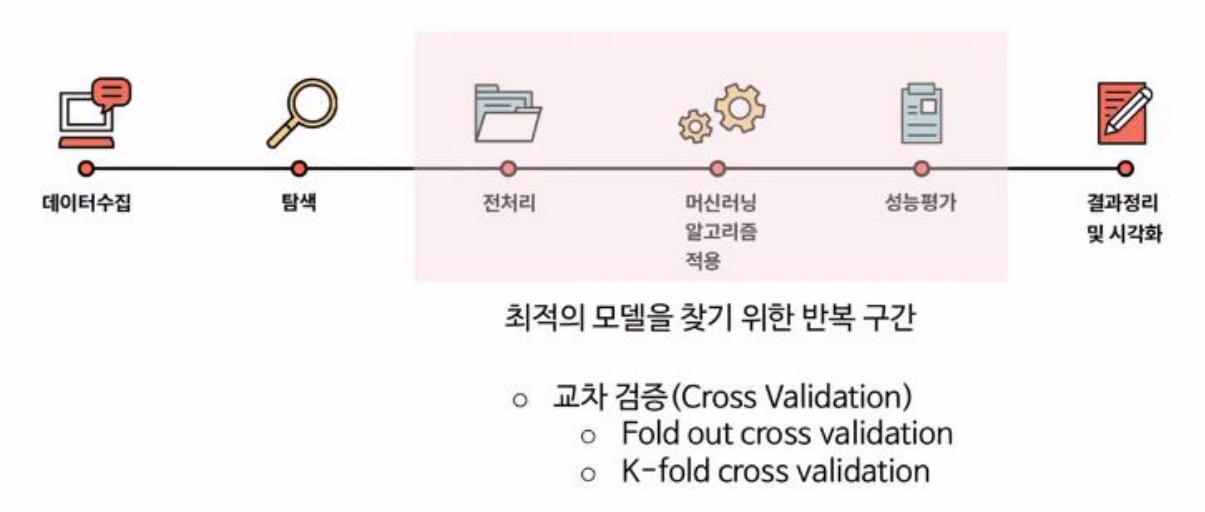
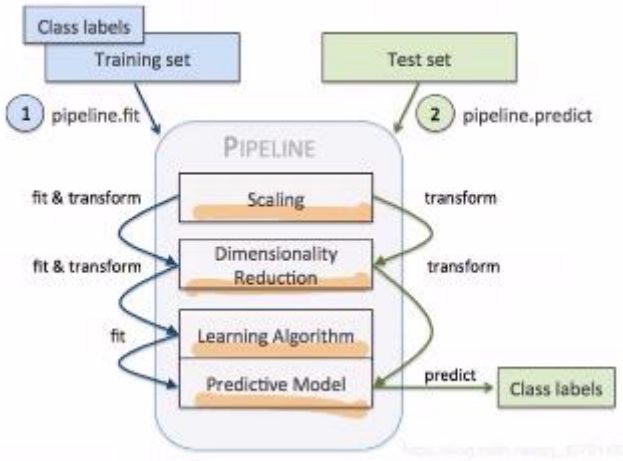


교차 검증에 대해 알아봅시다.

가장 좋은 모델을 찾기 위해서는 **전처리 – 알고리즘 적용 – 평가**의 과정을 반복해야 합니다. 이 과정을 반복하기 위해 **Pipeline** 클래스를 이용할 수 있습니다.

파이프라인 (Pipeline) : scikit-learn의 pipeline 클래스는 학습 과정에 필요한 여러 함수들을 묶어주는 유용한 Wrapper이다.

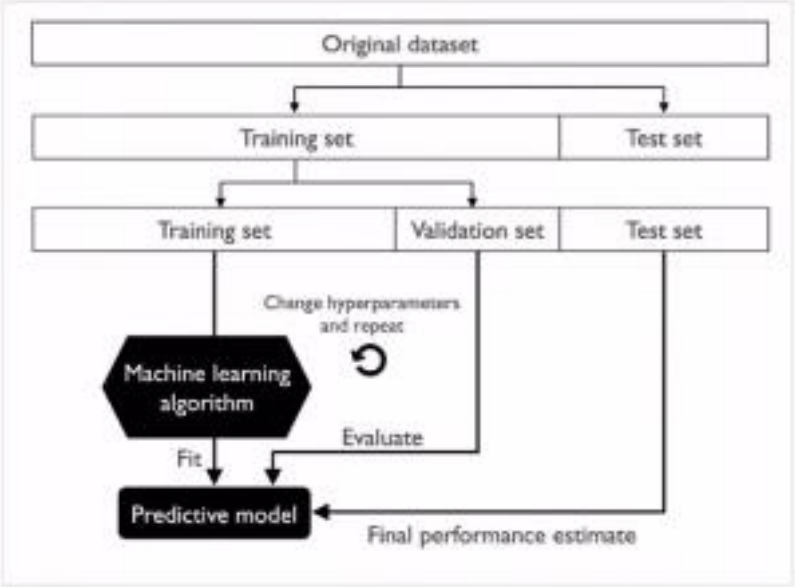
```
pipe_lr = make_pipeline(StandardScaler(),
                        PCA(n_components=2),
                        LogisticRegression(solver='liblinear', random_state=1))

pipe_lr.fit(X_train, y_train)
y_pred = pipe_lr.predict(X_test)
print('테스트 정확도: %.3f' % pipe_lr.score(X_test, y_test))
```



모델 성능 평가

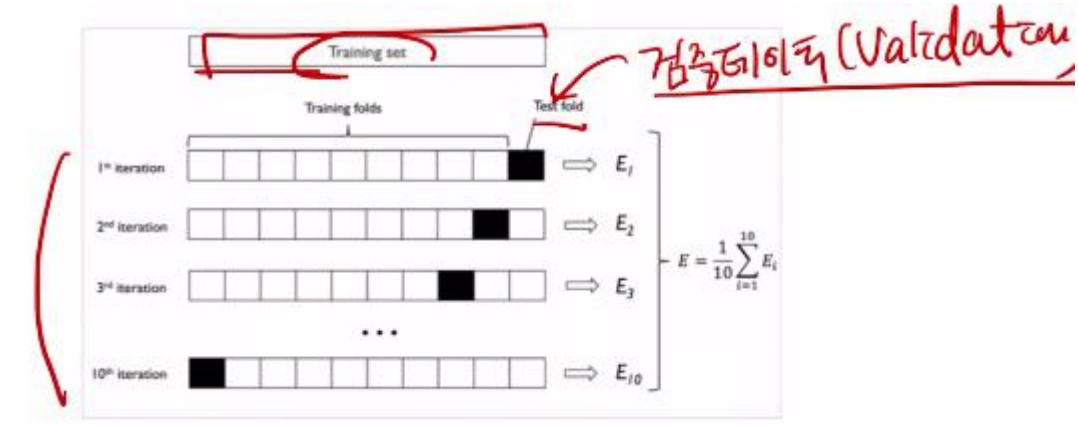
홀드아웃 교차 검증 (Holdout Cross Validation)

- 지금까지 해왔듯이 Test set와 Training set을 나누어서 학습한 뒤, Test set의 결과에 맞춰 Hyperparameter를 튜닝하는 것은 올바른 방법이 아니다.
- 따라서 Training Data의 일부를 Validation set로 떼어내어 튜닝에 이용한다.
- Training set은 모델 학습에, Validation set은 하이퍼 파라미터 튜닝에, Test set은 성능 추정에 사용한다.



K겹 교차 검증 (K fold Cross Validation)

- Training set을 중복 없이 K겹으로 Random하게 나눈다. 그 중 1개의 set으로 Validation하고 나머지 K-1개로 학습한다. K번 반복하여 K개의 서로 다른 set으로 학습된 모델을 얻을 수 있다.
- 각각의 Fold에서 얻은 성능을 기반으로 평균성능을 계산한다. 이 경우 홀드아웃 검증보다 데이터 분할에 덜 예민한 성능 평가 가능.
- K겹 교차 검증은 중복을 허락하지 않기 때문에 모든 샘플이 검증에 딱 한 번씩 사용된다.
- 추천하는 K값은 10이나 데이터가 크면 K를 줄여도 된다.



과적합 문제

과대적합(Overfitting) : 모델이 학습 데이터에 너무 잘 맞지만 일반화Generalization : Test data에 대한 높은 성능을 갖추는 것)이 떨어지는 상황.

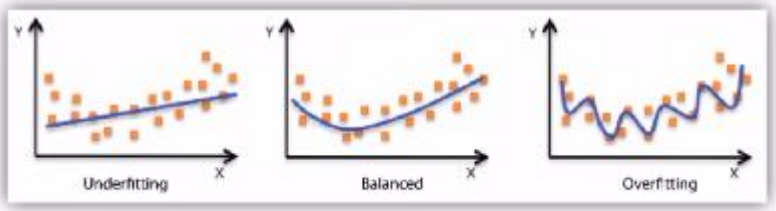
과대적합 해결방법

- 학습 데이터 추가 수집
- 모델 제약 (Regularization) 늘리기 (C값 낮추기)
- 학습 데이터 잡음을 줄임 (오류 수정 및 이상치 제거)

과소적합(Underfitting) : 모델이 너무 단순하여 데이터에 내재된 구조를 학습하지 못하는 현상

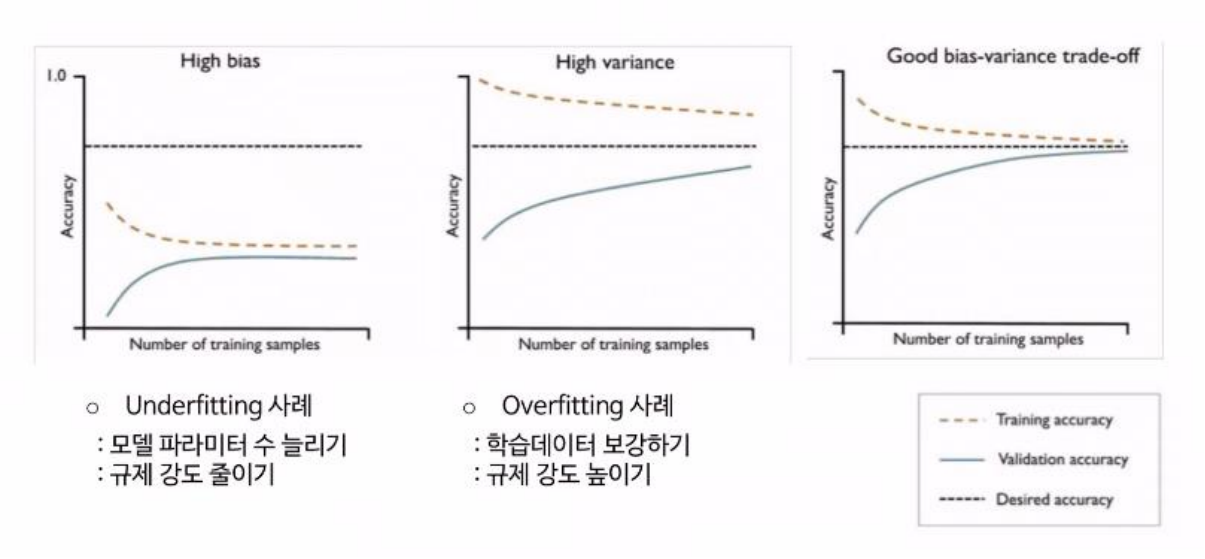
과소적합 해결방법

- 파라미터가 더 많은 복잡한 모델 선택
- 모델의 제약 줄이기 (C값 키우기)
- 과적합 이전까지 충분히 학습하기

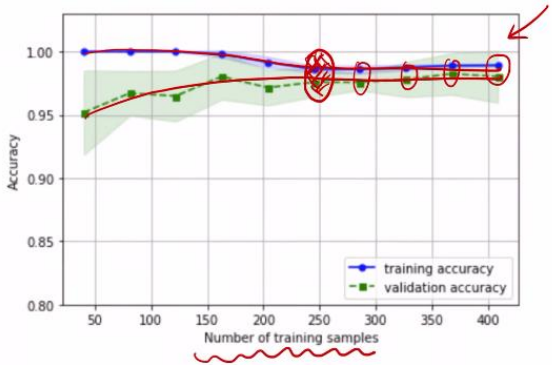


과적합 판단하기

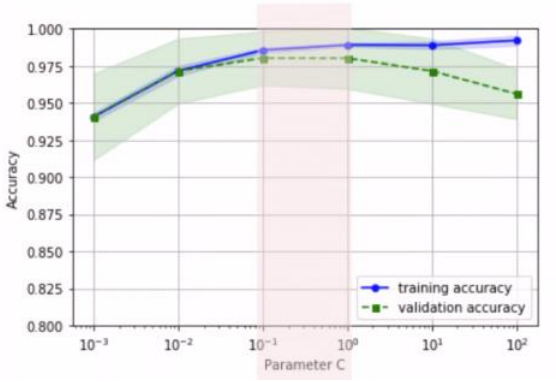
- **학습 곡선(Learning Curve)의 편향과 분산 분석 :** 샘플 데이터의 수에 따른 정확도 변화



▪ 아래의 예) 250개 이상의 샘플을 사용할 때 모델이 잘 작동함



▪ 로지스틱 회귀의 매개변수 C (규제 강도와 반비례)



앙상블 학습

목적 : 여러 분류기를 하나로 연결하여 개별 분류기 보다 더 좋은 일반화 성능을 달성하는 것

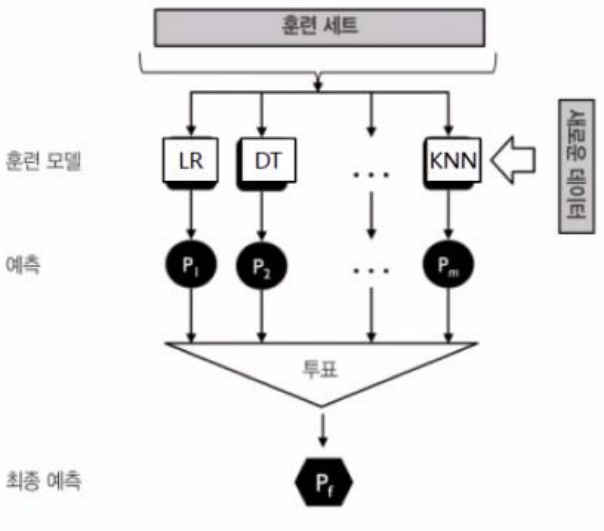
방법 :

- 여러 분류 알고리즘 사용 (Voting)
- 하나의 분류 알고리즘 여러 번 사용 (Bagging, Boosting)

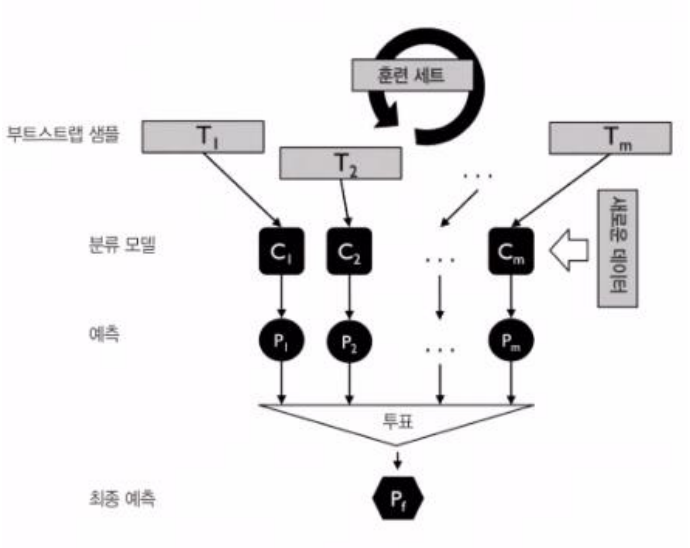
종류 :

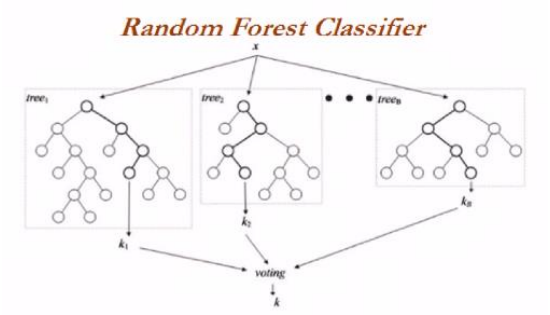
- 다수결 투표 (Majority Voting) : 동일한 학습 데이터 사용
- 배깅 (Bagging) : 알고리즘 수행 마다 서로 다른 학습 데이터 추출하여 사용 ex) Random Forest
- 부스팅 (Boosting) : 샘플 뽑을 때 잘못 분류된 데이터 50%를 재학습에 사용 또는 가중치를 부여해사용

다수결 투표 (Majority Voting) : 동일한 학습 데이터로 모델 구축. 샘플 뽑을 때 중복 없음



배깅 (Bagging) : 알고리즘 마다 별도의 학습 데이터를 추출(Sampling)하여 모델 구축에 사용. 부트스트랩(Bootstrap) -> 학습데이터 샘플링 시 복원 추출(중복) 허용 Ex) Random Forest



- 랜덤 포레스트 (Random Forest)
  - 배깅의 일종
  - 단일 분류 알고리즘 (Decision Tree) 사용
  - Forest 구축 : 무작위로 예측변수 선택하여 모델 구축
  - 결합 방식 : 투표(분류), 평균화(회귀 예측)
  - 

부스팅 (Boosting) : 샘플을 뽑을 때 잘못 분류된 데이터의 50%를 재학습에 사용.

AdaBoost : 전체 학습데이터를 사용하고 잘못 분류된 데이터에 가중치 적용