CS216: Image Understanding (**Recognition**)

Lecture 3

Alex Berg

Course structure

Reference text: Szeliski second edition available free online https://szeliski.org/Book/

1. Intro and image formation
   Read/review Chapter 1-3 from Szeliski book (optionally look at chapter 7)
2. Low-level vision: sampling, filtering, image statistics, edges, histogram features (*will get back to this with correspondence hw!*)
3. **Classification and beginning statistical machine learning**
   Read Hastie, Tibshirani, Friedman **book** **c**hapter 2
4. Deep learning
5. Progression on structured prediction: Detection, Pose Estimation
6. Progression on structure prediction: Segmentation (homework 3)
7. Vision and language and datasets (written test)
8. Synthesis NeRF, Dalle-2, Diffusion models (homework 4)
9. Embodied vision / Video
10. Looking back at the dreams of the past for the future of computer vision

Today is maybe end of warmup for recognition in computer vision

Images, filtering/convolution, image statistics, machine learning, (geometry)

Deep learning and computer vision

Today is maybe end of warmup for recognition in computer vision

Images, filtering/convolution, image statistics, **machine learning**, (geometry)

Deep learning and computer vision

Study topics (what is *x*? How do you compute/adjust/control *x*?)

- Classification
- PAC learning
- Empirical risk minimization
- Train error, test error
- Fitting
- Loss function
- Additive loss

- Cross validation
- Generalization
- Overfitting
- Sources of error
- Optimization

Warmup to the Machine Learning warmup ☺

What is a “Nearest Neighbor Classifier”?

What is a “Nearest Neighbor Classifier”?

Nearest-neighbor methods use those observations in the training set $\mathcal{T}$ closest in input space to x to form $\hat{Y}$. Specifically, the k-nearest neighbor fit for $\hat{Y}$ is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \tag{2.8}$$

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample. Closeness implies a metric, which for the moment we assume is Euclidean distance. So, in words, we find the k observations with x_i closest to x in input space, and average their responses.

What is a “Nearest Neighbor Classifier”?

Nearest-neighbor methods use those observations in the training set $\mathcal{T}$ closest in input space to x to form $\hat{Y}$. Specifically, the k-nearest neighbor fit for $\hat{Y}$ is defined as follows:

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i, \qquad (2.8)$$

where $N_k(x)$ is the neighborhood of x defined by the k closest points x_i in the training sample. Closeness implies a metric, which for the moment we assume is Euclidean distance. So, in words, we find the k observations with x_i closest to x in input space, and average their responses.

Classifier:

$C(x) = 1$ iff $\hat{Y}(x) > 0$

Where 0 is a threshold. This implies the y_i *are in* $\{-1, 1\}$.

What does the decision boundary for a nearest neighbor classifier look like if $k >= n$ the number of training samples.

What does the decision boundary for a nearest neighbor classifier look like if *k*>=*n* the number of training samples.

No decision boundary.

$$\hat{Y}(x) = \frac{1}{k} \sum_{x_i \in N_k(x)} y_i,$$

constant, independent of *x*,
mean of all training labels, y_i

Sources of expected error (generalization)?

Sources of expected error (generalization)?

The k-nearest-neighbor regression fit $\hat{f}_k(x_0)$ usefully illustrates the competing forces that effect the predictive ability of such approximations. Suppose the data arise from a model $Y = f(X) + \varepsilon$, with $\mathrm{E}(\varepsilon) = 0$ and $\mathrm{Var}(\varepsilon) = \sigma^2$. For simplicity here we assume that the values of x_i in the sample are fixed in advance (nonrandom). The expected prediction error at x_0, also known as *test* or *generalization* error, can be decomposed:

$$\begin{aligned}
\mathrm{EPE}_k(x_0) &= \mathrm{E}[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\
&= \sigma^2 + [\mathrm{Bias}^2(\hat{f}_k(x_0)) + \mathrm{Var}_{\mathcal{T}}(\hat{f}_k(x_0))] \qquad (2.46) \\
&= \sigma^2 + \left[f(x_0) - \frac{1}{k}\sum_{\ell=1}^{k} f(x_{(\ell)})\right]^2 + \frac{\sigma^2}{k}. \qquad (2.47)
\end{aligned}$$

Bias, how close is the estimate to $f(x_0)$?

Variance, in estimate

Variance of test samples (irreducible)

The subscripts in parentheses (ℓ) indicate the sequence of nearest neighbors to x_0.

There are three terms in this expression. The first term σ^2 is the *irreducible* error—the variance of the new test target—and is beyond our control, even if we know the true $f(x_0)$.

The second and third terms are under our control, and make up the *mean squared error* of $\hat{f}_k(x_0)$ in estimating $f(x_0)$, which is broken down into a bias component and a variance component. The bias term is the squared difference between the true mean $f(x_0)$ and the expected value of the estimate—$[\mathrm{E}_{\mathcal{T}}(\hat{f}_k(x_0)) - f(x_0)]^2$—where the expectation averages the randomness in the training data. This term will most likely increase with k, if the true function is reasonably smooth. For small k the few closest neighbors will have values $f(x_{(\ell)})$ close to $f(x_0)$, so their average should be close to $f(x_0)$. As k grows, the neighbors are further away, and then anything can happen.

The variance term is simply the variance of an average here, and decreases as the inverse of k. So as k varies, there is a *bias–variance tradeoff*.

What happens as the number of training samples increases?

$$
\begin{aligned}
\mathrm{EPE}_k(x_0) &= \mathrm{E}[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\
&= \sigma^2 + [\mathrm{Bias}^2(\hat{f}_k(x_0)) + \mathrm{Var}_{\mathcal{T}}(\hat{f}_k(x_0))] \qquad (2.46) \\
&= \sigma^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^{k} f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k}. \qquad (2.47)
\end{aligned}
$$

What happens as the number of training samples increases?

$$\begin{aligned}
\mathrm{EPE}_k(x_0) &= \mathrm{E}[(Y - \hat{f}_k(x_0))^2 | X = x_0] \\
&= \sigma^2 + [\mathrm{Bias}^2(\hat{f}_k(x_0)) + \mathrm{Var}_{\mathcal{T}}(\hat{f}_k(x_0))] \qquad (2.46) \\
&= \sigma^2 + \left[f(x_0) - \frac{1}{k} \sum_{\ell=1}^{k} f(x_{(\ell)}) \right]^2 + \frac{\sigma^2}{k}. \qquad (2.47)
\end{aligned}$$

Smaller?! Nearest neighbors are closer.

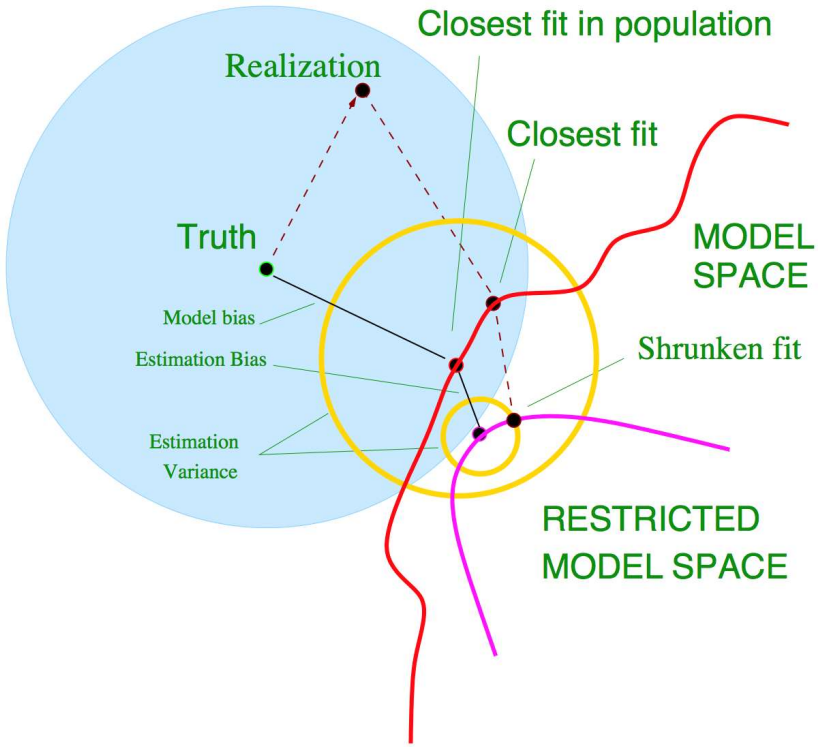
Bias, how close is the estimate to $f(x_0)$?

Variance, in estimate Same!

Variance of test samples (irreducible)

Same!

Real Life…



Hastie et al, Elements of Statistical Learning, 2008

Classification (making a choice)

http://www.flickr.com/photos/sgcallawayimages/3306849049/

Image processing and computer vision → Information → Choice

e.g. **Dog** or **not dog**

What is there to recognize?

http://www.flickr.com/photos/sgcallawayimages/3306849049/

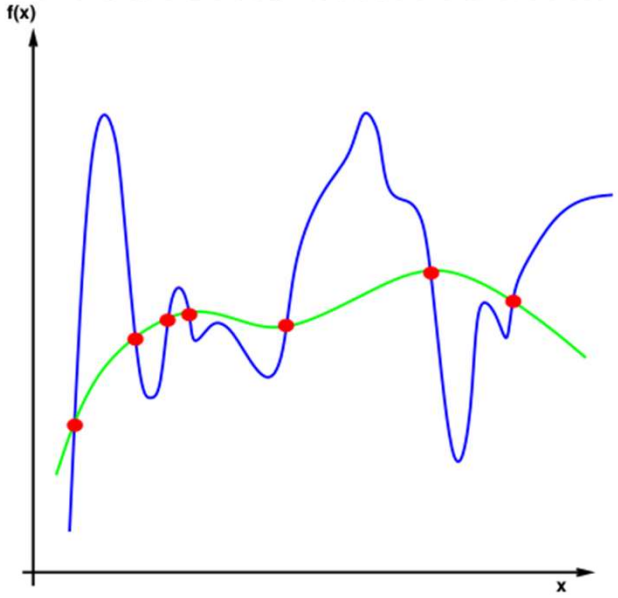
Increasing structural complexity

		What is the choice?
Dog	Single label	One of labels
Dog, Tree, Fence, Leaves, Autumn	Multiple labels	Subset of labels
	Localization/ Parsing	One label per pixel (many pixels)
"Happy shaggy Airedale poses in the autumn forest."	Description	Sequence of labels (words)

What is machine learning?

- [Tom Mitchell] Study of algorithms that improve their performance, P, at some task, T, with experience, E. <P,T,E>
- [Wikipedia] a scientific discipline that explores the construction and study of algorithms that can learn from data.
- This is like the scientific method:
  - observe some data
  - make some hypotheses (choose model)
  - perform experiments (fit and evaluate model)
  - (profit!)
- Machine learning focuses on the mathematical formulations and computation

Why is this hard?
Problems with learning a function from data



From Poggio & Smale 2003

(training) data in red

Problem: predict y = f(x)

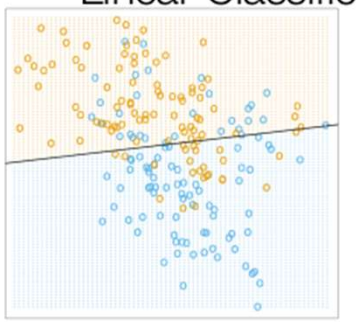
Possible f's shown in blue and green

Need something to limit possibilities!

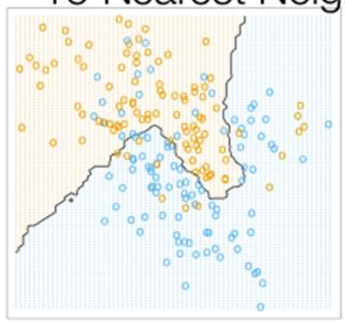
(possibilities are the hypothesis space)

Linear classifier vs nearest neighbor

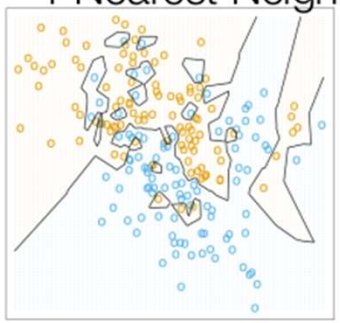
Linear Classifier



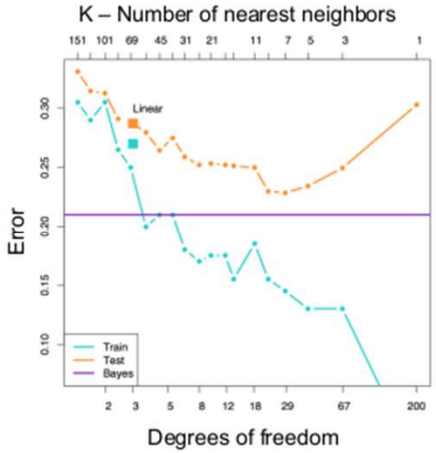
15 Nearest Neighbor



1 Nearest Neighbor



From Hastie, Tibshirani, Friedman Book



(training) data in wheat/blue

Problem: predict $y=f(x_1,x_2)>0$

Possibilities for $f(x_1,x_2)=0$ shown in black

Learning/fitting is a process…

Estimating the probability that a tossed coin comes up heads…

$X_i \in \{0,1\}$ The i'th coin toss

$$\widehat{\theta}_n(X^n) = \frac{1}{n}\sum_{i=1}^{n} X_i$$

Computation

Estimator based on n tosses

$$\mathsf{G}_{n,\varepsilon} \triangleq \left\{x^n \in \{0,1\}^n : |\widehat{\theta}_n(x^n) - \theta| \le \varepsilon\right\}$$ Estimate is within epsilon

$$\mathsf{B}_{n,\varepsilon} \triangleq \left\{x^n \in \{0,1\}^n : |\widehat{\theta}_n(x^n) - \theta| > \varepsilon\right\}$$ Estimate is not within epsilon

$$\mathbb{P}_\theta^n(\mathsf{B}_{n,\varepsilon}) \equiv \mathbb{P}_\theta^n\left(|\widehat{\theta}_n(X^n) - \theta| > \varepsilon\right) \le 2e^{-2n\varepsilon^2}$$

Probability of being bad is inversely proportional to the number of samples…
(the underlying computation is an example of a tail bound)

From Raginsky notes

Bad news for more dimensions

- Estimating a single variable (e.g. bias of a coin) within a couple of percent might take ~100 samples… *(check this!)*
- Estimating a function (e.g. the probability of being in class 1) for an *n-dimensional* binary variable requires estimating ~2^n variables. $100x2^n$ can be large.
- *In most cases our game will be finding ways to restrict the possibilities for the function and to focus on the decision boundary (where f(x)=0) instead of f(x) itself.*

What happens in high dimensions?

What happens in high dimensions?

Almost everything is far away!

Everything is almost equally far away!

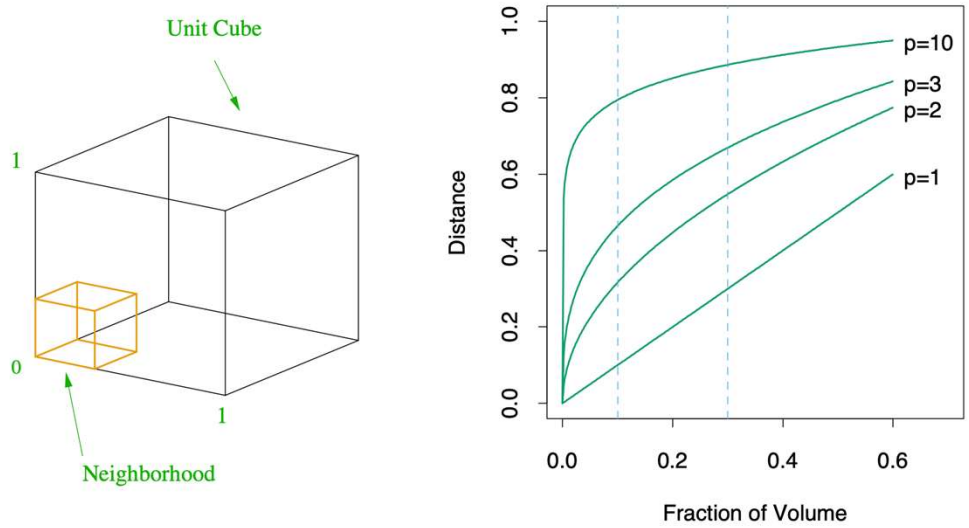


FIGURE 2.6. *The curse of dimensionality is well illustrated by a subcubical neighborhood for uniform data in a unit cube. The figure on the right shows the side-length of the subcube needed to capture a fraction r of the volume of the data, for different dimensions p. In ten dimensions we need to cover* 80% *of the range of each coordinate to capture* 10% *of the data.*

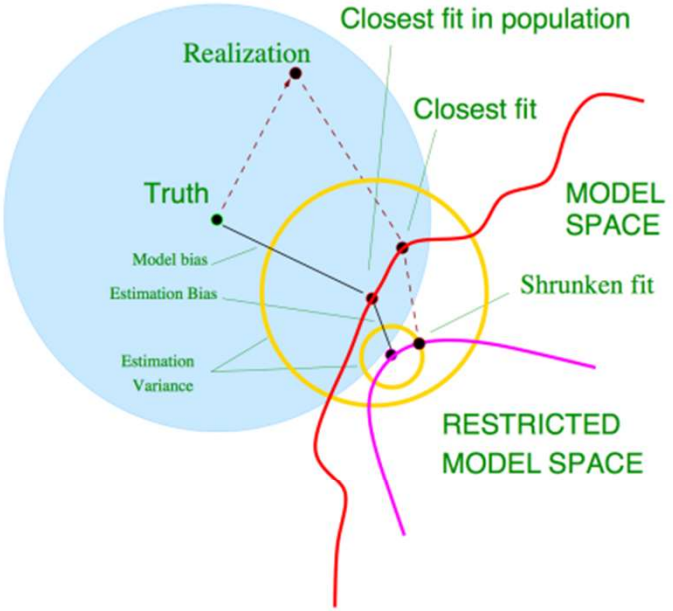
PAC Learning

Definition 1 *(The PAC Model, Preliminary Definition) Let $\mathcal{C}$ be a concept class over X. We say that $\mathcal{C}$ is* **PAC learnable** *if there exists an algorithm L with the following property: for every concept $c \in \mathcal{C}$, for every distribution $\mathcal{D}$ on X, and for all $0 < \epsilon < 1/2$ and $0 < \delta < 1/2$, if L is given access to $EX(c, \mathcal{D})$ and inputs ϵ and δ, then with probability at least $1 - \delta$, L outputs a hypothesis concept $h \in \mathcal{C}$ satisfying error$(h) \leq \epsilon$. This probability is taken over the random examples drawn by calls to $EX(c, \mathcal{D})$, and any internal randomization of L.*

If L runs in time polynomial in $1/\epsilon$ and $1/\delta$, we say that $\mathcal{C}$ is **efficiently** *PAC learnable. We will sometimes refer to the input ϵ as the* **error parameter**, *and the input δ as the* **confidence parameter**.

From Kearns and Vazirani, An introduction to computational learning theory

Real Life…



Seldom know anything beyond the measured samples and the restricted space. The rest are unknown.

Hastie et al, Elements of Statistical Learning, 2008

What is “supervised learning”?

What is "supervised learning"?

2.6.2 Supervised Learning

Before we launch into more statistically oriented jargon, we present the function-fitting paradigm from a machine learning point of view. Suppose for simplicity that the errors are additive and that the model $Y = f(X) + \varepsilon$ is a reasonable assumption. Supervised learning attempts to learn f by example through a *teacher*. One observes the system under study, both the inputs and outputs, and assembles a *training* set of observations $\mathcal{T} = (x_i, y_i)$, $i = 1, \ldots, N$. The observed input values to the system x_i are also fed into an artificial system, known as a learning algorithm (usually a computer program), which also produces outputs $\hat{f}(x_i)$ in response to the inputs. The learning algorithm has the property that it can modify its input/output relationship $\hat{f}$ in response to differences $y_i - \hat{f}(x_i)$ between the original and generated outputs. This process is known as *learning by example*. Upon completion of the learning process the hope is that the artificial and real outputs will be close enough to be useful for all sets of inputs likely to be encountered in practice.

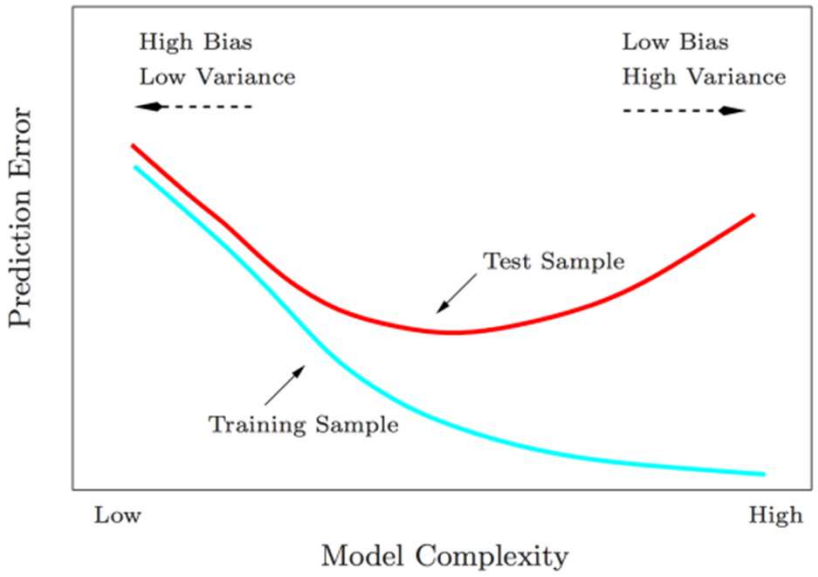
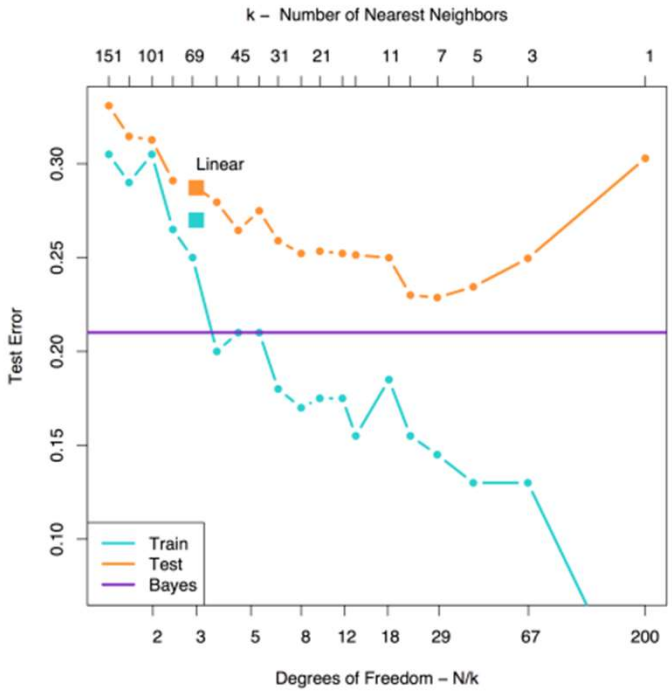
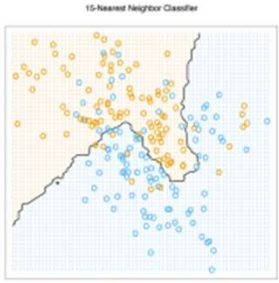
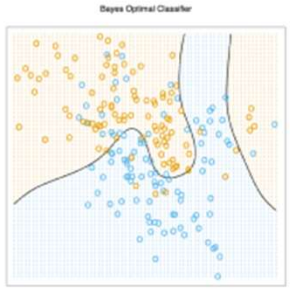


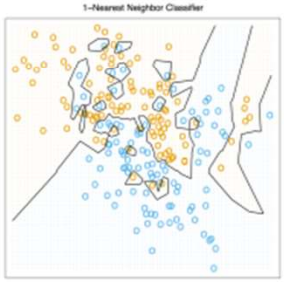
FIGURE 2.11. *Test and training error as a function of model complexity.*

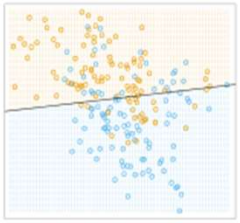
Hastie et al, Elements of Statistical Learning, 2008





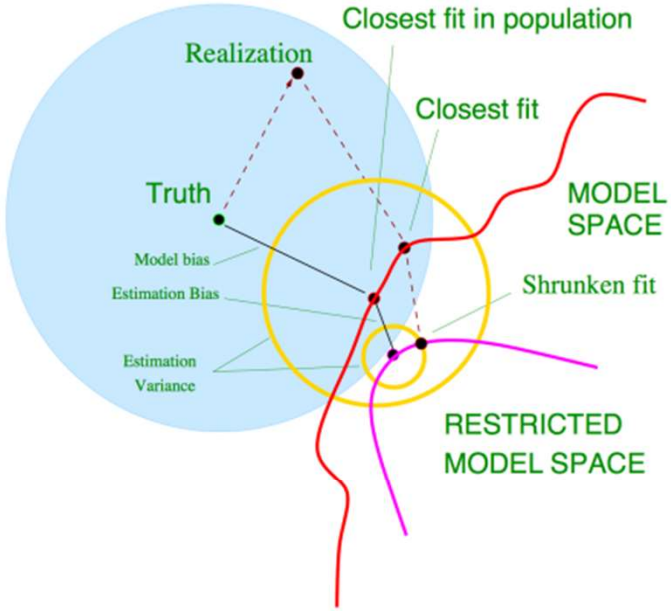




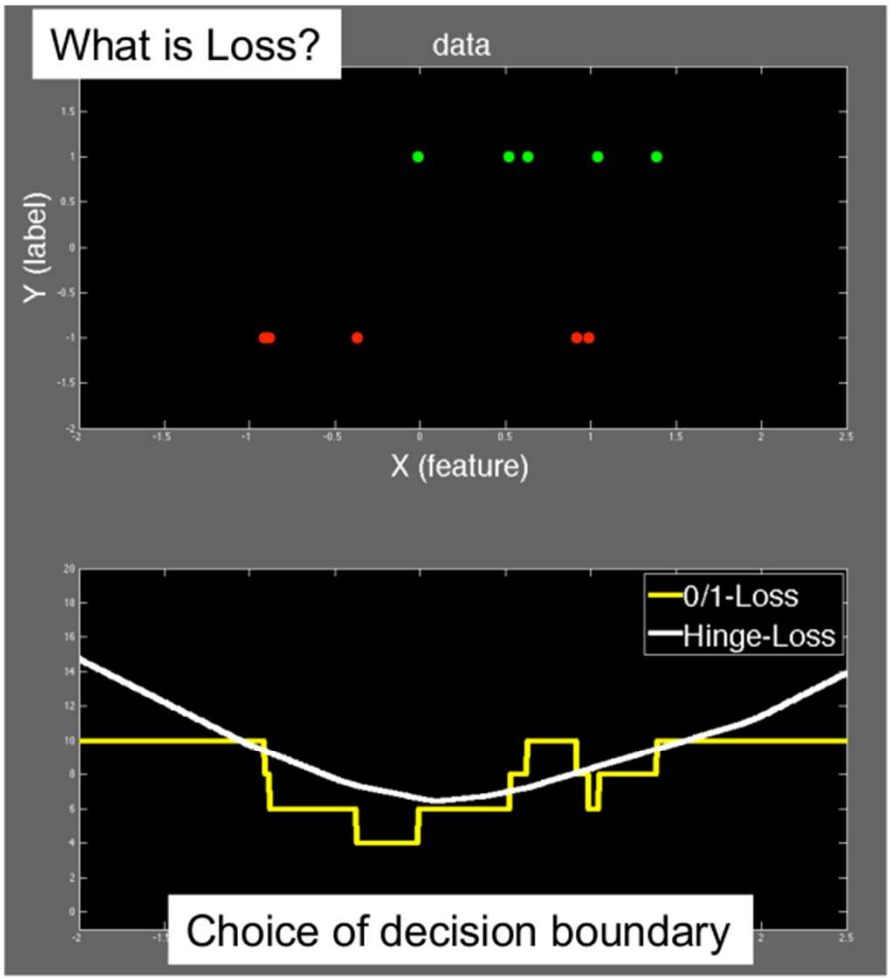


Hastie et al, Elements of Statistical Learning, 2008

Real Life…



Hastie et al, Elements of Statistical Learning, 2008



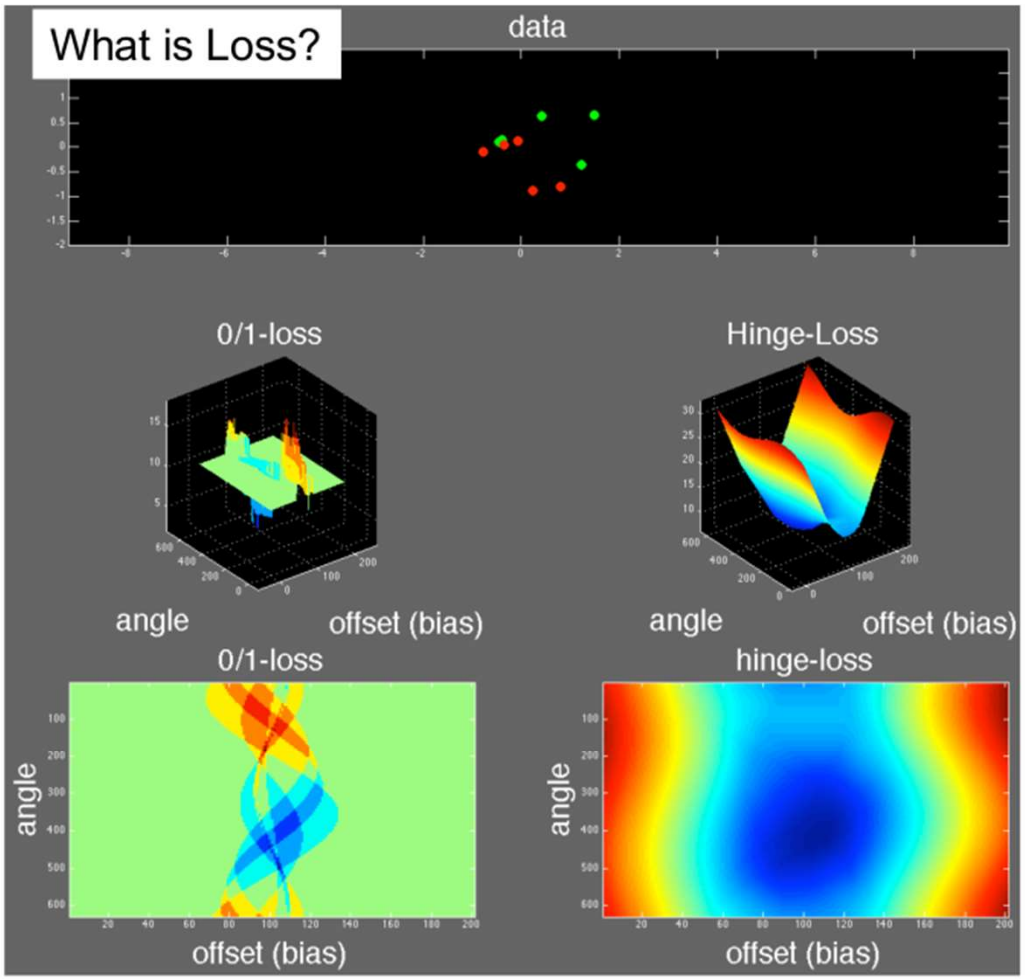
1 D Data with labels -1,1

(indicated by color and y-axis position)

We want red to the left of decision boundary!

0/1 Loss (how many correct decisions on data)
and Hinge Loss (one possible continuous approximation to this)

(y-axis is loss,
x-axis is decision boundary)



2 D Data with labels -1,1 (indicated by color)

0/1 Loss and Hinge Loss Indicated by color

y-location is angle of decision boundary

x-location is Offset of decision boundary

2.4 Statistical Decision Theory

In this section we develop a small amount of theory that provides a framework for developing models such as those discussed informally so far. We first consider the case of a quantitative output, and place ourselves in the world of random variables and probability spaces. Let $X \in \mathbb{R}^p$ denote a real valued random input vector, and $Y \in \mathbb{R}$ a real valued random output variable, with joint distribution $\Pr(X, Y)$. We seek a function $f(X)$ for predicting Y given values of the input X. This theory requires a *loss function* $L(Y, f(X))$ for penalizing errors in prediction, and by far the most common and convenient is *squared error loss*: $L(Y, f(X)) = (Y - f(X))^2$. This leads us to a criterion for choosing f,

$$\begin{aligned} \mathrm{EPE}(f) &= \mathrm{E}(Y - f(X))^2 && (2.9) \\ &= \int [y - f(x)]^2 \Pr(dx, dy), && (2.10) \end{aligned}$$

the expected (squared) prediction error . By conditioning[1] on X, we can write EPE as

$$\mathrm{EPE}(f) = \mathrm{E}_X \mathrm{E}_{Y|X} \left([Y - f(X)]^2 | X \right) \quad (2.11)$$

and we see that it suffices to minimize EPE pointwise:

$$f(x) = \mathrm{argmin}_c \mathrm{E}_{Y|X} \left([Y - c]^2 | X = x \right). \quad (2.12)$$

The solution is

$$f(x) = \mathrm{E}(Y | X = x), \quad (2.13)$$

the conditional expectation, also known as the *regression* function. Thus the best prediction of Y at any point $X = x$ is the conditional mean, when best is measured by average squared error.

The nearest-neighbor methods attempt to directly implement this recipe using the training data. At each point x, we might ask for the average of all those y_is with input $x_i = x$. Since there is typically at most one observation at any point x, we settle for

$$\hat{f}(x) = \mathrm{Ave}(y_i | x_i \in N_k(x)), \quad (2.14)$$

where "Ave" denotes average, and $N_k(x)$ is the neighborhood containing the k points in $\mathcal{T}$ closest to x. Two approximations are happening here:

- expectation is approximated by averaging over sample data;
- conditioning at a point is relaxed to conditioning on some region "close" to the target point.

For large training sample size N, the points in the neighborhood are likely to be close to x, and as k gets large the average will get more stable. In fact, under mild regularity conditions on the joint probability distribution $\Pr(X, Y)$, one can show that as $N, k \to \infty$ such that $k/N \to 0$, $\hat{f}(x) \to \mathrm{E}(Y | X = x)$. In light of this, why look further, since it seems we have a universal approximator? We often do not have very large samples. If the linear or some more structured model is appropriate, then we can usually get a more stable estimate than k-nearest neighbors, although such knowledge has to be learned from the data as well. There are other problems though, sometimes disastrous. In Section 2.5 we see that as the dimension p gets large, so does the metric size of the k-nearest neighborhood. So settling for nearest neighborhood as a surrogate for conditioning will fail us miserably. The convergence above still holds, but the *rate* of convergence decreases as the dimension increases.

How does linear regression fit into this framework? The simplest explanation is that one assumes that the regression function $f(x)$ is approximately linear in its arguments:

$$f(x) \approx x^T \beta. \quad (2.15)$$

This is a model-based approach—we specify a model for the regression function. Plugging this linear model for $f(x)$ into EPE (2.9) and differentiating we can solve for β theoretically:

$$\beta = [\mathrm{E}(XX^T)]^{-1} \mathrm{E}(XY). \quad (2.16)$$

Note we have *not* conditioned on X; rather we have used our knowledge of the functional relationship to *pool* over values of X. The least squares solution (2.6) amounts to replacing the expectation in (2.16) by averages over the training data.

So both k-nearest neighbors and least squares end up approximating conditional expectations by averages. But they differ dramatically in terms of model assumptions:

[1] Conditioning here amounts to factoring the joint density $\Pr(X, Y) = \Pr(Y|X)\Pr(X)$ where $\Pr(Y|X) = \Pr(Y, X)/\Pr(X)$, and splitting up the bivariate integral accordingly.

So both k-nearest neighbors and least squares end up approximating conditional expectations by averages. But they differ dramatically in terms of model assumptions:

- Least squares assumes $f(x)$ is well approximated by a globally linear function.
- k-nearest neighbors assumes $f(x)$ is well approximated by a locally constant function.

Although the latter seems more palatable, we have already seen that we may pay a price for this flexibility.

Many of the more modern techniques described in this book are model based, although far more flexible than the rigid linear model. For example, additive models assume that

$$f(X) = \sum_{j=1}^{p} f_j(X_j). \tag{2.17}$$

This retains the additivity of the linear model, but each coordinate function f_j is arbitrary. It turns out that the optimal estimate for the additive model uses techniques such as k-nearest neighbors to approximate *univariate* conditional expectations *simultaneously* for each of the coordinate functions. Thus the problems of estimating a conditional expectation in high dimensions are swept away in this case by imposing some (often unrealistic) model assumptions, in this case additivity.

Here the data $(x_i, y_i)_{i=1}^m$ is supposed random, so that there is an unknown probability measure ρ on the product space $X \times Y$ from which the data is drawn.

This measure ρ defines a function

$$f_\rho : X \to Y \tag{8}$$

satisfying $f_\rho(x) = \int y d\rho_x$, where ρ_x is the conditional measure on $x \times Y$.

From this construction f_ρ can be said to be the true input-output function reflecting the environment which produces the data. Thus a measurement of the error of f is

$$\int_X (f - f_\rho)^2 d\rho_X \tag{9}$$

where ρ_X is the measure on X induced by ρ (sometimes called the marginal measure).

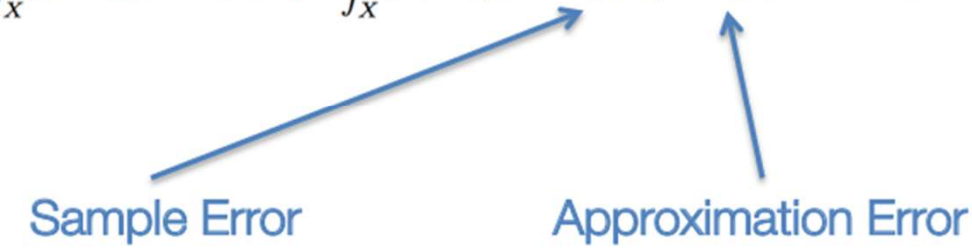
The goal of learning theory might be said to "find" f minimizing this error.

Starting from the data $(x_i, y_i)_{i=1}^m = z$ one may minimize $\frac{1}{m}\sum_{i=1}^m (f(x_i) - y_i)^2$ over $f \in \mathcal{H}$ to obtain a unique hypothesis $f_z : X \to Y$. This f_z is called the empirical optimum and we may focus on the problem of estimating

$$\int_X (f_z - f_\rho)^2 d\rho_X \tag{11}$$

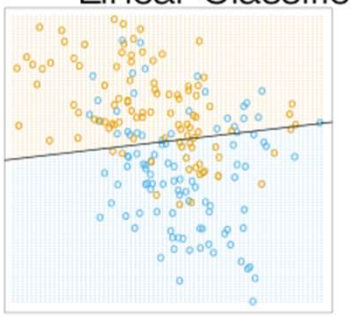
It is useful towards this end to break the problem into steps by defining a "true optimum" $f_{\mathcal{H}}$ relative to $\mathcal{H}$, by taking the minimum over $\mathcal{H}$ of $\int_X (f - f_\rho)^2$. Thus we may exhibit

$$\int_X (f_z - f_\rho)^2 = S(z, \mathcal{H}) + \int_X (f_{\mathcal{H}} - f_\rho)^2 = S(z, \mathcal{H}) + A(\mathcal{H}) \tag{12}$$

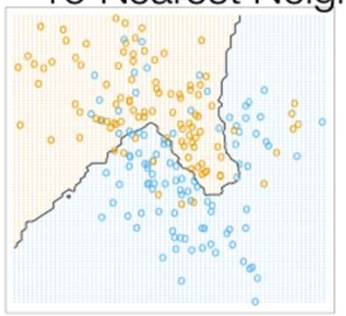


Identify Sample and Approximation Error in this setting…

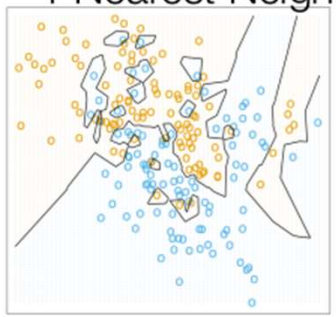
Linear Classifier



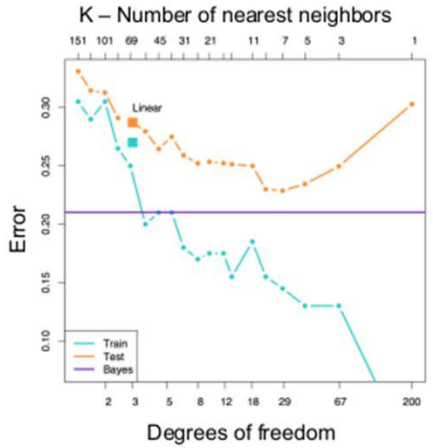
15 Nearest Neighbor



1 Nearest Neighbor



From Hastie, Tibshirani, Friedman Book



(training) data in wheat/blue

Problem: predict $y=f(x_1,x_2)>0$

Possibilities for $f(x_1,x_2)=0$ shown in black

Convolution is a linear transform!

What can it do?

- Linear decision boundaries in space of patches
- (Aside on invertibility)

What can't it do?

- Non-linear decisions on spaces of patches

How do we get around this?

- Deep learning

Work on white board for a bit, transition to deep learning…