PLA convirja com N = 100 pontos de treinamento? Escolha o valor mais próximo do seu resultado.

   (a) 50

   (b) 100

   (c) 500

   (d) 1000

   (e) 5000

   Justificativa:

   Para responder a este item foi utilizada a mesma função do item 1 (código 4), com *num_points* = 100.

   O resultado após 1000 execuções do experimento foi uma média de 32.981($\approx$ 33) iterações, com desvio padrão de 164.5924($\approx$ 165) iterações, mínimo de 2 iterações e máximo de41 49 iterações. Nota-se que novamente o número de iterações pode variar bastante entre uma iteração e outra. Como 33 está abaixo de 50 e não existe alternativa menor, o **item a** foi selecionado.

4. Qual das alternativas seguintes é mais próxima de $\mathbb{P}[f(x) \neq g(x)]$ para $N = 100$?

   (a) 0.001

   (c) 0.01

   (c) 0.1

   (d) 0.5

   (e) 1

   Justificativa:

   Para responder a este item foi utilizada a mesma função do item 2 (código 5), com *num_points* = 10100 e *train_size* = 100.

   O resultado após 1000 execuções foi $\mathbb{P}[f(x) \neq g(x)] = 0.0069 = 0.69\%$. Como 0.0069 está mais próximo de 0.01 do que de 0.001, o **item b** foi selecionado.

5. É possível estabelecer alguma regra para a relação entre N, o número de iterações até a convergência, e $\mathbb{P}[f(x) \neq g(x)]$?

Resposta:

Para responder a este item foram implementadas duas funções: uma para calcular o número de iterações e $\mathbb{P}[f(x) \neq g(x)]$ para uma faixa de diferentes números de pontos (código 6) e outra para plotar os resultados (código 7). O código das duas pode ser observado abaixo.

Código 6: Cálculo da probabilidade de erro e do número de iterações

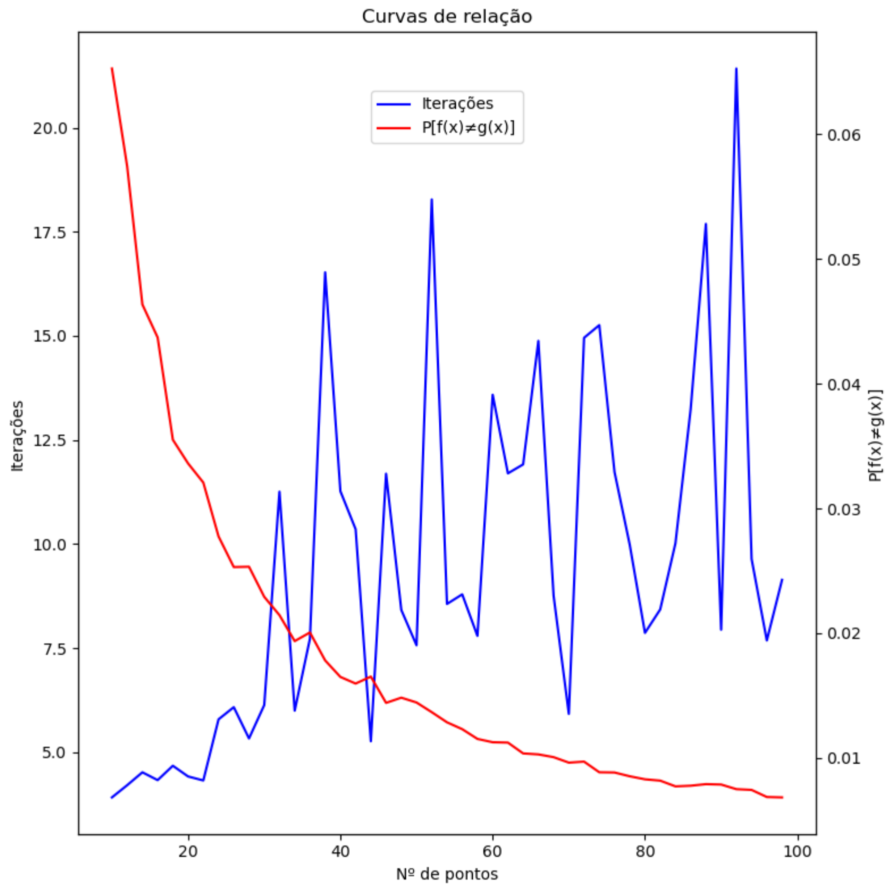
```
        def relationship(num_points_list):
            num_iter_medio = list()
            lista_erro_medio = list()
            for num_points in num_points_list:
                num_iter = list()
                lista_erro = list()
                for _ in range(1000):
                    dataset = Dataset(10000 + num_points)
                    data, labels = dataset.generate_dataset()
                    x_train, x_test, y_train, y_test = train_test_split(
                        data, labels, train_size=num_points, stratify = labels)
                    perceptron = Perceptron2D()
                    iter, _ = perceptron.fit(x_train,y_train)
                    num_iter.append(iter)
                    y_predicted = perceptron.classificar(x_test)
                    erro = np.mean(y_test != y_predicted)
                    lista_erro.append(erro)
                num_iter_medio.append(np.mean(num_iter))
                lista_erro_medio.append(np.mean(lista_erro))
            return num_iter_medio, lista_erro_medio
```

Código 7: Plot da probabilidade de erro e do número de iterações

```
        def plot_relationship(num_points_list, num_iter_medio, lista_erro_medio):
            fig, ax = plt.subplots(1, 1, figsize=(8, 8))
            ax.plot(num_points_list,num_iter_medio,c="blue",label="Iterações")
            ax.set_title("Curvas de relação")
            ax.set_xlabel('Nº de pontos')
            ax.set_ylabel('Iterações')
            ax2=ax.twinx()
            ax2.plot(num_points_list,lista_erro_medio,c="red", label='P[f(x)\u2260g(x)]')
            ax2.set_ylabel('P[f(x)\u2260g(x)]')
            fig.legend(loc='upper center', bbox_to_anchor=(0.5, 0.9))
            fig.tight_layout()
        plt.show()
```

O resultado para o tamanho N da amostra variando de 10 a 100 com passo 2 pode ser observado na figura 2.

Figura 2: Resultado para o tamanho do dataset variando de 10 a 100 com passo 2



A quantidade de iterações para o PLA convergir, como constatado nos itens 1 e 3, oscila bastante, porém, pela figura, é possível observar que ela tem uma tendencia de alta conforme N aumenta. Já a probabilidade de erro $\mathbb{P}[f(x) \neq g(x)]$ visivelmente apresenta uma queda exponencial com o aumento de N. Conclui-se que, com o aumento de N, o algoritmo se torna mais confiável, porém demora mais para ser treinado.

2 Regressão Linear

Nestes problemas, nós vamos explorar como Regressão Linear pode ser usada em tarefas de classificação. Você usará o mesmo esquema de produção de pontos visto na parte acima do Perceptron, com $d = 2$, $\mathcal{X} = [-1, 1] \times [-1, 1]$, e assim por diante.

1. Considere $N = 100$. Use Regressão Linear para encontrar g e calcule E_{in}, a fração de pontos dentro da amostra que foram classificados incorretamente (armazene os g's pois eles serão usados no item seguinte). Repita o experimento 1000 vezes. Qual dos valores abaixo é mais próximo do E_{in} médio?

2. Agora, gere 1000 pontos novos e use eles para estimar o $Eout$ dos g's que você encontrou no item anterior. Novamente, realize 1000 execuções. Qual dos valores abaixo é mais próximo do E_{out} médio?

3. Agora, considere $N = 10$. Depois de encontrar os pesos usando Regressão Linear, use-os como um vetor de pesos iniciais para o Algoritmo de Aprendizagem Perceptron (PLA). Execute o PLA até que ele convirja num vetor final de pesos que separa perfeitamente os pontos dentro-de-amostra. Dentre as opções abaixo, qual é mais próxima do número médio de iterações (sobre 1000 execuções) que o PLA demora para convergir?

4. Vamos agora avaliar o desempenho da versão pocket do PLA em um conjunto de dados que não é linearmente separável. Para criar este conjunto, gere uma base de treinamento com N2 pontos como foi feito até agora, mas selecione aleatoriamente 10% dos pontos e inverta seus rótulos. Em seguida, implemente a versão pocket do PLA, treine-a neste conjunto não-linearmente separável, e avalie seu E_{out} numa nova base de N2 pontos na qual você não aplicará nenhuma inversão de rótulos. Repita para 1000 execuções, e mostre o E_{in} e E_{out} médios para as seguintes configurações (não esqueça dos gráficos scatterplot, como anteriormente):

3 Regressão Não-Linear

Nestes problemas, nós vamos novamente aplicar Regressão Linear para classificação. Considere a função target

$$f(x_1, x_2) = sign(x_1^2 + x_2^2 - 0.6)$$

Gere um conjunto de treinamento de $N = 1000$ pontos em $\mathcal{X} = [-1, 1] \times [-1, 1]$ com probabilidade uniforme escolhendo cada $x \in \mathcal{X}$. Gere um ruído simulado selecionando aleatoriamente 10% do conjunto de treinamento e invertendo o rótulo dos pontos selecionados.

1. Execute a Regressão Linear sem nenhuma transformação, usando o vetor de atributos $(1, x_1, x_2)$ para encontrar o peso w. Qual é o valor aproximado de classificação do erro médio dentro da amostra E_{in} (medido ao longo de 1000 execuções)?

2. gora, transforme os $N = 1000$ dados de treinamento seguindo o vetor de atributos não-linear $(1, x_1, x_2, x_1x_2, x_1^2, x_2^2)$. Encontre o vetor $\tilde{w}$ que corresponda à solução da Regressão Linear. Quais das hipóteses a seguir é a mais próxima à que você encontrou? Avalie o resultado médio obtido após 1000 execuções.

3. Qual o valor mais próximo do erro de classificação fora da amostra E_{out} de sua hipótese na questão anterior? (Estime-o gerando um novo conjunto de 1000 pontos e usando 1000 execuções diferentes, como antes).