Faculdade XPe

Capítulo 03

TREINAMENTO E VALIDAÇÃO DE MODELOS

Luciano Tadeu P.A.Pereira

2024

No ciclo de vida de modelos de Machine Learning, a etapa de treinamento e a validação são fundamentais para garantir a eficácia e a precisão do modelo. Durante o treinamento, o modelo aprende a identificar padrões a partir dos dados históricos, ajustando seus parâmetros para representar da melhor forma possível o comportamento dos dados de entrada, enquanto a etapa de validação é onde se avalia o desempenho do modelo em dados de teste, evitando problemas como o overfitting e underfitting. Juntos, esses processos asseguram que o modelo seja robusto, e é sobre esses temas que discutiremos neste capítulo.

Treinamento de Modelos

Para treinar um modelo de Machine Learning com eficácia, é importante escolher e aplicar técnicas adequadas ao tipo de dado e ao problema em questão. Entre as técnicas mais comuns, destacam:

- Separar conjunto de dados em treino e teste, segue exemplo em Python:

```
from sklearn.model_selection import train_test_split
# Suponha que X sejam as features e y o target
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

- O cross-validation avalia o modelo em diferentes divisões dos dados:

```
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier

# Modelo de exemplo
model = RandomForestClassifier()

# Cross-validation com 5 folds
scores = cross_val_score(model, X, y, cv=5)
print("Scores de cada fold:", scores)
print("Média dos scores:", scores.mean())
```

Outra abordagem importante nessa etapa é o uso de hyperparameter tuning. Entre os diversos frameworks que ajudam nessa etapa, os mais conhecidos são:

- **Gridsearch:** é uma técnica de otimização de hiperparâmetros em machine learning, usada para encontrar a melhor combinação de parâmetros para um modelo, ele testa todas as combinações possíveis até encontrar o ponto ótimo, o que resulta em grande processamento de informação, causando demora no treinamento. Segue exemplo em Python:

```
from sklearn.model_selection import GridSearchCV

# Parâmetros a serem testados
param_grid = {
    'n_estimators' : [50, 100, 200],
    'max_depth' : [None, 10, 20, 30]
}

# GridSearch para otimizar o RandomForest
grid_search = GridSearchCV(RandomForestClassifier(), param_grid, cv=3)
grid_search.fit(X_train, y_train)

print("Melhores parâmetros:", grid_search.best_params_)
print("Melhor score:", grid_search.best_score_)
```

- **RandomizedSearchCV:** é uma técnica alternativa ao GridSearchCV para otimização de hiperparâmetros. Em vez de realizar uma busca exaustiva em todas as combinações possíveis de valores dos hiperparâmetros, o RandomizedSearchCV seleciona aleatoriamente algumas combinações a partir de um espaço de busca especificado. Isso torna o processo de busca mais rápido e eficiente em situações em que o GridSearchCV seria muito demorado devido à alta quantidade de combinações possíveis. Segue exemplo em Python:

```
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import RandomizedSearchCV
from scipy.stats import randint

# Define o modelo
model = RandomForestClassifier()

# Define o espaço de busca com distribuições para busca aleatória
param_dist = {
    'n_estimators': randint(50, 200),  # Gera números aleatórios entre 50 e 200
    'max_depth': randint(10, 30)  # Gera números aleatórios entre 10 e 30
}

# Configura o RandomizedSearchCV com 10 combinações aleatórias e 3 folds de cross-validation
random_search=RandomizedSearchCV(estimator=model, param_distributions=param_dist, n_iter=10, cv=3, scoring='accuracy')

# Treina o modelo usando RandomizedSearch
random_search.fit(X_train, y_train)

# Exibe a melhor combinação de hiperparâmetros e o melhor score
print("Melhores parâmetros:", random_search.best_params_)
print("Melhor score:", random_search.best_score_)
```

- **Hyperopt:** é uma biblioteca de otimização de hiperparâmetros que utiliza Otimização Bayesiana para realizar uma busca mais inteligente. Em vez de escolher combinações aleatórias, ele constrói um modelo probabilístico do espaço de busca, utilizando o histórico de avaliações para decidir qual combinação de hiperparâmetros testar em seguida. Segue exemplo em Python:

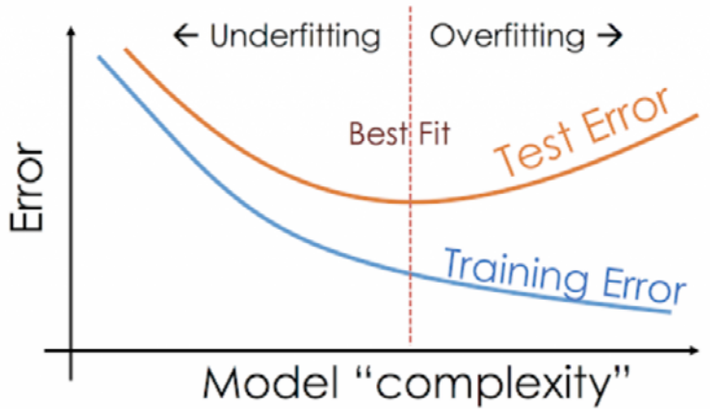
```
from hyperopt import fmin, tpe, hp, Trials, STATUS_OK
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score
from sklearn.datasets import load_iris
from sklearn.model_selection import train_test_split

# Carregando um conjunto de dados de exemplo
data = load_iris()
X = data.data
y = data.target

# Dividindo em treino e teste
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Definindo o espaço de busca para os hiperparâmetros
space = {
    'n_estimators': hp.quniform('n_estimators', 10, 200, 10),  # valores entre 10 e 200, passo de 10
    'max_depth': hp.quniform('max_depth', 3, 15, 1)  # valores entre 3 e 15, passo de 1
}

# Função de otimização
def objective(params):
    # Convertendo os valores dos hiperparâmetros para inteiros (necessário para RandomForest)
    n_estimators = int(params['n_estimators'])
    max_depth = int(params['max_depth'])

    # Criando e treinando o modelo com os hiperparâmetros atuais
    model = RandomForestClassifier(n_estimators=n_estimators, max_depth=max_depth, random_state=42)

    # Avaliando o modelo usando cross-validation
    score = cross_val_score(model, X_train, y_train, cv=3, scoring='accuracy').mean()

    # A função de objetivo minimiza, então retornamos o negativo da acurácia para maximizar o score
    return {'loss': -score, 'status': STATUS_OK}
```

```
# Inicializando Trials para armazenar os resultados das iterações
trials = Trials()

# Executando o Hyperopt com o algoritmo TPE (Tree-structured Parzen Estimator)
best = fmin(
    fn=objective,              # Função de objetivo
    space=space,               # Espaço de busca
    algo=tpe.suggest,          # Algoritmo de busca
    max_evals=50,              # Número máximo de avaliações
    trials=trials              # Armazenamento de resultados das iterações
)

# Exibindo os melhores hiperparâmetros encontrados
print("Melhores hiperparâmetros:", best)
```

Validações de modelos

A validação é uma etapa essencial no ciclo de vida de um projeto de Machine Learning, pois através dela é possível avaliar o desempenho do modelo em dados que ele não viu durante o treinamento, garantindo que ele seja capaz de generalizar para novos dados no ambiente de produção. Um conceito fundamental, além da definição de métricas adequadas, é o de overfitting e underfitting. O overfitting ocorre quando o modelo se ajusta tão bem aos dados de treinamento que perde a capacidade de generalização, capturando ruído e padrões específicos do conjunto de treino, o que leva a um desempenho ruim em dados novos, por outro lado, o underfitting acontece quando o modelo é incapaz de capturar padrões suficientes nos dados de treinamento, apresentando baixa precisão tanto nos dados de treino quanto nos dados de teste. A imagem a seguir representa graficamente o que é overfitting e o que é underfitting.

Figura 1 - Representação de Underfitting e Overfitting.



Fonte: Overfitting and Underfitting in Machine Learning

Exemplo Prático

Nesta etapa do capítulo vamos mostrar o exemplo de uma construção de pipeline de treinamento e validação de modelos, utilizando Scikit-learn e GridSearch, porém o mesmo racional pode ser replicado com outras bibliotecas de automl. Este exemplo consiste em criar um pipeline para treinar e validar 3 algoritmos que são (RandomForest, Regressão Logística e KNeigbors), utilizando técnicas de transformação de dados como: Standard Scaler, Min Maz Scaler e PCA, além de utilizar a biblioteca GridSearch para testar combinações de hiperparâmetros.

Pipeline de Treinamento de Modelos

Import Bibliotecas:

```
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.datasets import load_iris
```

Carregamento Conjunto de dados:

```
data = load_iris()X, y = data.data, data.targetX_train, X_test, y_train, y_test = train_test_split(X, y,     test_size=0.2, random_state=42)
```

Definir o Pipeline de Treinamento com várias opções:

- Usamos Pipeline e GridSearchCV para testar diferentes combinações de pré-processamento e modelos.

```
pipeline_treinamento = Pipeline([
    ( 'scaler' , StandardScaler()),
       ( 'pca' , PCA()),
    ( 'clf' , RandomForestClassifier())
])
```

Configurar a grade de hiperparâmetros para o GridSearchCV:

- Definimos as opções de transformações e modelos com seus parâmetros. Aqui estão algumas configurações:

```
param_grid = {
    'scaler' : [StandardScaler(), MinMaxScaler()],
  'pca' :       [PCA(n_components=2),       PCA(n_components=3), None],
  'clf' : [
          RandomForestClassifier(random_state=42, n_estimators=100),
        LogisticRegression(max_iter=1000),
        KNeighborsClassifier(n_neighbors=5)
    ]}
```

Rodar o GridSearchCV:

- Agora, vamos configurar o GridSearchCV para procurar a melhor combinação de transformações e modelos.

```
grid_search = GridSearchCV(pipeline_treinamento, param_grid, cv=5, scoring=' accuracy' )
grid_search.fit(X_train, y_train)
```

Exibir os resultados:

- Após rodar o GridSearchCV, podemos verificar os melhores parâmetros e a acurácia do modelo.

```
print( "Melhores Parâmetros Encontrados:", grid_search.best_params_)
print( "Melhor Acurácia no Treinamento:", grid_search.best_score_)
```

Pipeline de Validação

Agora que temos um pipeline de treinamento, vamos criar um pipeline de validação. Neste exemplo, utilizaremos validação cruzada para avaliar o desempenho do modelo.

Importação das bibliotecas de validação:

```
from sklearn.model_selection import cross_val_score
```

Executar Validação Cruzada:

- Usaremos a função cross_val_score para avaliar o modelo com validação cruzada de 5-folds.

```
scores = cross_val_score(pipeline_treinamento, X, y, cv=5)
```

Exibir resultados:

```
print( "Scores da Validação Cruzada:", scores)
print( "Acurácia Média:", scores.mean())
print( "Desvio Padrão:", scores.std())
```


Faculdade
XPe
xpeducacao.com.br