CENTRO UNIVERSITÁRIO FEI
Leonardo Contador Neves – 118315-1

Tópicos Especiais em Aprendizagem:

Perceptron

São Bernardo do Campo, SP
2018

SUMÁRIO

1 INTRODUÇÃO

Este relatório busca a implementação do algoritimo Perceptron, um algoritmo de classificação que faz o uso de similaridades do funcionamento biológico de um neurônio para criar um modelo matemático que pode separar linearmente dois estados ou classes.

2 REVISÃO BIBLIOGRÁFICA

O Perceptron foi desenvolvido nas décadas de 1950 e 1960 pelo cientista Frank Rosenblatt, inspirado em trabalhos anteriores de Warren McCulloch e Walter Pitts. É um modelo matemático que busca se espelhar no funcionamento de um neurônio biológico; Enquanto nos neurônios reais o dendrito recebe sinais elétricos dos axônios e de outros neurônios (estruturas básicas de neurônios biológicos demostrada na imagem 1), no Perceptron estes sinais elétricos são representados como valores numéricos. Nas sinapses entre dendritos e axônio, os sinais elétricos são modulados em várias quantidades. Isso também é modelado no Perceptron multiplicando cada valor de entrada por um valor chamado peso.

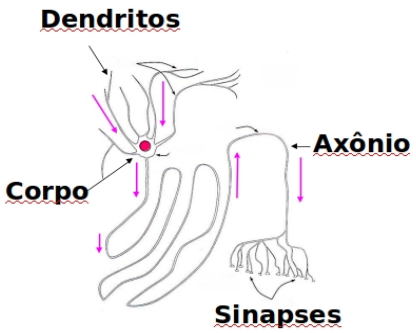


Imagem 1: Representação de um neurônio genérico e sua estrutura básica.

Um neurônio real dispara um sinal de saída somente quando a força total dos sinais de entrada excede um certo limiar. Esse fenômeno é modelado então em um Perceptron calculando a soma ponderada das entradas para representar a força total dos sinais de entrada e aplicando uma função de ativação na soma para determinar sua saída. Tal como nas redes neurais biológicas, esta saída é alimentada em outros Perceptrons.

O Perceptron então é uma função matemática, ou modelo, que mapeia as entradas e produz um valor binário de saída.

3 METODOLOGIA

A implementação do algoritmo a ser estudado neste relatório basea-se em duas frentes, a de treinamento do algoritmo, onde serão vistos métodos de obtenção e atualização dos parâmetros do nosso Perceptron e a de predição dos valores novos, que é responsável por nos dizer a qual grupo o dado de entrada pertece.

Basicamente o Perceptron é um classificador binário que mapeia todas suas entradas (vetor x1, … ,xn) para um novo valor de saída (vetor de F(x1), … , F(xn)) por meio de uma função de ativação simples que pode ser descrita na imagem à seguir. Essa função, uma vez que temos nosso vetor de entrada, vamos usar os “pessos” (vetor w), para compor uma somatória que será usada para a Equação 1.

$$f(x) = \begin{cases} 1 \; se \; w \times x + b \geq 0 \\ 0 \; caso \; contrário. \end{cases}$$

Equação 1: Função básica de ativação do Perceptron.

Com base em nossa função limiar de ativação do Perceptro, podemos notar, na imagem a seguir, que a resposta simples do modelo de um neurônio biológico é o vetor de entradas (que representa os dados de sensores ou de outros neurônios) vezes o vetor de pesos (que representa o peso que cada entrada tem na composição da resposta final) mais um valor de *bias* (que pode ser entendido como um valor que pode deslocar linearmente a função do neurônio aprendida de modo que não passe no ponto 0 do plano cartesiano para todas as soluções) e chegamos num valor que pela função de ativação limiar podemos ver à que classe binária o dado pertence.

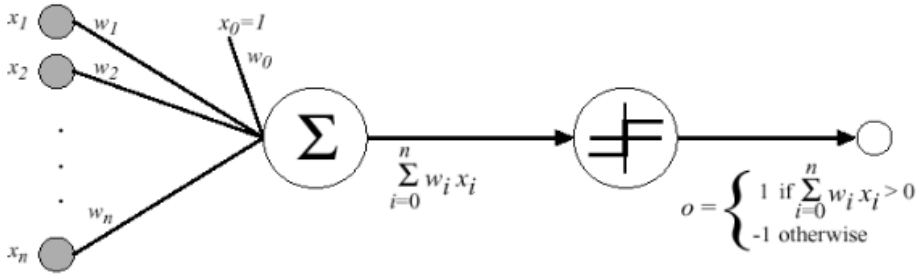


Imagem 2: Funcionamento de um Perceptron.

Uma vez que definimos como o Perceptron faz o mapeamento das entradas para fornecer uma saída que presenta o dado classificado, vamos agora entender um pouco mais sobre como podemos ensinar o algoritmo a função para chegar nesse intuito. Basicamente, o processo de aprendizagem se da pela atualização do vetor de pesos, que são os responsáveis por nos dizer o peso que cada entrada tem na composição de uma resposta para a função de ativação.

Podemos representar a atualização dos pesos com base na somatória do meu estado inicial mais um valor *deltaW*. Esse valor de *deltaW* pode ser composto pelo erro de cada iteração de aprendizagem, uma variável que controla o quanto ele aprende a cada iteração e o valor de entrada. A regra de atualização dos pesos (vetor w) pode ser representada na imagem a seguir.

$$w_i = w_i + \Delta w_i$$

Onde: $$\Delta w_i = \eta\,(t - o)\,x_i$$

Equação 2: Regra de atualização do vetor de pesos W.

Para melhores resultados, a função de aprendizado é iterada diversas vezes a partir do nosso conjunto de dados. As iterações são verificadas ao final para análise do erro da variável a ser predita e podemos tomar a decisão de parar as iterações com base então no erro tendendo ao valor zero.

4 DESENVOLVIMENTO

A implementação do algoritmo de classificação Perceptron foi feita neste trabalho através de uma classe na linguagem de programação *Python* que tem dois métodos principais. O método *fit()*, que faz o treinamento do Perceptron através da atualização dos pesos e o método *predict()* que faz a predição dos novos valores.

4.1 CLASSE PARA CÁLCULO DO CLASSIFICADOR PERCEPTRON

4.1.1 Método de treinamento *fit*

Esse método, fazendo analogia à teoria, foi criado para treinamento do Perceptron através da atualização dos pesos. Basicamente, o método espera valores de entrada “X” e valores de classificação”Y” e então os calculos são realizados.

Duas estruturas de laços são analisadas na figura a seguir que são a base do treinamento do neurônio artificial. A primeira delas é responsável por controlar quantas iterações a etapa de aprendizado vai ter através da variável de controle *self.iterations*. Essa variável controla quantas fazes de aprendizado o algoritmo vai passar pelo conjunto de dados e tem uma estrutura condicional, que se o erro (variável de contagem *count_error*) for zero as iterações se acabam mesmo não atingindo o valor final da variável *self.iterations.*

```
def fit(self, X, Y):
    for i in range(self.iteractions):
        # Começa uma nova iteração com contagem de erro = zero
        count_error = 0
        for inputs, label in zip(X, Y):
            #Calcula valor predito
            prediction = self.predict(inputs)
            # Verifica erro, se erro é diferente de zero, atualiza os pesos
            error = (label - prediction)
            if error != 0:
                count_error += 1
                self.weights[1:] = self.weights[1:] + (self.learning_rate * error * inputs)
                self.weights[0] = self.weights[0] + (self.learning_rate * error)
        print("Iteraction: {}".format(i))
        # Finaliza as iterações se todos os dados não possuem erro
        if count_error == 0:
            print("FINESHED")
            break
```

A variável *count_error* começa cada iteração com o valor de zero e incrementa toda vez que no conjunto de dados, o valor predito não condiz com o valor real da classe daquela parte dos dados. A medida que a variável *error* é medida através da subtração entre o valor real e o valor predito se a resultante for diferente de zero os pesos são atualizados

simplesmete somando eles mesmos mais o produto da variável de controle de aprendizado (*learning_rate)*, o valor do erro e o valor da entrada.

O foco principal deste método então é atualizar os pesos, treinando-os com base nas regras descritas, para que possamos prever valores futuros que não pertencem ao conjunto de dados treinado.

4.1.1 Método de classificação *predict*

O método de predição pode ser entendido como um processo matemático que diz, com base nos dados treinados, a que classe os dados de entrada pertencem. Essa função também é usada intrinsecamente no método de treinamento, a medida que o erro precisa ser calculado para que possamos atualizar os pesos.

Para o cálculo deste método, a função tem como entrada os valores *inputs* a serem preditos e basicamente, tem dois passos para fornecer uma resposta. No primeiro passo, vamos calcular a variável *sum*, que é descrita na imagem 2 deste relatório, onde cada entrada é multiplicada por um peso, somada uma a uma e acrescida do bias. No segundo passo, a etapa de ativação do neurônio artificial, a variável *sum* é comparada com o valor zero onde, se *sum* for maior que zero o neurônio é ativado e a resposta para a função *predict* é o valor de um, se não o retorno da função assume valor zero e analogicamente, o neurônio é inibido. A imagem a seguir mostra o processo descrito.

```
def predict(self, inputs):
    # Multiplicação das entradas pelos pesos + o bias
    sum = np.dot(inputs, self.weights[1:]) + self.weights[0]
    # Comparação para ativação
    if sum > 0:
      return 1
    else:
      return 0
```

5 RESULTADOS

A classe proposta nesse trabalho foi avaliada com dois problemas de classificação, onde um deles deveriamos classificar as funções lógicas AND e OU, avaliando também a função XOR e o segundo problema era classificar as classes “setosa” e “versicolor” da base de dados Iris.

Para o primeiro problema proposto, temos as seguintes tabelas verdade de mapeamente de entradas e saidas que serão treinadas no Perceptron pela função *fit*.

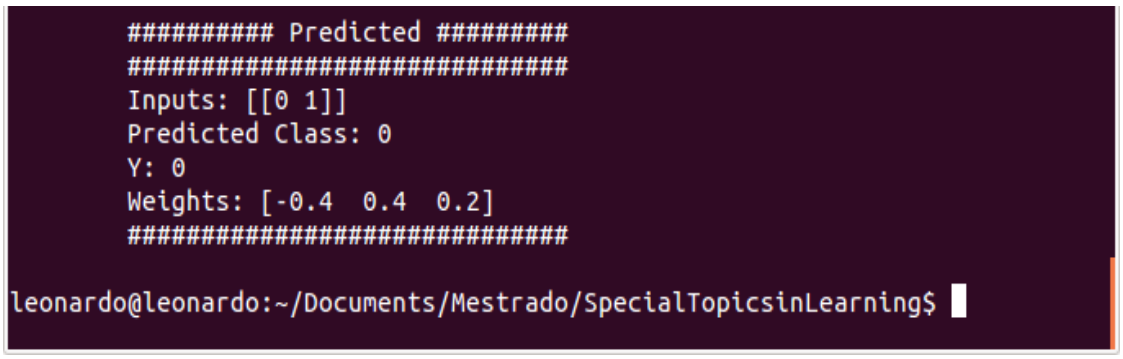
X1	X2	Y
0	0	0
0	1	0
1	0	0
1	1	1

Tabela 1: Tabela verdade da função AND.

X1	X2	Y
0	0	0
0	1	1
1	0	1
1	1	0

Tabela 2: Tabela verdade da função OR.

Para a função AND, a imagem a seguir mostra a resposta do algoritmo, que com 5 iterações aprendeu com o conjunto de dados a função que separa linearmente os dados.



Podemos notar que os pesos aprendidos foram -0.4, 0.4 e 0.2, que representam os dois pesos das variáveis de entrada X1 e X2 e o peso do *bias* que desloca a função aprendida pelo eixo da imagem.

A segunda imagem ilustra o mesmo teste para a função lógica OR, onde o valor de entrada foi o par 0 e 1, nossa resposta esperada é 1 e o classificador deu 1 de resposta.

```
########## Predicted #########
##############################
Inputs: [[0 1]]
Predicted Class: 1
Y: 1
Weights: [0.  0.2 0.2]
##############################

leonardo@leonardo:~/Documents/Mestrado/SpecialTopicsinLearning$
```

Podemos notar que agora os pesos mudaram, agora temos os valores de 0, 0.2 e 0.2 para as variáveis X1, X2 e o *bias*. Para todas as entradas o *Perceptron* resolveu o problema de maneira rápida e com cem por cento de atertividade. Para o último caso, a função XOR, a imagem a seguir mostra em gráfico que as classes não são separáveis linearmente.

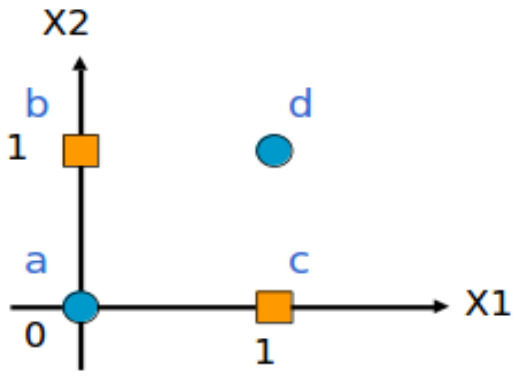
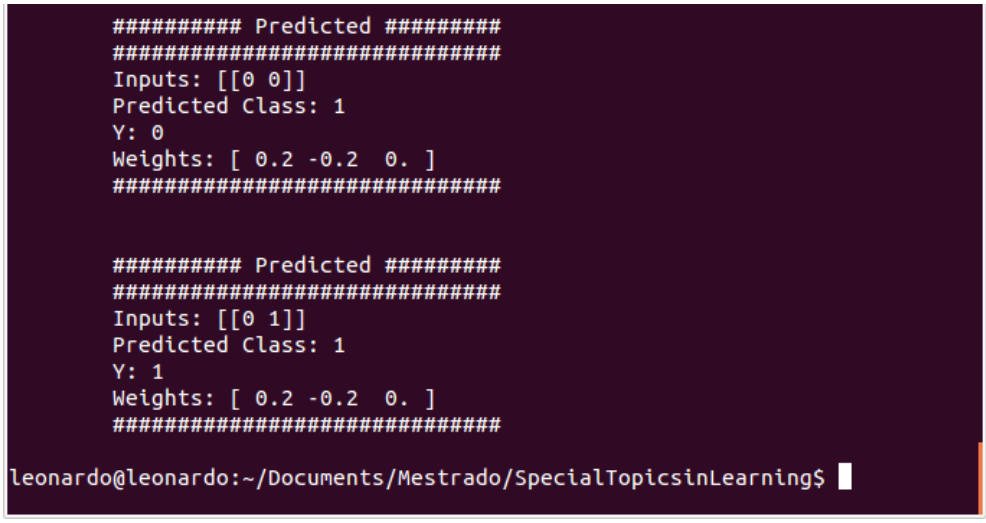
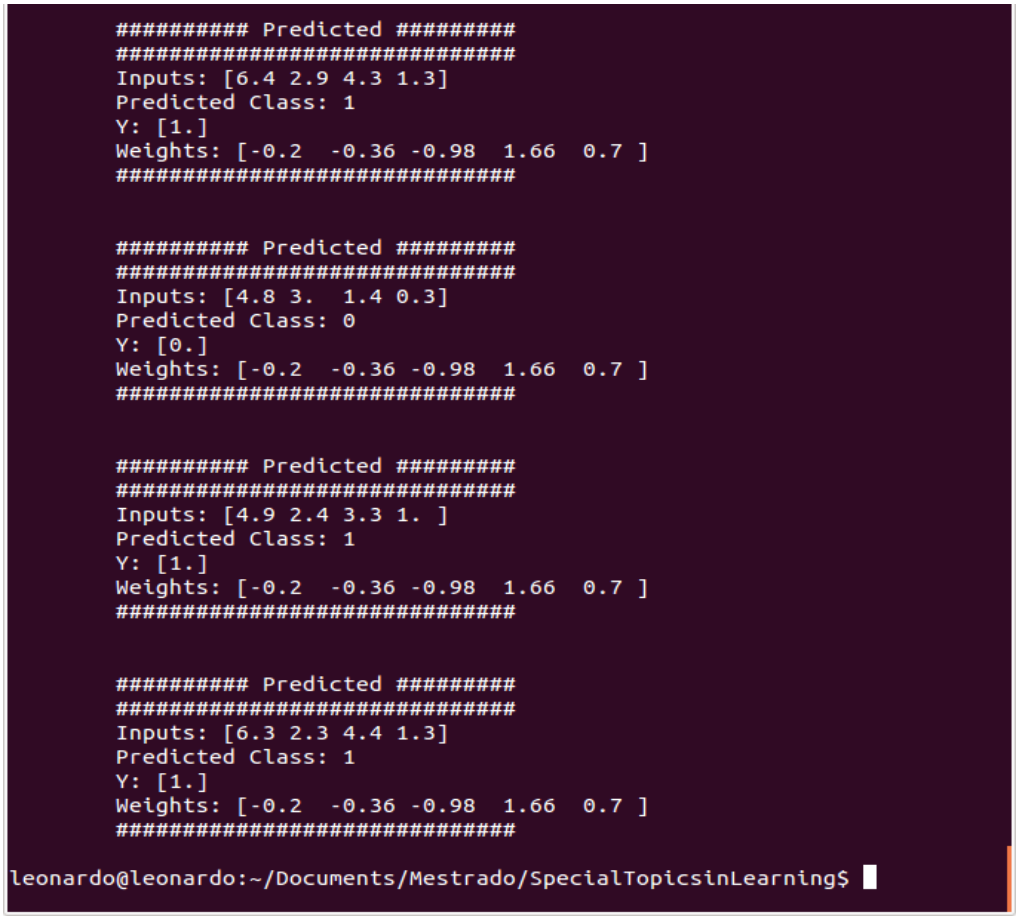


Gráfico 1: Função XOR.

Por esse motivo, o perceptron se mostrou ineficiente como mostra nas imagens a seguir, onde ele não classificou o conjunto de entradas [0, 0] e classificou o conjunto [0, 1].



O segundo problema proposto foi classificar, da base de dados Iris, as classes “Setosa” e “Versicolor”, que são linearmente separáveis do universo das 3 classes da base. Com uma entrada de 4 características das flores, a fase de aprendizado durou duas iterações para aprender o padrão linear de separação e classificou corretamente todos os dados de teste. A base foi separada em 80% treino e 20% teste, onde a imagem a seguir mostra alguns casos que foram mostrados pelo temrinal a cada iteração de predição.



Podemos notar que os pesos para as entradas agora mudaram, sendo -0.2, -0.36, -0.98, 1.66 para as variáveis X1, X2, X3, X4 e 0.7 para o *bias.*

5 CONCLUSÃO

O Algoritmo *Perceptron* mostrou grande eficiência na classificação dos das bases selecionadas, com exceção da função XOR por apresentar uma separação não linear, bem com a terceira classe da base de dados Iris. Os conjuntos de test foram todos passados pelo algoritmo com acertividade de 100% (os linearmente separáveis) e portanto com poucas iterações a função de cada conjunto de dados pode ser aprendida sem grande dificuldade.

Os próximos passos seriam criar redes a partir dos perceptrons desenvolvidos aprender funções mais complexas, como por exemplo a função XOR ou todas as classes da base de dados Iris.

REFERÊNCIAS

ADELI, H.; YEH, C. Perceptron learning in engineering design. **Computer-Aided Civil and Infrastructure Engineering**, v. 4, n. 4, p. 247-256, 1989.

PERCEPTRON. In: WIKIPÉDIA, a enciclopédia livre. Flórida: Wikimedia Foundation, 2018. Disponível em: <https://pt.wikipedia.org/w/index.php?title=Perceptron&oldid=52756657>. Acesso em: 26 jul. 2018.

NumPy Reference. Disponível em: <https://docs.scipy.org/doc/numpy-1.15.1/reference/index.html>. Acesso em: 10 set. 2018.