

Machine Learning

2022

PROFESSOR: Arthur Rocha
prof.arthur.rocha@gmail.com

Pré-processamento

Os modelos de machine learning em sua maioria só conseguem lidar com variáveis de entrada quantitativas (existem algumas poucas exceções). Por isso, quando estamos falando de aplicar dados aos nossos modelos, devemos realizar algumas etapas de pré-processamento para adequar os dados ao formato correto.

Primeiramente, devemos investigar se o nosso dataset possui dados nulos ou vazios. E havendo, devemos usar alguma estratégia para preencher esses vazios. Algumas possíveis estratégias são: Imputação de Valor Médio (inserir a média do restante da coluna nesse dado vazio), em vez da média ou mediana, também podemos inserir algum valor padrão como o zero ou -1 bem como qualquer outro valor que representa “ausência” de informação ou até mesmo o valor mais frequente da coluna. A estratégia a ser adotada depende muito de cada situação, do tipo de variável e que informação ela carrega. A ideia é inserir dados artificiais que tenham o menor impacto possível de geração de informação irrereal.

Outra estratégia que podemos adotar é a categorização de variáveis contínuas. Nesse tipo de pré-processamento, partimos de uma variável numérica que varia em uma grande faixa de valores e agrupamos em algumas categorias (preferencialmente representadas ainda de forma numérica). Por exemplo, transformar a variável idade em faixa etária, onde poderíamos incluir pessoas de 0 a 16 anos como faixa etária infantil, 17 a 25 como jovem, 26 a 60 como adulto e 60+ como idoso, onde a escala [criança; jovem; adulto; idoso] poderia ser representada pelos seguintes valores numéricos: [0, 1, 2, 3]. Perceba que essa variável é categórica mas não é simplesmente nominal, ela é ordinal.

Outra transformação de pré-processamento muito importante é o Encoding. Existem algumas estratégias, mas vamos abordar duas das mais importantes: Label/Ordinal Encoding e One Hot Encoding. Em geral (mas não sempre), Label/Ordinal Encoding é aplicado a variáveis que desejamos manter a propriedade de ordenação (variável categórica ordinal) e One Hot Encoding, para variáveis puramente nominais.

Label Encoding

Neste tipo de encoding, transformamos **N** valores categóricos em **N** valores numéricos inteiros sequenciais, como por exemplo: [0, 1, 2, 3, ... , N-1], veja o exemplo da figura abaixo.

SAFETY-LEVEL (TEXT)	SAFETY-LEVEL (NUMERICAL)
None	0
Low	1
Medium	2
High	3
Very-High	4

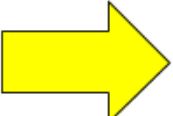
Na lib sklearn, eles diferenciam Label Encoding quando aplicado na variável alvo (target) e Ordinal Encoding quando aplicado às múltiplas variáveis de entrada. Mas o conceito é o mesmo.

Perceba que no exemplo da imagem acima, a variável categórica safety-level é ordinal (mesmo não sendo originalmente numérica), por isso, faz bastante sentido representá-la através de valores inteiros, pois eles preservam essa informação de ordem do menor para o maior.

One Hot Encoding

Neste tipo de encoding, transformamos uma **única** variável com **N** valores categóricos em **N** variáveis (uma variável para cada valor categórico) preenchidos com 0 ou 1. Valor **1**

quando o valor daquela linha é igual àquele representado pela nova coluna e **0** caso contrário. Veja o exemplo da figura abaixo:



Color		Red	Yellow	Green
Red		1	0	0
Red		1	0	0
Yellow		0	1	0
Green		0	0	1
Yellow		0	0	1

Existem algumas variações desse tipo de encoding (como por exemplo no caso de variáveis categóricas binárias, só precisamos de uma única coluna de hot encoding em vez de duas), mas vamos nos limitar a esse caso mais genérico.

O One Hot Encoding tem ainda uma vantagem de poder representar naturalmente valores desconhecidos apenas colocando todos os valores da linha iguais a zero. Para o Label Encoding, poderíamos obter um efeito parecido se colocarmos o valor -1, por exemplo.

Utilizamos o One Hot Encoding normalmente para variáveis que não possuem ordem, embora isso não seja uma regra.

Split do Dataset

O real propósito dos modelos supervisionados de machine learning é de (após a etapa de treinamento) efetivamente realizar previsões/inferências com uma alta de acerto em dados que nunca foram vistos antes.

Para isso, temos que fazer uso de algumas estratégias para evitar descobrir se um modelo está funcionando bem (ou não) posto em produção, já que isso poderia levar a gravíssimos prejuízos, tanto financeiros quanto de credibilidade. Só podemos expor um modelo para a aplicação real quando temos algumas garantias sobre seu desempenho e mesmo assim,

temos que monitorar constantemente como o desempenho do modelo vai se deteriorando com o tempo ou com alguns dados específicos que o modelo não está capturando bem.

Com o objetivo de validar o nosso modelo treinado antes de colocá-lo em produção, fazemos uso da estratégia de **split** (dividir). O objetivo de fazer o **split** do dataset é tentar emular mais ou menos o cenário que teríamos se o modelo já estivesse rodando definitivamente em produção. Então, a estratégia é a seguinte: A partir de todos os dados rotulados que temos disponíveis, dividimos em um conjunto de treinamento propriamente dito e um conjunto de teste. Para alguns tipos de modelos também utilizamos um terceiro split do dataset chamado conjunto de validação.

Algumas abordagens de validação (por exemplo, validação cruzada...) também existem e são mais robustas, porém não vamos abordar aqui (**pesquise depois!**).

Em geral, de maneira mais simples, temos que o conjunto de **treinamento**, que por exemplo, poderíamos ter selecionado como 60% do nosso dataset amostrado aleatoriamente (dependendo do caso, não faríamos amostragem aleatória, como por exemplo, quando trabalhamos com séries temporais). Esse conjunto de treinamento é efetivamente utilizado para **ajustar os parâmetros internos** do nosso modelo, esse processo é efetivamente responsável pelo aprendizado do nosso modelo.

O conjunto de **validação** é utilizado quando precisamos ajustar de maneira mais automática os **hiperparâmetros** dos modelos (hiperparâmetros são conceitos diferentes dos parâmetros). Por exemplo, se estamos falando de Redes Neurais, os parâmetros são os pesos das sinapses entre os neurônios, já os hiperparâmetros são conceitos mais abstratos, como por exemplo, a taxa de aprendizado, a quantidade de camadas de neurônios, a quantidade de épocas de treinamento e outros.

O conjunto de **teste** é responsável por metrificar o desempenho final do nosso modelo. Esse conjunto é que efetivamente está simulando os nossos dados desconhecidos como se o modelo estivesse em produção. A partir desse conjunto podemos calcular as métricas como por exemplo a acurácia (para classificação) ou o erro médio quadrático (para regressão). E esse desempenho final é que utilizamos para representar a qualidade do nosso modelo.

Essa etapa é muito importante e tem que ser tratada com muito cuidado, principalmente quando levando em consideração o risco de “**vazamento de dados**” (data leakage). Vazamento de dados pode ocorrer de diversas maneiras, uma delas é quando temos dados que possuem correlação entre si e por fazermos um split de maneira descuidada, esses dados acabam espalhados pelos conjuntos de treinamento, validação e teste. Isso acaba levando informação “privilegiada” de um conjunto para outro e seria uma situação que não ocorreria no mundo real. Outra maneira de acontecer os vazamentos de dados é quando fazemos pré-processamento de dados (como normalização, ou outros que envolvam computar uma média ou variância) que extraem certo tipo de informação privilegiada da distribuição dos dados inclusive do dataset de teste (que deveria emular o cenário dos dados desconhecidos). Então, para evitar esse tipo de vazamento, devemos realizar o fit pré-processamento dos dados após o split e somente no dataset de treinamento. Depois replicamos o mesmo pré-processamento nos outros conjuntos (validação e teste), mas a construção do pré-processamento é realizada somente baseada nos dados de treinamento.

Um outro tópico muito importante é quando temos dados **desbalanceados**. Isso acontece quando temos muito mais exemplos de uma classe do que de outra. Por exemplo, em dados financeiros sobre fraudes, a quantidade proporcional que temos de dados de fraudadores é muito inferior à quantidade de dados de transações corretas. Se fizéssemos o aprendizado de maneira ingênua diretamente com esses dados desbalanceados, teríamos um modelo enviesado que favorece a classe mais abundante e mesmo assim teria uma acurácia relativamente alta.

Para tentar contornar esse problema, podemos utilizar estratégias de geração de dados sintéticos como o SMOTE (procurem!) e também não fazer um split tão aleatório, como foi sugerido anteriormente, mas tentar equilibrar as distribuições pelos datasets e tentar alocar mais os dados reais no conjunto de testes e dados mais sintéticos no conjunto de treinamento. Essa abordagem, pode inclusive não funcionar muito bem, por isso é preciso que o cientista de dados esteja muito ciente desse contexto para tentar resolver da melhor maneira possível.

Overfitting e Underfitting

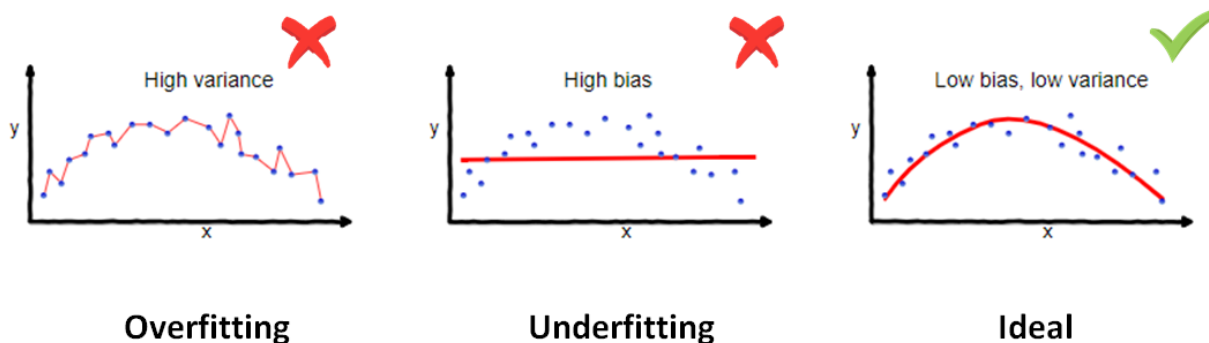
Esse conceito é um dos mais importantes no treinamento de modelos supervisionados de Machine Learning.

De uma forma muito resumida, **Overfitting** é quando temos um sobre ajuste do nosso modelo ao dataset de treinamento. Em outras palavras, também dizemos que o modelo tem uma alta variância. Podemos identificar esse tipo de situação quando o nosso modelo tem um alto desempenho no conjunto de treinamento, mas um desempenho ruim no conjunto de teste/validação. Nesse caso, dizemos que o nosso modelo “decorou”/memorizou os dados de treinamento apenas e por isso, tem uma baixa capacidade de extrapolar o “conhecimento” aprendido em outros dados desconhecidos.

Já na situação de **Underfitting**, dizemos que o modelo tem um alto viés (bias) e se caracteriza por um baixo desempenho inclusive no conjunto de treinamento. Nesse caso, o nosso modelo é muito ruim, ou está com hiperparâmetros ajustados de maneira insatisfatória. Para melhorar, temos que ou mudar completamente o nosso modelo, ou ajustar os hiperparâmetros desse modelo para deixá-lo mais complexo e consequentemente ser capaz de extrair melhor as nuances e “detalhes” dos nossos dados.

Geralmente, a situação “**ótima**” é alcançada como um meio termo entre o Overfitting e o Underfitting. Se o nosso modelo é simples demais, temos Underfitting, se é complexo demais, Overfitting, por isso, buscamos uma situação intermediária. Perceba que para identificarmos essas situações, precisamos obrigatoriamente do split do dataset (feito de maneira correta) em pelo menos treino e teste, mas para alguns modelos precisamos também de conjunto de validação.

Alguns exemplos de maneiras de evitar o Overfitting são: Para **Árvores de Decisão** - Poda da árvore, limite de profundidade da árvore, entre outros. Para **Redes Neurais** - Regularização, parada prematura do treinamento (para isso, podemos utilizar o conjunto de validação para encontrar um ponto “ótimo” máximo de iterações), entre outros.



Explicabilidade de Modelos

Alguns modelos possuem uma capacidade extra além de fazer boas inferências/predições.

Já outros, que costumamos chamar de “caixa-preta”, não possuem uma maneira fácil e direta de extrair regras ou um “passo a passo” racional de como a predição foi alcançada.

Alguns exemplos de modelos com uma boa explicabilidade são: **Regressão Linear**, **Árvores de Decisão** e **KNN**.

Já modelos como **Redes Neurais Artificiais** (Deep Learning ou não), **Ensembles** (principalmente boosting), **SVM** e outros, são mais difíceis de extrair esse tipo de informação. Embora existam algumas estratégias para extrair informações de explicação desses modelos, não é tão simples assim.

Normalmente, esses modelos com maior capacidade de explicabilidade não são os que vão possuir o melhor desempenho de predição, então, costumamos ter um trade-off. Quando precisamos de explicabilidade, costumamos perder desempenho e vice-versa.

É importante notar que o modelo de **Regressão Linear** é um modelo com boa explicabilidade, mas ele só serve para **regressões**. O seu similar para **classificação** (**Regressão Logística**) já não é tão boa para essa tarefa.

Tabela com alguns modelos e aplicações comuns

Abaixo temos uma tabela com exemplos de modelos de machine learning e algumas de suas aplicações mais comuns.

Dados Estruturados X Não-Estruturados	Aplicação	Tipos de modelos mais comuns
Não Estruturados (Imagens e Vídeos)	Processamento de Imagens e Visão Computacional	Redes Neurais Convolucionais (CNN - Deep Learning)
Não Estruturado (Textos e Áudio)	Processamento de Linguagem Natural (NLP)	Transformers, Redes Neurais Recorrentes (RNN: LSTM, GRU - Deep Learning)
Estruturados (Dados Tabulares de Maneira Genérica)	Casos genéricos de regressão ou classificação em dados tabulares	Ensembles (Gradient Boosting, Bagging - Random Forests ...)

Os exemplos na tabela acima não representam regras absolutas de melhor conduta, apenas tendências de modelos que têm apresentado melhores resultados em geral para cada um desses domínios de aplicação. Porém, diversos outros modelos podem ser testados para encontrar quais possuem melhor desempenho para uma dada aplicação. Modelos mais simples como Árvores de Decisão, KNN, Redes Neurais Artificiais Rasas (MLP de poucas camadas), Support Vector Machines, Extreme Learning Machines, Regressão Linear (regressão), Regressão Logística (classificação) e muitos outros podem ser testados em dados estruturados, e talvez obtenham bons resultados, embora Ensembles costumam extrair o melhor a partir de combinações desses modelos citados.

Para aplicações em dados não-estruturados, temos a possibilidade de extrair atributos tabulares a partir desses dados que originalmente não são tabulares (através de um processo chamado feature engineering). Essa abordagem era o padrão antes da explosão dos modelos de deep learning em 2012. Após o uso mais democrático de Deep Learning, esses modelos têm sido usados no que chamamos de aprendizado ponta-a-ponta (end to end) e com isso evitamos (ou diminuímos) a tarefa “manual” de feature engineering.

Então, em Machine Learning, não há essas regras bem definidas, podemos explorar possibilidades, desde que elas se apliquem com bom desempenho a dados desconhecidos a priori pelo modelo (extrapolação).