Boosting
Simple Model Selection
Cross Validation
Regularization

Machine Learning – 10701/15781

Carlos Guestrin

Carnegie Mellon University

October 3rd, 2007

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:
  - weight each training example by how incorrectly it was classified
  - Learn a hypothesis – h_t
  - A strength for this hypothesis – α_t
- Final classifier: $H(X) = \text{sign}\left\{ \sum_{t=1}^{T} \alpha_t h_t(x) \right\}$
- **Practically useful**
- **Theoretically interesting**

AdaBoost

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

- Train base learner using distribution D_t. (weights) ← as in previous slide
- Get base classifier $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

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

where Z_t is a normalization factor

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

Output the final classifier:

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

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 $h_t : X \to \mathbb{R}$.
- Choose $\alpha_t \in \mathbb{R}$.
- Update:

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

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

ϵ_t is weighted error of $h_t(x)$ at iteration t:

$$\epsilon_t = P_{i \sim D_t}[x^i \neq y^i]$$

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

as $\epsilon_t \to 0$, $\alpha_t \to +\infty$ ← really trust

as $\epsilon_t \to 1$, $\alpha_t \to -\infty$ ← really trust opposite

if $\epsilon_t = 0.5$, $\alpha_t = 0$, random classifiers are bad ⇒ zero weight

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

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

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

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

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

You'll prove this in your homework! ☺

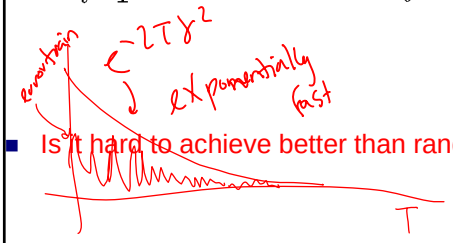
Strong, weak classifiers

- If each classifier is (at least slightly) better than random
  - $\varepsilon_t < 0.5$
- AdaBoost will achieve zero *training error* (exponentially fast):

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

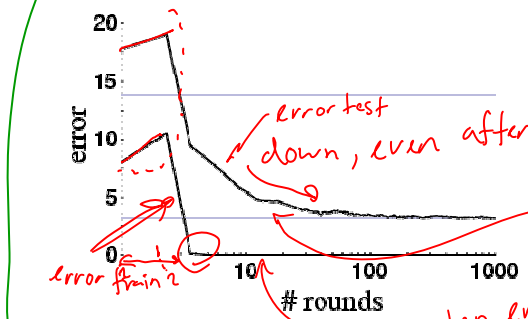
$(1/2-\varepsilon_t)^2$ ← how much better is ε_t than random

- Is it hard to achieve better than random training error? if $|1/2-\varepsilon_t| \geq \gamma$



Boosting results – Digit recognition

[Schapire, 1989]



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

Boosting generalization error bound

[Freund & Schapire, 1996]

$$error_{true}(H) \leq error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

true error — #iterations — train — extra terms

T small — large — small

T v. large — → 0 — large

bias — "variance"

- T – number of boosting rounds
- d – VC dimension of weak learner, measures complexity of classifier
- m – number of training examples

Boosting generalization error bound

[Freund & Schapire, 1996]

$$error_{true}(H) \leq error_{train}(H) + \tilde{O}\left(\sqrt{\frac{Td}{m}}\right)$$

- **Contradicts**: Boosting often
  - Robust to overfitting ("too many" iterations hurt)
  - Test set error decreases even after training error is zero
- **Need better analysis tools**
  - we'll come back to this later in the semester

if $error_{train}(H) = 0$, then bound gets worse with iterations

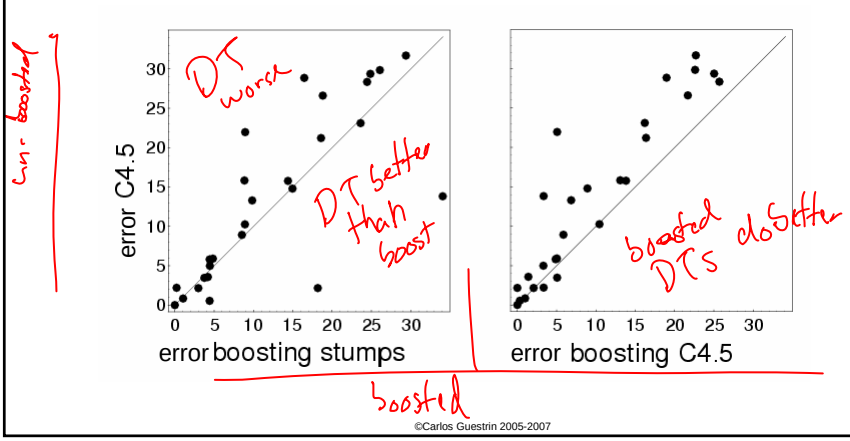
if we are lucky

- T – number of boosting rounds
- d – VC dimension of weak learner, measures complexity of classifier
- m – number of training examples

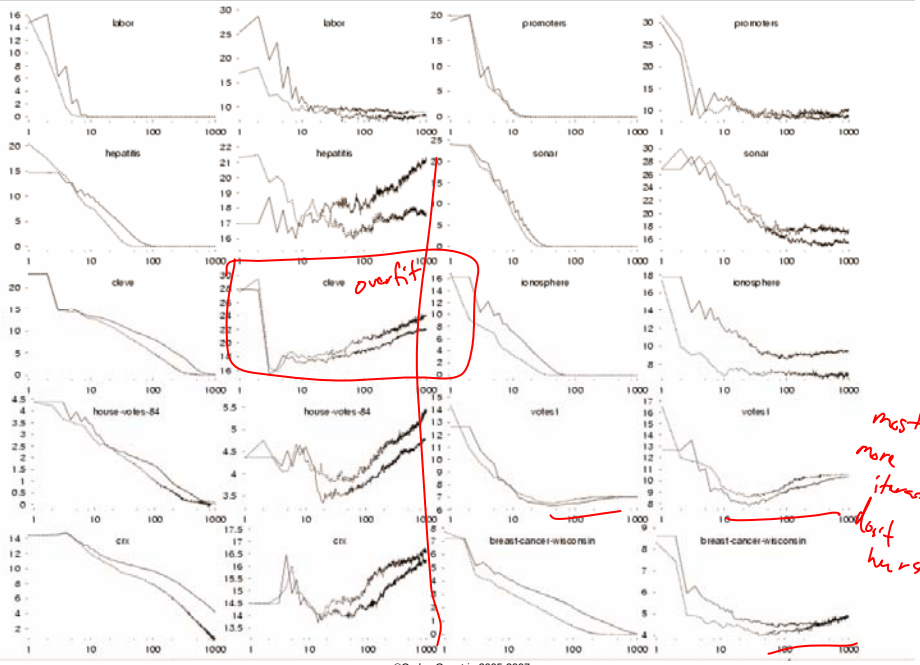
Boosting: Experimental Results

[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]



Boosting and Logistic Regression

Logistic regression assumes:

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

And tries to maximize data likelihood:

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

Equivalent to minimizing log loss

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

Boosting and Logistic Regression

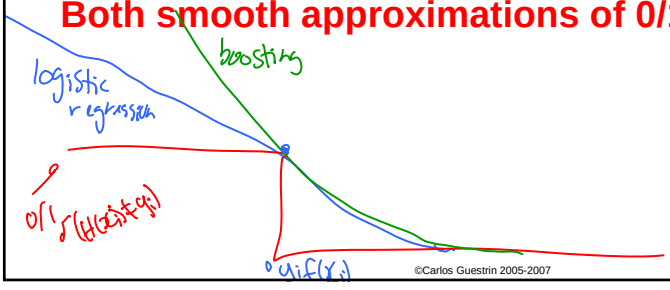
Logistic regression equivalent to minimizing log loss

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

Boosting minimizes similar loss function!!

$$\frac{1}{m} \sum_i \exp(-y_i f(x_i)) = \prod_t Z_t$$

Both smooth approximations of 0/1 loss!



Logistic regression and Boosting

boosting: "learning features of the data"

Logistic regression:

- Minimize loss fn (log loss)

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

- Define

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

where x_j predefined

use boosting to generate "features" → then use those features in LR

fixed

Boosting:

- Minimize loss fn (exp. loss)

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

- Define (weak classifier)

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

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

(not a linear classifier)

- Weights α_j learned incrementally

What you need to know about Boosting

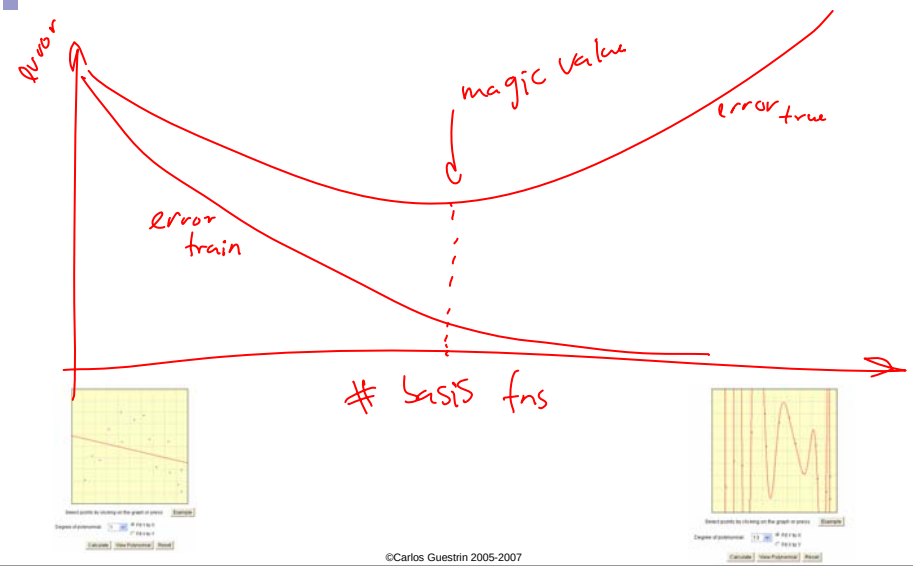
- Combine weak classifiers to obtain very strong classifier
  - Weak classifier – slightly better than random on training data
  - Resulting very strong classifier – can eventually provide zero training error
- AdaBoost algorithm
- Boosting v. Logistic Regression
  - Similar loss functions
  - Single optimization (LR) v. Incrementally improving classification (B)
- Most popular application of Boosting:
  - Boosted decision stumps!
  - Very simple to implement, very effective classifier

HW2 (is out !!) :)

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
- More generally, these are called **Model Selection** Problems
- Today:
  - Describe basic idea
  - Introduce very important concept for tuning learning approaches: **Cross-Validation**

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 $F_0 = \varnothing$
  - Run learning algorithm for current set of features F_t
    - Obtain h_t
  - Select **next best feature X_j**
    - 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

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)
  - Decision trees: Selecting leaves to expand
- Only a heuristic!
  - But, sometimes you can prove something cool about it
    - e.g., [Krause & Guestrin '05]: Near-optimal in some settings that include Naïve Bayes
- There are many more elaborate methods out there

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 $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - Recurse

2:59 pm when HW due?

When do you stop???

- When training error is low enough?

↳ overfitting: low train error $\not\Rightarrow$ low true error

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 $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - Recurse

When do you stop???

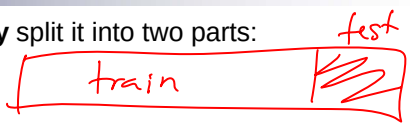
- ~~When training error is low enough?~~
- When test set error is low enough?

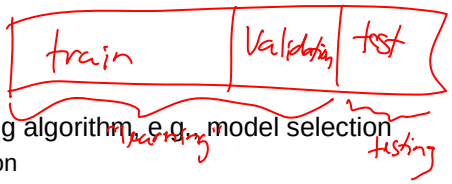
never, ever, ever ... train on test set!

:(Sad!

Validation set

- Thus far: Given a dataset, **randomly** split it into two parts:
  - Training data – $\{\mathbf{x}_1,\ldots, \mathbf{x}_{Ntrain}\}$
  - Test data – $\{\mathbf{x}_1,\ldots, \mathbf{x}_{Ntest}\}$
- 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 – $\{\mathbf{x}_1,\ldots, \mathbf{x}_{Ntrain}\}$
  - Validation data – $\{\mathbf{x}_1,\ldots, \mathbf{x}_{Nvalid}\}$
  - Test data – $\{\mathbf{x}_1,\ldots, \mathbf{x}_{Ntest}\}$
- 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 $\mathbf{X_i}$**
    - 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?~~
- ~~When test set error is low enough?~~
- When validation set error is low enough?

minimized

overfit to Validation Set

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 $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - 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
  - $D\backslash i$ – training data with ith data point moved to validation set
- **Learn classifier $h_{D\backslash i}$ with $D\backslash i$ dataset**
- **Estimate true error** as:
  - 0 if $h_{D\backslash i}$ classifies ith data point correctly
  - 1 if $h_{D\backslash i}$ is wrong about ith 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
  - In homework, you'll prove that 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!!!**

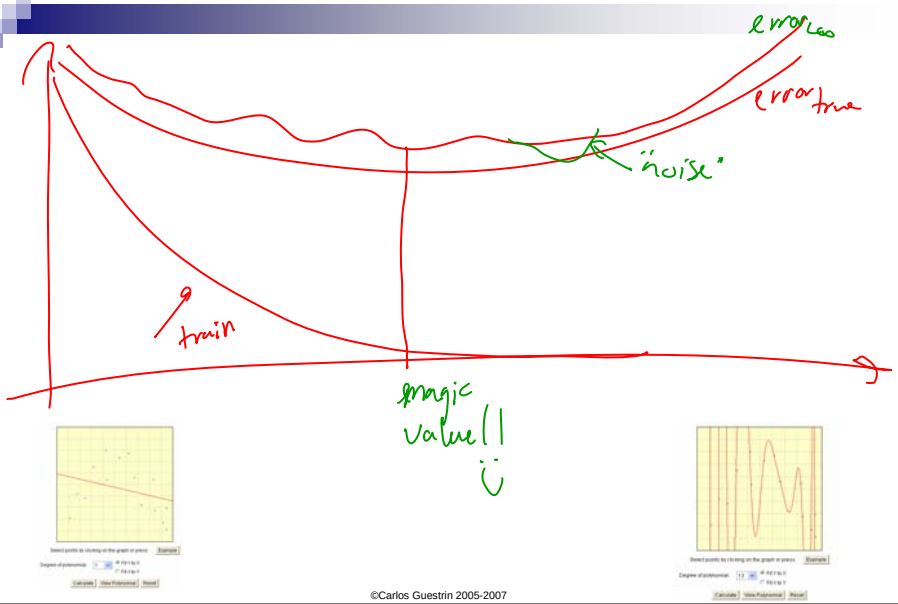
Simple greedy model selection algorithm

- Greedy heuristic:
  - …
  - Select **next best feature X_i**
    - e.g., X_j that results in lowest training error learner when learning with $F_t \cup \{X_j\}$
  - $F_{t+1} \leftarrow F_t \cup \{X_i\}$
  - 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 for each choice of basis functions, it will take fooooooreeeve'!!!
- Solution 1: Preferred, but not usually possible
  - Find a cool trick to compute LOO (e.g., see homework)

Solution 2 to complexity of computing LOO:

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

- 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:**

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

- ***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*
  - Usually, **k = 10** ☺

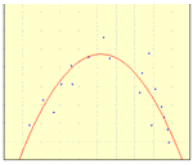
Regularization – Revisited

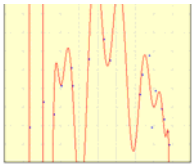
- Model selection 1: **Greedy**
  - Pick subset of features that have yield low LOO error
- Model selection 2: **Regularization**
  - Include **all possible features!**
  - **Penalize "complicated" hypothesis**

Regularization in linear regression

- Overfitting usually leads to very large parameter choices, e.g.:

-2.2 + 3.1 X – 0.30 X^2 -1.1 + 4,700,910.7 X – 8,585,638.4 X^2 + …





- Regularized least-squares (a.k.a. ridge regression), for $\lambda \geq 0$:

$$\mathbf{w}^* = \arg\min_{\mathbf{w}} \sum_j \left(t(\mathbf{x}_j) - \sum_i w_i h_i(\mathbf{x}_j) \right)^2 + \lambda \sum_{i=1}^{k} w_i^2$$

squared error regularization

Other regularization examples

- **Logistic regression** regularization
  - Maximize data likelihood minus **penalty for large parameters**

$$\arg\max_{\mathbf{w}} \sum_j \ln P(y^j | \mathbf{x}^j, \mathbf{w}) - \lambda \sum_i w_i^2$$

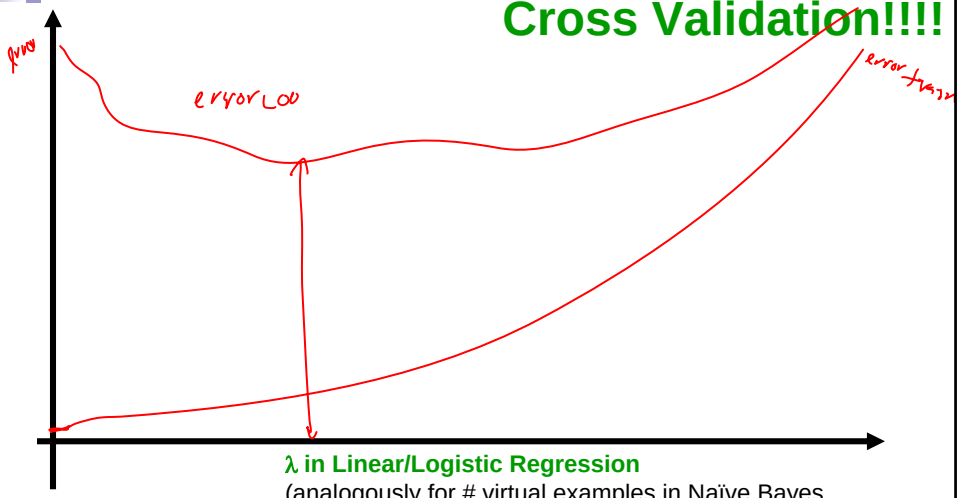
  - **Biases towards small parameter values**

- **Naïve Bayes** regularization
  - **Prior** over likelihood of features
  - **Biases away from zero probability** outcomes

$P(X_i | Y)$ "Spam" never saw a word, but added a small count

- **Decision tree** regularization
  - Many possibilities, e.g., **Chi-Square test and MaxPvalue** parameter
  - **Biases towards smaller trees**

How do we pick magic parameter?



Regularization and Bayesian learning

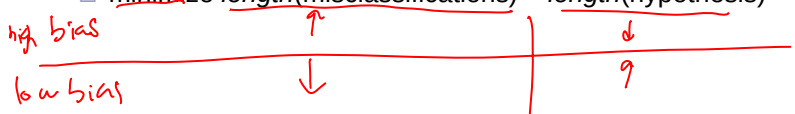
$$p(\mathbf{w} \mid Y, \mathbf{X}) \propto P(Y \mid \mathbf{X}, \mathbf{w}) p(\mathbf{w})$$

squared: $\lambda \sum_i w_i^2$

- We already saw that **regularization for logistic regression** corresponds to **MAP for zero mean, Gaussian prior** for **w**
- Similar interpretation for other learning approaches:
  - **Linear regression**: Also zero mean, Gaussian prior for **w**
  - **Naïve Bayes**: Directly defined as prior over parameters
  - **Decision trees**: Trickier to define… but we'll get back to this

Occam's Razor

- William of Ockham (1285-1349) *Principle of Parsimony*:
  - "One should not increase, beyond what is necessary, the number of entities required to explain anything."
- Regularization penalizes for *"complex explanations"*
- Alternatively (but pretty much the same), use *Minimum Description Length (MDL) Principle*:
  - minimize *length*(misclassifications) + *length*(hypothesis)



- *length*(misclassifications) – e.g., #wrong training examples
- *length*(hypothesis) – e.g., size of decision tree

Minimum Description Length Principle

- MDL prefers small hypothesis that fit data well:

$$h_{MDL} = \arg\min_h L_{C_1}(\mathcal{D} \mid h) + L_{C_2}(h)$$

  - $L_{C1}(D|h)$ – description length of data under code C_1 given h
    - Only need to describe points that h doesn't explain (classify correctly)
  - $L_{C2}(h)$ – description length of hypothesis h
- Decision tree example
  - $L_{C1}(D|h)$ – #bits required to describe data given h
    - If all points correctly classified, $L_{C1}(D|h) = 0$
  - $L_{C2}(h)$ – #bits necessary to encode tree
  - Trade off quality of classification with tree size

Bayesian interpretation of MDL Principle

- MAP estimate

$$h_{MAP} = \arg\max_h [P(\mathcal{D} \mid h) P(h)]$$
$$= \arg\max_h [\log_2 P(\mathcal{D} \mid h) + \log_2 P(h)]$$
$$= \arg\min_h [-\log_2 P(\mathcal{D} \mid h) - \log_2 P(h)]$$

- **Information theory fact**:
  - Smallest code for event of probability *p* requires $-\log_2 p$ bits
- MDL interpretation of MAP:
  - $-\log_2 P(D|h)$ – length of *D* under hypothesis *h*
  - $-\log_2 P(h)$ – length of hypothesis *h* (there is hidden parameter here)
  - MAP prefers simpler hypothesis:
    - minimize *length*(misclassifications) + *length*(hypothesis)
- **In general, Bayesian approach usually looks for simpler hypothesis** – Acts as a regularizer

What you need to know about Model Selection, Regularization and Cross Validation

- Cross validation
  - (Mostly) Unbiased estimate of true error
  - LOOCV is great, but hard to compute
  - *k*-fold much more practical
  - Use for selecting parameter values!
- Model selection
  - Search for a model with low cross validation error
- Regularization
  - Penalizes for complex models
  - Select parameter with cross validation
  - Really a Bayesian approach
- Minimum description length
  - Information theoretic interpretation of regularization
  - Relationship to MAP