13 - Prática: Predição e a Base de Aprendizado de Máquina (II)

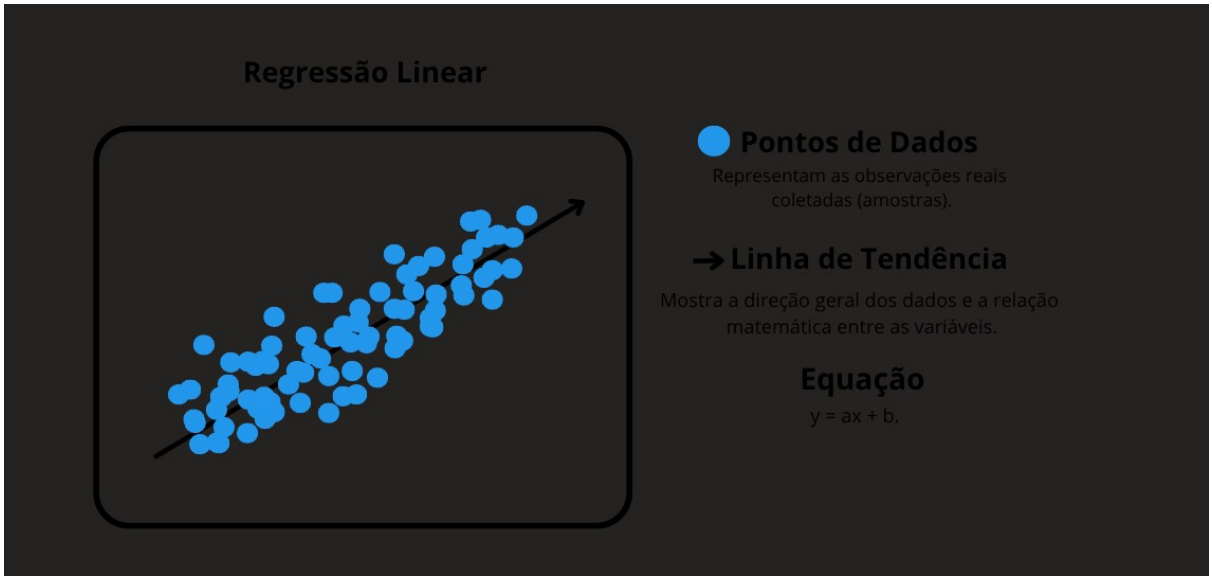
Ronny Gabryel Colatino de Souza

Descrição da atividada

Predictive Models

Linear Regression

Esse card começou falando sobre regressão linear que basicamente é tipo quando você quer prever um valor numérico baseado em outro valor a ideia é bem simples você pega um monte de pontos de dados e tenta traçar a melhor linha reta que passa por eles ou pelo menos perto deles



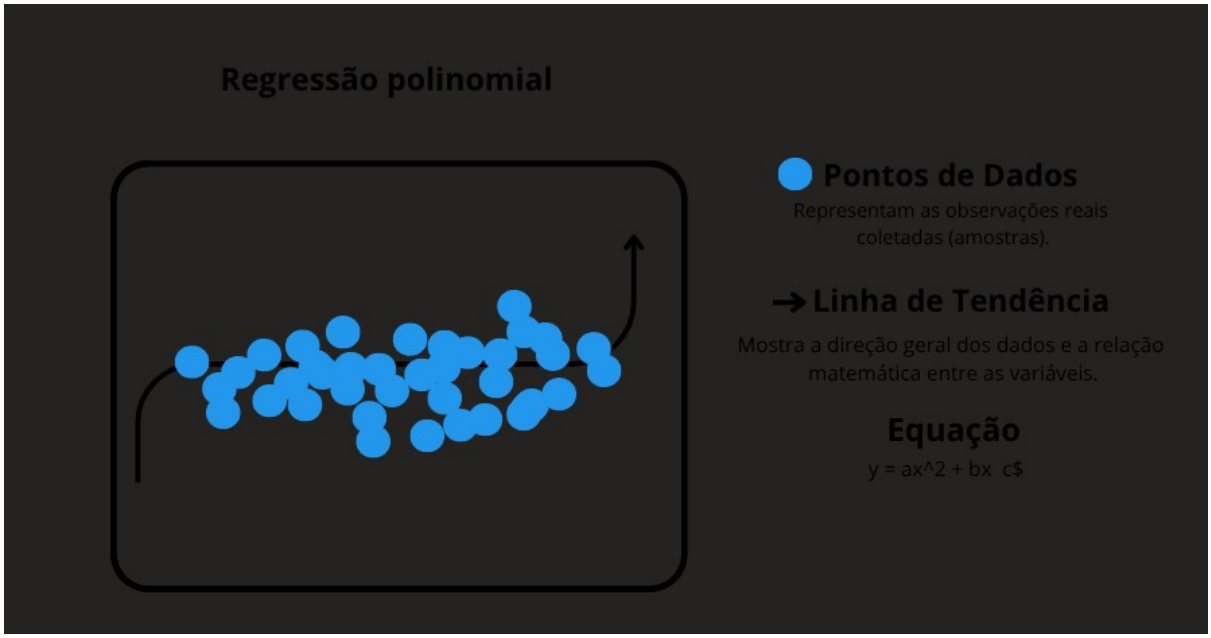
O que achei legal é que a regressão linear usa uma fórmula bem direta tipo y = ax + b onde m é a inclinação da linha e b é onde ela cruza o eixo Y o algoritmo vai ajustando esses valores até achar a linha que melhor se encaixa nos dados geralmente usando uma técnica chamada mínimos quadrados que basicamente minimiza a distância entre os pontos reais e a linha prevista

Na prática quando você implementa isso com scikit-learn é super simples só precisa importar a Regressão Linear, dar fit nos dados de treino e pronto já dá pra fazer previsões o modelo calcula automaticamente os melhores valores pros coeficientes

Uma coisa importante é que regressão linear assume que a relação entre as variáveis é linear mesmo tipo uma linha reta se a relação for mais complexa tipo uma curva aí a regressão linear não vai funcionar tão bem e é aí que entra a regressão polinomial

Polynomial Regression

A regressão polinomial é tipo uma evolução da linear ela serve pra quando os dados não seguem uma linha reta mas sim uma curva tipo imagina que você tá tentando prever o desempenho de um carro baseado na velocidade só que a relação não é linear depois de certo ponto aumentar a velocidade pode diminuir a eficiência por causa da resistência do ar e tal



O que o algoritmo faz é adicionar termos polinomiais na equação ao invés de só y = mx + b você tem tipo y = b + m1x + m2x² + m3x³ e assim vai dependendo do grau do polinômio que você escolher com x² você tem uma parábola com x³ você tem curvas mais complexas

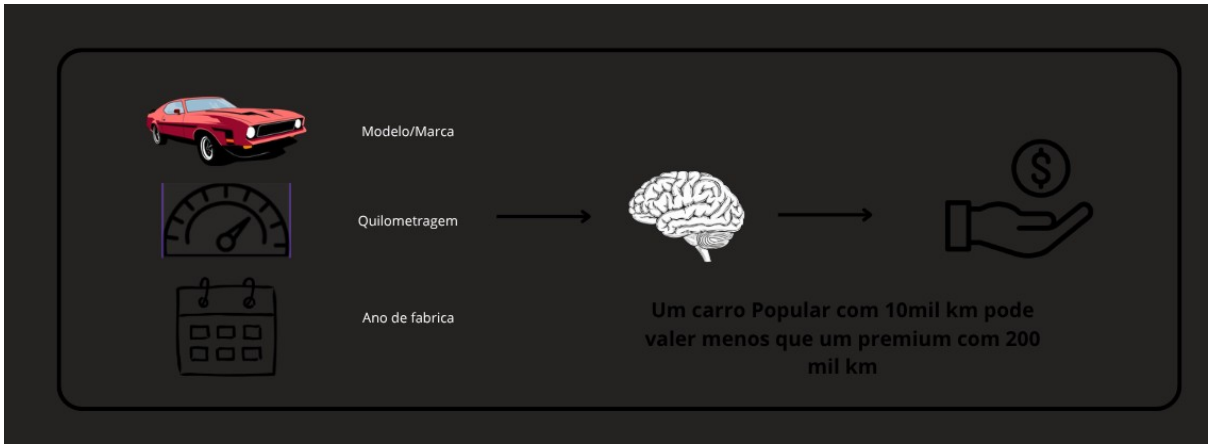
Achei interessante que tecnicamente regressão polinomial ainda é regressão linear só que nos coeficientes não na relação com x tipo você tá criando novas features que são potências de x mas o modelo ainda tá ajustando uma combinação linear dessas features

O desafio aqui é escolher o grau certo do polinômio se você escolhe um grau muito baixo tipo 1 ou 2 pode ser que não capture a complexidade dos dados mas se escolhe um grau muito alto tipo 10 ou 15 você cai no overfitting onde o modelo decora os dados de treino mas não generaliza bem pra dados novos

Multiple Regression

Aqui a coisa fica mais realista porque no mundo real você quase nunca prevê algo baseado em apenas uma variável tipo pra prever o preço de um carro você não olha só a kilometragem você olha o ano de fabricação a marca o modelo o estado de conservação se tem ar condicionado direção hidráulica essas coisas todas

Multiple regression é exatamente isso usar várias features ao mesmo tempo pra fazer a previsão a equação fica tipo y = b + m1x1 + m2x2 + m3x3 onde cada x é uma feature diferente e cada m é o quanto aquela feature influencia no resultado final



O exemplo de predição de preços de carros é perfeito pra isso porque tem um monte de fatores que influenciam no preço um carro com 10 mil km rodados vale muito mais que um com 200 mil km mas se o de 200 mil km for um modelo premium de uma marca boa pode valer mais que um popular com 10 mil km então você precisa considerar tudo junto

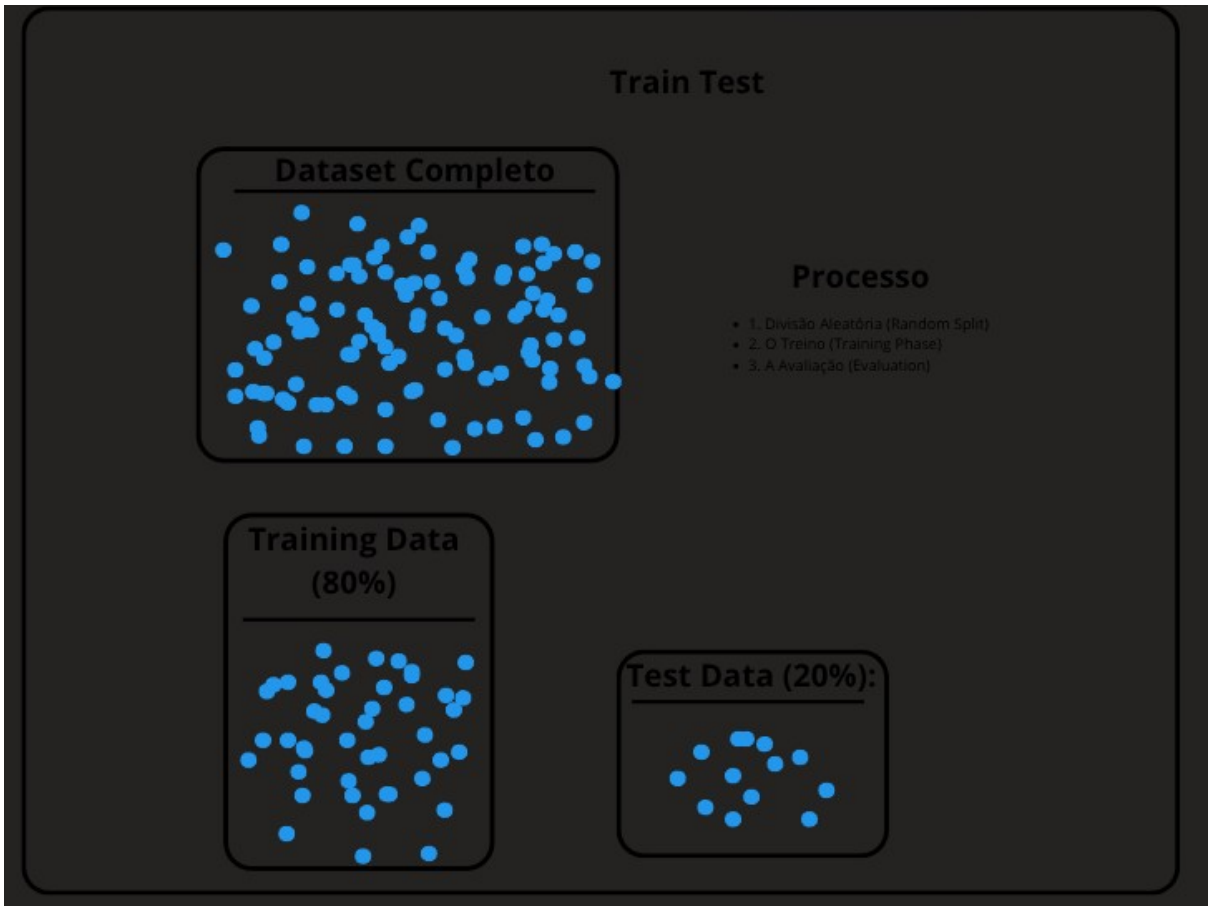
Uma coisa que ficou clara é que nem todas as features têm o mesmo peso tipo provavelmente o ano do carro e a kilometragem influenciam muito mais no preço do que a cor do carro o algoritmo descobre esses pesos automaticamente baseado nos dados de treino

Na prática você carrega um dataset tipo de anúncios de carros com várias colunas de características e uma coluna de preço depois seleciona as features que quer usar remove valores faltantes ou preenche eles de alguma forma inteligente divide em treino e teste treina o modelo e avalia com métricas tipo R^2 que mostra o quanto o modelo explica da variância dos dados ou MAE que é o erro médio absoluto

Machine Learning with Python

Train Test

Esse é tipo um conceito fundamental antes mesmo de ser um algoritmo a ideia é que você não pode treinar seu modelo e testar ele nos mesmos dados porque aí você não sabe se ele realmente aprendeu ou só decorou os dados de treino



O que você faz é dividir seu dataset em duas partes uma parte maior tipo 70% ou 80% pra treinar o modelo e uma parte menor tipo 30% ou 20% pra testar você treina o modelo só com os dados de treino e depois avalia ele com os dados de teste que ele nunca viu antes aí você tem uma ideia real de como ele vai performar com dados novos

Também é importante que a divisão seja aleatória mas estratificada especialmente se você tem classes desbalanceadas tipo se 90% dos seus dados são da classe A e 10% da classe B você quer que essa proporção seja mantida tanto no treino quanto no teste senão pode ser que seus dados de teste não representem bem a distribuição real

Naive Bayes

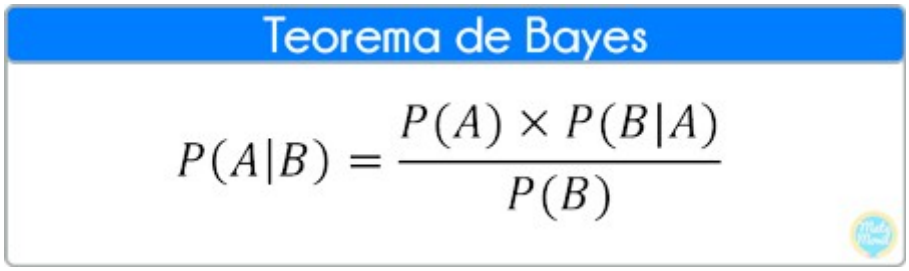


Imagem tirada do site: https://matemovil.com/teorema-de-bayes-ejercicios-resueltos/

Naive Bayes é baseado no teorema de Bayes que é tipo uma fórmula de probabilidade condicional o algoritmo é chamado de naive ingênuo porque assume que todas as features são independentes umas das outras o que geralmente não é verdade mas mesmo assim funciona surpreendentemente bem em vários casos

O uso mais famoso de Naive Bayes é classificação de spam em emails tipo você tem um email novo e quer saber se é spam ou não o algoritmo olha as palavras no email e calcula a probabilidade de ser spam baseado na frequência dessas palavras em emails de spam vs emails normais que ele viu no treino

Por exemplo se a palavra viagra aparece em 80% dos emails de spam mas só em 1% dos emails normais quando o algoritmo vê essa palavra num email novo ele já fica bem confiante que é spam aí ele faz isso pra todas as palavras e combina as probabilidades pra dar o veredito final

O legal do Naive Bayes é que ele é muito rápido de treinar e de fazer previsões porque os cálculos são bem simples só multiplicação de probabilidades então funciona bem até com datasets grandes também funciona bem quando você tem poucas amostras de treino diferente de algoritmos mais complexos que precisam de muitos dados

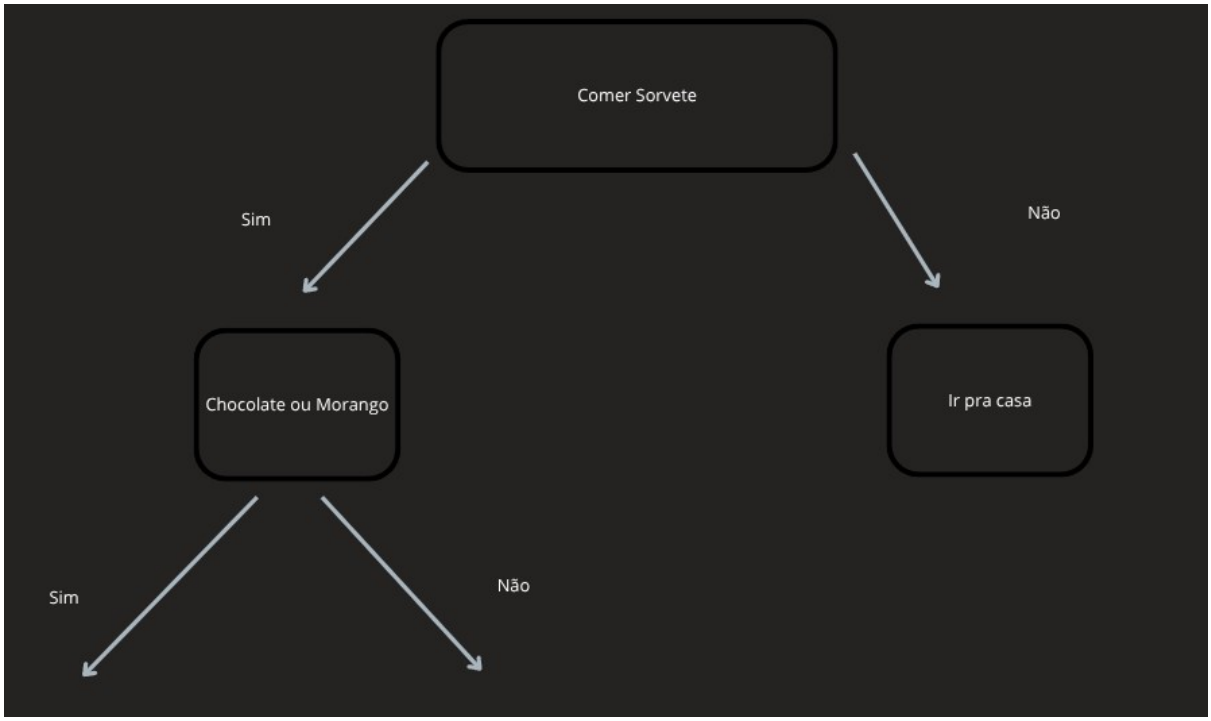
K-Means

K-Means é um algoritmo de clustering que é tipo aprendizado não supervisionado você não tem rótulos nos dados mas quer agrupar coisas parecidas juntas a ideia é dividir seus dados em K grupos onde cada grupo tem pontos que são similares entre si

O jeito que funciona é meio simples você escolhe quantos clusters quer tipo K=3 pra três grupos aí o algoritmo coloca K pontos aleatórios no espaço como centros dos clusters depois vai atribuindo cada ponto de dado pro centro mais próximo dele e recalcula onde fica o centro de cada cluster baseado nos pontos que foram atribuídos pra ele fica repetindo isso até os centros pararem de se mexer

O desafio maior do K-Means é escolher o valor de K tipo quantos clusters você quer tem uma técnica chamada método do cotovelo onde você plota a soma das distâncias dos pontos pros centros pra diferentes valores de K e procura o ponto onde a curva faz tipo um cotovelo mas nem sempre é óbvio onde tá esse ponto

Decision Tree



O Decision Tree ou árvore de decisão é tipo aqueles fluxogramas que a gente faz pra tomar decisões só que automatizado o algoritmo vai fazendo perguntas sobre os dados e dividindo eles em grupos menores até chegar numa conclusão tipo se você tá tentando prever se alguém vai comprar um produto ele pergunta a idade da pessoa é maior que 30 sim vai pra um lado não vai pro outro aí continua fazendo perguntas até classificar

O que achei legal é que árvores de decisão são super fáceis de entender e visualizar você literalmente consegue desenhar a árvore e ver o caminho que o algoritmo tá seguindo pra tomar cada decisão isso é ótimo quando você precisa explicar pro seu chefe ou cliente como o modelo funciona diferente de uma rede neural que é tipo uma caixa preta

O algoritmo usa métricas tipo Gini impurity ou entropy pra decidir qual pergunta fazer em cada ponto basicamente ele quer fazer as perguntas que melhor separam os dados em grupos homogêneos tipo se você pergunta se a pessoa tem mais de 18 anos e 90% das pessoas acima de 18 compram o produto enquanto 90% abaixo de 18 não compram essa é uma pergunta muito boa

O problema é que árvores de decisão sozinhas tendem a fazer overfitting fácil demais tipo elas vão crescendo e crescendo e ficam tão específicas pros dados de treino que não generalizam bem por isso tem técnicas de poda pra limitar o tamanho da árvore ou você pode usar um conjunto de várias árvores que é o que Random Forest faz

XGBoost

XGBoost significa Extreme Gradient Boosting e é tipo uma evolução turbinada de árvores de decisão o conceito de boosting é treinar vários modelos fracos em sequência onde cada modelo novo tenta corrigir os erros do modelo anterior no final você combina todos eles num modelo super forte

A ideia é assim você treina uma primeira árvore de decisão simples ela vai acertar algumas coisas e errar outras aí você treina uma segunda árvore que foca especialmente nos exemplos que a primeira errou depois treina uma terceira que foca nos erros das duas primeiras e assim vai no final você tem tipo um time de árvores onde cada uma é especialista em consertar os erros das outras

Uma parada que achei interessante é que XGBoost dominou competições de machine learning tipo Kaggle por muito tempo tipo se você olhava as soluções vencedoras quase sempre tinha XGBoost lá porque ele simplesmente funciona muito bem em problemas tabulares dados em forma de tabela

Support Vector Machines (SVM)

SVM é um algoritmo de classificação que tenta achar a melhor linha ou hiperplano que separa as classes dos seus dados a ideia é não só separar mas separar com a maior margem possível tipo se você tem pontos vermelhos de um lado e pontos azuis do outro o SVM quer achar a linha que fica o mais longe possível de ambos os grupos

O que diferencia SVM de outros algoritmos é que ele foca nos pontos mais difíceis de classificar os que tão mais perto da fronteira entre as classes esses pontos são chamados de support vectors e são os únicos que realmente importam pra definir a linha divisória os pontos que tão bem no meio de cada classe não influenciam nada

Uma parada interessante é o kernel trick que permite SVM trabalhar com dados que não são linearmente separáveis tipo imagina que você tem pontos vermelhos no meio e pontos azuis em volta não dá pra traçar uma linha reta que separa mas com kernel você pode tipo transformar os dados pra uma dimensão maior onde eles ficam separáveis é meio doido mas funciona

SVM funciona muito bem pra problemas de classificação binária tipo sim ou não spam ou não spam doente ou saudável e também pode ser adaptado pra classificação multi-classe e até pra regressão com SVR Support Vector Regression

Conclusão:

Essse card mas mostrou muito sobre machine learning e varias ferramentas que fazend de tudo cada problema precisa da ferramenta certa tipo Decision Tree quando você precisa explicar o modelo K-Means pra agrupar dados sem rótulo Naive Bayes pra classificação rápida SVM quando os dados são complexos e XGBoost quando você quer performance máxima o mais importante que aprendi é que não adianta ter o algoritmo mais sofisticado se você não entende bem os dados e não divide direito em treino e teste porque senão você acaba com um modelo que decora ao invés de aprender e na hora de usar com dados reais

Referencias:

Vídeo do card: 13 - Prática: Predição e a Base de Aprendizado de Máquina (II)

Uma imagem tirada do site: https://matemovil.com/teorema-de-bayes-ejercicios-resueltos/

Documentaçoes usadas:

Documentação oficial do scikit-learn:
https://scikit-learn.org/stable/modules/linear_model.html

Documentação scikit-learn:
https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LinearRegression.html

Documentação oficial: https://scikit-learn.org/stable/modules/tree.html

Documentação scikit-learn: https://scikit-learn.org/stable/modules/clustering.html#k-means

Documentação oficial: https://scikit-learn.org/stable/modules/naive_bayes.html

Documentação scikit-learn: https://scikit-learn.org/stable/modules/svm.html

Documentação oficial: https://scikit-learn.org/stable/modules/cross_validation.html

Documentação oficial: https://xgboost.readthedocs.io/

Scikit-learn user guide (essencial): https://scikit-learn.org/stable/user_guide.html