Mestrado Acadêmico IBILCE/São José do Rio Preto
Universidade Estadual Paulista "Júlio de Mesquita Filho"

APOSTILA DE TÓPICOS ESPECIAIS: APRENDIZADO DE MÁQUINA

Prof. Dr. João Paulo Papa

Contribuinte: Gustavo Henrique de Rosa

Departamento de Computação, Faculdade de Ciências,
Universidade Estadual Paulista Júlio de Mesquita Filho
Av. Eng. Luiz Edmundo Carrijo Coube, 14-01 - Vargem
Limpa, CEP 17033-360 - Bauru - SP.

BAURU
2016

Sumário

1 Introdução

Machine Learning (mais teórico) x *Pattern Recognition* x *Artificial Intelligence* (área maior, lógica fuzzy)

Podemos considerar o *Machine Learning* e o *Pattern Recognition* como uma inteligência artificial aplicada, isto é, redes neurais, SVM, *Deep Learning*.

Imagem $\longrightarrow$ Extração de características $\longrightarrow$ Espaço de características

O conjunto de dados é composto por um conjunto de treinamento, utilizado para aprender e determinar a função separadora do espaço (parâmetros). E também é composto por um conjunto de teste, utilizado em novas amostras e para definir o grupo das mesmas.

Machine Learning $\longrightarrow$ supervisionado (dados rotulados) ou não supervisionado

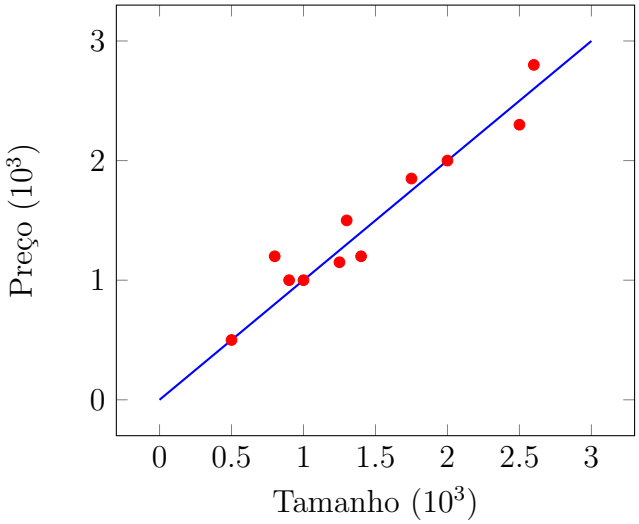
1.1 Exemplos de Técnicas

- SVM - support vector machine: parte do princípio que os dados podem ser separados. O problema se dá quando aumentamos o número de parâmetros, trazendo um maior custo computacional;
- Reconhecimento de padrões: o problema é dividido em classificação e regressão (encontrar o fitting, ou seja, a melhor função que modela o problema).

2 Regressão Linear

2.1 Uma Variável (Univariado)

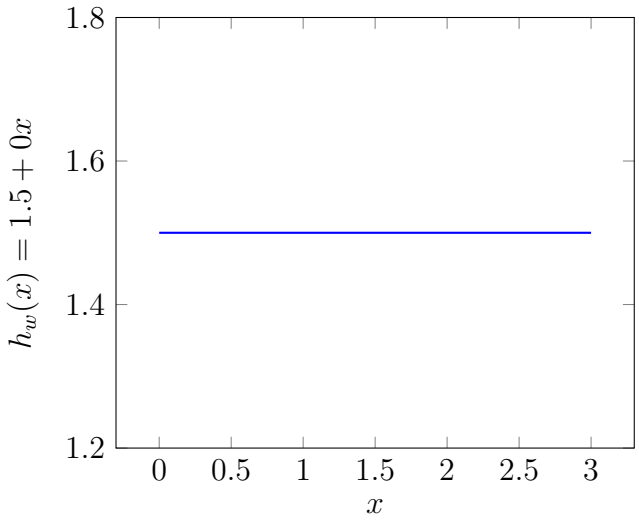
Suponha que tenhamos o problema de predizer preços de casas, como demonstradas abaixo. A regressão do problema tem por objetivo estimar (predizer) valores reais de saída, dado uma entrada. Considerando o problema mencionado, gostaríamos de estimar o preço de uma casa dado o seu tamanho, isto é, gostaríamos de encontrar a **linha reta** a qual melhor se adapta aos pontos da base de dados (dados de treinamento).

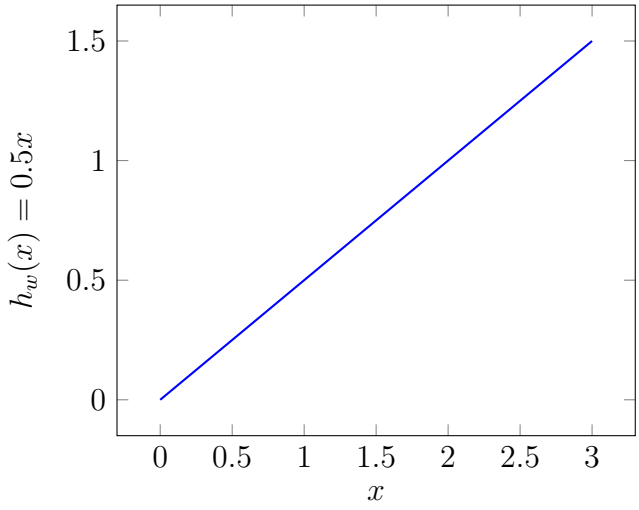


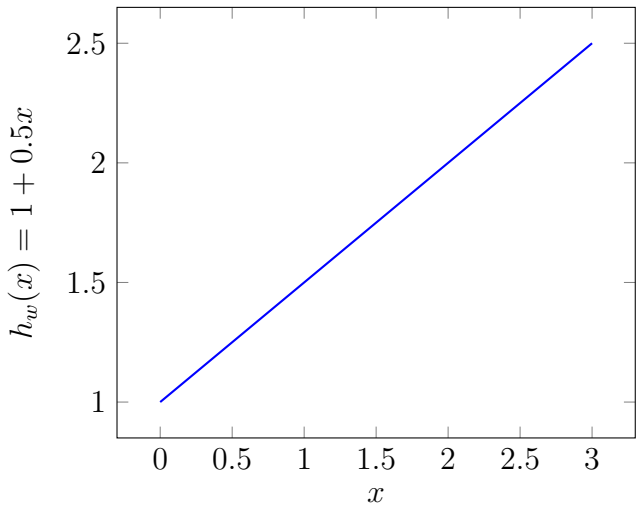
Exemplo: Seja m o número de exemplos de treinamento, $x \in \mathbb{R}$ a variável de entrada (características) e $y \in \mathbb{R}$ a variável de saída (objetivo). Então, um treinamento simples de treinamento é denotado por (x, y), sendo o conjunto de treinamento X composto por todas as amostradas de treinamento, isto é, $X = \{(x, y), (x_2, y_2), \ldots, (x_m, y_m)\}$. Usualmente, nós possuímos o seguinte fluxo de dados:

Dados de treinamento $X \longrightarrow$ Algoritmo de treinamento $\longrightarrow h$ (tamanho da casa) $\longrightarrow$ preço estimado

A função linear pode ser definida como $h_w(x) = w_0 + w_1 x$, onde w_0 e w_1 são os parâmetros do modelo. Suponha que tenhamos os seguintes exemplos:







Portanto, possuímos diferentes comportamentos envolvendo $h_w(x)$, o qual é chamado de **função hipótese**.

A ideia principal da regressão linear é escolher w_0 e w_1 tal que $h_w(x)$ esteja o mais próximo de y, considerendo os exemplos de treinamento (x_i, y_i), $i = 1, 2, \ldots, m$. Para cumprir tal propósito, temos que solucionar o seguinte problema de otimização:

$$minimizar_{w_0,w_1} \quad \frac{1}{2m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 \tag{1}$$

onde $h_w(x_i)$ é o **preço estimado** e y_i é o **preço real**.

A Equação 1 é usualmente chamada de **função de custo**, a qual também é conhecida como Erro Médio Quadrático (*Mean Squared Error* - MSE). Podemos simplificar a notação e reescrever a Equação 1 como seguinte:

$$minimizar_{w_0,w_1} \quad J(w_0, w_1) \tag{2}$$

onde $J(w_0, w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$. Também podemos simplificar ainda mais considerando que $w_0 = 0$. Portanto, possuíremos as seguintes equações para representar as funções de hipótese e custo, respectivamente:

$$h_w(x) = w_1 x \tag{3}$$

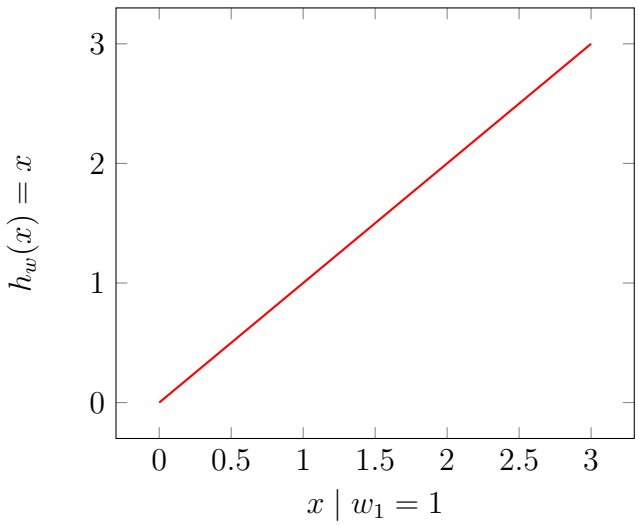
e

$$J(w_1) = \frac{1}{2m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 \tag{4}$$

O problema de minimização torna-se agora:

$$minimizar_{w_1} \quad J(w_1) \tag{5}$$

Tentemos entender ambas funções de hipótese e custo.

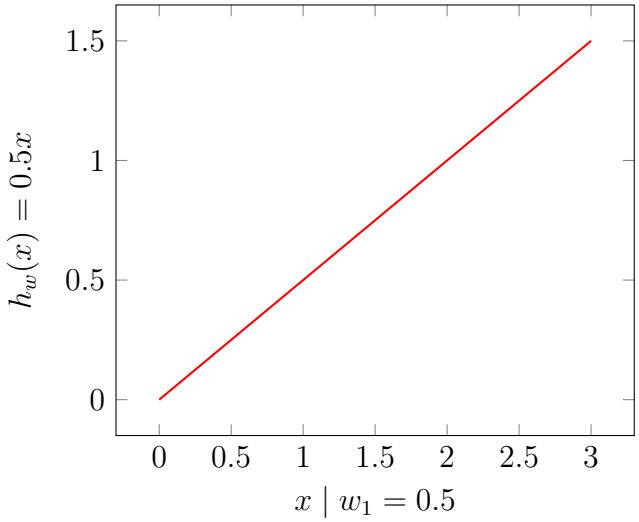


$$J(w_1) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x_i) - y_i)^2$$

$$= \frac{1}{2m}\sum_{i=1}^{m}(w_1 x_i) - y_i)^2$$

$$= \frac{1}{2\times 3}[(1-1)^2 + (2-2)^2 + (3-3)^2]$$

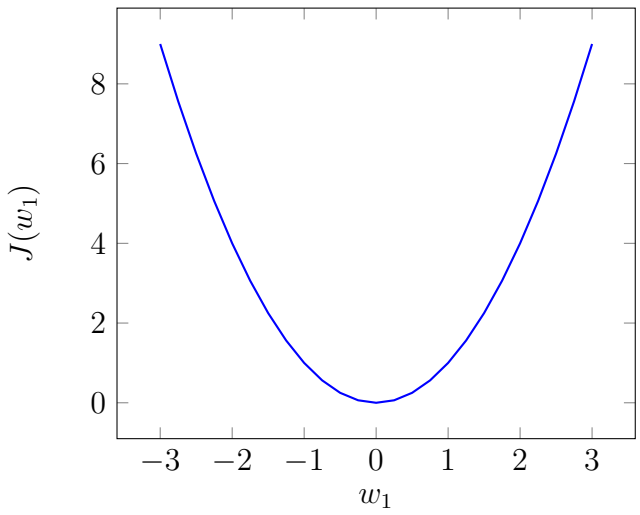
$$= \frac{1}{6} \times 0 = 0$$



$$J(w_1) = \frac{1}{2\times 3}[(0.5-1)^2 + (1-2)^2 + (1.5-3)^2]$$

$$= \frac{1}{6} \times (0.25 + 1 + 2.25) \approx 0.58$$

Se continuarmos efetuando os cálculos mencionados acima para diversos valores de w_1, devemos ter algo similar ao seguinte gráfico:



Erro Médio Absoluto:

$$J(w_1) = \frac{1}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)$$

Portanto, $w_1 = 1$ é o valor o qual minimiza $J(w_1)$ para o exemplo acima. Porque é melhor empregar o MSE ao invés do Erro Médio Absoluto (*Mean Absolute Error* - MAE)? Dado que os erros estão sendo elevados ao quadrado logo após serem ponderados, o MSE resulta em um peso maior relativo à erros maiores.

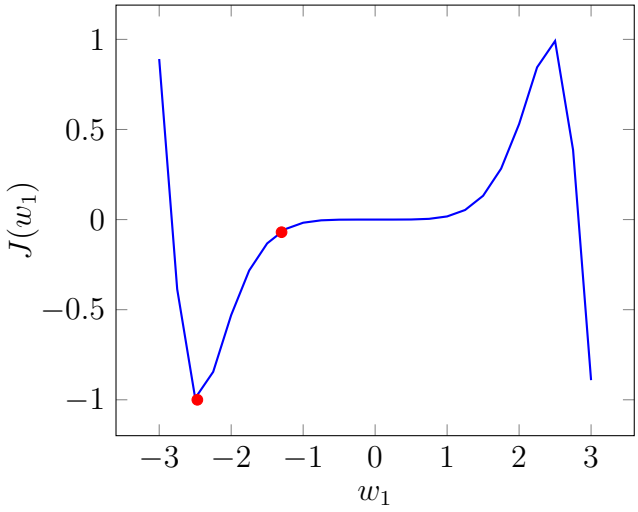
Então, a questão agora é: como podemos encontrar valores plausíveis para ambos w_0 e w_1? Podemos modelar tal problema como uma tarefa de otimização utilizando o algoritmo do **Gradiente Descendente** (*Gradient Descent* - GD) por exemplo. Em resumo, temos o seguinte:

Temos alguma função $J(w_0, w_1)$;
Queremos minimizar $J(w_0, w_1)$.

Algoritmo do Gradiente Descendente:

- comece com algum w_0, w_1;
- continue mudando w_0, w_1 de modo que reduza $J(w_0, w_1)$ até que atinja o mínimo (isso pode não acontecer!).

Observemos o mecanismo de busca do Gradiente Descendente:



A ideia acima pode ser matematicamente formulada como seguinte:

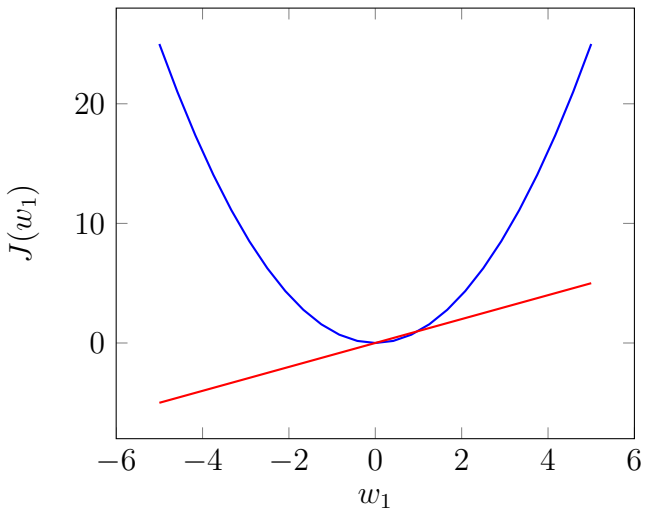
$$w_j = w_j - \alpha \frac{\partial J(w_0, w_1)}{\partial w_j}, j \in \{0, 1\} \tag{6}$$

onde α é a taxa de aprendizado e $\frac{\partial J(w_0,w_1)}{\partial w_j}$ é o termo derivativo.

A equação acima é chamada de **regra de atualização**. Segue abaixo a implementação correta desse procedimento:

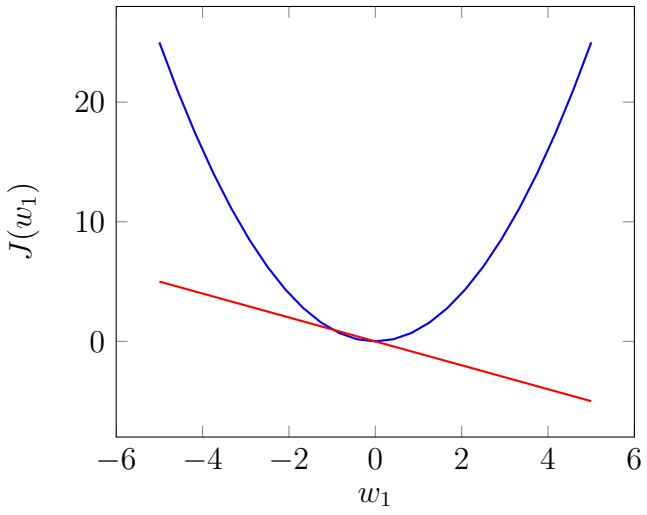
$$\begin{aligned} tmpw_0 &= w_0 - \alpha \frac{\partial J(w_0,w_1)}{\partial w_0} \\ tmpw_1 &= w_1 - \alpha \frac{\partial J(w_0,w_1)}{\partial w_1} \\ w_0 &= tmpw_0 \\ w_1 &= tmpw_1 \end{aligned}$$

A próxima questão que pode surgir é como computar o termo derivativo? Suponha que tenhamos apenas um parâmetro, por exemplo, w_1. Então, queremos minimizar $J(w_1)$ para algum $w_1 \in (R)$. Conhecemos o formato de $J(w_1)$:



onde a linha vermelha é a tangente e a inclinação $= w_1^{t+1} = w_1^t - \alpha \frac{\partial J(w_1)}{\partial w_1} > 0$.

O que é uma derivada em um ponto dado? É a **inclinação** da reta tangente em respeito ao ponto em questão. A inclinação nos indica o ângulo de uma reta em relação à linha horizontal.

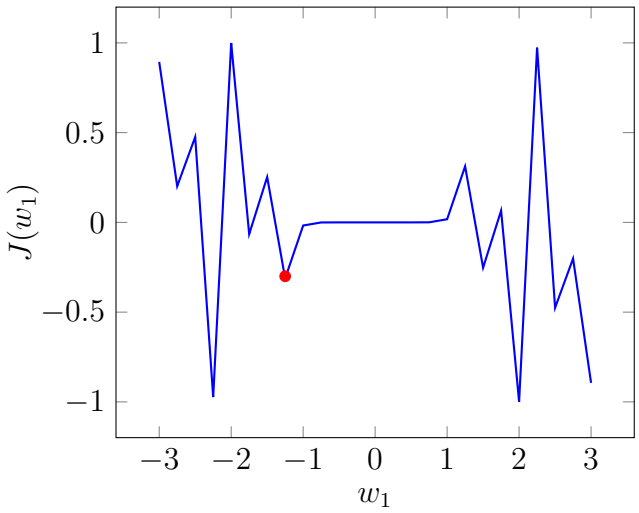


onde a inclinação é igual a $w_1^{t+1} = w_1^t - \alpha \frac{\partial J(w_1)}{\partial w_1} < 0$.

Agora, qual é a importância da taxa de aprendizado? Basicamente, pequenos valores de α tendem à uma lenta taxa de convergência, enquanto

valores altos de α nos leva à altas taxas de convergência.

O que acontece se inicializarmos o GD em um mínimo local?



Assim que nos aproximamos do mínimo global / local, o GD leva em consideração passos menores, até mesmo para um α fixo, dado que a inclinação também utiliza valores menores.

As derivadas parciais são computadas como seguinte:

$$\frac{\partial J(w_0,w_1)}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)$$

$$\frac{\partial J(w_0,w_1)}{\partial w_0} = \frac{1}{m}\sum_{i=1}^{m}[(h_w(x_i) - y_i)x_i]$$

Portanto, podemos sumarizar o algoritmo do GD como segue:

repita até convergência {

$w_0 \leftarrow w_0 - \alpha\frac{1}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)$

$w_1 \leftarrow w_1 - \alpha\frac{1}{m}\sum_{i=1}^{m}[(h_w(x_i) - y_i)x_i]$

}

2.2 Múltiplas Variáveis (Multivariado)

Suponha que tenhamos múltiplas características para representar o problema de estimar o preço de uma casa, por exemplo, o número de quartos, o número

de andares, dentre outros. Seja n o número de características, tal que nossa função de hipótese possa ser denotada como segue:

$$h_w(x) = w_0 + w_1x_1 + w_2x_2 + \cdots + w_nx_n = w_0 + \sum_{i=1}^{n} w_ix_i. \tag{7}$$

Neste caso, temos que $x_i \in \mathbb{R}$. Para a conveniência da notação, seja $x_0 = 1$, tal que $x = [x_0, x_1, x_2, \ldots, x_n] \in \mathbb{R}^{n+1}$, e $w = [w_0, w_1, w_2, \ldots, w_n] \in \mathbb{R}^{n+1}$. Portanto, a Equação 7 pode ser reescrita como segue:

$$h_w(x) = w_0x_0 + \sum_{i=1}^{n} w_ix_i = \sum_{i=0}^{n} w_ix_i = wx^T. \tag{8}$$

Então, como podemos utilizar o Gradiente Descendente para a regressão linear multivariável? Agora temos as seguintes proposições:

$$\text{função de hipóstese: } h_w(x) = wx^T$$
$$\text{função de custo: } J(w) = \frac{1}{2m}\sum_{i=1}^{m}(h_w(x_i) - y_n)^2$$
$$\text{parâmetro: w}$$

Gradiente Descendente:
repita até convergência {

$w_j \leftarrow w_j - \alpha\frac{\partial J(w)}{\partial w_j}$

não esqueça de atualizar cada w_j simultaneamente:

$\partial w_j = \frac{1}{m}\sum_{i=1}^{m}[(h_w(x_i) - y_i)x_i^j] \mid j = 1, 2, \ldots, n.$

}

Para trabalharmos com o Gradiente Descendente, podemos utilizar alguns artifícios. Um deles é o **dimensionamento das características**, i.e. temos que ter certeza de que as características estão em escalas similares.

Exemplo 1:
$x_1 = \text{tamanho}(0 - 200 \text{ feet})^2$ e $x_2 = \#$ de quartos.

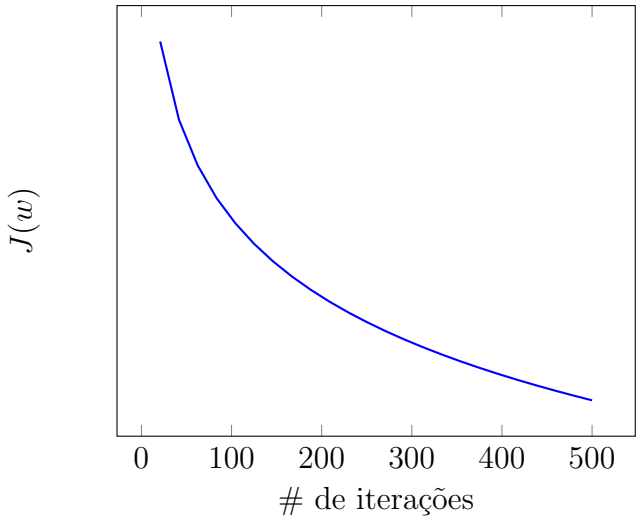
Exemplo 2:
$x_1' = \frac{tamanho}{2000}$ e $x_2' = \frac{\#dequartos}{5}$.

Portanto, podemos ter cada característica no intervalo aproximado de $-1 \leq x_i^j \leq 1$. Outra maneira é utilizar a **normalização média**:

$$x_i^j = \frac{x_i^j - \mu^j}{\beta_{max}^j - \beta_{min}^j}, \tag{9}$$

onde μ^j é o valor médio da característica j no conjunto de treinamento, e β_{max}^j e β_{min}^j denota os valores máximos e mínimo de cada características j no conjunto de treinamento, $i = 1, 2, \ldots, m$ e $j = 1, 2, \ldots, n$. Portanto, a ideia é fazer com que as características possuam aproximadamente média igual a zero.

Outro artifício para o Gradiente Descendente é a otimização da taxa de treinamento α. Entretanto, previamente, é necessário ter certeza de que o Gradiente Descendente está funcionando corretamente. Um comportamento esperado pode ser ilustrado como segue:

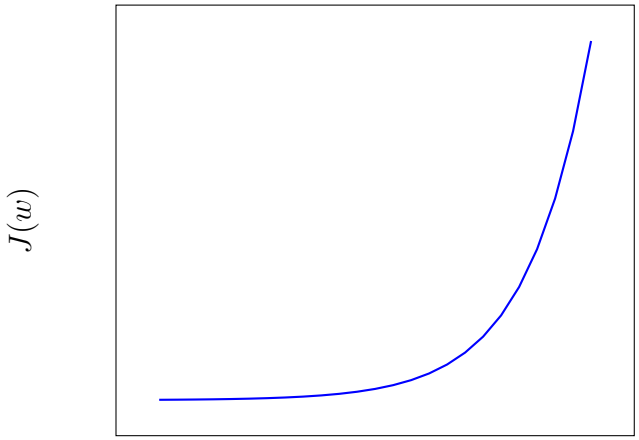


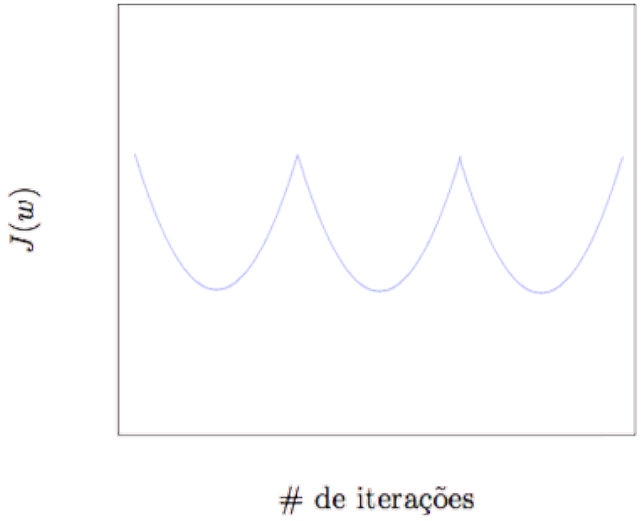
Problema: o número de iterações usualmente varia para cada problema. Portanto, é possível parar o algoritmo do GD utilizando a seguinte regra:

$$\mid J(w)^{t+1} - J(w)^t \mid < \epsilon, \tag{10}$$

onde $J(w)^t$ é o valor de cada função de custo na iteração i, e ϵ denota uma quantia muito pequena do **erro desejado** (usualmente $\epsilon = 10^{-3}$).

Entretanto, se valores de $J(w)$ estão crescendo, é possível utilizar valores menores para α. Situações possíveis:





2.3 Ferramentas Adicionais

No caso de possuirmos espaços de características complexos, pode ser necessário empregar funções de regressão diferentes. Podemos utilizar, por exemplo, polinômios de grau 2:

$$h_w(x) = w_0x_0 + w_1x_1 + w_2x^2. \tag{11}$$

Adicionalmente, é provável que os preços das casas comecem a estabilizar depois de um certo tamanho.

Considerando a regressão linear multivariável, ao invés de computar derivadas parciais para posterior computação do Gradiente Descendente, podemos resolver analiticamente a Equação 8. Basicamente, podemos reescrever a Equação 8 como segue:

$$y_i = w^T x_i, \forall i = 1, 2, \ldots, m. \tag{12}$$

Seja X a matriz contendo todas as amostras de treinamentos, por exemplo:

$$X = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \ldots & x_1^n \\ x_2^0 & x_2^1 & x_2^2 & \ldots & x_2^n \\ \vdots & \ddots & & & \vdots \\ x_m^0 & x_m^1 & x_m^2 & \ldots & x_m^n \end{bmatrix}, \in \mathbb{R}^{m \times (n+1)}$$

Adicionalmente, seja Y a matriz contendo todos os preços reais, por exemplo:

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \in \mathbb{R}^{m \times 1}$$

Podemos reescrever a Equação 12 considerando todos os pontos dos dados de treinamento como segue:

$$Y = w^T X. \tag{13}$$

Como podemos observar, a equação acima modela um sistema linear, que pode ser solucionado como segue:

$$w = (X^T X)^{-1} X^T Y, \tag{14}$$

onde A^{-1} é o inverso da matriz $A = X^T X$. Em suma, podemos sumarizar os seguintes pontos para cada técnica:

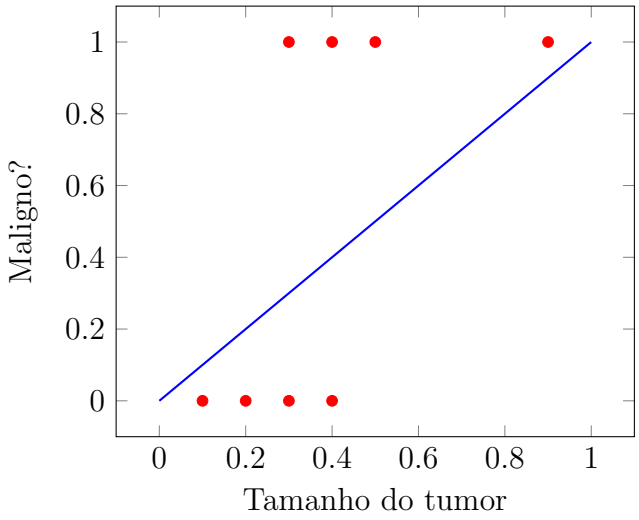
- **Gradiente Descendente:** necessita da escolha de *alpha*, necessita várias iterações, funciona muito bem até para $n \to \infty$;

- **Equação:** não necessita da escolha de α, não necessita iterar, necessita computar $(X^T X)^{-1}$, o qual pode ser lento quando $n \to \infty$.

3 Regressão Logística

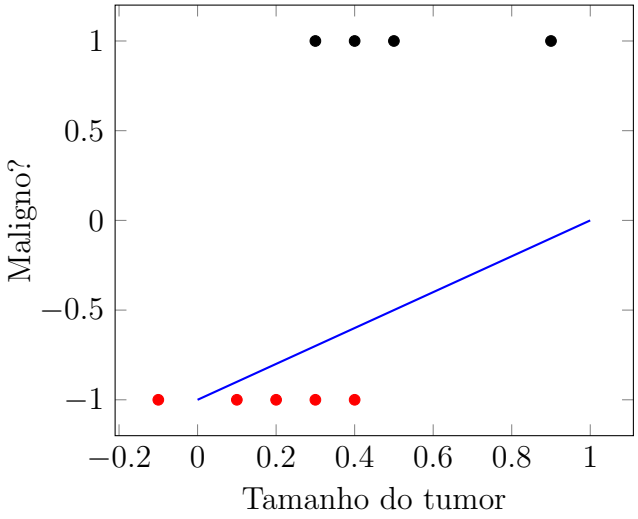
3.1 Problemas de duas classes

Suponha que tenhamos um problema de classificação binária (i.e. um problema de duas classes, $y \in \{0, 1\}$). Uma simples ilustração desse problema é dada a seguir:



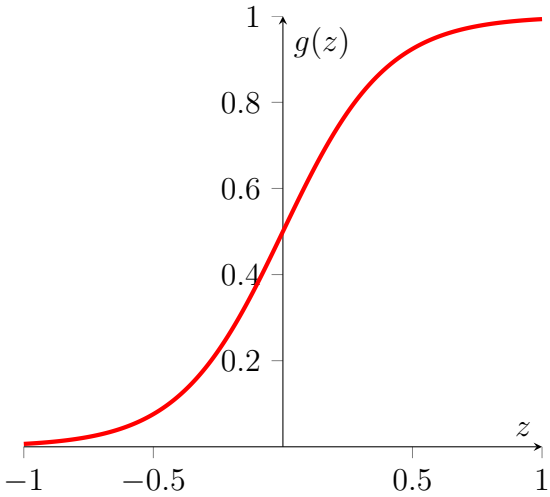
Podemos utilizar uma regressão linear aqui, com a seguinte regra:
se $(h_w(x) > 0.5)$ $y =$ SIM, se não $y =$ NÃO.

Entretanto, pode-se ter conjuntos de treinamentos mais complexos. Em tais casos, a Regressão Linear pode não ser plausível para solucionar o problema, como segue:



Portanto, mesmo quando sabemos que a saída de nosso problema é 0 ou 1, é estranho que $h_w(x)$ possa assumir valores > 1 ou < 0. Tal como, se sabemos que nossa saída está no intervalo $h_w(x) \in [0, 1]$, podemos aplicar a **Regressão Logística** para tal caso. Embora seja chamado de Regressão Logística, ela é uma técnica de **classificação**.

Modelo da Regressão Logística



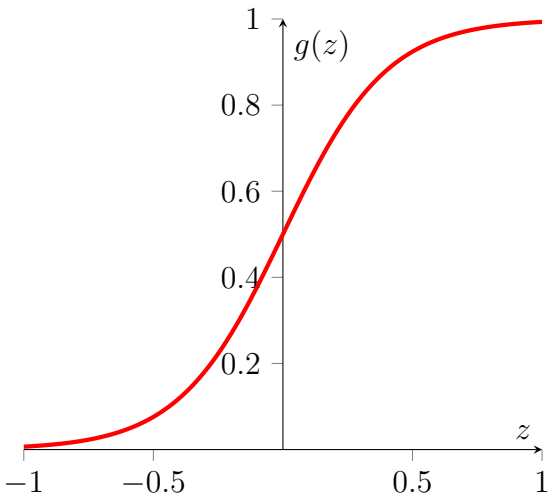
Queremos $0 \leq h_w(x) \leq 1$, com $h_w(x) = g(w^T x)$, onde $g(z) = \frac{1}{1+e^{-z}}$.
Portanto, temos que:

$$h_w(x) = \frac{1}{1 + e^{-w^T x}}. \tag{15}$$

Basicamente, a técnica de Regressão Logística tem por objetivo estimar w. Podemos interpretar a saída de hipótese como segue:

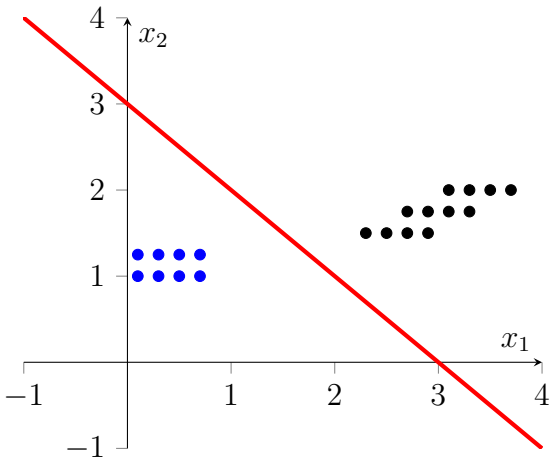
$$h_w(x) = P(y = 1 \mid x; w), \tag{16}$$

onde significa a probabilidade de $y = 1$ dado x parametrizado por w.



$$g(z) \geq 0.5 \text{ quando } z \geq 0$$
$$h_w(x) = g(w^T x) \geq 0 \text{ quando } w^T x \geq 0$$
$$h_w(x) = g(w^T x) < 0 \text{ quando } w^T x < 0$$

Suponha que tenhamos a seguinte situação:



$x_1 + x_2 = 3$ (LIMITE DE DECISÃO), corresponde a $h_w(x) = 0.5$

$$h_w(x) = g(w^T x) = g(w_0 + w_1 x_1 + w_2 x_2), \text{ onde}$$

$$W = \begin{bmatrix} -3 \\ 1 \\ 1 \end{bmatrix}$$

Portanto, queremos prever $y = 1$ se $-3 + x_1 + x_2 \geq 0 \Rightarrow x_1 + x_2 \geq 3$.

OBS: Falta exemplo do círculo.

Vamos sumarizar o nosso problema:

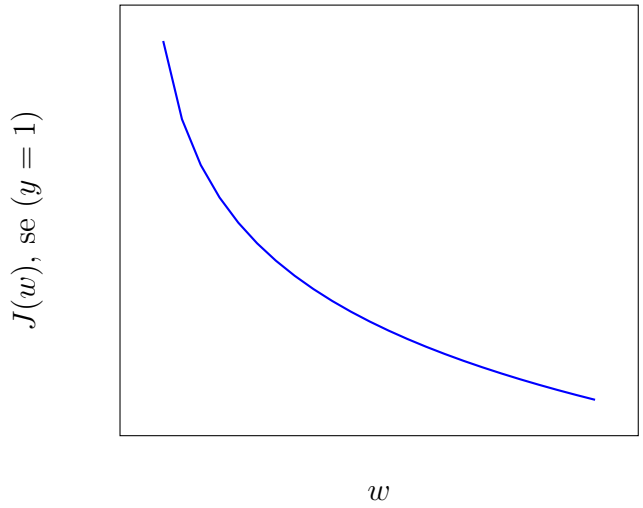
conjunto de treinamento: $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, $x_i \in \mathbb{R}^{n+1} \Rightarrow x_i^0 = 1$, $y_i \in \{0, 1\}$

função de hipótese: $h_w(x) = \frac{1}{1+e^{w^T x}}$
Como escolher parâmetros?

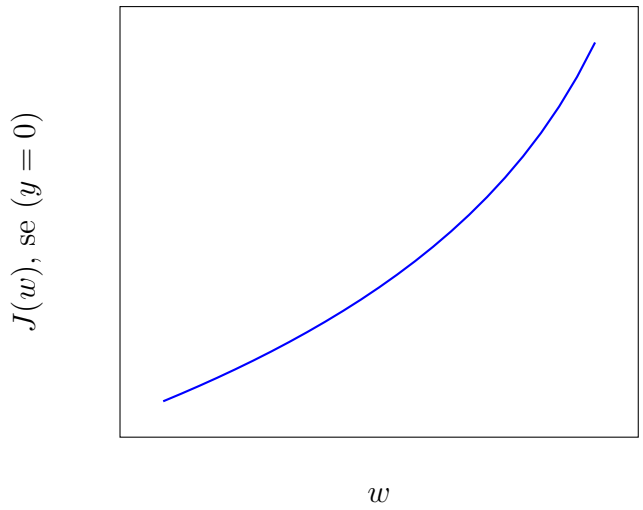
função de custo: $J(w) = \frac{1}{m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2$
Entretanto, como $h_w(x)$ agora é não linear, podemos ter uma função de custo não convexa, portanto há a necessidade de utilizar outra função de custo.

Seja $custo(h_w(x), y) = (h_w(x) - y)^2$ (MSE). Agora, devemos utilizar uma função de custo diferente:

$$custo(h_x(x), y) = \begin{cases} -log(h_w(x)), & \text{se } y = 1 \\ -log(1 - h_w(x)), & \text{se } y = 0. \end{cases} \tag{17}$$



Se $y = 1$ e $h_w(x) = 1$, então $custo(h_w(x), y) = 0$ (CORRETO).
Se $y = 1$ e $h_w(x) = 0$, então $custo(h_w(x), y) = \infty$ (ERRADO).



Se $y = 0$ e $h_w(x) = 0$, então $custo(h_w(x), y) = -log(1 - 0) = -log(1) = 0$ (CORRETO).
Se $y = 0$ e $h_w(x) = 1$, então $custo(h_w(x), y) = -log(1 - 1) = -log(0) = \infty$ (ERRADO).

Podemos reescrever a Equação 17 como segue:

$$custo(h_w(x), y) = -ylog(h_w(x)) - (1 - y)log(1 - h_w(x)). \quad (18)$$

Portanto, a função de custo geral (todo conjunto de treinamento) considerando a técnica de Regressão Logística pode ser dada por:

$$J(w) = \frac{1}{m}\sum_{i=1}^{m} custo(h_w(x_i), y_i) \tag{19}$$

$$= \frac{-1}{m}[\sum_{i=1}^{m} y_i log(h_w(x_i)) + (1 - y_1) log(1 - h_w(x_i))]$$

Novamente, temos o seguinte problema de minimização: $min_w \quad J(w)$. Podemos solucionar o seguinte problema de optimização:

repita até convergência {

$w_j \leftarrow w_j - \alpha \frac{\partial J(w)}{\partial w_j}$

}

Temos que:

$$\alpha \frac{\partial J(w)}{\partial w_j} = \frac{1}{m}\sum_{i=1}^{m}(h_w(x_i) - y_i)x^j \tag{20}$$

Então, o algoritmo do Gradiente Descendente é dado por:

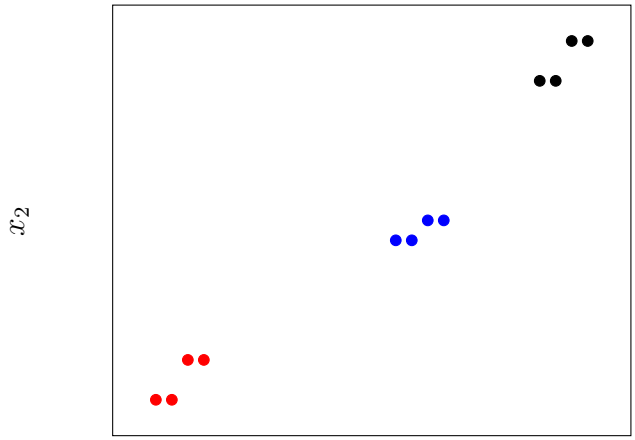
repita até convergência {

$w_j \leftarrow w_j - \alpha(\sum_{i=1}^{m}(h_w(x_i) - y_i)x_i^j$

}

3.2 Problema de múltiplas classes

A ideia da classificação de múltiplas classes é considerar o problema de aprender um espaço de características de C classes, como segue:



$$y = \{0, 1, \ldots, C-1\}$$

Para lidar com tal problema, podemos considerar a abordagem **um-contra-todos** (*one-versus-all* - OVA). Basicamente, se temos um problema de classificação de C-classes, podemos mapeá-lo em um problema de classificação de C-duas classes.

Portanto, temos a seguinte regra de decisão:

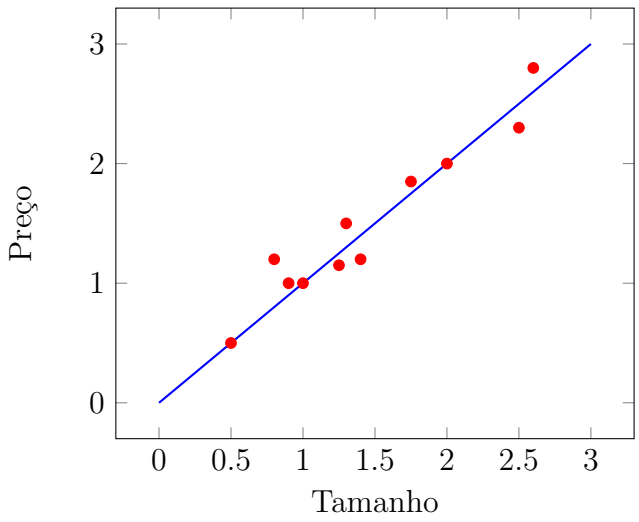
$$max_i \quad h_w^i(x) = P(y = 1 \mid x; w), \forall i = 1, 2, 3, \ldots, C \tag{21}$$

Em suma, necessitamos treinar um classificador de regressão logística $h_w^i(x)$ para cada classe i para prever o teste de probabilidade $y = i$. Em uma nova amostra de entrada x, podemos escolher sua classe i como a qual maximiza a Equação 21.

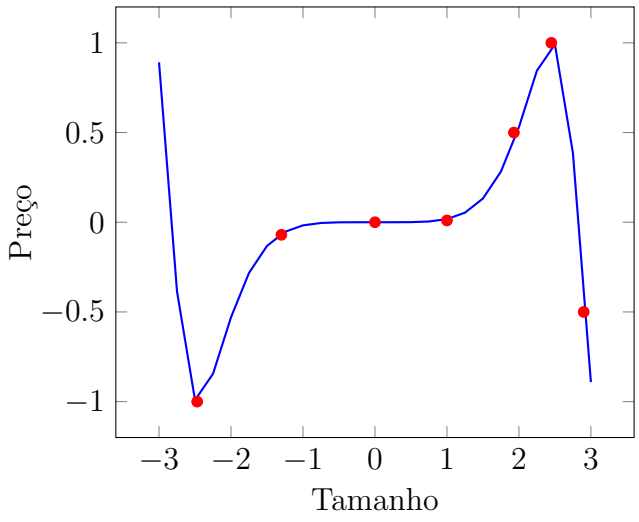
4 Regularização

4.1 *Overfitting*

Voltemos ao problema de predizer os preços das casas considerando os seguintes exemplos:



Podemos perceber que o preço das casas pode não crescer linearmente. Dentro desse espectro de linearidade, nosso modelo pode subtreinar os dados de treinamento.



A função tenta "passar"(adaptar) através de todas as amostras de treinamento, causando o problema de supertreinamento (modelo muito específico).

Portanto, se tivermos muitas características, a hipótese aprendida pode acomodar bem os dados de treinamento ($J(w) \approx 0$), mas também pode falhar em generalizar novos exemplos (predizer os preços de novas casas).

Como podemos checar se tal situação ocorre ou não em nosso modelo? Se tivermos apenas duas características, podemos apenas visualizar o espaço de características. Caso contrário, podemos tentar realizar o seguinte:

- reduzir a quantidade de características (pode não ser a melhor ideia);
- regularização: manter todas as características, mas reduzindo os valores dos parâmetros w_j (funciona bem quando possuímos muitas características).

Suponha que penalizemos e fazemos w_3 e w_4 muito pequenos:

$$min_w \frac{1}{2m} \sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + 1000w_3 + 1000w_4. \tag{22}$$

Com isso, temos que w_3 e $w_4 \approx 0$. Na equação acima, se quisermos abrandar a influência de w_3 e w_4, podemos multiplicá-los por um valor alto (por exemplo 1000). Basicamente, a ideia é ter valores **pequenos** para $w = (w_0, w_1, w_2, \ldots, w_n)$ para ter uma função de hipótese mais simples (suave).

Basicamente, podemos pegar nossa função de custo (Equação 1) e adicionar um segundo termo à ela, como segue:

$$J(w) = \frac{1}{2m} [\sum_{i=1}^{m} (h_w(x_i) - y_i)^2 + \lambda \sum_{j=1}^{n} w_j^2], \tag{23}$$

onde λ é o parâmetro de regularização que controla a relação entre a não regularização ($\lambda = 0$) e a regularização ($\lambda = \infty$).

4.2 Regressão Linear Regularizada

Nesta seção, iremos discutir sobre a regularização na abordagem de regressão linear. Novamente, nossa ideia é minimizar a Equação 23, ou seja, encontrar w o qual minimiza $J(w)$.

Portanto, podemos reescrever o algoritmo do Gradiente Descendente a fim de considerar a Equação 23 como segue:

repita até convergência {

$w_0 \leftarrow w_0 - \alpha \frac{1}{m} \sum_{i=1}^{m} (h_w(x_i) - y_i) x_i^0$

$w_j \leftarrow w_j - \alpha [\frac{1}{m} \sum_{i=1}^{m} (h_w(x_i) - y_i) x_i^j + \frac{\lambda}{m} w_j]$

}

Podemos generalizar o problema para uma formulação matricial também:

$$X = \begin{bmatrix} x_1^0 & x_1^1 & x_1^2 & \dots & x_1^n \\ x_2^0 & x_2^1 & x_2^2 & \dots & x_2^n \\ \vdots & \ddots & & & \vdots \\ x_m^0 & x_m^1 & x_m^2 & \dots & x_m^n \end{bmatrix}, \in \mathbb{R}^{m \times (n+1)}$$

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \in \mathbb{R}^{m \times 1}$$

Podemos reescrever a Equação 14 a fim de considerar o termo de regularização:

$$w = (X^T X + \lambda A)^{-1} X^T Y \tag{24}$$

onde A é uma matriz $(n+1) \times (n+1)$, como segue:

$$A = \begin{bmatrix} 0 & 0 & 0 & \dots & 0 \\ 0 & 1 & 0 & \dots & 0 \\ 0 & 0 & 1 & \dots & 0 \\ \vdots & & & \ddots & \vdots \\ 0 & 0 & 0 & \dots & 1 \end{bmatrix}, \in \mathbb{R}^{m \times (n+1)}$$

Exemplo para $n = 2$:

$$A = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

A razão para tal situação é garantir que $(X^T X + \lambda A)^{-1}$ seja inversível.

4.3 Regressão Logística Regularizada

Nesta seção, iremos discutir sobre o classificador de Regressão Logística com o termo de regularização. Podemos reescrever a Equação 19 como segue:

$$-1[\frac{1}{m}[\sum_{i=1}^{m} y_i log(h_w(x_i)) + (1-y_1)log(1-h_w(x_i))] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2. \quad (25)$$

Podemos implementar a equação acima considerando o Gradiente Descendente:

repita até convergência {

$$w_0 \leftarrow w_0 - \alpha(\sum_{i=1}^{m}(h_w(x_i) - y_i)x_i^0$$

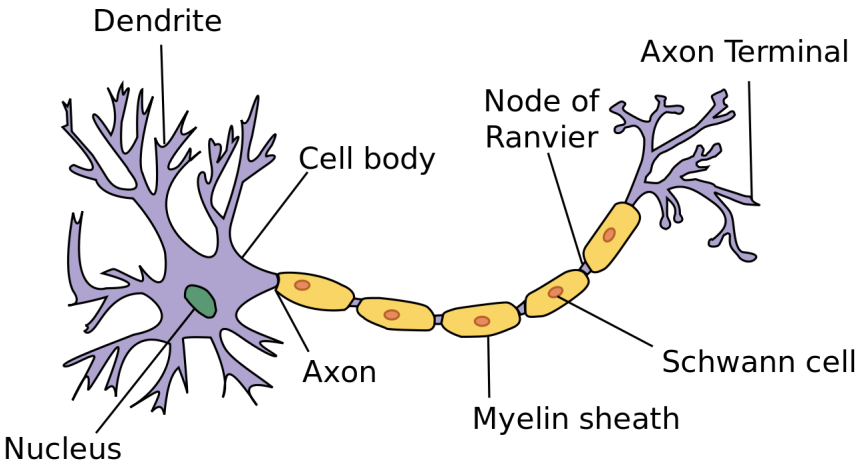
$$w_j \leftarrow w_j - \alpha(\sum_{i=1}^{m}(h_w(x_i) - y_i)x_i^j + \frac{\lambda}{m}w_j$$

}

5 Redes Neurais

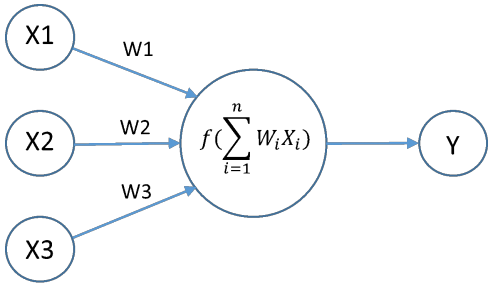
5.1 Neurônios e o Cérebro

Nesta seção, vamos discutir sobre os princípios das redes neurais. Tais técnicas tentam imitar o cérebro humano, sendo vastamente utilizadas nas décadas de 80 e 90. Com o surgimento de novas ferramentas por volta do meio da década de 90, sua popularidade acabou por decrescer. Entretanto, elas voltaram a ter um foco maior devido às abordagens baseadas em aprendizado em profundidade. Segue um exemplo de um simples neurônio:



Basicamente, os neurônios comunicam entre si por meio da emissão de pulsos elétricos a partir de suas entradas (dendrito) para suas saídas (axônio).

Matematicamente falando, podemos representar um simples neurônio como segue:



Uma **função de ativação** comum que pode ser utilizada é:

$$g(z) = \frac{1}{1 + e^{-z}}, \text{onde } z = w^T x \tag{26}$$

ou uma função limiar:

$$g(z) = \begin{cases} 1, & \text{if } z \geq \theta \\ 0, & \text{se } z < \theta, \end{cases} \tag{27}$$

onde θ é um limite dado.
A equação acima pode ser reescrita como segue:

$$h_w(x) = \begin{cases} 1, & \text{se } w^T x \geq \theta \\ 0, & \text{se } w^T x < \theta. \end{cases} \tag{28}$$

5.2 Perceptron

O Perceptron foi proposto formalmente por McCulloch e Pitts em 1943 com o intuito de modelar um neurônio biológico. Entretanto, ele possui uma capacidade limitada em termos de poder discriminativo, dado que ele é capaz de aprender apenas um único hiperplano.

Novamente, vamos considerar nosso conjunto de treinamento $X = \{(x_0, y_0), (x_1, y_1), \ldots, (x_n, y_n$ onde $x_i \in \mathbb{R}$ e $y_i \in \{0, 1\}$. Para uma dada amostra $x \in X$, temos a seguinte situação:

Podemos reescrever a função de ativação limiar como segue:

$$h_w(x) = \begin{cases} 1, & \text{se } w^T x - \theta \geq 0 \\ 0, & \text{se } w^T x - \theta < 0. \end{cases} \tag{29}$$

Se considerarmos $w_0 = -\theta$, então temos:

Rótulo Verdadeiro (y)	**Rótulo Proposto** $h_w(x)$	**Erro** ϵ
0	0	0
1	0	1
0	1	-1
1	1	0

$$h_w(x) = \begin{cases} 1, & \text{se } w^T x \geq 0 \\ 0, & \text{se } w^T x < 0. \end{cases} \tag{30}$$

Antes de aprender o mecanismo de funcionamento do Perceptron, temos que compreender alguns conceitos relacionados à projeções e produtos internos. Suponha que tenhamos dois vetores $w = [w_1 w_2]$ e $x = [x1x2]$. Podemos definir o seu produto interno (vetorial) entre w e x como: $w^T x$. Podemos visualizar tal ideia como segue:

$$\parallel w \parallel = \sqrt{w_i^2 + w_2^2} \in \mathbb{R} \Rightarrow \text{tamanho (norma) do vetor } w \tag{31}$$

E o produto interno pode ser representado como segue:

$$p = \text{tamanho da projeção de } w \text{ em } x.$$

Portanto, $w^T x = p \parallel w \parallel$.

Consideremos a situação onde $y = 1$ e $h_w(x) = 0$. Neste caso, $\epsilon = (y - h_w(x)) = (1 - 0) = 1$ (erro). Dado $h_w(x) = 0$, então temos que $w^T x < 0$ (função de ativação limiar).

Também temos que $w^T x < 0$ significa $p \parallel w \parallel \Rightarrow p < 0$. Entretanto, necessitamos $h_w(x) = 1$ para acertar a classificação. A fim de se obter isto, é desejável mover $w^{(t)}$ em direção à x, onde $w^{(t)}$ significa o valor de w na iteração t.

Nesta situação, podemos obter $w^{(t+1)}$ como segue: $w^{(t+1)} = w^{(t)} + \eta x$, onde η é a taxa de aprendizado.

Agora, suponha outra situação, onde $y = 0$ e $h_w(x) = 1$. Neste caso, temos que $\epsilon = (y - h_w(x)) = (0 - 1) = -1$ (erro). Portanto, temos que $w^T x \geq 0$, mas necessitamos $w^T x < 0$ dado que $y = 0$.

Portanto, podemos computar $w^{(t+1)}$ como segue: $w^{(t+1)} = w^{(t)} - \eta x$. Podemos juntar as duas equações em somente uma:

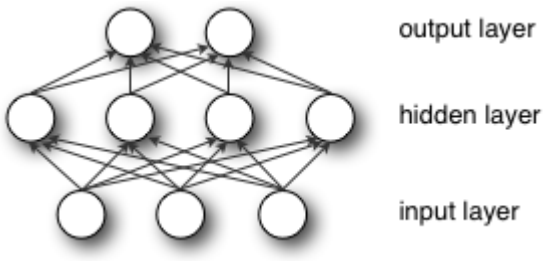
$$w^{(t+1)} = w^{(t)} + \eta \epsilon x, \text{onde } \epsilon \in \{-1, 0, 1\}. \tag{32}$$

Finalmente, o algoritmo de treinamento do perceptron é dado como segue:

1. Inicializar η;
2. Inicializar w (pesos aleatórios);
3. Aplicar $w^{(t+1)} = w^{(t)} + \eta\epsilon_i x_i, \forall i = 1, 2, \ldots, m$ e $\epsilon_i(y_i - h_w(x+i))$;
4. Repetir passo (3) até que $\epsilon_i = 0$ para todos os elementos em X.

5.3 Perceptron Multi-camadas

Uma rede neural é basicamente um grupo de neurônios, como segue:



Considerando a rede acima, temos:

$$a_1^{(2)} = g(w_{01}^{(1)}x_0 + w_{11}^{(1)}x_1 + \cdots + w_{n1}x^n)$$

$$a_2^{(2)} = g(w_{02}^{(1)}x_0 + w_{12}^{(1)}x_1 + \cdots + w_{n2}x^n)$$

$$h_w(x) = a_1^{(3)} = g(w_{01}^{(2)}a_0^{(2)} + w_{11}^{(1)}a_1^{(2)} + w_{21}^{(2)}a_2^{(2)})$$

Podemos também representar a rede neural como uma matriz de elementos como segue:

$$X = \begin{bmatrix} x^0 \\ x^1 \\ x^2 \\ \vdots \\ x^n \end{bmatrix}$$

$$Z^{(2)} = \begin{bmatrix} z_0^{(2)} \\ z_1^{(2)} \\ z_2^{(2)} \end{bmatrix} = (w^{(1)})^T X$$

$$a^{(2)} = g(z^{(2)} \Rightarrow h_w(x) = a^{(3)} = g(z^{(3)}$$

$$z^{(3)} = (w^{(2)})^T a^{(2)}$$

$Z_i^{(k)}$ refere-se aos dados de entrada do neurônio i da camada k. Note que o processo computacional das ativações das unidades de entrada, escondida e saída é chamado de *forward propagation*.

Se tivermos um problema de classificação de múltiplas classes, então o número de saída de neurônios será o mesmo que o número de classes.

Suponha que tenhamos um problema com 3 classes. Portanto, $h_w(x) \in \mathbb{R}$. Considerando o problema, cada unidade de saída é composta por 3 classificadores de regressão logística.

5.4 Parâmetros de Aprendizado

Agora, a próxima questão é: como podemos aprender o conjunto de parâmetros $w = (w^{(1)}, w^{(2)}, \ldots, w^{(L-1)})$, onde L é o número de camadas. Seja s_l o número de neurônios na camada l (não estamos considerando as unidades de *bias*).

Vamos rever a função de custo utilizada pela Regressão Logística.

$$J(w) = \frac{1}{m}[\sum_{i=1}^{m}(y_i log(h_w(x_i)) + (1 - y_i) log(1 - h_w(x_i)))] + \frac{\lambda}{2m}\sum_{j=1}^{m} w_j^2 \quad (33)$$

Considerando uma rede neural com k classes, então teremos k classificadores de regressão logística na unidade de saída. Portanto, a função de custo deve considerar esses k classificadores logísticos, como uma generalização da Equação 33:

$$J(w) = \frac{-1}{m}[\sum_{i=1}^{m}\sum_{k=1}^{k} y_i^k log(h_w^k(x_i)) + (1-y^k) log(1-h_w^k(x_i))] + \frac{\lambda}{2m}\sum_{l=1}^{L-1}\sum_{i=1}^{s_l}\sum_{j=1}^{s_{l+1}}(w_{ij}^{(l)})^2 \quad (34)$$

Nosso próximo passo consiste em utilizar um conhecido algoritmo para o aprendizado de parâmetros em redes neurais chamado de *Backpropagation*. Basicamente, queremos minimizar $J(w)$, e para tal propósito necessitamos computar as derivadas parciais $\frac{\partial J(w)}{\partial w_{ij}^{(l)}}$, onde $w_{ij}^{(0)} \in \mathbb{R}$.

Suponha que tenhamos apenas uma amostra de treinamento (x, y) e a seguinte arquitetura de rede neural:

Os passos da *forward propagation* são dados como segue:

- $a^{(1)} = x$
- $z^{(2)} = (w^{(1)})^T a^{(1)}$
- $a^{(2)} = g(z^{(2)})$
- $z^{(3)} = (w^{(2)})^T a^{(2)}$
- $a^{(3)} = g(z^{(3)})$
- $z^{(4)} = (w^{(3)})^T a^{(3)}$
- $a^{(4)} = h_w(x) = g(z^{(3)})$

O propósito do algoritmo do *Backpropagation* é relacionado ao cálculo do gradiente. Temos uma nova variável $\iota_j^{(l)}$ que significado o erro do nó j na camada l.

Considerando a rede acima, temos que:

- para cada camada de saída ($l = 4$): $\iota_j^{(4)} = a_j^{(4)} - y_j \Rightarrow \iota^{(4)} = a^{(4)} - y$
- para cada camada escondida $l \in \{2, 3\}$:

$$\iota^{(3)} = (w^{(3)})^T \iota^{(4)} * g^1(z^{(3)})$$

$$\iota^{(2)} = (w^{(2)})^T \iota^{(3)} * g^1(z^{(2)})$$

$$\iota^{(l)} = (w^{(l)})^T \iota^{(l+1)} * g^1(z^{(l)})$$

$$g^1 = \text{derivada da função de ativação}$$

Ex.: $g(z^{(3)}) = a^{(3)}$, $g^1(z^{(3)}) = a^{(3)} * (1 - a^{(3)})$

O nome *Backpropagation* refere-se ao fato de que são computadas $\iota^{(4)}$, $\iota^{(3)}$ e $\iota^{(2)}$, o que significa que retropropagamos o erro através da rede. Basicamente, as derivadas parciais são dadas por:

$$\frac{\partial J(w)}{\partial W_{ij}^{(l)}} = a_i^{(l)} \iota_j^{(l+1)} \tag{35}$$

Note que não estamos considerando o termo de regularização.

6 Máquinas de Vetores de Suporte

Com o intuito de obter os principais conceitos sobre as Máquinas de Vetores de Suporte (SVMs), podemos observar primeiramente o classificador de Regressão Logística, o qual será modificado para suprir nossas necessidades.

Novamente, temos a seguinte equação para descrever o classificador de Regressão Logística:

$$h_w(x) = \frac{1}{1+e^{-w^T x}}$$

Se $(y = 1)$, queremos que $h_w(x) \approx 1 \Rightarrow w^T x >> 0$. Se $(y = 0)$, queremos que $h_w(x) \approx 0 \Rightarrow w^T x << 0$.

Considerando a função de custo da Regressão Logística que já sabemos:

$$\begin{aligned} custo(h_w(x))y &= -(ylog(h_w(x)) + (1-y)log(1-h_w(x)) \\ &= -(ylog\frac{1}{1+e^{-w^T x}} + (1-y)log(1-\frac{1}{1+e^{-w^T x}}). \end{aligned} \tag{36}$$

Note que a função de custo acima considera apenas uma amostra de treinamento. Olhemos a função de custo geral do classificador de Regressão Logística:

$$min_w \frac{1}{m}[\sum_{i=1}^{m} y_i(-log(h_w(x_i)) + (1-y_i)(-log(1-h_w(x_i)))] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2, \tag{37}$$

a qual é basicamente a Equação 33, mas com um sinal de "−"dentro de cada termo.

Podemos reescrever a Equação 37 como segue:

$$min_w \frac{1}{m}[\sum_{i=1}^{m} y_i custo_1(w^T x) + (1-y_i)custo_o(w^T x)] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2 \tag{38}$$

Entretanto, para considerar a formulação do SVM, é necessário reescrever a Equação 38 como segue:

1. Vamos remover a constante $\frac{1}{m}$:
   $min_w[\sum_{i=1}^{m} y_i custo_1(w^T x) + (1-y_i)custo_o(w^T x)] + \frac{\lambda}{2m}\sum_{j=1}^{n} w_j^2$

2. Agora, retiremos o termo de regularização na segunda parte da Equação 38, e adicionemos ao primeiro termo, o qual é nomeado agora de C. A otimização é mantida similar se $C = \frac{1}{\lambda}$. Portanto, temos a seguinte função de custo geral considerando SVM:

$$min_w C \sum_{i=1}^{m} [y_i custo_1(w^T x) + (1 - y_i) custo_0(w^T x)] + \frac{1}{2} \sum_{j=1}^{n} w_j^2 \quad (39)$$

Note que as SVMs não retornam as probabilidades como a Regressão Logística faz, mas retorna a classe $h_w(x) \in \{0, 1\}$ para uma dada amostra x:

$$h_w(x) = \begin{cases} 1, & \text{se } w^T x \geq 0 \\ 0, & \text{caso contrário.} \end{cases} \quad (40)$$

Basicamente, a ideia agora é aprender w.

Quando C for muito grande, o primeiro termo da Equação 39 tende a 0. Neste contexto, temos algo parecido com isso:

$$min \quad C \times 0 + \frac{1}{2} \sum_{j=1}^{n} w_j^2 \quad (41)$$

$$\text{sujeito à } w^T x_i >> 1 \text{ se } (y_i = 1) \text{ ou } w^T x_i << -1 \text{ se } (y_i = 0).$$

Novamente, o problema de otimização refere-se à procura de parâmetros do hiperplano que melhor separam os dados. Portanto, as SVMs tentam encontrar o hiperplano com a margem **máxima**.

Relembrando:

$w^T x = p \parallel w \parallel$, onde p é o tamanho da projeção de x em w. Sabemos que $\parallel w \parallel = \sqrt{\sum_{j=1}^{n} w_j^2}$, a qual é a norma do vetor w. Podemos reescrever a Equação 41 como segue:

$$min_w \quad \frac{1}{2} \parallel w \parallel^2 \quad (42)$$

$$\text{sujeito à } w^T x_i >> 1 \text{ se } (y_i = 1) \text{ ou } w^T x_i << -1 \text{ se } (y_i = 0).$$

Portanto, temos que $w^T x_i = p_i \parallel w \parallel$. Podemos modificar a Equação 42 como segue:

$$min_w \quad \frac{1}{2} \parallel w \parallel^2 \tag{43}$$

$$\text{sujeito à } p_i \parallel w \parallel >> 1 \text{ se } (y_i = 1) \text{ ou } p_i \parallel w \parallel << -1 \text{ se } (y_i = 0).$$

A probabilidade 1 leva à $\sum_{i=1}^{m} p_i$ menor que a mesma soma considerando a possibilidade 2. Se $\sum_{i=1}^{m} p_i$ for muito grande, w necessita ser pequeno (lembre-se que estamos tentando minimizar w).

6.1 Trabalhando com Núcleos

Suponha que possamos representar o limite de decisão como segue:

$$h_w(x) = w_0 + w_1 l_1 + w_2 l_2 + \ldots, \text{ onde } h_1 = x_1, h_2 = x_2, \ldots \tag{44}$$

A questão é: podemos encontrar melhores calores para $l_1, l_2, \ldots$ de tal modo que o limite de decisão seja menos complexo? Suponha que tenhamos três marcações no espaço de características, por exemplo, l_1, l_2 e l_3.

Para uma dada amostra x, podemos definir:

$$l_1 = similaridade(x, l_1) = exp(\frac{-\|x-l_1\|}{2\sigma^2})$$
$$l_2 = similaridade(x, l_2) = exp(\frac{-\|x-l_2\|}{2\sigma^2})$$
$$l_3 = similaridade(x, l_3) = exp(\frac{-\|x-l_3\|}{2\sigma^2})$$

Está função $K(x, l_i)$ é chamada de função do núcleo, ativada por uma função de base radial do tipo gaussiana.

Se $x \approx l_1 \Rightarrow l_1 = exp(\frac{-0}{2\sigma^2}) = 1$.
Se x estiver distante de $l_1 \Rightarrow l_1 = exp(\frac{-\infty}{2\sigma^2}) = 0$.

Novamente, suponha que tenhamos $y = 1$ quando $w_0 + w_1 l_1 + w_2 l_2 + w_3 l_3 \geq 0$. Seja $w^T = [= 0.5\ 1\ 1\ 0]$, e considere duas amostras como x_1 e x_2:

- Para x_1: $l_1 \approx 1, l_2 \approx 0, l_3 \approx 0 \Rightarrow w_0 + w_1 l_1 + w_2 l_2 + w_3 l_3 = 0.5 + 1 = 0.5 > 0$.

- Para x_2: $l_1 \approx 0, l_2 \approx 0, l_3 \approx 0 \Rightarrow w_0 + w_1 l_1 + w_2 l_2 + w_3 l_3 = 0.5 < 0$.

Basicamente, também vamos terminar com uma função de decisão complexa.

A principal ideia é escolher cada exemplo de treinamento como uma marcação. Portanto, se tivermos m exemplos, deveremos ter $l_i = x_i, x = 1, 2, \ldots, m$.

Então, dado uma exemplo x:

$$f_i = similaridade(x, l_i) \Rightarrow f^T = [f_0 \ f_1 \ \ldots f_m].$$

Portanto, $x_i \in \mathbb{R}^{m+1}$ será representado como $f^i = [f_0^i \ f_1^i \ldots f_m^i] \ in \mathbb{R}^{m+1}$.

Em suma, considerando SVM com núcleos, temos:

- Hipótese: dado x, calcula características $f \in \mathbb{R}^{m+1}$. Predizer $y = 1$ se $w^T f \geq 0$.

A etapa de treinamento consiste em solucionar o seguinte problema de minimização.

$$min_w \ C \sum_{i=1}^{m} y_i \ custo_1(w^T f^i) + (1 - y_i) \ custo_0(w^T f^i) + \frac{1}{2} \sum_{j=1}^{m} w_j^2 \qquad (45)$$

O principal problema é escolher o parâmetro C, bem como os parâmetros do núcleo.

7 Aprendizado Não-Supervisionado

Vimos nas aulas anteriores que, dado um conjunto de treinamento rotulado $X = \{(x_1, y_1), (x_2, y_2), \ldots, (x_m, y_m)\}$, um algoritmo de aprendizado supervisionado tem por objetivo aprender um limite de decisão o qual melhor separa as amostras de diferentes classes.

Basicamente, a ideia principal é alimentar o algoritmo não-supervisionado com dados, com o intuito de encontrar padrões. Portanto, nossa base de dados é modelada como $X = \{x_1, x_2, \ldots, x_m\}$. Como a ideia é encontrar agrupamentos contidos nos dados, as técnicas não-supervisionadas geralmente são denominadas de agrupamentos.

7.1 K-médias

O algoritmo do k-médias é um dos mais utilizados para a classificação não-supervisionada. Ele possui duas etapas principais:

- atribuição dos agrupamentos;
- atualização dos centros.

Basicamente, utilizando o número de agrupamentos (médias) como uma entrada (k), escolhemos aleatoriamente k amostras. Após, cada amostra do conjunto de dados é associada ao seu vizinho mais próximo. Adicionalmente, podemos computar os novos centros baseados nos valores médios da amostra que pertence a cada agrupamento. O algoritmo termina de iterar quando os novos centros não mudam suas posições durante as iterações.

Quando os agrupamentos estão bem-separados, o algoritmo do k-médias deve funcionar muito bem. A maioria dos algoritmos supervisionados vistos são modelados por uma função a ser otimizada, assim como no caso do k-médias.

Seja k o número de agrupamentos, c_i o número do agrupamento para o qual x_i foi atribuído, μ_k o centro do agrupamento ($x, \mu_k \in \mathbb{R}^n$), e μ_{c_i} o centro do agrupamento para o qual x_i foi atribuído. A função de otimização é dada por:

$$J(c_1, c_2, \ldots, c_m, \mu_1, \mu_2, \ldots, \mu_k) = \frac{1}{m} \sum_{i=1}^{m} \parallel x_i - \mu_{c_i} \parallel . \tag{46}$$

Portanto, a ideia é solucionar o seguinte problema:

$$min\ J(c_1, c_2, \ldots, c_m, \mu_1, \mu_2, \ldots, \mu_k). \tag{47}$$

O k-médias pode ser implementado como segue:

inicialize aleatoriamente k centros de agrupamentos $\mu_1, \mu_2, \ldots, \mu_k \in \mathbb{R}^n$

repita até convergência{

para $i = 1$ até n

$c_i \leftarrow index \in \{1, 2, \ldots, k\}$ do centro do agrupamento mais perto de k_i

para $i = 1$ até k

$u_i \leftarrow$ média de todos os pontos atribuídos ao agrupamento i

}