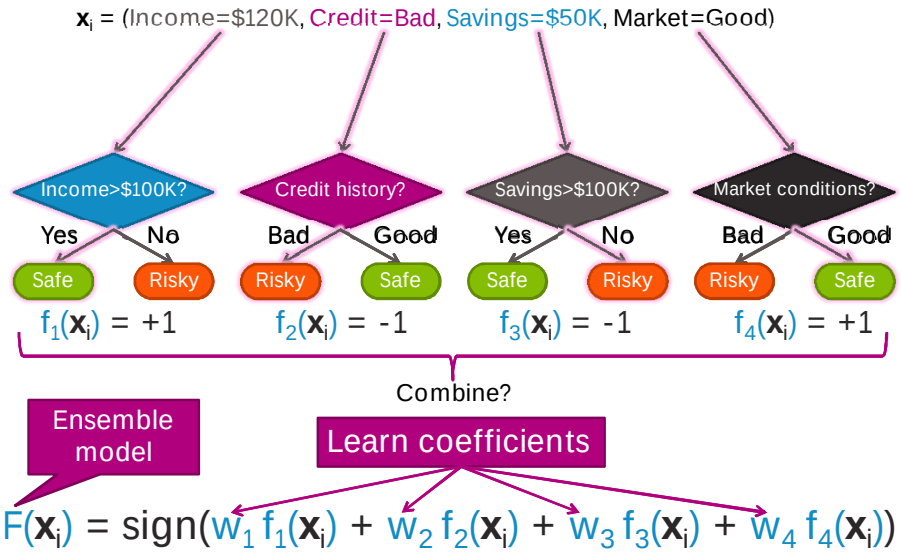
Boosting

CSE 446: Machine Learning

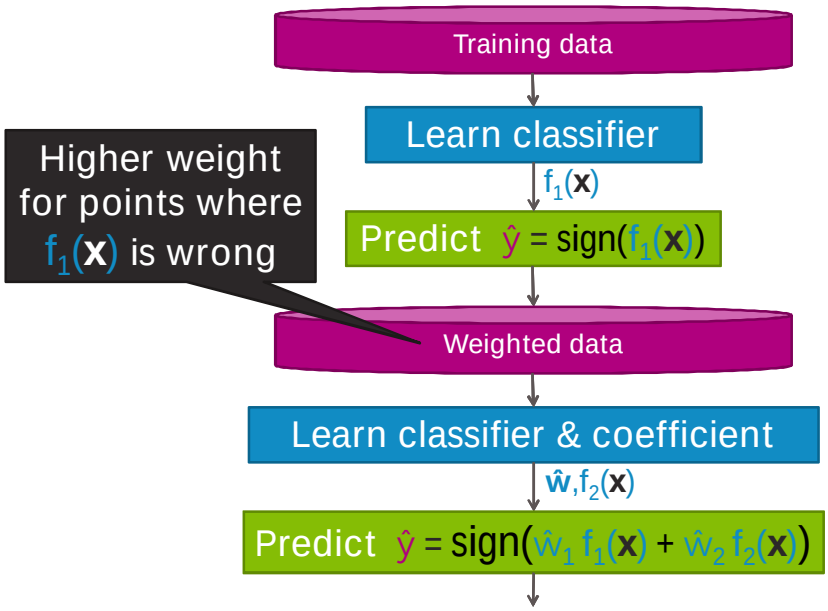
Emily Fox
University of Washington

February 1, 2017

Ensemble methods: Each classifier "votes" on prediction



Boosting = Greedy learning ensembles from data



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

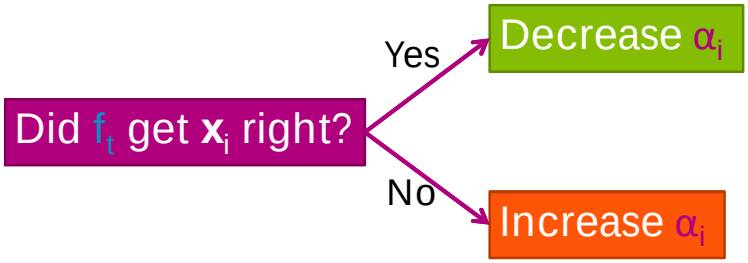
$$\hat{\mathbf{w}}_t = \frac{1}{2} \ln \left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)} \right)$$

- Final model predicts by:

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

Recompute weights α_i

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



AdaBoost: Formula for updating weights α_i

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

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	$e^{2.3} = 9.99$	increase importance of x_i, y_i
no	0	$e^{0} = 1$	no change

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:

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

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

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

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

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

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:

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

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

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

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

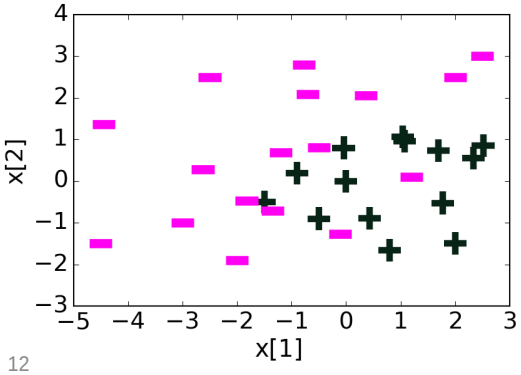
AdaBoost example:
A visualization

t=1: Just learn a classifier on original data

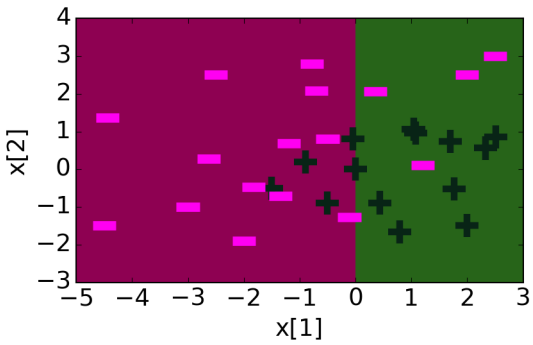
all pts have same weight

standard learning

Original data



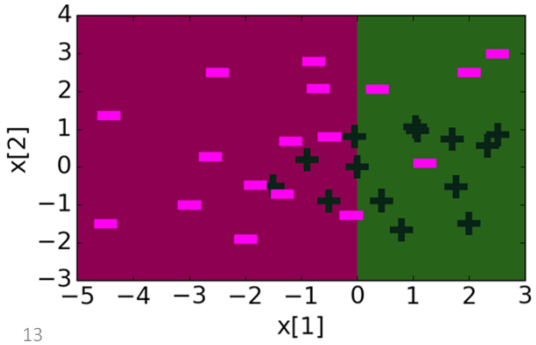
Learned decision stump $f_1(\mathbf{x})$



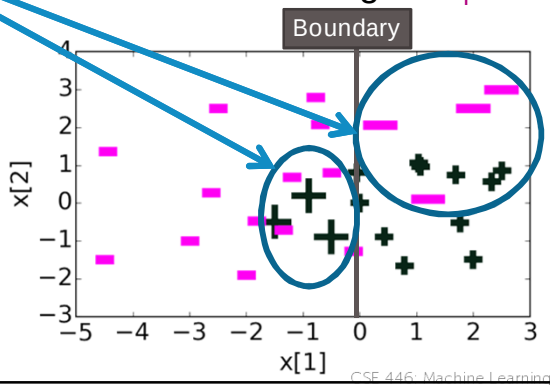
Updating weights α_i

Increase weight α_i of misclassified points

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

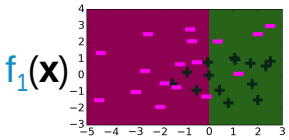


New data weights α_i

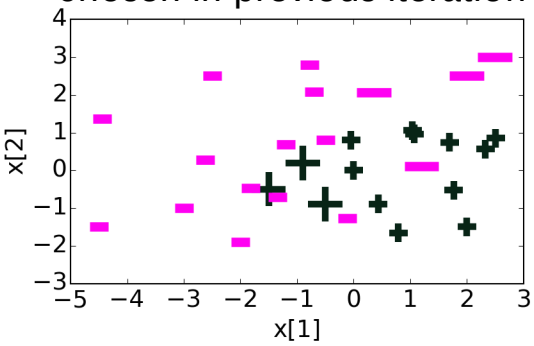


t=2: Learn classifier on weighted data

$f_1(\mathbf{x})$

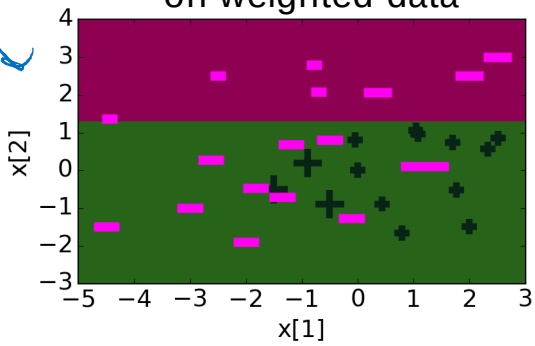


Weighted data: using α_i chosen in previous iteration



better split for these weights

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

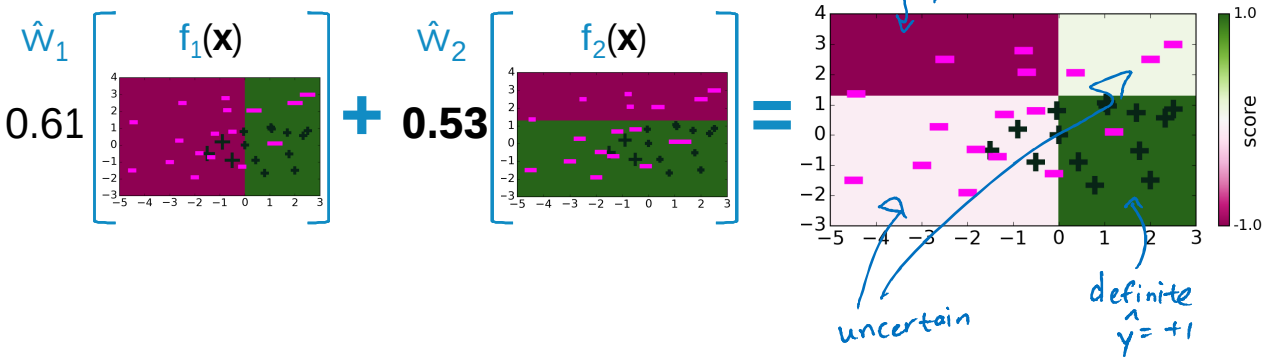


Ensemble becomes weighted sum of learned classifiers

score space

definite $\hat{y} = -1$

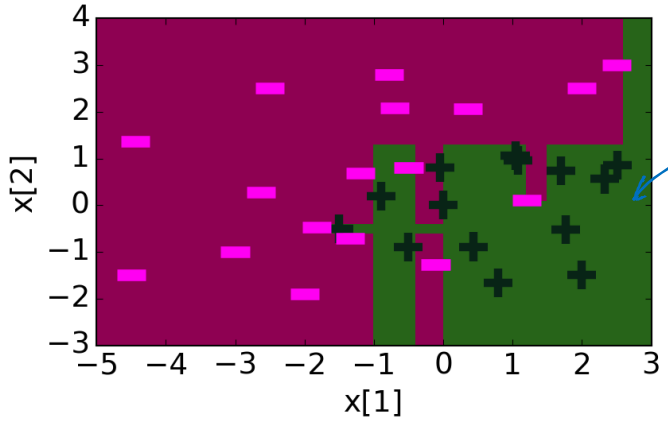
$\hat{w}_1$ 0.61 $f_1(\mathbf{x})$ + $\hat{w}_2$ **0.53** $f_2(\mathbf{x})$ =



uncertain

definite $\hat{y} = +1$

Decision boundary of ensemble classifier after 30 iterations



decision boundary is crazy!

probably overfitting

training_error = 0

AdaBoost example:
Boosted decision stumps step-by-step

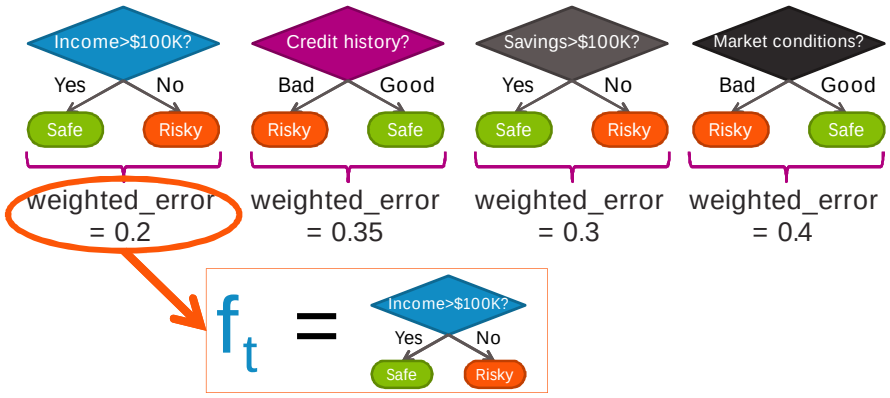
Boosted decision stumps

- 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:

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

Finding best next decision stump $f_t(\mathbf{x})$

Consider splitting on each feature:



$$\hat{W}_t = \frac{1}{2}\ln\left(\frac{1 - weighted\_error(f_t)}{weighted\_error(f_t)}\right) = 0.69$$

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:

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

Updating weights α_i

$$\alpha_i \leftarrow \begin{cases} \alpha_i e^{-\hat{W}_t} = \alpha_i e^{-0.69} = \alpha_i / 2 & \text{, if } f_t(\mathbf{x}_i) = y_i \\ \alpha_i e^{\hat{W}_t} = \alpha_i e^{0.69} = 2\,\alpha_i & \text{, if } f_t(\mathbf{x}_i) \neq y_i \end{cases}$$

Credit	Income	y	ŷ	Previous weight α	New weight α
A	$130K	Safe	Safe	0.5	0.5/2 = 0.25
B	$80K	Risky	Risky	1.5	0.75
C	$110K	Risky	Safe	1.5	2 * 1.5 = 3
A	$110K	Safe	Safe	2	1
A	$90K	Safe	Risky	1	2
B	$120K	Safe	Safe	2.5	1.25
C	$30K	Risky	Risky	3	1.5
C	$60K	Risky	Risky	2	1
B	$95K	Safe	Risky	0.5	1
A	$60K	Safe	Risky	1	2
A	$98K	Safe	Risky	0.5	1

Boosting convergence & overfitting

Boosting question revisited

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

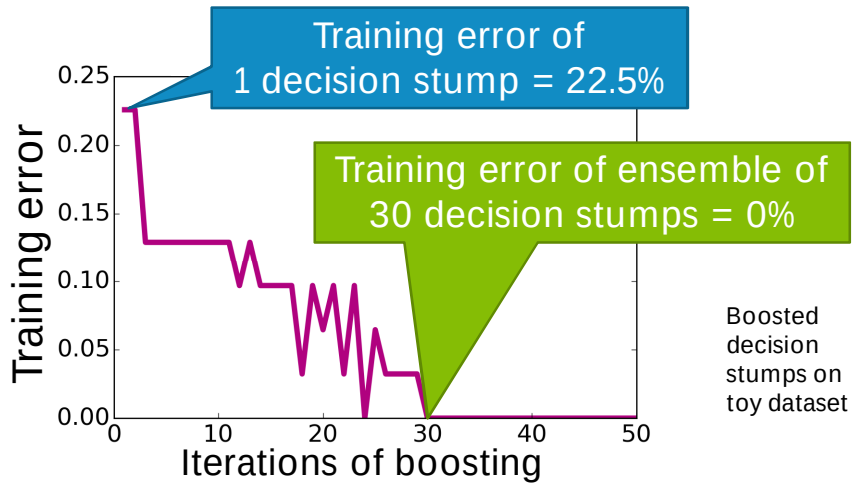
↓

Yes! Schapire (1990)

↓

Boosting

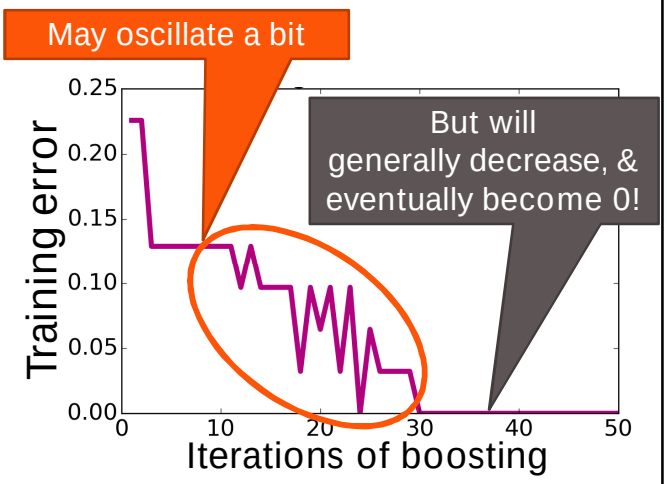
After some iterations, training error of boosting goes to zero!!!



AdaBoost Theorem

Under some technical conditions…

Training error of boosted classifier → 0 as T→∞



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

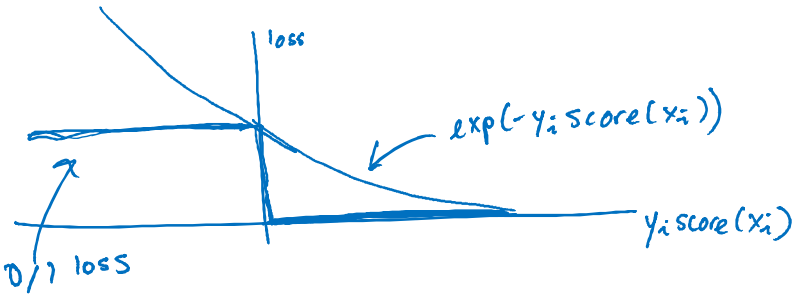
AdaBoost Theorem more formally

Training error of final classifier is bounded by:

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

0/1 loss

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



smooth upper bound on 0/1 loss

"surrogate loss"

AdaBoost Theorem more formally

Training error of final classifier is bounded by:

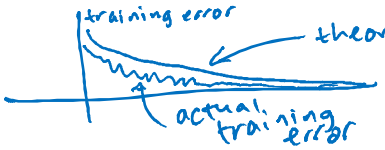
$$Z_t = \sum_{i=1}^{N} D_t(i)\exp(-\hat{w}_t y_i f_t(x_i))$$

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

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

Pf: HW "... "telescopic sums"

If $Z_t < 1\ \forall t \Rightarrow$ as $T \to \infty$, training error $\to 0$



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:

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

take der. + set = 0

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

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

weighted class. error

AdaBoost Theorem more formally

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

$$\epsilon_t < 0.5$$

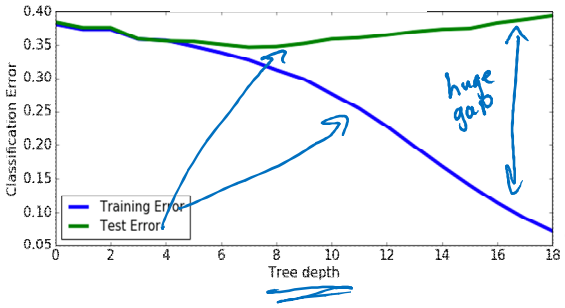
AdaBoost will achieve zero training error (exponentially fast):

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

upper bound for $\epsilon_t < 1/2$

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

Decision trees on loan data

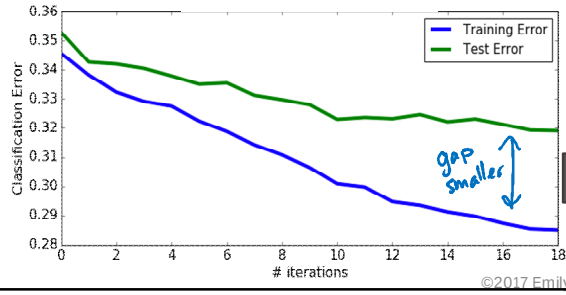


39% test error

Overfitting

8% training error

Boosted decision stumps on loan data



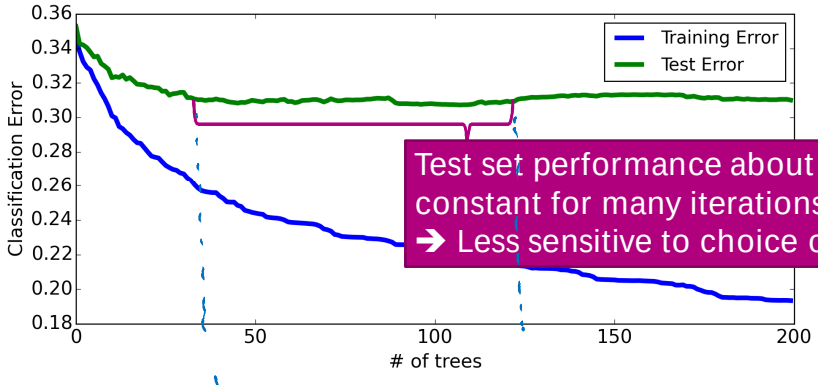
32% test error

better

Better fit & lower test error

28.5% training error

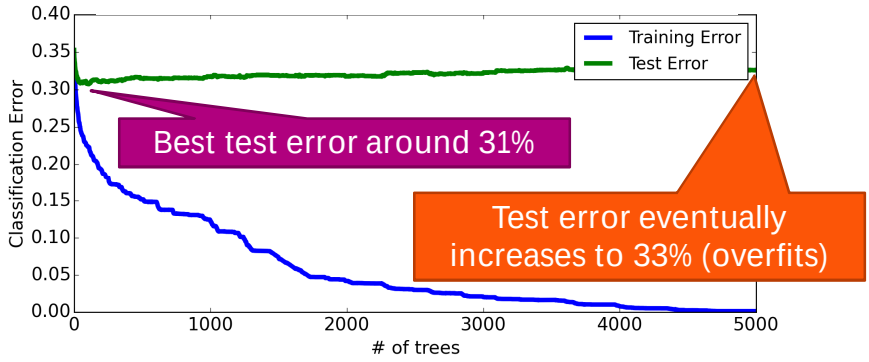
Boosting tends to be robust to overfitting



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

any of these values of T would be fine

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



How do we decide when to stop boosting?

Choosing T ?

Like selecting parameters in other ML approaches (e.g., λ in regularization)

- **Not on training data** — Not useful: training error improves as T increases
- **Never ever ever ever on test data**
- **Validation set** — If dataset is large
- **Cross-validation** — For smaller data

AdaBoost vs logistic regression

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:

training data

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

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

Equivalent to minimizing log loss

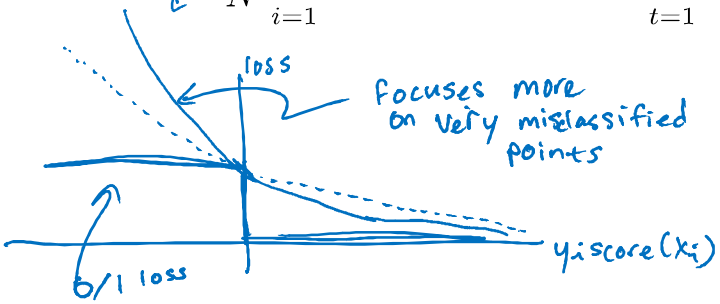
$$\sum_{i=1}^{N} \ln(1 + \exp(-y_i \text{score}(x_i)))$$

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!

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

Summary of boosting

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)

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

What you can do now…

- Identify notion ensemble classifiers
- Formalize ensembles as the 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
- Discuss convergence properties of AdaBoost & how to pick the maximum number of iterations T
- Derive why AdaBoost leads to zero training error exponentially fast
- Compare and contrast what AdaBoost and logistic regression are minimizing

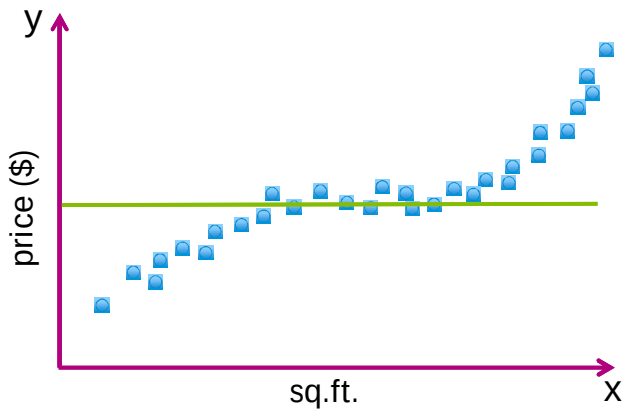
Instance-Based Learning:

Nearest neighbor and kernel regression and classificiation

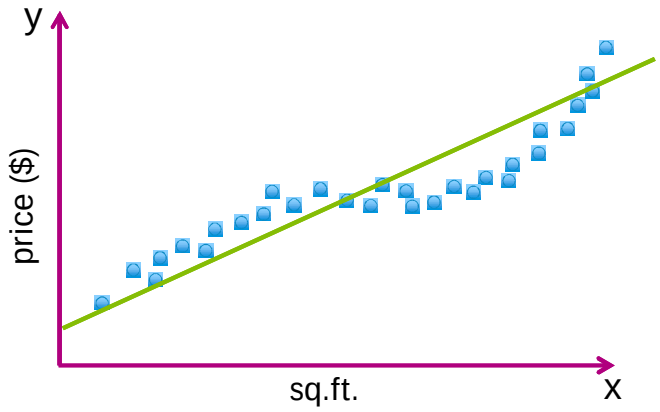
CSE 446: Machine Learning
Emily Fox
University of Washington
February 1, 2017

Fit globally vs. fit locally

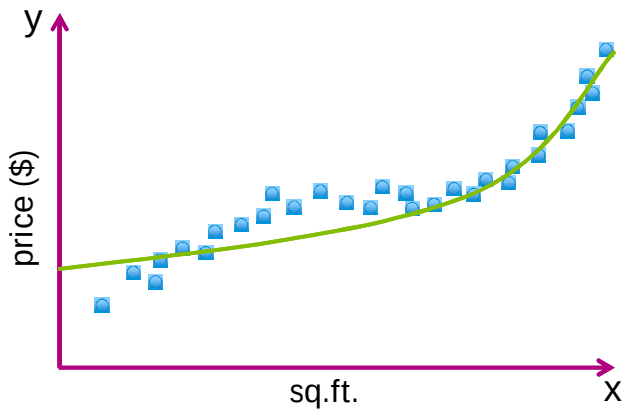
Parametric models of f(x)



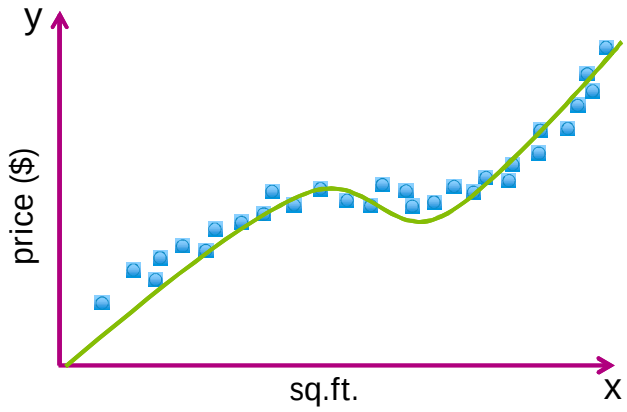
Parametric models of f(x)



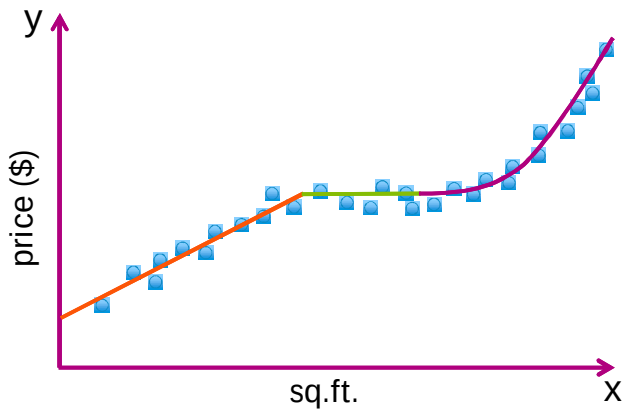
Parametric models of f(x)



Parametric models of f(x)



f(x) is not really a polynomial



What alternative do we have?

If we:

- Want to allow flexibility in f(**x**) having local structure
- Don't want to infer "structural breaks"

What's a simple option we have?

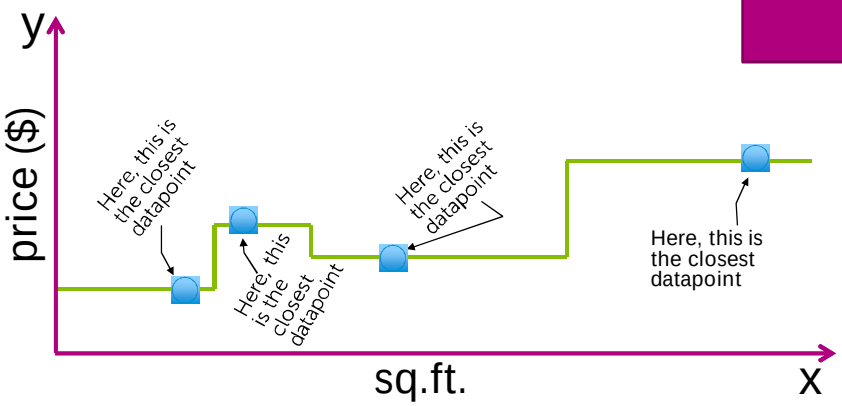
- Assuming we have plenty of data…

Simplest approach: Nearest neighbor regression

Fit locally to each data point

Predicted value = "closest" y_i

1 nearest neighbor (1-NN) regression



What people do naturally…

Real estate agent assesses value by finding sale of most similar house

1-NN regression more formally

Dataset of (🏠,$) pairs: $(\mathbf{x}_1,y_1)$, $(\mathbf{x}_2,y_2)$,…,$(\mathbf{x}_N,y_N)$

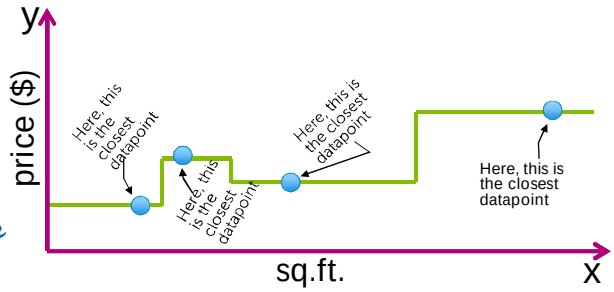
Query point: $\mathbf{x}_q$  big lime green house $?

1. Find "closest" $\mathbf{x}_i$ in dataset

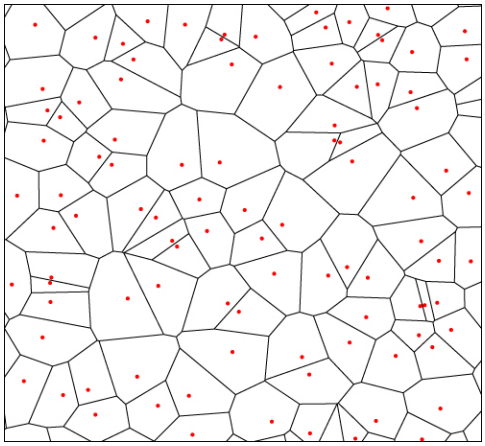
$x_{NN} \leftarrow \min_i \text{distance}(x_i, x_q)$ big pink house

2. Predict

$\hat{y}_q = y_{NN}$ sales price of big pink house



Visualizing 1-NN in multiple dimensions



Voronoi tesselation (or diagram):

- Divide space into N regions, each containing 1 datapoint
- Defined such that any **x** in region is "closest" to region's datapoint

Don't explicitly form!

Distance metrics: Defining notion of "closest"

In 1D, just Euclidean distance:

$$\text{distance}(x_j, x_q) = |x_j - x_q|$$

In multiple dimensions:

- can define many interesting distance functions
- most straightforwardly, might want to weight different dimensions differently

Weighting housing inputs

Some inputs are more relevant than others

 # bedrooms
 # bathrooms
sq.ft. living
sq.ft. lot
floors
year built
year renovated
waterfront

Scaled Euclidean distance

Formally, this is achieved via

$$\text{distance}(\mathbf{x}_j, \mathbf{x}_q) = \sqrt{a_1(\mathbf{x}_j[1]-\mathbf{x}_q[1])^2 + \ldots + a_d(\mathbf{x}_j[d]-\mathbf{x}_q[d])^2}$$

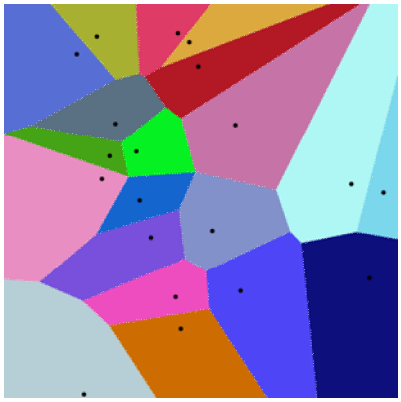
weight on each input
(defining relative importance)

Other example distance metrics:

- Mahalanobis, rank-based, correlation-based, cosine similarity, Manhattan, Hamming, …

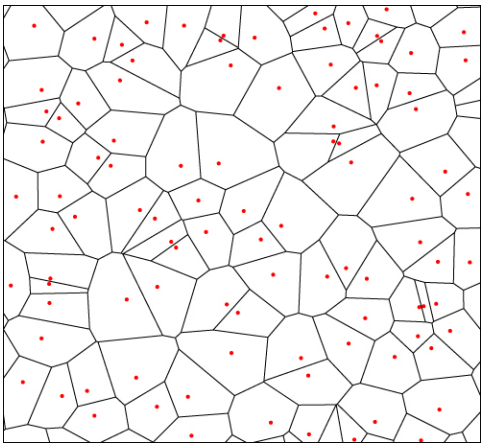
Different distance metrics lead to different predictive surfaces

Euclidean distance



Manhattan distance

Can 1-NN be used for classification?



Yes!!

Just predict class of neighbor