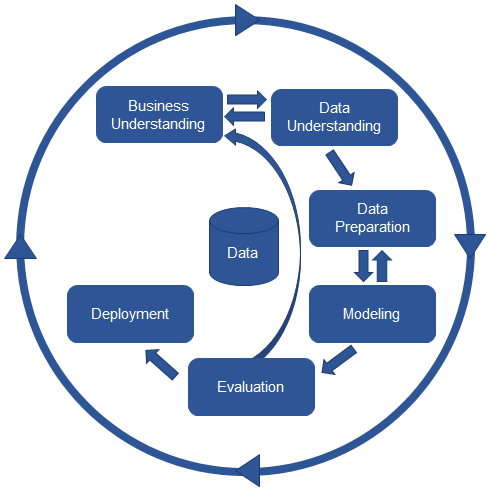
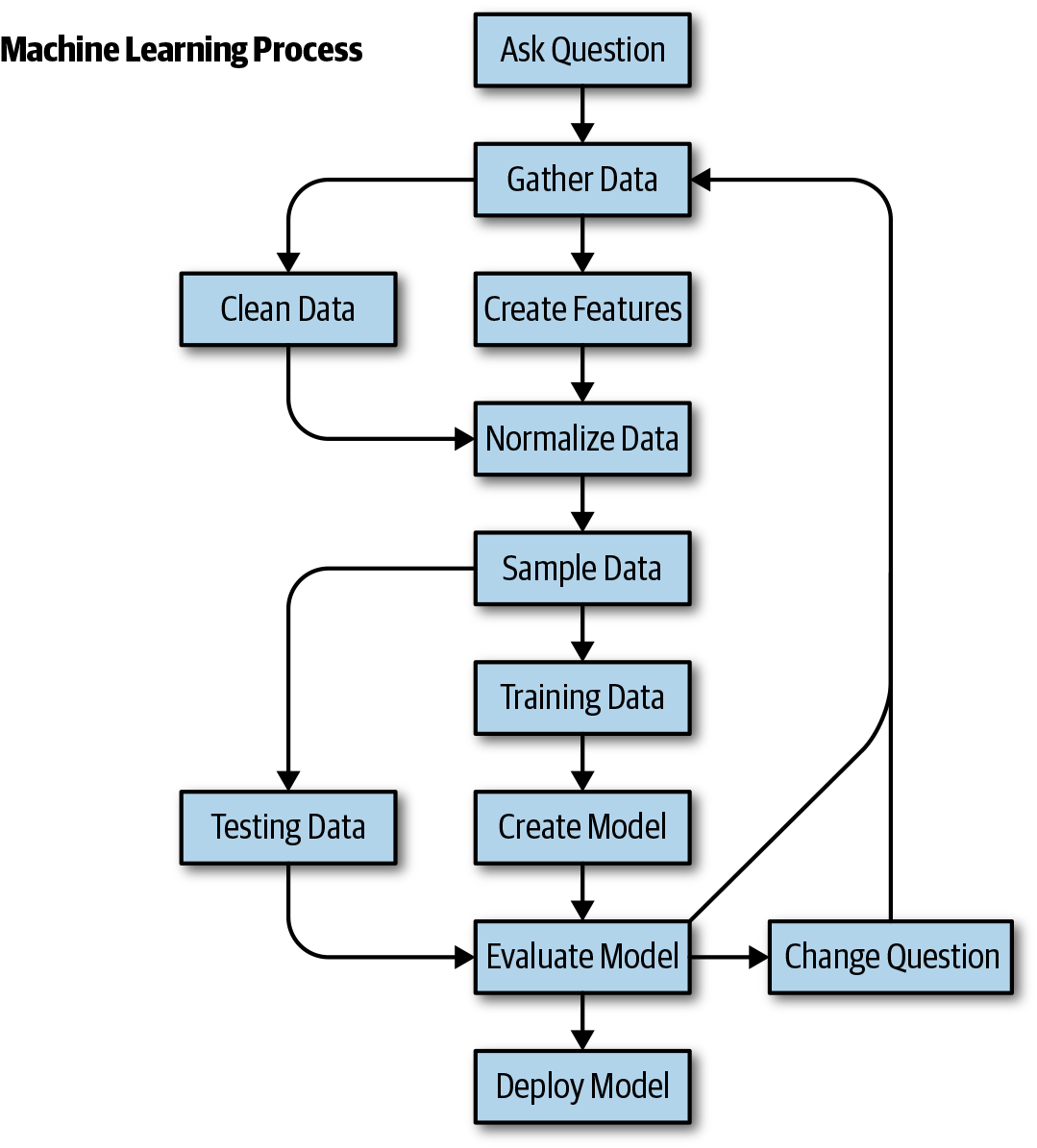
**CRISP-DM**

CRISP DM é a abreviação de *Cross Industry Standard Process for Data Mining, pode ser entendida como processo padrão da indústria cruzada para mineração de dados. É o processo para fazer mineração de dados, que contém vários passos que podem ser seguidos para uma melhoria contínua.*

*O CRISP DM surgiu justamente para atender aos projetos que estão diretamente envolvidos com o processamento e a análise de um grande volume de dados.*

* *Business Understanding*
* *Data Understanding*
* *Data Preparation*
* *Modeling*
* *Evaluation*
* *Deployment*

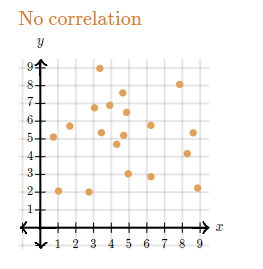
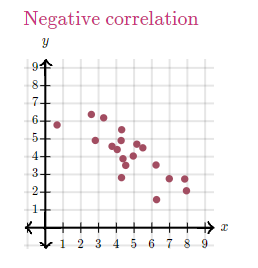
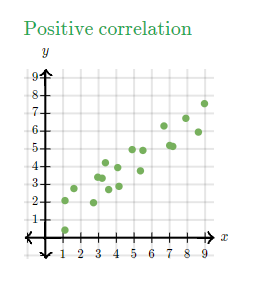
**

**

**Explore os dados**

Antes de criar o modelo é importante fazer análises de dados exploratórias. Isso dar noção dos dados, mas também uma ótima desculpa para conhecer as unidades do negócio que controlam esses dados e discutir problemas com elas.

* **Tamanho de dados -** df.shape
* **Estatísticas resumidas -** df.describe()
* **Gráfico de dispersão (scatter plot)** - Mostra o relacionamento entre duas colunas numéricas, e ajuda a identificar correlações.



* **Histograma -** Um histograma exibe dados numéricos agrupando dados em "caixas" de igual largura. Cada bin é plotado como uma barra cuja altura corresponde a quantos pontos de dados estão naquele bin.
* **Gráfico conjunto -** Gráfico de dispersão mais sofisticado (histogramas nas bordas e linha de regressão, que é chamada de joint plot)
* **Matriz de pares (pair grid) -** matriz de colunas e estimativas de densidade kernel.
* **Box plot -** fornece informação sobre as seguintes características do conjunto de dados: localização, dispersão, assimetria, comprimento da cauda e outliers.
  + Quanto maior for o quadrado do box plot maior é a variância.
  + Se a linha da mediana do box plot for mais baixo é dito com assimétrico positivo e ao contrário é assimétrico negativo, se tiver bem no meio são simétricos.
  + Amplitude (dispersão) = valor máximo - valor mínimo, quanto maior a amplitude, maior a variância dos dados.
* **Violin plot -** É semelhante a um gráfico de caixa, com a adição de um gráfico de densidade do kernel girado em cada lado.
* **Correlação -** Colunas com alto grau de correlação não agregam valor e podem prejudicar a interpretação da importância dos atributos e do coeficiente de regressão. Se tivéssemos colunas correlacionadas, poderíamos optar por remover.
* **Radviz -** Exibe todas as amostras em um círculo, com atributos nas circunferências. Você pode imaginar que cada figura tenha uma mola que puxa as amostras para ela com base no valor. Técnica para avaliar o grau de separação entre os alvos.
* **Coordenadas paralelas -** Para dados multivariados, podemos usar um gráfico de coordenadas paralelas para observar visualmente os agrupamentos.

**Limpe os dados**

De posse dos dados, devemos garantir que estejam em um formato que possamos criar um modelo.

* A maioria dos modelos do scikit-learn exige que os atributos sejam numéricos (integer or floats).
* Muitos modelos falham caso recebam valores ausentes (NaN).
* Alguns modelos terão melhor desempenho se os dados estiverem padronizados (têm um valor de média igual a 0 e um desvio padrão igual a 1).
* Alguns datasets podem possuir “**leaky features**”, que são variáveis que contêm dados sobre o futuro ou o objetivo.
* Uma limpeza nos dados pode ser um pouco demorada.
* Verifique os dados e garanta que os tipos façam sentido.
* Tipos de string com baixa cardinalidade são chamadas de colunas de categorias, e talvez valha a pena criar colunas dummy (pd.get\_dummies).
* **Pandas-profilling** é um excelente meio para obter informações estatística do dataframe
* O método **describe** é usado para obter informações estatísticas resumidas.
* O método **isnull** é usado para identificar colunas ou linhas com valores ausentes.
* É preciso avaliar quais colunas que têm muitos valores nulos podemos descartar ou utilizar.
* O método **value\_counts** serve para contar valores das amostras.
* É importante também padronizar os nomes das colunas limpando elas
  + Com o janitor podemos usar o método clean\_names

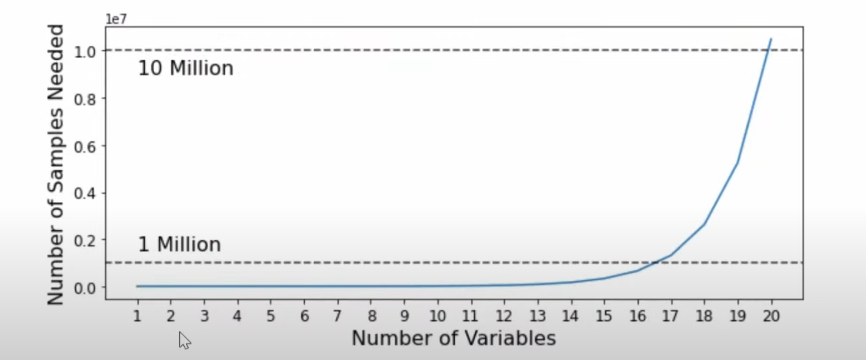
**Pré processamento de dados**

* **Padronize os dados (Normalize) -** Alguns algoritmos, como o SVM, apresentam melhores desempenhos com dados padronizados. Cada coluna deve ter um valor médio igual a 0 e um desvio padrão igual a 1. Qualquer amostra que você vá usar para predição mais tarde deverá ser padronizada com os mesmos valores que o modelo foi treinado, para evitar vazamento de informações.
  + MinMaxScaler = escala em intervalo, 0 a 1.
  + StandardScaler = baseado no desvio padrão.
* **Variáveis dummy -** Criar novas colunas baseadas em dados categóricos. Conhecida como one-hot-encoding ou indicator encoding. Dummies são particularmente úteis caso os dados sejam nominais (não ordenados).
* **Label Encoding -** Se tivermos dados nominais com alta cardinalidade, podemos usar uma codificação de rótulos, cada categoria se transformará em um número. O codificador impõe uma ordem que pode ser ou não desejável. Só consegue lidar com uma coluna de cada vez.
* **Codificador de frequência -** Outra opção de lidar com dados de categoria com alta cardinalidade, substituindo o nome da categoria pelo contador que ela tinha nos dados de treinamento. Usa o mapeamento dos dados de treino para que seja possível codificar dados futuros.
* **Extraindo categoria a partir de strings -** Encontrar padrões em colunas nominais, para depois criar variáveis dummies ou combinar colunas com contadores baixos em outras categorias (ou descartá-las).
* **Category\_enconding -** converte dados de categorias em dados numéricos É útil caso você não saiba com antecedência quantas categorias há, ou se estiver usando um conjunto de palavras para representar texto.
  + **Hash\_encoder -** Uma implementação de hash multivariada com dimensionalidade/precisão configurável.
  + **Ordinal\_encoder -** pode converter colunas de categorias que tenha uma ordem em uma única coluna de números. Se houver valores ausentes, recebe -1.
  + Diversos outros algoritmos existem no category\_enconding. Se você tiver dados de alta cardinalidade (um grande número de valores exclusivos), considere usar um dos codificadores bayesianos que geram uma única coluna por coluna categórica. Estes são TargetEncoder, LeaveOneOutEncoder, WOEEncoder, JamesSteinEncodere e MEstimateEncoder.
* **Engenharia de dados para datas -** gerar colunas com atributos de data com base em uma coluna de data e hora. Isso é útil, pois a maioria dos algoritmos de aprendizado de máquina não seria capaz de inferir esse tipo de sinal a partir de uma representação numérica de uma data. (add\_datepart do fastai).
* **Adição do atributo col\_na -** criar uma coluna para preencher um valor ausente (com a mediana) e indicar que estava faltando um valor. Pode haver algum sinal em saber que um valor estava faltando.
* **Engenharia de dados manual -** Podemos usar pandas para gerar novos recursos. Para obter dados agregados por cabine e mesclá-los novamente, use o método **.groupby** para criar os dados. Em seguida, alinhe-o de volta aos dados originais usando o método **.merge.** Se você quiser somar colunas “boas” ou “ruins”, poderá criar uma nova coluna que seja a soma das colunas agregadas (ou outra operação matemática). Isso é uma espécie de arte e também requer compreensão dos dados.

**Seleção de atributos**

Usamos a seleção de recursos (feature selection) para selecionar os recursos que são úteis para o modelo. Recursos irrelevantes podem ter um efeito negativo em um modelo. Os recursos correlacionados podem tornar os coeficientes na regressão (ou a importância do recurso em modelos de árvore) instáveis ​​ou difíceis de interpretar.

A **maldição da dimensionalidade (curse of dimensionality)** é outra questão a ser considerada. Conforme você aumenta o número de dimensões de seus dados, eles se tornam mais esparsos. Isso pode dificultar a obtenção de um sinal, a menos que você tenha mais dados. Os cálculos vizinhos tendem a perder sua utilidade à medida que mais dimensões são adicionadas.

* If we have more features than observations than we run the risk of massively overfitting our model — this would generally result in terrible out of sample performance.
* 

Além disso, o tempo de treinamento geralmente é uma função do número de colunas (e às vezes é pior do que linear). Se você puder ser conciso e preciso com suas colunas, poderá ter um modelo melhor em menos tempo.

* **Colunas colineares -** Podemos usar a **correlated\_columns** para encontrar colunas que tenham um coeficiente de correlação de 0,95. Evitando a multicolinearidade perfeita ou com uma correção positiva ou negativa bem alta (para identificar podemos realizar a Correlação de Pearson).
* **Regressão lasso -** A regressão Lasso é uma técnica de regularização. É usado em métodos de regressão para uma previsão mais precisa. Este modelo usa encolhimento. Encolhimento é onde os valores de dados são reduzidos em direção a um ponto central como a média. O procedimento de laço encoraja modelos simples e esparsos (ou seja, modelos com menos parâmetros). Esse tipo específico de regressão é adequado para modelos que mostram altos níveis de multicolinearidade ou quando você deseja automatizar certas partes da seleção do modelo, como seleção de variáveis/eliminação de parâmetros.
  + L1 Lasso - À medida que seu alfa aumenta, menor será o peso dados aos atributos de menor importância. Lasso encolhe o coeficiente do recurso menos importante para zero, removendo alguns recursos completamente. Então, isso funciona bem caso tenhamos um grande número de recursos.
  + L2 Ridge -
* **Eliminação recursiva de atributos -** é uma técnica que procura eliminar atributos irrelevantes ou redundantes, resultando em um dado mais representativo, e menor em tamanho. O método consiste em remover os atributos recursivamente, determinando pesos para atributos utilizando um estimador externo, como máquina de suporte de vetores ou florestas aleatórias, e removendo os atributos menos relevantes.
* **Informações mútuas -** Ele calcula o valor da informação mútua para cada uma das variáveis ​​independentes em relação à variável dependente e seleciona aquelas que têm mais ganho de informação. Em outras palavras, basicamente mede a dependência dos recursos com o valor alvo. A pontuação mais alta significa mais variáveis ​​dependentes.
* **SelectKBest -** Selecione os recursos de acordo com as k pontuações mais altas.
* SelectPercentile - Selecione os recursos de acordo com um percentil das pontuações mais altas.
* **Importância do recurso (Árvores) -** A maioria dos modelos de árvores fornece acesso a **.feature\_importances\_** após o treinamento. Uma importância mais alta geralmente significa que há um erro maior quando o recurso é removido do modelo.

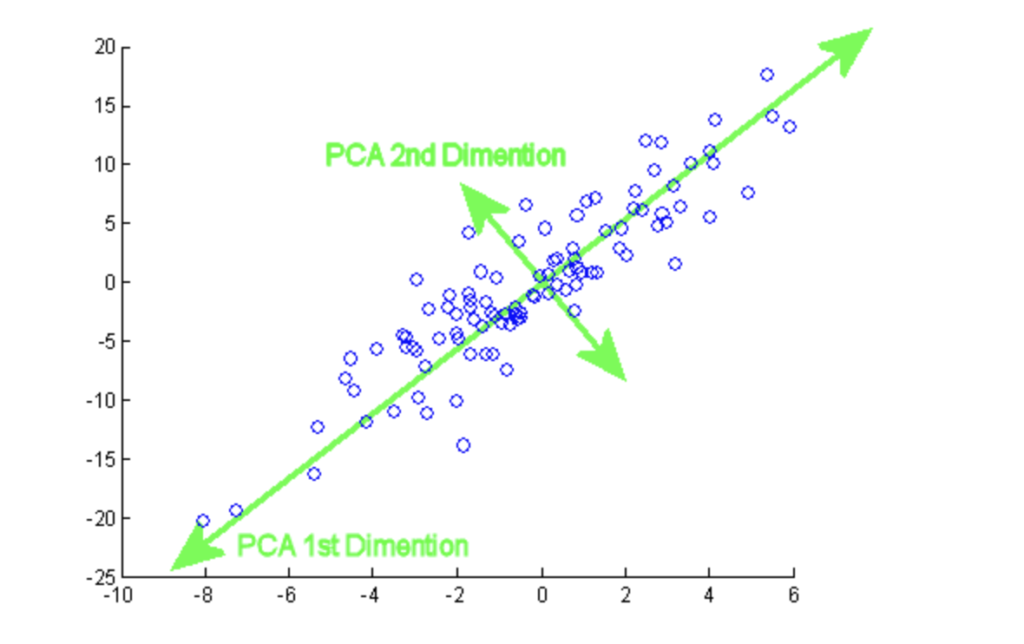
**Redução da Dimensionalidade**

Redução de dimensionalidade é um conjunto de técnicas que visam reduzir o número de dimensões (variáveis) em um conjunto de dados, sem perder muita informação. Isso pode ser útil em muitas situações, como para diminuir o tempo de treinamento de um modelo de aprendizado de máquina, para visualizar dados de alta dimensão ou para remover ruído e reduzir a complexidade de um conjunto de dados. Existem várias técnicas de redução de dimensionalidade, incluindo análise de componentes principais, t-SNE (t-distributed Stochastic Neighbor Embedding) e redução de dimensionalidade linear.

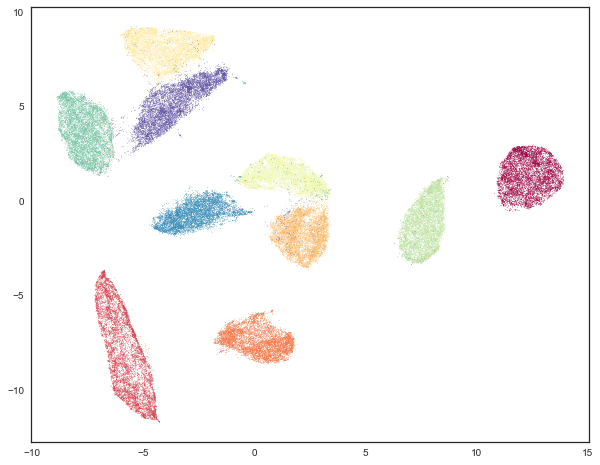
* **PCA -** Principal Component Analysis, ou Análise de Componentes Principais, em português) funciona rotacionando os dados de modo que os eixos de variação mais importantes fiquem nos planos principais. Os eixos de variação são as direções ao longo das quais os dados variam mais e os planos principais são os que capturam a maior parte da variação nos dados. Depois de rotacionar os dados, é possível selecionar as primeiras p componentes principais, onde p é o número de dimensões desejado, e descartar as demais. Isso reduz o número de dimensões dos dados sem perder muita informação. Ele é um método não supervisionado, o que significa que ele não precisa de rótulos ou etiquetas para realizar a análise.

Um componente principal é uma combinação linear das variáveis originais, são extraídos de tal forma que o primeiro componente principal explica a variação máxima no conjunto de dados. Já o segundo tenta explicar o restante e não está correlacionado com o primeiro componente, o mesmo do terceiro só que não é relacionado com o primeiro e segundo, e assim por diante. PCA só funciona quando os 2 principais componentes possuem a maior parte da variância de dados, se tiver dados complicados, PCA pode não ser a melhor escolha.

O PCA deve ser aplicado apenas em casos em que o seu dataset possui muitas colunas, realmente um número muito grande e o treino do seu modelo acaba por ser muito demorado ou inviável devido ao alto número de colunas, uma vez que o PCA é uma técnica na qual sempre haverá perda de informações.



* **UMAP -** UMAP é uma técnica de redução de dimensionalidade que é utilizada para mapear os dados de alta dimensão para um espaço de baixa dimensão de maneira que a similaridade entre os pontos é preservada o máximo possível. É um algoritmo relativamente novo, mas tem mostrado resultados promissores em várias tarefas de aprendizado de máquina. Utiliza manifold learning (aprendizado de variedades). Assim ele tende a manter itens similares topologicamente juntos e procura preservar tanto a estrutura global como local, em oposição ao t-SNE. Ás vezes a PCA é executada antes do UMAP para reduzir as dimensões e agilizar o processo.



* t-SNE -
* PHATE -

**Crie atributos**

* É realizado o processo de exclusão de colunas desnecessárias para o nosso modelo preditivo.
* Realizado get\_dummies.
* Evitando a multicolinearidade perfeita ou com uma correção positiva ou negativa bem alta (para identificar podemos realizar a Correlação de Pearson)

**OBS:.** Não existe consenso de quão alta deve ser a correlação, mas um ponto de partida é que relações acima de r = 0,9 já é bem alta para pesquisas em ciências humanas.

* Divisão do dataframe em X (atributos) e y (label).

**Classes desbalanceadas**

Se você estiver classificando dados e as classes não forem relativamente equilibradas em tamanho, a tendência para classes mais populares pode ser transportada para o seu modelo. Por exemplo, se você tiver 1 caso positivo e 99 casos negativos, poderá obter 99% de precisão simplesmente classificando tudo como negativo. Existem várias opções para lidar com *classes desequilibradas (inbalanced classes)* .

As classificações desequilibradas representam um desafio para a modelagem preditiva, pois a maioria dos algoritmos de aprendizado de máquina usados ​​para classificação foram projetados com base na suposição de um número igual de exemplos para cada classe.

Uma dica é usar uma medida diferente da precisão (AUC é uma boa escolha) para calibrar modelos. Precisão e recall também são opções melhores quando os tamanhos dos alvos são diferentes. No entanto, existem outras opções a considerar também.

**Os modelos baseados em árvore podem ter um desempenho melhor dependendo da distribuição da classe menor.** Se eles tendem a ser agrupados, **eles podem ser classificados mais facilmente.**

Os métodos de conjunto podem ajudar ainda mais a retirar as classes minoritárias. Bagging e boosting são opções encontradas em modelos de árvore como florestas aleatórias e Extreme Gradient Boosting (XGBoost).

Muitos modelos de classificação do scikit-learn suportam o parâmetro **class\_weight**. Definir isso como 'balanced 'tentará regularizar as classes minoritárias e incentivar o modelo a classificá-las corretamente. Como alternativa, você pode pesquisar em grade e especificar as opções de peso passando em um dicionário de mapeamento de classe para peso (dar peso maior a classes menores). O modelo KNN tem um parâmetro **weights** que pode influenciar os vizinhos mais próximos. Se as amostras da classe minoritária estiverem próximas, definir esse parâmetro como 'distance' pode melhorar o desempenho.

Em um problema de aprendizado de máquina, certifique-se de aumentar/diminuir a amostra **APENAS DEPOIS** de dividir em treinar, testar (e validar, se desejar). Se você fizer upsampling de seu conjunto de dados antes de dividir em treinar e testar, há uma grande possibilidade de que seu modelo seja exposto a vazamento de dados.

* **Oversampling** - O que essa técnica faz é basicamente aumentar o número de exemplos da classe minoritária
  + **Resample -** sklearn.resample não apenas gera pontos de dados extras para os conjuntos de dados por mágica, mas basicamente cria uma reamostragem aleatória (com/sem substituição) do seu conjunto de dados . Esse procedimento de equalização evita que o modelo de aprendizado de máquina se incline para a classe majoritária no conjunto de dados.
  + **RandomOverSample -** Objeto para superamostrar a(s) classe(s) minoritária(s) escolhendo amostras aleatoriamente com reposição. O bootstrap pode ser gerado de maneira suavizada.
  + **SMOTE -** Técnica da biblioteca imbalanced learn, O SMOTE funciona escolhendo um de seus k vizinhos mais próximos, conectando uma linha a um deles e escolhendo um ponto ao longo dessa linha.
  + **ADASYN -** Técnica da biblioteca imbalanced learn, ADASYN é semelhante ao SMOTE, mas gera mais amostras daqueles que são mais difíceis de aprender.
  + **Entre outros técnicas de oversampling na biblioteca imbalanced learn**
* **Undersampling -** Enquanto o oversampling aumenta a quantidade de exemplos da classe minoritária, o undersampling faz o contrário: ele diminui a quantidade de exemplos da classe majoritária.
  + **Resample -** sklearn.resample não apenas gera pontos de dados extras para os conjuntos de dados por mágica, mas basicamente cria uma reamostragem aleatória (com/sem substituição) do seu conjunto de dados . Esse procedimento de equalização evita que o modelo de aprendizado de máquina se incline para a classe majoritária no conjunto de dados.
  + **RandomUnderSample -** Esta classe seleciona amostras aleatoriamente.
  + **NearMiss -** Esta classe usa os vizinhos mais próximos para reduzir a amostragem.
  + **Entre outros técnicas de oversampling na biblioteca imbalanced learn**
* **Combine (Ambos) -**  Existem combinações de métodos de oversampling e undersampling que se mostraram eficazes e, juntos, podem ser considerados técnicas de reamostragem.
  + **SMOTETomek -** Over-sampling usando SMOTE e limpeza usando links Tomek.
  + **SMOTEENN -** Combine over e undersampling usando SMOTE e Edited Nearest Neighbours.

**Separe as amostras**

Sempre devemos fazer treinamentos e testes em dados distintos. Caso contrário você não saberá realmente quão bem seu modelo foi.

**Imputação de dados**

Se uma coluna tem valores ausentes precisamos imputar um valor numérico. Quando usamos alguma técnica de imputer preenchemos primeiro os dados de treinamento e então usamos esse imputer no teste. Caso contrário haverá vazamento de informações

* Fillna()
* IterativeImputer()
* KNNImputer()
* SimpleImputer()

**Dados Ausentes**

A maioria dos algoritmos não funcionará se houver dados ausentes, exceções para XGBoost, CatBoost e LightGBM.

Não há uma única resposta sobre como tratar dados ausentes. Antes de tratar precisamos saber porque aquele dados veio ausentes, tem algum motivo para ele?

Maneiras de lidar com dados ausentes:

* Remover qualquer linha com dados ausentes - dropna()
* Remover qualquer coluna com dados ausentes - drop(columns=”column” or axis=1)
* Imputar dados aos valores ausentes
  + **SimpleImputer -** Imputrador univariado para completar os valores ausentes com estratégias simples.
  + **Fillna -** Preencha os valores NA/NaN usando o método especificado.
  + **IterativeImputer -** Imputrador multivariado que estima cada recurso de todos os outros. Basicamente criar uma regressão para preve.r qual valor deve ser imputado baseado nas outras features
  + **KNNImputer -** Imputação para completar valores ausentes usando o valor médio dos k-vizinhos mais próximos.
  + **Fancyimpute -**  Fancyimpute usa algoritmo de aprendizado de máquina para imputar valores ausentes, ele usa toda a coluna para imputar os valores ausentes.
* Criar uma coluna para informar que os dados estavam ausentes;

Bibliotecas exclusivas de dados ausentes:

* missingno

**Normalize os dados**

Normalizar ou pré-processar os dados ajudará muitos modelos a terem um melhor desempenho depois, particularmente, para aqueles que dependam de uma métrica de distância para determinar semelhança. Vamos padronizar os dados para o pré-processamento, ou seja, traduzir os dados de modo que tenham um valor de média igual a 0 e um desvio padrão igual a 1. Aí **alguns modelos** não tratarão as variáveis com escala maiores como mais importantes que as variáveis com menor escala. Não é necessário padronizar colunas dummy.

* StandardScalar() = (X-mean)/std
* MinMaxScaler() (X-min/(max-min)
* Normalizer()

**Refatore**

Refatorar o código é interessante. Com isso podemos criar por exemplos duas funções:

* Limpeza geral do Dataset
* Fazer divisão entre train e test e mais algumas alterações, como normalizar e imputar dados.

**Modelo base**

Você pode usar o **DummyClassifier()** para comparar modelos, esse classificador serve como uma linha de base simples para comparar com outros classificadores mais complexos. Precisamos saber como comparar o modelo em questão com o classificador fictício.

**No Free Lunch**

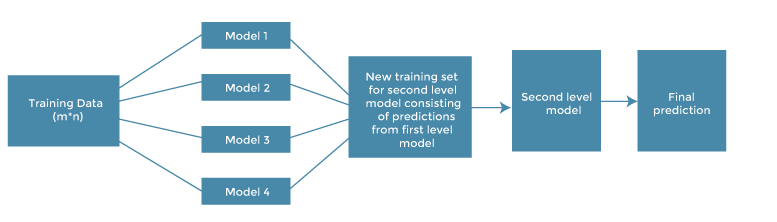
O teorema **“no free lunch” (NFL)** para aprendizado de máquina supervisionado é um teorema que implica essencialmente que nenhum algoritmo de aprendizado de máquina é universalmente o algoritmo de melhor desempenho para todos os problemas . Alguns algoritmos geralmente podem ter um desempenho melhor do que outros em certos tipos de problemas, mas todo algoritmo tem desvantagens e vantagens devido às suposições anteriores que acompanham esse algoritmo.]

**Para encontrar um bom modelo para um problema, pode ser necessário experimentar diferentes modelos e compará-los usando uma estratégia robusta de validação cruzada.**

**Stacking**

*Url:* [*https://www.youtube.com/watch?v=0BGLfYY26zQ&ab\_channel=DataProfessor*](https://www.youtube.com/watch?v=0BGLfYY26zQ&ab_channel=DataProfessor)

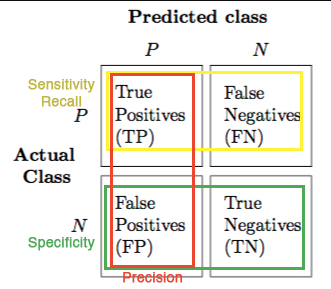
O **empilhamento** é uma das técnicas de aprendizado de máquina de conjunto mais populares usadas para prever vários nós para criar um novo modelo e melhorar o desempenho do modelo. O empilhamento nos permite treinar vários modelos para resolver problemas semelhantes e, com base na saída combinada, cria um novo modelo com desempenho aprimorado.



**Avalie o modelo**

Podemos usar os dados de teste para verificar se o mdelo pode ser generalizado para dados nunca visto antes.

* **score()** - Média de precisão da predição
* **precision\_score()** - Precisão do modelo
* **feature\_importance\_** (modelos baseados em árvores)
* **recall/sensitivity -** quantos positivos reais foram previstos como positivos. é melhor acertar o positivo que errar o positivo)
* **Specificity -** porcentagem de instâncias negativas do número total de instâncias negativas reais. (é melhor acertar o negativo que errar o negativo)
* **Precision -** porcentagem de instâncias positivas do total de instâncias positivas previstas.
* **Accuracy -** É a razão entre os valores previstos corretamente (VERDADEIRO) e todos os resultados.
* **Confusion\_matrix -** correlação entre as previsões do modelo e os rótulos de classe de pontos de dados reais
* **F1 score -** combina precision e recall considerando sua contribuição. Quanto maior a pontuação da F1, melhor.
* **PR and ROC curves**
* **Regression metrics**

****

**Otimize o modelo**

Os modelos têm *hiperparâmetros* que controlam o seu comportamento. São vários os valores desses parâmetros, alteramos o desempenho do modelo. O sklearn tem uma classe que busca em grade (grid search) para avaliar o modelo com diferentes combinações de parâmetros, devolvendo o melhor resultado.

**Matriz de Confusão**

Nos permite ver as classificações corretas, bem como FP e FN. Podemos otimizar em relação aos FP ou aos FN.

**Curva ROC e AOC**

A curva **ROC** mostra o quão bom o modelo criado pode distinguir entre duas coisas (já que é utilizado para classificação). Essas duas coisas podem ser 0 ou 1, ou positivo e negativo. Os melhores modelos conseguem distinguir com precisão o binômio.

O ROC possui dois parâmetros:

* Taxa de verdadeiro positivo (True Positive Rate), que é dado por true positives / (true positives + false negatives)
* Taxa de falso positivo (False Positive Rate), que é dado por false positives / (false positives + true negatives)

ROC é uma **curva de probabilidade**. Ela é criada traçando a taxa **verdadeiro-positivo** contra a taxa de **falsos-positivos.** Ou seja, o número de vezes que o **classificador acertou a predição conta o número de vezes que o classificador errou a predição.**

Assim, na tentativa de simplificar a análise da ROC, a AUC (“area under the ROC curve”) nada mais é que uma maneira de resumir a curva ROC em um único valor, agregando todos os limiares da ROC, calculando a “área sob a curva”.

O valor do AUC varia de 0,0 até 1,0 e o limiar entre a classe é 0,5. Ou seja, acima desse limite, o algoritmo classifica em uma classe e abaixo na outra classe.

Quanto maior o AUC, melhor.

Um modelo cujas previsões estão 100% erradas tem uma AUC de 0, enquanto um modelo cujas previsões são 100% corretas tem uma AUC de 1. Cada modelo apresentará um valor de AUC.

**Curva de Aprendizado**

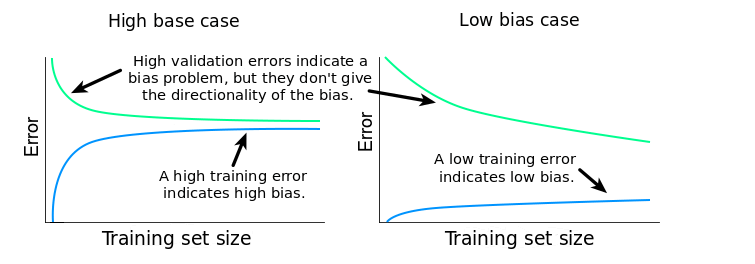
Ao contrário do viés, o erro devido à variância pode ser diminuído com a utilização de mais dados de treinamento. Nós podemos ver isso com as curvas de aprendizado, que mostram como os erros de treino e generalização evoluem conforme utilizamos mais dados de treinamento. Com isso podemos inferir que com mais dados de treino o resultado do modelo melhora, então seria interessante coletar mais dados.

Essas curvas nos dizem que coletar mais dados beneficiaria a performance de generalização do modelo. Quando o erro de treino e generalização estão próximos, coletar mais dados não beneficiaria a performance das nossas previsões; para isso, seria necessário aumentar a capacidade do nosso modelo. Por outro lado, se a performance de treino é mais alta do que a de generalização, recomenda-se coletar mais dados para treinar o modelo, até que o erro de generalização e treino convirjam.

A área sombreada em uma Learning Curve geralmente representa o intervalo de incerteza. Isso é usado para mostrar o quanto o desempenho do modelo pode variar com diferentes conjuntos de dados de treinamento.

A área sombreada fornece uma noção de quanto o desempenho do modelo pode variar dependendo do conjunto de dados de treinamento que ele recebe. Isso pode ser útil para avaliar se o modelo é robusto e está generalizando bem para novos dados. Se a área sombreada for muito ampla, isso pode indicar que o modelo não está sendo treinado de forma consistente e pode ser necessário obter mais dados de treinamento ou ajustar o modelo de alguma forma.

Se houver **variabilidade** (uma grande área sombreada) na pontuação de **treinamento**, o modelo sofrerá de **erro de viés e será muito simples** (underfit).Se houver **variabilidade na pontuação de validação cruzada**, o modelo sofre de **erro de variância** e **é muito complicado** (overfit). Outra indicação de que o modelo está com overfitting é que o desempenho do conjunto de validação é muito pior do que o conjunto de treinamento.



**Curva de Validação**

Criar uma curva de validação é uma maneira de determinar um valor apropriado para um hiperparâmetro. Uma curva de validação é um gráfico que mostra como o desempenho do modelo responde a mudanças no valor do hiperparâmetro. O gráfico mostra os dados de treinamento e os dados de validação. As pontuações de validação nos permitem inferir como o modelo responderia a dados não vistos. Normalmente, escolheríamos um hiperparâmetro que maximizasse a pontuação de validação.

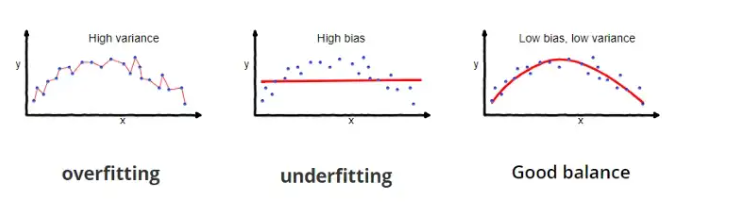
Se a curva de treinamento for alta e a curva de validação for baixa, isso indica **overfitting**. Se a curva de treinamento for baixa e a curva de validação for alta, isso indica **underfitting**. O melhor ponto de operação é onde a curva de validação é mais alta, pois isso indica que o modelo está se saindo bem tanto nos dados de treinamento quanto nos de validação.

**Bias e Variance**

**Bias** é a diferença entre a previsão média do nosso modelo e o valor correto que estamos tentando prever. O modelo com viés alto presta muito pouca atenção aos dados de treinamento e simplifica demais o modelo. Isso sempre leva a um alto erro nos dados de treinamento e teste

**Variance** é a variabilidade da previsão do modelo para um determinado ponto de dados ou um valor que nos informa sobre a distribuição de nossos dados .O modelo com alta variância presta muita atenção aos dados de treinamento e não generaliza nos dados que não viu antes.

Como resultado, tais modelos funcionam muito bem em dados de treinamento, mas apresentam altas taxas de erro em dados de teste.



**Explicando modelos**

Os modelos preditivos têm propriedades diferentes. Alguns são projetados para lidar com **dados lineares**. Outros são capazes de modelar **entradas mais complexas**. Alguns modelos podem ser interpretados com muita **facilidade**, outros são como **caixas pretas** e não oferecem muitos insights sobre como a previsão é feita.

* **Coeficiente de Regressão -** As interceptações e os coeficientes de regressão explicam o valor esperado e como os recursos afetam a previsão. **Um coeficiente positivo indica que, à medida que o valor de um recurso aumenta, a previsão também aumenta.**
* **Importância dos atributos -** Modelos baseados em árvore na biblioteca scikit-learn incluem um atributo **.fea⁠ture\_importances\_** para inspecionar como os recursos de um conjunto de dados afetam o modelo. Podemos inspecioná-los ou plotá-los.
* **LIME -** LIME (Local Interpretable Model-agnostic Explanations) é uma técnica de explicação de modelos de aprendizado de máquina que visa tornar os resultados de um modelo de aprendizado de máquina mais compreensíveis e interpretáveis para os seres humanos. Ela faz isso criando explicações simplificadas e interpretáveis para previsões feitas pelo modelo em questão.

O LIME não gosta de usar DataFrames como entrada. Então é melhor convertemos os dados em arrays numpy usando .values.

* **Interpretação de Árvores -** O **TreeInterpreter** é uma biblioteca do Python que permite interpretar e explicar previsões feitas por modelos de árvores de decisão. Ele pode ser usado para **entender como cada atributo dos dados de entrada afetou a previsão final do modelo**. Isso pode ser útil para obter uma compreensão mais aprofundada dos modelos de árvores de decisão e como eles tomam decisões. Ele calcula o bias e a contribuição de cada atributo.
* **Gráfico de Dependência Parcial -** nos permite visualizar a relação entre **mudanças em apenas um atributo e resultado.** No TreeInterpreter conseguimos saber quais atributos causaram impacto no resultado, mas não sabemos como o impacto varia à medida que o atributo muda. PDPBox (Partial Dependence Plot Box) é uma biblioteca do Python que permite visualizar a dependência parcial de uma previsão em um modelo de aprendizado de máquina em relação a um ou mais atributos de entrada. Isso pode ser útil para entender **como um determinado atributo está afetando a previsão do modelo e como o modelo está tomando decisões.**
* **Modelo substitutos -** Os modelos substitutos são modelos de aprendizado de máquina que são treinados para imitar o comportamento de outros modelos, geralmente modelos mais complexos. Eles são usados ​​para fornecer uma representação simplificada de um modelo mais complexo, de forma a torná-lo mais fácil de entender e interpretar. Por exemplo, criar um SVM mas treinar uma árvore de decisão a fim de explicá-lo.
* **Shapley -** É capaz de exibir contribuições dos atributos para qualquer modelo. É um pacote muito bom, pois não só funciona para maioria dos modelos como também é capaz de explicar predições individuais e as contribuições globais dos atributos. Funciona tanto para classificação quanto regressão, e gera valores ‘SHAP’.

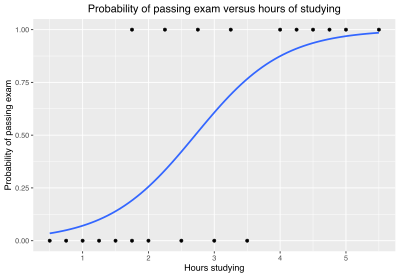
**Modelos de Classificação**

Classificação é um mecanismo de ***aprendizado supervisionado*** para rotular uma amostra com base nos recursos. Aprendizado supervisionado significa que temos rótulos para classificação ou números para regressão que o algoritmo deve aprender.

No sklearn, criamos uma instância de modelo e chamamos o método **.fit** nela com os dados de treinamento e os rótulos de treinamento. Em seguida podemos chamar o método .predict (ou o **.predict\_proba** ou **.predict\_log\_prob**) com o modelo ajustado. Para avaliar o modelo, usamos **.score** com dados de teste e rótulos de teste.

Veremos muitos modelos e discutiremos sua eficiência, as técnicas de pré-processamento que eles exigem, como evitar o overfitting e se o modelo oferece suporte à interpretação intuitiva dos resultados.

* **Logistic Regression -** A regressão logística estima probabilidades usando uma função logística. (Cuidado, embora tenha regressão no nome, é usado para classificação.) Esse tem sido o modelo de classificação padrão para a maioria das ciências. A regressão logística estima a probabilidade de ocorrência de um evento, como votou ou não votou, com base em um determinado conjunto de dados de variáveis ​​independentes. Como o resultado é uma probabilidade, a variável dependente é limitada entre 0 e 1.



* **Naive Bayes -** Baseia-se na ideia do teorema de Bayes, que afirma que a probabilidade de um evento ocorrer é igual à probabilidade anterior do evento multiplicada pela probabilidade do evento dado alguma evidência. O algoritmo funciona estimando a probabilidade de uma determinada classe (por exemplo, spam ou não spam) com base nas características dos dados (por exemplo, palavras em um e-mail). Em seguida, faz uma previsão sobre a classe dos dados com base na classe com a maior probabilidade.

É popular para aplicativos de classificação de texto , como captura de spam.

Uma vantagem desse modelo é que, por assumir independência de recursos, pode treinar um modelo com um pequeno número de amostras.

Uma desvantagem é que ele não pode capturar as interações entre os recursos.

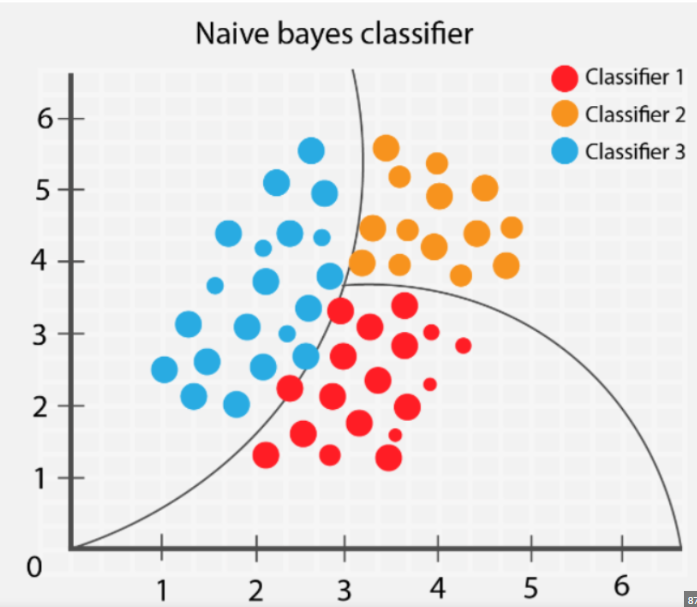
O desempenho deverá ser melhor após remoção de colunas colinerares.

Para dados numéricos contínuos, talvez seja melhor organizar em bins.

Treinamento O(Nd), onde N é o número de exemplos de treinamento e d é a dimensionalidade. Testando O(cd), onde c é o número de classes.

Esses modelos são suscetíveis ao problema de **probabilidade zero**. Se você tentar classificar uma nova amostra que não possui dados de treinamento, ela terá uma probabilidade zero. Uma solução é usar a **suavização de Laplace** . O Sklearn controla isso com o parâmetro **alpha**, cujo padrão é 1 e permite a suavização nos modelos MultinomialNB e BernoulliNB.

Existem 3 principais métodos que utilizam a teoria de Naive Bayes:



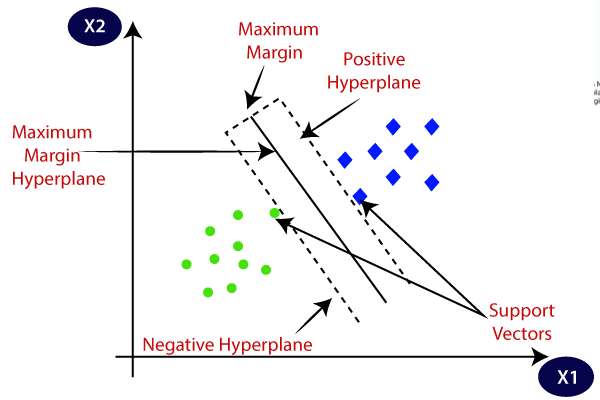
* + **BeunoulliNB -** é usado quando os dados de entrada são binários, ele assume que os dados seguem uma distribuição de probabilidade Bernoulli.
  + **GaussianNB -** usado quando os dados de entrada seguem uma distribuição normal (também conhecida como distribuição de Gauss). Ele é muito útil quando os atributos dos dados são continuamente distribuídos.
  + **MultinominalNB -** é usado quando os dados de entrada são contabilizados e assumem valores inteiros positivos. Ele assume que os dados seguem uma distribuição multinomial. É comumente usado em problemas de classificação de documentos, como classificação de spam de e-mail ou categorização de notícias.
* SVM - Uma **Support Vector Machine (SVM)** é um algoritmo que tenta encaixar uma linha (ou plano ou hiperplano) entre as diferentes classes que maximiza a distância da linha aos pontos das classes. Desta forma, tenta encontrar uma separação robusta entre as classes.

O SVM geralmente funciona bem e pode suportar espaços lineares ou não lineares usando um ***truque do kernel*** (transforma os dados de entrada em um espaço de dimensão maior, usando uma função chamada kernel, para tentar encontrar uma separação linear mais clara entre as classes, é controlado pelo parâmetro gamma).

A padronização dos dados é altamente recomendada.

A implementação do scikit-learn é O(n⁴), então pode ser difícil escalar para tamanhos grandes.

No SVM usamos principalmente SVC ou LinearSVC, a principal diferença é o tipo de kernel que cada uma delas usa. O SVC pode usar qualquer tipo de kernel, enquanto o LinearSVC só pode usar o kernel linear. Isso significa que o SVC é mais flexível e pode ser usado em uma ampla variedade de situações, mas também pode ser mais propenso ao overfitting em alguns casos. Já o LinearSVC é mais simples e pode ser mais rápido, mas só é adequado para casos em que os dados são linearmente separáveis. Outra diferença importante é que o SVC possui mais opções de parâmetros do que o LinearSVC.



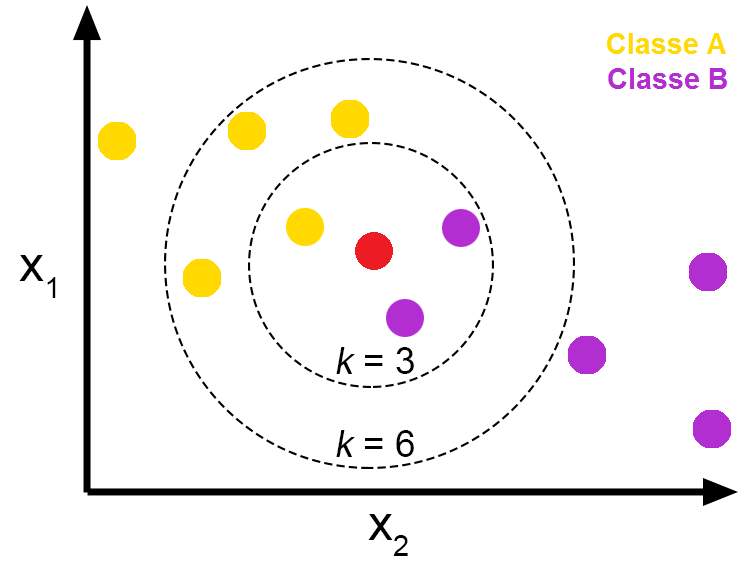
* KNN - classifica com base na distância para algum número (k) de amostras de treinamento.A família de algoritmos é chamada *de aprendizado baseado em instância,* pois não há parâmetros para aprender. Este modelo assume que a distância é suficiente para inferência; caso contrário, não faz suposições sobre os dados subjacentes ou suas distribuições.

A parte complicada é selecionar o valor k apropriado. Além disso, a maldição da dimensionalidade pode dificultar as métricas de distância, pois há pouca diferença em dimensões altas entre o vizinho mais próximo e o mais distante.

Treinamento O(1), mas precisa armazenar dados. Testando O(Nd) onde N é o número de exemplos de treinamento e d é a dimensionalidade.

Os cálculos baseados em distância funcionam melhor quando padronizados.

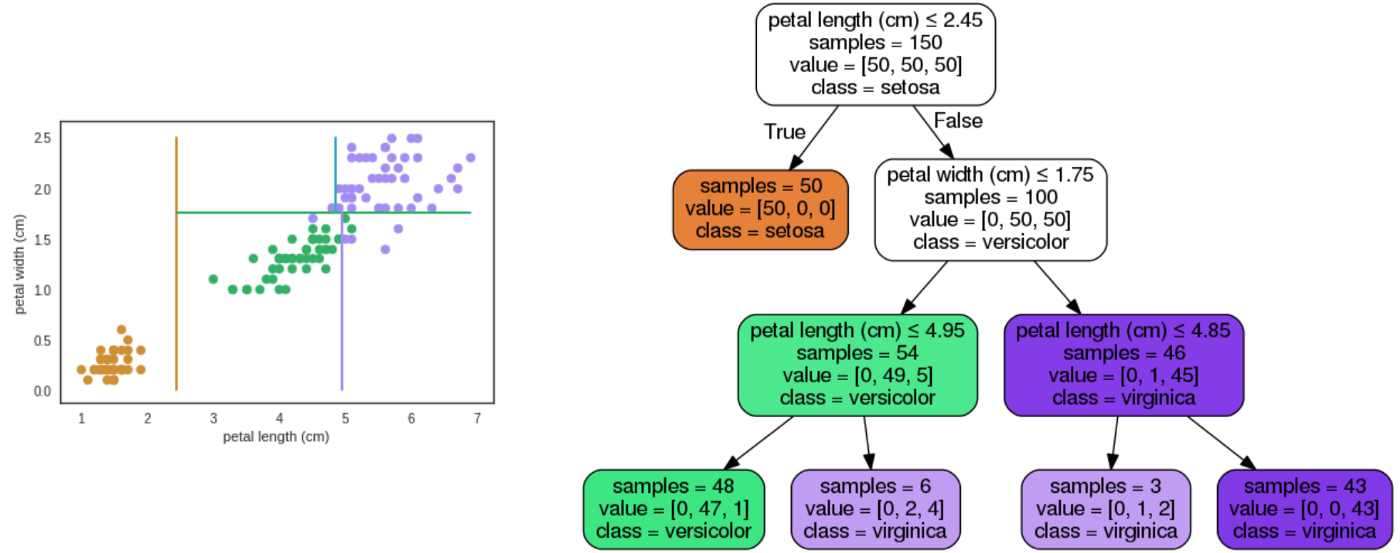
Para prevenir o overfitting, aumente **n\_neighbors**. Mude **p** para a métrica L1 ou L2 (regulização)



* **Árvores de Decisão -** Uma árvore de decisão é como ir a um médico que faz uma série de perguntas para determinar a causa de seus sintomas. Podemos usar um processo para criar uma árvore de decisão e fazer uma série de perguntas para prever uma classe de destino. As vantagens desse modelo incluem **suporte para dados não numéricos** (em algumas implementações), **pouca preparação** de dados (sem necessidade de escala), **suporte para lidar com relacionamentos não lineares**, **importâncias de recursos são reveladas** e **são fáceis de explicar**.

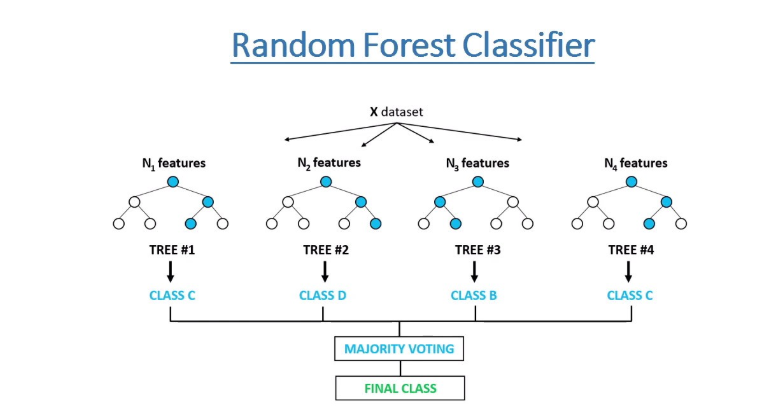
Os valores padrão levarão a uma árvore totalmente crescida overfitting. Use um mecanismo como **max\_depth** validação cruzada para controlar isso.

A árvore também é altamente dependente dos dados de treinamento. Uma pequena mudança pode mudar toda a árvore.



* **Floresta Aleatória -** Uma floresta aleatória é um conjunto de árvores de decisão.Ele usa ***bagging*** para corrigir a tendência de sobreajuste das árvores de decisão. Ao criar muitas árvores treinadas em subamostras aleatórias das amostras e características aleatórias dos dados, a variância é reduzida.

A ideia das florestas aleatórias é criar uma “floresta” de árvores de decisão treinadas em diferentes colunas dos dados de treinamento. Se cada árvore tiver mais de 50% de chance de classificação correta, você deve incorporar sua previsão. A floresta aleatória tem sido uma excelente ferramenta tanto para classificação quanto para regressão, embora recentemente tenha caído em desuso para árvores com aumento de gradiente.



* **XGBoost -** XGBoost é uma biblioteca de aprendizado de máquina de código aberto para o aprimoramento de árvores de decisão. Ele foi criado para ser muito rápido e eficiente, especialmente em grandes e complicados conjuntos de dados,

O nome XGBoost vem de eX\*treme \*Gradient Boosting, e representa uma categoria de algoritmo baseada em Decision Trees (árvores de decisão) com Gradient Boosting (aumento de gradiente)

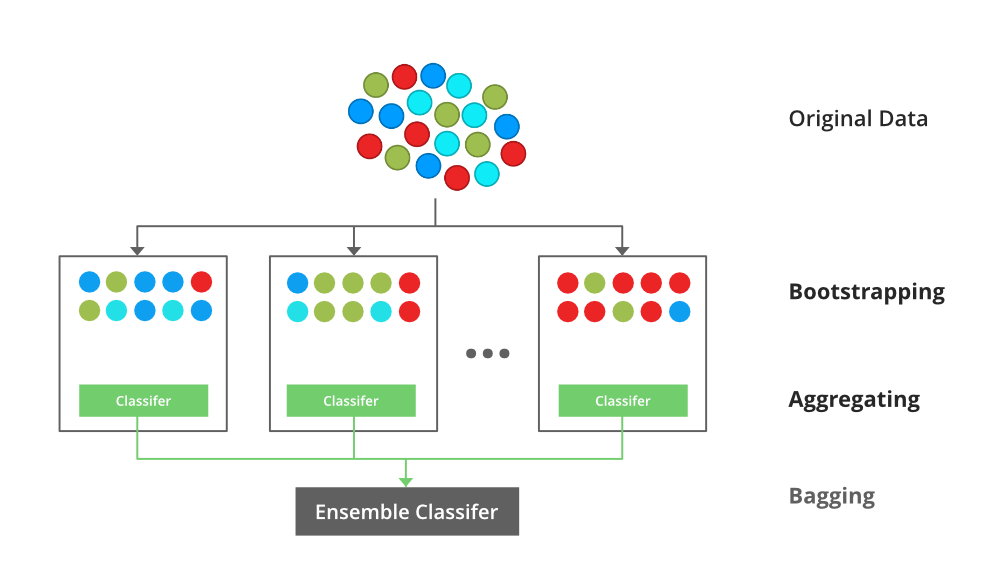
O XGBoost funciona construindo árvores de decisão em sequência, onde cada árvore tenta corrigir os erros feitos pelas árvores anteriores. Isso é chamado de "boosting". O XGBoost também inclui algumas outras características, como regularização e normalização, que ajudam a melhorar o desempenho do modelo.

XGBoost é paralelizável. Use a n\_jobsopção para indicar o número de CPUs. Use a GPU para um desempenho ainda melhor.

Não é necessário dimensionar com modelos de árvore. Necessidade de codificar dados categóricos.

O parâmetro early\_stopping\_rounds=N pode ser configurado para interromper o treinamento se não houver melhora após N rodadas. As regularizações L1 e L2 são controladas por reg\_alphae reg\_lambda, respectivamente. Números mais altos são mais conservadores.

Os termos "leaf positiva" e "leaf negativa" se referem às folhas de uma árvore de decisão que representam as classes positiva e negativa, respectivamente. Isso é comum em problemas de classificação binária, onde há apenas duas classes possíveis. "leaf positiva" e "leaf negativa" são apenas um modo de se referir às folhas da árvore de decisão e não têm nenhum significado especial no XGBoost em si. Eles são usados ​​apenas para ajudar a se referir às folhas da árvore de decisão de maneira clara e concisa.



* **Gradient Boosted com LightGBM -** é uma implementação da Microsoft. LightGBM usa um mecanismo de amostragem para lidar com valores contínuos. Isso permite a criação mais rápida de árvores (do que, digamos, XGBoost) e reduz o uso de memória.

Pode tirar proveito de várias CPUs. Ao usar o binning, pode ser 15 vezes mais rápido que o XGBoost.

* **TPOT -** usa um algoritmo genético para experimentar diferentes modelos e conjuntos. Isso pode levar de horas a dias para ser executado, pois considera vários modelos e etapas de pré-processamento, bem como os hiperparâmetros para esses modelos e opções de combinação. Em uma máquina típica, uma geração pode levar cinco ou mais minutos para ser executada.

Idealmente, os resultados devem usar validação cruzada para minimizar o overfitting.

**Bagging x Boosting**

*Bagging e Boosting são dois métodos de ensaio de conjunto que visam melhorar o desempenho de um modelo de aprendizado de máquina. Eles funcionam de maneira ligeiramente diferente e são usados ​​em situações diferentes.*

*Bagging, ou "bootstrap aggregation", é um método que consiste em treinar vários modelos de aprendizado de máquina de maneira independente a partir de amostras aleatórias do conjunto de treinamento, e depois combinar os resultados desses modelos para obter um resultado final. O objetivo do bagging é reduzir a variância do modelo, aumentando assim a precisão do modelo final. Alguns exemplos de algoritmos de aprendizado de máquina que usam bagging incluem o Random Forest e o Extra Trees.*

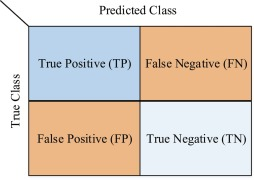
*Boosting, por outro lado, é um método que consiste em treinar modelos de aprendizado de máquina de maneira sequencial, onde cada modelo tenta corrigir os erros feitos pelo modelo anterior. O objetivo do boosting é aumentar a precisão do modelo, aumentando assim a capacidade de generalização do modelo. Alguns exemplos de algoritmos de aprendizado de máquina que usam boosting incluem o XGBoost e o AdaBoost.*

*Em resumo, o bagging visa reduzir a variância do modelo, enquanto o boosting visa aumentar a precisão do modelo.*

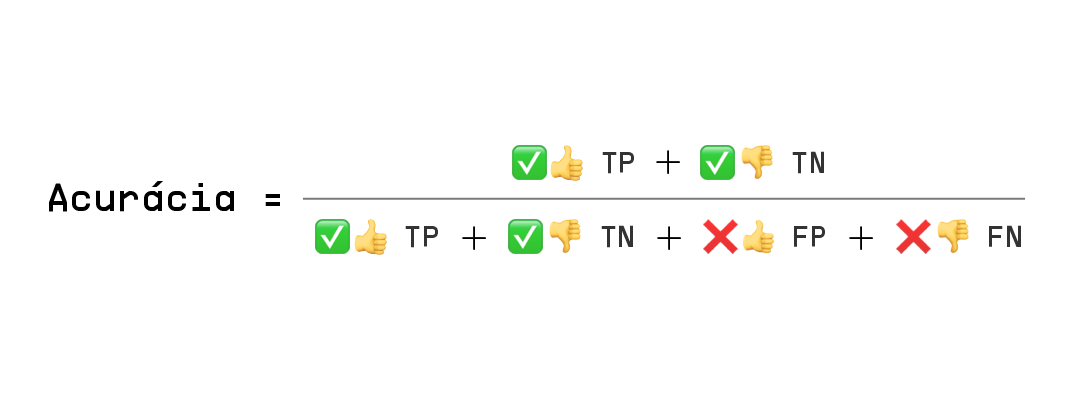
**Métricas e Avaliação de Classificação**

A avaliação de métricas de classificação é o processo de medir o desempenho de um modelo de classificação em relação a um conjunto de dados de teste. A escolha da métrica depende do problema de classificação em questão e do que é considerado importante.

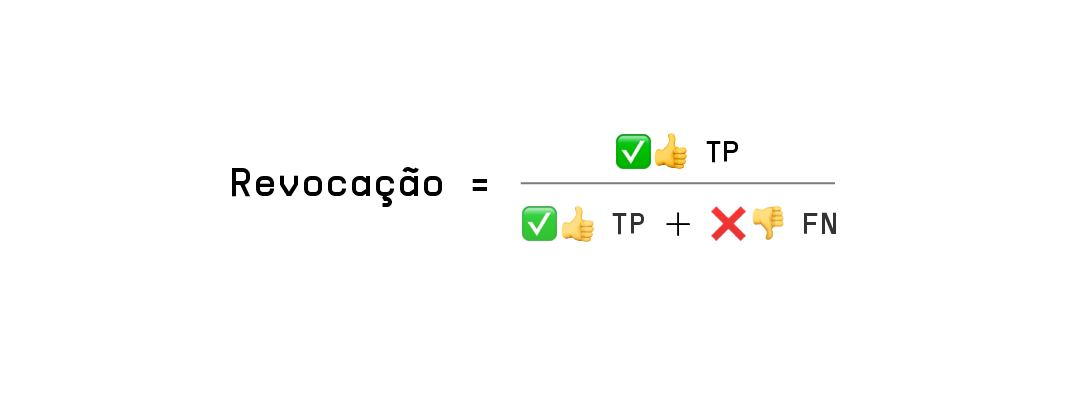
* **Matriz de Confusão -** Uma matriz de confusão pode ajudar a entender o desempenho de um classificador. Um classificador binário pode ter quatro resultados de classificação: verdadeiros positivos (TP), verdadeiros negativos (TN), falsos positivos (FP) e falsos negativos (FN). As duas primeiras são classificações corretas.



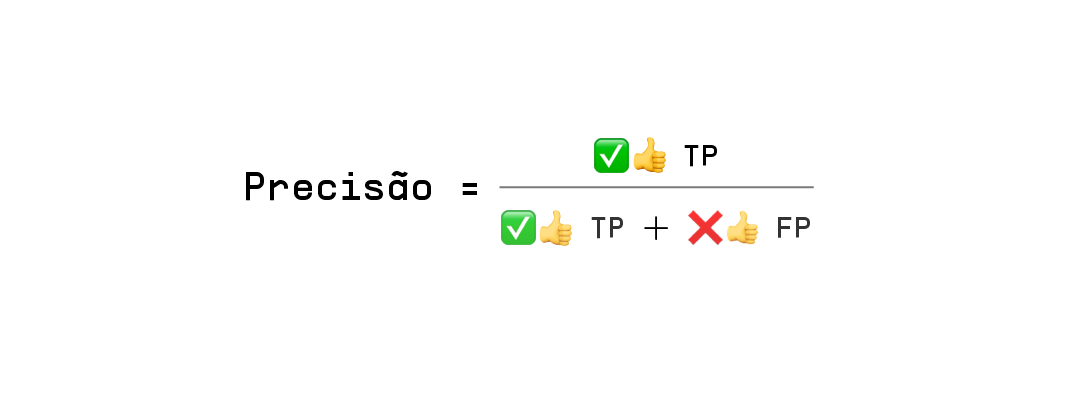
* **Acurácia** - é a porcentagem de classificações corretas. A acurácia é uma métrica útil quando o conjunto de dados de treinamento tem uma distribuição equilibrada de amostras positivas e negativas. No entanto, se o conjunto de dados tiver uma distribuição desbalanceada, a acurácia pode ser enganosa. Por exemplo, se você estiver tentando prever a ocorrência de um evento raro, mesmo que seu modelo faça poucas previsões corretas, ele ainda pode ter uma acurácia relativamente alta, pois a maioria das previsões serão negativas. Neste caso, é melhor usar outras métricas, como a precisão e o recall, que são mais robustas para conjuntos de dados desbalanceados.



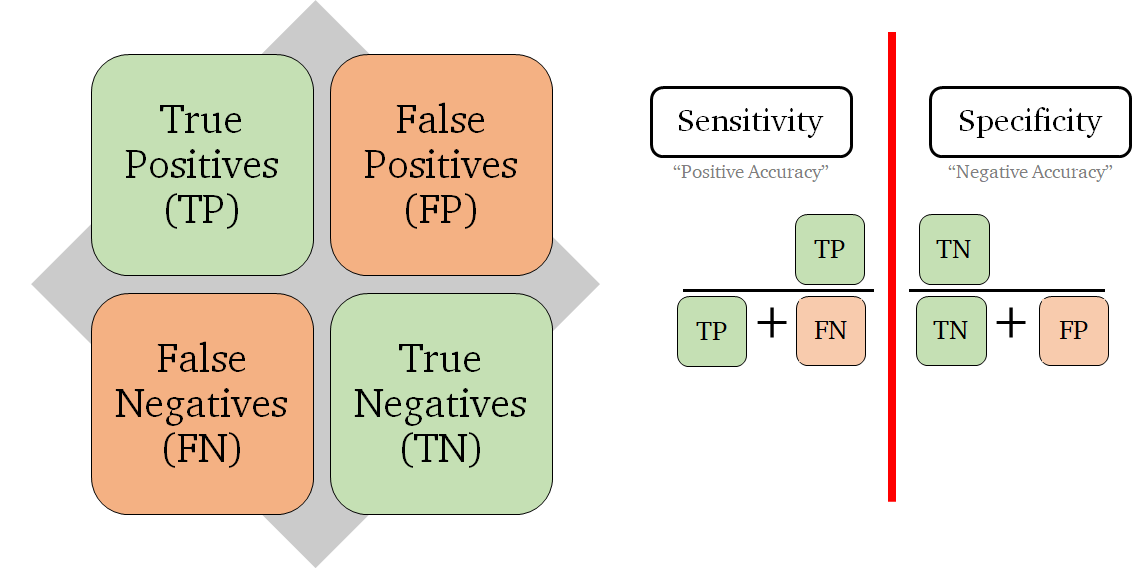
* **Recall/sensibilidade/revogação** - é a porcentagem de valores positivos classificados corretamente. (Quantos resultados relevantes são retornados?); **de todos os exemplos que são positivos, quantos foram classificados corretamente como positivos?**



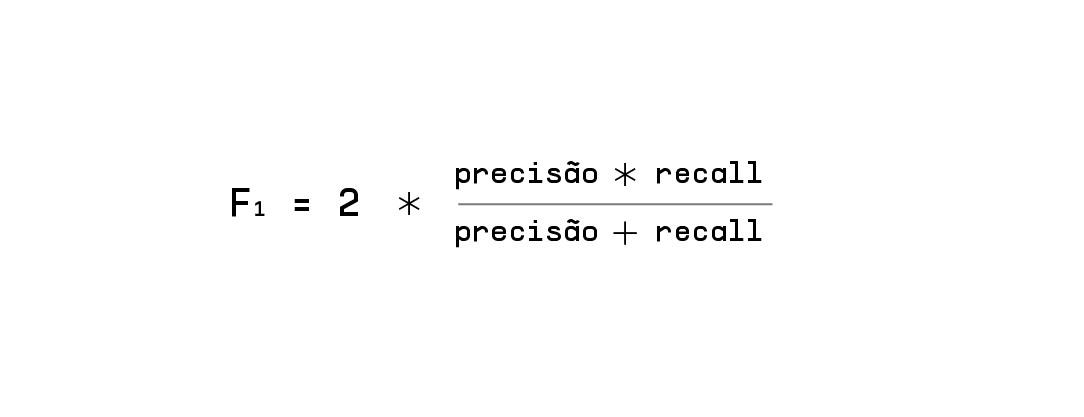
* **Precisão/precision -** A precisão é a porcentagem de previsões positivas que estavam corretas. (Quão relevantes são os resultados?). **dos exemplos classificados como positivos, quantos realmente são positivos?**

****

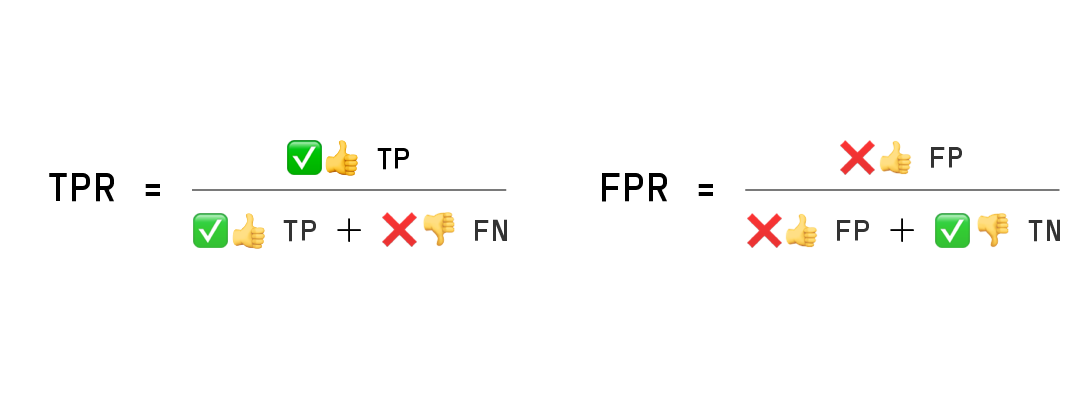
* **Especificidade/Specificity -** porcentagem de instâncias negativas do número total de instâncias negativas reais. (é melhor acertar o negativo que errar o negativo). **de todos os exemplos que são negativos, quantos foram classificados corretamente como negativos?**

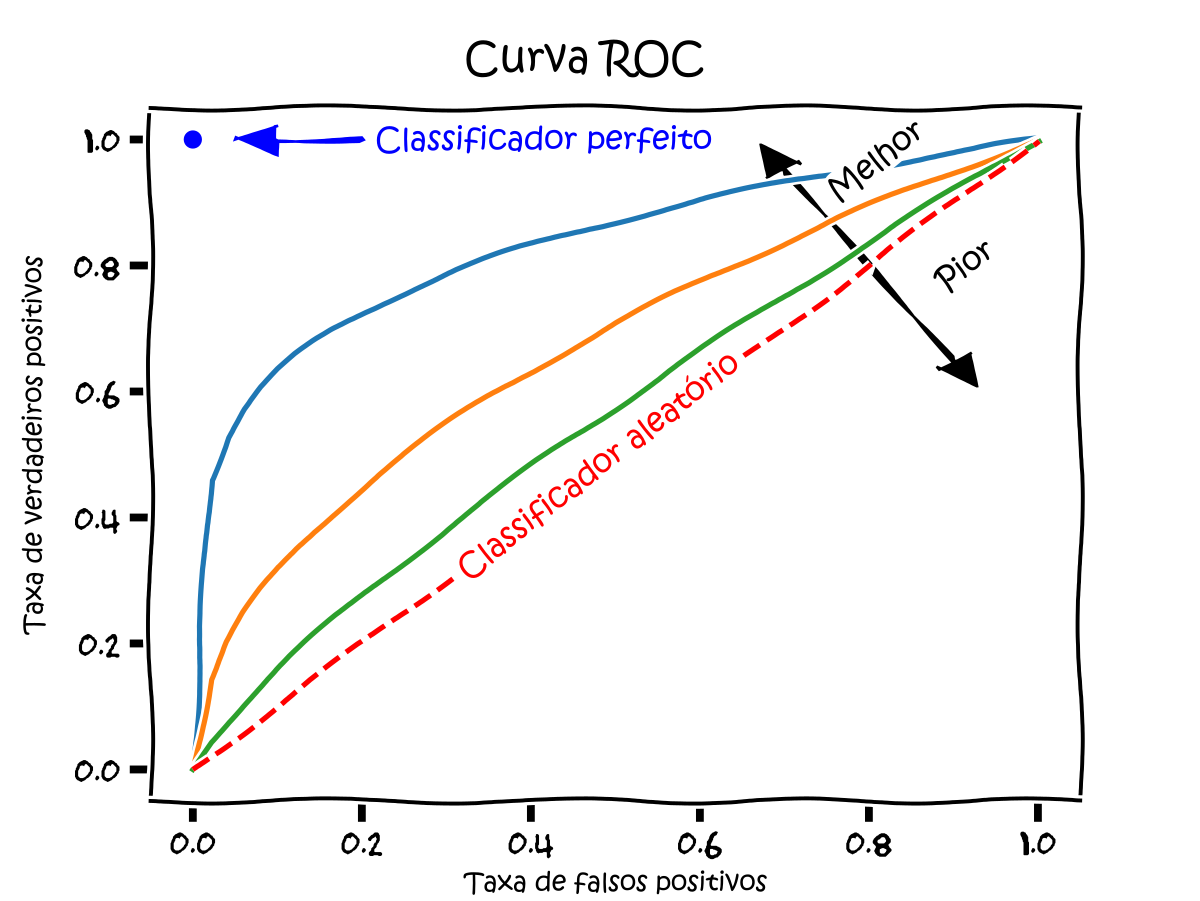


* **F1 Score/F-Measure** - leva em consideração tanto a precisão quanto a revocação. Ela é definida pela média harmônica entre as duas. Uma das características da média harmônica é que se a precisão ou a revocação for zero ou muito próximos disso, o F1-score também será baixo. Desta forma, para que o **F1-score seja alto, tanto a precisão como a revocação também devem ser altas**. Ou seja, um modelo que apresenta um bom F1-score é um modelo capaz tanto de acertar suas predições (precisão alta) quanto de recuperar os exemplos da classe de interesse (revocação alta). Portanto, esta métrica tende a ser um resumo melhor da qualidade do modelo.

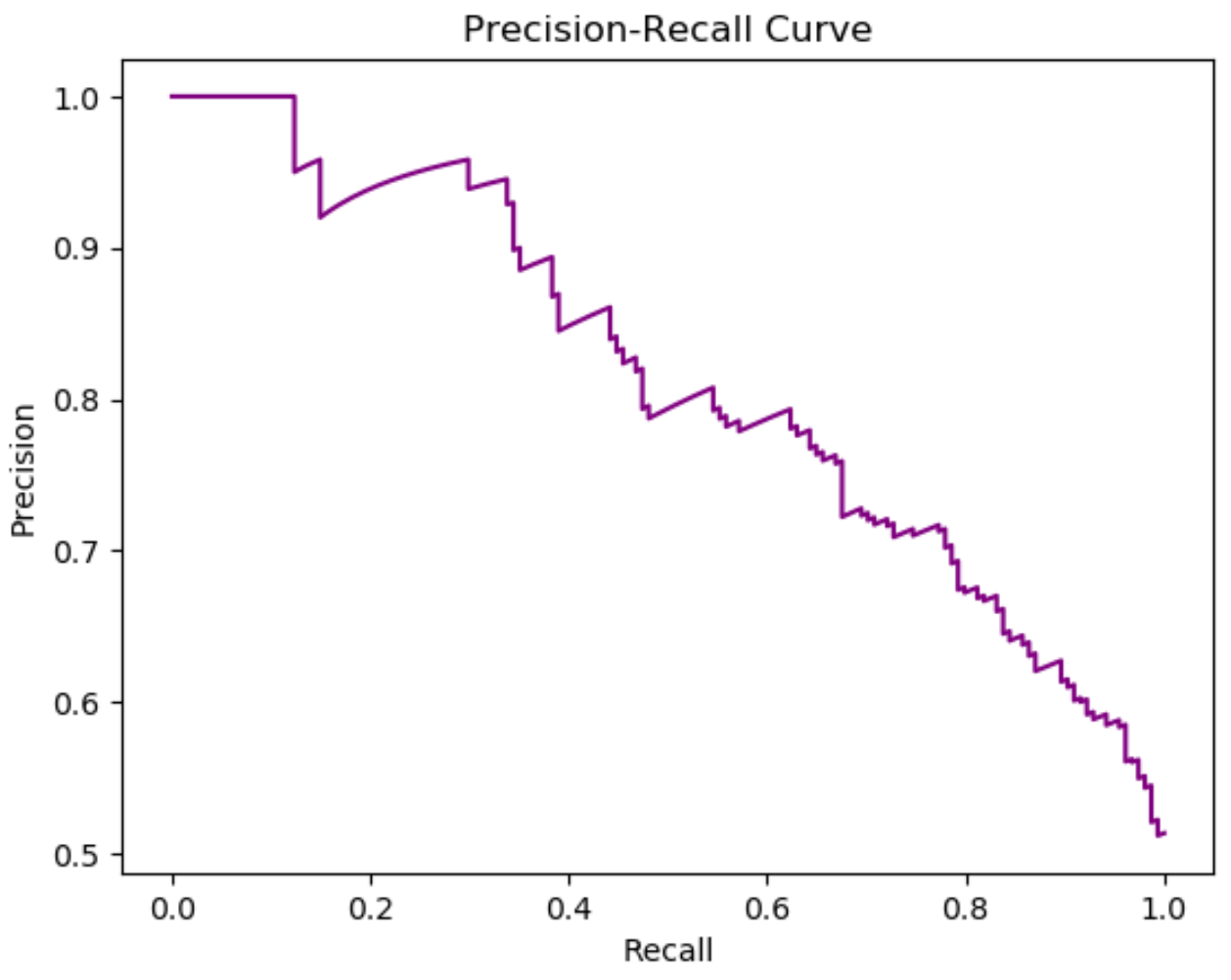


* **Relatório de classificação -** é uma função em algumas bibliotecas de machine learning que gera uma saída de resumo do desempenho de um modelo de classificação. Ele geralmente inclui métricas como precisão, sensibilidade (também conhecida como recall), especificidade e métrica F1. É útil porque fornece uma **visão geral do desempenho do modelo em um único relatório**, o que pode ser útil na avaliação e comparação de diferentes modelos.
* **ROC -** Uma curva ROC ilustra como o classificador executa rastreando otaxa de verdadeiros positivos (recall/sensibilidade) à medida que a taxa de falsos positivos (especificidade invertida) muda . Quanto mais próxima a curva estiver do canto superior esquerdo, melhor é a predição do modelo, dado que ele teria 100% de TPR e 0% de FPR. A linha tracejada indica qual seria curva de uma classificador que prevê classes de forma aleatória, e serve como um baseline de comparação. A área sob a curva ROC (AUC — Area Under the Curve ou AUROC — Area Under the Receiver Operating Characteristic curve) pode ser utilizada como métrica de qualidade de um modelo, dado que quanto mais próxima a curva estiver do canto superior esquerdo, maior será a área sob a curva e melhor será o modelo. Uma vantagem desta métrica é que ela não é sensível ao desbalanço de classes, como ocorre com a acurácia.

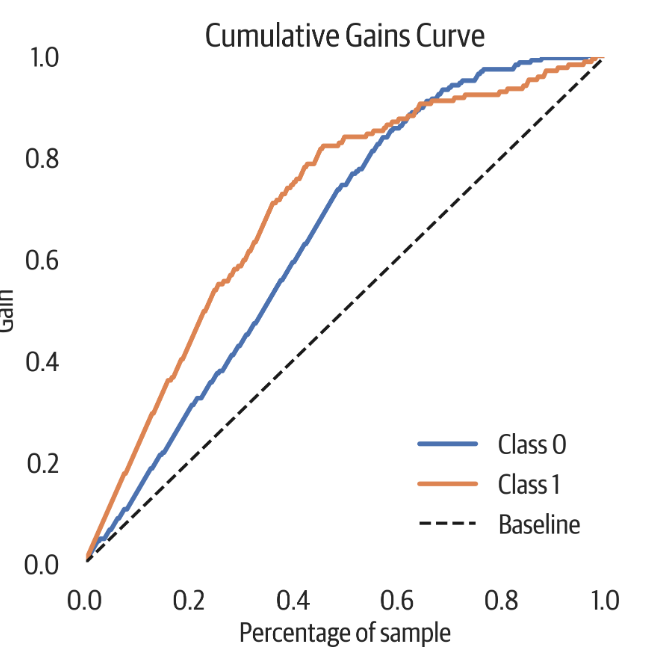




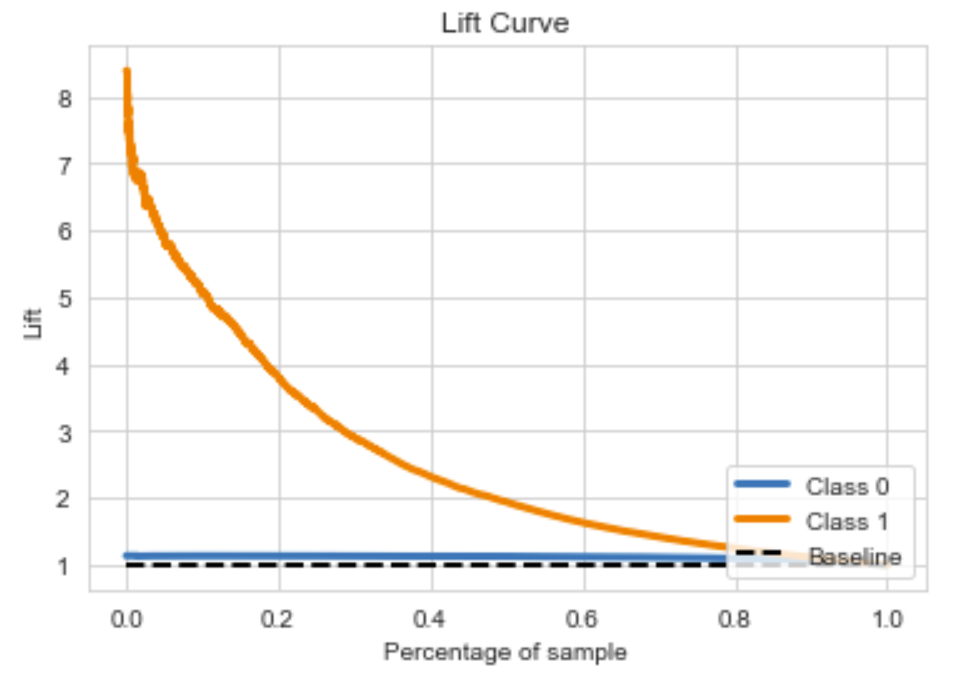
* **Curva de precisão/recall -** A curva ROC pode ser excessivamente otimista para classes desequilibradas. Outra opção para avaliar classificadores é usar uma curva de recuperação de precisão. A classificação é um ato de equilíbrio de encontrar tudo o que você precisa (recall) enquanto limita os resultados inúteis (precisão). Isso normalmente é uma troca. À medida que o recall aumenta, a precisão geralmente diminui e vice-versa.



* **Gráficos de ganho cumulativos** - podem ser usados ​​para visualizar como o desempenho de um modelo de aprendizado de máquina está mudando ao longo do tempo, por exemplo, como a precisão do modelo está mudando à medida que ele é treinado com mais dados. Em geral, um modelo que está à esquerda e acima de outro modelo é um modelo melhor.



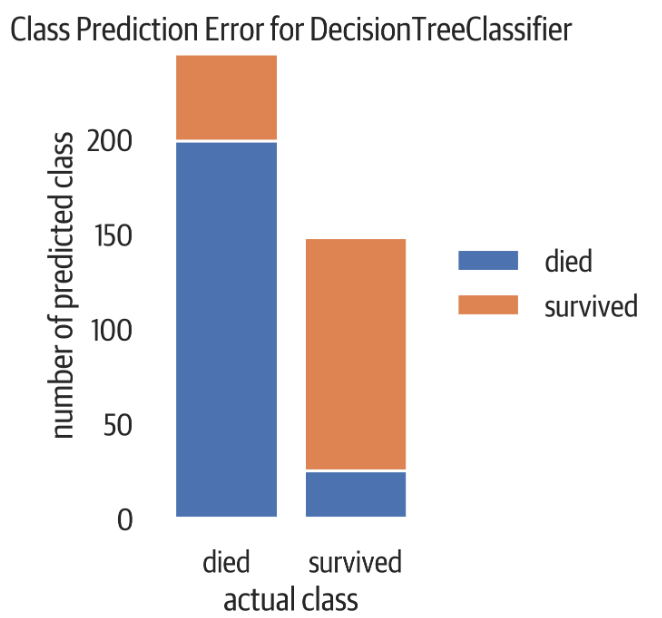
* **Gráfico de elevação -** Para criar um gráfico de elevação, você primeiro seleciona um conjunto de características e treina um modelo usando essas características. Em seguida, adiciona mais uma característica e treina um novo modelo usando o conjunto de características ampliado. Isso é repetido até que todas as características tenham sido incluídas no modelo. O desempenho de cada modelo é então plotado no gráfico como uma função do número de características incluídas.



* **Balanceamento das classes -** Yellowbrick tem um gráfico de barras simples para visualizar os tamanhos das turmas. Quando os tamanhos relativos das turmas são diferentes, a precisão não é uma boa métrica de avaliação.

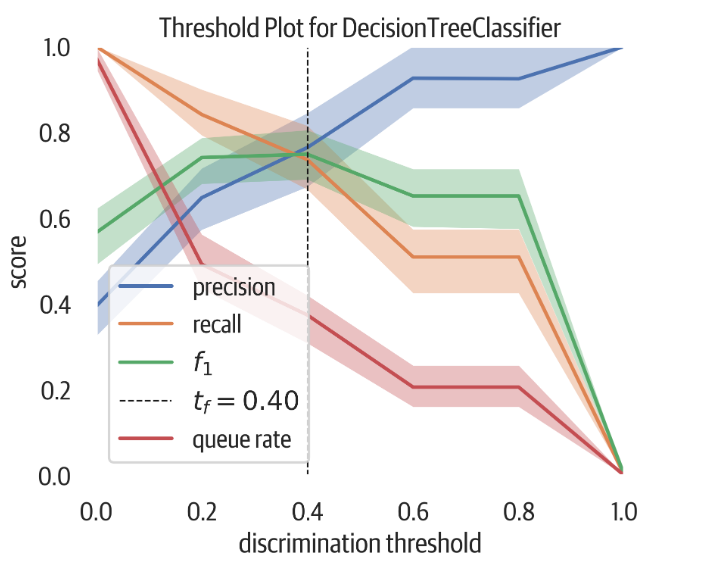
,

* **Erro de predição de classe -** O gráfico de erro de previsão de classe do Yellowbrick é um gráfico de barras que visualiza uma matriz de confusão



* **Limiar de discriminação -** é um conceito comumente usado em classificação binária, onde existem duas classes. Ele se refere ao ponto de corte usado para decidir se uma observação pertence à primeira classe ou à segunda classe. Por exemplo, imagine que você esteja treinando um modelo de aprendizado de máquina para prever se uma pessoa tem uma doença ou não. Se o modelo produzir uma probabilidade de a pessoa ter a doença, o limiar de discriminação será usado para decidir se a pessoa é classificada como tendo a doença (classe 1) ou não tendo a doença (classe 0).

O limiar de discriminação é geralmente definido como 50%, o que significa que se a probabilidade de uma observação pertencer à classe 1 for maior do que 50%, ela será classificada como classe 1; caso contrário, ela será classificada como classe 0. No entanto, o limiar de discriminação pode ser ajustado dependendo da situação. Por exemplo, se a detecção de uma doença for muito importante, pode ser necessário aumentar o limiar de discriminação para reduzir o número de falsos positivos (pessoas classificadas como tendo a doença, mas que na verdade não têm). Isso pode resultar em um aumento no número de falsos negativos (pessoas classificadas como não tendo a doença, mas que na verdade têm).



**Modelos de Regressão**

A regressão é um tipo de problema de aprendizado de máquina supervisionado em que o objetivo é prever um valor contínuo (numérico) a partir de um conjunto de características de entrada. Por exemplo, você poderia treinar um modelo de regressão para prever o preço de uma casa com base em características como tamanho, localização e número de quartos. A regressão é diferente da classificação, que é outro tipo de problema de aprendizado de máquina em que o objetivo é prever uma classe ou categoria a partir de características de entrada.

No sklearn o resultado score é coeficiente de determinação (r² ou R²). Esse número explica o percentual de variação de dados de entrada capturado pela predição.

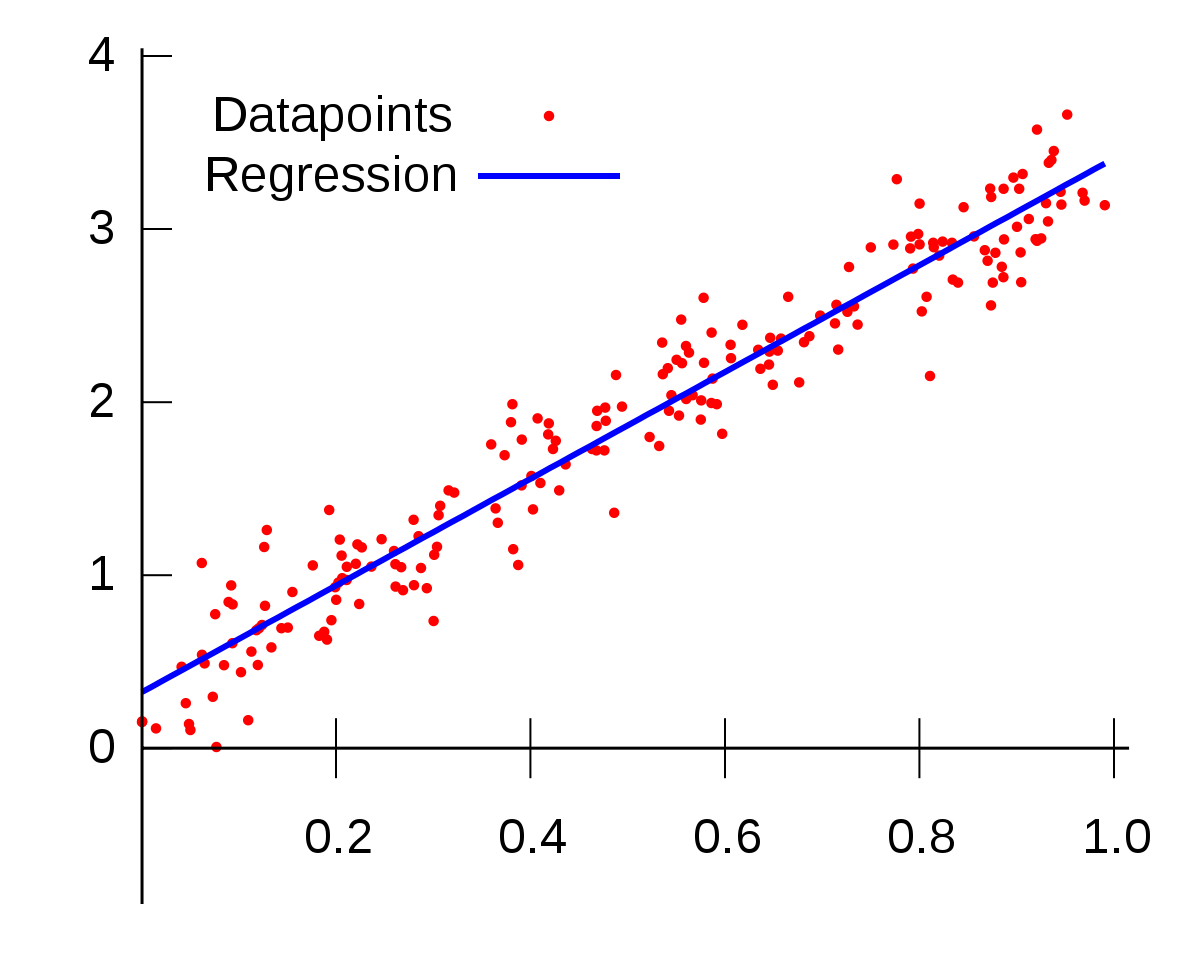
* **Regressão Linear -** É simples, ela tenta fazer a adequação da fórmula y = mx + b. Com ele temos um intercepto (b) e um coeficiente (m). Cada atributo terá um coeficiente.

Ele é chamado de "linear" porque o modelo assume que os dados seguem uma relação linear.

Pode usar **n\_jobs** para melhorar o desempenho.

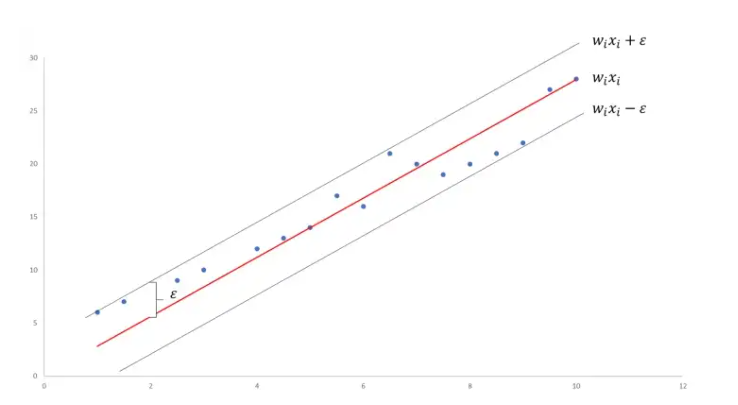
É importante padronizar os dados antes de fazer o treinamento do modelo.

Para evitar o overfitting deve simplificar o modelo não usado ou adicionado atributos polinomiais.



* **SVMs -** O Support Vector Machines (SVM) é um tipo de modelo de aprendizado de máquina que é amplamente utilizado para classificação, mas também pode ser usado para a regressão. No caso da regressão, o SVM é treinado para prever um valor contínuo em vez de uma classe.

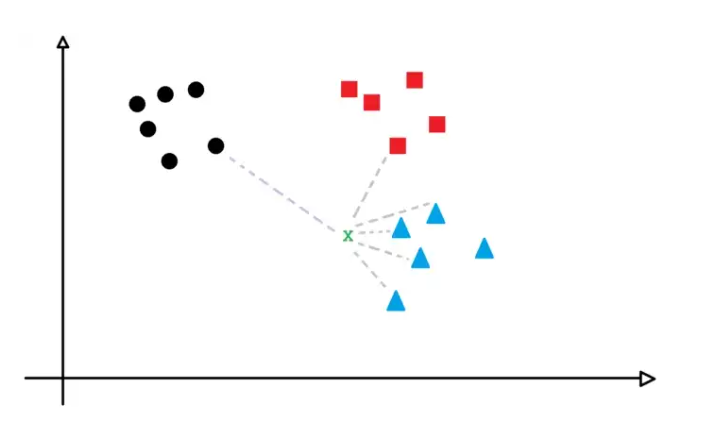
O SVM para regressão funciona de maneira semelhante ao SVM para classificação. Ele tenta encontrar um hiperplano de separação que maximize a margem entre os pontos de dados mais próximos (os vetores de suporte). No entanto, ao invés de classificar os pontos de dados em diferentes classes, o SVM para regressão é treinado para prever os valores alvo para cada ponto de dados.



* **KNN -** é um modelo de regressão baseado em instâncias que utiliza o conceito de vizinhança para prever o valor de saída de um dado exemplo com base nos valores de saída de outros exemplos semelhantes.

O KNN Regression funciona da seguinte maneira: dado um exemplo de entrada, o modelo seleciona os K exemplos mais semelhantes (vizinhos) a ele na base de dados de treinamento. Em seguida, calcula a média dos valores de saída dos K exemplos selecionados para prever o valor de saída do exemplo de entrada.

Para evitar o overfitting é recomendado aumentar o número de vizinhos, e mudar o parâmetro p.



* **Árvores de decisão -** O DecisionTreeRegressor é um modelo simples e fácil de interpretar, mas pode sofrer com o overfitting se a árvore crescer muito profunda. Para evitar isso, é possível utilizar técnicas de regularização, como a limitação da profundidade máxima da árvore ou o pruning, que remove os nós da árvore que não contribuem significativamente para o desempenho do modelo.

Não precisa normalizar mas não pode ter valores nulos.

* **Floresta Aleatória -** As árvores são boas, mas podem gerar overfitting.O RandomForest criando uma floresta de árvores de decisão, em que cada árvore é treinada com um subconjunto aleatório de exemplos e características do conjunto de dados de treinamento. As previsões de cada árvore são então combinadas para fazer a previsão final do modelo.

O RandomForestRegressor é um modelo poderoso e robusto, que tende a ter um bom desempenho mesmo com conjuntos de dados de treinamento pequenos ou com muita variância. Ele também é fácil de implementar e não requer muito ajuste de hiperparâmetros.

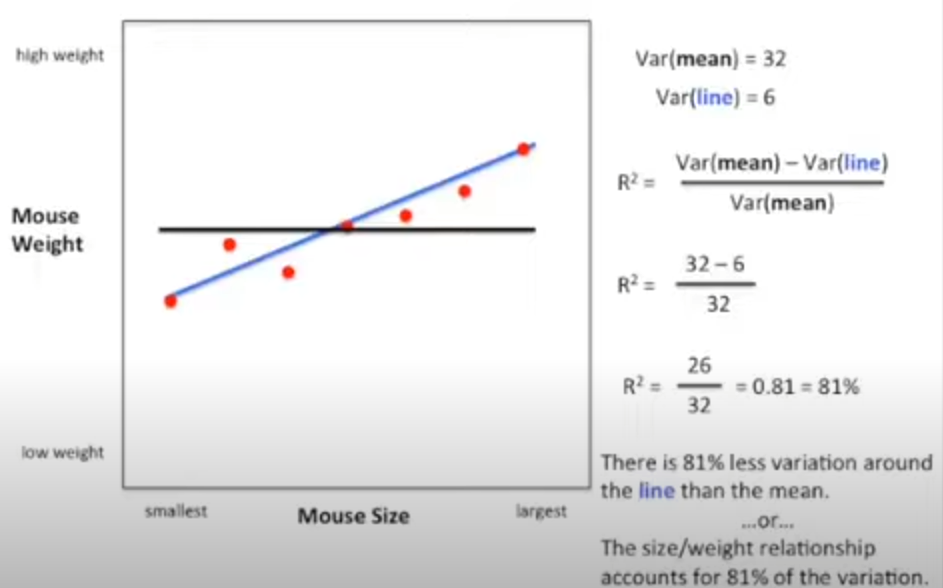
Para evitar o overfitting é só adicionar mais árvores (n\_estimators).

* **Regressão XGBoost -** A biblioteca XGBoost também aceita regressão. Ela constroí uma árvore de decisão simples, e em seguida faz uma melhoria (boosting) adicionando árvores subsequentes. Na prática, funciona muito bem com dados estruturados.
* Regressão LightGBM - LightGBM é uma estrutura de aumento de gradiente rápida, distribuída e de alto desempenho baseada em algoritmos de árvore de decisão. É uma das bibliotecas de aprendizado de máquina mais populares para dados estruturados e é usada para tarefas como regressão, classificação e classificação.

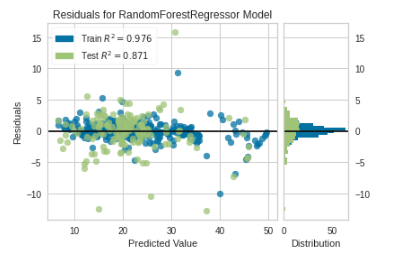
**Métricas e Avaliação de Regressão**

As métricas e avaliação de regressão são usadas para medir o desempenho de um modelo de regressão em prever valores numéricos. Essas métricas podem ser usadas para avaliar o desempenho de um modelo de regressão e comparar o desempenho de diferentes modelos. No entanto, é importante lembrar que nenhuma métrica é perfeita e que a escolha da métrica adequada depende do problema em questão.

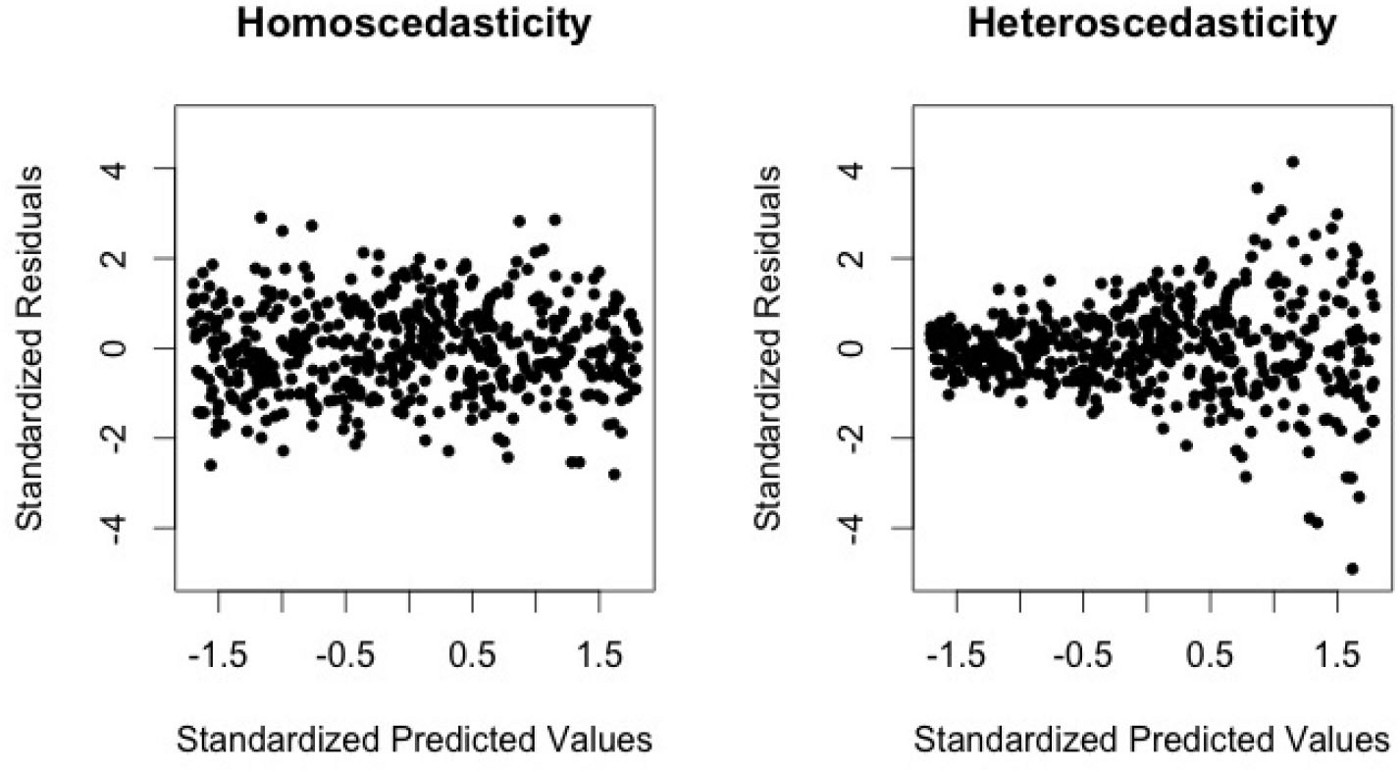
* **Coeficiente de determinação (R²) -** É uma métrica de regressão comum, em geral está entre 0 e 1. Representa o percentual de variância do target com o qual os atributos contribuem. Valores maiores são melhores, mas é difícil de avaliar exclusivamente a partir dessa métrica. A métrica default .scores, calcula ela.



* **Variância explicada -** A pontuação de variação explicada explica a dispersão de erros de um determinado conjunto de dados e a fórmula é escrita da seguinte forma: Aqui, e Var(y) é a variação de erros de previsão e valores reais, respectivamente. Escores próximos a 1,0 são altamente desejados, indicando melhores quadrados dos desvios padrão dos erros. É calculado como a soma dos quadrados das previsões do modelo dividida pela soma total dos quadrados das diferenças entre as previsões e o valor médio da variável alvo.
* **Erro médio Absoluto (MAE) -**  é definida como a média do valor absoluto das diferenças entre as previsões do modelo e os valores observados. Um modelo perfeito teria uma pontuação igual a 0. Se quiser ignorar outlier é uma métrica interessante. Essa medida não afirma se o modelo é bom, mas é usado para comparar modelos, o menor será o melhor. Esse número informa que o erro médio está acima ou abaixo do valor real. A métrica informa que pegamos os erros e tirarmos sua média, o valor resultante estará em torno de x.
* **Erro quadrático médio (MSE) -** é definido como a média do quadrado das diferenças entre as previsões do modelo e os valores observados. Como ele calcula a média do quadrado, erros maiores são penalizados. Se quiser penalizar erros maiores é uma boa métrica a ser usada. Essa medida não afirma se o modelo é bom, mas é usado para comparar modelos, o menor será melhor, que nem o MAE. A métrica informa que se elevarmos os erros ao quadrado e tiramos sua média, o valor resultante estará em torno de x.
* **Erro logarítmico quadrático médio (MSLE) -** Ela é semelhante ao MSE (Mean Squared Error), mas ao invés de calcular o erro quadrático como a diferença entre os valores previstos e observados, ela calcula o erro como a diferença entre os logaritmos dos valores previstos e observados. Seria o log do erro e eleva o quadrado, a média do resultado é x. Se você tiver alvos que apresentam crescimento exponencial (população, ações…) é uma boa métrica.
* **Gráfico de Resíduos -** bons modelos exibiram homocedasticidade, ou seja, que a variância é a mesma para todos os valores do target, independentemente da entrada. Um gráfico de resíduos ajuda a enxergar isso, já que os valores parecerão distribuídos aleatoriamente. Se houver padrões, é sinal de que o modelo ou os dados são problemáticos. Eles também mostram valores discrepantes. Se os pontos em um gráfico residual estiverem aleatoriamente dispersos em torno do eixo horizontal, é uma boa indicação de que o modelo de regressão é apropriado para os dados.



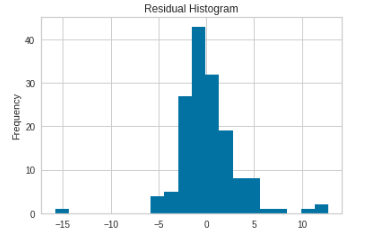
* **Heterocedasticidade -** A heterocedasticidade é um fenômeno estatístico no qual a variância do termo de erro em um modelo de regressão não é constante. Isso significa que a quantidade de erro no modelo pode variar ao longo do intervalo de valores do preditor. A heterocedasticidade pode ser um problema na análise de regressão porque pode levar a estimativas incorretas dos erros padrão dos coeficientes de regressão, o que, por sua vez, pode levar a conclusões incorretas sobre a significância estatística dos coeficientes.

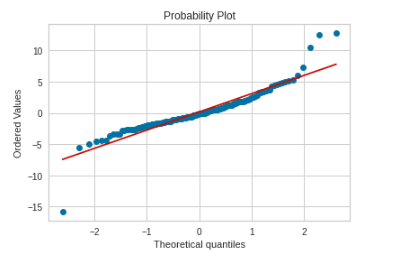


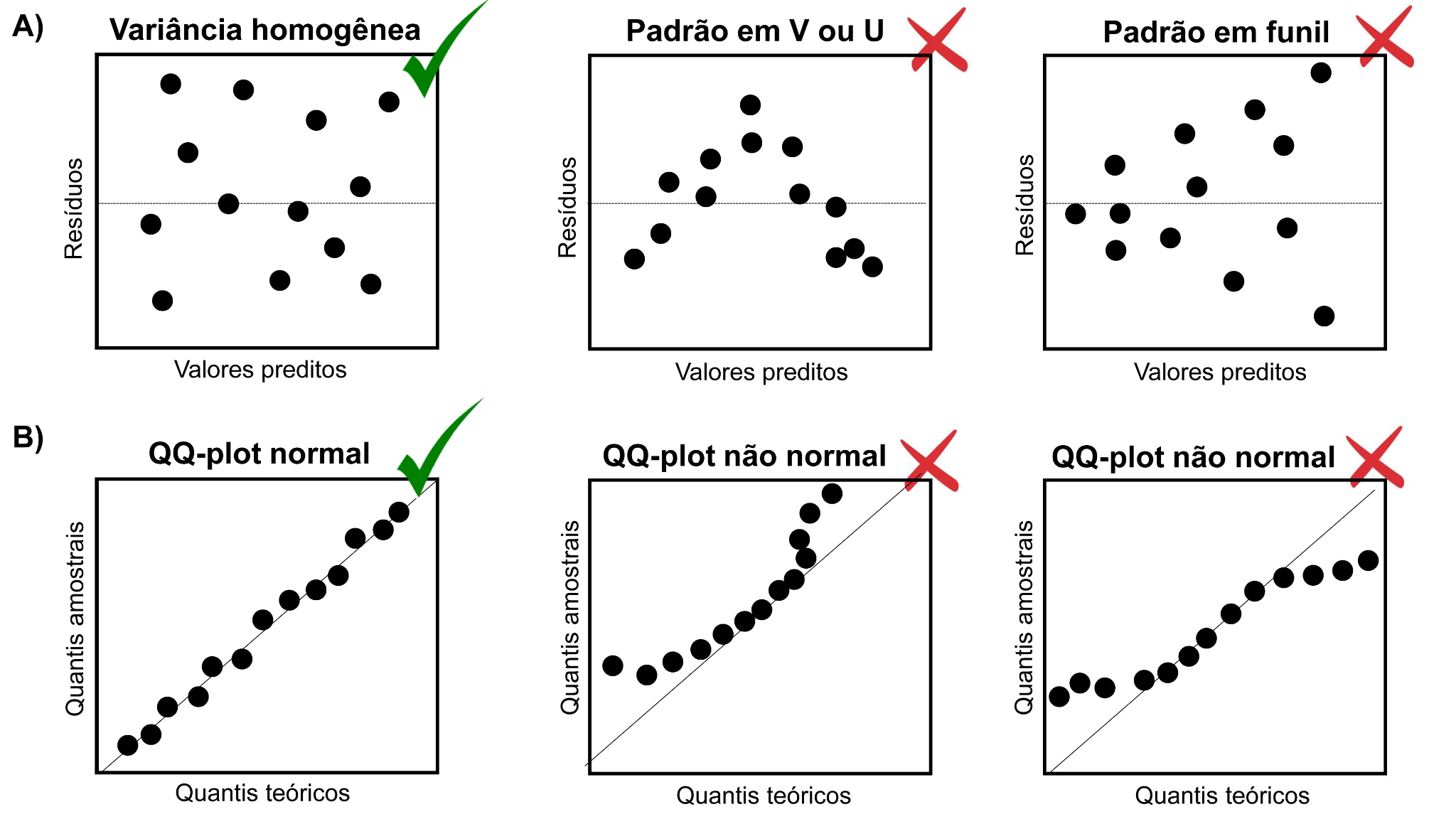
* **Resíduo com distribuição normal -** Em um modelo de regressão, os resíduos são a diferença entre os valores previstos pelo modelo e os valores observados na amostra de dados. Por exemplo, se você estiver tentando prever o preço de uma casa com base em suas características, como tamanho, localização e idade, o modelo de regressão produzirá uma previsão para cada casa na sua amostra de dados. Se o modelo prevê que a casa tem um preço de $ 300.000, mas o preço real da casa é de $ 280.000, então o resíduo para essa casa será de $ 20.000.  
   Em uma situação ideal, os resíduos de um modelo de regressão devem ter uma distribuição normal, ou seja, devem seguir uma curva de sino. Isso significa que a maioria dos resíduos deve estar concentrada em torno de zero, com poucos resíduos muito grandes ou muito pequenos.

A distribuição normal dos resíduos é importante porque muitas técnicas estatísticas assumem que os resíduos são normalmente distribuídos. Isso permite que você use essas técnicas para avaliar a confiabilidade do modelo e fazer previsões precisas. Se os resíduos não tiverem uma distribuição normal, essas técnicas podem não funcionar corretamente.

Se os resíduos formarem uma curva de sino, então é provável que eles tenham uma distribuição normal. Se os resíduos formarem uma forma estranha ou tiverem uma tendência clara, isso pode indicar que eles não são normalmente distribuídos.



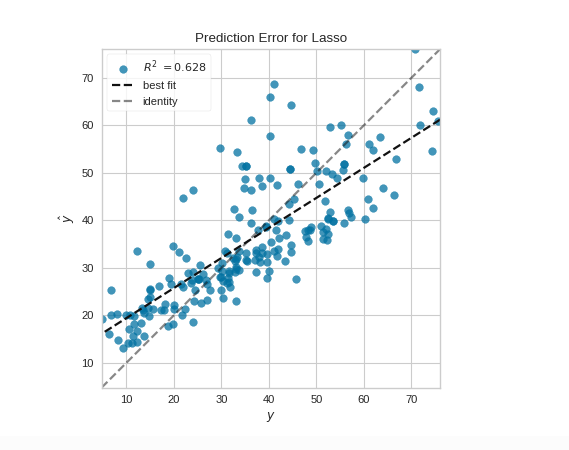




* **Gráfico de erros de predição -** Um gráfico de erros de previsão é um gráfico que mostra a diferença entre os valores previstos pelo modelo de regressão e os valores observados na amostra de dados. Os erros de previsão são geralmente mostrados em um gráfico de linhas ou pontos, com os valores observados no eixo y e os valores previstos no eixo x. (y = real e y circunflexo = previsto)

O gráfico de erros de previsão pode ser usado para avaliar a qualidade do modelo de regressão. Se os erros de previsão forem pequenos e distribuídos de forma uniforme ao longo do gráfico, isso indica que o modelo está se ajustando bem aos dados e está fazendo previsões precisas. Se os erros de previsão forem grandes ou formarem uma tendência clara, isso pode indicar que o modelo não está se ajustando bem aos dados e pode ser necessário ajustar o modelo ou usar um modelo diferente.

Além disso, o gráfico de erros de previsão também pode ser usado para detectar problemas com os dados, como valores atípicos ou outliers. Valores atípicos podem afetar negativamente o desempenho do modelo de regressão e devem ser tratados com cuidado.



**Clustering**

Clustering é um tipo de aprendizado de máquina não supervisionada que se concentra em **dividir um conjunto de dados em grupos (ou clusters) de itens similares.** O objetivo é agrupar itens de maneira que os itens em cada grupo sejam o mais semelhantes possível entre si, enquanto os itens em diferentes grupos sejam o mais diferentes possível. Ele simplesmente inspeciona os atributos e determina quais amostras são semelhantes e pertencem a um cluster.

* **K-Means -** O algoritmo k-means exige que o usuário selecione o número de clusters (k). Então ele escolhe aleatoriamente k centróides e atribui cada amostra a um cluster com base na métrica de distância a partir do centroide. Após a atribuição, os centróides são recalculados com base no centro de todas as amostras atribuídas a um rótulo. Em seguida, a atribuição das amostras aos clusters se repete com base nos novos centróides.

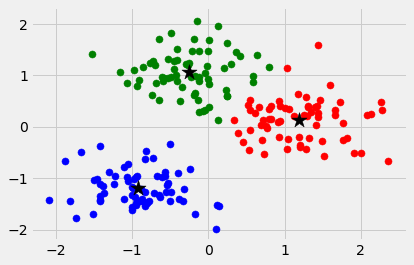
Como o clusteting utiliza métricas de distância para determinar quais amostras são semelhantes, o comportamento poderá mudar dependendo da escala de dados (padronizar é interessante, mas se alguns atributos têm mais importância não padronize).

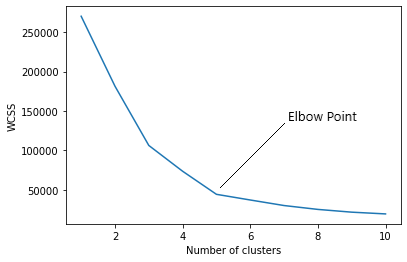
**OBS:.** O método do cotovelo é uma técnica utilizada para escolher o número adequado de clusters ao usar o algoritmo k-means, a ideia é procurar o ponto em que a curva dobra, que será uma boa opção para o número de clusters. Para gráficos sem um cotovelo, podemos usar outras métricas, inspecionar visualmente o clustering e ver se os clusters são visíveis, ou adicionar atributos aos dados e ver se isso ajuda no clustering. Como a técnica da silhueta.

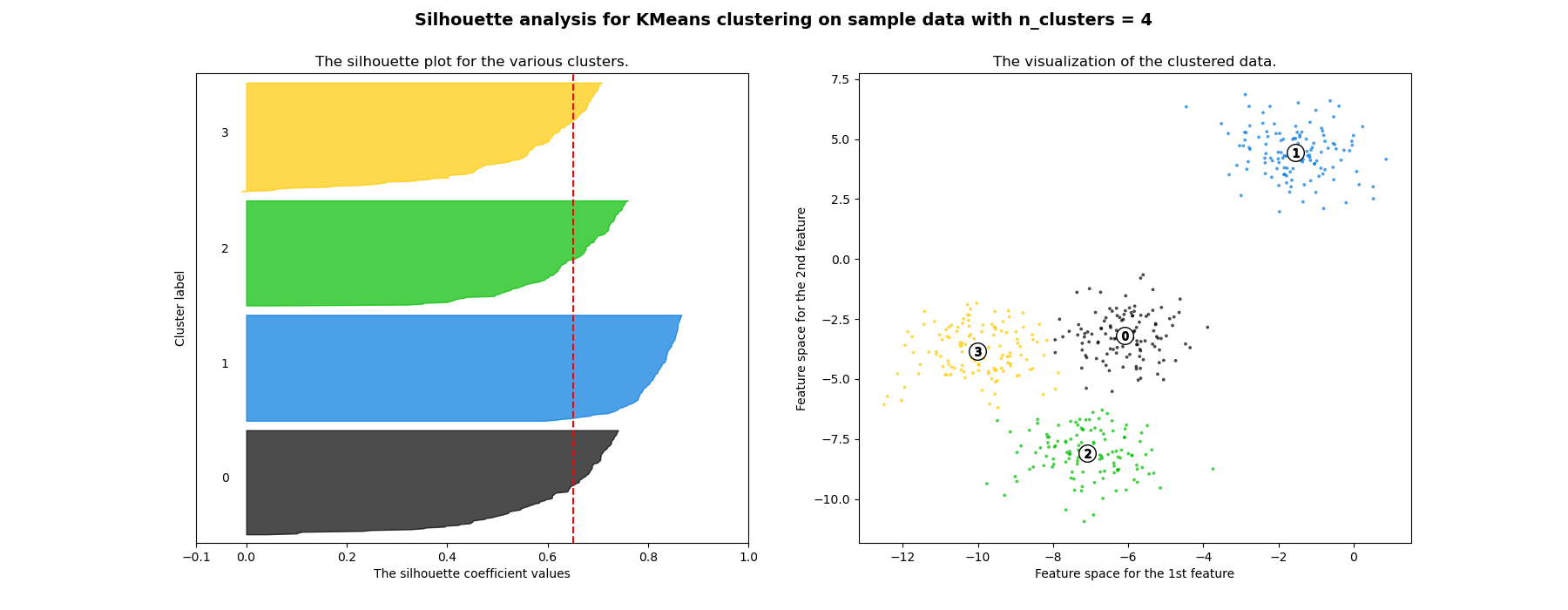
O **coeficiente de silhueta** é um valor entre -1 e 1. Quanto maior a pontuação, melhor. 1 indica clusters compactos e 0 significa clusters sobrepostos. A partir dessa medida, dois clusters nos dão a melhor pontuação.

O **Índice Calinski-Harabasz** é a razão entre a dispersão entre os clusters e a dispersão dentro dos clusters. Uma pontuação mais alta é melhor.

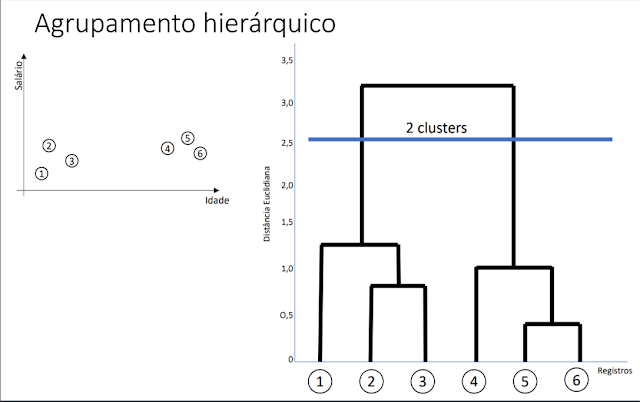
O **índice de Davis-Bouldin** é a similaridade média entre cada cluster e o cluster mais próximo. As pontuações variam de 0 para cima. 0 indica melhor clustering.







* **Clustering (hierarquico) aglomerativo -** é um algoritmo de clustering que constrói uma hierarquia de clusters. Ele começa considerando cada ponto do conjunto de dados como um cluster individual e, em seguida, iterativamente junta os dois clusters mais próximos até que todos os pontos estejam agrupados em um único cluster. Isso resulta em uma árvore de clusters conhecida como **dendrograma** (uma árvore que controla quando os clusters foram criados e qual a métrica das distâncias). Se você tiver muitas amostras, será difícil ler os nós do tipo folha. Para descobrir quantos cluster há nos dados, você poderia “passar” uma linha horizontal no ponto em que ela cruzaria as linhas mais altas.



**Implante o modelo**

Ao usar o módulo **pickle** de Python, podemos fazer a persistência dos modelos e carregá-los. Depois que tivermos um modelo, chamamos o método **.predict** para obter uma classificação ou uma resultado de regressão.

Outros métodos de deploy:

* Flask
* Clipper
* Pipeline
* Cloud Machine Learning Engine do Google