Module 8

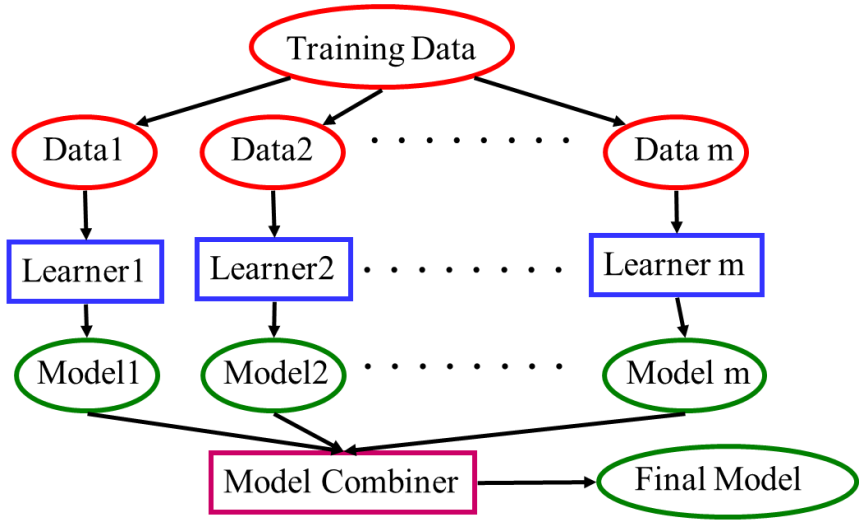
Part 2 – Ensembles

José Manuel García Nieto – University of Málaga

Table of contents

Ensembles: the role of diversity

- When combing multiple independent and diverse decisions each of which is at least more accurate than random guessing, random errors cancel each other out and correct decisions are reinforced.
- Learn multiple alternative definitions of a concept using different training data or different learning algorithms.
- Combine decisions of multiple definitions, e.g., using weighted voting



- Majority Voting – Analysis using many different classification models with one identical training set
- Use a single, arbitrary learning algorithm, but manipulate training data to make it learn multiple models.
  - Data1 ≠ Data2 ≠ … ≠ Data m
  - Learner1 = Learner2 = … = Learner m
- Different methods for changing training data:
  - Bagging: Resample training data
  - Boosting: Reweight training data

What is Ensemble Learning?

- The term ‘ensemble’ can be defined as follows:
  - In statistical mechanics, an ensemble of a system refers to the collection of equivalent systems.
- Instead of expecting performance results from a single model, ensemble learning draws a better result using the collective intelligence of different models, such as averaging out many different single models or making a decision based on the majority vote.
- There are many different ensemble methods using collective intelligence.
  - Voting – Drawing results through voting
  - Bagging – Bootstrap aggregating (duplicated creation of various samples)
  - Boosting – Weighting by supplementing previous errors
  - Stacking – A meta-model based on different models
- There can be other more different methods since ensemble learning applies a certain technique/methodology. Yet, the four methods listed above are the most representative ensemble techniques in the sklearn library.

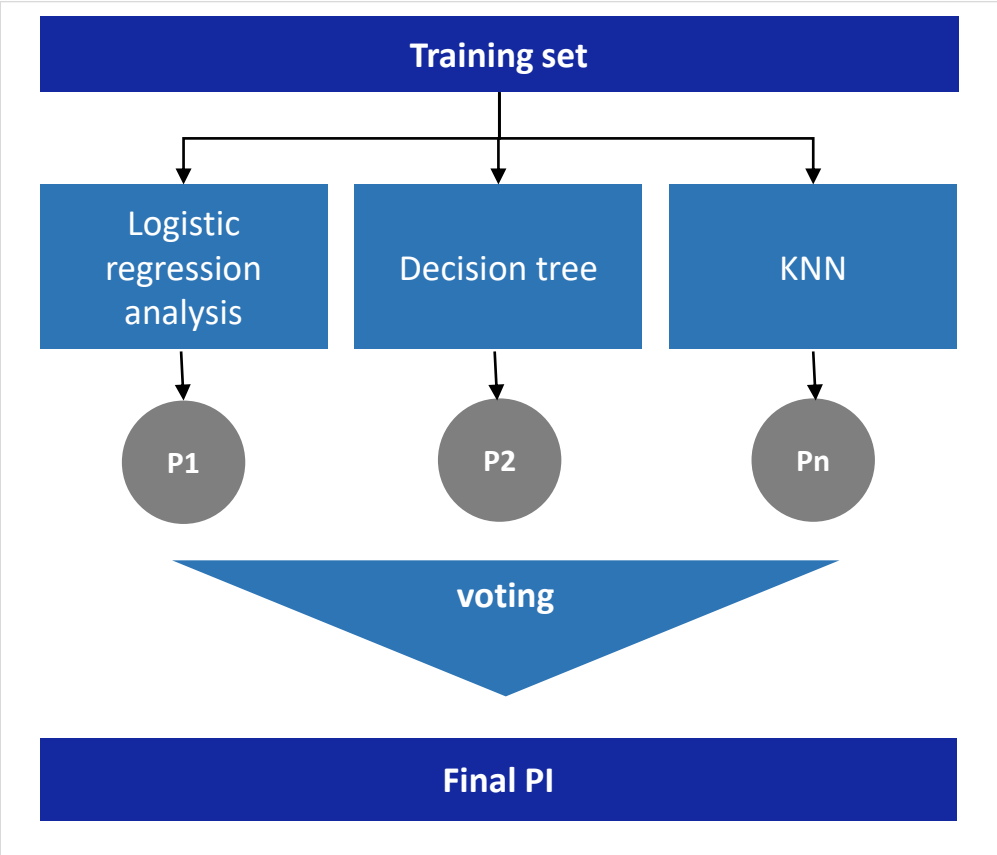
Ensemble algorithms

- Strong predictive model based on the weaker learners.
- **Voting type**:
  - A collection of basic learners that “vote”
  - An ensemble of different kinds of learners **Ex** Combine Tree, KNN, SVM, etc.
- **Bagging type**:
  - A collection of independent weak learners that “vote”
  - An ensemble of the same kind of weak learners **Ex** Random Forest
- **Boosting type**:
  - A series of weak learners that adaptatively learn and predict **Ex** AdaBoost, GBM, XGBoost, etc.
  - The series is grown by adding new learners multiplied by the “boosting weights.”

Voting Ensemble

- Can be applied to classification and regression.
- The learners that form an ensemble should be of **different kinds** for a good performance.
- Two voting methods for the classifier:
  - Hard: predicted class label is given by the majority rule voting.
  - Soft: predicted class label is given by the **argmax** of the sum of the predicted probabilities.

Voting Ensemble



- Use a single training set.
- Use different classification models.
- Predictive value
- If the predictive values of each analysis model are different from voting, choose the result with the most values.
  - hard voting
  - soft voting

→ Predict the highest class by averaging out the prediction of individual classifier.

Voting Ensemble

- Hard Voting
  - Taking classification as an example. Suppose the predictive values for classification are 1,0,0,1,1 since 1 has three votes and 0 has 2 votes. In that case, 1 becomes the final predictive value in the hard voting method.
- Soft Voting
  - Soft voting method calculates the average value of each probability and then determines the one with the highest probability.
  - Suppose the probability of getting class 0 is (0.4, 0.9, 0.9, 0.4, 0.4), and the probability of getting class 1 is (0.6, 0.1, 0.1, 0.6, 0.6). The final probability of getting class 0 is (0.4+0.9+0.9+0.4+0.4) / 5 = 0.44; the final probability of getting class 1 is (0.6+0.1+0.1+0.6+0.6) / 5 = 0.4. Therefore, the selected final value is different from the result of the hard voting above.
- In general, using the soft voting method is considered more reasonable than the hard voting method in competitions because the soft voting method provides a much better actual performance result.

Voting Ensemble

- To import the voting ensemble as class:

```
from sklearn.ensemble import VotingClassifier        # For classification
from sklearn.ensemble import VotingRegressor         # For regression
```

- To instantiate an object that implements voting ensemble:

```
myKNN = KNeighborsClassifier(n_neighbors = 3)
myLL = LogisticRegression()
myVotingEnsemble=VotingClassifier(estimators=[('lr',myLL),('knn',myKNN)],voting='hard')
```

- We can train and predict just like any other estimator:

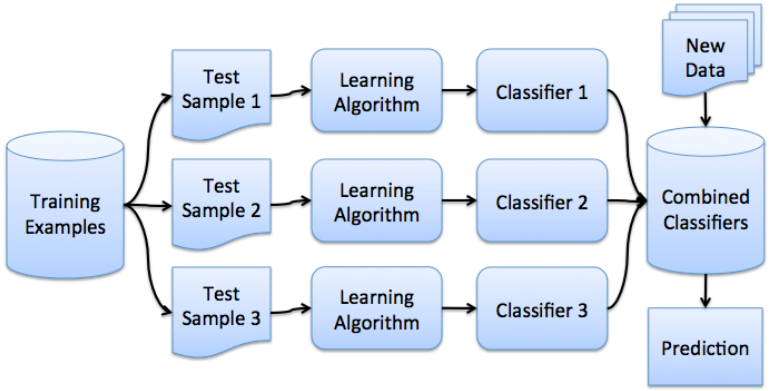
```
myVotingEnsemble.fit(X_train, Y_train)
myVotingEnsemble.predict(X_test)
```

Table of contents

Bagging Ensembles

- Create ensembles by repeatedly randomly resampling the training data (Brieman, 1996).
  - Given a training set of size n, create m samples of size n by drawing n examples from the original data, with replacement.
  - Each bootstrap sample will on average contain 63.2% of the unique training examples, the rest are replicates.
  - Combine the m resulting models using simple majority vote.

    - Decreases error by decreasing the variance in the results due to unstable learners, algorithms (like decision trees) whose output can change dramatically when the training data is slightly changed (e.g. random forests).



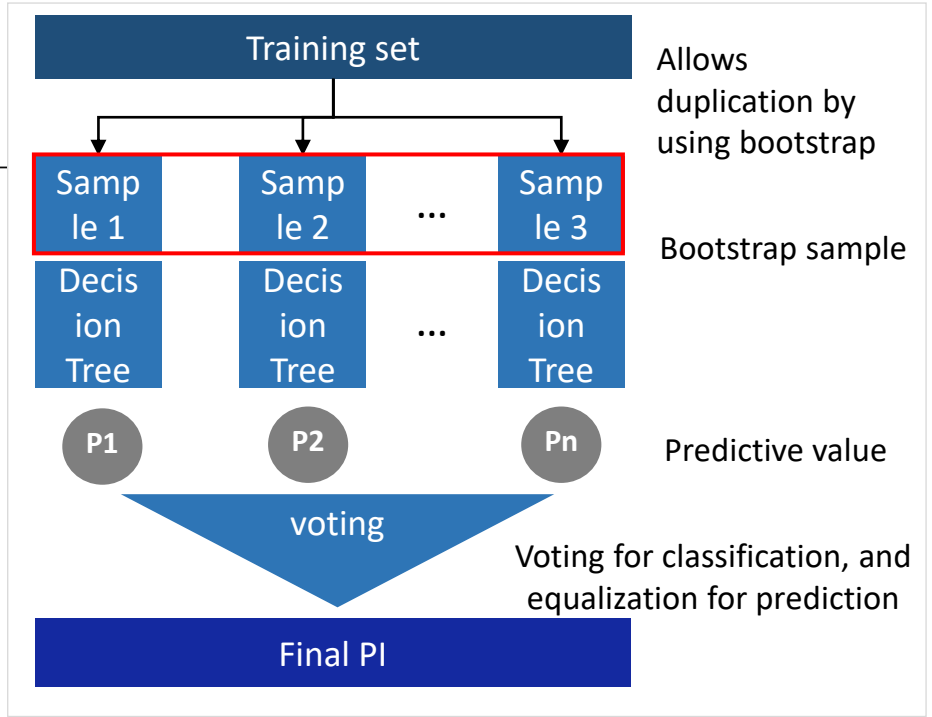
Bagging: delving into the algorithm

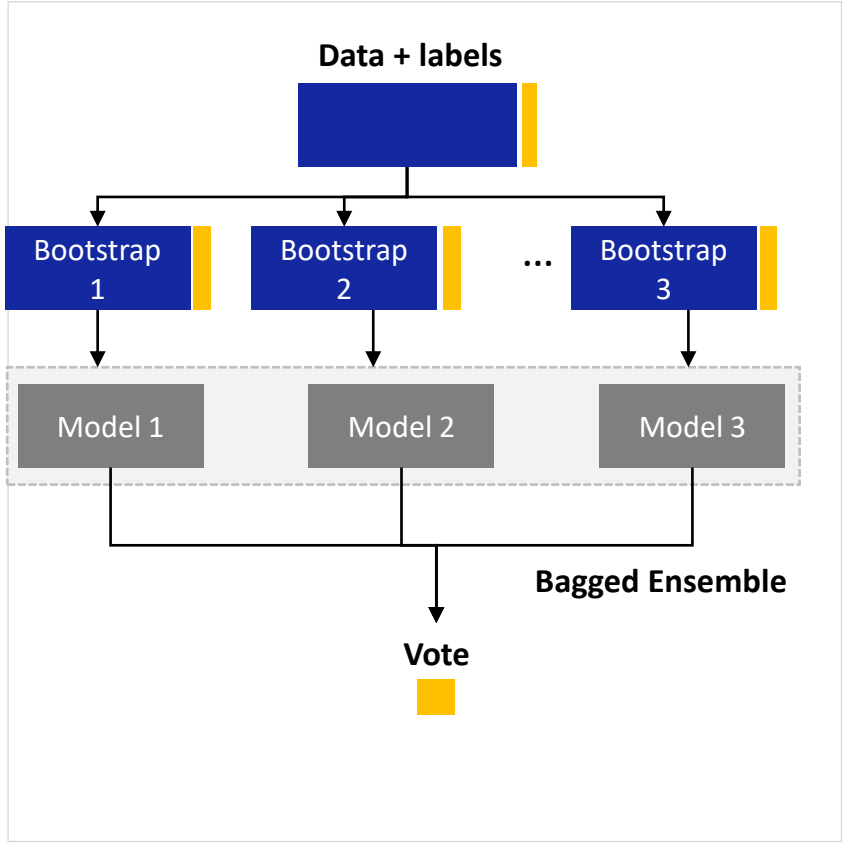
Algorithm 1 Bagging

Input: S: Training set; T: Number of iterations;
n: Bootstrap size; I: Weak learner

Output: Bagged classifier: $H(x) = sign\left(\sum_{t=1}^{T} h_t(x)\right)$ where $h_t \in [-1, 1]$ are the induced classifiers

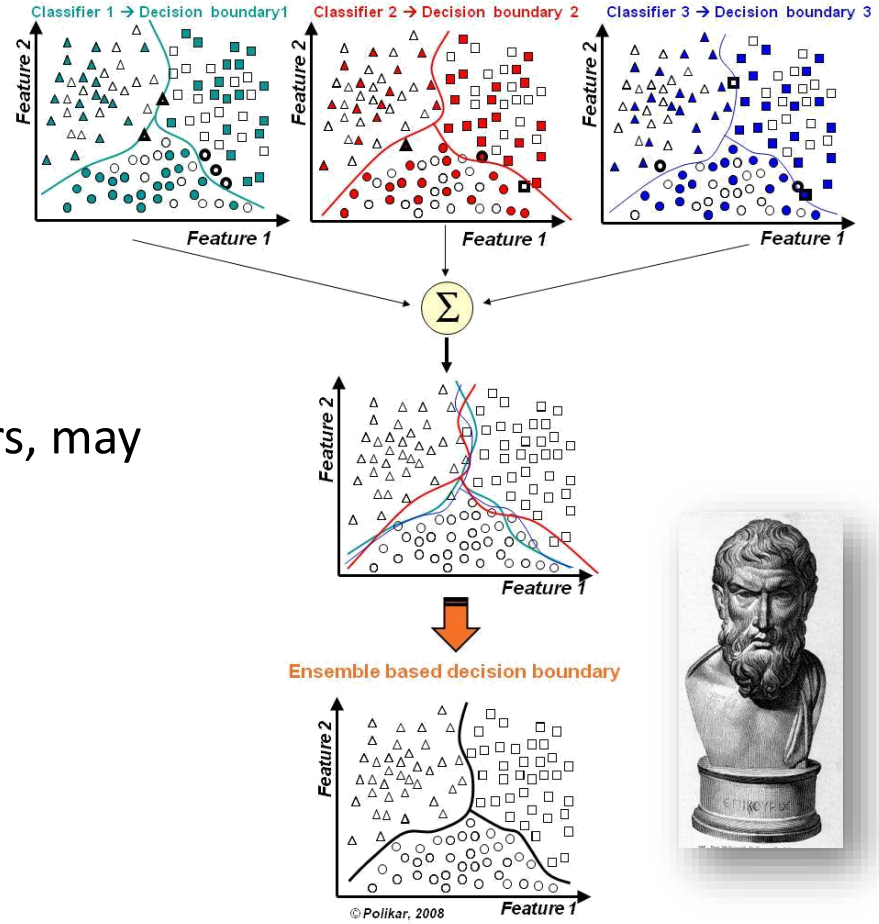
1: **for** $t = 1$ to T **do**
2: $S_t \leftarrow$ RandomSampleReplacement(n,S)
3: $h_t \leftarrow$ I(S_t)
4: **end for**





Bagging: when does it help?

- When learner is <u>unstable</u>
  - Small change to training set causes large change in the output classifier
  - True for decision trees, neural networks.
  - Not true for *k*-nearest neighbor, naïve Bayesian.
- Experimentally, bagging can help substantially for unstable learners, may somewhat degrade results for stable learners
- Diversity matters… a lot!
  - Epicurus' principle of multiple explanations
  - Can be achieved over several domains:
    - Training set sampling
    - Feature set sampling
    - Others (synthetic sample injection, hyper-parameter tuning…)



Bagging

- Bagging is an abbreviation of Bootstrap Aggregating.
  - Bootstrap = Sample
  - Aggregating = Adding up
- Bootstrap refers to a method that allows overlapping of different data sets for sampling and splitting.

Ex **Random Forest** is a typical bagging method algorithm.

- It creates multiple decision trees and performs sampling of different data sets while allowing overlapped data sets.
- If the data set consists of [1, 2, 3, 4, 5]:
  - Group 1 = [1, 2, 3]
  - Group 2 = [1, 3, 4]
  - Group 3 = [2, 3, 5]
- This is the bootstrap method. In the classification problem, voting is done on each tree trained with different sampling for the final prediction result.
- In regression problems, the average of each obtained value is calculated.

Bagging

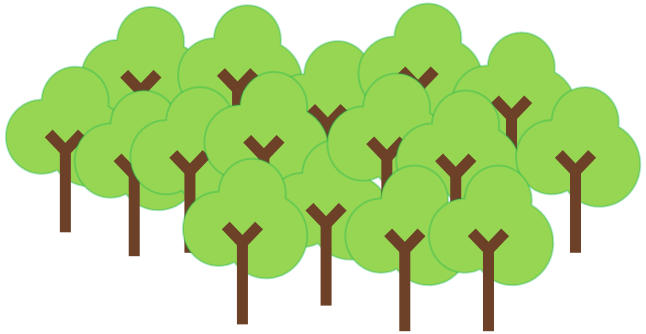
- Differences Between Bagging and Voting
  - The greatest difference between the bagging and voting methods is whether to use multiple single algorithms or apply various algorithms to the same sample data set.
  - In general, the bagging method is to train a single algorithm with different sampling data sets and perform voting. It has relatively better usability than the voting method because it uses a single algorithm. Thus, what is important is the hyperparameter of the single algorithm.
  - Taking the Random Forest algorithm as an example again, it is suitable to obtain the baseline score as it requires simple hyperparameter setting such as how many trees to use (n_estimators), maximum depth (max_depth), and the minimum number of samples for splitting (min_samples_leaf).
- Advantages of the Bagging Ensemble
  - When using the bagging method, it can reduce variance compared to making a prediction with a single model. There are three major training errors in a model: variance, noise, and bias. (Of course, more significant factors include overfitting/underfitting and many different preprocessing issues, but assume they are already being set.)
  - The ensemble method reduces variance, thus enhancing the performance of the final result.

Bagging

- sklearn.ensemble.BaggingClassifer/BaggingRegressor
  - The sklearn library package provides the wrapper class called BaggingClassifier/BaggingRegressor.
  - When designating the base algorithm to the base_estimator parameter, the BaggingClassifier/BaggingRegressor performs bagging ensemble.

Random Forest algorithm

1. Make trees with randomly selected variables and observations.
2. Keep only those with the lowest Gini impurity (or entropy).
3. Repeat from step 1) a given number of times.
4. Using the trees gathered during the training step, we can make predictions by majority vote.



Bagging

- Scikit-Learn RandomForestClassifier/Regressor Hyperparameters:

Hyperparameter	Explanation
n_estimators	The number of trees in the forest
max_depth	The maximum depth of a tree
min_samples_leaf	The minimum number of sample points required to be at a leaf node
min_samples_split	The minimum number of sample points required to split an internal node
max_features	The number of features to consider when looking for the best split

- Except for "n_estimators," the rest are analogous to those of DecisionTreeClassifier/Regressor.
- More information can be found at:

Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestRegressor.html

Table of contents

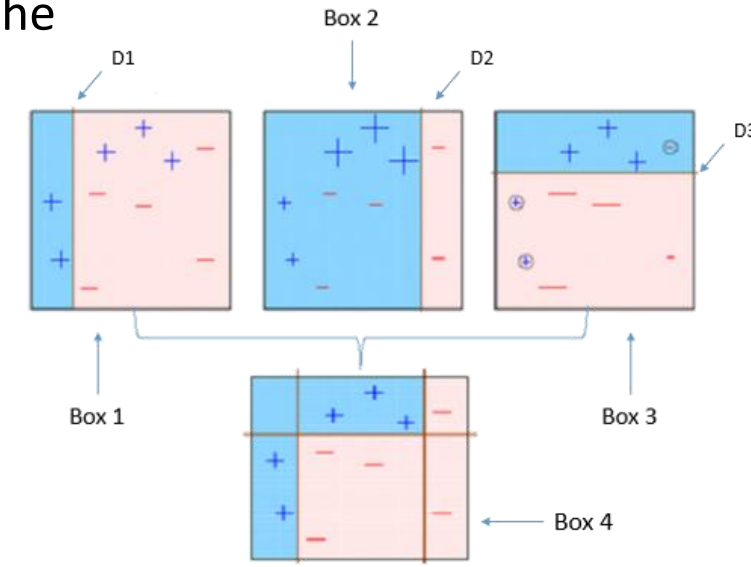
Boosting

- Examples are given weights. At each iteration, a new hypothesis is learned and the examples are reweighted to focus the system on examples that the most recently learned classifier got wrong.

- General Loop:
  - Set all examples to have equal uniform weights.
  - For t from 1 to T do:
    - Learn a hypothesis, ht, from the weighted examples
    - Decrease the weights of examples ht classifies correctly

- Base (weak) learner must focus on correctly classifying the most highly weighted examples while strongly avoiding over-fitting.

- During testing, each of the T hypotheses get a weighted vote proportional to their accuracy on the training data.



Boosting Ensemble: AdaBoost

- A sequence of weak learners such as trees
- A weighted ensemble of weak learners
  - One set of weights that increase the importance of the better-performing learners
  - One set of weights that increase the importance of the wrongly classified observations
- Similar pros and cons as the Random Forest

- AdaBoost Classification
  - Let's suppose n observations for the training step: $\boldsymbol{x}i$ and yi.
    We also suppose that $yi \in \{-1,+1\}$. (binary y)
  - We will make a series of weak learners $Gm\ (\boldsymbol{x})$ with $m= 1,\ldots,M$.
  - The ensemble classifier is made up of a linear combination of these weak learners:

$$G_{ensemble}(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

where α_m are the "boosting weights" that need to be calculated

  - Heavier weight is given to a better performing learner.

AdaBoost Classification

1) For the first step (m=1), equal weight is assigned to the observations: $w_i^{(1)}=\frac{1}{n}$

2) For the boost sequence m=1,…, M:

a) Train the learner $G_m(\boldsymbol{x})$ using observations weighted by $w_i^{(m)}$.

b) Calculate the error ratio: $$\varepsilon_m = \frac{\sum_{i=1}^{n} w_i^{(m)} I(y_i \neq G_m(\boldsymbol{x}_i))}{\sum_{i=1}^{n} w_i^{(m)}}$$

⇨ $I(y_i \neq G_m(\boldsymbol{x}_i))$ gives 1 for an incorrect prediction, else 0.

⇨ $0 \leq \varepsilon_m \leq 1$

AdaBoost Classification

3) For the boost sequence m=1,…, M:

c) Calculate the boosting weight: $$\alpha_m = \frac{1}{2} log\left(\frac{1-\varepsilon_m}{\varepsilon_m}\right)$$

⇨ As $\varepsilon_m \to 0$, α_m is a large positive number. The learner is given more importance!

⇨ As $\varepsilon_m \cong 0.5$, $\alpha_m \cong 0$.

⇨ As $\varepsilon_m \to 1$, α_m is a large negative number.

d) For the next step, the weights of the **wrongly** predicted observation are rescaled by a factor $e^{a}{}_{m}$.

This can be compactly expressed as: $$w_i^{(m+1)} = w_i^{(m)} \times e^{\alpha_m \cdot I\left(y_i \neq G_m(x_i)\right)}, \text{ where } i = 1, \dots, n$$

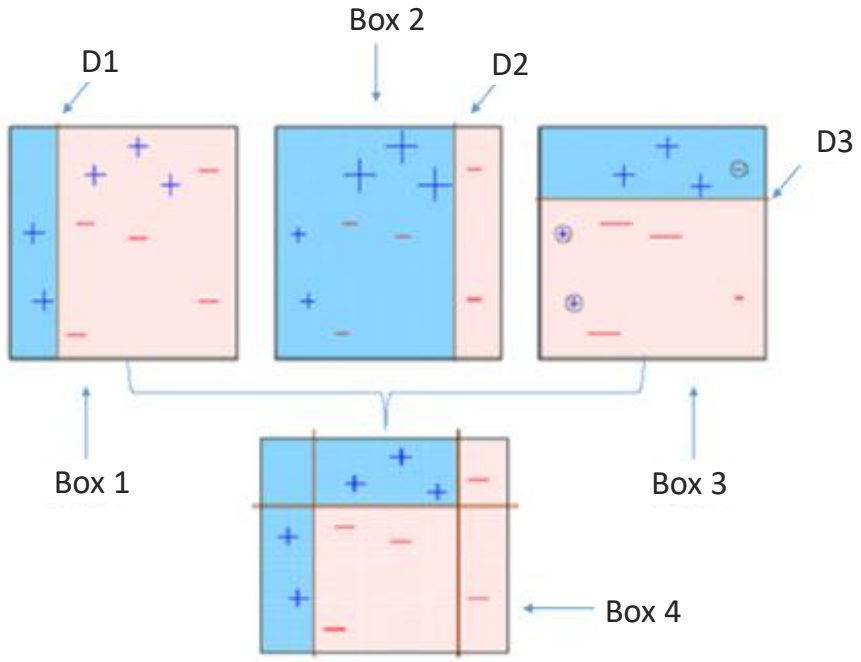
⇨ In the next sequence step, the wrongly predicted observations receive heavier weight.

AdaBoost Classification

4) The ensemble classifier is made up of a linear combination of weak learners:

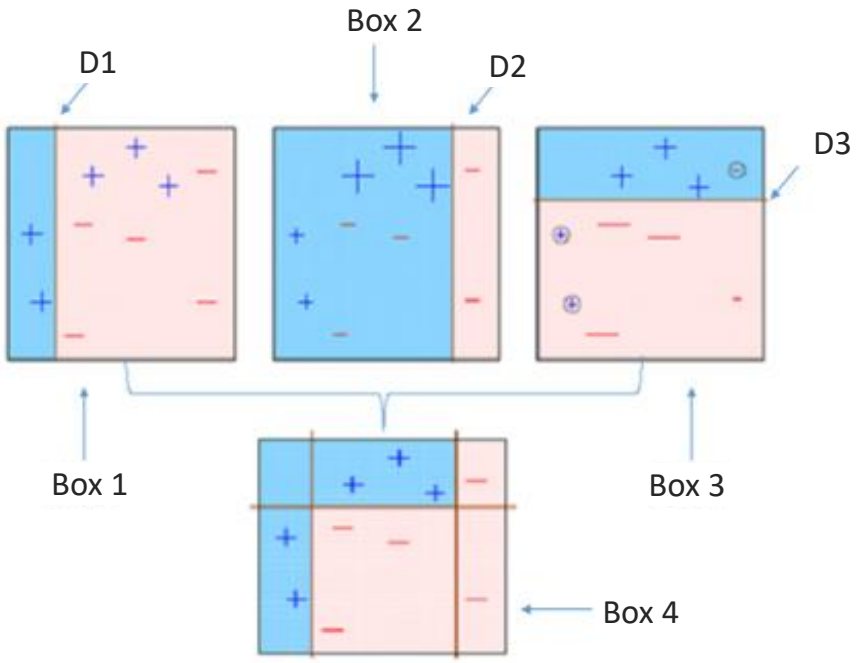
$$G_{ensemble}(\boldsymbol{x}) = sign\left(\sum_{m=1}^{M} \alpha_m G_m(\boldsymbol{x})\right)$$

5) For a new testing condition $\boldsymbol{x}'$, we can predict y' by $G\_ensemble\ (\boldsymbol{x}')$.



- Box 1 results from a weak learning machine classified as the D1 sector. However, the error rate is relatively high because some data sets are expressed in + distributed in the red sector.
- The D2 line in Box 2 moves to the right to supplement the error rate from Box 1. Here, the data sets expressed in – are distributed in the blue sector for better performance, but it is not satisfactory yet.
- The D3 line in Box 3 is horizontally drawn on the top. However, the data set expressed in – is incorrectly classified.
- By training the previous Box 1, 2, and 3, Box 4 finds the most ideal combination. It shows much better performance compared to the previous three individual learning machines.

AdaBoost Classification



- How is the weight applied for combination?

Ex Suppose that the following weight will be applied to the performance of Box 1~3.

- Performance of Box 1: weight = 0.2
- Performance of Box 2: weight = 0.5
- Performance of Box 3: weight = 0.6

It can be expressed as the following formula:

$$0.2 * \text{Box 1} + 0.5 + \text{Box 2} + 0.6 * \text{Box 3} = \text{Box 4}$$

Scikit-Learn AdaBoostClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
base_estimator	The base estimator with which the boosted ensemble is built
n_estimators	The maximum number of estimators at which boosting is terminated
learning_rate	The rate by which the contribution of each learner is shrunken
algorithm	Either 'SAMME' or 'SAMME.R'

- "base_estimator" is by default None, which means DecisionTreeClassifier(max_depth=1).
- More information can be found at:

  Classifier: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

  Regressor: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostRegressor.html

Boosting Ensemble: GBM & XGBoost

- Gradient Boosting Machine (GBM)
  - GBM is also made up of a sequence of weak learners similar to AdaBoost.
  - The disagreement between the predicted $\hat{y}$ and the true y can be represented by a "loss" function.
  - In the sequence of weak learners, the learner at step m, $Gm(\boldsymbol{x})$ can be obtained by moving the previous step learner $G\_(m-1)$ $(\boldsymbol{x})$ towards the direction that decreases the loss as defined above.
  - These updating movements are restricted to a set of base learners.
  - The sequence of weak learners is driven towards a better predictive performance by application of gradients.

GBM: Gradient descent

- The key to the boosting method is to supplement errors from the previous learning.
- The AdaBoosting and gradient descent methods are slightly different in how to supplement errors.
- Gradient descent uses differentiation to minimize the difference between the predicted value and actual data.
  - Weight
  - Input_data = feature data (input data)
  - Bias
  - Y_actual = actual data value
  - Y_predict = predicted value
  - Loss = error rate
- Y_predict = weight*input_data+bias
  - The predictive value can be obtained from the above formula.
    Calculating the difference with actual data will result in the total error rate.
- Loss = Y_predict – Y_actual
  - (There are many different functional formulas to define error rate, including root means square error and means absolute error, but the above definition is provided for convenience.)
  - The purpose of gradient descent is to find the weight that makes the loss closest to 0.

Extreme Gradient Boosting (XGBoost)

dmlc XGBoost

- Improves upon GBM in the execution speed.
- More resistant to the overfitting than GBM.
- Not included in the Scikit-Learn library. Requires installation of the “xgboost” library.
- XGBClassifier/Regressor Hyperparameters

Hyperparameter	Explanation
booster	gbtree or gblinear
n_estimators	The number of boosting steps (weak learners)
learning_rate	The contribution of each weak learner
subsample	The fraction of data that will be used by individual weak learner

- Need to be tuned for optimized performance.
- More information can be found at: https://xgboost.readthedocs.io/en/latest/python/index.htm

Table of contents

Practical Notes

- Ensembles-1.ipynb
  - Compare the Tree-like algorithms
    i. Read in data and explore
    ii. Data pre-processing
    iii. Classification with Tree (optimized hyperparameters)
    iv. Classification with Random Forest (optimized hyperparameters)
    v. Classification with AdaBoost (optimized hyperparameters)
- Ensembles-2.ipynb
  - *Voting Ensemble*
    i. Read in data
    ii. Predicting with individual estimator
    iii. Predicting with a voting ensemble
- Ensembles-3.ipynb
  - *A. Bagging ensemble*
    i. Read in data
    ii. Random Forest
    B. Boosting Ensemble
    i. AdaBoost
    ii. Gradient Boosting
    iii. XGBoost