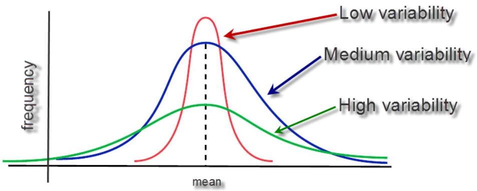
Ensemble methods

Def: An **ensemble method** is an approach that combines many simple "building ensemble block" models called **weak learners** obtain a single more powerful model.

Example: Supppose you have 100 different regression models $f_i(X)$. Then let $f^*(X) = (1/n)(f_1(X) + \cdots + f_n(X))$.

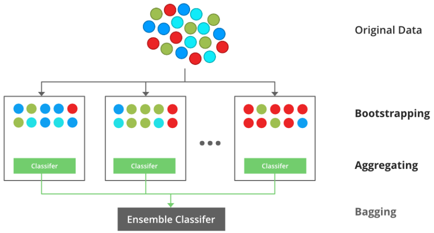
There are specialized ensemble methods for decision trees, which suffer from *high variance*:

- **high variance** - different samples can lead to very different outcomes
- **low variance** - different samples yield similar results



Bootstrap Aggregating

Key Point: Averaging a set of observations reduces variance.

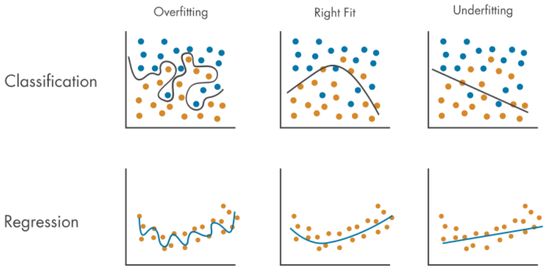


Suppse you have a (single) training sample $(x_1, y_1), \ldots, (x_n, y_n)$. Take B bootstrap samples from this dataset. For each b, compute a model $\hat{f}_b(x)$. Then the **bootstrap aggregation** (or **bagging**) model is

$$\hat{f}_{bag} = \frac{1}{B} \sum_{b=1}^{B} \hat{f}_b(x)$$

Bias-Variance Tradeoff

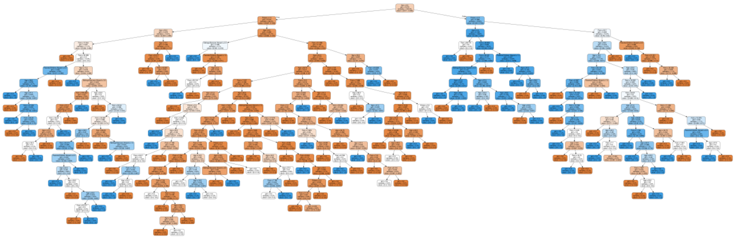
Key Point: There is a tension in building machine learning models that stems from different types of errors:



- **bias** stems from bad assumptions in the learning model. This is **underfitting**.
- **variance** (in test date) stems from a model learning the "noise" in the training data. This **overfitting**.

Issues with Decision Trees

Key Point: Decision Trees are particularly susceptible to *overfitting* . . .



This can be fixed by:

- **pruning** - reduce the size during construction (e.g., stop at a leaf if additional branching doesn't improve accuracy)
- **hyperparameter tuning** - place constraints before construction (e.g., max depth, min samples per leaf)
- **ensemble methods** - take "weak learners" and combine them together to make better learners

Bagging and Decision Trees

Regression Trees:

Procedure: Construct B regression trees using B bootstrapped training sets and average the resulting predictions.

Idea: These trees are “grown deep, and are not pruned”. The individual trees have high variance, but low bias. Average the trees to reduce the variance.

Classification Trees:

Procedure: Record the class predicted by each of the B trees, and take a majority vote. The overall prediction will be the most commonly occurring majority class among the B predictions.

Random Forests (Random Decision Trees)

A Problem: With bagging, a "strong predictor" will invariably end up in the top node. This means the predictors from a bagging ensemble will be "highly correlated", so averaging won't necessarily lead to the reduction in variance.

Idea: In addition to bootstrapping new samples of data, "decorrelate" the resulting trees by randomly *disallowing* certain nodes from being used at each decision.

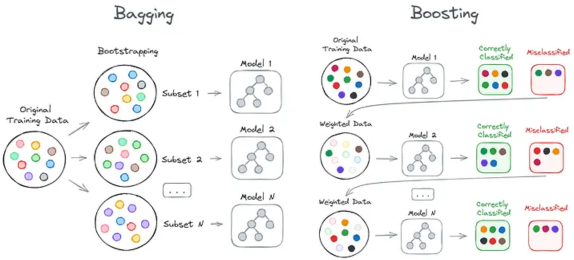
A **random forest** (trademark) performs "feature bagging" during the decision tree algorithm by selecting at each decision node a *random sample* of m feature variables that are allowed to be used to make the split.

Conventionally, $m \approx \sqrt{p}$, where p is the number of feature variables.

Boosting

Another approach for improving decision trees . . .

Idea: Bagging and random forests build decision trees *in parallel*. Boosting builds decision trees *sequentially*–i.e., starting with weak learners and adding to them at each step, by fitting a decision tree to the residuals (i.e., errors) in the existing model.



There are various implementations of this *strategy* . . .

Boosting Examples

SKLearn has boosting implementations for classification and regression:

- Adaboost

```
class sklearn.ensemble.AdaBoostClassifier(estimator=None, *,
n_estimators=50, learning_rate=1.0, random_state=None)
```

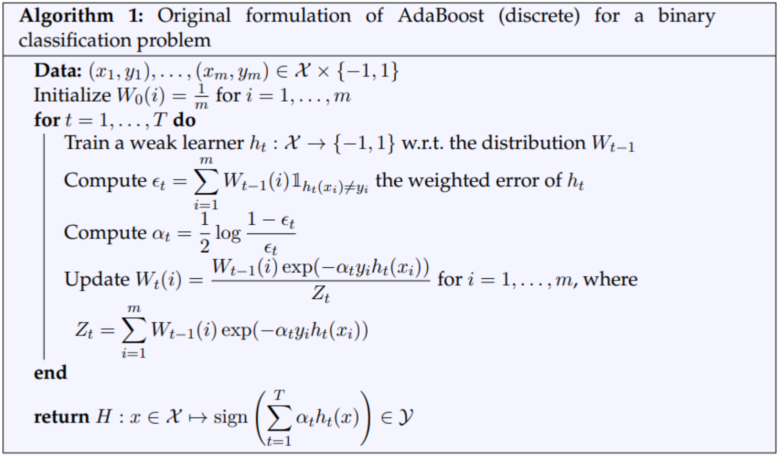
- Gradient Boosting

```
class sklearn.ensemble.GradientBoostingClassifier(*, loss='log_loss',
learning_rate=0.1, n_estimators=100, subsample=1.0, criterion='friedman_mse',
min_samples_split=2, min_samples_leaf=1, min_weight_fraction_leaf=0.0,
max_depth=3, min_impurity_decrease=0.0, init=None, random_state=None,
max_features=None, verbose=0, max_leaf_nodes=None, warm_start=False,
validation_fraction=0.1, n_iter_no_change=None, tol=0.0001, ccp_alpha=0.0)
```

Famous boosting algorithms like XGBoost and LightGBM are alternative implementations of gradient boosting and are available in external libraries.

These are generally regarded as the best algorithms for *tabular data*.

The Adaboost Algorithm



Algorithm 1: Original formulation of AdaBoost (discrete) for a binary classification problem

```
Data: $(x_1, y_1), \dots, (x_m, y_m) \in \mathcal{X} \times \{-1, 1\}$
Initialize $W_0(i) = \frac{1}{m}$ for $i = 1, \dots, m$
for $t = 1, \dots, T$ do
    Train a weak learner $h_t : \mathcal{X} \to \{-1, 1\}$ w.r.t. the distribution $W_{t-1}$
    Compute $\epsilon_t = \sum_{i=1}^{m} W_{t-1}(i) \mathbb{1}_{h_t(x_i) \neq y_i}$ the weighted error of $h_t$
    Compute $\alpha_t = \frac{1}{2} \log \frac{1 - \epsilon_t}{\epsilon_t}$
    Update $W_t(i) = \frac{W_{t-1}(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$ for $i = 1, \dots, m$, where
     $Z_t = \sum_{i=1}^{m} W_{t-1}(i) \exp(-\alpha_t y_i h_t(x_i))$
end
return $H : x \in \mathcal{X} \mapsto \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(x)\right) \in \mathcal{Y}$
```

"Weak learners" are generally very shallow decision trees.

The Adaboost Algorithm

"Boosting is an approach to machine learning based on the idea of creating a highly accurate prediction rule by combining many relatively weak and inaccurate rules. The AdaBoost algorithm of Freund and Schapire [10] was the first practical boosting algorithm, and remains one of the most widely used and studied, with applications in numerous fields. Over the years, a great variety of attempts have been made to "explain" AdaBoost as a learning algorithm, that is, to understand why it works, how it works, and when it works (or fails). It is by understanding the nature of learning at its foundation — both generally and with regard to particular algorithms and phenomena — that the field is able to move forward."

Robert Schapire (2015)

The Mathematics of Adaboost: A Loss Function Approach

The goal is to find a function

$$f(x) = \frac{1}{2}\sum_m \alpha_m f_m(x)$$

that will minimize the **exponential loss** function $L(x, y) = e^{-yf(x)}$. Given the data (x_i, y_i), this means

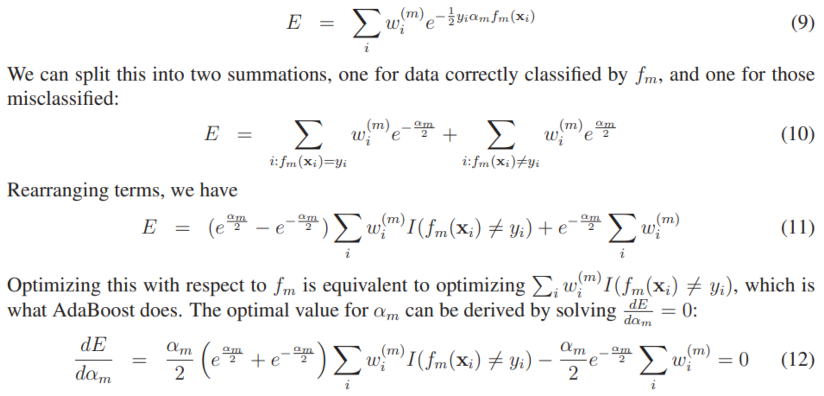
$$E = \sum_i e^{-\frac{1}{2} y_i \sum_{m=1}^{M} \alpha_m f_m(x_i)}$$

Now do algebra:

$$\begin{aligned} E &= \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i) - \frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \\ &= \sum_i e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(\mathbf{x}_i)} \, e^{-\frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \end{aligned}$$

The Mathematics of Adaboost: A Loss Function Approach

Let $w_i^{(m)} = e^{-\frac{1}{2} y_i \sum_{j=1}^{m-1} \alpha_j f_j(x)}$. Then



$$E = \sum_i w_i^{(m)} e^{-\frac{1}{2} y_i \alpha_m f_m(\mathbf{x}_i)} \tag{9}$$

We can split this into two summations, one for data correctly classified by f_m, and one for those misclassified:

$$E = \sum_{i: f_m(\mathbf{x}_i) = y_i} w_i^{(m)} e^{-\frac{\alpha_m}{2}} + \sum_{i: f_m(\mathbf{x}_i) \neq y_i} w_i^{(m)} e^{\frac{\alpha_m}{2}} \tag{10}$$

Rearranging terms, we have

$$E = (e^{\frac{\alpha_m}{2}} - e^{-\frac{\alpha_m}{2}}) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) + e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)} \tag{11}$$

Optimizing this with respect to f_m is equivalent to optimizing $\sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i)$, which is what AdaBoost does. The optimal value for α_m can be derived by solving $\frac{dE}{d\alpha_m} = 0$:

$$\frac{dE}{d\alpha_m} = \frac{\alpha_m}{2} \left(e^{\frac{\alpha_m}{2}} + e^{-\frac{\alpha_m}{2}} \right) \sum_i w_i^{(m)} I(f_m(\mathbf{x}_i) \neq y_i) - \frac{\alpha_m}{2} e^{-\frac{\alpha_m}{2}} \sum_i w_i^{(m)} = 0 \tag{12}$$

The Mathematics of Adaboost: A Loss Function Approach

Dividing both sides by $\frac{\alpha_m}{2\sum_i w_i^{(m)}}$, we have

$$\begin{aligned} 0 &= e^{\frac{\alpha_m}{2}}\epsilon_m + e^{-\frac{\alpha_m}{2}}\epsilon_m - e^{-\frac{\alpha_m}{2}} \\ e^{\frac{\alpha_m}{2}}\epsilon_m &= e^{-\frac{\alpha_m}{2}}(1-\epsilon_m) \\ \frac{\alpha_m}{2} + \ln\epsilon_m &= -\frac{\alpha_m}{2} + \ln(1-\epsilon_m) \\ \alpha_m &= \ln\frac{1-\epsilon_m}{\epsilon_m} \end{aligned}$$

Gradient Boosting (for regression)

Suppose you have a model $F(x)$ which isn't perfect ... and you want to improve it by adding on another model:

$$
\begin{aligned}
&F(x_1) + h(x_1) = y_1\\
&F(x_2) + h(x_2) = y_2\\
&\ldots\\
&F(x_n) + h(x_n) = y_n
\end{aligned}
$$

or

$$
\begin{aligned}
&h(x_1) = y_1 - F(x_1)\\
&h(x_2) = y_2 - F(x_2)\\
&\ldots\\
&h(x_n) = y_n - F(x_n)
\end{aligned}
$$

Idea: Fit a regression tree to this *residual* data ...

Gradient Boosting (for regression)

Idea: Apply gradient descent to the (RSS) loss function for this error ...

$$\frac{\partial J}{\partial F(x_i)} = \frac{\partial \sum_i L(y_i, F(x_i))}{\partial F(x_i)} = \frac{\partial L(y_i, F(x_i))}{\partial F(x_i)} = F(x_i) - y_i$$

This is just gradient descent:

$$\begin{aligned} F(x_i) &:= F(x_i) + h(x_i) \\ F(x_i) &:= F(x_i) + y_i - F(x_i) \\ F(x_i) &:= F(x_i) - 1\frac{\partial J}{\partial F(x_i)} \\ \theta_i &:= \theta_i - \rho\frac{\partial J}{\partial \theta_i} \end{aligned}$$