Kernel Methods and Support Vector Machines

CSCI 4360/6360 Data Science II

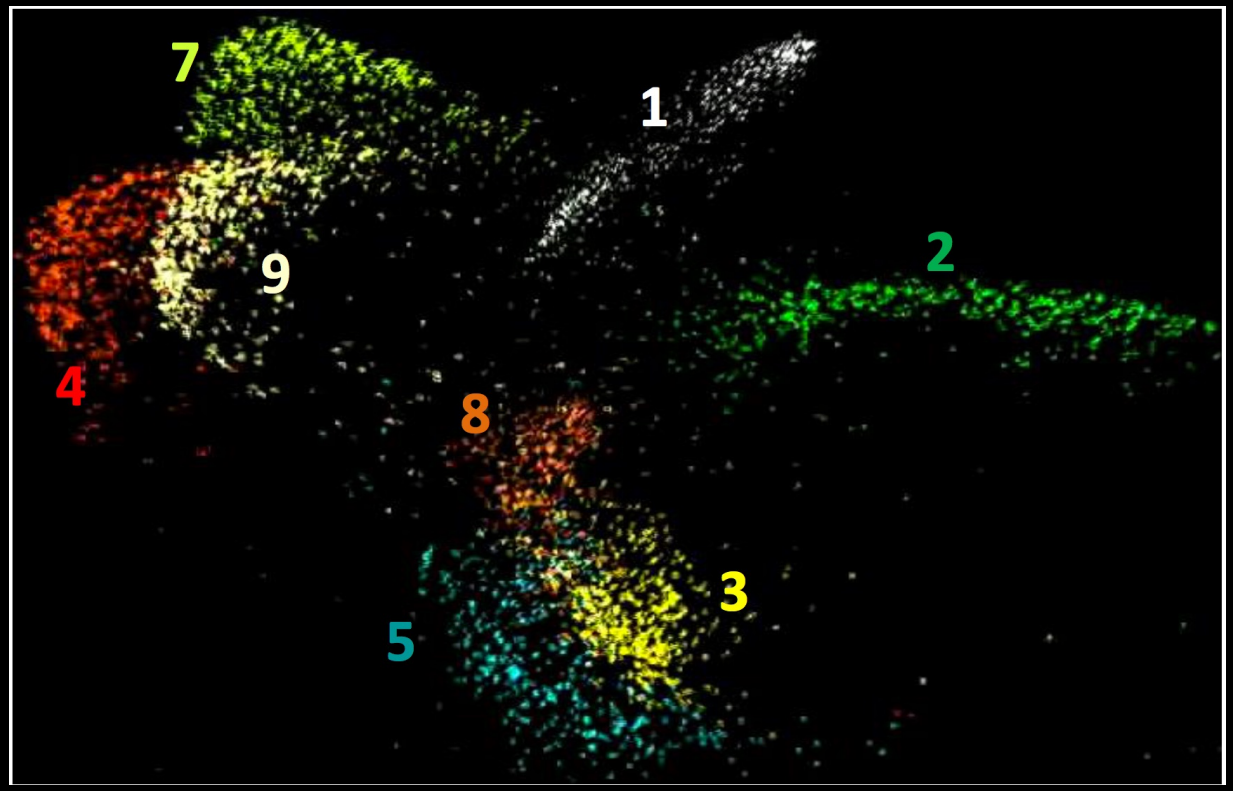
Parametric Statistics

- Assume some functional form (Gaussian, Bernoulli, Multinomial, logistic, linear) for
  - $P(X_i|Y)$ and $P(Y)$ as in Naïve Bayes
  - $P(Y|X)$ as in Logistic Regression
- Estimate parameters (μ, σ^2, θ, w, β) using MLE/MAP
  - Plug-n-chug

- **Advantages**: need relatively few data points to learn parameters
- **Drawbacks**: Strong assumptions rarely satisfied in practice

Embeddings

- Again!
- MNIST, projected into 2D embedding space
- What distribution do these follow?
- **Highly nonlinear**



Nonparametric Statistics

- Typically very few, if any, distributional assumptions
- Usually requires more data
- Let number of parameters scale with the data

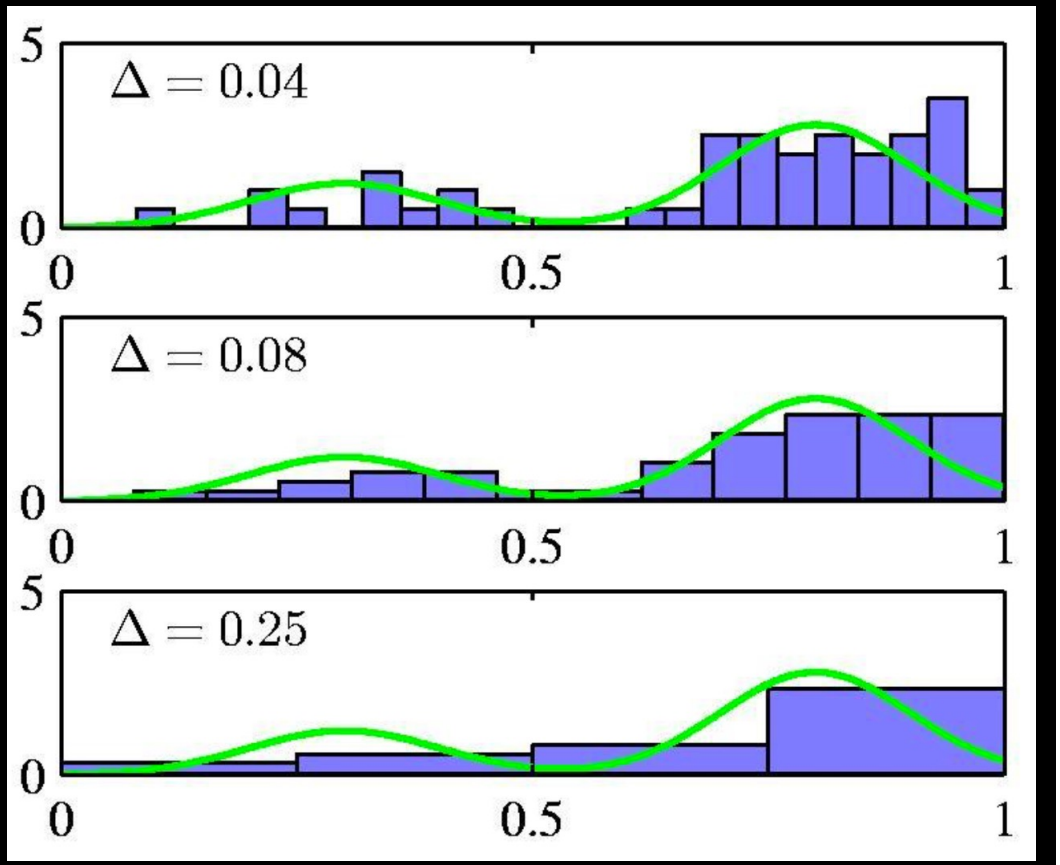
- **Today**
  - Kernel density estimation
  - K-nearest neighbors classification
  - Kernel regression
  - Support Vector Machines (SVMs) → not *exactly* nonparametric, but kernels are involved!

Density Estimation

- You've done this before—histograms!
- Partition feature space into distinct bins with specified widths and count number of observations n_i in each bin

$$\hat{p}(x) = \frac{n_i}{n\Delta_i} \mathbf{1}_{x \in \mathrm{Bin}_i}$$

- Same width is often used for all bins
- Bin width acts as **smoothing parameter**

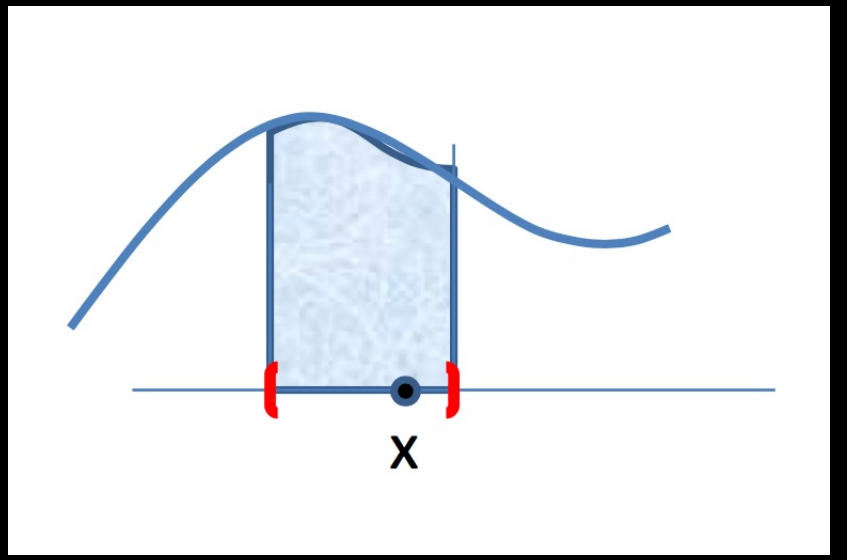


Effect of Δ

- # of bins = 1/Δ

$$\hat{p}(x) = \frac{n_i}{n\Delta} \mathbf{1}_{x \in \text{Bin}_i}$$

$$\hat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} \mathbf{1}_{X_j \in \text{Bin}_x}}{n}$$



- Bias of histogram density estimate

$$\mathbb{E}\left[\hat{p}(x)\right] = \frac{1}{\Delta} P(X \in \text{Bin}_x) = \frac{1}{\Delta} \int_{z \in \text{Bin}_x} p(z)dz \approx \frac{p(x)\Delta}{\Delta}$$

$$= p$$

Assuming density is roughly constant in each bin (roughly true, if Δ is small)

Bias-Variance Trade-off

- Choice of # of bins
  - if Δ is small
  - if Δ is large

$$\mathbb{E}\left[\hat{p}(x)\right] \approx p(x)$$
$$\mathbb{E}\left[\hat{p}(x)\right] \approx \hat{p}(x)$$

p(x) approximately constant per bin

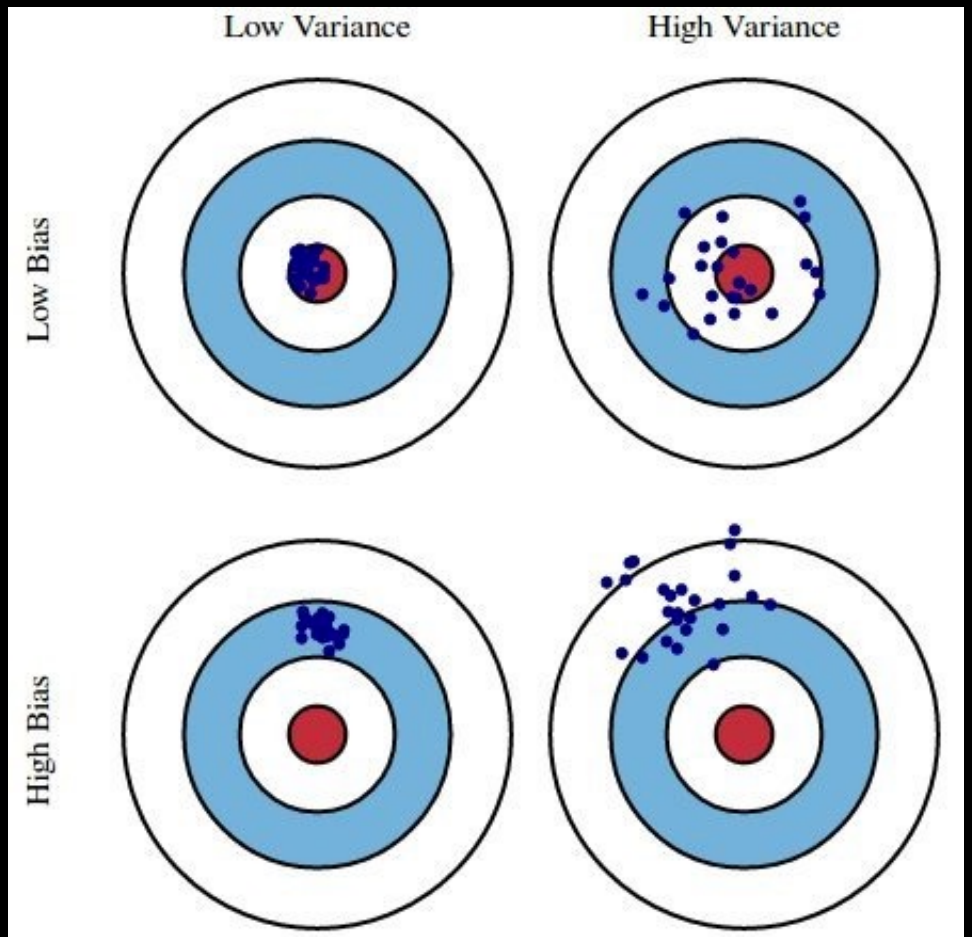
More data per bin stabilizes estimate

- **Bias**: how close is mean of estimate to the truth
- **Variance**: how much does estimate vary around the mean

Small Δ, large #bins ⟺ "**Small bias**, **Large variance**"

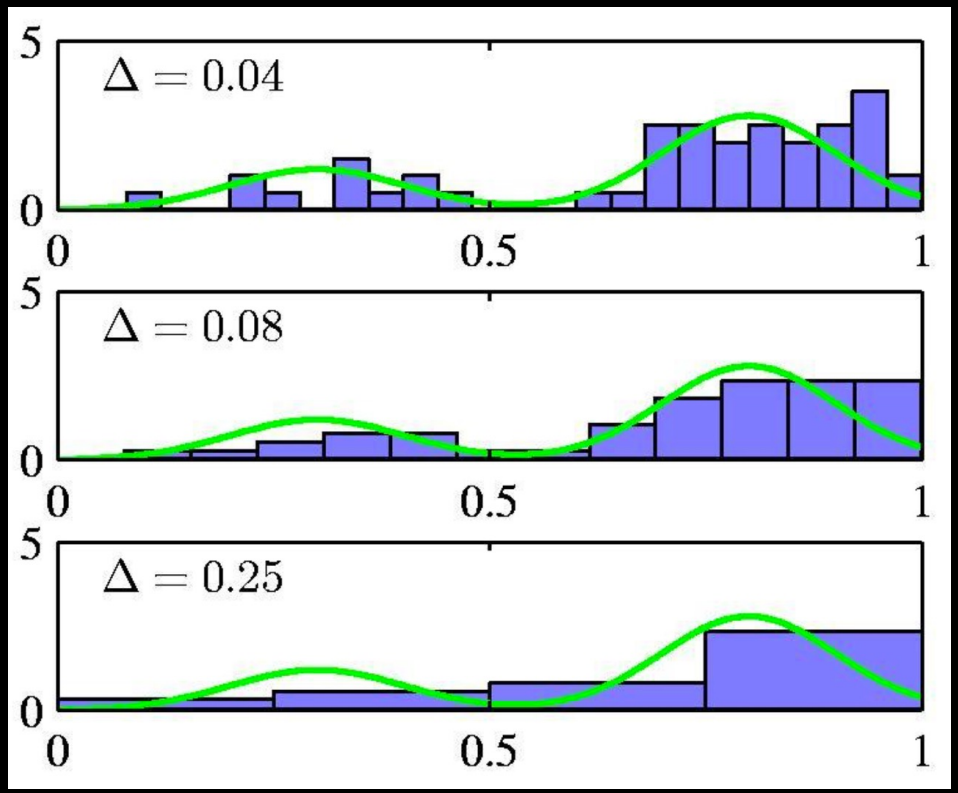
Large Δ, small #bins ⟺ "**Large bias**, **Small variance**"

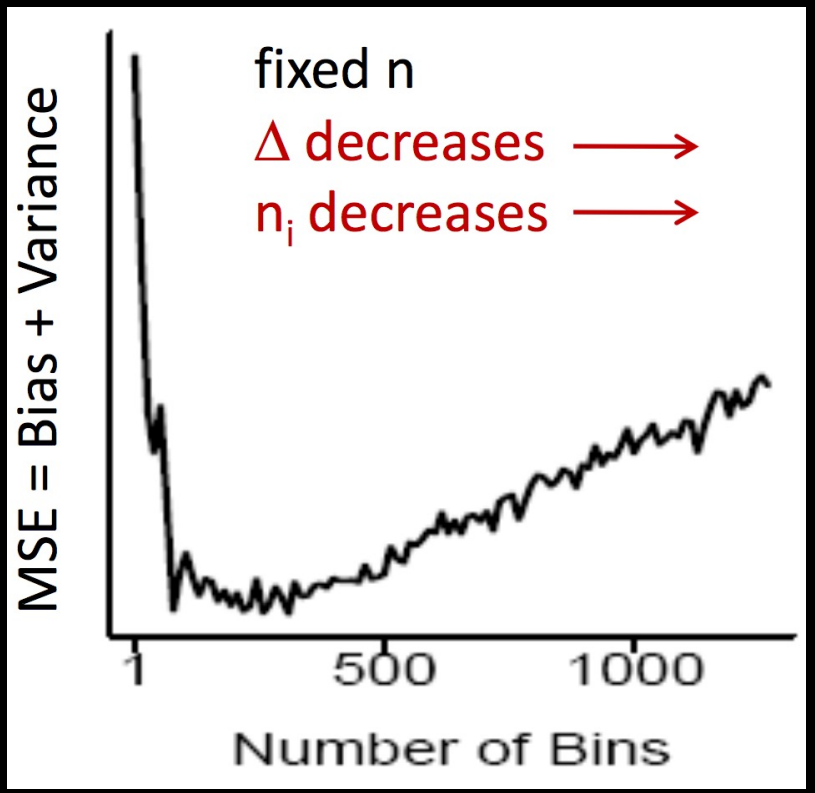
Bias-Variance Trade-off



Choice of number of bins

$$\hat{p}(x) = \frac{n_i}{n\Delta} \mathbf{1}_{x \in \mathrm{Bin}_i}$$





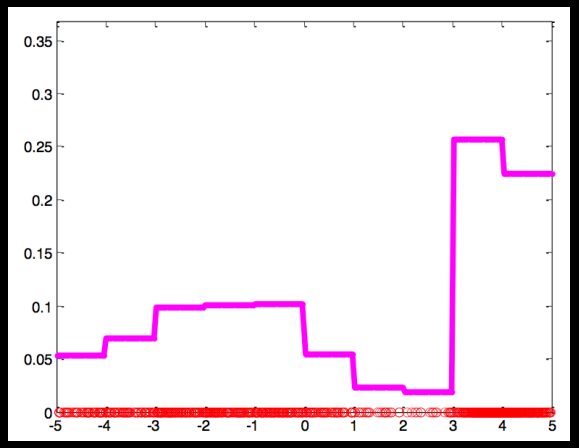
Kernel Density Estimation

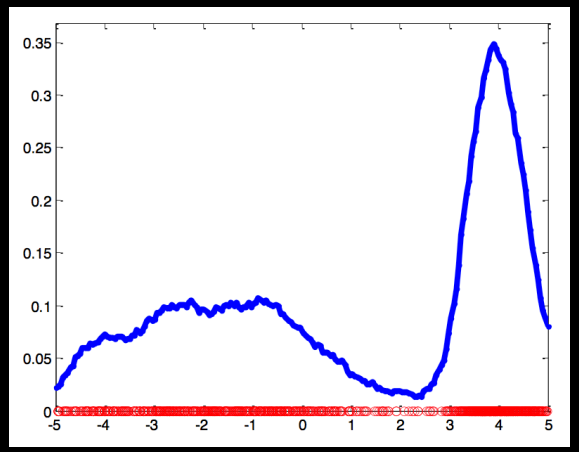
- Histograms are “blocky” estimates

$$\hat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} \mathbf{1}_{X_j \in \mathrm{Bin}_x}}{n}$$

- Kernel density estimate, aka “Parzen / moving window” method

$$\hat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} \mathbf{1}_{||X_j - x|| \leq \Delta}}{n}$$





Kernel Density Estimation

- More generally:

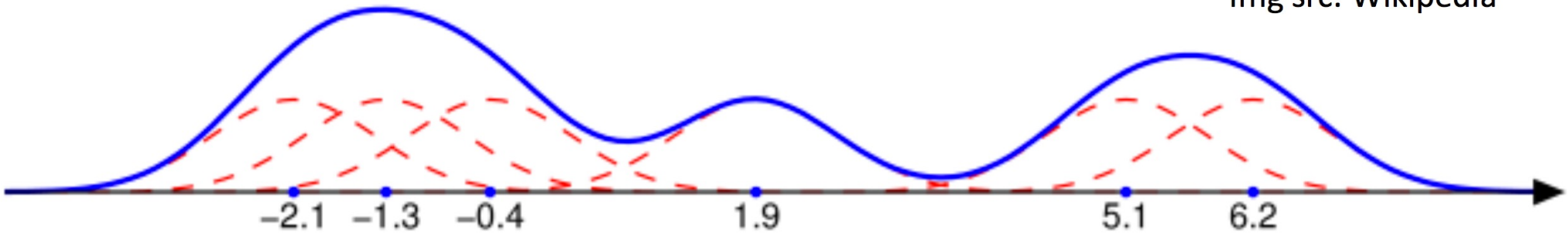
$$\hat{p}(x) = \frac{1}{\Delta} \frac{\sum_{j=1}^{n} K\left(\frac{X_j - x}{\Delta}\right)}{n}$$

- *K* is the kernel function
  - Much like kernels in Kernel PCA or SVMs: model a relationship between two data points
- Embodies any number of possible kernel functions

Kernel Density Estimation

- Places small “bumps” at each data point, determined by *K*
- Estimator itself consists of a [normalized] “sum of bumps”

Img src: Wikipedia



- Where points are denser, density estimate will be higher

Kernels

- Any function that satisfies

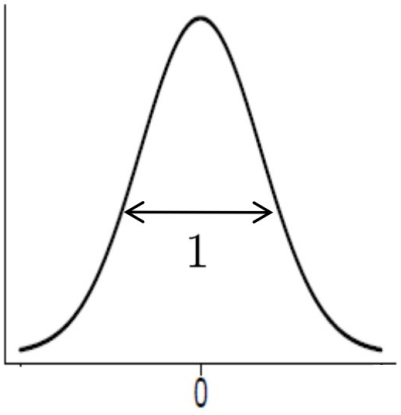
$$K(x) \geq 0$$

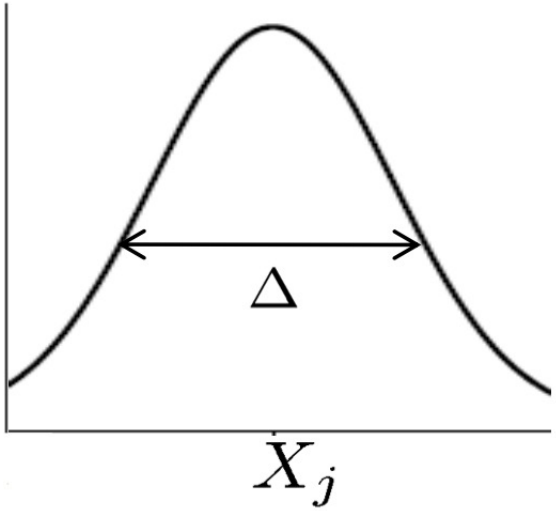
$$\int K(x)dx = 1$$

- SciPy has a **ton**
  - See “signal.get_window”

Gaussian kernel :

$$K(x) = \frac{1}{\sqrt{2\pi}} e^{-x^2/2}$$





Infinite support: need all points to compute estimate. **But quite popular.**

Kernels

- Deep theory associated with kernels and kernel functions
- Touched on in Kernel PCA lecture
- Foundational to Support Vector Machines and Deep Neural Networks

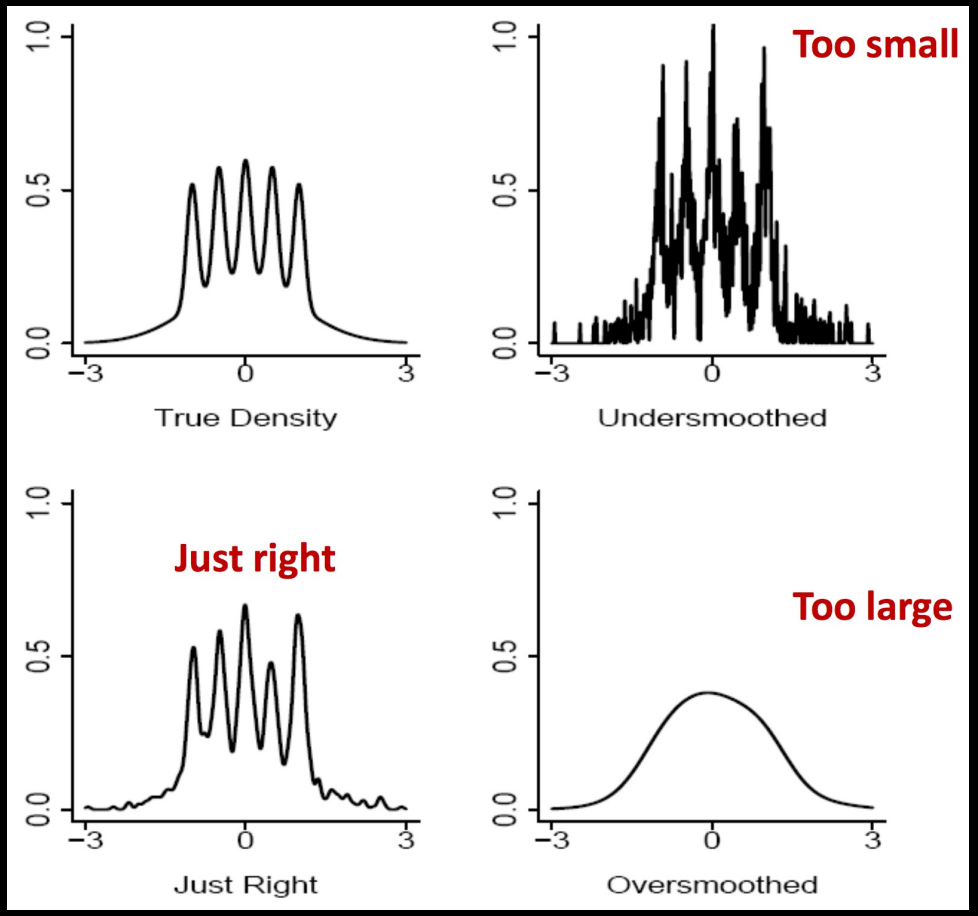
5.8 Regularization and Reproducing Kernel Hilbert Spaces

In this section we cast splines into the larger context of regularization methods and reproducing kernel Hilbert spaces. This section is quite technical and can be skipped by the disinterested or intimidated reader.

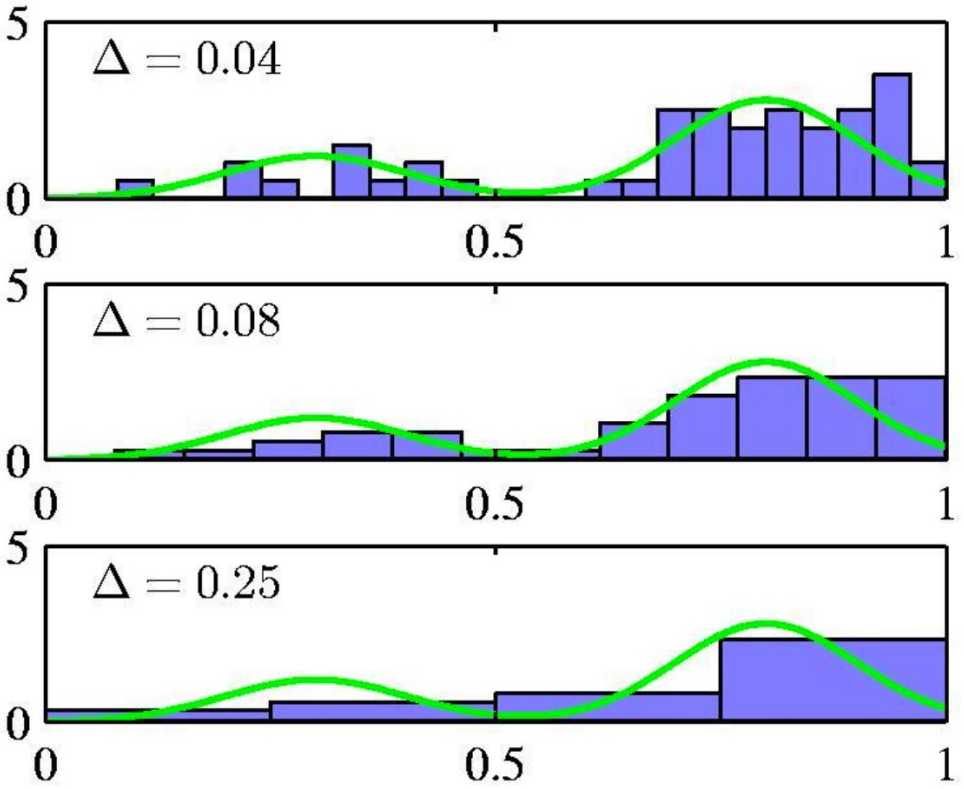
Elements of Statistical Learning, Chpt. 5

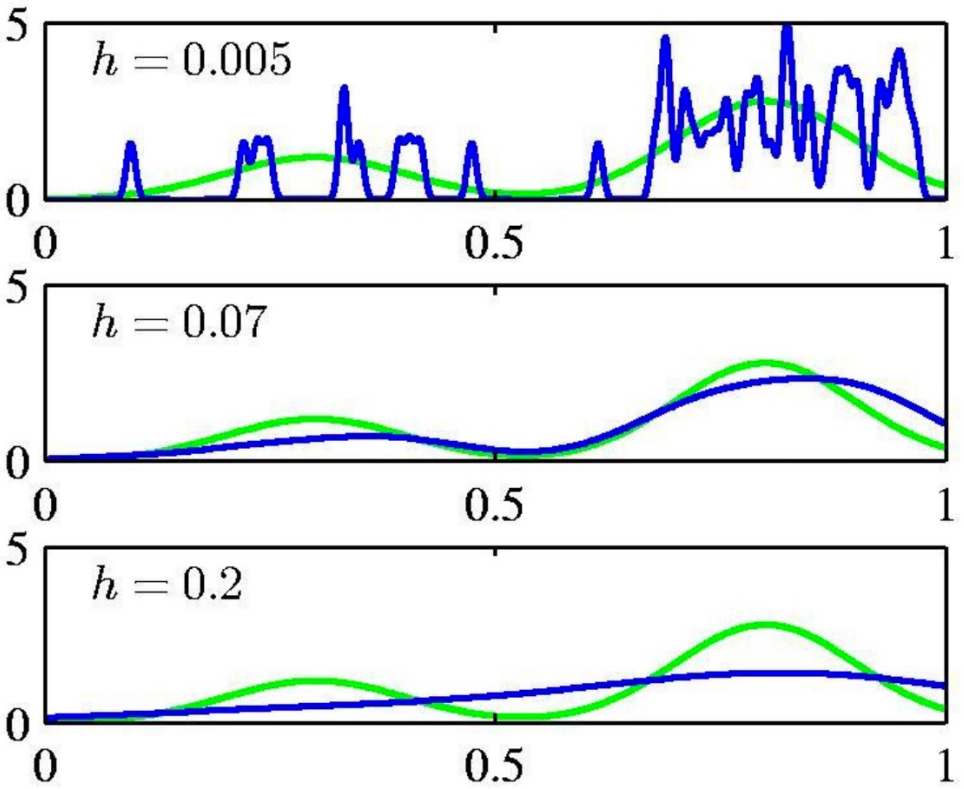
Choice of kernel bandwidth

The **Bart-Simpson Density**



Histograms versus KDE





KNN Density Estimation

- Recall
  - Histograms
  - KDE

$$\hat{p}(x) = \frac{n_i}{n\Delta} \mathbf{1}_{x \in \mathrm{Bin}_i}$$

$$\hat{p}(x) = \frac{n_x}{n\Delta}$$

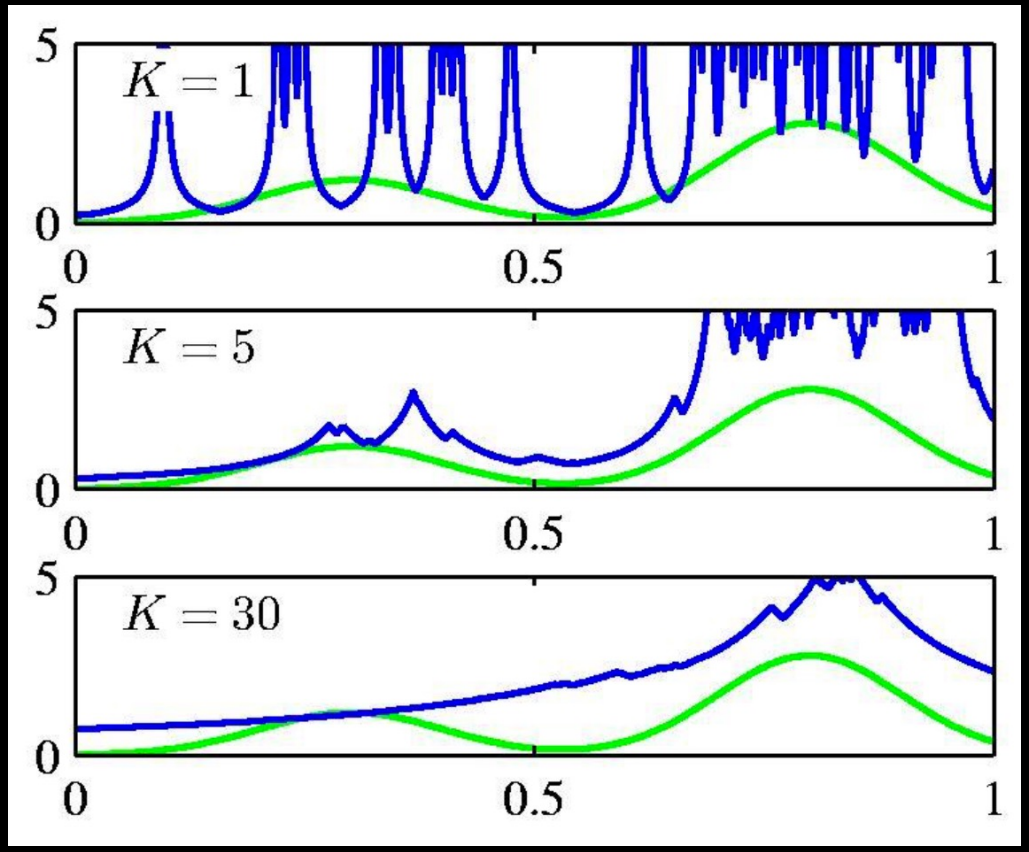
- Fix Δ, estimate number of points within Δ of *x* (n_i or n_x) from the data
- Fix $n_x = k$, estimate Δ from data (volume of ball around *x* with *k* data points)

- **KNN Density Estimation**

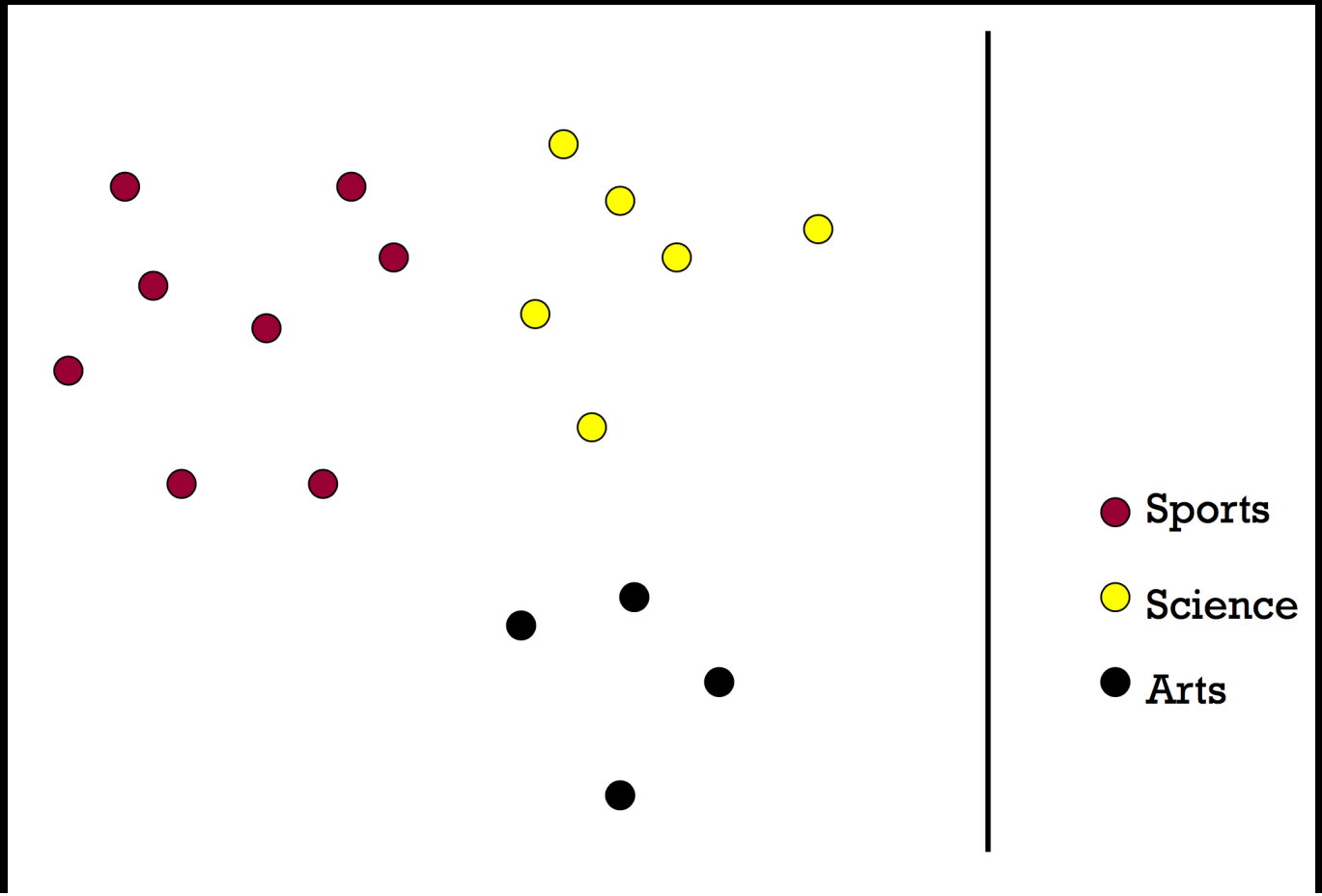
$$\hat{p}(x) = \frac{k}{n\Delta_{k,x}}$$

KNN Density Estimation

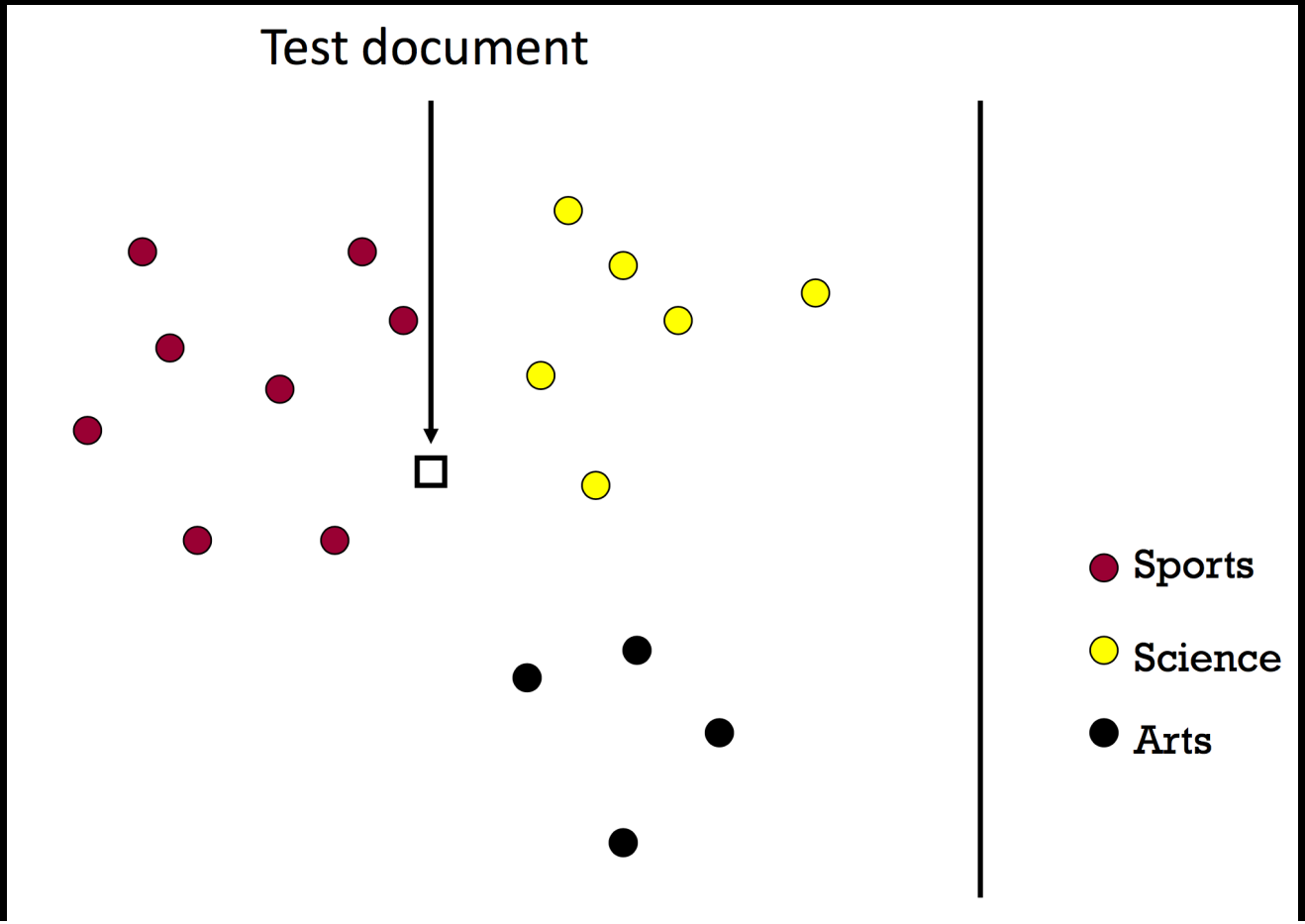
- *k* acts as a smoother
- Not very popular for density estimation
  - Computationally expensive
  - Estimates are poor
- **But related version for classification is very popular**



KNN Classification

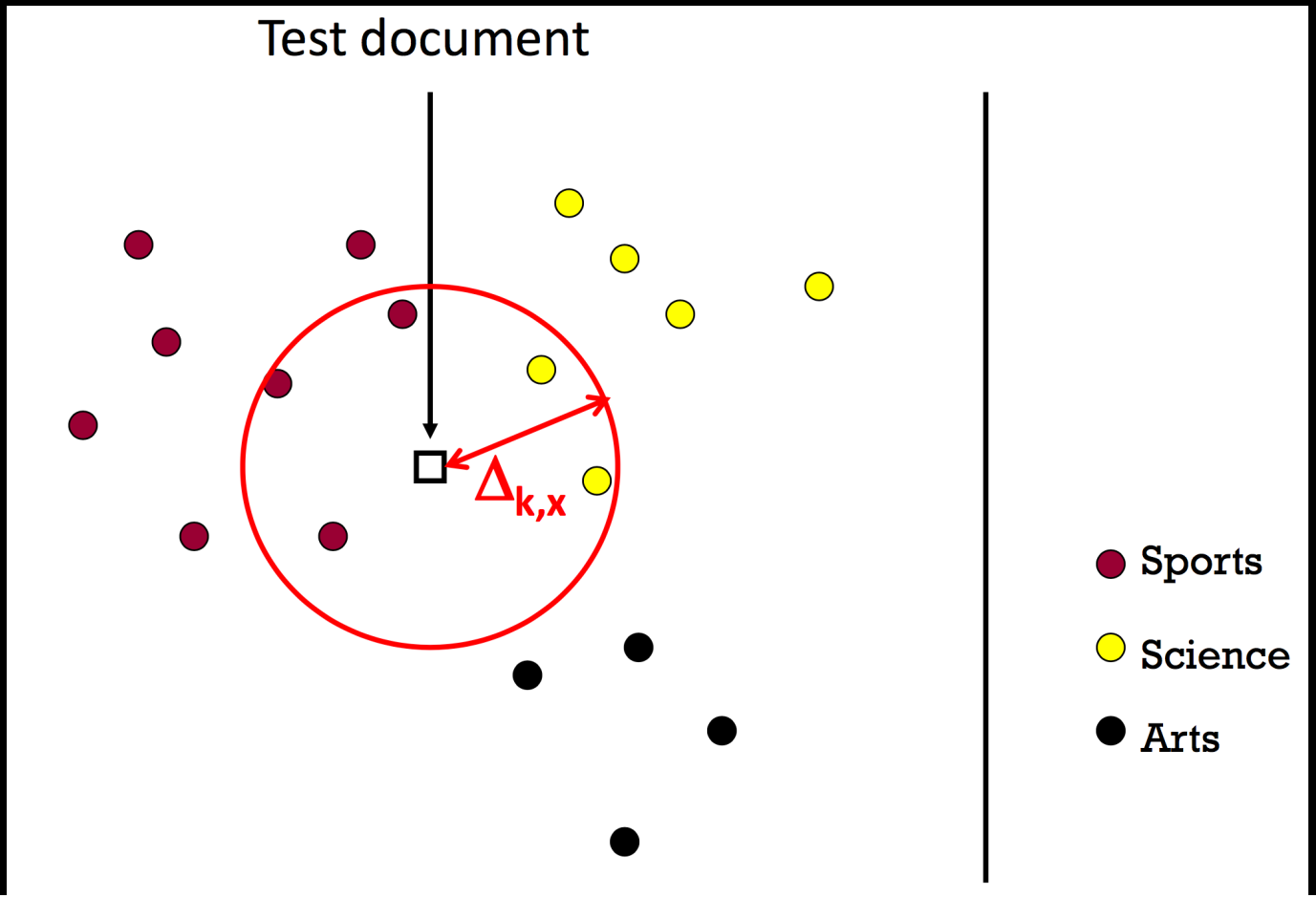


KNN Classification



KNN Classification

- *k = 4*
- What should we predict?
- Average? Majority? **Why?**



KNN Classification

- Optimal classifier

$$f^*(x) = \arg\max_y P(y|x)$$
$$= \arg\max_y P(x|y)P(y)$$

- KNN classifier

$$\hat{f}_{kNN}(x) = \arg\max_y \hat{p}_{kNN}(x|y)\hat{P}(y)$$
$$= \arg\max_y k_y$$

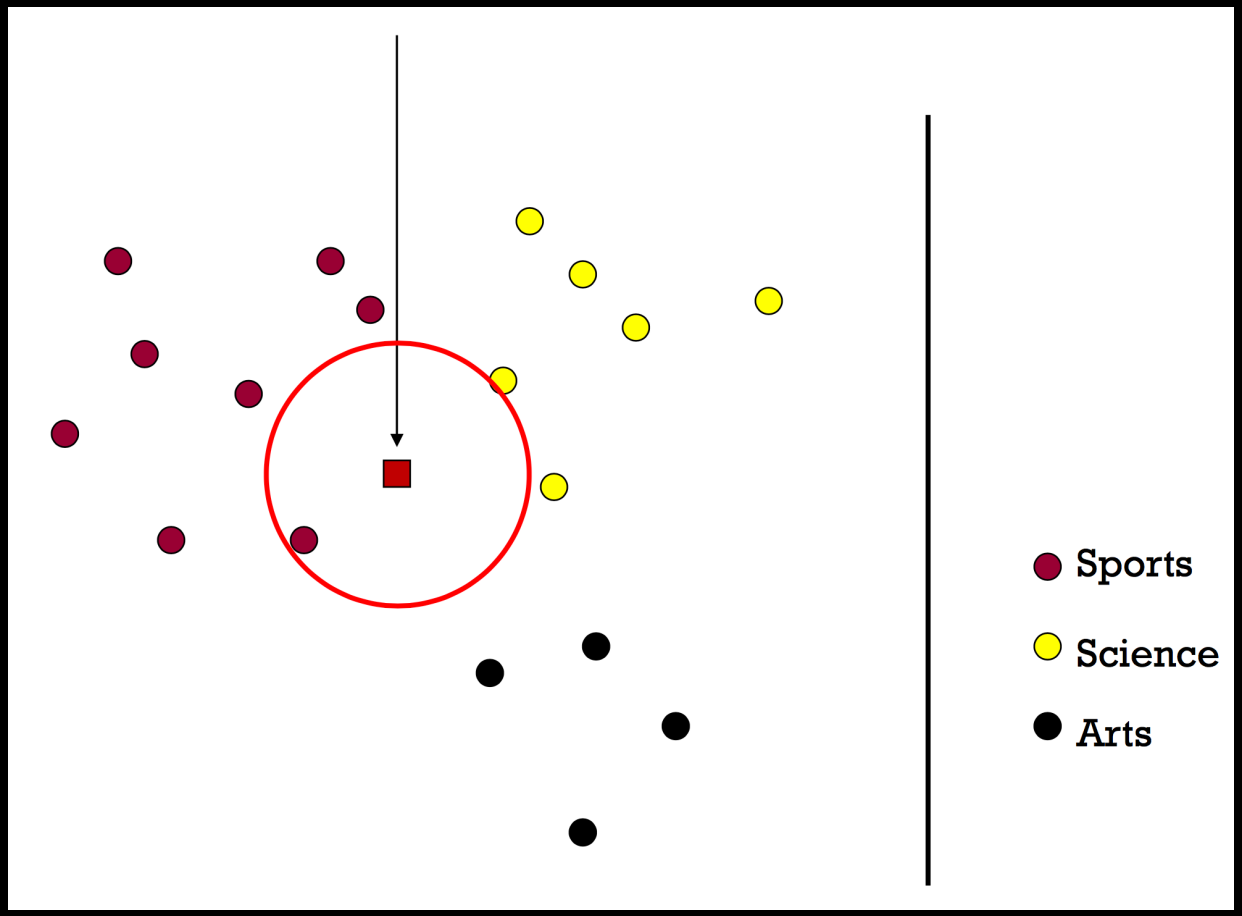
$$\hat{p}_{kNN}(x|y) = \frac{k_y}{n_y \Delta_{k,x}}$$

$$\sum_y k_y = k \quad \hat{P}(y) = \frac{n_y}{n}$$

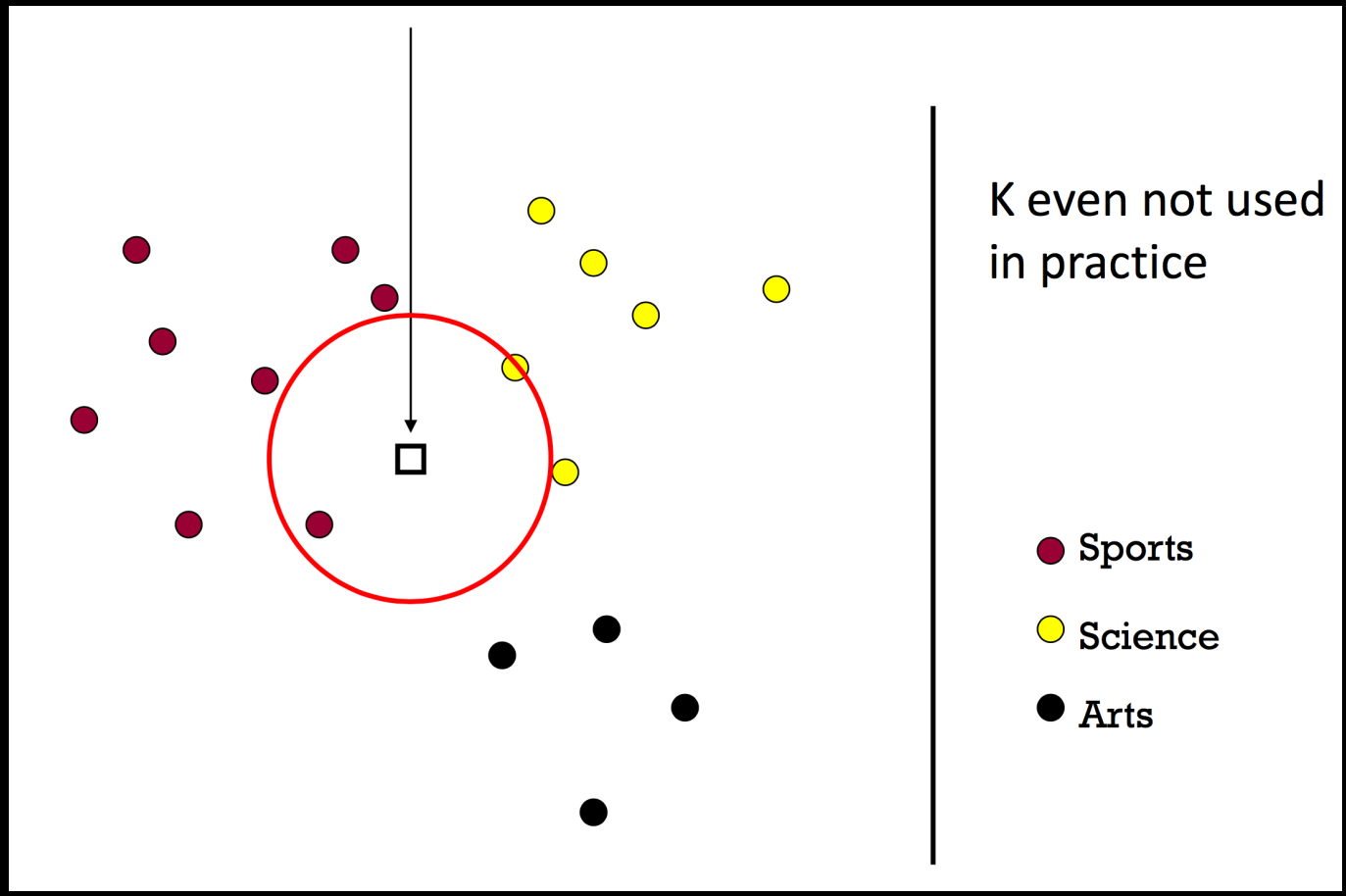
# of training points in class *y*

# of training points in class *y* that lie within Δ_k ball

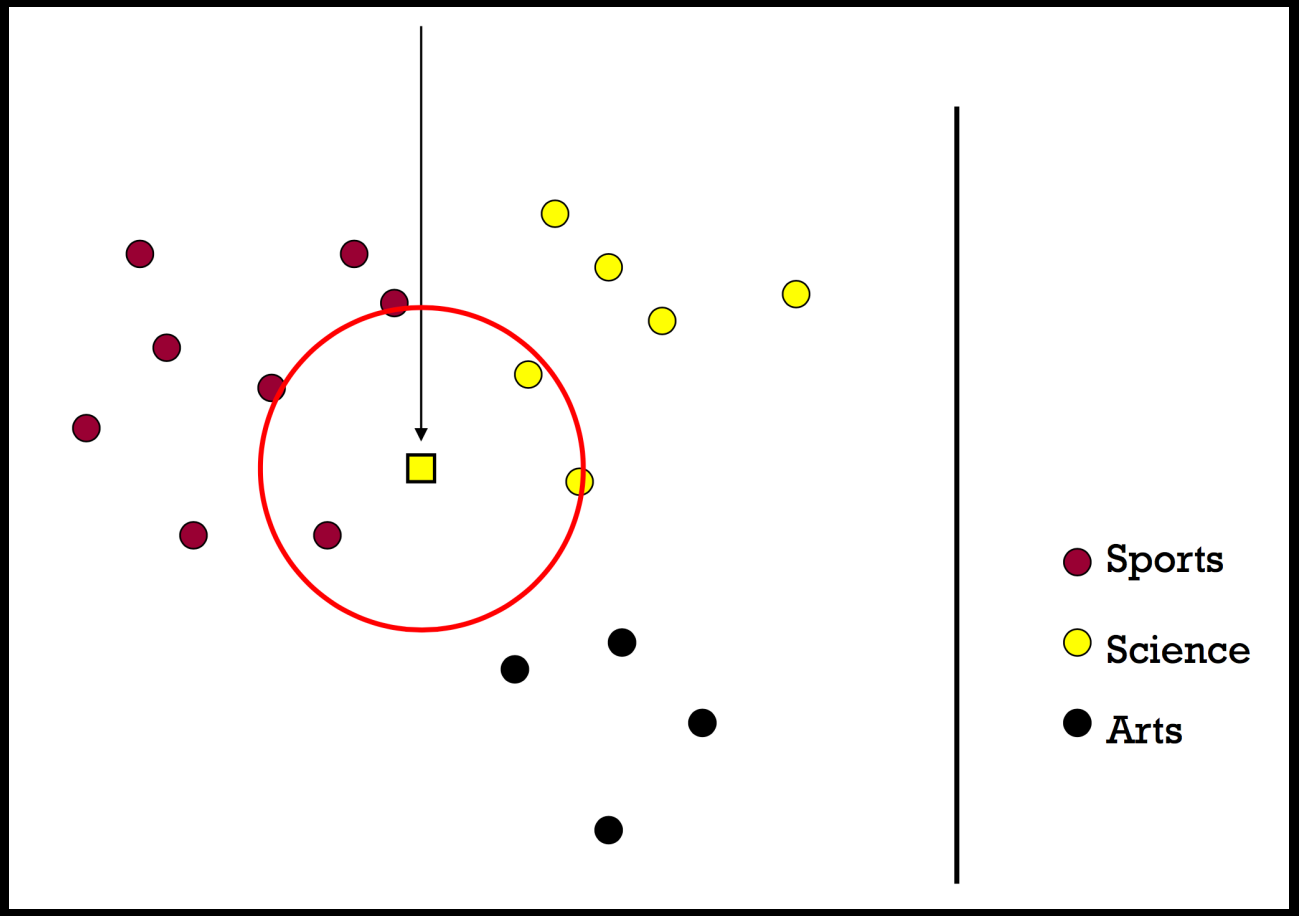
1-NN



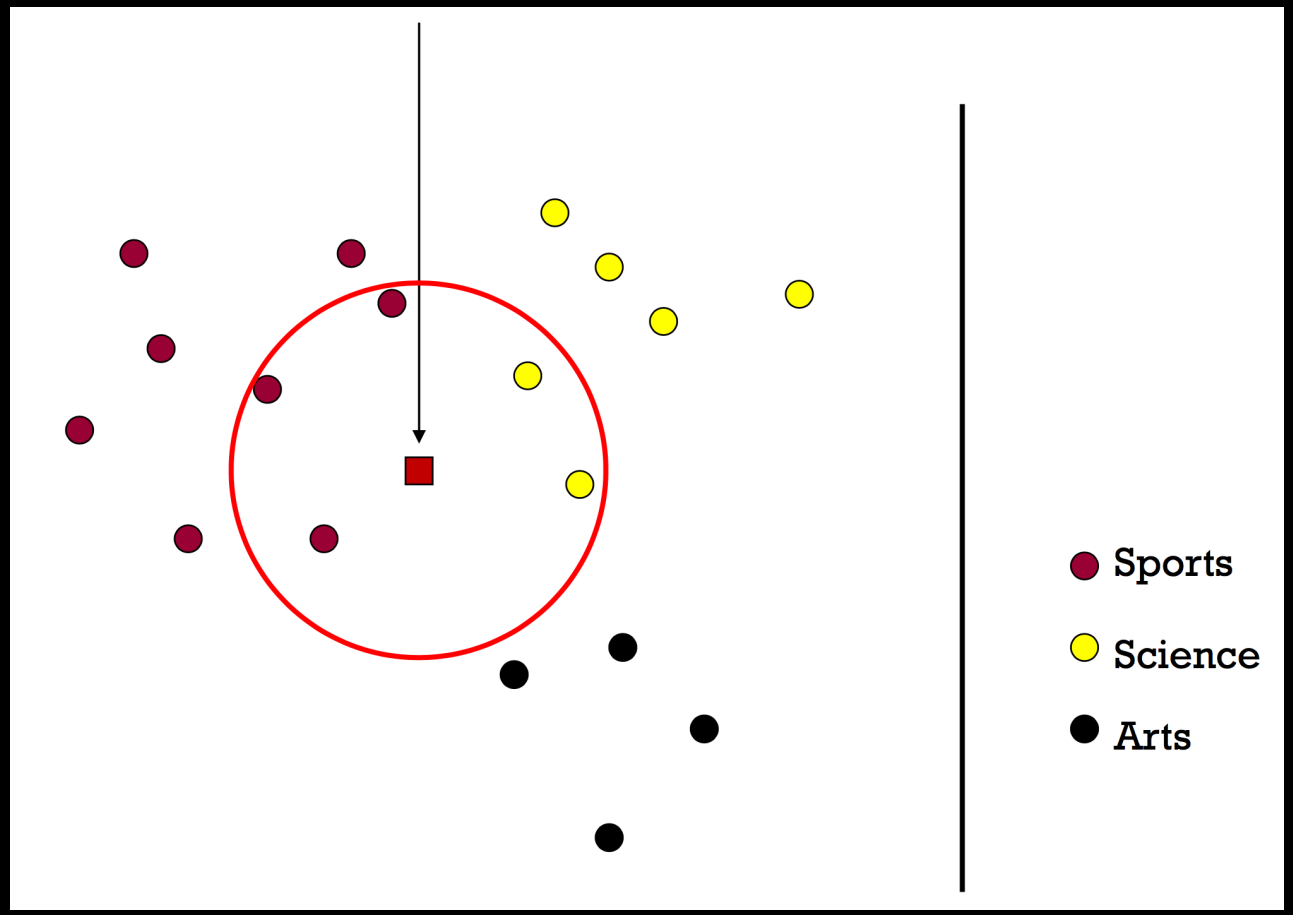
2-NN



3-NN



5-NN



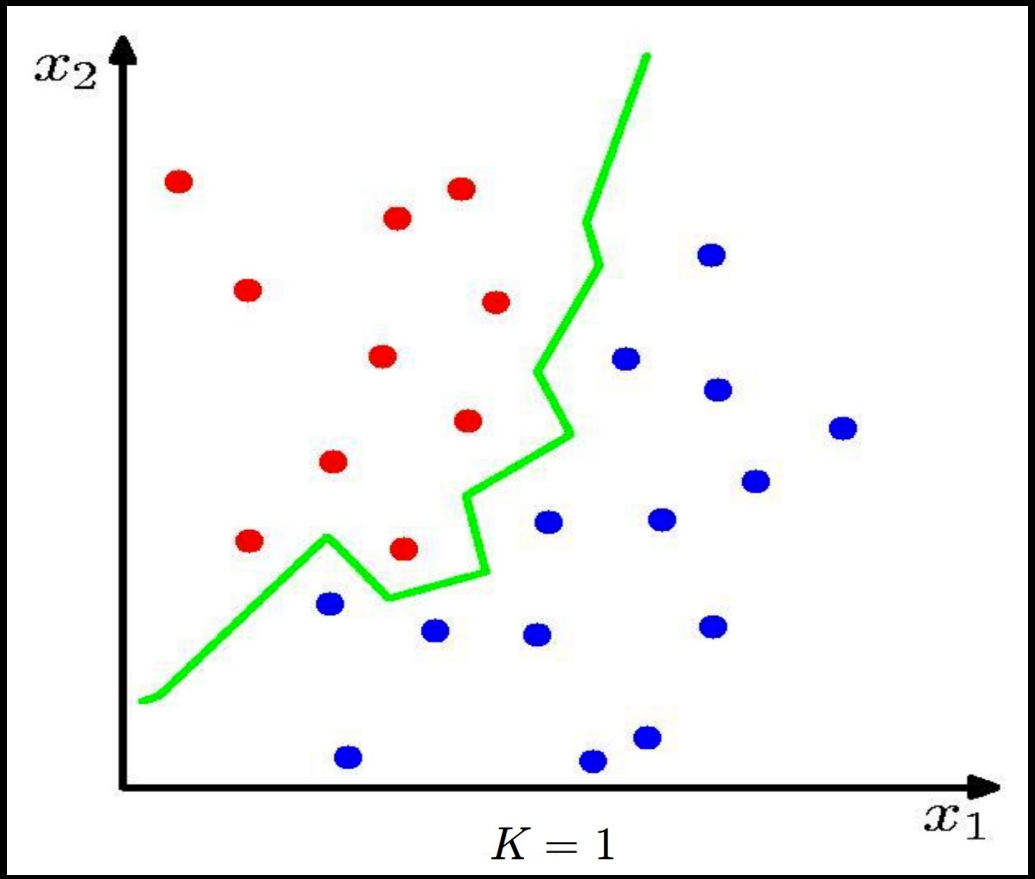
What is the best *k*?

- Bias-variance trade-off

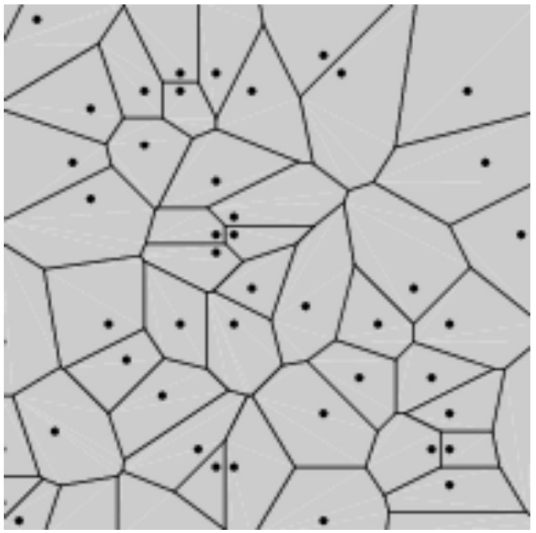
- **Large *k*** = predicted label is more stable
- **Small *k*** = predicted label is more accurate

- **Similar to density estimation**

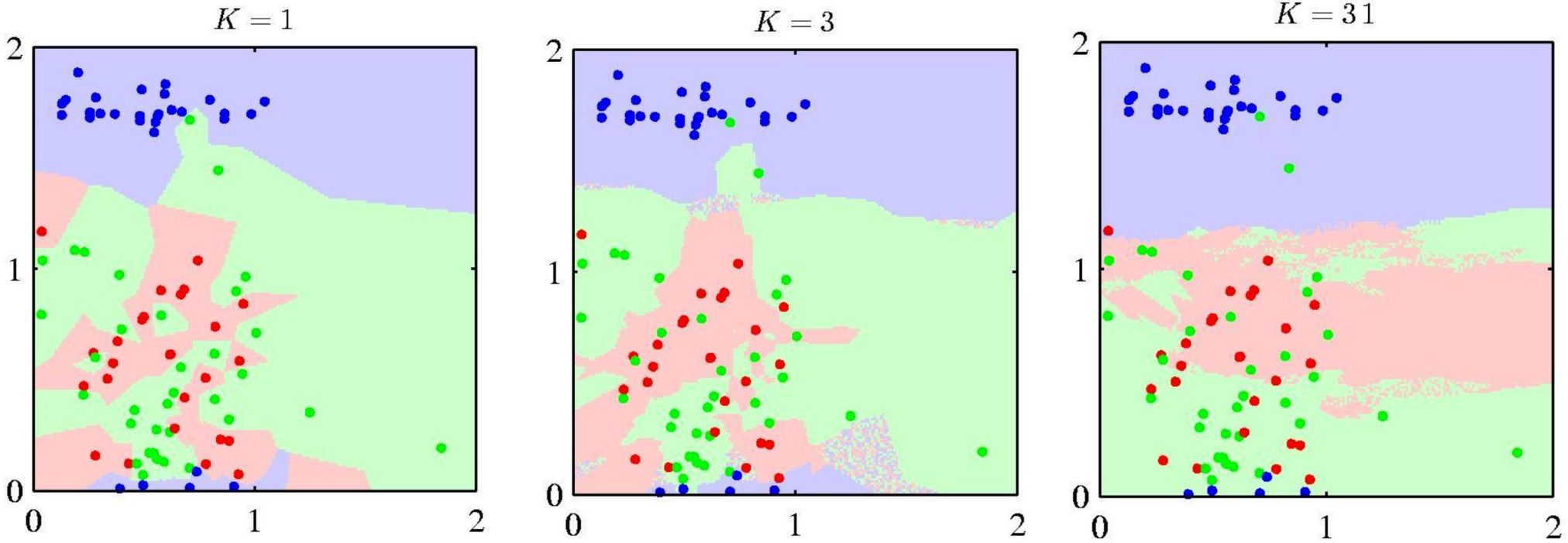
1-NN Decision Boundary



Voronoi Diagram



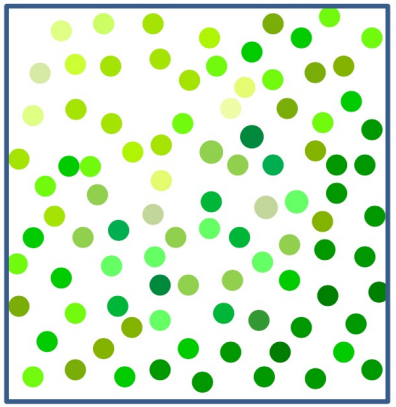
KNN Decision Boundaries



- **Guarantee:** For $n \rightarrow \infty$, error rate of 1-NN is never more than 2x optimal error rate

Temperature Sensing

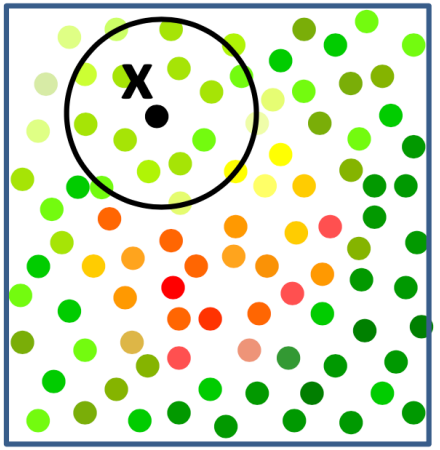
- What is the temperature in the room?



$$\widehat{T} = \frac{1}{n} \sum_{i=1}^{n} Y_i$$

Average

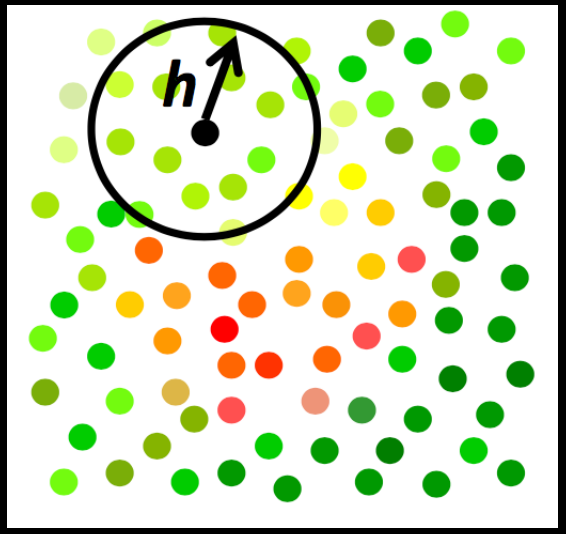
at location x?



$$\widehat{T}(x) = \frac{\sum_{i=1}^{n} Y_i 1_{||X_i - x|| \leq h}}{\sum_{i=1}^{n} 1_{||X_i - x|| \leq h}}$$

"Local" Average

Kernel Regression



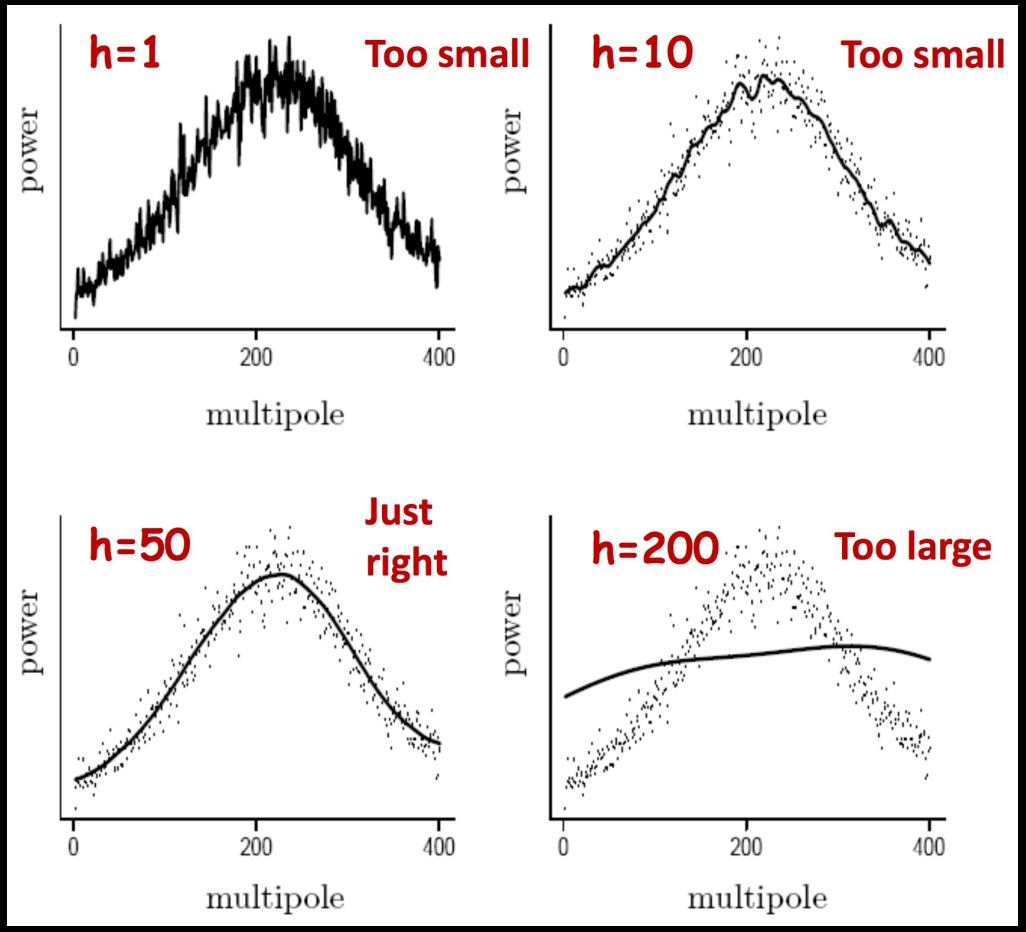
- Or “local” regression

- Nadaraya-Watson Kernel Estimator

$$\hat{f}_n(X) = \sum_{i=1}^{n} w_i Y_i \quad \text{...where} \quad w_i(X) = \frac{K\left(\frac{X - X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{X - X_i}{h}\right)}$$

- Weight each training point on distance to test point
- Boxcar kernel yields local average

Choice of kernel bandwidth



Choice of *kernel* is not terribly important!

Kernel Regression as WLS

- Weighted Least Squares (WLS) has the form

$$\min_f \sum_{i=1}^{n} w_i (f(X_i) - Y_i)^2$$

- Compare to Nadaraya-Watson form

$$w_i(X) = \frac{K\left(\frac{X - X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{X - X_i}{h}\right)}$$

- **Kernel regression corresponds to locally constant estimator obtained from [locally] weighted least squares**

- **Set** $f(X_i) = \beta$ **where $\boldsymbol{\beta}$ is constant**

Kernel Regression as WLS

$$\min_{\beta} \sum_{i=1}^{n} w_i(\beta - Y_i)^2 \qquad w_i(X) = \frac{K\left(\frac{X-X_i}{h}\right)}{\sum_{i=1}^{n} K\left(\frac{X-X_i}{h}\right)}$$

A constant value

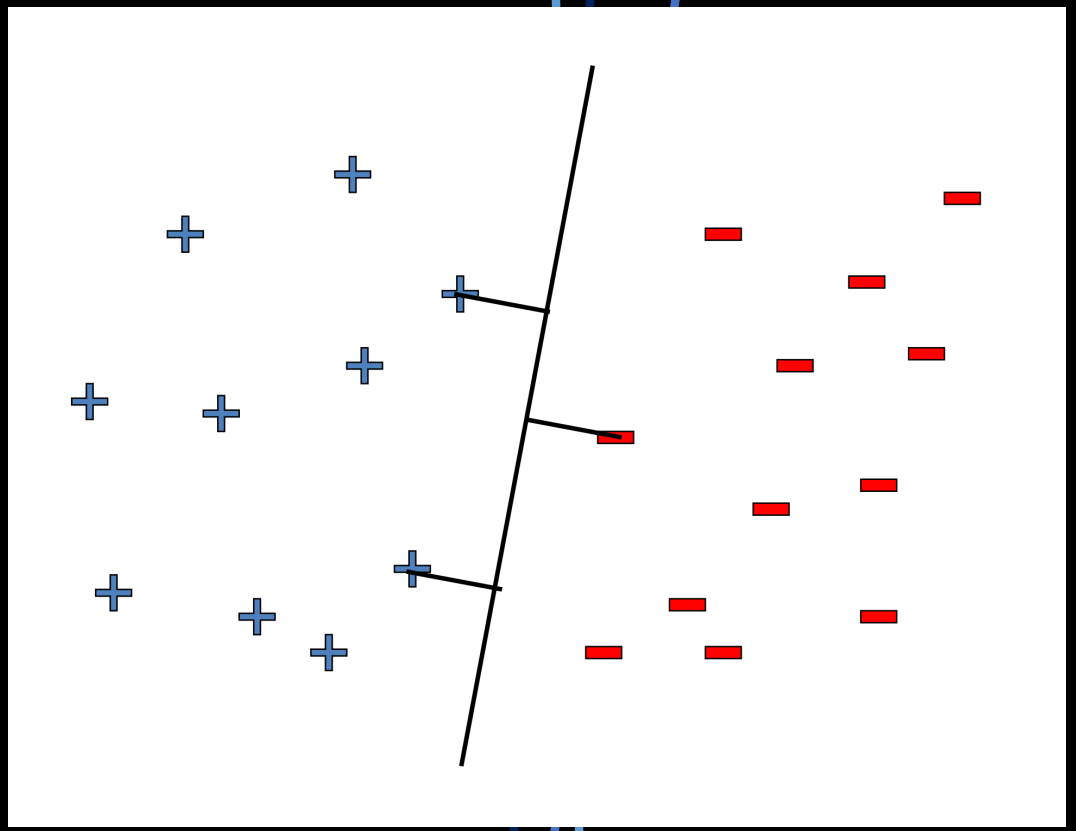
$$\frac{\partial J(\beta)}{\partial \beta} = 2\sum_{i=1}^{n} w_i(\beta - Y_i) = 0$$

Individual weights have to sum to 1

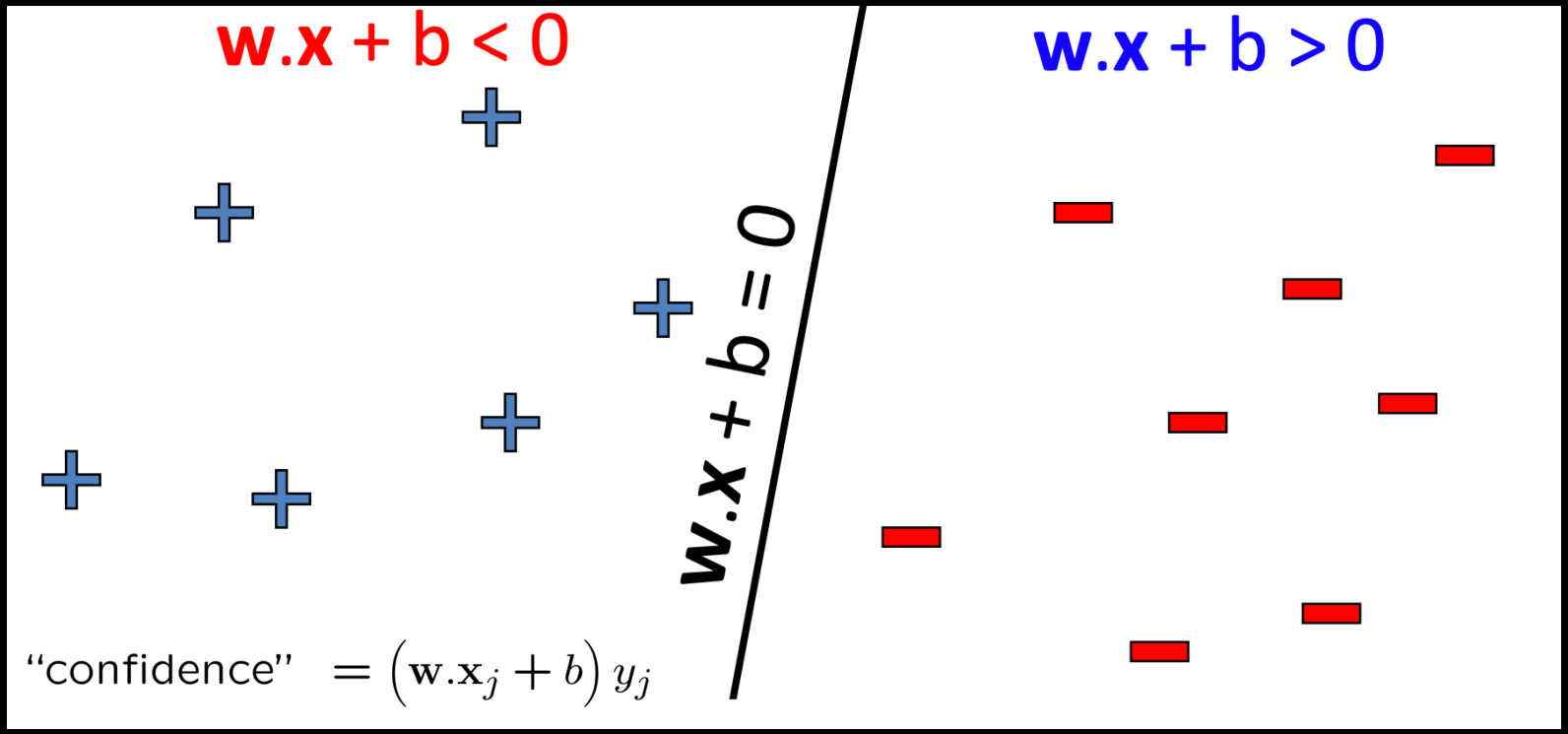
$$\rightarrow \hat{f}_n(X) = \hat{\beta} = \sum_{i=1}^{n} w_i Y_i$$

Support Vector Machines

- Linear classifiers—which is better?
- Pick the one with the **largest margin**



Support Vector Machines



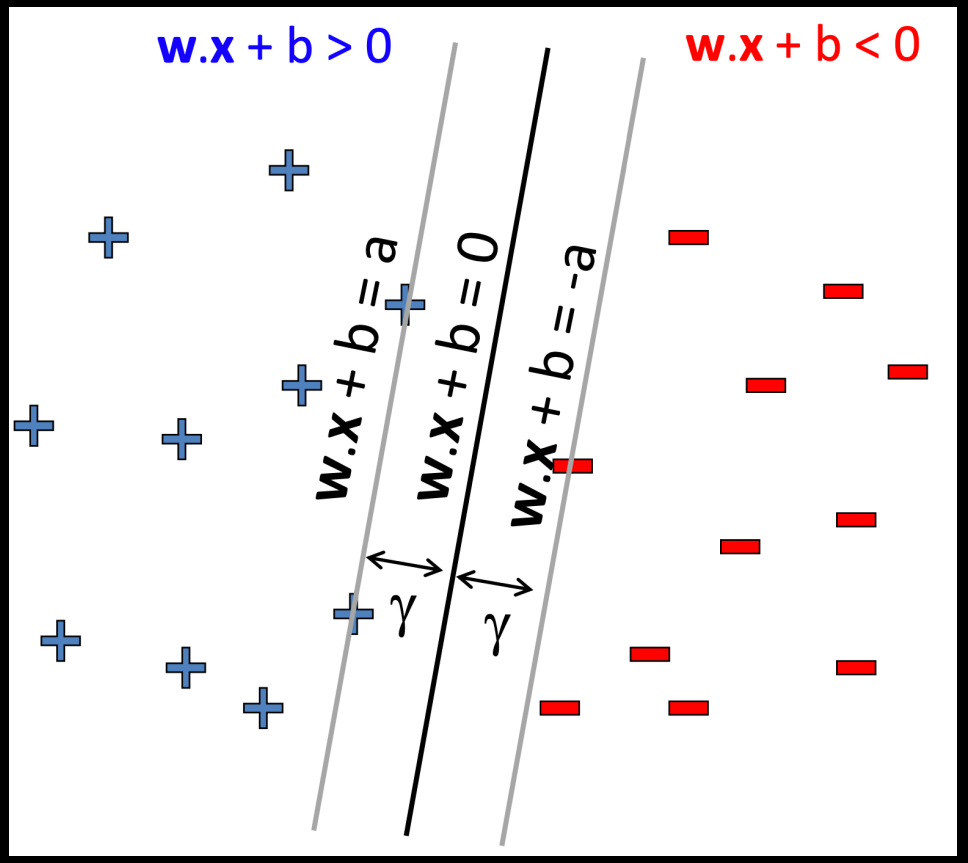
Support Vector Machines!

I promise I’m going somewhere with this…

Support Vector Machines

- Maximize the margin
- Distance of closest example / data point from the decision boundary / hyperplane:

$$\text{margin} = \gamma = 2a/\|w\|$$



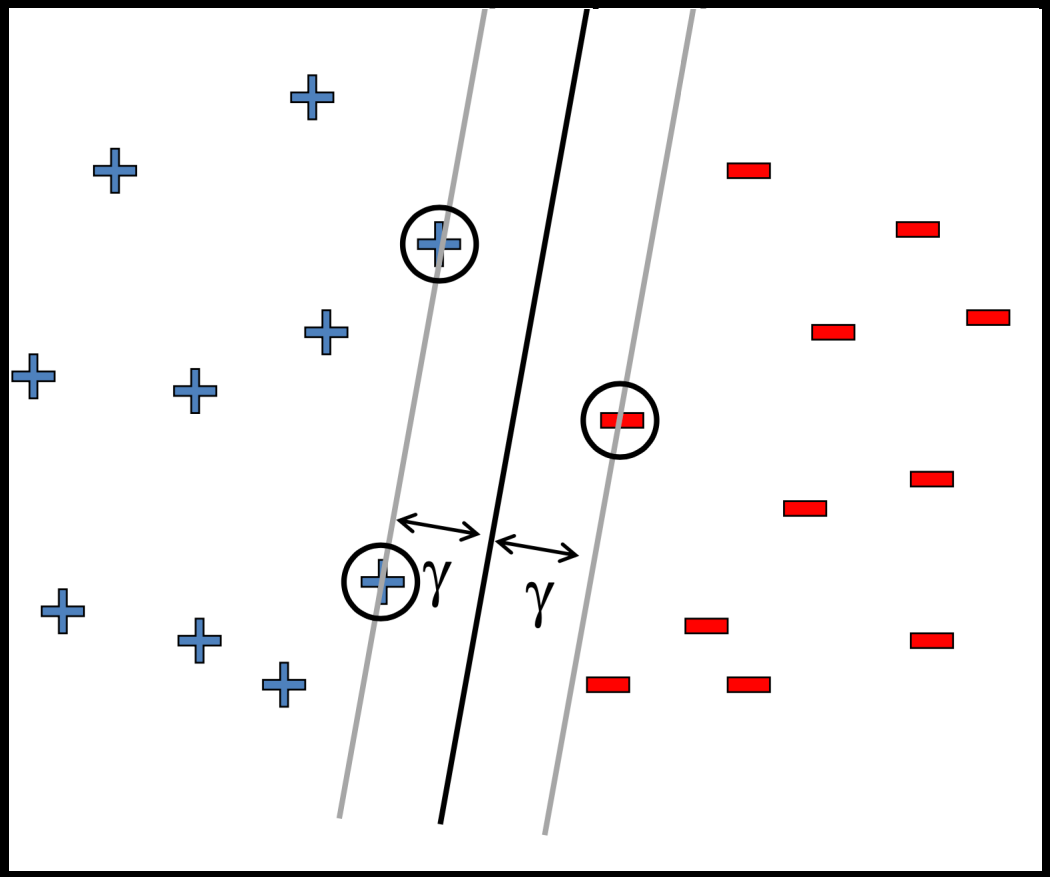
Support Vector Machines

- Rewrite the equation (drop *a* in favor of 1)

$$\min_{\mathbf{w},b} \ \mathbf{w}.\mathbf{w}$$

$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 \quad \forall j$$

- Solve via quadratic programming
- Data points along margin = **support vectors**

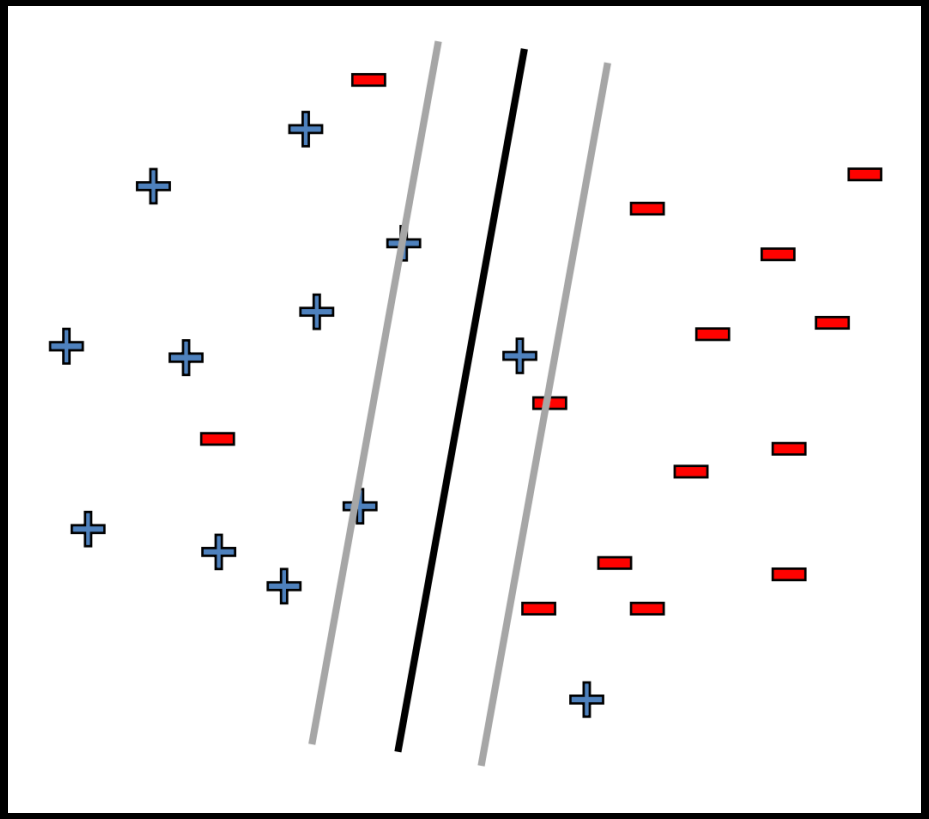


SVMs

- What if the data aren't linearly separable?
- Allow for "errors"

$$\min_{\mathbf{w},b} \ \mathbf{w}.\mathbf{w} + C \ \#\text{mistakes}$$
$$\text{s.t. } (\mathbf{w}.\mathbf{x}_j + b)\, y_j \geq 1 \quad \forall j$$

- Maximize margin AND minimize mistakes
  - C: tradeoff parameter



SVMs

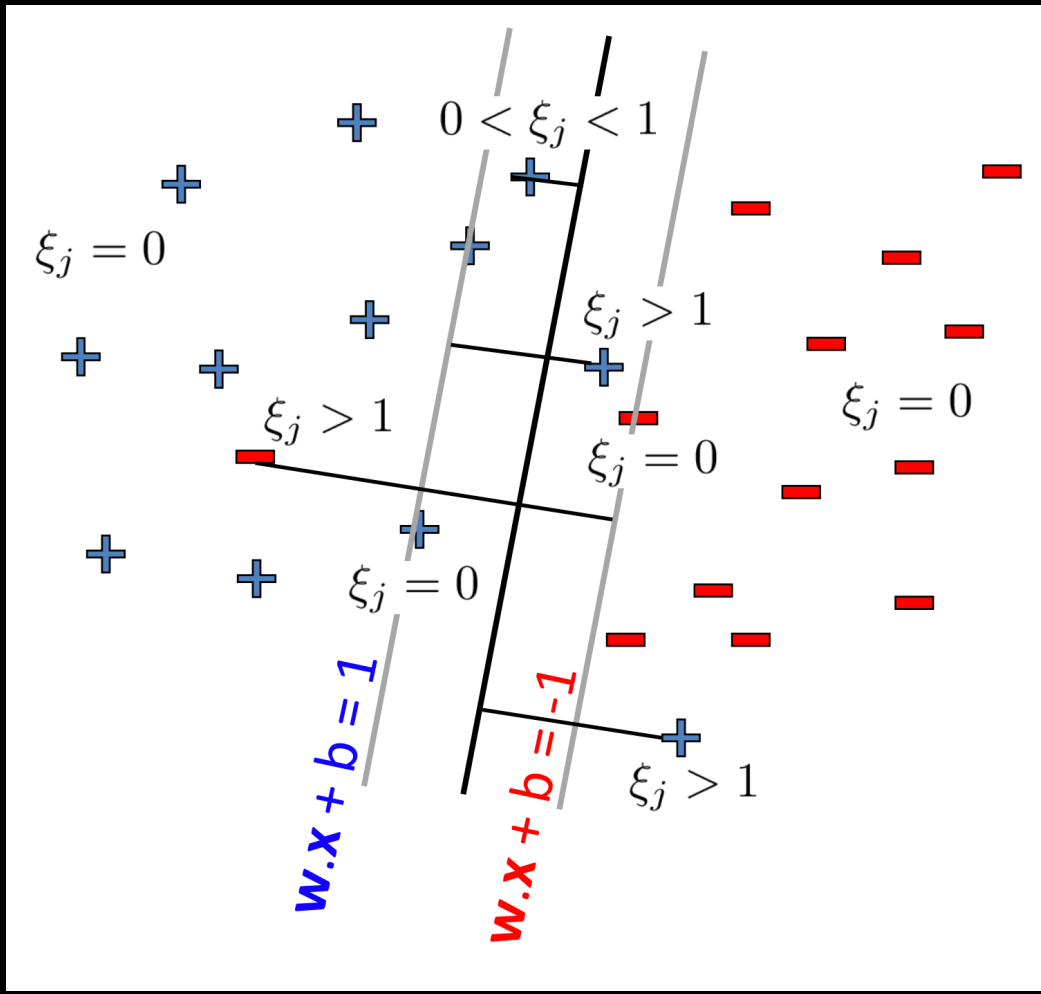
- What if the data *still* aren't linearly separable?
- **"Soft" margin**
  - (penalize misclassified data by how far it is from the margin)

$$(\mathbf{w}.\mathbf{x}_j+b)\, y_j \geq 1-\xi_j \quad \forall j$$

$$\xi_j \geq 0 \quad \forall j$$

  - Misclassification penalty: $C\,\xi_j$
  - Recover "hard" margin:

Set $C = \infty$



SVMs are great, but…

- Where is this going?
- **First**, SVMs were the "big thing" right before deep learning
  - Neural network research had been dead for 10+ years
  - SVMs were showing immense promise, especially with high-dim data
- **Second**, SVMs share a lot of theory with deep learning
  - Much of this theory found a second life in the Transformer architecture that powers all the modern large language models!

12.2.1 Computing the Support Vector Classifier

The problem (12.7) is quadratic with linear inequality constraints, hence it is a convex optimization problem. We describe a quadratic programming solution using Lagrange multipliers. Computationally it is convenient to re-express (12.7) in the equivalent form

- Start with core parameterization of SVM

$$\min_{\beta,\beta_0} \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i$$
$$\text{subject to} \quad \xi_i \geq 0,\ y_i(x_i^T\beta + \beta_0) \geq 1 - \xi_i\ \forall i,$$

- Write "primal" objective (Lagrange) function

$$L_P = \frac{1}{2}\|\beta\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\alpha_i[y_i(x_i^T\beta + \beta_0) - (1-\xi_i)] - \sum_{i=1}^{N}\mu_i\xi_i,$$

- Differentiate with respect to β, β_0, and ζ_i and set to 0

$$\begin{aligned}
\beta &= \sum_{i=1}^{N}\alpha_i y_i x_i, \\
0 &= \sum_{i=1}^{N}\alpha_i y_i, \\
\alpha_i &= C - \mu_i,\ \forall i,
\end{aligned}$$

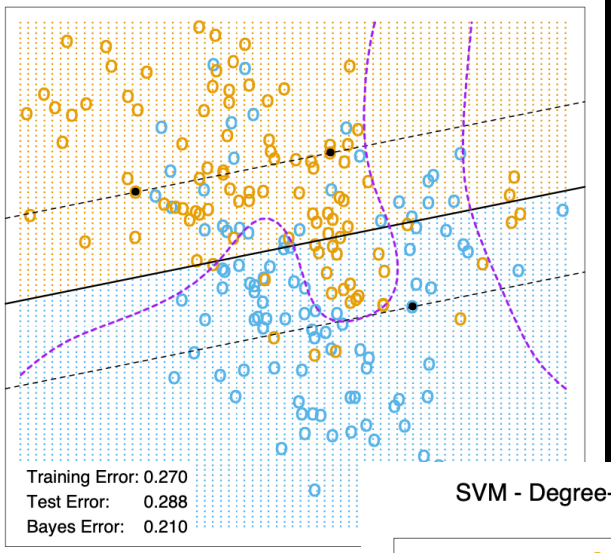
- Substitute back into primal equation, and get the Lagrangian **Dual**

$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} x_i^T x_{i'}.$$

- Notice anything?
- **Kernel!**

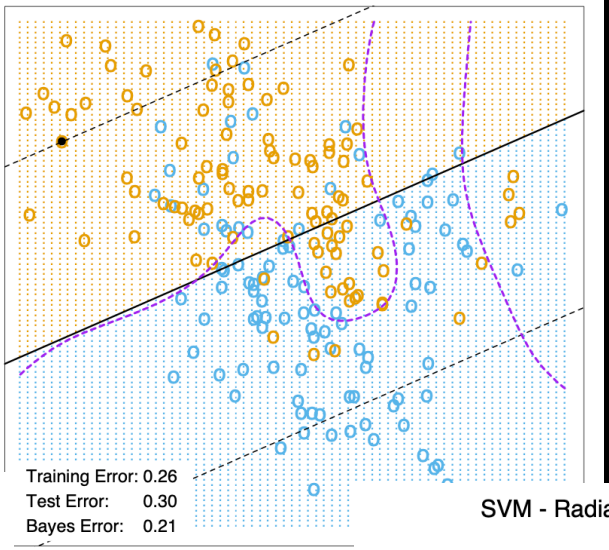
$$L_D = \sum_{i=1}^{N} \alpha_i - \frac{1}{2} \sum_{i=1}^{N} \sum_{i'=1}^{N} \alpha_i \alpha_{i'} y_i y_{i'} \langle h(x_i), h(x_{i'}) \rangle.$$

$$K(x, x') = \langle h(x), h(x') \rangle$$

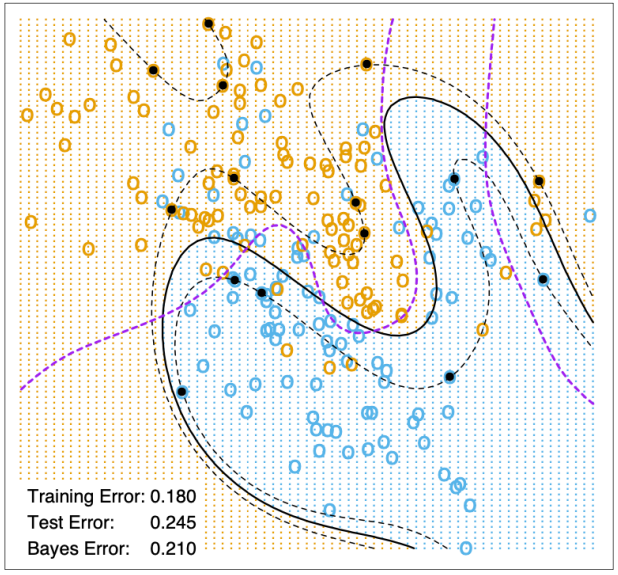


$C = 10000$

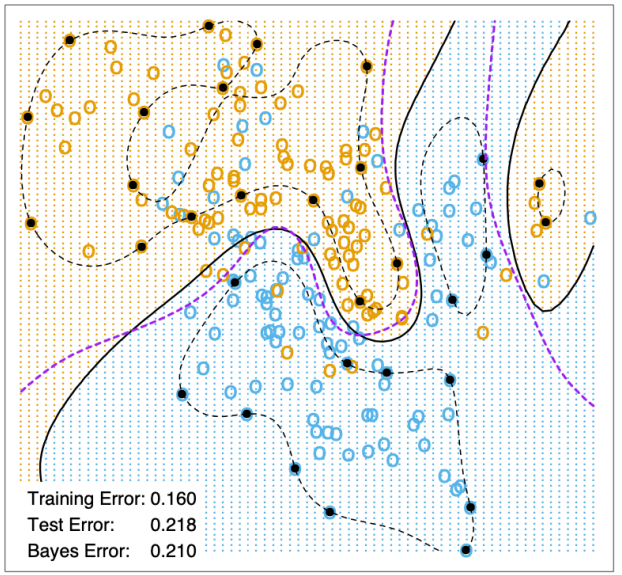
SVM - Degree-4 Polynomial in Feature Space



$C = 0.01$



SVM - Radial Kernel in Feature Space



Summary

- Nonparametric places mild assumptions on data; good models for complex data
  - Usually requires storing & computing with full dataset
- Parametric models rely on very strong, simplistic assumptions
  - Once fitted, they are much more efficient with storage and computation
- Effects of bin width & kernel bandwidth
  - Bias-variance trade-off
- Kernel regression
  - Comparison to weighted least squares
- Support Vector Machines
  - Powerful “shallow” models
  - Dual formulation of objective allows for kernel functions

Questions?

References

- “All of Nonparametric Statistics”, http://www.stat.cmu.edu/~larry/all-of-nonpar/index.html

Other slides

Case Study: Newsgroups Classification

- 20 Newsgroups
- 61,118 words
- 18,774 documents
- Class label descriptions

comp.graphics comp.os.ms-windows.misc comp.sys.ibm.pc.hardware comp.sys.mac.hardware comp.windows.x	rec.autos rec.motorcycles rec.sport.baseball rec.sport.hockey	sci.crypt sci.electronics sci.med sci.space
misc.forsale	talk.politics.misc talk.politics.guns talk.politics.mideast	talk.religion.misc alt.atheism soc.religion.christian

Case Study: Newsgroups Classification

- Training/Testing
  - 50%-50% randomly split
  - 10 runs
  - Report average results
- Evaluation Criteria

$$Accuracy = \frac{\sum_{i \in test\ set} \mathrm{I}(predict_i == true\ label_i)}{\#\ of\ test\ samples}$$

Case Study: Newsgroups Classification

- Results in binary class comparisons