Regressão Polinomial e Simbólica

Fabrício Olivetti de França

Universidade Federal do ABC

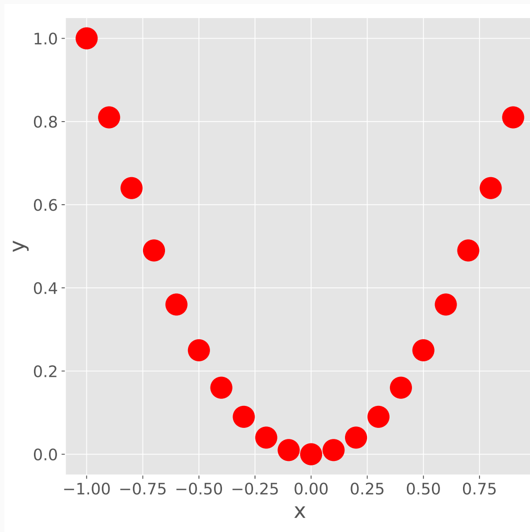
Tópicos

1. Variáveis Não-Lineares
2. Regressão Simbólica

UFABC

Variáveis Não-Lineares

Considere os seguintes pontos:



UFABC

Variáveis Não-Lineares

Uma alternativa para esses casos é criação de novos atributos como funções não lineares.

Esses novos atributos podem ser:

- **Transformação:** aplicação de uma função não-linear em uma combinação linear das variáveis.
- **Interação:** interação polinomial entre duas ou mais variáveis.

UFABC

Transformação

A aplicação de uma função não-linear, se escolhida adequadamente, pode gerar novos atributos que possuem relação linear com a variável alvo.

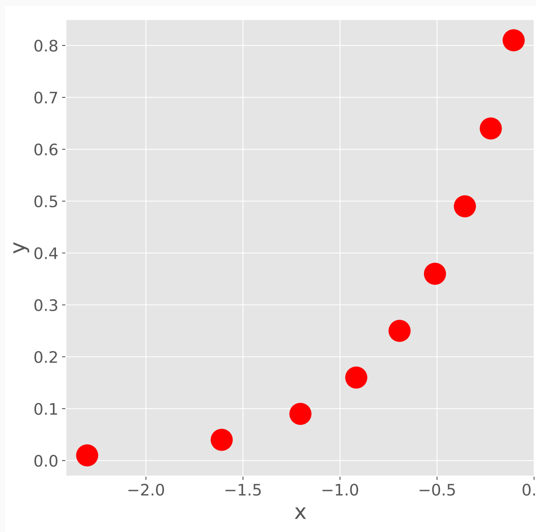
Funções comumentes utilizadas para essa tarefa:

- **Logística:** $\frac{1}{1+e^{-x}}$ cria uma variável sigmoidal, com valores entre 0 e 1.
- **Tangente Hiperbólica:** $\tanh(x)$ idem ao anterior, mas variando entre -1 e 1.
- **Logaritmo:** $\log(x)$ lineariza variáveis que seguem uma lei de potência.
- **Rectified Linear Units:** $\max(0, x)$, elimina os valores negativos, relacionado com Redes Neurais.
- **Softmax:** $\frac{e^{x_i}}{\sum_j e^{x_j}}$, faz com que a soma dos valores de x seja igual a 1.

UFABC

Transformação

Aplicando uma transformação logarítimica em nosso exemplo, temos:



UFABC

Interação

A interação polinomial gera interações de grau p entre as variáveis.

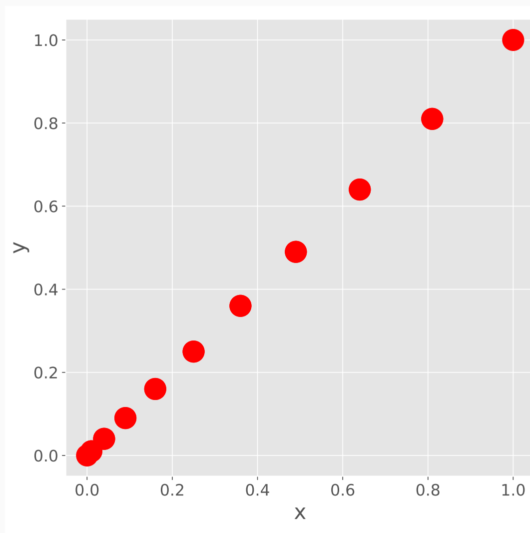
Por exemplo, em um problema com 2 variáveis e $p = 2$ teríamos as variáveis $x_1 \cdot x_1, x_1 \cdot x_2, x_2 \cdot x_2$.

Já para $p = 3$ teríamos $x_1^3, x_1^2 \cdot x_2, x_1 \cdot x_2^2, x_2^3$.

UFABC

Interação

Introduzindo a variável x_1^2 em nosso exemplo, temos:



UFABC

Gerando Atributos Polinomiais

Dado um vetor $x = [x_1, x_2, x_3, \ldots, x_d]$, queremos obter:

$$[[1], [x_1, x_2, \ldots, x_d], [x_1^2, x_1 \cdot x_2, \ldots], [x_1^3, x_1^2 \cdot x2, \ldots], \ldots].$$

Não queremos que repita $x_1 \cdot x_2 e x_2 \cdot x_1$.

UFABC

Gerando Atributos Polinomiais

Vamos iniciar criando uma lista inicial com tamanho $k + 1$, com k sendo o grau do polinômio que desejamos:

```
poly = [[1]] + [[] for _ in range(k)]
```

UFABC

Gerando Atributos Polinomiais

Agora, para cada variável x_i e para cada lista $l \leftarrow poly$, multiplicamos x_i a cada elemento de l e concatenamos a lista gerada com a lista seguinte.

Para x_0 teremos:

```
poly = [[1], [x[0]], [x[0]**2], [x[0]**3], ..]
```

UFABC

Gerando Atributos Polinomiais

Agora, para cada variável x_i e para cada lista $l \leftarrow ys$, multiplicamos x_i a cada elemento de l e concatenamos a lista gerada com a lista seguinte.

Para x_1 teremos:

```
ys = [[1], [x[0], x[1]], [x[0]**2, x[0]*x[1], x[1]**2],
      [x[0]**3, x[0]**2 * x[1], x[0] * x[1]**2, x[1]**3], ..
    ]
```

Queremos uma função que aplique *map (*x_i)* em cada elemento de uma lista e concatene com a seguinte.

UFABC

Gerando Atributos Polinomiais

Ou seja, queremos:

```
ys = [[1], [], [], ..]
     [[1], (map (*x1) [1]) ++ [],
       (map (*x1) (map (*x1) [1]) ++ []) ++ [], ..]
```

ou:

```
ys = [[1], [], [], ..]
     [[1], f([1], []), f (f ([1], [])) [], ..]
```

UFABC

Gerando Atributos Polinomiais

Ou seja, queremos:

```
def polyFeatures(k ,x):
    poly = [[1]] + [[] for _ in range(k)]

    for xi in x:
        poly = f(xi, poly)
    return poly
```

UFABC

Gerando Atributos Polinomiais

Com:

```
def f(x,poly):
    newpoly = [poly[0]]
    g = lambda pi : x*pi
    for i in range(len(poly)-1):
        newp2 = list(map(g,newpoly[i])) + poly[i+1]
        newpoly.append(newp2)
    return newpoly
```

UFABC

Gerando Atributos Polinomiais - Scikit-Learn

```
from sklearn.preprocessing import PolynomialFeatures

polyFeat = PolynomialFeatures(degree=2)

polyFeat = polyFeat.fit(X_data)
X_poly = polyFeat.transform(X_data)
```

UFABC

Regressão Simbólica

Regressão simples e poderosa

Uma característica da Regressão Linear, que pode ser encarada tanto como ponto forte ou fraco, é sua simplicidade.

Por ser simples, é fácil de interpretar.

Por ser simples, ela não consegue modelar muitas possíveis relações.

UFABC

Regressão simples e poderosa

A Regressão Polinomial tenta resolver esse problema adicionando um pouco de complexidade ao modelo sem torná-lo tão complexo.

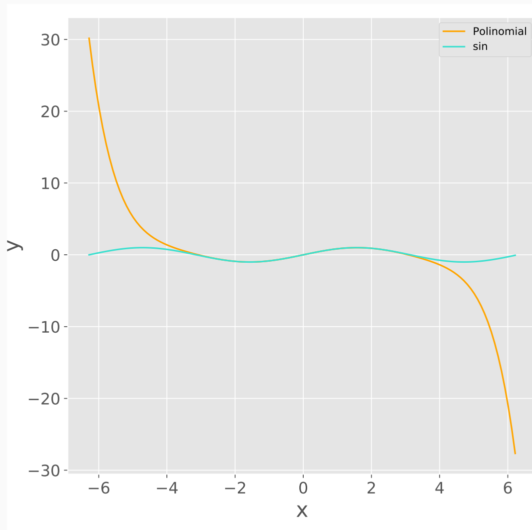
Podemos aproximar $\sin(x)$ como:

$$x - \frac{x^3}{3!} + \frac{x^5}{5!} - \frac{x^7}{7!}$$

UFABC

Regressão simples e poderosa

Mas até onde podemos aproximar?



UFABC

Regressão simples e poderosa

Além disso, não seria mais simples se o algoritmo de regressão retornasse $\sin(x)$?

UFABC

Regressão Simbólica

A *Regressão Simbólica* busca encontrar a forma da função e seus parâmetros que melhor se adapta a base de dados estudada.

UFABC

Diferente da Regressão Linear e Polinomial, não temos uma forma fechada para a função de aproximação que gera $\hat{y}$.

Programação Genética

Para resolver esse problema, costuma-se empregar a *Programação Genética*.

A Programação Genética é um algoritmo de otimização evolutiva que busca pela melhor árvore de expressão que resolve o problema de regressão.

UFABC

Programação Genética

Para resolver esse problema, costuma-se empregar a *Programação Genética*.

A Programação Genética é um algoritmo de otimização evolutiva que busca pela melhor árvore de expressão que resolve o problema de regressão.

UFABC

Árvore de Expressão

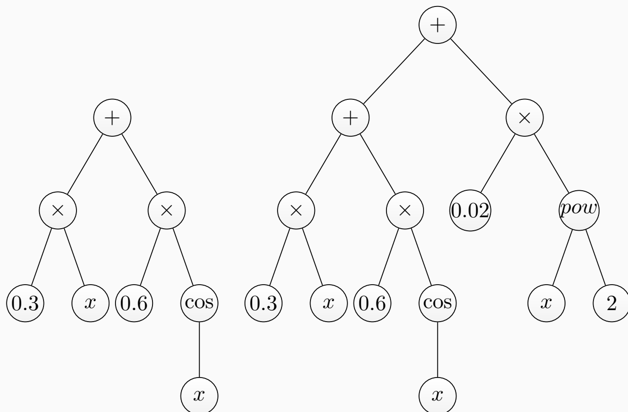
Uma árvore de expressão é uma árvore *n*-ária que representa uma expressão matemática.

A aridade dessa árvore é igual a maior aridade dentre as funções consideradas, que costuma ser 2.

UFABC

Árvore de Expressão

As árvores abaixo representam as expressões $0.3x + 0.6\cos(x)$ e $0.3x + 0.6\cos(x) + 0.02x^2$, respectivamente:



Programação Genética

O algoritmo de Programação Genética canônico é descrito de forma abstrata como:

```
def GP():
    P = geraPopulacao()
    for gen in range(max_gen):
        Filhos = cruzamento(P)
        Xmen   = mutacao(Filhos)
        P      = seleciona(P, Xmen)
    return melhor(P)
```

UFABC

Programação Genética

A função *geraPopulacao* cria uma população aleatória de árvores de expressão.

Para não criarmos árvores longas no início, adotamos a técnica *ramped half-and-half*:

Metade das árvores são criadas com uma altura fixa h.

Metade das árvores são criadas até uma altura h.

UFABC

A criação de árvores segue o seguinte procedimento:

Inicia uma árvore vazia e escolhe um nó aleatório para ser criado

O nó aleatório pode ser uma função, um valor numérico ou um atributo da base de dados

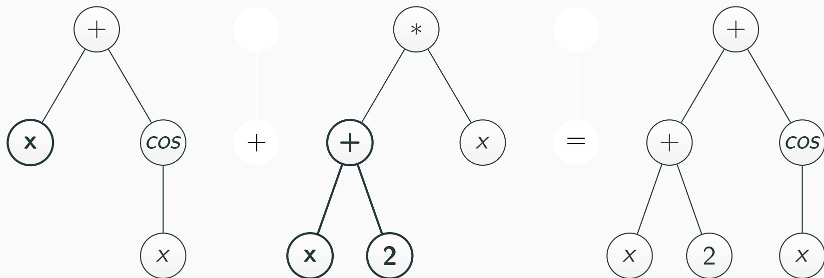
Para os casos de árvore com altura fixa, se a atulra é menor que h, escolhe-se apenas funções

Se a altura é igual a h escolhe-se apenas nós terminais

Se o nó atual for uma função, o procedimento é executado recursivamente para gerar n filhos aleatórios, com n igual a aridade da função

Cruzamento

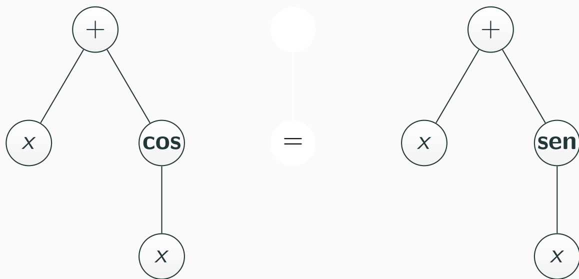
No cruzamento duas ou mais árvores de expressão são combinadas, para isso escolhe-se uma subárvore de uma solução que substitui a subárvore de outra solução:



UFABC

Mutação

Na mutação, cada solução pode ser alterada em um nó com uma certa probabilidade:



UFABC

Seleção

Para selecionar quais soluções serão consideradas para a próxima geração, são realizados diversos torneios entre a população atual e a nova população até que n indivíduos são selecionados, com n sendo o número de indivíduos da população inicial.

UFABC

Os problemas do GP

A flexibilidade de gerar a forma da função tem um preço:

- O espaço de busca é enorme e irregular, a tarefa se torna encontrar uma agulha no palheiro.
- Existem muitas soluções possíveis para uma determinada base de dados, a maioria delas é de funções extremamente complexas.

Regressão Interação-Transformação

Uma solução para os problemas do GP, cria um meio termo entre a flexibilidade de criar novas funções e um limite da complexidade que pode ser gerada.

A *Regressão Interação-Transformação* define uma forma fechada para a regressão como uma regressão linear de interações polinomiais com funções de transformação.

UFABC

Regressão Interação-Transformação

Uma expressão Interação-Transformação é definida como:

$$\hat{f}(x) = \sum_i w_i \cdot g_i(x),$$

com w_i sendo o i-ésimo coeficiente de uma Regressão Linear sobre transformações dos atributos do exemplo x.

UFABC

Regressão Interação-Transformação

A transformação $g_i(x)$ é a composição $g(.) = t(.) \circ p(.)$, sendo $t : \mathbb{R} \to \mathbb{R}$ qualquer função unidimensional de transformação (ex.: $\sin, \cos, \log$ e $p : \mathbb{R}^d \to \mathbb{R}$ uma função de interação d-dimensional da forma:

$$p(x) = \prod_{i=1}^{d} x_i^{k_i},$$

em que $k_i \in \mathbb{Z}$ é o expoente da i-ésima variável.

UFABC

Regressão Interação-Transformação

Supondo amostras de dimensão $d = 3$, temos que a Regressão Linear pode ser representada como:

$$
\begin{aligned}
y &= w_1 \cdot g_1(x) + w_2 \cdot g_2(x) + w_3 \cdot g_3(x) \\
g_1(x) &= id(x_1) \\
g_2(x) &= id(x_2) \\
g_3(x) &= id(x_3)
\end{aligned}
$$

,

sendo $id(x) = x$ a função identidade.

UFABC

Regressão Interação-Transformação

Da mesma forma, a função $f(x) = 3.5 \sin(x_1^2 \cdot x_2) + 5 \log(x_2^3 / x_1)$ é representada por:

$$
\begin{aligned}
\hat{f}(x) &= 3.5 \cdot g_1(x) + 5 \cdot g_2(x) \\
t_1(z) &= \sin(z) \\
p_1(\mathbf{x}) &= x_1^2 \cdot x_2 \\
t_2(z) &= \log(z) \\
p_2(\mathbf{x}) &= x_1^{-1} \cdot x_2^3.
\end{aligned}
$$

UFABC

Regressão Interação-Transformação

Uma representação computacional de uma expressão IT pode ser pensada como um conjunto de termos T e cada termo sendo representado por uma tupla (p, f), com p sendo um vetor com os expoentes do polinômio e f a função de transformação a ser aplicada.

UFABC

Regressão Interação-Transformação

O nosso exemplo de Regressão Linear ficaria:

$$
\begin{aligned}
T &= \{t_1, t_2, t_3\} \\
t_1 &= ([1, 0, 0], id) \\
t_2 &= ([0, 1, 0], id) \\
t_3 &= ([0, 0, 1], id).
\end{aligned}
$$

UFABC

Regressão Interação-Transformação

E a função $f(x) = 3.5 \sin(x_1^2 \cdot x_2) + 5 \log(x_2^3 / x_1)$:

$$
\begin{aligned}
T &= \{t_1, t_2\} \\
t_1 &= ([2, 1], \sin) \\
t_2 &= ([-1, 3], \log).
\end{aligned}
$$

UFABC

Algoritmo SymTree

Um algoritmo simples para encontrar a expressão IT de uma base de dados.

Basicamente inicia com uma Regressão Linear e aumenta gradualmente a complexidade do modelo a cada passo.

UFABC

Algoritmo SymTree

De forma abstrata os passos são:

- Inicializa com a estrutura representando uma Regressão Linear
- Repita:
- Gera todas as possíveis interações entre os termos da expressão atual
- Gera todas as possíveis transformações dos termos da expressão atual
- Cria novas expressões que maximizem o objetivo

UFABC

Algoritmo SymTree

```
expr = geraRegLinear()
for it in range(max_it):
    termos  = geraInteracao(expr)
    termos += geraTransformacao(expr)
    expr    = geraExpressao(expr, termos)
```

UFABC

Interações

Para gerar uma interação, basta somar os vetores de expoentes de dois termos. Supondo os termos:

$$t_1 = [2, 1]$$
$$t_2 = [-1, 3]$$

UFABC

Interações

Ao somar t_1 e t_2 temos:

$$t_1 = [2, 1]$$
$$t_2 = [-1, 3]$$
$$t_3 = [1, 4]$$

UFABC

Interações

E ao subtrair t_1 e t_2 temos:

$$t_1 = [2, 1]$$
$$t_2 = [-1, 3]$$
$$t_3 = [1, 4]$$
$$t_4 = [3, -2]$$

UFABC

Interações

Que representam:

$$
\begin{aligned}
t_1 &= [2, 1] & x_1^2 \cdot x_2 \\
t_2 &= [-1, 3] & x_1^{-1} \cdot x_2^3 \\
t_3 &= [1, 4] & x_1 \cdot x_2^4 \\
t_4 &= [3, -2] & x_1^3 \cdot x_2^{-2}
\end{aligned}
$$

UFABC

Algoritmo SymTree

```
def geraInteracao(expr):
  termos  = [somaExpoentes(t1, t2)
                  for t1, t2 in combinations(expr)]
  termos += [subExpoentes(t1, t2)
                  for t1, t2 in combinations(expr)]
  return termos
```

Transformações

Para gerar as transformações de um termo, basta aplicar todas as funções do conjunto de funções ao termo:

$$t_1 = ([2, 1], id)$$

UFABC

Transformações

Para gerar as transformações de um termo, basta aplicar todas as funções do conjunto de funções ao termo:

$$
\begin{aligned}
t_1 &= ([2, 1], id) \\
t_2 &= ([2, 1], \sin) \\
t_3 &= ([2, 1], \log) \\
t_4 &= ([2, 1], \sqrt{})
\end{aligned}
$$

UFABC

Algoritmo SymTree

```python
def geraTransformacao(expr):
  funcoes = [id, np.sqrt, np.sin, np.log]
  termos = [(expo, fi)
              for (expo, f) in expr
              for fi in funcoes
           ]
```

UFABC

Gerando Expressões

Dada a expressão atual como um conjunto T de termos e um conjunto T' com os novos termos gerados, a nova expressão é a combinação dos termos $T'' = T' \cup T$ que retorna a melhor expressão:

```
def geraExpressao(expr, termos):
  cjtTermos   = uniao(expr, termos)
  melhorExp, melhorScore = (expr, score(expr))

  for expr_i in combinations(cjtTermos):
    if score(expr_i) > melhorScore:
      melhorExp, melhorScore = (expr_i, score(expr_i))
  return melhorExp
```

UFABC

Algoritmo SymTree

Disponível online:

Algoritmos baseado em Interação-Transformação

UFABC

Próxima Aula

Na próxima aula aprenderemos sobre:

- Algoritmos de Classificação.
- Perceptron.
- Regressão Logística.

UFABC

Atividade 04

Complete o Laboratório:

Regularization_and_Gradient_Descent_Exercises.ipynb

UFABC