SUMÁRIO

O QUE VEM POR AÍ? 3
HANDS ON 4
SAIBA MAIS 5
MERCADO, CASES E TENDÊNCIAS 22
O QUE VOCÊ VIU NESTA AULA? 23
REFERÊNCIAS 24

O QUE VEM POR AÍ?

No aprendizado de máquina supervisionado, o processo de otimização e ajuste fino são etapas cruciais para alcançar o melhor desempenho possível de um modelo. Através de técnicas refinadas, podemos ajustar os parâmetros do modelo de forma precisa, aprimorando sua capacidade de generalização e robustez.

Nesta aula final, exploraremos as etapas da otimização de modelos e ajuste fino em machine learning supervisionado, desde seus fundamentos teóricos até a implementação prática com códigos em Python e as bibliotecas Pandas, Numpy e Scikit-learn. Prepare-se para dominar esta etapa essencial e elevar seus modelos a um novo patamar de excelência.

HANDS ON

Nessa vídeo aula veremos as técnicas para obter o melhor resultado de nossos modelos, otimizando-os ao testar diversos valores nos hiperparâmetros chaves de cada um. Chamamos esta técnica de **Grid Search**, em que cada coluna é um valor diferente de cada hiperparâmetro e (normalmente) a última coluna apresenta o valor da métrica que escolhemos para mensurar o desempenho do modelo.

Outra etapa importante é a forma que os dados são divididos para garantir que nosso modelo está, de fato, aprendendo. Para saber isso, mensuramos com algumas métricas o desempenho do modelo para exemplos nunca antes vistos por ele. Só que a divisão dos dados em treino e teste, mesmo que de forma aleatória, pode não ser feita da melhor forma e, com isso, seu treinamento pode não apresentar resultados confiáveis.

Vamos ver uma técnica de divisão de dados para evitar qualquer problema quanto a esta questão chamada de Validação Cruzada, na qual fazemos a divisão dos dados em folds e alternamos entre elas para os treinos e validações.

Ambas as técnicas podem consumir um tempo considerável dependendo da quantidade de valores candidatos nos parâmetros e dependendo também da quantidade de folds. Apesar disso, veremos que são etapas importantes para garantir uma modelagem adequada.

SAIBA MAIS

Validação Cruzada

A validação cruzada se posiciona como uma técnica fundamental para avaliar a robustez e a generalização dos modelos. Dizemos que um modelo generaliza bem quando houve aprendizado e ele consegue responder bem para novos exemplos nunca vistos antes, ou seja, não decorou o padrão dos dados (overfitting) e também não conseguiu encontrar relações entre as features e a classe (underfitting).

Através da divisão dos dados em conjuntos de treino e validação, podemos estimar com maior precisão o desempenho do modelo em dados não vistos, evitando o overfitting e garantindo que ele seja capaz de generalizar bem para novas situações.

Nesta jornada aprofundada, exploraremos os meandros da validação cruzada com Python e Scikit-Learn, desde seus fundamentos teóricos até a implementação prática com exemplos concretos. Mas antes, vamos entender mais a fundo os conceitos de overfitting, underfitting e generalização.

Overfitting

O overfitting, ou sobreajuste, ocorre quando um modelo se adapta excessivamente aos dados de treino, memorizando suas características específicas e perdendo a capacidade de generalizar bem para novos dados. Dizemos neste caso que o modelo "overfitou". Essa situação pode ser causada por diversos fatores:

- **Escassez de Dados:** um conjunto de dados de treinamento muito pequeno não fornece amostras suficientes para representar com precisão todos os valores de dados de entrada possíveis. O modelo, buscando se ajustar ao máximo aos dados disponíveis, acaba aprendendo padrões específicos que não se generalizam para novos dados.
- **Ruído nos Dados:** dados de treinamento com grandes quantidades de informações irrelevantes, chamadas de ruído, podem confundir o modelo durante o aprendizado. O modelo, ao tentar se adaptar ao ruído, perde a capacidade de identificar os padrões reais presentes nos dados e, consequentemente, não se generaliza bem para novos dados.

- **Treinamento Excessivo:** treinar o modelo por muito tempo em um único conjunto de dados de amostra pode levar ao overfitting. O modelo, após aprender todas as características do conjunto de dados de treinamento, não consegue se adaptar a novos dados que podem conter padrões diferentes.
- **Complexidade Excessiva do Modelo:** um modelo com complexidade muito alta possui um grande número de parâmetros e flexibilidade para se adaptar aos dados de treinamento. Essa flexibilidade pode levar o modelo a aprender padrões específicos do conjunto de dados de treinamento, ignorando as características mais relevantes para a generalização.

Para evitar o overfitting e garantir a generalização dos modelos, diversas técnicas podem ser utilizadas:

- **Aumento do Conjunto de Dados:** ampliar o conjunto de dados de treinamento com novas amostras aumenta a diversidade dos dados e fornece ao modelo uma base mais abrangente para aprender os padrões reais. Isso reduz a chance de o modelo se adaptar excessivamente a características específicas do conjunto de dados original.
- **Limpeza de Dados:** remover dados ruidosos do conjunto de treinamento impede que o modelo seja influenciado por informações irrelevantes. Técnicas como a detecção de outliers e a imputação de dados podem ser utilizadas para limpar o conjunto de dados e melhorar a qualidade da informação.
- **Validação Cruzada:** a validação cruzada divide o conjunto de dados em subconjuntos e treina o modelo em cada subconjunto, utilizando os outros como conjuntos de validação. Essa técnica avalia o desempenho do modelo em dados não vistos durante o treinamento e permite identificar o momento ideal para interromper o treinamento e evitar o overfitting.
- **Seleção de Modelos:** a seleção de modelos envolve a escolha de um modelo com a complexidade adequada para o problema em questão. Modelos muito simples podem subestimar os padrões presentes nos dados, enquanto modelos muito complexos podem levar ao overfitting. Técnicas como a validação cruzada e a seleção de hiperparâmetros podem auxiliar na escolha do modelo ideal.

- **Técnicas de Dropout:** técnica amplamente utilizada em redes neurais. Durante o treinamento, uma proporção aleatória de neurônios é desativada (recebe valor zero) em cada camada da rede. Isso força a rede a aprender representações robustas dos dados, pois não pode confiar sempre na saída de neurônios específicos. Ao final do treinamento, todos os neurônios são utilizados novamente para fazer previsões.
- **Early Stopping:** a técnica de Early Stopping interrompe o treinamento do modelo quando o desempenho no conjunto de validação começa a piorar. Isso evita que o modelo continue treinando e aprendendo padrões específicos do conjunto de treinamento que não se generalizam para novos dados.

Underfitting

O underfitting é o oposto: ele surge quando um modelo não aprende suficientemente os padrões presentes nos dados de treinamento. Essa situação pode ser causada por diversos fatores:

- **Modelo Muito Simples:** um modelo com complexidade muito baixa possui um número limitado de parâmetros e flexibilidade para se adaptar aos dados de treinamento. O modelo, incapaz de capturar as nuances dos dados, simplifica excessivamente os padrões e não consegue fazer previsões precisas.
- **Poucos Dados de Treinamento:** um conjunto de dados de treinamento muito pequeno não fornece ao modelo amostras suficientes para aprender os padrões presentes nos dados. O modelo, com base em uma quantidade limitada de informações, não consegue generalizar para novos dados e apresenta um desempenho inferior.
- **Treinamento Insuficiente:** treinar o modelo por um tempo muito curto pode levar ao underfitting. O modelo, sem tempo suficiente para aprender os padrões dos dados, não consegue convergir para uma solução ideal e apresenta um desempenho inferior.
- **Ruído nos Dados:** dados de treinamento com grandes quantidades de informações irrelevantes, chamadas de ruído, podem confundir o modelo

durante o aprendizado. O modelo, ao tentar se adaptar ao ruído, perde a capacidade de identificar os padrões reais presentes nos dados e, consequentemente, não consegue aprender com eficiência.

Generalização

No contexto do aprendizado de máquina, a generalização se refere à capacidade de um modelo de realizar previsões precisas em dados novos e não vistos durante o treinamento. Um modelo que generaliza bem é capaz de identificar padrões nos dados de treinamento e aplicá-los de forma eficaz para fazer previsões em situações que o modelo nunca encontrou antes.

O modelo provavelmente terá um bom desempenho ao ser testado com os mesmos dados que foi treinado, pois nestes o modelo assimilou o padrão. O que queremos é que ele consiga obter a resposta correta, na média, mediante exemplos novos.

Pela figura 1 podemos ter uma melhor noção das três situações mencionadas para a tarefa de regressão. Repare que no primeiro caso, o de underfitting, não houve quase aprendizado, acertando alguns casos (a reta fica próxima de alguns pontos), mas não aprende o padrão todo.

No segundo caso, da generalização, vemos que, apesar da função não passar "em cima" de todos os pontos, é uma função que se aproxima da realidade e terá um bom resultado, na média, na métrica utilizada para avaliar.

No terceiro, o de overfitting, o modelo decorou o padrão dos dados e a função passa exatamente em cada ponto nos exemplos de treinamento, o que significa que não terá um bom desempenho para novos exemplos não vistos e sua métrica de avaliação será comprometida.

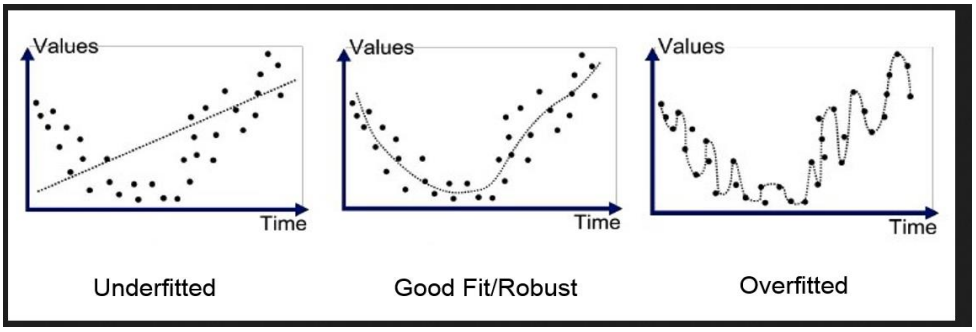

Figura 1 – Representação de Underfitting, Generalização e Overfitting para regressão
Fonte: Hu (2024)

Para a tarefa de classificação (figura 2), vemos os fenômenos de aprendizado de forma semelhante. No underfitting não houve aprendizado suficiente para todos os exemplos, no overfitting forçou-se a separação das classes e no caso que houve uma boa generalização vemos que a classificação, apesar de errar em alguns, obterá um resultado melhor para novos exemplos. É importante comentar que, com o tempo e a entrada de novos dados, pode ser que o padrão mude e que seja necessário retreinar o seu modelo.

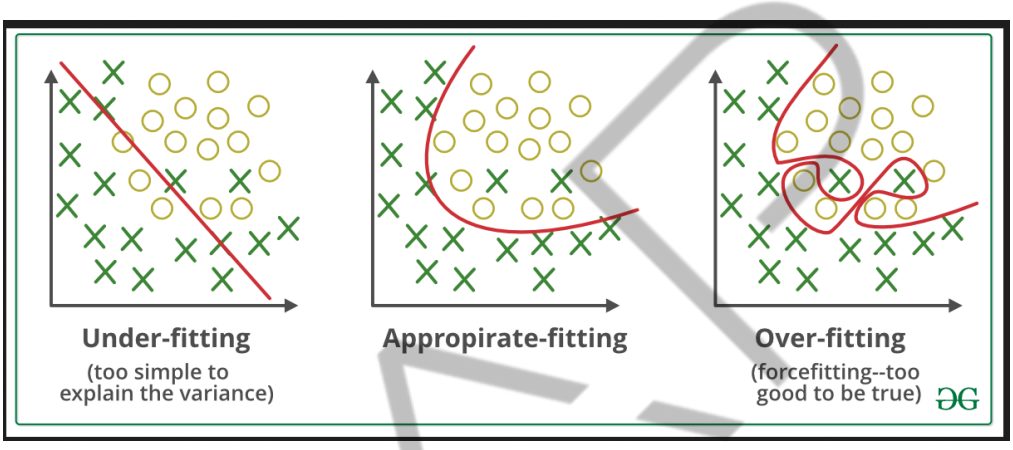

Figura 2 – Representação de Underfitting, Generalização e Overfitting para classificação
Fonte: Gour (2018)

Um conceito importante para o cientista de dados ter em mente é que há um equilíbrio, como tudo na vida, entre treinar pouco seu modelo e treinar muito (figura 3). Na aula anterior tivemos uma noção disso, de certa forma, no modelo de k-vizinhos mais próximos quando aumentamos a quantidade de vizinhos e observamos o respectivo resultado. Vimos que há um intervalo de valores ótimos de vizinhos para aquele modelo.

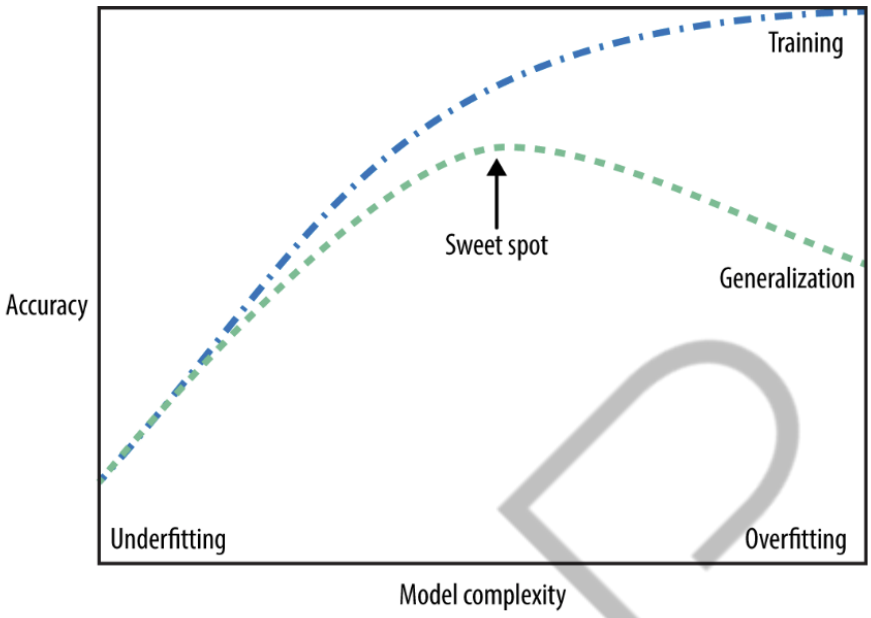

Figura 3 – Trade off entre overfitting e underfitting
Fonte: Muller, Guido (2016)

Para avaliar a generalização de um modelo, podemos utilizar técnicas como a **validação cruzada** e a **avaliação em um conjunto de teste independente**. Nestes métodos, o modelo é treinado em um conjunto de dados e avaliado em outro conjunto de dados diferente, que não foi utilizado para o treinamento. Se o modelo apresentar um bom desempenho em ambos os conjuntos de dados, podemos considerar que ele generaliza bem e é capaz de realizar previsões precisas em dados novos. Há alguns tipos de validação cruzada, vamos entender:

- **K-Fold Cross-Validation:** uma das técnicas mais comuns de validação cruzada. Nela, os dados são divididos em **k** subconjuntos iguais (folds). O modelo é treinado **k** vezes, utilizando cada fold como conjunto de validação em uma iteração. A performance média do modelo em todas as iterações é considerada como uma estimativa imparcial do seu desempenho em dados não vistos.
    - Para exemplificar, na figura 4 temos uma representação de uma validação cruzada para k sendo 5. Dividimos o conjunto de treinamento em 5 partes iguais. Na primeira iteração pegamos o fold 1 para validação e o fold 2 até o 5 para o treinamento e medimos com alguma métrica. Na segunda iteração usamos o fold 2 para validação e os demais para treino

e guardamos a métrica. Fazemos isso até a quinta iteração. A média desta métrica é uma estimativa que esperamos ser similar quando o modelo fizer as inferências no conjunto de teste.

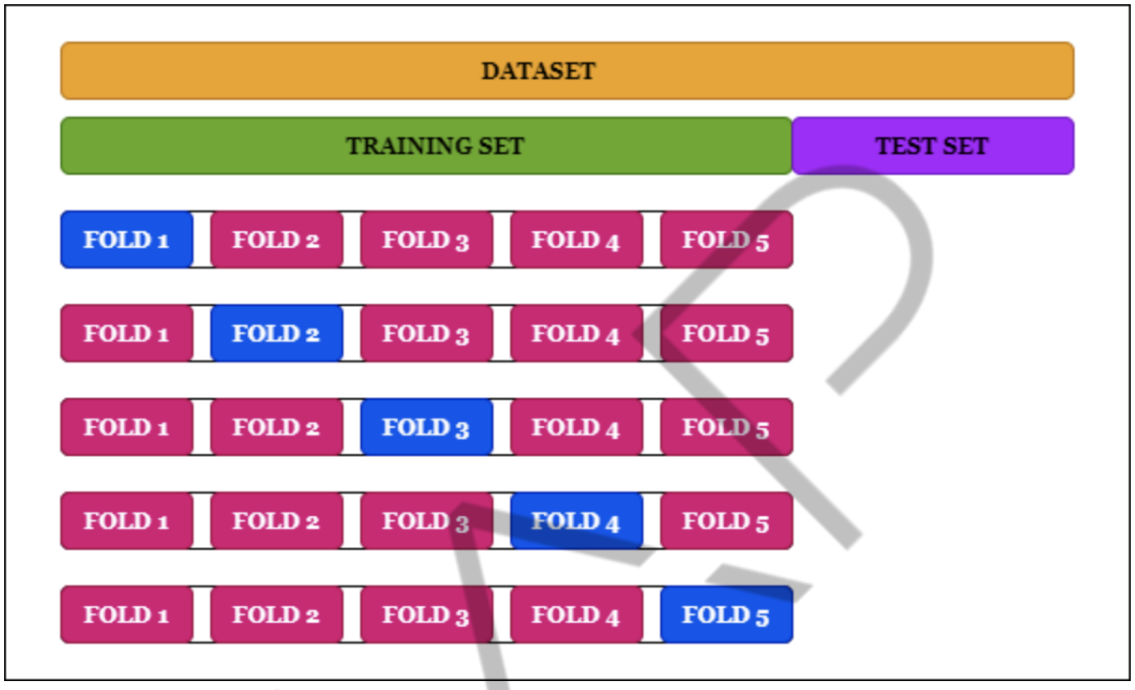

Figura 4 – Representação do K-Fold Cross Validation, com K = 5
Fonte: Manna (2020)

- **Stratified K-Fold Cross-Validation:** em conjuntos de dados com classes desbalanceadas, a Stratified K-Fold Cross-Validation garante que cada fold contenha a mesma proporção de classes que o conjunto de dados original. Isso evita que o modelo seja enviesado para a classe majoritária.
- **Leave-One-Out Cross-Validation:** nesta proposta, o modelo é treinado n vezes, sendo que n é igual ao número de amostras no conjunto de dados. Em cada iteração, uma única amostra é utilizada como conjunto de validação, enquanto as demais são utilizadas para o treinamento. Esse método fornece uma estimativa imparcial do desempenho do modelo. É a melhor forma de garantir um bom treinamento, mas é a técnica mais custosa computacionalmente, principalmente para conjuntos de dados grandes.

Vejamos agora um exemplo de código utilizando a técnica K-Fold Cross-Validation. Usaremos o mesmo dataset da aula anterior com os mesmos tratamentos, recapitulando aqui (código-fonte 1).

```
import pandas as pd
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.linear_model import LinearRegression
from sklearn.metrics import mean_absolute_percentage_error

df = pd.read_csv('kc_house_data.csv')
df  =  df[['price',  'bedrooms',  'bathrooms',  'sqft_living',
'sqft_lot', 'floors', 'waterfront']]

x = df.drop('price', axis=1)
y = df['price']

# normalização dos dados
min_max_scaler = StandardScaler()
x = min_max_scaler.fit_transform(x)
```

Código-fonte 1 – Importações, load do dataset e tratamento
Fonte: Elaborado pelo autor (2024)

```
# Criando o modelo de regressão linear
linear_regressor = LinearRegression()

# Definindo o número de folds
k = 5

# Criando o objeto KFold
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Armazenará os scores de cada fold
mape_scores = []

# Realizando o K-Fold Cross-Validation
for train_index, val_index in kf.split(x):
    x_train, x_val = x[train_index], x[val_index]
    y_train, y_val = y[train_index], y[val_index]

    # Treinando o modelo no conjunto de treino
    linear_regressor.fit(x_train, y_train)

    # Fazendo previsões no conjunto de validação
    y_pred = linear_regressor.predict(x_val)

    # Calculando o erro percentual absoluto médio (MAPE)
    mape = mean_absolute_percentage_error(y_val, y_pred)

    # Armazenando o MAPE para cada fold
    mape_scores.append(mape)
```

```
# Calculando o MAPE médio
mape_mean = np.mean(mape_scores)

print(f"MAPE médio: {mape_mean}")
MAPE médio: 0.34622668025790687
```

Código-fonte 2 – Aplicação do K-Fold Cross-Validation, com K = 5
Fonte: Elaborado pelo autor (2024)

Podemos visualizar a pontuação de cada fold na lista `mape_scores` (código-fonte 3).

```
print(mape_scores)
[0.34395818800589684,
 0.34797610372682075,
 0.34657027263314844,
 0.34294743291873186,
 0.34968140400493636]
```

Código-fonte 3 – Resultados do MAPE na lista `mape_scores`
Fonte: Elaborado pelo autor (2024)

No exemplo da validação cruzada anterior foi utilizado todo o conjunto de dados para treino e validação. Vamos agora fazer de forma similar à figura 3, separando 10% dos dados para o teste final (código-fonte 4).

```
# Separando os dados de treino (para a validação cruzada) e de
teste
x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.1, random_state=0)

# Criando o modelo de regressão linear
linear_regressor = LinearRegression()

# Definindo o número de folds
k = 5

# Criando o objeto KFold
kf = KFold(n_splits=k, shuffle=True, random_state=42)

# Armazenará os scores de cada fold
mape_scores = []

# Realizando o K-Fold Cross-Validation
for train_index, val_index in kf.split(x_train, y_train):
    x_train_cv, x_val = x[train_index], x[val_index]
    y_train_cv, y_val = y[train_index], y[val_index]

    # Treinando o modelo no conjunto de treino
    linear_regressor.fit(x_train_cv, y_train_cv)

    # Fazendo previsões no conjunto de validação
```

```
    y_pred = linear_regressor.predict(x_val)

    # Calculando o erro percentual absoluto médio (MAPE)
    mape = mean_absolute_percentage_error(y_val, y_pred)

    # Armazenando o MAPE para cada fold
    mape_scores.append(mape)

# Calculando o MAPE médio
mape_mean = np.mean(mape_scores)

print(f"MAPE médio: {mape_mean}")
MAPE médio: 0.3511047791823444
```

Código-fonte 4 – Outro exemplo de validação cruzada com o conjunto de teste separado
Fonte: Elaborado pelo autor (2024)

Agora, vamos ver o desempenho do modelo para o conjunto de testes (código-fonte 5). Repare que agora não iremos fazer o treinamento (função `.fit()`) pois é justamente este treinamento que queremos medir a performance.

```
# Vendo a performance agora do modelo para o conjunto de teste

# Fazendo previsões no conjunto de teste
y_pred_2 = linear_regressor.predict(x_test)

# Calculando o erro percentual absoluto médio (MAPE)
mape_test = mean_absolute_percentage_error(y_test, y_pred_2)

print(f"MAPE: {mape_test}")
MAPE: 0.3513107105323694
```

Código-fonte 5 – Análise do modelo com o conjunto de teste
Fonte: Elaborado pelo autor (2024)

Vemos que o valor obtido agora não é muito diferente do obtido com a validação cruzada. Isso mostra que o nosso modelo é consistente e generaliza bem para exemplos não vistos antes (`x_test`).

Este exemplo demonstra a implementação básica do K-Fold Cross-Validation em Python. O código pode ser adaptado para diferentes tipos de modelos e métricas de avaliação.

Recursos Adicionais

É altamente recomendado que você leia sempre a documentação oficial das bibliotecas, veja outros parâmetros das funções, entenda sua implementação e treine o inglês. Veja aqui a documentação do Scikit-Learn sobre K-Fold Cross-Validation.

Sugestões:

- Experimentar diferentes valores de k para observar a influência do número de folds nos resultados.
- Utilizar outras métricas de avaliação.
- Visualizar os resultados utilizando gráficos e tabelas.

A validação cruzada é uma ferramenta essencial para avaliar a generalização dos modelos e garantir sua confiabilidade em situações reais. Treine esta técnica para poder criar modelos mais robustos e eficazes!

Tuning de Hiperparâmetros

O ajuste fino dos hiperparâmetros (valores ou categorias definidas como argumentos nas funções dos modelos) é outra etapa importante para alcançar melhores desempenhos dos seus modelos. Estes hiperparâmetros definem o comportamento do algoritmo de otimização e do processo de ajuste.

Repare que na aula anterior houve poucos casos que eram definidos hiperparâmetros dos modelos, pois estes normalmente possuem valores pré-definidos. Por exemplo, no modelo Support Vector Machines Regressor da biblioteca Scikit-Learn dois hiperparâmetros que costumamos alterar são o kernel e o valor C (código-fonte 6). O kernel diz respeito ao tipo de kernel a ser usado no algoritmo e o valor de C é um parâmetro de regularização.

De acordo com a documentação, os valores pré-definidos para kernel e C são, respectivamente, 'rbf' e 1. O kernel possui outros valores pré-estabelecidos, observando sua documentação: ‘linear’, ‘poly’, ‘sigmoid’, ‘precomputed’.

```
class sklearn.svm.SVR(*, kernel='rbf', degree=3, gamma='scale', coef0=0.0, tol=0.001, C=1.0, epsilon=0.1, shrinking=True, cache_size=200, verbose=False, max_iter=-1)
```

Código-fonte 6 – Modelo Support Vector Machines do Scikit-Learn com seus hiperparâmetros
Fonte: Scikit-Learn (2024)

Então podemos escolher alguns hiperparâmetros e testar outros valores para medir o desempenho do modelo (código-fonte 7).

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.1, random_state=0)

# Definindo os parâmetros a serem ajustados
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}

# Criando o modelo
svr = SVR()

# Ajuste fino
clf = GridSearchCV(svr, parameters)

# Treinando o modelo com otimização
clf.fit(x_train, y_train)
```

Código-fonte 7 – Otimização de dois hiperparâmetros com SVR
Fonte: Elaborado pelo autor (2024)

No Grid Search será feito o treinamento com cada combinação de hiperparâmetro. Como estamos fazendo o ajuste em dois hiperparâmetros e cada um deles possui dois valores, são quatro combinações ao todo e, portanto, quatro vezes que o treinamento será feito.

Os resultados estão em um dicionário `cv_results_`. Vamos observar algumas chaves dele, como as combinações dos hiperparâmetros (código-fonte 8).

```
print(clf.cv_results_['params'])
[{'C': 1, 'kernel': 'linear'},
 {'C': 1, 'kernel': 'rbf'},
 {'C': 10, 'kernel': 'linear'},
 {'C': 10, 'kernel': 'rbf'}]
```

Código-fonte 8 – Apresentação dos resultados da validação cruzada
Fonte: Elaborado pelo autor (2024)

E podemos ver qual das combinações obteve o melhor resultado (código-fonte 9).

```
print(clf.best_params_)
{'C': 10, 'kernel': 'linear'}
```

Código-fonte 9 – Apresentação dos melhores resultados da validação cruzada
Fonte: Elaborado pelo autor (2024)

Outra forma de visualizar os melhores hiperparâmetros é colocando o dicionário em um DataFrame do Pandas, filtrando pelo melhor resultado em `rank_test_score` igual a 1 e obtendo os melhores valores dos hiperparâmetros (código-fonte 10).

```
df_results = pd.DataFrame(clf.cv_results_)
df_results.query("rank_test_score == 1")['params']

{'C': 10, 'kernel': 'linear'}
```

Código-fonte 10 – Transformação em DataFrame e filtragem do melhor resultado
Fonte: Elaborado pelo autor (2024)

Validação Cruzada com ajuste fino

Vimos cada técnica de forma detalhada. Felizmente o Scikit-Learn tem um módulo pronto para realizarmos ambas as técnicas de uma vez: é a mesma biblioteca que utilizamos anteriormente no ajuste fino, definindo explicitamente a quantidade de folds que queremos para a validação cruzada (código-fonte 11). Neste caso faremos com k = 10 folds. É importante mencionar que este processo é demorado e levou quase 9 minutos para ser processado.

```
from sklearn.svm import SVR
from sklearn.model_selection import GridSearchCV

x_train, x_test, y_train, y_test = train_test_split(x, y,
test_size=0.1, random_state=0)

# Definindo os parâmetros a serem ajustados
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}

# Criando o modelo
svr = SVR()

# Ajuste fino com validação cruzada
clf = GridSearchCV(svr, parameters, cv=10)

# Treinando o modelo com otimização
clf.fit(x_train, y_train)
```

Código-fonte 11 – Grid Search com validação cruzada, 10 folds
Fonte: Elaborado pelo autor (2024)

E os melhores resultados foram estes (código-fonte 12).

```
print(clf.best_params_)
{'C': 10, 'kernel': 'linear'}
```

Código-fonte 12 – Apresentação dos melhores resultados da validação cruzada com tuning de hiperparâmetros
Fonte: Elaborado pelo autor (2024)

Observamos que foram as mesmas combinações de valores que obtivemos para o caso anterior. Com isso, você já sabe quais parâmetros utilizar para fazer o treinamento definitivo do seu modelo para subir em produção.

Viés e Variância

O viés e a variância se posicionam como dois desafios cruciais que podem comprometer a performance dos modelos.

- **Viés**: representa o **erro** de generalização dado a suposições erradas. Por exemplo: um modelo de classificação de spam que classifica e-mails que são legítimos como spam. Esse modelo apresenta um viés alto, pois está constantemente errando em um tipo de e-mail. É provável que um modelo de alto viés não se ajuste aos dados de treinamento (underfitting).
- **Variância**: representa a **sensibilidade** do modelo a pequenas variações nos dados de treinamento. Um modelo com alta variância tende a apresentar resultados inconsistentes, variando significativamente em diferentes conjuntos de dados. Imagine um modelo de regressão linear que aprende padrões muito específicos do conjunto de dados de treinamento. Esse modelo apresenta alta variância, pois não generaliza bem para novos dados que podem conter padrões ligeiramente diferentes.

Causas do Viés:

- **Dados de treinamento não representativos**: se os dados de treinamento não representarem adequadamente a população real, o modelo pode apresentar viés, favorecendo as características presentes nos dados de treinamento e ignorando as características da população real.
- **Algoritmos tendenciosos**: alguns algoritmos podem apresentar viés inerente, especialmente se forem baseados em dados históricos ou decisões humanas que podem conter vieses.
- **Falta de diversidade nos dados**: a falta de diversidade nos dados de treinamento pode levar o modelo a apresentar viés em relação a grupos sub-representados.

Impactos do Viés:

- **Previsões incorretas**: o viés pode levar a previsões incorretas, especialmente para grupos sub-representados nos dados de treinamento.
- **Falta de generalização**: modelos com viés alto podem não generalizar bem para novos dados, pois não aprenderam os padrões reais da população.
- **Perda de confiança**: o viés pode levar à perda de confiança nos modelos, especialmente quando as previsões são inconsistentes ou injustas.

Causas da Variância:

- **Modelo muito complexo**: um modelo com muitos parâmetros e flexibilidade pode aprender padrões muito específicos dos dados de treinamento, mas não generalizar bem para novos dados.
- **Poucos dados de treinamento**: um modelo treinado em um conjunto de dados pequeno pode ser muito sensível a pequenas mudanças nos dados, apresentando alta variância e resultados inconsistentes.
- **Ruído nos dados**: a presença de ruído nos dados de treinamento pode confundir o modelo durante o aprendizado, levando a alta variância e previsões inconsistentes.

Impactos da Variância:

- **Previsões inacuradas e instáveis**: a alta variância pode levar a previsões inacuradas e instáveis que variam significativamente em diferentes conjuntos de dados.
- **Overfitting**: modelos com alta variância tendem a apresentar overfitting, se adaptando excessivamente aos dados de treinamento e não generalizando bem para novos dados.
- **Dificuldade na interpretação**: modelos com alta variância podem ser difíceis de interpretar, pois seus resultados podem variar muito em diferentes situações.

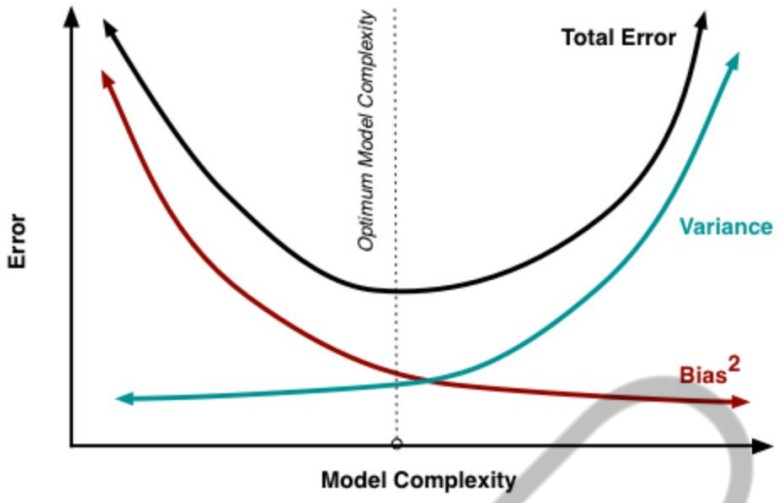

Figura 5 – Contribuição do viés e variância para o erro total
Fonte: Fortmann-Roe (2012)

Podemos reduzir o viés dos modelos com uma coleta de dados representativos, utilizando técnicas de desvio de viés e validando o modelo com diversos dados. É fundamental coletar dados de treinamento que representem adequadamente a população real, evitando vieses de amostragem e garantindo que o modelo aprenda os padrões presentes na população real.

Sobre as técnicas, há o resampling e a ponderação de classes que podem ser utilizadas para balancear conjuntos de dados desbalanceados e reduzir o viés em relação a grupos sub-representados. A validação do modelo com conjuntos de dados diversos, que representam diferentes grupos e situações, identifica e corrige possíveis vieses presentes no modelo.

E para reduzir a variância, podemos simplificar o modelo; utilizar modelos com menor complexidade (menos parâmetros) pode reduzir a capacidade do modelo de se adaptar excessivamente aos dados de treinamento e, consequentemente, diminuir a variância. Outra possibilidade é aumentar o conjunto de dados, pois treinar o modelo com um conjunto de dados maior fornece mais informações para o modelo aprender os padrões reais e reduz a sensibilidade a pequenas variações nos dados, diminuindo a variância.

Manter o equilíbrio entre viés e variância é crucial para construir modelos de aprendizado de máquina eficazes. Um viés muito alto pode levar a previsões

incorretas e falta de generalização, enquanto uma variância muito alta pode levar a previsões instáveis e overfitting.

Experimentar diferentes técnicas, analisar os dados e o problema específico que você está abordando são fundamentais para encontrar o equilíbrio ideal entre viés e variância para cada situação. Com as ferramentas apresentadas nesta aula, você dispõe de meios para construir modelos robustos, precisos e generalizáveis, preparados para enfrentar os desafios do mundo real.

MERCADO, CASES E TENDÊNCIAS

A Genie AI é uma IA da Google que promete criar jogos a partir de imagens e prompts de texto. O modelo foi treinado por mais de 200.000 horas com plataformas 2D disponíveis gratuitamente na internet. Veja aqui.

O Google estaria trabalhando em uma integração do Gemini com streamings de música, que pode sugerir combinar o uso da assistente virtual da Google com seu streaming de música favorito. Leia sobre aqui.

O QUE VOCÊ VIU NESTA AULA?

Nesta aula vimos técnicas de otimização e ajuste fino de modelos para obter o melhor resultado possível de um modelo de machine learning. Entendemos os conceitos de overfitting, underfitting e generalização dos modelos.

Analisamos técnicas como a validação cruzada para avaliar a generalização da modelagem e o grid search para fazer a combinações de valores para cada hiperparâmetro que queremos ajustar. Por fim, vimos os conceitos de viés e variância, que estão totalmente relacionados com os conceitos vistos anteriormente.

REFERÊNCIAS

FORTMANN-ROE, S. **Understanding the bias-variance tradeoff**. 2012. Disponível em: <https://scott.fortmann-roe.com/docs/BiasVariance.html>. Acesso em: 05 jul. 2024.

GÉRON, A. **Hands-on machine learning with scikit-learn, keras, and tensorflow**. 3. ed. [s.l.] O'Reilly Media, 2022.

GOUR, R. **Deep neural networks with python - rinu Gour**. 2018. Disponível em: <https://medium.com/@rinu.gour123/deep-neural-networks-with-python-6b599a5b1af9>. Acesso em: 05 jul. 2024.

HU, M. **Utilizing Machine Learning to Predict the Malignancy of a Breast Tumor**. 2024. Disponível em: <https://www.researchgate.net/publication/379846879_Utilizing_Machine_Learning_to_Predict_the_Malignancy_of_a_Breast_Tumor>. Acesso em: 05 jul. 2024.

MANNA, S. **K-Fold Cross Validation for Deep Learning Models using Keras**. 2020. Disponível em: <https://medium.com/the-owl/k-fold-cross-validation-in-keras-3ec4a3a00538>. Acesso em: 05 jul. 2024.

MULLER, A. C.; GUIDO, S. **Introduction to machine learning with python:** a guide for data scientists. [s.l.] O'Reilly Media, 2016.

SCIKIT-LEARN. **Sklearn.Model_selection.GridSearchCV**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html>. Acesso em: 05 jul. 2024.

SCIKIT-LEARN. **Sklearn.Svm.SVR**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVR.html>. Acesso em: 05 jul. 2024.

SCIKIT-LEARN. **Tuning the hyper-parameters of an estimator**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/grid_search.html>. Acesso em: 05 jul. 2024.

SCIKIT-LEARN.**Sklearn.Model_selection.KFold**. 2024. Disponível em: <https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html>. Acesso em: 05 jul. 2024.

PALAVRAS-CHAVE

Palavras-chave: Otimização. Ajuste fino. Generalização. Overfitting. Underfitting. Grid Search. Validação Cruzada. Pandas. Numpy. Scikit-Learn. Python. Viés. Variância.