Seleção de Modelos de Aprendizado de Máquina

Alberto de Sá Cavalcanti de Albuquerque

2020

Seleção de Modelos de Aprendizado de Máquina

Alberto de Sá Cavalcanti de Albuquerque

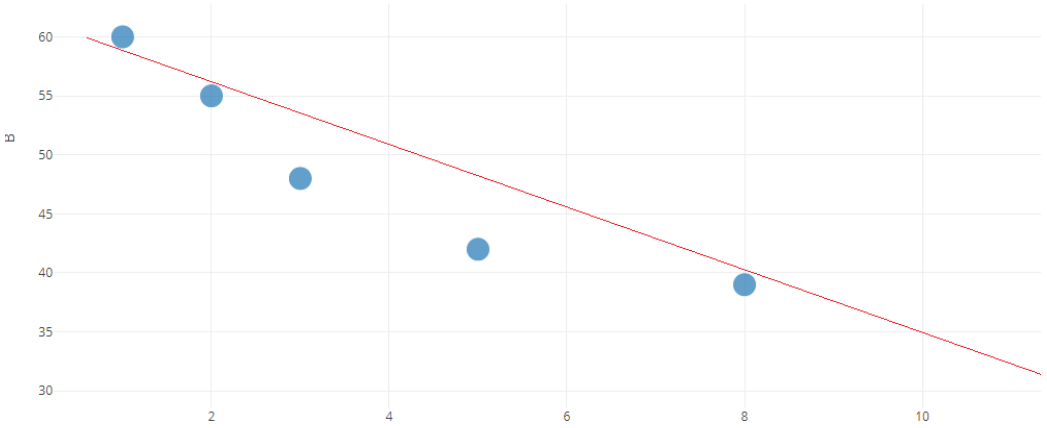
Sumário

Capítulo 1. Métricas para Regressão

Em Aprendizado de Máquina, uma regressão é utilizada para criar um algoritmo que faça previsões com valores numéricos. A intenção é estimar relacionamentos entre variáveis. Por exemplo: “Se o meu preço de televisões sobe até determinado ponto, meu lucro sobe. Depois disso, ele começa a cair”.

Regressões são uma forma de analisar dados passados para tentar compreender o futuro. Os valores analisados devem ser sempre numéricos e contínuos. “Como meu lucro se comporta na medida em que meu preço aumenta?” é uma pergunta que uma regressão responde bem. “Como cada cor de carro impacta nas minhas vendas” já não é: nesse caso, temos categorias (cores de carro), e não valores numéricos.

Figura 1 - Exemplo de conjunto de dados com regressão.



Um algoritmo de regressão busca minimizar uma determinada métrica de erro para o conjunto de dados analisado. Isto é: dado o seu conjunto de dados, o algoritmo busca traçar uma reta que melhor “se encaixe” nesses pontos, dada uma determinada métrica de erro. Estudamos aqui quatro métricas importantes:

- Erro médio absoluto.
- Erro médio quadrático.
- Erro mediano absoluto.

- Coeficiente de determinação.

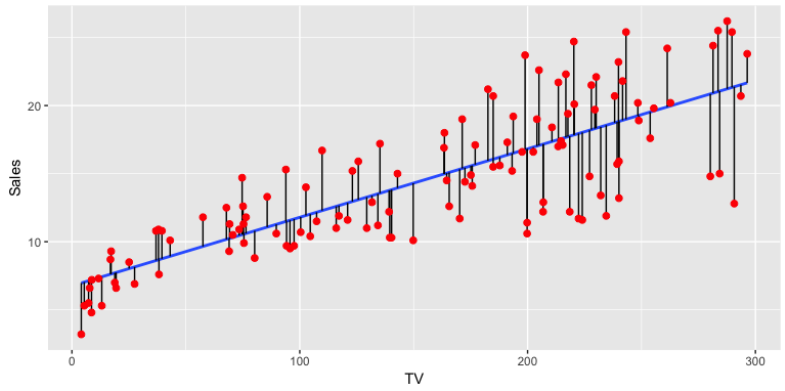
Seguindo essa lógica, é instrutivo imaginar que cada ponto do gráfico é como um ímã que repele a reta: o objetivo do algoritmo é encontrar a reta que se equilibre melhor entre todos aqueles ímãs. A força com que cada ponto repele a reta varia de acordo com a métrica estudada.

Nas demais seções deste capítulo, vemos em detalhes cada uma dessas métricas. As três primeiras, erro médio absoluto, quadrático e erro mediano, são variações do mesmo conceito: a análise da distância dos pontos para a reta traçada para a regressão. O coeficiente de determinação tem uma ideia um pouco diferente, que será vista mais adiante. Por fim, abordaremos alguns dos cuidados que o cientista de dados precisa ter em mente ao utilizar regressões para entender o seu problema.

Erro médio absoluto

O erro médio absoluto é a métrica de qualidade mais básica para se analisar uma regressão. Toda regressão se constrói seguindo o mesmo princípio: dado um conjunto de pontos, encaixe sua reta entre eles de forma a melhorar o mais possível a sua métrica de qualidade. Em inglês, é chamado de *Mean Absolute Error.*

Figura 2 - Regressão linear com as distâncias absolutas de cada ponto a ela.



Fonte: http://uc-r.github.io/public/images/analytics/regression/sq.errors-1.png.

Observando a Figura 2, veja que cada ponto de dado observado está distante um certo tanto da regressão. Essa distância é o erro, para aquele ponto, daquela regressão. **O erro médio absoluto é a soma de todos esses erros dividido pelo número de pontos**.

Figura 3 - Equação para o erro médio absoluto.

$$\frac{1}{\#\text{amostras}} * \sum_{0}^{\#\text{amostras}-1} |\text{previsto} - \text{real}|$$

O significado disto é direto: quanto mais longe um ponto está da reta que faz a regressão dos dados, maior aquele erro. **Na medida em que a distância do ponto à regressão aumenta, o erro aumenta de forma linear.** Uma regressão feita sob essa métrica tentará se aproximar de todos os pontos igualmente, da melhor forma possível.

Erro médio quadrático

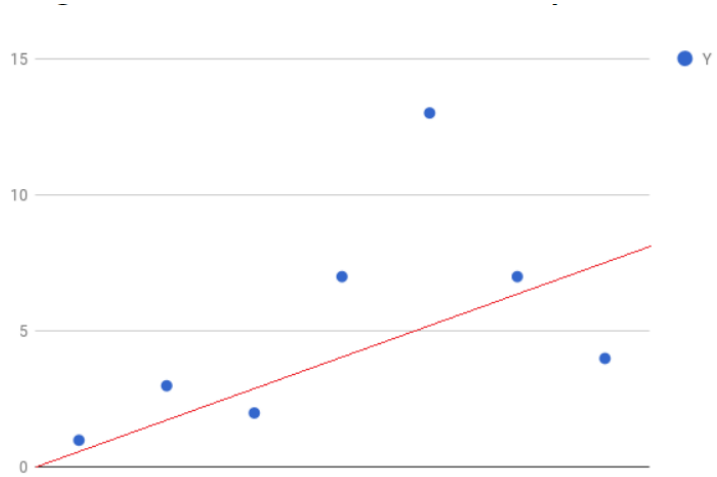
O erro médio quadrático tem como base o mesmo conceito do erro médio absoluto: ele utiliza as distâncias dos pontos à regressão para ser calculado. A diferença é que o que é levado em consideração não é a distância absoluta do ponto à regressão, mas essa distância **elevada ao quadrado.** Em inglês, é chamado de *Mean Quadratic Error.*

Figura 4 - Equação para o erro médio quadrático.

$$\frac{1}{\#\text{amostras}} * \sum_{0}^{\#\text{amostras}-1} (\text{previsto} - \text{real})^2$$

Ou seja: ao se considerar o erro da regressão para aquele ponto, toma-se a distância entre ambos e a eleva ao quadrado, em todos os casos. Na prática, no entanto, o que isso significa? Observe a Figura 5.

Figura 5 - Exemplo de erro quadrático contra erro absoluto.



Erro Absolutos	Erros Quadráticos
0.1	0.01
1	1
0.5	0.25
3	9
9	81
0.2	0.04
3	9

Observe que os erros de valor inferior a zero se tornam **menores** quando elevados ao quadrado. Os erros de valor superior a zero, no entanto, crescem, em proporção **quadrática.** Um erro de 3 se torna 9, 6 unidades maior. Um erro de 9 se torna 81, 72 unidades maior.

Podemos ver, portanto, que o erro médio quadrático, ao elevar ao quadrado a distância entre o ponto e a regressão para fazer sua soma, penaliza de forma cada vez maior um ponto distante da reta. Ter um ponto muito longe da regressão é muito pior que ter dois pontos a média distância. É preferível ter dois pontos a uma distância de 4 (dois erros de 16, 32 de erro no total) a ter um ponto a uma distância de 1 e outro a distância de 7 (erro de 1 e erro de 49, 50 de erro no total).

Isso se reflete na forma como a regressão é montada. Ela vai tentar sempre ficar **em igual distância de todos os pontos do gráfico.** Um ponto muito mais longe que os demais é algo que a regressão evitará ao máximo ter.

No entanto: num conjunto de dados quase todo uniforme, mas com alguns poucos pontos anômalos, ao tentar deixar todos em igual distância, a regressão pode acabar distorcida. Essa é a métrica costumeiramente mais usada para regressões.

Erro mediano absoluto

A mediana de um conjunto de valores é o valor que, dado que os valores foram colocados em ordem crescente, ficou no meio da sequência. Por exemplo: pense que listamos todos os erros absolutos de uma regressão conforme abaixo:

- Erros absolutos = [0.1, 1.0, 0.5, 3.0, 9.0, 0.2, 3.0]
- Erros absolutos em ordem = [0.1, 0.2, 0.5, **1.0**, 3.0, 3.0, 9.0]

Se calcularmos a média dos erros absolutos, isto é, o nosso erro médio absoluto, obtemos o valor de 2.4. O erro mediano absoluto, no entanto, é 1.0, conforme mostra a sequência ordenada de forma crescente. O erro mediano absoluto é a mediana dos erros absolutos da regressão.

O que isso significa para regressões formadas tendo essa métrica como base? Veja que, ainda que o último erro tivesse um valor muito maior que os demais, como 1000, o erro mediano absoluto continuaria sendo 1.0, e a regressão resultante não seria diferente.

Ou seja: uma regressão feita com o **erro quadrático absoluto** tenta se encaixar em todos os pontos da forma mais homogênea possível. A regressão do **erro mediano absoluto** capta a tendência geral dos dados, ignorando pontos muito distantes, tomando-os como anomalias.

Vemos nisso vantagens e desvantagens no uso de cada métrica para construir a sua regressão: o erro mediano absoluto é robusto a anomalias nos dados, mas pode perder precisão se os dados variarem muito. O quadrático é o contrário: ele se encaixa melhor conforme os dados variam, mas uma anomalia pode deixa-lo

distorcido. O erro médio absoluto é quase que um meio termo: ele não se deixa distorcer tanto por anomalias, e não é fraco para dados que variam muito.

Cada uma dessas três métricas têm vantagens e desvantagens claras, e a escolha de qual deverá ser usada é fortemente dependente de como seu conjunto de dados é. Projetar os pontos num gráfico e observá-lo pode dar boas pistas de qual dessas métricas de regressão se comportará melhor ali.

Coeficiente de Determinação

As três outras métricas estudadas nesse capítulo – erro médio absoluto, erro médio quadrático e erro mediano absoluto – fornecem um número absoluto que representa o quão “afastados” os dados estão, na média, da sua regressão. A métrica que orienta a escolha da regressão determina o tipo de regressão que você terá – uma regressão que ignora anomalias nos dados, que busca deixar os erros mais “homogêneos”, e assim por diante.

O Coeficiente de Determinação se propõe a dizer se duas variáveis têm correlação forte ou fraca. Isto é: se uma das variáveis varia, a outra varia também? Sua saída é uma porcentagem, de 0 a 100%, que indica correlação entre as duas variáveis. Uma correlação de 100% significa que, para os seus dados, sempre que uma das variáveis varia, a outra varia também. Uma correlação de 0% indica que as duas variáveis não parecem estar associadas.

Como exemplo, suponhamos que você queira medir se o aumento da incidência de um produto nos recomendados de um e-commerce impacta em suas vendas. Você anota, então, a quantidade de vezes que o produto foi exibido como recomendado naquele dia, por exemplo, ao longo de meses, e depois compara com as vendas do produto naqueles dias. Seu valor de R^2 dá 80%. Isso significa que 80% da variação nas vendas do produto foi acompanhada por variação na exibição nos recomendados. Há relação entre as duas coisas. É um indicativo de que as duas coisas estão relacionadas.

Figura 6 - Equação para cálculo do Coeficiente de Determinação.

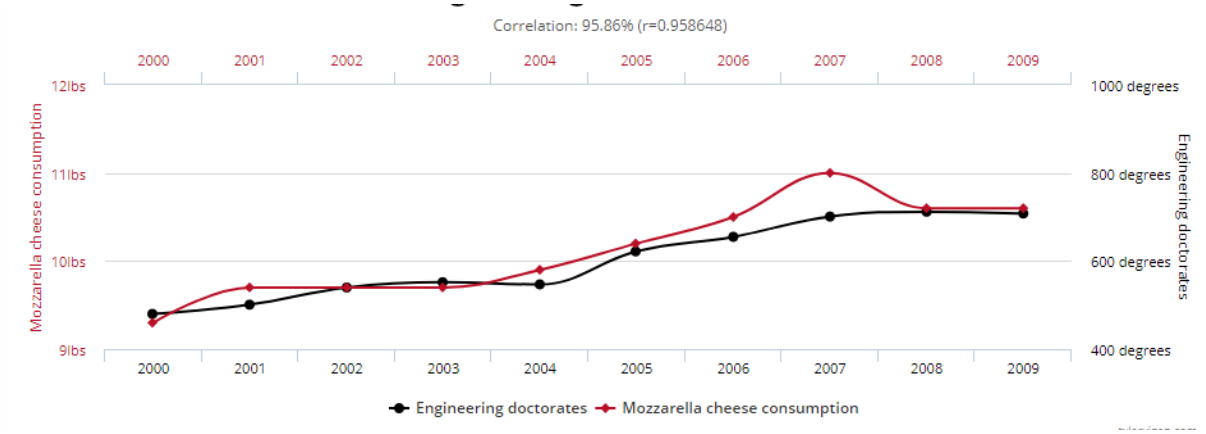
$$1 - \frac{\sum_{0}^{\#\text{amostras - 1}} (\text{previsto - dado real})^2}{\sum_{0}^{\#\text{amostras - 1}} (\text{previsto - media dos dados reais})^2}$$

O cálculo do coeficiente de determinação se dá dividindo a soma dos erros quadráticos da regressão com a soma dos erros quadráticos dos dados em relação à sua média. Em termos simples, divida a soma dos erros da regressão pela soma da variação dos valores de sua variável em relação à sua média. **Essa métrica só pode ser usada para regressões lineares.**

O coeficiente de determinação, em resumo, faz o mesmo que as métricas anteriores: ele diz o quanto da variação naqueles dados pode ser explicada pela sua regressão. Sua utilidade se dá por ser uma métrica dada em porcentagem – é fácil de entender e independe da métrica associada aos seus valores. Um erro médio quadrático de 200, por exemplo, pode ser muito ou pouco a depender do que você está medindo. Um coeficiente de determinação de 80%, no entanto, sempre indica forte correlação.

É preciso que fique claro, no entanto, como será discutido na seção seguinte, que **correlação não é causalidade.** O coeficiente de determinação diz que duas variáveis, para aquele conjunto de dados, estão relacionadas, mas cabe ao cientista de dados entender o problema em estudo para avaliar se essa correlação existe, ou se ela é só um acaso. Em outras palavras: não é porque a maior parte dos assassinados nos EUA estão usando camisetas (uma correlação forte, com coeficiente de determinação alto) que camisetas geram assassinatos.

Figura 7 - Gráfico comparando o consumo per capita de queijo mozzarella com o número de doutorados em engenharia civil dados ao longo dos anos, nos Estados Unidos. Correlação não é causalidade.



Fonte: http://www.tylervigen.com/spurious-correlations.

Cuidados com a Regressão

Ao se usar regressões em situações problema de Aprendizado de Máquina, é preciso ter em mente o que essas ferramentas são e o que são capazes de fazer. Mais do que saber usar a ferramenta, é preciso entende-la bem. Ter alguns cuidados é fundamental para não ser induzido ao erro:

- **Correlação não é causalidade:** conforme o livro *Spontaneous Correlations*, de Tyler Virgen (Correlações espontâneas, em tradução livre. Indisponível em português) mostra muito bem, duas variáveis quaisquer podem estar fortemente correlacionadas sem que isso indique que uma leva à variação da outra. **Estar correlacionado não significa estar relacionado.** Uma correlação entre variáveis pode, antes de mais nada, ser uma coincidência. Descobrir uma correlação entre variáveis usando uma regressão é uma pista, não uma solução. É preciso compreender o problema antes de afirmar que aquela correlação implica em causalidade.

- **A regressão não é o objetivo:** todo cientista de dados tem um problema em mãos para resolver. Esse problema, geralmente, envolve entregar valor a alguém: a um cliente, uma organização, um usuário, o que for. Esse valor vem das conclusões que os dados te permitem ter. Por isso, é preciso entender: uma boa regressão, com métricas de qualidade altas, não é por si só valiosa. É preciso saber no que esse modelo implica, e como isso pode se tornar valor.

- **Cuidado com o overfitting:** em aprendizado de máquina, faz-se a análise de um conjunto de dados limitado na tentativa de encontrar uma regra geral para esses dados. Isso significa que **um modelo que se adeque perfeitamente ao seu conjunto de dados pode não se adequar à situações reais.** Em outras palavras: cuidado com o overfitting ao fazer regressões lineares. Uma regressão com um erro pequeno, mas robusta à análise de novos dados daquele domínio, pode ser preferível a uma regressão perfeita, mas que cai por terra diante da entrada de dados novos.

- **Saiba quando dar um passo atrás:** um modelo, em aprendizado de máquina, será sempre tão bom quanto seus dados o permitem ser. Em certas situações, pode ser que a dificuldade se chegar a bons resultados não advenha do seu modelo, mas dos seus dados: eles podem ser insuficientes, podem ter viés, podem estar desbalanceados. Saber quando “voltar atrás” e voltar sua análise para os dados, e não para o modelo, é fundamental.

Exemplo - Regressão para previsão de diabetes

Para um exemplo de como implementar uma regressão, temos o código abaixo, que gera uma regressão para prever, baseado em determinados atributos, uma métrica quantitativa da progressão da diabetes após um ano em pacientes diferentes. Neste link é possível ler mais sobre esse conjunto de dados: https://scikit-learn.org/stable/datasets/index.html#diabetes-dataset.

```python
import matplotlib.pyplot as plt
import numpy as np
from sklearn import datasets, linear_model
from sklearn.metrics import mean_squared_error, median_absolute_error
from sklearn.model_selection import train_test_split

# Carregue o dataset
X, y = datasets.load_diabetes(return_X_y=True)

# Separamos um atributo para usar
X = X[:, np.newaxis, 2]

# Separe dados pra testar
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)

# Instancie
regr = linear_model.LinearRegression()

# Treine o modelo
regr.fit(X_train, y_train)

# Teste o modelo treinado
y_pred = regr.predict(X_test)

# Métricas de qualidade
print('Erro médio quadrático: %.2f'
      % mean_squared_error(y_test, y_pred))

print('Erro mediano absoluto: %.2f'
      % median_absolute_error(y_test, y_pred))

# Gráfico
plt.scatter(X_test, y_test,  color='black')
plt.plot(X_test, y_pred, color='blue', linewidth=3)
plt.xticks(())
plt.yticks(())
plt.show()
```

Os comentários do código são auto-explicativos - e a documentação do scikit-learn é a maior aliada na hora de se escrever código para gerar modelos em Aprendizado de Máquina. Nesse código nós carregamos o dataset, separamos os dados em dados para treino do modelo da regressão e dados de teste, geramos o nosso modelo, medimos sua performance e geramos um gráfico da regressão ao final de tudo.

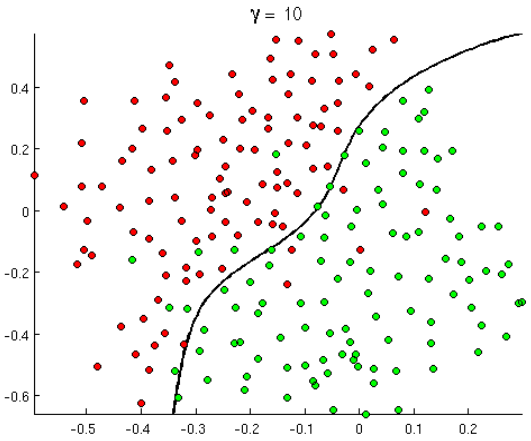
Capítulo 2. Métricas para Classificação

Tem-se uma base de dados com um número de objetos. Existe um conjunto de classes associado a essa base de dados, e cada objeto pertence a uma classe. Cada objetivo possui um conjunto de atributos. Um algoritmo de classificação é aquele que, dado isso, busca aprender a classificar um objeto de acordo com os seus atributos.

Por exemplo, suponha que você quer determinar a espécie de uma flor. Mil amostras de flor foram fotografadas e tiveram medidas o seu tamanho de caule, largura e comprimento das pétalas. Um botânico classifica duzentas dessas fotos uma a uma, e pede que você desenvolva um sistema computacional que classifique as restantes, e todas as fotos que irão ser feitas no futuro.

Um algoritmo de classificação é aquele que resolve esse problema. Ele é, acima de tudo, um buscador de **padrões** entre esses dados. “O que as flores dessa classe tem que nenhuma outra tem?”. Ele busca encontrar, portanto, o que *diferencia* as classes de flores, de modo que, ao receber uma foto sem classificação, ele saiba classifica-la, a partir desses atributos diferenciadores.

Figura 8 - Um algoritmo de classificação busca entender o que separa duas ou mais classes de objetos.



Fonte: https://qph.fs.quoracdn.net/main-qimg-53d5f0110a715905b8742719a858bdbe.

Observe que, na figura oito, temos dois tipos de bolinhas: as verdes, à esquerda, e as vermelhas, à direita. O algoritmo de classificação, no caso dessa imagem, buscou separar as duas e obteve um bom resultado, apesar de alguns erros: algumas bolinhas vermelhas ficaram do lado das verdes, e vice-versa.

Em qualquer problema de classificação é isso que o algoritmo busca fazer. “Qual padrão de pixels determina que essa foto é a de um número seis, e qual padrão indica o número um?”. “Qual tamanho de pétala e caule pertence a uma determinada espécie?”, e assim por diante.

Entre os diversos problemas que um algoritmo de classificação pode resolver, temos: o reconhecimento de letra escrita à mão, para leitura de livros, formulários e cheques; os reconhecimentos de faces ou objetos em imagens, como árvores, frutas, e assim por diante; reconhecimento de spam enviado a e-mails.

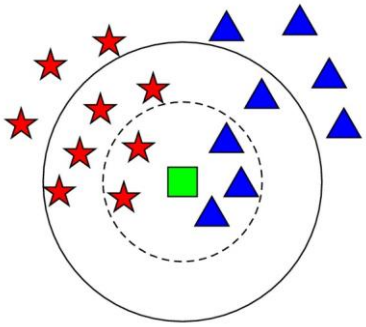
Figura 9 - Classificar imagens, como as que contém uvas, nesse exemplo, é uma aplicação para algoritmos de classificação.

Entre os algoritmos de classificação temos exemplos como:

- Classificadores Lineares.
- Classificadores Quadráticos.
- SVM: Support Vector Machines.
- K-Nearest Neighbors.
- Árvores de Decisão.

- Redes Neurais.

Figura 10 – Exemplo de algoritmo de classificação: K-Nearest Neighbors.



Neste capítulo, nós estudaremos as diversas métricas de qualidade que nos permitem avaliar os resultados fornecidos por algoritmos de classificação. Iremos estudar as métricas de Acurácia, Precisão, Recall, F-Score, Matriz de Confusão, Área sob a curva ROC e Log Loss.

Acurácia e Precisão

As métricas mais fundamentais dos algoritmos de classificação buscam dizer, de maneira bem direta, quão bem ele soube classificar os objetos sob análise. Duas das métricas mais fundamentais de classificação são a acurácia e a precisão.

Como exemplo base, suponha que você criou um classificador feito para separar frutas na linha de produção de uma fazenda. Você quer separar frutas maduras de frutas verdes automaticamente, para economizar o máximo de trabalho manual de separação. Após fazer o classificador, tendo como “positivo” a classificação de uma fruta como “verde”, você o testa com seis frutas. Essa é uma amostra de testes excessivamente pequena, mas a mantemos assim para fins didáticos:

Tabela 1 - Amostragem de frutas maduras e verdes contra resultados da classificação.

	Verdes	Maduras
Amostragem	3	3
Classificadas	4	2

Intuitivamente, pode-se querer dizer que nosso classificador foi bem: afinal de contas, ele encontrou somente uma fruta verde a mais e somente uma fruta madura a menos. Um resultado próximo do ideal. Mas fazer tal afirmação exige uma análise mais aprofundada dos resultados:

Tabela 2 - Classificação de frutas verdes e maduras : resultados detalhados.

	Ela é...	Classificação	Resultado	Significado
Fruta 1	Verde	Verde	Correto	
Fruta 2	Verde	Madura	Errado	Falso Negativo
Fruta 3	Verde	Verde	Correto	
Fruta 4	Madura	Madura	Correto	
Fruta 5	Madura	Verde	Errado	Falso Positivo
Fruta 6	Madura	Verde	Errado	Falso Positivo
...	...	...	...	...

Vemos que a realidade não era o que aparentava. Apenas três das frutas classificadas estavam corretas, o que dá uma acurácia de 50%. A **acurácia** nada mais é do que a proporção de classificações corretas pelo total de classificações naquele domínio. Uma acurácia de 73%, por exemplo, significa que 73 de cada 100 objetos classificados o serão corretamente.

A **precisão** já diz outra coisa. “Quantos dos meus resultados positivos são *realmente* positivos? Observe que, no exemplo anterior, tivemos dois resultados falsos positivos. Duas frutas maduras classificadas como verdes erroneamente. Como foram quatro frutas classificadas como verdes, temos uma precisão de 50%. Isto é: de todas as frutas classificadas como verde, metade foi uma classificação falsa. A *precisão é uma medida da confiabilidade de sua classificação “positiva”.*

Reflete-se acerca do assunto: quando a precisão é realmente importante? Idealmente, nosso modelo é bom em todas as métricas. Entretanto, há casos e casos. No caso de um separador de frutas, uma precisão baixa significa mais trabalho manual dos trabalhadores que conferem a separação.

Há casos onde um falso positivo representa um erro grave. Imagine um sistema que dá diagnósticos para paciente. O dito sistema classifica um paciente como portador de câncer, e, no entanto, isso foi um falso positivo. As consequências de tal classificação seriam muito mais severas.

Recall e F-Score

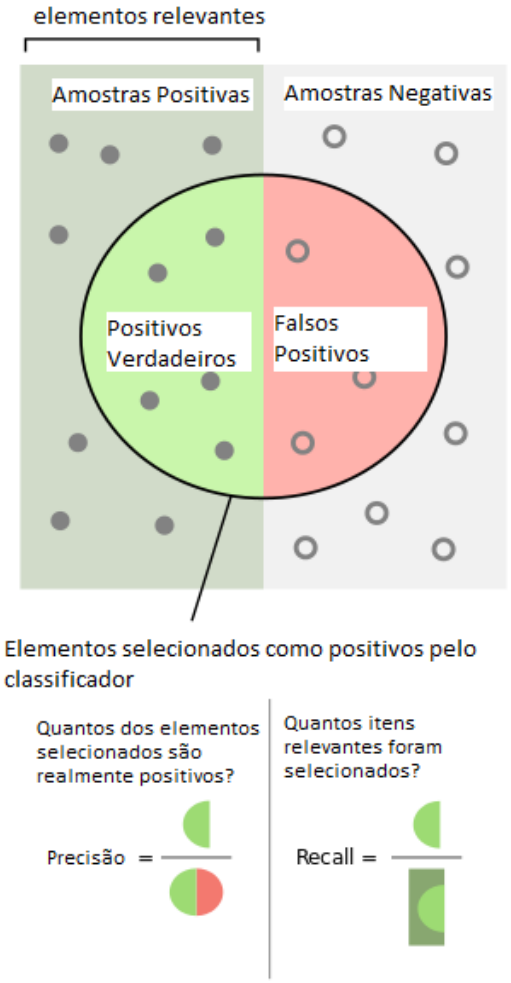
Sabendo que a acurácia de um modelo e o quanto que ele acertou em suas previsões, e que a precisão é quanto de seus resultados não foram falsos positivos, chegamos ao recall, ou sensibilidade. O recall é a medida que mostra quanto de todos os nossos resultados positivos em teste foram encontrados pelo nosso modelo.

Em nosso separador de frutas verdes e maduras, vemos que, das três frutas verdes em teste, encontramos duas. Nosso recall, portanto, foi de 66%. O recall

significa *quanto de todas as amostras positivas o nosso classificador consegue capturar*.

Assim, já é possível tirar algumas conclusões: em um cenário onde a classificação correta vem em primeiro lugar, a acurácia deve ser maximizada em sua análise. Quando é crucial não ter falsos positivos, a precisão deve ser maximizada. Quando o importante é não deixar de “pegar” nenhum resultado, o ideal é buscar um recall alto.

Figura 11 - Precisão e Recall.



Fonte: https://en.wikipedia.org/wiki/File:Precisionrecall.svg.

A figura 11 ilustra bem esses cenários e as diferenças entre as métricas. Nela, temos 12 amostras positivas (buscadas pelo classificador) e 10 amostras negativas. O classificador encontrou 8 objetos positivos, três dos quais falsos positivos. Daí extraímos as métricas:

- A acurácia é 12 para 22, ou 55%, pois foi a quantidade de objetos classificados corretamente como positivos ou negativos.
- A precisão é de 5 para 8, ou 62,5%, pois é a fração dos objetos encontrados positivos que realmente são positivos.
- O recall é 5 para 12, ou 42%, pois foi a quantidade de objetos positivos que o classificador corretamente encontrou.

Por fim, temos o F-score. A métrica que reúne em um valor só os valores de precisão é recall. Usável quando ambas métricas têm igual relevância. A métrica é simplesmente a média harmônica das duas métricas, como mostra a figura 12. No exemplo da figura 11, o F-score é 50,2%. No exemplo do separador de frutas, ele é 57%.

Figura 12 - Fórmula do F-score.

$$F_1 = 2 \times \frac{precisão * recall}{precisão + recall}$$

Faz-se o questionamento de qual métrica é a mais relevante e a resposta não é simples: depende do tipo de problema. Imagine um classificador de um carro que dirige sozinho, em piloto automático. Ele, recebendo os dados de inúmeros sensores, deve classificar se a situação em que o transito ao redor de si permite que ele siga seu trajeto, ou exige que ele desvie ou freie para evitar uma colisão.

Um classificador responsável por resolver esse problema pode se der ao luxo de ter falsos positivos, isto é, classificar como “normais” situações que exigem ação para evitar colisões. A resposta é não. Nesse caso, tem-se como precisão a métrica

essencial do classificador. Sob nenhuma hipótese uma colisão deve acontecer. Se houver um falso negativo, isto é, classificar como anormais situações normais, o carro desviará ou freará desnecessariamente. Isso não é tão grave quanto colidir.

No caso do separador de frutas, no entanto, tanto frutas maduras classificadas como verdes quanto as verdes classificadas como maduras gerarão o mesmo resultado: elas terão que ser separadas manualmente mais adiante, na linha de montagem. Nesse caso, o F-score interessa mais: seja precisão ou recall, é importante que ele seja o maior possível. O único denominador comum de todos esses casos é que uma acurácia alta é sempre preferível.

Matriz de Confusão

A Matriz de Confusão ou Matriz de Classificação, é menos uma métrica e mais uma forma de visualizar os seus resultados de classificação. Utilizando-a, é possível enxergar o desempenho do seu classificador com muito mais clareza. Além disso, é possível calcular as demais métricas básicas, acurácia, precisão e recall, através dela.

Tabela 3 - Matriz de Confusão de um classificador de maçãs.

	Maçã	**Pêssego**
Maçã	150	90
Pêssego	30	130

Tabela 4 - Esquema da Matriz de Confusão.

	Positivo	**Negativo**
Positivo	Positivos verdadeiros	Falsos negativos
Negativo	Falsos positivos	Negativos verdadeiros

Como exemplo, tomemos o classificador de frutas verdes e maduras anterior. Suponhamos que agora você precisa diferenciar frutas, e não só dizer se elas são maduras. Você precisa diferenciar maçãs, bananas, peras e pêssegos. Ao testar o seu classificador com mil frutas, 250 de cada tipo, você obtém os seguintes resultados:

Tabela 5 - Matriz de Confusão de um classificador de vários tipos de frutas.

	Maçã	**Pêssego**	**Pera**	**Banana**
Maçã	150	90	10	0
Pêssego	30	130	80	10
Pera	30	30	180	10
Banana	0	5	15	230

Cada linha e cada coluna é uma classe de seu classificador. Os números de uma linha representam quantos membros daquela classe foram classificados com a classe daquela coluna. Ou seja, no exemplo da Tabela 3, 90 maçãs foram erroneamente classificadas como pêssegos, e 150 foram corretamente classificadas como maçãs. Utilizando a tabela, conseguimos facilmente calcular as outras métricas:

- Acurácia. Para calculá-la, utiliza-se a linha associada à classe sob análise, para somar os falsos negativos, e a coluna, para os falsos positivos. Assim, temos:
  - Maçãs: 840 corretas, 160 erradas **84%.**
  - Pêssegos: 755 corretas, 245 erradas **75%.**
  - Peras: 825 corretas, 175 erradas **82,5%.**
  - Bananas: 960 corretas, 40 erradas **96%.**
- Precisão, sendo as classificações corretas contra a soma do restante da linha (os falsos positivos):
  - Maçãs: 150 corretas, 210 encontradas: **71.4%.**
  - Pêssegos: 130 corretas, 255 encontrados: **51%.**
  - Peras: 180 corretas, 285 encontradas: **63%.**
  - Bananas: 230 corretas, 250 encontradas: **92%.**
- Recall, sendo as classificações corretas contra o total de membros daquela classe no teste.
  - Maçãs: 150 corretas, 250 no total **60%.**
  - Pêssegos: 130 corretas, 250 no total **52%.**
  - Peras: 180 corretas, 250 no total **72%.**
  - Bananas: 230 corretas, 250 no total **92%.**
- F-Score:
  - Maçãs: **65%.**
  - Pêssegos: **51,5%.**
  - Peras: **67,2%.**

– Bananas: **92%.**

Mais do que simplesmente um modo mais fácil de calcular as outras métricas e visualizar nosso resultado, a matriz de confusão nos permite enxergar novas informações. No nosso exemplo, muitas maçãs se confundiram com pêssegos, mas o contrário não aconteceu tanto, por que será? E o que fez tantos pêssegos se assimilarem a peras? Durante a análise de um problema, esse tipo de pergunta te ajuda a entender o comportamento do classificador – e ajuda a torna-lo melhor.

Área sob a curva ROC

A área sob a curva ROC, em inglês denominada *Area Under Receiver Operating Characteristic*, ou *AUROC*, é uma forma de determinar e visualizar graficamente, qual classificador dentre um conjunto deles se aproxima mais do desempenho ideal.

Seu desenho se faz de forma relativamente simples: no eixo Y do gráfico, tem-se o recall (sensibilidade) do classificador. No eixo X, tem-se a sua taxa de falsos positivos, isto é, a quantidade de classificações falsas positivas sobre a quantidade de exemplos negativos no seu teste.

Ou seja, são duas as grandezas comparadas na curva ROC: a sensibilidade de um classificador, que se quer ter maximizada, e a taxa de falsos positivos, que se quer minimizada. Um classificador que marca todas as entradas como positivas teria 100% de sensibilidade, mas 100% de falsos positivos também. Um classificador que acerta metade de suas classificações é tão bom quanto classificação via cara ou coroa. Assim sendo, temos a intuição que nos permite visualizar o gráfico da curva ROC, de acordo com os valores exemplificados nas tabelas 6 a 9.

Tabela 6 - Matriz de Confusão de um classificador de maçãs, exemplo 1.

	Maçã	**Pêssego**
Maçã	150	90
Pêssego	30	130

Tabela 7 - Matriz de Confusão de um classificador de maçãs, exemplo 2.

	Maçã	**Pêssego**
Maçã	200	40
Pêssego	60	100

Tabela 8 - Matriz de Confusão de um classificador de maçãs, exemplo 3.

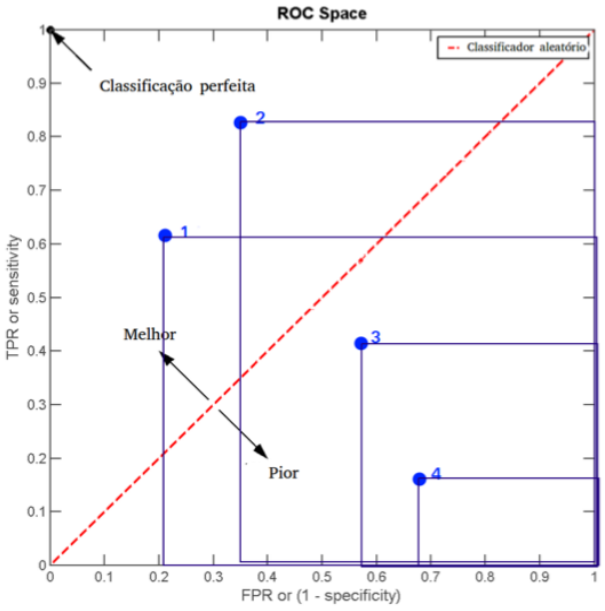
	Maçã	**Pêssego**
Maçã	100	140
Pêssego	90	70

Tabela 9 - Matriz de Confusão de um classificador de maçãs, exemplo 4.

	Maçã	**Pêssego**
Maçã	40	200
Pêssego	110	50

A linha vermelha na diagonal do gráfico, exibido na figura 13, representa um classificador que é tão bom quanto um classificador que sorteia suas classificações, ou seja, que acerta rigorosamente metade de suas tentativas de classificação. Os pontos azuis representam os classificadores descritos nas tabelas 6 a 9, e os quadrados que os acompanham são suas áreas sob a curva ROC. A curva ROC é a curva que vai da origem do gráfico até o ponto (1,1), passando pelo ponto do classificador. Quanto maior a área sob a curva, melhor o classificador é.

Figura 13 - Classificadores 1 a 4 com suas respectivas áreas sob a curva.



Fonte: https://commons.wikimedia.org/w/index.php?curid=8326140.

Exemplo - Classificação de vinhos

As imagens abaixo mostram um exemplo de seleção de modelos para classificação single label de um vinho - baseado em seus atributos, devemos determinar qual é a sua uva. Fazemos a importação do dataset, separação dos dados para treino e teste, instanciação do classificador, criação do modelo baseado nos dados de treino e subsequente teste desse. Fazemos isso com o Classificador Random Forest e com o KNN, medimos suas métricas de qualidade (aqui medimos as três para fins didáticos) e comparamos os nossos resultados.

```
# Carregando o dataset
from sklearn.datasets import load_wine
wine = load_wine()

# Exemplos de acesso aos dados
X = wine.data[:, :] # Features de cada elemento
y = wine.target # Classes de cada elemento

# -----------------

# É preciso treinar o classificador, e testar o seu desempenho com dados "novos".
# Aqui, dividimos os dados em treino e teste, para podermos testar nosso desempenho depois.

from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.33, random_state=42)
# O uso dessa função facilita, mas não é obrigatório. Você pode dividir os seus dados manualmente.

# -----------------

# Carregando e treinando os classificadores
# Random forests
from sklearn.ensemble import RandomForestClassifier
rfc = RandomForestClassifier()
rfc.fit(X_train,y_train)
y_pred = rfc.predict(X_test)

# Métricas do Random Forests

from sklearn.metrics import accuracy_score, recall_score, precision_score

rfc_acc = round(accuracy_score(y_test, y_pred), 6) # round é para arredondar
rfc_recall = round(recall_score(y_test, y_pred, average='weighted'), 6)
rfc_precision = round(precision_score(y_test, y_pred, average='weighted'), 6)

```

```
35
36 # -----------------
37
38 # KNN
39 from sklearn.neighbors import KNeighborsClassifier
40 knn = KNeighborsClassifier(n_neighbors=3)
41 knn.fit(X_train, y_train)
42 y_pred = knn.predict(X_test)
43
44 # Métricas do KNN
45 knn_acc = round(accuracy_score(y_test, y_pred), 6)
46 knn_recall = round(recall_score(y_test, y_pred, average='weighted'), 6)
47 knn_precision = round(precision_score(y_test, y_pred, average='weighted'), 6)
48
49 # -----------------
50
51 # Comparação
52 print("KNN vs Random Forests\n")
53 print("Classes: {0}\n".format(wine.target_names))
54 print("Acurácia: {0} vs {1}".format(knn_acc, rfc_acc))
55 print("Recall: {0} vs {1}".format(knn_recall, rfc_recall))
56 print("Precisão: {0} vs {1}".format(knn_precision, rfc_precision))
57
58 # -----------------
```

Capítulo 3. Métricas para Algoritmos Multilabel

Algoritmos multilabel são como uma extensão dos algoritmos de classificação binários, que, usualmente, determinam se o objeto sob análise pertence ou não pertence a uma classe específica. Entenda aqui que *labels* são como classes. Assim, o nome da categoria de algoritmos já nos indica o que eles fazem: são algoritmos que atribuem várias classes a cada objeto.

Isto é: dados N objetos e L labels possíveis, um algoritmo multilabel busca atribuir a cada objeto as labels que lhe pertencem. Isto é: busca encontrar os padrões que define que uma label pertence a um conjunto de objeto específico, e somente a ele.

Figura 14 - Um classificador multilabel poderia querer atribuir às imagens abaixo as seguintes labels: (1) contém comida, (2) contém prato, (3) foto tirada a menos de 100m de um restaurante.

Fonte: http://ghk.h-cdn.co/assets/16/17/980x490/landscape-1461861317-gettyimages-142156665.jpg, https://upload.wikimedia.org/wikipedia/commons/e/e7/SM_Supermarket_BF_Para%C3%B1aque_storefront_%28Front_part%29.jpg, https://lh3.googleusercontent.com/GlGayfoebCwqAy-gMql49olP9K794pbZa1ZPVl1e0LnSg1-nA1P5eBkoAJ-DU4duGP1YYpiSDdrmXVkQk06nCIMvCT9c_dspYBDiuP_hEA6hNx-oeQ0, https://thateconomist.files.wordpress.com/2014/11/img_6268.jpg.

Como exemplo simples, podemos imaginar classificadores de músicas: essa música é *animada?* De *rock?* Dos *anos 50?* Cada pergunta é uma label a se atribuir, ou não, a um objeto. Isso se estende a fotos também (essa foto contém Árvores? Carros? Placas?), dentre vários outros exemplos.

Tabela 10 - Exemplos de datasets para treino e testes de algoritmos multilabel.

Nome	Descrição	Domínio	Instâncias	# de labels
bibtex	Classificador de artigos	Texto	7395	159
birds	Identificar cantos de pássaros	Som	645	19
corel16k	Associar imagens a palavras	Imagens	13811	161
genbase	Classificação de proteínas	Biologia	662	27

Observe que um algoritmo de classificação comum buscaria perguntar “Essa música é animada ou não?”, ou “Essa música é de rock ou não?”. Um algoritmo de classificação atribui uma classe a um objeto ou não a atribui. Um algoritmo multilabel, observa-se, tem ideia semelhante a ele. A diferença é que várias perguntas são feitas por objeto, ao invés de apenas uma.

Assim sendo, há duas técnicas principais de resolução de problemas multilabel, e ambas têm a ver com problemas de classificação. A primeira técnica é a **transformação do problema,** onde buscamos transformar o nosso problema multilabel em um problema de vários classificadores binários. A segunda técnica é a **adaptação do algoritmo**, onde adaptamos um algoritmo de classificação para problemas multilabel.

Como exemplos de formas de resolver os problemas multilabel utilizando a primeira técnica, temos, considerando L o número de labels do problema:

- Um contra todos (*one-versus-all*) – Utiliza-se L classificadores binários sobre os seus dados, cada um atribuindo, ou não, uma determinada label a um objeto. Por exemplo, para o caso do problema da música, teríamos três classificadores binários: um que diz se a música é dos anos 50 ou não, outro que diz se a música é animada ou não, e outro que diz se a música é de rock ou não.

- Label Powerset – Crie um classificador que indica se seu objeto possui um conjunto de labels ou não. Esse objeto é só dos anos 50? É dos anos 50 **e** animado? É só animado? É dos anos 50, animado e rock? É só rock? Em outras palavras, transforme o problema multilabel em um problema de classificação multiclasses, e resolva-o com um classificador multiclasses.

Considerando a técnica de **adaptação do algoritmo** para a resolução de problemas multilabel, consideramos dois exemplos:

- K-Vizinhos mais próximos (KNN) - O algoritmo KNN tradicional atribui a um objeto a classe mais comum dentre os seus K-vizinhos mais próximos. O MLKNN atribui a um objeto as labels mais presentes entre seus vizinhos.

- Árvore de decisão – a árvore de decisão multilabel é como a tradicional, com uma alteração: ao invés de um objeto percorrer um único caminho na árvore, para ser classificado, ele percorre todos os caminhos que lhe são pertinentes, e cada “folha” que ele atinge é uma label que ele possui.

Cardinalidade e Densidade de labels

Cardinalidade e densidade são duas métricas que nos ajudam a caracterizar um conjunto de dados para algoritmos muiltilabel. A **cardinalidade** de um conjunto

de dados é a quantidade média de labels associadas a cada objeto. A **densidade** é a fração do total de labels, média, que cada objeto possui.

Por exemplo: suponhamos um dataset imaginário de classificação multilabel de fotos. Há 10 labels (Y1 a Y10) e mil fotos (X1 a X1000) a classificar. Para as cinco primeiras fotos, temos os seguintes resultados de classificação:

- X1: Y1, Y3, Y4, Y8
- X2: Y1, Y3, Y7
- X3: Y1, Y4, Y9
- X4: Y1, Y5, Y6, Y10
- X5: Y1, Y4, Y10

Qual é a cardinalidade desse dataset, considerando só esses 5 exemplos?

- X1: 4 labels
- X2: 3 labels
- X3: 3 labels
- X4: 4 labels
- X5: 3 labels

A cardinalidade é média da quantidade de labels por objeto, ou seja, 3,4. A densidade, por outro lado, é a cardinalidade divida pelo número de labels, ou seja, a fração média do total de labels que há em cada objeto analisado. No caso deste exemplo, 0.34.

Isso significa que um dataset com densidade alta é um dataset onde uma grande fração de suas labels está em cada objeto analisado. É um dataset *pouco separado.* Um dataset com cardinalidade alta, por outro lado, é *pouco específico*, pois

uma grande quantidade de labels está associada a cada objeto. Datasets podem, portanto:

- Muito específicos e muito separados (Baixa cardinalidade, baixa densidade).
- Pouco específicos e muito separados (Alta cardinalidade, baixa densidade).
- Muito específicos e pouco separados (Baixa cardinalidade, alta densidade).
- Pouco específicos e pouco separados (Alta cardinalidade, alta densidade).

Podemos intuir que os melhores dados de se analisar são os dados muito específicos (padrões únicos de dados a diferenciar) e muito separados (padrões muito diferentes), mas especificidade demais pode dar pouca informação. No fim das contas, o resultado para as métricas que você buscar dependerão do seu problema. Só atente que, uma vez que a densidade é cardinalidade divida pelo número de labels, essas duas métricas estão relacionadas. Situações onde uma é alta e a outra é baixa são raras e não tendem a acontecer.

Perda de Hamming e Perda 0-1

A Perda de Hamming e a Perda 0-1 são duas métricas que calculam o desempenho dos algoritmos de classificação multilabel. Suas análises se assemelham nas ideias, mas tem implicações bastante diferentes, como veremos mais adiante. Primeiramente, um exemplo:

Tabela 11 - Classificador de fotos multilabel. Quais labels se aplicam a cada foto?

	Contém pessoas?	Contém carros?	Contém plantas?	Contém animais?
Foto 1	**Sim**	**Não**	**Sim**	**Não**

Foto 2	Não	Sim	Não	**Sim**
Foto 3	Sim	Não	Não	**Sim**
Foto 4	Sim	Não	Não	**Não**
Foto 5	**Sim**	**Sim**	**Não**	**Não**

A Tabela 11 representa a amostra dos resultados de um classificador multilabel. Dado um conjunto de fotos, o classificador deveria atribuir a cada foto as labels “contém pessoas”, “contém carros”, “contém plantas”, “contém animais”. Uma forma mais concisa de representar resultados como esse, está na tabela 12.

Tabela 12 - Resultados do classificador multilabel, numa notação mais concisa.

	Resultados Esperados	**Resultados Obtidos**
Foto 1	1,0,1,0	1,1,0,0
Foto 2	0,1,0,1	0,1,0,0
Foto 3	1,0,0,1	1,0,0,1
Foto 4	1,0,0,0	1,0,0,1
Foto 5	**1,1,0,0**	**1,1,0,0**

A Perda de Hamming de um resultado é calculada de forma simples: conte quantas labels erradas seu resultado tem e divida pelo número total de labels a classificar. No caso deste exemplo, temos 4 erros para 20 labels, o que caracteriza uma Perda de Hamming de 0,2, ou 20%.

Para a Perda 0-1, por outro lado, soma-se a quantidade de objetos que tiveram uma label qualquer classificada incorretamente. No caso do nosso exemplo, a foto 1 teve duas labels erradas, e as foto 2 e 4 tiveram uma label errada cada. Assim, temos 3 fotos com erros para 5 fotos no total, o que dá uma Perda 0-1 de 60%.

Ou seja: ambas as métricas calculam *perda*. Isso significa que, quanto menor os seus resultados, melhor. Para a Perda de Hamming, considera-se a capacidade do classificador de classificar labels corretamente. A Perda 0-1 considera a capacidade do classificador de classificar objetos corretamente.

Tabela 13 - Um novo resultado do classificador.

	Resultados Esperados	Resultados Obtidos
Foto 1	1,0,1,0	1,0,1,1
Foto 2	0,1,0,1	1,0,1,0
Foto 3	1,0,0,1	1,0,0,1
Foto 4	1,0,0,0	1,0,0,0
Foto 5	**1,1,0,0**	**1,1,0,0**

Neste segundo exemplo, temos 5 labels incorretas de 20 no total. Isso dá uma Perda de Hamming de 25%. Temos, por outro lado, 2 dos 5 objetos classificados incorretamente, o que dá uma Perda 0-1 de 40%.

Comparando o conceito das duas métricas, Perda de Hamming e Perda 0-1, vemos uma clara diferença: a Perda de Hamming analisa **classificações de labels**. A Perda 0-1 analisa **classificações de objetos.**

Tabela 14 - Exemplo de resultado de classificador multilabel. A Perda de Hamming é 30%, e a Perda 0-1 é 60%.

	Resultados Esperados	Resultados Obtidos
Foto 1	1,0,1,0	1,0,0,1
Foto 2	0,1,0,1	0,1,0,1
Foto 3	1,0,0,1	1,1,1,1
Foto 4	1,0,0,0	1,0,1,1
Foto 5	**1,1,0,0**	**1,1,0,0**

Assim sendo, a Perda de Hamming se torna uma métrica, via de regra, mais fácil de se minimizar do que a Perda 0-1. A ideia é simples: ela analisa a capacidade do analisador de detectar a presença de padrões, nos dados, que devem ser associados a uma label específica. E isso é feito *independentemente de contexto.* Ou seja: as outras labels associadas àquele objeto não importam para a classificação.

A Perda 0-1, por outro lado, exige que o contexto seja levado em consideração: o que importa não são simplesmente labels bem classificadas, mas labels bem classificadas *juntas*. Você quer que um conjunto de labels sempre apareça

juntas. A classificação de sua label por si só não basta. Ela é avaliada conjuntamente com as outras labels ao seu redor. Isso é levar o contexto em consideração.

Tabela 15 - Exemplo de resultado de classificador multilabel. A Perda de Hamming é 20%, e a Perda 0-1 é 80%.

	Resultados Esperados	Resultados Obtidos
Foto 1	1,0,1,0	1,0,1,1
Foto 2	0,1,0,1	0,1,1,1
Foto 3	1,0,0,1	1,0,1,1
Foto 4	1,0,0,0	1,0,1,0
Foto 5	**1,1,0,0**	**1,1,0,0**

No exemplo ilustrado na Tabela 15, temos um resultado com ótima Perda de Hamming, e péssima Perda 0-1. O classificador identifica bem as labels, mas a falta de consideração do contexto o leva a cometer erros inaceitáveis.

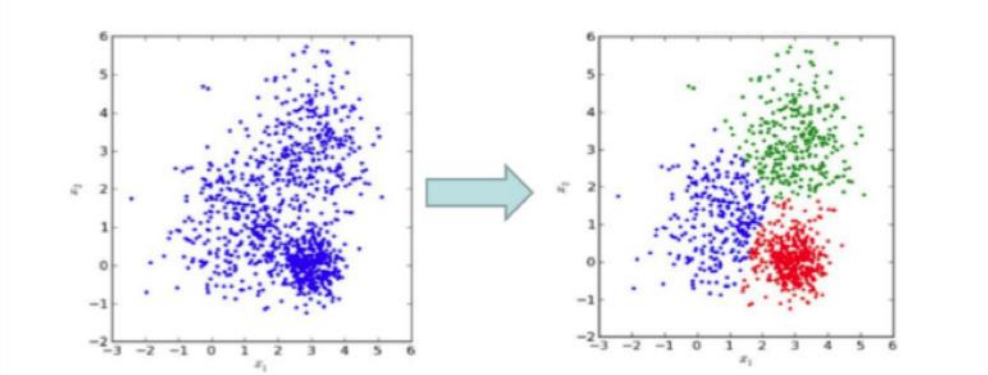
Resta a questão: qual métrica é melhor de se utilizar? A resposta, como já é de se esperar, é “depende”. O seu problema exige classificação correta de labels, pura e simplesmente, ou exige que os seus objetos tenham as labels 100% classificadas? Idealmente usa-se várias métricas para se analisar um classificador multilabel. Quais métricas devem ser levadas em consideração é uma questão para o Analista de Dados responder, de acordo com o que seu problema exige – de acordo com o valor que ele precisa entregar.

Capítulo 4. Métricas para Algoritmos de Agrupamento

Algoritmos de agrupamento, ou clustering, tem uma lógica diferente de técnicas de classificação multilabel. Nestas você dá a elas um conjunto de dados previamente classificado, e as técnicas usam esses conjuntos para tentar entender quais padrões de dados implicam naquele objeto ser de uma classe ou de outra. Em algoritmos de agrupamento, ocorre o inverso: você dá um conjunto de dados, e o algoritmo busca as classes ou labels em que se pode dividi-lo.

A execução de um algoritmo de agrupamento visa responder a diversas perguntas:

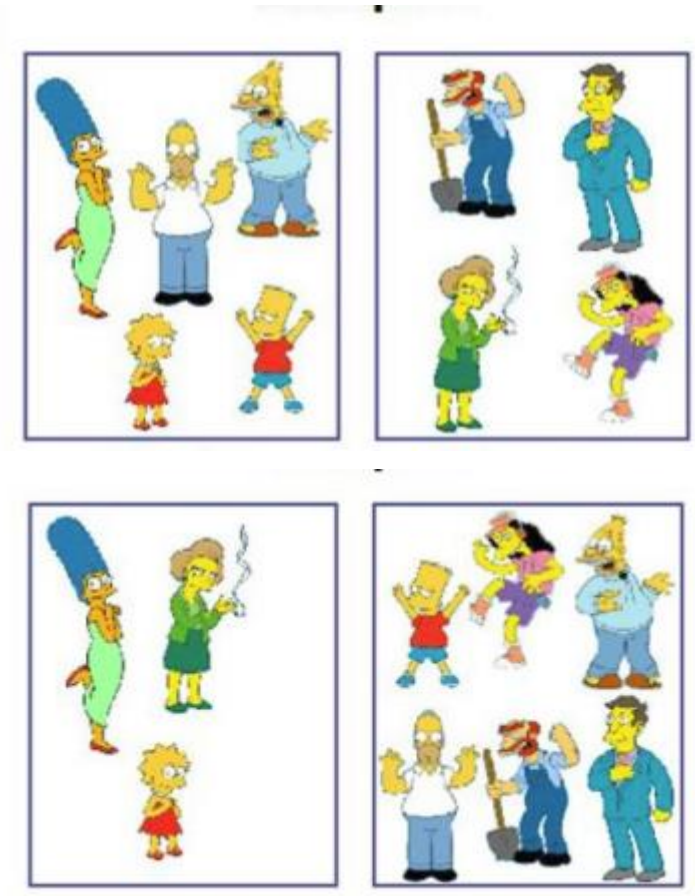
Figura 15 - Dados não agrupados, à esquerda, e dados agrupados, à direita.



- Eu conseguirei encontrar padrões distintos nesses dados, que me permita classificá-los? Ou devo buscar mais dados?
- Dado que eu tenho dados categorizados, esses dados estão bem separados ou mal separados? Em outras palavras, classificar esses dados será fácil ou difícil?
- O que é preciso levar em consideração para separar esses dados.

Por exemplo. Suponhamos que você, por diversão, resolva classificar por agrupamento os personagens do desenho The Simpsons. A figura 16 mostra duas separações possíveis para o mesmo conjunto de dados, ambas igualmente válidas:

Figura 16 - Personagens do desenho The Simpsons, separados de duas formas diferentes, igualmente válidas. Acima, a separação entre personagens da família e personagens da escola. Abaixo, a separação é entre homens e mulheres.



Um mesmo conjunto de dados, portanto, pode ser separado de várias maneiras diferentes e igualmente válidas. Mais uma vez o trabalho do Analista de Dados se faz fundamental: é preciso compreender o seu problema para dizer qual separação agrega valor que você busca e qual separação é irrelevante.

Tipos de Algoritmos de Agrupamento

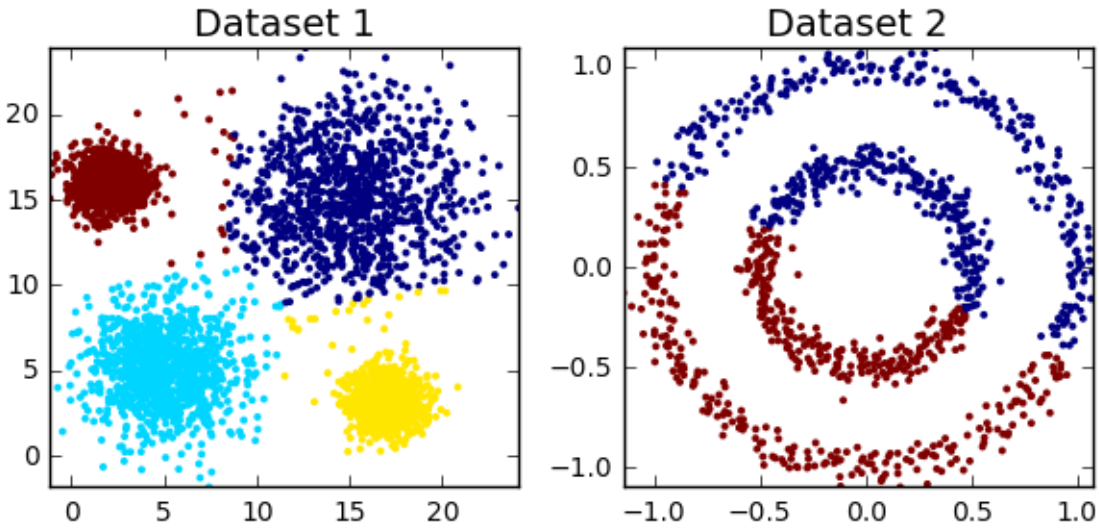
Existe uma grande miríade de tipos de algoritmos de agrupamento, e nessa seção descreveremos dois muito usados, evidenciando suas forças e fraquezas. Eles

são o algoritmo de K-Médias (K-Means) e o Agrupamento Hierárquico (Hierarchical Clustering).

O algoritmo K-Médias funciona da seguinte forma. Primeiro estabelece-se um número de centros, distribuídos pelos seus dados aleatoriamente ou segundo certos critérios. O número de centros é o número de conjuntos que você busca. Em seguida, faz-se o seguinte:

- Itere sobre todos os pontos dos dados. Os pontos “pertencem” ao centro mais próximo deles. A métrica de distância pode ser, por exemplo, a distância euclidiana entre os pontos.
- Mova o centro para o centroide de todos os seus pontos.
- Repita até o algoritmo se reduzir a mudanças mínimas, ou se estabilizar.

Figura 17 - Resultado da execução do algoritmo K-Médias sobre dois conjuntos de dados diferentes. O conjunto de dados da direita teve desempenho sofrível, um vez que a separação dos dados está claramente mal feita.

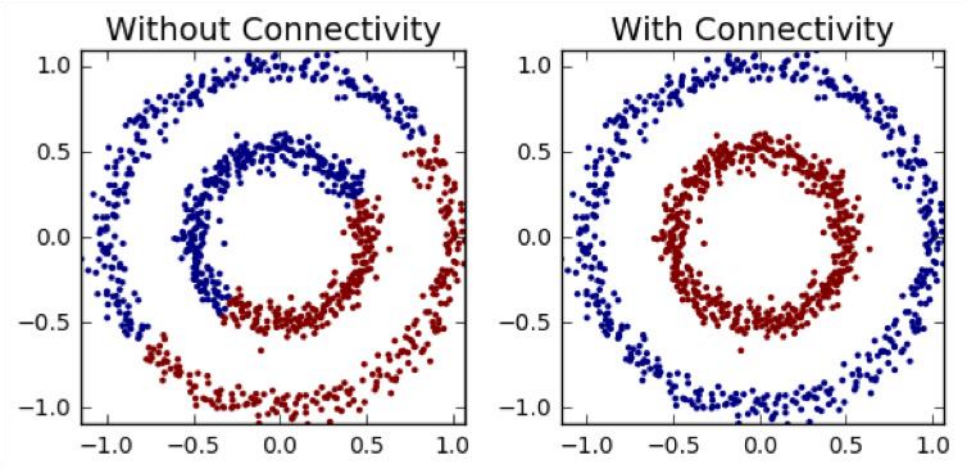


O algoritmo de agrupamento hierárquico segue uma lógica semelhante: primeiro, escolhe-se o número de clusters que você desejar obter. Em seguida:

- Agrupe os objetos sob análise de dois em dois. Cada grupo resultante é um cluster.
- Una um cluster com o outro que estiver mais próximo a ele.
- Repita até ter a quantidade de clusters desejados.

E há uma variação do algoritmo que leva em consideração a conectividade: isto é, limite a união de clusters. Cada ponto só pode se unir aos seus N (por exemplo, n=5) vizinhos mais próximos. A depender dos seus dados, há gritante diferença de desempenho.

Figura 18 - Resultado sobre um conjunto de dados da aplicação de um algoritmo de agrupamento hierárquico. O resultado da direita, levou em conta a conectividade dos objetos, e o da esquerda não.



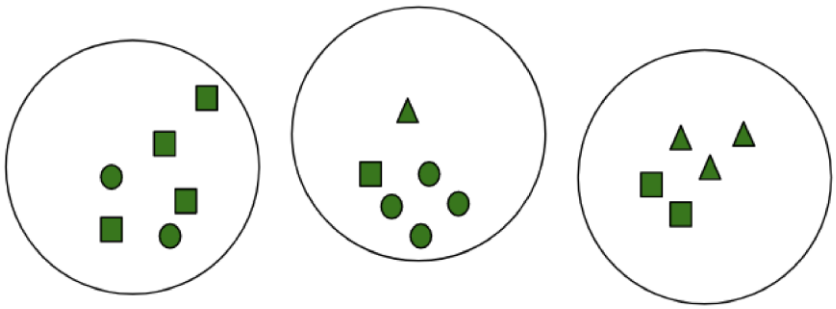
E há vários outros tipos de algoritmos de agrupamento. Alguns deles sendo:

- Maximização de Expectativas (*Expectancy Maximization*).
- Deslocamento da Média (*Mean Shift*).
- Propagação de Afinidade (*Affinity Propagation*).

Pureza e Entropia

Pureza e entropia são duas métricas básicas de entendimento dos agrupamentos gerados pelo seu algoritmo. A pureza é quanto de um agrupamento é formado por uma só classe. A entropia mede o quão pouco separados, caóticos, os seus dados estão. Tome como base o conjunto de dados da figura 19. Um algoritmo de agrupamento a separou em três grupos diferentes.

Figura 19 - Conjunto de dados com três classes, triângulo, círculo e quadrado, separado em três clusters.



A pureza de um agrupamento pode ser definida como a fração que a sua classe mais comum ocupa nele. Por exemplo: na figura 19, descrita na tabela 16, o agrupamento 1 possui da classe 2 como classe mais comum. De seus 6 elementos, 4 pertencem a essa classe. Ou seja, 4 de cada 6 elementos do agrupamento pertencem a sua classe mais comum. 4 dividido por 6 resulta em 67%, e esta é a pureza do agrupamento 1.

Fazendo raciocínios análogos, os agrupamentos 2 e 3 tem pureza, respectivamente, de 67% e 60%. Na média, o resultado do algoritmo e agrupamento tem pureza de 65%.

Tabela 16 - Distribuição das classes da figura 19 nos agrupamentos.

	Classe 1	Classe 2	Classe 3
Agrupamento 1	**2**	**4**	**0**
Agrupamento 2	**4**	**1**	**1**
Agrupamento 3	**0**	**2**	**3**

É possível entender a pureza e a entropia se fizemos uma analogia com gases. Imagine, para fins didáticos, que cada ponto representa uma unidade de um gás, e cada classe é um gás diferente. Suponhamos que o papel do algoritmo seja encher balões com os gases que você está analisando. A pureza é o quanto que a mistura de gases dentro de um balão é pura. A entropia é o quanto que os gases estão misturados. Uma mistura com pouca entropia é bem separada.

Ou seja: a entropia é uma métrica que mede o caos de uma distribuição. Em termos simples, isto é "quão misturadas as classes estão." A sua medida é dada pela soma da fração que uma classe representa no todo do cluster, vezes o logaritmo do mesmo valor. Faz-se isso para todas as classes por cluster, e some os resultados de forma ponderada e tem-se a entropia. Ou seja:

m = #Elem. da classe 1/#Elem. do agrupamento

Entropia no agrupamento = -1* m * log(m)

Some para todas as classes

Fazendo o cálculo para o exemplo dado:

*Entropia Ag.1: -1 * ((2/6)*(log(2/6)) + (4/6)*(log(4/6)) + (0/6)*(log(0/6))) =* ***0.650***
*Entropia Ag. 2: -1 * (1/6)*(log(1/6)) + (4/6)*(log(4/6)) + (1/6)*(log(1/6))) =* ***1.252***
*Entropia Ag. 3: -1 * ((2/5)*(log(2/6)) + (0/5)*(log(0/5)) + (3/5)*(log(3/5))) =* ***0.971***

Entropia total: (0.650*(6/17)) + (1.252 *(6/17)) + (0.971*(5/17)) = 0.956

Como o objetivo e um algoritmo de agrupamento é separar as classes, quanto menor a entropia de seus resultados, melhor. Calcular a entropia dos seus dados também te ajuda a entender quão fácil – ou difícil – o trabalho do algoritmo vai ser. Quanto maior a entropia, menos separáveis os dados são. E, é claro, quanto mais puros os clusters, melhor.

Homogeneidade, Completude e Métrica V

Homogeneidade e completude são métricas cuja noção deriva da ideia da entropia, explicada na seção anterior. Mais especificamente, essas métricas são casos mais específicos da entropia. A homogeneidade mede a concentração das classes em agrupamentos únicos. Quanto menos homogêneo, mais classes o agrupamento tem. A completude “olha” para as classes: cada classe deve ter representantes no menor número de agrupamentos possível.

Ou seja, completude e homogeneidade são métricas “espelhadas”. Nesse caso, isso significa que as duas falam a mesma coisa, mas a completude fala das classes e a homogeneidade dos agrupamentos. A métrica V é a média harmônica dessas duas métricas, como o F-score é para algoritmos de classificação.

Para a homogeneidade, o ideal é que cada agrupamento contenha elementos de só uma classe. Para essa métrica, **quanto maior o valor, melhor**. Quanto menos classes num agrupamento, melhor. “Cada balão tem um gás diferente”

Para a completude, o ideal é que cada classe esteja contida em um só cluster. **Quanto maior a completude, melhor**. Quanto menos agrupamentos contendo uma classe, melhor. “Em quantos balões eu encontro todo o hidrogênio que procuro?”

Figura 20 - Fórmula da Homogeneidade.

$$1 - \frac{H(C|K)}{H(C)}$$

$$1 - \frac{\text{Entropia dos agrupamentos nas classes}}{\text{Entropia dos agrupamentos}}$$

Figura 21 - Fórmula da Completude.

$$1 - \frac{H(K|C)}{H(K)}$$

$$1 - \frac{\text{Entropia das classes nos agrupamentos}}{\text{Entropia das classes}}$$

Como exemplo, considere o conjunto de dados, e os respectivos agrupamentos encontrados, apresentados na figura 22.

Figura 22 - Um novo conjunto de dados e os agrupamentos encontrados.

A homogeneidade e a completude são encontradas pelos cálculos abaixo:

Homogeneidade

H(C\|K) **=** *((2/6)*log(2/2) + (1/6)*log(1/2) + 0) + ((2/6)*log(2/2) + (1/6*log(2/2) + 0)*

H(C) **=** *((3/6)*log(3/6) + (3/6)*log(3/6))*

1 - H(C|K)/H(C) = +-0.66

Completude

H(K|C) ***=*** *((2/6)*log(2/3) + (1/6)*log(1/3) + 0) + ((2/6)*log(2/3) + (1/6)*log(1/3) + 0)*

H(K) ***=*** *((3/6)*log(3/6) + (3/6)*log(3/6) + (3/6)*log(3/6))*

1 - H(C|K)/H(C) = +-0.41

A Métrica-V é a média harmônica das duas anteriores. Ou seja:

Métrica-V

2 * ((0.66*0.41)/(0.66+0.41)) = 0.51

Índice de Rand

O Índice de Rand é uma métrica de similaridade entre diversos agrupamentos de um mesmo conjunto de dados. Intuitivamente, podemos dizer que ele busca a fração de agrupamentos de pontos nos quais concordam. Por exemplo, considere que temos 5 pontos, identificados de 1 a 5 e fizemos dois agrupamentos diferentes deles.

Tabela 17 – Dois agrupamentos distintos para o mesmo conjunto de 5 pontos.

	1	**2**	**3**	**4**	**5**
Agrupamento 1	Grupo A	Grupo B	Grupo B	Grupo A	Grupo A
Agrupamento 2	Grupo B	Grupo A	Grupo B	Grupo A	Grupo A

Para calcular o índice, é preciso categorizar cada par de pontos possível. Os pares, nesse caso, são: [1,2], [1,3], [1,4], [1,5], [2,3], [2,4], [2,5], [3,4], [3,5], [4,5]. Para cada par, classifique de acordo com uma das categorias abaixo. Este par…

- Pertence ao mesmo grupo no primeiro agrupamento **e** no segundo?
- Não pertence ao mesmo grupo em nenhum agrupamento?
- Pertence ao mesmo grupo só no primeiro agrupamento?

- Pertence ao mesmo grupo só no segundo agrupamento?

Tabela 18 – As quatro classificações possíveis pra o Índice de Rand.

	Mesmo grupo em A	**Grupos diferentes em A**
Mesmo grupo em B	a	b
Grupos diferentes em B	c	d

Fazendo a classificação para todos os pares de pontos, obtemos os seguintes resultados: [1,2]: d, [1,3]: **b**, [1,4]: c, [1,5]: c, [2,3]: c, [2,4]: **b**, [2,5]: **b**, [3,4]: d, [3,5]: d, [4,5]: **a.** A fórmula para se calcular o Índice de Rand é simples: some os pares com os quais os dois agrupamentos concordam, e divida pela quantidade de pares.

Figura 23 - Fórmula para cálculo do Índice de Rand.

$$\frac{a+b}{a+b+c+d}$$

Para o nosso exemplo, isso dá 4 dividido por 10, ou seja, 40%.

Capítulo 5. Parametrização de Algoritmos

Até agora este curso se focou nas métricas de qualidade usáveis, para avaliar e entender os resultados dos algoritmos para as principais categorias de problemas em aprendizado de máquina: regressão, classificação, classificação multilabel e agrupamento. Esta seção dá um passo adiante, e se concentra em outra etapa de aprimoramento de resultados que auxiliam o analista de dados a escolher o modelo que solucionará o seu problema: a parametrização dos algoritmos.

Afinal de contas, pergunta-se: quando você vai executar o algoritmo K-vizinhos mais próximos para solucionar um problema de classificação, ou um problema de classificação multilabel, quantos vizinhos são esses K? Um valor de K muito pequeno pode me dar resultados totalmente errados, assim como um valor muito grande. Qual valor de K é bom para o meu problema?

E seguindo adiante nesse raciocínio, há mais: quantas folhas minha árvore de decisão multilabel vai permitir cada elemento ocupar? Quantos agrupamentos meu algoritmo de agrupamento hierárquico deve formar? O objetivo desta seção é fornecer conhecimento de ferramentas que o ajudarão a tomar decisões melhores nesse sentido.

Os hiperparâmetros dos algoritmos

É preciso, antes de mais nada, diferenciar os parâmetros dos algoritmos dos seus hiperparâmetros, que são os que buscaremos otimizar. No contexto de aprendizado de máquina, os parâmetros de um algoritmo são aqueles que o próprio algoritmo busca ajustar para adequar o seu modelo aos dados.

Por exemplo, pense numa regressão linear. Uma reta é caracterizada pela equação $ax + b$, sendo a e b os parâmetros desta reta. Um algoritmo de regressão linear busca ajustar os valores de a e b, partindo de uma dupla de valores quaisquer, encontrar uma reta que minimize a sua métrica de qualidade escolhida – por exemplo, o erro médio quadrático. No caso de um algoritmo SVM, os parâmetros são os pesos

do vetor que ele busca. No caso de uma rede neural, estes são os pesos que ligam os neurônios. Esse são parâmetros dos algoritmos.

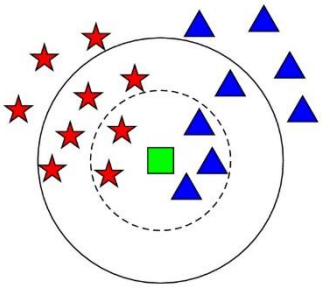
Os hiperparâmetros são as configurações dos algoritmos. Os ajustes e configurações escolhidos para sua melhor execução. No caso de uma rede neural, estes seriam a quantidade de nós da rede, a quantidade de camadas, a largura de cada camada, para citar alguns exemplos. O número de agrupamentos final, o número de vizinhos a considerar, a métrica de similaridade entre pontos, todos esses são hiperparâmetros: eles são fixos para cada execução do algoritmo.

As técnicas que estudaremos nas próximas seções são formas mais sofisticadas de escolher esses hiperparâmetros. Estudaremos a busca exaustiva em grid, a otimização aleatória e veremos princípios gerais, dicas que ajudam nesse processo, independentemente da técnica de escolha de parâmetros que se optar por usar.

Busca exaustiva em grid

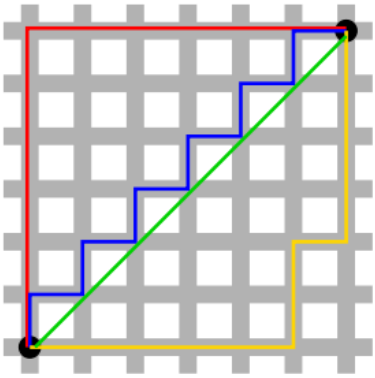
A busca exaustiva em grid é a forma mais simples de se escolher os melhores de um conjunto de parâmetros – a força bruta. Imagine, por exemplo, que você vai rodar o algoritmo KNN, o K-vizinhos mais próximos, sobre um conjunto de dados qualquer. É preciso definir dois hiperparâmetros ao se instanciar o algoritmo: a métrica de similaridade ou distância, entre os pontos, e a quantidade de vizinhos a se levar em consideração.

Figura 24 - O algoritmo KNN, ou K-vizinhos mais próximos.



Suponhamos que você, baseado em outras execuções de KNN sobre dados semelhantes, decida que seu k ideal não é maior que trinta. E entre as várias métricas de similaridade, você escolhe duas: a euclidiana, que é a distância vetorial entre os pontos, e a manhattan, ou distância do taxi, que é a soma absoluta das distâncias cartesianas entre os pontos.

Figura 25 - A distância manhattan entre dois pontos é a soma absoluta das suas distâncias cartesianas. É como calcular a distância entre os pontos em quarteirões percorridos e não em linha reta.



Assim sendo, o seu *grid* de busca será *k = [0, 30], distance = {`euclidean`, `manhattan`}*. O algoritmo, então, buscará em seu grid, um a um, pelos hiperparâmetros que derem o melhor resultado, segundo a métrica que se escolher. Ou seja, ele executará da seguinte forma:

- Para k = 1, distancia euclidiana
- Para k = 1, distância manhattan
- Para k = 2, distância euclidiana
- Para k = 3, distância manhattan
- ...

Ou seja. Sendo 30 valores possíveis para k e 2 para a distância, o algoritmo KNN será executado 60 vezes. Agora imagine que você tem três hiperparâmetros

diferentes, e cada um deles tem vinte valores possíveis. Isso dá 20 * 20 * 20 = 8000 execuções! Claramente isso não é escalável, e este é o maior problema da busca exaustiva em grid. Se os seus hiperparâmetros ficam um pouco mais complexos, ele já demora muito para executar.

É possível mitigar isso deixando a busca menos granular. No exemplo acima você pode, por exemplo, buscar somente os valores ímpares de k e não os pares, já que a diferença de um k para o outro não é tão significativa. Mas isso ainda traz problemáticas:

- Na medida em que o k fica menos granular, você vai perdendo precisão e corre o risco de pular os bons valores para o seu problema. Se o seu k fosse de 5 em 5 e o melhor valor fosse 7, tanto os valores 5 quanto 10 poderiam acabar ruins e você jamais encontraria um bom valor.

- Às vezes um parâmetro ou outro não influenciam nos seus resultados. Um parâmetro específico pode ser 0 ou 8000 e isso não faz diferença nos resultados. A busca exaustiva em grid não percebe isso e se poupa do trabalho, ela multiplica por 8000 a quantidade de vezes que seu algoritmo executará.

Assim sendo, a busca em grid garante o melhor resultado, dado que ela passa por todos os parâmetros, mas perde precisão na medida em que o problema aumenta, para ser executada em tempo viável e acaba não sendo escalável.

Otimização aleatória de parâmetros

A otimização aleatória de parâmetros se baseia numa ideia simples: “caminhe” pelos parâmetros possíveis e, a cada passo, veja se houve melhora ou piora. Havendo melhora, dê outro passo nessa direção. Sem melhora, caminhe em outra direção e repita. Ela parte, vê-se, do princípio que parâmetros bons são semelhantes, mas não está limitada a isso.

Seu algoritmo, então, é algo como:

- Inicie com uma tupla, um “par” de *t* elementos, aleatória de hiperparâmetros.
- Varie-os um pouco. Dê um ‘salto’.
- Se houve melhora, varie-os mais um pouco nessa direção.
- Se houve piora, vá para outra direção.
- Repita.

Ou seja: a otimização aleatória se inicia com k = 20. No primeiro passo, ela varia o k para 16 e percebe que há piora nos resultados. Ela, então, volta a 20 e varia o k para 23 no passo seguinte, e percebe melhora. Ela segue a partir daí até não ter mais para onde melhorar.

É possível se inferir, a partir daí, que a otimização aleatória não garante o melhor conjunto de hiperparâmetros, e isto está correto. Ela garante um bom resultado, mas não o melhor. A depender dos passos que ela der, ela pode perder o melhor. Experimentalmente, no entanto, vê-se que ela é bem mais escalável, leva uma fração do tempo da busca exaustiva em grid para retornar seu resultado e tem qualidade comparável.

É possível customizar a forma como o algoritmo salta, associando a ele diferentes distribuições probabilísticas. A ideia é que a distribuição probabilística se assemelha à distribuição de qualidade entre os hiperparâmetros. Qual usar é algo que depende de do problema em questão. Na dúvida, deve-se testar vários experimentos, com uma fração dos seus dados, se o teste, com todos eles, for muito lento.

Dicas para a busca de parâmetros

Há algumas boas práticas que se pode adotar para ter resultados melhores mais rapidamente na busca por parâmetros. Elas são: especifique suas métricas de

qualidade; use mais de uma métrica quando pertinente; separe seus dados em treino e teste; use paralelismo.

- **Especifique suas métricas de qualidade:** como padrão, as implementações dos algoritmos de otimização de hiperparâmetros utilizam a acurácia, ou métrica similar, como critério de qualidade de resultados. A depender do seu problema, no entanto, essa métrica pode não ser útil, não fazer sentido ou ser pior que outra. Defina a métrica que o algoritmo vai usar para melhores resultados.

- **Utilize mais de uma métrica, se necessário:** É possível usar mais de uma métrica também. Você pode querer usar o F-Score e a Acurácia, num problema de classificação. Num problema de classificação multilabel, você pode querer usar a Perda de Hamming e a Perda 0-1. Use esta opção.

- **Separe seus dados em treino e teste:** Para qualquer algoritmo, a modelagem que você cria e aufere usando as métricas de qualidade representa uma *amostra* dos dados que o mundo real vai lhe dar. Uma fração do que você realmente vai ver. Conforme veremos em mais detalhes o capítulo 6, uma ótima prática é dividir os seus dados em alguma proporção – 70-30, por exemplo – e usar a maior para criar o seu modelo, como “treino”. Em seguida, teste o seu algoritmo, com os parâmetros encontrados, nos dados de teste. Isso simula uma situação onde seu algoritmo já parametrizado encontrou dados desconhecidos e é uma boa forma de auferir a robustez do modelo que você criou.

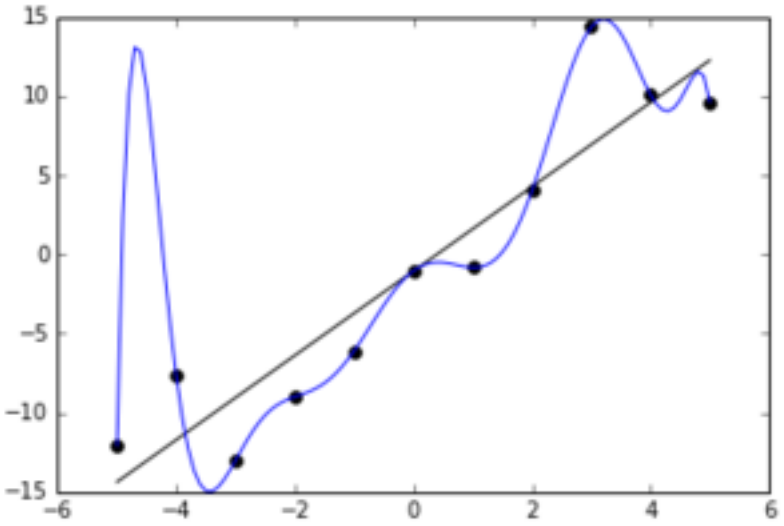
- **Use paralelismo:** Utilize a opção de executar testes em paralelo, se o seu algoritmo permitir. Isso acelera em muitas vezes o algoritmo e te permite fazer testes mais extensos. Caso o algoritmo não te dê essa opção, divida os seus hiperparâmetros em grupos e rode várias instâncias do algoritmo, uma com cada subgrupo.

Capítulo 6. Validação Cruzada

A validação cruzada é uma das formas mais poderosas – e importantes – de se garantir que o desempenho de um modelo será bom após o fim do ciclo de treinos e parametrizações. Fundamentalmente, o que ela faz é testar como o modelo se comporta diante de dados desconhecidos – isto é, de dados que não foram usados em seu treino.

Ou seja: a validação cruzada é a forma de garantir que seu modelo não sofre de overfitting.

Figura 26 - Regressão linear contra regressão curva com overfitting.



O overfitting, em essência, é uma situação onde um modelo se adequa perfeitamente aos dados de treino, mas tem desempenho ruim em casos reais. A lógica é simples: os dados que você usa para gerar o seu modelo são sempre *parciais*. Sempre representam uma parte do que o mundo real pode "jogar" em seu modelo. Eles não representam perfeitamente o que o seu modelo encontrará no mundo real, em suma. Assim sendo, uma vez que os seus dados são uma aproximação do que há no mundo real, um modelo que se *aproxime* dos seus dados, havendo certo erro, é preferível a um modelo perfeito.

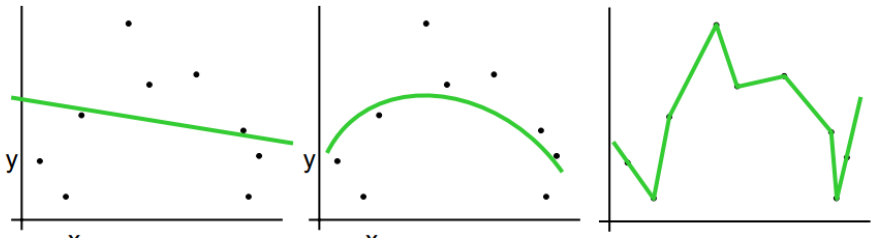
A validação cruzada é uma técnica que testa seu modelo diante de dados desconhecidos, para simular uma situação em que seu problema está no mundo real, diante de dados nunca antes vistos e não usados em seu treino. Um bom desempenho na validação cruzada indica que o modelo é robusto e lida bem com dados desconhecidos. Um desempenho ruim evidencia o overfitting: o modelo é ótimo para o treino, mas péssimo fora dele.

Nas demais seções deste capítulo, serão estudadas três técnicas de validação cruzada: “deixe P elementos de fora”, “divisão em treino e teste” e “validação cruzada em k grupos”.

Separação em treino e teste e “deixar P elementos de fora”

Todas as técnicas de validação cruzada seguem a mesma lógica: construa um modelo utilizando parte dos dados que você tem para treino e teste-o com a parte dos dados que ficou de fora. Para fins de exemplo, considere as três regressões da figura 27.

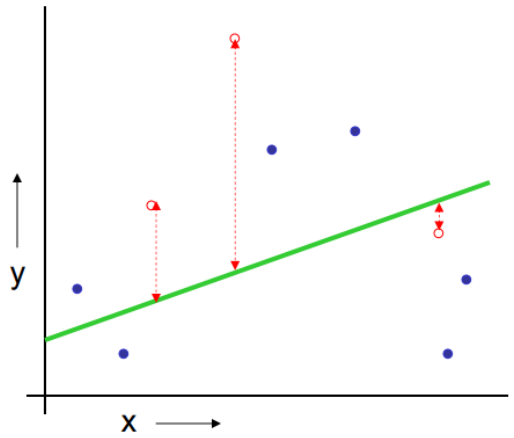
Figura 27 - Uma regressão linear, uma regressão curva e uma regressão "ligue os pontos". Qual melhor representa a estrutura dos dados sob análise?



Na figura 27, temos três regressões feitas sobre o mesmo conjunto de dados. Uma regressão linear, uma regressão curva, e uma regressão estilo “ligue os pontos”, que traça retas entre os pares de pontos mais próximos dos dados analisados para se construir. Considerando uma métrica de qualidade como o erro médio quadrático, vemos com clareza que, ao contrário das demais, o desempenho da técnica de ligar os pontos se mostra superior, com erro zero.

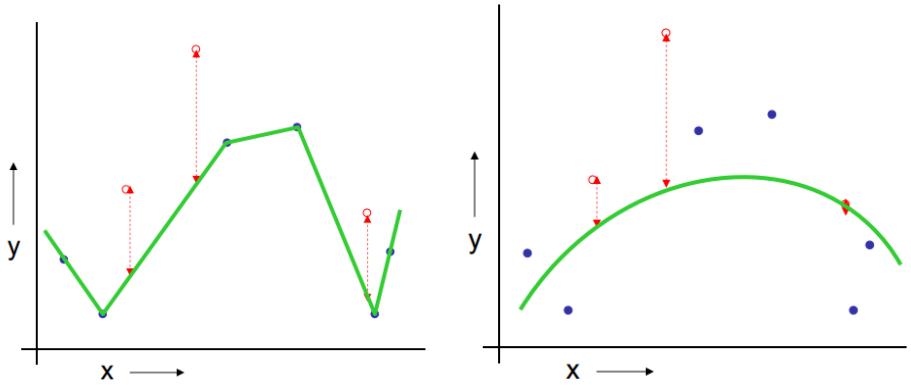
Entretanto, ao se fazer a mesma análise utilizando a validação cruzada, vê-se que os resultados mudam. Primeiramente, façamos uma validação utilizando a divisão dos dados em treino e teste. Essa técnica é simples: pegue uma fração pequena de seus dados, por exemplo, 30%, e a separe. Construa seu modelo utilizando os 70% dos dados restantes e teste o desempenho de seu modelo sobre os 30% separados.

Figura 28 - Validação cruzada sobre a regressão linear. Constrói-se o modelo sobre a maior parte dos seus dados e testa-se seu desempenho sobre a menor parte que foi separada. Aqui, o erro médio quadrático é de 2.4. Observa-se que a regressão construída é bem diferente da construída sobre 100% dos dados.



Vê-se que a regressão mudou, já que um conjunto de dados diferente foi utilizado em sua construção. Os pontos marcados de vermelho são os pontos sobre os quais seu modelo é testado. São pontos “desconhecidos” para a regressão.

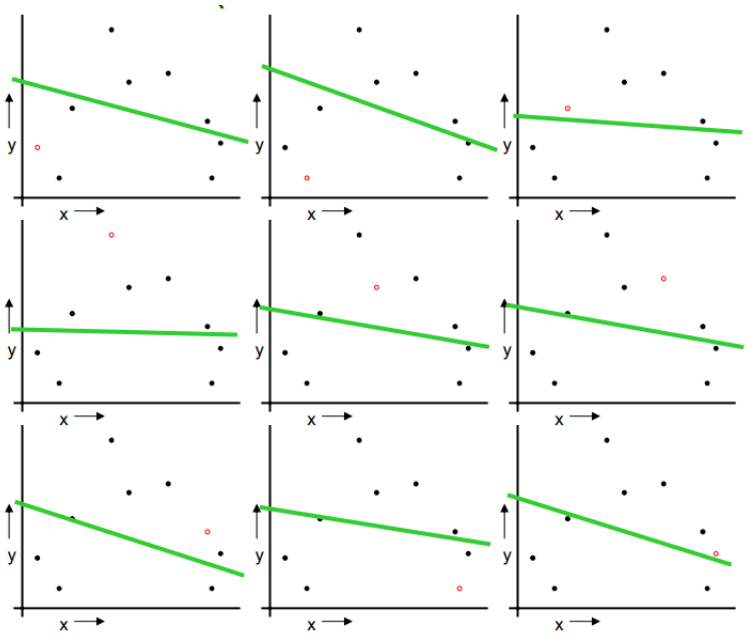
Figura 29 – Regressões, de ligue os pontos e curva, testadas com a validação cruzada, sem 30% de seus pontos. Os erros médios quadráticos para as duas são, respectivamente, 2.2 e 0.9.



Podemos observar que, quando confrontadas com dados diferentes dos usados em seu treino, a ótica sobre as regressões muda. Enquanto a regressão linear e a regressão curva são boas, mas imprecisas, conservam seus resultados, os resultados da regressão que teve desempenho perfeito nos treinos são bem ruins. É a consequência de se adequar bem demais ao seu treino. Overfitting.

As vantagens da validação cruzada de divisão de teste e treino é que o método é simples em sua execução e análise: o modelo que obtiver o menor erro na validação cruzada é o melhor a ser usado para aquele conjunto de dados. Como desvantagem, temos o fato de que você perde parte dos seus dados de treino para fazer a sua análise.

Figura 30 - Validação cruzada com a técnica de deixar P elementos de fora, P sendo igual a 1. Faz-se a média dos resultados de cada regressão e escolhe-se o algoritmo com o menor erro.



Na técnica de deixar P elementos de fora, o que se faz é parecido: a cada iteração, remove-se P elementos dos seus dados, e faz-se a validação cruzada da mesma forma que com a técnica de divisão de treino e teste: constrói-se a regressão com os dados restantes, e testar-se-á com os dados removidos. A diferença é que, aqui, faz-se uma validação cruzada para cada possibilidade de ponto a ser removida, como se mostra na figura 30. Calcula-se, em seguida, a média dos erros da validação cruzada para todos esses casos.

O objetivo, aqui, não é gerar um modelo robusto pura e simplesmente: é ver qual algoritmo tem, na média, a maior robustez a dados desconhecidos. Faz-se o teste com todos os algoritmos sob análise e o que tiver o melhor resultado médio para a métrica de qualidade escolhida deve ser escolhido, sendo o algoritmo que tende a ter melhores resultados quando diante de dados desconhecidos.

A vantagem desta técnica é que ela dá um resultado mais robusto, uma vez que não se gera um modelo usando uma fração arbitrária dos seus dados. A média de todas as validações te diz que, diante de grande variação de dados entrada, como

seu modelo se comporta. A desvantagem é o preço da técnica. Se você usa p = 2, por exemplo, para um conjunto de dados com mil pontos, a validação cruzada fará a média de desempenho de 500.000 modelos. Em compensação, após a escolha do melhor algoritmo, é possível construir o modelo usando 100% dos seus dados, pois o teste mais extensivo garante o desempenho que ele terá.

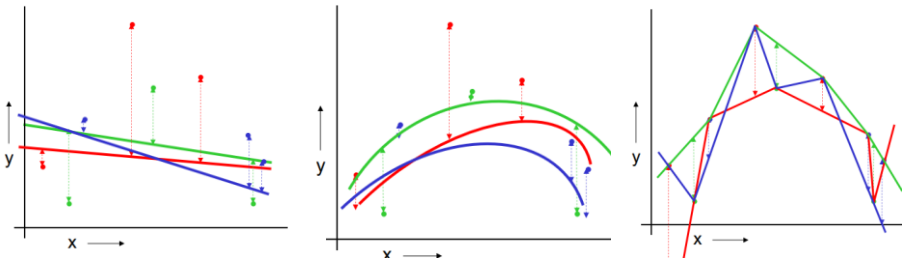
Faz-se o tradeoff, então: a divisão em treino e teste perde parte dos seus dados, mas é barata. A deixar P elementos de fora é muito cara, mas dá um resultado mais poderoso. Na próxima seção, estudamos a técnica de validação cruzada em K grupos, que busca um meio termo entre essas duas técnicas, para ter parte dos benefícios das duas.

Validação cruzada em K-grupos

A validação cruzada em K-grupos, ou *K-fold cross validation*, é a técnica que busca unir os benefícios da técnica de divisão de treino e teste e da técnica de deixar P elementos de fora. Sua lógica é simples como as demais:

- Divida seus dados em K-grupos.
- A cada iteração, faça uma regressão com K-1 grupos de dados e teste-a com o grupo restante.
- A média destas regressões que tiver melhor resultado será sua regressão escolhida.

Figura 31 - Modelos sendo criados utilizando a validação cruzaada em K-grupos. A média das regressões que tiver melhor desempenho deve ser escolhida. Aqui, os erros médios quadráticos encontrados são, respectivamente 2.05, 1.11, 2.93.



Vemos que a técnica busca agregar as vantagens das duas outras. Na prática, ela se mostra a escolha mais frequente para testar a robustez de modelos.

Tabela 19 - Trade offs das três técnicas estudadas de validação cruzada.

Técnica	Vantagens	Desvantagens
Treino e teste	**Barata**	**Perde 30% dos dados**
P elementos de for a	**Não desperdiça dados na análise**	**Cara**
K-grupos	**Desperdiça poucos dados na análise (10% para 10 grupos)**	**Ainda desperdiça dados** **K vezes mais cara que a técnica de treino e teste.**

Exemplo (continuação) - Classificação de vinhos

O exemplo abaixo continua o que vimos no capítulo anterior: a classificação de vinhos baseado em alguns de seus atributos. Aqui, além de gerar os modelos e

comparar suas métricas de qualidade, fazemos a validação cruzada de ambos e comparamos seus desempenhos, tendo como base a acurácia. Vemos, então, como ambos os modelos provavelmente se comportarão quando no mundo real, diante de dados desconhecidos.

```
60 # Na validação cruzada
61 from sklearn.model_selection import cross_val_score
62 cv_rfc = cross_val_score(rfc, X, y)
63 cv_knn = cross_val_score(knn, X, y)
64 print("\nValidação cruzada: {0} vs {1}".format(cv_knn, cv_rfc))
65
66 sum_cv_rfc = 0
67 for cv_score in cv_rfc:
68   sum_cv_rfc += cv_score
69
70 print("\nResultado Random Forest: {0}".format(sum_cv_rfc/5))
71
72 sum_cv_knn = 0
73 for cv_score in cv_knn:
74   sum_cv_knn += cv_score
75
76 print("\nResultado KNN: {0}".format(sum_cv_knn/5))
```

A seguir, fazemos a otimização de hiperparâmetros dos nossos dois classificadores com a função *GridSearchCV,* que embute a sua lógica a validação cruzada e a otimização. Definimos um objeto que diz quais parâmetros de cada classificador queremos explorar, por quais intervalos e geramos o nosso modelo normalmente. O retorno é um objeto que tem todas as informações acerca do modelo gerado, inclusive quais foram os melhores hiperparâmetros para ele.

```
78 # -----------------
79
80 #Buscando hiper parâmetros
81 from sklearn.model_selection import GridSearchCV
82
83 #RFC
84 parameters = {'min_samples_split':(2,6)}
85 rfc_hps = GridSearchCV(rfc, parameters)
86 rfc_hps.fit(X, y)
87 print("Melhor valor para min_samples_split: {0}".format(rfc_hps.best_params_['min_samples_split']))
88
89 #KNN
90 parameters = {'n_neighbors':(1,20)}
91 knn_hps = GridSearchCV(knn, parameters)
92 knn_hps.fit(X, y)
93 knn_hps.best_params_['n_neighbors']
94 print("Melhor valor para n_neighbors: {0}".format(knn_hps.best_params_['n_neighbors']))
```

Capítulo 7. Curvas de Validação

A análise da curva de validação é uma técnica que permite visualizar, através do acúmulo de diversos modelos construídos com quantidades diferentes de dados de treino, se seu modelo tende a estar com underfitting, overfitting e se vale a pena buscar mais dados de treino ou se não adianta mais aumentar sua quantidade de dados, porque o modelo já atingiu seu potencial de análise.

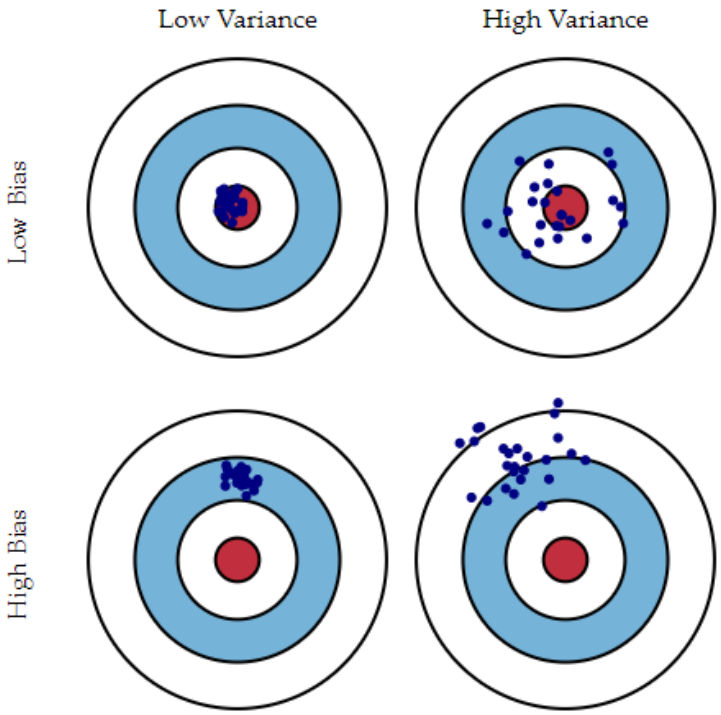
A tendência e variância são características de algoritmos que ajudam a compreender seu comportamento e prever se seus modelos serão bons e, se ruins, como falharão.

Tendência e Variância

A tendência de um algoritmo representa aquilo que ele assume para fazer suas previsões. Um erro em função de tendência é um erro que ocorre decorrente dela. Por exemplo: um algoritmo de regressão linear sempre *presume* que os dados têm uma relação linear. Se a relação dos dados for curva, boa parte do erro que sua regressão tiver decorrerá da tendência que os modelos do algoritmo possuem por sempre presumirem uma relação linear entre os dados. Um algoritmo como o KNN presume que um elemento terá a mesma classe da maior parte de seus vizinhos. Ele tende a fazer suas classificações dessa forma, portanto. Elementos isolados em meio a classes diferentes de si sempre *tenderão* a ter classificação incorreta.

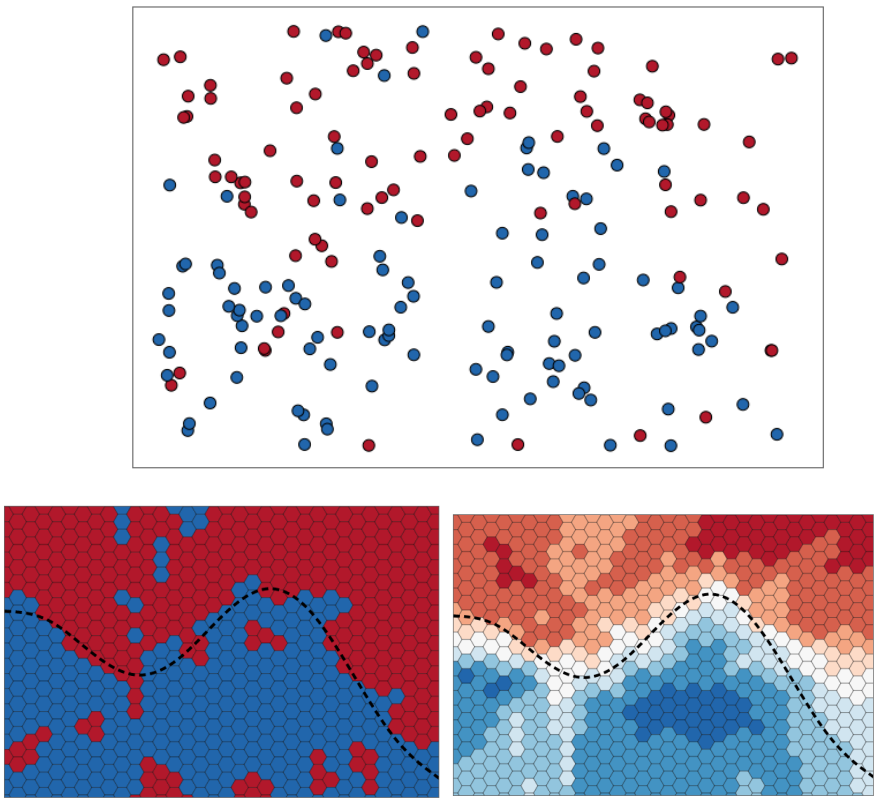
Já variância, por outro lado, é o tanto que o desempenho de um algoritmo sobre diferentes conjuntos de dados é previsível. Um algoritmo com muita variância é ótimo para um conjunto de dados e muito ruim para outro conjunto não tão diferente assim. Um algoritmo com pouca variância tem desempenhos semelhantes para conjuntos de dados semelhantes.

Figura 32 - Cada algoritmo tem suas características de tendência e variância. O ideal é o algoritmo que tem ambas baixas, mas geralmente tem-se um trade-off, onde busca-se o ponto ótimo onde as duas estão igualmente baixas, o tanto quanto possível.



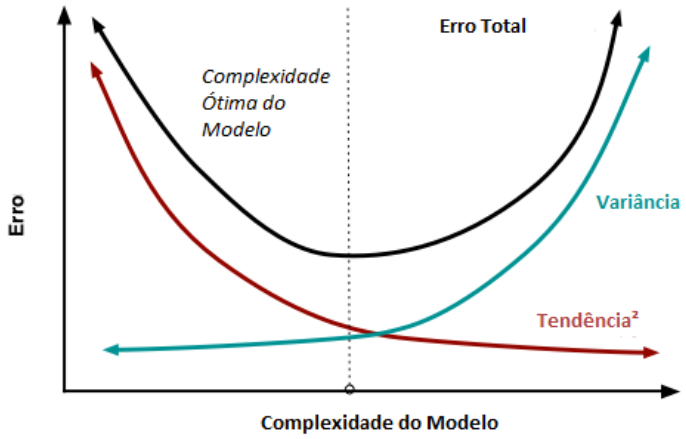
As características de tendência e a variância de um algoritmo podem variar de acordo com a forma com que ele é parametrizado. O KNN, por exemplo, apresenta muita variância e pouca tendência para valores baixos de K, e o inverso para valores altos. A lógica é simples. Com valores de K baixos, o algoritmo considera uma quantidade muito pequena de vizinhos para classificar seus pontos. Ou seja, qualquer ponto de classe distinta dos seus k pontos mais próximos terá classificação errada e isso pode acontecer em qualquer local dos seus dados. Quando os valores de K forem altos, todos os pontos de uma determinada região terão a classe que a maior parte dos pontos dali tem. Os erros, então, tendem a acontecer em regiões específicas.

Figura 33 – A tendência e a variância do algoritmo KNN varia conforme o valor de k aumenta. Para valores baixos, o algoritmo se comporta de forma errática, e pode errar muito ou pouco, a depender de leves variações em seus dados. Quando o valor de k é alto, por outro lado, os erros tendem a se concentrar em regiões com concentração maior de pontos de uma única classe. Os pontos que são minoria sempre estarão errados ali.



Tendência e variância, portanto, são características únicas a cada algoritmo que gera modelos, e podem variar a depender da hiperparametrização destes algoritmos. Elas ajudam a entender como o algoritmo tende a se comportar naquele cenário, ou seja, como serão os erros que ele terá.

Figura 34 - Busca-se uma hiperparametrização que encontre o ponto ótimo onde a tendência e a variância são tão baixas quanto possível. A localização destes pontos varia a depender dos dados do seu problema.

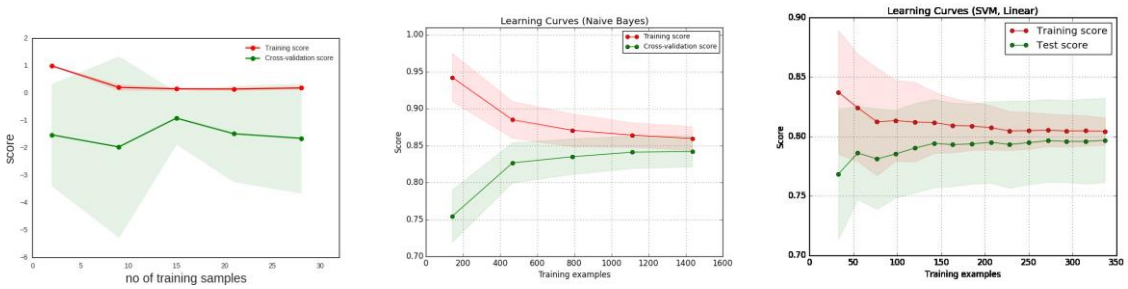


Curva de aprendizado

A curva de aprendizado é um gráfico que confronta a sua métrica de qualidade, no eixo Y, com a quantidade de exemplos de teste usados na construção do seu modelo, no eixo X. Faz-se duas curvas: a do desempenho dos seus modelos, e a do desempenho “esperado” para aquela quantidade de exemplos.

O desempenho esperado pode ser, por exemplo, o desempenho do algoritmo sobre um conjunto de teste separado para comparar diferentes algoritmos, ou o resultado de uma validação cruzada. Dado que a validação cruzada diz qual erro médio aquele algoritmo tende a ter para aqueles dados, comparar a qualidade média da validação cruzada com a qualidade real do seu modelo é uma forma de confrontar “o que eu tenho” com “o que eu quero”.

Figura 35 – Diferentes tipos de curvas de aprendizado.



E que tipo de informação um gráfico como este trás? Primeiramente, dado que o gráfico confronta o modelo que você tem com o desempenho que você espera, quando as curvas do gráfico se aproximam, você tem a situação em que seu modelo está se aproximando do desempenho máximo que ele consegue ter para aqueles dados. Ou seja, o desempenho do seu modelo está se aproximando do “esperado” (por exemplo, o desempenho médio previsto pela validação cruzada).

Um gráfico onde tanto o seu modelo quanto a comparação têm a quantidade de erros alta e muito parecida para diferentes quantidades de exemplos de teste, é um gráfico onde seu modelo não tende a melhorar e não é bom: há alta tendência no algoritmo, porque os resultados são consistentemente ruins. Há underfit.

Um gráfico onde os desempenhos comparados não convergem ou onde o desempenho do modelo varia muito conforme a quantidade de exemplos, é um gráfico de um modelo com alta variância. Um gráfico onde o modelo tem desempenho muito melhor que o esperado pela comparação é um gráfico que mostra overfit.

Por outro lado, quando o desempenho do modelo converge com o esperado e a qualidade destes é alta, é um modelo muito bom. O convergir do modelo com o esperado é sempre um sinal de que aquele modelo atingiu o seu potencial: dar mais dados a ele não trará melhorias.

Vemos, portanto, que a curva de validação é uma forma de auferir a tendência, variância, presença de underfit e overfitting no seu modelo. Além disso, é uma técnica que te ajuda a decidir se vale a pena acrescentar mais testes ao seu treino ou se ele não tende a melhorar mais.

Referências

BARNARD, Kobus. et. al. Matching Words and Pictures. In: *Journal of Machine Learning Research*, vol 3, p. 1107-1135.

BELVECCHIOUK. Rand Index in Statistics - A Worked Example - Cluster Analysis. Disponível em: <https://www.youtube.com/watch?v=6rjTIwn0yWI>. Acesso em: 10 jun. 2020.

BERGSTRA, James; BENGIO, Yoshua. Random search for hyper-parameter optimization. In: *Journal of Machine Learning Research*, v. 13, n. Feb, 2012, p. 281-305.

BERNARDINI, F. C.; DA SILVA, R. B.; MEZA, E. B. M. Analyzing the influence of cardinality and density characteristics on multi-label learning. *Proc. X Encontro Nacional de Inteligencia Artificial e Computacional-ENIAC 2013.* Rio de Janeiro, Brasil, 2013.

BRIGGS, Forrest. et. al. The 9th annual MLSP competition: New methods for acoustic classification of multiple simultaneous bird species in a noisy environment. In: *2013 IEEE International Workshop on Machine Learning for Signal Processing (MLSP)*, Southampton, 2013, p. 1-8.

CROSS VALIDATED. *About learning curves in Machine Learning.* Disponível em: <https://stats.stackexchange.com/questions/291379/about-learning-curves-in-machine-learning>. Acesso em: 10 jun. 2020.

CROSS VALIDATED. *How to calculate purity?* Disponível em:<https://stats.stackexchange.com/questions/95731/how-to-calculate-purity>. Acesso em: 10 jun. 2020.

CROSS VALIDATED. *How to choose a predictive model after k-fold cross-validation*. Disponível em: <https://stats.stackexchange.com/questions/52274/how-to-choose-a-predictive-model-after-k-fold-cross-validation>. Acesso em: 10 jun. 2020.

CROSS VALIDATED. *How to know if a learning curve from SVM model suffers from bias or variance?*. Disponível em: <https://stats.stackexchange.com/questions/220827/how-to-know-if-a-learning-curve-from-svm-model-suffers-from-bias-or-variance/220852#220852>. Acesso em: 10 jun. 2020.

CROSS VALIDATED. How to know if a learning curve from SVM model suffers from bias or variance? Disponível em: <https://stats.stackexchange.com/questions/220827/how-to-know-if-a-learning-curve-from-svm-model-suffers-from-bias-or-variance>. Acesso em: 10 jun. 2020.

CROSS VALIDATED. *What does AUC stand for and what is it?* Disponível em: <https://stats.stackexchange.com/questions/132777/what-does-auc-stand-for-and-what-is-it>. Acesso em: 10 jun. 2020.

DIPLARIS, Sotiris; MITKAS, Pericles A.; TSOUMAKAS, Grigorios; VLAHAVAS, I. Protein Classification with Multiple Algorithms. In: *Proc. 10th Panhellenic Conference on Informatics (PCI 2005)*, Volos, Grécia, 2005, p. 448-456.

ELITE DATA SCIENCE. *WTF is the Bias-Variance Tradeoff? (Infographic)*. Disponível em: <https://elitedatascience.com/bias-variance-tradeoff>. Acesso em: 10 jun. 2020.

Evaluation of clustering. Disponível em: <https://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html>. Acesso em: 10 jun. 2020.

FAWCETT, Tom. An introduction to ROC analysis. In: *Pattern Recognition Letters*. n. 8. v. 27. 2006. p. 861-874. Disponível em: <https://web.archive.org/web/20110827012406/https://cours.etsmtl.ca/sys828/REFS/A1/Fawcett_PRL2006.pdf>. Acesso em: 10 jun. 2020.

FORTMANN-ROE, Scott. *Understanding the bias-variance tradeoff*. 2012.

GIT HUB. *Código fonte do Scikit para cálculo de homogeneidade, completude, métrica V e outras*. Disponível em: <https://github.com/scikit-learn/scikit-

learn/blob/a24c8b46/sklearn/metrics/cluster/supervised.py#L365>. Acesso em: 10 jun. 2020.

KATAKIS, I.; TSOUMAKAS, G.; VLAHAVAS, I. *Multilabel Text Classification for Automated Tag Suggestion*. Proceedings of the ECML/PKDD 2008 Discovery Challenge, Antwerp, Bélgica, 2008.

MOORE. Andrew W. *Cross-validation for detecting and preventing overfitting*. School of Computer Science Carnegie Mellon University. Disponível em: <https://www.slideshare.net/guestfee8698/crossvalidation>. Acesso em: 10 jun. 2020.

MULAN. *Datasets multilabel.* Disponível em: <http://mulan.sourceforge.net/datasets-mlc.html>. Acesso em: 10 jun. 2020.

NIST. *Manhattan Distance.* Disponível em: <https://xlinux.nist.gov/dads/HTML/manhattanDistance.html>. Acesso em: 10 jun. 2020.

RAND, W. M. Objective criteria for the evaluation of clustering methods. In: *Journal of the American Statistical Association*. v. 66 (336), 1917, p.846–850.

RASTRIGIN, L.A. The convergence of the random search method in the extremal control of a many parameter system. In: *Automation and Remote Control*. v. 24, n. 10, 1963, p. 1337–1342.

READ. Jesse. *Multi-label Classification*. Universidad Carlos III de Madrid, Madri, Espanha, 2013. Disponível em: <https://users.ics.aalto.fi/jesse/talks/Multilabel-Part01.pdf>. Acesso em: 10 jun. 2020.

RITCHIE NG. *Machine Learning: Learning Curve*. Disponível em: <http://www.ritchieng.com/machinelearning-learning-curve/>. Acesso em: 10 jun. 2020.

ROSEBROCK. Adrian. *How to tune hyperparameters with Python and scikit-learn*. Pyimagesearch, 2016. Disponível em:

<https://www.pyimagesearch.com/2016/08/15/how-to-tune-hyperparameters-with-python-and-scikit-learn/>. Acesso em: 10 jun. 2020.

ROSENBERG, Andrew; HIRSCHBERG, Julia. V-measure: A conditional entropy-based external cluster evaluation measure. In: *Proceedings of the 2007 joint conference on empirical methods in natural language processing and computational natural language learning (EMNLP-CoNLL).* 2007.

SCIKIT LEARN. *3.2. Tuning the hyper-parameters of an estimator*. Disponível em: <http://scikit-learn.org/stable/modules/grid_search.html>. Acesso em: 10 jun. 2020.

SCIKIT LEARN. *3.5. Validation curves: plotting scores to evaluate models*. Disponível em: <http://scikit-learn.org/stable/modules/learning_curve.html>. Acesso em: 10 jun. 2020.

SHALIZI, Cosma. R^2. In: *Lecture 10: F-Tests, R2 , and Other Distractions*, 2015. p. 16-19. Disponível em: <http://www.stat.cmu.edu/~cshalizi/mreg/15/lectures/10/lecture-10.pdf>. Acesso em: 19 jun. 2020.

SHEEHAN, David. *Clustering with Scikit with GIFs*. Disponível em: <https://dashee87.github.io/data%20science/general/Clustering-with-Scikit-with-GIFs/>. Acesso em: 10 jun. 2020.

STACK OVERFLOW. *How to calculate clustering entropy? A working example or software code [closed]*. Disponível em: <https://stackoverflow.com/questions/35709562/how-to-calculate-clustering-entropy-a-working-example-or-software-code/35780505#35780505>. Acesso em: 10 jun. 2020.

STACK OVERFLOW. *Input matches no features in training set; how much more training data do I need?* Disponível em: <https://stackoverflow.com/questions/35077270/input-matches-no-features-in-training-set-how-much-more-training-data-do-i-need?rq=1>. Acesso em: 10 jun. 2020.