# Lecture 22: Combining Different Models for Ensemble Learning – boosting approach

Dr. Huiping Cao

# AdaBoost - Adaptive Boosting

- Adaptive Boosting (**AdaBoost**) is a common implementation of the *boosting* method.
- The original idea was formulated by Robert E. Schapire in 1990. *The strength of weak learnability, R. E. Schapire, Machine Learning, 5(2), 197-227, 1990.*

# AdaBoost - idea

- The ensemble consists of very simple base classifiers (called **weak learners**). These weak learners are slightly better than random guessing. For example, a weak learner can be a decision stump.
  - A **decision stump** is a model consisting of a one-level decision tree. That is, it is a decision tree with one internal node which is immediately connected to the terminal nodes. A decision stump makes a prediction based on the value of just a single input feature.
- Focus on **training samples that are hard to classify** and let the weak learners subsequently learn from misclassified training samples to improve the performance.

# General procedure of boosting

- Draw a random **subset of training samples $\boldsymbol{d_1}$** without replacement from training set to train a weak learner $C_1$.
- Draw a second **random training subset $\boldsymbol{d_2}$** without replacement from training set and add 50 percent of the samples that were previously misclassified to train a weak learner $C_2$.
- Find the training samples $d_3$ in training set $D$, which **$\boldsymbol{C_1}$ and $\boldsymbol{C_2}$ disagree upon**, to train a third weak learner $C_3$.
- Combine the weak learners $C_1$, $C_2$, and $C_3$ via majority voting.

# Discussions

- Theoretically, Boosting can decrease both bias and variance.
- Practically, AdaBoost still has high variance (leads to overfitting).

# AdaBoost - idea

- AdaBoost uses the **complete training dataset** to train the weak learners.
- In each boosting round, we update the **weights** of all the instances. For the correctly predicted samples, their weights are reduced. For the wrongly predicted samples, their weights are increased.
- Combine the three weak learners by a **weighted majority vote**.

# Algorithm pseudocode

1. Set the weight vector **w** to uniform weights where $\sum_i w_i = 1$
2. For *j* in *m* boosting rounds, do the following
    - (a) Train a weighted weak learner: $C_j$ = *train*($\mathbf{X}$, **y**, **w**)
    - (b) Predict class labels $\hat{\mathbf{y}} = predict(C_j, \mathbf{X})$
    - (c) Compute weighted error rate $\boldsymbol{\epsilon} = \mathbf{w} \cdot (\hat{\mathbf{y}} \neq \boldsymbol{y})$ (dot product of w and $\hat{\mathbf{y}} \neq \boldsymbol{y}$)
    - (d) Compute coefficient: $\alpha_j = 0.5 \ln \frac{1-\epsilon}{\epsilon}$
    - (e) Update weights: $\mathbf{w} = \mathbf{w} \times \exp(-\alpha_j \times \hat{\mathbf{y}} \times \mathbf{y})$
    - (f) Normalize weights to sum to 1: $\mathbf{w} = \frac{\mathbf{w}}{\sum_i w_i}$
3. Compute the final predictions
    - $\hat{y} = \left( \sum_{j=1}^{m} \left( \alpha_j \times predict(C_j, X) \right) > 0 \right)$

# Example

| sample indices | $x$ | $y$ | initial weights | $\hat{y}$ | correct? | updated weights |
|---|---|---|---|---|---|---|
| 1 | 1.0 | 1 | 0.1 | 1 | Yes | 0.072 |
| 2 | 2.0 | 1 | 0.1 | 1 | Yes | 0.072 |
| 3 | 3.0 | 1 | 0.1 | 1 | Yes | 0.072 |
| 4 | 4.0 | -1 | 0.1 | -1 | Yes | 0.072 |
| 5 | 5.0 | -1 | 0.1 | -1 | Yes | 0.072 |
| 6 | 6.0 | -1 | 0.1 | -1 | Yes | 0.072 |
| 7 | 7.0 | 1 | 0.1 | -1 | No | 0.167 |
| 8 | 8.0 | 1 | 0.1 | -1 | No | 0.167 |
| 9 | 9.0 | 1 | 0.1 | -1 | No | 0.167 |
| 10 | 10.0 | -1 | 0.1 | -1 | Yes | 0.072 |

# Steps

- 2(c) Compute weighted error rate $\boldsymbol{\epsilon} = \mathbf{w} \cdot (\hat{\mathbf{y}} \neq \boldsymbol{y})$

$$\text{Error rate } \epsilon = (0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1, 0.1)\begin{pmatrix} 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 1 \\ 1 \\ 0 \end{pmatrix} = 0.3$$

- 2(d) Compute coefficient: $\alpha_j = 0.5 \ln \frac{1-\epsilon}{\epsilon}$

$$\alpha_j = 0.5 \ln\left(\frac{1-\epsilon}{\epsilon}\right) \approx 0.424$$

# Steps

- 2(e) Update weights: $\mathbf{w} = \mathbf{w} \times \exp\left(-\alpha_j \times \hat{\mathbf{y}} \times \mathbf{y}\right)$
  - $\hat{\mathbf{y}} \times \mathbf{y}$ is an element-wide multiplication between the y and the $\hat{\mathbf{y}}$ vector.
  - If $\hat{y}_i$ is correct, $\hat{y}_i \times y_i$ has positive sign, the weight value will be decreased
    - $0.1 \times \exp(-0.424 \times 1 \times 1) \approx 0.065$
    - $0.1 \times \exp(-0.424 \times (-1) \times (-1)) \approx 0.065$
  - If $\hat{y}_i$ is incorrect, $\hat{y}_i \times y_i$ has negative sign, the weight value will be increased
    - $0.1 \times \exp(-0.424 \times 1 \times (-1)) \approx 0.153$
    - $0.1 \times \exp(-0.424 \times (-1) \times 1) \approx 0.153$
- 2(f) Normalize weights to sum to 1: $\mathbf{w} = \dfrac{\mathbf{w}}{\sum_i w_i}$
  - $\sum_i w_i = 7 \times 0.065 + 3 \times 0.153 = 0.914$
  - $0.065/0.914 \approx 0.072, 0.153/0.914 \approx 0.167$

# Weight updates

| **error rate $\epsilon$** | **0.2** | **0.3** | **0.5** | **0.7** | **0.9** |
|---|---|---|---|---|---|
| $\alpha_j$ | 0.693147181 | 0.42364893 | 0 | -0.42364893 | -1.0986123 |
| w for correctly predicted instances | 0.05 | 0.065465367 | 0.1 | 0.152752523 | 0.3 |
| w for incorrectly predicted instances | 0.2 | 0.152752523 | 0.1 | 0.065465367 | 0.03333333 |
| w sum $\sum_i w_i$ | 0.95 | 0.916515139 | 1 | 0.916515139 | 1.13333333 |
| Normalized w for correctly predicted instances | 0.052631579 | 0.071428571 | 0.1 | 0.166666667 | 0.26470588 |
| Normalized w for incorrectly predicted instances | 0.210526316 | 0.166666667 | 0.1 | 0.071428571 | 0.02941176 |

- When there are more correctly predicted instances, the updated weights for incorrectly predicted instances are bigger and for correctly predicted instances are smaller.
- On the other hand (more incorrectly predicted instances), the weights are updated in opposite direction.

# Use AdaBoost in scikit-learn library

```
from sklearn.ensemble import AdaBoostClassifier

tree = DecisionTreeClassifier(criterion='entropy',
                              max_depth=1,
                              random_state=1)

ada = AdaBoostClassifier(base_estimator=tree,
                         n_estimators=500,
                         learning_rate=0.1,
                         random_state=1)
```

- **learning_rate**: Learning rate shrinks the contribution of each classifier. There is a trade-off between learning_rate and n_estimators.

# Regression analysis

# Random forest regressor

- A limitation of the **DecisionTreeRegressor** is that it does not capture the continuity and differentiability of the desired prediction.
- **Random forest is an ensemble of multiple decision trees**. It can be understood as the sum of piecewise linear functions, in contrast to the global linear and polynomial regression models.
- Compared with decision tree, it has the following advantages.
  - (1) It usually has a better generalization performance than an individual decision tree due to randomness.
  - (2) Random forests are less sensitive to outliers in the dataset.
  - (3) Do not require much parameter tuning. The only parameter we need is the number of decision trees in the ensemble.

# Random Forest Regressor

*class* 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*)

*class sklearn.ensemble.**RandomForestRegressor**(n_estimators=100, *, criterion='**squared_error**', 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, bootstrap=True, oob_score=False, n_jobs=None, random_state=None, verbose=0, warm_start=False, ccp_alpha=0.0, max_samples=None)*

# Random forest regressor

```
from sklearn.ensemble import RandomForestRegressor

X = df[['MedInc', 'AveRooms']].values
y = df['MEDV'].values

forest = RandomForestRegressor(n_estimators=1000, criterion='squared_error', random_state=1, n_jobs=10)
forest.fit(X, y)
y_pred = forest.predict(X)

error = mean_squared_error(y, y_pred)
r2 = r2_score (y, y_pred)
print('MSE: %.3f, R2:%.3f' % (error, r2))
```

```
MSE: 0.048, R2:0.940
```

This is much better than other regression results.

# AdaBoost regressor

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

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

# References


- Chapter 7,  Sebastian Raschka and Vahid Mirjalili: Python Machine Learning (Machine learning and deep learning with Python, scikit-learn, and TensorFlow), 3rd Edition.
- Other ensemble approaches: https://scikit-learn.org/stable/modules/ensemble.html