Fine Tuning Série Temporal

# Aula	17
☑ Ready	☑
☑ Finished	☑
≡ Ciclos	Ciclo 02: Fundamentos

Objetivo da Aula:

- [ ] O que é Fine-Tuning?
- [ ] O método Random Search
- [ ] Implementação em Python

Conteúdo:

▼ 1. O que é Fine-Tuning?

Fine-Tuning é o processo de ajuste fino de um modelo estatístico ou de aprendizado de máquina para melhorar seu desempenho.

Isso envolve a seleção e otimização dos hiperparâmetros do modelo, garantindo que ele tenha um bom equilíbrio entre viés e variância, além de evitar problemas como overfitting ou underfitting.

▼ 1.1. Métodos de Fine-Tuning

Há várias formas de buscar os melhores valores de hiperparâmetros:

- **Grid Search:** Testa todas as combinações possíveis de um conjunto pré-definido de valores.

- **Random Search:** Escolhe aleatoriamente algumas combinações e avalia o desempenho.
- **Bayesian Optimization:** Usa inferência estatística para escolher os melhores valores com menos tentativas.
- **AutoML:** Algoritmos automatizados ajustam os hiperparâmetros com base em heurísticas.

2. O método Random Search

Random Search (Busca Aleatória) é um método de otimização de hiperparâmetros que seleciona combinações aleatórias de valores dentro de um espaço pré-definido. Diferente do Grid Search, que testa todas as combinações possíveis, o Random Search amostra aleatoriamente um subconjunto dessas combinações, tornando o processo mais eficiente em termos de tempo computacional.

2.1. Passo a passo do Random Search

O processo do Random Search segue algumas etapas principais:

Passo 1: Definição do Espaço de Hiperparâmetros

Antes de iniciar a busca, é necessário definir quais hiperparâmetros serão ajustados e seus respectivos intervalos de valores. Esses valores podem ser contínuos (ex: taxa de aprendizado) ou discretos (ex: número de neurônios em uma camada).

Exemplo para um modelo ARIMA:

- p (ordem da parte autoregressiva): [1,2,3,...,10]
- d (número de diferenciações): [0,1,2]
- q (ordem da média móvel): [1,2,3,...,10]

Passo 2: Seleção Aleatória de Combinações

Ao invés de testar todas as combinações possíveis, o Random Search seleciona aleatoriamente um número limitado de combinações dentro do espaço de hiperparâmetros.

Exemplo:

- Iteração 1: (p=2,d=1,q=5)
- Iteração 2: (p=7,d=1,q=3)
- Iteração 3: (p=4,d=0,q=8)
- ...
- Última Iteração: (p=9,d=2,q=1)

  (p=9,d=2,q=1)(p=9, d=2, q=1)

Passo 3: Treinamento e Avaliação do Modelo

Para cada conjunto de hiperparâmetros amostrado, um modelo é treinado e avaliado utilizando uma métrica de desempenho, como:

- **RMSE (Root Mean Squared Error)**
- **MAPE (Mean Absolute Percentage Error)**
- **AIC e BIC** (para modelos estatísticos como ARIMA)

O modelo com o melhor desempenho na métrica escolhida é selecionado.

Passo 4: Seleção do Melhor Modelo

Após testar um número predefinido de combinações, o conjunto de hiperparâmetros que apresentou a melhor métrica de desempenho é escolhido para treinar o modelo final.

3. Implementação em Python

3.1. Código Python para 1 previsão

```
# Fine-tuning do ARIMA
lags_values = [1, 2, 3, 4 ]

# escolha aleatoria dos parametros ( Random Search )
lag = np.random.choice( lags_values)

# treinamento do modelo
model = AR( train_series, lags=lag )
model_fit = model.fit()
```

```
# previsao
forecast = model_fit.forecast( steps=1 ).iloc[0]

# guardando previsoes
print( f"Previsao: {forecast}" )
print( f"Actual: {validation.iloc[0]}" )
```

3.2. Código Python para múltiplas previsões

```
# Fine-tuning do ARIMA
lags_values = [1, 2, 3, 4 ]

# escolha aleatoria dos parametros ( Random Search )
lag = np.random.choice( lags_values)

predictions = []
actuals = []
train_series = train.copy()
for t in range( len( validation ) ):
    # treinamento do modelo
    model = AR( train_series, lags=lag )
    model_fit = model.fit()

    # previsao
    forecast = model_fit.forecast( steps=1 ).iloc[0]

    # guardando previsoes
    predictions.append( forecast )
    actuals.append( validation.iloc[t] )

    # update training dataset
    train_series = pd.concat( [train_series, pd.Series( validation.iloc[t], inde

df = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, index=
```

```
print( f"Previsao: {forecast}" )
print( f"Actual: {validation.iloc[0]}" )

# compute metrics
errors = df["Predictions"] - df["Actuals"]
rmse = np.sqrt( np.mean( errors ** 2 ) )
mae = np.mean( np.abs( errors ) )
mape = np.mean( np.abs( errors / df["Actuals"] ) ) * 100

# calculo das métricas AIC e BIC
log_likelihood = model_fit.llf
num_params = model_fit.params.shape[0]
num_obs = len( train_series )

aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log( num_obs )

print( f"RMSE: {rmse}" )
print( f"MAE: {mae}" )
print( f"MAPE: {mape}" )
print( f"AIC: {aic}" )
print( f"BIC: {bic}" )
```

▼ 3.3. Código Python pra múltiplas previsões fine-tuning

```
# Fine-tuning do ARIMA
lags_values = [1, 2, 3, 4 ]
num_iteration = 2
results = pd.DataFrame()

for _ in range( num_iteration ):
    # escolha aleatoria dos parametros ( Random Search )
    lag = np.random.choice( lags_values)
    print( f"Testando parâmetro: lag={lag}" )
```

```
# Start Rolling Forecast Origin
predictions = []
actuals = []
train_series = train.copy()
for t in range( len( validation ) ):
    # treinamento do modelo
    model = AR( train_series, lags=lag )
    model_fit = model.fit()

    # previsao
    forecast = model_fit.forecast( steps=1 ).iloc[0]

    # guardando previsoes
    predictions.append( forecast )
    actuals.append( validation.iloc[t] )

    # update training dataset
    train_series = pd.concat( [train_series, pd.Series( validation.iloc[t], inc

df = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, inde

#print( f"Previsao: {forecast}" )
#print( f"Actual: {validation.iloc[0]}" )

# compute metrics
errors = df["Predictions"] - df["Actuals"]
rmse = np.sqrt( np.mean( errors ** 2 ) )
mae = np.mean( np.abs( errors ) )
mape = np.mean( np.abs( errors / df["Actuals"] ) ) * 100

# calculo das métricas AIC e BIC
log_likelihood = model_fit.llf
num_params = model_fit.params.shape[0]
num_obs = len( train_series )
```

```
    aic = -2 * log_likelihood + 2 * num_params
    bic = -2 * log_likelihood + num_params * np.log( num_obs )

    performance = pd.DataFrame({'Lag': lag,
                                "RMSE": rmse,
                                "MAE": mae,
                                "MAPE": mape,
                                "AIC": aic,
                                "BIC": bic}, index=[0]
                               )

    results = pd.concat( [results, performance] )

results
```

▼ 3.4. Código Completo até o momento

```
import numpy as np
import seaborn as sns
import pandas as pd
import warnings

from matplotlib import pyplot as plt
from statsmodels.tsa.ar_model import AutoReg as AR

# configuracao da series
np.random.seed(42)
n = 500

# criacao da series perfeita
trend = np.linspace( 0, 0, n)
noise = np.random.normal( 0, 1, n)
serie_perfeita = trend + noise

dates = pd.date_range( start='2023-01-01', periods=n, freq='D' )
serie_perfeita = pd.Series( serie_perfeita, index=dates, name='serie_perfe
```

```
# Quebra Premissa 1: Linearidade
trend_break = np.linspace( 0, 10, n )
serie_nao_linear = serie_perfeita + trend_break

# Quebra Premissa 2: Estacionariedade
seasonality = 3 * np.sin( 2 * np.pi * np.arange( n ) / 50  )
serie_nao_estacionaria = serie_perfeita + seasonality

# Quebra Premissa 3: Autocorrelacao dos residucos
autoregressive = np.zeros( n)
autoregressive[0] = noise[0]
for t in range( 1, n ):
    autoregressive[t] = 0.8 * autoregressive[t-1] + np.random.normal( 0, 0.5

serie_nao_autocorrelacao = serie_perfeita + autoregressive

# Quebra Premissa 4: Homoscedasticidade
non_normal = noise * np.linspace( 1, 3, n )
serie_nao_homoscedastica = serie_perfeita + non_normal

# Quebra Premissa 5: Não nomralidade dos residuos
non_normal_noise = np.random.exponential( scale=1, size= n )
serie_nao_normal = trend + non_normal_noise

# Combinar as series
serie_final = (
    serie_perfeita
    + trend_break
    + seasonality
    + autoregressive
    + non_normal
    + non_normal_noise
)

serie_final = pd.Series( serie_final, index=dates, name='serie_final' )
```

```
serie_final = serie_final - serie_final.min() + 1

# Visualizacao das series
plt.figure( figsize=( 16, 8 ) )
sns.lineplot( serie_final )

# Divisão da Série em Conjuntos de Treinamento, Validação e Teste
train_size = int( 0.8 * len( serie_final ) )
validation_size = int( 0.1 * len( serie_final ) )

train = serie_final[:train_size]
validation = serie_final[train_size:train_size + validation_size]
test = serie_final[train_size + validation_size:]

warnings.filterwarnings( "ignore" )

# Fine-tuning do ARIMA
lags_values = [1, 2, 3, 4 ]
num_iteration = 2
results = pd.DataFrame()

for _ in range( num_iteration ):
    # escolha aleatoria dos parametros ( Random Search )
    lag = np.random.choice( lags_values)
    print( f"Testando parâmetro: lag={lag}" )

    # Start Rolling Forecast Origin
    predictions = []
    actuals = []
    train_series = train.copy()
    for t in range( len( validation ) ):
        # treinamento do modelo
        model = AR( train_series, lags=lag )
        model_fit = model.fit()

        # previsao
```

```
    forecast = model_fit.forecast( steps=1 ).iloc[0]

    # guardando previsoes
    predictions.append( forecast )
    actuals.append( validation.iloc[t] )

    # update training dataset
    train_series = pd.concat( [train_series, pd.Series( validation.iloc[t], inc

df = pd.DataFrame( {"Predictions": predictions, "Actuals": actuals}, inde

#print( f"Previsao: {forecast}" )
#print( f"Actual: {validation.iloc[0]}" )

# compute metrics
errors = df["Predictions"] - df["Actuals"]
rmse = np.sqrt( np.mean( errors ** 2 ) )
mae = np.mean( np.abs( errors ) )
mape = np.mean( np.abs( errors / df["Actuals"] ) ) * 100

# calculo das métricas AIC e BIC
log_likelihood = model_fit.llf
num_params = model_fit.params.shape[0]
num_obs = len( train_series )

aic = -2 * log_likelihood + 2 * num_params
bic = -2 * log_likelihood + num_params * np.log( num_obs )

performance = pd.DataFrame({'Lag': lag,
                        "RMSE": rmse,
                        "MAE": mae,
                        "MAPE": mape,
                        "AIC": aic,
                        "BIC": bic}, index=[0]
                    )
```

```
    results = pd.concat( [results, performance] )

results
```