

## 과적합 피하기

세즈노프스키 교수가 만들어 고개한 음파 탐지기 데이터 정보를 이용해 광물과 돌을 구분하는 실험을 해보자.

지금이나 쉽게 해결하지 못하는, 하지만 머신러닝을 하는 사람이라면 반드시 극복해야 하는 문제인 '**과적합(overfitting)**'에 대해서도 알아보자.

### 1. 데이터의 확인과 실행

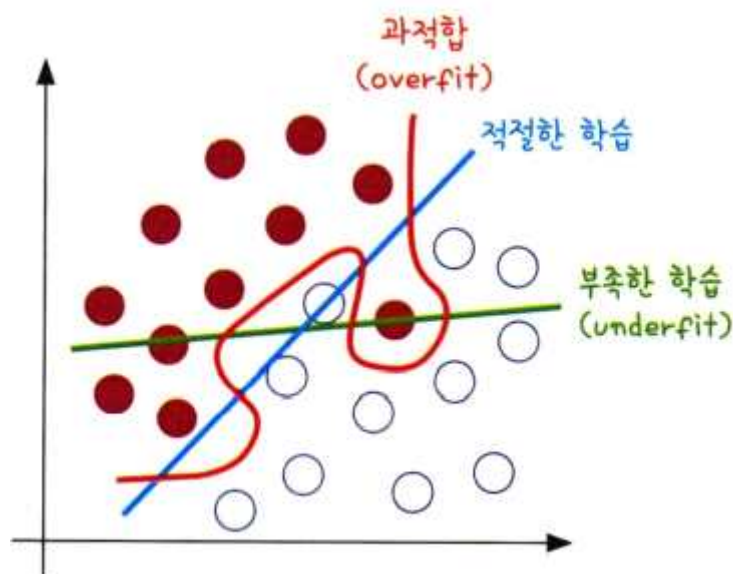
(실습코드는 github 참고)

주어진 데이터(sonar.csv)는 샘플의 수는 208개, 60개의 속성과 1개의 클래스로 이루어져 있다. 맨 마지막 칼럼만 객체형인 것으로 보아 마지막에 나오는 칼럼은 클래스이며 **데이터형 변환**이 필요한 것을 알 수 있다.

학습을 시켜보면 모델의 정확도는 100%로 나온다. 하지만 정말로 100% 정확도의 모델이 만들어진 것일까?

### 2. 과적합 이해하기

**과적합(overfitting)** - 모델이 학습 데이터셋 안에서는 일정 수준 이상의 예측 정확도를 보이지만, 새로운 데이터에 적용하면 잘 맞지 않는 것



**빨간색 선** - 주어진 샘플에 정확히 맞게끔 선이 그어져 있음. 하지만 이 선은 너무 이 경우에 **최적화**되어 있어서 완전히 새로운 데이터에 적용하면 이 선을 통해 **정확히 두 그룹으로 나누지 못하게 된다.**

과적합은 층이 너무 많거나 변수가 복잡해서 발생하기도 하고 테스트셋과 학습셋이 중복될 때 생기기도 함.

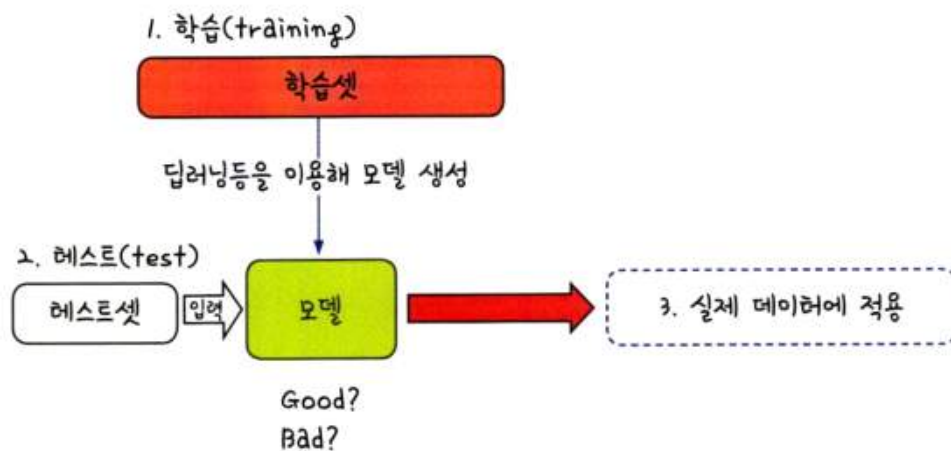
### 3. 학습셋과 테스트셋

(실습코드는 github 참고)

과적합을 방지하는 방법 - 학습을 하는 데이터셋과 이를 테스트할 데이터셋을 완전히 구분한 다음 학습과 동시에 테스트를 병행하며 진행하는 것이 한 방법

ex. 100개의 샘플로 이루어진 데이터셋 -> 70개 샘플은 학습셋, 30개 샘플은 테스트셋으로 사용

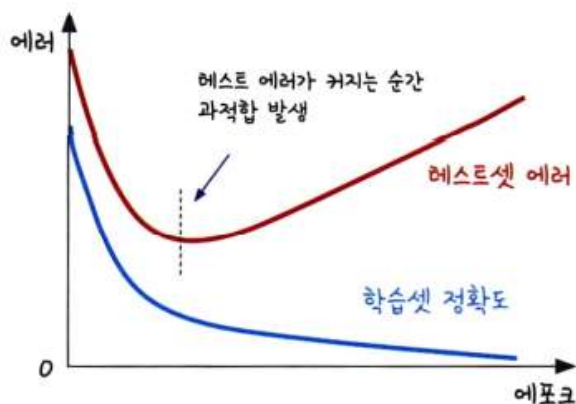
학습을 하여 저장된 결과 파일을 '모델'이라고 부른다. 모델은 다른 셋에 적용할 경우 학습 단계에서 각인되었던 그대로 다시 수행한다. 테스트셋으로 쓸 샘플들로 실험해서 정확도를 살펴보면 학습이 얼마나 잘 되었는지를 알 수 있다!



지금까지 학습 데이터를 이용해 정확도를 측정한 것은 데이터에 들어있는 모든 샘플을 그대로 테스트에 활용한 결과이다. 이러한 방법은 빠른 시간에 모델 성능을 파악하고 수정할 수 있도록 도와 주지만 머신러닝의 최종 목적에 맞지 않음.

머신러닝의 최종 목적 - 과거의 데이터를 토대로 새로운 데이터르 예측하는 것

18



식이 복잡해지고 학습량이 늘어날수록 학습 데이터를 통한 예측률은 계속해서 올라가지만, 테스트셋을 이용한 예측률은 오히려 떨어진다.

sklearn 라이브러리의 `train_test_split()` 함수 - 불러온 X 데이터와 Y 데이터에서 각각 정해진 비율(%)만큼 구분하여 한 그룹은 학습에 사용하고 다른 한 그룹은 테스트에 사용하게 하는 함수

#### 4. 모델 저장과 재사용

학습이 끝난 후 테스트해 본 결과가 만족스러울 때 이를 모델로 저장하여 새로운 데이터에 사용할 수 있다.

(실습코드는 github 참고)

#### 5. k겹 교차 검증

딥러닝 혹은 머신러닝 작업을 할 때 늘 어려운 문제 중 하나는 알고리즘을 충분히 테스트하였어도 데이터가 충분치 않으면 좋은 결과를 내기가 어렵다는 것이다. 이러한 문제를 해결하기 위해 만든 방법이 바로 k겹 교차 검증 (k-fold cross validation)이다.

k겹 교차 검증 - 데이터셋을 여러 개로 나누어 하나씩 테스트셋으로 사용하고 나머지를 모두 합해서 학습셋으로 사용하는 방법. 이렇게 하면 가지고 있는 데이터의 100%를 테스트셋으로 사용할 수 있다.

데이터를 원하는 숫자만큼 쪼개 각각 학습셋과 테스트셋으로 사용되게 만드는 함수는 sklearn의 `StratifiedKFold()` 함수이다.

ex. 5겹 교차 검증(5-fold cross validation)



## 베스트 모델 만들기

비뉴 베르드 지방에서 만들어진 와인을 측정한 데이터를 가지고 레드와인과 화이트와인을 구분하는 실험을 진행해 보자

### 1. 데이터의 확인과 실행

(실습코드는 github 참고)

**sample()** 함수 - 원본 데이터에서 정해진 비율만큼 랜덤으로 뽑아오는 함수

frac = 1 -> 원본 데이터의 100%를 랜덤으로 불러오기

frac = 0.5 -> 원본 데이터의 50%만 랜덤으로 불러오기

주어진 데이터는 총 6497개의 샘플로 이루어져 있으며 13개의 속성을 가지고 있다.

0	주석산 농도	7	밀도
1	아세트산 농도	8	pH
2	구연산 농도	9	황산칼륨 농도
3	잔류 당분 농도	10	알코올 도수
4	염화나트륨 농도	11	와인의 맛(0~10등급)
5	유리 아황산 농도	12	class(1: 레드와인, 0: 화이트와인)
6	총 아황산 농도		

이항 분류 문제이므로 오차 함수는 binary\_crossentropy를 사용. 전체 샘플이 200회 반복되어 입력될 때까지 실험을 반복하고 한 번에 입력되는 입력 값은 200개씩 하고 종합하도록 함.

### 2. 모델 업데이트하기

(실습코드는 github 참고)

이번에는 모델을 그냥 저장하는 것이 아니라 에포크(epoch)마다 모델의 정확도를 함께 기록하면서 저장해보자.

에포크 횟수와 이때의 테스트셋 오차 값을 이용해 파일 이름을 만들어 hdf5라는 확장자로 저장해보자.

모델을 저장하기 위해 케라스의 콜백 함수 중 **ModelCheckpoint()** 함수를 불러오기

그리고 checkpointer라는 변수를 만들어 모니터할 값을 지정

**테스트 오차**는 케라스 내부에서 **val\_loss**로 기록됨(**학습 정확도** - **acc**, **테스트셋 정확도** - **val\_acc**, **학습셋 오차** - **loss**)

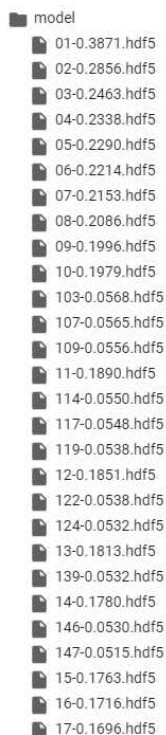
**modelpath** - 모델이 저장될 곳으로 지정

**verbose** - 1이면 해당 함수의 진행 사항을 **출력**, 0이면 **출력하지 않음**

이렇게 하면 모델을 학습할 때마다 정한 checkpoint의 값을 받아 지정된 곳에 모델을 저장한다.

(ModelCheckpoint() 함수에 모델이 앞서 저장된 모델보다 나아졌을 때만 저장하게끔 하려면 **save\_best\_only** 값을 **True**로 지정한다.)

(참고. 콜백 함수 - 다른 함수의 인자로써 넘겨진 후 특정 이벤트에 의해 호출되는 함수)



(이런 식으로 모델이 저장됨)

### 3. 그래프로 확인하기

(실습코드는 github 참고)

모델의 학습 시간에 따른 정확도와 테스트 결과를 그래프를 통해 확인해 보자.

모델이 학습되는 과정을 **history** 변수로 만들어 저장

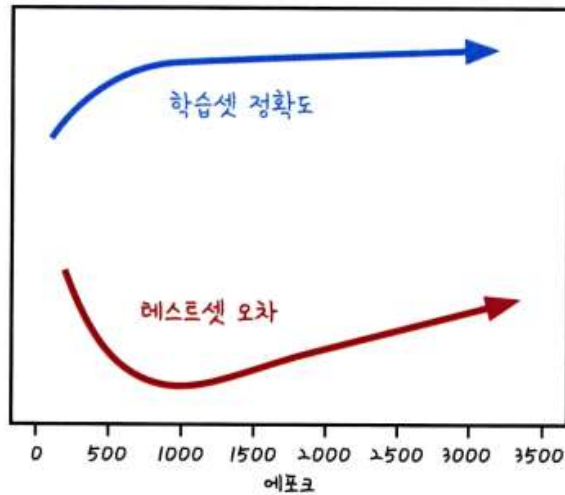
긴 학습의 예를 지켜 보기 위해 에포크를 3500으로 조정하고 시간이 너무 오래 걸리지 않도록 **sample()** 함수를 이용해 전체 샘플 중 15%만 불러오도록 함.

배치 크기는 500으로 늘려 한 번 딥러닝을 가동할 때 더 만히 입력되게끔 하고 불러온 샘플 중 33%는 분리하여 테스트셋으로 사용함

y\_vloss에 테스트셋(33%)으로 실험한 결과의 오차 값을 저장하고, y\_acc에 학습셋(67%)으로 측정한 정확도의 값을 저장한다.

여기서는 앞에서 배운 `train_test_split()` 함수와 다르게 `model.fit` 과정에서 `validation_split` 인자로 학습셋과 테스트셋을 나눔!

(출력된 그래프의 단순화 ver)



앞서 공부한 것처럼 학습셋의 정확도는 시간이 흐를수록 좋아지나 어느정도 시간이 흐르면 더 나아지지 않는다는 것을 볼 수 있음.

#### 4. 학습의 자동 중단

(실습코드는 github 참고)

학습이 진행될수록 학습셋의 정확도는 올라가지만 과적합 때문에 테스트셋의 실험 결과는 점점 나빠지게 된다.

`EarlyStopping()` 함수 - 학습이 진행되어도 테스트셋 오차가 줄지 않으면 학습을 멈추게 하는 함수

(모니터할 값(`monitor`)과 테스트 오차가 좋아지지 않아도 몇 번(`patience`)까지 기다릴지 정해야 함)

참고 자료

<https://aileen93.tistory.com/97>

<https://deep-deep-deep.tistory.com/53>

<https://snowdeer.github.io/machine-learning/2018/01/11/keras-use-history-function/>

<https://vision-ai.tistory.com/entry/%EB%94%A5%EB%9F%AC%EB%8B%9D%EC%9D%98-%EB%AA%A8%EB%8D%B8-%EC%84%B1%EB%8A%A5-%ED%8F%89%EA%B0%80-1-Keras%EC%9D%98-validationssplit-%EC%9D%B4%EC%9A%A9%ED%95%98%EB%8A%94-%EB%B0%A9%EB%B2%95>