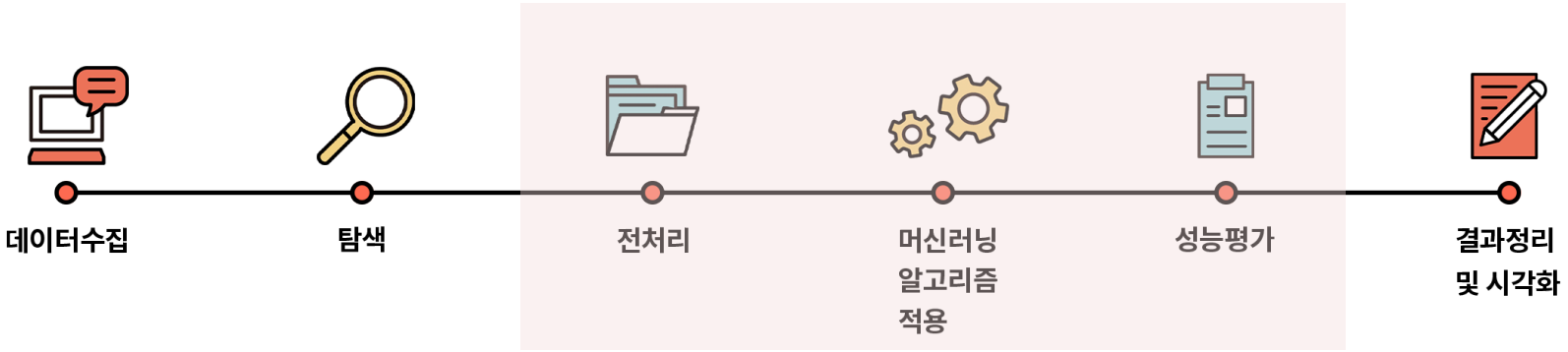
교차 검증

Cross Validation

데이터 분석 과정

- **머신러닝을 이용한 데이터 분석 과정**
  - 모델 최적화: 주어진 데이터 성능 평가 결과 가장 좋은 모델을 찾는 과정



최적의 모델을 찾기 위한 반복 구간

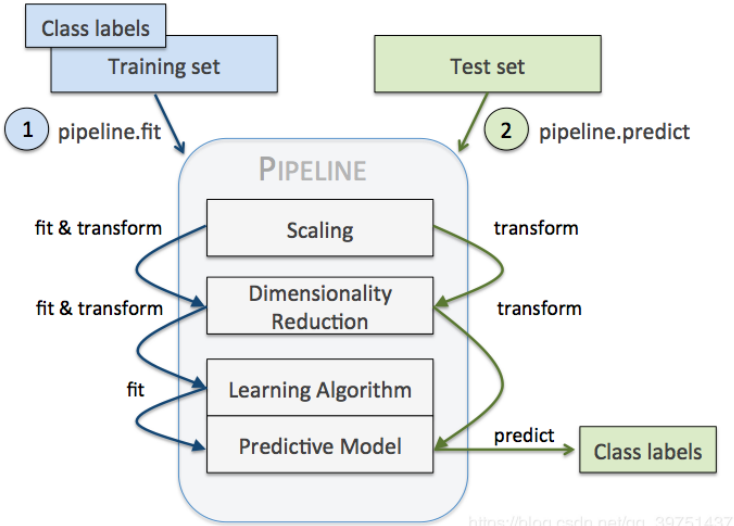
- 교차 검증(Cross Validation)
  - Fold out cross validation
  - K-fold cross validation

파이프라인: 모델 성능 평가 도구

- **파이프 라인(Pipeline)**
  - 사이킷런의 Pipeline 클래스는 연속된 변환을 순차적으로 처리할 수 있는 기능을 제공하는 유용한 래퍼(Wrapper) 도구

```
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))
```

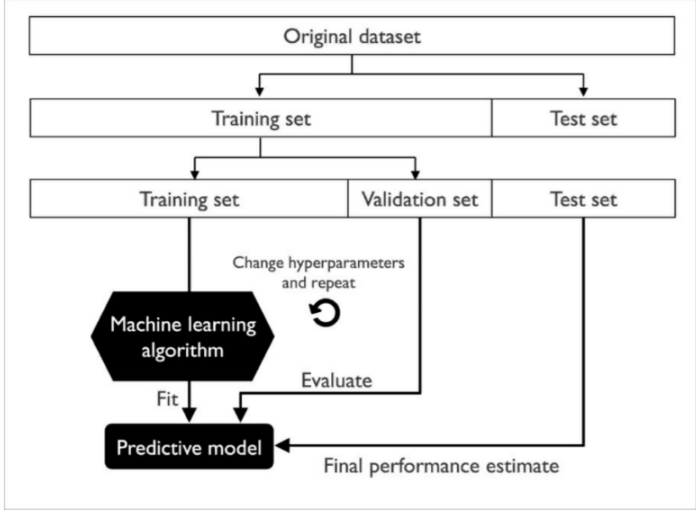


모델 성능 평가

- **교차 검증 (Cross Validation)**
  - **모델 성능 검증하기 위한 방법**
  - 홀드아웃 교차 검증(Holdout Cross Validation)
    - 지금까지 여러분이 실습에 사용한 방법
  - **K-겹 교차 검증(K-fold Cross)**
    - 앞으로 여러분이 실습에 사용하면 좋은 방법

모델 성능 평가

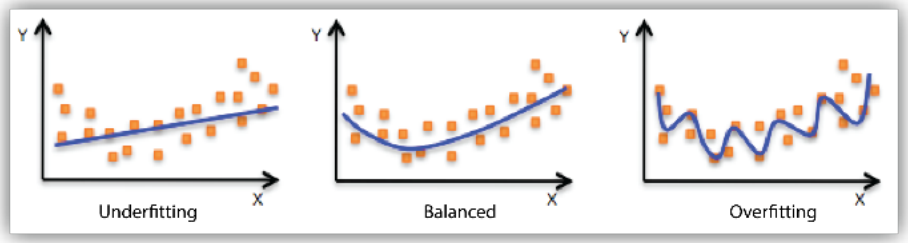
- 홀드아웃 교차 검증(Holdout Cross Validation)
  - 전체 데이터를 학습데이터와 테스트 데이터로 나눔
  - 학습데이터는 모델 학습에, 테스트 데이터는 일반화 성능 추정을 위해 사용
  - 모델 선택을 통한 하이퍼 파리미터 튜닝
    - **테스트 데이터 기반 튜닝을 시도하나 이는 올바른 방법이 아님!!!**
  - 홀드아웃 방법은 전체 데이터를 1)학습 데이터 2)검증 데이터 3)테스트 데이터로 나누고, 학습 데이터는 모델학습에, 검증 데이터는 하이퍼파라미터 튜닝에, 테스트 데이터는 성능 추정에 사용함



모델 성능 평가

- K겹 교차 검증(K fold Cross Validation)
  - K겹 교차 검증에서는 중복없이 훈련 데이터를 K겹으로 랜덤하게 나눔
    - K-1겹으로 모델을 훈련하고, 나머지 하나로 성능을 평가함
    - 즉, K번 반복하므로 K개의 서로 다른 모델을 얻을 수 있음
  - K겹 교차검증은 각각의 폴드에서 얻은 성능을 기반으로 평균 성능을 계산함
    - **이 경우에는 홀드아웃 방법보다 데이터 분할에 덜 예민한 성능 평가 가능**
    - K겹 교차 검증은 중복을 허락하지 않기 때문에 모든 샘플이 검증에 딱 한번 사용됨
  - 추천하는 K 값은 10 임
    - K가 크면 실행 시간이 길어짐, 따라서 큰 데이터는 작은 K 값을 선택해도 됨

모델 성능 최적화



- **과적합 문제**
  - **과대적합(overfitting)이란?**
    - 모델이 학습 데이터에 너무 잘 맞지만 **일반화(generalization)**가 떨어지는 상황

일반화(generalization)란?
테스트 데이터에 대한 높은 성능을 갖추는 것

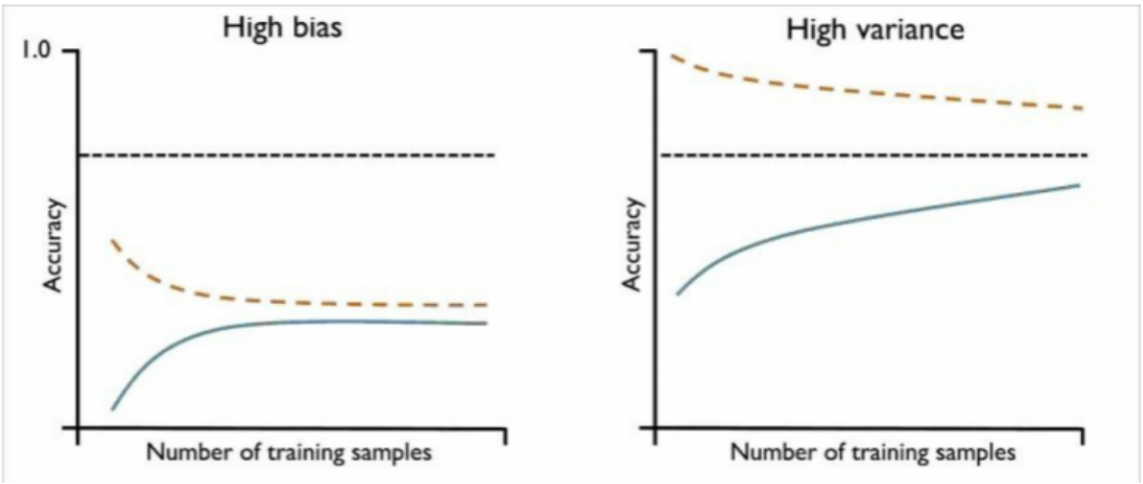
  - 과대적합 해결방법
    - 학습 데이터 추가 수집
    - 모델 제약 늘리기: 규제(regularization) 값 늘리기
    - 학습 데이터 잡음을 줄임 (오류 수정 및 이상치 제거)

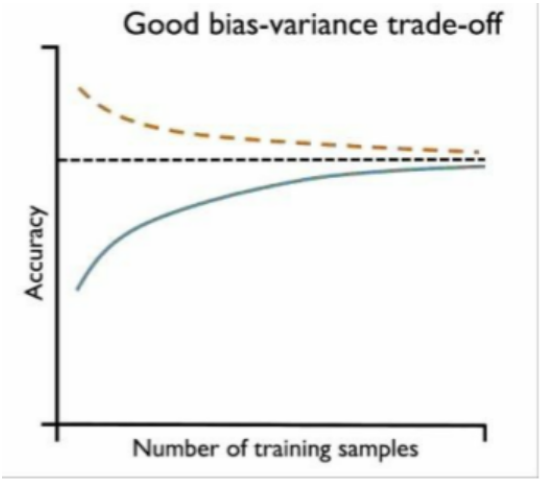
class `sklearn.linear_model.` **`LogisticRegression`**(*penalty='l2', *, dual=False, tol=0.0001, C=1.0, fit_intercept=True, intercept_scaling=1, class_weight=None, random_state=None, solver='lbfgs', max_iter=100, multi_class='auto', verbose=0, warm_start=False, n_jobs=None, l1_ratio=None*) [source]

  - **과소적합(underfitting)이란?**
    - 모델이 너무 단순하여 데이터에 내재된 구조를 학습하지 못하는 현상
  - 과소적합 해결방법
    - 파라미터가 더 많은 모델 선택
    - 모델의 제약 줄이기: 규제(regularization) 값 줄이기
    - 과적합 이전까지 충분히 학습하기

모델 최적화

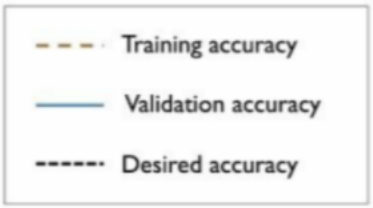
- **과대적합/과소적합 판단하기**
  - **학습 곡선(Learning Curve)의 편향과 분산 분석**
    - 샘플 데이터의 수에 따른 정확도 변화





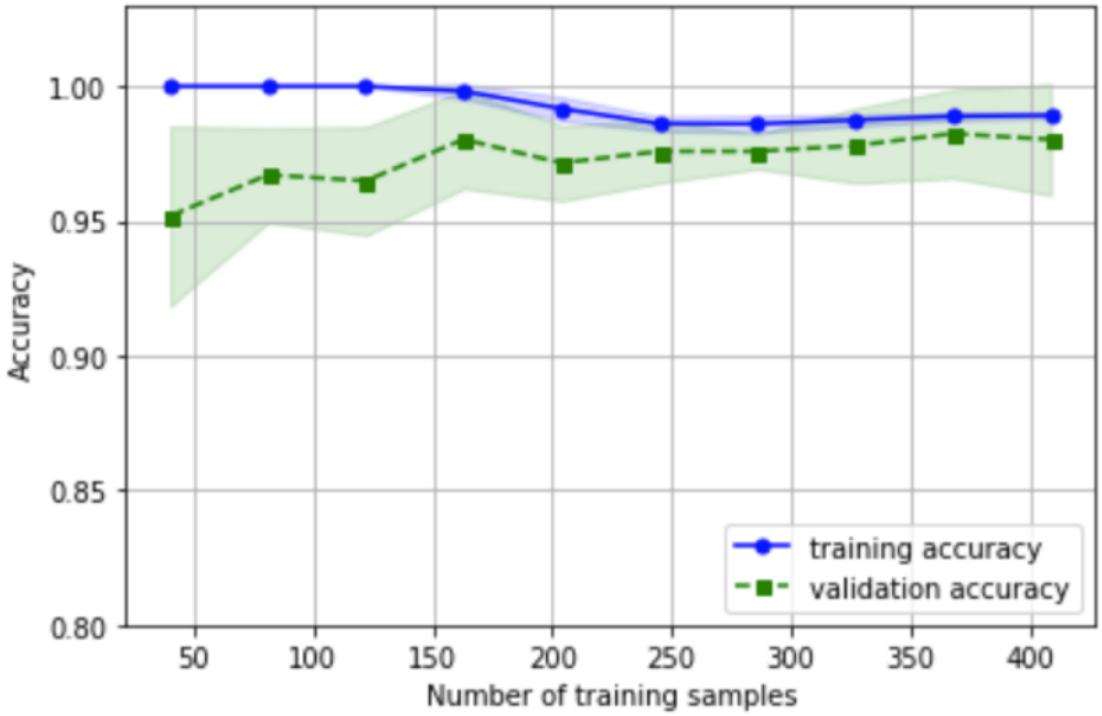
- Underfitting 사례
  : 모델 파라미터 수 늘리기
  : 규제 강도 줄이기

- Overfitting 사례
  : 학습데이터 보강하기
  : 규제 강도 높이기



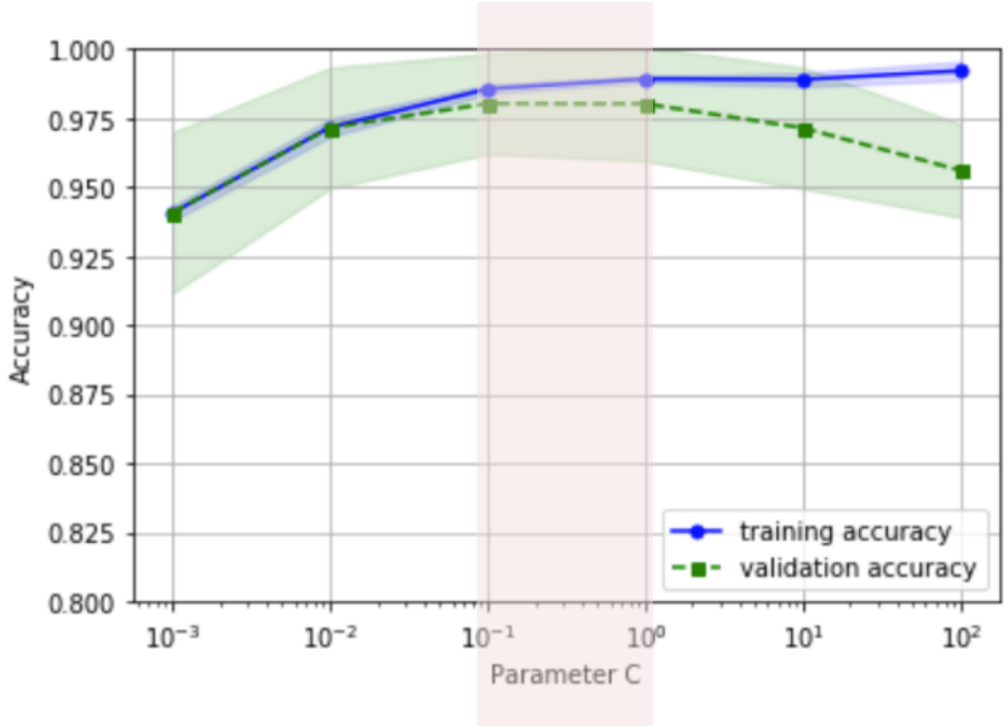
모델 최적화

- **과대적합/과소적합 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



모델 최적화

- 과대적합/과소적합 판단하기
  - 검증 곡선(Validation Curve)
    - 매개변수에 따른 정확도 변화
      - 로지스틱 회귀의 매개변수 C (규제 강도와 반비례)



앙상블

Ensemble Model

앙상블 학습

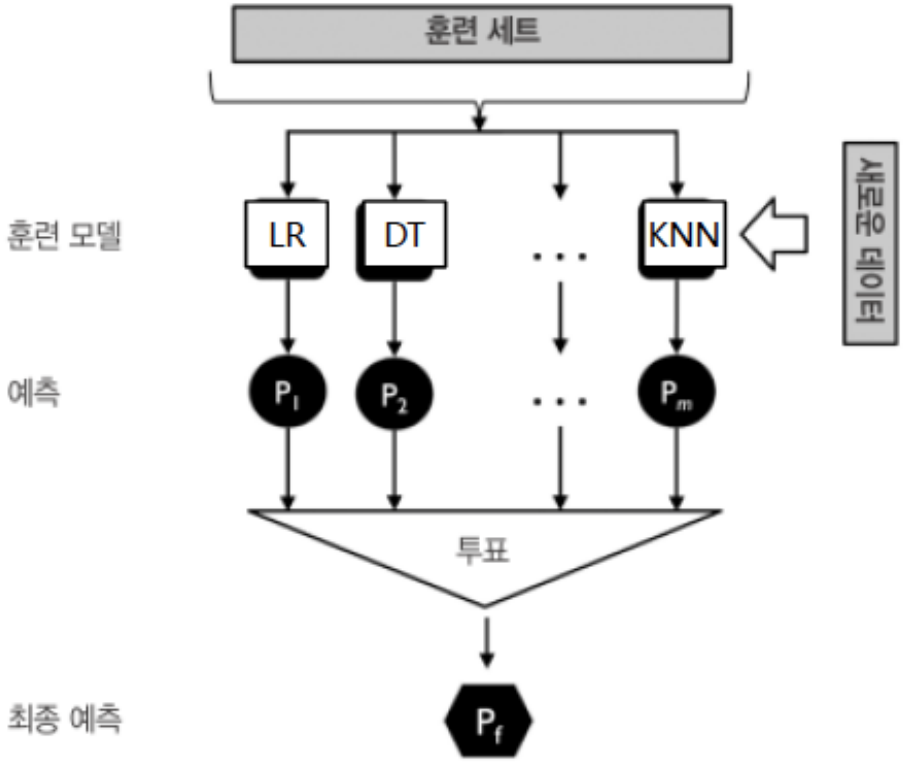
- **목적**
  - 여러 분류기를 하나로 연결하여 개별 분류기 보다 더 좋은 일반화 성능을 달성하는 것

- **방법**
  - 여러 분류 알고리즘 사용: 다수결 투표(Voting)
  - 하나의 분류 알고리즘을 여러 번 사용: 배깅(Bagging), 부스팅(Boosting)

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

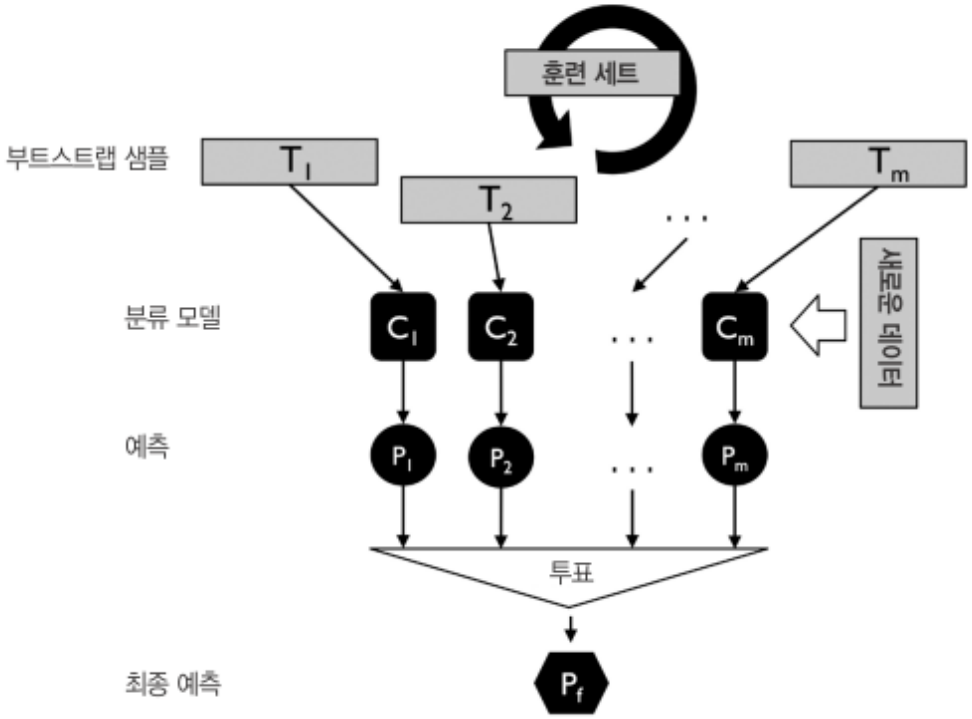
다수결 투표 (Majority Voting)

- **다수결 투표**
  - 동일한 학습데이터로 모델 구축
  - 샘플 뽑을 때 중복 없음



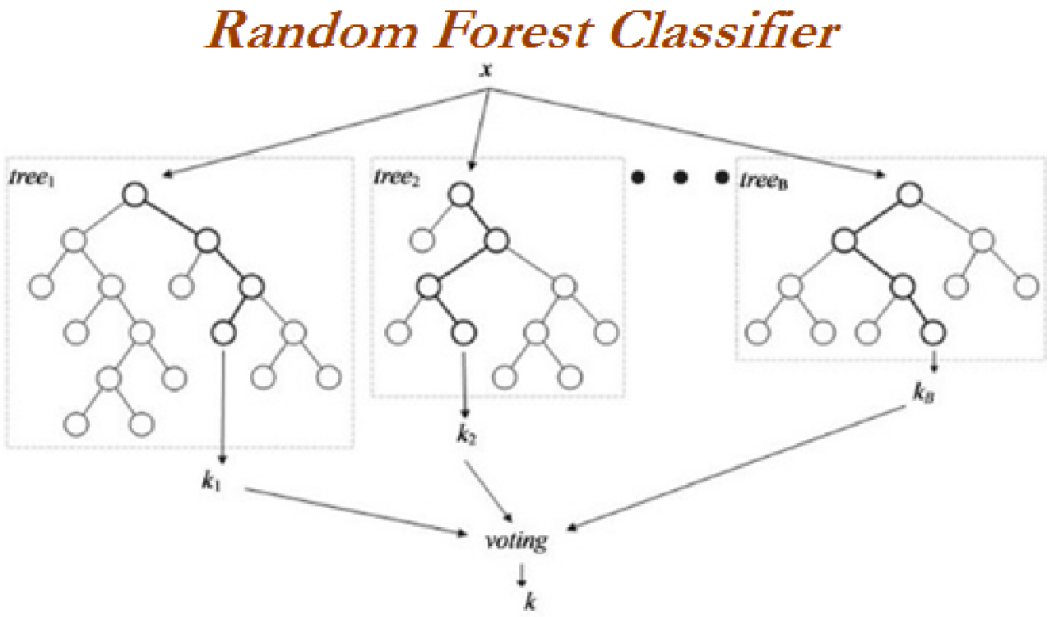
배깅(Bagging)

- 배깅
  - 알고리즘마다 별도의 학습 데이터를 추출(샘플링)하여 모델 구축에 사용
  - 부트스트랩(Bootstrap) 사용
    - 학습데이터 샘플링 시 복원 추출(중복)을 허용



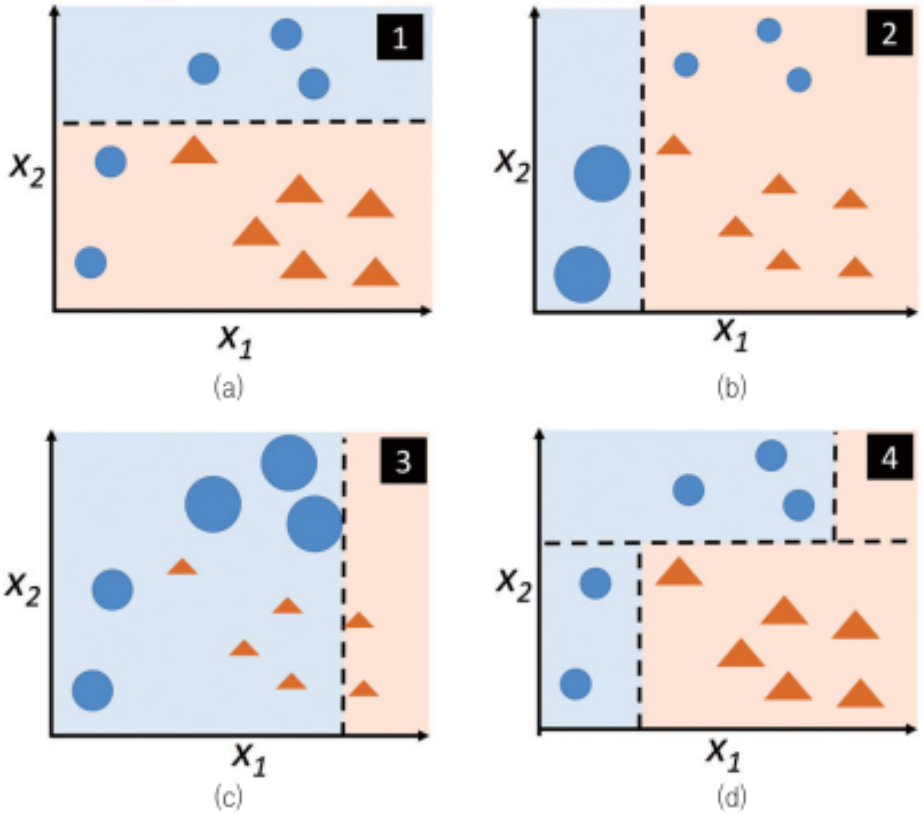
배깅(Bagging)

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



부스팅(Boosting)

- **부스팅**
  - 샘플 뽑을 때 잘못 분류된 데이터의 50%를 재 학습에 사용
  - AdaBoost: 전체 학습데이터를 사용하고 잘못 분류된 데이터에 가중치 적용



앙상블 실습

Majority Voting / Bagging / Boosting

데이터 설명

- 데이터셋: 유니버셜 은행
- 개인대출 제안에 대한 수락 여부
- 총 데이터: 5000개 (학습: 3000개, 테스트 2000개)
- 성공율: 9.6% (480명)

나이, 경력, 소득, 가족 수, 신용카드 월평균 사용 액, 교육, 담보 부채권, **개인대출**, 증권계좌, CD계좌, 온라인 뱅킹, 신용카드

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
	25	1	49	4	1.6	1	0	0	1	0	0	0
	45	19	34	3	1.5	1	0	0	1	0	0	0
	39	15	11	1	1.0	1	0	0	0	0	0	0
	35	9	100	1	2.7	2	0	0	0	0	0	0
	35	8	45	4	1.0	2	0	0	0	0	0	1

위의 특성 변수를 이용하여 개인대출 가능 여부를 예측하는 분류기를 설계하는 문제

투표 방식(Voting) 실습 #1 (링크)

- 데이터 로더

```
1 import pandas as pd
2
3 bank_df = pd.read_csv('UniversalBank.csv')
4 bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- 특성 변수 선택

```
1 X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
2 y = bank_df['PersonalLoan']
```

- 데이터 분할

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

투표 방식(Voting) 실습 #1 (분류)

- 앙상블로 사용할 개별 모델 정의

```
from sklearn.tree import DecisionTreeClassifier # 결정 트리
from sklearn.neighbors import KNeighborsClassifier # K-최근접 이웃
from sklearn.linear_model import LogisticRegression # 로지스틱 회귀 모델


logistic = LogisticRegression(solver='liblinear',
                              penalty='l2',
                              C=0.001,
                              random_state=1)

tree = DecisionTreeClassifier(max_depth=None,
                              criterion='entropy',
                              random_state=1)

knn = KNeighborsClassifier(n_neighbors=1,
                           p=2,
                           metric='minkowski')

```

- 앙상블-Voting 정의

```
from sklearn.ensemble import VotingClassifier # 과반수 투표(Majority Voting)
voting_estimators = [('logistic', logistic), ('tree', tree), ('knn', knn)]
voting = VotingClassifier(estimators = voting_estimators,
                          voting='soft')
```

투표 방식(Voting) 실습 #1 (분류)

- K-fold 교차 검증

```
[ ]  1 from sklearn.model_selection import cross_val_score # 교차타당도 # 추가
     2
     3 clf_labels = ['Logistic regression', 'Decision tree', 'KNN', 'Majority voting']
     4 all_clf = [logistic, tree, knn, voting]
     5
     6 for clf, label in zip(all_clf, clf_labels):
     7     scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
     8     print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9276195033668649, 0.01984630447185705, 'Logistic regression')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9499227861055811, 0.032735680131811405, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.7120816883018249, 0.04722587272864442, 'KNN')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9724206139059355, 0.01593603549077189, 'Majority voting')
```

투표 방식(Voting) 실습 #1 (분류)

- GridSearch방식을 이용한 모델 최적화

```
]  1 from sklearn.model_selection import GridSearchCV # 하이퍼파라미터 튜닝
   2
   3 params = {'logistic__C': [0.001, 0.1, 100.0],
   4           'tree__max_depth': [1, 3, 5],
   5           'knn__n_neighbors': [1, 3, 5]}
   6
   7 grid = GridSearchCV(estimator=voting,
   8                      param_grid=params,
   9                      cv=10,
  10                      scoring='roc_auc',
  11                      iid=False)
  12 grid.fit(X_train, y_train)
  13
  14 for r, _ in enumerate(grid.cv_results_['mean_test_score']):
  15     print("%0.3f +/- %0.3f %r"
  16           % (grid.cv_results_['mean_test_score'][r],
  17              grid.cv_results_['std_test_score'][r] / 2.0,
  18              grid.cv_results_['params'][r]))
  19
  20 print('최적의 파타미터: %s' % grid.best_params_)
  21 print('ACU: %.3f' % grid.best_score_)
```

Scoring	Function
Classification	
'accuracy'	metrics.accuracy_score
'balanced_accuracy'	metrics.balanced_accuracy_score
'average_precision'	metrics.average_precision_score
'neg_brier_score'	metrics.brier_score_loss
'f1'	metrics.f1_score
'f1_micro'	metrics.f1_score
'f1_macro'	metrics.f1_score
'f1_weighted'	metrics.f1_score
'f1_samples'	metrics.f1_score
'neg_log_loss'	metrics.log_loss
'precision' etc.	metrics.precision_score
'recall' etc.	metrics.recall_score
'jaccard' etc.	metrics.jaccard_score
'roc_auc'	metrics.roc_auc_score
'roc_auc_ovr'	metrics.roc_auc_score
'roc_auc_ovo'	metrics.roc_auc_score
'roc_auc_ovr_weighted'	metrics.roc_auc_score
'roc_auc_ovo_weighted'	metrics.roc_auc_score
Clustering	
'adjusted_mutual_info_score'	metrics.adjusted_mutual_info_score
'adjusted_rand_score'	metrics.adjusted_rand_score
'completeness_score'	metrics.completeness_score
'fowlkes_mallows_score'	metrics.fowlkes_mallows_score
'homogeneity_score'	metrics.homogeneity_score
'mutual_info_score'	metrics.mutual_info_score
'normalized_mutual_info_score'	metrics.normalized_mutual_info_score
'v_measure_score'	metrics.v_measure_score
Regression	
'explained_variance'	metrics.explained_variance_score
'max_error'	metrics.max_error
'neg_mean_absolute_error'	metrics.mean_absolute_error
'neg_mean_squared_error'	metrics.mean_squared_error
'neg_root_mean_squared_error'	metrics.mean_squared_error
'neg_mean_squared_log_error'	metrics.mean_squared_log_error
'neg_median_absolute_error'	metrics.median_absolute_error
'r2'	metrics.r2_score
'neg_mean_poisson_deviance'	metrics.mean_poisson_deviance
'neg_mean_gamma_deviance'	metrics.mean_gamma_deviance

```
최적의 파타미터: {'knn__n_neighbors': 3, 'logistic__C': 100.0, 'tree__max_depth': 5}
ACU: 0.986
```

배깅 방식(Bagging) 실습 #2 (링크)

- 데이터 로더

```
1 import pandas as pd
2
3 bank_df = pd.read_csv('UniversalBank.csv')
4 bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- 특성 변수 선택

```
1 X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
2 y = bank_df['PersonalLoan']
```

- 데이터 분할

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

배깅 방식(Bagging) 실습 #2 (분류)

- 앙상블로 사용할 개별 모델 정의

```
[ ]  1 from sklearn.tree import DecisionTreeClassifier # 결정 트리
     2
     3 tree = DecisionTreeClassifier(max_depth=None,
     4                                criterion='entropy',
     5                                random_state=1)
     6
```

- 앙상블-Bagging 정의

```
[ ]  1 from sklearn.ensemble import BaggingClassifier # 배깅(Bagging)
     2
     3 bagging = BaggingClassifier(base_estimator=tree, # 수정
     4                             n_estimators=500,
     5                             max_samples=1.0,
     6                             max_features=1.0,
     7                             bootstrap=True,
     8                             bootstrap_features=False,
     9                             n_jobs=1,
    10                             random_state=1)
```

배깅 방식(Bagging) 실습 #2 (분류)

- K-fold 교차 검증

```
[ ]  1 from sklearn.model_selection import cross_val_score # 교차타당도 # 추가
     2
     3 clf_labels = ['Decision tree', 'Bagging']
     4 all_clf = [tree, bagging]
     5
     6 for clf, label in zip(all_clf, clf_labels):
     7     scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
     8     print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9499227861055811, 0.032735680131811405, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9976668161065998, 0.001775473982951, 'Bagging')
```

부스팅 방식(Boosting) 실습 #3 (링크)

- 데이터 로더

```
1 import pandas as pd
2
3 bank_df = pd.read_csv('UniversalBank.csv')
4 bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- 특성 변수 선택

```
1 X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
2 y = bank_df['PersonalLoan']
```

- 데이터 분할

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

부스팅 방식(Boosting) 실습 #3 (분류)

- 앙상블로 사용할 개별 모델 정의

```
1 from sklearn.tree import DecisionTreeClassifier # 결정 트리
2
3 # 배깅과 차이점: mex_depth =1 로 변경
4 tree = DecisionTreeClassifier(max_depth=1,
5                                criterion='entropy',
6                                random_state=1)
7
8
```

- 앙상블-Boosting 정의

```
[ ]  1 from sklearn.ensemble import AdaBoostClassifier # 부스팅(Boosting)
     2
     3 adaboost = AdaBoostClassifier(base_estimator=tree, # 수정
     4                               n_estimators=500,
     5                               learning_rate = 0.1, # 수정
     6                               random_state=1)
     7
```

부스팅 방식(Boosting) 실습 #3 (분류)

- K-fold 교차 검증

```
[ ]  1 from sklearn.model_selection import cross_val_score # 교차타당도 # 추가
     2
     3 clf_labels = ['Decision tree', 'Ada boost']
     4 all_clf = [tree, adaboost]
     5 for clf, label in zip(all_clf, clf_labels):
     6     scores = cross_val_score(estimator=clf,X=X_train,y=y_train,cv=10,scoring='roc_auc')
     7     print("ROC AUC: %0.3f (+/- %0.3f) [%s]", (scores.mean(), scores.std(), label))
```

```
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.8829135713666967, 0.023406169666276122, 'Decision tree')
ROC AUC: %0.3f (+/- %0.3f) [%s] (0.9835566978095359, 0.010774714837632118, 'Ada boost')
```

파이프라인

pipeline

데이터 설명

- 데이터셋: 유니버셜 은행
- 개인대출 제안에 대한 수락 여부
- 총 데이터: 5000개 (학습: 3000개, 테스트 2000개)
- 성공율: 9.6% (480명)

나이, 경력, 소득, 가족 수, 신용카드 월평균 사용 액, 교육, 담보 부채권, **개인대출**, 증권계좌, CD계좌, 온라인 뱅킹, 신용카드

	Age	Experience	Income	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
	25	1	49	4	1.6	1	0	0	1	0	0	0
	45	19	34	3	1.5	1	0	0	1	0	0	0
	39	15	11	1	1.0	1	0	0	0	0	0	0
	35	9	100	1	2.7	2	0	0	0	0	0	0
	35	8	45	4	1.0	2	0	0	0	0	0	1

위의 특성 변수를 이용하여 개인대출 가능 여부를 예측하는 분류기를 설계하는 문제

파이프라인 (링크)

- 데이터 로더

```
1 import pandas as pd
2
3 bank_df = pd.read_csv('UniversalBank.csv')
4 bank_df.head()
```

	ID	Age	Experience	Income	ZIPCode	Family	CCAvg	Education	Mortgage	PersonalLoan	SecuritiesAccount	CDAccount	Online	CreditCard
0	1	25	1	49	91107	4	1.6	1	0	0	1	0	0	0
1	2	45	19	34	90089	3	1.5	1	0	0	1	0	0	0
2	3	39	15	11	94720	1	1.0	1	0	0	0	0	0	0
3	4	35	9	100	94112	1	2.7	2	0	0	0	0	0	0
4	5	35	8	45	91330	4	1.0	2	0	0	0	0	0	1

- 특성 변수 선택

```
1 X = bank_df.drop (['ID','ZIPCode','PersonalLoan'], axis=1)
2 y = bank_df['PersonalLoan']
```

- 데이터 분할

```
1 from sklearn.model_selection import train_test_split
2 X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=1, stratify=y)
```

파이프라인

- 개별 모델 정의

```
from sklearn.tree import DecisionTreeClassifier # 결정 트리

tree = DecisionTreeClassifier(max_depth=None, criterion='gini',random_state=1)
tree.fit(X_train, y_train)

```

- 개별 모델 검정

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, roc_auc_score # 정

y_pred = tree.predict(X_test)
print('잘못 분류된 샘플 개수: %d' % (y_test != y_pred).sum())
print('정확도: %.3f' % accuracy_score(y_test, y_pred))
print('정밀도: %.3f' % precision_score(y_true=y_test, y_pred=y_pred))
print('재현율: %.3f' % recall_score(y_true=y_test, y_pred=y_pred))
print('F1: %.3f' % f1_score(y_true=y_test, y_pred=y_pred))

```

파이프라인

- 파이프라인 모델 정의

```
from sklearn.preprocessing import StandardScaler
from sklearn.decomposition import PCA
from sklearn.model_selection import GridSearchCV

#pipe_tree = make_pipeline( StandardScaler(), PCA(n_components=10), DecisionTreeClassifier())  # 98.514
pipe_tree = make_pipeline(DecisionTreeClassifier())

```

- 파이프라인 모델 학습

```

param_range1 = [1,2,3,4,5,6,7,8,9,10] # 수정
param_range2 = [10,20,30,40,50] # 수정

param_grid = [{'decisiontreeclassifier__max_depth': param_range1, # 수정
               'decisiontreeclassifier__min_samples_leaf': param_range2}] # 수정

gs = GridSearchCV(estimator=pipe_tree, # 수정
                  param_grid=param_grid,
                  scoring='accuracy',
                  cv=10,
                  n_jobs=-1)

gs = gs.fit(X_train, y_train)

print(gs.best_score_)
print(gs.best_params_)

```

파이프라인

- 파이프라인 모델 검정

```
1 from sklearn.metrics import confusion_matrix, classification_report
2
3 best_tree = gs.best_estimator_
4 best_tree.fit(X_train, y_train)
5 y_pred = best_tree.predict(X_test)
6
7 print('Classification Report')
8 print(classification_report(y_test, y_pred))
```