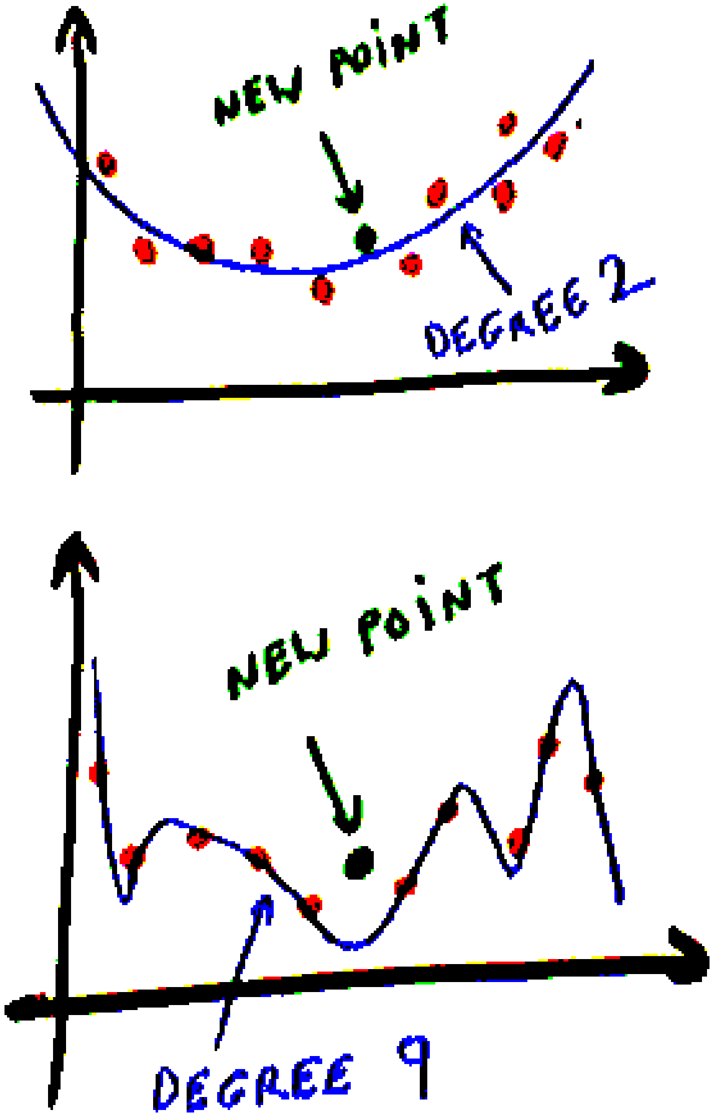
MACHINE LEARNING AND PATTERN RECOGNITION

Spring 2004, Lecture 7: Model Selection, Cross-Validation, Bootstrap/Bagging, and Boosting.

Yann LeCun
The Courant Institute,
New York University
http://yann.lecun.com

Model Selection



- Say we train a model M^1 (e.g. degree 2 polynomials) on a training set and we get a training error E^1_{train}
- Say we train a second model M^2 (e.g. degre 9 polynomials) on the same training set, and we get a training error E^2_{train}
- Presumably, E^2_{train} will be smaller than E^1_{train} because the family of functions in M^2 contains the set of functions in M^1.
- How do we decide which model is best?
- This is the model selection problem.

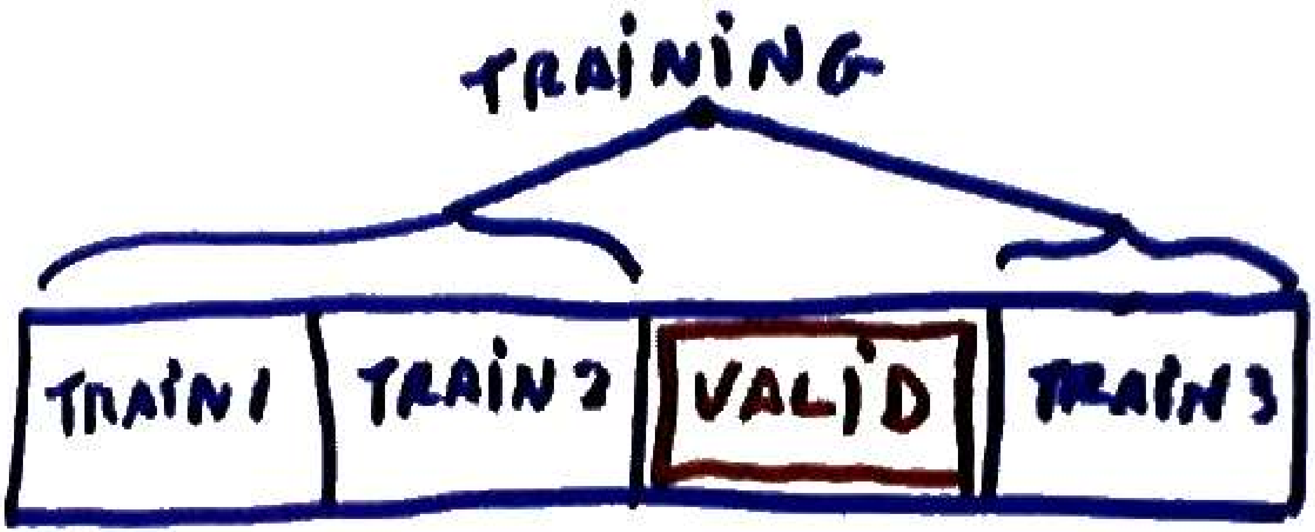
Test Set

- The most natural idea is to divide the set of samples into a training set and a test set.
- Train both models on the training set, and pick the one that performs best on the test set.
- PROBLEM 1: The result depends on the particular test set we chose. Another drawing of the test set might have given slightly different results.
- PROBLEM 2: This forces us to sacrifice some data that could be otherwise used for training
- PROBLEM 3: If we use the “test set” to select models, then it’s not a test set anymore, because we allow ourselves to look at it to select a model.

Using a Validation Set

- We further split the training set into a reduced training set and a *validation* set. We train on the reduced training set, and select our model on the basis of the result on the validation set.
- Once the model is select, we test it on the test set.
- PROBLEM: This takes away even more training samples, when we should use all the samples we have to train a machine.
- SOLUTION: K-Fold Validation and Leave-one-out.

K-Fold Validation



- To prevent the effect of picking a particular validation set, we make many partitions into reduced training set and validation set, and average the results.
- We split the training set in K equally sized subsets.
- We train on $K-1$ subsets, and test (validate) on the remaining subset.
- We repeat the process for all possible choices of the validation subset and average the error rates thereby obtained.
- We pick the model with the lowest average error.
- This process is called *K-fold cross-validation*.

Leave-One-Out

- The Leave-One-Out method is the limit case of K-fold cross-validation for $K = P$: Each validation set has a single sample, and the reduced training set has $P - 1$ samples.
- Bounds/Estimates of the leave-one-out error can be computed analytically for some classes of models (e.g. linear models).
- PROBLEM: this method is quite expensive computationally, and therefore is only practical for fairly small problems.

Procedure for Model Selection

- A conceptually simple, but computationally expensive method for model selection:
- choose/design a set of models of various capacities (e.g. neural nets with more and more hidden units, polynomials of higher and higher degree, etc...)
- Train all the available models and make an *estimate* of the “test” error for each of them *without touching the actual test*.
- pick the model with the best estimated “test” error.
- test it on the (yet untouched) test set.
- PROBLEM: How can we estimate the *expected risk* (a.k.a. the test error, a.k.a. the generalization error) without touching the test set?

Estimating the Expected Risk

- Use cross validation (but it throws away training data)
- Use a formula from Learning Theory (but they are inaccurate).
- Most formulae for the expected risk are of the form:

$$E_{\text{expected}} = E_{\text{train}} + c\frac{h}{P^{\alpha}}$$

Where c is a constant, P is the number of training samples, $alpha$ is a constant between 0.5 and 1.0, and h is a measure of *capacity* of the family of functions implemented by the model.

Estimating the Expected Risk

- PROBLEM 1: In fact, many formulae are probabilistic upper bounds rather than equalities: “with probability $1 - \delta$”:

$$E_{\text{expected}} < E_{\text{train}} + c(\delta)\frac{h}{P^{\alpha}}$$

- PROBLEM 2: The bound formulae derived from general learning theories are generally extremely loose, so you can’t just plug in numbers and hope that the formula will give you useful information. That is because general formulae cannot take into account the peculiarities of the problem at hand.
- PROBLEM 3: The “capacity” term h is generally impossible to know for a complex model family. Intuitively, h is akin to some sort of *effective number of free parameters* (which may or may not reflect the actual number of free parameters).

Bayesian Formula for Model Selection

- Bayesian Information Criterion (BIC):

$$E_{\text{expected}} = E_{\text{train}} + \frac{log(P)}{2}\frac{h}{P}$$

- can be derived from a Bayesian inference formula with a so-called “Laplace” approximation (2nd order taylor expansion of the log likelihood around the MLE estimate).

Structural Risk Minimization

- Structural Risk Minimization: these are very general bounds derived from first principles. The bad news is that they are distribution free, and therefore very loose.
- SRM bound for classification: with probability $1 - \delta$:

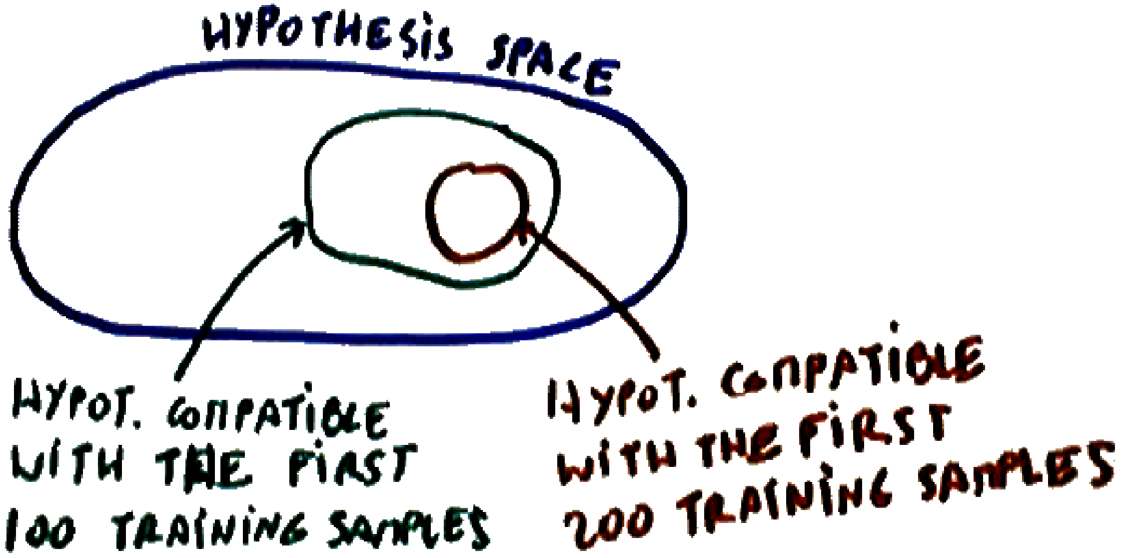
$$E_{\text{expected}} \leq E_{\text{train}} + \frac{\epsilon}{2}\left(1 + \sqrt{1 + \frac{4E_{\text{train}}}{\epsilon}}\right)$$

  with $\epsilon = a_1 \frac{h[log(a_2 P/h)+1] - log(\delta/4)}{P}$, where a_1, a_2 are constant not furnished by the theory, and h is the *Vapnik-Chervonenkis Dimension* of the family of function.
- SRM bound for regression

$$E_{\text{expected}} \leq \frac{E_{\text{train}}}{(1 - c\sqrt{(\epsilon)})_+}$$

  with the same ϵ as above.

An Example of How to get a Bound



- The hypothesis space of our model is the family of all possible functions indexed by our parameter W:

$$\mathcal{H} : \{F(Y, X, W), \forall W\}$$

  That set may have an infinite (possibly uncountable) cardinality.
- One view of learning: each training example eliminates elements of the hypothesis space that disagree with it.

An Example of How to get a Bound

- Let's assume that the number of hypotheses is finite and equal to k: $W_1, W_2, ...W_k$.
- Let's assume the loss is binary: 0 if correct, 1 if error.
- Let's define the expected error for a particular hypothesis $E_{exp}(W_i)$, the error that W_i over an infinite training set.
- Let's define the empirical error for P examples: $E^P_{emp}(W_i)$, the error that W_i over a particular training set of size P.
- The Hoeffding/Chernoff bound tells us:

$$P(|E_{exp}(W_i) - E^P_{emp}(W_i)| > \epsilon) < 2\exp(-2\epsilon^2 P)$$

- The Hoeffding/Chernoff bound is a wonderful formula that tells us how fast the average of a variable computed over a finite set of sample converges to the true expectation (average over an inifinte number of samples) as we increase the number of samples.

Bound on the Expected Error

Since the probability of the union of a set of events is bounded above by the sum of the individual event probabilities, we can write:

$$P(\exists W_i, |E_{exp}(W_i) - E_{emp}^P(W_i)| > \epsilon) \leq \sum_{i=1}^{k} 2\exp(-2\epsilon^2 P) = 2k\exp(-2\epsilon^2 P)$$

this can be rewritten as

$$P(\forall W_i, |E_{exp}(W_i) - E_{emp}^P(W_i)| \leq \epsilon) > 1 - 2k\exp(-2\epsilon^2 P)$$

This is a uniform convergence bound, because it holds for all hypotheses.

Bound on the Expected Error

Let's define $\delta = 2k\exp(-2\epsilon^2 P)$. Suppose we hold P and δ, and solve for ϵ, we get with probability $1-\delta$:

$$E_{exp}(W) \leq E_{emp}^{P}(W) + \sqrt{\frac{1}{2P}\log\frac{2k}{\delta}}$$

In particular, the previous inequality is true for the hypothesis W_{train} that minimizes the training set error:

$$E_{exp}(W_{\mathrm{train}}) \leq E_{emp}^{P}(W_{\mathrm{train}}) + \sqrt{\frac{1}{2P}\log\frac{2k}{\delta}}$$

VC-dimension

- The previous bound assumed the space of hypotheses was finite (with k hypotheses).
- The Vapnik-Chervonenkis approach derives similar results for infinite hypotheses spaces.
- key idea: as far as we are concerned, two hypotheses are identical if they produce the same classification on our dataset: identical hypotheses are put into equivalence classes.
- The formulae we obtain are the SRM formulae shown a few slides back where h is the VC-dimension of the family of functions
- The VC-dim is defined as the largest number of points (in any position) that our family of function could classify in every possible ways.
- EXAMPLE: The VC-dim of linear classifier with N inputs is $N + 1$: in dimension 2, there is a set of 3 points on which all 2^3 dichotomies are linearly separable. There is no such set with 4 points.

VC-dimension, continued

- The VC-dim and the SRM formulae are very interesting conceptually, because they are derived with a minimal set of assumptions.
- The VC theory allows us to “derive” and quantify the intuitive notion of Occam’s Razor from *first principles*.
- Occam’s Razor is the idea that simple models are preferable to complex one.
- Until the VC theory, Occam’s Razor was assumed to be a good idea, or was derived from rather contrived sets of assumptions.

How Can we get the Constants?

Recall that a good general form for the generalization error is:

$$E_{\text{expected}} = E_{\text{train}} + c\frac{h}{P^{\alpha}}$$

The constants c depends on the task and the learning algorithm used, h depends on the model, and $0.5 \leq \alpha \leq 1$ depends on the type of task (e.g. regression versus classification).
We can *measure* the constants c and α for a particular task by running a learning machine with a known h on the task (e.g. a linear classifier for which $h = N + 1$) with various size of the training set.
Then, if we assume c is constant from model to model, we can measure h for unknown learning machine by running on several size of training set and fitting the curve.
This process is rather long and expensive, and rarely worth it.

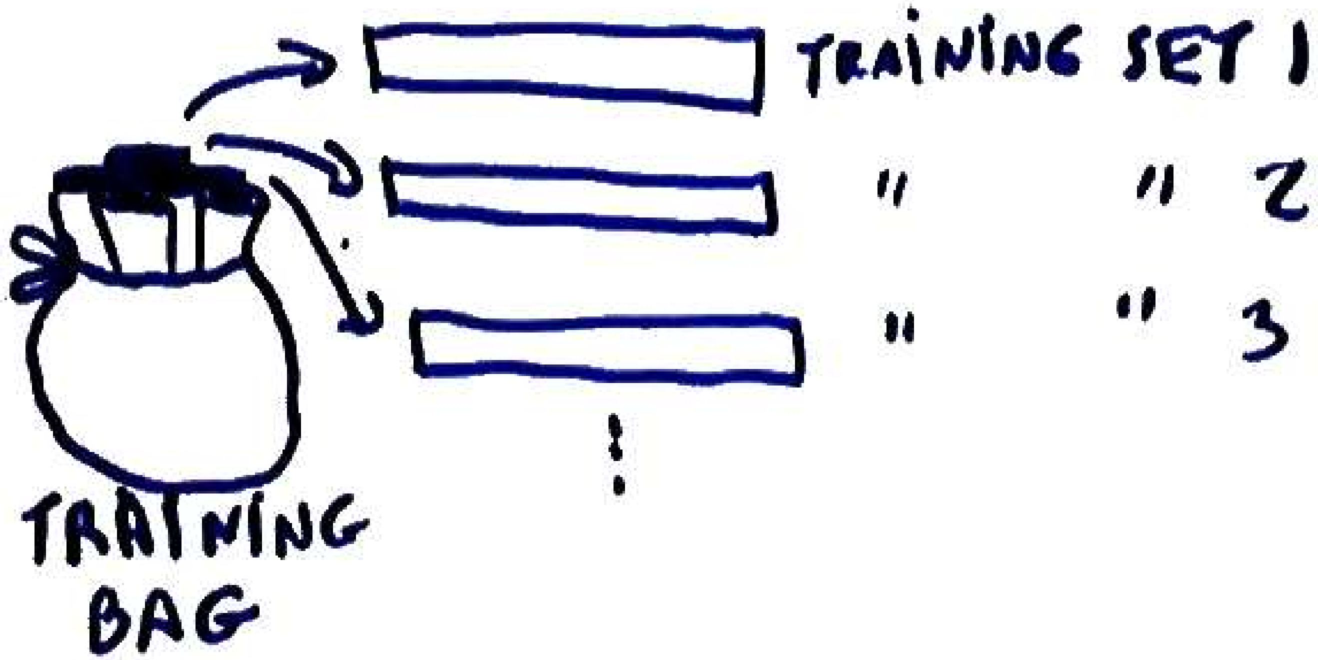
Conclusion on Model Selection

- It's really hard to beat validation or cross-validation.
- When the size/capacity of the family of functions is controled by a continuous parameter (e.g. the coefficient λ of a regularization term) we can use more efficient techniques than exhaustive search.
- Some learning architectures/algorithms come with bounds that are pretty accurate: e.g. Support Vector Machines, Adaboost.... That's because they are essentially linear machines.

Ensemble Methods

- It is sometimes useful to train many instances of a learning machine (in slightly different ways) and combine their outputs.
- Example: Bayesian learning. The outputs produced by replicas of the machine for each possible value of the parameter vector are added with weights that reflect the conditional probaility of the parameter vector given the training dataset.
- The key idea of ensemble methods: find a way to make each replica of the learning machine different from the others, yet useful.

Bootstrap/Bagging



- A simple idea: use different subsets of the training set.
- The Bootstrap method consists in generating multiple training set by drawing samples from the original training set *with replacement*.
- The Bagging method consists simply in averaging of the outputs produced by each of the instances.