Boosting

CS 229: Machine Learning

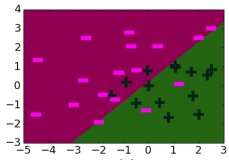
Emily Fox
Stanford University

February 14, 2024

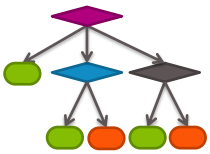
Slides include content developed by and co-developed with Carlos Guestrin

1

Simple (weak) classifiers are good!



Logistic regression w. simple features



Shallow decision trees

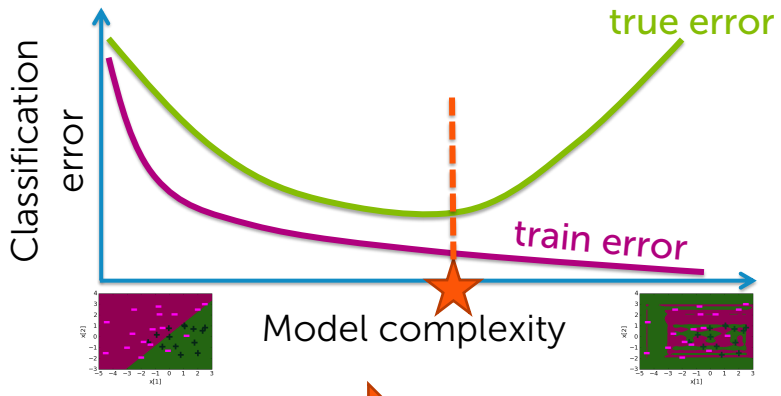
Decision stumps

Low variance. Learning is fast!

But high bias…

2

Finding a classifier that's just right



Weak learner → Need stronger learner

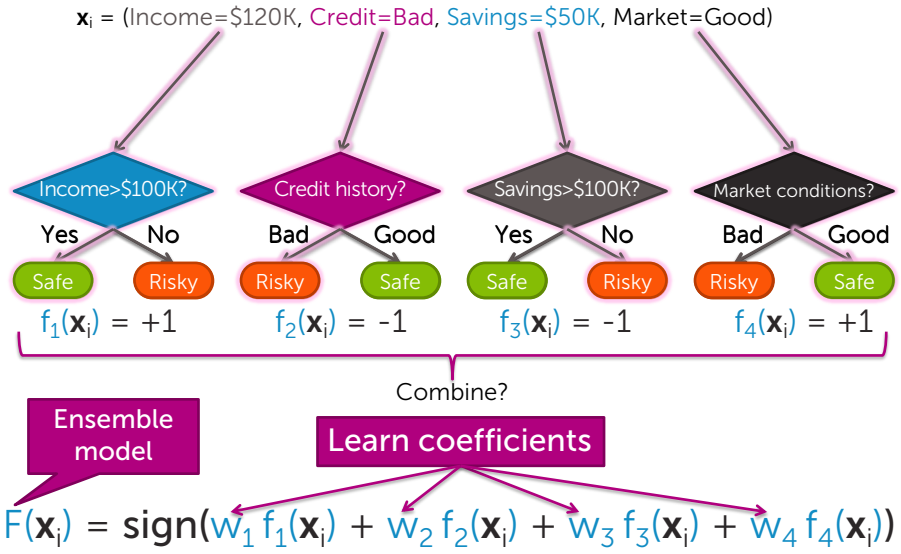
Option 1: add more features or depth
Option 2: ?????

3

Ensemble classifier

4

Ensemble methods: Each classifier "votes" on prediction



5

Ensemble classifier in general

- Goal:
  - Predict output y
    - Either +1 or -1
  - From input **x**
- Learn ensemble model:
  - Classifiers: $f_1(\mathbf{x}), f_2(\mathbf{x}), \ldots, f_T(\mathbf{x})$
  - Coefficients: $\hat{w}_1, \hat{w}_2, \ldots, \hat{w}_T$
- Prediction:

total # of classifiers in ensemble

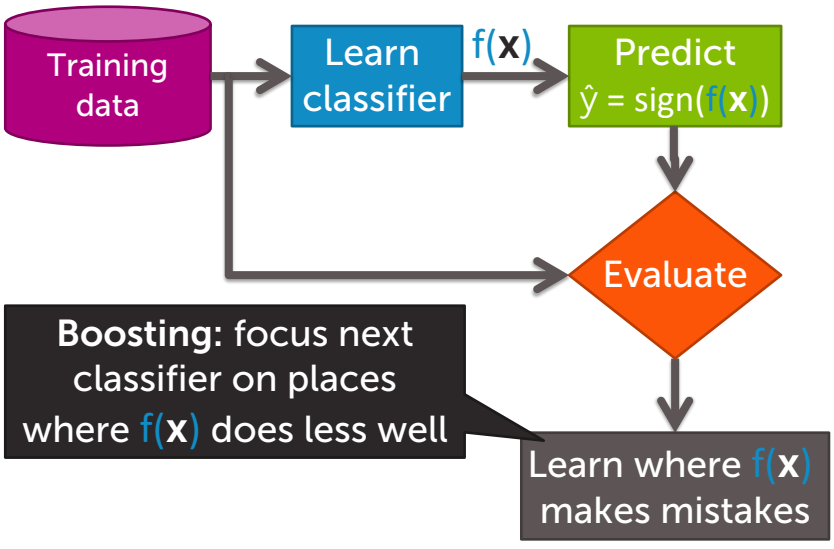
$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

6

Boosting

7

Boosting = Focus learning on "hard" points

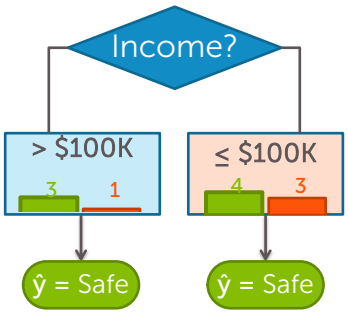


8

More weight on "hard" or more important points

Increase weight **α** of harder/misclassified points

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	**Risky**	**1.5**
C	$110K	**Risky**	**1.2**
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	**Risky**	**3**
C	$60K	**Risky**	**2**
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

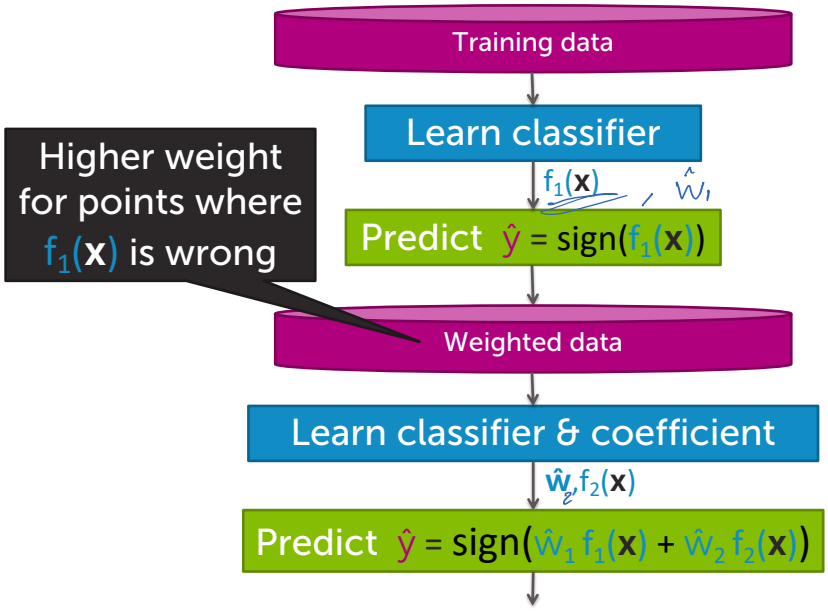


Weighted dataset:

- Each $\mathbf{x}_i, y_i$ weighted by α_i
  - More important point = higher weight α_i

9

Boosting = Greedy learning ensembles from data



10

Learning from weighted data

11

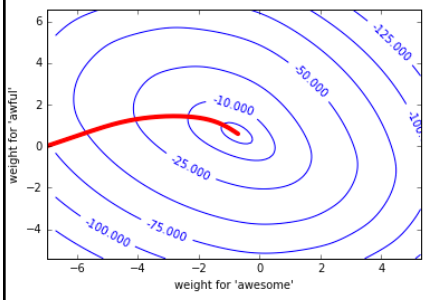
Learning from weighted data in general

Often, learning from weighted data treats data point i as α_i replicates of that data point

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

12

Gradient ascent for logistic regression on *unweighted data*



init $\mathbf{w}^{(1)}=0$ (or randomly, or smartly), t=1

while $||\nabla \ell(\mathbf{w}^{(t)})|| > \varepsilon$

Difference between truth and prediction

for j=0,…,D

partial[j] $= \sum_{i=1}^{N} h_j(\mathbf{x}_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\right)$

$w_j^{(t+1)} \leftarrow w_j^{(t)} + \eta$ partial[j]

t ← t + 1

Modify the logistic regression gradient update for *weighted data*

Sum over data points

$$\mathbf{w}_j^{(t+1)} \leftarrow \mathbf{w}_j^{(t)} + \eta \sum_{i=1}^{N} \alpha_i h_j(\mathbf{x}_i)\left(\mathbb{1}[y_i = +1] - P(y = +1 \mid \mathbf{x}_i, \mathbf{w}^{(t)})\right)$$

Weigh each point by α_i

How to learn the "best" decision stump?

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

Goal:
Choose best feature (categorical input) or feature/threshold pair (real-valued input)

Questions:

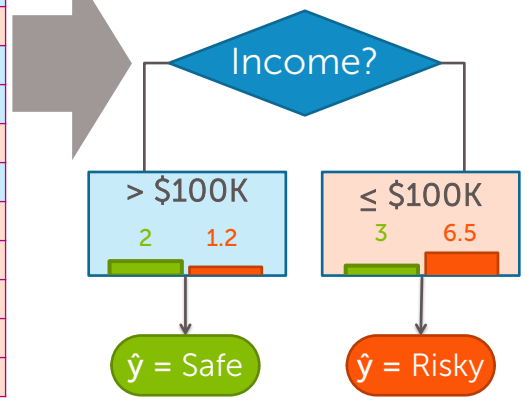
1. For given feature or feature/threshold, how to determine classifier output using weighted data?
2. For that classifier, how to compute its accuracy on the training data?
3. For real-valued features, how to select the best threshold on weighted data?
4. Based on the above, select the best decision stump

15

Learning a decision stump on weighted data

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

For a given potential split, learn classifier:



16

How to learn the "best" decision stump?

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	1.2
A	$110K	Safe	0.8
A	$90K	Safe	0.6
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	0.8
A	$60K	Safe	0.7
A	$98K	Safe	0.9

Goal:
Choose best feature (categorical input) or feature/threshold pair (real-valued input)

Questions:

1. For given feature or feature/threshold, how to determine classifier output using weighted data?
2. For that classifier, how to compute its accuracy on the training data?
3. For real-valued features, how to select the best threshold on weighted data?
4. Based on the above, select the best decision stump

17

Learning a decision stump on weighted data

Credit	Income	y	Weight α
A	$130K	Safe	0.5
B	$80K	Risky	1.5
C	$110K	Risky	**1.2**
A	$110K	Safe	0.8
A	$90K	Safe	**0.6**
B	$120K	Safe	0.7
C	$30K	Risky	3
C	$60K	Risky	2
B	$95K	Safe	**0.8**
A	$60K	Safe	**0.7**
A	$98K	Safe	**0.9**

Compute weighted error:

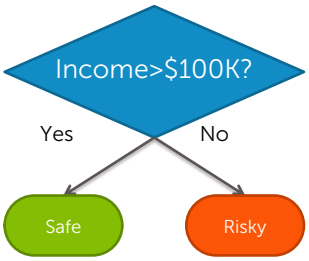
$$\text{Weighted error} = \frac{\text{total weight of mistakes}}{\text{total weight}}$$

$$= \frac{1.2+0.6+0.8+0.7+0.9}{12.7}$$

$$= \frac{4.2}{12.7} = 0.33$$

18

How to learn the "best" decision stump?



Goal:
Choose best feature (categorical input) or feature/threshold pair (real-valued input)

Questions:

1. For given feature or feature/threshold, how to determine classifier output using weighted data?
2. For that classifier, how to compute its accuracy on the training data?
3. For real-valued features, how to select the best threshold on weighted data?
4. Based on the above, select the best decision stump

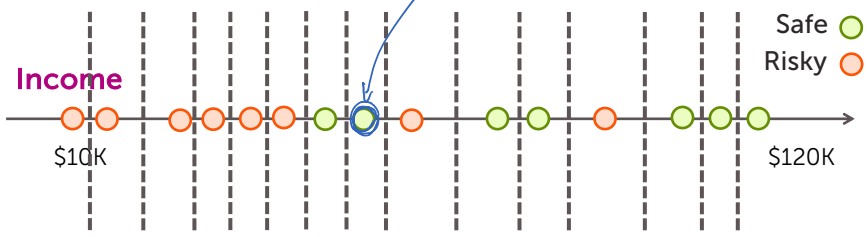
19

Only need to consider mid-points

Finite number of splits to consider

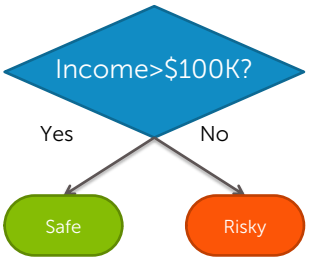
weight

Same alg. as before, but using weighted error to choose t^*



20

How to learn the "best" decision stump?



Goal:
Choose best feature (categorical input) or feature/threshold pair (real-valued input)

Questions:

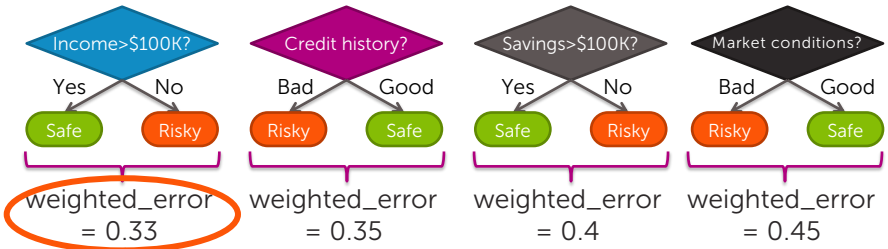
1. For given feature or feature/threshold, how to determine classifier output using weighted data?
2. For that classifier, how to compute its accuracy on the training data?
3. For real-valued features, how to select the best threshold on weighted data?
4. Based on the above, select the best decision stump

21

Finding best next decision stump $f_t(x)$

Which weak learner to choose?

Consider splitting on each feature:



f_t =

22

AdaBoost algorithm

AdaBoost: learning ensemble

[Freund & Schapire 1999]

- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

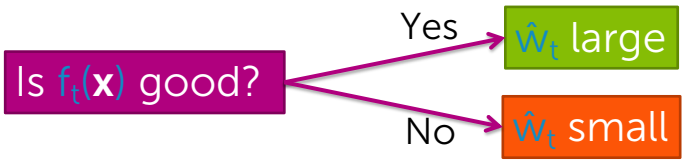
Problem 1: How much do I trust f_t?

Problem 2: Weight mistakes more

Computing coefficient $\hat{w}_t$

25

AdaBoost: Computing coefficient $\hat{w}_t$ of classifier $f_t(\mathbf{x})$



$f_t(\mathbf{x})$ is good ➔ f_t has low training error

26

AdaBoost:
Formula for computing coefficient $\hat{w}_t$ of classifier $f_t(x)$

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

Is $f_t(x)$ good?

- Yes
- No

weighted_error(f_t) on training data	$\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}$	$\hat{w}_t$
0.01	$\frac{1-0.01}{0.01} = 99$	2.3
0.5	1	0
0.99	0.01	-2.3

terrible classifier! But, $-f_t$ is awesome!

27

AdaBoost: learning ensemble

- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(x)$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(x)\right)$$

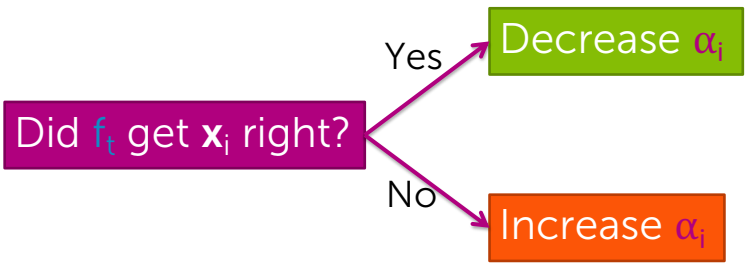
$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

28

Recompute weights α_i

29

AdaBoost: Updating weights α_i based on where classifier $f_t(\mathbf{x})$ makes mistakes



30

AdaBoost: Formula for updating weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i)=y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i)\neq y_i \end{cases}$$

$\alpha_i \leftarrow \alpha_i \exp(-\hat{w}_t f_t(x_i) y_i)$

Did f_t get $\mathbf{x}_i$ right? Yes / No

$f_t(\mathbf{x}_i)=y_i$?	$\hat{w}_t$	Multiply α_i by	Implication
yes	2.3	$e^{-2.3} = 0.1$	decrease importance of x_i, y_i
yes	0	$e^{-0} = 1$	no change
no	2.3	$e^{2.3} = 9.98$	increase importance of x_i, y_i
no	0	$e^{0} = 1$	no change

31

AdaBoost: learning ensemble

- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{w}_t f_t(\mathbf{x})\right)$$

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i)=y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i)\neq y_i \end{cases}$$

32

AdaBoost: Normalizing weights α_i

If $\mathbf{x}_i$ often mistake, weight α_i gets very **large**

If $\mathbf{x}_i$ often correct, weight α_i gets very **small**

Can cause numerical instability after many iterations

Normalize weights to add up to 1 after every iteration

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

33

AdaBoost: learning ensemble

- Start with same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$ with data weights α_i
  - Compute coefficient $\hat{w}_t$
  - Recompute weights α_i
  - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

$$\hat{\mathbf{w}}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right)$$

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{w}_t}, & \text{if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

34

AdaBoost example:
A visualization

35

Boosted decision stumps

- Start same weight for all points: $\alpha_i = 1/N$
- For t = 1,…,T
  - Learn $f_t(\mathbf{x})$: pick **decision stump** with lowest weighted training error according to α_i
    - Compute coefficient $\hat{w}_t$
    - Recompute weights α_i
    - Normalize weights α_i
- Final model predicts by:

$$\hat{y} = sign\left(\sum_{t=1}^{T} \hat{\mathbf{w}}_t f_t(\mathbf{x})\right)$$

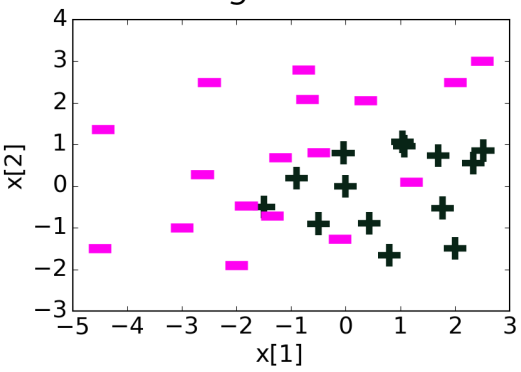
36

t=1: Just learn a classifier on original data

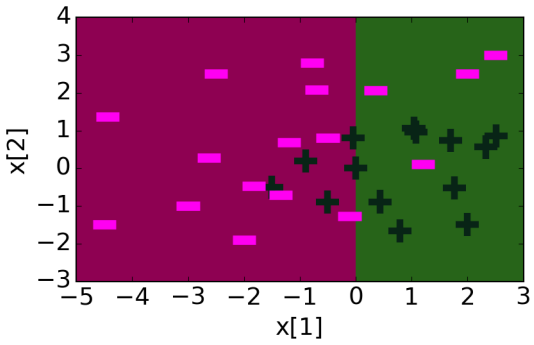
all pts have same weight

standard learning

Original data



Learned decision stump $f_1(\mathbf{x})$



37

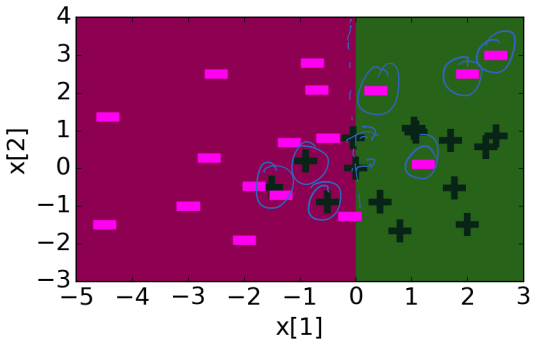
Determine weight (trust) of classifier

$$\hat{W}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

$$= \frac{1}{2} \ln \left(\frac{1 - \frac{7}{31}}{\frac{7}{31}} \right)$$

$$= 0.62$$

Learned decision stump $f_1(\mathbf{x})$



38

Updating weights α_i

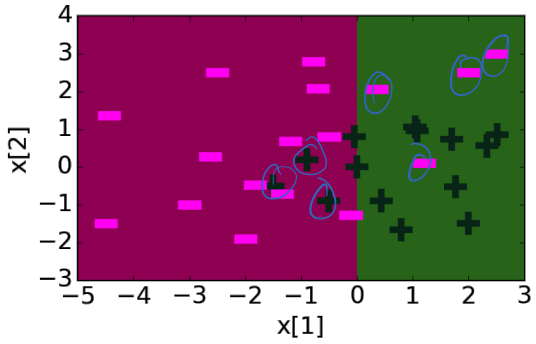
$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{w}_t} = 0.54\alpha_i, & \text{if } f_t(\mathbf{x}_i)=y_i \\ \alpha_i e^{\hat{w}_t} = 1.85\alpha_i, & \text{if } f_t(\mathbf{x}_i)\neq y_i \end{cases}$$

0.61

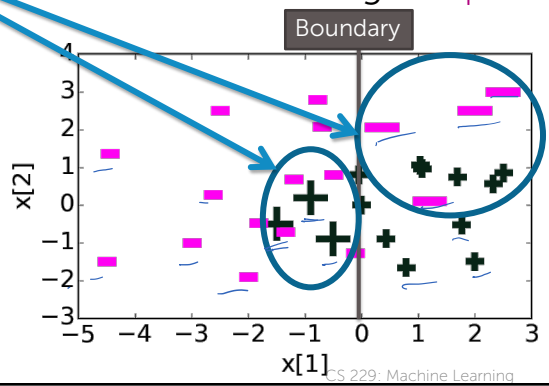
0.61

Increase weight α_i of misclassified points

Learned decision stump $f_1(\mathbf{x})$



New data weights α_i

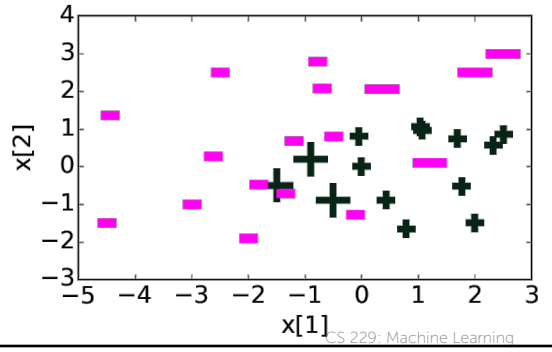


39

Normalize weights…

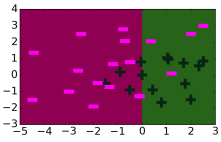
$$\alpha_i \leftarrow \frac{\alpha_i}{\sum_{j=1}^{N} \alpha_j}$$

New data weights α_i

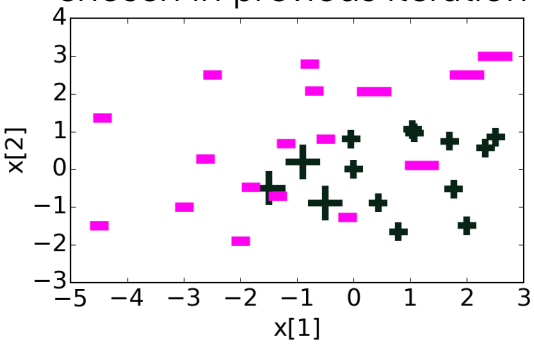


40

t=2: Learn classifier on weighted data

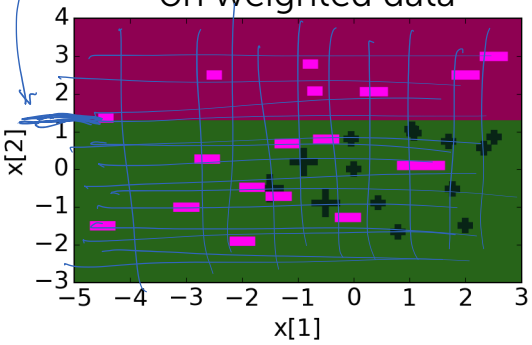
$f_1(\mathbf{x})$ 

Weighted data: using α_i chosen in previous iteration



Learned decision stump $f_2(\mathbf{x})$ on weighted data

best fit for this weighted dataset (lowest weighted classification error)



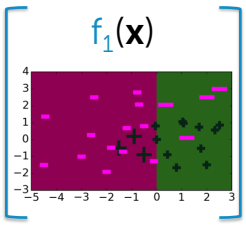
41

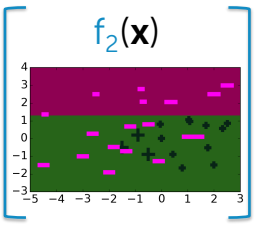
Ensemble becomes weighted sum of learned classifiers

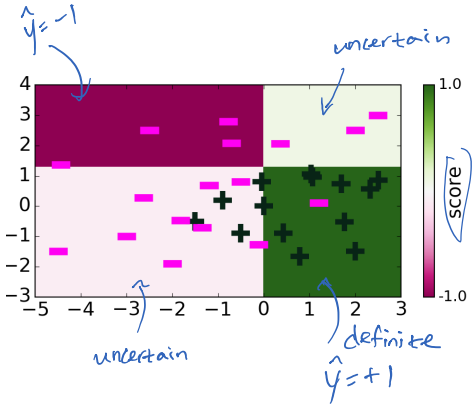
$\hat{y} = \text{sign}(\hat{w}_1 f(x) + \hat{w}_2 f(x))$

$\text{score}(x)$

$\text{score}(x) =$

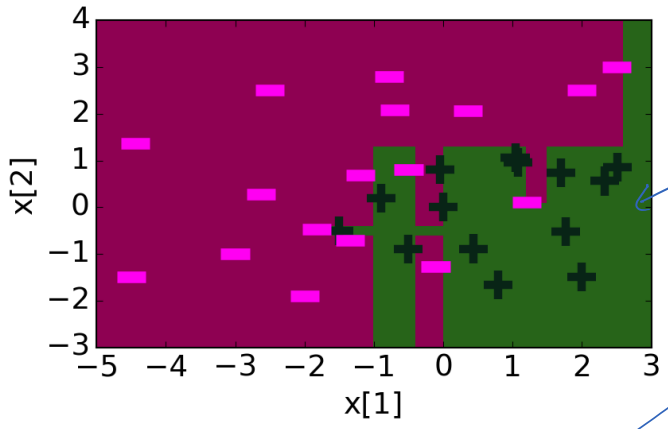
$\hat{w}_1$ $f_1(\mathbf{x})$ 

0.62 + $\hat{w}_2$ $f_2(\mathbf{x})$  =



42

Decision boundary of ensemble classifier after 30 iterations



complex decision boundary → probably overfitting

training_error = 0

43

AdaBoost more formally

44

AdaBoost convergence

45

Boosting question

"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*

↓

Yes! *Schapire (1990)*

↓

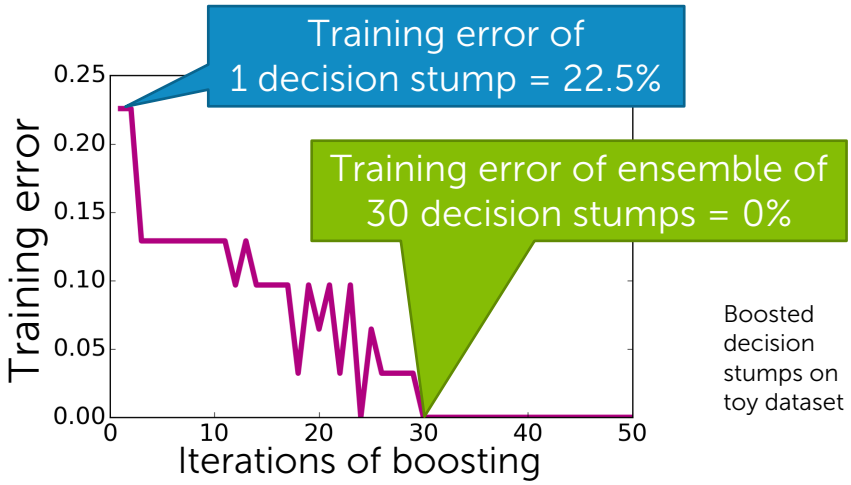
Boosting

↓

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions

46

After some iterations, training error of boosting goes to zero!!!



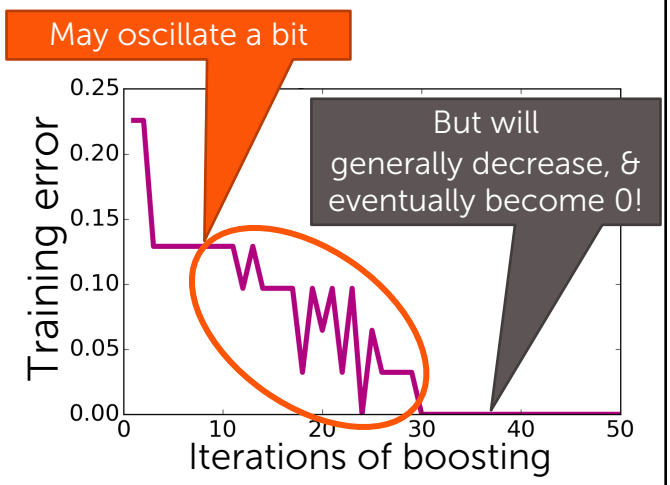
Boosted decision stumps on toy dataset

47

AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞



48

Condition of AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞

Condition = At every t, can find a weak learner with weighted_error(f_t) < 0.5

Not always possible

Extreme example: No classifier can separate a +1 on top of -1

Nonetheless, boosting often yields great training error

49

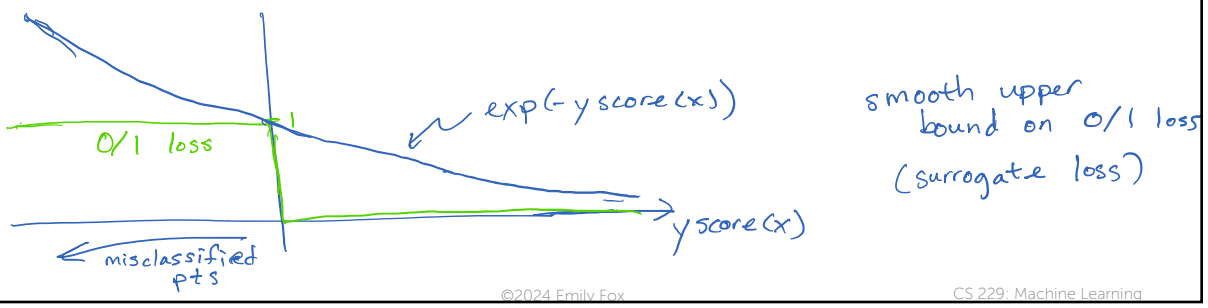
AdaBoost Theorem more formally

Training error of final classifier is bounded by:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[F(x_i)\neq y_i] \leq \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i \text{score}(x_i))$$

indicator

Where $\text{score}(x) = \sum_t \hat{w}_t f_t(x)$; $F(x) = sign(\text{score}(x))$



50

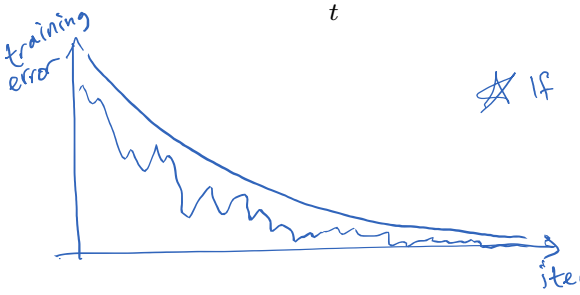
AdaBoost Theorem more formally

Training error of final classifier is bounded by:

$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[F(x_i)\neq y_i] \leq \frac{1}{N}\sum_{i=1}^{N}\exp(-y_i \text{score}(x_i)) = \prod_{t=1}^{T} Z_t$$

$$Z_t = \sum_{i=1}^{N} \alpha_{i,t} \exp(-\hat{w}_t y_i f_t(x_i))$$

Where $\text{score}(x) = \sum_t \hat{w}_t f_t(x)$; $F(x) = sign(\text{score}(x))$



If $Z_t < 1 \ \forall t$, $\prod_t Z_t$ decreases exp. fast ⇒ upper bound on training error ⇒ training error → 0 exp-fast as $T \to \infty$

51

AdaBoost Theorem more formally

If we minimize $\prod_{t=1}^{T} Z_t$, we minimize our training error

We can tighten this bound greedily by choosing $\hat{w}_t$, f_t on each iteration to minimize Z_t:

$$(f_t, \hat{w}_t) = \arg\min_{f,w} \sum_{i=1}^{N} \alpha_{i,t} e^{-w y_i f(\mathbf{x}_i)}$$

For Boolean target function, this is accomplished by [Freund & Schapire '97]:

f_t min weighted class. error

$$\hat{w}_t = \frac{1}{2}\ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$$

ϵ_t: weighted class. error

52

AdaBoost Theorem more formally

If each classifier is (at least slightly) better than random

$$weighted\_error(f_t) = \epsilon_t < 0.5$$

AdaBoost will achieve zero training error (exponentially fast):

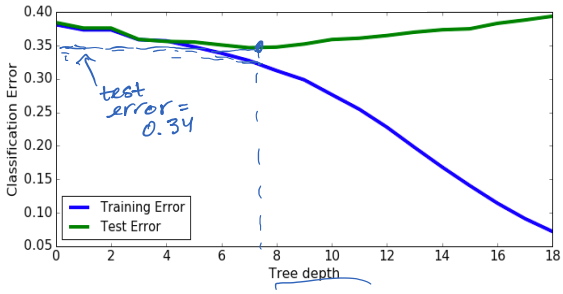
$$\frac{1}{N}\sum_{i=1}^{N}\mathbb{I}[F(x_i) \neq y_i] \leq \prod_{t=1}^{T} Z_t \leq \exp\left(-2\sum_{t=1}^{T}(1/2-\epsilon_t)^2\right)$$

upper bound for $\epsilon_t < 0.5$

as ϵ_t moves away from $\frac{1}{2}$, exponents gets bigger

Boosting is robust to overfitting

Decision trees on loan data

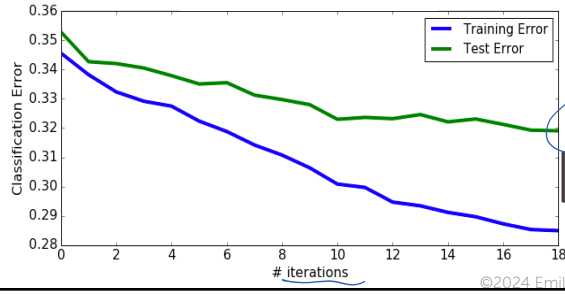


39% test error

Overfitting

8% training error

Boosted decision stumps on loan data



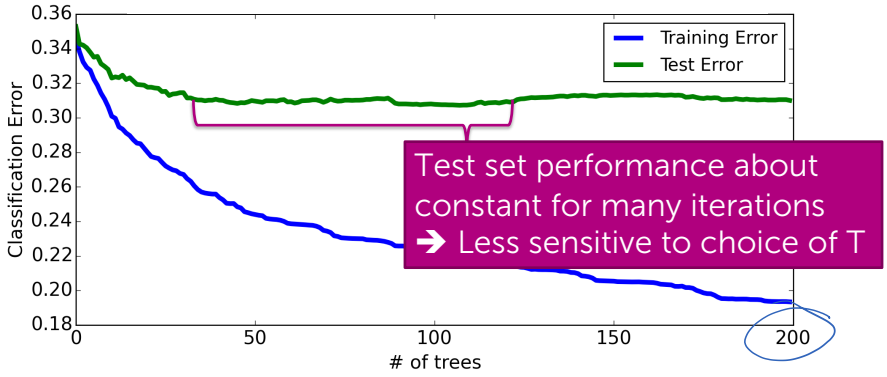
32% test error

Better fit & lower test error

28.5% training error

55

Boosting tends to be robust to overfitting

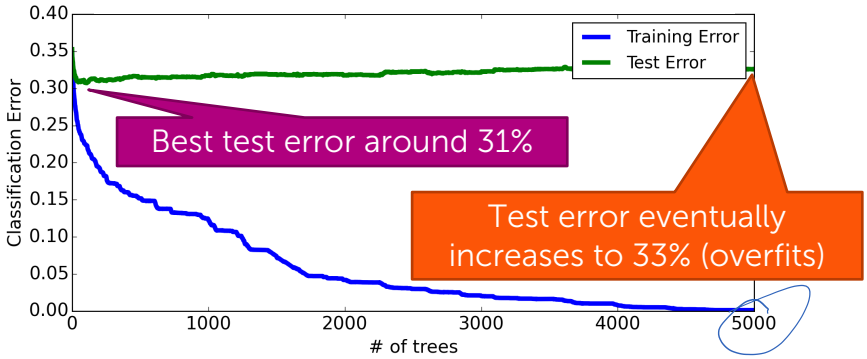


Test set performance about constant for many iterations
➔ Less sensitive to choice of T

any of these values of T would be fine

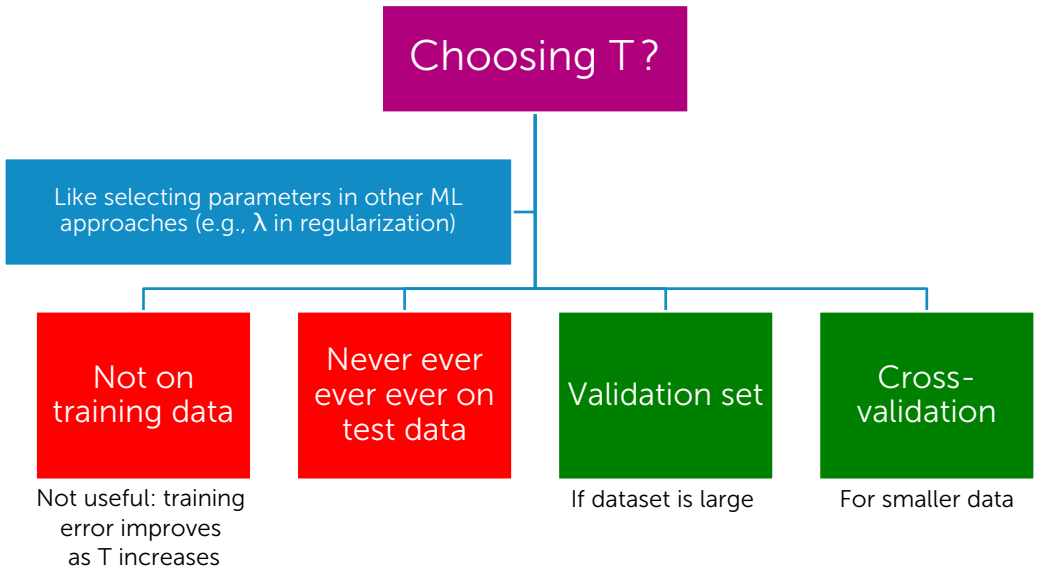
56

But boosting will eventually overfit, so must choose max number of components T



57

How do we decide when to stop boosting?



58

AdaBoost vs logistic regression

59

What is logistic regression minimizing?

Logistic regression assumes:

$$P(y = +1 \mid x) = \frac{1}{1 + \exp(-\text{score}(x_i))}$$

$\text{score}(x_i) = \sum_j w_j h_j(x_i)$

And tries to maximize data likelihood:

$$P(\mathcal{D}|\mathbf{w}, \mathbf{x}) = \prod_{i=1}^{N} \frac{1}{1 + \exp(-y_i \text{score}(x_i))}$$

$\Rightarrow \min_{\mathbf{w}} -\ln p(\mathcal{D}|\mathbf{w}, \mathbf{x})$

Equivalent to minimizing log loss

$$\sum_{i=1}^{N} \ln(1 + \exp(-y_i \text{score}(x_i)))$$

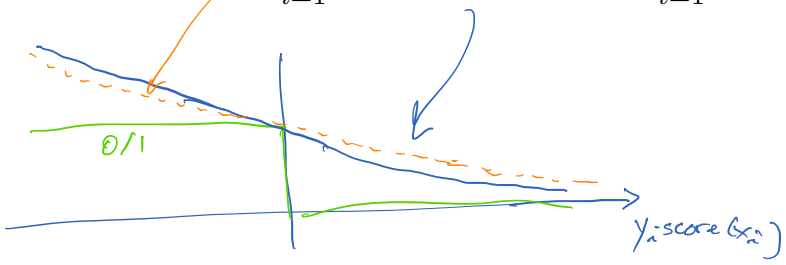
60

Logistic regression vs boosting objectives

Logistic regression minimizes: $\sum_{i=1}^{N} \ln(1+\exp(-y_i \text{score}(x_i)))$

Boosting minimizes similar loss function: $\text{score}(x) = \sum_t \hat{w}_t f_t(x)$

$$\frac{1}{N}\sum_{i=1}^{N} \exp(-y_i \text{score}(x_i)) = \prod_{t=1}^{T} Z_t$$



Both smooth approximations of 0/1 loss!

61

Logistic regression vs boosting overview

Logistic regression:

- Minimize loss fn
$$\sum_{i=1}^{N} \ln(1+\exp(-y_i \text{score}(x_i)))$$
- Define
$$\text{score}(x) = \sum_j \hat{w}_j h_j(x)$$
where features $h_j(\mathbf{x})$ are predefined
- Weights $\hat{w}_j$ learned in joint optimization

Boosting:

- Minimize loss fn
$$\sum_{i=1}^{N} \exp(-y_i \text{score}(x_i))$$
- Define
$$\text{score}(x) = \sum_t \hat{w}_t f_t(x)$$
where $f_t(\mathbf{x})$ defined dynamically to fit data
(not a linear classifier)
- Weights $\hat{w}_t$ learned incrementally

62

Summary of boosting

63

Variants of boosting and related algorithms

There are hundreds of variants of boosting, most important:

Gradient boosting

- Like AdaBoost, but useful beyond basic classification

Many other approaches to learn ensembles, most important:

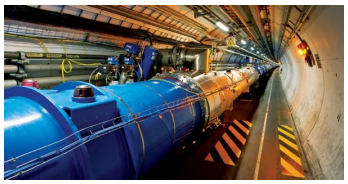
Random forests

- **Bagging**: Pick random subsets of the data
  - Learn a tree in each subset
  - Average predictions
- Simpler than boosting & easier to parallelize
- Typically higher error than boosting for same # of trees (# iterations T)

64

XGBoost [Chen & Guestrin KDD 2016]

https://github.com/dmlc/xgboost



System for fast, robust, and versatile gradient boosting

- Regularizer on tree complexity
- 2nd order approximation of loss
- Samples features at internal nodes (as in random forests)
- Handles missing values
- Out-of-core computation for large datasets for scalability
- …

De-facto tool for ML & data science

Used by more than 50% of Kaggle winning solutions

kaggle

65

Impact of boosting (*spoiler alert… HUGE IMPACT*)

Amongst most useful ML methods ever created

Extremely useful in computer vision	• Standard approach for face detection, for example
Used by **most winners** of ML competitions (Kaggle, KDD Cup,…)	• Malware classification, credit fraud detection, ads click through rate estimation, sales forecasting, ranking webpages for search, Higgs boson detection,…
Most deployed ML systems use model ensembles	• Coefficients chosen manually, with boosting, with bagging, or others

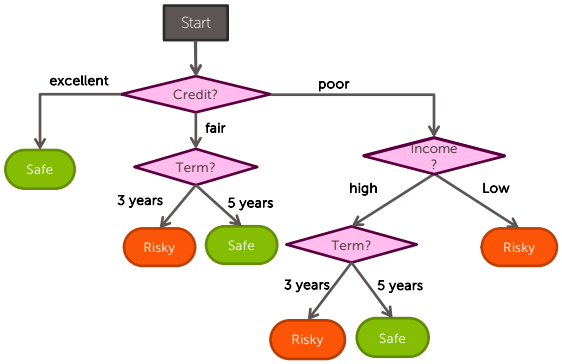
66

What you can do now…

- Identify notion ensemble classifiers
- Formalize ensembles as weighted combination of simpler classifiers
- Outline the boosting framework – sequentially learn classifiers on weighted data
- Describe the AdaBoost algorithm
  - Learn each classifier on weighted data
  - Compute coefficient of classifier
  - Recompute data weights
  - Normalize weights
- Implement AdaBoost to create an ensemble of decision stumps

67

Decision trees Wrap-up



CS 229: Machine Learning
Emily Fox
Stanford University
February 14, 2024

Slides include content developed by and co-developed with Carlos Guestrin

68

Decision trees pros & cons

Pros
- Easy to interpret
- Easily handle mixed discrete/continuous inputs
- Insensitive to monotone transformations of inputs (don't need to standardize data)
- Performs automatic variable selection
- Fairly robust to outliers
- Fast to fit (scales to large datasets)
- Handles missing input features straightforwardly

Cons
- Do not predict accurately
- Unstable (small changes to input → large differences in tree structure)

69

Boosted trees

Pros
- Targets improving predictive performance
- Maintains **low variance** of component weak learners (shallow trees) while **reducing bias** (ensemble is more complex)
- Robust to overfitting
- Can handle many differentiable loss functions via gradient boosting
- Efficient implementations (e.g., XGBoost)

Cons
- Sequential/adaptive algorithm
- Loss in interpretability

70

Random forests

Pros

- Targets improving predictive performance
- Maintains **low bias** of component (overfit) trees while **reducing variance**
- Trees can be trained in parallel
- Very few tuning parameters that tend to matter much in practice

Cons

- Loss in interpretability
- Does not perform as well in general as boosting

71

Gradient Boosting

CS 229: Machine Learning
Emily Fox
Stanford University
February 14, 2024

72

Gradient boosting

73

So far for boosting we have focused on…

- **Task:** Binary classification
- **Loss:** Exponential loss

} AdaBoost

- What about other loss functions (and tasks)?

L: sq. error → least squares boosting (regression)
log loss → logit boost

74

What about more general loss functions?

- Instead of deriving boosting algorithms for every loss function, derive **generic version** (for differentiable loss)
- **Gradient boosting**

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

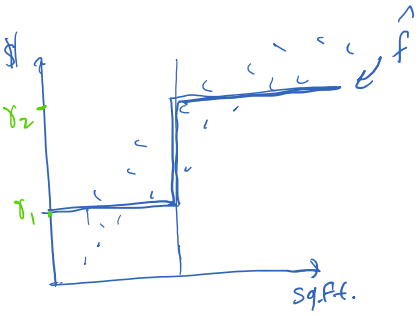
Table 10.2 from ESL

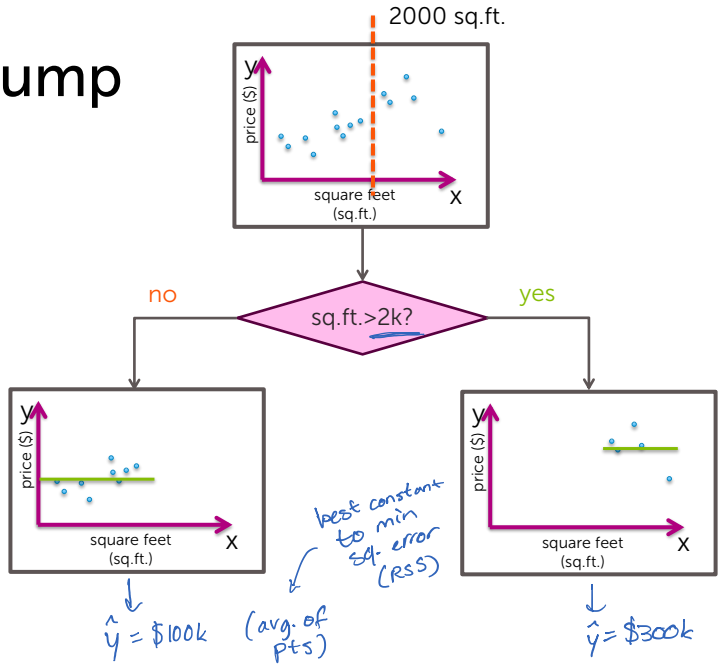
Name	Loss	$-\partial \ell(y_i, f(\boldsymbol{x}_i))/\partial f(\boldsymbol{x}_i)$
Squared error	$\frac{1}{2}(y_i - f(\boldsymbol{x}_i))^2$	$y_i - f(\boldsymbol{x}_i)$
Absolute error	$\lvert y_i - f(\boldsymbol{x}_i)\rvert$	$\text{sgn}(y_i - f(\boldsymbol{x}_i))$
Exponential loss	$\exp(-\tilde{y}_i f(\boldsymbol{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\boldsymbol{x}_i))$
Binary Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$
Multiclass logloss	$-\sum_c y_{ic} \log \pi_{ic}$	$y_{ic} - \pi_{ic}$

Table 18.1 from Murphy Book 1

Aside: Regression trees visualized

Regression stump

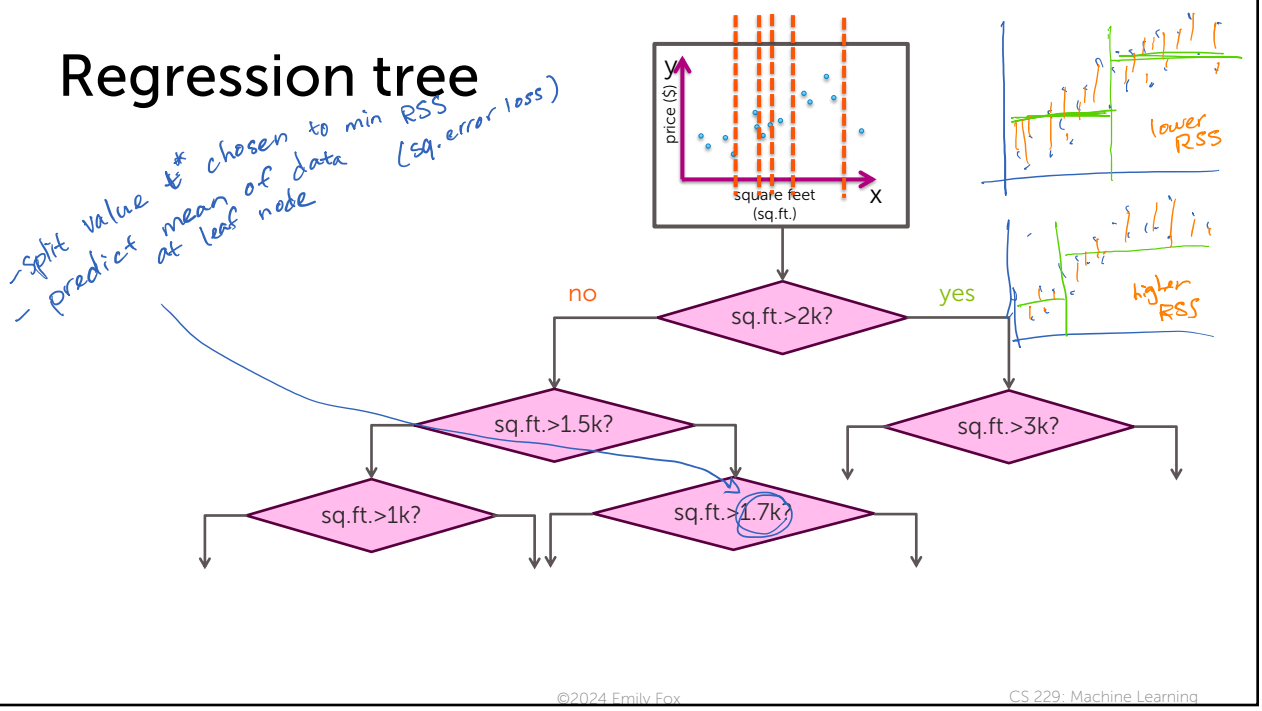




77

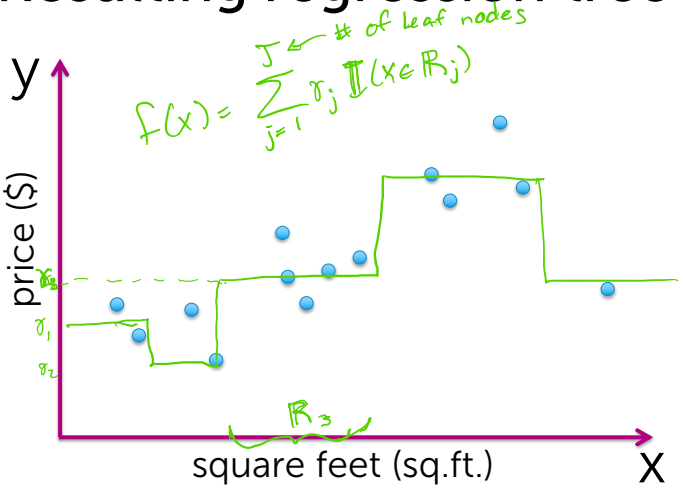
Regression tree

- split value * chosen to min RSS
- predict mean of data at leaf node (sq. error loss)

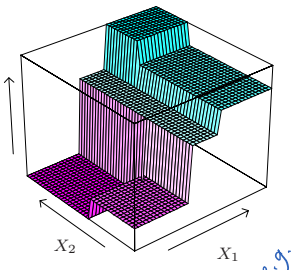


78

Resulting regression tree fit



In higher dimensions (here, 2D)



ESL Fig. 9.2

Formally: $\{\hat{\gamma}_j, \hat{R}_j\} = \arg\min_{\{\gamma_j, R_j\}} \sum_{j=1}^{J} \sum_{x_i \in R_j} L(y_i, \gamma_j)$ (e.g. sq. error)

- greedy, top-down partitioning to find $\hat{R}_j$ (e.g. min RSS)
- given $\hat{R}_j$, then $\hat{\gamma}_j$ is easy (e.g. mean)

79

Back to gradient boosting...
AdaBoost as forward stagewise additive modeling

80

Forward stagewise additive modeling

- Aim to fit $F_T(\mathbf{x};\boldsymbol{\theta}) = \sum_{t=1}^{T} w_t f(\mathbf{x};\theta_t)$ by minimizing

  $\sum_{i=1}^{N} L(y_i, F_T(\mathbf{x_i};\boldsymbol{\theta}))$

- Do so by sequentially optimizing the objective:

  $(w_t, \theta_t) = \arg\min_{w,\theta} \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i) + w f(x_i;\theta))$

- Set

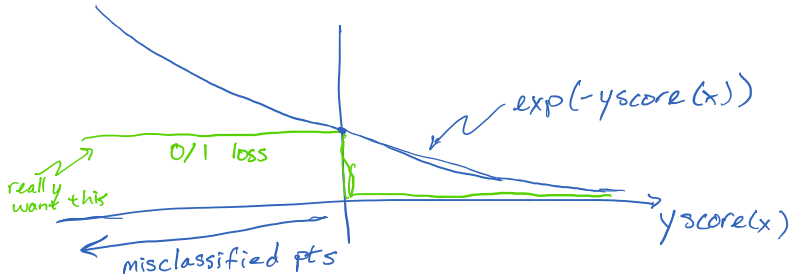
  $F_t(x) = F_{t-1}(x) + w_t f(x;\theta_t)$

81

AdaBoost is performing forward stagewise additive modeling

- **Loss:** L(y,score(**x**)) = exp(-y score(**x**))

$$\text{score}(x) = \sum_t \hat{w}_t f_t(x);\ F(x) = sign(\text{score}(x))$$



82

AdaBoost is performing forward stagewise additive modeling

- Step of forward stagewise additive modeling:

$$(f_t, \hat{w}_t) = \arg\min_{f,w} \sum_{i=1}^{N} e^{-y_i(F_{t-1}(\mathbf{x}_i)+wf(\mathbf{x}_i))} = \sum_{i=1}^{N} \alpha_{i,t} e^{-wy_i f(\mathbf{x}_i)}$$

exp. loss; fixed; F_t; argmin; $e^{-y_i F_{t-1}(x_i)}$ const. wrt w, f

(1) $F_t = F_{t-1} + \hat{w}_t f_t \Rightarrow \alpha_{i,t+1} = e^{-y_i(F_{t-1}+\hat{w}_t f_t)} = \alpha_{i,t+1} = \alpha_{i,t} e^{-y_i \hat{w}_t f_t(x_i)}$ ✓

(2) (Assume $w>0$), If $y_i \neq f(x_i)$, $e^{-wy_i f(x_i)}$ is larger than had $y_i = f(x_i)$

$\Rightarrow f_t = \arg\min_f \sum_{i=1}^{N} \alpha_{i,t} \mathbb{I}(y_i \neq f(x_i))$ ← $\mathbb{I}(A) = \begin{cases} 1 & \text{if } A \text{ true} \\ 0 & \text{o/w} \end{cases}$

min weighted class. error ✓

(3) Plug in f_t, take deriv., set $= 0$ ✓

$\Rightarrow \hat{w}_t = \frac{1}{2} \log\left(\frac{1-\epsilon_t}{\epsilon_t}\right)$ ϵ_t = weighted class. error of f_t

Forward stagewise additive modeling

- Aim to fit $F_T(\mathbf{x};\boldsymbol{\theta}) = \sum_{t=1}^{T} w_t f(\mathbf{x};\theta_t)$ by minimizing

T; weight; params

e.g. neural networks, wavelets, splines

empirical loss $\sum_{i=1}^{N} L(y_i, F_T(\mathbf{x}_i;\boldsymbol{\theta}))$

If sq. error $\frac{1}{2}\sum_{i=1}^{N}(y_i - F_{t-1}(x_i) - wf)^2 = \frac{1}{2}\sum_{i=1}^{N}(r_{it} - wf)^2$; residuals $y_i - F_{t-1}(x_i)$; e.g., tree $T(x_i)$

- Do so by sequentially optimizing the objective:

$$(w_t, \theta_t) = \arg\min_{w,\theta} \sum_{i=1}^{N} L(y_i, F_{t-1}(x_i) + wf(x_i;\theta))$$

don't change what we've learned so far

- Set

$$F_t(x) = F_{t-1}(x) + w_t f(x;\theta_t)$$

"least sq. boosting"

fit e.g. reg tree to min RSS of residuals at t

Gradient boosting

What about more general loss functions?

- Instead of deriving boosting algorithms for every loss function, derive **generic version** (for differentiable loss)
- **Gradient boosting**

Setting	Loss Function	$-\partial L(y_i, f(x_i))/\partial f(x_i)$
Regression	$\frac{1}{2}[y_i - f(x_i)]^2$	$y_i - f(x_i)$
Regression	$\lvert y_i - f(x_i)\rvert$	$\text{sign}[y_i - f(x_i)]$
Regression	Huber	$y_i - f(x_i)$ for $\lvert y_i - f(x_i)\rvert \le \delta_m$ $\delta_m \text{sign}[y_i - f(x_i)]$ for $\lvert y_i - f(x_i)\rvert > \delta_m$ where $\delta_m = \alpha$th-quantile$\{\lvert y_i - f(x_i)\rvert\}$
Classification	Deviance	kth component: $I(y_i = \mathcal{G}_k) - p_k(x_i)$

Table 10.2 from ESL

Name	Loss	$-\partial \ell(y_i, f(\boldsymbol{x}_i))/\partial f(\boldsymbol{x}_i)$
Squared error	$\frac{1}{2}(y_i - f(\boldsymbol{x}_i))^2$	$y_i - f(\boldsymbol{x}_i)$
Absolute error	$\lvert y_i - f(\boldsymbol{x}_i)\rvert$	$\text{sgn}(y_i - f(\boldsymbol{x}_i))$
Exponential loss	$\exp(-\tilde{y}_i f(\boldsymbol{x}_i))$	$-\tilde{y}_i \exp(-\tilde{y}_i f(\boldsymbol{x}_i))$
Binary Logloss	$\log(1 + e^{-\tilde{y}_i f_i})$	$y_i - \pi_i$
Multiclass logloss	$-\sum_c y_{ic} \log \pi_{ic}$	$y_{ic} - \pi_{ic}$

Table 18.1 from Murphy Book 1

Gradient-based optimization of loss in function space

- Consider generic objective

$$\hat{f} = \arg\min_f L(f)$$

- Since f (function) is infinite-dimensional, simplify by optimizing only over function evaluated at training points, $\mathbf{f} = (f(\mathbf{x}_1), ..., f(\mathbf{x}_N))'$

where

$$\mathbf{f}_t = \mathbf{f}_{t-1} - \eta_t \mathbf{g}_t$$

grad. descent

η_t step size: $\arg\min_\eta L(f_{t-1} - \eta g_t)$ line search

$$g_{it} = \left[\frac{\partial L(y_i, f(x_i))}{\partial f(x_i)}\right]_{f(x_i)=f_{t-1}(x_i)}$$

gradient of $L(f)$ evaluated at $f = f_{t-1}$ ← prev. f

87

Gradient boosting

- How to generalize beyond the training points?
- Fit weak learner to approx. negative gradient at training points

$$f_t = \arg\min_f \sum_{i=1}^{N} (-g_{it} - f(\mathbf{x}_i))^2$$

treat as "data" we are trying to fit

weak learner

sq. error

88

Gradient boosting algorithm

init ensemble with single term $F_0(\mathbf{x})=\text{argmin}_f \sum_{i=1}^{N} L(y_i, f(\mathbf{x}_i))$

for t=1,…,T

compute gradient terms $g_{it} = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i)=F_{t-1}(\mathbf{x}_i)}$

fit weak learner to negative gradient

$$f_t = \arg\min_f \sum_{i=1}^{N} (-g_{it} - f(\mathbf{x}_i))^2$$

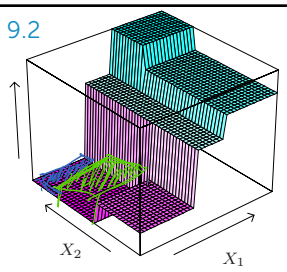
update ensemble $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \eta_t f_t(\mathbf{x})$

Return $F_T(\mathbf{x})$

89

Gradient boosted trees

ESL Fig. 9.2



init ensemble with single tree $F_0(\mathbf{x})$

for t=1,…,T

compute gradient terms $g_{it} = \left[\frac{\partial L(y_i, f(\mathbf{x}_i))}{\partial f(\mathbf{x}_i)}\right]_{f(\mathbf{x}_i)=F_{t-1}(\mathbf{x}_i)}$

fit regression tree to "pseudo" residuals $-g_{it}$ as targets

at leaves, predict $\gamma_{jt} = \arg\min_{\gamma} \sum_{\mathbf{x}_i \in R_{jt}} L(y_i, F_{t-1}(\mathbf{x}_i) + \gamma)$ $j=1,\dots,J_t$

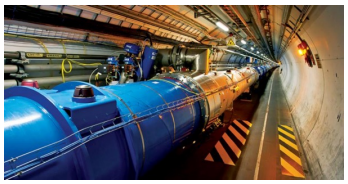
update ensemble $F_t(\mathbf{x}) = F_{t-1}(\mathbf{x}) + \sum_{j=1}^{J_t} \gamma_{jt} \mathbb{I}(\mathbf{x} \in R_{jt})$

Return $F_T(\mathbf{x})$

90

XGBoost [Chen & Guestrin KDD 2016]

https://github.com/dmlc/xgboost



System for fast, robust, and versatile gradient boosting

- Regularizer on tree complexity
- 2nd order approximation of loss
- Samples features at internal nodes (as in random forests)
- Handles missing values
- Out-of-core computation for large datasets for scalability
- …

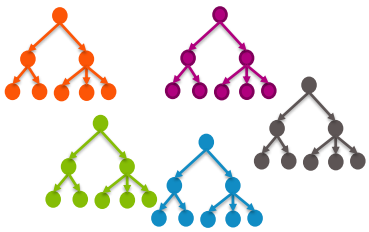
De-facto tool for ML & data science

Used by more than 50% of Kaggle winning solutions

kaggle

91

Random Forests

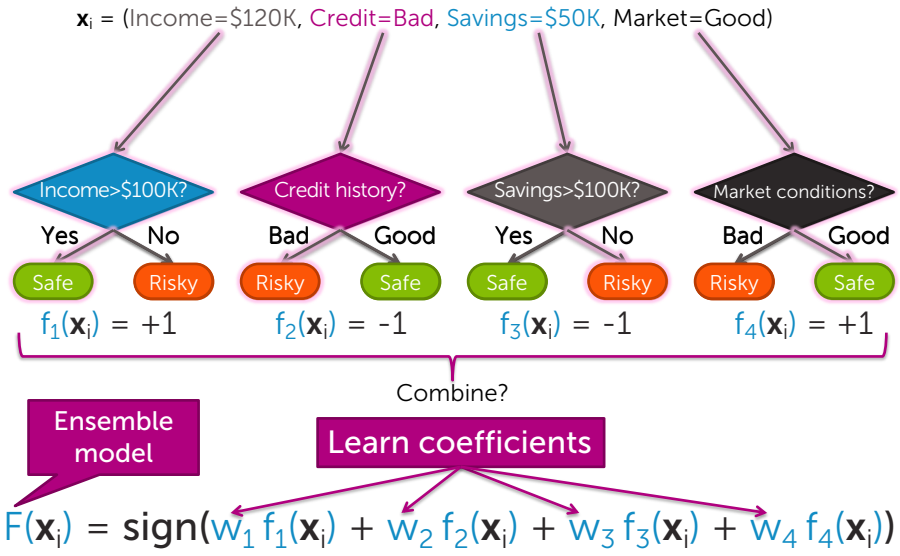


CS 229: Machine Learning
Emily Fox
Stanford University
February 14, 2024

OPTIONAL

92

Ensemble methods: Each classifier "votes" on prediction



93

Boosting question

"Can a set of weak learners be combined to create a stronger learner?" *Kearns and Valiant (1988)*

↓

Yes! *Schapire (1990)*

↓

Boosting

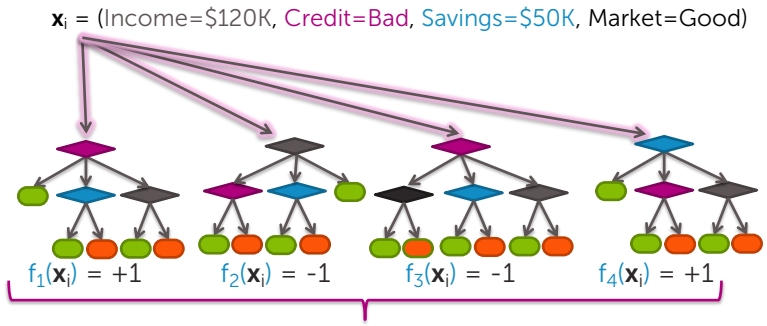
↓

Amazing impact: • simple approach • widely used in industry • wins most Kaggle competitions

94

Another option…

- Instead of **boosting** performance of low-variance decision trees (e.g., *decision stumps*), what if we reduce the variance of low-bias (*complex*) trees?
- **How?**

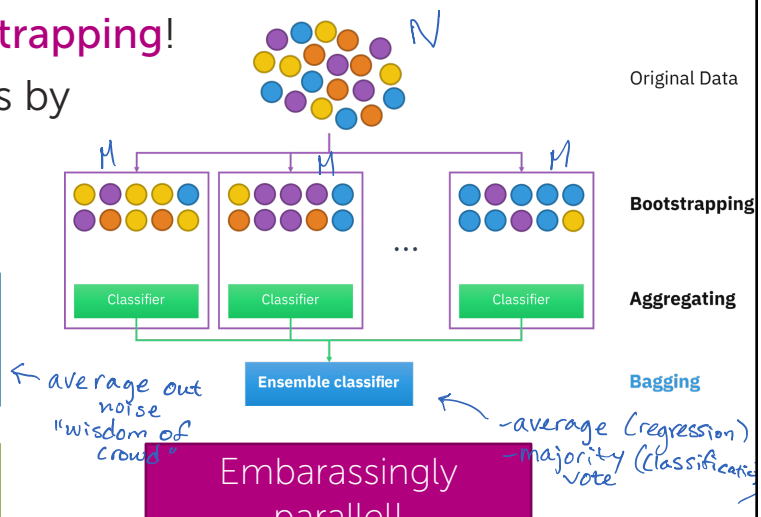


$$F(x_i) = \text{sign}(w_1 f_1(x_i) + w_2 f_2(x_i) + w_3 f_3(x_i) + w_4 f_4(x_i))$$

95

Bootstrap aggregating (Bagging)

- Solve this with **bootstrapping**!
- Create many datasets by randomly sampling with replacement



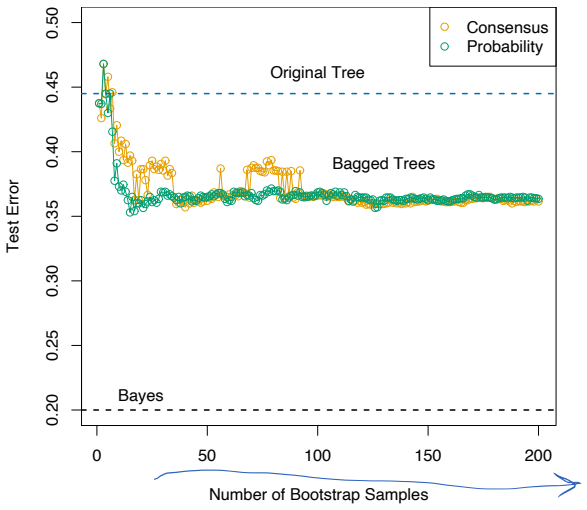
Reduces variance leaving bias unchanged
(under sq. error loss)

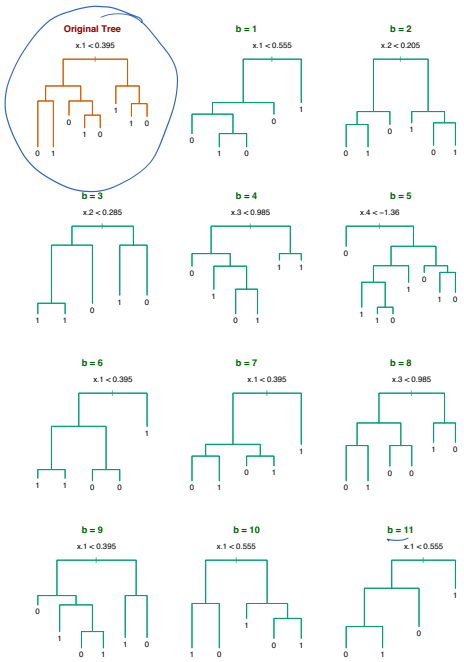
Helpful for unstable learning procedures

Embarassingly parallel!

96

Bagging on simulated data: ESL Example 8.7.1





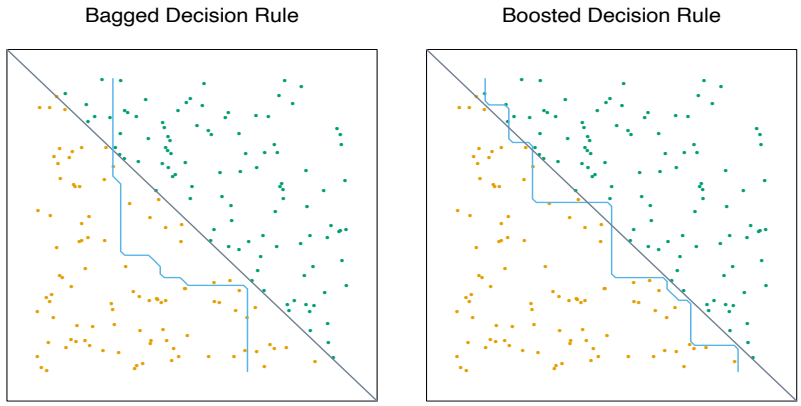
97

Notes on bagging

- For regression tasks using squared-error loss, can show that bagging reduces variance (due to averaging) while leaving bias unchanged
- For classification tasks using 0/1 loss, bagging a bad classifier can make it worse
- Works particularly well for low-bias, high-variance models like trees
- Bagged trees are no longer a tree → loss in interpretability

98

Comparing bagging and boosting: ESL Figure 8.12



Boosting tends to dominate bagging

99

Random forests

100

Improving variance reduction

- Consider average of T i.i.d. random variables, each with variance σ^2 $X_1, \dots, X_T$, $\text{Var}(X_i) = \sigma^2$

$$\text{Var}\left(\frac{1}{T}\sum_{t=1}^{T} X_t\right) = \frac{1}{T}\sigma^2$$

- If i.d., but not independent, and have correlation ρ

$$\text{Var}\left(\frac{1}{T}\sum_{t=1}^{T} X_t\right) = \rho\sigma^2 + \frac{1-\rho}{T}\sigma^2$$ (see ESL exercise 15.1)

- Increasing T→ ∞ … $\text{Var}(\text{avg}) \to \rho\sigma^2$ can't be reduced by increasing T

Random forests seek to **reduce correlation** between trees

Improving variance reduction

If a few features are very predictive of outcome, will be selected in many trees leading to increased correlation

Random forests seek to **reduce correlation** between trees by randomizing features available at each step

Random forest algorithm

For T=1,…,many! (1000+)

- Randomly select a set of N samples (rows)

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22	1	0	7.25	S
1	1	female	38	1	0	71.2833	C
1	3	female	26	0	0	7.925	S
1	1	female	35	1	0	53.1	S
0	3	male	35	0	0	8.05	S
0	3	male		0	0	8.4583	Q
0	1	male	54	0	0	51.8625	S
0	3	male	2	3	1	21.075	S
1	3	female	27	0	2	11.1333	S
1	2	female	14	1	0	30.0708	C

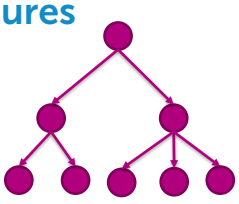
Kaggle Titanic dataset

103

Random forest algorithm

For T=1,…,many! (1000+)

- Randomly select a set of N samples (rows)
- Build a deep tree (overfit)
  - At each considered split, randomly sample **d candidate features**



Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22	1	0	7.25	S
1	1	female	38	1	0	71.2833	C
1	3	female	26	0	0	7.925	S
1	1	female	35	1	0	53.1	S
0	3	male	35	0	0	8.05	S
0	3	male		0	0	8.4583	Q
0	1	male	54	0	0	51.8625	S
0	3	male	2	3	1	21.075	S
1	3	female	27	0	2	11.1333	S
1	2	female	14	1	0	30.0708	C

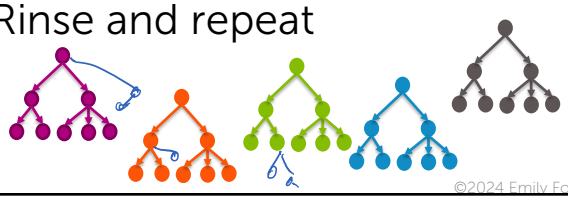
Kaggle Titanic dataset

104

Random forest algorithm

For T=1,…,many! (1000+)

- Randomly select a set of N samples (rows)
- Build a deep tree (overfit)
  - At each considered split, randomly sample **d candidate features**
- Rinse and repeat



Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22	1	0	7.25	S
1	1	female	38	1	0	71.2833	C
1	3	female	26	0	0	7.925	S
1	1	female	35	1	0	53.1	S
0	3	male	35	0	0	8.05	S
0	3	male		0	0	8.4583	Q
0	1	male	54	0	0	51.8625	S
0	3	male	2	3	1	21.075	S
1	3	female	27	0	2	11.1333	S
1	2	female	14	1	0	30.0708	C

Kaggle Titanic dataset

← apologies for lazy powerpointing. really diff. structures.

105

Random forest algorithm

For T=1,…,many! (1000+)

- Randomly select a set of N samples (rows)
- Build a deep tree (overfit)
  - At each considered split, randomly sample **d candidate features**
- Rinse and repeat
- **Ensemble classification =** majority vote

Survived	Pclass	Sex	Age	SibSp	Parch	Fare	Embarked
0	3	male	22	1	0	7.25	S
1	1	female	38	1	0	71.2833	C
1	3	female	26	0	0	7.925	S
1	1	female	35	1	0	53.1	S
0	3	male	35	0	0	8.05	S
0	3	male		0	0	8.4583	Q
0	1	male	54	0	0	51.8625	S
0	3	male	2	3	1	21.075	S
1	3	female	27	0	2	11.1333	S
1	2	female	14	1	0	30.0708	C

Kaggle Titanic dataset

106

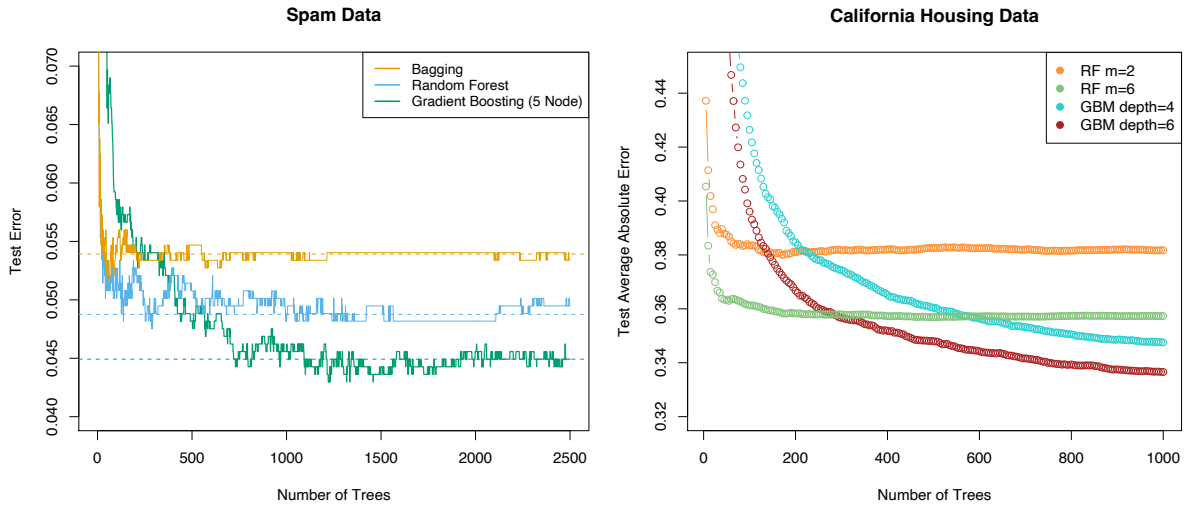
Terminating and assessing performance

Measure of validation error:

- For each $(\mathbf{x}_i, y_i)$, compute ensemble prediction (average or majority vote) on trees **not** constructed using $(\mathbf{x}_i, y_i)$
- Out-of-bag error
- Approximates K-fold cross validation
- Can sequentially add in new trees and monitor OOB error to determine when to stop

107

Comparing random forests, bagging, and boosting: ESL Figures 15.1 & 15.3



108

Extremely widely used…

Microsoft used random forests in Kinect system to identify "pose" of person from depth camera

Real-Time Human Pose Recognition in Parts from Single Depth Images

Jamie Shotton Andrew Fitzgibbon Mat Cook Toby Sharp Mark Finocchio
Richard Moore Alex Kipman Andrew Blake
Microsoft Research Cambridge & Xbox Incubation

Abstract

We propose a new method to quickly and accurately predict 3D positions of body joints from a single depth image, using no temporal information. We take an object recognition approach, designing an intermediate body parts representation that maps the difficult pose estimation problem into a simpler per-pixel classification problem. Our large and highly varied training dataset allows the classifier to estimate body parts invariant to pose, body shape, clothing, etc. *Finally we generate confidence-scored 3D proposals of several body joints by reprojecting the classification result and finding local modes.*

The system runs at 200 frames per second on consumer hardware. Our evaluation shows high accuracy on both synthetic and real test sets, and investigates the effect of several training parameters. We achieve state of the art accuracy in our comparison with related work and demonstrate improved generalization over exact whole-skeleton nearest neighbor matching.

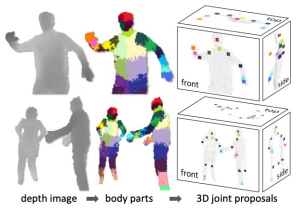


Figure 1. **Overview.** From an single input depth image, a per-pixel body part distribution is inferred. (Colors indicate the most likely part labels at each pixel, and correspond in the joint proposals). Local modes of this signal are estimated to give high-quality proposals for the 3D locations of body joints, even for multiple users.