1부 딥러닝 기초 - 4장 머신 러닝의 기본 요소

시작일	
링크	
목표	월/0725
종료일	

4장 머신 러닝의 기본 요소

- 분류/회귀 이외의 머신 러닝 형태
- 머신 러닝 올바른 평가 과정
- 딥러닝을 위한 데이터 전처리
- 특성 공학
- 과대적합 문제 해결
- 머신 러닝 문제를 다루는 일반적인 작업 흐름

4.1 머신 러닝의 네 가지 분류

4.1.1 지도 학습

- 시퀀스 생성(sequence generation)
  - 사진이 주어지면 이를 설명하는 캡션을 생성합니다
  - 시퀀스 생성: 단어/토큰(token)을 반복 예측처럼 분류 문제로 재구성 가능
- 구문 트리 (syntax tree) 예측
  - 문장이 주어지면 분해된 구문 트리 예측
- 물체 감지(object detection)
  - 사진 주어지면, 사진 안의 특정 물체 주위에 경계 상자(bounding box)를 그림
  - 각 상자의 내용을 분류하는 : 분류 문제
  - 경계 상자의 좌표를 벡터 회귀로 에측: 예측하는 회귀/분류 문제

- 이미지 분할(image segmentation)
    - 픽셀 단위로 특정 물체에 마스킹(masking)

4.1.2 비지도 학습

- 데이터 시각화, 데이터 압축, 데이터 노이즈 제거 등 데이터의 상관관계 이해
    - 종종 지도 학습 문제 전, 데이터셋 이해를 위해 사용되기도 함
    - 차원 축소(dimensionality reduction)
    - 군집(clustering)
- 학습 과정에 사람이 개입하기 않음
- 레이블이 여전히 필요하지만, 경험적인 알고리즘(heuristic algorithm) 사용, 입력 데이터로부터 생성
- ex. 오토인코더(Autoencoder)
    - 생성된 타깃은 수정하지 않은 원본 입력
- 시간에 따른 지도 학습
    - temporally supervised learning
    - 지난 프레임 → 다음 프레임 예측
    - 이전 단어 → 다음 단어 예측 (미래의 입력 데이터로부터 지도되기 때문에

4.1.4 강화 학습

- reinforcement learning
- 에이전트(agent)는 환경에 대한 정보를 받아 **보상을 최대화**하는 행동을 선택하도록 학습됨

분류와 회귀에서 사용하는 용어

- 샘플/입력 → 예측/출력 (타깃)
- 예측 오차/손실 값
- 클래스/레이블 (클래스 할당의 구체적 사례)
- 참 값/꼬리표
- 이진분류/다중분류/다중 레이블 분류
- 스칼라 회귀/벡터 회귀(연속적인 값의 집합)

- 미니 배치: 소량의 샘플 묶음

4.2 머신 러닝 모델 평가

- 과대적합을 막고 새로운 데이터셋에 대해서도 잘 작동하는 일반화된 모델을 얻어야 함

4.2.1 훈련, 검증, 테스트 세트

- 하이퍼파라미터(hyperparameter) : 층의수, 층의 유닛 수
- 학습: 어떤 파라미터 공간에서 좋은 설정을 찾음
- 정보 누설(information leak)
  - 검증 세트의 모델 성능에 기반하여 모델의 하이퍼파라미터 조정할 때마다 검증 데이터에 관한 정보가 모델로 샘

훈련/검증/테스트 세트로 나누는 고급 기법

1. `단순 홀드아웃 검증(hold-out validation)`
2. `K-겹 교차 검증(K-fold cross-validation)`
3. `셔플링(shuffling) 을 사용한 K겹 교차 검증(iterated K-fold cross-validation)`

1. **단순 홀드아웃 검증**

- 데이터의 일정량을 테스트 세트로 떼어 놓음
  - 훈련/ 홀드아웃 검증 세트

  - train_test_split() 로 보통 나눔
  - 단점: 데이터가 적을 대, 주어진 전체 데이터를 통계적으로 대표하지 못할 수 있음

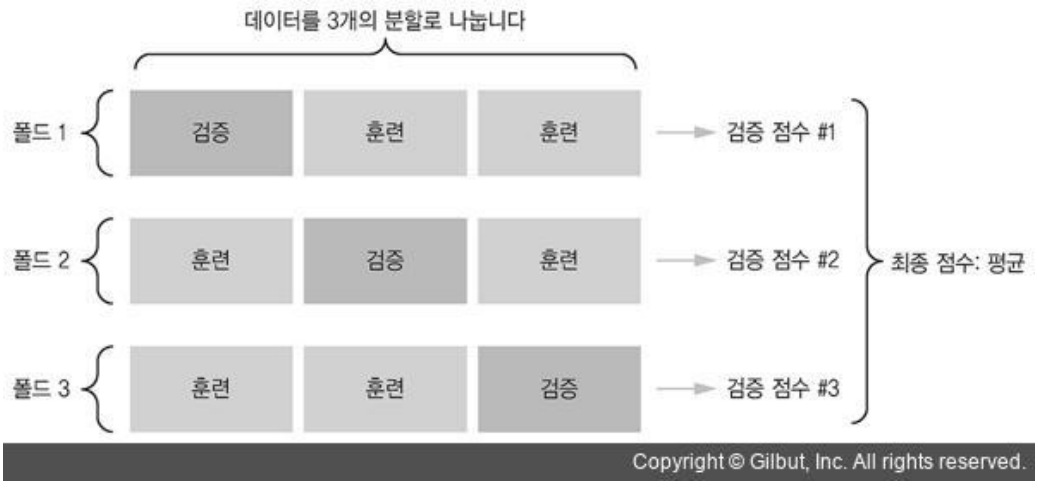
- 난수 초깃값 셔플링해서 모델의 성능이 매우 달라지는걸로 확인 가능

  → K겹 교차 검증 / 반복 K-겹 교차 검증 필요

```
# 홀드아웃 검증 구현 예
num_validation_samples = 10000

np.random.shuffle(data) # 데이터를 섞는 것(셔플링)이 일반적으로 좋음

validation_data = data[:num_validation_samples] # 검증 데이터셋
data = data[num_validation_samples:]
training_data = data[:]

# 훈련 세트에서 모델을 훈련하고 검증세트로 평가함
model = get_model()
model.train(training_data)
val_score= model.evaluate(vaildation_data)

# 여기서 모델 다시 튜닝, 다시 훈련, 평가, 튜닝 ...
  # 하이퍼파라미터 튜닝이 끝나면 테스트 데이터를 제외한 모든 데이터를 사용하여
  # 모델을 다시 훈련시킴
model = get_model()
model.train(np.concatenate([training_data, validation_data])
test_score= model.evaluate(test_data)
```

2. K겹 교차 검증(K-fold cross-validation)

- 데이터를 동일한 크기를 가진 K개 분할로 나눔
  - K-1개로 훈련, 분할 i에서 모델을 평가 / 이렇게 얻은 K개의 점수를 평균
  - K겹 교차 검증: cross_validate() 으로 구현
  - 케라스 모델을 사이킷런과 호환되도록 KerasClssifier / KerasRegressor 클래스 모델로 감쌈



```
# K-겹 교차 검증 구현 예
k = 4
num_validation_samples = len(data) // k

np.random.shuffle(data)

validation_scores = []
for fold in range(k):
  # 검증 데이터 선택
  validation_data = data[num_validation_samples * fold:
                         num_validation_samples * (fold+1)]
  # 남은 데이터를 훈련 데이터로 사용
  # 리스트에서 + 연산자는 두 리스트를 연결
  # training_data = np.concatenate(data[:num_validation_samples * (fol
d)],
  #                                data[num_validation_samples* (fold+1)
:])
  training_data = data[:num_validation_samples * (fold)] + data[num_valid
ation_samples* (fold+1) :]

# 훈련되지 않는 새로운 모델을 만듦
model = get_model()
model.train(training_data)
val_score = model.evaluate(validation_data)
validation_scores.append(val_score)

# 검증 점수 : K개 폴드의 검증 점수 평균
val_score = np.average(validation_scores)

# 새로운 model
  # 테스트 데이터를 제외한 전체 데이터로 최종 모델 훈련
model = get_model()
model.train(data)
test_score= model.evaluate(test_data)
```

3. **셔플링을 사용한 반복 K겹 교차 검증(K-fold cross-validation)**

- K-겹 교차 검증을 여러 번 적용하되 K개의 분할로 나누기 전에 매번 데이터를 무작위로 섞음
- 최종 점수는 모든 K-겹 교차 검증 점수의 평균이 됨
- P * K개(P는 반복 횟수)의 모델 훈련/평가

4.2.2 평가 방식 선택 시, 기억해야 할 것

- `대표성 있는 데이터`
  - 훈련 세트, 테스트 세트가 주어진 데이터에 대한 `대표성`이 있어야 함
  - ex. 클래스 순서대로 나열된 경우 (8:2로 나누면)
    - 훈련: 0~7만 학습하고 / 테스트: 8~9만 담김

    - 세트로 나누기 전 데이터를 `무작위로 섞어야 함`
- `시간의 방향`
    - 과거로부터 미래를 예측 (날씨/주식 시세)
    - `데이터를 무작위로 섞으면 안됨 -> 미래 데이터가 누설되기 때문`
    - 훈련 세트에 있는 데이터보다 테스트 세트에 있는 데이터가 미래의 것이어야 함
- `데이터 중복`
    - 한 데이터셋에서 어떤 데이터 포인트가 두 번 등장
    - 훈련 & 검증 → 데이터 포인트가 중복될 수 있음
    - 데이터 일부로 테스트 하는 최악의 경우가 됨
    - `훈련 / 검증 세트가 중복되지 않도록 함`

4.3 데이터 전처리, 특성 공학, 특성 학습

- 많은 데이터 전처리 & 특성 공학 → 도메인 종속적

4.3.1 신경망을 위한 데이터 전처리

- 벡터화, 정규화, 누락된 값 다루기, 특성 추출
    - `벡터화(vectorization)`
        - 입력 & 타깃이 텐서여야 함 → 텐서로 변환(데이터 벡터화, data vectorization)
        - ex. 텍스트를 정규 시퀀스로 변환 → 원-핫 인코딩
    - `값-정규화(normalization)`
        - Image grayscale: 0~255
        - float 타입으로 변경 → 값/255 → 0~1 소수값으로 변경
        - 주택 가격 : 각 특성별 범위를 독립적으로 정규화 → 평균 0, 표준편차 1
            - 네트워크를 쉽게 학습하기 위해
            - 1) 작읍 값 사용 : 대부분 0~1
            - 2) 균일해야 함 : 모든 특성 대체로 비슷한 범위
            - 즉, 각 특성별 평균이 0이 되도록 정규화 / 특성별 표준 편차가 1이 되도록 정규화

```
x -= x.mean(axis= 0) # x가 (샘플, 특성) 크기인 2D 행렬
x /= x.std(axis = 0)
```

- 누락된 값 다루기
  - 일반적으로 신경망 0
    - 사전에 정의된 의미 있는 값 아니면 0으로 대체해도 됨
    - 누락된 데이터 뜻하며, 이 값을 무시할 것
  - 테스트 데이터 누락된 값 포함될 가능성 있음
    - 누락된 값이 없는 데이터서 훈련되었다면, 누락된 값 무시하는 방법 학습 X
    - 이런 경우, 누락된 값이 있는 훈련 샘플을 고의적으로 만들어야 함
    - 훈련 샘플의 일부를 여러벌 복사해서 테스트 데이터에서 빠질 것 같은 특성 제거해야 함

4.3.2 특성 공학

```
# ex. 시계의 시간을 읽기 위한 특성 공학
  # 원본 데이터 : 2차원 픽셀 데이터 (시계 그림)
  # 더 나은 특성: 시계 바늘의 좌표 { x1: 0.7, y1:0.7 } { x2 : 0.5, y2 : 0.0 }
  # 훨씬 더 좋은 특성: 시계 바늘의 각도 : theta : 45, theta : 0
```

- 최근 딥러닝에서는 특성 공학이 필요하지 않음
- 신경망이 자동으로 원본 데이터에 유용한 특성을 추출함
  - But, 좋은 특성은 적은 자원을 사용하여 문제를 풀 수 있음
  - 좋은 특성은 더 적은 데이터로 문제를 풀 수 있음
    - 딥러닝 모델이 스스로 특성을 학습하는 능력은 가용한 훈련 데이터가 많을 때 발휘됨
    - 샘플의 개수가 적다면 특성에 있는 정보가 매우 중요함

4.4 과대적합과 과소적합

- 영화 리뷰 예측, 토픽 분류, 주택 가격 회귀
  - 모델이 훈련 데이터에 과대적합(overfitting)/과소적합(underfitting)됨

- 머신러닝: 1) 최적화 2) 일반화의 줄다리기
  - 1) 최적화(optimization) : 가능한 훈련 데이터에서 최고의 성능을 얻으려고 모델을 조정하는 과정
  - 2) 일반화(generalization) : 훈련된 모델이 이전에 본 적 없는 데이터에서 잘 수행되도록 함
- 과대적합 처리 과정 → 규제(regularization)

4.4.1 네트워크 크기 축소

- 모델의 크기, 즉 모델에 있는 학습 파라미터 수를 줄이는 것
- 층의 수, 각 층의 유닛 수에 결정됨 : 모델의 용량(capacity)
- 데이터에 알맞은 모델 크기를 찾으려면 각기 다른 구조로 평가해야 함

```
# 원본 모델
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input = (10000,)))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

```
# 작은 용량의 모델
from keras import models
from keras import layers

model = models.Sequential()
model.add(layers.Dense(6, activation = 'relu', input = (10000,)))
model.add(layers.Dense(6, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

- 작은 네트워크가 기본 네트워크보다 더 나중에 과대적합됨
- 과대적합에서 성능이 점점 감소함

```
# 큰 용량의 모델
model = models.Sequential()
model.add(layers.Dense(1024, activation = 'relu', input = (10000,)))
model.add(layers.Dense(1024, activation = 'relu'))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

- 용량이 큰 네트워크, 거의 바로 과대적합 시작되어 갈수록 더 심해짐

- 용량이 큰 네트워크 훈련 손실 매우 빠르게 0에 가까워짐, 과대적합에 민감해짐
- 간단한 모델이 복잡한 모델보다 덜 과대적합될 가능성이 높음
- 과대적합 완화 → 간단한 모델
  - 파라미터 값 분포의 엔트로피가 적은 모델
  - 적은 수의 파라미터를 가진 모델
  - 가중치가 작은 값을 가지도록 강제 → 가중치 규제 (weight regularization)
    - `L1 규제`
      - 가중치의 절댓값에 비례하는 비용 추가 (L1 norm)
    - `L2 규제`
      - 가중치의 제곱에 비례하는 비용 추가 (L2 norm
      - 가중치 감쇠(weight decay)

```
# 모델에 L2 가중치 추가하기
  # 가중치 규제 객체를층의 키워드 매개변수로 전달
  # l2(0.001)는 가중치 행렬의 모든 원소를 제곱, 0.01을 곱하여 네트워크의 전체 손실에 더해짐
  # penalty는 훈련할 때만 추가됨

from keras import regularizers

model = models.Sequential()
model.add(layers.Dense(16,kernel_regularizer = regularizers.l2(0.01),
                       activation = 'relu', input_shape = (10000,)))
model.add(layers.Dense(16,kernel_regularizer = regularizers.l2(0.01),
                       activation = 'relu'))
model.add(layers.Dense(1, activation ='sigmoid'))
```

```
# 케라스에서 사용할 수 있는 가중치 규제
from keras import regularizers

regularizers.l1(0.001) # l1규제

regularizers.l1_l2(l1 = 0.001, l2 = 0.001) # l1, l2규제 병행 -> ElasticNet
```

4.4.3 드롭아웃 추가

- 드롭아웃(Dropout)
  - 무작위로 층의 일부 출력 특성을 제외시킴 (0으로 만듦)
  - 층의 출력 값에 노이즈를 추가하여 중요하지 않은 우연한 패턴을 깨트리는 것

- 노이즈가 없다면 네트워크가 이 패턴을 기억하기 시작할 것
- 드롭아웃 비율 : 0이 될 특성의 비율 (0.2~ 0.5)
- (batch_size, features), 훈련 시 행렬 값의 일부가 랜덤하게 0이 됨
- `layer_output *= np.random.randint(0, high = 2, size = layer_output.shape)` # 유닛의 출력 중 50%를 버림
- 테스트할 때는 드롭아웃 비율로 출력을 낮추어 주어야 함 (0.5배)
- `layer_output *= 0.5` # 테스트 단계
- layer_output * = np.random.randint(0, hight= 2, size = layer_output.shape) # 훈련 단계
- layer_output /= 0.5

```
# keras에서 층의 출력 바로 뒤에 Dropout 층을 추가
# 네트워크게 드롭아웃 적용

model.add(layers.Dropout(0.5))
```

```
# IMDB 네트워크에 드롭아웃 추가하기
model = models.Sequential()
model.add(layers.Dense(16, activation = 'relu', input_shape = (10000,)))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(16, activation = 'relu'))
model.add(layers.Dropout(0.5))
model.add(layers.Dense(1, activation = 'sigmoid'))
```

- 과대적합 방지
  - 훈련 데이터 수집
  - 네트워크 용량 감소
  - 가중치 규제 추가
  - 드롭아웃 추가

4.5 보편적인 머신 러닝 작업 흐름

- 문제 정의, 평가, 특성 공학, 과대적합의 개념

4.5.1 문제 정의와 데이터셋 수집

- 문제 정의: 가용 데이터의 유무, 문제의 종류(분류/회귀)

- 가설
  - 주어진 입력으로 출력을 예측할 수 있다고 가설을 세움
  - 가용한 데이터에 입력/출력 사이의 관계를 학습
- 시간에 따라는 문제 (nonstationary problem) → 풀기 어려운 문제
  - 최근의 데이터로 주기적으로 모델을 다시 훈련 or
  - 시간 분포에 맞게 데이터를 수집, 시간에 따라 변하지 않는 문제로 바꿈

4.5.2 성공 지표 선택

- 클래스 분포가 균일한 분류 문제
  - 정확도 / ROC AUC 가 일반적
- 클래스 분포가 균일하지 않은 문제
  - 정밀도 / 재현율
- 랭킹 문제 / 다중 레이블 문제
  - 평균 정밀도 사용

4.5.3 평가 방법 선택

- 홀드아웃 검증 세트 분리
  - 데이터가 풍부할 때 사용
- K-겹 교차 검증
  - 홀드아웃 검증을 사용하기에, 샘플의 수가 너무 적을 때 사용
- 반복 K-겹 교차 검증
  - 데이터가 적고 매우 정확한 모델 평가가 필요할 때 사용

4.5.4 데이터 준비

- 1. 무엇을 훈련할지 2. 무엇을 최적화할지 3. 어떻게 평가할지 → 훈련 준비
- 모델에 주입할 데이터 구성
  - 데이터는 텐서로 구성됨
  - 텐서에 있는 값, 작은 값으로 스케일 조정되어 있음 [-1,1], [0,1]
  - 특성마다 범위가 다르면 정규화
  - 특성 공학 (특히, 데이터가 적을 때)

4.5.5 기본보다 나은 모델 훈련하기

- 통계적 검정력(statistical power)을 달성
  - 적어도 데이터셋에 있는 `클래스별 분포보다 모델의 정확도가 높아야 함`
  - 아주 단순한 모델보다 나은 수준의 작은 모델을 개발
  - MNIST (0.1보다 높은 정확도) : 0~9 (10개) / 100/10 = 0.1
  - IMDB (0.5보다 높은 정확도) : 이진분류 / 100/2 = 0.5
- 통계적 검정력을 위한 가설
  - 주어진 입력으로 출력을 예측할 수 있어야 함
  - 가용한 데이터에 입력과 출력 사이의 관계를 학습하는데 충분한 정보가 있다는 가설
- 모델을 만들기 위한 중요한 선택
  - 마지막 층의 활성화 함수
    - 네트워크 출력에 필요한 제한
      - 분류: ex. IMDB 이중 분류 - sigmoid
      - 회귀 - 마지막 층 활성화 함수 X
  - 손실 함수
    - 이중분류 IMDB : binary_crossentropy
    - 회귀 : mse
  - 최적화 설정
    - 어떤 옵티마이저? ex. 대부분 rmsprop
    - 학습률? ex. 대부분 기본 학습률
    - 옵티마이저 - 기본 학습률
      - rmsprop, adam - 0.001
      - sgd, adagrad - 0.01

문제 유형	마지막 층의 활성화 함수	손실 함수
이진 분류	시그모이드	binary_crossentropy
단일 레이블 다중 분류	소프트맥스	categorical_crossentropy
다중 레이블 다중 분류	시그모이드	binary_crossentropy

임의 값에 대한 회귀	없음	mse
0과 1 사이 값에 대한 회귀	시그모이드	mse 또는 binary_crossentropy

4.5.6 몸집 키우기: 과대적합 모델 구축

- 과대적합된 모델 만들기 위해 - 1. 층 추가 2. 층의 크기 키움 3. 더 많은 에포크 훈련
  - 훈련 손실, 검증 손실을 모니터링
  - 검증 데이터에서 모델 성능 감소하기 시작할 때 → 과대적합

4.5.7 모델 규제와 하이퍼파라미터 튜닝

- 드롭아웃 추가
- 층을 추가하거나 제거해서 다른 구조를 시도해봄
- L1, L2 또는 두가지 모두 추가
- 최적의 설정을 찾기 위해 하이퍼파라미터를 바꾸어 시도 (층의 유닛 수/ 옵티마이저의 학습률 등)
- 선택적으로 특성 공학을 시도 (새로운 특성 추가 / 유용하지 않을 특성 제거)

4.6 요약

- 주어진 문제 , 훈련할 데이터 정의
  - 데이터 수집 & 레이블 태깅
  - 성공 지표 결정 & 검증 데이터에서 모니터링 할 지표
  - 평가 방법 (홀드아웃 검증, K-겹 교차 검증) & 검증 데이터셋 양
  - 단순한 랜덤 선택 모델보다 나은 통계적 검정력 있는 모델 생성
  - 과대적합된 모델을 만듦
  - 검증 데이터의 성능에 기초하여 모델에 규제를 적용 & 하이퍼파라미터 튜닝