Week3 - Homework

Sanghyuk Lee

Contents

AdaBoost

Bagging

RandomForest

AdaBoost - Classifier

In sklearn,

A meta-estimator which starts by fitting a model on the original data.

After that, it fits additional copies of the classifier on **the same data but where the weights of incorrectly classified instances are adjusted.**

-> Is there no subsampling?

AdaBoost - Classifier

base_estimator : *object, default=None*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

n_estimators : *int, default=50*

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate : *float, default=1.0*

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters.

algorithm : *{'SAMME', 'SAMME.R'}, default='SAMME.R'*

If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

random_state : *int, RandomState instance or None, default=None*

Controls the random seed given at each `base_estimator` at each boosting iteration. Thus, it is only used when `base_estimator` exposes a `random_state`. Pass an int for reproducible output across multiple function calls. See Glossary.

No n_jobs.
No bootstrap.
…

AdaBoost - Classifier - review a paper

- Zhu, H. Zou, S. Rosset, T. Hastie, “Multi-class AdaBoost”, 2009.

problem and the AdaBoost algorithm [8]. Suppose we are given a set of training data $(\boldsymbol{x}_1, c_1), \ldots, (\boldsymbol{x}_n, c_n)$, where the input (prediction variable) $\boldsymbol{x}_i \in \mathbb{R}^p$, and the output (response variable) c_i is qualitative and assumes values in a finite set, e.g. $\{1, 2, \ldots, K\}$. K is the number of classes. Usu-

AdaBoost - Classifier - review a paper

- Zhu, H. Zou, S. Rosset, T. Hastie, “Multi-class AdaBoost”, 2009.

Algorithm 1. *AdaBoost [8]*

1. *Initialize the observation weights* $w_i = 1/n$, $i = 1, 2, \ldots, n$.
2. *For* $m = 1$ *to* M:
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$

   (c) *Compute*

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}}.$$

   (d) *Set*

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$

   for $i = 1, 2, \ldots, n$.

Algorithm 2. *SAMME*

1. *Initialize the observation weights* $w_i = 1/n$, $i = 1, 2, \ldots, n$.
2. *For* $m = 1$ *to* M:
   (a) *Fit a classifier* $T^{(m)}(\boldsymbol{x})$ *to the training data using weights* w_i.
   (b) *Compute*

$$err^{(m)} = \sum_{i=1}^{n} w_i \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right) / \sum_{i=1}^{n} w_i.$$

   (c) *Compute*

$$\alpha^{(m)} = \log \frac{1 - err^{(m)}}{err^{(m)}} + \log(K - 1). \quad (1)$$

   (d) *Set*

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$

   for $i = 1, \ldots, n$.
   (e) *Re-normalize* w_i.
3. *Output*

$$C(\boldsymbol{x}) = \arg\max_k \sum_{m=1}^{M} \alpha^{(m)} \cdot \mathbb{I}(T^{(m)}(\boldsymbol{x}) = k).$$

Note that Algorithm 2 (SAMME) shares the same simple modular structure of AdaBoost with a *simple but subtle* difference in (1), specifically, the extra term $\log(K - 1)$. Obviously, when $K = 2$, SAMME reduces to AdaBoost. However, the term $\log(K - 1)$ in (1) is critical in the multi-class case ($K > 2$). One immediate consequence is that now in order

AdaBoost - Classifier - review a paper

- Zhu, H. Zou, S. Rosset, T. Hastie, “Multi-class AdaBoost”, 2009.

(d) *Set*

$$w_i \leftarrow w_i \cdot \exp\left(\alpha^{(m)} \cdot \mathbb{I}\left(c_i \neq T^{(m)}(\boldsymbol{x}_i)\right)\right),$$

for $i = 1, \ldots, n$.

AdaBoosting: Algorithm

Algorithm 2 Adaboost

Input: Required ensemble size T

Input: Training set $S = \{(x_1, y_1), (x_2, y_2), ..., (x_N, y_N)\}$, where $y_i \in \{-1, +1\}$

Define a uniform distribution $D_1(i)$ over elements of S.

for $t = 1$ to T **do**

Train a model h_t using distribution D_t.

Calculate $\epsilon_t = P_{D_t}(h_t(x) \neq y)$

If $\epsilon_t \geq 0.5$ break

Set $\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$

Update $D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$

where Z_t is a normalization factor so that D_{t+1} is a valid distribution.

end for

For a new testing point (x', y'),

$H(x') = sign\left(\sum_{t=1}^{T} \alpha_t h_t(x')\right)$

Weight is updated in all the samples of original data. Also, **wi is retained if sample i is appropriately classified.**

AdaBoost - Classifier - review a paper

From the context I found, I thought that Adaboost in sklearn doesn't use subsampling while we had studied. Also the method applied to update weight is slightly different from stuff that we studied.

AdaBoost - Classifier - Hyperparameters

Parameters:

base_estimator : *object, default=None*

The base estimator from which the boosted ensemble is built. Support for sample weighting is required, as well as proper `classes_` and `n_classes_` attributes. If `None`, then the base estimator is `DecisionTreeClassifier` initialized with `max_depth=1`.

n_estimators : *int, default=50*

The maximum number of estimators at which boosting is terminated. In case of perfect fit, the learning procedure is stopped early.

learning_rate : *float, default=1.0*

Weight applied to each classifier at each boosting iteration. A higher learning rate increases the contribution of each classifier. There is a trade-off between the `learning_rate` and `n_estimators` parameters.

algorithm : *{'SAMME', 'SAMME.R'}, default='SAMME.R'*

If 'SAMME.R' then use the SAMME.R real boosting algorithm. `base_estimator` must support calculation of class probabilities. If 'SAMME' then use the SAMME discrete boosting algorithm. The SAMME.R algorithm typically converges faster than SAMME, achieving a lower test error with fewer boosting iterations.

random_state : *int, RandomState instance or None, default=None*

Controls the random seed given at each `base_estimator` at each boosting iteration. Thus, it is only used when `base_estimator` exposes a `random_state`. Pass an int for reproducible output across multiple function calls. See Glossary.

AdaBoost - Classifier - Hyperparameters

Base_estimator (default : DecisionTreeClassifier with max_depth = 1)
- The base estimator used as **weak learner**.
- It supports for **sample weighting**, classes_ and n_classes attributes.

n_estimators (default = 50)
- The max number of estimators at which boosting is terminated.
- In case of perfect fit, stop early.

learning_rate (default = 1.0)
- Weight applied to each classifier at each boosting iter.

Algorithm (default = SAMME.R)
- **SAMME.R -> support calculation of class prob -> converges faster.**
- SAMME -> discrete boosting

random_state (default = None)
- It is only used when base_estimator exposes a random_state.

AdaBoost - Classifier - Hyperparameters

A visual explanation of the trade-off between learning rate and iterations

This post is based on the assumption that the AdaBoost algorithm is similar to the M1 or SAMME implementations which can be sumarized as follows:

Let $G_m(x)\ m = 1, 2, \ldots, M$ be the sequence of weak classifiers, our objective is:

$$G(x) = \text{sign}\left(\alpha_1 G_1(x) + \alpha_2 G_2(x) + \ldots \alpha_M G_M(x)\right) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m G_m(x)\right)$$

AdaBoost.M1

1. Initialize the obervation weights $w_i = 1/N$
2. For $m = 1, 2, \ldots, M$
   - Compute the weighted error $Err_m = \frac{\sum_{i=1}^{N} w_i \mathcal{I}(y^{(i)} \neq G_m(x^{(i)}))}{\sum_{i=1}^{N} w_i}$
   - Compute the estimator coefficient $\alpha_m = L \log\left(\frac{1 - err_m}{err_m}\right)$ where $L \leq 1$ is the learning rate
   - Set data weights $w_i \leftarrow w_i \exp[\alpha_m \mathcal{I}(y^{(i)} \neq G_m(x^{(i)}))]$
   - To avoid numerical instability, normalize the weights at each step $w_i \leftarrow \frac{w_1}{\sum_{i=1}^{N} w_i}$
3. Output $G(x) = \text{sign}\left[\sum_{m=1}^{M} \alpha_m G_m(x)\right]$

The impact of Learning Rate L and the number of weak classifiers M

From the above algorithm we can understand intuitively that

- **Decreasing the learning rate** L makes the coefficients α_m smaller, which reduces the amplitude of the sample_weights at each step (since $w_i \leftarrow w_i e^{\alpha_m \mathcal{I} \cdots}$). This translates into smaller variations of the weighted data points and therefore fewer differences between the weak classifier decision boundaries
- **Increasing the number of weak classifiers** M increases the number of iterations, and allows the sample weights to gain greater amplitude. This translates into 1) more weak classifiers to combine at the end, and 2) more variations in the decision boundaries of these classifiers. Put together these effects tend to lead to more complex overall decision boundaries.

From this intuition, it would make sense to see a **trade-off** between the parameters L and M. Increasing one and decreasing the other will tend to cancel the effect.

AdaBoost - Regressor - Hyperparameters

Base_estimator (default : DecisionTreeRegressor with max_depth = 3)
- The base estimator used as **weak learner**.

n_estimators (default = 50)
- The max number of estimators at which boosting is terminated.
- In case of perfect fit, stop early.

learning_rate (default = 1.0)
- Weight applied to each classifier at each boosting iter.

loss (default =linear)
- **linear , square, exponential**
- **The loss to use when updating the weights after each boosting iter.**

random_state (default = None)
- It is only used when base_estimator exposes a random_state.

AdaBoost - Regressor - Hyperparameters

- Drucker. “Improving Regressors using Boosting Techniques”, 1997.

2. Construct a regression machine t from that training set. Each machine makes a hypothesis: $h_t: x \rightarrow y$

3. Pass *every* member of the training set through this machine to obtain a prediction $y_i^{(p)}(\boldsymbol{x}_i)$ $i=1,...N_1$.

4. Calculate a loss for each training sample $L_i = L\left[\,|\,y_i^{(p)}(\boldsymbol{x}_i) - y_i\,|\,\right]$. The loss L may be of any functional form as long as $L \in [0,1]$. If we let

$$D = sup\,|\,y_i^{(p)}(\boldsymbol{x}_i) - y_i\,| \quad i=1,...,N_1$$

then we have three candidate loss functions:

$$L_i = \frac{|\,y_i^{(p)}(\boldsymbol{x}_i) - y_i\,|}{D} \qquad (linear)$$

$$L_i = \frac{|\,y_i^{(p)}(\boldsymbol{x}_i) - y_i\,|^2}{D^2} \qquad (square\ law)$$

$$L_i = 1 - exp\left[\frac{-|\,y_i^{(p)}(\boldsymbol{x}_i) - y_i\,|}{D}\right] \qquad (exponential)$$

5. Calculate an average loss: $\bar{L} = \sum_{i=1}^{N_1} L_i p_i$

6. Form $\beta = \frac{\bar{L}}{1-\bar{L}}$. β is a measure of confidence in the predictor. Low β means high confidence in the prediction.

7. Update the weights: $w_i \rightarrow w_i \beta^{**}[1-L_i]$, where ** indicates exponentiation. The smaller the loss, the more

Bagging- Classifier - Hyperparameters

base_estimator(*default=None)*
- The base estimator to fit on random subsets of the dataset.
- If None, then the base estimator is a DecisionTreeClassifier.

n_estimators*int (default=10)*
- The number of base estimators in the ensemble.

max_samples *(default=1.0)*
- The number of samples to draw from X to train each base estimator (with replacement by default, see bootstrap for more details).

Bagging- Classifier - Hyperparameters

max_features ***(default=1.0)***
The number of features to draw from X to train each base estimator

bootstrap***(default=True)***
Whether samples are drawn with replacement.

bootstrap_features ***(default=False)***
Whether features are drawn with replacement.

oob_score ***(default=False)***

warm_start ***default=False***
When set to True, reuse the solution of the previous call to fit and add more estimators to the ensemble.

Bagging- Classifier - Hyperparameters

Warm Start

When fitting an estimator repeatedly on the same dataset, but for multiple parameter values (such as to find the value maximizing performance as in grid search), it may be possible to reuse aspects of the model learned from the previous parameter value, saving time. When `warm_start` is true, the existing fitted model attributes are used to initialize the new model in a subsequent call to fit.

Bagging- Classifier - Hyperparameters

random_state*(default=None)*

verbose *(default=0)*
Controls the verbosity when fitting and predicting.

```
building tree 1 of 500
building tree 2 of 500
[Parallel(n_jobs=1)]: Done   1 out of   1 | elapsed:    4.6s remaining:    0.0s
building tree 3 of 500
building tree 4 of 500
building tree 5 of 500
building tree 6 of 500
building tree 7 of 500
building tree 8 of 500
building tree 9 of 500
building tree 10 of 500
building tree 11 of 500
```

Bagging- Regressor- Hyperparameters

There are no big differences in regressor of bagging.

RandomForest- Classifier- Hyperparameters

n_estimators (default=100)
- The number of trees in the forest.

criterion (default=”gini”)
- “gini”, “entropy”

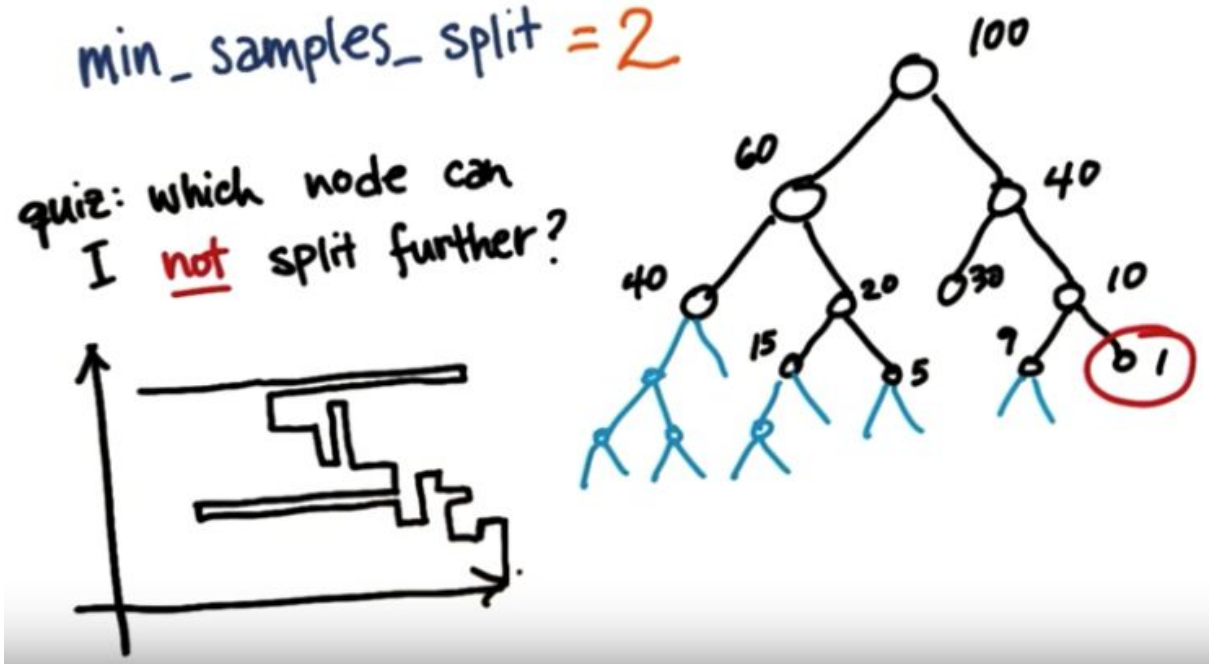
max_depth (default=None)
- The maximum depth of the tree.
- If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_split (default=2)
- The minimum number of samples required to split an internal node

RandomForest- Classifier- Hyperparameters

min_samples_split (default=2)

- The minimum number of samples required to split an internal node:



RandomForest- Classifier- Hyperparameters

min_samples_leaf (default=1)

- **The minimum number of samples required to be at a leaf node.**

min_weight_fraction_leaf (default=0.0)

- The minimum weighted fraction of the sum total of weights (of all the input samples) required to be at a leaf node. Samples have equal weight when sample_weight is not provided.

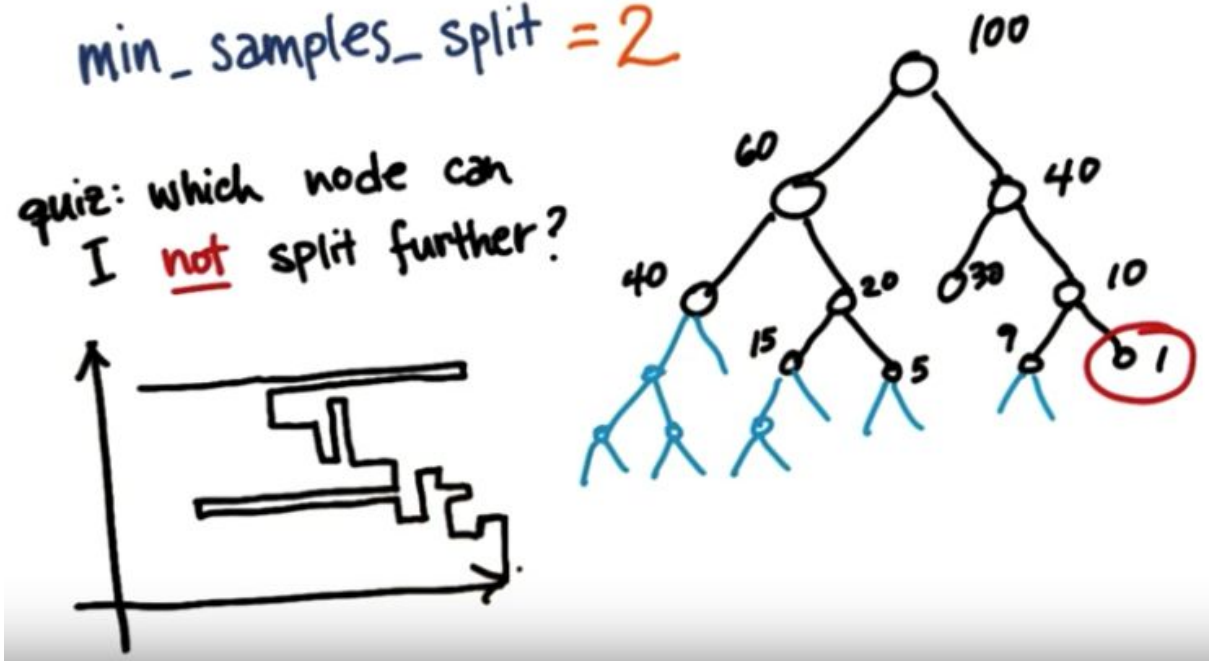
max_features (default=”auto”)

- **{“auto”, “sqrt”, “log2”}**
- **If “sqrt” ,sqrt(n_features) (same as “auto”).**

RandomForest- Classifier- Hyperparameters

min_samples_leaf (default=1)

- **The minimum number of samples required to be at a leaf node.**



RandomForest- Classifier- Hyperparameters

max_leaf_nodes (default=None)

- Grow trees with max_leaf_nodes in best-first fashion. Best nodes are defined as relative reduction in impurity. If None then unlimited number of leaf nodes.

min_impurity_decrease(default=0.0)
A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

bootstrap (default=True)
oob_score (default=False)
n_jobs (default=None)
random_state (default=None)
verbose (default=0)
warm_start(default=False)

RandomForest- Classifier- Hyperparameters

class_weight (default = None)

- {“balanced”, “balanced_subsample”}
- Weights associated with classes in the form {class_label: weight}.
- If not given, all classes are supposed to have weight one. For multi-output problems, a list of dicts can be provided in the same order as the columns of y.

RandomForest- Classifier- Hyperparameters

ccp_alpha (default=0.0)

- Complexity parameter used for Minimal Cost-Complexity Pruning.

max_samples (default=None)

- If bootstrap is True, the number of samples to draw from X to train each base estimator.
- If None (default), then draw X.shape[0] samples.
- If int, then draw max_samples samples.
- If float, then draw max_samples * X.shape[0] samples. Thus, max_samples should be in the interval (0.0, 1.0].

RandomForest- Regressor- Hyperparameters

There are no big differences in regressor of bagging.