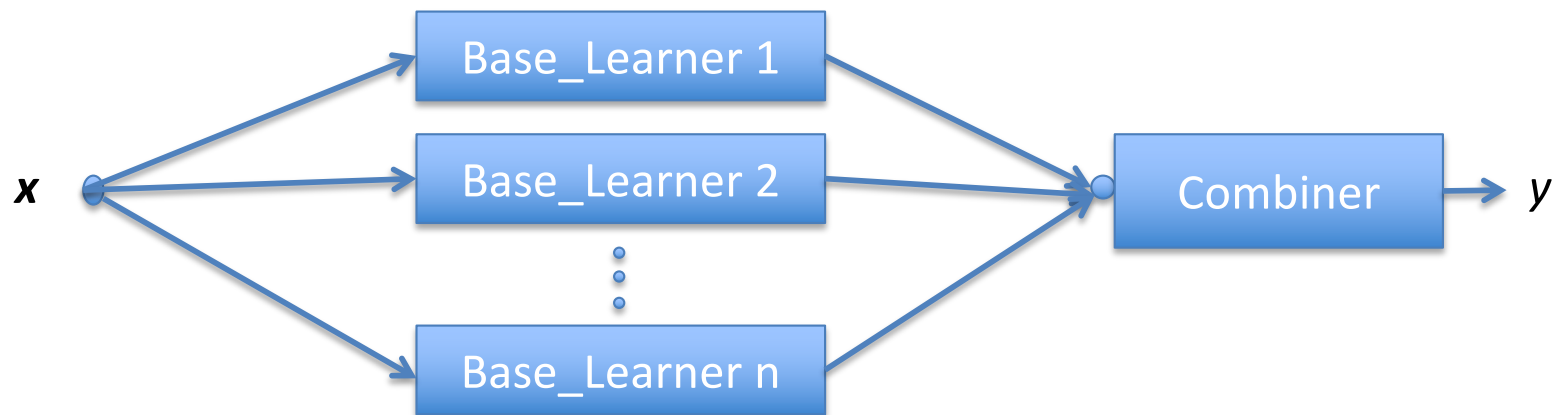


Ensemble Methods

Ishwar K Sethi

Ensemble Methods

- These methods train multiple learners for the same task
- These methods are also known as *committee-based methods*, or *multiple classifier systems*



Homogeneous vs. Heterogeneous Ensembles

- Homogeneous ensembles have all base learners of same type, e.g., decision trees
- Heterogeneous ensembles use a variety of base learners
- Base learners are also known as *weak learners* because these learners exhibit performance only slightly better than random guess. However, together they perform very well



Why a Collection of Weak Learners Performs Well?

- Suppose we have 5 completely independent learners each with an accuracy of 70%. We combine their decision using the majority rule. Then

- $(.7^5) + 5(.7^4)(.3) + 10(.7^3)(.3^2)$
- **83.7% majority vote accuracy**
- 101 such classifiers
- **99.9% majority vote accuracy**

Wisdom of crowds

Note: Binomial Distribution: The probability of observing x heads in a sample of n independent coin tosses,

where in each toss the probability of heads is p , is

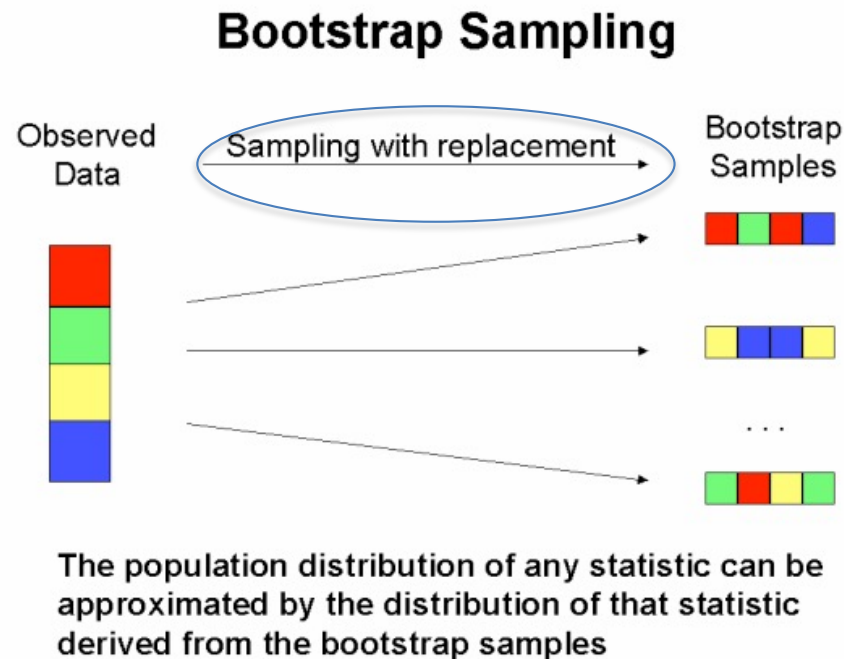
$$P(X = x|p, n) = \frac{n!}{x!(n-x)!} p^x (1-p)^{(n-x)}$$

How Do We Train Weak Learners?

- These learners must not be strongly correlated. Rather they should compliment each other. Why?
- Possible training approaches:
 - Different algorithms for different learners (Heterogeneous ensemble)
 - Same algorithm but different parameters
 - Different features for different learners (Random subspace)
 - Different data subsets for different learners (Bagging and boosting)

Bagging

- Bagging is an abbreviation of *Bootstrap Aggregating* (Breiman 1996)
- It uses bootstrap sampling to generate different data subsets for training different learners
- The output of different learners is combined using voting for classification and averaging for regression



Bootstrap Example using Sklearn

```
from sklearn.utils import resample
data = [1,2,5,7,9,13,15, 19]
#Perform bootstrap sampling
sample1 = resample(data, replace=True, n_samples=5, random_state=11)
print('Bootstrap Sample: %s', sample1)
```

Bootstrap Sample: %s [2, 19, 1, 7, 2]

Note the second data item appears twice in the bootstrap sample

Bagging

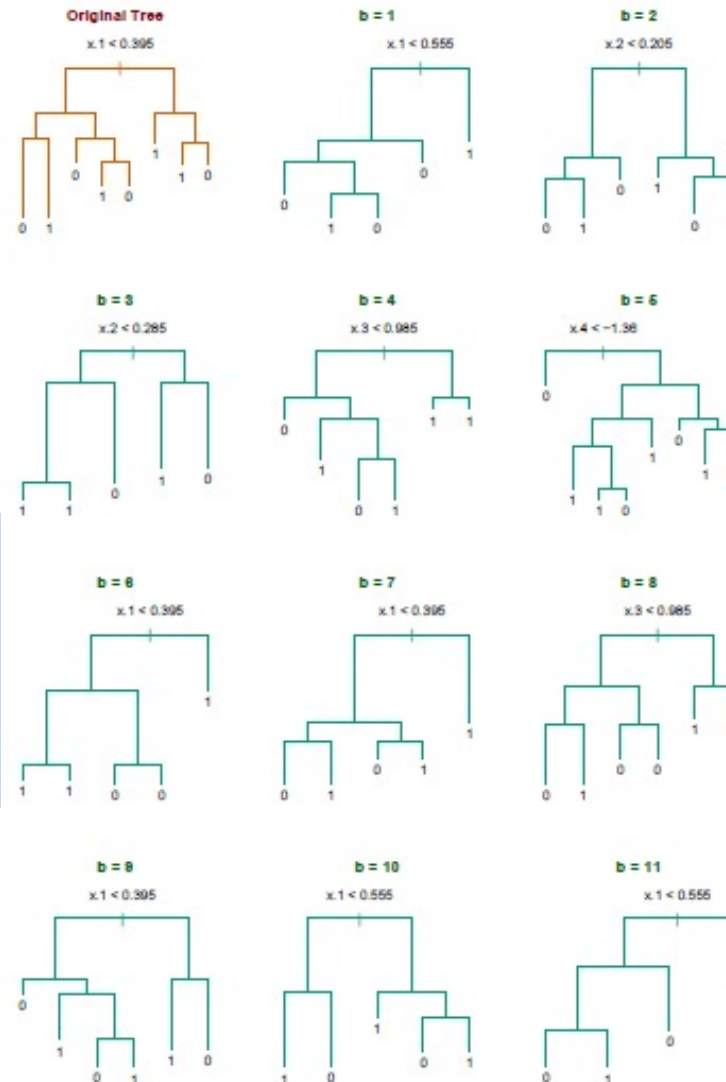
- Given m training examples, the probability that the i^{th} example is selected k times is approximated by Poisson distribution with $\lambda = 1$

$$p(k, \lambda) = e^{-\lambda} \lambda^k / k! \text{ for } k = 0, 1, 2, \dots$$

- Thus, the probability that the i^{th} example will occur at least once is $1 - (1/e) \approx 0.632$. This means that each base learner uses only about 63.2% of the original training examples. Thus, the remaining 36.8% of the examples, called **out-of-bag** examples, not used by a base learner can be used as test examples for evaluating its performance. Such an estimate is called **out-of-bag estimate**.

Bagging

- Some learners are more susceptible to addition or deletion of training examples. These kinds of learners, for example decision trees, are known as *unstable learners*.
- Learners such as k -NN classifier are hardly affected by some additions/deletions of training examples. Such learners are called *stable learners*.
- Bagging works well with unstable learners.



Tree pruning is not desirable because un-pruned trees are more unstable and consequently better Suited for bagging

Simulated data: sample of size $N = 30$, with two classes and 5 features, each having a standard Gaussian distribution with pairwise correlation 0.95. The response Y was generated according to $\Pr(Y = 1 | x_1 \leq 0.5) = 0.2$, $\Pr(Y = 1 | x_1 > 0.5) = 0.8$. The Bayes error is 0.2. A test sample of size 2000 was also generated from the same population. We fit classification trees to the training sample and to each of 200 bootstrap samples

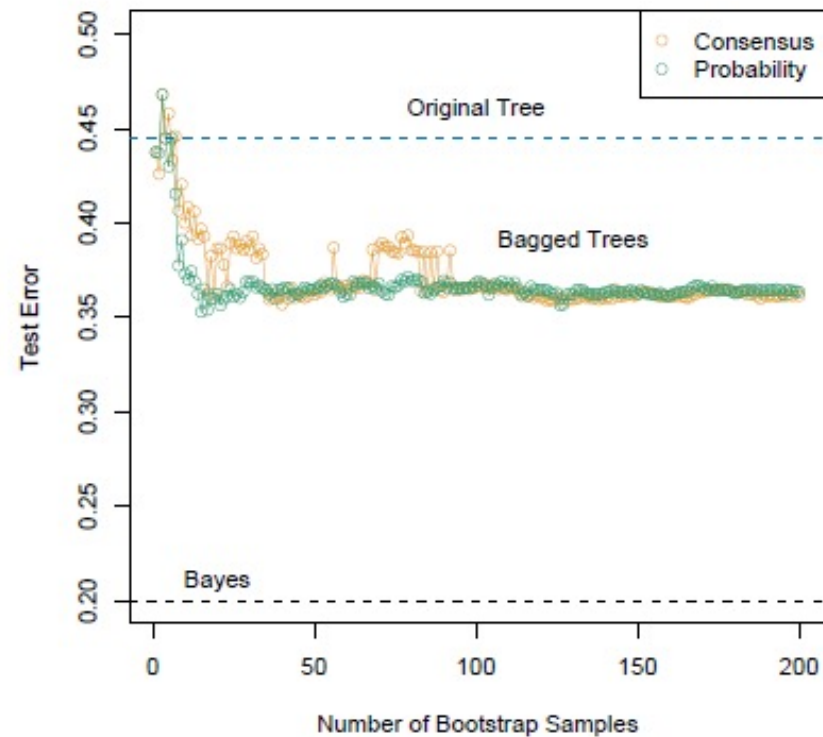


FIGURE 8.10. Error curves for the bagging example of Figure 8.9. Shown is the test error of the original tree and bagged trees as a function of the number of bootstrap samples. The orange points correspond to the consensus vote, while the green points average the probabilities.

```
import pandas as pd
from sklearn.tree import DecisionTreeClassifier
from sklearn.model_selection import train_test_split
from sklearn import metrics
```

```
col_names = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age', 'label']
pima = pd.read_csv("https://raw.githubusercontent.com/jbrownlee/Datasets/master/pima-indians-diabetes.data.csv",
                  header=None, names=col_names)
```

```
#split features and target variable
feature_cols = ['pregnant', 'glucose', 'bp', 'skin', 'insulin', 'bmi', 'pedigree', 'age']
X = pima[feature_cols]
y = pima.label
```

```
#Split dataset to create train and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.1, random_state=3)
```

```
clf = DecisionTreeClassifier(max_depth=3)
classifier = clf.fit(X_train, y_train)
y_pred = classifier.predict(X_test)
print('Accuracy:', metrics.accuracy_score(y_test, y_pred))
```

Accuracy: 0.5844155844155844

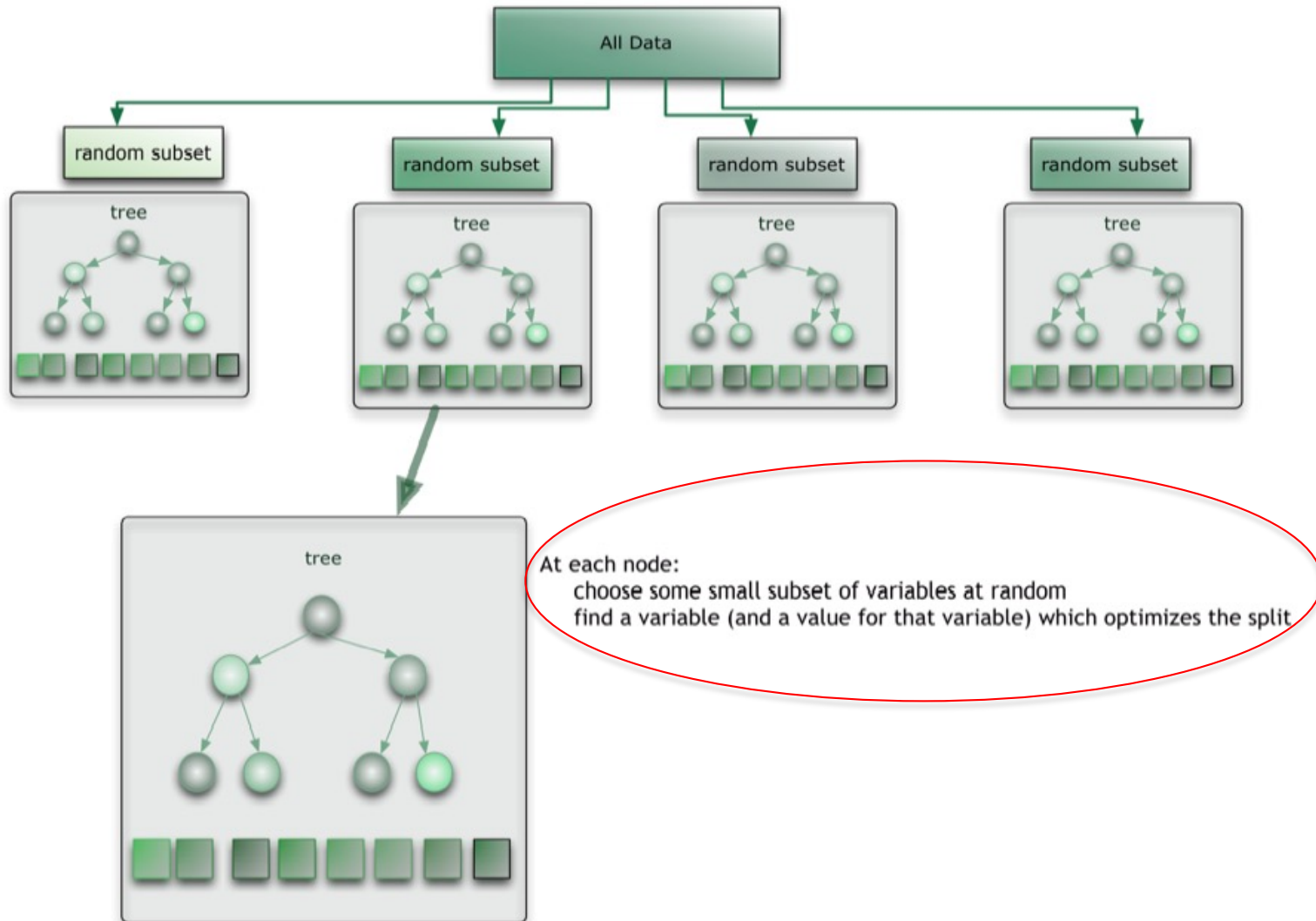
```
# Now lets apply bagging
from sklearn.ensemble import BaggingClassifier
for i in range (10, 110, 10):
    clf = BaggingClassifier(n_estimators=i, random_state=3).fit(X_train, y_train)
    y_pred = clf.predict(X_test)
    print('Number of Learners:', i, 'Accuracy:', metrics.accuracy_score(y_test, y_pred))
```

```
Number of Learners: 10 Accuracy: 0.6233766233766234
Number of Learners: 20 Accuracy: 0.6883116883116883
Number of Learners: 30 Accuracy: 0.6883116883116883
Number of Learners: 40 Accuracy: 0.6753246753246753
Number of Learners: 50 Accuracy: 0.6623376623376623
Number of Learners: 60 Accuracy: 0.6623376623376623
Number of Learners: 70 Accuracy: 0.6623376623376623
Number of Learners: 80 Accuracy: 0.6623376623376623
Number of Learners: 90 Accuracy: 0.6623376623376623
Number of Learners: 100 Accuracy: 0.6883116883116883
```

Random Forest

- Random Forest (RF) (Breiman, 2001)
- RF is an extension of bagging with decision trees. In RF, a random subset of features is used to select splits at each tree node. This is in addition to bootstrap sampling to create different data subsets.

Random Forest



Random Forest Performance: Pima Indians Data

```
from sklearn.ensemble import RandomForestClassifier
rf = RandomForestClassifier(n_estimators=50,
                           random_state=0)
rf.fit(X_train, y_train)
rf.score(X_train, y_train)
1.0
rf.score(X_test, y_test)
0.6753246753246753
```

Boosting

- Boosting refers to any procedure that combines many weak learners to yield a much higher performance. Unlike bagging, the base learners in boosting are sequentially generated by focusing on examples misclassified by earlier weak learners in the chain.

Key Idea of Boosting

- Each training example is assigned a weight. The weight determines the training example's probability of being selected for training by the next base learner.
- Initially, all examples have identical weight. The misclassified examples see their weight go up to increase their selection chance for training
- The outputs of the learners are combined using weights to yield the final response

Boosting: Basic Algorithm

- General Loop:

Set all examples to have equal uniform weights.

For t from 1 to T do:

 Learn a hypothesis, h_t , from the weighted examples

 Decrease the weights of examples h_t 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.

AdaBoost Pseudocode

TrainAdaBoost(D , BaseLearn)

For each example d_i in D let its weight $w_i = 1/|D|$

Let H be an empty set of hypotheses

For t from 1 to T do:

 Learn a hypothesis, h_t , from the weighted examples: $h_t = \text{BaseLearn}(D)$

 Add h_t to H

 Calculate the error, ϵ_t , of the hypothesis h_t as the total sum weight of the examples that it classifies incorrectly.

 If $\epsilon_t > 0.5$ then exit loop, else continue.

 Let $\alpha_t = \epsilon_t / (1 - \epsilon_t)$

 Multiply the weights of the examples that h_t classifies correctly by α_t

 Rescale the weights of all of the examples so the total sum weight remains 1.

Return H

TestAdaBoost(ex , H)

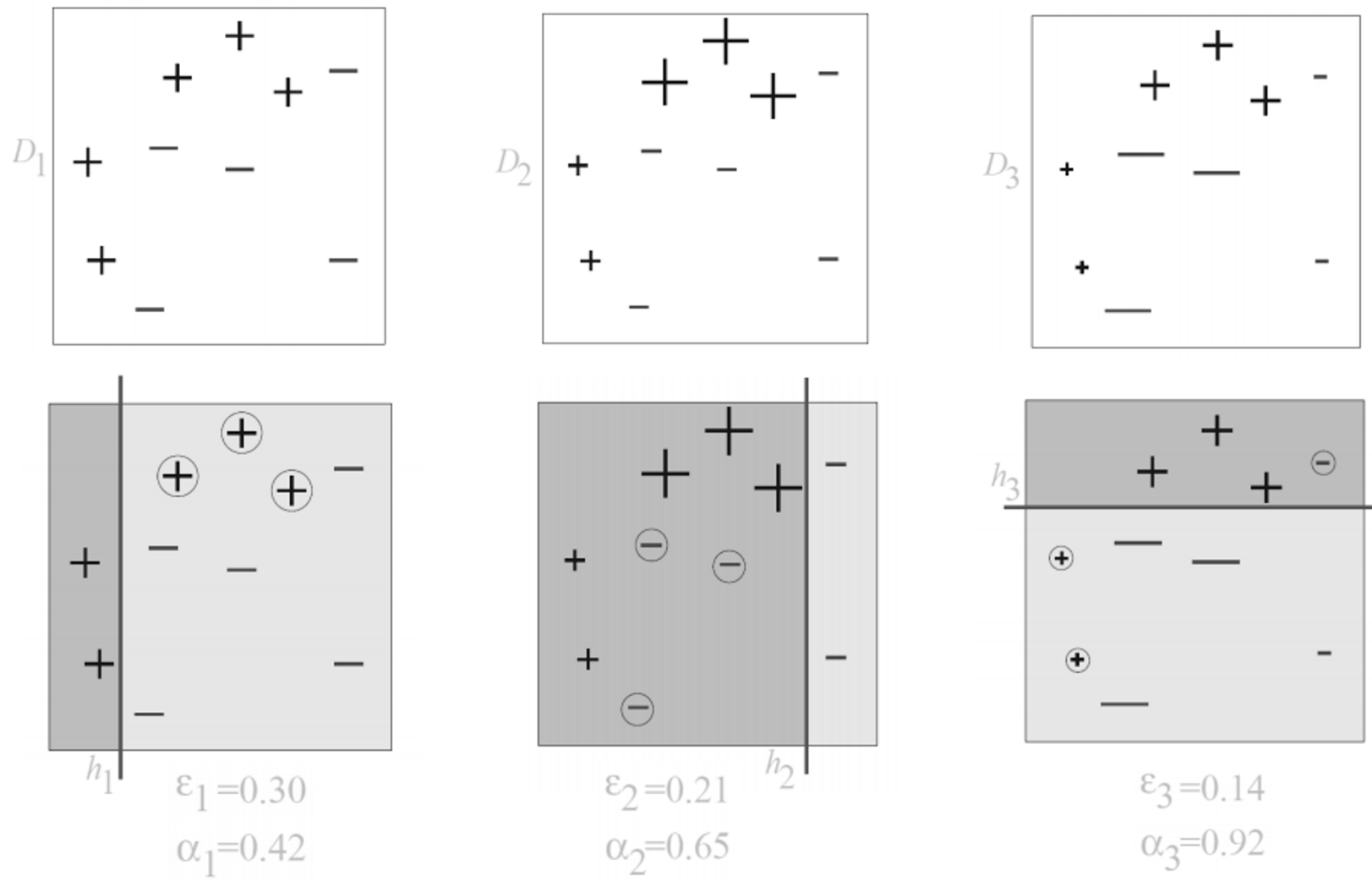
 Let each hypothesis, h_t , in H vote for ex 's classification with weight $\log(1/\alpha_t)$

 Return the class with the highest weighted vote total.


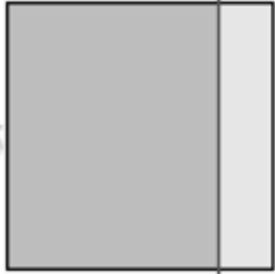

How is Example Weighting Used?

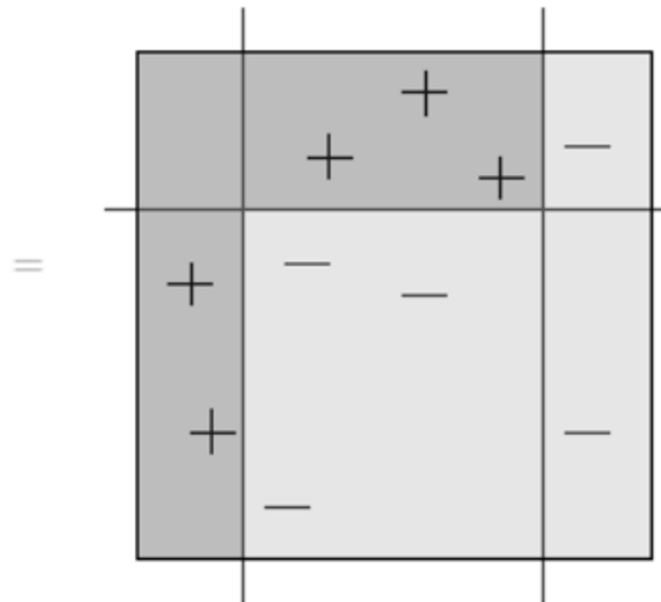
- Generic approach is to replicate examples in the training set proportional to their weights (e.g., 10 replicates of an example with a weight of 0.01 and 100 for one with weight 0.1).
- Most algorithms can be enhanced to efficiently incorporate weights directly in the learning algorithm so that the effect is the same. For example, when calculating information gain with decision tree learning, simply increment the count by w_i for example i rather than by 1.

Visual Illustration of AdaBoost



The final boundary is a weighted linear sum of different learners weighted by the respective α values

$$H_{\text{final}} = \text{sign} \left(0.42 \begin{array}{|c|} \hline \text{[Diagram 1]} \\ \hline \end{array} + 0.65 \begin{array}{|c|} \hline \text{[Diagram 2]} \\ \hline \end{array} + 0.92 \begin{array}{|c|} \hline \text{[Diagram 3]} \\ \hline \end{array} \right)$$






Combining Weak Learners

- Averaging (simple or weighted) for regression
- Voting (simple or weighted) for classification tasks
- Stacking. Here the combiner is trained to combine the output of weak learners. The training data for this is different from the data used for weak learners. The combiner in stacking is often called the *second-level learner* or *meta learner*. The weak learners are called the *first-level learners*.

Gradient Boosted Trees

- The gradient boosting is another way of building a sequence of trees whose outputs are added to obtain the final prediction.
- The sequence of trees are really stumps or trees with depth of 2.
- Unlike the AdaBoost where the training examples vary in their importance, the successive trees in the gradient boosted algorithm are generated by minimizing a loss function.

Steps for Constructing an Ensemble of GBTs

- **Step1:** Construct a base tree with single root node. It is the initial guess for all the samples.
- **Step2:** Build a tree from errors of the previous tree.
- **Step3:** Scale the tree by learning rate (value between 0 and 1). This learning rate determines the contribution of the tree in the prediction
- **Step4:** Combine the new tree with all the previous trees to predict the result and repeat step 2 until maximum number of trees is achieved or until the new trees don't improve the fit.
- The final prediction model is the combination of all the trees.

GBT Illustration

We will first look at the regression problem consisting of 3 predictors and one output variable.

Height	Age	Gender	Weight
5.4	28	Male	88
5.2	26	Female	76
5	28	Female	56
5.6	25	Male	73
6	25	Male	77
4	22	Female	57

We will construct our first tree with only one output node. We will use the average of the Weight column as the output of this node. The average is 71.2.

GBT Illustration

Step2 is to build a tree based on errors from previous tree. The errors that the previous tree made is the difference between the actual weight and the predicted weight. This difference is called residual or pseudo residual.

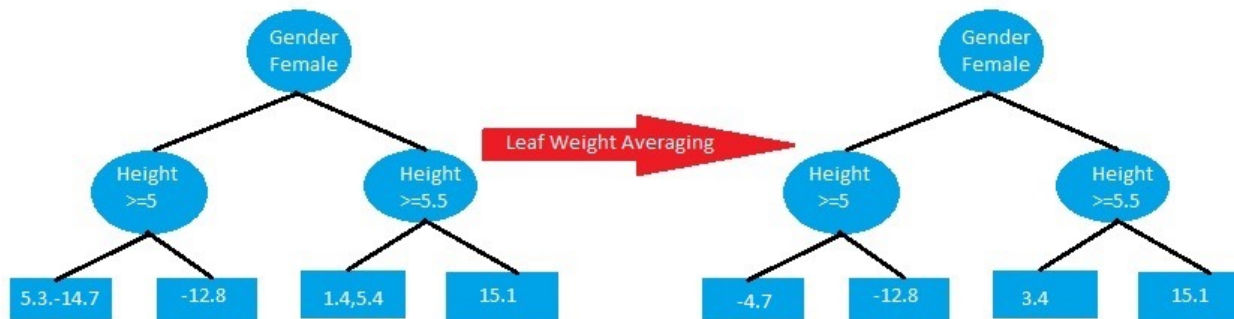
Height	Age	Gender	Weight	Predicted Weight 1	Pseudo Residuals 1
5.4	28	Male	88	71.2	$88 - 71.2 = 16.8$
5.2	26	Female	76	71.2	$76 - 71.2 = 4.8$
5	28	Female	56	71.2	$56 - 71.2 = -15.2$
5.6	25	Male	73	71.2	$73 - 71.2 = 1.8$
6	25	Male	77	71.2	$77 - 71.2 = 5.8$
4	22	Female	57	71.2	$57 - 71.2 = -14.2$



GBT Illustration

Step 3 is scaling tree with learning rate. Assuming the learning rate as 0.1.
Step 4 is combining the trees to make the new prediction. So, we start with initial prediction 71.2 and run the sample data down the new tree and sum them. Next, get the new set of values for the residuals.

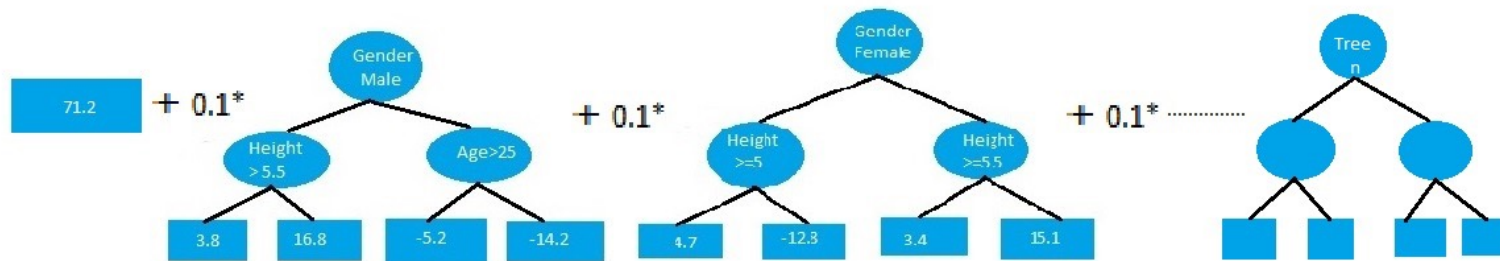
Height	Age	Gender	Weight	Predicted weight 2	Pseudo Residuals2
5.4	28	Male	88	$71.2 + 0.1 * 16.8 = 72.9$	$88 - 72.9 = 15.1$
5.2	26	Female	76	$71.2 + 0.1 * (-5.2) = 70.7$	$76 - 70.7 = 5.3$
5	28	Female	56	$71.2 + 0.1 * (-5.2) = 70.7$	$56 - 70.7 = -14.7$
5.6	25	Male	73	$71.2 + 0.1 * 3.8 = 71.6$	$73 - 71.6 = 1.4$
6	25	Male	77	$71.2 + 0.1 * 3.8 = 71.6$	$77 - 71.6 = 5.4$
4	22	Female	57	$71.2 + 0.1 * (-14.2) = 69.8$	$57 - 69.8 = -12.8$



GBT Illustration

Now we combine the new tree with all the previous trees to predict the new weights and get the residuals and a new tree.

Height	Age	Gender	Weight	Predicted Weight 3
5.4	28	Male	88	$71.2 + 0.1 * 16.8 + 0.1 * 15.1 = 74.4$
5.2	26	Female	76	$71.2 + 0.1 * (-5.2) + 0.1 * (-4.7) = 70.2$
5	28	Female	56	$71.2 + 0.1 * (-5.2) + 0.1 * (-4.7) = 70.2$
5.6	25	Male	73	$71.2 + 0.1 * 3.8 + 0.1 * 3.4 = 71.9$
6	25	Male	77	$71.2 + 0.1 * 3.8 + 0.1 * 3.4 = 71.9$
4	22	Female	57	$71.2 + 0.1 * (-14.2) + 0.1 * (-12.8) = 68.5$



Continuing this process to meet the desired goal, the process will terminate at some point.

GBTs

- For classification, we need to convert the qualitative output to quantitative output. This is needed because the underlying minimization procedure uses the gradient search.
- The conversion is done using the $\log(\text{odds})$, the average of classification labels. [<https://blog.paperspace.com/gradient-boosting-for-classification/>]
- There are a few other variations on GBTs. The XGBoost is the latest version which is fast and highly accurate. [<https://xgboost.ai>]

Ensembles' Performance

- Ensembles have been used to improve generalization accuracy on a wide variety of problems.
- On average, Boosting provides a larger increase in accuracy than Bagging.
- Boosting on rare occasions can degrade accuracy.
- Bagging more consistently provides a modest improvement.
- Boosting is particularly subject to over-fitting when there is significant noise in the training data.