Boosting

CSE 575: Statistical Machine Learning

Jingrui He

CIDSE, ASU

Boosting

Fighting the bias-variance tradeoff

- **Simple (a.k.a. weak) learners are good**
  - e.g., naïve Bayes, logistic regression, decision stumps (or shallow decision trees) Not Covered
  - Low variance, don't usually overfit
- **Simple (a.k.a. weak) learners are bad**
  - High bias, can't solve hard learning problems

- Can we make weak learners always good???
  - **No!!!**
  - **But often yes…**

Neural networks are Ensemble model of perceptrons
But not Boost

Voting (Ensemble Methods)

- Instead of learning a single (weak) classifier, learn **many weak classifiers** that are **good at different parts of the input space**

Team

Specialists

- **Output class:** (Weighted) vote of each classifier
  - Classifiers that are most "sure" will vote with more conviction
  - Classifiers will be most "sure" about a particular part of the space
  - On average, do better than single classifier!

Esemble methods create a team of specialist
models

- **But how do you ???**
  - force classifiers to learn about different parts of the input space?
  - weigh the votes of different classifiers?

Boosting: [Schapire, 1989]

- Idea: given a weak learner, run it multiple times on (reweighted) training data, then let learned classifiers vote

- On each iteration *t*: high error => higher weight
  - Weigh each training example by how incorrectly it was classified
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t – weight of the model

- Final classifier: $T :=$ # of iterations

$$F(x) = \sum_{t=1}^{T} \alpha_t h_t(x) \qquad H(x) = \text{sign}(F(x))$$

$> 0 \Rightarrow +1$

$< 0 \Rightarrow -1$

- **Practically useful**
- **Theoretically interesting**

Learning from weighted data

Boosting used for both Regression and Classification,

- **Sometimes not all data points are equal** but only Classification discussed here
  - Some data points are more important than others
- **Consider a weighted dataset**
  - D(i) – weight of *i* th training example ($\mathbf{x}^i, y^i$)
  - Interpretations:
    - *i* th training example counts as D(i) examples
    - If I were to "resample" data, I would get more samples of "heavier" data points
- **Now, in all calculations, whenever used, *i* th training example counts as D(i) "examples"**
  - e.g., MLE for Naïve Bayes, redefine *Count(Y=y)* to be weighted count

$\hat{P}(Y=1) = \frac{\#(Y=1)}{m}$ by default

$\hat{P}(Y=0) = 1 - \hat{P}(Y=1)$

weighted with $D(i)$

$\hat{P}(Y=1) = \frac{\sum_{i=1}^{m} D(i)\,\delta(Y=1)}{\sum_{i=1}^{m} D(i)}$ indicator function 1 if $Y=1$, 0 else

$C_1 \quad C_2$

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$

Initialize $D_1(i) = 1/m$. uniform weight

Iteration number

For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \rightarrow \{-1, +1\}$.
- Choose $\alpha_t \in \mathbb{R}$. Compute this, method discussed later. Does not depend on samples
- Update:

$$D_{t+1}(i) = \frac{D_t(i) \exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

a probability due to Z_t

correct label => Lower weight

where Z_t is a normalization factor

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

same over all examples

Output the final classifier:

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

Week 7 review

Figure 1: The boosting algorithm AdaBoost.

Given: $(x_1, y_1), \ldots, (x_m, y_m)$ where $x_i \in X, y_i \in Y = \{-1, +1\}$
Initialize $D_1(i) = 1/m$.
For $t = 1, \ldots, T$:

- Train base learner using distribution D_t.
- Get base classifier $h_t : X \to \{-1, +1\}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

$$\boxed{\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

$$D_{t+1}(i) = \frac{D_t(i)\exp(-\alpha_t y_i h_t(x_i))}{Z_t}$$

Listen to later this point for Threshold discussion

$$\boxed{\epsilon_t = P_{i \sim D_i}[\mathbf{x}^i \neq y^i]}$$

$$\boxed{\epsilon_t = \frac{1}{\sum_{i=1}^m D_t(i)} \sum_{i=1}^m D_t(i)\delta(h_t(x_i) \neq y_i)}$$

Probability of error

What α_t to choose for hypothesis h_t?

easy to understand upper bound on training error

[Schapire, 1989]

Training error of final classifier is bounded by:

$\hat{y}_i \in R$ since $\alpha_T \in R$

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i) \leq \frac{1}{m}\sum_{i=1}^{m}\exp(-y_i f(x_i))$$

0 or 1

wrong => exp(+) > 1

right => exp(-) OK < 1

Where $f(x) = \sum_t \alpha_t h_t(x); H(x) = sign(f(x))$

$\forall$ values in the sum the RHS > LHS

> 1 or > 0

vs 0 or 1

What α_t to choose for hypothesis h_t?

[Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i)\leq\frac{1}{m}\sum_{i=1}^{m}\exp(-y_i f(x_i))=\prod_t Z_t$$

Where $f(x)=\sum_t \alpha_t h_t(x)$; $H(x)=sign(f(x))$

$$\boxed{Z_t=\sum_{i=1}^{m}D_t(i)\exp(-\alpha_t y_i h_t(x_i))}$$

no proof, just results
"not difficult to prove"
visit Office Hours for more
detail

What α_t to choose for hypothesis h_t?

[Schapire, 1989]

Training error of final classifier is bounded by:

$$\frac{1}{m}\sum_{i=1}^{m}\delta(H(x_i)\neq y_i)\leq\frac{1}{m}\sum_i \exp(-y_i f(x_i))=\prod_t Z_t$$

Where $f(x)=\sum_t \alpha_t h_t(x); H(x)=sign(f(x))$

If we minimize $\prod_t Z_t$, we minimize our training error

why do classification problems not minimize training error directly?

We can tighten this bound greedily, by choosing α_t and h_t on each iteration to minimize Z_t. ⇐ Loss function to minimize

$$Z_t=\sum_{i=1}^{m} D_t(i)\exp(-\alpha_t y_i h_t(x_i))$$

What α_t to choose for hypothesis h_t?

[Schapire, 1989]

We can minimize this bound by choosing α_t on each iteration to minimize Z_t.

$$Z_t = \sum_{i=1}^{m} D_t(i) \exp(-\alpha_t y_i h_t(x_i))$$

For binary weak classifiers, this is accomplished by [Freund & Schapire '97]:

training error

$$\boxed{\alpha_t = \frac{1}{2} \ln\left(\frac{1-\epsilon_t}{\epsilon_t}\right)}$$

Sample Test Question: How to prove it?

↑ Scary

Take partial derivative a la MLE

$Z_t = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x_i))$

$\frac{\partial Z_t}{\partial \alpha_t} = \sum_i D_t(i) \exp(-\alpha_t y_i h_t(x)) (-y_i h_t(x))$

$y_i h_t(x_i) =$ 1 when correct or -1 when wrong

$-\sum_{i \in \text{Correct}} D_t(i) \exp(-\alpha_t)$

$\Rightarrow + \sum_{i \in \text{Incorrect}} D_t(i) \exp(\alpha_t) = 0$

Then solve for α_t

$\Rightarrow \underbrace{e^{\alpha_t} \sum_{i \in \text{Incorrect}} D_t(i)}_{(e^{\alpha_t})\, e^{\alpha_t} \varepsilon_t} = e^{-\alpha_t} \sum_{i \in \text{correct}} D_t(i) = e^{-\alpha_t}(1-\varepsilon_t)(e^{\alpha_t})$

$\Rightarrow e^{2\alpha_t} \varepsilon_t = 1 - \varepsilon$ $\cdot \ln$

$2\alpha_t = \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

$\alpha_t = \frac{1}{2} \ln\left(\frac{1-\varepsilon_t}{\varepsilon_t}\right)$

Strong, weak classifiers

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$

- AdaBoost will achieve zero *training error* (exponentially fast):

$$\frac{1}{m} \sum_{i=1}^{m} \delta(H(x_i) \neq y_i) \leq \prod_t Z_t \leq \exp\left(-2 \sum_{t=1}^{T} (1/2 - \epsilon_t)^2\right)$$

To get accuracy < 50%, purposefully classify incorrectly ⇒ negative of all $\hat{y}$

- Is it hard to achieve better than random training error?

> 50% accuracy should not be hard

Boosting results – Digit recognition

[Schapire, 1989]

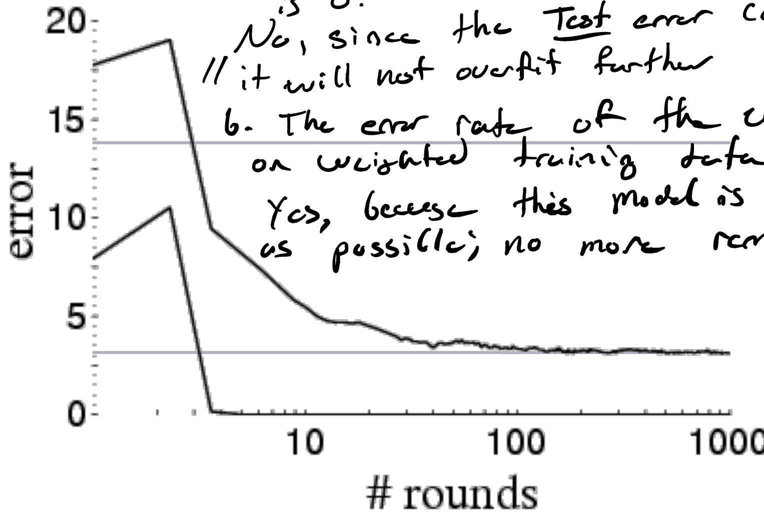
"In Boosting, would you stop the iteration if the following happens? Justify your answer in at most two sentences"

a. error rate of combined classifier on the original data is 0.

No, since the Test error can still reduce but it will not overfit further

b. The error rate of the current weak classifier on weighted training data is 0.

Yes, because this model is as close to perfect as possible; no more remains for it to learn.

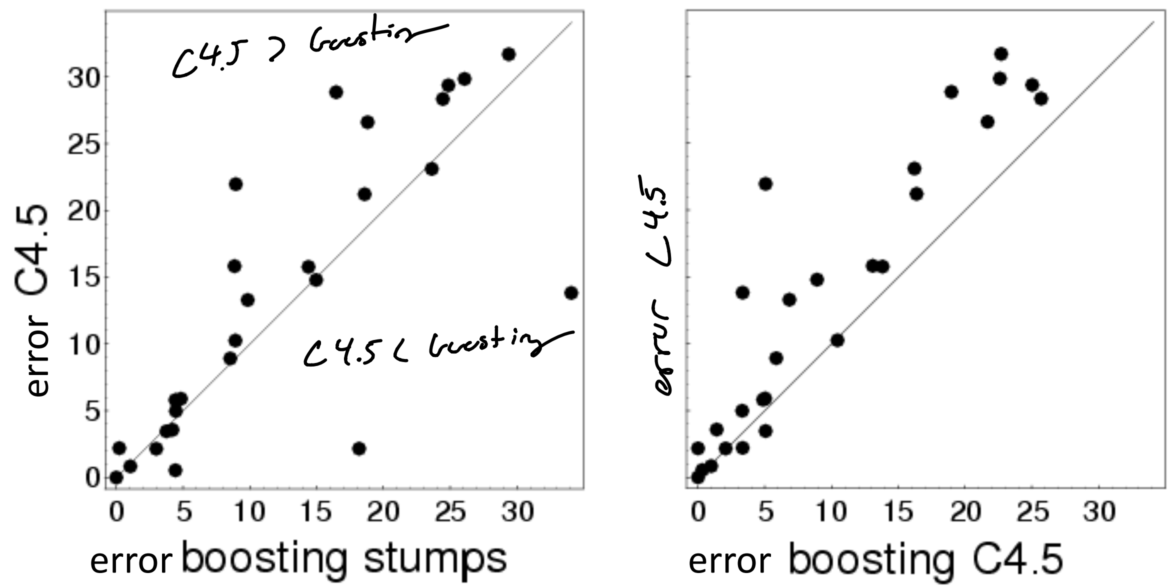


- Boosting often
  - Robust to overfitting
  - Test set error decreases even after training error is zero

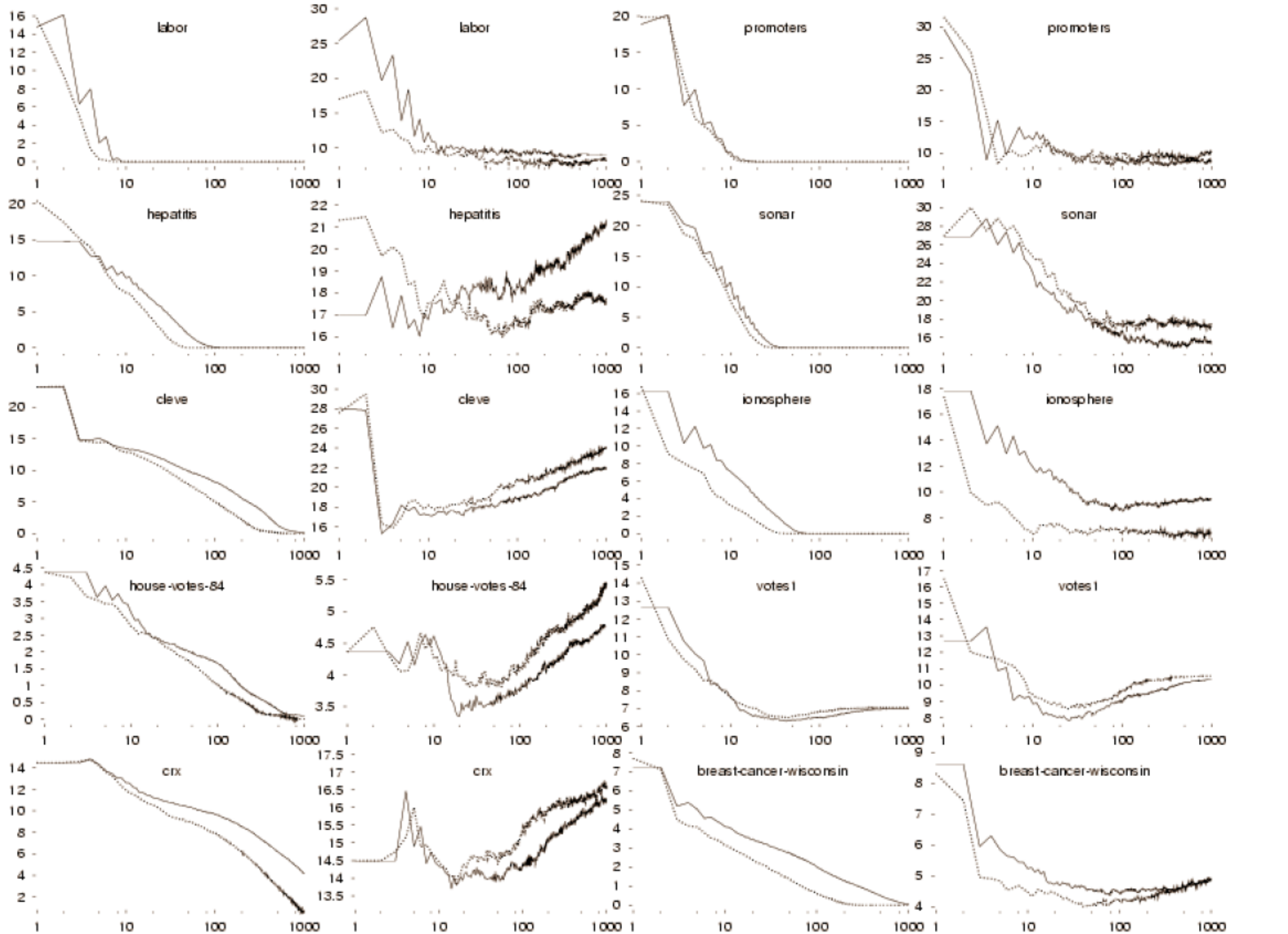
Boosting: Experimental Results

[Freund & Schapire, 1996]

Comparison of C4.5, Boosting C4.5, Boosting decision stumps (depth 1 trees), 27 benchmark datasets



AdaBoost and AdaBoost.MH on Train (left) and Test (right) data from Irvine repository. [Schapire and Singer, ML 1999]



Discriminative Models

Boosting and Logistic Regression

Logistic regression assumes:

$$P(Y=1|X) = \frac{1}{1+\exp(-f(x))}$$

And tries to maximize data likelihood:

$$P(\mathcal{D}|H) = \prod_{i=1}^{m} \frac{1}{1+\exp(-y_i f(x_i))}$$

$$f(x_i) = w \cdot x_i = w_0 + \sum w_j x_{ij}$$

Equivalent to minimizing log loss

negative likelihood

$$\sum_{i=1}^{m} \ln(1+\exp(-\underbrace{y_i f(x_i)}_{\text{very big}}))$$

very big => log is very small

Boosting and Logistic Regression

Logistic regression equivalent to minimizing log loss

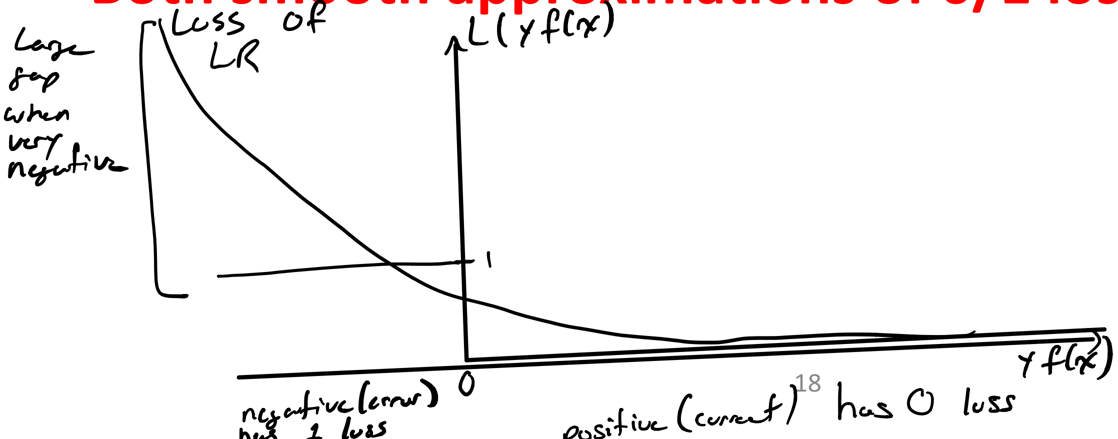
$$\sum_{i=1}^{m} \ln(1 + \exp(-y_i f(x_i)))$$

Boosting minimizes similar loss function!!

Sum over each iteration $\alpha_t h_t(x_i)$

$$\frac{1}{m}\sum_{i} \exp(-y_i f(x_i)) = \prod_{t} Z_t$$

Both smooth approximations of 0/1 loss!



LR: $f(x) = \sum_i w_i x_i$

Boosting: $f(x) = \sum_t \alpha_t h_t(x)$

$\text{sign}(f(x)) = \begin{cases} +1 & \text{if } f(x) > 0 \\ -1 & \text{if } f(x) < 0 \end{cases}$

LR: Loss $\ln(1 + \exp(-yf(x)))$

Hard to optimize

Logistic regression and Boosting

Logistic regression:

- Minimize loss fn

$$\sum_{i=1}^{m} \ln(1+\exp(-y_i f(x_i)))$$

- Define

$$f(x) = \sum_j w_j x_j$$

where x_j predefined

Boosting:

- Minimize loss fn

$$\sum_{i=1}^{m} \exp(-y_i f(x_i))$$

- Define

$$f(x) = \sum_t \alpha_t h_t(x)$$

where $h_t(x_i)$ defined dynamically to fit data

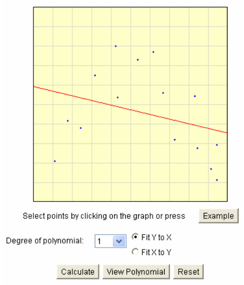
(not a linear classifier)

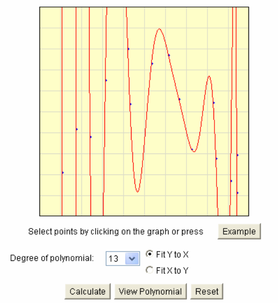
- Weights α_t learned incrementally

OK… now we'll learn to pick those darned parameters…

- **Selecting features (or basis functions)**
  - Linear regression
  - Naïve Bayes
  - Logistic regression
- **Selecting parameter value**
  - Prior strength
    - Naïve Bayes, linear and logistic regression
  - Regularization strength
    - Naïve Bayes, linear and logistic regression
  - Decision trees
    - MaxpChance, depth, number of leaves
  - Boosting
    - Number of rounds
- These are called **Model Selection** Problems

Test set error as a function of model complexity





Simple greedy model selection algorithm

- Pick a dictionary of features
  - e.g., polynomials for linear regression
- Greedy heuristic:
  - Start from empty (or simple) set of features $F_0 = \emptyset$
  - Run learning algorithm for current set of features F_t
    - Obtain h_t
  - Select **next best feature $\mathbf{X_i}$**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

should be sam basis function

Greedy model selection

- Applicable in many settings:
  - Linear regression: Selecting basis functions
  - Naïve Bayes: Selecting (independent) features $P(X_i|Y)$
  - Logistic regression: Selecting features (basis functions)
- Only a heuristic!
- There are many more elaborate methods out there

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature $\mathbf{X_i}$**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

When do you stop???

- When training error is low enough?

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature X_i**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

When do you stop???

- ~~When training error is low enough?~~
- When test set error is low enough?

Validation set

- Thus far: Given a dataset, **randomly** split it into two parts:
  - Training data – {$\mathbf{x}_1$,…, $\mathbf{x}_{\text{Ntrain}}$}
  - Test data – {$\mathbf{x}_1$,…, $\mathbf{x}_{\text{Ntest}}$}
- But **Test data must always remain independent**!
  - Never ever ever ever learn on test data, including for model selection
- Given a dataset, **randomly** split it into three parts:
  - Training data – {$\mathbf{x}_1$,…, $\mathbf{x}_{\text{Ntrain}}$}
  - Validation data – {$\mathbf{x}_1$,…, $\mathbf{x}_{\text{Nvalid}}$}
  - Test data – {$\mathbf{x}_1$,…, $\mathbf{x}_{\text{Ntest}}$}
- Use validation data for tuning learning algorithm, e.g., model selection
  - Save test data for very final evaluation

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature X_i**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

When do you stop???

- ~~When training error is low enough?~~
- ~~When test set error is low enough?~~
- When validation set error is low enough?

Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature X_i**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

When do you stop???

- ~~When training error is low enough?~~
- ~~When test set error is low enough?~~
- ~~When validation set error is low enough?~~
- Man!!! OK, should I just repeat until I get tired???
  - I am tired now…
  - **No, "There is a better way!"**

(LOO) Leave-one-out cross validation

- Consider a **validation set with 1 example**:
  - *D* – training data
  - *D*\i – training data with *i* th data point moved to validation set
- **Learn classifier $h_{D\backslash i}$ with *D*\i dataset**
- **Estimate true error** as:
  - 0 if $h_{D\backslash i}$ classifies *i* th data point correctly
  - 1 if $h_{D\backslash i}$ is wrong about *i* th data point
  - Seems really bad estimator, but wait!
- **LOO cross validation**: Average over all data points *i*:
  - **For each data point you leave out, learn a new classifier $h_{D\backslash i}$**
  - **Estimate error** as:

$$error_{LOO} = \frac{1}{m} \sum_{i=1}^{m} \mathbb{1}\left(h_{\mathcal{D}\backslash i}(\mathbf{x}^i) \neq y^i\right)$$

LOO cross validation is (almost) unbiased estimate of true error!

- When computing **LOOCV error, we only use *m-1* data points**
  - So it's not estimate of true error of learning with *m* data points!
  - Usually pessimistic, though – learning with less data typically gives worse answer
- **LOO is almost unbiased**!
  - Let $error_{true,m-1}$ be true error of learner when you only get *m-1* data points
  - LOO is unbiased estimate of $error_{true,m-1}$:

$$E_{\mathcal{D}}[error_{LOO}] = error_{true,m-1}$$

- **Great news!**
  - **Use LOO error for model selection!!!**

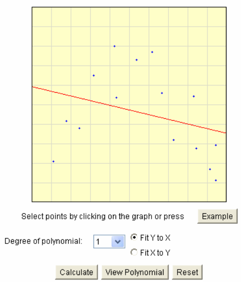
Simple greedy model selection algorithm

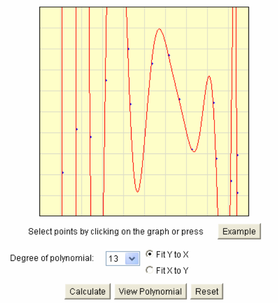
- Greedy heuristic:
  - …
  - Select **next best feature X_i**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_j\}$
  - Recurse

When do you stop???

- ~~When training error is low enough?~~
- ~~When test set error is low enough?~~
- ~~When validation set error is low enough?~~
- **STOP WHEN error_{LOO} IS LOW!!!**

Using LOO error for model selection





Computational cost of LOO

- Suppose you have 100,000 data points
- You implemented a great version of your learning algorithm
  - Learns in only 1 second
- Computing LOO will take about 1 day!!!
  - If you have to do so for each choice of basis functions, it will take fooooooreeever!!!
- Solution 1: Preferred, but not usually possible
  - Find a cool trick to compute LOO

Solution to complexity of computing LOO:

(More typical) **Use *k*-fold cross validation**

Know LOO definition
Complexity cost of LOO
Def of k-fold
how it relates to LOO

- Randomly **divide training data into *k* equal parts**
  - $D_1,\ldots,D_k$
- For each *i*
  - **Learn classifier $h_{D\backslash Di}$ using data point not in D_i**
  - **Estimate error of $h_{D\backslash Di}$ on validation set D_i:**

$$error_{\mathcal{D}_i} = \frac{k}{m} \sum_{(\mathbf{x}^j, y^j) \in \mathcal{D}_i} \mathbb{1}\left(h_{\mathcal{D}\backslash\mathcal{D}_i}(\mathbf{x}^j) \neq y^j\right)$$

- ***k*-fold cross validation error is average** over data splits:

$$error_{k-fold} = \frac{1}{k} \sum_{i=1}^{k} error_{\mathcal{D}_i}$$

- *k*-fold cross validation properties:
  - **Much faster to compute** than LOO
  - **More (pessimistically) biased** – using much less data, only *m(k-1)/k* ← no free lunch
  - Usually, **k = 10** ☺
    K runs