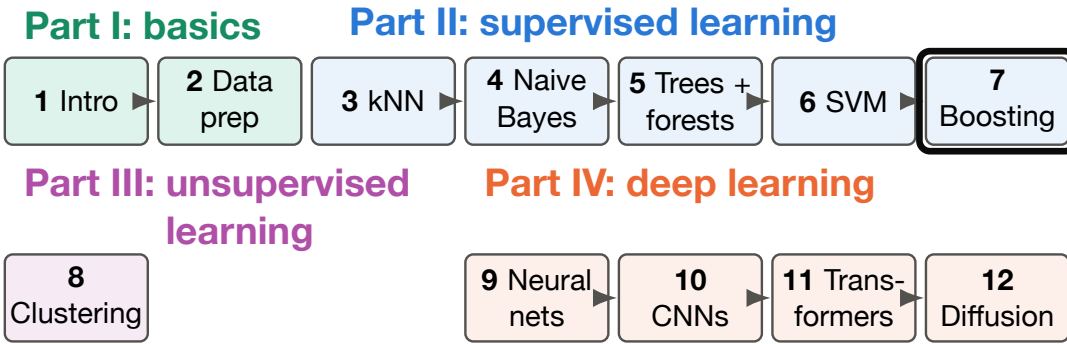
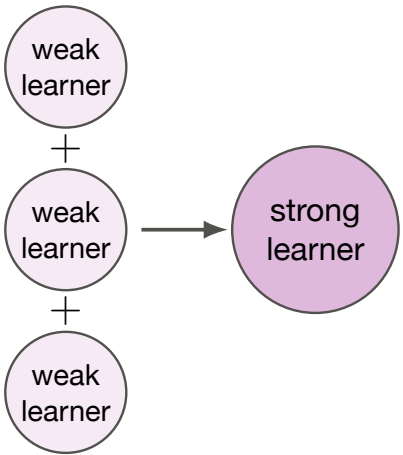
Boosting

229352 Lecture 7: boosting

Course map


Part I: basics
Part II: supervised learning
1 Intro
2 Data prep
3 kNN
4 Naive Bayes
5 Trees + forests
6 SVM
7 Boosting
Part III: unsupervised learning
Part IV: deep learning
8 Clustering
9 Neural nets
10 CNNs
11 Trans-formers
12 Diffusion

Main idea of boosting



A **weak learner** is only slightly better than random guessing.

Example: a **stump**, a decision tree with one split.

Boosting combines many weak learners into one **strong learner**, a model with a small error.

Boosting (Schapire, 1990)

- Schapire proved that weak learners can be boosted into a strong learner.
- AdaBoost (later in this lecture) uses weights: each new learner gives more weight to the points that the earlier learners got wrong.

R. E. Schapire, *The strength of weak learnability*, Machine Learning 5 (1990), 197–227

The additive model

An **additive model** is a weighted sum of M weak learners:

$$f(x) = \sum_{m=1}^{M} \alpha_m T_m(x)$$

$T_m(x)$: weak learner m, usually a small tree; α_m: its coefficient, the size of its vote.

Regression: predict $f(x)$. Classification (labels ± 1): $\widehat{y} = \operatorname{sign} f(x)$.

Data and loss function

Data: $(x_1, y_1), \ldots, (x_n, y_n)$.

x_i: the features of data point i; y_i: its label,
in $\{-1, +1\}$ for classification, a number for regression.

A **loss function** $L\big(y, f(x)\big)$ measures how far the prediction $f(x)$ is from the label y.

The model fits the training data well when the total loss $\sum_{i=1}^{n} L\big(y_i, f(x_i)\big)$ is small.

Boosting: the general algorithm

Start with $f_0(x) = 0$. For $m = 1, \ldots, M$:

1. Find α_m and T_m that minimize the **total** loss

$$\sum_{i=1}^{n} L\big(y_i,\ f_{m-1}(x_i) + \alpha_m T_m(x_i)\big).$$

2. Set $f_m(x) = f_{m-1}(x) + \alpha_m T_m(x)$; the earlier terms stay fixed.

Output $f_M(x) = \alpha_1 T_1(x) + \cdots + \alpha_M T_M(x)$.

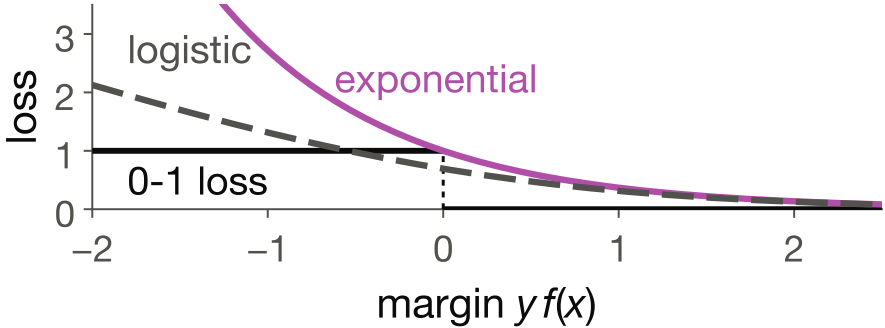
Examples of loss functions

Regression: squared loss $L\big(y, f(x)\big) = \frac{1}{2}\big(y - f(x)\big)^2$; the $\frac{1}{2}$ makes the derivative simpler.

Classification, labels $y \in \{-1, +1\}$: the **margin** $y\, f(x)$ (here a score, not the band width of Lecture 6) is positive when the sign of $f(x)$ is right.

- exponential loss $L\big(y, f(x)\big) = e^{-y f(x)}$ (used by AdaBoost)
- logistic loss $L\big(y, f(x)\big) = \log\big(1 + e^{-y f(x)}\big)$ (natural log)
- 0-1 loss: 1 if the sign of $f(x)$ is wrong ($y\, f(x) \leq 0$), else 0

Classification losses against the margin



For negative margins the exponential loss grows much faster than the logistic loss, which grows about linearly (7.39 against 2.13 at margin -2): AdaBoost gives a mislabeled point a large weight.

Cross-entropy and the logistic loss

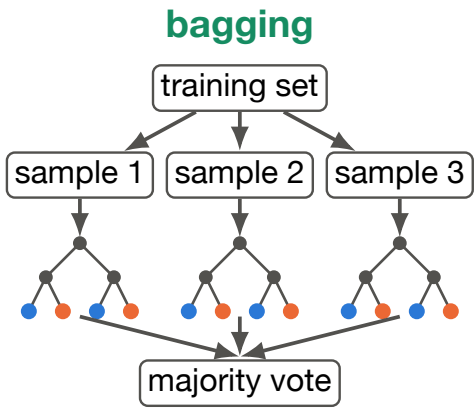
Labels $\tilde{y} \in \{0, 1\}$, predicted probability q of class 1:
the **binary cross-entropy** is $-\tilde{y} \log q - (1 - \tilde{y}) \log(1 - q)$.

With labels $y \in \{-1, +1\}$, set $\tilde{y} = (1 + y)/2$.
With a score $f(x)$, set $q = 1/\left(1 + e^{-f(x)}\right)$.

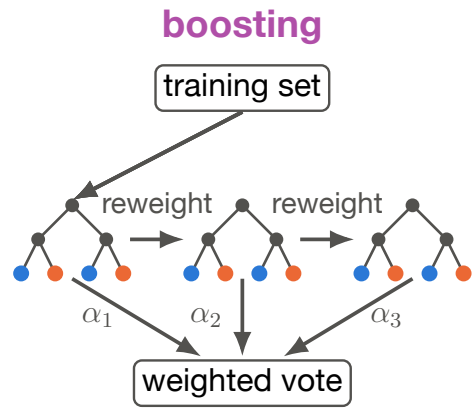
$$
\begin{aligned}
y = +1: \quad & -\log q = \log\left(1 + e^{-f(x)}\right) \\
y = -1: \quad & -\log(1 - q) = \log\left(1 + e^{f(x)}\right)
\end{aligned}
$$

In both cases, the loss is $\log\left(1 + e^{-yf(x)}\right)$, the logistic loss.

Bagging against boosting



in parallel, on bootstrap samples



in sequence, on reweighted data

Summary: boosting

- Boosting adds many weak learners: $f(x) = \sum_m \alpha_m T_m(x)$.
- One learner at a time: α_m and T_m minimize the total loss; earlier learners stay fixed.
- Losses: squared (regression); exponential and logistic, functions of the margin $y\, f(x)$ (classification).
- Bagging: trees in parallel, on bootstrap samples.
  Boosting: trees in sequence, on reweighted data.

AdaBoost

- **AdaBoost** (adaptive boosting): labels $y_i \in \{-1, +1\}$; weak learners $T_m(x) \in \{-1, +1\}$, here stumps.
- Each training point has a **weight** w_i. After each round, the points that T_m misclassified get more weight.
- Final classifier: the weighted vote $\widehat{y} = \operatorname{sign} f(x)$, $f(x) = \sum_m \alpha_m T_m(x)$.

Y. Freund and R. E. Schapire, *A decision-theoretic generalization of on-line learning and an application to boosting*, J. Computer and System Sciences 55 (1997), 119–139

The AdaBoost algorithm

1. Start with equal weights $w_i = 1/n$; they sum to 1.
2. For $m = 1, \ldots, M$:
   (a) Fit T_m **with weights** w_i: an error on point i costs w_i.
   (b) Weighted error $\mathrm{err}_m = \sum_{i=1}^{n} w_i \, I\big(y_i \neq T_m(x_i)\big)$; $I(\cdot)$: 1 if true, else 0.
   (c) $\alpha_m = \frac{1}{2} \log \dfrac{1 - \mathrm{err}_m}{\mathrm{err}_m}$ (natural log)

The AdaBoost algorithm (continued)

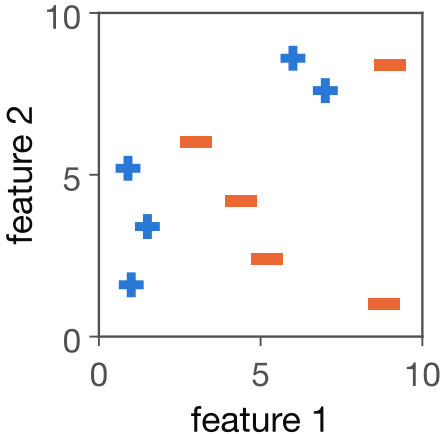
2. For $m = 1, \ldots, M$ (continued):

   (d) $w_i \leftarrow w_i \exp\big(-\alpha_m\, y_i\, T_m(x_i)\big)$, then divide each w_i by the sum.

   Misclassified: weight $\times\, e^{\alpha_m}$; correct: weight $\times\, e^{-\alpha_m}$.

3. Output $\widehat{y}(x) = \operatorname{sign} f(x)$, where $f(x) = \sum_{m=1}^{M} \alpha_m T_m(x)$.

Toy data

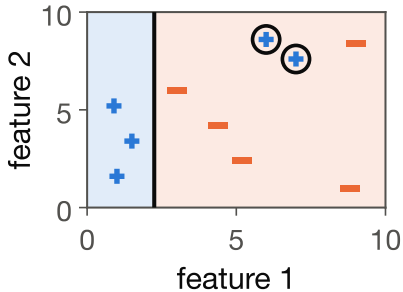


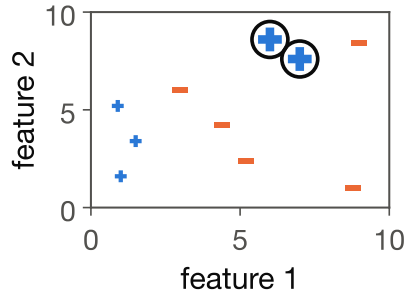
The data: 10 made-up points, 5 of class $+1$ (blue $+$), 5 of class -1 (orange $-$). Start: $w_i = 1/10$.

No stump classifies all 10 points correctly: the best one misclassifies 2.

Each round fits the stump with the smallest weighted error. Marker area: proportional to w_i.

AdaBoost round 1





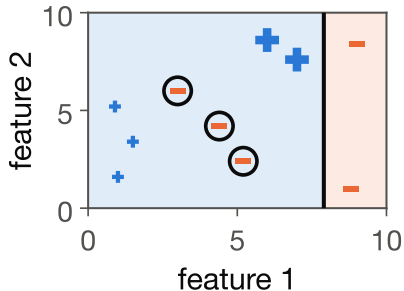
T_1: $+1$ if feature 1 < 2.25, else -1; shaded: its prediction

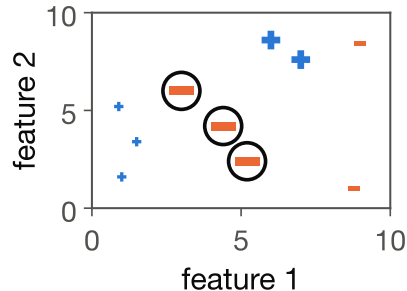
circled: misclassified; $\text{err}_1 = 2 \times \frac{1}{10} = 0.20$, $\alpha_1 = 0.69$

right: new weights $\frac{1}{4}$ (circled) and $\frac{1}{16}$; circled total $\frac{1}{2}$,

so T_1's new weighted error is $\frac{1}{2}$, as for random guessing

AdaBoost round 2



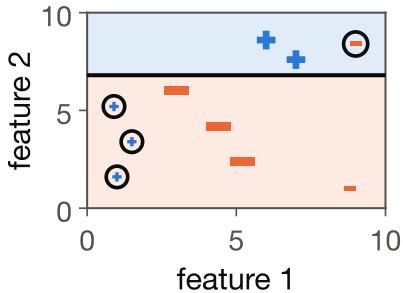


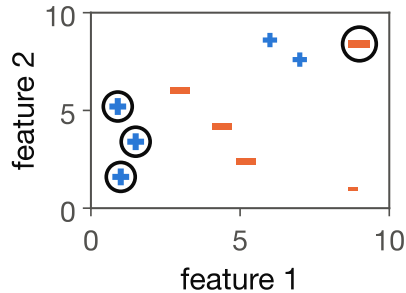
T_2: $+1$ if feature 1 < 7.9, else -1, fitted with the new weights

circled: misclassified; $\text{err}_2 = 3 \times \frac{1}{16} = 0.19$, $\alpha_2 = 0.73$

right: new weights

AdaBoost round 3



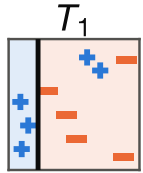
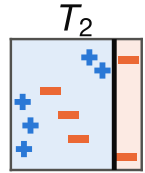
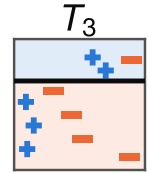


T_3: -1 if feature 2 < 6.8, else $+1$, fitted with the new weights

circled: misclassified; $\text{err}_3 = 4 \times \frac{1}{26} = 0.15$, $\alpha_3 = 0.85$

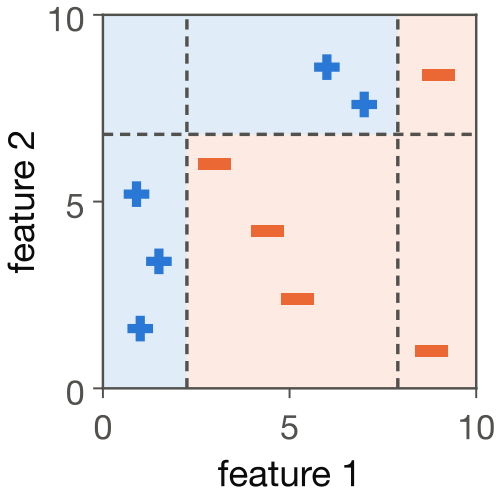
right: new weights, for a fourth round

AdaBoost final model

$$f(x) = 0.69$$  $$+ \; 0.73$$  $$+ \; 0.85$$ 

$\widehat{y}(x) = \text{sign}\, f(x)$: a weighted vote of three stumps

The final model classifies all 10 points



Shaded: the sign of $f(x)$.

Dashed: the three stump splits; f is constant in each of the six cells.

All 10 training points are classified correctly; each stump alone misclassifies 2 or more.

The three rounds in numbers

round m	err_m	α_m	**misclassified by** $\mathrm{sign}\, f_m$
1	1/5 = 0.20	0.69	2
2	3/16 = 0.19	0.73	3
3	2/13 = 0.15	0.85	0

Training points misclassified by $\mathrm{sign}\, f_m$: 2, 3, 0.

Checkpoint 1

A weak learner T_m has $\text{err}_m = 0.5$.

What is α_m, and what does it mean?

Checkpoint 1

A weak learner T_m has $\text{err}_m = 0.5$.

What is α_m, and what does it mean?

$$\alpha_m = \frac{1}{2} \log \frac{0.5}{0.5} = \frac{1}{2} \log 1 = 0$$

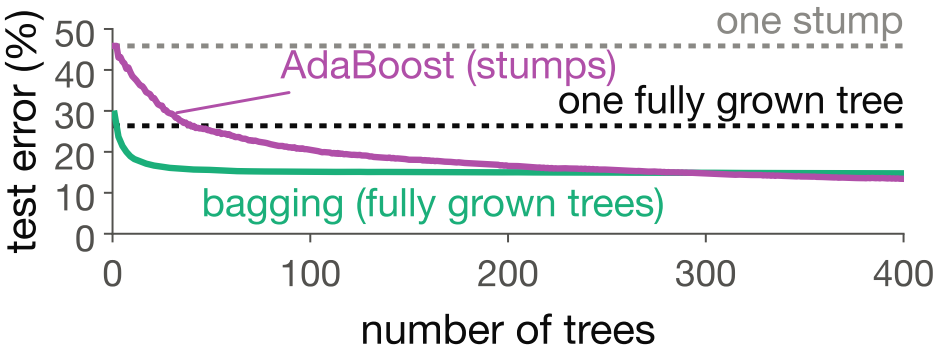
T_m gets no vote, and the weights do not change.

A simulated data set

- X_j: feature j (a random variable, so a capital), $j = 1, \ldots, 10$; the ten features are independent and standard normal (mean 0, variance 1).
- Class $Y = +1$ if $X_1^2 + \cdots + X_{10}^2 > 9.34$, else $Y = -1$.
- 9.34: the median of $X_1^2 + \cdots + X_{10}^2$, so the classes have about equal size.
- 10 training sets of 2,000 points each, one test set of 10,000 points.

Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Section 10.1, eq. (10.2)

Test error against the number of trees



mean test error over the 10 training sets

bagging's error levels off early; AdaBoost's still decreases

Test errors after 400 trees

model	test error (%)	sd (%)
one stump	45.9	0.4
one fully grown tree	26.4	0.9
bagging, 400 fully grown trees	14.8	0.6
AdaBoost, 400 stumps	13.4	0.4

mean and sd (standard deviation) over the 10 training sets;
no hyperparameter tuned

Summary: AdaBoost

- Weights w_i start at $1/n$; each round fits a stump with the current weights.
- $\alpha_m = \frac{1}{2} \log \frac{1-\text{err}_m}{\text{err}_m}$: a smaller error, a larger vote.
- Misclassified points gain weight ($\times\, e^{\alpha_m}$), so the next stump is fitted with larger weights on them.

From AdaBoost to gradient boosting

Round m of the general algorithm:
find α_m and T_m that minimize

$$\sum_{i=1}^{n} L\big(y_i,\ f_{m-1}(x_i) + \alpha_m T_m(x_i)\big)$$

- With $L\big(y, f(x)\big) = e^{-yf(x)}$, this step gives AdaBoost.
- For a general loss, this minimization is hard.
- **Gradient boosting** uses the gradient of the loss instead.

Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Section 10.4

Gradient

Let $g(\theta_1, \theta_2) = \theta_1^2 + \theta_2^2$. Compute:

1. $\dfrac{\partial g}{\partial \theta_1} =$

2. $\dfrac{\partial g}{\partial \theta_2} =$

3. $\nabla g(\theta_1, \theta_2) =$

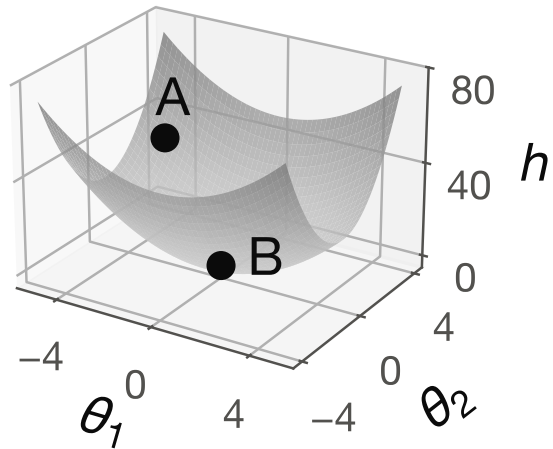
Why use the gradient?

Example:
$h(\theta_1, \theta_2) = \theta_1^2 + 3\theta_2^2$.

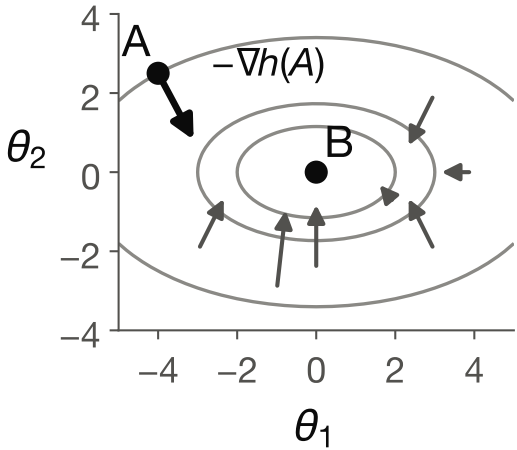
Start: $A = (-4, 2.5)$.

Goal: the minimum
$B = (0, 0)$.

Which way should we go from A?



Why use the gradient? (2)



Contours: curves on which h is constant.

The **gradient**: $\nabla h = \left(\frac{\partial h}{\partial \theta_1}, \frac{\partial h}{\partial \theta_2}\right)$. Arrows: $-\nabla h$, shortened.

$-\nabla h$: the direction in which h decreases fastest, at right angles to the contours.

$-\nabla h$ need not point at B.

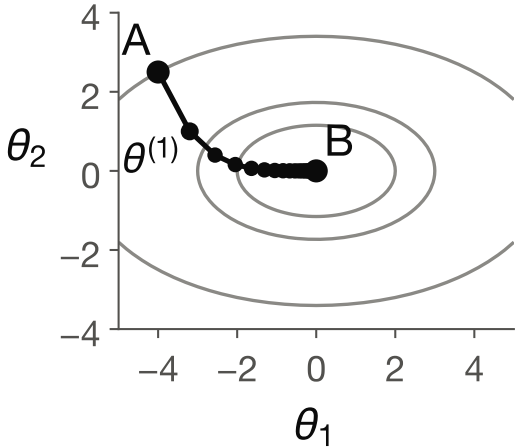
Gradient descent algorithm

Gradient descent finds $\theta = (\theta_1, \ldots, \theta_p)$ that minimizes a function $g(\theta)$.

$\eta > 0$: the **learning rate**; K: the number of steps.

1. Choose a starting point $\theta^{(0)}$, for example at random.
2. For $t = 1, \ldots, K$: $\quad \theta^{(t)} = \theta^{(t-1)} - \eta \, \nabla g\big(\theta^{(t-1)}\big)$
3. Return $\theta^{(K)}$.

Gradient descent: an example



$h(\theta_1, \theta_2) = \theta_1^2 + 3\theta_2^2, \ \eta = 0.1,$
$\theta^{(0)} = A = (-4, 2.5).$

$\nabla h(A) = (-8, 15),$ so
$\theta^{(1)} = A - \eta \nabla h(A) = (-3.2, 1).$

After $K = 20$ steps:
$\theta^{(20)} = (-0.05,\ 0.00) \approx B.$

Summary: gradient descent

- The gradient ∇g: the vector of the partial derivatives of g.
- $-\nabla g(\theta)$: the direction in which g decreases fastest at θ.
- Gradient descent repeats $\theta^{(t)} = \theta^{(t-1)} - \eta \nabla g\left(\theta^{(t-1)}\right)$, for $t = 1, \ldots, K$; η is the learning rate.

Gradient descent on the predictions

- The total loss $\sum_i L\big(y_i, f(x_i)\big)$ depends on f only through $f(x_1), \ldots, f(x_n)$.
- Take $\theta = \big(f(x_1), \ldots, f(x_n)\big)$ and $g(\theta) = \sum_i L(y_i, \theta_i)$, the total loss; so $p = n$. A gradient step moves each $f(x_i)$ in the direction

$$r_i = -\frac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}.$$

- This changes f only at the training points. A regression tree fitted to the r_i gives a step at every x.

Gradient boosting algorithm

1. Start with the constant $f_0(x) = \gamma$ that minimizes $\sum_{i=1}^{n} L(y_i, \gamma)$.
2. For $m = 1, \ldots, M$:
   (a) For $i = 1, \ldots, n$: $\quad r_{im} = -\left[\dfrac{\partial L\big(y_i, f(x_i)\big)}{\partial f(x_i)}\right]_{f=f_{m-1}}$
   (b) Fit a **regression** tree to the targets $r_{1m}, \ldots, r_{nm}$; its J_m leaves give regions R_{jm}, $j = 1, \ldots, J_m$.

Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Algorithm 10.3

Gradient boosting algorithm (continued)

2. For $m = 1, \ldots, M$ (continued):
   (c) For $j = 1, \ldots, J_m$ (R_{jm}: region of leaf j):

$$\gamma_{jm} = \arg\min_{\gamma} \sum_{x_i \in R_{jm}} L\big(y_i,\ f_{m-1}(x_i) + \gamma\big)$$

   (d) Update $f_m(x) = f_{m-1}(x) + \sum_{j=1}^{J_m} \gamma_{jm}\, I(x \in R_{jm})$

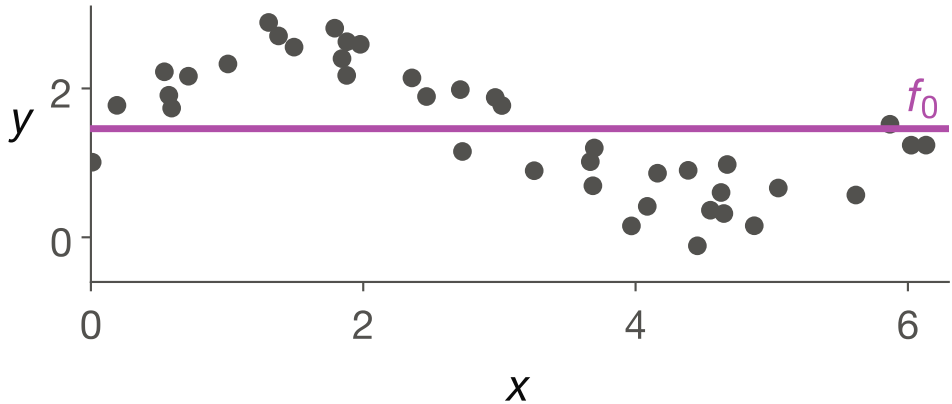
3. Output $f_M(x)$.

Squared loss: fit the residuals

$$L\big(y, f(x)\big) = \tfrac{1}{2}\big(y - f(x)\big)^2 \quad \Longrightarrow \quad r_{im} = y_i - f_{m-1}(x_i)$$

- The negative gradient is the **residual**: the part of y_i that f_{m-1} has not fitted yet.
- f_0 is the average of the y_i. γ_{jm} is the average residual of the points in R_{jm}: the leaf value of the regression tree.
- So each round fits a regression tree to the residuals.

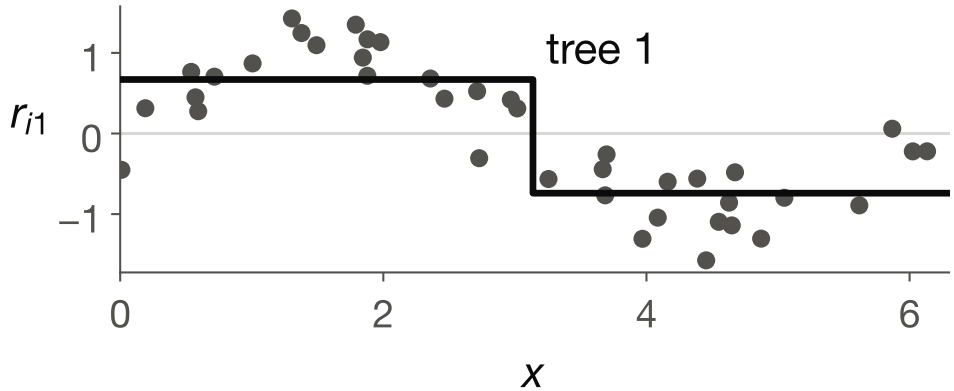
Example: 40 points, squared loss



each tree is a regression stump (one split), added in full

$f_0(x) = 1.46$, the average of the y_i

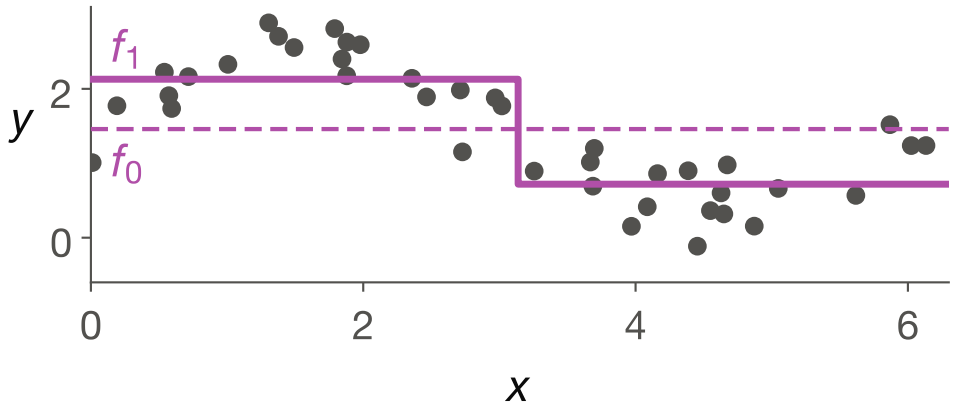
Round 1: a stump fitted to the residuals



residuals $r_{i1} = y_i - f_0(x_i)$

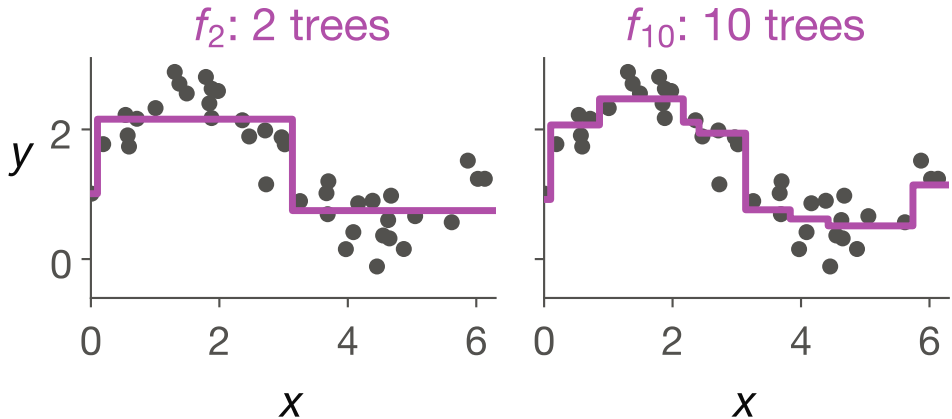
tree 1: split at $x = 3.14$; leaf values $+0.67$ and -0.74

Round 1: the new model



$f_1(x) = f_0(x) +$ tree 1(x), the leaf value of tree 1 at x

After 2 and 10 rounds



each round adds a stump fitted to the current residuals

Learning rate

Libraries scale each new tree by the learning rate η of gradient descent, $0 < \eta \leq 1$:

$$f_m(x) = f_{m-1}(x) + \eta \sum_{j=1}^{J_m} \gamma_{jm} \, I(x \in R_{jm})$$

With squared loss, f moves by η times a tree fitted to the negative gradient.

In the example, each tree was added in full: $\eta = 1$.

Learning rate in practice

- A smaller η needs more trees M; η and M are chosen together, for example by cross-validation.
- scikit-learn's gradient boosting classes use `learning_rate=0.1` by default.

Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Section 10.12.1

Summary: gradient boosting

- Each round computes the negative gradient r_{im} of the loss at the predictions $f_{m-1}(x_i)$.
- A regression tree is fitted to the r_{im}; each leaf value γ_{jm} minimizes the loss in its leaf.
- Squared loss: $r_{im} = y_i - f_{m-1}(x_i)$, the residual; γ_{jm}: the average residual.
- A learning rate $0 < \eta \leq 1$ scales each tree; a smaller η needs more trees.

Checkpoint 2

Squared loss $\frac{1}{2}\big(y - f(x)\big)^2$.

After round m: $y_i = 5$, $f_m(x_i) = 3$.

What target does tree $m+1$ get at this point?

Checkpoint 2

Squared loss $\frac{1}{2}\big(y - f(x)\big)^2$.

After round m: $y_i = 5$, $f_m(x_i) = 3$.

What target does tree $m+1$ get at this point?

$r_{i,m+1} = y_i - f_m(x_i) = 5 - 3 = 2$: the residual

Gradient boosting in scikit-learn (1)

```
from sklearn.datasets import load_breast_cancer
from sklearn.ensemble import HistGradientBoostingClassifier
from sklearn.metrics import classification_report
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
X, y = load_breast_cancer(return_X_y=True)
y = 1 - y  # 1 = malignant (scikit-learn: 0 = malignant)
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.3, stratify=y, random_state=0)
```

Breast Cancer Wisconsin (Diagnostic), UCI, 1993, doi:10.24432/C5DW2B, CC BY 4.0

Precision, recall and F1

- For class c: **precision**: the share of the points predicted as c that truly belong to c; **recall**: the share of the points in c that are predicted as c.
- **F1** $= 2 \cdot \text{precision} \cdot \text{recall} / (\text{precision} + \text{recall})$.

Gradient boosting in scikit-learn (2)

```
# max_iter: number of trees M (binary: one per iteration)
parameters = {"max_iter": [100, 200, 300],
              "max_depth": [2, 4, 6],
              "min_samples_leaf": [1, 10, 100]}
hgb = HistGradientBoostingClassifier(random_state=0)
clf = GridSearchCV(hgb, parameters, scoring="f1", cv=5)
clf.fit(X_train, y_train)
print(clf.best_params_)
best = clf.best_estimator_  # refit on all of X_train
y_pred = best.predict(X_test)
print(classification_report(y_test, y_pred, digits=3))
```

`GridSearchCV`: 27 hyperparameter settings, each scored by 5-fold cross-validation on the training set.

Output of the code

```
{'max_depth': 6, 'max_iter': 100, 'min_samples_leaf': 10}
              precision    recall  f1-score   support

           0      0.972     0.972     0.972       107
           1      0.953     0.953     0.953        64

    accuracy                          0.965       171
   macro avg      0.963     0.963     0.963       171
weighted avg      0.965     0.965     0.965       171
```

Labels 0 and 1 here (1 = malignant). `scoring="f1"`: the F1 of class 1.

Reading the classification report

- **support**: the number of test points in class c.
- **accuracy**: the share of test points classified correctly.
- **macro avg**: the plain average over the two classes;
  weighted avg: the average weighted by support.

Gradient boosting with xgboost

```
# install once, in a terminal: pip install xgboost
from sklearn.metrics import accuracy_score, f1_score
from xgboost import XGBClassifier
# default hyperparameters, except the seed and the threads
xgb = XGBClassifier(random_state=0, n_jobs=4)
xgb.fit(X_train, y_train)
y_pred = xgb.predict(X_test)
print(f"F1 of class 1: {f1_score(y_test, y_pred):.3f}")
print(f"accuracy: {accuracy_score(y_test, y_pred):.3f}")
```

Output:

```
F1 of class 1: 0.904
accuracy: 0.924
```

Comparison of learning methods (1)

Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels	MART (boosted trees)
Natural handling of data of "mixed" type	▼	▼	▲	▲	▼	▲
Handling of missing values	▼	▼	▲	▲	▲	▲
Robustness to outliers in input space	▼	▼	▲	▼	▲	▲
Insensitive to monotone transformations of inputs	▼	▼	▲	▼	▼	▲

▲ good, ♦ fair, ▼ poor. MARS: multivariate adaptive regression splines.

Neural nets to k-NN: Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Table 10.1.
MART: T. Hastie, *Boosting* (talk), Stanford, 2003, slide 43.

Comparison of learning methods (2)

Characteristic	Neural nets	SVM	Trees	MARS	k-NN, kernels	MART (boosted trees)
Computational scalability (large n)	▼	▼	▲	▲	▼	▲
Ability to deal with irrelevant inputs	▼	▼	▲	▲	▼	▲
Ability to extract linear combinations of features	▲	▲	▼	▼	♦	♦
Interpretability	▼	▼	♦	▲	▼	▲
Predictive power	▲	▲	▼	♦	▲	▲

▲ good, ♦ fair, ▼ poor. MARS: multivariate adaptive regression splines.

Neural nets to k-NN: Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Table 10.1.
MART: T. Hastie, *Boosting* (talk), Stanford, 2003, slide 43.

Boosted trees in this comparison

- Trees have most of the good properties in the table, but poor predictive power.
- Boosting often improves the accuracy of trees greatly and preserves most of these properties.

Hastie, Tibshirani and Friedman, ESL, 2nd ed., 2009, Section 10.7