



## Ciclo 04 - Garantias de Aprendizado

### Fundamentos Machine Learning

A garantia do aprendizado

A estratégia do Treino-Teste

A estratégia da validação holdout

A sequência prática de aprendizado

Cuidados

Generalização e Overfitting

As principais causas do Overfitting e Underfitting

## A garantia do aprendizado

A única forma real de garantir o aprendizado do algoritmo é comparar as classificações com os rótulos reais, no caso das classificações. É exatamente nessa comparação entre as previsões e os valores reais que conseguimos estimar o quanto nosso algoritmo aprendeu.



Algoritmo de classificação → Estamos **classificando (classificação)** uma classe.

Algoritmo de regressão → Estamos **predizendo (predição)** um valor.

No processo de aprendizagem de um algoritmo de classificação, estamos interessados em definir qual classe um determinado datapoint (linha do meu dataset) pertence, no caso de aprendizado supervisionado, é passado a classe real de cada ponto da base de dados em uma coluna chamada de **target**. No processo de aprendizado estamos interessados em descobrir como os outros parâmetros da nossa base afetam a classe que o nosso ponto recebe.

No exemplo ao lado a nossa variável target é **Tipo de Veículo**, e utilizando os outros parâmetros do dataset (*cor, combustível, preço e número de portas*) o algoritmo atribuirá uma classe para cada veículo (linha) da nossa base de dados (coluna **Classificação**).

No processo de aprendizado, o algoritmo mudará os seus parâmetros internos de modo a acertar o maior número de veículos possíveis presentes no nosso conjunto de treino, ou seja, através de processos estatísticos, o nosso algoritmo é capaz de “aprender” o quanto cada feature (variável de treino) influencia na classificação de um veículo.

Cor	Combustível	Preço	Número de Portas	Tipo de Veículo
Azul	Gasolina	R\$ 30.000	4	Carro
Vermelho	Gasolina	R\$ 22.500	2	Carro
Verde	Álcool	R\$ 40.000	4	Carro
Cinza	Diesel	R\$ 300.000	2	Ônibus
...	...	...	...	...
Prata	Gasolina	R\$ 10.000	0	Moto

Porém, nós queremos que os algoritmos sejam capazes de fazer classificações corretas para dados que nunca foram vistos, ou seja, não possuem o rótulo real no dataframe. Para isso, é necessário realizar testes para validar o aprendizado. Nesse teste, o algoritmo recebe um novo conjunto de dados, que não foram usados durante o treinamento e precisa ser capaz de classificar corretamente os dados nunca vistos.



A capacidade de um algoritmo de classificar dados que não foram vistos por ele é chamada de **generalização**.

Como avaliar a capacidade de generalização do algoritmo antes de colocá-lo em Produção?

## A estratégia do Treino-Teste

Existe uma estratégia melhor para medir a capacidade de generalização de um algoritmo de Machine Learning do que colocá-lo diretamente em Produção. Essa estratégia consiste em separar um conjunto de dados em 2 subconjuntos: **Treinamento e Teste**.

A separação dos dados em conjunto de treinamento e teste deve ser feita de maneira aleatória mantendo a proporção original dos exemplos entre as classes. O conjunto de teste deve ser uma representação fiel do ambiente que o algoritmo encontrará quando estiver em produção, em outras palavras, nosso conjunto de teste deverá simular o cenário real no qual o algoritmo treinado será posto a uso. É em produção que o algoritmo realmente encontrará dados nunca vistos anteriormente, por isso é de fundamental importância que nós consigamos avaliar a performance do modelo em um ambiente que simule o cenário em produção, por esse motivo que a avaliação no conjunto de testes é importante.



A avaliação do algoritmo no conjunto treino simulará a performance do algoritmo em um cenário real.

Devemos separar o conjunto de dados original em treinamento e teste aleatoriamente, pois assim garantimos que uma proporção semelhante de cada classe do conjunto original seja mantida nos conjuntos de treinamento e teste. Caso não fizéssemos isso, poderia haver casos em que uma classe não fosse adequadamente representada no dataset de treinamento, de modo que as chances dela ser classificada erroneamente no conjunto de testes aumentaria.

Um exemplo de separação de conjunto de dados está apresentado ao lado. É possível observar que apesar do dataset ter sido dividido em 80% treino e 20% teste, a proporção entre as classes A e B no dataset é a mesma, que por sua vez é a mesma do dataset original.

1. **Conjunto de dados original** (100% dos dados)
  - a. 25 colunas e 10.000 linhas
  - b. 60% classe A e 40% classe B
2. **Conjunto de dados de treino** (80% dos dados)
  - a. 25 colunas e 8.000 linhas
  - b. 60% classe A e 40% classe B
3. **Conjunto de dados de teste** (20% dos dados)
  - a. 25 colunas e 10.000 linhas
  - b. 60% classe A e 40% classe B



As proporções mais comuns de separação são:

1. 80% treinamento e 20% teste
2. 70% treinamento e 30% teste

Após fazermos essa separação e escolhermos qual algoritmo queremos implementar, devemos escolher os valores dos parâmetros do algoritmo (o *número de vizinhos do KNN*, o *parâmetro de regularização em regressão linear*, o *valor da entropia em decision tree*, ...). Para cada valor do parâmetro obteremos uma precisão diferente da performance no conjunto de teste, o que nos leva a tarefa de tentar determinar qual é o valor do parâmetro que produz o menor erro no conjunto de teste, ou seja, uma acurácia maior. Uma maneira de fazermos isso, é repetir o treinamento e avaliar a performance no conjunto de testes para diferentes valores do parâmetro e escolhermos para colocar em produção o modelo cujo o parâmetro produziu o menor erro no conjunto teste. Esse raciocínio parece estar correto em princípio, porém, ele não produzirá a melhor performance quando estiver em produção. Mas por que isso acontece?



O problema é que o algoritmo mediu o erro de generalização várias vezes no conjunto de teste e adaptou o modelo e os parâmetros para produzir o melhor modelo para **esse conjunto de teste em particular**. Isso significa que o modelo provavelmente não funcionará tão bem em novos dados, quando ele estiver em produção.

Para resolvermos esse problema usamos a estratégia de **validação holdout**.

## A estratégia da validação holdout

Uma solução comum para o problema da adaptação do algoritmo, durante o experimento que busca o melhor valor para o parâmetro é a validação holdout. Na validação holdout, você simplesmente separa um terceiro conjunto de dados que será usado exclusivamente para selecionar os parâmetros que trazem a melhor performance ao algoritmo.

Assim como anteriormente, a separação dos dados em conjunto de treinamento e teste deve ser feita de maneira aleatória, mantendo a proporção entre as classes, porém nesse caso fazemos mais um split para validar o modelo. Nesse novo split também devemos selecionar os dados de maneira aleatória, de modo que a proporção entre as classes nesse novo dataset seja a mesma que a do conjunto original. Também é preferível que a quantidade de dados do conjunto de validação seja a mesma que a do conjunto de teste.



As proporções mais comuns de separação são:

1. 70% treinamento, 15% validação e 20% teste
2. 80% treinamento, 10% validação e 10% teste

1. **Conjunto de dados original** (100% dos dados)
  - a. 25 colunas e 10.000 linhas
  - b. 60% classe A e 40% classe B
2. **Conjunto de dados de treino** (70% dos dados)
  - a. 25 colunas e 7.000 linhas
  - b. 60% classe A e 40% classe B
3. **Conjunto de dados de validação** (15% dos dados)
  - a. 25 colunas e 1.500 linhas
  - b. 60% classe A e 40% classe B
4. **Conjunto de dados de teste** (15% dos dados)
  - a. 25 colunas e 1.500 linhas
  - b. 60% classe A e 40% classe B



Devemos escolher a proporção de cada split tendo em mente que cada conjunto desempenha uma função.

- **Conjunto de Treinamento:** Responsável por construir o modelo, nele devemos ter o maior conjunto de dados possível para que o algoritmo consiga ver um número suficiente de dados de cada classe e assim generalizar o processo de classificação.

- **Conjunto de Validação:** Responsável por determinar o melhor valor dos parâmetros do modelo, se tivermos poucos dados no conjunto de validação, corremos o risco de utilizar um modelo não-otimizado.

- **Conjunto de Teste:** Responsável por simular/representar a produção, se tivermos poucos dados no conjunto de teste, corremos o risco de obter um valor que não representa o **erro de generalização**, ou seja, temos o risco de não descobrir com exatidão o quão bom o nosso modelo generalizou a tarefa para a qual ele foi treinado.

!! Em casos de não possuímos um conjunto de dados grande o suficiente, é preferível que uma porcentagem maior dos dados seja designado para teste do que para validação, uma vez que mesmo que os parâmetros dos algoritmos não estejam totalmente otimizados, saber como o modelo performaria em um caso real (ambiente de produção) é considerado mais importante.

## A sequência prática de aprendizado

Ao coletar um conjunto de dados para treinar, validar e testar um algoritmo de Machine Learning, siga os seguintes passos:

1. Defina a proporção de dados para os subconjuntos de treinamento, validação e teste.
2. Separe a porcentagem dos dados de teste logo no início do projeto, antes mesmo de fazer qualquer análise e guarde em algum local.
3. Separe a porcentagem dos dados de validação também no início do projeto, antes mesmo de fazer qualquer análise e guarde em algum local.
4. Com o split de dados de treinamento, realize o treinamento do algoritmo e valide sua performance usando o split dos dados de validação.
5. Teste inúmeros valores para os principais parâmetros do algoritmo que está sendo treinado. Escolha um valor para o parâmetro, treine o algoritmo, valide sua performance e guarde o resultado. Após testar todos os parâmetros disponíveis, defina os melhores valores.
6. Definido os melhores valores para os parâmetros, junte os dados de treinamento com os dados de validação e treine novamente o algoritmo com a união desses dados, usando os melhores parâmetros.
7. O resultado dos passos de 1 a 5, fornecerá um algoritmo treinado com os melhores valores para os parâmetros.
8. Por último, avalie a capacidade de generalização desse modelo final, utilizando os dados de teste separados no começo do projeto, para estimar o erro de generalização.
  - a. Se a performance do algoritmo treinado com os dados de treinamento e validado com os dados de validação, for muito maior que a performance sobre os dados de teste (que ele nunca viu), o algoritmo pode estar "**overfitado**". Por exemplo, a acurácia do algoritmo na validação é de 80% e a acurácia do algoritmo no teste é de 60%. **O algoritmo não conseguiu generalizar.**
  - b. Se a performance do algoritmo treinado com os dados de treinamento e validado com os dados de validação, não for muito diferente que a performance sobre os dados de teste (que ele nunca viu), o algoritmo está generalizando bem.

Por exemplo, a acurácia do algoritmo na validação é de 80% e a acurácia do algoritmo no teste é de 75%. **O algoritmo conseguiu generalizar.**

- i. Caso a condição “b.” seja satisfeita devemos unir os dados de treinamento, validação e teste, retreinar o algoritmo com os melhores valores para os principais parâmetros e publicar em produção.

## Cuidados

1. Se o conjunto de dados de validação é muito pequeno, a avaliação da performance pode ser imprecisa: você pode acabar não selecionando o melhor modelo.
2. Se o conjunto de validação é muito grande, então o conjunto para treinar o algoritmo será menor do que aquele utilizado para avaliar a sua performance. É como se você estivesse treinando o algoritmo para correr 100 *m* rasos e avaliando sua performance em uma maratona de 25 *km*.

Uma forma de resolver esse problema é utilizando uma técnica chamada **Cross-Validation**, onde cada modelo é avaliado com uma parte do conjunto de dados e treinado com o resto. A cada iteração, uma porção dos dados de treinamento se tornam os novos dados para validação enquanto o conjunto usado na validação durante a iteração anterior é incorporado ao dados de treinamento.

Depois da última iteração, também chamada de “**fold**”, é realizada a média dos valores da performance de cada iteração.

### Benefícios da estratégia de Train-Validation-Test

A estratégia de Train-Validation-Test tem diversos benefícios, incluindo:

- Melhorar a precisão do modelo de Machine Learning.
- Ajudar a evitar o overfitting e o underfitting.
- Prevenir o uso de dados de teste para ajustar os parâmetros do modelo.
- Fornecer uma boa estimativa do desempenho do modelo no mundo real.

### Desvantagens da estratégia de Train-Validation-Test

No entanto, a estratégia Train-Validation-Test também possui algumas desvantagens, incluindo:

- Pode ter custos computacionais elevados.
- O conjunto de dados pode não ser suficientemente grande para treinar e validar.
- O resultado pode não ser generalizável para dados não vistos anteriormente.

<https://towardsdatascience.com/what-is-stratified-cross-validation-in-machine-learning-8844f3e7ae8e>

## Generalização e Overfitting


A generalização é algo que o ser humano faz muito frequentemente, ele usa um único exemplo ou poucos exemplos e define o comportamento geral de um seguimento. Infelizmente, os algoritmos também podem cair nessa armadilha se não tomarmos cuidados. Em Machine Learning, esse problema é chamado de “**overfitting**”.

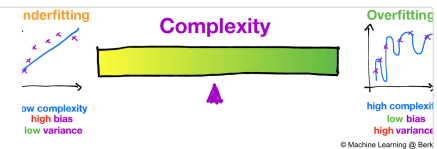
Overfitting significa que o modelo tem uma boa performance nos dados de treinamento e validação, mas não consegue manter a mesma performance para dados não vistos, ou seja, não consegue generalizar. Ele comete muitos erros para exemplos não vistos.

## As principais causas do Overfitting e Underfitting

Machine Learning Crash Course: Part 4—The Bias-Variance Dilemma

13 Jul 2017 | Daniel Geng and Shannon Shih

 <https://medium.com/@ml.at.berkeley/machine-learning-crash-course-part-4-the-bias-variance-dilemma-a94e60ec1d3>



### As principais causas do Overfitting

1. **Complexidade do Modelo:** modelos muito complexos podem se ajustar demais aos dados de treinamento e ter um mal desempenho em novos dados.
2. **Conjunto de Dados Pequenos:** modelos podem aprender padrões aleatórios nos dados de treinamento quando o conjunto de dados é muito pequeno.
3. **Treinamento Excessivo:** quando o modelo é treinado por muitas épocas, pode se ajustar demais aos dados de treinamento e não generalizar bem.
4. **Vazamento:** informações dos dados de teste vazam para o modelo durante o treinamento, o que pode levar a um ajuste excessivo aos dados de teste.

### Soluções para o Overfitting

1. **Validação cruzada:** dividir o conjunto de dados em conjuntos de treinamento e validação para avaliar o desempenho do modelo em novos dados.
2. **Regularização:** adicionar uma penalidade à função de perda durante o treinamento para evitar que o modelo se ajuste demais aos dados de treinamento.
3. **Early stopping:** interromper o treinamento do modelo quando o desempenho no conjunto de validação começa a piorar, evitando que o modelo se ajuste demais aos dados de treinamento.
4. **Aumentar o tamanho do conjunto de dados:** ajuda o modelo a aprender padrões mais relevantes e evitar o ajuste excessivo aos dados de treinamento.

### As principais causas do Underfitting

1. **Modelo muito simples:** modelos muito simples não conseguem capturar a complexidade dos dados e podem ter um desempenho inferior.
2. **Dados insuficientes:** quando o conjunto de dados é muito pequeno, o modelo pode não ter informações suficientes para aprender padrões relevantes.
3. **Falta de treinamento suficiente:** o modelo pode não ter sido treinado por tempo suficiente para aprender padrões relevantes nos dados.
4. **Modelo mal projetado:** um modelo mal projetado pode ter dificuldade em capturar as nuances e, portanto, ter um desempenho inferior.

### Soluções para o Underfitting

1. **Aumentar a complexidade do modelo:** aumentar a complexidade do modelo pode ajudar a capturar a complexidade dos dados e melhorar o desempenho.
2. **Adicionar mais recursos:** adicionar mais recursos ao conjunto de dados pode fornecer mais informações adicionais para o modelo aprender.
3. **Aumentar o tempo de treinamento:** treinar o modelo por mais tempo pode ajudar a capturar padrões mais complexos nos dados.
4. **Alterar o design do modelo:** alterar o design do modelo pode ajudar a capturar nuances adicionais nos dados e melhorar o desempenho.

<https://towardsdatascience.com/learning-curve-to-identify-overfitting-underfitting-problems-133177f38df5>