Foundations of Machine Learning

Module 3: Instance Based Learning and Feature Selection

Part D: Instance Based Learning

Sudeshna Sarkar
IIT Kharagpur

Instance-Based Learning

- One way of solving tasks of approximating discrete or real valued target functions
- Have training examples: $(x_n, f(x_n))$, n=1..N.
- Key idea:
  - just store the training examples
  - when a test example is given then find the closest matches

Inductive Assumption

- Similar inputs map to similar outputs
  - If not true => learning is impossible
  - If true => learning reduces to defining "*similar*"
- Not all similarities created equal
  - predicting a person's weight may depend on different attributes than predicting their IQ

Basic k-nearest neighbor classification

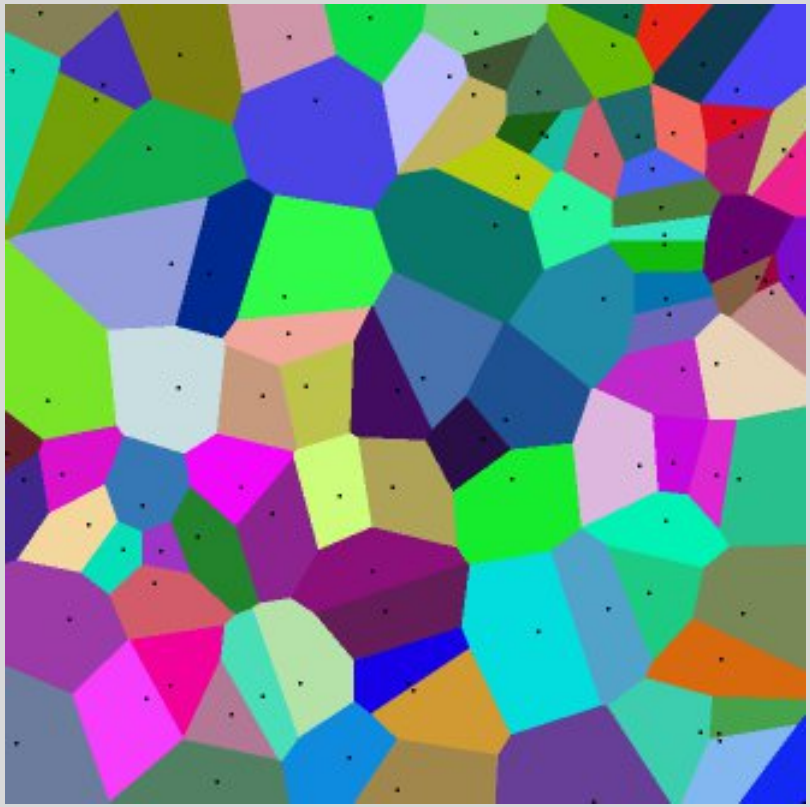
- Training method:
  - Save the training examples
- At prediction time:
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those y_i's.

- Example:

  http://cgm.cs.mcgill.ca/~soss/cs644/projects/simard/

What is the decision boundary?

Voronoi diagram



Basic k-nearest neighbor classification

- Training method:
  - Save the training examples
- At prediction time:
  - Find the *k* training examples $(x_1,y_1),\ldots(x_k,y_k)$ that are closest to the test example *x*
  - Predict the most frequent class among those y_i's.

- Improvements:
  - *Weighting* examples from the neighborhood
  - Measuring "*closeness*"
  - Finding "close" examples in a large training set *quickly*

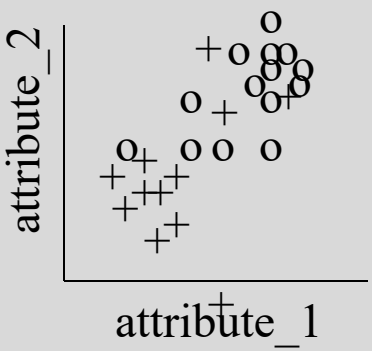
k-Nearest Neighbor

$$\mathrm{Dist}(c_1, c_2) = \sum_{i=1}^{N} \left(\mathrm{attr}_i(c_1) - \mathrm{attr}_i(c_2)\right)^2$$

$$k - \mathrm{NearestNeighbors} = \left\{k - \mathrm{MIN}(\mathrm{Dist}(c_i, c_{\mathrm{test}}))\right\}$$

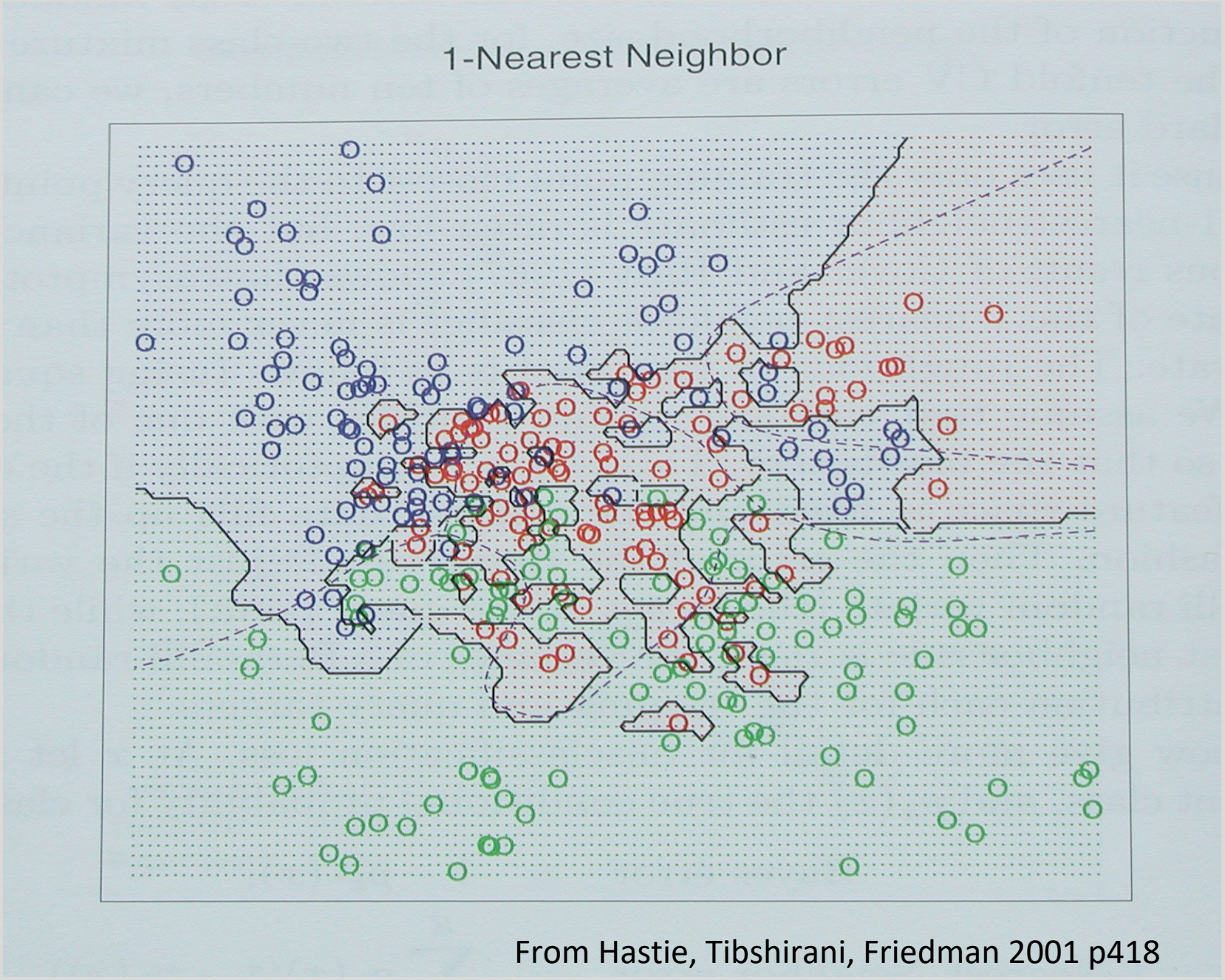
$$\mathrm{prediction}_{\mathrm{test}} = \frac{1}{k}\sum_{i=1}^{k} \mathrm{class}_i \ (\text{or } \frac{1}{k}\sum_{i=1}^{k} \mathrm{value}_i)$$

- Average of k points more reliable when:
  - noise in attributes
  - noise in class labels
  - classes partially overlap

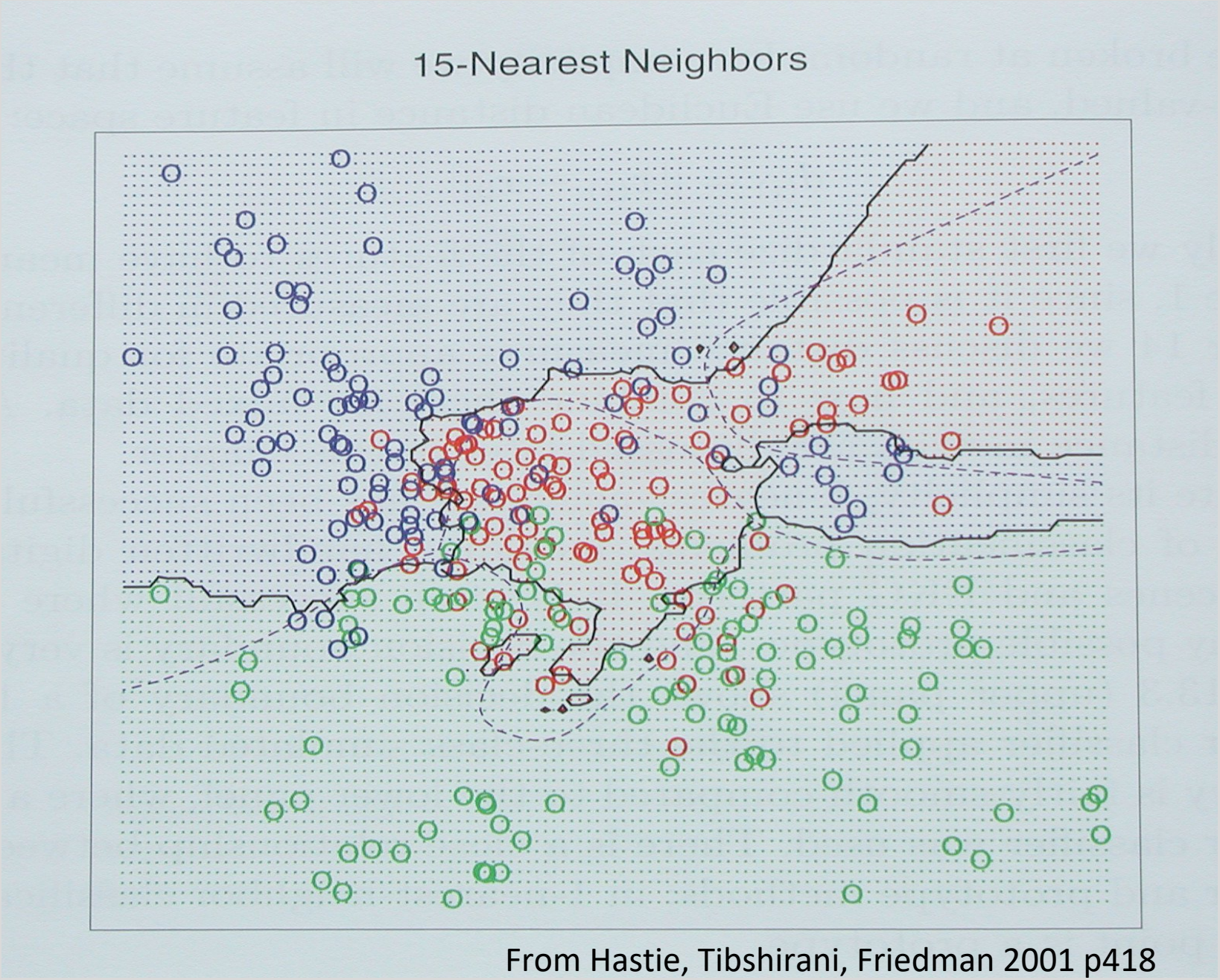


How to choose “k”

- Large k:
  - less sensitive to noise (particularly class noise)
  - better probability estimates for discrete classes
  - larger training sets allow larger values of k
- Small k:
  - captures fine structure of problem space better
  - may be necessary with small training sets
- Balance must be struck between large and small k
- As training set approaches infinity, and k grows large, kNN becomes Bayes optimal

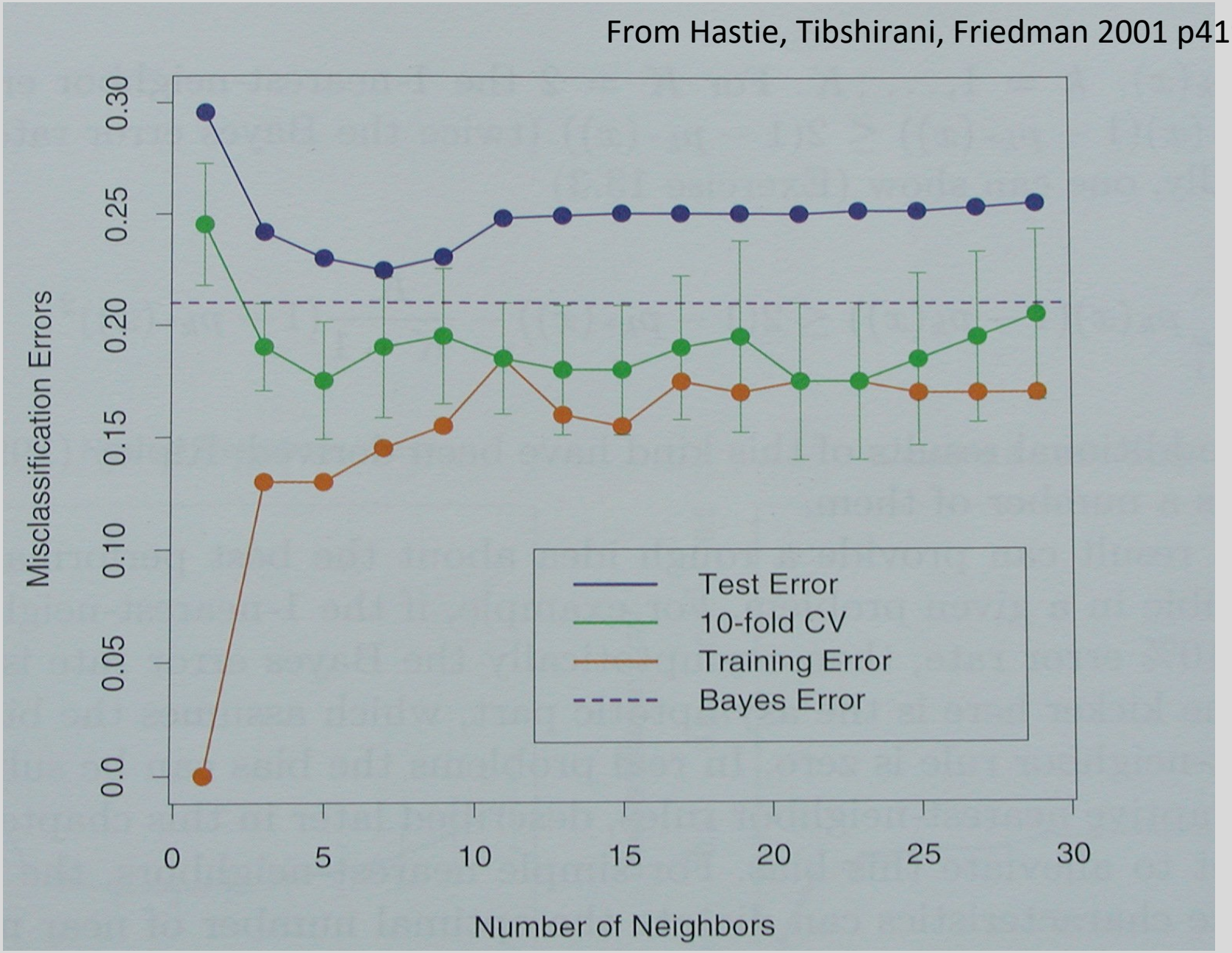


From Hastie, Tibshirani, Friedman 2001 p418



From Hastie, Tibshirani, Friedman 2001 p418

From Hastie, Tibshirani, Friedman 2001 p419



Distance-Weighted kNN

- tradeoff between small and large k can be difficult
  - use large k, but more emphasis on nearer neighbors?

$$\text{prediction}_{\text{test}} = \frac{\sum_{i=1}^{k} w_i * \text{class}_i}{\sum_{i=1}^{k} w_i} \quad \left(\text{or } \frac{\sum_{i=1}^{k} w_i * \text{value}_i}{\sum_{i=1}^{k} w_i}\right)$$

$$w_k = \frac{1}{\text{Dist}(c_k, c_{\text{test}})}$$

Locally Weighted Averaging

- Let k = number of training points
- Let weight fall-off rapidly with distance

$$\text{prediction}_{\text{test}} = \frac{\sum_{i=1}^{k} w_i * \text{class}_i}{\sum_{i=1}^{k} w_i} \quad (\text{or} \quad \frac{\sum_{i=1}^{k} w_i * \text{value}_i}{\sum_{i=1}^{k} w_i})$$

$$w_k = \frac{1}{e^{\text{KernelWidth} \cdot \text{Dist}(c_k, c_{\text{test}})}}$$

- KernelWidth controls size of neighborhood that has large effect on value (analogous to k)

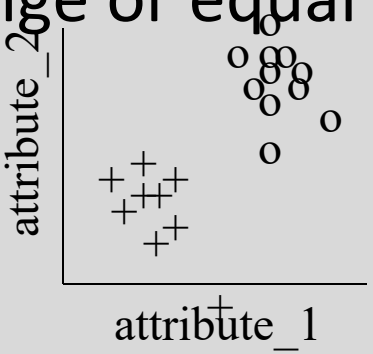
Locally Weighted Regression

- All algs so far are strict averagers: interpolate, but can't extrapolate
- Do weighted regression, centered at test point, weight controlled by distance and KernelWidth
- Local regressor can be linear, quadratic, n-th degree polynomial, neural net, …
- Yields piecewise approximation to surface that typically is more complex than local regressor

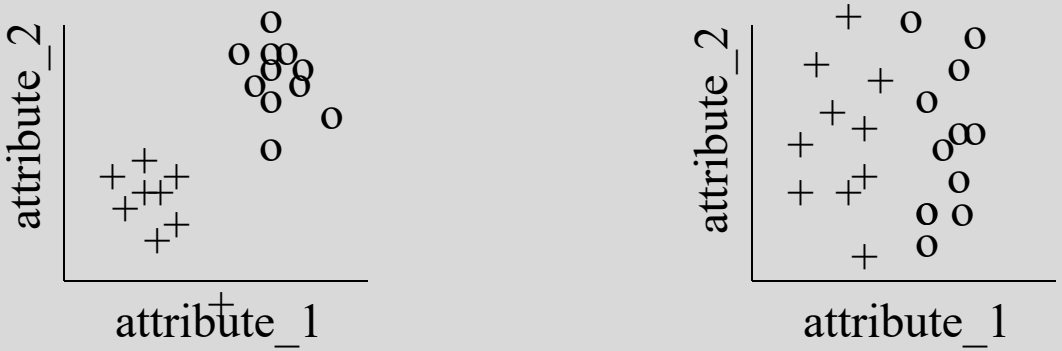
Euclidean Distance

$$D(c1, c2) = \sum_{i=1}^{N} \left(attr_i(c1) - attr_i(c2)\right)^2$$

- gives all attributes equal weight?
  - only if scale of attributes and differences are similar
  - scale attributes to equal range or equal variance
- assumes spherical classes



Euclidean Distance?



- if classes are not spherical?
- if some attributes are more/less important than other attributes?
- if some attributes have more/less noise in them than other attributes?

Weighted Euclidean Distance

$$D(c1, c2) = \sum_{i=1}^{N} w_i \cdot \left(attr_i(c1) - attr_i(c2)\right)^2$$

- large weights => attribute is more important
- small weights => attribute is less important
- zero weights => attribute doesn't matter

- Weights allow kNN to be effective with axis-parallel elliptical classes
- Where do weights come from?

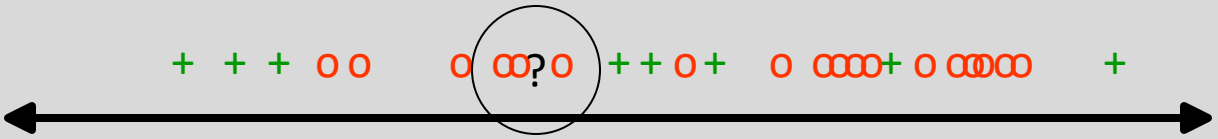
Curse of Dimensionality

- as number of dimensions increases, distance between points becomes larger and more uniform
- if number of relevant attributes is fixed, increasing the number of less relevant attributes may swamp distance

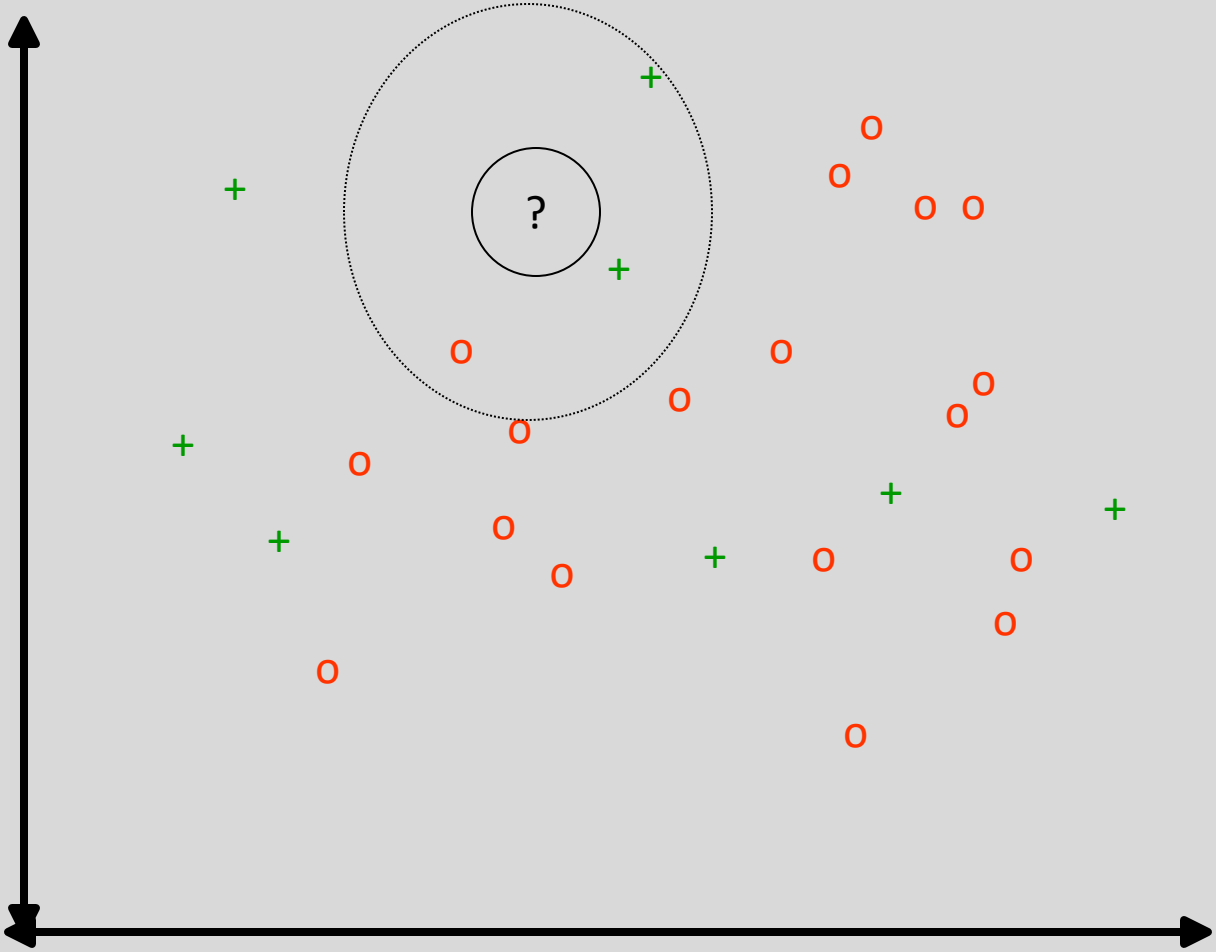
- when more irrelevant than relevant dimensions, distance becomes less reliable
- solutions: larger k or KernelWidth, feature selection, feature weights, more complex distance functions

$$D(c1,c2) = \sum_{i=1}^{\text{relevant}} \left(\text{attr}_i(c1) - \text{attr}_i(c2)\right)^2 + \sum_{j=1}^{\text{irrelevant}} \left(\text{attr}_j(c1) - \text{attr}_j(c2)\right)^2$$

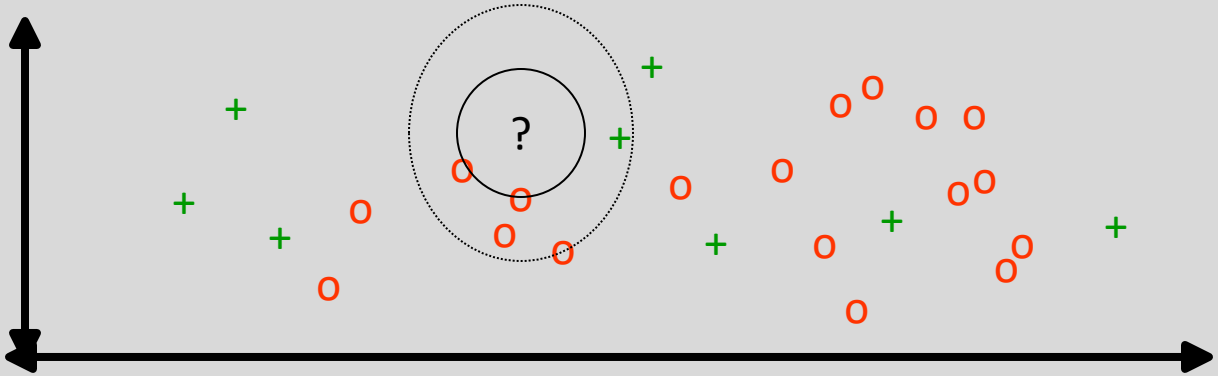
K-NN and irrelevant features



K-NN and irrelevant features



K-NN and irrelevant features



Ways of rescaling for KNN

Normalized L1 distance:

$$\Delta(X,Y) = \sum_{i=1}^{n} \delta(x_i, y_i)|$$

where:

$$\delta(x_i, y_i) = \begin{cases} abs(\frac{x_i - y_i}{max_i - min_i}) & \text{if numeric, else} \\ 0 & \text{if } x_i = y_i \\ 1 & \text{if } x_i \neq y_i \end{cases}$$

Scale by IG:

$$\Delta(X,Y) = \sum_{i=1}^{n} w_i \, \delta(x_i, y_i)$$

$$w_i = H(C) - \sum_{v \in V_i} P(v) \times H(C|v)$$

Modified value distance metric:

$$\delta(v_1, v_2) = \sum_{i=1}^{n} |P(C_i|v_1) - P(C_i|v_2)|$$

Ways of rescaling for KNN

Dot product:

$$\Delta(X,Y) = dot_{max} - \sum_{i=1}^{n} w_i x_i y_i$$

Cosine distance:

$$\Delta(X,Y) = cos_{max} - \frac{\sum_{i=1}^{n} w_i x_i y_i}{\sqrt{\sum_{i=1}^{n} w_i x_i^2 \sum_{i=1}^{n} w_i y_i^2}}$$

TFIDF weights for text: for doc j, feature i: $x_i = tf_{i,j} * idf_i$:

#occur. of term i in doc j

$$\mathrm{tf}_{i,j} = \frac{n_{i,j}}{\sum_k n_{k,j}}$$

$$\mathrm{idf}_i = \log \frac{|D|}{|\{d_j : t_i \in d_j\}|}$$

#docs in corpus

#docs in corpus that contain term i

Combining distances to neighbors

Standard KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$C(y, D') \equiv |\{(x', y') \in D' : y' = y\}|$$

Distance-weighted KNN:

$$\hat{y} = \arg\max_y C(y, Neighbors(x))$$

$$C(y, D') \equiv \sum_{\{(x', y') \in D' : y' = y\}} (SIM(x, x'))$$

$$C(y, D') \equiv 1 - \prod_{\{(x', y') \in D' : y' = y\}} (1 - SIM(x, x'))$$

$$SIM(x, x') \equiv 1 - \Delta(x, x')$$

Advantages of Memory-Based Methods

- Lazy learning: don't do any work until you know what you want to predict (and from what variables!)
  - never need to learn a global model
  - many simple local models taken together can represent a more complex global model
  - better focussed learning
  - handles missing values, time varying distributions, ...
- Very efficient cross-validation
- Intelligible learning method to many users
- Nearest neighbors support explanation and training
- Can use ***any*** distance metric: string-edit distance, …

Weaknesses of Memory-Based Methods

- Curse of Dimensionality:
  - often works best with 25 or fewer dimensions
- Run-time cost scales with training set size
- Large training sets will not fit in memory
- Many MBL methods are strict averagers
- Sometimes doesn't seem to perform as well as other methods such as neural nets
- Predicted values for regression not continuous