AI 모델 개발 Part 2 (심화 강의)

목차

AI 모델 개발 Part 2

체계적 평가와 효율적 튜닝: "측정할 수 없으면 개선할 수 없다"

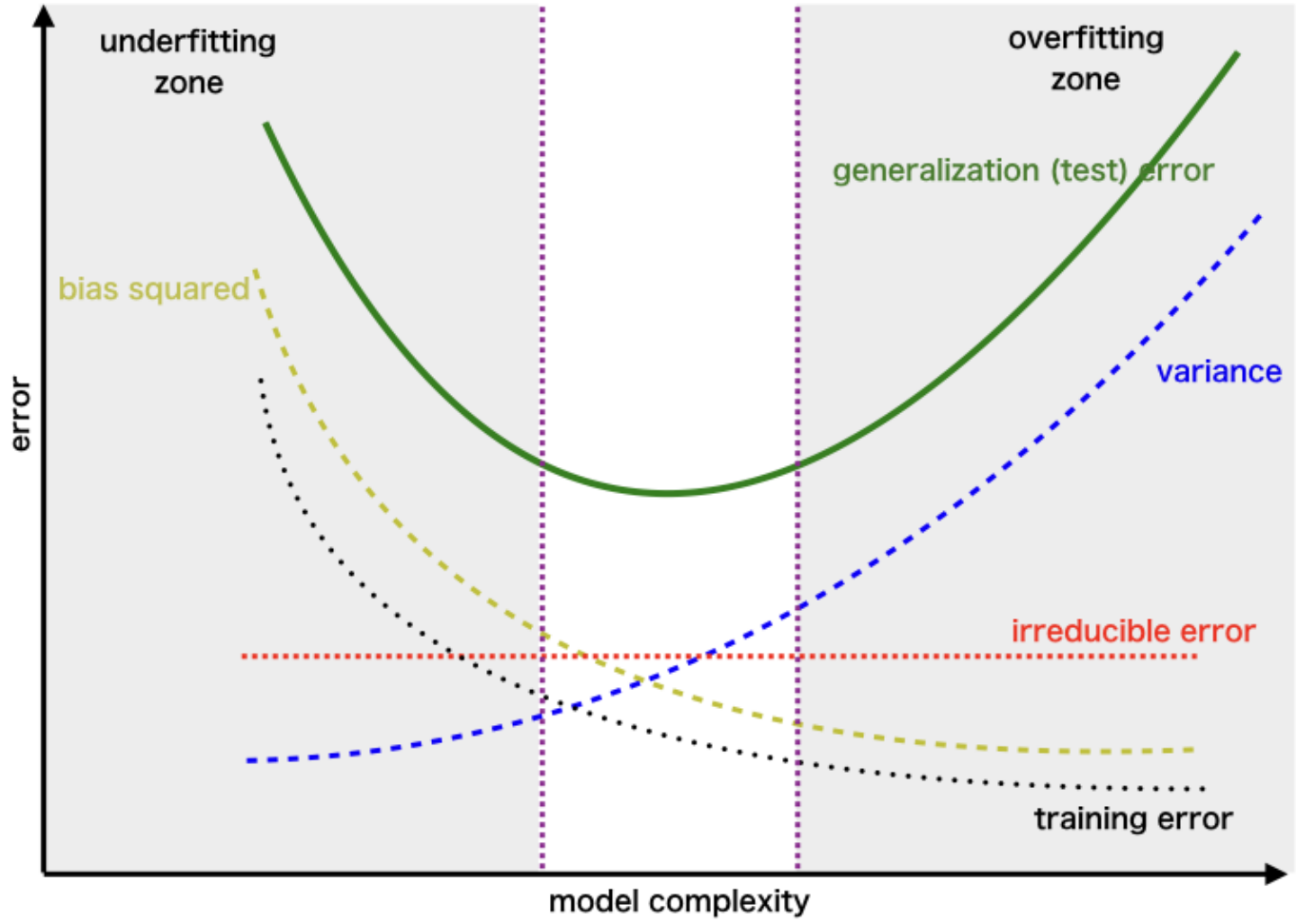
Part 1: AI 모델 학습 및 평가

AI 모델 평가의 근본적 목적과 철학

AI 모델 평가의 궁극적 목표는 모델이 훈련 데이터뿐만 아니라, 실제 환경에서 마주할 새로운 데이터에 대해 얼마나 잘 동작할지 예측하는 것입니다. 이는 통계학의 추론 이론과 깊은 관련이 있습니다.

1.1.1 Bias-Variance Trade-off

모델의 총 오차는 편향, 분산, 그리고 데이터 자체의 노이즈로 분해될 수 있습니다.




```
총 오차 = 편향² + 분산 + 노이즈
```

- 편향 (Bias): 모델의 체계적 오류. 모델이 너무 단순하여 데이터의 복잡한 패턴을 학습하지 못할 때 발생하며, 과소적합(Underfitting)의 지표입니다.
- 분산 (Variance): 모델의 불안정성. 훈련 데이터의 작은 변화에도 모델의 예측이 크게 달라질 때 발생하며, 과적합(Overfitting)의 지표입니다.
- 노이즈 (Noise): 데이터 자체에 내재된 불확실성. 이는 모델로 줄일 수 없는 최소 오차입니다.

수학적 분석

```
# 편향-분산 분해의 수학적 표현
E[(y - f̂(x))²] = (E[f̂(x)] - f(x))² + E[(f̂(x) - E[f̂(x)])²] + σ²

# 편향² = (E[f̂(x)] - f(x))²
# 분산 = E[(f̂(x) - E[f̂(x)])²]
# 노이즈 = σ²
```

실무 구현 예시

```
# 편향-분산 분석 구현
import numpy as np

def bias_variance_analysis(model, X_train, y_train, X_test, y_test,
n_iterations=100):
    predictions = []

    for _ in range(n_iterations):
        # 부트스트랩 샘플링
        indices = np.random.choice(len(X_train), len(X_train), replace=True)
        X_boot, y_boot = X_train[indices], y_train[indices]

        # 모델 학습 및 예측
        model.fit(X_boot, y_boot)
        pred = model.predict(X_test)
        predictions.append(pred)

    predictions = np.array(predictions)

    # 편향 계산
    mean_pred = np.mean(predictions, axis=0)
    bias = np.mean((mean_pred - y_test) ** 2)

    # 분산 계산
    variance = np.mean(np.var(predictions, axis=0))

    return bias, variance
```

평가는 이 트레이드오프를 이해하고, 비즈니스 목표에 맞는 최적의 균형점을 찾는 과정입니다.

1.1.2 교차 검증과 모델 선택

교차 검증의 중요성

교차 검증은 모델의 일반화 성능을 정확히 평가하고 과적합을 방지하는 핵심 기법입니다. 단순한 홀드아웃 검증보다 더 신뢰할 수 있는 성능 추정을 제공합니다.

교차 검증 기법들

- K-Fold Cross Validation: 데이터를 K개 폴드로 나누어 K번 검증. 가장 일반적이고 안정적인 방법.
- Stratified K-Fold: 클래스 비율을 유지하면서 K개 폴드로 분할. 불균형 데이터에 효과적.
- Leave-One-Out (LOO): 각 샘플을 하나씩 검증 세트로 사용. 작은 데이터셋에 적합하지만 계산 비용 높음.
- Time Series CV: 시간 순서를 고려한 교차 검증. 시계열 데이터에 필수.

실무 구현 예시

```
# 교차 검증 구현
from sklearn.model_selection import cross_val_score, StratifiedKFold
from sklearn.metrics import make_scorer, f1_score
```

```
# Stratified K-Fold 교차 검증
cv = StratifiedKFold(n_splits=5, shuffle=True, random_state=42)
scores = cross_val_score(model, X, y, cv=cv, scoring='f1')

print(f"평균 F1 점수: {scores.mean():.3f} (+/- {scores.std() * 2:.3f})")

# 시간 순서 고려 교차 검증
def time_series_cv(X, y, n_splits=5):
    fold_size = len(X) // n_splits
    for i in range(n_splits):
        test_start = i * fold_size
        test_end = (i + 1) * fold_size
        train_indices = list(range(0, test_start)) + list(range(test_end, len(X)))
        test_indices = list(range(test_start, test_end))
        yield train_indices, test_indices
```

1.1.3 평가 지표의 설계 원칙

SMART 원칙의 AI 적용

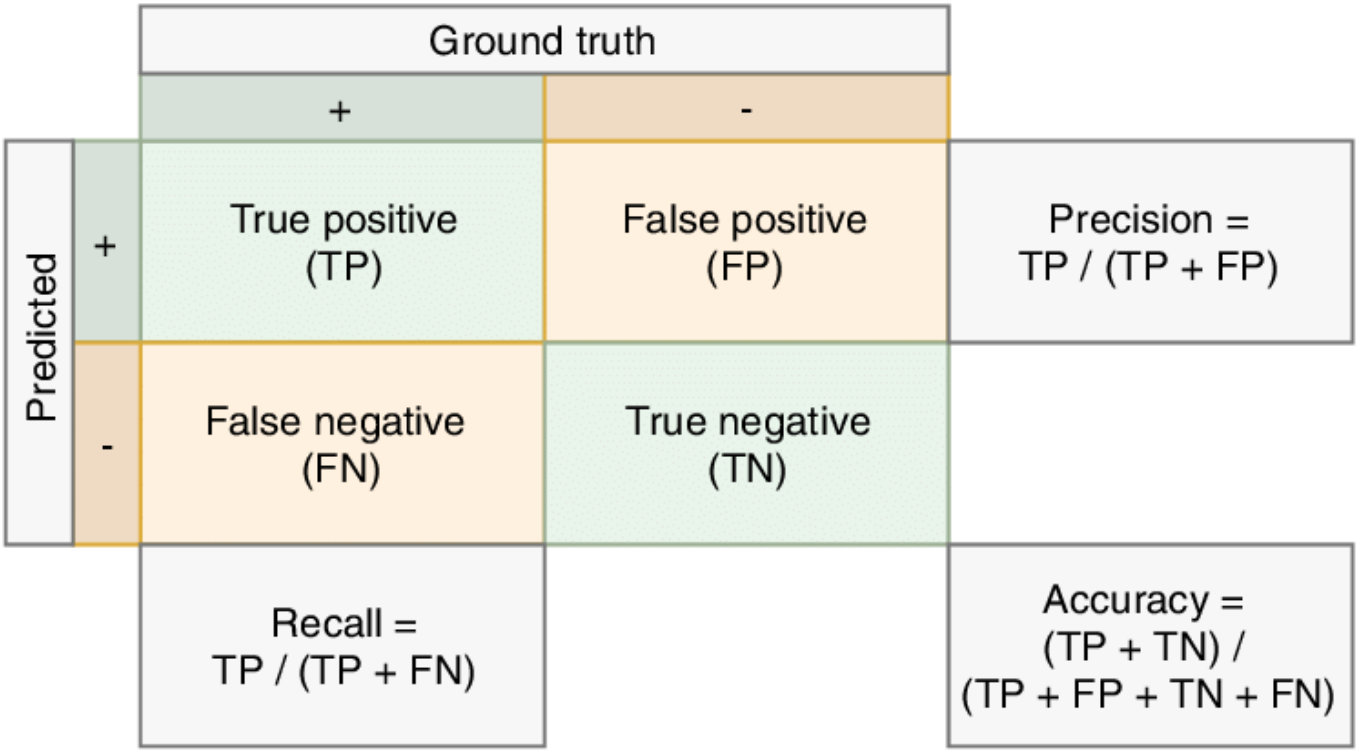
- Specific(구체적): 측정하고자 하는 능력이 명확해야 합니다.
- Measurable(측정가능): 정량적 수치로 표현 가능해야 합니다.
- Achievable(달성가능): 현실적으로 달성 가능한 목표여야 합니다.
- Relevant(관련성): 비즈니스 목표와 직결되어야 합니다.
- Time-bound(시한성): 평가 시점과 주기가 명확해야 합니다.

도메인별 평가 철학

- 분류: 결정 경계의 품질.
- 회귀: 연속값 예측의 정확성.
- 생성: 창의성과 현실성의 균형.
- 강화학습: 장기적 보상 최적화.

1.2.1 분류 모델 평가: Confusion Matrix

혼동 행렬(Confusion Matrix)은 분류 모델의 예측 결과를 실제 클래스와 비교하여 시각화하는 가장 기본적인 도구입니다. 이진 분류에서 2x2 행렬은 네 가지 경우를 담습니다.



- True Positive (TP): 실제 Positive를 Positive로 올바르게 예측 (정탐)
- False Negative (FN): 실제 Positive를 Negative로 잘못 예측 (미탐, Type II 오류, 놓침)
- False Positive (FP): 실제 Negative를 Positive로 잘못 예측 (오탐, Type I 오류, 거짓 경보)
- True Negative (TN): 실제 Negative를 Negative로 올바르게 예측 (정탐)

이 네 가지 값으로부터 다양한 평가 지표가 파생됩니다.

1.2.2 정밀도(Precision)와 재현율(Recall)

정밀도와 재현율은 혼동 행렬에서 파생되는 가장 중요한 두 지표이며, 비즈니스 목표에 따라 중요도가 달라지는 트레이드오프 관계입니다.

- 정밀도 (Precision = TP / (TP+FP)): "모델이 Positive라고 예측한 것 중 실제로 Positive인 비율". 거짓 경보(False Positive)를 최소화하는 것이 중요한 상황(e.g., 스팸 메일 필터, 금융 사기 탐지)에서 핵심 지표입니다. 이는 보수성(Conservatism)의 지표입니다.
- 재현율 (Recall = TP / (TP+FN)): "실제 Positive 중에서 모델이 찾아낸 비율". 양성 사례를 놓치는 것(False Negative)이 치명적인 상황(e.g., 암 진단, 보안 침입 탐지)에서 핵심 지표입니다. 이는 민감성(Sensitivity)의 지표입니다.
- F1-Score: 정밀도와 재현율의 조화평균. 두 지표가 모두 중요할 때 사용하며, 산술평균과 달리 어느 한쪽이 극단적으로 낮으면 점수가 크게 하락하는 특징이 있어, 두 지표를 모두 고려하게 만듭니다.

추가 평가 지표들

- Specificity (특이도): TN / (TN + FP). 실제 Negative를 올바르게 식별하는 능력.
- Balanced Accuracy: (Sensitivity + Specificity) / 2. 불균형 데이터에서 유용.
- Cohen's Kappa: 우연에 의한 일치를 제거한 일치도 측정.
- Matthews Correlation Coefficient (MCC): 모든 클래스에 균등한 가중치를 주는 지표.

실무 구현 예시

```
# 평가 지표 계산
from sklearn.metrics import precision_score, recall_score, f1_score,
balanced_accuracy_score

def calculate_metrics(y_true, y_pred):
    precision = precision_score(y_true, y_pred)
    recall = recall_score(y_true, y_pred)
    f1 = f1_score(y_true, y_pred)
    balanced_acc = balanced_accuracy_score(y_true, y_pred)

    return {
        'precision': precision,
        'recall': recall,
        'f1': f1,
        'balanced_accuracy': balanced_acc
    }

# 비즈니스 목표에 따른 임계값 최적화
import numpy as np

def optimize_threshold(y_true, y_scores, cost_fp=1, cost_fn=10):
    thresholds = np.arange(0.1, 0.9, 0.01)
    best_cost = float('inf')
    best_threshold = 0.5

    for threshold in thresholds:
        y_pred = (y_scores >= threshold).astype(int)
        fp = np.sum((y_pred == 1) & (y_true == 0))
        fn = np.sum((y_pred == 0) & (y_true == 1))
        total_cost = fp * cost_fp + fn * cost_fn

        if total_cost < best_cost:
            best_cost = total_cost
            best_threshold = threshold

    return best_threshold
```

1.2.3 회귀 모델 평가 지표

회귀 모델의 핵심 평가 지표

- Mean Squared Error (MSE): 예측 오차의 제곱 평균. 큰 오차에 더 큰 페널티를 부여.
- Root Mean Squared Error (RMSE): MSE의 제곱근. 원본 스케일과 동일한 단위로 해석 가능.
- Mean Absolute Error (MAE): 예측 오차의 절댓값 평균. 이상치에 덜 민감.
- R^2 (Coefficient of Determination): 모델이 설명하는 분산의 비율. 0~1 범위, 높을수록 좋음.

고급 회귀 평가 지표

- Mean Absolute Percentage Error (MAPE): 상대적 오차의 절댓값 평균. 스케일에 무관한 비교 가능.
- Symmetric MAPE (sMAPE): MAPE의 대칭 버전. 0에 가까운 값에서 더 안정적.

- Huber Loss: MSE와 MAE의 조합. 이상치에 강건하면서도 미분 가능.

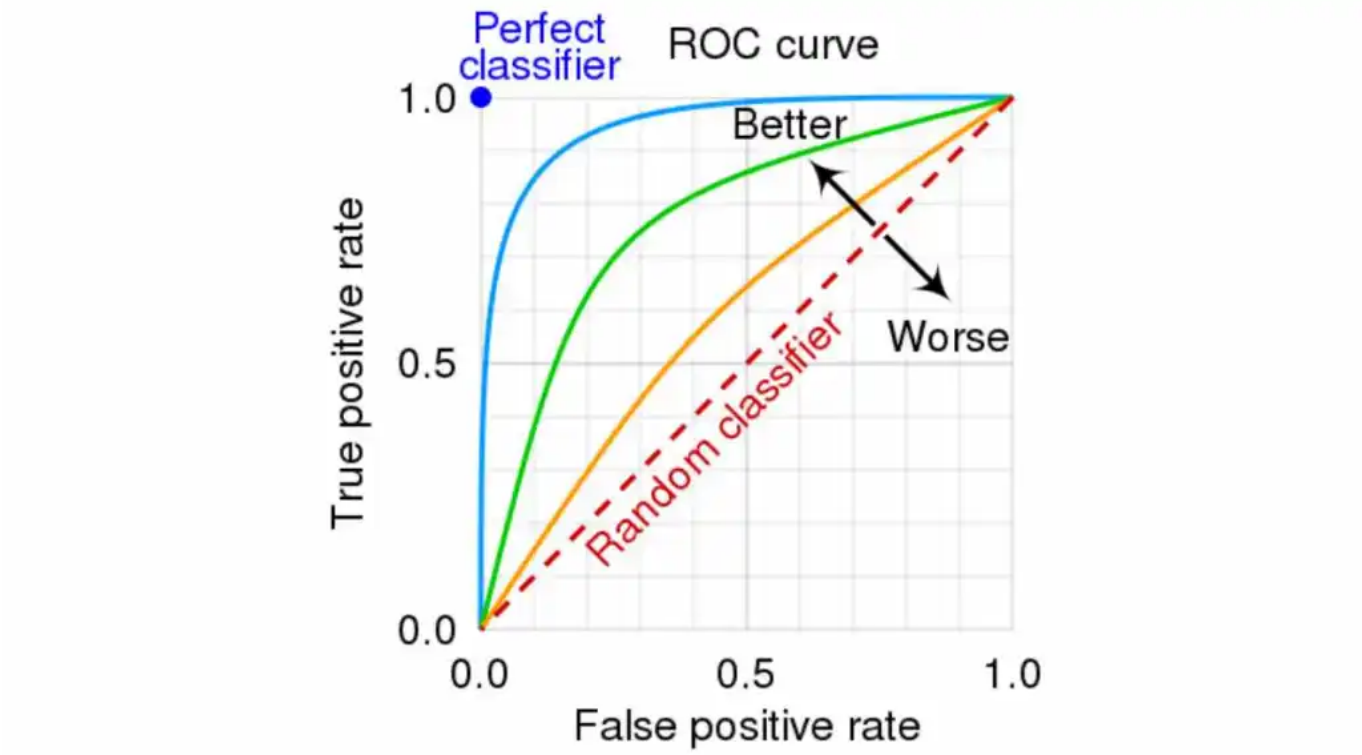
실무 구현 예시

```
# 회귀 평가 지표 계산
from sklearn.metrics import mean_squared_error, mean_absolute_error, r2_score
import numpy as np

def calculate_regression_metrics(y_true, y_pred):
    mse = mean_squared_error(y_true, y_pred)
    rmse = np.sqrt(mse)
    mae = mean_absolute_error(y_true, y_pred)
    r2 = r2_score(y_true, y_pred)

    # MAPE 계산
    mape = np.mean(np.abs((y_true - y_pred) / y_true)) * 100

    # sMAPE 계산
    smape = 2.0 * np.mean(np.abs(y_pred - y_true) / (np.abs(y_true) +
np.abs(y_pred))) * 100

    return {
        'mse': mse,
        'rmse': rmse,
        'mae': mae,
        'r2': r2,
        'mape': mape,
        'smape': smape
    }

# Huber Loss 구현
import numpy as np

def huber_loss(y_true, y_pred, delta=1.0):
    error = y_true - y_pred
    abs_error = np.abs(error)
    quadratic = np.minimum(abs_error, delta)
    linear = abs_error - quadratic
    return np.mean(0.5 * quadratic**2 + delta * linear)
```

1.3.1 ROC Curve와 PR Curve 비교 분석

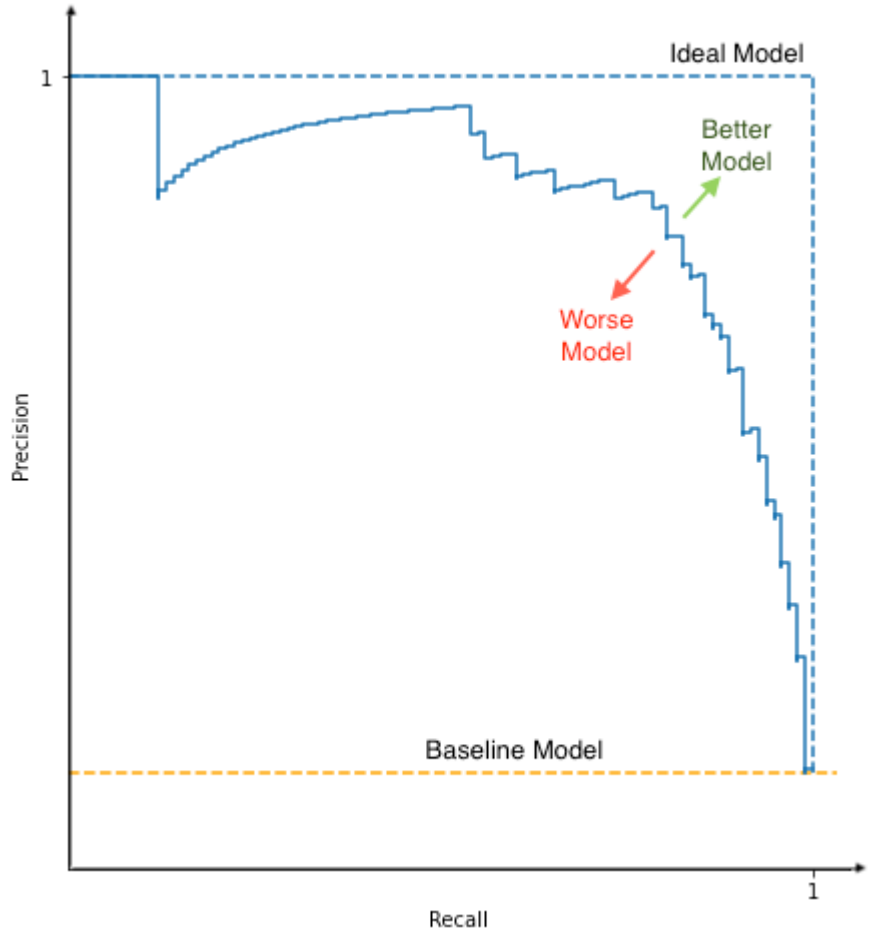
모델의 판별 능력을 임계값(Threshold) 변화에 따라 종합적으로 평가하는 두 가지 핵심 시각화 도구입니다.

ROC Curve (Receiver Operating Characteristic Curve)



가로축(FPR: False Positive Rate)과 세로축(TPR: True Positive Rate, 재현율)의 관계로 모델 성능을 시각화합니다. 곡선 아래 면적인 AUC (Area Under the Curve)는 "무작위 양성 샘플이 무작위 음성 샘플보다 높은 점수를 받을 확률"을 의미하며, 1에 가까울수록 완벽한 모델입니다. 클래스 분포에 비교적 둔감하여 일반적인 분류 성능을 평가하기에 좋습니다.

PR Curve (Precision-Recall Curve)



가로축(재현율)과 세로축(정밀도)의 관계로 성능을 시각화합니다. 클래스 불균형이 심한 데이터셋에서 ROC보다 훨씬 유용합니다. 다수 클래스(Negative)의 영향을 받지 않고 소수 클래스(Positive) 탐지 성능에 집중하기 때문입니다. 곡선 아래 면적인 AP(Average Precision)로 성능을 정량화합니다.

핵심: 일반적인 성능은 ROC, 불균형 데이터의 소수 클래스 성능은 PR Curve를 확인하는 것이 효과적입니다.

1.4.1 자연어 처리(NLP) 모델 평가 지표

NLP 모델, 특히 생성 모델의 평가는 정답이 하나로 정해져 있지 않아 더 복잡하고, 인간의 판단과 유사한 평가 지표가 필요합니다.

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

- BLEU (Bilingual Evaluation Understudy): 기계 번역 평가를 위해 개발. 생성된 문장이 정답 문장과 얼마나 많은 단어/구문(n-gram)을 공유하는지 측정. 짧은 문장에 페널티를 부여하는 Brevity Penalty를 포함하며, 정밀도에 중점을 둡니다.
- ROUGE (Recall-Oriented Understudy...): 자동 요약 평가에 주로 사용. 정답 요약의 핵심 내용(n-gram)이 생성된 요약에 얼마나 포함되었는지 측정. 재현율에 중점을 둡니다. (ROUGE-1, ROUGE-2, ROUGE-L 등)
- METEOR: BLEU/ROUGE의 한계를 넘어, WordNet 등을 활용한 동의어 매칭, 어간 추출, 단어 순서 페널티 등을 적용하여 인간의 평가와 더 높은 상관관계를 보입니다.
- 신경망 기반 지표 (BERTScore, BLEURT): BERT와 같은 사전 훈련된 언어 모델을 활용하여 단순한 단어 매칭이 아닌, 문맥적 의미 유사도를 측정하는 최신 평가 방식입니다.

1.4.2 현대적 평가 지표의 등장

신경망 기반 평가 지표

- BERTScore: BERT 임베딩을 활용하여 문맥적 의미 유사도를 측정. 전통적인 n-gram 기반 지표의 한계를 극복.
- BLEURT: BERT를 평가 태스크로 파인튜닝하여 인간 평가와의 상관관계를 최적화. 다양한 언어와 도메인에 일반화 가능.
- MoverScore: Word Mover's Distance 개념을 확장하여 단어 임베딩 간 최적 운송 문제로 모델링. 의미적 거리의 정확한 측정.

LLM 평가의 새로운 도전

- 창발적 능력 평가: Chain-of-Thought 추론 능력, Few-shot 학습 성능, 도메인 간 전이 능력.
- 안전성과 정렬 평가: 유해 콘텐츠 생성 방지, 인간 가치와의 정렬도, 적대적 공격 강건성.

Part 2: AI 모델 튜닝

AI 모델 튜닝의 단계별 접근법

딥러닝 모델의 성공적인 학습은 올바른 튜닝 전략에 크게 의존합니다. 이는 손실 함수라는 복잡한 지형에서 최적의 해를 찾아가는 과정과 같습니다.

2.1.1 전통적 머신러닝 튜닝: 하이퍼파라미터 최적화

하이퍼파라미터 튜닝은 본질적으로 블랙박스 최적화 문제입니다. 목적 함수 f(θ)가 미분 불가능하고 노이즈가 있는 상황에서 최적해를 찾아야 합니다.

하이퍼파라미터 최적화 기법들

- Grid Search: 모든 조합을 체계적으로 탐색. 완전하지만 계산 비용 높음.
- Random Search: 무작위 샘플링으로 탐색. 효율적이고 병렬화 가능.
- Bayesian Optimization: 이전 결과를 활용한 지능적 탐색. 가장 효율적.
- Population-based Methods: 유전 알고리즘, PSO 등 진화적 방법.

베이지안 최적화의 원리

이전 평가 결과를 바탕으로 다음 탐색할 하이퍼파라미터 조합을 예측하여 효율적으로 최적점을 찾습니다.

- 대리 모델 (Surrogate Model): 가우시안 프로세스 또는 Tree-structured Parzen Estimator (TPE) 등을 사용하여 목적 함수의 형태를 추정.
- 획득 함수 (Acquisition Function): Expected Improvement (EI), Upper Confidence Bound (UCB) 등을 사용하여 다음으로 평가할 최적의 하이퍼파라미터 조합을 선택.

실무 구현 예시

```
# Optuna를 이용한 베이지안 최적화
import optuna
from sklearn.model_selection import cross_val_score
from xgboost import XGBClassifier

def objective(trial):
    # 하이퍼파라미터 정의
    n_estimators = trial.suggest_int('n_estimators', 100, 1000)
    max_depth = trial.suggest_int('max_depth', 3, 10)
    learning_rate = trial.suggest_float('learning_rate', 0.01, 0.3, log=True)

    # 모델 학습
    model = XGBClassifier(
        n_estimators=n_estimators,
        max_depth=max_depth,
        learning_rate=learning_rate
    )

    # 교차 검증
    scores = cross_val_score(model, X_train, y_train, cv=5, scoring='f1')
    return scores.mean()

# 최적화 실행
study = optuna.create_study(direction='maximize')
study.optimize(objective, n_trials=100)

print(f"최적 하이퍼파라미터: {study.best_params}")
print(f"최적 성능: {study.best_value:.3f}")
```

2.1.2 딥러닝 특화 튜닝 전략

학습률 스케줄링 (Learning Rate Scheduling)

학습 과정에 따라 학습률을 동적으로 조절하는 기법입니다. 초기에는 높은 학습률로 빠르게 최적점 근처로 이동하고, 점차 학습률을 낮춰 안정적으로 수렴하도록 돕습니다.

- Step Decay: 특정 epoch마다 학습률을 일정 비율로 감소시키는 가장 간단한 방법입니다.
- Cosine Annealing: 코사인 함수 형태를 따라 부드럽게 학습률을 감소시켜, 급격한 변화 없이 안정적인 수렴을 유도합니다.
- Exponential Decay: 학습률이 지수적으로 감소.
- One Cycle Policy: 학습률을 먼저 증가시킨 후 감소시키는 방법. 빠른 수렴과 안정성 동시 확보.

가중치 초기화 (Weight Initialization)

학습 시작 전 가중치를 어떻게 설정하느냐가 학습 안정성과 속도에 큰 영향을 줍니다. 활성화 함수에 맞는 초기화 방법을 사용해야 그래디언트 소실/폭발 문제를 예방할 수 있습니다. (e.g., ReLU에는 He 초기화, Sigmoid/tanh에는 Xavier/Glorot 초기화)

실무 구현 예시

```
# PyTorch 학습률 스케줄러
import torch.optim as optim
import torch.nn as nn

# Cosine Annealing
scheduler = optim.lr_scheduler.CosineAnnealingLR(optimizer, T_max=100)

# One Cycle Policy
scheduler = optim.lr_scheduler.OneCycleLR(
    optimizer, max_lr=0.1, epochs=100, steps_per_epoch=len(train_loader)
)

# 가중치 초기화
def init_weights(m):
    if isinstance(m, nn.Linear):
        torch.nn.init.xavier_uniform_(m.weight)
        m.bias.data.fill_(0.01)

model.apply(init_weights)

# 그래디언트 클리핑
torch.nn.utils.clip_grad_norm_(model.parameters(), max_norm=1.0)
```

2.1.3 딥러닝의 고급 튜닝 기법

정규화 기법들

- Dropout: 훈련 중 일부 뉴런을 무작위로 비활성화하여 과적합 방지. 테스트 시에는 모든 뉴런 사용.
- Batch Normalization: 미니배치 단위로 정규화하여 내부 공변량 이동 문제 해결.
- Layer Normalization: 각 샘플별로 정규화. 시퀀스 모델에서 효과적.
- Weight Decay (L2 Regularization): 가중치의 크기에 페널티를 부여하여 복잡한 모델 방지.

최적화 기법들

- Adam Optimizer: 적응적 학습률과 모멘텀을 결합한 최적화 기법.
- AdamW: Weight decay를 올바르게 구현한 Adam의 개선 버전.
- RAdam: Rectified Adam. 초기 학습 불안정성 해결.
- Lookahead: 빠른 최적화와 안정성의 균형.

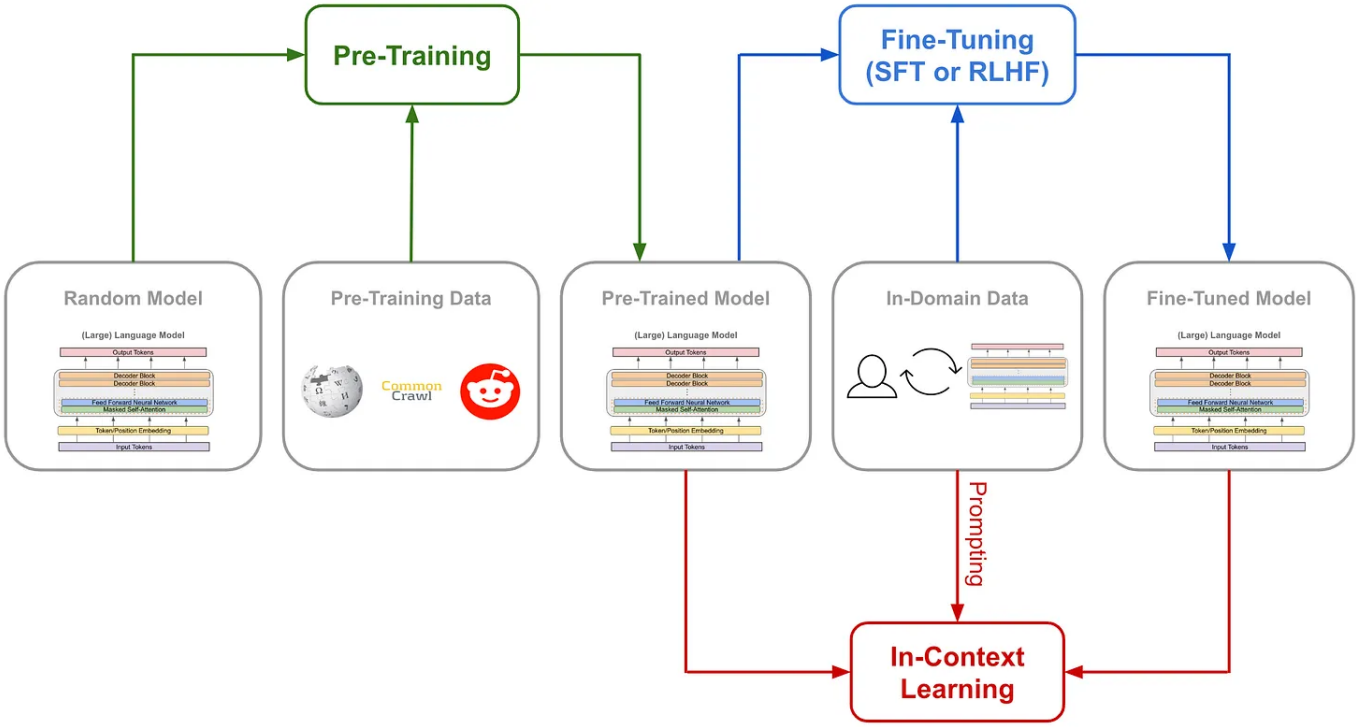
실무 구현 예시

```
# 고급 정규화 기법 구현
import torch.nn as nn

class AdvancedModel(nn.Module):
    def __init__(self):
        super().__init__()
        self.layers = nn.Sequential(
            nn.Linear(784, 512),
            nn.BatchNorm1d(512),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(512, 256),
            nn.LayerNorm(256),
            nn.ReLU(),
            nn.Dropout(0.3),
            nn.Linear(256, 10)
        )

    def forward(self, x):
        return self.layers(x)

# AdamW 최적화기
import torch.optim

optimizer = torch.optim.AdamW(
    model.parameters(),
    lr=1e-3,
    weight_decay=0.01,
    betas=(0.9, 0.999)
)

# Lookahead 래퍼
from torch.optim import Adam
base_optimizer = Adam(model.parameters(), lr=1e-3)
# optimizer = Lookahead(base_optimizer, k=5, alpha=0.5)  # Lookahead는 별도 설치 필요
```

2.1.4 생성형 AI 튜닝의 특수성

언어 모델 파인튜닝의 단계별 접근

생성형 AI는 일반적인 지도학습과 다른 특수한 고려사항이 있습니다.



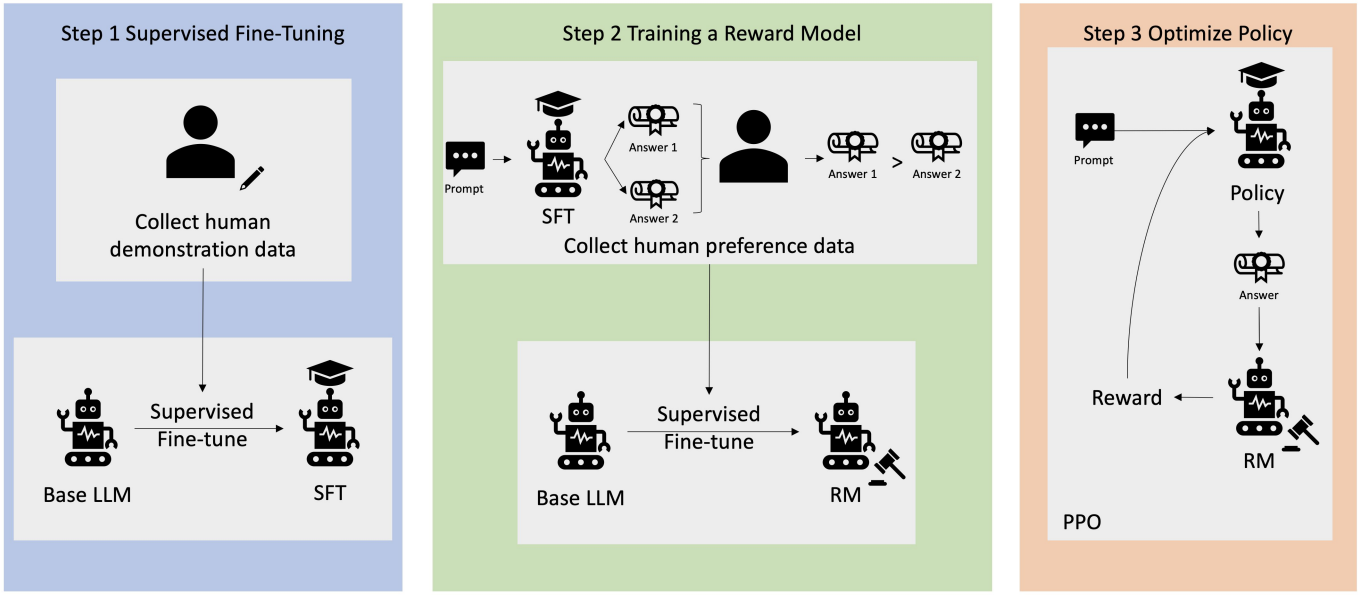
1. 사전 훈련 (Pre-training): 대규모 텍스트 코퍼스에서 자기지도 학습을 통해 언어의 통계적 구조와 표현 학습.
2. 지도 파인튜닝 (Supervised Fine-Tuning, SFT): 고품질 instruction-response 쌍으로 학습하여 태스크별 행동 패턴 학습.
3. 인간 피드백 기반 강화학습 (Reinforcement Learning from Human Feedback, RLHF): 인간 선호도 모델 학습 및 강화학습 알고리즘 적용을 통해 안전성과 유용성 균형.

2.2.1 강화학습 기반 인간친화적 튜닝: RLHF

LLM이 단순히 정확한 정보를 넘어, 유용하고(Helpful), 정직하며(Honest), 무해한(Harmless) 답변을 생성하도록 인간의 가치와 선호도를 모델에 정렬하는 과정입니다.

RLHF (Reinforcement Learning from Human Feedback)

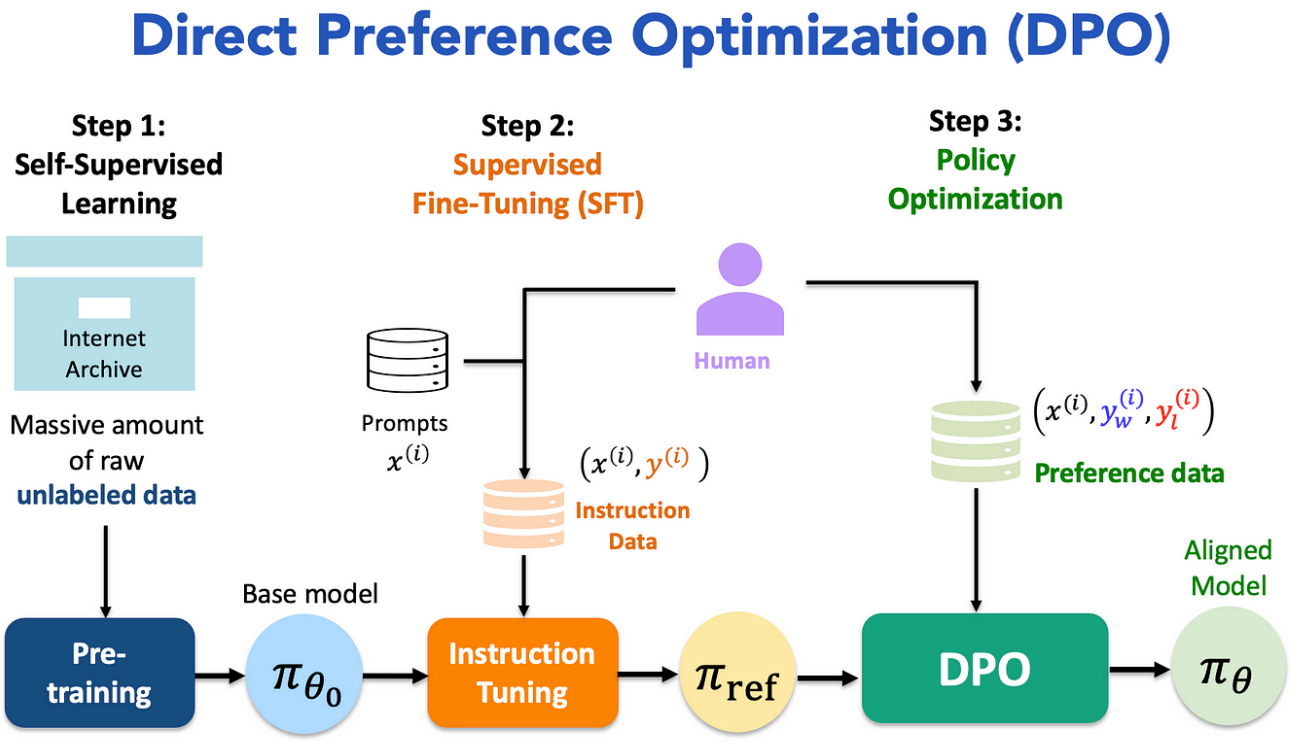
3단계로 구성된 복잡하지만 강력한 튜닝 파이프라인입니다.



1. 지도 파인튜닝 (SFT): 고품질 대화 데이터로 기본 응답 능력을 학습시킵니다.
2. 보상 모델 학습: 인간 평가자가 여러 답변 중 더 선호하는 답변을 선택한 데이터를 이용해, 좋은 답변에 높은 점수를 주는 보상 모델을 학습시킵니다.
3. 강화학습 (PPO): 이 보상 모델의 점수를 보상으로 삼아, LLM이 더 높은 점수를 받는 답변을 생성하도록 강화학습 알고리즘(PPO)으로 파인튜닝합니다.

2.2.2 강화학습 기반 인간친화적 튜닝: DPO

DPO(Direct Preference Optimization)는 RLHF의 복잡한 강화학습 단계를 제거한 최신 기법입니다. 인간의 선호도 데이터를 사용하여 보상 모델 없이 LLM을 직접 최적화하여, 더 단순하고 안정적인 학습이 가능합니다.



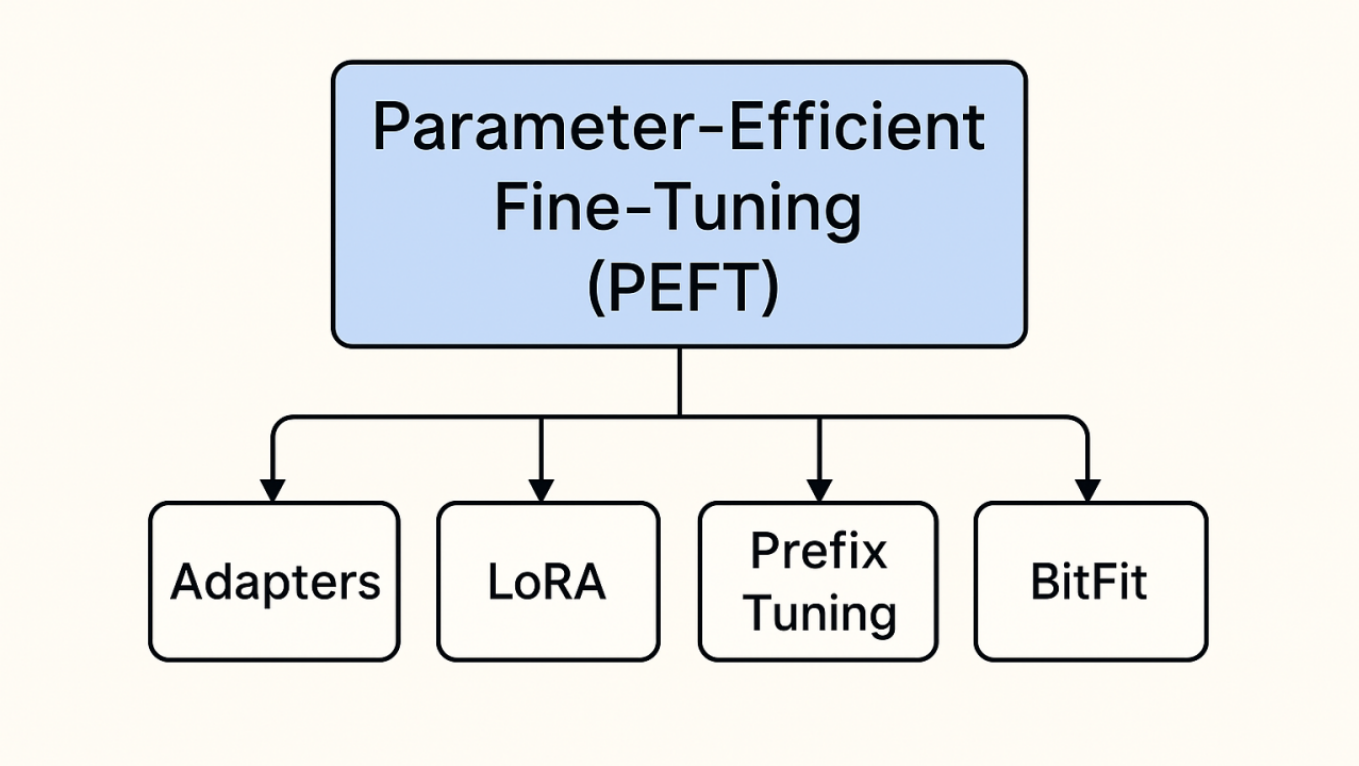
- 장점: 보상 모델 학습 단계 제거, 하이퍼파라미터 민감도 감소, 더 안정적이고 효율적인 학습, 분포 이동 문제 완화.
- 핵심 아이디어: Bradley-Terry 모델을 활용하여 인간 선호도를 직접 정책 최적화에 반영.

2.3.1 모델 경량화 기법 : PEFT와 LoRA

수천억 개의 파라미터를 가진 거대 모델을 효율적으로 파인튜닝하고 배포하기 위한 필수 기술입니다. 전체 모델을 튜닝하는 대신, 극히 일부의 파라미터만 수정하여 비용과 시간을 절약합니다.

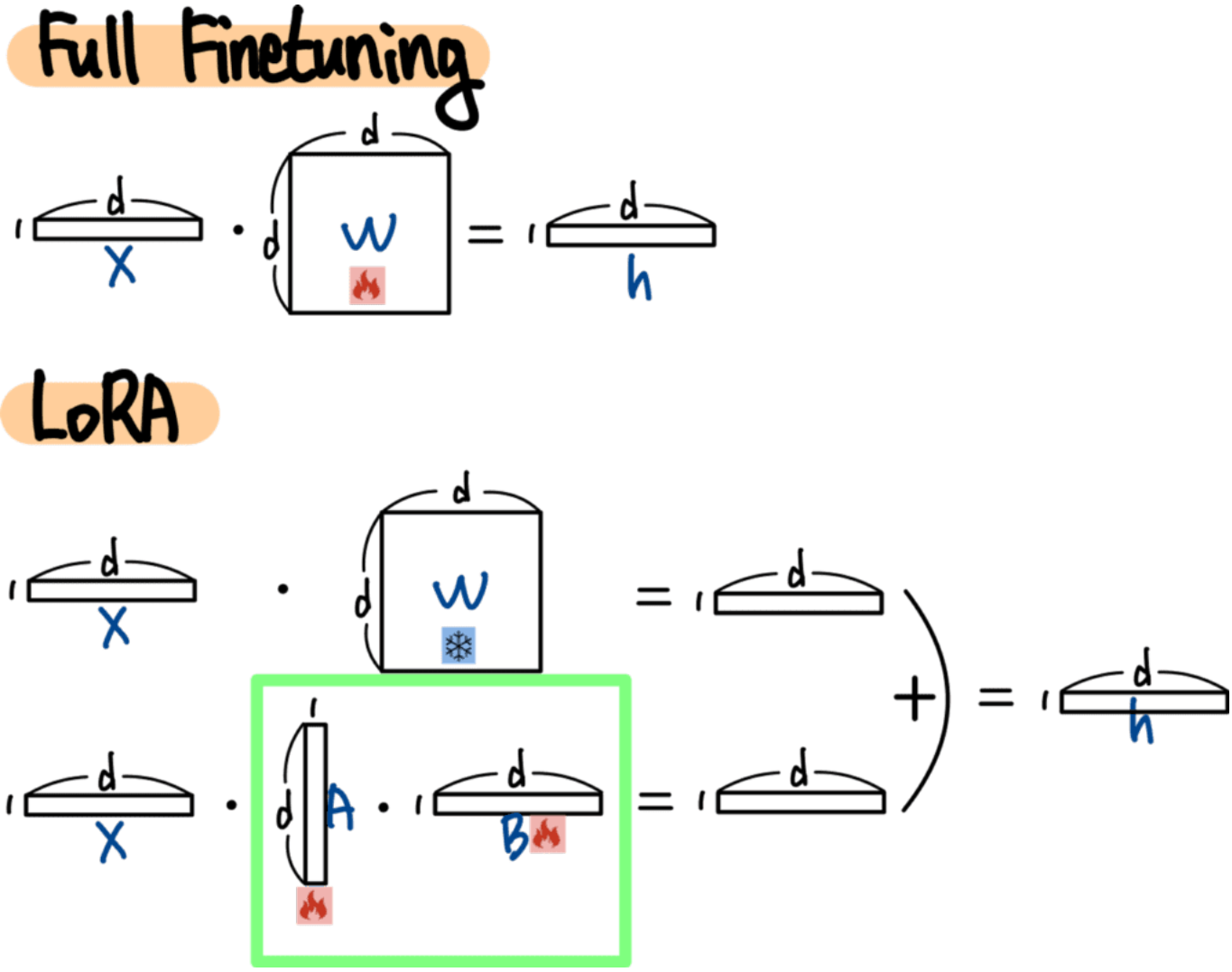
PEFT (Parameter-Efficient Fine-Tuning)

사전 훈련된 LLM의 대부분의 가중치는 고정(freeze)하고, 극히 일부의 파라미터만 새로 추가하여 학습하는 방식의 총칭입니다. (e.g., Adapter Layers, Prefix Tuning, Prompt Tuning)



LoRA (Low-Rank Adaptation)

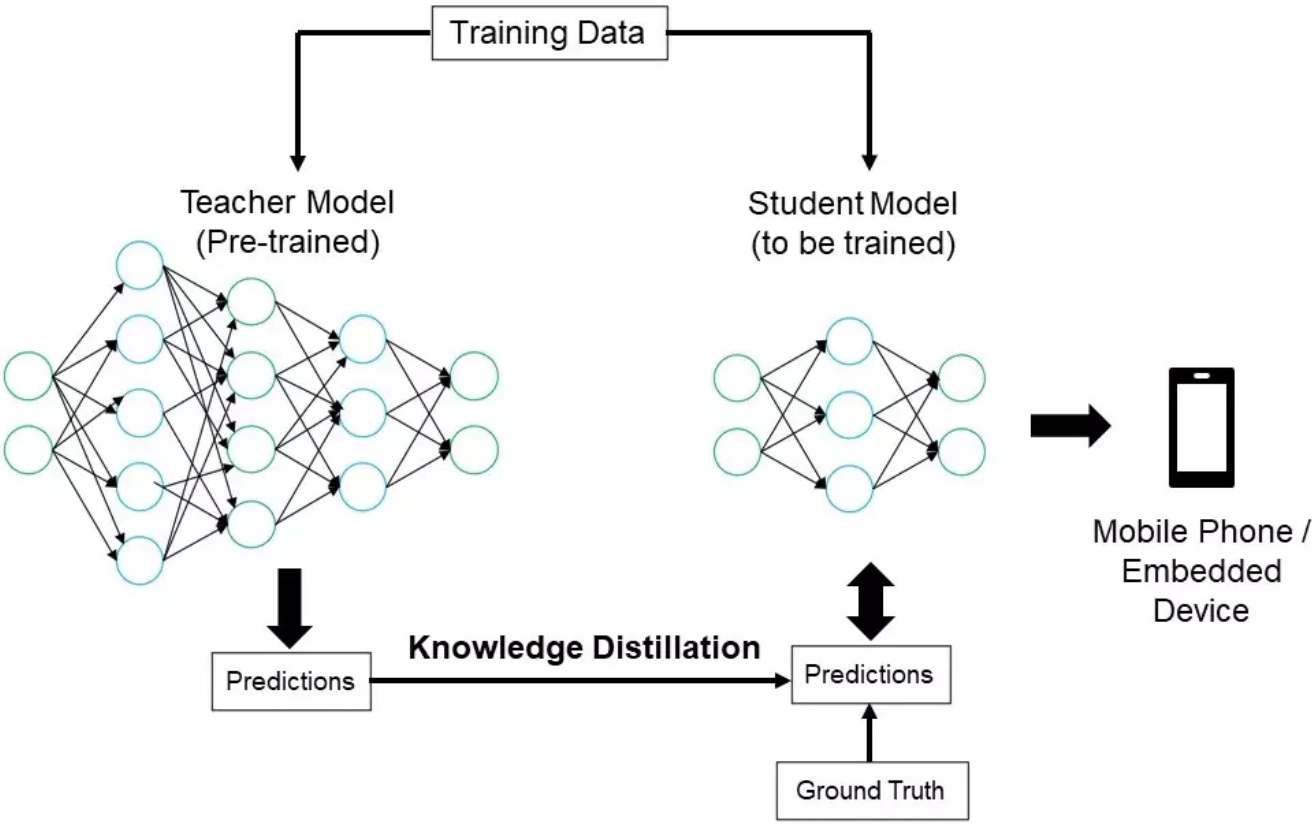
PEFT의 가장 대표적이고 효과적인 기법입니다. 거대한 가중치 행렬의 '변화량(Update)'이 실제로는 저차원(Low-Rank)의 특성을 가질 것이라는 가정하에, 이 변화량을 두 개의 작은 행렬의 곱으로 근사(분해)합니다. 이를 통해 학습 대상 파라미터 수를 99% 이상 줄이면서도, 전체를 파인튜닝하는 것과 유사한 성능을 달성할 수 있습니다.




```
# LoRA의 핵심 아이디어: 거대한 업데이트 행렬을 두 개의 작은 행렬로 분해
Weight_Update (d x k) ≈ Low_Rank_Matrix_B (d x r) * Low_Rank_Matrix_A (r x k) , (r
<< d, k)
```

2.3.2 모델 경량화 기법 : Knowledge Distillation

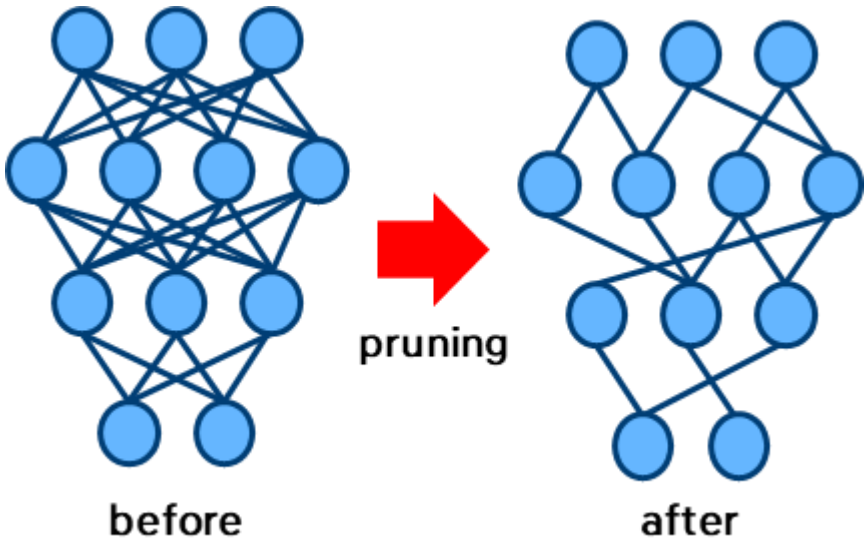
크고 성능 좋은 '교사 모델'의 예측 결과(클래스별 확률 분포인 Soft Label)를 정답으로 삼아, 작고 가벼운 '학생 모델'을 학습시키는 기법입니다. 학생 모델은 교사 모델의 정답뿐만 아니라, 정답 클래스와 다른 클래스 간의 관계까지 학습하여 더 높은 성능을 달성할 수 있습니다.



- 소프트 타겟의 정보 이론적 의미: 교사 모델의 소프트맥스 출력은 클래스 간 관계에 대한 풍부한 정보를 담고 있습니다.
- 온도의 역할: 온도(T) 매개변수를 통해 소프트맥스 분포의 부드러움을 조절하여 더 많은 정보를 전달.

2.3.3 모델 경량화 기법 : Pruning, Quantization

Pruning (가지치기)

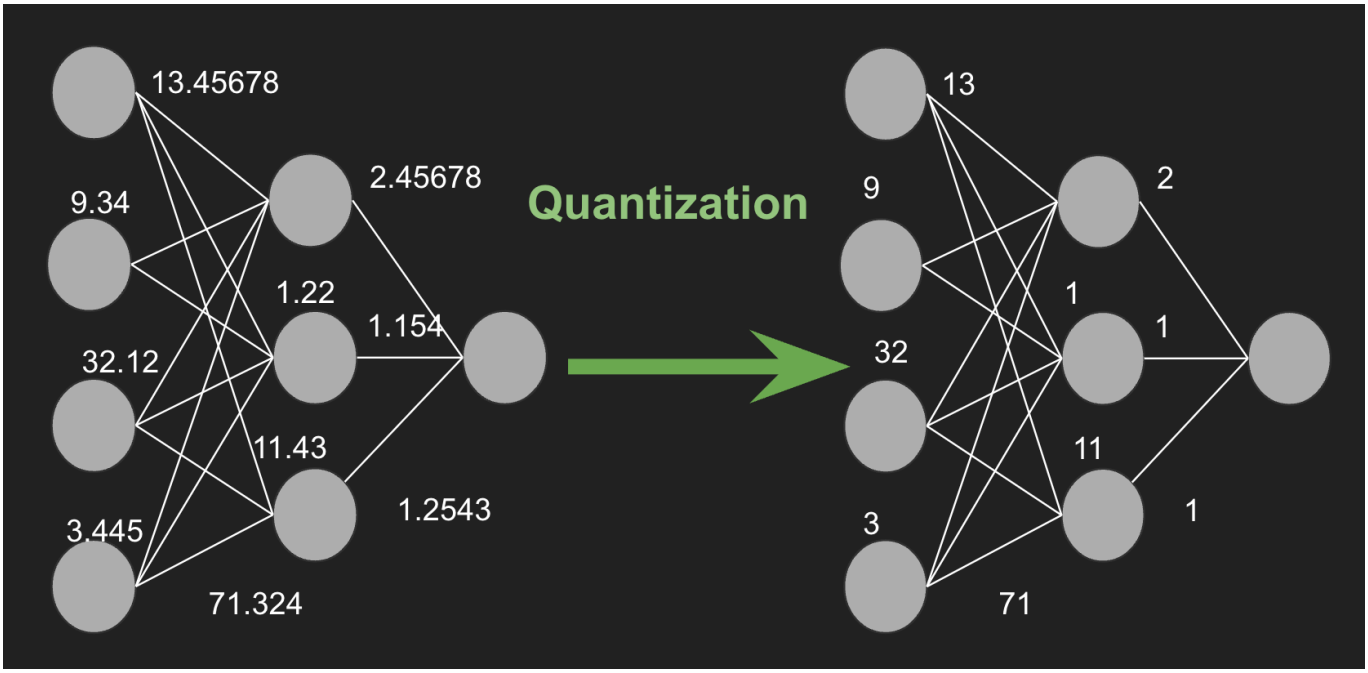


훈련된 네트워크에서 중요도가 낮은(일반적으로 가중치의 절댓값이 작은) 연결이나 뉴런을 체계적으로 제거하여 모델을 희소(sparse)하게 만드는 기술입니다. Lottery Ticket Hypothesis는 잘 훈련된 거대 네트워크 안에는 처음부터 좋은 성능을 낼 수 있었던 작은 서브네트워크가 존재한다는 이론적 기반을 제공합니다.

- 구조적 Pruning: 채널, 필터, 블록 단위로 제거하여 일반적인 하드웨어에서 가속 가능.

- 비구조적 Pruning: 개별 가중치 단위로 제거하여 높은 압축률 가능하나 특수 하드웨어 필요.
- Dynamic Pruning: 추론 시 동적으로 불필요한 계산을 건너뛰는 기법.

Quantization (양자화)



모델의 가중치를 표현하는 데이터 타입을 32비트 부동소수점(FP32)에서 16비트(FP16/BF16)나 8비트 정수(INT8) 등으로 변환하여, 모델 크기를 획기적으로 줄이고 추론 속도를 향상시키는 기술입니다. 훈련 후 적용하는 PTQ(Post-Training Quantization)와 훈련 과정에 양자화를 시뮬레이션하는 QAT(Quantization-Aware Training)가 있습니다.

실무 구현 예시

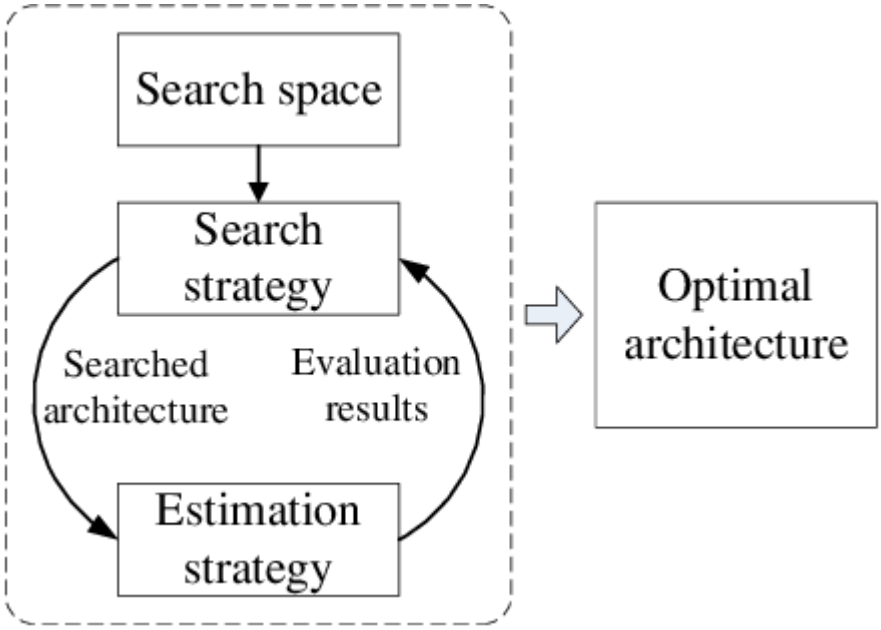
```
# PyTorch Pruning 구현
import torch.nn.utils.prune as prune
import torch.nn as nn

# L1-norm 기반 구조적 가지치기
def prune_model(model, amount=0.3):
    for name, module in model.named_modules():
        if isinstance(module, nn.Conv2d):
            prune.l1_unstructured(module, name='weight', amount=amount)
            prune.remove(module, 'weight')
    return model

# Quantization 구현
import torch.quantization

def quantize_model(model):
    # 동적 양자화
    quantized_model = torch.quantization.quantize_dynamic(
        model, {nn.Linear, nn.Conv2d}, dtype=torch.qint8
    )
    return quantized_model
```

```
# QAT (Quantization-Aware Training)
def prepare_qat(model):
    model.train()
    model.qconfig = torch.quantization.get_default_qat_qconfig('fbgemm')
    torch.quantization.prepare_qat(model, inplace=True)
    return model
```

2.3.4 모델 경량화의 고급 기법

Neural Architecture Search (NAS)

자동으로 최적의 신경망 아키텍처를 탐색하는 기법입니다. 수동 설계보다 더 효율적이고 성능 좋은 모델을 찾을 수 있습니다.



- Reinforcement Learning based NAS: 강화학습을 사용하여 아키텍처 탐색.
- Evolutionary NAS: 유전 알고리즘을 사용한 아키텍처 진화.
- One-Shot NAS: 슈퍼넷을 한 번 훈련하고 서브넷을 평가하는 효율적 방법.

Model Distillation의 고급 기법

- Feature Distillation: 중간 레이어의 특징 맵을 전달하여 더 풍부한 정보 학습.
- Attention Transfer: 어텐션 패턴을 전달하여 모델의 해석 가능성 향상.
- Progressive Distillation: 점진적으로 작은 모델로 지식 전달.

실무 구현 예시

```
# Feature Distillation 구현
import torch.nn as nn
import torch.nn.functional as F

class DistillationLoss(nn.Module):
```

```
    def __init__(self, alpha=0.7, temperature=4.0):
        super().__init__()
        self.alpha = alpha
        self.temperature = temperature
        self.ce_loss = nn.CrossEntropyLoss()
        self.kl_loss = nn.KLDivLoss(reduction='batchmean')

    def forward(self, student_logits, teacher_logits, targets):
        # Hard target loss
        ce_loss = self.ce_loss(student_logits, targets)

        # Soft target loss
        soft_loss = self.kl_loss(
            F.log_softmax(student_logits / self.temperature, dim=1),
            F.softmax(teacher_logits / self.temperature, dim=1)
        ) * (self.temperature ** 2)

        return self.alpha * ce_loss + (1 - self.alpha) * soft_loss

# 모델 압축 평가
def evaluate_compression(original_model, compressed_model, test_loader):
    original_size = sum(p.numel() for p in original_model.parameters())
    compressed_size = sum(p.numel() for p in compressed_model.parameters())
    compression_ratio = original_size / compressed_size

    # 성능 비교
    original_acc = evaluate_model(original_model, test_loader)
    compressed_acc = evaluate_model(compressed_model, test_loader)

    return {
        'compression_ratio': compression_ratio,
        'accuracy_drop': original_acc - compressed_acc,
        'size_reduction': (1 - 1/compression_ratio) * 100
    }
```

2.4.1 평가 지표 선택 전략

비즈니스 목표를 명확히 정의하는 것이 우선입니다. 의료 AI에서는 재현율이, 금융 AI에서는 정밀도와의 균형이, NLP에서는 태스크 특화 지표와 인간 평가가 중요합니다. 단일 지표에 매몰되지 말고, 여러 지표를 종합적으로 해석하는 능력이 필요합니다.

도메인별 핵심 지표

- 의료 AI: 민감도(Recall), 특이도, AUC, Positive Predictive Value.
- 금융 AI: 정밀도, 재현율, F1-Score, ROI 기반 지표.
- 자연어 처리: BLEU, ROUGE, Perplexity, Human Evaluation.

2.4.2 튜닝 및 경량화 전략 수립

프로젝트의 제약 조건(메모리, 지연시간, 정확도)에 따라 최적의 기법을 조합해야 합니다.

- 정확도 손실 최소화가 최우선일 때: LoRA, QAT(Quantization-Aware Training)
- 메모리 및 속도 최우선일 때: Post-Training Quantization, Pruning
- 인간의 선호도와 가치 반영이 중요할 때: RLHF, DPO

성능 모니터링과 지속적인 피드백 루프를 통해 모델을 최신 상태로 유지하고, 데이터 드리프트에 대응하는 자동화된 재훈련 체계를 구축하는 것이 성공적인 AI 모델 운영의 핵심입니다.