Machine Learning and Intelligent Systems

Ensemble Methods

Maria A. Zuluaga

Jan 12, 2023

EURECOM - Data Science Department

Table of contents

Recap

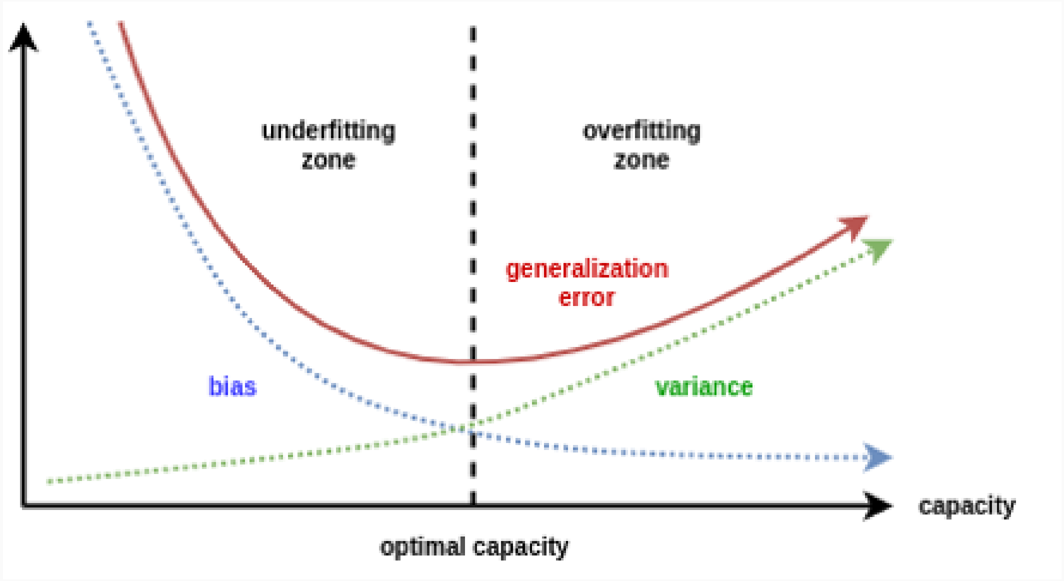
Bias-Variance Decomposition

$$\mathbb{E}_{\mathbf{x},y,\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x})-y)^2] = \underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x})-\bar{h}(\mathbf{x}))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}_{\mathbf{x},y}[(\bar{y}(\mathbf{x})-y)^2]}_{\text{Noise}} + \underbrace{\mathbb{E}_{\mathbf{x}}[(\bar{h}(\mathbf{x})-\bar{y}(\mathbf{x}))^2]}_{\text{Bias}^2}$$

Noise: The error associated to the data. It measures ambiguity due to your data distribution and feature representation. **You can never beat this**.

Variance: Error caused from sensitivity to fluctuations in the training set. How much does the model change if it is trained in a different dataset. High variance can cause an algorithm to model noise from the training data rather than the intended targets (overfitting)

Bias: The inherent error that you obtain from the model even with infinite training data. This is due to the classifier being biased to a particular solution (e.g. linear classifier)



Source: `https://djsaunde.wordpress.com/2017/07/17/the-bias-variance-tradeoff/`

Bias-Variance Trade-off

- The **bias-variance trade-off** suggests that reducing one of the two, bias or variance, comes at the cost of an increase in the other term
- **Question:** Is it possible to reduce one of them without sacrificing the other one?
- **Short answer:** Yes
- **How?** Instead of learning a single model, we will learn multiple models and combine them

- In this lecture, we will see two different strategies to combine such models
  - **Bagging:** Models are combined **in parallel**
  - **Boosting:** Models are combined **sequentially**

Bagging

Reducing Variance

Goal: Reduce the variance without affecting the bias

$$\mathbb{E}_{\mathbf{x},y,\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - y)^2] = \underbrace{\mathbb{E}_{\mathbf{x},\mathcal{D}}[(h_{\mathcal{D}}(\mathbf{x}) - \bar{h}(\mathbf{x}))^2]}_{\text{Variance}} + \underbrace{\mathbb{E}_{\mathbf{x},y}[(\bar{y}(\mathbf{x}) - y)^2]}_{\text{Noise}} + \underbrace{\mathbb{E}_{\mathbf{x}}[(\bar{h}(\mathbf{x}) - \bar{y}(\mathbf{x}))^2]}_{\text{Bias}^2}$$

Reducing variance, in this context, accounts to $h_{\mathcal{D}}(\mathbf{x}) \to \bar{h}(\mathbf{x})$

We will achieve this by averaging multiple models

Weak Law of Large Numbers

For a set of i.i.d. random variables $\mathbf{x}_i$ with mean $\bar{\mathbf{x}}$

$$\frac{1}{M}\sum_{i=1}^{M}\mathbf{x}_i \to \bar{\mathbf{x}} \qquad \text{as } M \to \infty$$

Idea: Apply the weak law of large numbers to classifiers:

1. M datasets available $\mathcal{D}_1, \dots, \mathcal{D}_M$ drawn from $\mathcal{P}^N$
2. Train a classifier on each dataset and then average.

Ensemble of classifiers:

$$H = \frac{1}{M}\sum_{i=1}^{M} h_{\mathcal{D}_i}(\mathbf{x}) \to \bar{h}(\mathbf{x}) \qquad \text{as } M \to \infty$$

Problem: There is only one training dataset available

Solution: Bagging

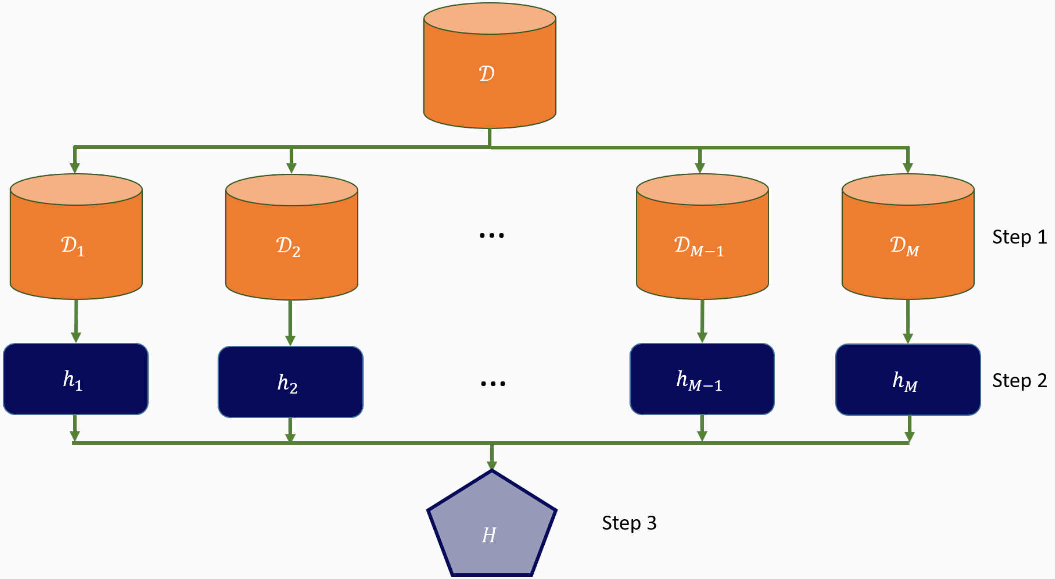
- Proposed by Leo Breiman in 1994
- The term **bagging** is an abbreviation to **bootstrap aggregating**
- The core idea behind bagging is to take repeated **bootstrap samples** from the training set $\mathcal{D}$.
- **Bootstrap sampling:** Given a set $\mathcal{D}$ containing N training samples, create $\mathcal{D}$' by drawing N samples at random with replacement from $\mathcal{D}$.

Solution: Bagging

- Proposed by Leo Breiman in 1994
- The term **bagging** is an abbreviation to **bootstrap aggregating**
- The core idea behind bagging is to take repeated **bootstrap samples** from the training set $\mathcal{D}$.
- **Bootstrap sampling:** Given a set $\mathcal{D}$ containing N training samples, create $\mathcal{D}$' by drawing N samples at random with replacement from $\mathcal{D}$.

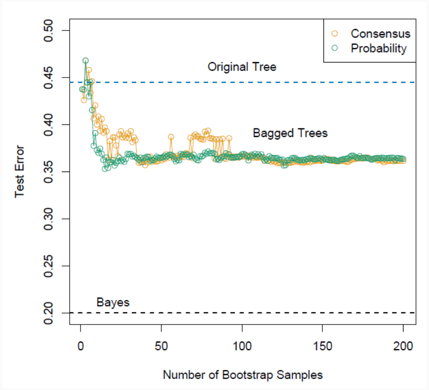
Algorithm

1. Create M bootstrap samples $\mathcal{D}_1, \ldots, \mathcal{D}_M$
2. Train a classifier on each $\mathcal{D}_i$
3. Classify new instance by majority vote or average


$\mathcal{D}$
$\mathcal{D}_1$
$\mathcal{D}_2$
...
$\mathcal{D}_{M-1}$
$\mathcal{D}_M$
Step 1
h_1
h_2
...
h_{M-1}
h_M
Step 2
H
Step 3

Analysis

- In bagging, $h_{\mathcal{D}}(\mathbf{x}) \nrightarrow \bar{h}(\mathbf{x})$
- The weak law of large numbers cannot be applied because the i.i.d. condition does not hold
- Despite this, they are efficient at reducing the variance



Out-of-Bag Error

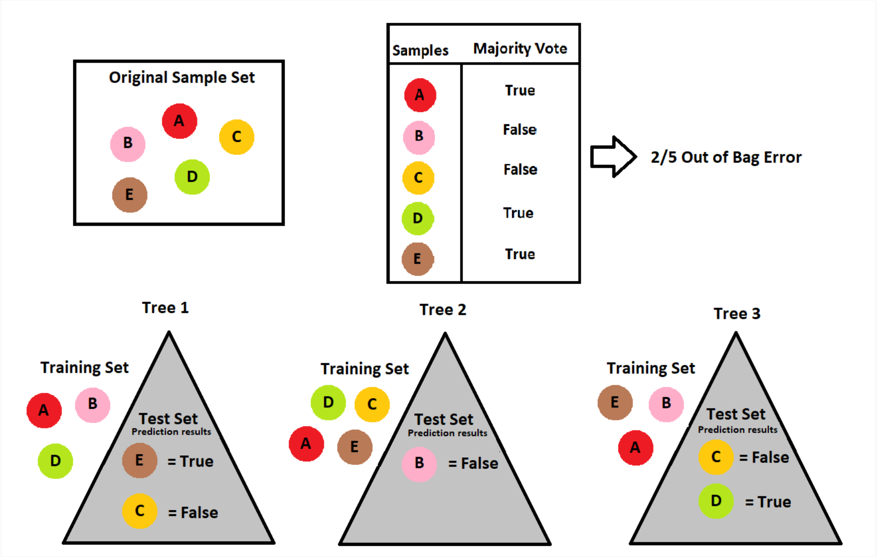
- Bagging provides and unbiased estimate of the test error
- **Idea:** Exploit the sample points that are not selected for a given classifier. These points are denoted the out-of-bag (OOB) set (or instance)
- **Advantage:** No need to reduce the training set to get an idea of the generalization error

Out-of-Bag Error

- Bagging provides and unbiased estimate of the test error
- **Idea:** Exploit the sample points that are not selected for a given classifier. These points are denoted the out-of-bag (OOB) set (or instance)
- **Advantage:** No need to reduce the training set to get an idea of the generalization error

Algorithm

1. Find all models that are not trained by the OOB instance.
2. Take the majority vote of these models' result for the OOB instance, compared to the true value of the OOB instance.
3. Estimate the OOB error for all instances in the OOB dataset.



Source: Wikipedia

Bagging: Summary

Advantages:

- Easy to implement
- Reduces variance keeping bias unaltered
- As prediction is the average of many classifiers, we can obtain a score and a variance that can be interpreted as uncertainty
- Out of bag error

Bagging: Summary

Advantages:

- Easy to implement
- Reduces variance keeping bias unaltered
- As prediction is the average of many classifiers, we can obtain a score and a variance that can be interpreted as uncertainty
- Out of bag error

Disadvantages:

- Computationally more expensive
- Correlated training sets

Wikipedia Facts: Bootstrapping

- Instatistics,bootstrapping is any test or metric that relies onrandom sampling with replacement.
- As a metaphor, means to better oneself by one's own unaided efforts. In use in 1922
- This metaphor spawned additional metaphors for a series of self-sustaining processes that proceed without external help.

"Pull yourself out with your own bootstraps"

Source: Wikipedia

Random Forests

Limitations of bagging:

- Often the decision trees look very similar.
- If one or more features are very informative, they will be selected by almost every tree in the bag, reducing the diversity (and potentially increasing the bias).

Solution: Random forests

- Ensemble method specifically designed for decision tree classifiers
- One of most (if not the most) famous bagging algorithm also proposed by Leo Breiman.
- Among the easiest to use ML algorithms
- **Idea:** Reduce correlation between trees in the bag without increasing variance too much

Randomness of Forests

The Randon Forests algorithm introduces two sources of randomness:

1. **Bagging:** Each tree is grown using a bootstrap sample of the training data
2. **Random vector method:** At each node, best split is chosen from a random sample of $k < D$ attributes

The random vector method alleviates the correlation among bootstrap samples

Algorithm

1. For $b = 1$ to B:
    1.1 Sample M datasets $\mathcal{D}_1, \dots \mathcal{D}_M$ from $\mathcal{D}$ with replacement (bootstrap samples)
    1.2 Grow a random forest tree T_b to the bootstrap data (i.e. $\mathcal{D}_1, \dots \mathcal{D}_M$) by repeating the following steps:
        1.2.1 Select k variables at random from the D features
        1.2.2 Pick the best variable/split-point among k
        1.2.3 Split the node into two child nodes
        1.2.4 Repeat until reaching a leaf node
2. Output the ensemble of trees $\{T_b\}_1^B$
3. Prediction: Average for regression, majority voting for classification

Important: Once k variables are selected at a given node, it will not be possible to select splits using the remaining $D - k$ features for that branch.

Tips

- The RF only has two hyper-parameters M and k. It is quite insensitive to these. A good choice for these is:
  - For k: $k = \sqrt{D}$ for classification; $k \approx \frac{D}{3}$ for regression.
  - For M: as large as possible
  - **Important:** Smaller k implies more randomness, less tree correlation and more bias

Tips

- The RF only has two hyper-parameters M and k. It is quite insensitive to these. A good choice for these is:
  - For k: $k = \sqrt{D}$ for classification; $k \approx \frac{D}{3}$ for regression.
  - For M: as large as possible
  - **Important:** Smaller k implies more randomness, less tree correlation and more bias

- Decision trees do not require a lot of preprocessing.
  - The features can be of different scale, magnitude, or slope.
  - Advantageous in scenarios with heterogeneous data, which is recorded in completely different units

Limitations of Bagging and Random Forests

- **Bagging:** Significant correlation between trees that are learnt on different training datasets
- **Random Forest**s try to resolve this by doing random feature sampling, but some correlation still remains
- All B trees are given the same weight when taking the average

Solution: Boosting methods try to force classifiers to learn on different parts of the feature space, and take their weighted average

Boosting

Boosting

- **Context:** Hypothesis class $\mathcal{H}$, whose set of classifiers has a large bias and high training error
- **Question:** Can weak learners $\in \mathcal{H}$ be combined to generate a strong learner?
  Michael Kearns (Prof. at UPenn) in his ML course project [1988]
- **Answer:** Yes
  Robert Schapire [1990]

Boosting

- **Context:** Hypothesis class $\mathcal{H}$, whose set of classifiers has a large bias and high training error
- **Question:** Can weak learners $\in \mathcal{H}$ be combined to generate a strong learner?
  Michael Kearns (Prof. at UPenn) in his ML course project [1988]
- **Answer:** Yes
  Robert Schapire [1990]

Definitions

Weak learner One whose error rate is only slightly better than random guessing (< 0.5).

Strong learner One who is arbitrarily well-correlated with the true classification

High-level idea

- Sequentially construct weak classifiers and combine them to obtain a complex decision boundary (i.e. a strong classifier)

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{1}$$

- At each iteration t, the classifier $\alpha_t h_t(\mathbf{x})$ is added to the ensemble

High-level idea

- Sequentially construct weak classifiers and combine them to obtain a complex decision boundary (i.e. a strong classifier)

$$H(\mathbf{x}) = \sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}) \tag{1}$$

- At each iteration t, the classifier $\alpha_t h_t(\mathbf{x})$ is added to the ensemble
- At test time, all classifiers are evaluated and return the weighted sum

$$H(\mathbf{x}^*) = \text{sign}\left(\sum_{t=1}^{T} \alpha_t h_t(\mathbf{x}^*)\right)$$

- Analogous to gradient descent: Instead of updating model parameters at each iteration, functions are added to the ensemble

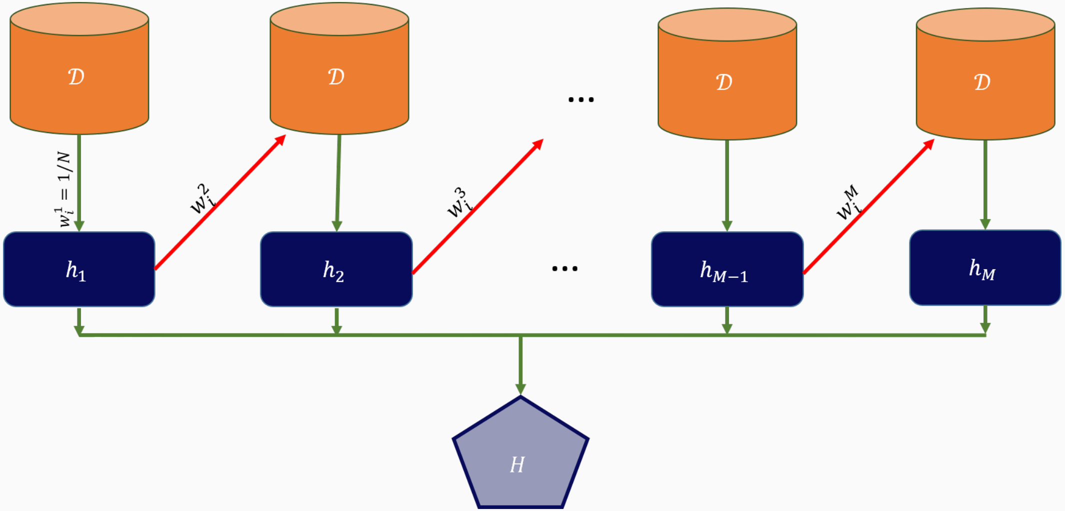
AdaBoost

- Adaboost (Freund and Schapire, 1996) is one of the most used boosting algorithms, which is the short for Adaptive Boosting
- Each base classifier is trained using a weighted form of the dataset
- The weighting coefficient associated to each point depends on the performance of the previous classifiers
- Misclassified points get more weight
- Predictions are done through a combined majority weighted scheme (or averaging)

AdaBoost

- Adaboost (Freund and Schapire, 1996) is one of the most used boosting algorithms, which is the short for Adaptive Boosting
- Each base classifier is trained using a weighted form of the dataset
- The weighting coefficient associated to each point depends on the performance of the previous classifiers
- Misclassified points get more weight
- Predictions are done through a combined majority weighted scheme (or averaging)
- We will look into the version proposed by Freidman et al (2000), which minimizes the exponential loss:

$$\mathcal{L}(H) = \sum_{i=1}^{N} \exp(-y_i H(\mathbf{x}_i))$$



$\mathcal{D}$
$\mathcal{D}$
…
$\mathcal{D}$
$\mathcal{D}$
$w_i^1 = 1/N$
w_i^2
w_i^3
w_i^M
h_1
h_2
…
h_{M-1}
h_M
H

AdaBoost Algorithm Setup

Setup: The training set $\mathcal{D}$ is composed of N inputs $\mathbf{X} \in \mathbb{R}^D$ with corresponding binary labels $y \in \{-1, +1\}$.

Initialization: Each data point $\mathbf{x}_i$ is given an associated weighting parameter $w_i^1 = \frac{1}{N}$

Training Algorithm: We will suppose there is a procedure in place to train a base classifier using the weighted $\mathcal{D}$ to obtain a a weak classifier $h_m(\mathbf{x})$

Weak Classifier: The error rate of a weak classifier $h_m(\mathbf{x})$ is calculated empirically over the training data:

$$\epsilon(h_m(\mathbf{x})) = \frac{1}{N} \sum_{i=1}^{N} \delta(h_m(\mathbf{x}_i) \neq y_i) < \frac{1}{2}$$

AdaBoost Algorithm (Friedman et al, 2000)

1. Initialize weights $w_i^1 = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    2.1 Train a weak classifier $h_m(\mathbf{x})$ that minimizes the weighted sum error for misclassified points:

$$\epsilon_m = \sum_{i=1}^{N} w_i^m \delta(h_m(\mathbf{x}_i) \neq y_i)$$

AdaBoost Algorithm (Friedman et al, 2000)

1. Initialize weights $w_i^1 = \frac{1}{N}$
2. For $m = 1, \dots, M$:
    2.1 Train a weak classifier $h_m(\mathbf{x})$ that minimizes the weighted sum error for misclassified points:

    $$\epsilon_m = \sum_{i=1}^{N} w_i^m \delta(h_m(\mathbf{x}_i) \neq y_i)$$

    2.2 If $\epsilon_m < \frac{1}{2}$:

        2.2.1 Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$

AdaBoost Algorithm (Friedman et al, 2000)

1. Initialize weights $w_i^1 = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    2.1 Train a weak classifier $h_m(\mathbf{x})$ that minimizes the weighted sum error for misclassified points:

$$\epsilon_m = \sum_{i=1}^{N} w_i^m \delta(h_m(\mathbf{x}_i) \neq y_i)$$

    2.2 If $\epsilon_m < \frac{1}{2}$:

        2.2.1 Compute $\alpha_m = \frac{1}{2} \log \left(\frac{1 - \epsilon_m}{\epsilon_m} \right)$

        2.2.2 Add to the ensemble: $H_m = H_{m-1} + \alpha_m h_m(\mathbf{x})$

AdaBoost Algorithm (Friedman et al, 2000)

```
1. Initialize weights $w_i^1 = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    2.1 Train a weak classifier $h_m(\mathbf{x})$ that minimizes the weighted sum error for misclassified points:
        $$\epsilon_m = \sum_{i=1}^{N} w_i^m \delta(h_m(\mathbf{x}_i) \neq y_i)$$
    2.2 If $\epsilon_m < \frac{1}{2}$:
        2.2.1 Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$
        2.2.2 Add to the ensemble: $H_m = H_{m-1} + \alpha_m h_m(\mathbf{x})$
        2.2.3 Update the data weights coefficients:
              $$w_i^{(m+1)} = w_i^m \exp[-\alpha_m y_i h_m(\mathbf{x}_i)]$$
              and normalize so that $\sum_i w_i^{(m+1)} = 1$
```

AdaBoost Algorithm (Friedman et al, 2000)

1. Initialize weights $w_i^1 = \frac{1}{N}$
2. For $m = 1, \ldots, M$:
    2.1 Train a weak classifier $h_m(\mathbf{x})$ that minimizes the weighted sum error for misclassified points:

    $$\epsilon_m = \sum_{i=1}^{N} w_i^m \delta(h_m(\mathbf{x}_i) \neq y_i)$$

    2.2 If $\epsilon_m < \frac{1}{2}$:

        2.2.1 Compute $\alpha_m = \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right)$

        2.2.2 Add to the ensemble: $H_m = H_{m-1} + \alpha_m h_m(\mathbf{x})$

        2.2.3 Update the data weights coefficients:

        $$w_i^{(m+1)} = w_i^m \exp[-\alpha_m y_i h_m(\mathbf{x}_i)]$$

        and normalize so that $\sum_i w_i^{(m+1)} = 1$
3. Make predictions using the final model:

$$H(\mathbf{x}^*) = \text{sign}\left(\sum_{m=1}^{M} \alpha_m h_m(\mathbf{x}^*)\right)$$

Analysis

Weak classifiers:

- The selected weight for each new weak classifier is always positive

$$\epsilon_m < \frac{1}{2} \Rightarrow \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right) > 0$$

- The smaller the classification error, the bigger its associated weight α and **the weak classifier will have more impact in the final strong classifier**.

Analysis

Weak classifiers:

- The selected weight for each new weak classifier is always positive

$$\epsilon_m < \frac{1}{2} \Rightarrow \frac{1}{2} \log\left(\frac{1-\epsilon_m}{\epsilon_m}\right) > 0$$

- The smaller the classification error, the bigger its associated weight α and **the weak classifier will have more impact in the final strong classifier**.

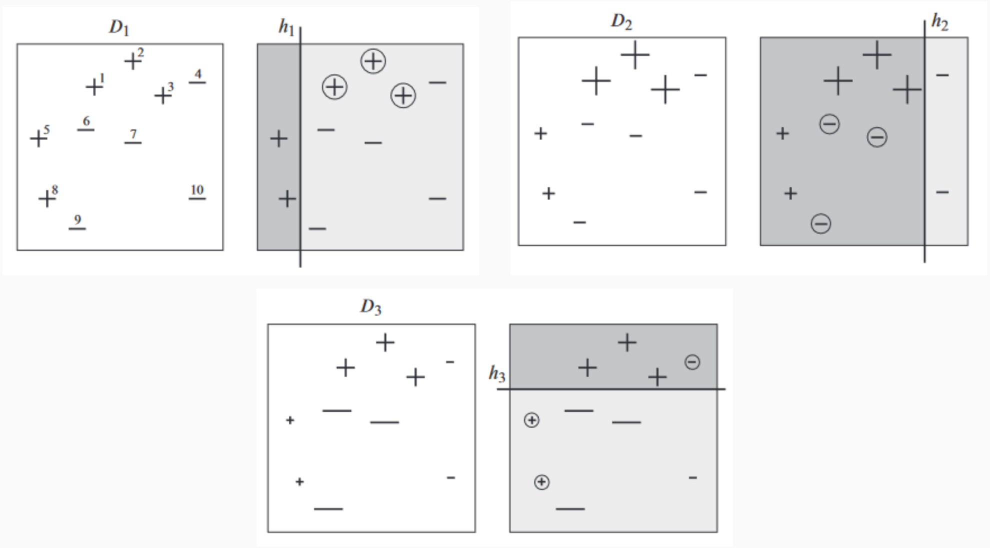
Weighted training points:

- The weights of the data points are multiplied by $\exp[-\alpha_m y_i h_m(\mathbf{x}_i)]$.

$$\exp[-\alpha_m y_i h_m(\mathbf{x}_i)] = \begin{cases} \exp[-\alpha_m], & \text{if} \quad h_m(\mathbf{x}_i)] = y_i \\ \exp[\alpha_m], & \text{if} \quad h_m(\mathbf{x}_i)] \neq y_i \end{cases}$$

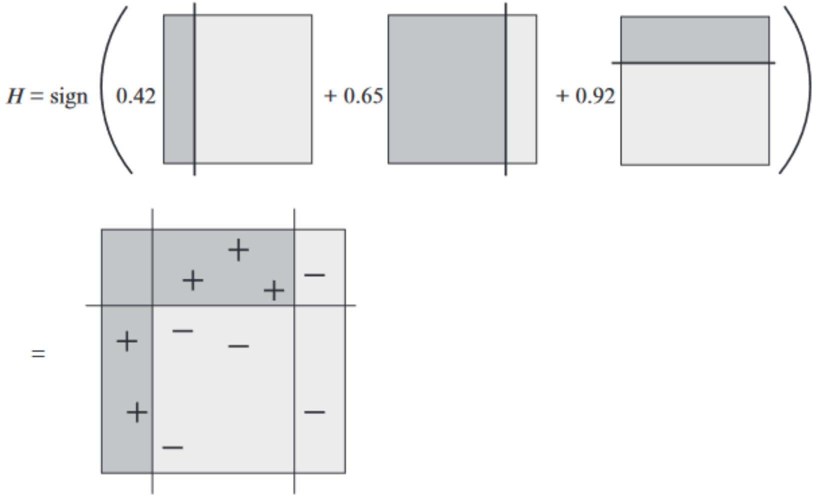
- The weights of correctly classified points are reduced and the weights of misclassified points increase. **Misclassified points will receive more attention in the next iteration**.

Example



Source: Fig 1.1. from Schapire & Freund 2012

Example



Source: Fig 1.2. from Schapire & Freund 2012

Gradient Boosting Trees

Gradient Boosting = Gradient Descent + Boosting

Gradient Boosting Trees

$$\text{Gradient Boosting} = \text{Gradient Descent} + \text{Boosting}$$

Key idea:

- Fit an additive model (ensemble) $\sum_m \alpha_m h_m$ in an iterative manner
- At each iteration, introduce a weak learner to compensate the shortcomings of existing weak learners.

Gradient Boosting Trees

$$\text{Gradient Boosting} = \text{Gradient Descent} + \text{Boosting}$$

Key idea:

- Fit an additive model (ensemble) $\sum_m \alpha_m h_m$ in an iterative manner
- At each iteration, introduce a weak learner to compensate the shortcomings of existing weak learners.
- **Recall:** In Adaboost, *shortcomings* are identified by high-weight data points.

Gradient Boosting Trees

$$\text{Gradient Boosting} = \text{Gradient Descent} + \text{Boosting}$$

Key idea:

- Fit an additive model (ensemble) $\sum_m \alpha_m h_m$ in an iterative manner
- At each iteration, introduce a weak learner to compensate the shortcomings of existing weak learners.
- **Recall:** In Adaboost, *shortcomings* are identified by high-weight data points.
- **Gradient Boosting:** *Shortcomings* are identified by gradients.

Gradient Boosting Trees

$$\text{Gradient Boosting} = \text{Gradient Descent} + \text{Boosting}$$

Key idea:

- Fit an additive model (ensemble) $\sum_m \alpha_m h_m$ in an iterative manner
- At each iteration, introduce a weak learner to compensate the shortcomings of existing weak learners.
- **Recall:** In Adaboost, *shortcomings* are identified by high-weight data points.
- **Gradient Boosting:** *Shortcomings* are identified by gradients.

Both high-weight data points and gradients provide information on how to improve the global model (ensemble)

Intuition

Game:

- You are given a set of points $\{(\mathbf{x}_i, y_i)\}_{i=1}^N$. Your task is to fit a model $H(\mathbf{x})$ to minimize the square loss
- To get started, they give you a model H.
- You check the provided model. It is good, but not perfect:
  $H(\mathbf{x}_1) = 0.7 \Rightarrow y_1 = 0.65$
  $H(\mathbf{x}_2) = 4.1 \Rightarrow y_2 = 4.2 \ldots$
- **Your goal:** To improve H

Intuition

Game:

- You are given a set of points $\{(\mathbf{x}_i, y_i)\}_{i=1}^{N}$. Your task is to fit a model $H(\mathbf{x})$ to minimize the square loss
- To get started, they give you a model H.
- You check the provided model. It is good, but not perfect:
  $H(\mathbf{x}_1) = 0.7 \Rightarrow y_1 = 0.65$
  $H(\mathbf{x}_2) = 4.1 \Rightarrow y_2 = 4.2 \ldots$
- **Your goal:** To improve H

Rules of the Game:

1. You cannot change anything in H (i.e. re-train)
2. You can add an additional model to H, so that the new prediction would be $H(\mathbf{x}) + h(\mathbf{x})$

Intuition - A simple solution to the game

You wish to improve your model such that:

$$\begin{aligned} H(\mathbf{x}_1) + h(\mathbf{x}_1) &= y_1 \\ H(\mathbf{x}_2) + h(\mathbf{x}_2) &= y_2 \\ &\dots \\ H(\mathbf{x}_N) + h(\mathbf{x}_N) &= y_N \end{aligned}$$

or equivalently:

$$\begin{aligned} h(\mathbf{x}_1) &= y_1 - H(\mathbf{x}_1) \\ h(\mathbf{x}_2) &= y_2 - H(\mathbf{x}_2) \\ &\dots \\ h(\mathbf{x}_N) &= y_N - H(\mathbf{x}_N) \end{aligned}$$

Is it possible to train a regression tree that accomplishes the goal?

Intuition - Residuals

Idea: To train the regression tree h to fit the **residuals**

Residuals: the parts were the existing model $H(\cdot)$ cannot do well, i.e. $y_i - H(\mathbf{x}_i)$

Intuition - Residuals

Idea: To train the regression tree h to fit the **residuals**

Residuals: the parts were the existing model $H(\cdot)$ cannot do well, i.e. $y_i - H(\mathbf{x}_i)$

This means that the new training set looks like:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i - H(\mathbf{x}_i))\}_{i=1}^{N}$$

Intuition - Residuals

Idea: To train the regression tree h to fit the **residuals**

Residuals: the parts were the existing model $H(\cdot)$ cannot do well, i.e. $y_i - H(\mathbf{x}_i)$

This means that the new training set looks like:

$$\mathcal{D} = \{(\mathbf{x}_i, y_i - H(\mathbf{x}_i))\}_{i=1}^{N}$$

The role of the learned h is to compensate the shortcoming of existing model $H(\cdot)$.

If the new model $H(\mathbf{x}) + h(\mathbf{x})$ is still not satisfactory, more trees can be added in an iterative fashion

How this relates to gradient descent?

Our final aim is to minimize the residuals. For such purpose, we can pick a convenient loss such as the sum of squared errors:

$$\mathcal{L}(H) = \frac{1}{2}\sum_{i=1}^{N}(y_i - H(\mathbf{x}))^2$$

How this relates to gradient descent?

Our final aim is to minimize the residuals. For such purpose, we can pick a convenient loss such as the sum of squared errors:

$$\mathcal{L}(H) = \frac{1}{2}\sum_{i=1}^{N}(y_i - H(\mathbf{x}))^2$$

Notice that $H(\mathbf{x}_1), H(\mathbf{x}_2), \ldots, H(\mathbf{x}_N)$ are just numbers, so we can treat $H(\mathbf{x}_i)$ as parameters and take derivatives.

How this relates to gradient descent?

Our final aim is to minimize the residuals. For such purpose, we can pick a convenient loss such as the sum of squared errors:

$$\mathcal{L}(H) = \frac{1}{2}\sum_{i=1}^{N}(y_i - H(\mathbf{x}))^2$$

Notice that $H(\mathbf{x}_1), H(\mathbf{x}_2), \ldots, H(\mathbf{x}_N)$ are just numbers, so we can treat $H(\mathbf{x}_i)$ as parameters and take derivatives.

In such scenario, it is easy to proof that:

$$-\frac{\partial \mathcal{L}}{\partial H} = y_i - H(\mathbf{x})$$

How this relates to gradient descent?

Our final aim is to minimize the residuals. For such purpose, we can pick a convenient loss such as the sum of squared errors:

$$\mathcal{L}(H) = \frac{1}{2}\sum_{i=1}^{N}(y_i - H(\mathbf{x}))^2$$

Notice that $H(\mathbf{x}_1), H(\mathbf{x}_2), \ldots, H(\mathbf{x}_N)$ are just numbers, so we can treat $H(\mathbf{x}_i)$ as parameters and take derivatives.

In such scenario, it is easy to proof that:

$$-\frac{\partial \mathcal{L}}{\partial H} = y_i - H(\mathbf{x})$$

which are nothing else but the residuals!

We can interpret the residuals as negative gradients

How this relates to gradient descent?

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + h(\mathbf{x}_i)$$

How this relates to gradient descent?

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + h(\mathbf{x}_i)$$
$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + y_i - H(\mathbf{x}_i)$$

How this relates to gradient descent?

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + h(\mathbf{x}_i)$$

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + y_i - H(\mathbf{x}_i)$$

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} - 1 \cdot \frac{\partial \mathcal{L}}{\partial H}$$

How this relates to gradient descent?

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + h(\mathbf{x}_i)$$

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + y_i - H(\mathbf{x}_i)$$

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} - 1 \cdot \frac{\partial \mathcal{L}}{\partial H}$$

From the gradient descent lecture, this looks just like:

$$\boldsymbol{\theta}^{(\tau+1)} \longleftarrow \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$

How this relates to gradient descent?

$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + h(\mathbf{x}_i)$$
$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} + y_i - H(\mathbf{x}_i)$$
$$H(\mathbf{x}_i)^{(m+1)} \longleftarrow H(\mathbf{x}_i)^{(m)} - 1 \cdot \frac{\partial \mathcal{L}}{\partial H}$$

From the gradient descent lecture, this looks just like:

$$\boldsymbol{\theta}^{(\tau+1)} \longleftarrow \boldsymbol{\theta}^{(\tau)} - \alpha \nabla_{\boldsymbol{\theta}} J_i(\boldsymbol{\theta})$$
$$\boldsymbol{\theta}_j^{(\tau+1)} \longleftarrow \boldsymbol{\theta}_j^{(\tau)} - \alpha \frac{\partial J}{\partial \boldsymbol{\theta}_j}$$

How this relates to gradient descent?

In the setup of regression with the squares loss:

- residual $\Leftrightarrow$ negative gradient

How this relates to gradient descent?

In the setup of regression with the squares loss:

- residual $\Leftrightarrow$ negative gradient
- fit h to residual $\Leftrightarrow$ fit h to negative gradient

How this relates to gradient descent?

In the setup of regression with the squares loss:

- residual $\Leftrightarrow$ negative gradient
- fit h to residual $\Leftrightarrow$ fit h to negative gradient
- update H based on residual $\Leftrightarrow$ update H based on negative gradient

We are actually updating the model using gradient descent

Gradient Boosting Regression Trees Algorithm

1. Set a learning rate α
2. Train an initial model H_0
3. For $m = 1, \ldots, M$
    - 3.1 For every training point, estimate the negative gradient:
    $$-g(\mathbf{x}_i) = \frac{\partial \mathcal{L}}{\partial H(\mathbf{x}_i)}$$
    - 3.2 Obtain h_m that minimizes the negative gradient $g(\mathbf{x}_i)$
    - 3.3 Update H:
    $$H_{m+1} = H_m + \alpha h_m$$

Negative Gradients

In general, negative gradients $\not\Leftrightarrow$ residuals.

This parallel holds only when using the squared loss. However, we may be interested in using other loss functions which are better fitted to the targeted problem.

Solution: Update always by negative gradient rather than residuals

Boosting, Kaggle competitions & Reproducibility

Wrap-up

Wrap-up

- We introduced ensembles techniques, which are some of the best out of the box methods in ML
- We saw two techniques to build them: bagging and boosting
- Bagging: Methods are built in parallel.
- Bagging's most famous algorithm: Random forests
- Boosting: Methods are built sequentially.
- Boosting's notable algorithms: Adaboost and Gradient Boosting Trees

Key Concepts

- Bagging
- Boosting
- Gradient Descent over functions
- Out-of-Bag Error
- Negative Gradient

References

Further Reading and Useful Material

Source	**Notes**
The Elements of Statistical Learning	Ch. 10, 15, 16
Pattern Recognition and Machine Learning	Ch. 14