머신 러닝 모델 평가

머신 러닝의 목표는 처음 본 데이터에서 잘 작동하는 일반화된 모델을 얻는 것이다. 여기서 과대적합은 주요 장애물이다. 과대적합을 완화하고 일반화를 최대화하기 위한 전략을 배워보자.

훈련, 검증, 테스트 세트

모델 평가의 핵심은 가용한 데이터를 항상 훈련, 검증, 테스트 3개의 세트로 나누는 것이다.

훈련 세트에서 모델을 훈련하고, 검증 세트에서 모델을 평가한다. 모델을 출시할 준비가 되면 테스트 세트에서 최종적으로 딱 한 번 모델을 테스트한다.

훈련 세트와 테스트 세트 2개를 사용하면 어떤가? 훨씬 간단하지 않나?

그렇게 하지 않는 이유는 모델을 개발할 때 항상 모델의 설정을 튜닝하기 때문이다.

예를 들어 층의 수나, 층의 유닛 수를 선택해야 한다. 검증 세트에서 모델의 성능을 평가하여 이런 튜닝을 수행한다. 본질적으로 이런 튜닝도 어떤 파라미터 공간에서 좋은 설정을 찾는 **학습**이다. 결국 검증 세트의 성능을 기반으로 모델의 성절을 튜닝하면 검증 세트로 모델을 직접 훈련하지 않더라도 빠르고 **검증 세트에 과대적합**될 수 있다.

이 현상의 핵심은 정보 누설 개념에 있다. 검증 세트의 모델 성능에 기반하여 모델의 하이퍼파라미터를 조정할 때마다 검증 데이터에 관한 정보가 모델로 새는 것이다. 하나의 파라미터에 대해서 단 한 번만 튜닝한다면 아주 적은 정보가 누설된다. 이런 검증 세트로는 모델을 평가할 만하다. 하지만 한 번 튜닝하고 나서 검증 세트에 평가한 결과를 가지고 모델을 조정하는 과정을 여러 번 반복하면, 검증 세트에 관한 정보를 모델에 아주 많이 노출시키게 된다.

모델은 간접적으로라도 테스트 세트에 대한 어떤 정보도 얻어서는 안된다. 테스트 세트 성능에 기초하여 튜닝한 모델의 모든 설정은 일반화 성능을 왜곡시킬 것이다.

데이터를 훈련, 검증, 테스트 세트로 나누는 것은 간단해 보일 수 있지만, 데이터가 적을 때는 몇가지 고급 기법을 사용하면 도움이 된다. 대표적인 세 가지 평가 방법인 단순홀드아웃 검증, K-겹 교차 검증, 셔플링을 사용한 반복 K-겹 교차 검증을 살펴보자

단순 홀드아웃 검증

데이터의 일정량을 테스트 세트로 떼어 놓는다. 남은 데이터에서 훈련하고 테스트 세트로 평가한다. 정보 누설을 막기 위해 테스트 세트를 사용하여 모델을 튜닝하면 안된다. 검증 세트도 따로 떼어 놓아야 한다.

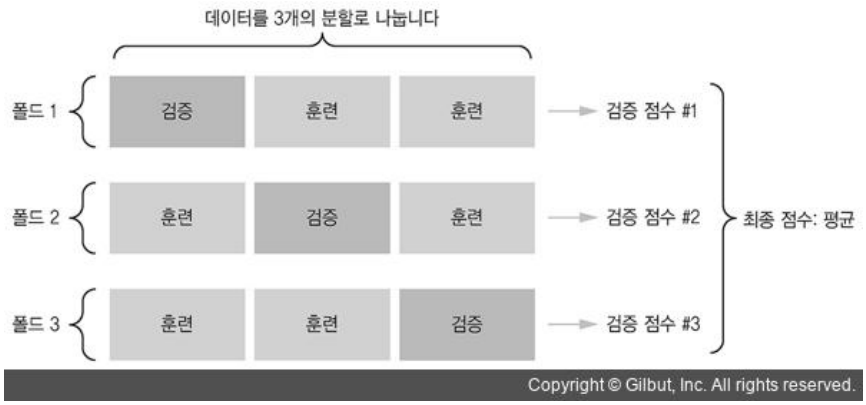
```
num_validation_samples = 10000

np.random.shuffle(data) #데이터를 섞는 것(셔플링)이 일반적으로 좋다.
validation_data = data[:num_validation_samples] #검증 세트를 만든다.
data = data[num_validation_samples:]

training_data = data[:] #훈련 세트를 만든다.

model = get_model()
model.train(training_data)
validation_score = model.evaluate(validation_data)

#여기에서 모델을 튜닝하고,
#다시 훈련하고, 평가하고, 또 다시 튜닝하고,,,

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

이 평가 방법은 단순해서 한 가지 단점이 있다. 데이터가 적을 때 검증 세트와 테스트 세트의 샘플이 너무 적어 주어진 전체 데이터를 통계적으로 대표하지 못할 수 있다.

다음에 나올 K-겹 교차 검증과 반복 K-겹 교차 검증이 이 문제를 해결할 수 있다.

K-겹 교차 검증

이 방식은 데이터를 동일한 크기를 가진 K개 분할로 나눈다. 각 분할 I에 대해 남은 K - 1개의 분할로 모델을 훈련하고 분할 I에서 모델을 평가한다. 최종 점수는 이렇게 얻은 K개의 점수를 평균한다. 이 방법은 모델의 성능이 데이터 분할에 따라 편차가 클 때 도움이 된다. 홀드 아웃 검증처럼 이 방법은 모델의 튜닝에 별개의 검증 세트를 사용하게 된다.




```
k = 4
num_validation_samples = len(data) // k

np.random.shuffle(data)

validation_scores = []
for fold in range(k):
  validation_data = data[num_validation_samples * fold:
  num_validation_samples * (fold + 1)] # 검증 데이터 부분을 선택한다.
  training_data = data[:num_validation_samples * fold] +
  data[num_validation_samples * (fold + 1):]
# 남은 데이터를 훈련 데이터로 사용한다. 리스트에서 + 연산자는 두 리스트를 더하는 것이 아니고 연결한다.

  model = get_model()
  model.train(training_data)
  validation_score = model.evaluate(validation_data)
  validation_scores.append(validation_score)

validation_score = np.average(validation_scores)

model = get_model() #테스트 데이터를 제외한 전체 데이터로 최종 모델을 훈련한다.
model.train(data)
test_score = model.evaluate(test_data)
```

셔플링을 사용한 반복 K-겹 교차 검증

이 방법은 비교적 가용 데이터가 적고 가능한 정확하게 모델을 평가하고자 할 때 사용한다. 캐글 경연에서는 이 방법이 아주 크게 도움이 된다. 이 방법은 K-겹 교차 검증을 여러 번 적용하되 K개의 분할로 나누기 전에 매번 데이터를 무작위로 섞는다. 최종 점수는 모든 K-겹 교차 검증을 실행해서 얻은 점수의 평균이 된다. 결국 P x K개(P는 반복 횟수)의 모델을 훈련하고 평가하므로 비용이 매우 많이 든다.

기억해야 할 것

평가 방식을 선택할 때 다음 사항을 유념해야 한다.

- **대표성 있는 데이터** : 훈련 세트와 테스트 세트가 주어진 데이터에 대한 대표성이 있어야한다. 예를 들어 숫자 이미지를 분류하는 문제에서 샘플 배열이 클래스 순서대로 나열되어 있다고 가정한다. 이 배열의 처음 80%를 훈련 세트로 나머지 20%를 테스트 세트로 만들면 훈련 세트에는 0~7 숫자만 담겨 있고 테스트 세트에는 8~9 숫자만 담기게 된다. 어처구니 없는 실수처럼 보이지만 놀랍게도 자주 일어나는 일이다. 이런 이유 때문에 훈련 세트와 테스트 세트로 나누기 전에 데이터를 무작위로 섞는 것이 일반적이다.
- **시간의 방향** : 과거로부터 미래를 예측하려고 한다면 데이터를 분할하기 전에 무작위로 섞어서는 절대 안된다. 이렇게 하면 미래의 정보가 누설되기 때문이다. 즉 모델이 사실상 미래 데이터에서 훈련될 것이다. 이런 문제에서는 훈련 세트에 있는 데이터보다 테스트 세트에 있는 모든 데이터가 미래의 것이어야 한다.
- **데이터 중복** : 한 데이터셋에 어떤 데이터 포인트가 두 번 등장하면, 데이터를 섞고 훈련 세트와 검증 세트로 나누었을 때 훈련 세트와 검증 세트에 데이터 포인트가 중복될 수 있다. 이로 인해 훈련 데이터의 일부러 테스트하는 최악의 경우가 된다. 훈련 세트와 검증 세트가 중복되지 않는지 확인하자.