

# Curso Machine Learning

## Aprendizado supervisionado

### Ridge regression

\* Cost function é uma função que tem como objetivo retornar o erro quadrático médio de uma predição em relação à saída verdadeira, pertencente aos dados de treinamento.

Cost function para uma feature

$$J(w, b) = \frac{1}{2m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})^2$$

Predito

desconhecido

$m \rightarrow$  número de dados de treinamento  
 $x^{(i)} \rightarrow$   $i$ -ésimo dado

O objetivo é encontrar os valores adequados para que a função  $J(w, b)$  seja mínima.

$$\min_{w_1, \dots, w_n, b} J(w_1, w_2, \dots, w_n, b)$$

OBS: Podem existir vários mínimos locais.  
 Em um gráfico de 3 dimensões é possível identificar isso através dos raios que surgem ao longo da superfície

### Gradiente Descendente

Para ajustar os valores de  $w$  e  $b$  utilizando derivadas de  $J(w, b)$  em cada termo. Assim  $w$  e  $b$  serão atualizados desta maneira:

$$w = w - \alpha \frac{d}{dw} J(w, b)$$

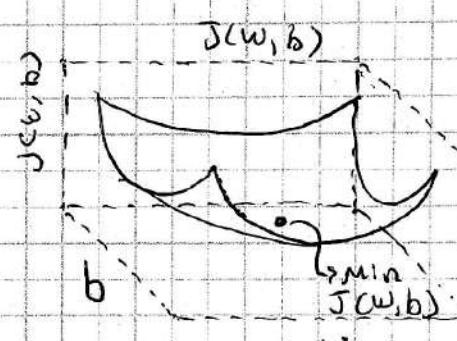
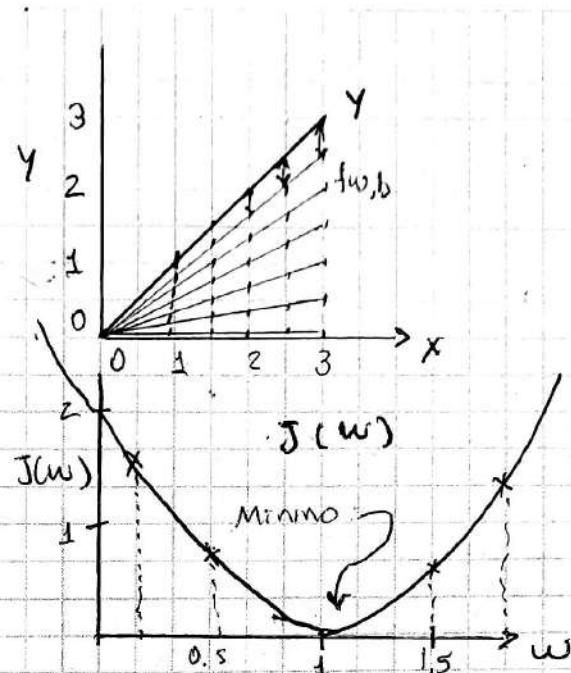
$$b = b - \alpha \frac{d}{db} J(w, b)$$

$\alpha$  = learning rate (o quão rápido os termos são ajustados. é uma constante)

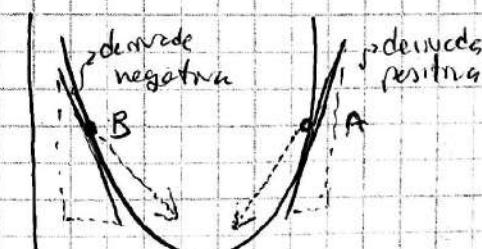
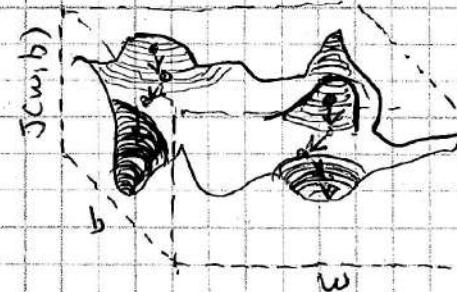
Este cálculo ocorre simultaneamente.

Ou seja, a cada iteração, os termos são calculados com o valor da rede anterior (não inserindo o  $w$  atual no cálculo de  $b$ )

O learning rate precisa ser bem ajustado pois o modelo pode demorar a aprender ou pode divergir do mínimo local.



Gradiente descendente



$$A \rightarrow w = w - \alpha \frac{d}{dw} J(w, b)$$

$$B \rightarrow w = w - \alpha \frac{d}{dw} J(w, b)$$

## Algoritmo do Gradiente descendente

Repeat until convergence um feature I

$$w = w - \alpha \left\{ \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)}) x^{(i)} \right\}$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(x^{(i)}) - y^{(i)})$$

$$\text{I} \rightarrow \frac{d}{dw} J(w, b) \quad \text{II} \rightarrow \frac{d}{db} J(w, b)$$

## Múltiplas features

Quando se trata de múltiplos entradas, recorremos à representação matemática vetorial. Isso é conhecido como vectorization e pode ser entendido como operações com matrizes e vetores, em geral, para implementação dos algoritmos com performance otimizada.

Sem vectorization os algoritmos são executados como em um loopatório, uma iteração. Mas com vectorization temos o processamento em paralelo dos dados, o que fazia o aprendizado muito rápido e computacionalmente mais eficiente do ponto de vista do código fonte.

## Feature Scaling

Parâmetros podem apresentar, em conjunto com features, um comportamento não desejado. Pequenas variações em um deles parâmetro podem impactar mais do que pequenas variações em outros parâmetros. Isso ocorre porque as features estão em escalas diferentes e, partindo de  $w_1, w_2, \dots, w_n$ , se variarem, causam maior ou menor efeito.

Exemplo de um dataset de preço de imóveis

$x_1$	$x_2$
Nº de quartos	Nº de m²
5	2000
1	1
0	300

$$w_1 \cdot x_1 \quad w_2 \cdot x_2$$

Alteração em  $w_2$  causa um impacto menor no preço do casal para  $x_2$  apresenta valores entre 300 e 2000

n features ( $n \geq 2$ )

repeat {

$$w_1 = w_1 - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(\vec{x}^{(i)}) - y^{(i)}) x_1^{(i)}$$

$$w_n = w_n - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(\vec{x}^{(i)}) - y^{(i)}) x_n^{(i)}$$

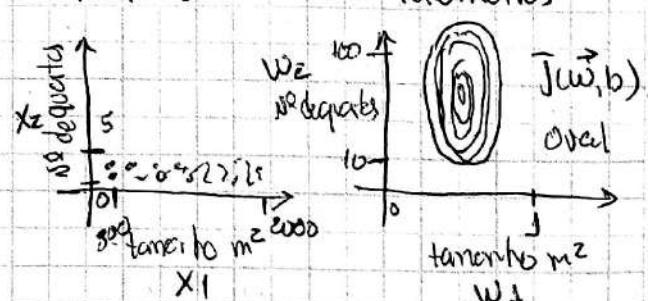
$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{w,b}(\vec{x}^{(i)}) - y^{(i)})$$

Simultaneous update

$w_j$  (for  $j = 1, \dots, n$ ) and  $b$

}

Features



Parâmetros



Isso aumenta a velocidade de convergência do aprendizado (Feature Scaling)

## Mean normalization

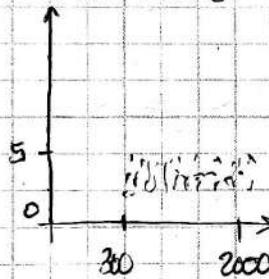
É uma técnica para ajustar os features em uma escala que tem como base a média aritmética simples dos dados. Supomos que  $300 \leq X_1 \leq 2000$  e  $0 \leq X_2 \leq 5$ . Além disso temos que o valor médio de  $X_1$  é  $\mu_1 = 600$  e o valor médio de  $X_2$  é  $\mu_2 = 2.3$ . Os novos valores para  $X_1$  e  $X_2$  (escalados) são dados por:

$$X_1' = \frac{X_1 - \mu_1}{X_{1\text{max}} - X_{1\text{min}}} \quad \text{e} \quad X_1' = \frac{X_1 - 600}{2000 - 300}$$

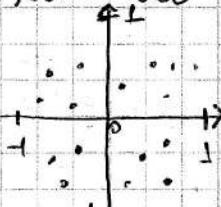
$$X_2' = \frac{X_2 - \mu_2}{X_{2\text{max}} - X_{2\text{min}}} \quad \text{e} \quad X_2' = \frac{X_2 - 2.3}{5 - 0}$$

$$-0.18 \leq X_1' \leq 0.82 \quad -0.46 \leq X_2' \leq 0.54$$

Não Normalizado

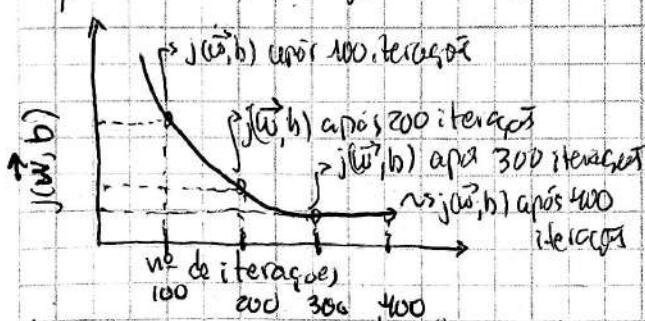


Normalizado



## Gradiente descendente e convergência

Um ponto importante ao lidar com o gradiente descendente é observar sua convergência. Nessa perspectiva, avaliaremos a cost function sua evolução ao longo do tempo, ou mais precisamente, ao longo das iterações.



- \* O teste de convergência automática considera um "epsilon". É de ordem de  $10^{-3}$  ou 0,001. Se  $J(w, b)$  decresce a uma taxa  $\leq \epsilon$  em uma iteração, então, declararmos convergência.

## Z-score normalization

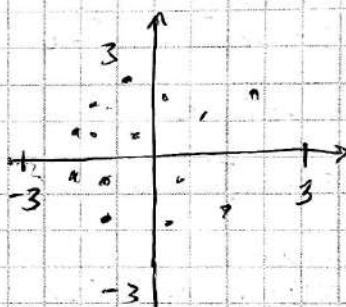
Outra técnica - normalização mais utilizada para ajuste de escala das features, que utiliza como base o desvio padrão e o valor médio de determinado vetor ( $X_1$ , ou  $X_2$ , ou ...  $X_n$ )

Essa técnica é descrita pelas seguintes relações:

$$X_{1\text{min}} \leq X_1 \leq X_{1\text{max}} \quad X_{2\text{min}} \leq X_2 \leq X_{2\text{max}}$$

$$X_1 = \frac{X_1 - \mu_1}{\sigma_1} \quad X_2 = \frac{X_2 - \mu_2}{\sigma_2}$$

Sendo  $\mu_1$  e  $\mu_2$  as médias de  $X_1$  e  $X_2$  respectivamente e  $\sigma_1$  e  $\sigma_2$  seus desvios padrão. Aplicando o Z-score, temos

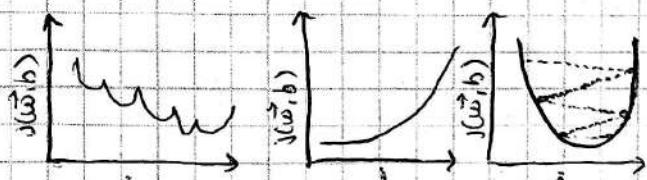


Usar-se:

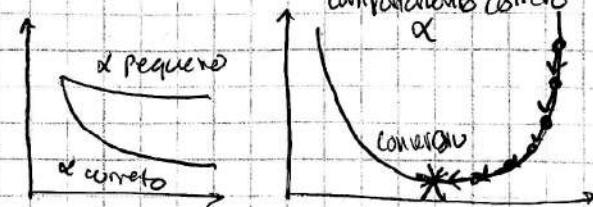
- \* X foi números reais de que 3, ou valores de que -3
- \* se forem too pequenos que a convergência fique preguiçoso.

## Escolhendo o learning Rate $\alpha$

Se o gráfico de evolução de  $J(w, b)$  apresentar oscilações crescentes quando tendeões esferas ao infinito ou formato parabólico,  $\alpha$  está muito grande

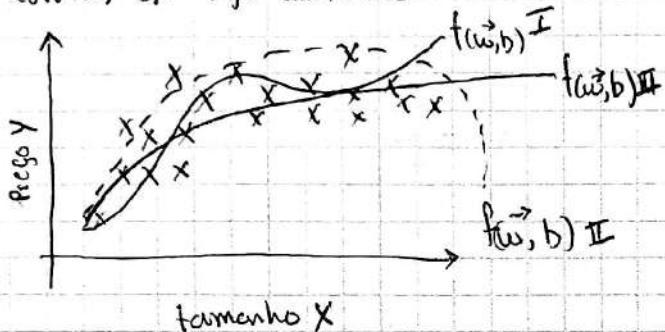


Quando  $J(w, b)$  decai vagamente, o gráfico da evolução se parece com uma curva suave comportamento correto



## Régressão Polinomial

Uma visão geral sobre os dados pode nos mostrar que a regressão linear não satisfaz nossas necessidades. Quando a distribuição dos dados é não-linear, deve-se fazer uma engenharia na de features, aplicando novas colunas que são combinações ou potências/raízes de colunas ( $X^i$ ) já existentes.



$$I \rightarrow f(\vec{w}, b)(x) = w_1 x_1 + w_2 x^2 + w_3 x^3 + b$$

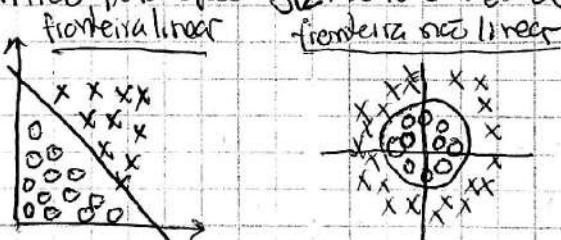
$$II \rightarrow f(\vec{w}, b)(x) = w_1 x + w_2 x^2 + b$$

$$III \rightarrow f(\vec{w}, b)(x) = w_1 x + w_2 \sqrt{x} + b$$

Obs: Aplicando engenharia de features deve-se fazer também a feature scaling, pois se fizermos termos que são potências de uma feature, também termos escala queremos essas potências.

## Frontera de decisão

Para um threshold de 0.5 na função sigmoid,  $\hat{y}$  será 1 se  $z \geq 0$ , e  $\hat{y}$  será 0 se  $z < 0$ . A fronteira de decisão é uma função que descreve os valores das features ou como se combinam esses valores em que  $z$  assume de fronteira de valores crítico para que  $g(z)$  bata o threshold.



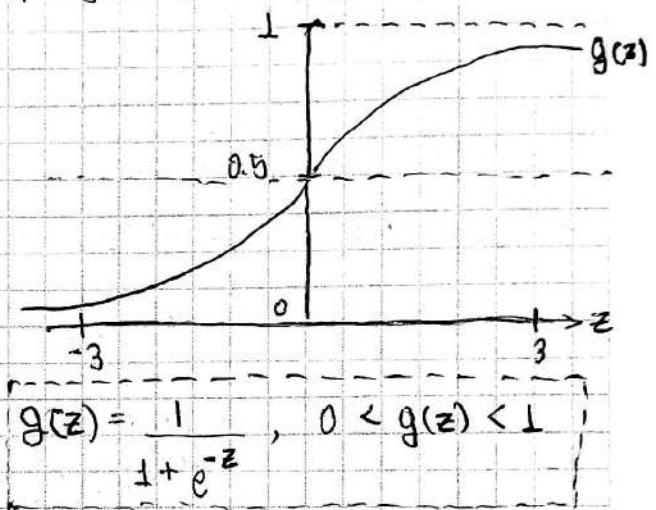
No gráfico, a fronteira de decisão demarca da complexidade de  $z$

$$y = \begin{cases} 1, & z \geq \text{threshold} \\ 0, & z < \text{threshold} \end{cases}$$

4

## Régressão Logística

Técnica utilizada para decidir acerca de uma condição booleana. Ex: se um e-mail é ou não é um spam, se a foto é bonita ou não é. O algoritmo usa como base a fronteira de decisão, um ponto em que considera-se sim ou não em relação a algo que está sendo avaliado. Isso é determinado por uma função matemática conhecida como sigmoid function ou função sigmoidide,  $g(z)$ .



Avaliável  $z$  da função sigmoidide é dada por:

$$z = \vec{w} \cdot \vec{x} + b$$

Logo, a probabilidade da regressão logística é uma função  $f(\vec{w}, b)(\vec{x})$  do tipo:

$$f(\vec{w}, b)(\vec{x}) = g(\vec{w} \cdot \vec{x} + b) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

Usando a notação de probabilidade, sabemos que a probabilidade de  $y=0 + y=1$  é:

$$P(Y=0) + P(Y=1) = 1$$

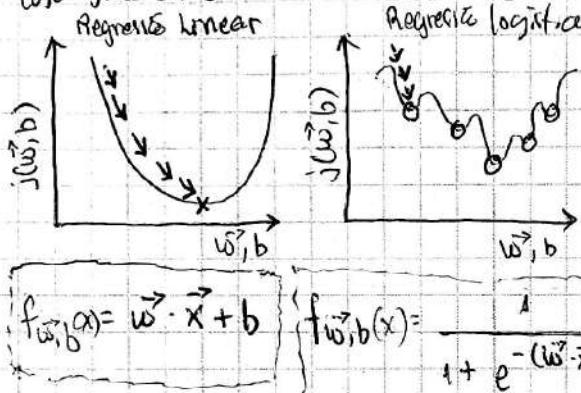
Assim,  $f(\vec{w}, b)$  é uma função tal que:

$$f(\vec{w}, b)(\vec{x}) = P(Y=1 | \vec{x}; \vec{w}, b)$$

Obs: Numa lógica estatística também usam  $P(1)$  para representar a probabilidade de ser 1.

## cost function for logistic regression

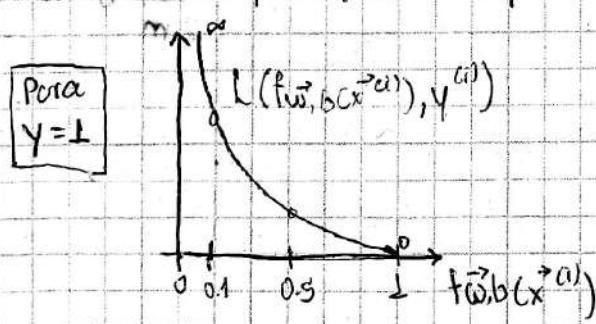
um dos motivos pelos quais não se usa a função de custo utilizada na teoria de regressão linear é devido à natureza não-convexa da função em regressão logística. Isso é importante pois por não ser convexa ( $J(\vec{w}, b)$ ) permite a existência de vários mínimos locais cujo gradiente descendente convergeira.



Portanto, temos uma função  $L$  que retorna uma medida de erro sobre a predição feita pelo algoritmo, e é dada por:

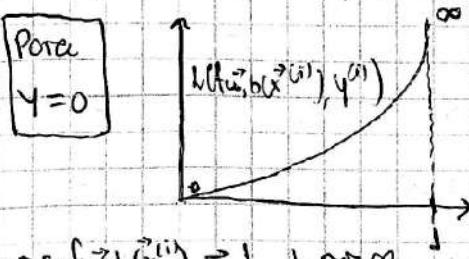
$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = \begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{se } y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})) & \text{se } y^{(i)} = 0 \end{cases}$$

A função  $L$  (significa loss) apresenta estas características para  $y^{(i)} = 0$  ou  $y^{(i)} = 1$ :



$\Rightarrow$  Se  $f_{\vec{w}, b}(\vec{x}^{(i)}) \rightarrow 1$ ,  $L \rightarrow 0$

$\Rightarrow$  Se  $f_{\vec{w}, b}(\vec{x}^{(i)}) \rightarrow 0$ ,  $L \rightarrow \infty$



$\Rightarrow$  Se  $f_{\vec{w}, b}(\vec{x}^{(i)}) \rightarrow 1$ ,  $L \rightarrow \infty$

$\Rightarrow$  Se  $f_{\vec{w}, b}(\vec{x}^{(i)}) \rightarrow 0$ ,  $L \rightarrow 0$

Com base nesse entendimento, temos uma função que indica o quanto estouvo errado no processo de regressão logística, e esse erro, com todos os dados de treinamento, pode ser expresso por:

$$J(\vec{w}, b) = \frac{1}{m} \sum_{i=1}^m L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)})$$

sendo a função  $L$  igual a:

$$\begin{cases} -\log(f_{\vec{w}, b}(\vec{x}^{(i)})), & y^{(i)} = 1 \\ -\log(1 - f_{\vec{w}, b}(\vec{x}^{(i)})), & y^{(i)} = 0 \end{cases}$$

### \* Loss function simplificada.

Nas formas matematicamente enxutas da escrevermos a função loss é necessário todos os termos em um única linha:

$$L(f_{\vec{w}, b}(\vec{x}^{(i)}), y^{(i)}) = -y^{(i)} \cdot \log(f_{\vec{w}, b}(\vec{x}^{(i)}))$$

$$+ (1 - y^{(i)}) \cdot \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))$$

Assim,  $(1 - y^{(i)})$  e  $y^{(i)}$  são constantes que valem 0 ou 1 e anulam as partes da equação quando convenientemente.

### \* cost function simplificada

Aplicando a simplificação na função loss, temos a seguinte expressão para a cost function simplificada:

$$\begin{aligned} J(\vec{w}, b) = & -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(f_{\vec{w}, b}(\vec{x}^{(i)})) + \\ & + (1 - y^{(i)}) \cdot \log(1 - f_{\vec{w}, b}(\vec{x}^{(i)}))] \end{aligned}$$

### gradient descent for logistic regression

O conceito de gradiente descendente para regressão logística se assemelha ao que é feito no caso da regressão linear, exceto pela função  $f_{\vec{w}, b}(\vec{x}^{(i)})$  assumir um formato diferente.

## Gradient descent for logistic regression

repeat {

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) x_j^{(i)} \right]$$

$$b = b - \alpha \left[ \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}) \right]$$

} simultaneous updates

$$f_{\vec{w}, b}(\vec{x}) = \frac{1}{1 + e^{-(\vec{w} \cdot \vec{x} + b)}}$$

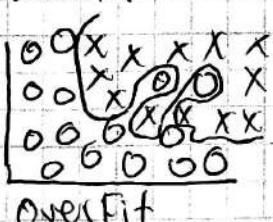
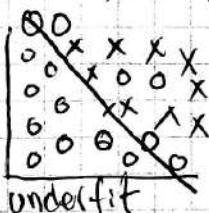
## Problema de overfitting

O grau do polinômio que construímos ~~deve~~ trabalhar com a engenharia da features ateta a forma com que nossas previsões são executadas.

under-fit: estado em que nosso polinômio não se ajusta aos dados, e portanto tem uma performance ruim. também conhecido como high bias.

just right: ou "ajustado", é o estado em que o polinômio possui boa capacidade de generalização e faz uma performance preditiva.

over-fit: quando há um sobreajuste no polinômio e isso faz com que ele seja muito bom em dados de treinamento, mas pior em previsões.



\* O grau do polinômio é responsável por esses erros no processo de predição.

Para resolver esse problema, podemos tomar as seguintes estratégias:

1. Coletar mais dados: isso ajuda a descrever melhor o formulário da função a ser construída.

2. Seleção de features: escolher quais features devem entrar no polinômio. normalmente, isso é algo da intuição do cientista de dados e pode não ser tão eficiente, mas ajuda.

3. Regularização: técnicas que reduzem a influência de algumas features multiplicando-a por uma escala, por exemplo,  $\alpha = 0,0001$ .

## Cost function com regularização

Adicionamos um termo à cost function que será responsável por reduzir a influência dos  $w_n$  valores (pesos) de forma a controlar o poder de generalização de nosso modelo.

$$+ \underbrace{\frac{\lambda}{2m} \sum_{j=1}^n w_j^2}_{\text{optional}}$$

$$+ \frac{\lambda}{2m} b^2$$

Nesse termo,  $\lambda$  é um parâmetro, também conhecido como parâmetro de regularização, que ajusta a influência do termo adicionado à cost function ( $\lambda > 0$ ). Desta forma, temos a seguinte cost function:

$$J(\vec{w}, b) = \frac{1}{2m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})^2 + \frac{\lambda}{2m} \sum_{j=1}^n w_j^2$$

$$f_{\vec{w}, b}(\vec{x}) = w_1 x + w_2 x^2 + w_3 x^3 + b$$

- Se  $\lambda \rightarrow 0$ , temos um overfit
- Se  $\lambda \rightarrow \infty$ , temos um underfit
- Se  $\lambda \rightarrow a$ , temos um ajuste bom feito

OBS: "a" representa um parâmetro razoável que apreende um resultado satisfatório

## Regularized linear regression (and logistic)

Um pequeno ajuste preciso ser feito para utilizar regularização em regressão linear. basta adicionarmos a derivada do termo de regularização ao gradiente descendente:

repetir f

$$w_j = w_j - \alpha \left[ \frac{1}{m} \sum_{i=1}^m [f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)}] x_j^{(i)} \right] +$$

$$+ \frac{\lambda}{m} w_j$$

$$b = b - \alpha \frac{1}{m} \sum_{i=1}^m (f_{\vec{w}, b}(\vec{x}^{(i)}) - y^{(i)})$$

} simultaneous update

\* Vale apenas ressaltar que esse termo é usado em regressão linear e regressão logística. Daí seja, podemos evitar o overfitting ou o underfitting apenas acrescentando o termo e ajustando a constante da regularização.

OBS: a única coisa que muda entre a regressão linear e a regressão logística em relação à regularização é a função  $f_{\vec{w}, b}(\vec{x})$ . O resto é idêntico.

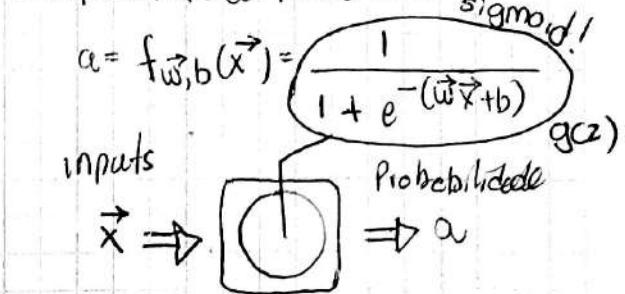
## Neural Networks

Redes neurais artificiais foram inspiradas em redes neurais biológicas. Elas se baseiam na comunicação entre os neurônios artificiais para obter um output.

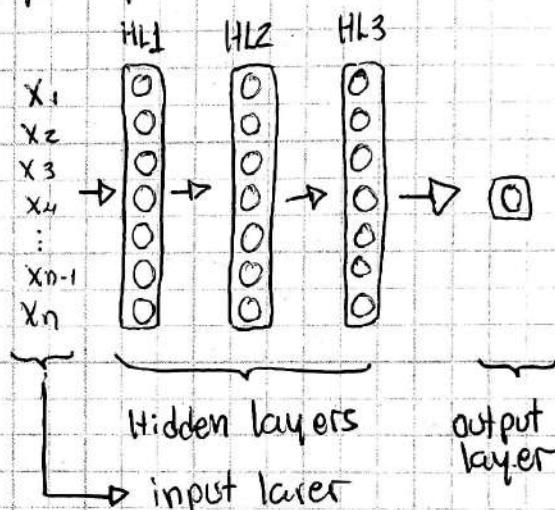
RNNs possuem camadas (layers) que processam, cada uma, uma parte dos impulsos, na verdade todos os impulsos, mas evoluem padronizados ao longo das n camadas. Ou seja, a camada 100 sera responsável por identificar mais condições do que a camada 1. Usa-se processamento sobre processamento de forma a obter de rede um output muito mais poderoso.

Cada neurônio artificial é uma unidade de regressão logística que calcula probabilidade

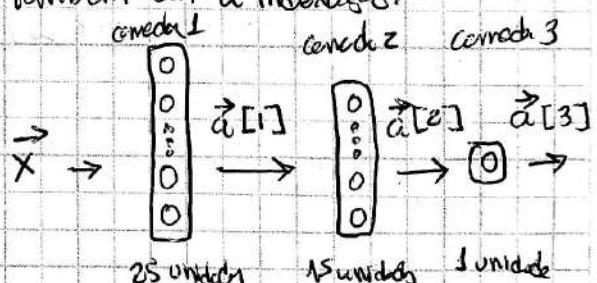
\* Representação do neurônio artificial



Os neurônios artificiais são organizados em camadas e agrupados por camadas. O resultado de um neurônio é enviado com atividades e essa "activation" flui para o processamento para o próximo neurônio.



As redes neurais artificiais são representadas também com a indexação:



$$\vec{a}[z] = \begin{bmatrix} g(\vec{w}_1^{[z]}, \vec{a}^{[z-1]} + b_1^{[z]}) \\ \vdots \\ g(\vec{w}_{15}^{[z]}, \vec{a}^{[z-1]} + b_{15}^{[z]}) \end{bmatrix}$$

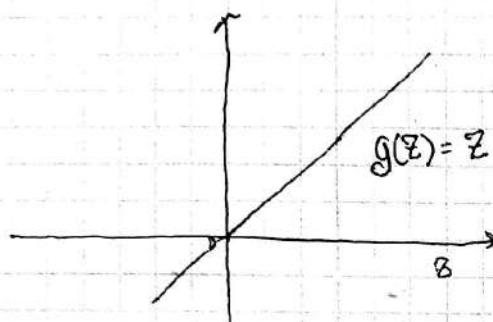
No final, temos com  $\vec{a}[3]$  a probabilidade, e dai fazendo a decisão:

$$\text{sim } a_1^{[3]} \geq 0.5? \rightarrow \begin{cases} \hat{y} = 1 & \text{sim} \\ \hat{y} = 0 & \text{não} \end{cases}$$

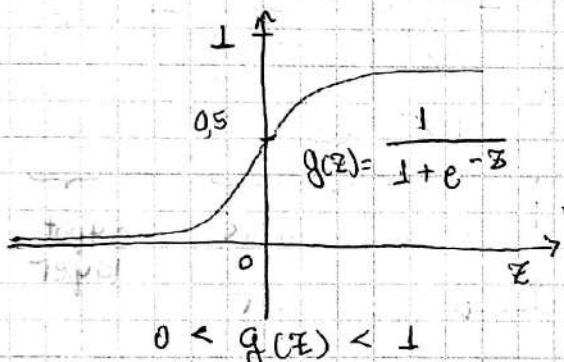
## Exemplos de Funções de Ativação

O tipo de função de ativação utilizada em redes neurais pode afetar sua performance. Vamos os modelos de funções de ativação:

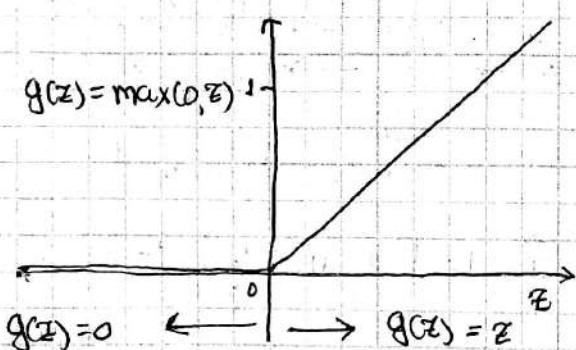
### 1. Linear activation function



### 2. Sigmoid function



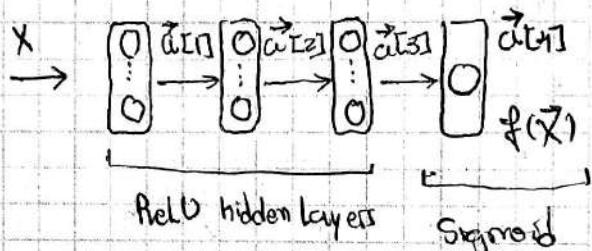
### 3. Rectified Linear Unit (ReLU)



- Linear activation function é uma boa opção se o objetivo do modelo é prever algum valor  $y$  que seja negativo ou positivo, como o saldo de uma conta bancária, que pode ser negativo ou positivo.

- Sigmoid function: normalmente usada em classificações binárias. Escolha esta opção se o objetivo for algo como o bel
- ReLU: usada quando a predição é um valor positivo,  $y > 0$ , como o preço de uma casa.

Dicas: uma estratégia interessante é empregar ReLU nos caminhos escondidos da rede neural pois elas permitem "saltos" maiores de aprendizado. Fazemos caminhos com ReLU e a última camada como sendo Sigmoid, assim, teremos um sistema de classificação binária que convergirá com uma velocidade maior



Para números muito grandes ou muito pequenos de  $z$ ,  $g(z)$  possui pequena variação,  $dg(z)/dz \rightarrow 0$ , é assim que acontece na função sigmoid.

### Multiclass Classification

Em alguns casos, desejamos escolher uma opção entre um conjunto finito de possibilidades. Por exemplo, escolher (classificar) qual número está escrito em uma imagem (MNIST).

Neste caso, para cada opção teremos um  $z_i$  associado. Se tivermos 4 opções:

$$z_1, z_2, z_3, z_4$$

A probabilidade de cada opção será dada pela divisão entre o  $z$  da opção e a soma de todos os  $z$ . Por exemplo, para cálculo, temos:

$$p_1 = \frac{e^{z_1}}{e^{z_1} + e^{z_2} + e^{z_3} + e^{z_4}}$$

## Softmax regression

Técnica usada em modelos de multiclass em que se tem  $N$  possíveis saídas, ou seja,  $Y = 1, 2, 3, \dots, N$

$$z_j = \vec{w} \cdot \vec{x} + b_j \quad j=1, \dots, N$$

As probabilidades são dadas por

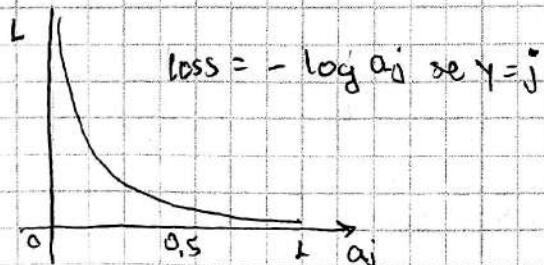
$$a_j = \frac{e^{z_j}}{\sum_{k=1}^N e^{z_k}} = P(Y=j | \vec{x})$$

Observe que, por calcular as probabilidades, a soma das  $N$  probabilidades deve ser igual a 1

$$a_1 + a_2 + a_3 + \dots + a_N = 1$$

A loss function, neste caso, assume uma hipótese com poucas alterações, ela avalia qual foi a predição do modelo e calcula o negativo do logaritmo da probabilidade daquela predição.

$$\text{Loss}(a_1, a_2, \dots, a_N) = \begin{cases} -\log a_1, \text{ se } y=1 \\ -\log a_2, \text{ se } y=2 \\ -\log a_3, \text{ se } y=3 \\ \vdots \\ -\log a_N, \text{ se } y=N \end{cases}$$



Por exemplo, se a saída escolhida for  $z_2$ ,  $y=2$ , a loss function é:

$$\text{loss} = -\log a_2$$

Observe que a loss function compensa a probabilidade quando ela se aproxima de zero e diminui em um.

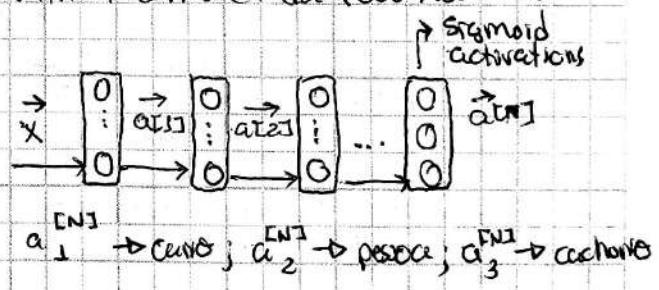
Assim, softmax entra na última camada da rede e compõe a probabilidade da cada classe, usando seu modelo probabilístico.

Em TensorFlow, softmax pode produzir entradas de precisão. Por isso, uma abordagem mais precisa para utilizar essa biblioteca seria configurar a última camada da rede neural como activation = "linear" e passar como parâmetro da função loss from\_logits = True.

## Multi-label Classification

Em alguns casos, desejamos classificar se em determinados dados temos saídas multiplas, isto é, quando a saída pode assumir o formato de um vetor contendo mais de uma escolha. Um exemplo disso seria um modelo que fosse capaz de de detectar se em uma imagem existe um carro, uma pessoa ou um cachorro. Assim, uma imagem contendo esses três elementos deveria ter um input com os dados da imagem e um output com três afirmativas.

Para fazer isso, basta empregar a função sigmoid como função de ativação na última camada da rede neural.



$a_1^{Nj} \rightarrow \text{carro}; a_2^{Nj} \rightarrow \text{pessoa}; a_3^{Nj} \rightarrow \text{cachorro}$

## Algoritmo Adam para gradiente descendente

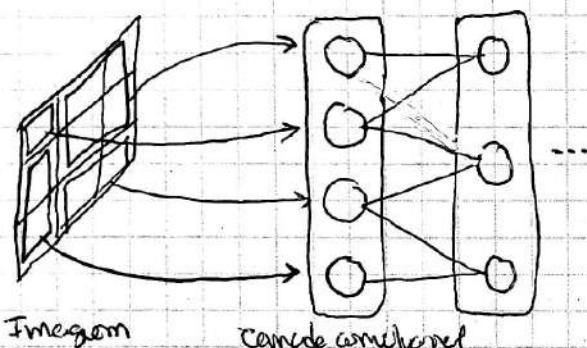
É uma forma de otimizar o treinamento do modelo. Basicamente, o parâmetro  $\alpha$  muda o tamanho do treinamento e faz com que as mudanças sejam aplicadas quando os saltos normalmente não oscilam e é reduzido quando oscilações significativas são observadas.

$$\begin{aligned} w_j &= w_j - \alpha_j \frac{\partial}{\partial w_j} j(\vec{w}, b) \\ w_{j0} &= w_{j0} - \alpha_{j0} \frac{\partial}{\partial w_{j0}} j(\vec{w}, b) \\ b &= -\alpha_b \frac{\partial}{\partial b} j(\vec{w}, b) \end{aligned}$$

$\alpha$  assume  $j$  valores  $\rightarrow \alpha_j$

## Camada Convolucional (Resumo)

Não será abordada com profundidade no curso, mas se tratam de camadas que são diferentes das outras por ter em cada de neurônios uma responsabilidade parcial dos dados. Por exemplo, em uma imagem, uma camada convolucional segmentaria e distribuiria partes para seus neurônios, que processam informações em um contexto específico.



Isto é feito para melhorar a performance da rede e também para obter resultados mais precisos. Por exemplo, certas partes da imagem parecem mais relevantes, por isso convoluções é uma forma interessante de se trabalhar.

## Avaliação de modelos

Após construirmos um modelo de ML precisamos avaliar o quanto generalizável ele é. Uma forma de fazer isso é separar uma parte dos dados para ser o conjunto de treinamento e outra para ser o conjunto de teste.

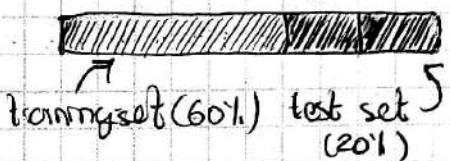
Treinando o modelo com o conjunto de treinamento, podemos verificar sua performance gerando suas previsões com os dados do conjunto de teste. Fazendo o cálculo de erro quadrático, ou seja,  $J$ , entre as previsões e o resultado do grupo de teste, o valor gerado indica o quanto o modelo está em relação ao conjunto de teste.

Existe outra forma de auxiliar, mas um modelo quando nosso objetivo é prever qual é a melhor escolha a se fazer.

na composição dos dados de treinamento, ou seja, se aumentarmos o grau do polinômio, como isso afetará a performance desse modelo em relação à generalização?

- Divisão do conjunto de dados: particionamos o conjunto de dados em três categorias: conjunto de treinamento, conjunto de cross-validation e conjunto de testes. Normalmente, seguindo a seguinte proporção:

cross-validation set (20%)  
2



O cross-validation set é o conjunto que usaremos para testar estratégias dif. entre elas, e compará-las com o modelo comum. Fazemos modelos diferentes para cada estratégia e o confrontamos com os dados do cross-validation set.

Poderemos testar polinômios com graus diferentes, novas features e até mesmo mudanças na arquitetura da rede neural afim de encontrarmos um estratégia que cominja de forma generalizativa.

O fluxo de avaliação deve ser neste sentido:

Treinar o modelo no training set      Testá-lo no test set      Calcular o erro ( $J$ )



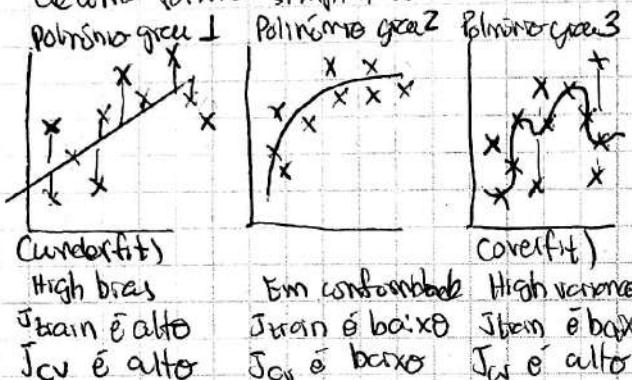
Treinar o modelo no training set      Testá-lo no cross-validation set      Calcular o erro ( $J$ )



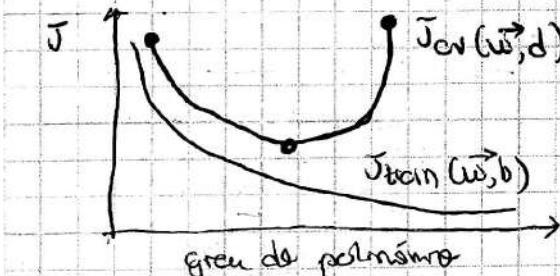
Agora, variamos a estratégia para saber qual delas apresentará o menor erro, esse será a escolha.

## Análise de Biass e Variâncias

Uma das formas que temos de avaliar o que é generalista é nosso modelo é utilizando o conceito de biass e variância. Biass significa a tendência do modelo ser impreciso por fazer suposições inadequadas, normalmente, por tentar abordar um conjunto de dados que exigem maior complexidade de uma forma simplista.



Uma forma de visualizarmos isto é verificando um gráfico que exibe o valor de  $J_{train}$  e  $J_{CV}$  em função do grau do polinômio.



Se temos um alto biass, nesse modelo está com underfit,  $J_{train}$  será alto e  $J_{CV}$  será próximo de  $J_{train}$ .

Se temos uma alta variância, ou seja, nosso modelo está com overfit, temos que  $J_{CV}$  será muito menor do que  $J_{train}$  e  $J_{train}$  será baixo.

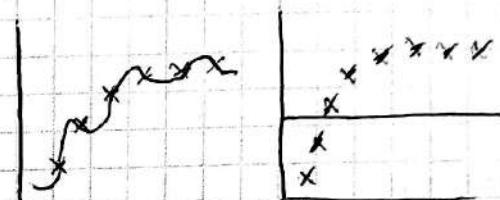
Também temos o caso em que tanto biass quanto a variância são altos.  $J_{train}$  será alto e  $J_{CV}$  será muito mais alto. É um caso mais específico em que o modelo diverge dos padrões esperados pelos dados.

## Biass e variâncias em função de $\lambda$

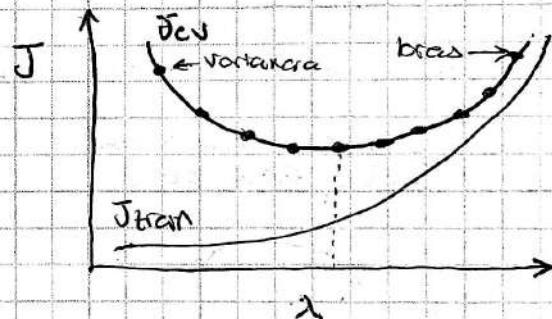
Quando utilizamos a parte de regularização

$$\frac{\lambda}{2m} \sum_{j=1}^m w_j^2$$

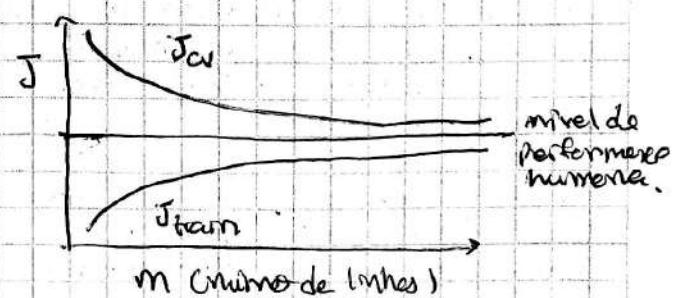
o parâmetro  $\lambda$  influencia no comportamento do modelo por atenuar os pesos do polinômio.



Se compararmos  $J_{train}$  e  $J_{CV}$  em função do parâmetro  $\lambda$ , podemos qualificar como se dei a melhor escolha:



É comum estabelecermos uma baseline para entender quais os biass e variâncias estão altas ou baixas. Um exemplo de baseline é a performance humana.



Se o aprendizado do algoritmo sofre de uma alta variância, consegue mais dados de treinamento provavelmente ruim.

Resumo de debug de aprendizado da classificação

Em geral, as estratégias que podemos considerar em cada caso abaixo são:

- Obter mais exemplos de treinamento  
Resolve a variância alta
- Tentar reduzir o conjunto de features, ou seja, eliminar features de grau menor  
Resolve a variância alta
- Tentar obter mais features  
Resolve o bias alto
- Tentar obter mais features polinomiais, ou seja,  $x_1^2, x_2^2, x_1 \cdot x_2$ , etc.  
Resolve o bias alto
- Tentar reduzir  $\lambda$   
Resolve o bias alto
- Tentar aumentar  $\lambda$   
Resolve a variância alta.

Loop interativo do desenvolvimento de ML

Escolher a arquitetura  
(modelo, dados, etc...)



Suponha que o modelo tenha classificado 100 e-mails de forma errada em um conjunto de cross-validation de 500 e-mails. Analisar os erros significa verificar manualmente o que cada e-mail com classificação errada tinha em seu conteúdo. Você pode fazer uma "estatística" sobre o que mais aparece nas características dos e-mails errados. Tudo isso servirá para formatação de dados no loop de desenvolvimento do modelo.

Obs: a análise de erros é mais útil no caso de termos problemas em que o set humano é capaz de fazer, como classificar e-mails ou imagens. Caso contrário, não serve muito trivial identificar padrões nos erros.

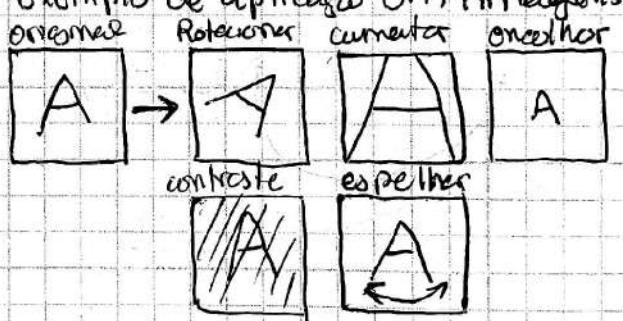
Adicionando mais dados

Conseguir mais dados é uma medida lenta e normalmente caro, por isso, o recomendado é obter vários tipos de dados mas focar nos parâmetros em que o modelo está errando com mais frequência.

Por exemplo, se o modelo erra com muita frequência em determinado parâmetro, obter mais dados que possuam este parâmetro pode ajudar.

- **Data augmentation:** técnica que nos permite aumentar o número de exemplos a partir do conjunto de dados de treinamento

Exemplo de aplicação em imagens:



Em geral, significa gerar mais dados através de distorções e variações das imagens originais / dados originais.

Uma aplicação em processamento de áudio seria montar outros áudios, que representam ruídos ao redor de voz de uma pessoa. Pode-se também distorcer o áudio para criar variações de timbre.

Aumentar ruídos e distorções ajudam a criar novos dados que ajudam o modelo a aceitar mais.

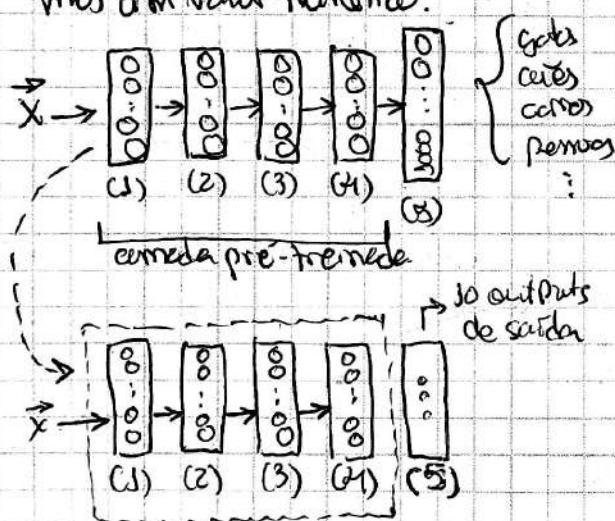
Isto nos leva ao ponto de considerar duas abordagens:

- Abordagem convencional centrada no modelo: foca em trabalhar no código, otimizando o modelo através das configurações.
- Abordagem centrada nos dados: foca em melhorar a qualidade dos dados e garantir uma performance melhor pelo melhoramento dos datasets.

A síntese de dados ou data augmentation se concentra na segunda abordagem.

Transfer learning.

Técnica usada para aproveitarmos uma rede pré-treinada (apenas ligadas) à nossa rede. Por exemplo, se temos uma rede neural capaz de classificar cães, gatos, pessoas, carros, etc... podemos usá-la para integrar a outra rede cujo output não é uma classificação com valores numéricos.



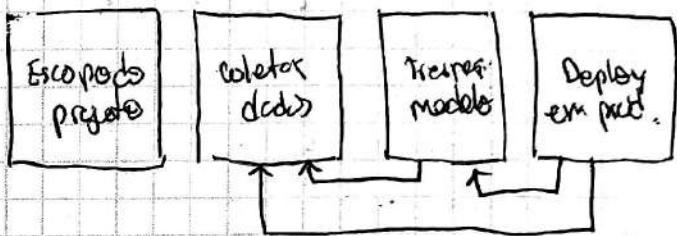
Termos duas opções mente certo: treinar apenas os caminhos de saída, que é melhor no caso de termos apenas alguns dados (dataset pequeno).

Outra opção seria treinar todos os parâmetros da rede, funcionando mais em datasets grandes. Ou seja, como rede pré-treinada, treinar todos os dados juntando com as redes pré-certas.

Chama-se supervised pretraining o treinamento de uma rede que será usada em outro modelo.

Chamou-se fine tuning a técnica de usar uma rede pré-treinada com uma ou mais camadas adicionais no final da rede que ela só será construída.

O ciclo completo de um projeto de ML



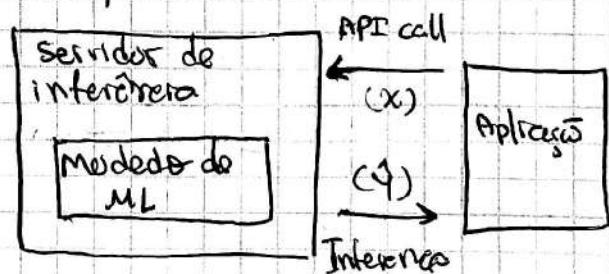
① O escopo considera o tipo de projeto, se é o domínio da visão computacional, ou processamento de áudio, ou classificação de imagens..

② coletar dados pode considerar a preparação dos dados para o treinamento da rede. Muitas vezes os dados já se encontram na base de dados do cliente e o que devemos fazer é separá-los.

③ O treinamento da rede pode gerar novos dados com técnicas de data augmentation e sintetização de dados. Também pode treinar a rede com dados vindos de deploy.

④ O deploy alimenta a base de dados e serve para treinar a rede com novos dados.

### Deploy



O deploy muitas vezes exige o trabalho de um ML devops ou MLops. Machine Learning Operations.

calculando precisão em problemas específicos

em alguns casos, podemos lidar com problemas que exigem uma precisão minuciosa. Considere uma doença rara que afeta 0,5% dos pacientes do dataset e nossa precisão na previsão é de 95%. Daí seja, a cada 100 pacientes, erros 1 previsão.

1% é menor do que 0,5%. e tem uma enorme relevância na previsão real. Bem como como esse, teremos uma taxa de falso.

classes atuais

		1	0
		Verdadeiros Positivos	Falso Positivo
Classe Predictor	1	15	5
	0	Falso Negativo 10	Verdadeiros Negativos 70

Temos os seguintes valores:

Precisão

- De todos os pacientes em que predizemos  $y=1$ , qual fração na verdade corresponde à doença verdadeiramente?

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\#\text{positivos preditos}} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falso Pos.}} = \frac{15}{15+5} = 0,75$$

Recall

- De todos os pacientes que possuem a doença rara, qual fração deles nós predizemos com sucesso?

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\#\text{positivos atuais}} = \frac{\text{Verdadeiros Positivos}}{\text{Verdadeiros Positivos} + \text{Falso Negativo}} = \frac{15}{15+10} = 0,6$$

O trading off entre precisão e recall

Regressão logística:  $0 < f_w^T b(x) < 1$

Prediz 1 se  $f_w^T b(x) \geq 0,5$  Threshold  
Prediz 0 se  $f_w^T b(x) < 0,5$

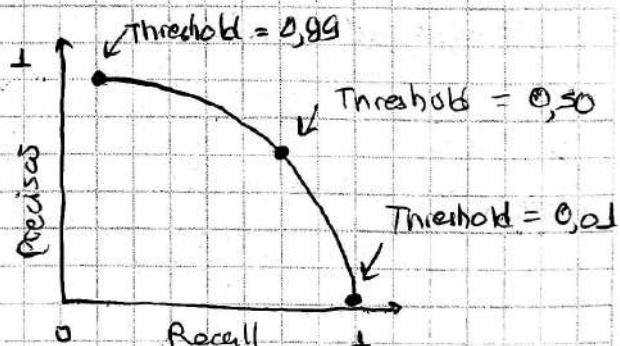
O que fazemos aqui é alterar os parâmetros de previsão. Como isso, o modelo pode prever 1 apenas quando tiver 99% de certeza ou 0 mínimo de dúvida.

- Se o threshold tende a 1, por exemplo 0,9, temos uma alta precisão e um baixo recall. Isso quer dizer que nossos positivos são mais frios e os resultados real mais pegajosos pacientes.

- Se o threshold tende a 0, por exemplo 0,1 temos uma baixa precisão e um alto recall, ou seja, encaramos mais os predizidos como positivos que mais pacientes sejam considerados.

$$\text{Precisão} = \frac{\text{Verdadeiros Positivos}}{\text{Total de positivos preditos}}$$

$$\text{Recall} = \frac{\text{Verdadeiros Positivos}}{\text{Total que efetivamente é positivo}}$$



Como podemos escolher um threshold adequado?

Trabalhamos com um conceito não chamado F1 score. Esse score ranknea os algoritmos dando a melhor opção de acordo com o score.

## O cálculo do F<sub>1</sub> score

	Precisão (P)	Recall (R)	F <sub>1</sub> score
Algoritmo 1	0,3	0,4	0,1144
Algoritmo 2	0,7	0,1	0,175
Algoritmo 3	0,02	3,0	0,0392

Analisamos o algoritmo com maior chance de ser bom nesse trading off através do F<sub>1</sub> score, calculado por:

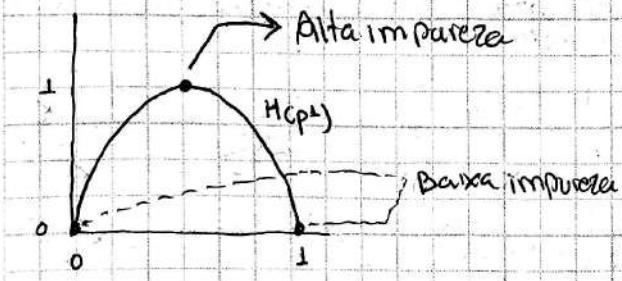
$$F_1 \text{ score} = \frac{\frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right)}{\frac{1}{2} \left( \frac{1}{P} + \frac{1}{R} \right)} = 2 \frac{P \cdot R}{P+R}$$

## Árvore de decisão (Decision Tree)

Estrutura em formato de árvore que descreve relações existentes entre as entradas e as saídas. É composta por nós, sendo o nó raiz (root node) o primeiro e os nós de decisão os demais nós.

## Entropy as a measure of impurity

Decidir como devemos construir a árvore envolve o conceito de "pureza", que significa a quantidade de elementos algo que são separados por uma característica. Por exemplo: em um conjunto de ameixas, a formatação da areia determina uma separação que resulta em um subconjunto contendo 80% de gatos. A entropia é uma relação que podemos usar como medida de pureza.



A entropia segue a seguinte relação

$$\begin{aligned} H(p_1) &= -p_1 \log_2(p_1) - p_0 \log_2(p_0) \\ &= -p_1 \log_2(p_1) - (1-p_1) \log_2(1-p_1) \end{aligned}$$

Fazemos isto pois  $p_0 = 1 - p_1$

## Information Gain

Para escolher qual feature usar em determinada nó podemos recorrer à relação de ganho de informação. Basicamente, a feature que apresenta o maior ganho de informação deve ser usada no nó.

## Information Gain =

$$= H(p_1^{\text{root}}) - (w \cdot H(p_1^{\text{left}}) + w \cdot H(p_1^{\text{right}}))$$

OBS:  $p_1^{\text{root}}$  é a pureza do nó, w se refere às proporções de elementos em relações à ramificação (porcentagem de elementos que são alocações para determinadas folhas.)

# CURSO Machine Learning

## Unsupervised learning

### Clustering

Uma técnica que é utilizada para agrupar os dados e então entendê-los sobre uma determinada óptica.

#### K-means algorithm

Supomos que existam dados de duas dimensões. Geramos dois centroides, ou seja, pontos, em posições aleatórias, e em seguida dividimos os dados por aproximação em relação a estes pontos. Após isso, movemos os centroides para o ponto médio do conjunto de dados pertencente a eles. Repetimos estes passos até formar grupos definidos.

- Randomicamente, inicia K centroides de clusters  $\mu_1, \mu_2, \dots, \mu_K$

#### • Repita:

- # Relacione os pontos aos centroides para  $i = 1$  até  $m$   
 $c^{(i)}$  é definido como o índice do centroide mais próximo ao ponto

#### # Move os centroides dos clusters

Para  $k = 1$  até  $K$

$\mu_k$  é definido pelo ponto médio entre os pontos pertencentes ao centroide do cluster  $k$ .

}

#### Cálculo da posição média

$$\mu_k = \frac{1}{n} [x^{(1)} + x^{(2)} + \dots + x^{(n)}]$$

O algoritmo é simples, resumidamente:

1 - Ligue os pontos aos centroides mais próximos

2 - Move os centroides até a posição média das pontos relacionados a eles

3 - Volte ao passo 1 até que o algoritmo comunique.

cost function para K-means

$$J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K) = \frac{1}{m} \sum_{i=1}^m \|x^{(i)} - \mu_{c^{(i)}}\|^2$$

distância

Em que:

$c^{(i)}$ : índice do cluster ( $1, 2, \dots, K$ ) para o qual cada  $x^{(i)}$  está atualmente relacionado

$\mu_k$ : centroide do cluster  $K$

$\mu_{c^{(i)}}$ : centroide do cluster de cada  $x^{(i)}$

O objetivo é semelhante aos objetivos das técnicas de Machine learning: tentar minimizar o erro, ou seja, reduzir o valor de  $J$ .

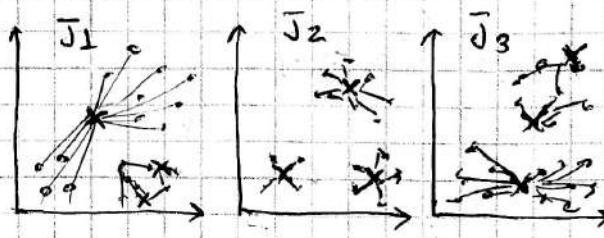
Essa redução do erro ocorre naturalmente no algoritmo k-means uma vez que sempre buscamos por reduzir a distância entre os pontos e os centroides relativos a esses pontos.

#### Iniciando K-means

Sabendo que devemos escolher  $K < m$  para garantir que os clusters tenham pontos associados

Uma das formas de escolhermos os centroides é aleatoriamente escolhermos  $K$  exemplos de treinamento e atribuirmos  $\mu_1, \dots, \mu_K$  iguais a esses  $K$  exemplos.

O problema de seguir essa estratégia sem algum controle é que, dependendo do ponto de inicialização, podemos ter clusters que convergiram para mínimos locais. Por isso, utilizaremos a função de custo para avaliarmos se os clusters são de fato os mínimos.



Melhor escolha  $J$

$J_2 \rightarrow$  menor  $J$

## Inicialização aleatória

Devido à necessidade de avaliarmos o nível de distorção para definirmos um mínimo, seguimos o algoritmo:

Pode variar de 50 - 1000

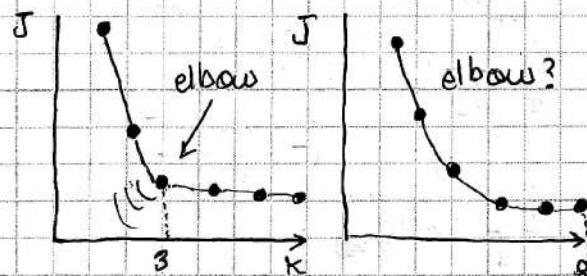
```

for i = 1 to 100 {
    K-random examples →
    Initialize K-means aleatoriamente
    Execute o K-means, pegue  $c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K$ 
    calcule a função de custo (distorção)
     $J(c^{(1)}, \dots, c^{(m)}, \mu_1, \dots, \mu_K)$ 
}
  
```

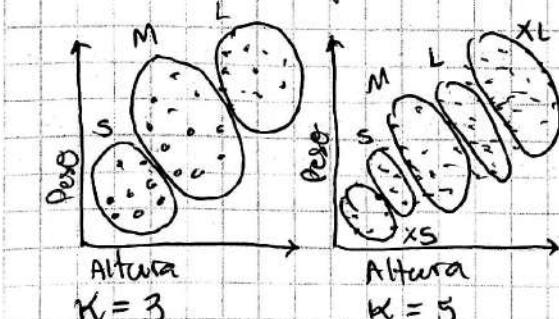
Escolha o conjunto de clusters com resultados que indiquem a menor distorção, ou seja, o menor  $J$ .

## Escolhendo o número de clusters

Neste caso, não é muito indicado observar unicamente a distorção  $J$ , pois o resultado pode ser combigoso. Por isso, técnicas como "elbow" ou "cotovelo" podem não funcionar muito bem.

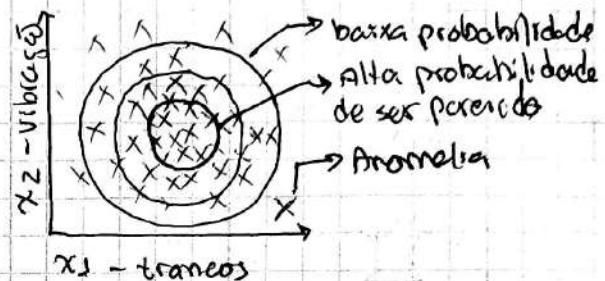


Poderemos nos guiar pelo contexto. Por exemplo, agrupando pessoas pela altura e pelo peso em grupos de "tamanhos de família". Fazemos um trade-off entre granularidade.



## Deteção de anomalias

Técnica utilizada para compararmos o que próximo está um dado de um determinado conjunto de dados com o objetivo de identificarmos anomalias.



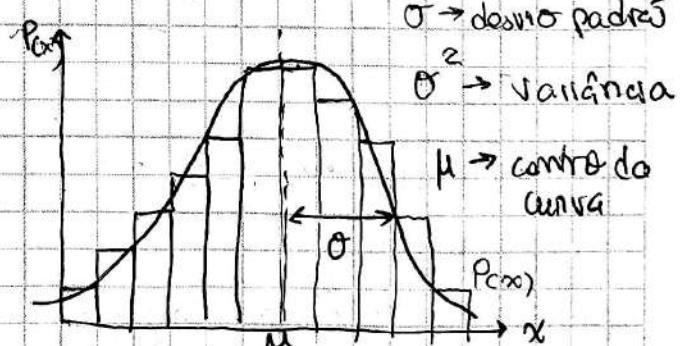
Dizemos que  $p(x_{\text{test}}) < \epsilon$ , ou seja, a probabilidade deve ser menor do que  $\epsilon$ , com número normalmente pequeno, para ser uma anomalia.

De modo contrário,  $p(x_{\text{test}}) \geq \epsilon$  indica um dado OK, normal.

Deteção de anomalias pode ser usada na detecção de fraudes em sistemas, monitoramento de data centers e em processos de manufatura, com a detecção de falhas durante os processos industriais.

## Distribuição Gaussiana ou Normal

Um tipo de distribuição de probabilidades em forma de sino



De tal modo que  $P(x)$  é dada por:

$$P(x) = \frac{1}{\sqrt{2\pi}\sigma} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$$

Outro ponto importante é que a curva da distribuição de probabilidades é tal que sua área é igual a 1.

$\mu$  e  $\sigma^2$  são dados por expressões

seja o dataset  $\{x^{(1)}, x^{(2)}, \dots, x^{(m)}\}$

$$\mu = \frac{1}{m} \sum_{i=1}^m x^{(i)}$$

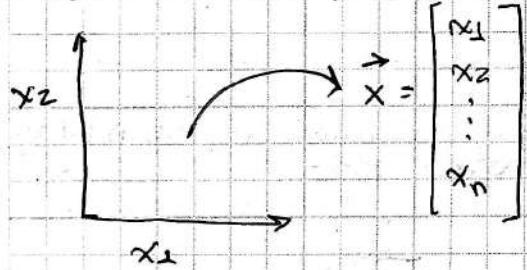
$$\sigma^2 = \frac{1}{m} \sum_{i=1}^m (x^{(i)} - \mu)^2$$

$x$  é um número, representa uma feature.

Density estimation

Seja um conjunto de treinamento tal que  $\{\vec{x}^{(1)}, \vec{x}^{(2)}, \dots, \vec{x}^{(m)}\}$ , em que cada exemplo  $\vec{x}^{(i)}$  tem  $n$  features

Considerando que as features são variáveis independentes, podemos construir uma expressão para a probabilidade envolvendo todas essas features



$$p(\vec{x}) = p(x_1; \mu_1, \sigma_1^2) \cdot p(x_2; \mu_2, \sigma_2^2) \cdot \dots$$

$$\cdot p(x_n; \mu_n, \sigma_n^2)$$

Poderia interpretar  $\vec{x}$  como um conjunto de features e  $p(\vec{x})$  as probabilidades:

$$p(x_1 = \text{alta temperatura}) = 1/30$$

$$p(x_2 = \text{alta vibração}) = 1/20$$

$$p(x_1, x_2) = p(x_1) \cdot p(x_2) = \frac{1}{200}$$

em outras palavras, a probabilidade é dada pelo produto das probabilidades de cada feature (dimensão).

$$p(\vec{x}) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2)$$

O algoritmo de detecção de anomalias

① Escolher  $n$  features  $x_i$  que podem indicar anomalias

② Ajustar os parâmetros  $\mu_1, \dots, \mu_n, \sigma_1^2, \dots, \sigma_n^2$

Sendo:

$$\mu_j = \frac{1}{m} \sum_{i=1}^m x_j^{(i)} \quad \sigma_j^2 = \frac{1}{m} \sum_{i=1}^m (x_j^{(i)} - \mu_j)^2$$

Fórmula vetorializada:

$$\vec{\mu} = \frac{1}{m} \sum_{i=1}^m \vec{x}^{(i)} \quad \vec{\mu} = \begin{bmatrix} \mu_1 \\ \mu_2 \\ \vdots \\ \mu_n \end{bmatrix}$$

③ Dado um novo exemplo  $x$ , computar  $p(x)$ :

$$p(x) = \prod_{j=1}^n p(x_j; \mu_j, \sigma_j^2) = \\ = \prod_{j=1}^n \frac{1}{\sqrt{2\pi\sigma_j^2}} \cdot e^{-\frac{(x_j - \mu_j)^2}{2\sigma_j^2}}$$

A anomalia é detectada se  $p(x) < \epsilon$

Desenvolvendo e criando um sistema de detecção de anomalias.

Normalmente, temos poucos dados que indicam anomalias. Em um dataset de 10 000 linhas de dados podemos ter 20 linhas que representam de fato anomalias. Por isso, podemos desenvolver alguns testes de validação.

# Curso Machine Learning

Se temos 10 000 dados normais e 20 dados de anomalias, podemos:

Separar os dados normais,  $y = 0$  e dados anormais,  $y = 1$ .

Separar em conjuntos:

- Conjunto de treino: 6 000 exemplares bons. ( $y = 0$ )
- Conjunto de validação cruzada com 2 000 exemplares bons e 10 apresentando anomalias ( $y = 1$ )
- Conjunto de teste contendo 2 000 exemplares normais e 10 anormais

Com essas partes em mãos, treinamos o algoritmo com o conjunto de treinamento, testamos sua performance com o conjunto de teste e compararmos com a validação cruzada.

- Ajuste  $\epsilon$  para melhorar o resultado com o conjunto de validação cruzada, isso pode aumentar ou diminuir a sensibilidade
- Adicionar ou remover features e usar o conjunto de validação cruzada para ver se o sistema possui uma resposta mais precisa.

OBS: No caso de se ter poucos dados de anomalia, por exemplo, 6 000 bons exemplares de treino, 4 000 bons exemplares de teste e apenas 2 exemplares de anomalia:

- Exclua o teste e use 6 000 exemplares de ( $y = 0$ ) com um conjunto de validação cruzada com 4 000 exemplares normais e 2 anormais.
- Ajuste  $\epsilon$  e  $X_j$

Atenção: Há grande chance de acontecer um overfitting.

Outra abordagem possível seria prever valores de saída e obter métricas:

- ① Treine um modelo,  $p(x)$  com o conjunto de teste  $x^{(1)}, x^{(2)}, \dots, x^{(n)}$
- ② No conjunto de teste / validação cruzada, use os exemplos para prever

$$y = \begin{cases} 1 & \text{se } p(x) < \epsilon \text{ (anomalia)} \\ 0 & \text{se } p(x) \geq \epsilon \text{ (normal)} \end{cases}$$

Análise:

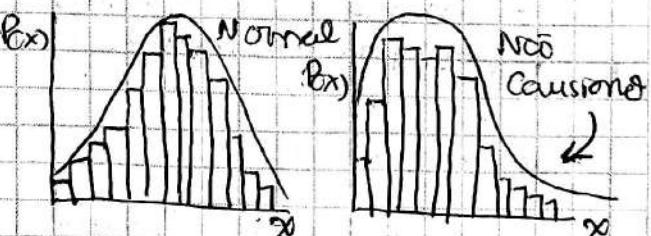
- Verdadeiros positivos, falsos positivos, verdadeiros negativos, falsos negativos
- Precisão / Recall
- F1-score

Com isso, ajuste  $\epsilon$

Escolhendo features

A escolha das features em sistemas de detecção de anomalias se torna mais importante do que em outros algoritmos. A seguir, algumas dicas para melhorar resultados de detecção de anomalias.

Features não-Gaussianas: quando o histograma dos dados da feature não corresponde a um formato de sino, ou é uma Gaussiana.



Modifique  $x_j$  até que essa feature assuma o formato gaussiano. Exemplo:

$$\begin{aligned}x_1 &\rightarrow \log(x_1) \\x_2 &\rightarrow \log(x_2 + c) \\x_3 &\rightarrow -\sqrt{x_3} = x_3^{-1/2}\end{aligned}$$

$$x_4 \rightarrow x_4^{1/3}$$

Análise de Erros para detecção de Anomalias.

Deseja-se o seguinte critério:

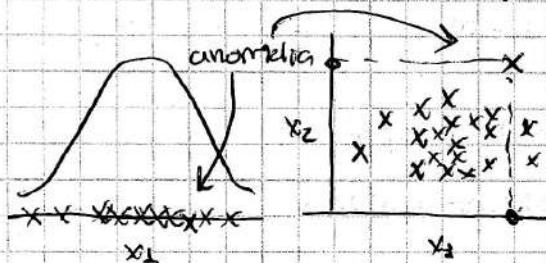
p(x) grande para exemplares normais

p(x) pequeno para exemplares anômalos

- Um problema comum:

p(x) é parecido para exemplares normais e anômalos (grande porcentagem)

Para resolver esse problema, podemos acrescentar features que batam uma nova dimensão ao sistema.



Podemos também transformar features, dividindo uma pela outra, dividindo e elevando ao quadrado, etc...

Decidimos as escolher de features baseando-nos no critério que desejamos:

p(x) grande para exemplares normais  
p(x) pequeno para anômalos.

sistemas de Recomendações

O objetivo de um sistema de recomendação é prever o que bom seriam determinados ítem para certo usuário de acordo com o que sabemos de sua experiência com outros ítems.

Sejam os dados:

Filme	Usuário 1	Usuário 2	...	Comprador	Custo
1	3	5		0,9	0
2	5	?		1,0	0,01
3	2	4		0,99	0

Indica se o filme tem características

Notas

XL

X2

Filme | Usuário 1 Usuário 2 ... Comprador Custo

1	3	5		0,9	0
2	5	?		1,0	0,01
3	2	4		0,99	0

Em outras palavras, dado um filme, os usuários podem dar notas de 0 à 5 ou podem não ter visto o filme, sendo informados com (?). Além disso, temos, de 0 à 1, uma restrição sobre se o filme passou certos critérios de filmes da romântica e etc.

- $r(i, j) = 1$  se o usuário classificou o filme  $i$ , ou 0 de modo contrário
- $y^{(i,j)}$  classificação dada pelo usuário  $j$  sobre o filme  $i$ , se definida
- $w^{(j)}, b^{(j)}$  parâmetros do usuário  $j$
- $x^{(i)}$  vetor de features para o filme  $i$
- $w^{(j)} \cdot x^{(i)} + b^{(j)}$  é a predição da classificação para o usuário  $j$  e o filme  $i$
- $m^{(j)}$  é o número de filmes classificados pelo usuário  $j$

Função de custo ( $J$ )

Para fincar os parâmetros  $w^{(j)}$  e  $b^{(j)}$  para o usuário  $j$ :

$$J(w^{(j)}, b^{(j)}) = \frac{1}{2} \sum (w^{(j)} \cdot x^{(i)} + b^{(j)} - y^{(i,j)})^2 +$$

$i: r(i,j) = 1$

$$+ \frac{1}{2} \sum_{k=1}^m (w_k^{(j)})^2$$

Regularização