Machine Learning

Ilya Narsky, Caltech

Plan

1. Problem of classification. Forms of empirical loss, or classification error. Training, validation and test. Cross-validation and bootstrap. Bayes (Schwartz) information criterion. Criteria for choosing classifiers.
2. Linear and quadratic discriminant analysis. Logistic regression. Naive Bayes classifier. Local averaging and k-NN methods. Neural networks. Radial basis functions. Support vector machines. Curse of dimensionality.
3. Decision trees. Bump hunting. Boosting: AdaBoost and arc-x4. Bagging and random forest. Comparison of reviewed classifiers.
4. Multi-class problems. Multi-class versions of Neural Networks, Decision Trees, Support Vector Machines and AdaBoost. Reduction of a multi-class problem to a set of binary problems. Combining classifiers.
5. Ideas for Lecture 5: Variable selection and dimension reduction methods. Basics of unsupervised learning (discovering structures in data). Software for multivariate classification.

Things to keep in mind

- 10:00am – 11:30am Panofsky; Monday – Friday, August 28 – September 1
- 1 hour for lecture, 30 mins for questions… but I welcome active participation by the audience. Ask questions as we go along, don't wait until the end.
- The content of lecture 5 is very fluid right now. If you would like to hear about a topic in machine learning, feel free to submit your request. I may be able to cover it in lecture 5. Email to: narsky at hep.caltech.edu Deadline: Wednesday night.

Textbooks

- Hastie, Tibshirani and Friedman, *The Elements of Statistical Learning,* Springer Series in Statistics 2001
- Webb, *Statistical Pattern Recognition,* 2nd ed., John Wiley & Sons 2002
- Haykin, *Neural Networks,* 2nd ed., Prentice Hall 1999
- Kuncheva, *Combining Pattern Classifiers,* John Wiley & Sons 2004

All are good and comprehensive. I found the first two especially useful.

Lecture 1

Problem of classification. Forms of empirical loss, or classification error. Training, validation and test. Cross-validation and bootstrap. Bayes (Schwartz) information criterion. How to choose a classifier.

Problem of classification

- Notation
  - input vectors x of dimension D (D coordinates)
  - output vectors y of dimension K (K categories, or classes)

$$(x, y): \quad x = \{x^{(d)}\}_{d=1}^{D} \quad y = \{y^{(k)}\}_{k=1}^{K}$$

- Build a predictor function **f(x)** that approximates **y**

$$f: \ \Re^D \mapsto \Re^K$$

- Example of classification problem
  - signal y=1 and background y=−1 (or y=0) (K=2)
  - signal y=(1,0,0); 1st bgrnd y=(0,1,0); 2nd bgrnd y=(0,0,1) (K=3)

For the most part we will deal only with K=2 categories: signal and background.

What do we need to build f(x)?

- A training sample with **N** data points:

$$\left\{\left(x_n, y_n\right)\right\}_{n=1}^{N} \quad x_n = \left\{x_n^{(d)}\right\}_{d=1}^{D} \quad y_n = \left\{y_n^{(k)}\right\}_{k=1}^{K}$$

- In the training sample all $\mathbf{y_n}$'s are known. No ambiguity is allowed.
- This is supervised learning. Teacher gives student correct answers.
- If correct class labels were unknown, this would be unsupervised learning (discovering structures in data).
- As a side note, there is a lot of similarity between classification (discrete **y** takes a few allowed values) and regression (continuous **y** spans the whole space of real numbers). Regression will not be discussed here.

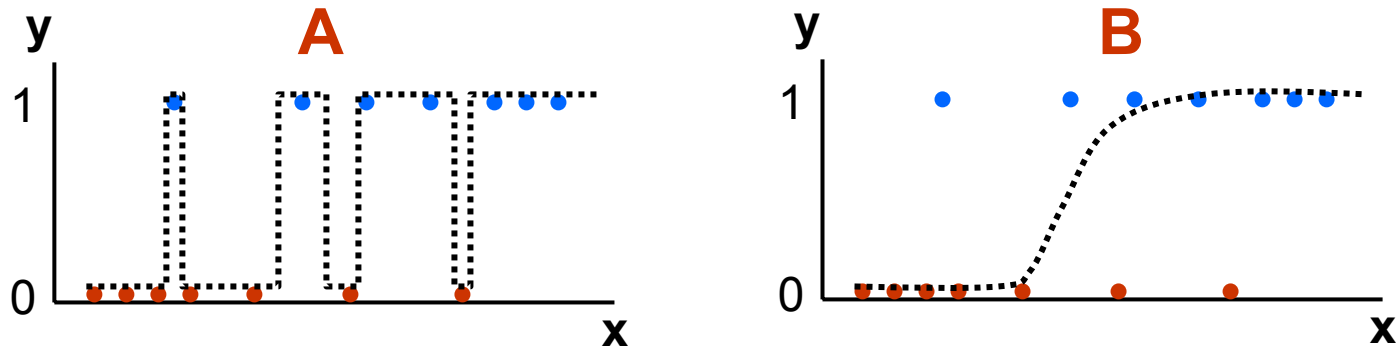
How well does f(x) approximate y?

- Define per-event *loss*, L(y,f(x))
- Find f(x) such that minimizes *empirical risk,* or *classification error*

$$R = \frac{1}{N}\sum_{n=1}^{N} L\left(y_n, f\left(x_n\right)\right)$$

- Forms of empirical loss:
  - 0-1 **L(y,f(x)) = 1 – I(y=f(x))** **f(x)=1 or −1**
  - Quadratic **L(y,f(x)) = (y-f(x))^2** **−1≤f(x)≤1**
  - Exponential **L(y,f(x)) = exp(-yf(x))** **−∞<f(x)<+∞**
  - Binomial log-likelihood **L(y,f(x)) = log[1+exp(-yf(x))]**
  - Support Vector Machine **L(y,f(x)) = [1-yf(x)]$_+$** (positive part)
- Suppose you have a training dataset {(x_n,y_n)}; n=1,…,N. If you put enough flexibility into your model f(x), you can make the empirical loss arbitrarily small on the training dataset. Is it good?

Overtraining and how one can avoid it



Model A gives a smaller classification error on the training set.

Model B is obviously better for predicting future values.

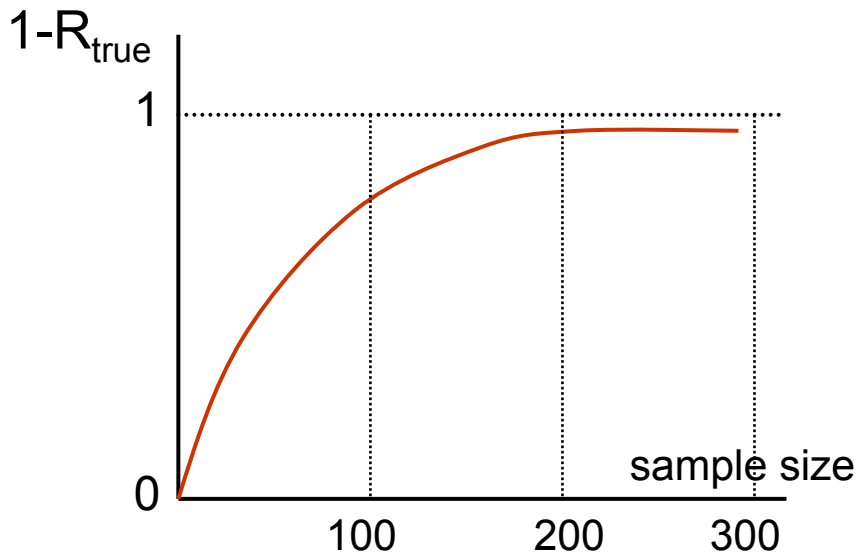
Solution: build predictor function f(x) on dataset 1 and monitor its performance on an independent dataset 2. This will let you choose the best model.

But the estimate of the model performance obtained using dataset 2 is biased because you have chosen the model with minimal classification error. Now apply your model to an independent dataset 3 to assess its performance.

Notation: 1 = training; 2 = validation; 3 = test

How well does a classifier learn on dataset of size N?

- The answer varies versus classifier and dataset. No generic rule applies.
- Often you gain a lot by including a few hundred events but not a lot by adding a few thousand more



$$R_{\text{true}} = E\left(L\left(Y, \hat{f}(X)\right)\middle| N\right)$$

expectation of classification error on random variable (X,Y) for a training sample of size N

$$R = \frac{1}{N}\sum_{n=1}^{N} L\left(y_n, f(x_n)\right)$$

estimator of classification error on a training sample of size N

Two things to remember:

- **expect $R_{\text{training}} < R_{\text{true}}$**
- **the smaller the dataset, the greater the true classification error**

How do I split data into training/validation/test sets?

- Rule of thumb: 50-25-25 (if you have lots of data)
- More sophisticated techniques (especially useful when you don't have much data):
  - Cross-validation
  - Bootstrap
  - These are good but slow. Don't use them on big datasets.

Training error typically underestimates true model error.

Validation error may underestimate true model error.

Test error is in principle an unbiased estimate of the true model error.

In principle, if the validation set is fairly large and the classifier is robust, the test classification error should be close to the validation classification error. The test stage can be omitted.

Do we really need a test dataset?

- Make sure that the size of the validation set is adequate
  - One often cares only about the classifier performance in the region of high signal purity. In this case, we effectively deal with a small subset of the validation sample.
- Make sure the classifier is robust on the training and validation sets
  - For example, Linear Discriminant Analysis and Logistic Regression are simple and robust classifiers. You likely don't need a test dataset for those.
  - Neural Network and Random Forest, however, are not so robust. You may need a test set even if your validation sample is big.
  - You can always swap training and validation sets to check if you can reproduce your classification error estimate.

Example from real life: single top at D0

- An exercise at Banff Statistical Challenges Workshop (July 2006)
- 20k signal sample (single top)
- 15k background sample (Wbb)
- Split into training-test as 50-50
  - First I tried odd lines for training and even lines for test…
    - …but turns out the input files have almost duplicate entries on neighbor lines => the classifier performance was much better than it should have been because many test events were almost identical to training events
  - Then I took the 1st half for training and the 2nd half for test
    - For 50% signal efficiency, I got 9.5% background efficiency by applying Random Forest
    - Competitive with the Neural Network used by D0
  - Then I swapped the test and training sets
    - Now for 50% signal efficiency, I got 7.5% background efficiency by applying Random Forest with the same classifier parameters
- If you worry about the classifier performance at such level, you better be really careful

Cross-validation

- Split data into M subsets
- Remove one subset, optimize your model on the kept M-1 subsets and estimate prediction error for the removed subset
- Repeat for each subset: m(n) shows indices of subsets containing this event

$$m(n):\ \{1,2,\ldots,N\}\mapsto\{1,2,\ldots,M\}$$

$$R_{CV}=\frac{1}{N}\sum_{n=1}^{N}L\left(y_n,\ f_{-m(n)}(x_n)\right)$$

- How to choose M?
  - M=N => CV error estimator is unbiased for the model error but can have large variance because essentially you are dealing with the same training set
  - M«N => CV error estimator has small variance but can have large bias for small datasets
- Rule of thumb: choose M=5-10
- More advice in the textbooks

Bootstrap

- Randomly draw N events with replacement out of the data sample of size N => one bootstrap replica
- Make B=100-200 replicas
- Optimize model on a bootstrap replica and see how well it predicts the behavior of the original sample

$$R_B = \frac{1}{BN}\sum_{b=1}^{B}\sum_{n=1}^{N} L\left(y_n, f_b^*(x_n)\right)$$

- Do a little better: To estimate error at point n, use only those replicas that do not contain point *n* ("leave-one-out" method)

$$R_B^{(1)} = \frac{1}{N}\sum_{n=1}^{N}\frac{1}{\left|C_{-n}\right|}\sum_{b\in C_{-n}} L\left(y_n, f_b^*(x_n)\right)$$

Bootstrap .632 estimator

- If you have a small sample, the "leave-one-out" estimator may significantly overestimate the true model error for a training sample of this size. (Remember the learning curve plot?)
- Fix it by using

$$R^{(0.632)} = 0.368 \cdot R_{training} + 0.632 \cdot R_B^{(1)}$$

- Why 0.632? Because

$$P(\text{point } n \in \text{replica } b) = 1 - \left(1 - \frac{1}{N}\right)^N \approx 1 - e^{-1} = 0.632$$

- …but in general derivation of this formula is not so trivial.

Quick and dirty shortcuts

- Methods
  - Akaike Information Criterion
  - Bayes (Schwartz) Information Criterion
  - Vapnik-Chernovenkis dimension
- Idea: penalize on the number of free parameters in the model
  - Two questions:
    1. What approach to use?
    2. What is the number of free parameters in the model?
  - Answer to both not obvious.

Bayes (Schwartz) Information Criterion

- Recommended for use with Max Likelihood fits
- Derived using Bayes methodology: select model with highest posterior probability
- Assuming that all models have equal prior probabilities and making some generic simplifying assumptions about the model behavior, one obtains:

$$BIC = -2 \log L(\hat{\theta} \mid x) + p \log N$$

where **L** is likelihood maximized over the parameter space

p is the number of free parameters in the likelihood

- Choose model with minimal BIC

What have we learned today?

- We defined a formal model for training a classifier f(x) on data $\{(x_n,y_n)\}$ and monitoring its performance.
- This formal model has a lot of flexibility: various forms of classification error, various classifiers, various optimization techniques.
- We discussed how to split data into training/validation/test sets and how to use more sophisticated techniques for data splitting (cross-validation and bootstrap).
- We discussed Bayes Information Criterion, a useful tool for selecting likelihood fits with varying numbers of free parameters.
- Let us step back now and think – what are we trying to accomplish?

Classification or density estimation?

- It is obvious that even with an infinite sample we can only do so well. We can't achieve zero true classification error. There has to be a limit past which there can be no improvement (e.g., *Bayes error rate*).
- Suppose we know the true multivariate densities, s(x) and b(x).
  - Do we need to know anything else to separate signal from background? – No. This is full info.
  - Then why not simply estimate the two densities? – Because it is hard! Harder than separating two classes in a data set!!!
- The described formalism for classification has been shown empirically to work well on real-world problems (limited data size, many dimensions etc).
- If we had a magic box that could give us the true signal and background densities any time we asked, there would be no need for neural networks, decision trees etc etc.

A bit of statistical decision theory

$$L(y, f(x)) = \begin{cases} 0, & y = f(x) \\ 1, & y \neq f(x) \end{cases} \qquad 0\text{-}1 \text{ loss}$$

Consider a problem with **K** classes and assume that the predictor function **f(x)** returns a discrete class label **$Y^{(k)}$, k=1,…,K.**

How can we minimize expected loss?

$$E_{(X,Y)} L\left(Y, \hat{f}(X)\right) = E_X E_{Y|X} L\left(Y, \hat{f}(X)\right) = E_X \sum_{k=1}^{K} L\left(Y^{(k)}, \hat{f}(X)\right) P\left(Y^{(k)} \middle| X\right)$$

$$\sum_{k=1}^{K} L\left(Y^{(k)}, \hat{f}(X)\right) P\left(Y^{(k)} \middle| X\right) = \sum_{\hat{f}(X) \neq Y^{(k)}} P\left(Y^{(k)} \middle| X\right) = 1 - P\left(\hat{f}(X) \middle| X\right)$$

$$\min E_{(X,Y)} L\left(Y, \hat{f}(X)\right) \Leftrightarrow \max E_X P\left(\hat{f}(X) \middle| X\right)$$

In plain English, classification error is minimized by choosing the most probable class for point X.

What is the most probable class at point X?

Suppose we have **K** classes with densities $\mathbf{g_k(x)}$ and $\mathbf{N_k}$ events in each class.

$$\text{Bayes formula}: \quad P(k|X)P(X) = P(X|k)P(k)$$

$$P(X|k) = g_k(X) \qquad P(k) = \frac{N_k}{\sum_{i=1}^{K} N_i} \qquad P(X) = \sum_{k=1}^{K} P(X|k)P(k)$$

$$P(k|X) = \frac{N_k g_k(X)}{\sum_{i=1}^{K} N_i g_i(X)}$$

A simple classification rule using class densities gives the minimal classification error **for 0-1 loss**.

This classification error is referred to as **Bayes error rate**. Represents the minimal error any model can achieve **for 0-1 loss**.

Of course, in practice we know neither $\mathbf{g_k(x)}$ nor $\mathbf{N_k}$.

What to consider when you need to choose a classifier?

- Classification power
- Training stability
  - Sensitivity of the training routine to outliers, strongly correlated inputs, types of input variables, missing data
- Scalability
  - Is performance robust in many dimensions? (curse of dimensionality)
  - How does training time scale versus dimensionality and sample size?
- Timing characteristics
  - Can afford a long training time?
  - Can afford a long time needed to classify new events after training has been completed? (Typically not an issue for physics analysis.)
- Interpretability
  - Rectangular cuts are easy to visualize. Neural Networks are not.
- Availability of software