AI 보안 기술개발 교육

머신러닝(Machine Learning) 모델 3

KISA 한국인터넷진흥원

AI 보안 기술개발 교육

머신러닝 모델 3

1. 앙상블 학습 개요
2. Bagging
3. Boosting
4. Stacking

KISA 한국인터넷진흥원

앙상블 학습 (Ensemble Learning)

▪ 앙상블(Ensemble)

- 여러 개의 의사결정 트리(Decision Tree)를 결합하여 하나의 의사결정 트리를 사용하는 것 보다 더 좋은 성능을 내도록 하는 머신러닝 기법

- 앙상블 학습의 핵심은 여러 개의 약 분류기 (Weak Classifier)를 결합하여 강 분류기(Strong Classifier) 를 생성

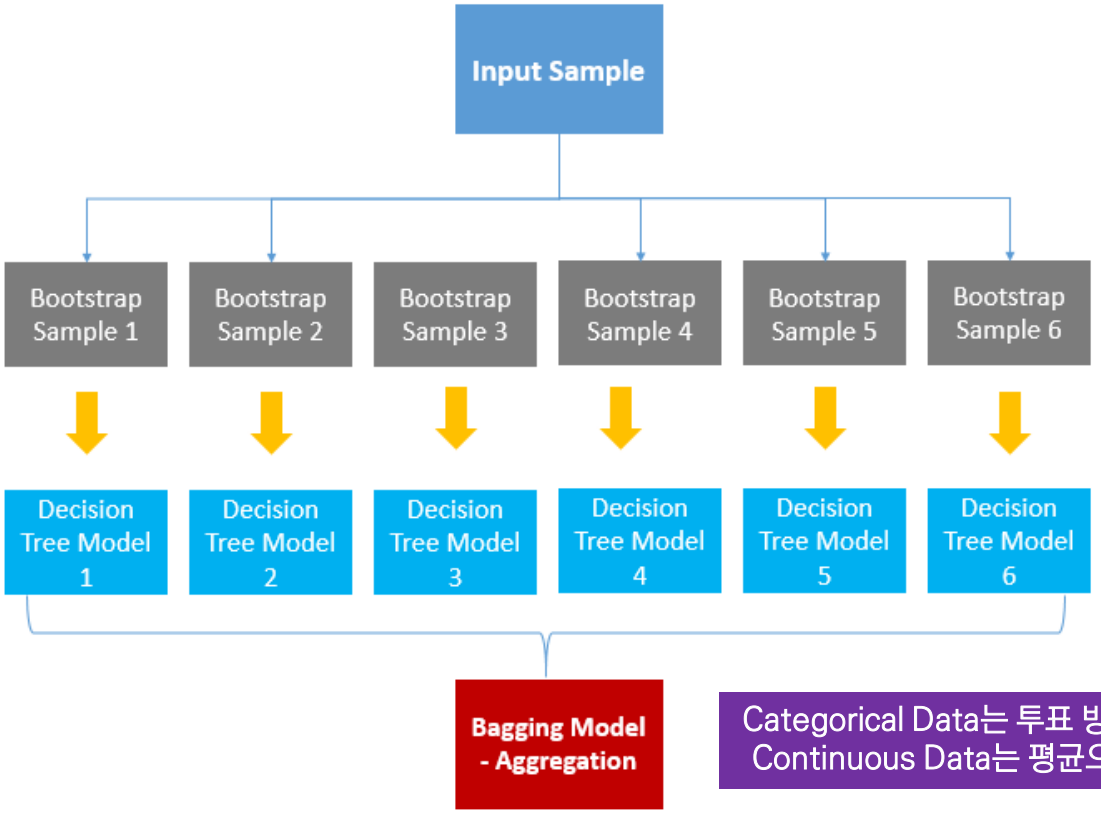
- 앙상블 학습 기법에는 배깅(Bagging), 부스팅(Boosting) 및 스태킹(Stacking)

앙상블 학습 (Ensemble Learning)

- 배깅(Bagging)

  Bagging은 Bootstrap Aggregation의 약자로 데이터로부터 부트스트랩(복원 랜덤 샘플링) 을 수행

  부트스트랩한 데이터로 모델을 학습, 학습된 모델의 결과를 집계하여 최종 결과 값 도출

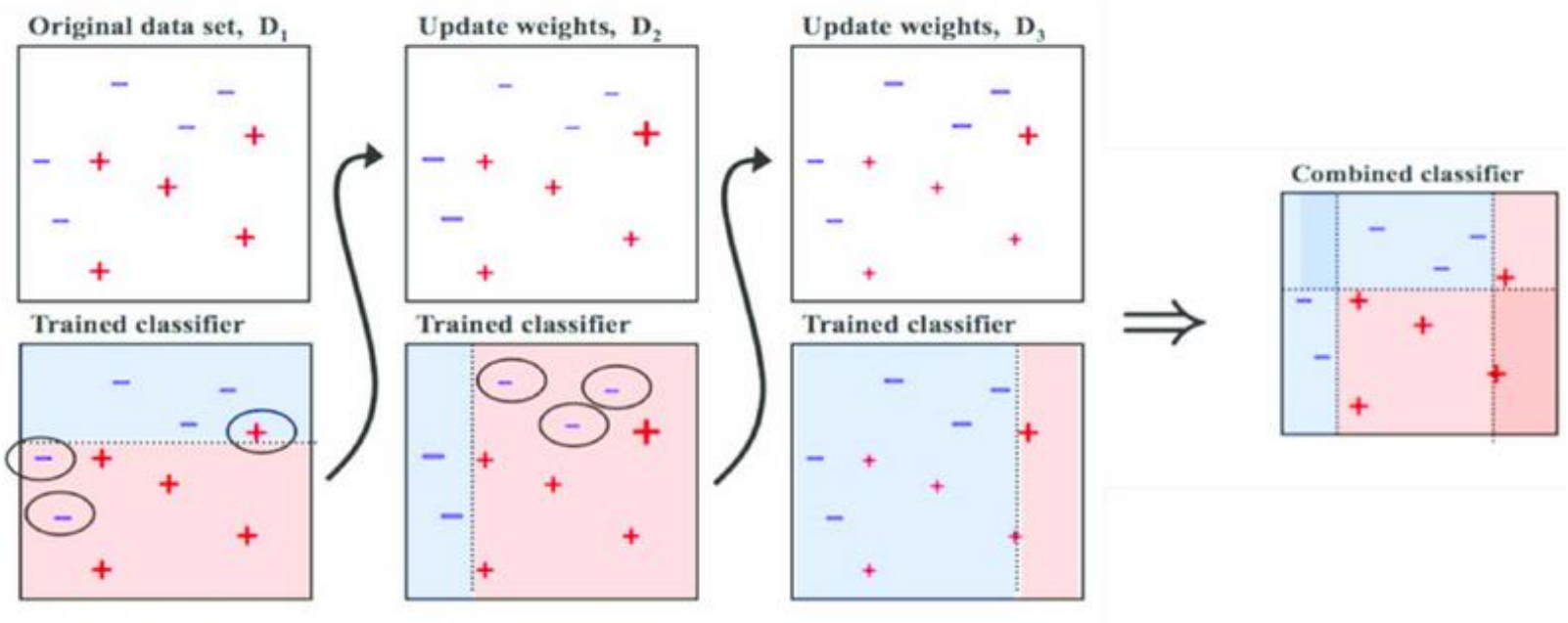


Categorical Data는 투표 방식(Votinig)으로 결과를 집계
Continuous Data는 평균으로 집계

출처: swallow.github.io

앙상블 학습 (Ensemble Learning)

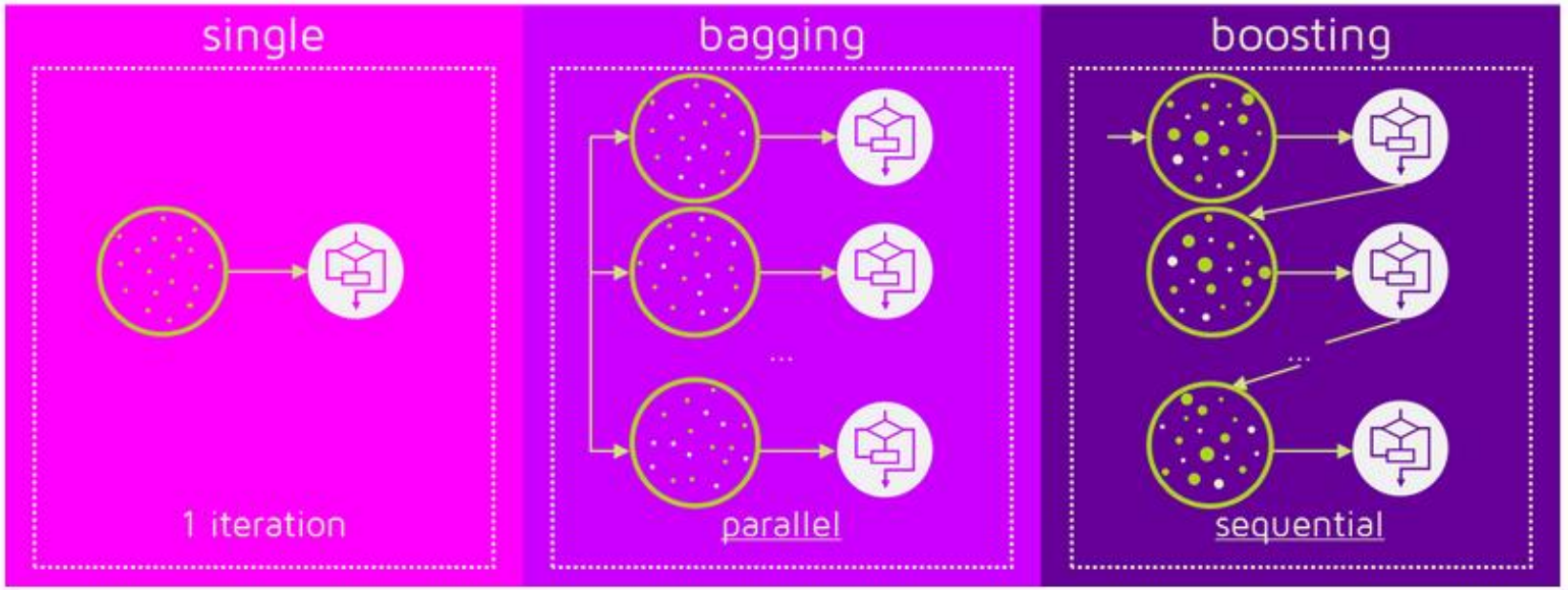
- 부스팅(Boosting)
  - 부스팅은 가중치를 활용하여 weak classifier로 strong classifier를 생성하는 과정
  - 오분류 데이터에 높은 가중치 부여, 정분류 데이터는 낮은 가중치 부여
  - 선행 모델의 오분류 데이터에 가중치를 부여하여 후행 모델에 반영



출처: swallow.github.io

앙상블 학습 (Ensemble Learning)

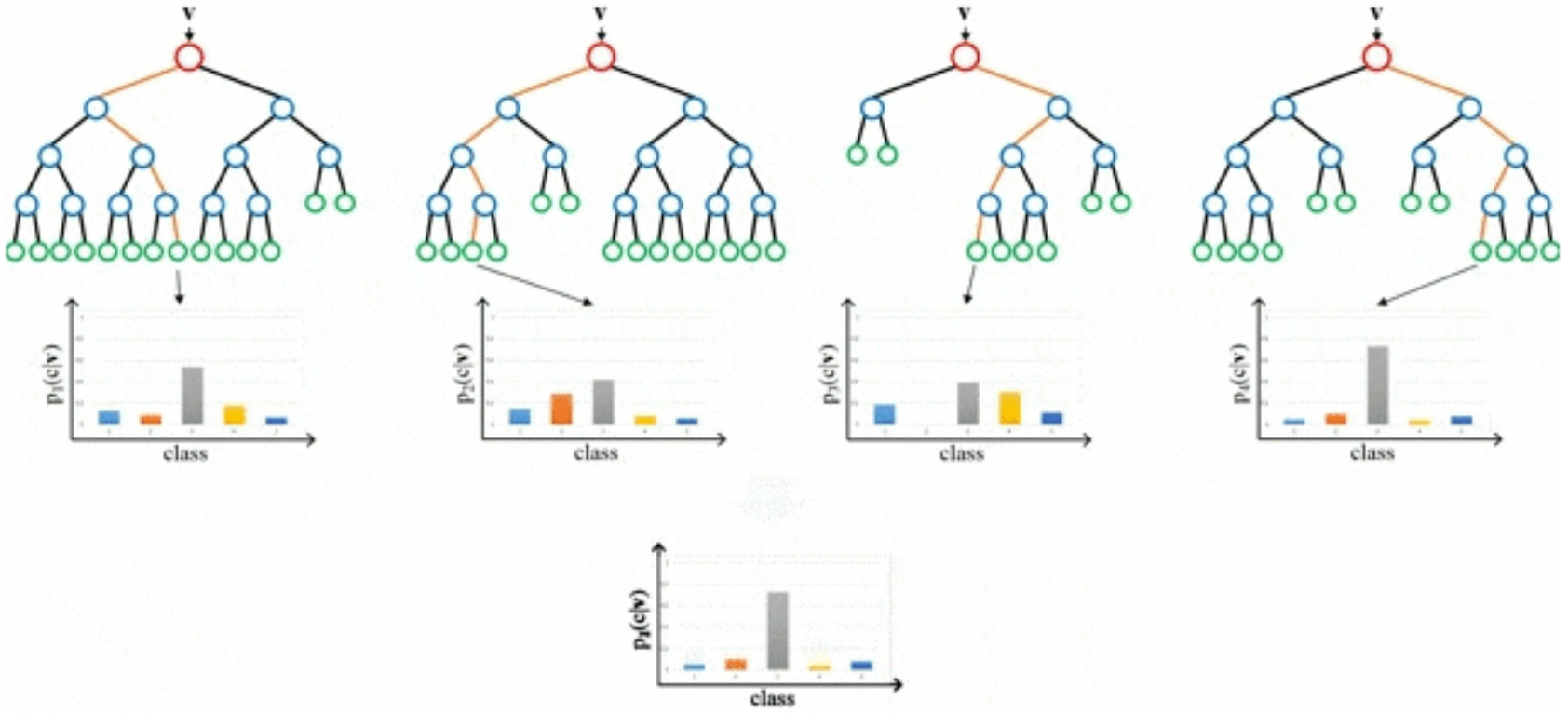
- 배깅(Bagging)과 부스팅(Boosting)
  - 배깅은 병렬로 학습하는 반면, 부스팅은 순차적으로 학습
  - 부스팅은 배깅에 비해 error가 적고 성능이 좋으나 속도가 느리고 오버피팅 가능성 높음
  - 개별 결정 트리의 낮은 성능이 문제라면 부스팅이 적합하고, 오버 피팅이 문제라면 배깅이 적합



출처: swallow.github.io

랜덤 포레스트(Random Forest)

- 랜덤 포레스트는 특징 기반 배깅 기법을 적용한 의사결정 트리의 앙상블
- 트리 배깅은 의사결정 트리 모델의 단점 중 하나인 고분산을 줄여 단일 트리보다 고성능
- 랜덤 포레스트의 무작위적 특성은 트리를 보다 다양하게 생성하고, 분산을 축소



출처 : https://m.blog.naver.com/samsjang/220979751089

랜덤 포레스트(Random Forest)

▪ 랜덤 포레스트 성능 개선을 위한 하이퍼 파라미터

- max_features: 최적의 분할 지점을 찾기 위해 검토할 특징의 개수로
  일반적으로 n차원의 데이터 세트의 $\sqrt{n}$ 의 반올림 값을 설정('auto')

- n_estimators : 트리의 개수가 많을수록 성능이 더 좋지만 계산 시간 장시간 소요
  일반적으로 100, 200, 500을 설정

- min_sample_splits : 노드에서 추가 분할을 위해 필요한 샘플의 최소 개수
  숫자가 너무 작으면 오버피팅, 너무 크면 언더피팅 발생가능
  일반적으로 10, 30, 50으로 설정

Adaboost

- Adaboost(Adaptive Boosting)

1996년에 Freund와 Schapire이 제안한 알고리즘으로 2003년 괴델상 수상

다른 학습 알고리즘(weak learner)의 결과물들에 가중치를 두어 더하는 방법으로 최종 모델 생성

여러 알고리즘을 사용하여 일반화 성능이 좋음

매우 간단한 구현으로 가능하여 효율성이 좋음

속도나 성능적인 측면에서 Desision Tree를 weak learner로 사용

Adaboost

- AdaBoost의 학습 절차

① 각 weak 모델에서 학습할 데이터 선택

② 모든 데이터의 가중치 초기화

③ 학습진행 후, error 계산, 모델 별 가중치 계산, 데이터 가중치를 갱신

④ 오류가 0이 되거나, weak learner 수의 max까지 반복

⑤ 마지막으로 각 분류기의 가중치를 고려한 선형 summation으로 예측값 계산

noise나 outlier에 민감한 모델

Adaboost

- AdaBoostClassifier

```
from sklearn.ensemble import AdaBoostClassifier

model = AdaBoostClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
```

- AdaBoostRegressor

```
from sklearn.ensemble import AdaBoostRegressor
model = AdaBoostRegressor()
fitting_model(model)
```

Hyper parameter인 learning_rate와 n_estimator는 trade off 관계이 적절한 조절 필요

Gradient Boost

- GBM(Gradient Boosting Machine)

  Friedman이 2001년에 소개한 Boosting 알고리즘입

  MART(Multiple Additive Regression Trees), GBRT (Gradient Boosted Regression Trees)

  AdaBoost와 같은 순서로 진행합

  모델의 가중치를 계산하는 방식에서 Gradient Descent를 이용하여 파라미터 계산

  모델에 따른 최적의 가중치로 Gradient decent 계산, 최적화된 결과 도출

  (AdaBoost에서는 모델 가중치를 고려한 linear summation 으로 prediction)

GBM은 greedy algorithm으로 과적합(overfitting) 가능성 높음

옵션 파라미터에 제약을 두어 모델의 일반화 성능을 향상 필요

Gradient Boost

▪ GBM(Gradient Boosting Machine) 옵션 파라미터

Tree 관련 옵션 파라미터

파라미터	설명
max_depth	트리의 최대 깊이, tree가 깊어질 수록 모델은 복잡, 과적합 가능성
max_leaf_nodes	노드의 최대 개수, decision tree, 전체 노드들의 수를 한정
min_samples_split	leaf의 분할 가능 최소 개수, 분할 기준을 제한하여 모델의 복잡도 감소
min_sample_leaf	leaf 노드가 되기 위한 최소 개수, min_samples_split와 조정하여 사용
min_weight_fraction_leaf	min_sample_leaf와 유사, 개수가 아닌 비율로 조정하며 동시 사용은 불가
max_features	노드 분할 기준이 되는 feature의 개수, feature수의 sqrt 정도 사용 권고

Boosting 관련 옵션 파라미터

파라미터	설명
n_estimators	tree의 갯수, tree가 많으면 학습 속도, 최소 갯수 elbow chart
learning_rate	학습률 조정 hypter parameter
subsample	개별 tree가 사용할 데이터 subset, tree간 상관도 축소, 랜덤하게 데이를 선택

Gradient Boost

- GradientBoostringClassifier

```
from sklearn.ensemble import GradientBoostingClassifier

model = GradientBoostingClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
```

- GradientBoostringRegressor

```
from sklearn.ensemble import GradientBoostingRegressor
model = GradientBoostingRegressor()
fitting_model(model)
```

XGBoost

▪ XGBoost(eXtream GBM)

2014년, Tianqi Chen이 GBM을 기반의 알고리즘

2016년에 Kaggle에서 우승하며 python 버전 및 논문 발표

기존 GBM에서 시스템 최적화와 알고리즘 성능 개선

- 시스템 최적화

  병렬화(Parallerlization), 가지치기(Tree Prunning), 하드웨어 최적화(Hardware Optimization)

- 알고리즘 성능 향상

  Ridge나 Lasso를 regularization

  missing value를 모두 제외하고 분할한 후 loss가 제일 적은 곳에 할당

  교차 검증(built-in)을 통한 일반화

XGBoost

- XGBoostClassifier

```
# !pip install xgboost
```

```
from xgboost import XGBClassifier

model = XGBClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
```

- XGBoostRegressor

```
from xgboost import XGBRegressor
model = XGBRegressor()
fitting_model(model)
```

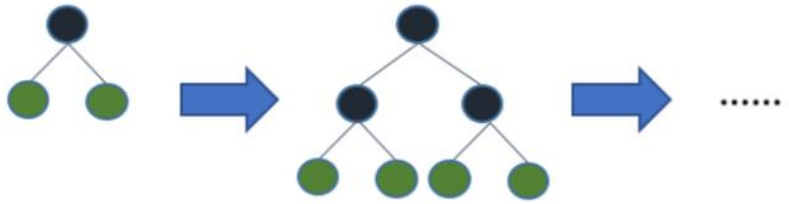
LightGBM

▪ LightGBM

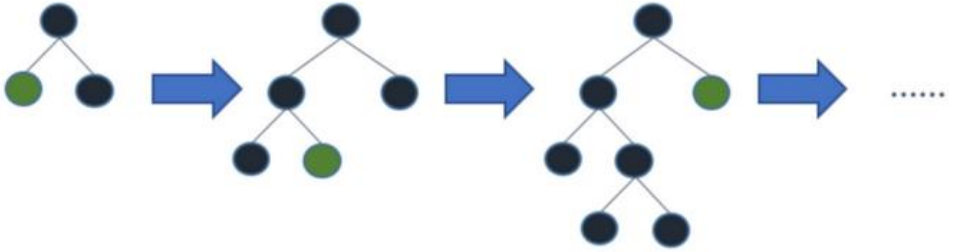
2016년 Microsoft에서 발표한 XGBoost의 뒤를 이은 GBM 기반의 모델

Kaggle에서 여러 차례 우승한 알고리즘

기존 GBM을 비롯한 booting 알고리즘은 level-wise tree growth



LightGBM은 leaf-wise tree growth



LightGBM

- LightGBM

  LightGBM은 히스토그램 기반 알고리즘을 사용

  (분할을 위해 모든 데이터를 확인하지 않고, 히스토그램으로 근사치를 측정)

  local voting과 global voting을 병렬화에서 소모되는 communication 연산 축소, 속도 개선

  XGBoost와 같이 Missing Value 무시

메모리 사용량 감소 : 기존 GBM보다 적은 메모리를 사용합니다.

leaf-wise 방식을 사용하여 복잡한 모델을 만들고, 정확도 개선

과적합될 가능성이 있으므로 max_depth로 조정

XGBoost와 비교하여 속도는 빠르고 성능은 유사,

GPU 사용 가능

LightGBM

- LGBMClassifier

```
from lightgbm import LGBMClassifier

model = LGBMClassifier()
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
```

- LGBMRegressor

```
from lightgbm import LGBMRegressor
model = LGBMRegressor()
fitting_model(model)
```

CatBoost

- CatBoost(Categorical Boost)

2017년에 발표된 Categorical 데이터에 특화된 알고리즘

Categorical Feature가 많은 데이터 셋에서 성능 우수

Categorical Feature 를 자동으로 전처리

missing data는 처리해 주지 않음

Numeric Feature가 많다면 LightGBM보다 느림

다른 GBM에 비해 비교적 overfitting이 적음

CatBoost

- CatBoostClassifier

```
from catboost import CatBoostClassifier

model = CatBoostClassifier(verbose=0, n_estimators=100)
model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
```

- CatBoostRegressor

```
from catboost import CatBoostRegressor
model = CatBoostRegressor()
fitting_model(model)
```

ExtraTrees

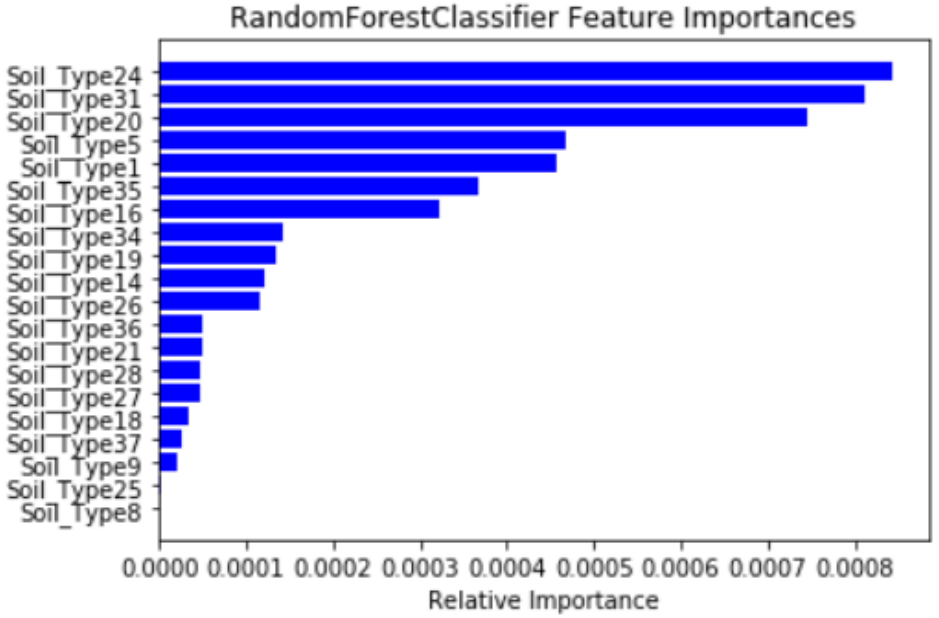
- ExtraTrees(Extremely Randomized Trees)

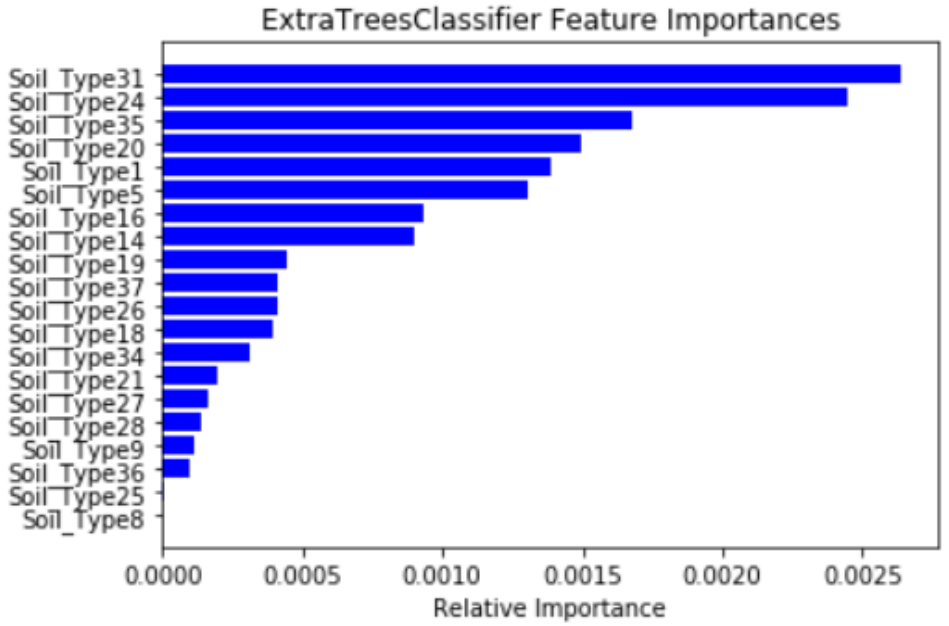
  랜덤 서브셋을 사용하여 다수의 트리를 만들고 노드를 분할한다는 측면에서 RandomForest와 동일

  부트스트랩(랜덤 복원 샘플링)을 사용하지 않고 비복원 샘플링

  샘플링 수도 랜덤화하여 훈련 데이터 셋을 구성

  노드의 분할은 모든 노드에서 랜덤하게 선택된 feature들의 부분집합에서 랜덤하게 분할한다.





ExtraTrees

- ExtraTreesClassifier

```
from sklearn.ensemble import ExtraTreesClassifier

model = ExtraTreesClassifier()

model.fit(x_train, y_train)
y_pred = model.predict(x_test)
print('Accuracy: {:.3f}'.format(accuracy_score(y_test, y_pred)))
print(classification_report(y_test, y_pred, target_names = target_names))
importance_plot('ExtraTreesClassifier', model.feature_importances_)
```

- ExtraTreesClassifier

```
from sklearn.ensemble import ExtraTreesRegressor

model = ExtraTreesRegressor()
fitting_model(model)
```

Stacking

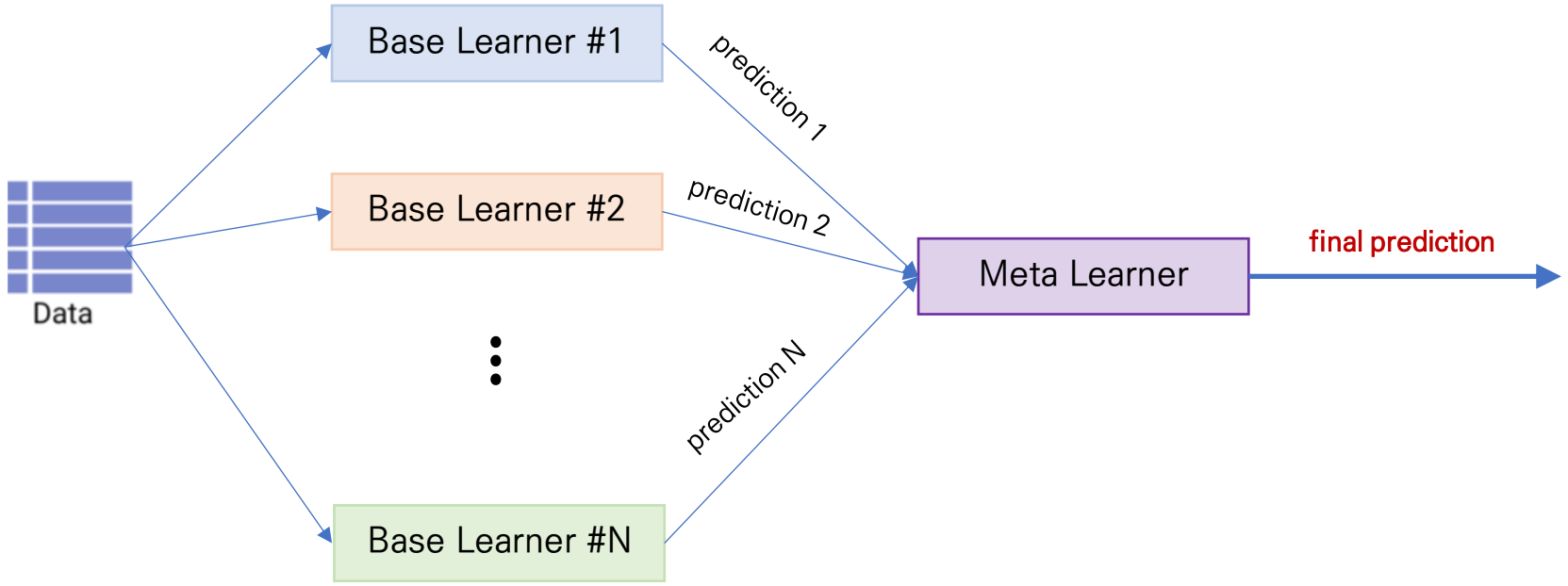
- Stacking

  Meta learner의 컨셉으로 소개되었으며 여러 모델들을 결합하는 방법

  bagging과 boosting과는 달리 이기종 모델간의 통해 모델을 생성

  알고리즘이 서로 결합되었을 경우에 보다 예측력이 높은 모형을 개발할 수 있는 가능성

  단일 모형만으로도 충분히 예측력이 높은 경우에는 모형 결합을 통한 향상 효과가 미미할 수도 있음



Stacking

- Stacking 예시


```
def get_stacking():
    level0 = []
    level0.append(('RF', RandomForestClassifier()))
    level0.append(('SVM', SVC(C=10, gamma=0.1)))

    level1 = LogisticRegression()

    ensemble = StackingClassifier(estimators=level0, final_estimator=level1, cv=5)
    model = OneVsRestClassifier(ensemble)
    return model
```

Stacking

- Stacking 예시

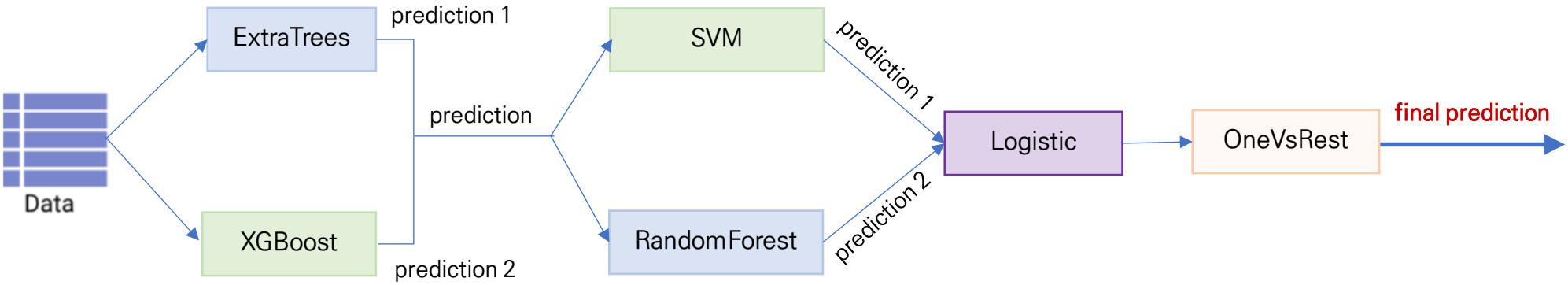

```
def get_stacking():
    level0 = list()
    level0.append(('RF', RandomForestClassifier()))
    level0.append(('EXT', ExtraTreesClassifier()))

    level1 = LogisticRegression()

    ensemble = StackingClassifier(estimators=level0, final_estimator=level1)
    model = OneVsRestClassifier(ensemble)
    return model
```

Stacking

- Multi layer Stacking 예시




```
layer_one_estimators = [
                        ('EXT_1', ExtraTreesClassifier()),
                        ('XGB_1', XGBClassifier())
                       ]
layer_two_estimators = [
                        ('SVM_2', SVC(C=10, gamma=0.1)),
                        ('RF_2', RandomForestClassifier())
                       ]
```

```
layer_two = StackingClassifier(estimators=layer_two_estimators, final_estimator=LogisticRegression(), cv = 5)
```

```
ensemble = StackingClassifier(estimators=layer_one_estimators, final_estimator=layer_two)
model = OneVsRestClassifier(ensemble)
```

27