



Data Science: Ensemble Learning

Won Kim
2022



Roadmap: End-to-End Process

- 1. Objective Setting
- 2. Data Curation
- 3. Data Inspection
- 4. Data Preparation
- 5. Data Analysis
- 6. **Evaluation**
- 7. Deployment



Roadmap: Evaluation

- Motivation
- Evaluation Methods
- **Ensemble Learning**
- Evaluation Metrics



Acknowledgments

- 김원의 SW중심세상 -- https://blog.naver.com/wonkim_sw-world
- <http://www.inf.u-szeged.hu/~tothl/ML/10.%20Ensemble%20learning.ppt>
- <http://www2.cs.uh.edu/~ceick/ML/Topic12.ppt>



Roadmap: Ensemble Learning

- Overview
- Bagging (and Random Forest)
- Boosting



Note

- Ensemble learning methods apply to supervised learning algorithms only; that is, to classifiers and regressors.
- They do not apply to unsupervised learning algorithms, such as clustering.
- These are to improve the prediction accuracy of supervised learning.
- Although the topic is usually covered under “Evaluation Metrics”, it may be regarded as an extension of the supervised learning algorithms.

Ensemble

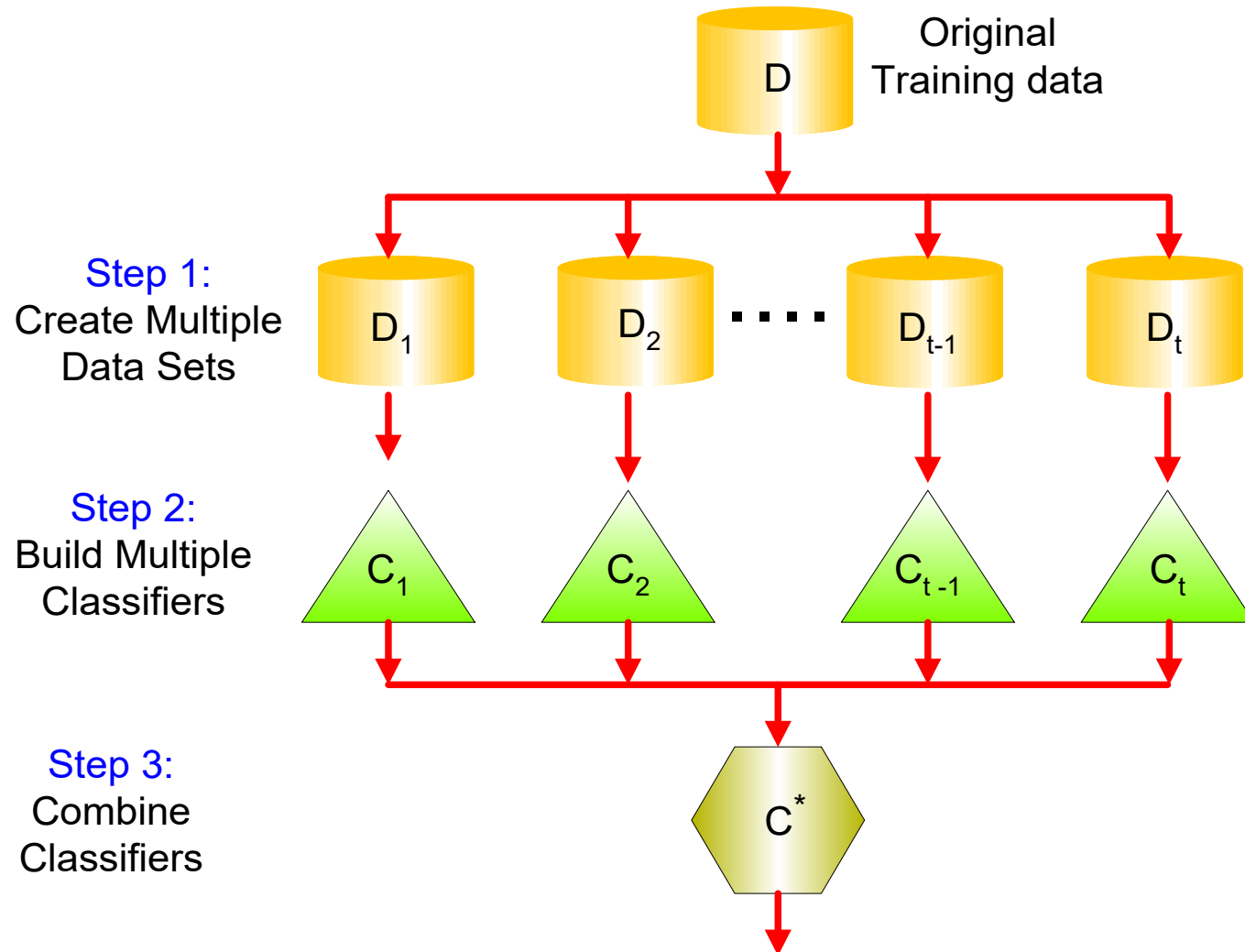




Ensemble of Classifiers

- An ensemble of classifiers is a set of classifiers whose individual decisions are combined in some way to classify new examples (Thomas Dietterich, 2000)
- Base learner
 - Arbitrary learning algorithm which could be used on its own
- Ensemble learner
 - A learning algorithm composed of a set of base learners. The base learners may be organized in some structure.

General Idea



Fixed Combination Rules

| Rule | Fusion function $f(\cdot)$ |
|--------------|---|
| Sum | $y_i = \frac{1}{L} \sum_{j=1}^L d_{ji}$ |
| Weighted sum | $y_i = \sum_j w_j d_{ji}, w_j \geq 0, \sum_j w_j = 1$ |
| Median | $y_i = \text{median}_j d_{ji}$ |
| Minimum | $y_i = \min_j d_{ji}$ |
| Maximum | $y_i = \max_j d_{ji}$ |
| Product | $y_i = \prod_j d_{ji}$ |

| | C_1 | C_2 | C_3 |
|---------|-------|-------------|-------|
| d_1 | 0.2 | 0.5 | 0.3 |
| d_2 | 0.0 | 0.6 | 0.4 |
| d_3 | 0.4 | 0.4 | 0.2 |
| Sum | 0.2 | 0.5 | 0.3 |
| Median | 0.2 | 0.5 | 0.4 |
| Minimum | 0.0 | 0.4 | 0.2 |
| Maximum | 0.4 | 0.6 | 0.4 |
| Product | 0.0 | 0.12 | 0.032 |



Why Does It Work?

- Suppose there are 25 base classifiers
 - Each classifier has error rate, $\varepsilon = 0.35$
 - Assume classifiers are independent
 - Probability that the ensemble classifier makes a wrong prediction:

$$\sum_{i=13}^{25} \binom{25}{i} \varepsilon^i (1 - \varepsilon)^{25-i} = 0.06$$

note:
combination

$$\binom{n}{k} = C(n, k) = \frac{n!}{(n - k)!k!}$$



Rationale for an Ensemble

- Different learners use different
 - algorithms
 - hyperparameters
 - training sets
 - and/or (solve) different subproblems
- They each make different contributions to classification




















































Main Challenge

- The main challenge is to **obtain base models which make different kinds of errors** (not to obtain highly accurate base models).
- The individual classifiers composing an ensemble must be accurate and diverse:
 - An accurate classifier is one that has an error rate better than random (50%) when guessing new examples
 - Two classifiers are diverse if they make different errors on new data points.

Example – ensemble of classifier outputs

- Predictions of weather from 5 different classifiers – each with errors
- Combining the outputs of the classifiers results in perfect (^^) prediction.

| Reality |  |  |  |  |  |  |  |
|---------|---|---|--|---|---|---|---|
| 1 |  |  |  |  |  |  |  |
| 2 |  |  |  |  |  |  |  |
| 3 |  |  |  |  |  |  |  |
| 4 |  |  |  |  |  |  |  |
| 5 |  |  |  |  |  |  |  |
| Combine |  |  |  |  |  |  |  |



Ensemble Approaches (1/2)

- Combine different learning algorithms ("hybridization")
 - (e.g.) We can train a decision tree, k-nearest neighbors, GMM (Gaussian Mixture Modeling), SVM (Support Vector Machine), etc. over the same dataset, and then combine their outputs
- Use the same algorithm over the same dataset, but with different weights over the data instances



Ensemble Approaches (2/2)

- Combine the same learning algorithm trained over different subsets of the training data
 - We can also try using different subsets of the features
 - or different subsets of the target classes (in case of a multi-classification task, or when there are a lot of classes)



Ensemble Techniques

- Aggregation Methods
 - voting
 - stacking
- Ensemble Methods
 - cascading
 - bagging
 - random forest
 - AdaBoost
 - gradient boosting

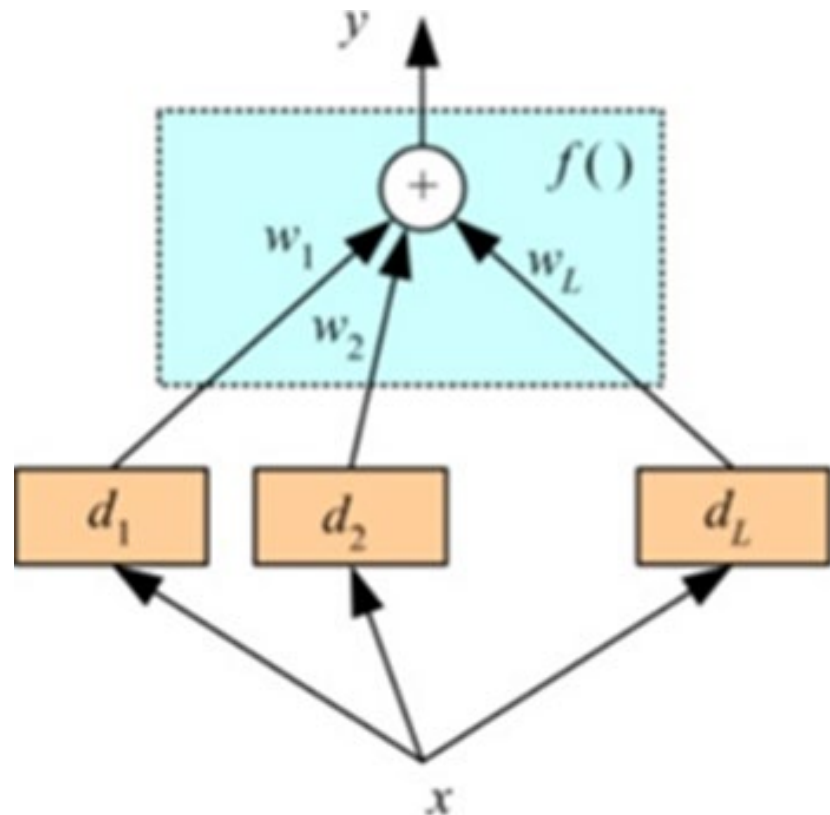


Aggregation Methods

- Voting
 - When the output is a class label (categorical)
 - take majority voting over all base classifiers
 - take weighted majority voting (weight each base classifier by its reliability)
 - When the output is numeric (e.g. a probability estimate for each class c_i)
 - combine the d_j scores by taking their (weighted) mean, product, minimum, maximum, ...
- Stacking
 - train yet another classifier on the output values of the base classifiers

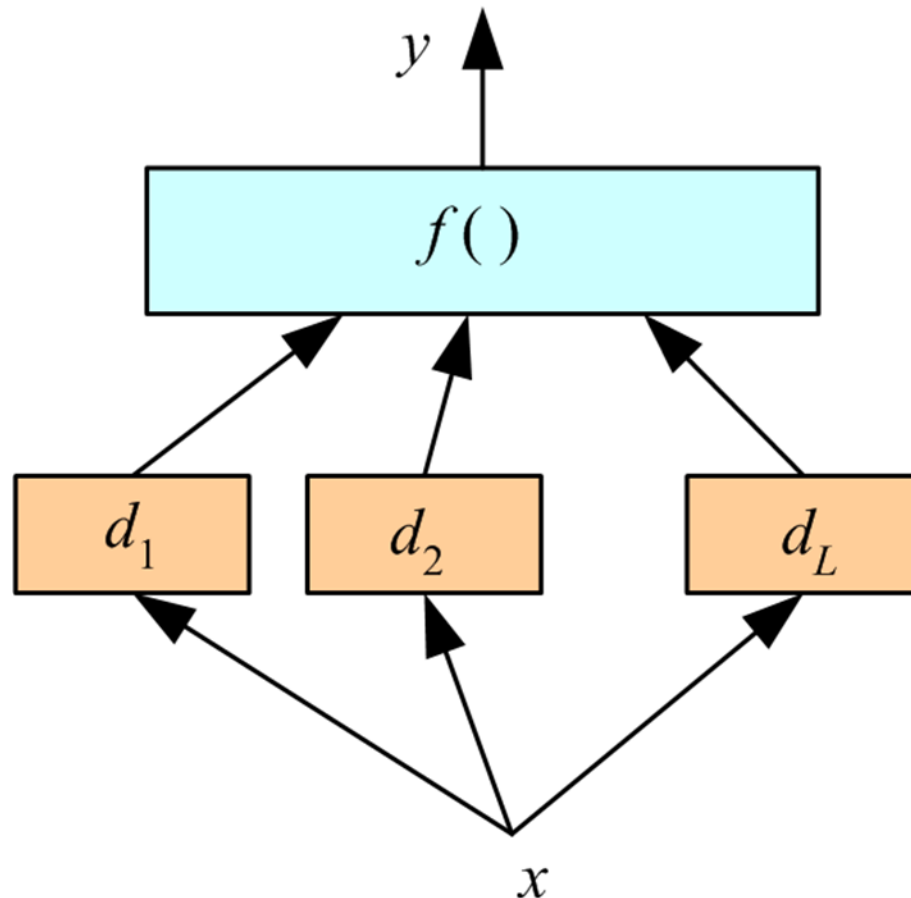
Voting on the Outputs of Classifiers

- d_i : output from classifier i
- w_i : weight assigned to output d_i
- y : ensemble output



Stacking (1/2)

- In stacked aggregation, the combiner $f()$ is another learner (classifier) as in voting.



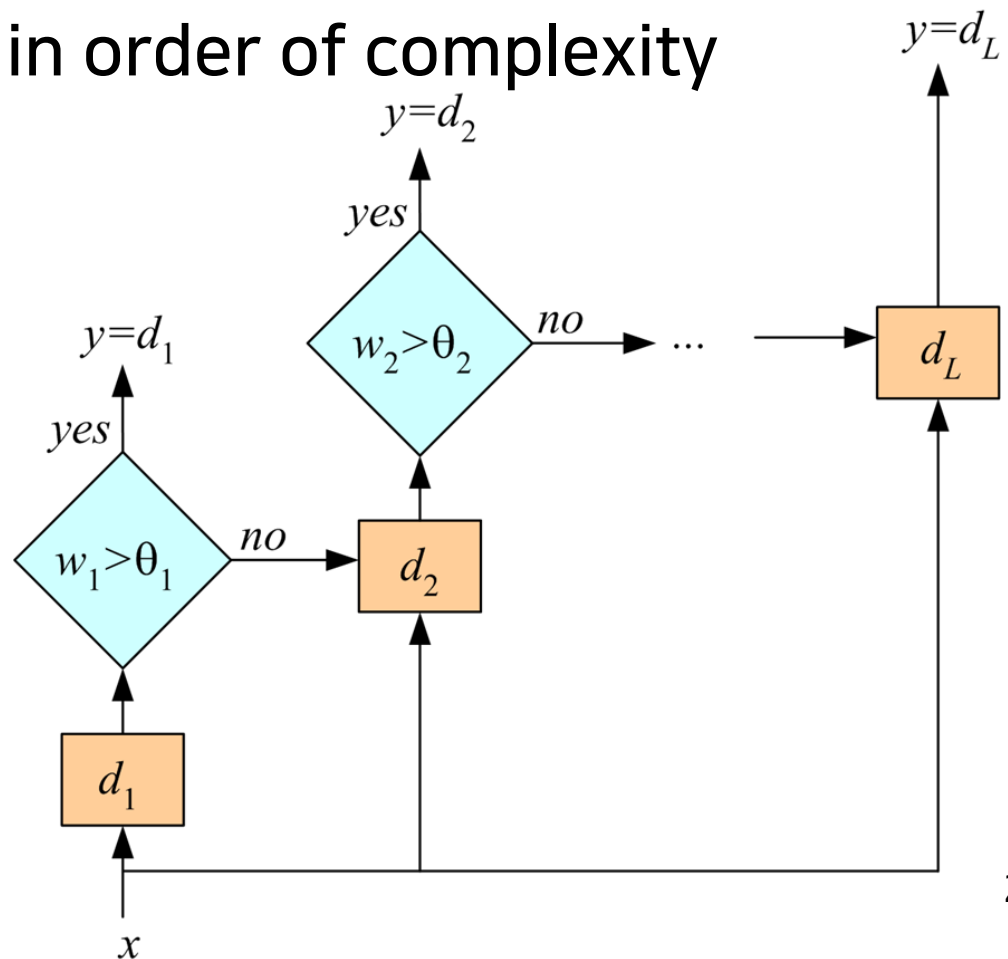


Stacking (2/2)

- The combiner should learn how the base learners make errors.
 - Stacking is a means of estimating and correcting for the biases of the base learners.
 - The combiner should be trained on data unused in training the base learners

Cascading (1/2)

- Use d_j only if the preceding classifiers outputs are not confident (i.e., below a certain threshold θ)
- Cascade learners in order of complexity





Cascading (2/2)

- Cascading is a multistage method, and we use d_j only if all preceding learners are not confident.
- Associated with each learner is a confidence w_j such that we say d_j is confident of its output and can be used if $w_j > \theta_j$ (the threshold).
- Confident: Misclassifications as well as the instances for which the posterior probability is not high enough.
- Important: The idea is that an early simple classifier handles the majority of instances, and a more complex classifier is used only for a small percentage, so does not significantly increase the overall complexity.



Bias and Variance in Ensemble Learning

- Minimize variance (use many training datasets)
 - Bagging
 - Random forests
- Minimize bias (use refined classifiers)
 - Boosting
 - Functional gradient descent
 - Ensemble selection



Roadmap: Ensemble Learning

- Overview
- Bagging (and Random Forest)
- Boosting



Bootstrap Resampling

- Create M different training datasets from an original training dataset with n samples.
- Bootstrap resampling takes random samples from the original dataset with replacement.
- As the M training sets are different, the result of the training over these sets will also be more or less different, regardless of the training algorithm we use
 - Works better with unstable learners (e.g. decision trees, neural networks)
 - Not really effective with stable learners (e.g. k-NN, SVM)
 - ** A stable learner is one for which the prediction does not change much when the training data is modified slightly. (e.g.) It produces similar predictions with 1000-element and 999-element training sets.



Bagging

- Bagging = Bootstrap + Aggregating
- Invented by Leo Breiman (an inventor of CART decision tree)
- It uses bootstrap resampling to generate M different training datasets from the original training dataset
- On the M training sets it trains M base learners (classifiers)
- During testing, it aggregates the M learners by taking their average (using uniform weights for each classifier), or by majority voting

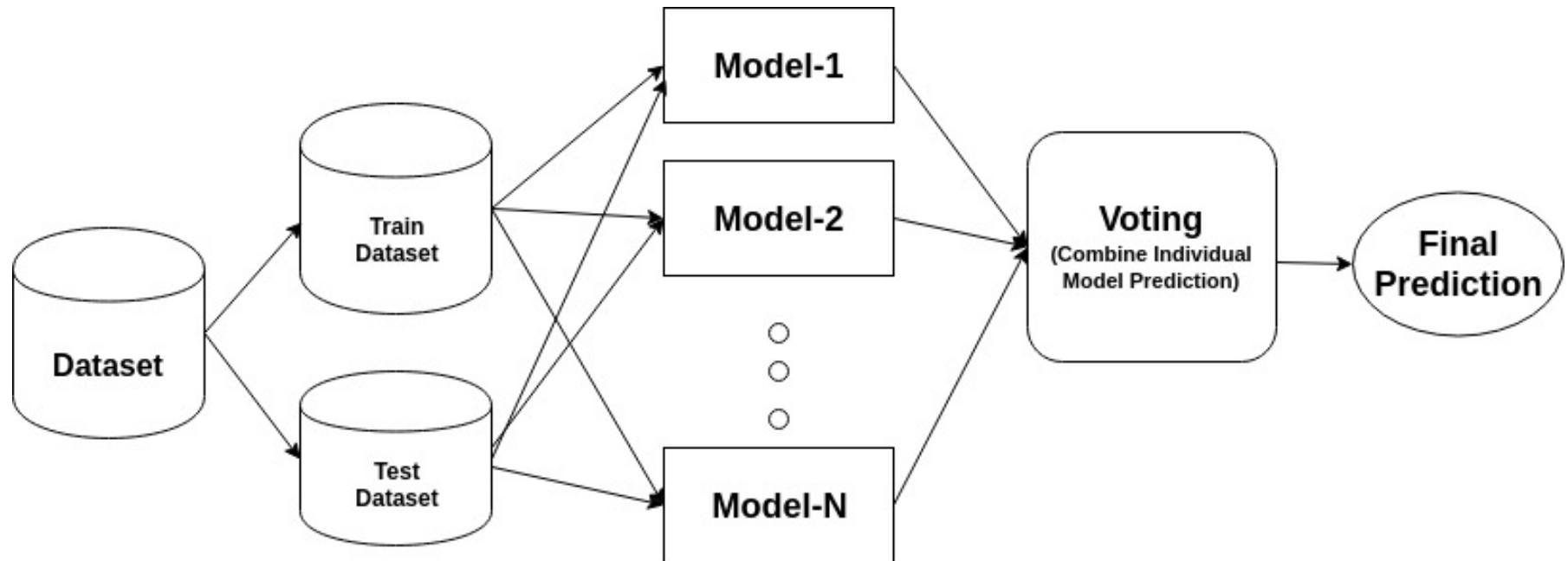


Bagging Algorithm

Algorithm 5.6 Bagging Algorithm

- 1: Let k be the number of bootstrap samples.
 - 2: for $i = 1$ to k do
 - 3: Create a bootstrap sample of size n , D_i .
 - 4: Train a base classifier C_i on the bootstrap sample D_i .
 - 5: end for
 - 6: $C^*(x) = \arg \max_y \sum_i \delta(C_i(x) = y)$, $\{\delta(\cdot) = 1$ if its argument is true, and 0 otherwise. $\}$
-

Bagging: Concept Visualization





Illustration

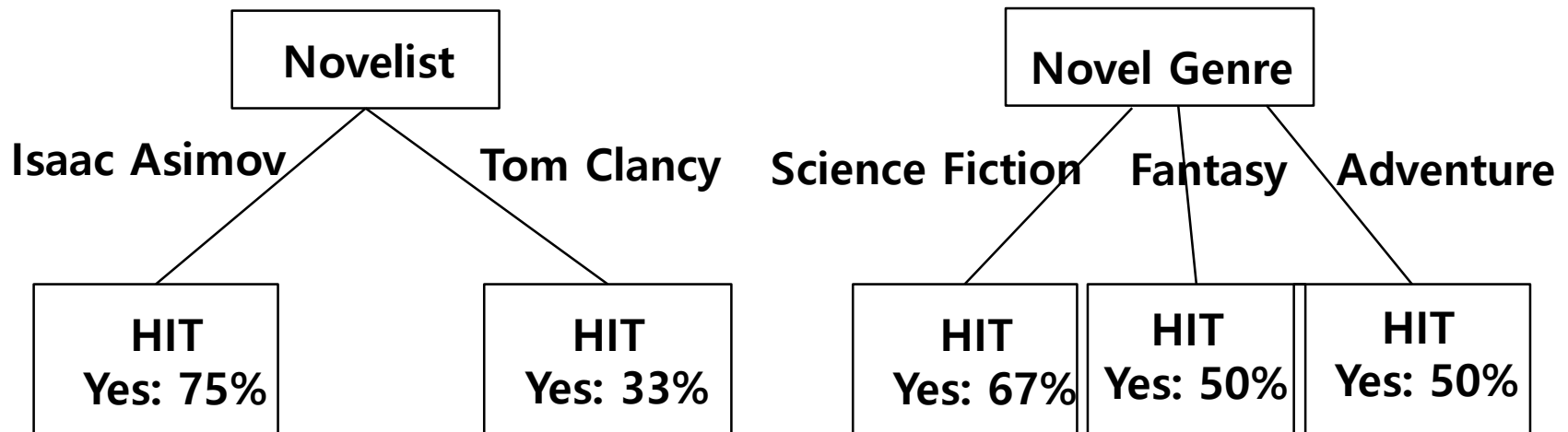
- Sampling with replacement

| | | | | | | | | | | |
|-------------------|---|---|----|----|---|---|----|----|---|----|
| Original Data | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| Bagging (Round 1) | 7 | 8 | 10 | 8 | 2 | 5 | 10 | 10 | 5 | 9 |
| Bagging (Round 2) | 1 | 4 | 9 | 1 | 2 | 3 | 2 | 7 | 3 | 2 |
| Bagging (Round 3) | 1 | 8 | 5 | 10 | 5 | 5 | 9 | 6 | 3 | 7 |

- Build a classifier on each bootstrap sample.
- Each sample has probability $(1 - 1/n)^n$ of being selected for testing.

Decision Stump (Remember?)

- A decision stump is a decision tree with only the root node (and the leaf nodes)
- We saw this earlier (see the diagram below).
- This is a very “weak” decision tree (classifier).
- But is useful for “Ensemble Learning”.



Bagging Example

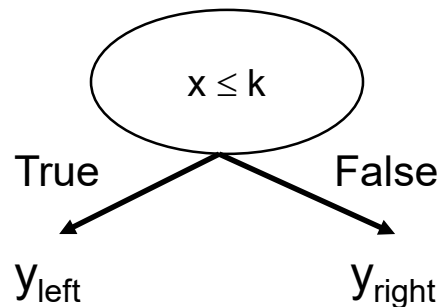
(using decision stumps)

- Consider a 1-dimensional dataset:

Original Data:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

- Classifier is a decision stump
 - decision rule: $x \leq k$ versus $x > k$
 - Split point k is chosen based on entropy.



Ensemble of Decision Stumps



How Decision Stumps Are Used

test dataset

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

bagging round-1 dataset and training

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.2 | 0.2 | 0.3 | 0.4 | 0.4 | 0.5 | 0.6 | 0.9 | 0.9 |
| y | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 |

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

for 0.1, 0.2, 0.3, $y=1$.

for 0.4, 0.5, 0.6, 0.9, 1.0, $y=-1$

bagging round-1 prediction (against test dataset)

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|----|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |



Notes

- The dataset in each bagging round (bagging dataset) is used only for training. It should not be split into “train set” and “test set”.
 - The testing of the trained classifier (decision stump) is done against a separate test set (which in this example is the original dataset).
- The bagging dataset does not have to be the same size as the original dataset.
 - If the original dataset has N samples, the bagging dataset may each have n ($n < N$) samples.

Bagging Walkthrough (1/2)

red bar | indicates the split value for x

Bagging Round 1:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.2 | 0.2 | 0.3 | 0.4 | 0.4 | 0.5 | 0.6 | 0.9 | 0.9 |
| y | 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 |

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

Bagging Round 2:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|---|---|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.5 | 0.9 | 1 | 1 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 |

$x \leq 0.7 \rightarrow y = 1$

$x > 0.7 \rightarrow y = 1$

Bagging Round 3:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.4 | 0.5 | 0.7 | 0.7 | 0.8 | 0.9 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

Bagging Round 4:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.1 | 0.2 | 0.4 | 0.4 | 0.5 | 0.5 | 0.7 | 0.8 | 0.9 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

$x \leq 0.3 \rightarrow y = 1$

$x > 0.3 \rightarrow y = -1$

Bagging Round 5:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|---|---|---|
| x | 0.1 | 0.1 | 0.2 | 0.5 | 0.6 | 0.6 | 0.6 | 1 | 1 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

$x \leq 0.35 \rightarrow y = 1$

$x > 0.35 \rightarrow y = -1$

Bagging Walkthrough (1/2) cont'd

Bagging Round 6:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.2 | 0.4 | 0.5 | 0.6 | 0.7 | 0.7 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

$x \leq 0.75 \rightarrow y = -1$

$x > 0.75 \rightarrow y = 1$

Bagging Round 7:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.4 | 0.4 | 0.6 | 0.7 | 0.8 | 0.9 | 0.9 | 0.9 | 1 |
| y | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 | 1 | 1 |

$x \leq 0.75 \rightarrow y = -1$

$x > 0.75 \rightarrow y = 1$

Bagging Round 8:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.5 | 0.5 | 0.5 | 0.7 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

$x \leq 0.75 \rightarrow y = -1$

$x > 0.75 \rightarrow y = 1$

Bagging Round 9:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|---|---|
| x | 0.1 | 0.3 | 0.4 | 0.4 | 0.6 | 0.7 | 0.7 | 0.8 | 1 | 1 |
| y | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

$x \leq 0.75 \rightarrow y = -1$

$x > 0.75 \rightarrow y = 1$

Bagging Round 10:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.1 | 0.1 | 0.1 | 0.3 | 0.3 | 0.8 | 0.8 | 0.9 | 0.9 |
| y | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

$x \leq 0.05 \rightarrow y = 1$

$x > 0.05 \rightarrow y = 1$



Bagging Walkthrough (1/2) cont'd

- Summary of Training Results

| Round | Split Point | Left Class | Right Class |
|-------|-------------|------------|-------------|
| 1 | 0.35 | 1 | -1 |
| 2 | 0.7 | 1 | 1 |
| 3 | 0.35 | 1 | -1 |
| 4 | 0.3 | 1 | -1 |
| 5 | 0.35 | 1 | -1 |
| 6 | 0.75 | -1 | 1 |
| 7 | 0.75 | -1 | 1 |
| 8 | 0.75 | -1 | 1 |
| 9 | 0.75 | -1 | 1 |
| 10 | 0.05 | 1 | 1 |

Bagging Walkthrough (2/2)

- The predictions are organized in a 10x10 matrix (10 classifiers x 10 test samples)
- Use majority vote to determine the predicted class of the ensemble classifier.

| Round | x=0.1 | x=0.2 | x=0.3 | x=0.4 | x=0.5 | x=0.6 | x=0.7 | x=0.8 | x=0.9 | x=1.0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 4 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 5 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| 6 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 7 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 8 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 9 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 10 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| Sum | 2 | 2 | 2 | -6 | -6 | -6 | -6 | 2 | 2 | 2 |
| Sign | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

Predicted
Class



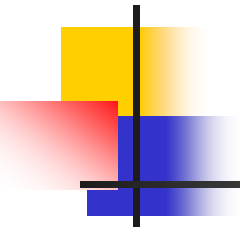
In-Class Exercises

- (1) Change the example original (test) dataset by replacing $(x=0.8, y=1)$ to $(x=0.8, y=-1)$.

Original Data:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

- (2) Change the k value 0.75 to 0.85 for bagging rounds 6,7,8,9. (Use the above modified test dataset.)
- Reuse the same 10 bagging round datasets as the example, and determine the predicted class of the ensemble classifier.



Random Forest



Random Forest

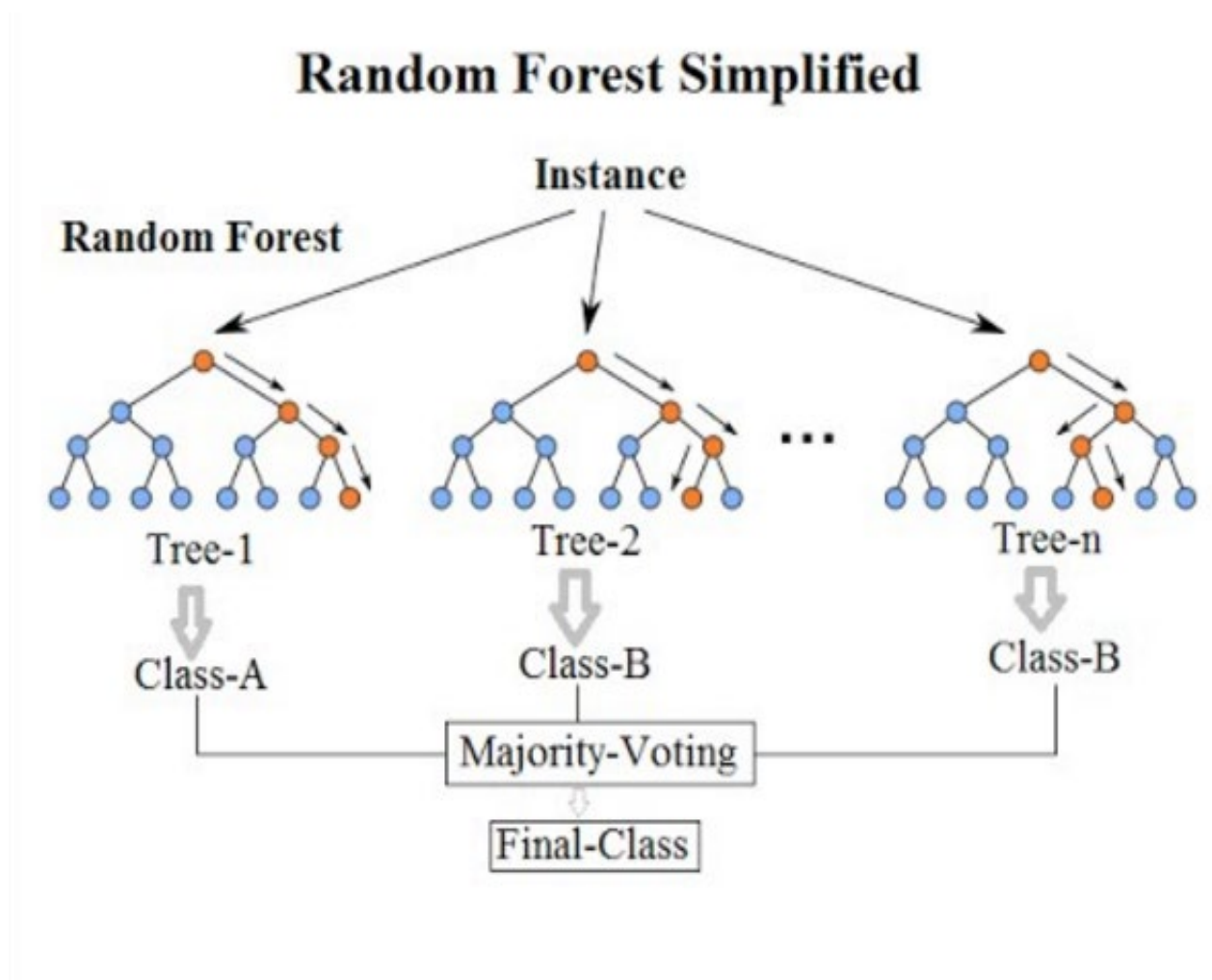
- Invented by Tin Kam Ho in 1995, followed by Leo Breiman, Adele Cutler
- A random forest is an ensemble of decision trees.
- It is created to use a set of random decision trees as weak base learners, and turn them into a strong learner.
- Why decision trees as base learners?
 - (Small) decision trees are not really efficient classifiers.
 - But their training is very simple and fast, so it is very easy to create an ensemble learner from a huge set of (small) decision trees.



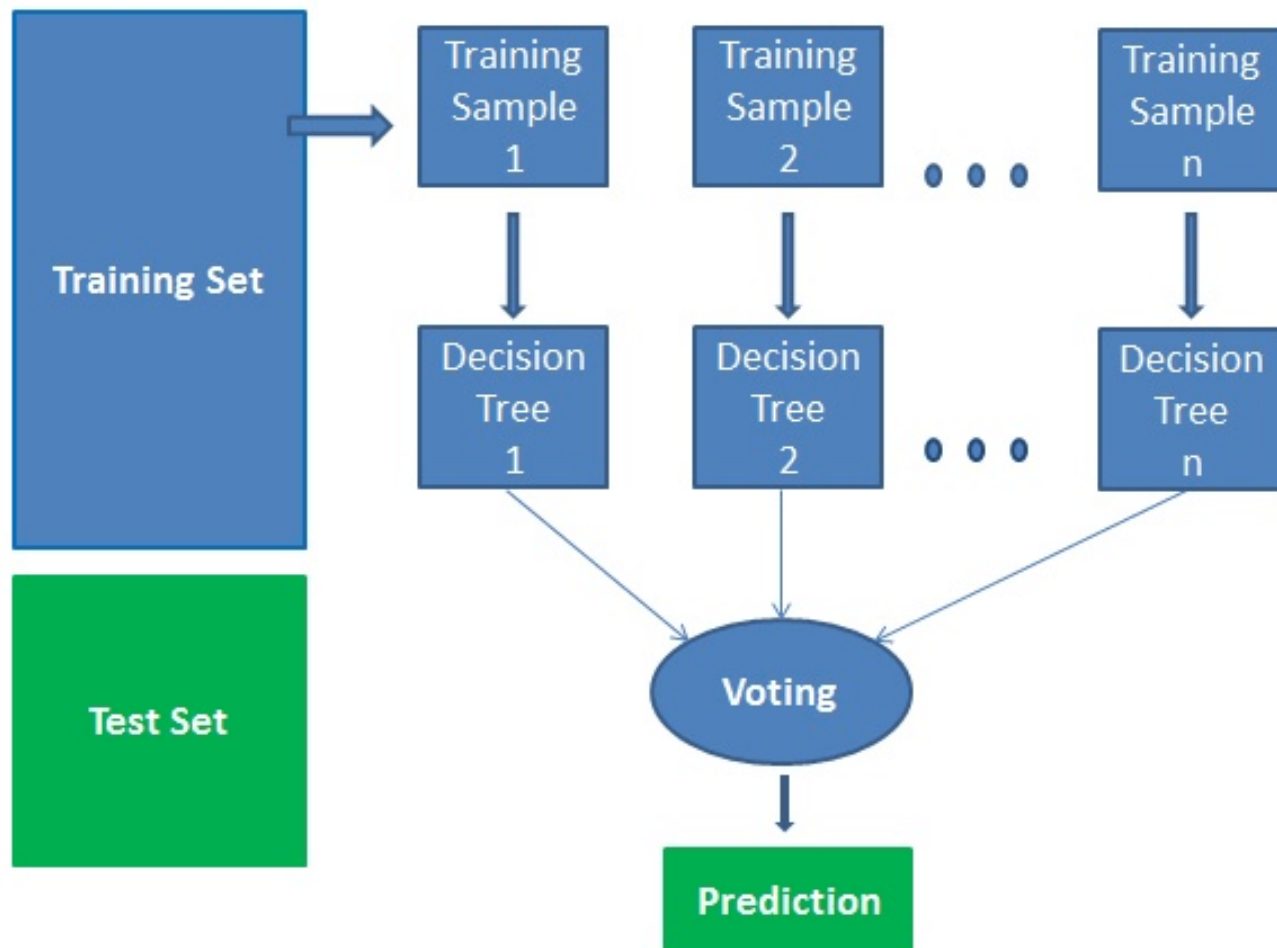
Creating a Random Forest

- How to produce different decision trees at different runs (where the 'random' in the name 'random forest' comes from)
 - Use bootstrap resampling to create M random subsets of training data for each decision tree.
 - Use a random subset of the features (not all features) for each decision tree

Random Forest: Concept Visualization



Random Forest Training and Testing





Random Forest Algorithm

- Input: training data set, T, m
 - Choose T , the number of decision trees to build.
 - Choose m , the number of variables used to split each node ($m \ll M$, the total number of input variables).
- Build T decision trees. For each tree,
 - Construct a bootstrap sample of size n from the original dataset.
 - When building a tree, at each node select m variables at random to find the best split.
 - Build the tree to the maximum extent, without pruning.
- To classify a new data, collect votes from every tree in the forest, and use majority voting for the predicted class label.



Hyperparameters of Random Forest

- `n_estimators` = number of trees in the ensemble.
 - default 100
- `max_features` = maximum number of features considered when splitting a node.
- `criterion` = the function used to evaluate the quality of a split.
 - gini (default), entropy
- `max_depth` = maximum number of levels allowed in each tree.
- `min_samples_leaf` = minimum number of samples which can be stored in a tree leaf.
 - default 1
- `min_samples_split` = minimum number of samples necessary in a node to cause node splitting.
 - default 2



Grid Search with Scikit-Learn

- <https://stackabuse.com/cross-validation-and-grid-search-for-model-selection-in-python/>

```
from sklearn.model_selection import GridSearchCV
```

```
# define grid parameters
```

```
grid_param = { 'n_estimators': [100, 300, 500, 800,  
1000], 'criterion': ['gini', 'entropy'], 'bootstrap': [True,  
False] }
```

- * Grid search will check 20 combinations of parameters: (5 x 2 x 2 = 20).



Initialize and Train the Grid Search Class

(omitted) load dataset, prepare train set and test set

initialize the GridSearchCV class

```
gd_sr = GridSearchCV(estimator=classifier,  
param_grid=grid_param, scoring='accuracy', cv=5, n_jobs=-  
1)
```

- * n_jobs=-1 means “use all available computing power”

train the class

```
gd_sr.fit(X_train, y_train)
```

- The method will execute 100 times (20 combinations of parameters x 5-fold cross validation)



Check the Result

```
best_parameters = gd_sr.best_params_  
print(best_parameters)
```

output

```
{'bootstrap': True, 'criterion': 'gini', 'n_estimators': 1000}
```

- * best_params_ is an attribute of the GridSearchCV object
- * The output means the highest accuracy is achieved when the n_estimators is 1000, bootstrap is True and criterion is "gini".



Find the Accuracy Obtained Using the Best Parameters

```
best_result = gd_sr.best_score_  
print(best_result)
```



Scikit-Learn Support for Random Forest (to prepare for hypertuning)

- <https://towardsdatascience.com/random-forest-in-python-24d0893d51c0>
- Example: Predicting the maximum temperature for tomorrow in a city, using one year of past weather data.
- ** Let us assume that model has been built (details are in the above reference).
- ** We will only show how you may improve the model using some Scikit-learn classes.



Take Just One Tree and Save It as Image

```
# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot

# Pull out one tree from the forest
tree = rf.estimators_[5]

# Import tools needed for visualization
from sklearn.tree import export_graphviz
import pydot

# Pull out one tree from the forest
tree = rf.estimators_[5]

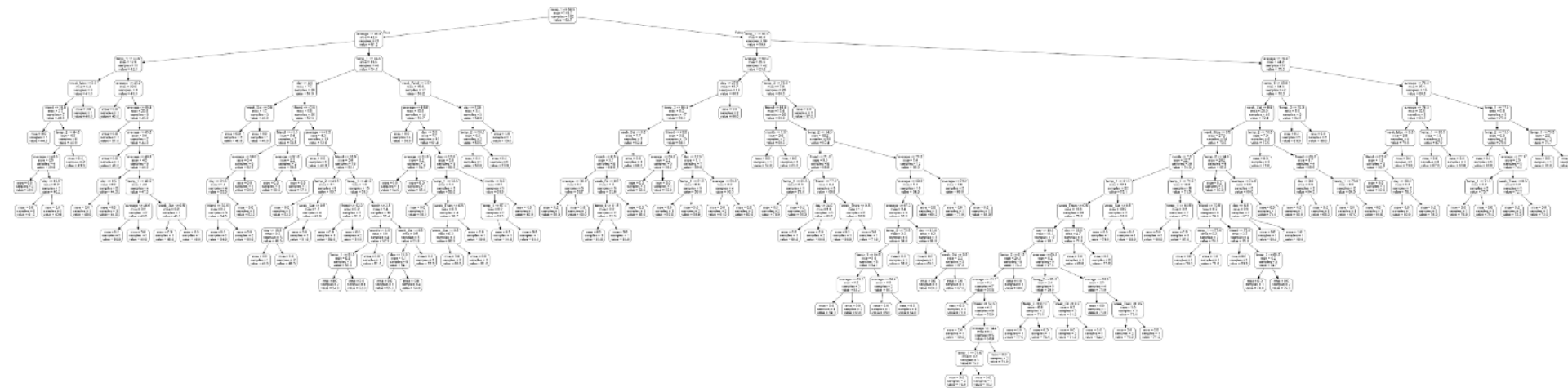
# Export the image to a dot file
export_graphviz(tree, out_file = 'tree.dot', feature_names = feature_list, rounded
= True, precision = 1)

# Use dot file to create a graph
(graph, ) = pydot.graph_from_dot_file('tree.dot')

# Write graph to a png file
graph.write_png('tree.png')
```

Single Decision Tree in the Forest

(* too much *)





Limit the Tree Depth to 3

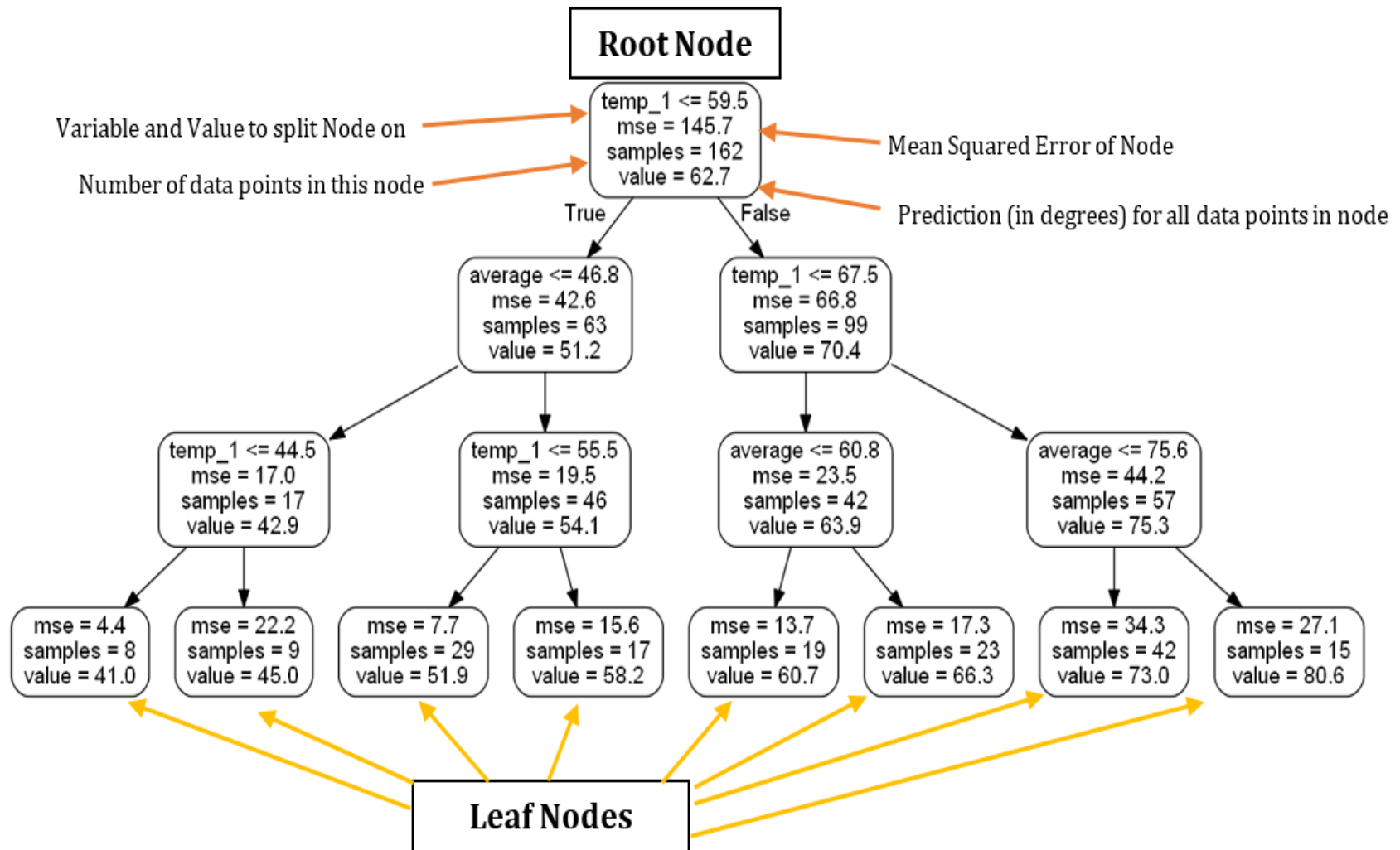
```
# Limit depth of tree to 3 levels
rf_small = RandomForestRegressor(n_estimators=10, max_depth = 3)
rf_small.fit(train_features, train_labels)

# Extract the small tree
tree_small = rf_small.estimators_[5]

# Save the tree as a png image
export_graphviz(tree_small, out_file = 'small_tree.dot',
    feature_names = feature_list, rounded = True, precision = 1)
(graph,) = pydot.graph_from_dot_file('small_tree.dot')
graph.write_png('small_tree.png');
```

Result: One Decision Tree of Depth 3

(use this to visually determine most useful features)





Select the Most Useful Features

```
# Get numerical feature importances
importances = list(rf.feature_importances_)

# List of tuples with variable and importance
feature_importances = [(feature, round(importance, 2)) for
feature, importance in zip(feature_list, importances)]

# Sort the feature importances by most important first
feature_importances = sorted(feature_importances, key =
lambda x: x[1], reverse = True)

# Print out the feature and importances
[print('Variable: {:20} Importance: {}'.format(*pair)) for pair in
feature_importances];
```



Result

(* for simplicity, the full set of features not shown here *)

Variable: temp_1 Importance: 0.7 # temperature one day before

Variable: average Importance: 0.19 # temperature historical average

Variable: day Importance: 0.03

Variable: temp_2 Importance: 0.02

Variable: friend Importance: 0.02

Variable: month Importance: 0.01

Variable: year Importance: 0.0

Variable: week_Fri Importance: 0.0

Variable: week_Mon Importance: 0.0

Variable: week_Sat Importance: 0.0

Variable: week_Sun Importance: 0.0

Variable: week_Thurs Importance: 0.0

Variable: week_Tues Importance: 0.0

Variable: week_Wed Importance: 0.0



Roadmap: Ensemble Learning

- Overview
- Bagging (and Random Forest)
- **Boosting**



Acknowledgments

- <https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464>
- <https://sefiks.com/2018/10/04/a-step-by-step-gradient-boosting-decision-tree-example/>
- <https://hackernoon.com/boosting-algorithms-adaboost-gradient-boosting-and-xgboost-f74991cad38c>



Boosting

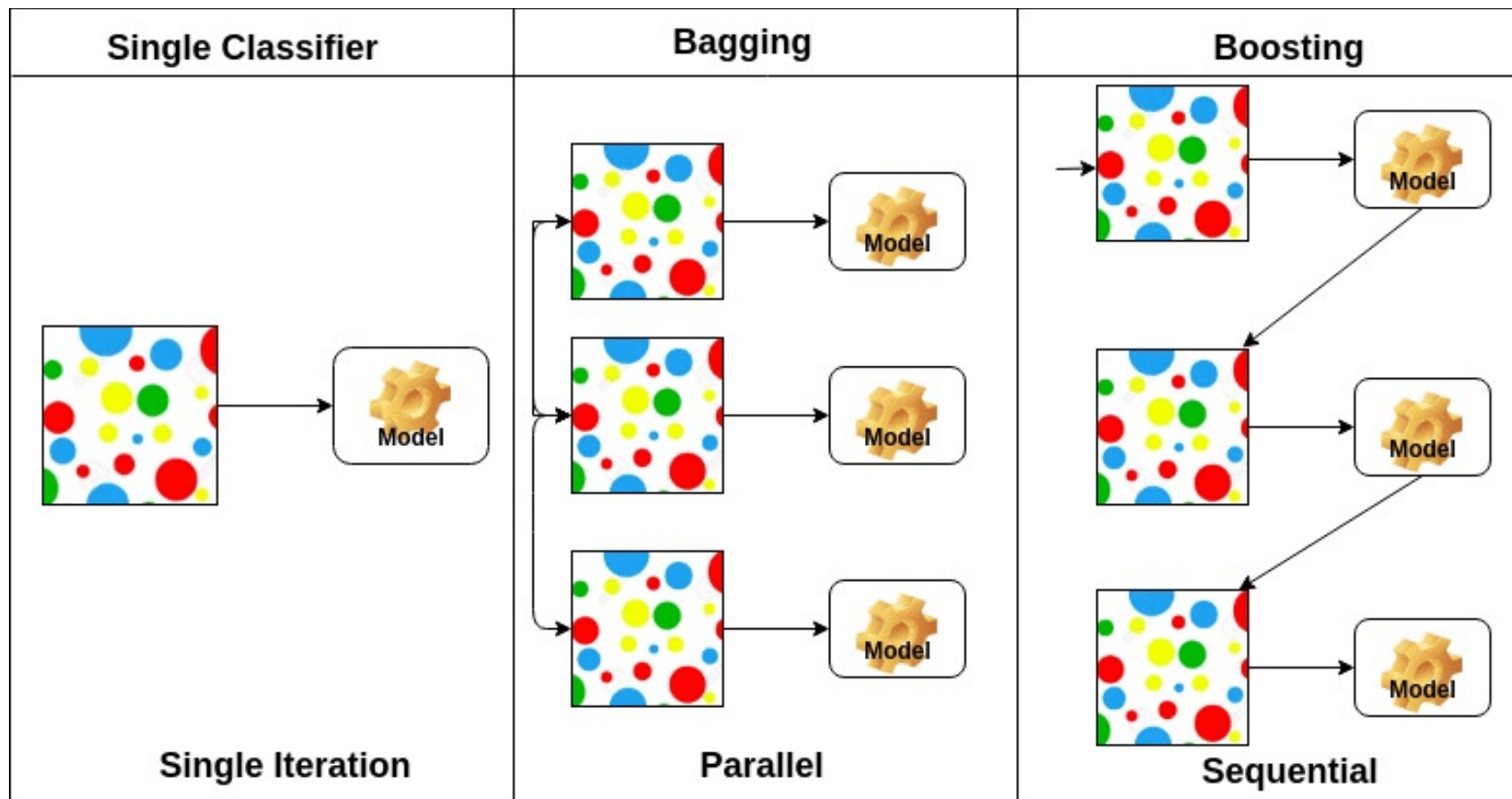
- AdaBoost
- Gradient Boosting
- (variants of gradient boosting)
 - XGBoost (eXtreme Gradient Boost)
 - LightGB
 - Catboost
- ** For all of these, the default base classifier is the decision tree.
- ** Boosting classifiers are very powerful and popular.
 - They win machine learning competitions.



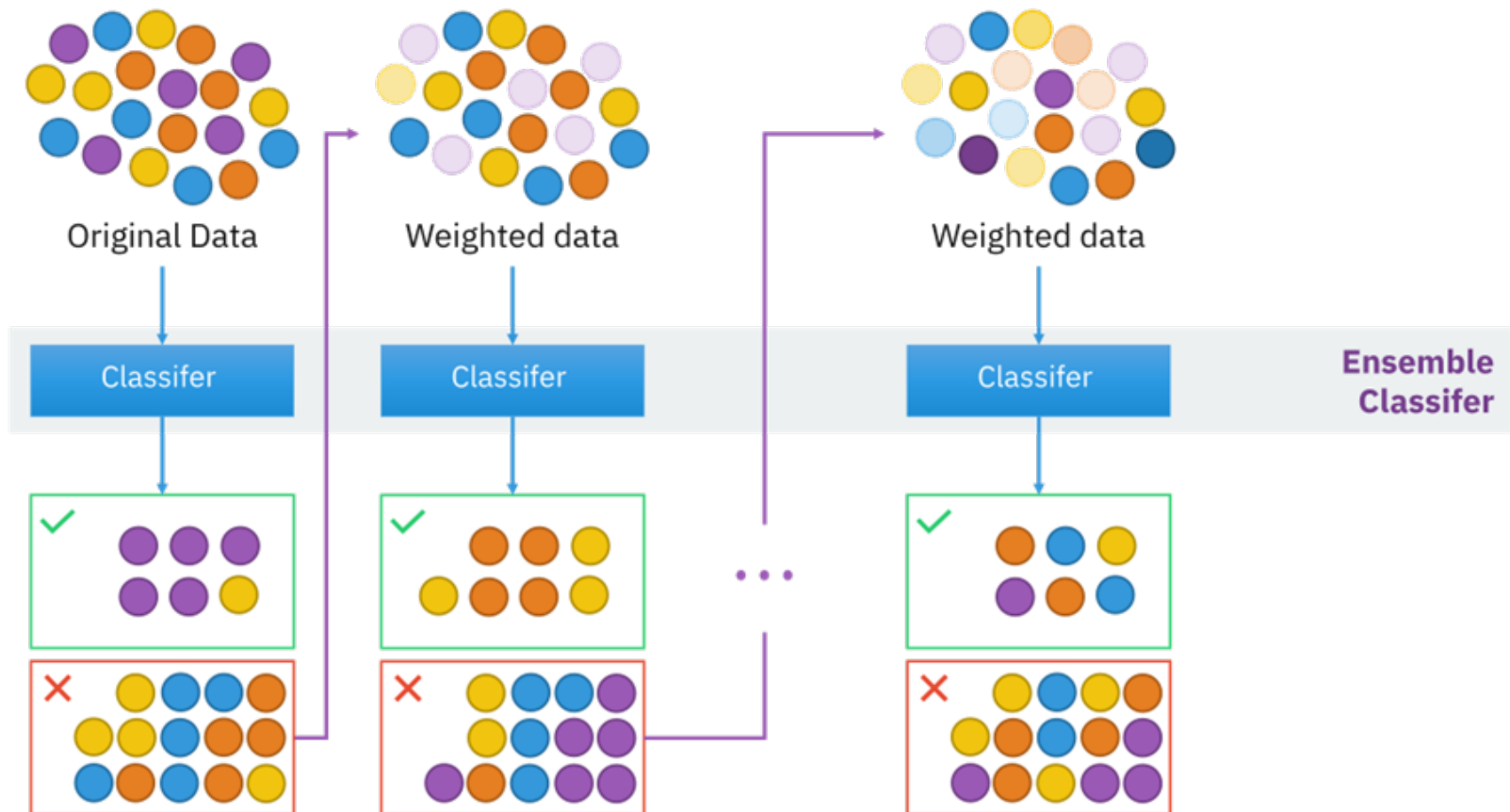
Bagging (and Random Forest) vs. Boosting

- Bagging
 - Simply takes a majority voting of the M base learners
- Boosting
 - Adds one base learner at a time to the ensemble
 - Each base learner attempts to correct the error (misclassification) of the previous base learner, by 'boosting' the weights assigned to misclassified data samples.

Bagging vs. Boosting: Visualization



Boosting: Visualization





Roadmap: Boosting

- AdaBoost
- Gradient Boosting
- XGBoost



AdaBoost

- Invented by Yoav Freund and Robert Schapire in 1996
 - Received the Goedel Prize for the work
- Builds many weak base learners (classifiers) one at a time, ultimately leading to a strong learner.
 - The weak classifiers are decision stumps
- Updates the sample weights of all samples at every iteration by increasing the weights for misclassified samples and decreasing the weights for correctly classified samples
- The final prediction is a weighted average of all the base learners, where more weights are given to stronger learners



AdaBoost Algorithm

D_1 = initial dataset with equal weights

for $i=1$ to k *do*

 Learn new classifier C_i

 Compute α_i (classifier's importance)

 Update sample weights

 Create new training set D_{i+1} (using weighted sampling)

end for

Construct ensemble which uses C_i weighted by α_i ($i=1,k$)

Math Behind AdaBoost (1/2)

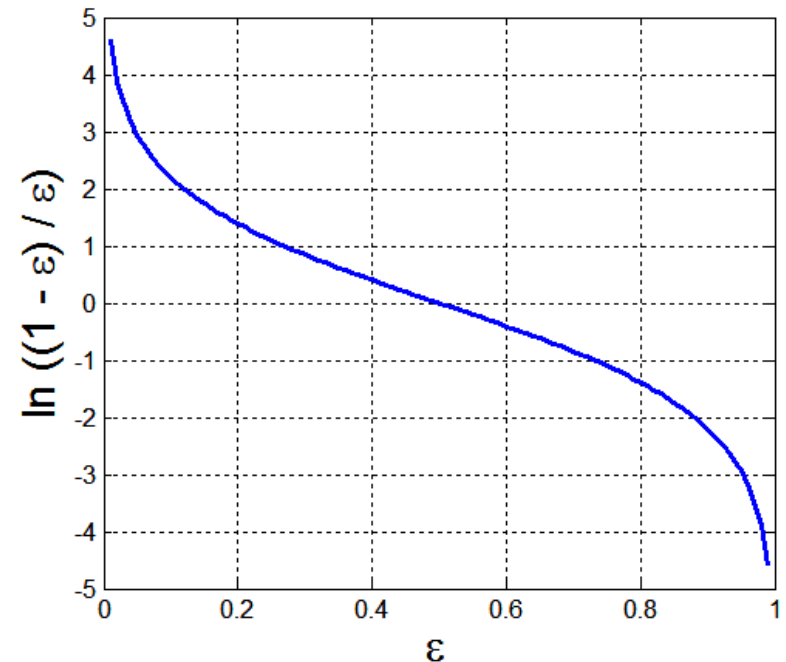
- base classifiers: C_1, C_2, \dots, C_T
- total error (sample weights add up to 1):

$$\varepsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

each sample error

- importance of a classifier:

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \varepsilon_i}{\varepsilon_i} \right)$$



Math Behind AdaBoost (2/2)

- sample weight update:

Increase weight if misclassification;
Increase is proportional to classifier's importance.

$$w_i^{(j+1)} = \frac{w_i^{(j)}}{Z_j} \begin{cases} \exp^{-\alpha_j} & \text{if } C_j(x_i) = y_i \\ \exp^{\alpha_j} & \text{if } C_j(x_i) \neq y_i \end{cases}$$

where Z_j is the normalization factor

- If any intermediate rounds produce error rate higher than 50%, the weights are reverted back to $1/n$ and the re-sampling procedure is repeated
- Classification (α_j is a classifier's importance for the whole dataset):

$$C^*(x) = \arg \max_y \sum_{j=1}^T \alpha_j \delta(C_j(x) = y)$$

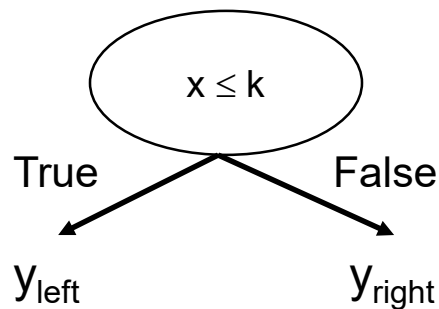
AdaBoost Example 1 (1/3)

- Consider a 1-dimensional dataset

Original Data:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

- Classifier is a decision stump.
 - decision rule: $x \leq k$ versus $x > k$
 - Split point k is chosen based on entropy.



AdaBoost Example 1 (2/3)

- Training sets for the first 3 boosting rounds:
red bar | indicates the split value for x

Boosting Round 1:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 | 0.7 | 0.7 | 0.8 | 1 |
| y | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 |

Boosting Round 2:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.1 | 0.1 | 0.2 | 0.2 | 0.2 | 0.2 | 0.3 | 0.3 | 0.3 | 0.3 |
| y | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

Boosting Round 3:

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|-----|
| x | 0.2 | 0.2 | 0.4 | 0.4 | 0.4 | 0.4 | 0.5 | 0.6 | 0.6 | 0.7 |
| y | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |

- Summary

| Round | Split Point | Left Class | Right Class | alpha |
|-------|-------------|------------|-------------|--------|
| 1 | 0.75 | -1 | 1 | 1.738 |
| 2 | 0.05 | 1 | 1 | 2.7784 |
| 3 | 0.3 | 1 | -1 | 4.1195 |

Example Explained (1/2)

original dataset (test dataset)

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

boosting round-1 prediction (against the test dataset)

for $x \leq 0.75$, $y = -1$, and $x > 0.75$, $y = 1$

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

For $x = 0.1, 0.2, 0.3$, y predictions are wrong.
The weights for $x = 0.1, 0.2, 0.3$ are to be increased.

Example Explained (2/2)

original dataset

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

boosting round-2 prediction

for $x > 0.05$, $y = 1$

| | | | | | | | | | | |
|---|-----|-----|-----|-----|-----|-----|-----|-----|-----|---|
| x | 0.1 | 0.2 | 0.3 | 0.4 | 0.5 | 0.6 | 0.7 | 0.8 | 0.9 | 1 |
| y | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

For $x = 0.4, 0.5, 0.6, 0.7$, y predictions are wrong.
The weights for $x = 0.4, 0.5, 0.6, 0.7$ are to be increased.

AdaBoost Example (3/3)

- Sample weights for Data

| Round | x=0.1 | x=0.2 | x=0.3 | x=0.4 | x=0.5 | x=0.6 | x=0.7 | x=0.8 | x=0.9 | x=1.0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 | 0.1 |
| 2 | 0.311 | 0.311 | 0.311 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 | 0.01 |
| 3 | 0.029 | 0.029 | 0.029 | 0.228 | 0.228 | 0.228 | 0.228 | 0.009 | 0.009 | 0.009 |

- Ensemble Classification

| Round | x=0.1 | x=0.2 | x=0.3 | x=0.4 | x=0.5 | x=0.6 | x=0.7 | x=0.8 | x=0.9 | x=1.0 |
|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|-------|
| 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |
| 2 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |
| 3 | 1 | 1 | 1 | -1 | -1 | -1 | -1 | -1 | -1 | -1 |
| Sum | 5.16 | 5.16 | 5.16 | -3.08 | -3.08 | -3.08 | -3.08 | 0.397 | 0.397 | 0.397 |
| Sign | 1 | 1 | 1 | -1 | -1 | -1 | -1 | 1 | 1 | 1 |

Predicted
Class

Note: **sum** is a sum weighted by alpha
(the classifier's importance)



Note

- How the sample weights and the alpha values are computed will be shown in Example 3.



Scikit-learn Support for AdaBoost

```
from sklearn.ensemble import AdaBoostClassifier
from sklearn.tree import DecisionTreeClassifier
```

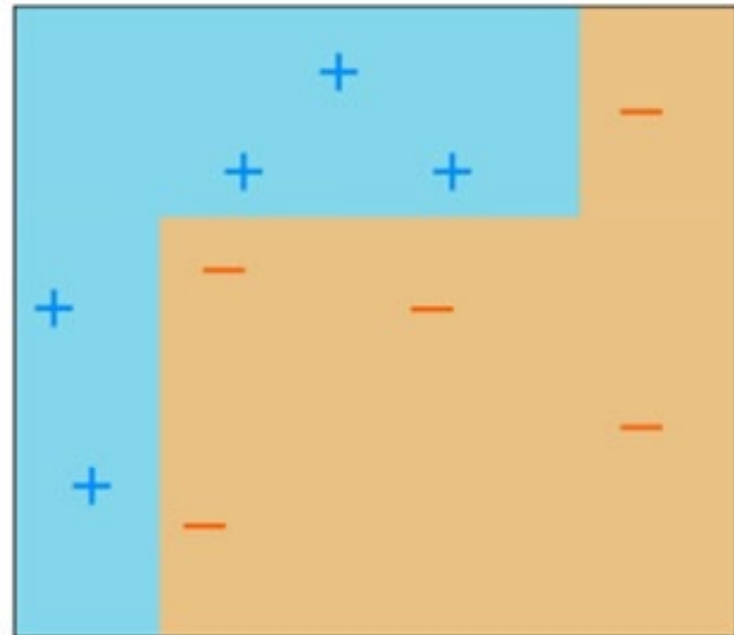
```
# create and fit the model to the dataset
classifier = AdaBoostClassifier(
    DecisionTreeClassifier(max_depth=1),
    n_estimators=200)
classifier.fit(train_X, train_y)
```

```
# predict and evaluate the model
predictions = classifier.predict(test_X)
confusion_matrix(test_y, predictions)
```

AdaBoost Example 2

- training dataset with two classes ($y=+$ or $y=-$), each with 5 training samples (x_1, x_2) values. (on the left below)
- the samples as points with a possible correct decision boundary. (on the right below)

| x_1 | x_2 | y |
|-------|-------|-----|
| 1 | 5 | + |
| 2 | 3 | + |
| 3 | 2 | - |
| 4 | 6 | - |
| 4 | 7 | + |
| 5 | 9 | + |
| 6 | 5 | - |
| 6 | 6 | + |
| 8 | 5 | - |
| 8 | 8 | - |





Computing Steps: Initialization and Iteration 1

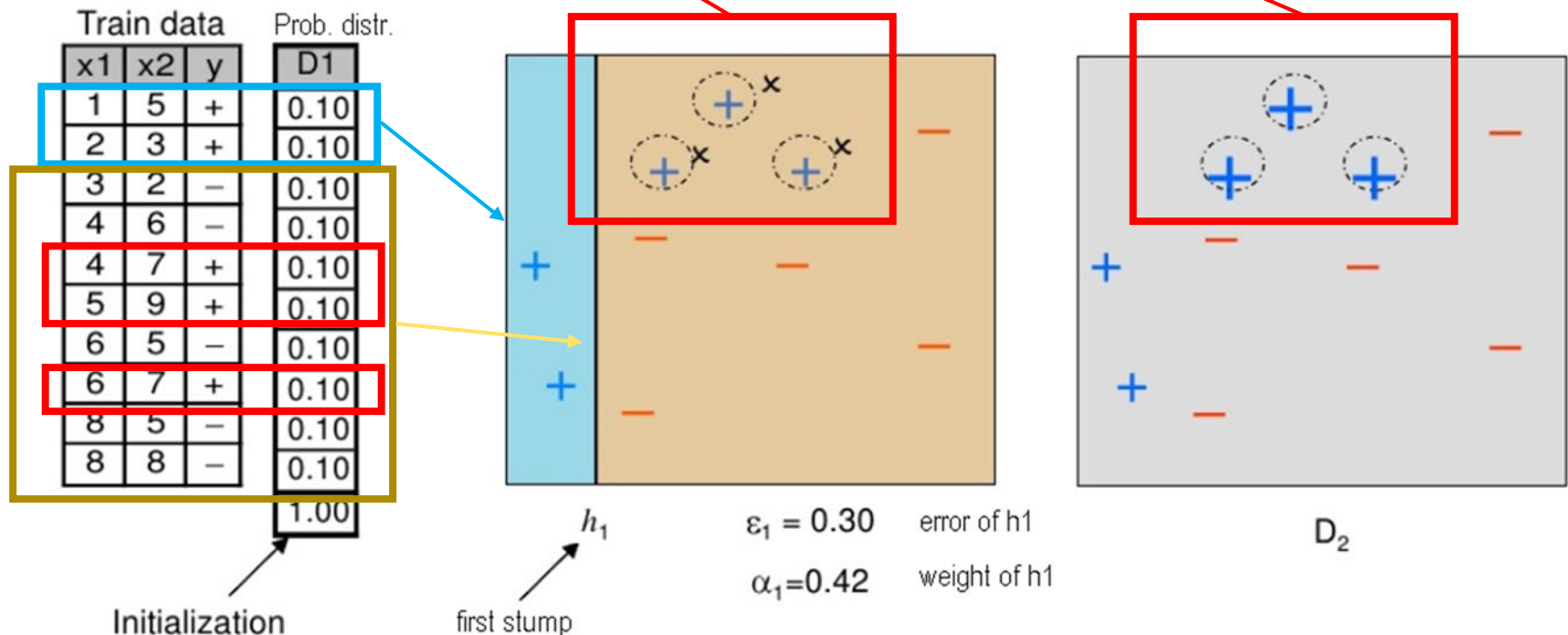
- As base learners, we use decision stumps.
 - Outputs 1 on one side, -1 on the other side.
- Initialize the weights to equal weights $\rightarrow D_1$.
- Train the first stump and calculate its error and combination weight (alpha).
- Increase the sample weights of the missed samples \rightarrow new weighted D_2 .
- ** (Note: The term 'combination weight' is actually 'alpha'; it is different from 'sample weight'.)

Result of Iteration 1

(Note the 3 different colored rectangles)

3 points on wrong side of h_1

Increase their sample weight



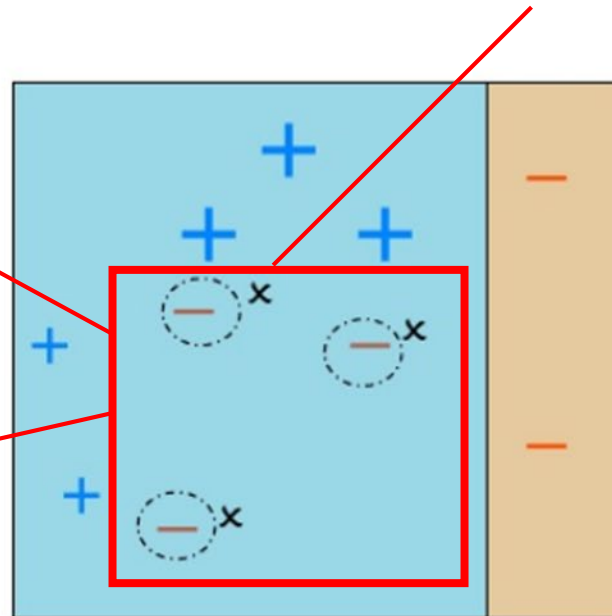
ϵ = error rate of classifier
 α = weight of classifier

Computing Steps: Iteration 2

- Classify the data using the new weighted D_2 .
- Train the second stump and calculate its error and combination weight (alpha).
- Increase the sample weights of the missed samples

Round 1

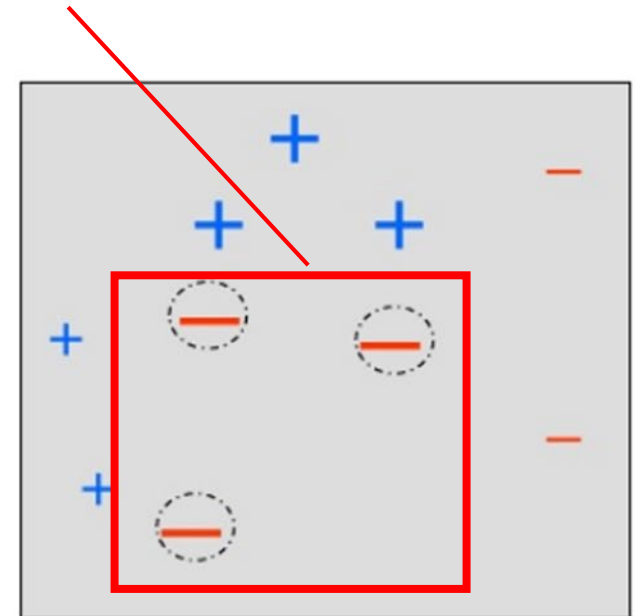
| h1e | ε | D2 |
|-----------------|---------------|--------------|
| 0 | 0.00 | 0.07 |
| 0 | 0.00 | 0.07 |
| 0 | 0.00 | 0.07 |
| 0 | 0.00 | 0.07 |
| 1 | 0.10 | 0.17 |
| 1 | 0.10 | 0.17 |
| 0 | 0.00 | 0.07 |
| 1 | 0.10 | 0.17 |
| 0 | 0.00 | 0.07 |
| 0 | 0.00 | 0.07 |
| ε_1 | 0.30 | 1.00 |
| α_1 | 0.42 | \Downarrow |
| Zt | | 0.92 |



$$\varepsilon_2 = 0.21$$

h_2

$$\alpha_2 = 0.65$$



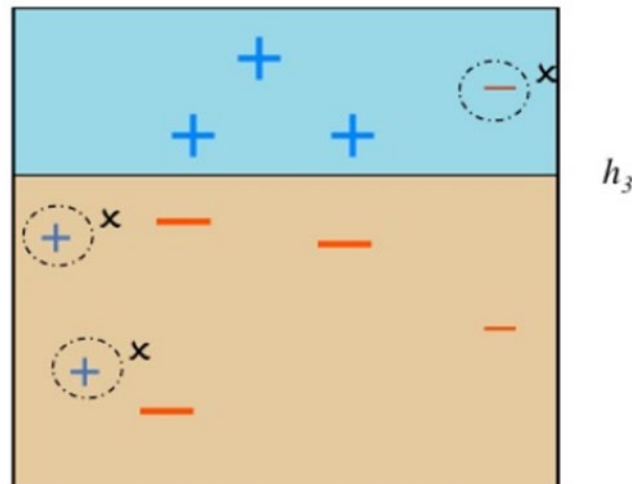
D_3

Computing Steps: Iteration 3

- Classify the data using the new weighted D_3 .
- Train the third stump and calculate its error and combination weight (alpha).
- Then stop, because the combined error becomes 0.

Round 2

| h2e | ϵ | D3 |
|--------------|------------|------|
| 0 | 0.00 | 0.05 |
| 0 | 0.00 | 0.05 |
| 1 | 0.07 | 0.17 |
| 1 | 0.07 | 0.17 |
| 0 | 0.00 | 0.11 |
| 0 | 0.00 | 0.11 |
| 1 | 0.07 | 0.17 |
| 0 | 0.00 | 0.11 |
| 0 | 0.00 | 0.05 |
| 0 | 0.00 | 0.05 |
| ϵ_2 | 0.21 | 1.00 |
| α_2 | 0.65 | ⬇ |
| Zt | | 0.82 |



$$\epsilon_3 = 0.14$$

$$\alpha_3 = 0.92$$

Computing Steps – final classifier

- Combine the 3 stumps using their alphas.

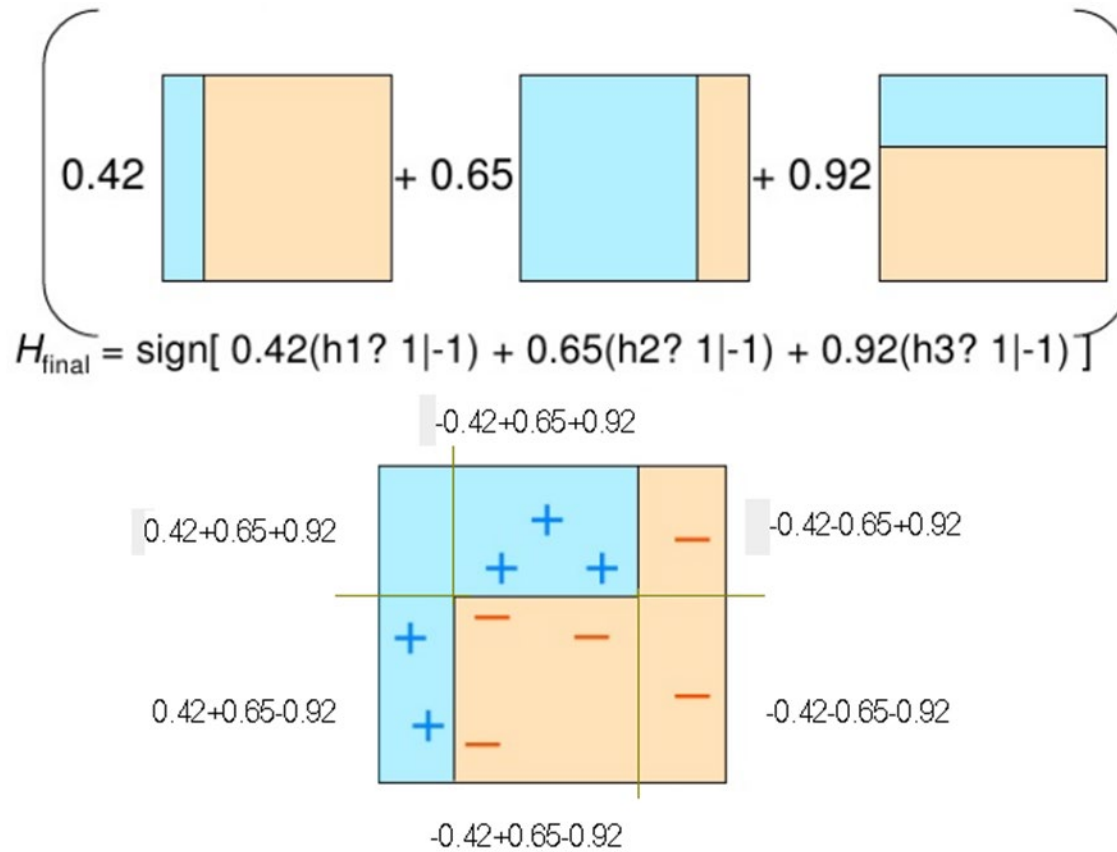
| Train data | | | Round 1 | | | Round 2 | | | Round 3 | | |
|------------|----|---|---------|--------------|------------|--------------|--------------|------------|--------------|--------------|------------|
| x1 | x2 | y | D1 | h1e | ϵ | D2 | h2e | ϵ | D3 | h3e | ϵ |
| 1 | 5 | + | 0.10 | 0 | 0.00 | 0.07 | 0 | 0.00 | 0.05 | 1 | 0.05 |
| 2 | 3 | + | 0.10 | 0 | 0.00 | 0.07 | 0 | 0.00 | 0.05 | 1 | 0.05 |
| 3 | 2 | - | 0.10 | 0 | 0.00 | 0.07 | 1 | 0.07 | 0.17 | 0 | 0.00 |
| 4 | 6 | - | 0.10 | 0 | 0.00 | 0.07 | 1 | 0.07 | 0.17 | 0 | 0.00 |
| 4 | 7 | + | 0.10 | 1 | 0.10 | 0.17 | 0 | 0.00 | 0.11 | 0 | 0.00 |
| 5 | 9 | + | 0.10 | 1 | 0.10 | 0.17 | 0 | 0.00 | 0.11 | 0 | 0.00 |
| 6 | 5 | - | 0.10 | 0 | 0.00 | 0.07 | 1 | 0.07 | 0.17 | 0 | 0.00 |
| 6 | 7 | + | 0.10 | 1 | 0.10 | 0.17 | 0 | 0.00 | 0.11 | 0 | 0.00 |
| 8 | 5 | - | 0.10 | 0 | 0.00 | 0.07 | 0 | 0.00 | 0.05 | 0 | 0.00 |
| 8 | 8 | - | 0.10 | 0 | 0.00 | 0.07 | 0 | 0.00 | 0.05 | 1 | 0.05 |
| | | | 1.00 | ϵ_1 | 0.30 | 1.00 | ϵ_2 | 0.21 | 1.00 | ϵ_3 | 0.14 |
| | | | | α_1 | 0.42 | \downarrow | α_2 | 0.65 | \downarrow | α_3 | 0.92 |
| | | | | Z_t | 0.92 | | Z_t | 0.82 | | | |

Initialization

Importance of each learner

The Ensemble Classifier

- Combine the 3 stumps using their weights



Walkthrough Example 3 (1/10)

(to show the calculation of weights and alpha)

- <https://towardsdatascience.com/machine-learning-part-17-boosting-algorithms-adaboost-in-python-d00faac6c464>
- Dataset (N = 8 samples, 5 features)
(The “attractive” feature is the target variable.)

| weight | smart | polite | fit | attractive |
|--------|-------|--------|-----|------------|
| 180 | no | no | no | no |
| 150 | yes | yes | no | no |
| 175 | no | yes | yes | yes |
| 165 | yes | yes | yes | yes |
| 190 | no | yes | no | no |
| 201 | yes | yes | yes | yes |
| 185 | yes | yes | no | yes |
| 168 | yes | no | yes | yes |



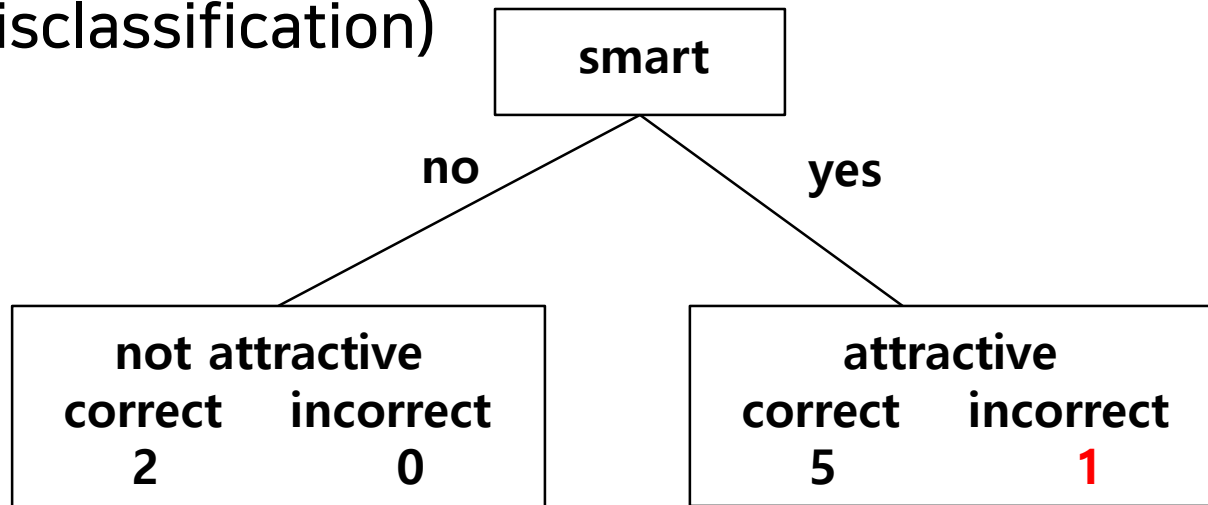
Walkthrough Example 3 (2/10)

- Set the initial weight for each sample ($1/N = 1/8$)
- The sample weight indicates the sample's importance to classification

| weight | smart | polite | fit | attractive | sample weight |
|--------|-------|--------|-----|------------|---------------|
| 180 | no | no | no | no | 1/8 |
| 150 | yes | yes | no | no | 1/8 |
| 175 | no | yes | yes | yes | 1/8 |
| 165 | yes | yes | yes | yes | 1/8 |
| 190 | no | yes | no | no | 1/8 |
| 201 | yes | yes | yes | yes | 1/8 |
| 185 | yes | yes | no | yes | 1/8 |
| 168 | yes | no | yes | yes | 1/8 |

Walkthrough Example 3 (3/10)

- For each feature, we build a decision stump (total 4).
- Compare the prediction by each decision stump against the labels in the training set.
- The decision stump (and its feature) with the best prediction becomes the next decision tree in the ensemble.
- Suppose the best classifier is the following (1 misclassification)



Walkthrough Example 3 (3/10)

- Compute the significance of the classifier
significance (α) =
 $\frac{1}{2} \ln \left(\frac{1 - \text{total_error}}{\text{total_error}} \right)$
- total_error (ϵ) = sum of sample weights for miscalculated samples = 1/8
- $\alpha = 0.97$ (* note: This number will later be used to determine the prediction by the ensemble classifier)

$$\epsilon_i = \frac{1}{N} \sum_{j=1}^N w_j \delta(C_i(x_j) \neq y_j)$$

$$\alpha_i = \frac{1}{2} \ln \left(\frac{1 - \epsilon_i}{\epsilon_i} \right)$$



Walkthrough Example 3 (4/10)

- Compute new sample weights for all samples.
- First, increase the sample weight for each of the samples incorrectly classified by the best decision tree in the current iteration

new sample weight =

$$\text{current sample weight} \times e^{\alpha} = 0.33$$

- Next, decrease the sample weight for each of the samples correctly classified

new sample weight =

$$\text{current sample weight} \times e^{-\alpha} = 0.05$$



Walkthrough Example 3 (5/10)

- The sum of the new sample weights is less than 1.
- Since the sample weights are probabilities, their sum must be 1.
- We need to normalize the new sample weights.



Walkthrough Example 3 (6/10)

- The new normalized sample weights are as follows:

| weight | smart | polite | fit | attractive | sample weight | new smp weight | normalized smp weight |
|--------|-------|--------|-----|------------|---------------|----------------|-----------------------|
| 180 | no | no | no | no | 1/8 | 0.05 | 0.07 |
| 150 | yes | yes | no | no | 1/8 | 0.33 | 0.49 |
| 175 | no | yes | yes | yes | 1/8 | 0.05 | 0.07 |
| 165 | yes | yes | yes | yes | 1/8 | 0.05 | 0.07 |
| 190 | no | yes | no | no | 1/8 | 0.05 | 0.07 |
| 201 | yes | yes | yes | yes | 1/8 | 0.05 | 0.07 |
| 185 | yes | yes | no | yes | 1/8 | 0.05 | 0.07 |
| 168 | yes | no | yes | yes | 1/8 | 0.05 | 0.07 |



Walkthrough Example 3 (7/10)

- We now need to randomly sample a new training dataset.
- The previously misclassified samples have a higher probability of being selected in the new training dataset, because of the new higher sample weights.

Walkthrough Example 3 (8/10)

- The newly sampled training dataset is as follows:
- Note the previously misclassified sample is included 4 times in the updated training dataset.

| weight | smart | polite | fit | attractive |
|--------|-------|--------|-----|------------|
| 201 | yes | yes | yes | yes |
| 150 | yes | yes | no | no |
| 180 | no | no | no | no |
| 150 | yes | yes | no | no |
| 150 | yes | yes | no | no |
| 150 | yes | yes | no | no |
| 185 | yes | yes | no | yes |
| 165 | yes | yes | yes | yes |



Walkthrough Example 3 (9/10)

- Repeat the previous steps until there is no misclassification or the pre-specified number of base classifiers (estimators) has been built.



Walkthrough Example 3 (10/10)

- Finally, divide all the base classifiers into groups according to their predictions (“is attractive” and “is not attractive”).
- To take a vote, for each group, add up the significance (α) of every base classifier.
- The ensemble classifier (i.e., the AdaBoost classifier) selects the prediction of the group with the largest sum.
- (e.g.) “is attractive” (total 1.98) (* selected *)
0.97, 0.22, 0.79
“is not attractive” (total 0.96)
0.46, 0.19, 0.31



Roadmap: Boosting

- AdaBoost
- Gradient Boosting
- XGBoost



Acknowledgments

- <https://sefiks.com/2018/10/04/a-step-by-step-gradient-boosting-decision-tree-example/>
- <https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/>
- <https://medium.com/@gabrieltseng/gradient-boosting-and-xgboost-c306c1bcfaf5>
- <https://blog.paperspace.com/implementing-gradient-boosting-regression-python/>



AdaBoost vs. Gradient Boosting

- Both build weak base learners (classifiers) one at a time, ultimately leading to a strong learner.
- AdaBoost
 - Try to fix the errors (miscalculations) in each iteration (epoch) by adjusting sample weights
 - AdaBoost is a special case of Gradient Boosting
- Gradient Boosting
 - Try to fix the errors in each iteration by using gradient descent on the error function
 - Compute the prediction errors in each iteration and use them as target values to predict in the next iteration



Inventors of Gradient Boosting

- Original concept by Leo Breiman
- Invented by Jerome Friedman (inventor of CART and MARS decision trees)
- Followed by Llew Mason, Jonathan Baxter, Peter Bartlett, Marcus Frean

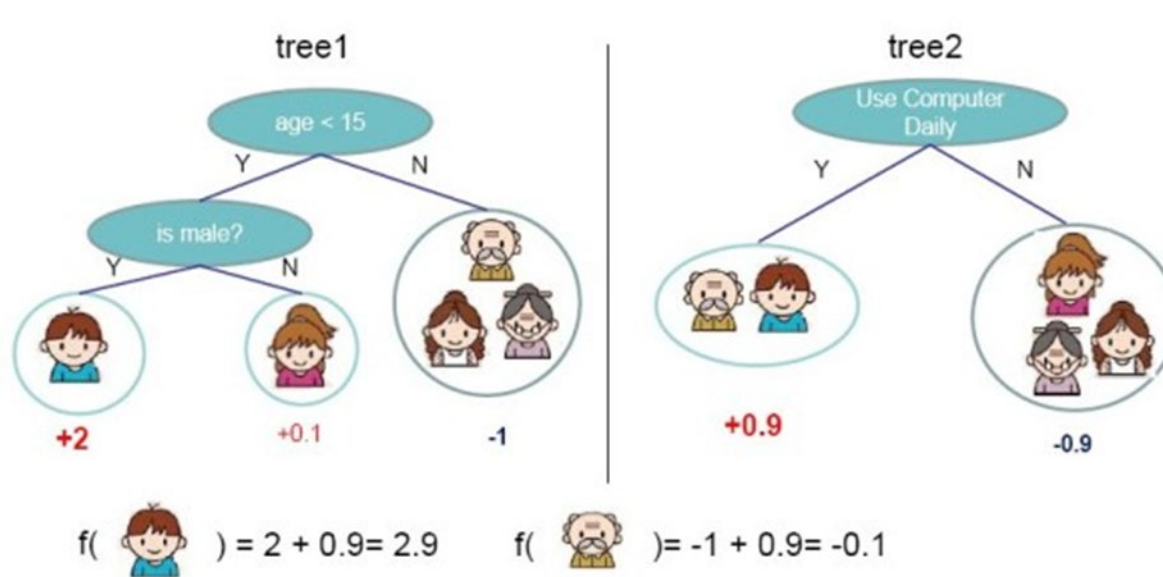


Gradient Boosting Steps

- Build the first model on a train dataset.
- Using this model, predictions are made on the dataset.
- Errors are calculated by comparing the predictions and the actual (target) values.
- Repeat the following process until the error function does not change, or the specified number of estimators is reached.
 - Build a new model using the previously calculated errors as target variable.
 - The predictions made by this new model are combined with the predictions of the previous model.
 - New errors are calculated using these predictions and the actual values.

Intuitive Introduction

- First, we build decision tree 1: It returns 2 as a result for 'boy'.
- Next, we build decision tree 2 based on errors by decision tree 1: It returns 0.9 for 'boy'.
- Final decision for 'boy' would be 2.9 (which sums the predictions of the two sequential decision trees)





Walkthrough Example (1/12)

- Dataset

- 14 samples, 6 features
- 4 features are weather conditions that may affect the number of golf players
- “golfers” is the target variable, and means the number of golf players

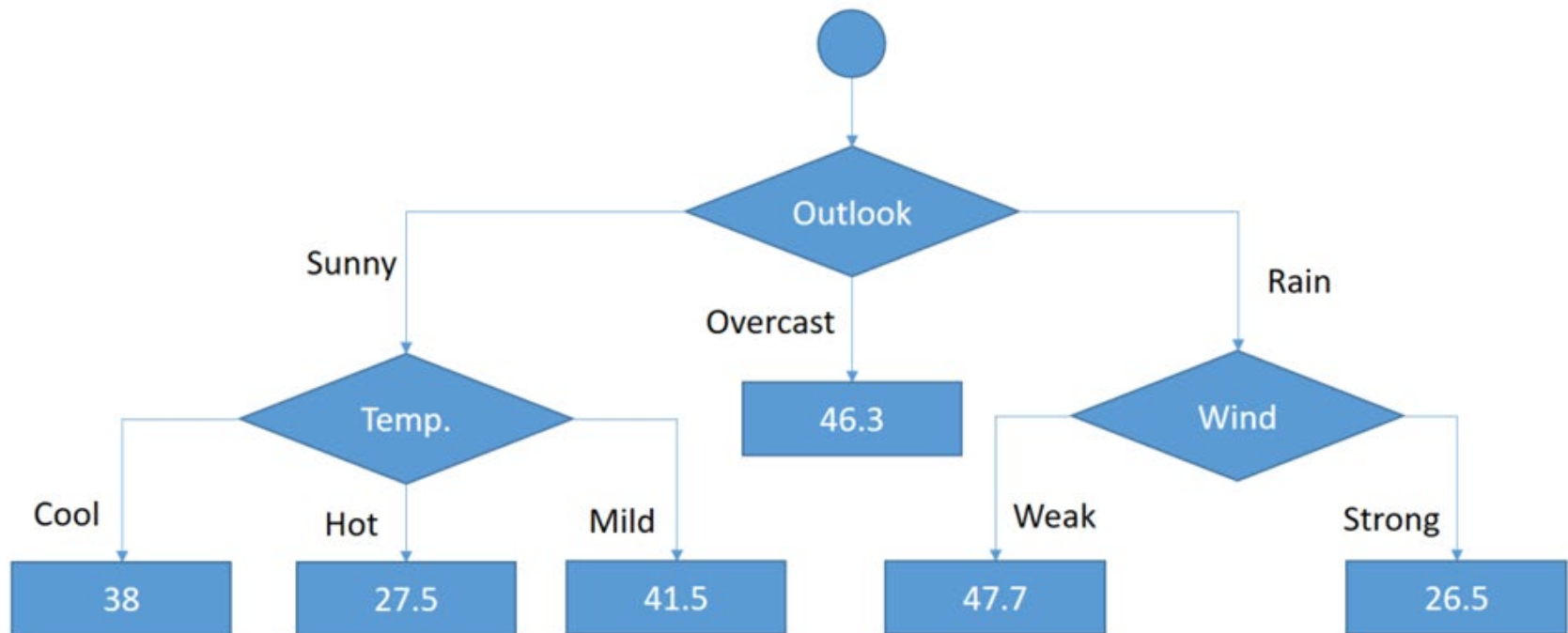


dataset

| Day | Outlook | Temp. | Humidity | Wind | golfers |
|-----|----------|-------|----------|--------|---------|
| 1 | Sunny | Hot | High | Weak | 25 |
| 2 | Sunny | Hot | High | Strong | 30 |
| 3 | Overcast | Hot | High | Weak | 46 |
| 4 | Rain | Mild | High | Weak | 45 |
| 5 | Rain | Cool | Normal | Weak | 52 |
| 6 | Rain | Cool | Normal | Strong | 23 |
| 7 | Overcast | Cool | Normal | Strong | 43 |
| 8 | Sunny | Mild | High | Weak | 35 |
| 9 | Sunny | Cool | Normal | Weak | 38 |
| 10 | Rain | Mild | Normal | Weak | 46 |
| 11 | Sunny | Mild | Normal | Strong | 48 |
| 12 | Overcast | Mild | High | Strong | 52 |
| 13 | Overcast | Hot | Normal | Weak | 44 |
| 14 | Rain | Mild | High | Strong | 30 |

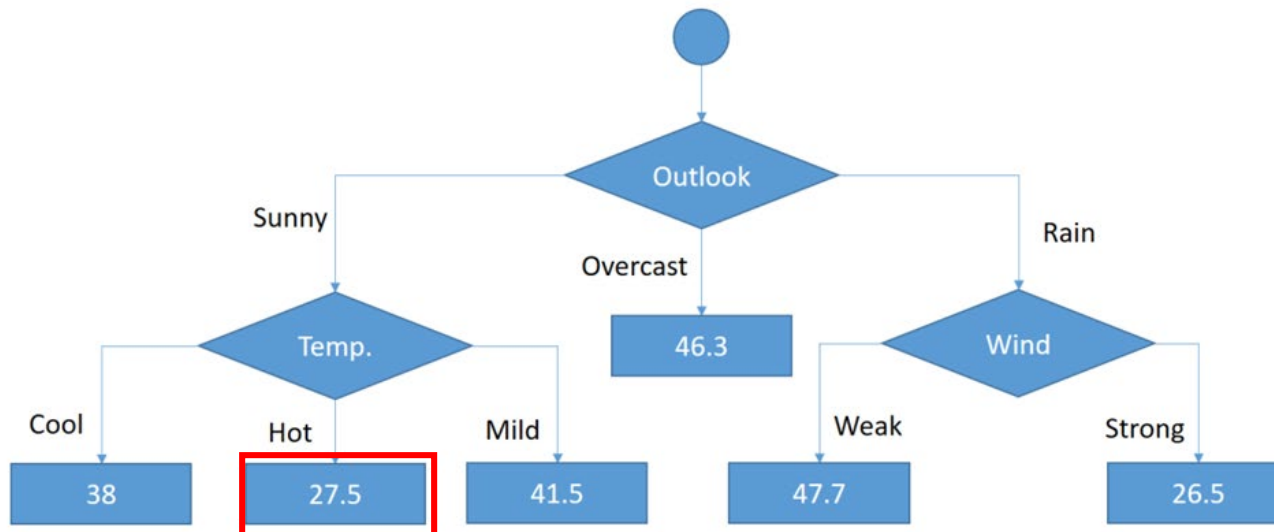
Walkthrough Example (2/12)

- Build decision tree 1



Walkthrough Example (3/12)

- Compute errors (for all 14 samples)
 - (e.g) Day 1 sample: outlook=sunny, temp=hot
prediction=27.5, actual=25, error=-2.5
 - (e.g) Day 2 sample: outlook=sunny, temp=hot
prediction=27.5, actual=30, error=+2.5





Walkthrough Example (4/12)

- Let us assume our loss function is mean squared error (MSE)

$$\text{loss} = (1/2) \times (y - y')^2$$

where y is the actual value and y' is the prediction.

- 'Gradient' (partial derivative) means gradient descent in gradient boosting.
- We will update each prediction as partial derivative of the loss function with respect to the prediction.
- Let's find this derivative first.

$$\begin{aligned}\partial \text{loss} / \partial y' &= \partial((1/2) \times (y - f(x))^2) / \partial y' \\ &= 2 \times (1/2) \times (y - y') \times \partial(-y') / \partial y' \\ &= 2 \times (1/2) \times (y - y') \times (-1) = y' - y\end{aligned}$$



Walkthrough Example (5/12)

- Now, we can update the predictions by applying the following formula. Here, α is the learning rate.

$$y' = y' - \alpha \times (\partial \text{loss} / \partial y')$$

- Focus on the updating term only. We will set α to 1 to make the formula simpler.

$$\begin{aligned} -\alpha \times (\partial \text{loss} / \partial y') &= -\alpha \times (y' - y) \\ &= \alpha \times (y - y') = y - y' \end{aligned}$$

- This is the label that we are going use to build a new decision tree (decision tree 2).

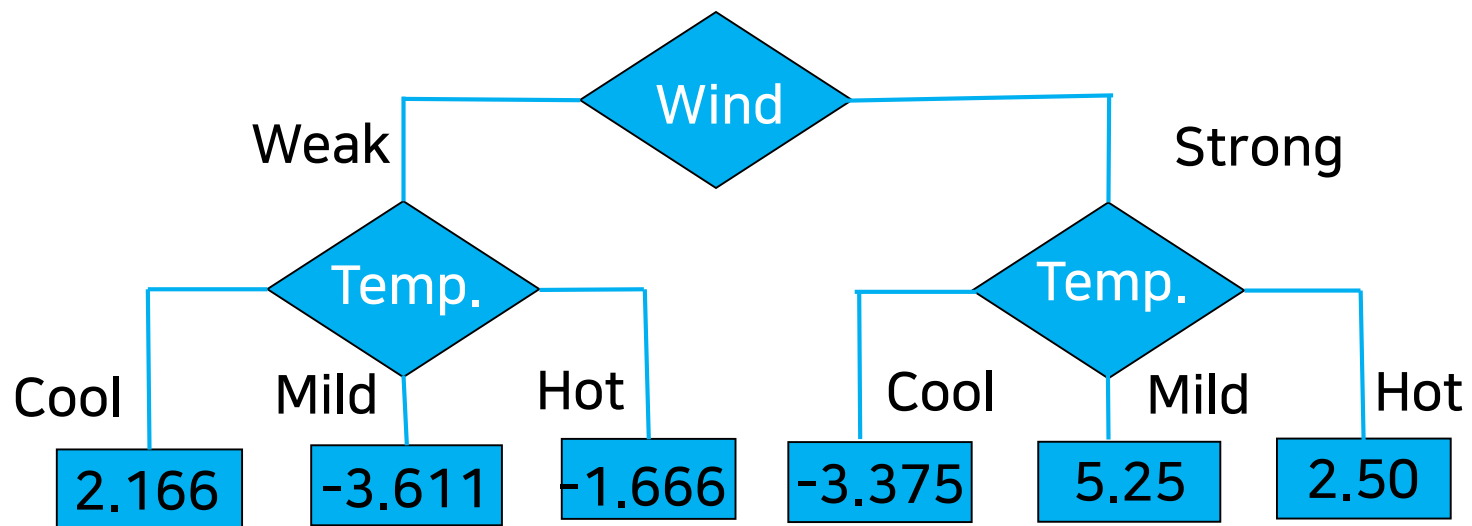
Walkthrough Example (6/12):

updated dataset (* note the 'residual' values)

| Day | Outlook | Temp. | Humidity | Wind | residual |
|-----|----------|-------|----------|--------|----------|
| 1 | Sunny | Hot | High | Weak | -2.5 |
| 2 | Sunny | Hot | High | Strong | 2.5 |
| 3 | Overcast | Hot | High | Weak | -0.25 |
| 4 | Rain | Mild | High | Weak | -2.66 |
| 5 | Rain | Cool | Normal | Weak | 4.333 |
| 6 | Rain | Cool | Normal | Strong | -3.5 |
| 7 | Overcast | Cool | Normal | Strong | -3.25 |
| 8 | Sunny | Mild | High | Weak | -6.5 |
| 9 | Sunny | Cool | Normal | Weak | 0 |
| 10 | Rain | Mild | Normal | Weak | -1.66 |
| 11 | Sunny | Mild | Normal | Strong | 6.5 |
| 12 | Overcast | Mild | High | Strong | 5.75 |
| 13 | Overcast | Hot | Normal | Weak | -2.25 |
| 14 | Rain | Mild | High | Strong | 3.55 |

Walkthrough Example (7/12)

- Using the updated dataset, build decision tree 2.
- Note the values in the leaf nodes. These are residuals.



- From now, each decision tree computes the residuals, not the original target values.
- The goal is to sum all residuals computed by decision trees and add it to the result of decision tree 1.



Walkthrough Example (8/12)

- Compute errors (for all 14 samples)
 - (e.g) Day 1 sample: wind=weak, temp=hot
prediction=1.666, actual=-2.5, error=-0.833
 - (e.g) Day 2 sample: wind=strong, temp = hot
prediction=2.5, actual=2.5, error=0

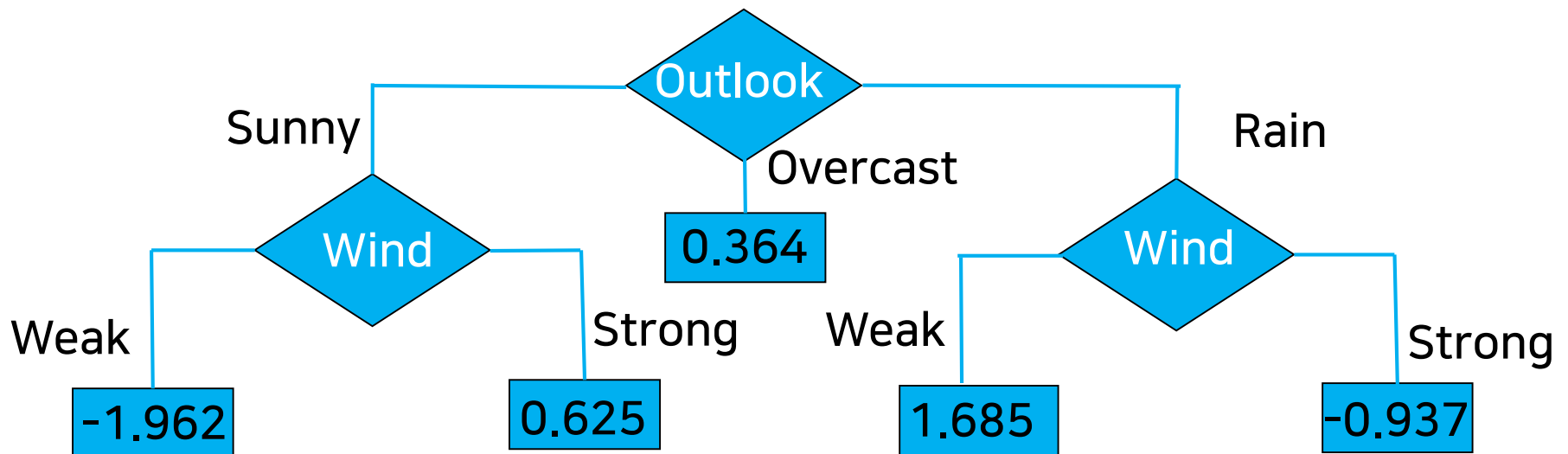


Walkthrough Example (9/12): updated dataset

| Day | Outlook | Temp. | Humidity | Wind | residual | |
|-----|----------|-------|----------|--------|----------|-----|
| 1 | Sunny | Hot | High | Weak | -0.833 | |
| 2 | Sunny | Hot | High | Strong | 0.0 | |
| 3 | Overcast | Hot | High | Weak | 1.416 | |
| 4 | Rain | Mild | High | Weak | 0.944 | |
| 5 | Rain | Cool | Normal | Weak | 2.166 | |
| 6 | Rain | Cool | Normal | Strong | -0.125 | |
| 7 | Overcast | Cool | Normal | Strong | 0.125 | |
| 8 | Sunny | Mild | High | Weak | -2.888 | |
| 9 | Sunny | Cool | Normal | Weak | -2.166 | |
| 10 | Rain | Mild | Normal | Weak | 1.944 | |
| 11 | Sunny | Mild | Normal | Strong | 1.25 | |
| 12 | Overcast | Mild | High | Strong | 0.5 | |
| 13 | Overcast | Hot | Normal | Weak | -0.583 | 110 |
| 14 | Rain | Mild | High | Strong | -1.75 | |

Walkthrough Example (10/12)

- Using the updated dataset, build decision tree 3.





Walkthrough Example (11/12):

predictions by 5 sequential decision trees

| Day | Actual | epoch 1 | epoch 2 | epoch 3 | epoch 4 | epoch 5 | prediction sum |
|-----|--------|---------|---------|---------|---------|-----------|----------------|
| 1 | 25 | 27.5 | -1.667 | -1.963 | 0.152 | 5.55E-17 | 24.023 |
| 2 | 30 | 27.5 | 2.5 | 0.625 | 0.152 | 5.55E-17 | 30.777 |
| 3 | 46 | 46.25 | -1.667 | 0.365 | 0.152 | 5.55E-17 | 45.1 |
| 4 | 45 | 47.667 | -3.611 | 1.685 | -0.586 | -1.88E-01 | 44.967 |
| 5 | 52 | 47.667 | 2.167 | 1.685 | 0.213 | 1.39E-17 | 51.731 |
| 6 | 23 | 26.5 | -3.375 | -0.938 | 0.213 | 1.39E-17 | 22.4 |
| 7 | 43 | 46.25 | -3.375 | 0.365 | 0.213 | 1.39E-17 | 43.452 |
| 8 | 35 | 41.5 | -3.611 | -1.963 | -0.586 | -7.86E-02 | 35.261 |
| 9 | 38 | 38 | 2.167 | -1.963 | 0.213 | 1.39E-17 | 38.416 |
| 10 | 46 | 47.667 | -3.611 | 1.685 | 0.442 | -1.88E-01 | 45.995 |
| 11 | 48 | 41.5 | 5.25 | 0.625 | 0.442 | -7.86E-02 | 47.739 |
| 12 | 52 | 46.25 | 5.25 | 0.365 | -0.586 | 7.21E-01 | 52 |
| 13 | 44 | 46.25 | -1.667 | 0.365 | 0.152 | 5.55E-17 | 45.1 |
| 14 | 30 | 26.5 | 5.25 | -0.938 | -0.586 | -1.88E-01 | 30.038 |



Walkthrough Example (12/12)

- In the previous table, we summed the predictions of all 5 sequential decision trees.
- The table shows that, for each sample, the absolute error is reduced from epoch 1 (iteration 1) to epoch 5.



Making the Final Prediction

- Once training is completed, use all the trees in the ensemble to make a final prediction for the value of the target variable (using a test dataset).
- The final prediction will be equal to the values computed in the first epoch, plus all of the residuals predicted by the trees that make up the ensemble multiplied by the learning rate.



Scikit-learn Support for Gradient Boosting (for regression) (1/3)

<https://towardsdatascience.com/machine-learning-part-18-boosting-algorithms-gradient-boosting-in-python-ef5ae6965be4>

```
from sklearn.ensemble import GradientBoostingRegressor
```

```
# (omitted) load dataset, prepare train set and test set
```

```
regressor = GradientBoostingRegressor(  
    max_depth=2,  
    n_estimators=3,  
    learning_rate=1.0  
)  
regressor.fit(X_train, y_train)
```



Scikit-learn Support for Gradient Boosting (for regression) (2/3)

```
errors = [mean_squared_error(y_test, y_pred) for y_pred in  
          regressor.staged_predict(X_test)]
```

staged_predict method measures the validation error at each iteration of training (i.e. with one tree, with two trees...) to find the optimal number of trees

```
best_n_estimators = np.argmin(errors)
```

Build and fit the model using the optimal number of trees.

```
best_regressor = GradientBoostingRegressor(  
    max_depth=2,  
    n_estimators=best_n_estimators,  
    learning_rate=1.0  
)  
best_regressor.fit(X_train, y_train)
```



Scikit-learn Support for Gradient Boosting (for regression) (3/3)

let us use the mean absolute error (average distance from the predictions and the actual values).

```
y_pred = best_regressor.predict(X_test)
mean_absolute_error(y_test, y_pred)
```



Scikit-learn Support for Gradient Boosting (for classification)

```
from sklearn.ensemble import GradientBoostingClassifier
# (omitted) load dataset, prepare train set and test set
```

```
# Set different learning rates, so we can compare the
  classifier's performance at different learning rates.
```

```
lr_list = [0.05, 0.075, 0.1, 0.25, 0.5, 0.75, 1]
```

```
for learning_rate in lr_list:
```

```
    gb_clf = GradientBoostingClassifier(n_estimators=20,
        learning_rate=learning_rate, max_features=2,
        max_depth=2, random_state=0)
```

```
    gb_clf.fit(X_train, y_train)
```

```
    print("Learning rate: ", learning_rate)
```

```
    print("Accuracy score (training):
```

```
        {0:.3f}".format(gb_clf.score(X_train, y_train)))
```

```
    print("Accuracy score (validation):
```

```
        {0:.3f}".format(gb_clf.score(X_val, y_val)))
```



Hyperparameters of Gradient Boosting

- `n_estimators` = number of trees in the ensemble.
 - default 100
- `max_features` = maximum number of features considered when splitting a node.
- `criterion` = the function used to evaluate the quality of a split.
 - gini (default), entropy
- `max_depth` = maximum number of levels allowed in each tree.
 - default 3
- `min_samples_leaf` = minimum number of samples which can be stored in a tree leaf.
 - default 1
- `min_samples_split` = minimum number of samples necessary in a node to cause node splitting.
 - default 2



Roadmap: Boosting

- AdaBoost
- Gradient Boosting
- XGBoost



Acknowledgments

- <https://www.datacamp.com/community/tutorials/xgboost-in-python>
- <https://blog.quantinsti.com/xgboost-python/>
- <https://stackabuse.com/gradient-boosting-classifiers-in-python-with-scikit-learn/>



XGBoost

- eXtreme Gradient Boost
- Invented by Tianqi Chen
- Advantages (over Gradient Boosting)
 - High speed and performance
 - Implements parallel processing
 - Implements regularization to reduce overfitting
 - Allows custom optimization objectives and evaluation criteria
- A wide variety of tuning parameters



Installing and Using XGBoost

```
# install XGBoost
!pip install xgboost (on cmd)

# import libraries needed
from xgboost import XGBClassifier

# assume dataset is prepared
# build and fit a model
xgb_clf = XGBClassifier()
    # n_estimators = 100 (default)
    # max_depth = 3 (default)
xgb_clf.fit(X_train, y_train)

# evaluate the model
score = xgb_clf.score(X_val, y_val)
```



Note: Validation Set

- To tune a trained model (before testing and predicting), we need to use a validation set.
- The validation set comes out of the training set.
- Test (prediction) is done only with the test set.

- The initial full training dataset is split into training set (X_train, y_train) and test set (X_test)
- The training set is further split (using train_test_split) into (reduced) training set and validation set.

```
X_train, X_val, y_train, y_val = train_test_split(X_train,  
y_train, test_size=0.3, random_state=12)
```



XGBoost Hyperparameters (1/2)

many

- **n_estimators**: number of trees you want to build.
- **learning_rate**: step size shrinkage used to prevent overfitting. Range is $[0, 1]$
- **objective**: determines the loss function to be used like reg:linear for regression problems, reg:logistic for classification problems with only decision, binary:logistic for classification problems with probability
- **max_depth**: determines how deeply each tree is allowed to grow during any boosting round. Used to control overfitting.
- **max_features** : number of features it should consider while searching for the best split.



XGBoost Hyperparameters (2/2)

- **subsample**: percentage of samples used per tree. Low value can lead to underfitting.
- **colsample_bytree**: percentage of features used per tree. High value can lead to overfitting.
- **max_lead_nodes** : maximum number of terminal leaves in a tree. If this is defined max_depth is ignored.
- **min_samples_split**: Minimum number of observation which is required in a node to be considered for splitting. It is used to control overfitting.
- **min_samples_leaf** : Minimum samples required in a terminal or leaf node. Lower values should be chosen for imbalanced class problems since the regions in which the minority class will be in the majority will be very small.
- **min_weight_fraction_leaf** : similar to the previous but defines a fraction of the total number of observations instead of an integer.



XGBoost Regularization Parameters

- XGBoost supports regularization parameters to penalize models as they become more complex and reduce them to simple (parsimonious) models.
- **alpha**: L1 regularization on leaf weights. A large value leads to more regularization.
- **lambda**: L2 regularization on leaf weights and is smoother than L1 regularization.
- **gamma**: controls whether a given node will split based on the expected reduction in loss after the split. A higher value leads to fewer splits. Supported only for tree-based learners.



End of SubModule
