**머신러닝 모델 적중률 향상을 위한 전체 절차**

**1. 문제 정의 및 목표 설정 (Problem Definition & Goal Setting)**

* **명확한 목표**: 무엇을 예측하거나 분류하고 싶은가? 어떤 종류의 문제를 해결하려는가? (예: 고객 이탈 예측, 이미지 내 객체 분류, 주택 가격 예측 등)
* **성능 지표 선택**: 적중률(Accuracy)만으로는 충분하지 않을 수 있습니다. 문제의 특성에 따라 Precision, Recall, F1-Score, ROC AUC (분류), RMSE, MAE (회귀) 등 적절한 평가 지표를 선택해야 합니다. 불균형 데이터셋에서는 Accuracy가 오해를 줄 수 있습니다.
* **비즈니스 가치**: 모델의 성능 향상이 실제 비즈니스에 어떤 가치를 가져올지 이해합니다.

**2. 데이터 수집 및 이해 (Data Collection & Understanding)**

* **관련성 높은 데이터 수집**: 문제를 해결하는 데 필요한 데이터를 최대한 많이, 그리고 다양하게 수집합니다. 내부 데이터뿐만 아니라 외부 데이터(경제 지표, 날씨, 소셜 미디어 트렌드 등)도 고려합니다.
* **탐색적 데이터 분석 (EDA, Exploratory Data Analysis)**:
  + 데이터의 분포, 통계량, 고유값 등을 파악합니다.
  + 변수 간의 상관관계, 이상치, 결측치를 시각적으로 확인합니다.
  + 도메인 지식을 활용하여 데이터의 특징을 이해합니다.
  + 데이터 불균형 여부를 확인합니다.

**3. 데이터 전처리 (Data Preprocessing)**

모델이 학습하기 적합한 형태로 데이터를 가공하는 과정입니다. 적중률에 가장 큰 영향을 미치는 단계 중 하나입니다.

* **결측치 처리 (Missing Value Imputation)**:
  + 결측치가 있는 행/열을 삭제하거나, 평균, 중앙값, 최빈값, 예측 모델 등으로 채웁니다.
  + 결측치 발생 원인을 분석하여 적절한 방법을 선택합니다.
* **이상치 처리 (Outlier Handling)**:
  + 이상치를 식별하고(박스 플롯, Z-score 등), 제거하거나, 다른 값으로 대체(Capping), 또는 변환(로그 변환)합니다.
  + 이상치가 실제 중요한 정보를 담고 있을 수 있으므로 신중하게 접근합니다.
* **범주형 변수 인코딩 (Categorical Encoding)**:
  + 순서가 없는 범주형 변수: 원-핫 인코딩(One-Hot Encoding)
  + 순서가 있는 범주형 변수: 라벨 인코딩(Label Encoding) 또는 순서 매핑
* **스케일링 (Scaling)**:
  + 수치형 변수들의 스케일(값의 범위)을 통일합니다.
  + **정규화(Normalization)**: 0과 1 사이로 변환 (Min-Max Scaler)
  + **표준화(Standardization)**: 평균 0, 분산 1로 변환 (StandardScaler)
  + 이는 Gradient Descent 기반 모델(선형 회귀, 로지스틱 회귀, 신경망)의 성능 향상에 필수적입니다.
* **데이터 노이즈 제거**: 오타, 중복 데이터 등을 정리합니다.

**4. 특징 공학 (Feature Engineering)**

모델의 예측 성능에 가장 큰 영향을 미칠 수 있는 창의적인 단계입니다. 기존 변수를 조합하거나 변환하여 새로운 의미 있는 변수(특징)를 생성합니다.

* **새로운 특징 생성**: 날짜/시간에서 요일, 월, 계절, 근무 시간 등을 추출하거나, 여러 컬럼을 조합하여 비율, 상호작용 항 등을 만듭니다.
* **차원 축소 (Dimensionality Reduction)**:
  + 너무 많은 특징은 과적합(Overfitting)을 유발하거나 모델 학습 시간을 늘릴 수 있습니다.
  + 주성분 분석(PCA), 독립성분 분석(ICA), LSA 등 기법을 사용하여 차원을 줄입니다.
* **특징 선택 (Feature Selection)**:
  + 모델 예측에 가장 중요한 특징들만 선택하여 불필요한 특징을 제거합니다.
  + 필터(Filter) 방법, 래퍼(Wrapper) 방법, 임베디드(Embedded) 방법 등이 있습니다.

**5. 모델 선택 및 학습 (Model Selection & Training)**

* **다양한 알고리즘 시도**: 문제의 유형(분류, 회귀 등)과 데이터의 특성(선형, 비선형 등)에 따라 다양한 머신러닝 알고리즘(선형 모델, 트리 기반 모델, SVM, 신경망 등)을 시도하고 비교합니다. 특정 알고리즘이 모든 데이터셋에 최적은 아닙니다.
* **데이터 분할**: 학습 데이터(Train Set), 검증 데이터(Validation Set), 테스트 데이터(Test Set)로 분할합니다.
  + **학습 데이터**: 모델을 훈련하는 데 사용됩니다.
  + **검증 데이터**: 모델 훈련 중 하이퍼파라미터 튜닝 및 모델 선택에 사용됩니다. 과적합 방지에 중요합니다.
  + **테스트 데이터**: 모델의 최종 성능을 평가하는 데 한 번만 사용됩니다.
* **교차 검증 (Cross-Validation)**:
  + 데이터 분할에 따른 모델 성능 편향을 줄이고, 모델의 일반화 성능을 더 신뢰성 있게 평가하기 위해 사용합니다. K-Fold Cross-Validation이 대표적입니다.

**6. 하이퍼파라미터 튜닝 (Hyperparameter Tuning)**

선택된 모델의 성능을 최적화하기 위해 모델 학습 전에 설정하는 값(하이퍼파라미터)들을 조정하는 과정입니다.

* **그리드 서치 (Grid Search)**: 모든 하이퍼파라미터 조합을 시도합니다.
* **랜덤 서치 (Random Search)**: 하이퍼파라미터 공간에서 무작위로 샘플링하여 시도합니다. 그리드 서치보다 효율적일 수 있습니다.
* **베이지안 최적화 (Bayesian Optimization)**: 이전 시도 결과를 바탕으로 다음 최적의 하이퍼파라미터 조합을 탐색합니다.
* **자동화된 ML (AutoML)**: 하이퍼파라미터 튜닝, 특징 엔지니어링, 모델 선택 등을 자동화하는 기술입니다.

**7. 모델 평가 (Model Evaluation)**

* **선택된 지표 사용**: 1단계에서 설정한 평가 지표(Accuracy, Precision, Recall, F1-Score, RMSE 등)를 사용하여 모델의 성능을 객관적으로 평가합니다.
* **과적합/과소적합 진단**:
  + **과적합(Overfitting)**: 학습 데이터에서는 성능이 좋지만 테스트 데이터에서는 성능이 떨어지는 경우 (모델이 너무 복잡할 때 발생).
  + **과소적합(Underfitting)**: 학습 데이터와 테스트 데이터 모두에서 성능이 좋지 않은 경우 (모델이 너무 단순할 때 발생).
  + 이러한 문제를 진단하고 3~6단계를 다시 반복하여 해결책을 찾습니다.
* **학습 곡선 (Learning Curve)**: 학습 데이터 크기에 따른 모델 성능 변화를 시각화하여 과적합/과소적합을 진단하는 데 도움을 줍니다.

**8. 앙상블 (Ensemble Learning) (선택 사항)**

단일 모델로 성능 한계에 도달했을 때, 여러 모델의 예측을 결합하여 더 나은 성능을 얻는 기법입니다.

* **배깅 (Bagging)**: 여러 모델을 병렬로 학습시키고 결과를 평균/투표하여 결합 (예: 랜덤 포레스트)
* **부스팅 (Boosting)**: 여러 모델을 순차적으로 학습시키며 이전 모델의 오류를 보완 (예: Gradient Boosting, XGBoost, LightGBM, CatBoost)
* **스태킹 (Stacking)**: 여러 기본 모델의 예측을 입력으로 사용하여 메타 모델을 학습시킵니다.

**9. 모델 배포 및 모니터링 (Model Deployment & Monitoring)**

* **실제 환경 적용**: 학습 및 검증이 완료된 모델을 실제 서비스 환경에 배포합니다.
* **지속적인 모니터링**: 배포된 모델의 성능을 지속적으로 모니터링합니다. 실제 데이터가 시간이 지남에 따라 변할 수 있기 때문입니다 (데이터 드리프트, 개념 드리프트). 성능 저하가 감지되면 모델을 재학습하거나 업데이트해야 합니다.
* **피드백 루프**: 실제 환경에서 발생하는 새로운 데이터와 피드백을 수집하여 다음 모델 개선 사이클에 활용합니다.

차원 축소(Dimensionality Reduction)는 데이터셋의 특징(feature) 수를 줄이는 과정입니다. 특징의 수가 너무 많으면 모델 학습 시간이 길어지고, 과적합(overfitting)의 위험이 증가하며, 데이터를 시각화하기 어려워집니다. 차원 축소는 이러한 문제를 해결하고 데이터의 핵심 정보를 보존하면서 더 효율적인 모델 학습을 돕습니다.

주요 차원 축소 기법에는 \*\*주성분 분석(PCA)\*\*과 **t-SNE**가 있습니다. 여기서는 이 두 가지 기법에 대한 예제를 보여드리겠습니다.

**예제 1: 주성분 분석 (PCA - Principal Component Analysis)**

**PCA**는 데이터의 분산을 가장 잘 설명하는 새로운 직교(orthogonal) 축(주성분)을 찾아 데이터를 이 축에 투영하여 차원을 축소하는 선형 차원 축소 기법입니다. 데이터의 특징 간 상관관계가 높을 때 특히 효과적입니다.

**시나리오:** 붓꽃(Iris) 데이터셋은 4개의 특징(꽃받침 길이/너비, 꽃잎 길이/너비)을 가지고 있습니다. 이 4차원 데이터를 2차원으로 축소하여 시각화하고 싶습니다.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_iris

from sklearn.preprocessing import StandardScaler

from sklearn.decomposition import PCA

# 1. 데이터 로드 및 준비

iris = load\_iris()

X = iris.data # 특징 데이터

y = iris.target # 타겟 (붓꽃 품종)

feature\_names = iris.feature\_names

# 2. 데이터 스케일링 (PCA는 스케일에 민감하므로 필수)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

# 3. PCA 모델 생성 및 학습 (4차원 -> 2차원)

pca = PCA(n\_components=2) # 2개의 주성분으로 축소

X\_pca = pca.fit\_transform(X\_scaled) # PCA 적용

# 4. 축소된 데이터프레임 생성 (시각화를 위해)

df\_pca = pd.DataFrame(data=X\_pca, columns=['principal component 1', 'principal component 2'])

df\_pca['target'] = y # 품종 정보 추가

print("원본 데이터 X.shape:", X.shape)

print("PCA 적용 후 X\_pca.shape:", X\_pca.shape)

print("\nPCA로 축소된 데이터 (상위 5개 행):")

print(df\_pca.head())

# 5. 시각화

plt.figure(figsize=(8, 6))

targets = [0, 1, 2]

colors = ['r', 'g', 'b']

for target, color in zip(targets, colors):

indicesToKeep = df\_pca['target'] == target

plt.scatter(df\_pca.loc[indicesToKeep, 'principal component 1'],

df\_pca.loc[indicesToKeep, 'principal component 2'],

c=color,

s=50)

plt.xlabel('Principal Component 1')

plt.ylabel('Principal Component 2')

plt.title('2 Component PCA of Iris Dataset')

plt.legend(iris.target\_names)

plt.grid()

plt.show()

# 각 주성분이 설명하는 분산 비율 확인

print("\n각 주성분이 설명하는 분산 비율:", pca.explained\_variance\_ratio\_)

print("설명된 총 분산:", sum(pca.explained\_variance\_ratio\_))

 **스케일링의 중요성**: PCA는 분산을 기반으로 하기 때문에 각 특징의 스케일이 다르면 스케일이 큰 특징의 영향이 과대평가될 수 있습니다. StandardScaler를 사용하여 데이터를 표준화합니다.

 **n\_components**: PCA로 축소할 최종 차원 수를 지정합니다 (여기서는 2).

 **fit\_transform**: PCA 모델을 데이터에 학습시키고, 동시에 데이터를 변환합니다.

 **시각화**: 2차원으로 축소된 데이터를 산점도(scatter plot)로 그려 품종별 분포를 확인합니다. PCA를 통해 품종들이 비교적 잘 분리됨을 볼 수 있습니다.

 **explained\_variance\_ratio\_**: 각 주성분이 원본 데이터의 총 분산 중 얼마를 설명하는지 보여줍니다. 이 값이 높을수록 해당 주성분이 데이터의 중요한 정보를 많이 담고 있다는 의미입니다.

**예제 2: t-SNE (t-Distributed Stochastic Neighbor Embedding)**

**t-SNE**는 비선형 차원 축소 기법으로, 특히 고차원 데이터의 복잡한 비선형 구조를 저차원(주로 2D 또는 3D)에서 시각화하는 데 매우 강력합니다. t-SNE는 데이터 포인트 간의 고차원 공간에서의 유사도를 저차원 공간에서도 유사하게 유지하려고 노력하여 군집 구조를 잘 보여줍니다.

**시나리오:** MNIST 필기체 숫자 데이터셋(64개의 픽셀 특징)을 2차원으로 축소하여 숫자들이 어떻게 군집을 이루는지 시각화하고 싶습니다.

import pandas as pd

import matplotlib.pyplot as plt

from sklearn.datasets import load\_digits

from sklearn.preprocessing import StandardScaler

from sklearn.manifold import TSNE

import numpy as np

import time # t-SNE 계산 시간 측정용

# 1. 데이터 로드 및 준비

digits = load\_digits()

X = digits.data # 특징 데이터 (64 픽셀)

y = digits.target # 타겟 (숫자 0-9)

# 2. 데이터 스케일링 (t-SNE도 스케일링 권장)

scaler = StandardScaler()

X\_scaled = scaler.fit\_transform(X)

print("원본 데이터 X.shape:", X.shape)

# 3. t-SNE 모델 생성 및 학습 (64차원 -> 2차원)

# t-SNE는 계산 비용이 높으므로, 작은 데이터셋이나 샘플링하여 사용

# perplexity: 이웃의 수를 조절하는 파라미터 (일반적으로 5~50 사이 값 사용)

# learning\_rate: 학습 속도

# n\_iter: 최적화 반복 횟수

tsne = TSNE(n\_components=2, perplexity=30, learning\_rate=200, n\_iter=1000, random\_state=42)

print("t-SNE 계산 시작...")

start\_time = time.time()

X\_tsne = tsne.fit\_transform(X\_scaled) # t-SNE 적용

end\_time = time.time()

print(f"t-SNE 계산 완료! 소요 시간: {end\_time - start\_time:.2f} 초")

# 4. 축소된 데이터프레임 생성 (시각화를 위해)

df\_tsne = pd.DataFrame(data=X\_tsne, columns=['tsne component 1', 'tsne component 2'])

df\_tsne['target'] = y # 숫자 정보 추가

print("\nt-SNE로 축소된 데이터 (상위 5개 행):")

print(df\_tsne.head())

# 5. 시각화

plt.figure(figsize=(10, 8))

colors = plt.cm.get\_cmap('jet', 10) # 0-9까지 10가지 색상

for i in range(10): # 0부터 9까지의 숫자

indicesToKeep = df\_tsne['target'] == i

plt.scatter(df\_tsne.loc[indicesToKeep, 'tsne component 1'],

df\_tsne.loc[indicesToKeep, 'tsne component 2'],

color=colors(i),

label=str(i),

s=10, alpha=0.7)

plt.xlabel('t-SNE Component 1')

plt.ylabel('t-SNE Component 2')

plt.title('2 Component t-SNE of MNIST Digits Dataset')

plt.legend(title='Digit')

plt.grid(True)

plt.show()