DS Lv.1-8

データサイエンス講座

Level 01-8

第８章：モデル評価・チューニング

mjin.space

第８章：モデル評価・チューニング

モデル評価基礎

モデル評価指標とビジネスKPIの関連づけの**重要性**

機械学習モデルの評価指標をビジネスKPI（重要業績評価指標）と関連づけることは、モデルの成果がビジネス目標にどの程度貢献しているかを理解する上で非常に重要です。評価指標単独では技術的な性能を示すに過ぎませんが、ビジネスKPIと結びつけることで、次のような価値を引き出すことができます。

モデル評価指標

指標	概要	事例
Accuracy (正解率)	正しく予測されたインスタンスの割合。バランスの取れたデータセットで有用。	クラスバランスが取れた分類タスクで使用。
Precision (適合率)	正と予測されたうち、実際に正である割合。予測の正確さを測る。	偽陽性のコストが高いシナリオ（例: スパムフィルタリング）。
Recall (再現率)	実際に正のうち、正しく予測された割合。予測の網羅性を測る。	偽陰性のコストが高いシナリオ（例: 疾病検出）。
F1スコア	PrecisionとRecallの調和平均。正確性と網羅性のバランスを取る。	偽陽性と偽陰性のバランスが重要な状況。
ROC/AUC	ROC曲線下面積。クラス間を区別する能力を測る。	二値分類モデルの性能評価。
RMSE (平方平均二乗誤差)	予測と実際の値の誤差の平均の平方根。連続値予測の誤差を測定。	予測誤差の大きさを把握する必要がある回帰問題。
MAE (平均絶対誤差)	予測と実際の値の絶対誤差の平均。予測の平均誤差に焦点を当てる。	平均的な予測誤差を最小化することが重要な回帰問題。

ハイパーパラメーターチューニング

ハイパーパラメーターチューニングとは

ハイパーパラメータチューニングは、機械学習モデルの性能を最大化するために、学習率や木の深さ、クラスタ数などの設定値を最適化するプロセス。適切なハイパーパラメータを選ぶことで、モデルの過学習や未学習を防ぎ、予測精度や汎化性能を向上させることができます。

指標	概要	特徴	事例
グリッドサーチ (Grid Search)	全てのパラメータ組み合わせを試行する網羅的な方法。	小規模なパラメータ空間に適し、正確だが計算コストが高い。	学習率や正則化パラメータの最適化。
ランダムサーチ (Random Search)	指定範囲からランダムにサンプリングして探索する方法。	広範囲を探索可能で、大規模なパラメータ空間に適している。	ニューラルネットの隠れ層数やユニット数の最適化。
ベイズ最適化 (Bayesian Optimization)	過去の試行結果に基づき、次に試す値を効率的に推定する方法。	試行回数を抑えつつ、高精度な結果が得られる。	強化学習のハイパーパラメータ調整。
ハイパーバンド (Hyperband)	リソースを考慮し、性能が低い設定を早期に除外する方法。	リソース効率が良く、パラメータ空間が広い場合に有効。	ディープラーニングのドロップアウト率やバッチサイズの調整。

Cross-validationのメリット

DS Lv.1-8

データを複数の分割に分けてモデルを評価する手法で、限られたデータから汎化性能を正確に測定できる点が最大のメリットです。これにより、過学習のリスクを低減し、モデルの安定性や信頼性を向上させることができます。

汎化性能の向上

限られたデータを効率的に活用し、モデルのテストデータでの性能をより正確に評価可能

過学習の検出

モデルが学習データに過剰適合していないか確認できる

安定した評価

データ分割の偶然性に依存せず、複数のスコアの平均を取ることで信頼性の高い評価が可能

適用可能な幅広さ

回帰や分類など、ほぼすべての機械学習アルゴリズムに適用可能

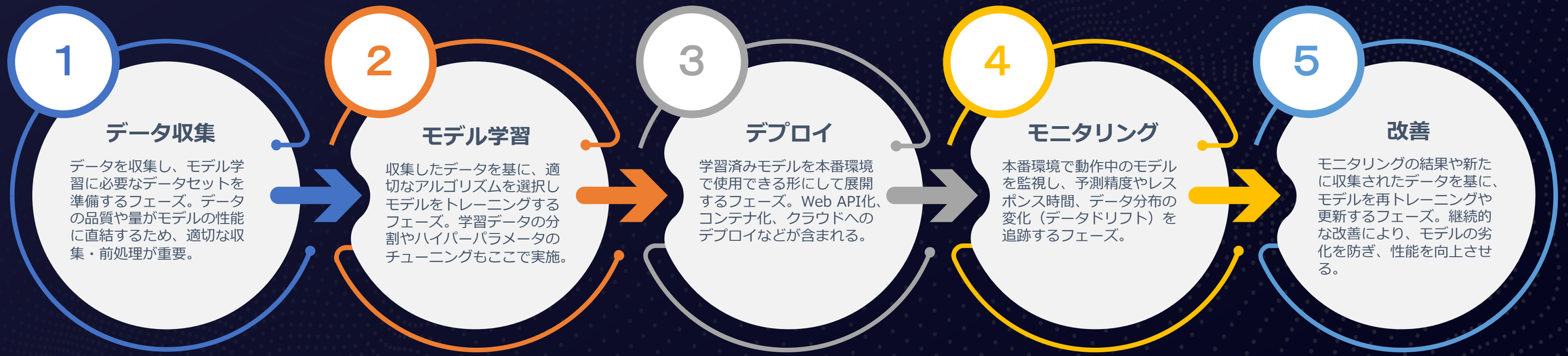
Cross-validation

手法	概要	メリット	デメリット	使用例
k-fold	データをk個に分割し、1つをテスト用、残りを学習用にしてk回繰り返す。	計算コストが低く、バランスの取れた評価が可能。	データの分布に偏りがある場合、不正確な評価となる可能性。	sklearn.model_selection.KFold
Leave-One-Out	データの1つをテスト用、残りを学習用として、データ全体で繰り返す。	データセットが小さい場合に有効で、評価が高精度。	計算コストが非常に高い。	sklearn.model_selection.LeaveOneOut
Stratified k-fold	クラス分布を保ちながらk個に分割し、k-foldを実施。	クラス不均衡なデータでのモデル評価に有効。	分類問題に限定される。	sklearn.model_selection.StratifiedKFold
Shuffle Split	データをランダムに分割して学習用とテスト用を設定し、複数回繰り返す。	ランダム性が高く、データの分割数や比率を自由に設定可能。	分割がランダムなので、再現性が低い可能性。	sklearn.model_selection.ShuffleSplit

MLOpsの概要

DS Lv.1-8

機械学習モデルの開発から運用までのライフサイクルを効率化し、継続的に価値を提供する



ログデータやセンサーデータを収集し、欠損値や異常値を処理。

データをトレーニングセットとテストセットに分割し、分類モデルを構築。

FlaskやFastAPIでREST APIを構築し、Webアプリに統合。

PrometheusやGrafanaでリアルタイムモニタリングを実施。

新データでモデルを再学習し、CI/CDパイプラインでデプロイ。

モデルのデプロイ方法

DS Lv.1-8

データを複数の分割に分けてモデルを評価する手法で、限られたデータから汎化性能を正確に測定できる点が最大のメリットです。これにより、過学習のリスクを低減し、モデルの安定性や信頼性を向上させることができます。

Web API化

- FlaskやFastAPIを用いてREST APIを作成。
- サービスの他の部分と連携可能。

クラウドベースのデプロイ

- AWS SageMaker、Google AI Platform、Azure MLを利用。
- スケーラビリティと管理のしやすさが特徴。

コンテナ化

- Dockerを使ってモデルをパッケージ化し、どこでも動作可能。
- Kubernetesでスケーリング対応。

エッジデプロイ

- モデルを軽量化してモバイル端末やIoTデバイスに搭載。

CI/CD概要

DS Lv.1-8

CI/CDの概念

- **CI (Continuous Integration):**
  開発中に変更されたコードやモデルを頻繁に統合・テスト。
- **CD (Continuous Delivery/Deployment):**
  テスト済みモデルを本番環境に自動的にデプロイ。

機械学習用パイプライン構築の要素

1. データ前処理とフィーチャエンジニアリング。
2. モデルのトレーニングと評価。
3. モデルのデプロイ。
4. モニタリングと再トレーニング。

ツール例

GitHub Actions、Jenkins、Kubeflow

モデル運用フェーズの課題と解決方法

課題

1. モデルの劣化（データドリフト）。
2. モニタリングとパフォーマンス管理。
3. スケーリングとリソース管理。

解決法

1. モデルモニタリングツールの導入（Prometheus、Grafana）。
2. リアルタイム推論パフォーマンスの監視。
3. スケーリングにクラウドリソースを活用。

演習：モデル評価とハイパーパラメータチューニング

学習のポイント

1. **Cross-validationの重要性:**モデルの汎化性能を正確に評価するために、Cross-validationを活用するメリットを理解。
2. **グリッドサーチの実践:** ハイパーパラメータの全ての組み合わせを試行して最適なモデルを見つける方法を体験。
3. **モデル評価指標の理解:** AccuracyやClassification Reportを通じてモデル性能を多角的に評価する。
4. **汎化性能向上のテクニック:** 適切なハイパーパラメータの選択が、過学習や未学習の防止にどう役立つかを学ぶ。

8-01.py

```
import numpy as np
from sklearn.datasets import load_iris
from sklearn.model_selection import cross_val_score, GridSearchCV
from sklearn.ensemble import RandomForestClassifier
from sklearn.metrics import classification_report

# Load Iris dataset
data = load_iris()
X, y = data.data, data.target

# Define model and parameter grid
model = RandomForestClassifier(random_state=42)
param_grid = {
    "n_estimators": [10, 50, 100],
    "max_depth": [3, 5, 10]
}

# Perform Grid Search with Cross-Validation
grid_search = GridSearchCV(model, param_grid, cv=5, scoring='accuracy')
grid_search.fit(X, y)
 . . .
```

演習：モデルデプロイ（Flask API）

学習のポイント

1. **APIを使ったモデル運用の基本:** 学習済みモデルをWeb APIとして展開し、外部システムとの連携方法を学ぶ。
2. **Flaskの基礎操作:** Flaskを使ってRESTfulなAPIを構築する方法を理解。
3. **JSON形式のデータ受け渡し:**モデルに入力データを渡し、予測結果を返すAPI設計を体験。
4. **デプロイの実務的な応用:**サービスとして動作する機械学習モデルの基本的な構築方法を習得。

8-02.py

```
from flask import Flask, request, jsonify
import pickle
import numpy as np

# Load pre-trained model (assuming it is saved as 'model.pkl')
with open("model.pkl", "rb") as file:
    model = pickle.load(file)

# Initialize Flask app
app = Flask(__name__)

@app.route('/predict', methods=['POST'])
def predict():
    data = request.get_json()  # Input should be JSON
    features = np.array(data['features']).reshape(1, -1)
    prediction = model.predict(features)
    return jsonify({"prediction": int(prediction[0])})

if __name__ == '__main__':
    app.run(debug=True)
```

演習：CI/CDパイプラインシミュレーション

学習のポイント

1. **CI/CDの基礎概念:** 継続的インテグレーションと継続的デリバリーが、モデルの効率的な開発と運用にどう貢献するかを学ぶ。
2. **GitHub Actionsの活用:** シンプルなワークフローでモデルの訓練とデプロイを自動化する手順を理解。
3. **パイプライン構築の基本ステップ:** コードのチェックアウト、依存関係のインストール、モデル訓練、デプロイまでの一連の流れを体験。
4. **再現性と効率性の向上:** パイプラインを使うことで、モデルの更新やデプロイがどれだけ効率化されるかを実感。

8-03.yml

```
# Simulated YAML for GitHub Actions CI/CD
# File: .github/workflows/ml_pipeline.yml

name: ML Pipeline

on:
  push:
    branches:
      - main

jobs:
  train-deploy:
    runs-on: ubuntu-latest

    steps:
    - name: Checkout Code
      uses: actions/checkout@v3

    - name: Set up Python
      uses: actions/setup-python@v3
. . .
```

DS Lv.1-8

お疲れ様でした

info@mjin.space