Machine Learning

Ensemble methods

Emanuele Rodolà
rodola@di.uniroma1.it

SAPIENZA
UNIVERSITÀ DI ROMA

2nd semester a.y. 2024/2025 · May 12, 2025

Y8ML0ZCL

Main idea: Combine many weak learners into a stronger ensemble model.

Objective: To improve performance and robustness.

Main idea: Combine many weak learners into a stronger ensemble model.

Objective: To improve performance and robustness.

Weak learners: Models that perform slightly better than random guessing.

- Simple and fast.
- High bias (underfitting), low variance (stable predictions).

Main idea: Combine many weak learners into a stronger ensemble model.

Objective: To improve performance and robustness.

Weak learners: Models that perform slightly better than random guessing.

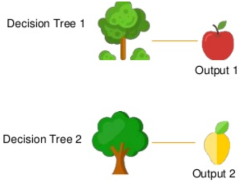
- Simple and fast.
- High bias (underfitting), low variance (stable predictions).

Reduce the high bias by aggregation!

Random forest: Intuition

Figures by Richard Kershner

Random forest: Intuition

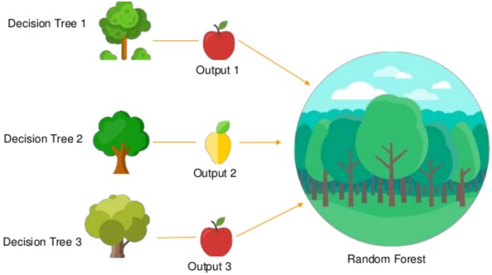


Construct multiple decision trees.

Each decision tree outputs a prediction for a given input.

Figures by Richard Kershner

Random forest: Intuition

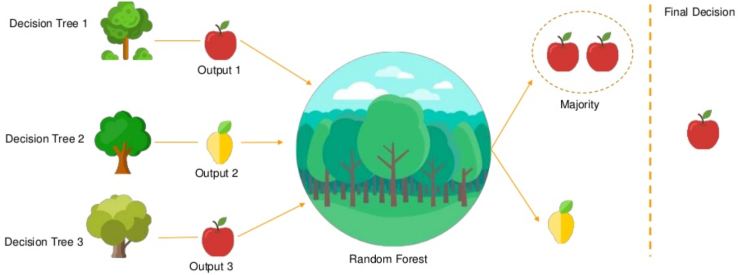


Construct multiple decision trees.

Each decision tree outputs a prediction for a given input.

Figures by Richard Kershner

Random forest: Intuition



Construct multiple decision trees.

Each decision tree outputs a prediction for a given input.

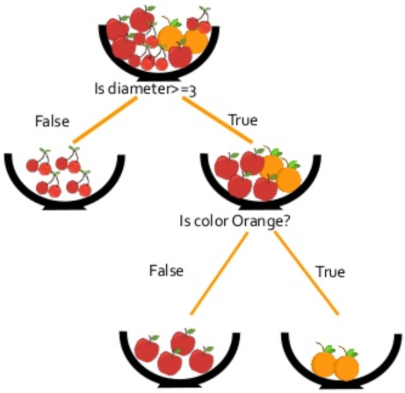
The predictions from each tree are combined into one final prediction.

In the example, the final decision is "apple with probability 66%".

Figures by Richard Kershner

Decision tree

A decision tree is a binary tree in which each branch represents a possible decision. A path through the tree is therefore a course of action.

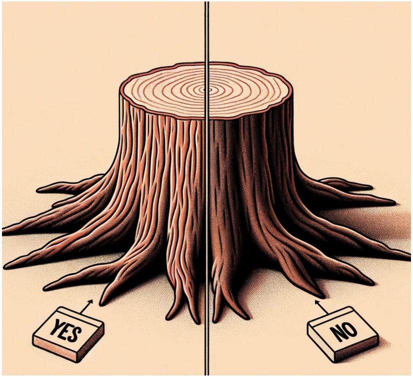


The data is iteratively partitioned into subsets from the root to the leaves.

Decision stump

An even weaker model is a decision stump.

A one-level decision tree, splitting data based on a single feature.



Information gain

Intermediate nodes are decision nodes.

Leaf nodes store the final decision (i.e. the classification result).

Information gain

Intermediate nodes are decision nodes.

Leaf nodes store the final decision (i.e. the classification result).

We use entropy to choose what is done at each node.

Entropy measures the uncertainty for a given data distribution.

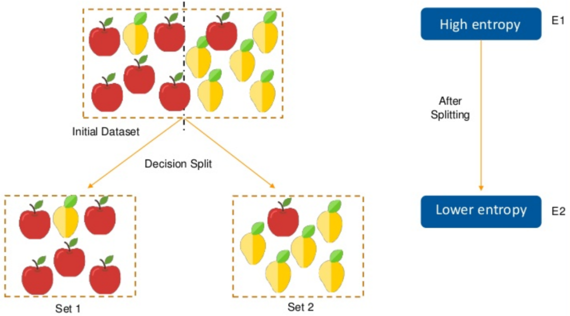
Information gain

Intermediate nodes are decision nodes.

Leaf nodes store the final decision (i.e. the classification result).

We use entropy to choose what is done at each node.

Entropy measures the uncertainty for a given data distribution.



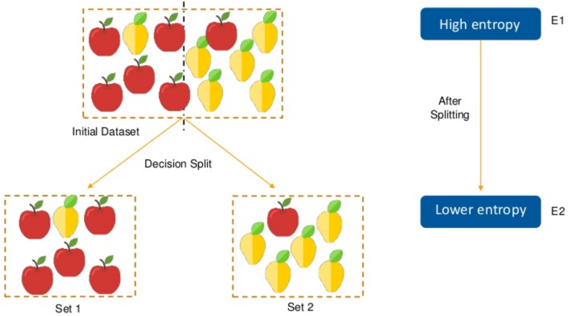
Information gain

Intermediate nodes are decision nodes.

Leaf nodes store the final decision (i.e. the classification result).

We use entropy to choose what is done at each node.

Entropy measures the uncertainty for a given data distribution.



The split should be as discriminative as possible: maximize the information gain $E2 - E1$.

Training

Problem: Classify the types of fruit based on their features.

We start from a set of data points that have high entropy.

Training

Problem: Classify the types of fruit based on their features.

We start from a set of data points that have high entropy.

Define the features (diameter, color, etc.) on which to base our decisions.

Training

Problem: Classify the types of fruit based on their features.

We start from a set of data points that have high entropy.

Define the features (diameter, color, etc.) on which to base our decisions.

At each node, choose the question that maximizes the information gain.

Training

Problem: Classify the types of fruit based on their features.

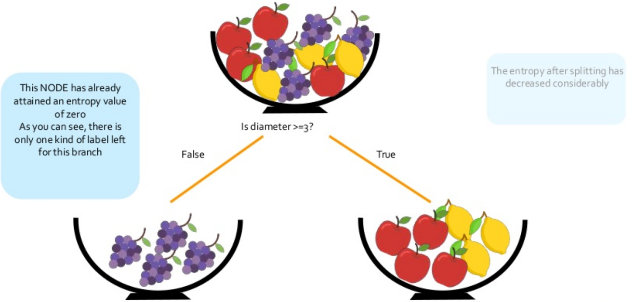
We start from a set of data points that have high entropy.

Define the features (diameter, color, etc.) on which to base our decisions.

At each node, choose the question that maximizes the information gain.

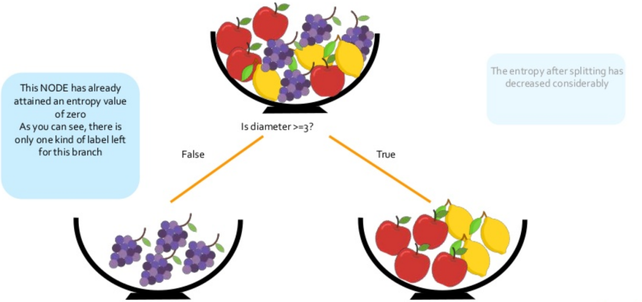
Keep splitting until the dataset is empty or accuracy is high enough.

Accuracy



A leaf is where no more splitting is required or possible (zero entropy).

Accuracy

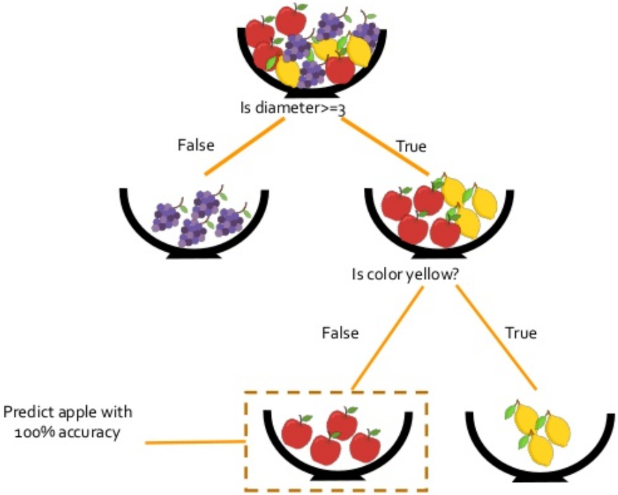


A leaf is where no more splitting is required or possible (zero entropy).

At each leaf, accuracy with respect to label ℓ is measured as:

$$\frac{\#\text{ data points with label } \ell}{\#\text{ data points}}$$

Accuracy



Is diameter>=3
False
True
Is color yellow?
False
True
Predict apple with
100% accuracy

Feature interaction

Each tree splits features to make predictions.

Thus, deeper nodes represent interactions between features.

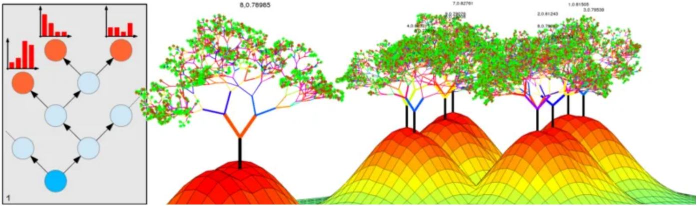


Figure: Sonali Dasgupta

Feature interaction

Each tree splits features to make predictions.

Thus, deeper nodes represent interactions between features.

- At depth 2, the split is done using only one feature.

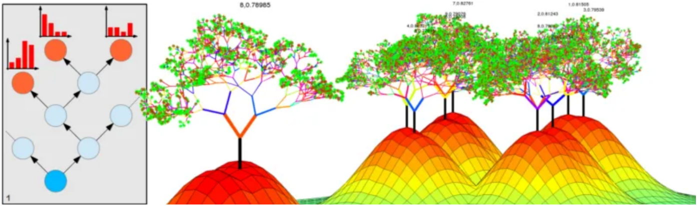


Figure: Sonali Dasgupta

Feature interaction

Each tree splits features to make predictions.

Thus, deeper nodes represent interactions between features.

- At depth 2, the split is done using only one feature.
- At depth 3, the splits are based on two features.

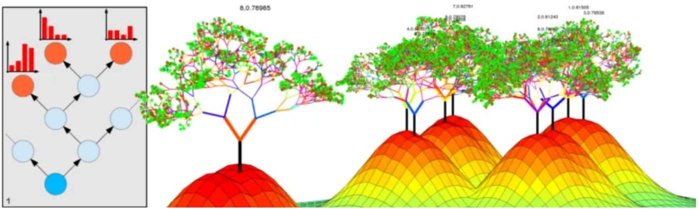


Figure: Sonali Dasgupta

Feature interaction

Each tree splits features to make predictions.

Thus, deeper nodes represent interactions between features.

- At depth 2, the split is done using only one feature.
- At depth 3, the splits are based on two features.
- At depth k, the splits consider interaction between $k-1$ features.

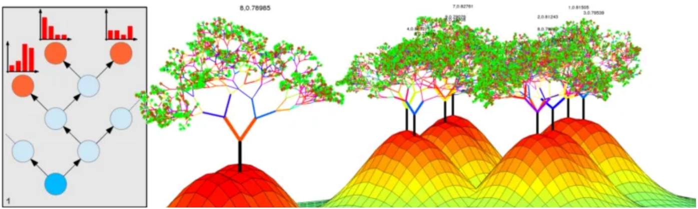


Figure: Sonali Dasgupta

Feature interaction

Each tree splits features to make predictions.

Thus, deeper nodes represent interactions between features.

- At depth 2, the split is done using only one feature.
- At depth 3, the splits are based on two features.
- At depth k, the splits consider interaction between $k - 1$ features.
- Deep trees capture complex interactions but risk overfitting.

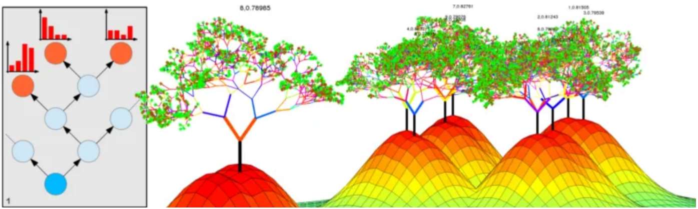


Figure: Sonali Dasgupta

Regression forests

For classification problems:

- Each tree outputs a probability distribution over all the classes.
- Internal splits maximize the information gain.
- Leaf nodes contain the majority class label.

Regression forests

For classification problems:

- Each tree outputs a probability distribution over all the classes.
- Internal splits maximize the information gain.
- Leaf nodes contain the majority class label.

For regression problems:

- Each tree outputs a continuous value.
- Internal splits minimize the MSE.
- Leaf nodes contain the average value.

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

- Create a potential split, resulting in two child nodes.

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

- Create a potential split, resulting in two child nodes.
- In each child node, compute the average temperature.

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

- Create a potential split, resulting in two child nodes.
- In each child node, compute the average temperature.
- For each mobile phone in the child node, compute the difference between its temperature and the average.

  Take the mean of these differences.

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

- Create a potential split, resulting in two child nodes.
- In each child node, compute the average temperature.
- For each mobile phone in the child node, compute the difference between its temperature and the average.

  Take the mean of these differences.
- Compute the weighted average $\frac{n_{\text{left}}}{n}\text{MSE}_{\text{left}} + \frac{n_{\text{right}}}{n}\text{MSE}_{\text{right}}$

Regression forests: Example

- **Training data:** mobile phones
- **Features:** screen brightness, battery level, number of apps
- **Target value** to predict: temperature

Since temperature is continuous, we need a regression forest.

- Create a potential split, resulting in two child nodes.
- In each child node, compute the average temperature.
- For each mobile phone in the child node, compute the difference between its temperature and the average.

  Take the mean of these differences.
- Compute the weighted average $\frac{n_{\text{left}}}{n}\text{MSE}_{\text{left}} + \frac{n_{\text{right}}}{n}\text{MSE}_{\text{right}}$
- Keep the split that results in the lowest weighted average MSE.

Pipeline

1. Split the training data into random partitions (bagging).

Breiman, “Random forests”, 2001

Pipeline

1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

Breiman, "Random forests", 2001

Pipeline

1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

   Each tree asks different questions, has different depth, and different accuracy at the leaves.

Breiman, "Random forests", 2001

Pipeline

1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

   Each tree asks different questions, has different depth, and different accuracy at the leaves.

   However, all the trees try to solve the same problem.

Breiman, "Random forests", 2001

Pipeline

1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

   Each tree asks different questions, has different depth, and different accuracy at the leaves.

   However, all the trees try to solve the same problem.
3. Given a new data point, route it through each tree.

Breiman, "Random forests", 2001

Pipeline

1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

   Each tree asks different questions, has different depth, and different accuracy at the leaves.

   However, all the trees try to solve the same problem.
3. Given a new data point, route it through each tree.

   Each tree gives its own prediction (e.g. a probability distribution).

Breiman, "Random forests", 2001

Pipeline

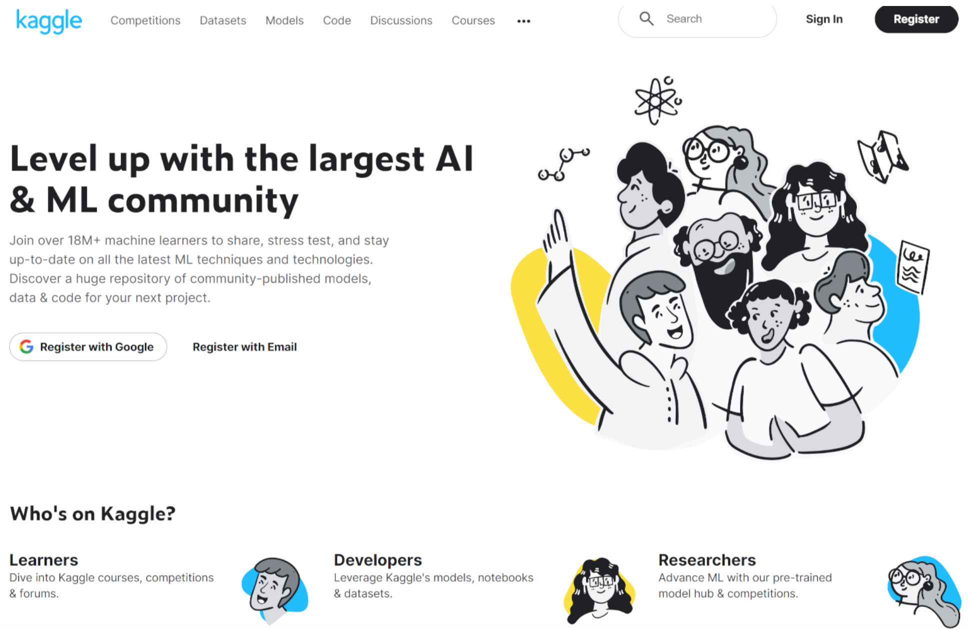
1. Split the training data into random partitions (bagging).
2. Train a tree for each partition.

   Each tree asks different questions, has different depth, and different accuracy at the leaves.

   However, all the trees try to solve the same problem.
3. Given a new data point, route it through each tree.

   Each tree gives its own prediction (e.g. a probability distribution).
4. Average the tree predictions to get the forest prediction.

Breiman, “Random forests”, 2001



https://www.kaggle.com/

Bagging vs. Boosting

- **Bagging:** Reduces variance by training on different subsets of data.

  Example: Random Forests.

Bagging vs. Boosting

- **Bagging:** Reduces variance by training on different subsets of data.

  Example: Random Forests.

- **Boosting:** Reduces bias by focusing on errors of previous models.

  Example: AdaBoost, XGBoost, LightGBM.

Main idea: Sequentially train weak learners, focusing each time on the incorrectly predicted data; then, combine these models.

Main idea: Sequentially train weak learners, focusing each time on the incorrectly predicted data; then, combine these models.

- AdaBoost: At each iteration, re-weight the training data to focus on the difficult samples.

Boosting

Main idea: Sequentially train weak learners, focusing each time on the incorrectly predicted data; then, combine these models.

- AdaBoost: At each iteration, re-weight the training data to focus on the difficult samples.
- Gradient Boosting: At each iteration, focus on the residual errors of the previous model.

AdaBoost

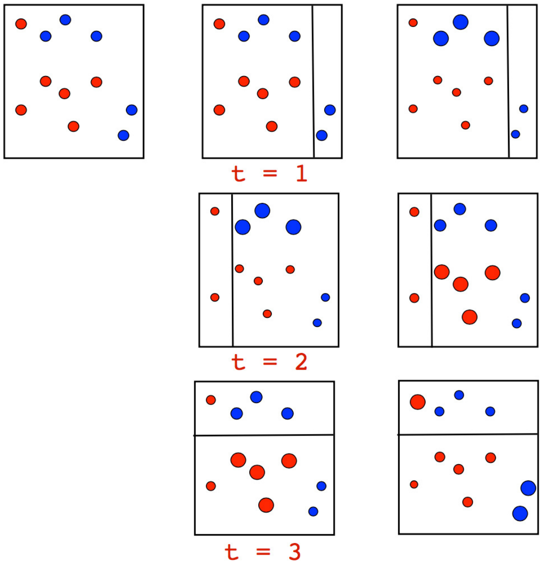
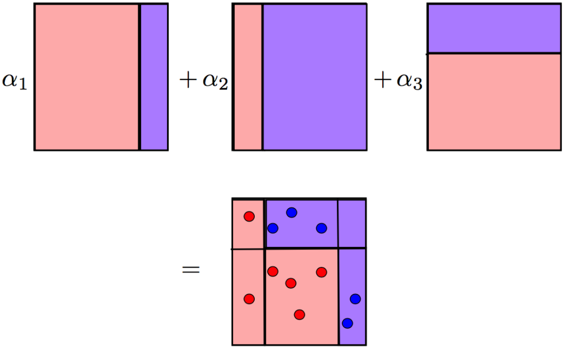


Figure: Yury Kashnitsky

AdaBoost



The α_i are proportional to the accuracy of each weak learner.

AdaBoost tends to overfit.

Figure: Yury Kashnitsky

AdaBoost

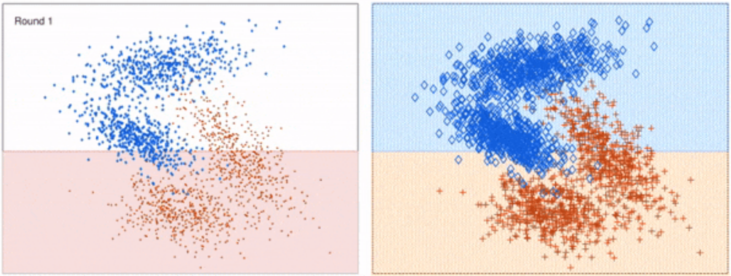


Figure: Kai O. Arras

Example: Toxic content

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

- Train a weak learner F_m.
  Example: for $m = 0$, define $F_m(x_i) = \frac{1}{n} \sum_i y_i$

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

- Train a weak learner F_m.
  Example: for $m = 0$, define $F_m(x_i) = \frac{1}{n} \sum_i y_i$
- Consider $F_{m+1}(x_i) = F_m(x_i) + \underbrace{h_{m+1}(x_i)}_{\text{residual}} = y_i$

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

- Train a weak learner F_m.
  Example: for $m = 0$, define $F_m(x_i) = \frac{1}{n} \sum_i y_i$
- Consider $F_{m+1}(x_i) = F_m(x_i) + \underbrace{h_{m+1}(x_i)}_{\text{residual}} = y_i$
- Train a new model h_{m+1} such that $h_{m+1}(x_i) \approx y_i - F_m(x_i)$

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

- Train a weak learner F_m.
  Example: for $m = 0$, define $F_m(x_i) = \frac{1}{n} \sum_i y_i$
- Consider $F_{m+1}(x_i) = F_m(x_i) + \underbrace{h_{m+1}(x_i)}_{\text{residual}} = y_i$
- Train a new model h_{m+1} such that $h_{m+1}(x_i) \approx y_i - F_m(x_i)$

Each F_{m+1} tries to correct the errors of its predecessor F_m.

Gradient boosting

Let's illustrate this with least-squares regression for $F : X \to \mathbb{R}$:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Now iterate for $m = 0, \ldots$:

- Train a weak learner F_m.
  Example: for $m = 0$, define $F_m(x_i) = \frac{1}{n} \sum_i y_i$
- Consider $F_{m+1}(x_i) = F_m(x_i) + \underbrace{h_{m+1}(x_i)}_{\text{residual}} = y_i$
- Train a new model h_{m+1} such that $h_{m+1}(x_i) \approx y_i - F_m(x_i)$

Each F_{m+1} tries to correct the errors of its predecessor F_m.

The final solution is the combination $F_0(x) + \sum_m h_m(x)$.

Gradient boosting

Why is it called "gradient" boosting?

Given the MSE loss:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Its (negative) derivative w.r.t. the predictions $F(x)$ is:

$$-\frac{\partial L(y, F(x))}{\partial F(x)} = 2 \sum_i (y_i - F(x_i))$$

Gradient boosting

Why is it called "gradient" boosting?

Given the MSE loss:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Its (negative) derivative w.r.t. the predictions $F(x)$ is:

$$-\frac{\partial L(y, F(x))}{\partial F(x)} = 2\sum_i (y_i - F(x_i))$$

Which is equal to:

$$2\sum_i h_{m+1}(x_i)$$

Gradient boosting

Why is it called "gradient" boosting?

Given the MSE loss:

$$L(y, F(x)) = \sum_i (F(x_i) - y_i)^2$$

Its (negative) derivative w.r.t. the predictions $F(x)$ is:

$$-\frac{\partial L(y, F(x))}{\partial F(x)} = 2 \sum_i (y_i - F(x_i))$$

Which is equal to:

$$2 \sum_i h_{m+1}(x_i)$$

For this reason, the derivatives $-\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}$ are also called pseudo-residuals r_i.

Gradient boosting

Since $h_{m+1} \propto -\frac{\partial L}{\partial F}$, this really looks like standard gradient descent!

$$F_{m+1}(x) = F_m(x) + \gamma h_{m+1}(x)$$

where γ is the learning rate.

Gradient boosting

Since $h_{m+1} \propto -\frac{\partial L}{\partial F}$, this really looks like standard gradient descent!

$$F_{m+1}(x) = F_m(x) + \gamma h_{m+1}(x)$$

where γ is the learning rate.

However:

- Gradient boosting is typically initialized with a constant F_0.

Gradient boosting

Since $h_{m+1} \propto -\frac{\partial L}{\partial F}$, this really looks like standard gradient descent!

$$F_{m+1}(x) = F_m(x) + \gamma h_{m+1}(x)$$

where γ is the learning rate.

However:

- Gradient boosting is typically initialized with a constant F_0.
- Instead of just computing $\frac{\partial L}{\partial F}$, you fit a weak model h_{m+1} to it.

Gradient boosting

Since $h_{m+1} \propto -\frac{\partial L}{\partial F}$, this really looks like standard gradient descent!

$$F_{m+1}(x) = F_m(x) + \gamma h_{m+1}(x)$$

where γ is the learning rate.

However:

- Gradient boosting is typically initialized with a constant F_0.
- Instead of just computing $\frac{\partial L}{\partial F}$, you fit a weak model h_{m+1} to it.
- Instead of fixing γ, compute it via line search:

$$\gamma_{m+1} = \arg\min_{\gamma} L(y, F_m(x) + \gamma h_{m+1}(x))$$

Gradient boosting

Since $h_{m+1} \propto -\frac{\partial L}{\partial F}$, this really looks like standard gradient descent!

$$F_{m+1}(x) = F_m(x) + \gamma h_{m+1}(x)$$

where γ is the learning rate.

However:

- Gradient boosting is typically initialized with a constant F_0.
- Instead of just computing $\frac{\partial L}{\partial F}$, you fit a weak model h_{m+1} to it.
- Instead of fixing γ, compute it via line search:

$$\gamma_{m+1} = \arg\min_{\gamma} L(y, F_m(x) + \gamma h_{m+1}(x))$$

Note: The weak model $h_{m+1}(x)$ is trained using the pseudo-residuals for the training set $\{(x_i, r_i)\}$, **not** using the labels $\{(x_i, y_i)\}$!

Gradient boosting in practice

- Designed to learn scalar-valued models F.

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.
- Remember: you compute the pseudo-residuals as the derivatives:

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

  and then fit weak learners to them.

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.
- Remember: you compute the pseudo-residuals as the derivatives:

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

  and then fit weak learners to them.
- Conceptually there are two losses:

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.
- Remember: you compute the pseudo-residuals as the derivatives:

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

  and then fit weak learners to them.
- Conceptually there are two losses:
  - The loss L defined on the original data $\{(x_i, y_i)\}$, that is minimized over all iterations.

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.
- Remember: you compute the pseudo-residuals as the derivatives:

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

  and then fit weak learners to them.
- Conceptually there are two losses:
  - The loss L defined on the original data $\{(x_i, y_i)\}$, that is minimized over all iterations.
  - An inner loss to fit the residuals $\{(x_i, r_i)\}$ by the weak learners. This loss may not be defined explicitly, e.g. when using regression forests.

Gradient boosting in practice

- Designed to learn scalar-valued models F.
- For multinomial classification, typically use the one-vs-rest paradigm.
- Remember: you compute the pseudo-residuals as the derivatives:

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

  and then fit weak learners to them.
- Conceptually there are two losses:
  - The loss L defined on the original data $\{(x_i, y_i)\}$, that is minimized over all iterations.
  - An inner loss to fit the residuals $\{(x_i, r_i)\}$ by the weak learners. This loss may not be defined explicitly, e.g. when using regression forests.

This ensures that each subsequent model in the boosting process focuses on correcting the errors of the previous models.

Gradient boosting vs. Deep learning

Gradient boosting:

- Works well with heterogeneous tabular data.
- Requires less training data.
- Easier to interpret.
- Faster to train.

Gradient boosting vs. Deep learning

Gradient boosting:

- Works well with heterogeneous tabular data.
- Requires less training data.
- Easier to interpret.
- Faster to train.

Deep learning:

- Works very well with very large datasets.
- Learns features by itself, e.g. as local spatial patterns.
- Allows transfer learning, e.g. via fine-tuning.
- Flexible for many tasks (e.g., generation, sequence prediction, etc.).

Suggested reading

"Understanding random forests: from theory to practice"

`https://arxiv.org/pdf/1407.7502`

"Ensembles: Gradient boosting, random forests, bagging, voting, stacking"

`https://scikit-learn.org/stable/modules/ensemble.html`