

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al.)
 - 1.5.2 AdaBoost.SAMME with binary classifier
 - 1.5.3 AdaBoost.SAMME with multi-class classifier
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

Ensemble models

- Machine learning algorithm의 선택과 구성은 overfitting 과의 전쟁이다.
- Ensemble method는 bias-variance trade-off로 인한 손실을 최소화하는 방법으로 볼 수 있다.
- 여러 개의 동질적이거나 이질적인 base model 혹은 weak learner를 동일한 문제에 대해 다른 '표본'으로 훈련시킨 후 결과를 종합하거나 순차적으로 훈련시키는 방법을 ensemble learning이라고 한다.
- Ensemble learning의 기능
 1. Anomaly에 해당하는 특정 관찰값에 대한 의존성을 감소시켜 핵심적인 pattern을 중점적으로 학습(sampling, aggregating)
 2. Outlier를 가급적 모형 안에서 설명할 수 있도록 하여 예측에 반영(weight adjustment)
 3. 단순한 모형을 사용하여 overfitting 가능성 축소하면서 충분한 숫자의 weak learner를 사용하여 pattern 학습
- Ensemble 모형에서 사용하는 기본 단위모형을 base model 혹은 weak learner라고 부르며, 성능이 우수한 모형이나 ensemble model 자체를 strong learner이라고 부르기도 한다.
 - weak learner은 overfitting의 우려가 없는, random guess보다 (살짝) 나은 모형을 의미한다.
 - Stump, k-Nearest Neighbors, Naive Bayes
 - strong learner는 weak learner에 비해 월등한 성과를 보이는 모형을 의미한다.
 - logistic regression, SVM
 - ensemble method는 weak learner들을 strong learner로 바꾸는 것이다.
- 훈련과정과 최종결과를 얻는 과정에 따라 bagging, boosting, stacking으로 구분한다.

1 Bagging

- Bagging은 "bootstrap aggregating"을 의미한다.

1.1 Bootstrapping

- bootstrap은 복원추출로 자료를 생성하는 방법으로
- 표본을 집단으로 간주하여 모집단의 통계값을 계산하는 numerical method이다.
- Subsample의 크기
 - Subsample의 크기를 전체 표본의 크기로 하면, 복원추출의 경우 특정표본이 subsample에 포함되지 않을 확률은 $(1 - 1/n)^n$ 이 되고 그 극한은 e^{-1} 이다.
 - 보통 subsample의 크기는 원표본의 크기와 같도록 하며, 이때 표본의 특정 관찰값이 subsample에 포함될 확률과 포함되는 표본의 비율은 $(1 - 1/n)^n \approx 0.632$ 으로 동일하다. 참고 사이트 (<https://stats.stackexchange.com/questions/88980>)

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준



1.2 Aggregation

- 여러 개의 동질적인 base model을 독립적으로 훈련시켜 결과의 '평균'을 사용한다.
- Bootstrapping은 반복해 추출한 subsample로 계산한 추정값을 이용하여 추정량의 분포를 계산하는 방법으로 신뢰구간이나 복잡한 통계량을 계산할 때 유용한 방법이다.
- Bagging은 bootstrapping으로 구한 통계값을 aggregating 하여 예측에 사용한다.

1.2.1 Regression

- 예측결과의 산술평균

1.2.2 Classification

- Classification은 voting 방식을 사용
 - majority voting
 - weighted voting
 - simple average
 - weighted average
 - stacking
 - bootstrapping average
- hard-voting: 분류결과에 다수결원칙을 적용
- soft-voting: 분류확률을 평균하여 최종결과 도출

(0.1, 0.9), (0.6, 0.4), (0.6, 0.4)

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - ▼ 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.3 Sklearn implementation

1.3.1 Random forest

- [Random forest \(<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html>\)](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html)은 support vector machine과 함께 가장 많이 사용하는 machine learning 분야 분류모형 중 하나이다.
- Keyword argument의 대부분은 DecisionTreeClassifier 와 같다.
- Tree의 수가 많을수록 예측에 유리하지만 시간이 많이 걸리고, 한계적인 이득은 체감한다.

```
sklearn.ensemble.RandomForestClassifier(n_estimators=100, *, criterion='gini', max_depth=None, min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_features='auto', max_leaf_nodes=None, min_impurity_decrease=0.0, min_impurity_split=None, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, class_weight=None, ccp_alpha=0.0, max_samples=None)
```

- Random forest의 base model로는 보통 deep tree를 사용한다.
- shallow tree는 상대적으로 분산이 작지만 상단한 편의를 발생시킨다.
- 반면 deep tree는 분산이 큰 반면 편의가 작으므로 여러 결과의 평균을 예측에 사용하면 분산을 줄이는 효과가 있다.
- Random features
 - Random forest는 훈련과정에서 무작위로 추출한 feature set만을 사용한다.
 - 모형들 사이의 correlation을 감소시킨다.
 - Missing value와 관련된 문제를 효과적으로 완화시킨다.

1.3.2 Bagging

- 분류에서 decision tree를 사용한 bagging이 가장 효과적인 것으로 알려져 있어, random forest는 bagging과 거의 동의어로 사용
- Sklearn에선 classification에 사용하는 모든 machine learning algorithm으로 bagging 모형을 구성할 수 있는 algorithm을 제공

```
sklearn.ensemble.BaggingClassifier(base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)
```

```
sklearn.ensemble.BaggingRegressor(base_estimator=None, n_estimators=10, *, max_samples=1.0, max_features=1.0, bootstrap=True, bootstrap_features=False, oob_score=False, warm_start=False, n_jobs=None, random_state=None, verbose=0)
```

computing time: 0 minutes 42.76seconds

best model: RandomForestClassifier(max_depth=10, random_state=42)

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - ▼ 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
 - ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
 - ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
 - ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
 - 3 Ensemble 모형의 특징 및 선택기준

Random Forest - training dataset

	precision	recall	f1-score	support
died	0.91	0.97	0.94	474
survived	0.95	0.85	0.90	309
accuracy			0.93	783
macro avg	0.93	0.91	0.92	783
weighted avg	0.93	0.93	0.93	783

Random Forest - test dataset

	precision	recall	f1-score	support
died	0.83	0.88	0.86	333
survived	0.76	0.69	0.72	189
accuracy			0.81	522
macro avg	0.80	0.78	0.79	522
weighted avg	0.81	0.81	0.81	522

1.4 Data preparation

- Boosting은 순차적으로 모형의 misclassifications를 보강해가는 방법으로 직전 round의 훈련결과로 다음 단계의 표본 추출에 사용하는 가중값을 결정한다. 어떤 관찰값의 예측오차가 크다면 그 관찰값에 '더욱' 집중하여 모형을 수정해간다.
- 예측오차를 가중값 결정에 반영하므로 regular한 관찰값의 가중값은 점차 줄어드는 반면 irregular한 data point에 대한 가중값은 전과정을 통해 높은 수준으로 유지된다.
- 따라서 outlier는 boosting에서도 여전히 효과적인 학습을 어렵게하고 변수들의 관계를 왜곡시킬 수 있다.
- 일반적으로, 그리고 특히 weak learner로 decision tree를 사용한다면, boosting 모형은 feature의 noisy data에 별다른 영향을 받지 않는다. 다만 형태에 따라 target의 noise는 훈련성과에 상당한 영향을 미칠 수도 있다.
- Target의 noise나 outlier들의 효과는 regression과 classification에서 다르게 나타날 수 있다.
- 실증적으로 boosting이 overfitting과 outliers, noise에 강한 것으로 알려져 있지만 사전에 outlier와 anomaly를 정리하는 것을 추천한다.

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund and Schapire, 1997)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.5 Adaptive boosting, Adaboosting

- 훈련결과를 이용하여 예측오차가 큰 표본에 더 큰 가중값을 부여하여 재추출한 표본으로 다시 훈련한다.

1.5.1 classification, AdaBoost (Freund and Schapire, 1997)

- Weak learner로는 depth가 1인 decision stump을 많이 사용한다. sklearn default는 max_depth=3.
- 전 단계에 사용한 weak learner의 예측오차를 이용하여 설명력이 상대적으로 더 좋은 weak learner가 잘못 예측한 관찰값에 더 큰 가중값을 주면서 순차적으로 예측한다.
- 아래 설명은 label이 $(-1, 1)$ 인 binary classification의 경우이다.
- n 은 표본, m 은 weak learner의 index이다.

1. Initialize the observation weights, $w_i = 1/n, i = 1, \dots, n$

1. for $m = 1, \dots, M$,

(i) find a weak learner $T_m(\cdot)$ using weight w_i that determines observation i 's sampling probability

(ii) compute weighted misspecification rate

$$\text{error}_m = \frac{\sum_{i=1}^n w_i \times 1(c_i \neq T_m(x_i))}{\sum_{i=1}^n w_i}$$

(iii) compute stage value

$$\alpha_m = \frac{1}{2} \ln \frac{1 - \text{error}_m}{\text{error}_m} > 0$$

(iv) update weights

$$w_{m+1,i} = w_{m,i} \times \exp(-c_i \cdot T_m(x_i) \times \alpha_m)$$

(v) normalize weights

$$\sum_{i=1}^n w_{m+1,i} = 1$$

1. output

$$T(x) = \sum_{m=1}^M \alpha_m \cdot T_m(x)$$

$$C(x) = \arg \max_c \sum_{m=1}^M \alpha_m \cdot 1(T_m(x) = c)$$

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-class classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.5.2 AdaBoost.SAMME with binary classification

- Sklearn의 AdaBoostClassifier는 SAMME(Stagewise Additive Modeling using a Multi-class Exponential loss function) algorithm을 사용한다.
- 기본적인 구조는 adaboost와 동일하고, weight를 계산하고 update하는 방법이 약간 다르다.
- compute stage value

$$\alpha_m = \ln \frac{1 - \text{error}_m}{\text{error}_m}$$

- update weights

$$w_{m+1,i} = w_{m,i} \times \exp(\alpha_m \times \mathbf{1}(c_i \neq T_m(x_i)))$$

1.5.3 AdaBoost.SAMME with multi-class classification

- α 계산에 $\ln(C - 1)$ 를 추가하여 weak learner가 random guessing보다는 나아야 한다는 조건을 명시적으로 고려한다.

$$\alpha_m = \ln \frac{1 - \text{error}_m}{\text{error}_m} + \ln(C - 1)$$

- 오분류에 대한 가중값은 SAMME가 더 크며, $C = 2$ 일때는 Adaboost와 같다.
- 여기에서 (<https://hastie.su.domains/Papers/samme.pdf>) 다양한 확장 모형을 확인할 수 있다.

1.5.4 implementation

```
sklearn.ensemble.AdaBoostClassifier(base_estimator=None, *, n_estimators=50, learning_rate=1.0, algorithm='SAMME.R', random_state=None)
```

- learning rate ν 는 다른 모형들과 마찬가지로 훈련속도를 조정하기 위해 사용하며 다음과 같이 적용된다.

$$\alpha_m = \frac{\nu}{2} \ln \frac{1 - \text{error}_m}{\text{error}_m} > 0$$

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund and Schapire, 1997)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.5.5 regression

- Freund and Schapire, 1997, [AdaBoost.R \(\[https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf\]\(https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf\)\)](https://www.face-rec.org/algorithms/Boosting-Ensemble/decision-theoretic_generalization.pdf)
- Drucker AdaBoost.R2, 1997, [AdaBoost.R2 \(<https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.314&rep=rep1&type=pdf>\)](https://citeseerx.ist.psu.edu/viewdoc/download?doi=10.1.1.31.314&rep=rep1&type=pdf)
- Solomatine and Shrestha, 2004, [AdaBoost.RT \(\[https://www.researchgate.net/publication/4116773_AdaBoostRT_A_boosting_algorithm_for_regression_problems\]\(https://www.researchgate.net/publication/4116773_AdaBoostRT_A_boosting_algorithm_for_regression_problems\)\)](https://www.researchgate.net/publication/4116773_AdaBoostRT_A_boosting_algorithm_for_regression_problems)

• Sklearn의 algorithm은 AdaBoost.R2를 사용한다.

1. Initialize the observation weights, $w_{0i} = 1/n, i = 1, \dots, n$

1. for $m = 1, \dots, M$,

(i) find a weak learner $F_m(\cdot)$ using weight $w_{i,m-1}$ that determines observation i 's sampling probability

(ii) compute weighted misspecification rate

$$D_m = \max_i |y_i - F_m(x_i)|$$

$$\text{Linear: } L_{im} = \frac{|y_i - F_m(x_i)|}{D_m}$$

$$\text{Square: } L_{im} = \left(\frac{y_i - F_m(x_i)}{D_m} \right)^2$$

$$\text{Exponential: } L_{im} = 1 - \exp \frac{-(y_i - F_m(x_i))}{D_m}$$

(iii) calculate the model error

$$L_m = \sum_{i=1}^n w_i L_{im}$$

If $L_m > 0.5$, terminate the loop.

(iv) compute stage value (small β means high confidence in the prediction)

$$\beta_m = \frac{L_m}{1 - L_m}$$

(v) update and normalize weights

$$w_{m+1,i} = \frac{w_{im}\beta^{(1-L_{im})}}{\sum_{i=1}^n w_{im}\beta^{(1-L_{im})}}$$

1. output

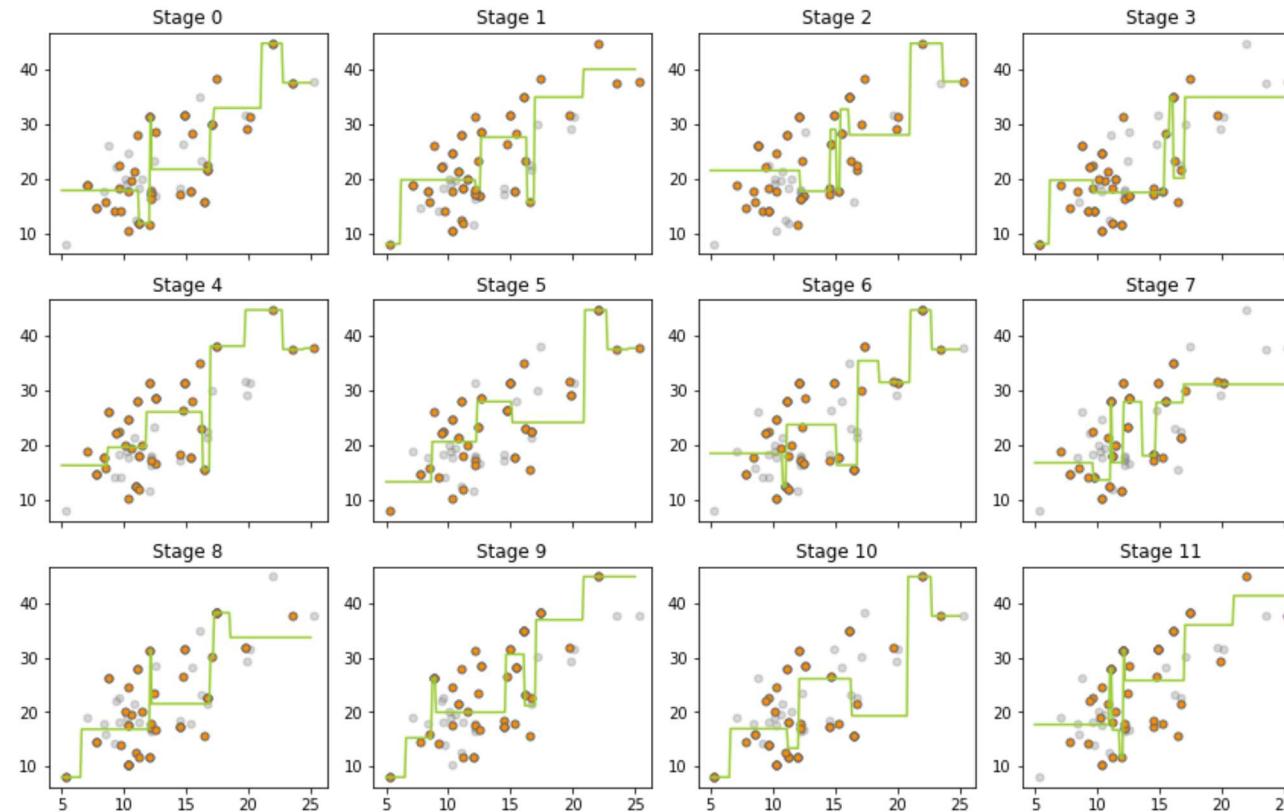
$$F(x) = \inf \left[y \in Y \mid \sum_{m: F_m(x) < y} \ln \frac{1}{\beta_m} \geq \frac{1}{2} \ln \frac{1}{\beta_m} \right], \quad Y = \{y_1, \dots, y_M\}$$

```
sklearn.ensemble.AdaBoostRegressor(base_estimator=None, *, n_estimators=50, learning_rate=1.0, loss='linear', random_state=None)
```

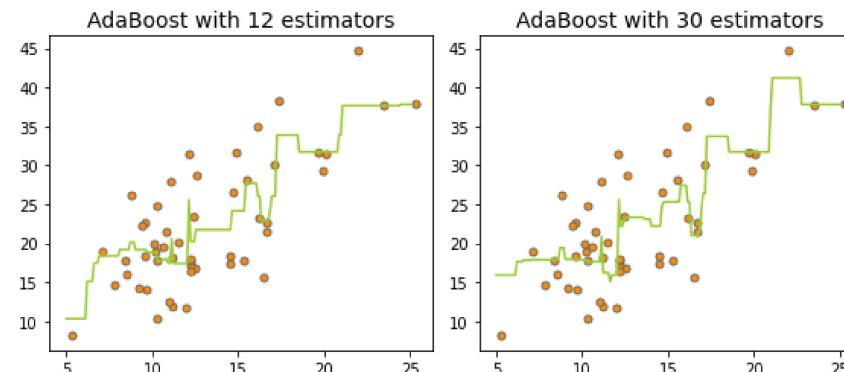
Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, AdaBoosting
 - 1.5.1 classification, AdaBoost (Freund et al.)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

Gradient Boost with 12 stages of depth 3 tree



AdaBoost Regression Tree of depth 3



Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
 - 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
 - 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.6 Gradient boosting

Jerome Friedman, 2001, Greedy Function Approximation: A Gradient Boosting Machine <https://doi.org/10.1214/aos/1013203451>
(<https://doi.org/10.1214/aos/1013203451>)

- Adaboost는 예측을 하고 이에 따른 예측오차를 계산하는 반면 Gradient boosting에선 이전 단계의 예측오차를 target으로 학습한다.
- '예측한' 예측오차에 근거해 이전 모형을 수정보완하다.
- Gradient boost는 최적화과정에서 gradient descent algorithm을 사용해서 붙은 이름이고, 분류문제에서도 target을 연속변수로 취급한다.
- Regression과 classification 모두에서 weak learner는 leaf의 수를 제한한 restricted tree를 사용한다.
- 훈련과정에 loss function의 도함수를 이용하므로 loss function은 미분이 가능해야 한다.

1.6.1 Residual fitting 과 gradient

- Loss function가 MSE라면 도함수는 잔차에 (-)를 곱한 값, 즉 negative residual이다.

$$L(y_i, F(x_i)) = \frac{1}{2} \sum_{i=1}^n (y_i - F(x_i))^2$$
$$\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = \sum_{i=1}^n -(y_i - F(x_i))$$

- Gradient는 함수값을 가장 크게 증가시킬 수 있는 변화를 표시하므로 이 정보를 이용하여 오차를 줄여간다.
- Stochastic Gradient Boosting은 표본의 일부로 모형을 훈련시킨 후 전체 표본에 대한 예측값을 계산해 사용한다.

Contents ☰

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
 - 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-class classification
 - 1.5.4 implementation
 - 1.5.5 regression
 - 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
 - 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
 - 3 Ensemble 모형의 특징 및 선택기준

1.6.2 Sketch

1. 실제 값과 $(m - 1)$ 단계에서 구한 예측값의 차이인 예측오차 혹은 residual r_{im} 을 구한다.

$$y_i = F_0(x_i) + r_{i0}$$
$$r_{i,m-1} = F_{m-1}(x_i) + r_{im}, \quad m \geq 1$$

1. r_{im} 을 weak learner로 훈련시켜 예측값 $\gamma_m = (\gamma_{1m}, \dots, \gamma_{J_m m})$ 을 계산하고 이를 이용해 예측값을 update한다.

$$F_m(x_i) = F_{m-1}(x_i) + \nu_m \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x_i \in R_j)$$
$$= \bar{y} + \nu_m \sum_{m=1}^M \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x_i \in R_j)$$

ν_m 의 크기는 일변수함수의 극대화문제로 결정한다.

$$\nu_m = \arg \min_{\nu} \sum_i^n L(y_i, F_{m-1}(x_i) + \nu \gamma_m(x_i))$$

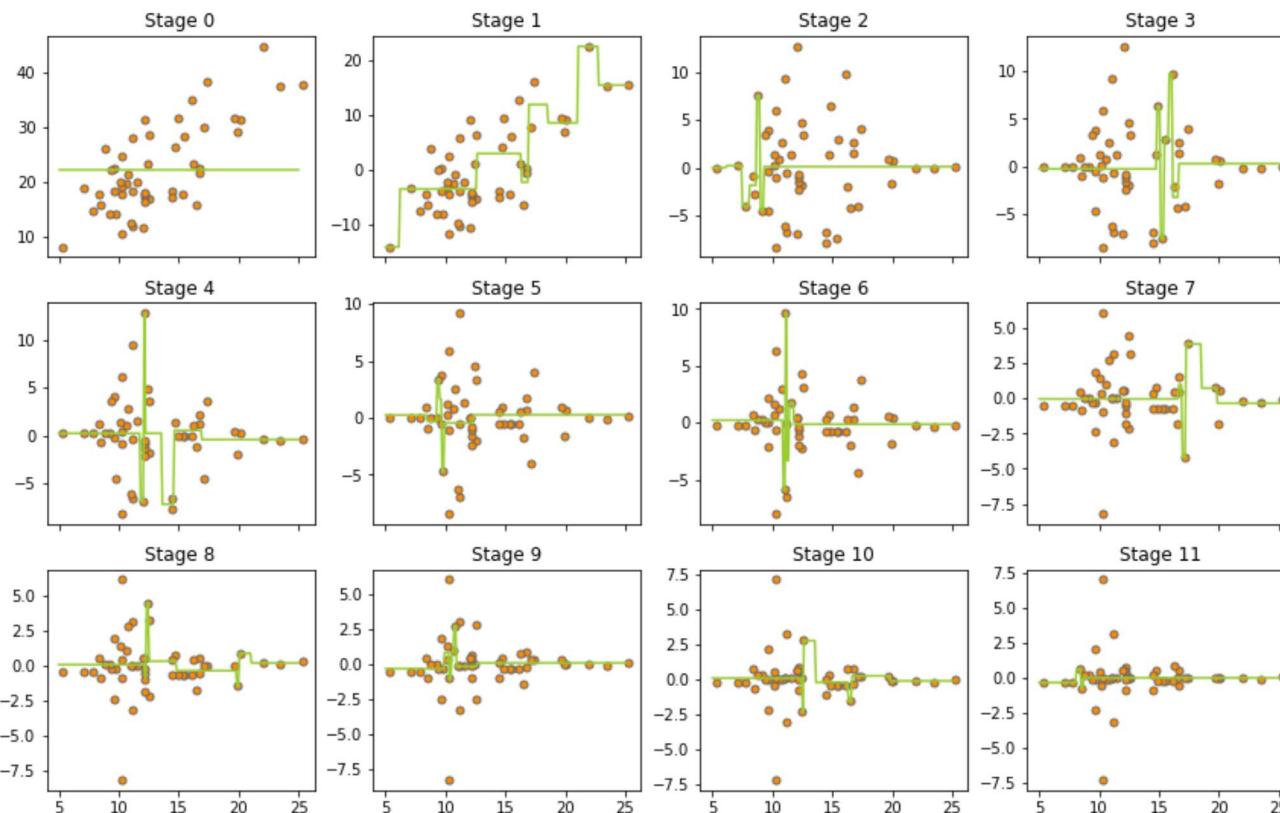
$$F_m(x_i) = F_{m-1}(x_i) + \nu \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x_i \in R_j)$$

- ν_m 은 learning rate로 훈련을 효율적으로 만들지만 실제 적용에선 대부분 $\nu > 0$ 를 이용해 정확도에 관계없이 m 개의 훈련결과를 모두 동일하게 반영한다. Shrinkage라고 부르기도 한다
- 보통 0.0001과 0.1 사이의 값을 많이 사용하며 learning rate이 작을수록 결과가 수렴하는데 더 많은 시간이 걸린다.

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al.)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

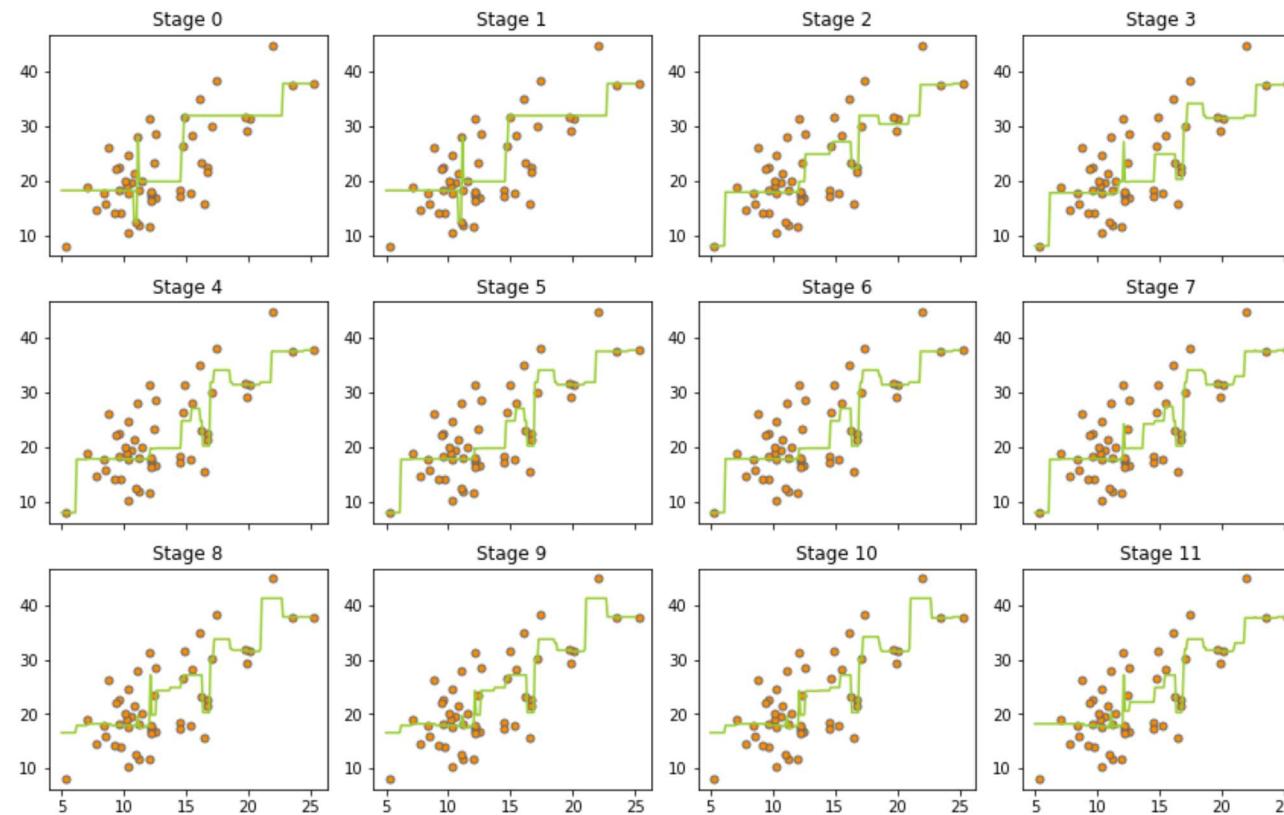
Gradient Boost with 12 stages of depth 3 tree



Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, AdaBoosting
 - 1.5.1 classification, AdaBoost (Freund et al.)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

Stage Predictions of Gradient Boost



Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, AdaBoosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.6.3 Classification

- 확률이나 log-odd를 이용한 예측 결과는 동일하고, 오차를 극소화하는 문제이므로 (sklearn에서 logit이라고 부르는) log-odd를 사용한다.
- 1. initialize model: 첫 단계에선 모든 관찰값의 예측확률이 동일하다고 가정하며, 그 값으로 표본비율을 사용한다. $n-$ 과 $n+$ 를 $y_i = 0, 1$ 인 표본의 수, n 을 전체 표본의 크기라고 하면,

$$\Pr(y_i = 1) = p_{i0} = \pi_1 = \frac{n+}{n}$$

(i) Pseudo-residual은 실제값과 prediction값인 log-odd의 차이로 계산하므로 prediction을 구해보면,

$$F_0(x_i) = \ln \frac{p_{i0}}{1 - p_{i0}}$$

(ii) 따라서 첫 stage의 pseudo-residual은 다음과 같다.

$$r_{i0} = y_i - F_0(x_i), \quad \Pr(y_i = 1|x_i \in R_{j0}) = \frac{e^{r_{i0}}}{1 + e^{r_{i0}}} (= \pi_1)$$

1. for $m = 1$ to M ,

(i) $r_{i,m-1}$ 을 target으로 tree를 훈련시켜 partition R_m 을 구한다.

(ii) Partition R_m 을 기준으로 stage $m - 1$ 의 log-odd 예측오차로 pseudo-residual을 계산한다. 여기서 $R_m(i)$ 는 i 가 속한 partition을 표시한다.

$$r_{im} = \frac{\sum_{j \in R_m(i)} r_{jm}}{\sum_{j \in R_m(i)} p_{j,m-1} (1 - p_{j,m-1})}$$

$$r_{jm} = y_i - F_{m-1}(x_j), \quad p_{j,m-1} = \frac{\exp(F_{m-1}(x_j))}{1 + \exp(F_{m-1}(x_j))}$$

(iii) $p > 0.5$ 이고 $p(1 - p)$ 은 예측오차가 클수록 값이 작아지므로, 오차가 큰 관찰값이 속한 leaf에 더 큰 가중값을 부여한다.

$$r_{im} = -\frac{\partial \text{loss}_m(y_i, F(x_i))}{\partial F(x_i)} \Big|_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n$$

- [Gradient Boosting Visualization \(\[http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html\]\(http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html\)\)](http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html)

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - ▼ 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
 - ▼ 1.5 Adaptive boosting, AdaBoosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
 - ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
 - ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.6.4 Regression

1. Initialize model with a constant value

$$F_0(x) = \arg \min_{\gamma} \sum_{i=1}^n L(y_i, \gamma)$$

1. for $m = 1$ to M ,

- (i) pseudo-residual을 계산한다.

$$r_{im} = -\frac{\partial \text{loss}_m(y_i, F(x_i))}{\partial F(x_i)} \Big|_{F(x)=F_{m-1}(x)} \quad \text{for } i = 1, \dots, n$$

loss function이 MSE라면 pseudo-residual은 deviation이 된다.

- (ii) r_{im} 을 target으로 하는 regression tree를 훈련시켜 terminal node, $R_{jm}, j = 1, \dots, J_m$ 를 구한다.

- (iii) terminal node 별로 전 단계의 모형 $F_{m-1}(x_i)$ 을 보정할 γ_{ij} 계산

$$\gamma_{jm} = \arg \min_{\gamma} \sum_{x_i \in R_{jm}} L(y_i, F_{m-1}(x_i) + \gamma) \quad \text{for } i = 1, \dots, n$$

- (iv) update the model

$$F_m(x) = F_{m-1}(x) + \nu_m \sum_{j=1}^{J_m} \gamma_{jm} \mathbf{1}(x \in R_{jm})$$

여기서 ν_m 는 learning rate

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

- Regression과 classification에 동일한 algorithm을 사용하며 자세한 설명은 [여기](https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502) (<https://towardsdatascience.com/all-you-need-to-know-about-gradient-boosting-algorithm-part-1-regression-2520a34a502>)를 참고

```
sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

```
sklearn.ensemble.GradientBoostingRegressor(*, loss='squared_error', learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse', min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0, max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None, max_features=None, alpha=0.9, verbose=0, max_leaf_nodes=None, warm_start=False, validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

- Documentation에선 10,000개 이상의 큰 표본이라면 [Histogram-based Gradient Boosting Classification Tree](https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html?highlight=gradientboost#sklearn-ensemble-histgradientboostingclassifier) (<https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.HistGradientBoostingClassifier.html?highlight=gradientboost#sklearn-ensemble-histgradientboostingclassifier>)나 [LightGBM](https://github.com/Microsoft/LightGBM#light-gradient-boosting-machine) (<https://github.com/Microsoft/LightGBM#light-gradient-boosting-machine>) 같은 algorithm 사용을 추천한다.

1.7 XGBoost

Tianqi Chen and Carlos Guestrin, 2016, XGBoost: A Scalable Tree Boosting System, <https://arxiv.org/abs/1603.02754> (<https://arxiv.org/abs/1603.02754>)

- eXtreme Gradient Boost을 의미하며 Gradient boost에 효과적인 최적화 algorithm과 분기 가능성 등 여러가지 기능을 추가한 것으로 연산속도에 엄청난 장점이 있다.
- Tianqi Chen and Carlos Guestrin algorithm의 특징
 - 각각의 decision tree 계산에서 제한적인 병렬연산 (GPU연산지원)
 - feature 별로 정렬을 해 놓고 최적 split를 탐색하여 사용
 - depth-first 방식의 tree pruning을 통한 feature selection
 - 최적화에 Newton method 사용
 - hardware optimization
 - regularization이 가능, XGBoost는 regularized boosting 으로 부르기도 한다.
 - Sparsity-aware Split Finding algorithm으로 missing value를 효과적으로 처리
 - distributed weighted Quantile Sketch algorithm으로 효과적인 분기
 - built-in cross-validation

```
conda install -c conda-forge py-xgboost
```

```
import xgboost as xgb  
print(xgb.__version__)
```

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
- 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-class classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.7.1 Classification

- XGboost의 기본골격은 AdaBoost와 유사하다.

1. initialize model:

- AdaBoost와는 달리 초기값은 $p_{i0} = 0.5$ 로 시작을 한다.
- Pseudo-residual은 AdaBoost와 마찬가지로 실제값과 log-odd의 차이이므로 첫 stage의 pseudo-residual은 다음과 같다.

$$F_0(x_i) = \ln \frac{p_{i0}}{1 - p_{i0}} = 0, \quad r_{i0} = y_i - F_0(x_i) = y_i$$

1. for $m = 1$ to M ,

(i) $r_{i,m-1}$ 을 target으로 하는 tree를 훈련시켜 partition R_m 을 구한다.

(ii) Partition R_m 을 기준으로 stage $m - 1$ 의 log-odd 예측오차로 pseudo-residual을 계산한다. 여기서 $R_m(i)$ 는 i 가 속한 partition을 표시한다.

$$\text{Similarity}_{im} = \frac{\left(\sum_{j \in R_m(i)} r_{jm} \right)^2}{\sum_{j \in R_m(i)} p_{j,m-1} (1 - p_{j,m-1}) + \lambda}$$
$$r_{jm} = y_i - F_{m-1}(x_j), \quad p_{j,m-1} = \frac{\exp(F_{m-1}(x_j))}{1 + \exp(F_{m-1}(x_j))}$$

1. 분기는 cover로 결정한다. (sklearn의 min_child_weight)

(i) cover는 similarity의 분모에서 λ 를 제외한 값으로 $p(1 - p)$ 는 예측이 정확할수록 작은값이 되므로 해당 node에 속한 관찰값들의 similarity 합이 cover 보다 작으면 분기를 하지 않는다.

(ii) 한편 각 branch의 가치는 gain으로 측정하며, $gain - \gamma$ 가 0보다 작으면 해당 가지를 자른다.

$$gain = \text{Similarity}_{left} + \text{Similarity}_{right} - \text{Similarity}_{parent}$$
$$F_m(x_i) = F_{m-1}(x_i) + \nu \cdot r_{im}$$

1. output

$$\text{predicted log-odd} = F_0(x_i) + \nu \sum_{m=1}^M F_m(x_i)$$
$$\text{predicted probability} = \frac{e^{\log\text{-odd}}}{1 + e^{\log\text{-odd}}}$$

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

1.7.2 Regression

- Initial stage 에선 classification과 동일하게 모든 관찰값들이 동일한 예측값을 갖는 것으로 가정하여 Pseudo residual 계산

1. Pseudo residual로 중간단계의 각 leaf의 similarity를 계산한다.

$$\text{Similarity Score} = \frac{\text{sum of squared residuals}}{\text{number of residuals} + \lambda}$$

1. 분기의 효과는 Gain으로 측정한다. Gain이 0보다 크면 가능한 선에서 분기를 계속한다.

$$\text{gain} = \text{Similarity}_{\text{left}} + \text{Similarity}_{\text{right}} - \text{Similarity}_{\text{parent}}$$

1. Tree 구성을 마친 후 gain과 threshold $\gamma \geq 0$ 을 비교하여 가지를 정리한다. 즉 $\text{Gain} - \gamma > 0$ 일 때만 해당 가지를 남겨두고 나머지는 제거한다.

1. Leaf의 output value를 결정한다.

$$\frac{\text{sum of residuals}}{\text{number of residuals} + \lambda}$$

1. 완성된 tree로 이전 tree를 update 한다.

$$F_m(x_i) = F_{m-1}(x_i) + \nu \times \text{residual}_{im}$$

1. Update한 tree로 residual을 계산하고 stopping rule을 만족할 때까지 위 과정을 반복한다.

Contents ⚙

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
- ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
- ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
- ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
- ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
 - 3 Ensemble 모형의 특징 및 선택기준

1.7.3 Implementation

- XGBoost는 python 이외에도 여러가지 언어를 지원하며 sklearn와 호환가능한 [wrapper](https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn) (https://xgboost.readthedocs.io/en/stable/python/python_api.html#module-xgboost.sklearn)도 제공한다.
- 대부분의 keyword arguments 다른 boost 모형들과 동일하게 사용한다.

```
xgboost.XGBRegressor(*, objective='reg:squarederror', **kwargs)
```

- Attribute 중 feature_importances_ 몇가지 기준으로 각 feature의 중요도를 보여준다. 관련 keyword는 importance_type 이다.
 - For tree model, it's either "gain", "weight", "cover", "total_gain" or "total_cover".
 - For linear model, only "weight" is defined and it's the normalized coefficients without bias.

```
from xgboost import XGBRegressor
from xgboost import plot_importance
```

```
xgb = XGBClassifier(max_depth=3, learning_rate=0.02, objective= 'binary:logistic')
xgb.fit(X_train, y_train)
```

```
print(f'Accuracy of XGB classifier on training set: {xgb.score(X_train, y_train):.2f}')
print(f'Accuracy of XGB classifier on test set: {xgb.score(X_test[X_train.columns], y_test):.2f}'
```

```
fig, ax = plt.subplots(figsize=(10,8))
plot_importance(xgb, ax=ax)
```

- 이외에도 각 feature의 예측에 대한 기여도는 permutation_importance (https://scikit-learn.org/stable/modules/generated/sklearn.inspection.permutation_importance.html#sklearn.inspection.permutation_importance)나 Shapley value를 이용한 SHAP (<https://github.com/slundberg/shap>) (SHapley Additive exPlanations)으로 계산할 수 있다.

Contents

- ▼ 1 Bagging
 - 1.1 Bootstrapping
 - ▼ 1.2 Aggregation
 - 1.2.1 Regression
 - 1.2.2 Classification
 - ▼ 1.3 Sklearn implementation
 - 1.3.1 Random forest
 - 1.3.2 Bagging
 - 1.4 Data preparation
 - ▼ 1.5 Adaptive boosting, Adaboosting
 - 1.5.1 classification, AdaBoost (Freund et al., 1995)
 - 1.5.2 AdaBoost.SAMME with binary classification
 - 1.5.3 AdaBoost.SAMME with multi-classification
 - 1.5.4 implementation
 - 1.5.5 regression
 - ▼ 1.6 Gradient boosting
 - 1.6.1 Residual fitting 과 gradient
 - 1.6.2 Sketch
 - 1.6.3 Classification
 - 1.6.4 Regression
 - ▼ 1.7 XGBoost
 - 1.7.1 Classification
 - 1.7.2 Regression
 - 1.7.3 Implementation
- ▼ 2 Ensemble Stacking
 - 2.1 base model 과 meta model의 선택
 - 2.2 훈련과정
 - 2.3 Implementation
- 3 Ensemble 모형의 특징 및 선택기준

2 Ensemble Stacking

2.1 base model 과 meta model의 선택

- Base model로는 보통 서로 다른 strong learner들을 사용
- 자료를 여러 측면으로 분석하기 위해 기본 가정들이 다른 모형을 사용하는 경우가 많다.
 - linear models, logistic regression, Naive Bayes, (decision trees), support vector machines, neural network
- meta model은 해석이 용이한 모형을 주로 사용한다.
 - regression meta model: Linear regression
 - classification meta-model: Logistic regression
- Mixture of experts (MoE)라고 부르기도 한다.

2.2 훈련과정

1. 표본을 둘로 나누어 base model들과 meta model 훈련에 사용한다.
2. Base model은 k-fold cross-validation으로 훈련
3. Meta-model은 overfitting 문제를 완화시키기 위해 훈련을 마친 base model의 `sklearn.model_selection.cross_val_predict`로 예측한 값을 이용해 훈련시킨다.

2.3 Implementation

```
sklearn.ensemble.StackingClassifier(estimators, final_estimator=None, *, cv=None, stack_method='auto', n_jobs=None, passthrough=False, verbose=0)
```

- Keyword cv로 훈련에 사용할 자료를 선택할 수 있다. 별도로 훈련시킨 base model을 사용하려면 'prefit' option을 사용한다. passthrough는 예측값과 원자료를 동시에 사용 여부를 결정한다.

3 Ensemble 모형의 특징 및 선택기준

- 분산의 감소가 주목적인 경우가 많으므로 보통은 편의가 작은 weak learner를 선택한다.
- 각 모형은 독립적으로 훈련하므로 병렬연산이 가능하다.
- 분산감소에는 효과적이지만 편의는 줄이지 못하는 것으로 알려져 있다.
- 극단적으로 예측에 중점을 둔 모형으로 추론은 불가능하다.
- Bagging: 예측의 정확성보다는 분산을 감소시키는 것이 중요한 경우
- Boosting: 분산을 희생하더라도 예측의 정확성이 중요할 경우
- stacking: 예측의 정확성과 분산을 균형있게 감소시킬 경우