📙

Ciclo 06 - Outros sistemas de aprendizado

🤖 Fundamentos Machine Learning

Logistic Regression

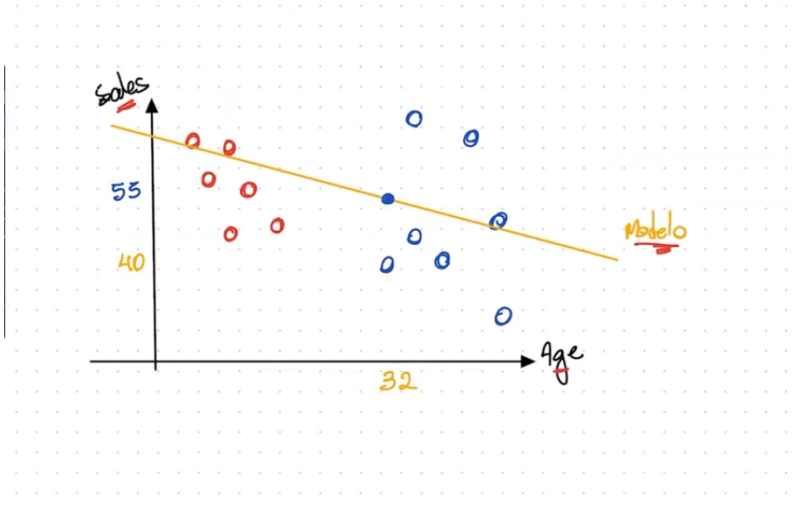
O algoritmo de Regressão Logística é uma combinação de modelos da família das regressões com uma função de ativação, chamada função Sigmoid.

Essa união permite o algoritmo de Regressão conseguir classificar exemplos de um conjunto de dados entre as classes A ou B.

> 💡 Apesar de possuir “Regressão” no nome, o algoritmo de Regressão Logística na verdade é um algoritmo de classificação. Isso ocorre porque nesse tipo de algoritmo, não estamos interessado em utilizar as features de cada observação para estimar um target real, como preço, altura etc, mas sim em estimar a probabilidade que cada observação tem de pertencer a uma classe. Nesse sentido, estamos usando um conjunto de features X para estimar uma variável numérica $\hat{y}$ (probabilidade) a qual, por sua vez, será utilizada para segmentar o espaço de features em regiões de classificação baseado em algum threshold (th), e então determinar o valor da variável categórica y (target).

Imagine que tenhamos um conjunto de dados com as features: (“Sales” e “Age”). Quando fazemos um problema de regressão estamos interessados em encontrar a uma reta que descreva a relação entre as variáveis, de modo que seja possível determinar o valor de um “Target” a partir das variáveis do conjunto de dados.

> 💡 A reta ajustada em uma regressão é chamada de modelo, em outras palavras, a reta é uma representação do fenômeno.



Quando utilizamos o modelo para fazer uma predição em uma regressão, nós identificamos qual o valor que um determinado conjunto de inputs (registro, nesse caso o registro contém apenas `"Age"`) nos retorna como output (`“Sales”`). Nesse tipo de problema, nosso interesse é conhecer como as mudanças nas nossas features afetam o valor do nosso target.

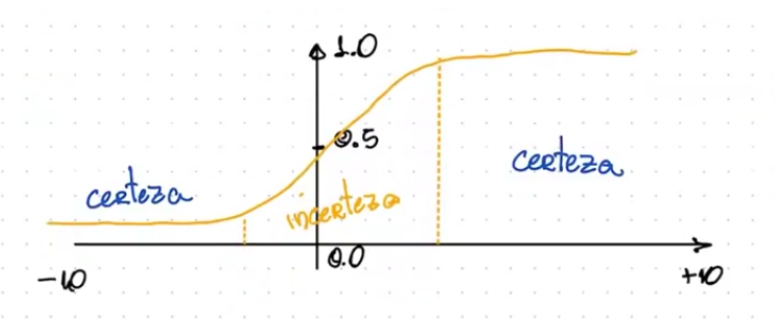
$$\text{Sales} = \theta_0 + \theta_1 \text{Age}$$

Em problemas de Classificação estamos interessado em dividir o espaço de features em regiões de classificação. Todos os pontos que ficarem à direita da reta classificadora será classificado como classe azul e todo ponto à esquerda terá classe vermelha.

> 💡 Nosso problema agora é construir uma reta que divide o espaço em duas regiões de classificação e não mais utilizar a própria reta para estimar valores.

Anteriormente, na regressão linear vista acima, nosso modelo previa o valor de `“Sales”` a partir da variável `“Age”`. Agora queremos definir a qual classe cada registro representado pelo par $(\text{Age}, \text{Sales})$ pertence. Digamos que nesse exemplo

Curva de Regressão Logística



> 💡 Quanto mais afastado da da região central, maior será a certeza da classificação feita pelo modelo.

as classes indicam se o vendedor receberá ou não uma bonificação ao final do ano (Classe 1 → Sim) e (Classe 0 → Não).

O modelo deve ser uma função que mapeia qualquer ponto do plano (x_1, x_2) no intervalo $[0, 1]$ (No caso geral: $f : \mathbb{R}^n \to [0, 1]$), ou seja, para uma determinada observação $\vec{x}$, o valor da função $f(\vec{x})$, representará uma probabilidade, a qual definimos como sendo a probabilidade de pertencer a classe positiva. Em regressão logística a função, chamada de sigmóide, possui a forma:

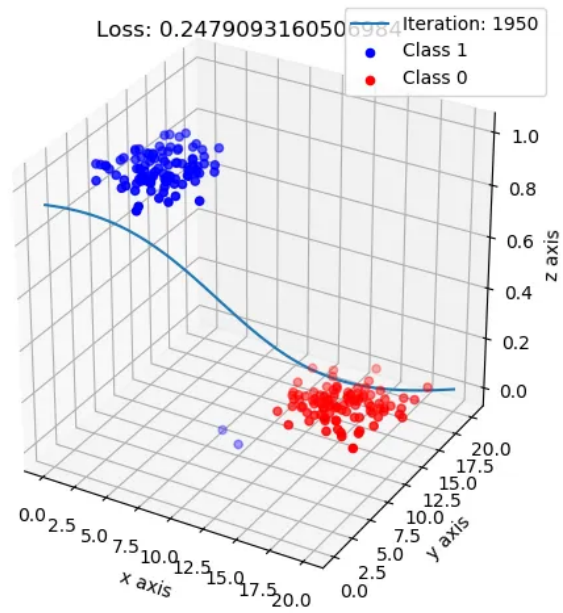
$$\hat{y} = f(x_1, x_2, ..., x_n) = \frac{1}{1 + e^{-\theta}} = \frac{1}{1 + e^{-(\theta_0 + \sum_{i=1}^{n} \theta_i x_i)}}$$

Onde os parâmetros θ_1, θ_1,..., θ_n, são ajustados pelo modelo de modo a minimizar a função custo, a qual será discutida na próxima seção. Como discutido acima, podemos interpretar a função acima como sendo a probabilidade de classificar um registro com a classe positiva dada uma função linear θ, ou seja, se para uma observação identificada por $\vec{x} = (x_1, x_2, ..., x_n)$ obtermos $f(\vec{x}) = 0.18$ concluímos que tal observação possui 18% de chances de pertencer a classe positiva, ou seja, ela terá 82% de pertencer a classe negativa. As probabilidades calculadas dependem do valor de θ:

$$\theta = \theta(x_1, x_2, ..., x_n) = \theta_0 + \sum_{i=1}^{n} \theta_i x_i \quad = \quad \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n$$

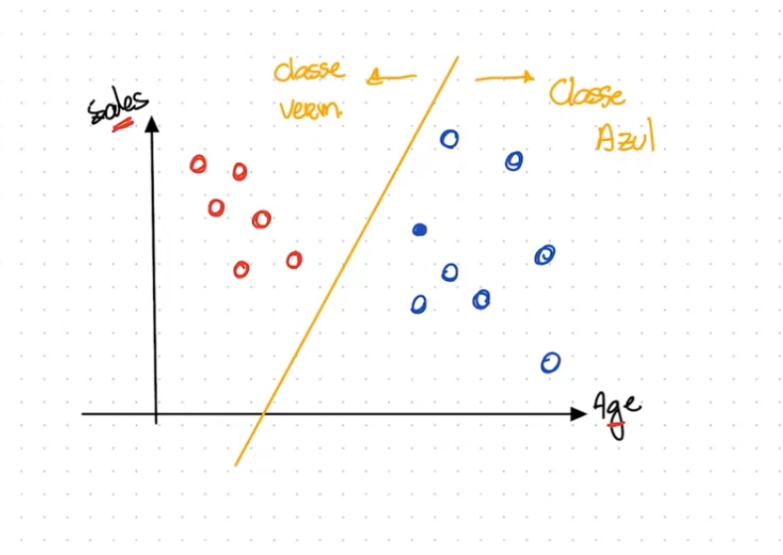
Após treinarmos o modelo e determinarmos os valores dos parâmetros de ajuste, podemos dividir o espaço de features entre as regiões de classificação. Por padrão, o threshold (probabilidade mínima atribuída pelo modelo necessária para classificar um registro com a classe positiva) é 0.5, ou seja, se o valor predito pelo modelo $\hat{y}$ for maior que 0.5 classificamos como classe 1 e classe 0 caso contrário.

- $\hat{y} > 0.5 \quad \Rightarrow$ classe 1
- $\hat{y} < 0.5 \quad \Rightarrow$ classe 0



Mas antes vamos descobrir para quais valores de x_1 e x_2 obtemos $\hat{y} = 0.5$:

$$\hat{y} = \frac{1}{2} = \frac{1}{1 + e^{-\theta}} \quad \Rightarrow \quad \theta = 0$$



Os pontos $(x_1, x_2, ..., x_n)$ para os quais $\theta = 0$ são aqueles que possuem a probabilidade de 50% de pertencerem classe 1, para obtermos os pontos que satisfazem tal condição precisamos resolver a equação:

$$\theta \quad = \quad \theta_0 + \theta_1 x_1 + \theta_2 x_2 + ... + \theta_n x_n = 0$$

Nosso problema de classificação, para encontrarmos a reta de separação precisamos resolver a seguinte equação:

$$\theta_0 + \theta_1 \text{Age} + \theta_2 \text{Sales} = 0$$

A reta amarela no gráfico ao lado é composta pelos pares $(\text{Age}, \text{Sales})$ que satisfazem a equação:

$$\text{Sales} = -\left(\frac{\theta_1}{\theta_2}\right) \text{Age} - \frac{\theta_0}{\theta_2}$$

Logistic Regression: Maximum Likelihood Estimation & Gradient Descent

In this blog, we will be unlocking the Power of Logistic Regression by mastering Maximum Likelihood and Gradient Descent which will also…

https://medium.com/@ashisharora2204/logistic-regression-maximum-likelihood-estimation-gradient-descent-a7962a452332

O processo de treinamento da Regressão Logística

Todo algoritmo de machine learning tem como objetivo de minimizar uma função custo, a qual é definida de acordo com o algoritmo em questão; de maneira simplificada essa função representa o erro entre a predição feita pelo algoritmo e valor real que a variável possui. No processo de aprendizagem vamos ajustando os parâmetros que compõem o algoritmo de modo a diminuir o erro a cada iteração de treinamento. Cada algoritmo possui uma função de custo diferente, uma vez que cada um possui um mecanismo de aprendizado diferente e portanto é necessário que sejam utilizados diferentes maneiras de penalizar o erro da predição de um ponto.

Nosso conjunto de dados X e nossa variável target y podem ser representados pela matriz e vetor abaixo respectivamente:

$$X = \begin{bmatrix} x_{11} & x_{12} & ... & x_{1n} \\ \vdots & \vdots & & \vdots \\ x_{j1} & x_{j2} & ... & x_{jn} \\ \vdots & \vdots & \ddots & \vdots \\ x_{m1} & x_{m2} & ... & x_{mn} \end{bmatrix}_{(m \times n)} \qquad e \qquad y = \begin{bmatrix} y_1 \\ \vdots \\ y_j \\ \vdots \\ y_m \end{bmatrix}_{(m \times 1)}$$

Em regressão logística o variável target é discreta podendo ser 0 ou 1 em problemas de classificação binária ($y \in \{0, 1\}$), no entanto o valor da nossa predição ($\hat{y}$) é um valor contínuo que varia no intervalo $[0, 1]$ sendo calculada através de $\hat{y} = f_{\vec{\theta}}(\vec{x})$ em que $f_{\vec{\theta}}$ representa a função sigmoide definida pelo conjunto de parâmetros $\vec{\theta} = (\theta_0, \theta_1, \ldots, \theta_n)$ e $\vec{x}$ é o vetor de features que representa um registro (linha do conjunto de dados).

Como o vetor $\vec{\theta}$ possui um índice a mais (começa pelo índice 0), portanto vamos adicionar uma coluna extra à X com todos inputs iguais à 1, isso ficará mais claro a diante (deixará a notação mais simples), de modo que a nossa matriz será representada por:

$$X = \begin{bmatrix} 1 & x_{11} & x_{12} & ... & x_{1n} \\ \vdots & \vdots & \vdots & & \vdots \\ 1 & x_{j1} & x_{j2} & ... & x_{jn} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & x_{m1} & x_{m2} & ... & x_{mn} \end{bmatrix} \qquad x^{(j)} = \begin{bmatrix} 1 \\ x_{j1} \\ x_{j2} \\ \vdots \\ x_{jn} \end{bmatrix} \qquad e \qquad \vec{\theta} = \begin{bmatrix} \theta_0 \\ \theta_1 \\ \theta_2 \\ \vdots \\ \theta_n \end{bmatrix}$$

Através da notação de vetores podemos escrever a somatória de maneira mais compacta:

$$\vec{\theta}^{\,T} x^{(j)} \quad = \quad \begin{bmatrix} \theta_0 & \theta_1 & \theta_2 & \ldots & \theta_n \end{bmatrix} \begin{bmatrix} 1 \\ x_{j1} \\ x_{j2} \\ \vdots \\ x_{jn} \end{bmatrix} \quad = \quad \theta_0 + \theta_1 x_{j1} + \theta_2 x_{j2} + ... + \theta_n x_{jn} \quad = \quad \theta_0 + \sum_{i=1}^{n} \theta_i x_{ji}$$

De modo que, a predição uma determinada observação j (linha do dataset) é feita através da equação:

$$\hat{y}^{(j)} = f_{\vec{\theta}}(x^{(j)}) = \frac{1}{1 + e^{-(\theta_0 + \sum_{i=1}^{n} \theta_i x_{ji})}} = \frac{1}{1 + e^{-\vec{\theta}^{\,T} x^{(j)}}}$$

No processo de treinamento queremos que a diferença do valor predito da observação j ($\hat{y}^{(j)} = f_{\vec{\theta}}(x^{(j)})$) e a classe real ($y^{(j)}$) seja a menor possível quando levamos em contas todas as possíveis observações j caracterizadas pelos valores das features em $x^{(j)}$, ou seja, queremos minimizar soma de todos os erros, também conhecido como `Custo`, entre as predições e os valores reais.

$$J(\vec{\theta}) = \frac{1}{m} \sum_{j=1}^{m} \text{Cost}(\ f_{\vec{\theta}}(x^{(j)}), y^{(j)}\ )$$

A função $Cost()$ tem que ser definida de modo que quando a classe da observação j for 0, ou seja, $y^{(j)} = 0$ e a predição for próxima de 0 ($\hat{y}^{(j)} \approx 0$), seu valor deve ser próximo de zero ($Cost \approx 0$) uma vez que previsão e o valor real tem valores próximos. Também devemos ter $Cost \approx 0$ no caso em que $y^{(j)} = 1$ e $\hat{y}^{(j)} \approx 1$, pois novamente tanto o valor predito quanto o real possui valores similares.

No entanto, a função $Cost$ deverá assumir um valor alto quando o modelo prever uma classe com certa certeza e na verdade a observação pertencer a classe oposta, sendo esse valor maior o quanto maior for a discordância entre predição e valor real, nesse cenário devemos aplicar uma grande punição, de modo que o algoritmo evite combinações dos parâmetros do vetor $\vec{\theta}$ que gere tal situação.

Portanto, estamos a procura de uma função que possua as seguintes características:

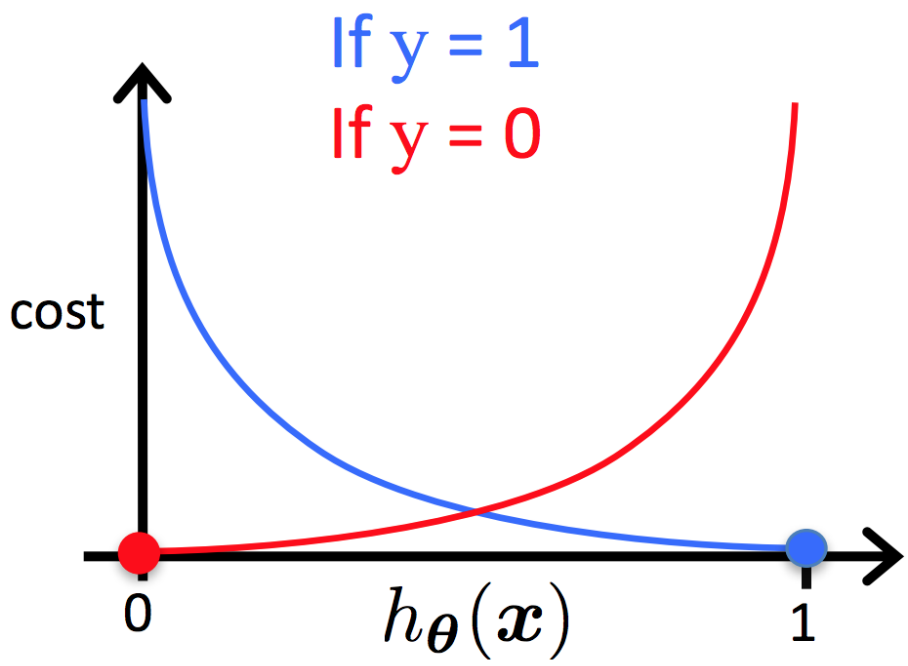
- $\text{Cost} \approx 0$
  - $y^{(j)} = 0$ e $\hat{y}^{(j)} \approx 0$
- $\text{Cost} \to \infty$
  - $y^{(j)} = 1$ e $\hat{y}^{(j)} \approx 0$

- $y^{(j)} = 1$ e $\hat{y}^{(j)} \approx 1$
- $y^{(j)} = 0$ e $\hat{y}^{(j)} \approx 1$

A função que satisfaz as condições acima e portanto é utilizada para ponderar o “Erro” cometido pelo modelo tem a seguinte forma:
(lembrando que $\hat{y}^{(j)} = f_{\vec{\theta}}(x^{(j)})$)

$$\text{Cost}(\ \hat{y}^{(j)}, y^{(j)}\ ) = \begin{cases} -\log(\ \hat{y}^{(j)}\ ) & \text{se} \quad y^{(i)} = 1 \\ -\log(\ 1 - \hat{y}^{(j)}\ ) & \text{se} \quad y^{(i)} = 0 \end{cases}$$

$$\text{Cost}(\ f_{\vec{\theta}}(x^{(j)}), y^{(j)}\ ) = \begin{cases} -\log(\ f_{\vec{\theta}}(x^{(j)})\ ) & se \quad y^{(i)} = 1 \\ -\log(\ 1 - f_{\vec{\theta}}(x^{(j)})\ ) & se \quad y^{(i)} = 0 \end{cases}$$



De modo geral, essa função pode ser reescrita da seguinte forma:

$$\text{Cost}(\ f_{\vec{\theta}}(x^{(j)}), y^{(j)}\ ) = -y^{(j)}\ \log(\ f_{\vec{\theta}}(x^{(j)})\ )\ -\ (1 - y^{(j)})\ \log(\ 1 - f_{\vec{\theta}}(x^{(j)})\ )$$

Ao levarmos em conta todas as m observações do conjunto de dados obtemos a função de custo da regressão logística:

$$J(\vec{\theta}) = \frac{1}{m} \sum_{j=1}^{m} \text{Cost}(\ f_{\vec{\theta}}(x^{(j)}), y^{(j)}\ )$$

$$J(\vec{\theta}) = -\frac{1}{m} \sum_{j=1}^{m} \left[y^{(j)}\ \log(\ f_{\vec{\theta}}(x^{(j)})\ )\ +\ (1 - y^{(j)})\ \log(\ 1 - f_{\vec{\theta}}(x^{(j)})\ ) \right]$$

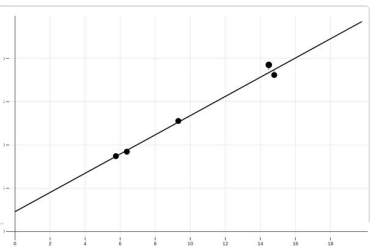
Para encontrarmos o melhor ajuste da regressão logística, temos que determinar quais são os valores dos parâmetros $\theta_1, \theta_2, ..., \theta_n$ que minimizam a função $J(\vec{\theta})$ em relação as observações contidas no conjunto dataset X.

Algoritmos de otimização

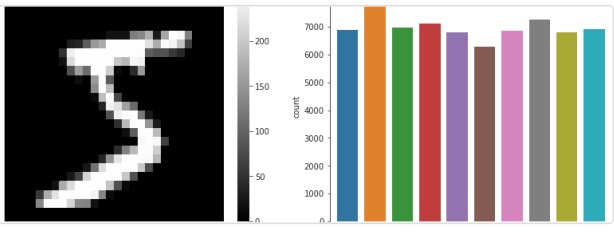
1. Para conjuntos de dados pequenos, `liblinear` é uma boa escolha, enquanto `sag` e `saga` são mais rápidos para grandes conjuntos de dados;
2. Para problemas de classificação multi-classe, apenas `newton-cg`, `sag`, `saga` e `lbfgs` lidam com perda multinomial;
3. `liblinear` é limitado a esquemas de um-contra-todos;
4. `newtow-cholesky` é uma boa escolha para `n_amostras >> n_caracteristicas`, especialmente com recursos categóricos como one-hot com categorias raras. Observe que ele é limitado à classificação binária e à redução um-contra-todos para classificação multi-classe. Esteja ciente de que o uso de memória desse solucionador tem uma dependência quadrática em relação a `n_caracteristicas`, pois ele explicitamente calcula a matriz Hessiana.

The cost function in logistic regression
Preparing the logistic regression algorithm for the actual implementation.
https://www.internalpointers.com/post/cost-function-logistic-regression

Interactive Visualization of Linear Regression
Improve your intuition about the behaviour of linear regression! Here are the interactions that are possible: Click anywhere on the plot to insert a new
https://observablehq.com/@yizhe-ang/interactive-visualization-of-linear-regression



Logistic Regression with Gradient Descent and Regularization: Binary & Multi-class Classification
Learn how to implement logistic regression with gradient descent optimization from scratch.
https://medium.com/@msayef/logistic-regression-with-gradient-descent-and-regularization-binary-multi-class-classification-cc25ed63f655



Regularização em Regressões

Os algoritmos paramétricos, aqueles que possuem uma formula definida (regressão linear, logística e etc…), são mais fáceis de overfittar o modelo, pois os parâmetros ajustados durante o treinamento podem ficar “fora de controle”, podemos pensar que esses parâmetros modificam a maleabilidade da

curva ajustada e em alguns casos, pode ocorrer da curva se tornar tão maleável que ela praticamente decora o conjunto de dados de treinamento. Uma maneira de evitar tal comportamento é penalizar certas “abordagens” de ajuste, de modo que, a curva ajustada possua um melhor poder de generalização sem que haja uma perda muito grande performance de aprendizado, essas penalizações são chamadas de regularizações.

> 💡 As regularizações criam uma penalidade extra à função de custo, de forma que conseguimos evitar combinações de parâmetros que overfittem o modelo.

As 3 principais regularizações em problemas de regressão são:

1. A regularização L1 - `Lasso`
2. A regularização L2 - `Ridge`
3. A regularização L1 e L2 - `Elastic Net`

A regularização L1 - `Lasso`

A regularização L1 adiciona uma penalidade à função custo para evitar que os parâmetros do modelo se tornem muito grandes.

> 💡 A adição da penalidade L1, **incentiva o modelo a utilizar apenas os parâmetros mais importantes**, zerando os parâmetros menos relevantes.

Função de Custo com a Regularização L1

$$J(\vec{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\ \log(\ f_{\vec{\theta}}(x^{(j)})\ )\ +\ (1-y^{(j)})\ \log(\ 1-f_{\vec{\theta}}(x^{(j)})\ )\right] + \lambda$$

O parâmetro λ controla a força da regularização

> 💡 Como a **regularização L1** zera muitos dos parâmetros de ajuste do modelo, ela é bastante utilizada como uma **ferramenta de seleção de features**.

▼ **Quando usar a regularização L1?**

> *Quando não houver uma seleção de features antes do treinamento do algoritmo.*

A regularização L1 é particularmente útil quando o conjunto de dados de treinamento tem muitas características, algumas das quais podem ser irrelevantes ou redundantes. Ela pode ajudar a selecionar as características mais relevantes para a predição, eliminando as características irrelevantes.

A regularização L2 - `Ridge`

A penalidade L2 é proporcional ao quadrado da magnitude dos parâmetros, incentivando o modelo a ter parâmetros menores.

> 💡 A penalidade L2 **incentiva o modelo a ter parâmetros menores**, pois nesse caso a função de custo $J(\vec{\theta})$ experiencia um aumento considerável caso os parâmetros possuam valores altos devido a uma dependência quadrática da função $J(\vec{\theta})$ em relação os parâmetros $\theta_i(\ \sum(\theta_i)^2\ )$.

Função de Custo com a Regularização L2

$$J(\vec{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\left[y^{(j)}\ \log(\ f_{\vec{\theta}}(x^{(j)})\ )\ +\ (1-y^{(j)})\ \log(\ 1-f_{\vec{\theta}}(x^{(j)})\ )\right] + \lambda$$

O parâmetro λ controla a força da regularização

> 💡 A **regularização L2** também tem uma propriedade chamada **“smoothing”** (suavização), que **suaviza as diferenças entre os valores dos parâmetros,** reduzindo a complexidade do modelo e **evitando o overfitting**.

▼ **Quando usar a regularização L2?**

> *Quando tiver ocorrido uma seleção de features antes do treinamento do algoritmo.*

Em geral, a regularização L2 é útil quando o modelo tem muitos recursos e o conjunto de dados de treinamento é relativamente pequeno. Ela também é eficaz quando os recursos são altamente correlacionados, pois ajuda a diminuir a multicolinearidade.

A regularização L1 e L2 - `Elastic Net`

A regularização Elastic Net é útil quando há muitas características nos dados que podem ser irrelevantes para a predição. O termo da regularização L1 ajuda a tornar os coeficientes dessas características irrelevantes iguais a zero, enquanto o termo L2 ajuda a evitar a superestimação dos coeficientes restantes.

$$J(\vec{\theta}) = -\frac{1}{m}\sum_{j=1}^{m}\ \left[y^{(j)}\ \log(\ f_{\vec{\theta}}(x^{(j)})\ )\ +\ (1-y^{(j)})\ \log(\ 1-f_{\vec{\theta}}(x^{(j)})\ )\right] + \lambda_1 \sum_{i=1}^{n} |\theta_i| + \lambda_2 \sum_{i=1}^{n} (\theta_i)^2$$

Onde:

- *O parâmetro λ_1 controla a força da regularização da penalidade L1*
- *O parâmetro λ_2 controla a força da regularização da penalidade L2*

▼ **Quando usar a regularização Elastic Net?**

> *Quando tiver ocorrido uma seleção de features antes do treinamento do algoritmo e você quer usar uma segunda seleção de features.*

A regularização Elastic Net é uma combinação das regularizações L1 e L2 e permite controlar a força relativa de cada técnica de regularização usando um parâmetro de elasticidade. Isso pode torná-la mais flexível do que as outras duas técnicas, mas também é mais difícil de ajustar.

Polynomial Regression

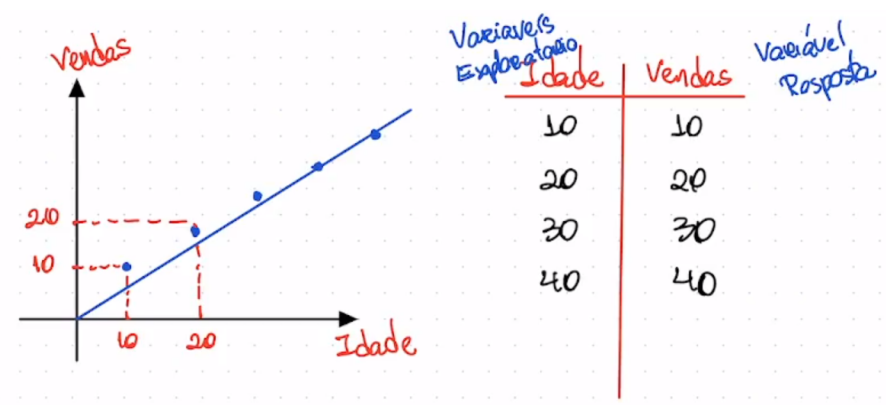
A regressão polinomial é um método estatístico para modelar a relação entre uma variável independente e uma variável dependente, assumindo que essa relação possa ser aproximada por uma função polinomial.

Todo o fenômeno de estudo possui uma variável que é responsável por encapsular ele, por exemplo, se estamos analisando a performance financeira de uma startup, a variável `Vendas` teria esse papel; comumente denominamos essa variável de **variável resposta**. Além da variável resposta, também existem outras variáveis que caracterizam o fenômeno como: `dia_da_semana`, `tipo_produto`, `quantidade` e etc…, as quais são chamadas de **variáveis exploratórias** e são usadas para criação de uma modelo responsável por tentar prever o valor da variável resposta. De maneria simples, queremos determinar, através de um modelo fenômeno, qual será o comportamento da variável resposta (dependente) quando variamos uma ou mais das variáveis exploratórias (independentes).

Anteriormente fizemos estudo da regressão linear, que é utilizada quando a variável resposta possui um comportamento linear em relação as variáveis exploratórias, em outras, a variável resposta cresce / decresce em intervalos (incrementos) iguais conforme aumentamos ou diminuímos as demais variáveis do problema.

No exemplo, temos as vendas possuem uma resposta linear quando aumentamos a idade. A relação entre idade e número de vendas é linear e pode ser modelada pela equação de uma reta.

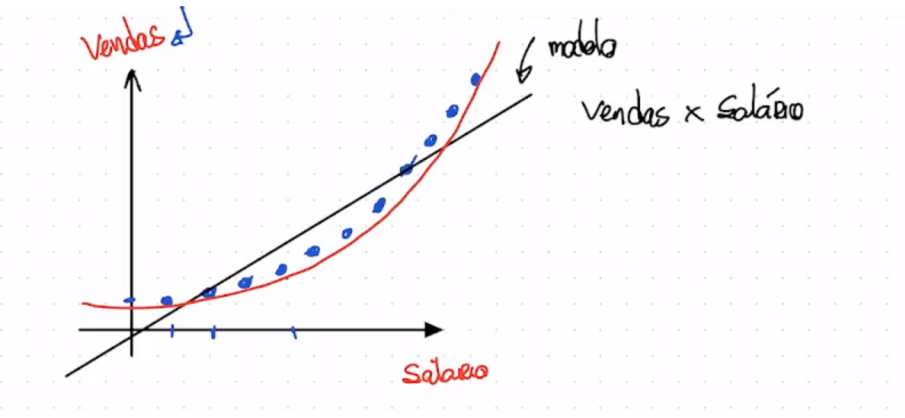
$$y = \beta_0 + \beta_1 x_1 + \epsilon$$



Em situações mais complexas, geralmente a relação entre a variável reposta e as exploratórias é não-linear e portanto não pode ser descrita por uma reta.

Podemos imaginar a situação em que queremos prever o número de vendas com base no salário do funcionário; esse salário pode depender de fatores como o tempo de experiência e um acréscimo de comissão com base nas vendas realizadas no último ano. Sabendo disso, podemos esperar que os funcionários com maiores salários são aqueles que esperamos que vendam mais no próximo mês, uma vez que eles possuem no geral mais experiência e supostamente um histórico de vendas melhor que os demais.

Nesse exemplo a relação entre as variáveis é não-linear, isso é, o incremento nas vendas (variável resposta) não é constante com a variação do salário recebido (exploratória), mas sim o número de vendas aumenta cada vez mais rápido quanto melhor remunerado for o funcionário. Portanto precisamos inserir um termo quadrático na equação do modelo ($\beta_2\ x_1^2$) qual será responsável por representar a não-linearidade da variável resposta.



$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_1^2 + \epsilon$$

No caso geral, podemos usar polinômios para modelar fenômenos que são complexos demais para usarmos uma regressão linear. Podemos imaginar que os termos que envolvem potências maiores que 1 são responsáveis por “entortar” a reta da regressão linear, de modo que agora o modelo consiga capturar melhor a relação e as nuances entre as variáveis.

Polinômio de grau n de uma variável

$$y = \left(\sum_{i=0}^{n} \beta_i x_i^i \right) + \epsilon \quad = \quad \beta_0 + \beta_1 x_1 + ... + \beta_n x^n + \epsilon$$

A função dos algoritmos de regressão polinomial é encontrar os parâmetros β_0, β_1, …, β_n que minimizem uma função de custo, ou seja, gera a curva que melhor se ajusta os pontos.

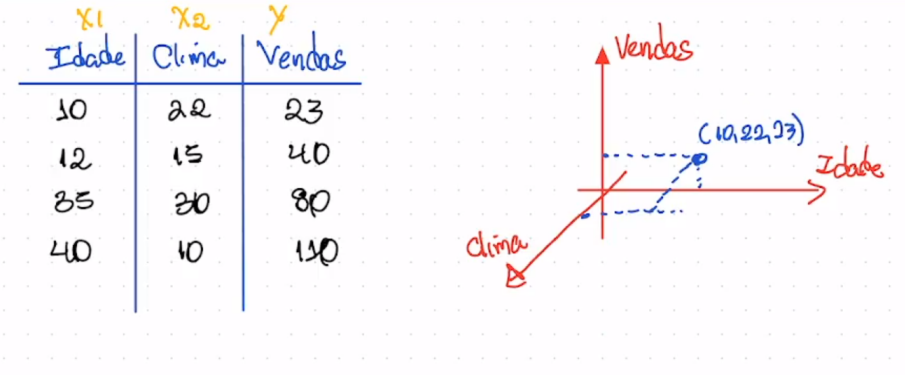
Regressão polinomial com mais de uma variável

A maioria esmagadora dos problemas em que lidamos o comportamento da variável resposta do fenômeno é modelado em função de mais de uma variável.

No exemplo, supomos que as vendas de quiosque depende do clima e da idade do cliente; se quisermos modelar tal fenômeno com um polinômio de segundo grau, usamos a seguinte equação:

$$y = \beta_0 + (\beta_1 x_1 + \beta_2 x_2) + \left(\beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2\right) + \epsilon$$

Vemos que além dos termos lineares e quadráticos surge um termo cruzado ($\beta_5 x_1 x_2$), que captura o comportamento de vendas conforme variamos a idade e o clima juntos.



Um dos problemas da regressão polinomial é que equação do modelo se torna muito grande conforme vamos aumentando o grau do polinômio.

$$y = \beta_0 + (\ \beta_1 x_1 + \beta_2 x_2\ ) + \left(\ \beta_3 x_1^2 + \beta_4 x_2^2 + \beta_5 x_1 x_2\ \right) + \left(\ \beta_6 x_1^3 + \beta_7 x_2^3 + \beta_8 x_1^2 x_2 + \beta_9 x_1 x_2^2\ \right) + \epsilon$$

> 💡 No processo de treinamento estamos interessados em encontrar o nível de curvatura (grau do polinômio) e os parâmetros de ajuste que criem um modelo (curva) que explique a variável resposta sem gerar overfit.

sklearn.preprocessing.PolynomialFeatures

Examples using sklearn.preprocessing.PolynomialFeatures: Release Highlights for scikit-learn 0.24 Time-related feature engineering Comparing Linear Bayesian Regressors Poisson regression and non-no...

https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.PolynomialFeatures.html