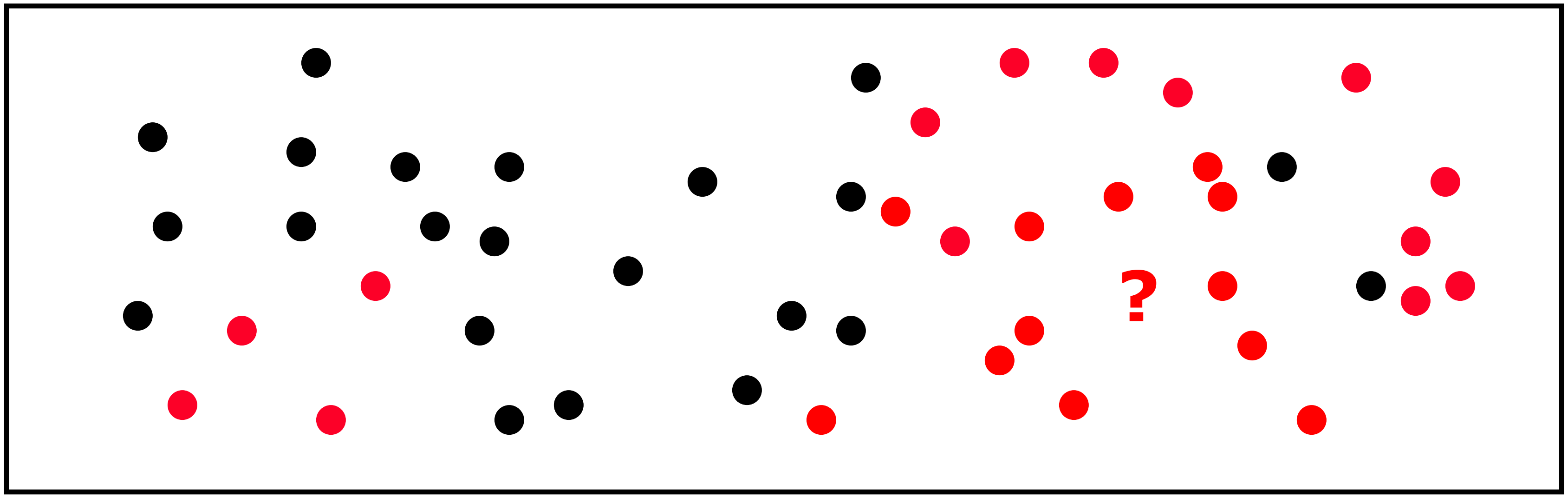
CS340 Machine learning
Lecture 4
K-nearest neighbors

Nearest neighbor classifier

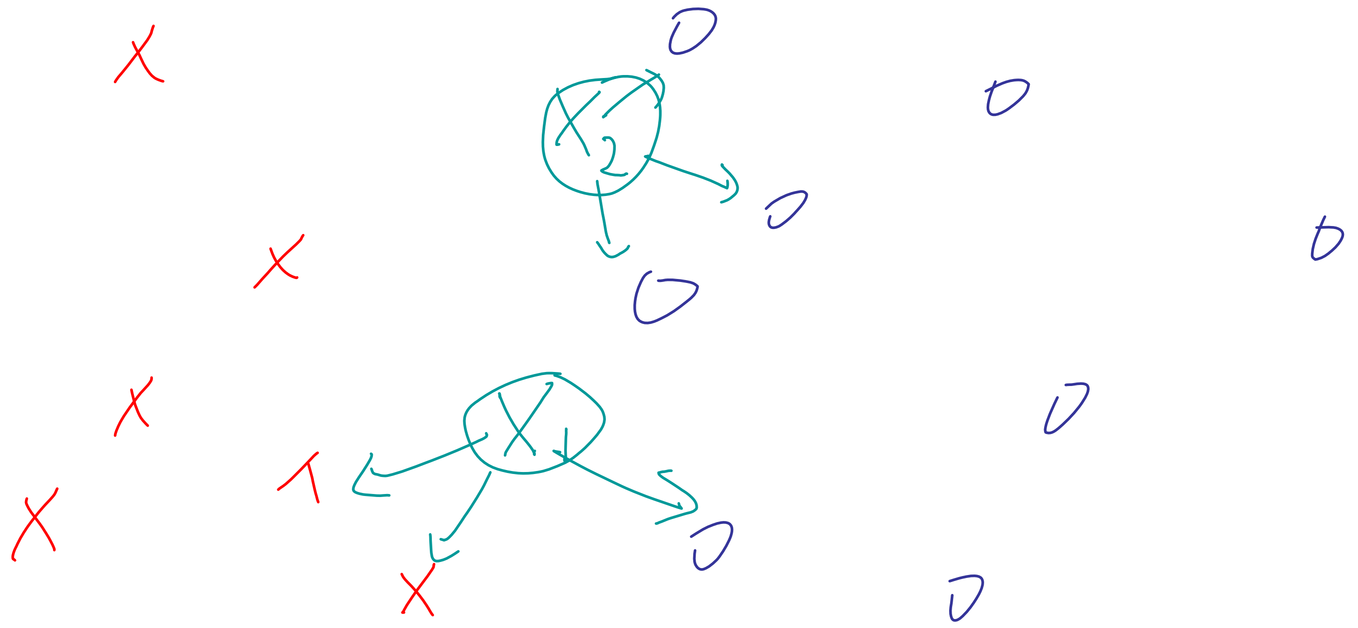
- Remember all the training data (non-parametric classifier)
- At test time, find closest example in training set, and return corresponding label

$$\hat{y}(x) = y_{n^*} \text{ where } n^* = \arg\min_{n \in D} dist(x, x_n)$$



K-nearest neighbor (kNN)

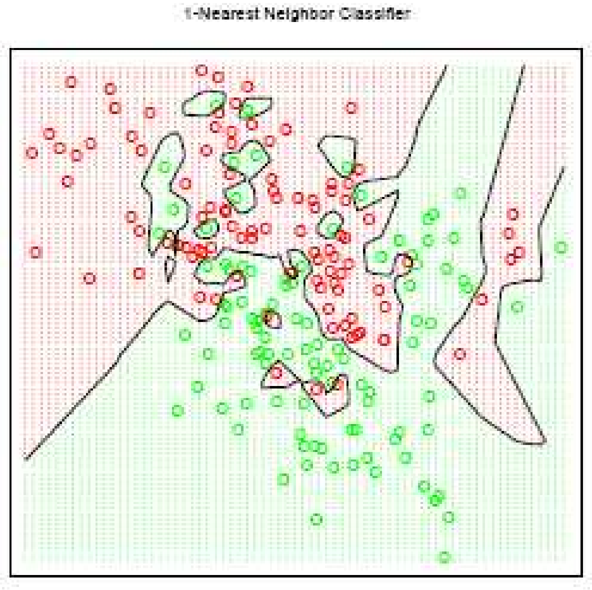
- We can find the K nearest neighbors, and return the majority vote of their labels
- Eg y(X1) = x, y(X2) = o



Effect of K

- K yields smoother predictions, since we average over more data
- K=1 yields y=piecewise constant labeling
- K = N predicts y=globally constant (majority) label

K=1



K=15

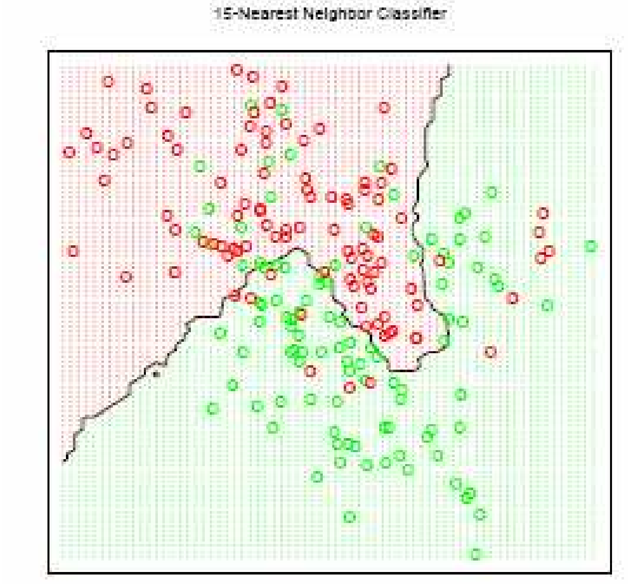
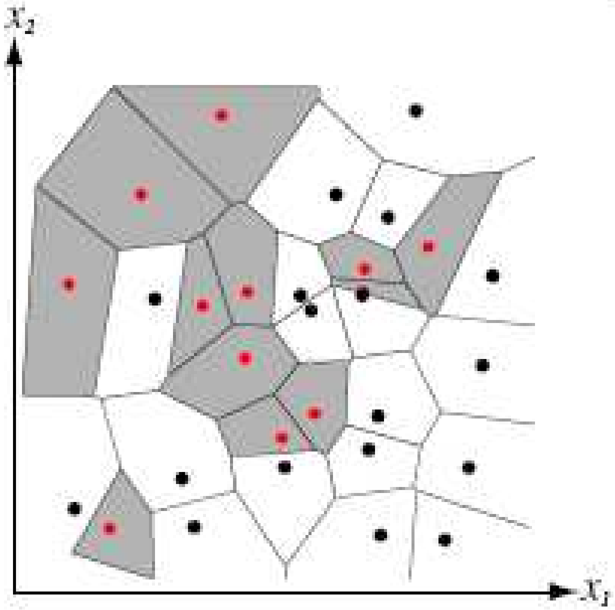


Fig 2.2, 2.3 of HTF01

Decision boundary for K=1

- Decision boundary is piecewise linear; each piece is a hyperplane that is perpendicular to the bisector of pairs of points from different classes (Voronoi tessalation)



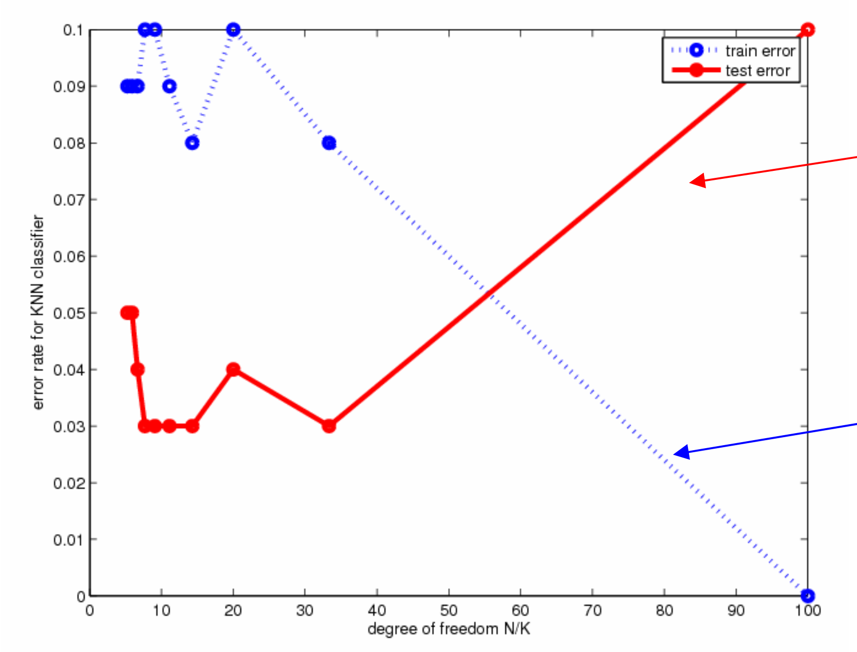
DHS 4.13

Model selection

- Degrees of freedom ≈ N/K, since if neighborhoods don’t overlap, there would be N/K n’hoods, with one label (parameter) each
- K=1 yields zero training error, but badly overfits

K=20

K=1



Test error

Train error

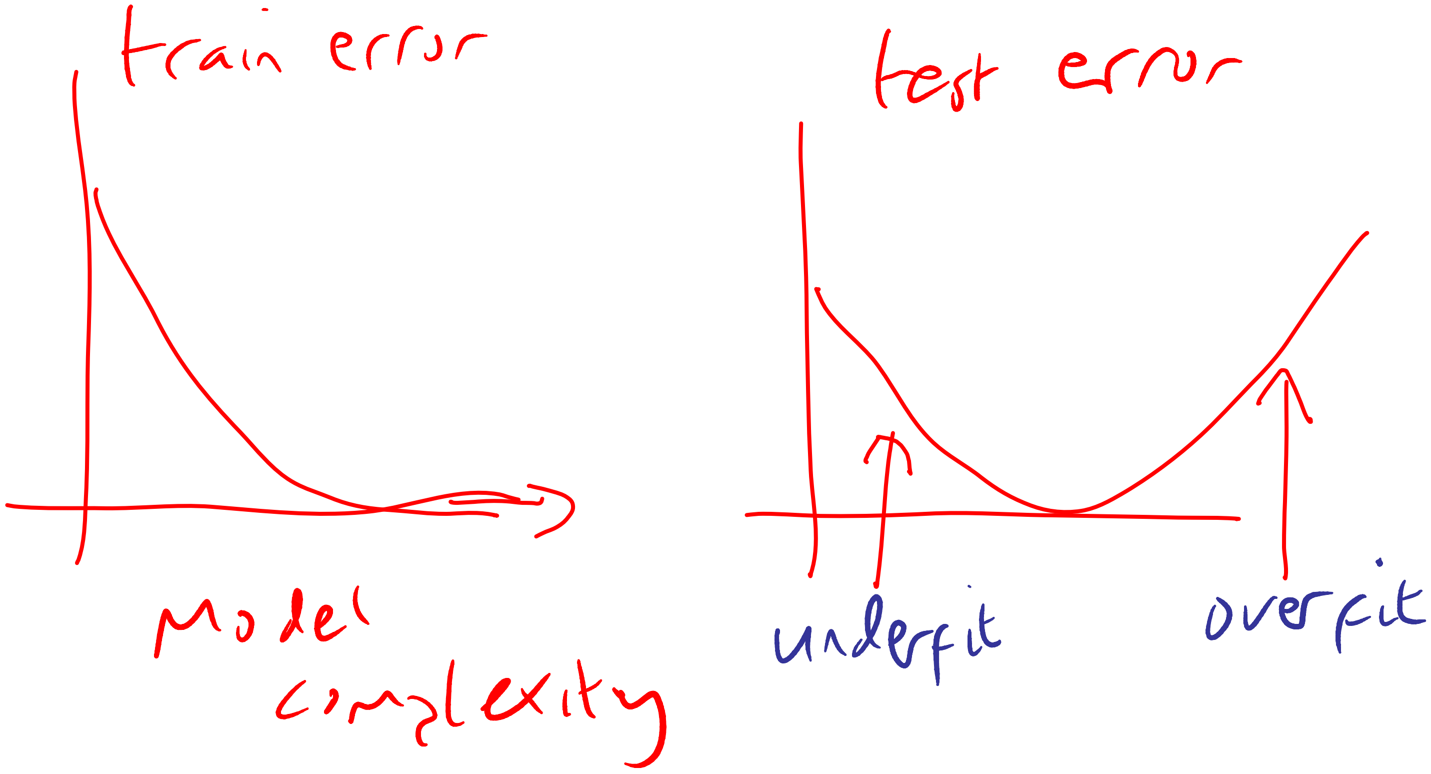
error

dof=5

dof=100

Model selection

- If we use empirical error to choose H (models), we will always pick the most complex model



Approaches to model selection

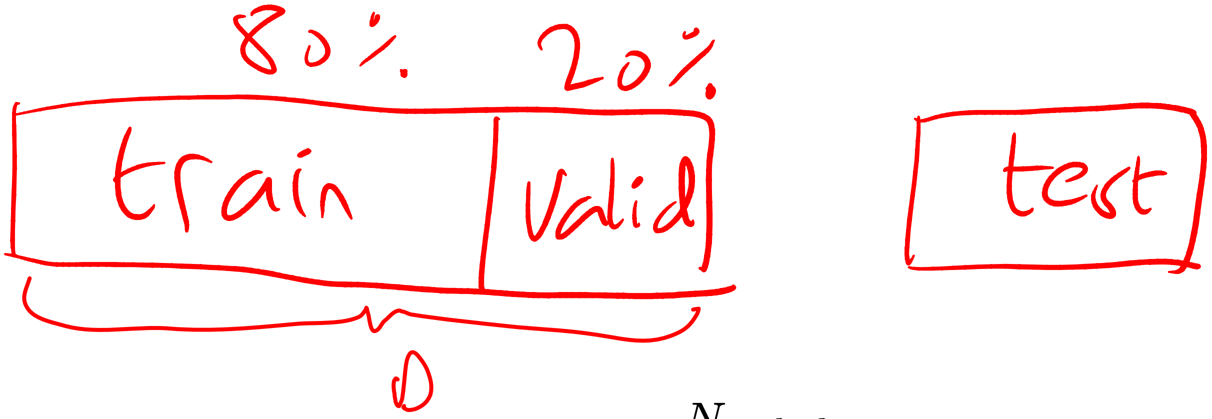
- We can choose the model which optimizes the fit to the training data minus a complexity penalty

$$H^* = \arg\max_H \mathrm{fit}(H|D) - \lambda \mathrm{complexity}(H)$$

- Complexity can be measured in various ways
  - Parameter counting
  - VC dimension
  - Information-theoretic encoding length
- We will see some examples later in class

Validation data

- Alternatively, we can estimate performance of each model on a validation set (not used to fit the model) and use this to select the right H.
- This is an estimate of the generalization error.
- Once we have chosen the model, we refit it to all the data, and report performance on a test set.



$$E[err] \approx \frac{1}{N_{valid}} \sum_{n=1}^{N_{valid}} I(\hat{y}(x_n) \neq y_n)$$

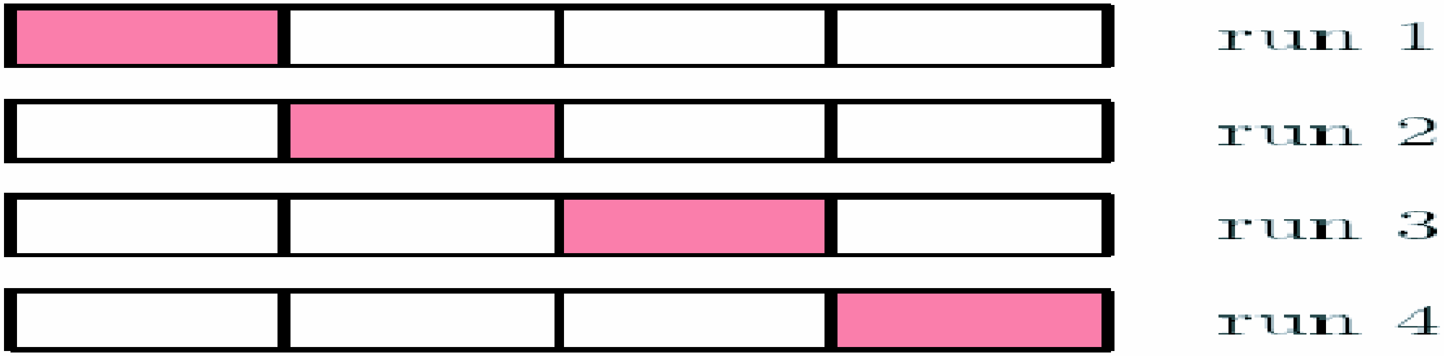
K-fold cross validation

If D is so small that N_{valid} would be an unreliable estimate of the generalization error, we can repeatedly train on all-but-1/K and test on 1/K'th. Typically K=10.

If K=N-1, this is called leave-one-out-CV.

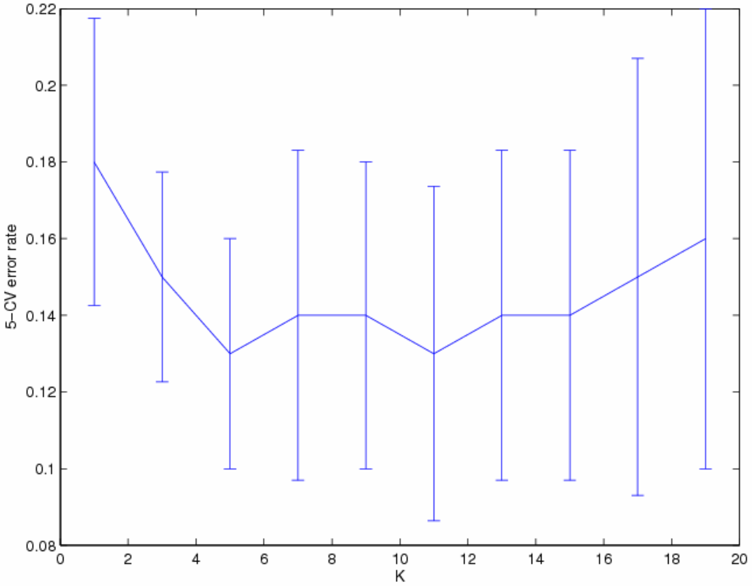
$$e\hat{r}r_k = \frac{1}{N_k} \sum_{n \in fold(k)} I(\hat{y}(x_n) \neq y_n)$$

$$e\hat{r}r = \frac{1}{K} e\hat{r}r_k$$



CV for kNN

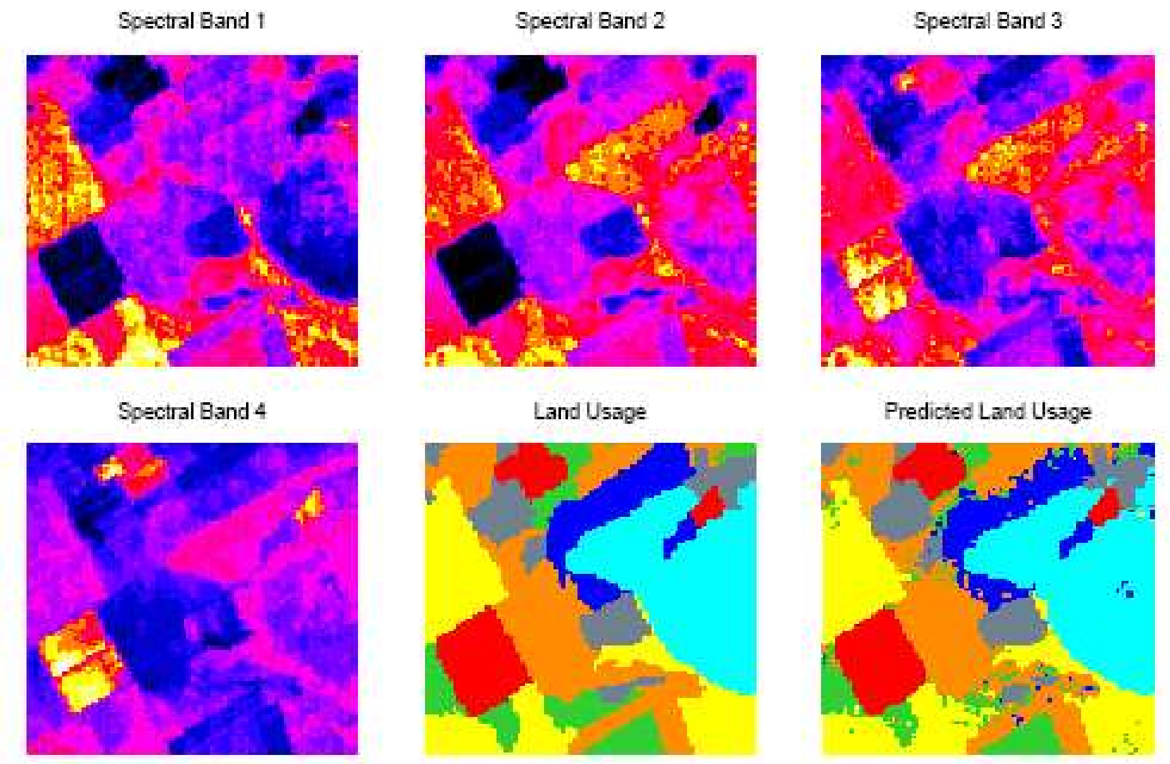
- In hw1, you will implement CV and use it to select K for a kNN classifier
- Can use the “one standard error” rule*, where we pick the simplest model whose error is no more than 1 se above the best.
- For KNN, dof=N/K, so we would pick K=11.

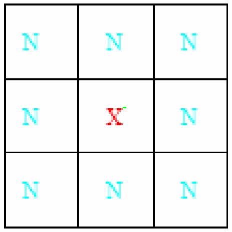


* HTF p216

Application of kNN to pixel labeling

LANDSAT images for an agricultural area in 4 spectral bands;
manual labeling into 7 classes (red soil, cotton, vegetation, etc.);
Output of 5NN using each 3x3 pixel block in all 4 channels (9*4=36 dimensions).
This approach outperformed all other methods in the STATLOG project.





N	N	N
N	X	N
N	N	N

HTF fig 13.6, 13.7

Problems with kNN

- Can be slow to find nearest nbr in high dim space

$$n^* = \arg\min_{n \in D} dist(x, x_n)$$

- Need to store all the training data, so takes a lot of memory
- Need to specify the distance function
- Does not give probabilistic output

Reducing run-time of kNN

- Takes O(Nd) to find the exact nearest neighbor
- Use a branch and bound technique where we prune points based on their partial distances

$$D_r(a,b)^2 = \sum_{i=1}^{r} (a_i - b_i)^2$$

- Structure the points hierarchically into a kd-tree (does offline computation to save online computation)
- Use locality sensitive hashing (a randomized algorithm)

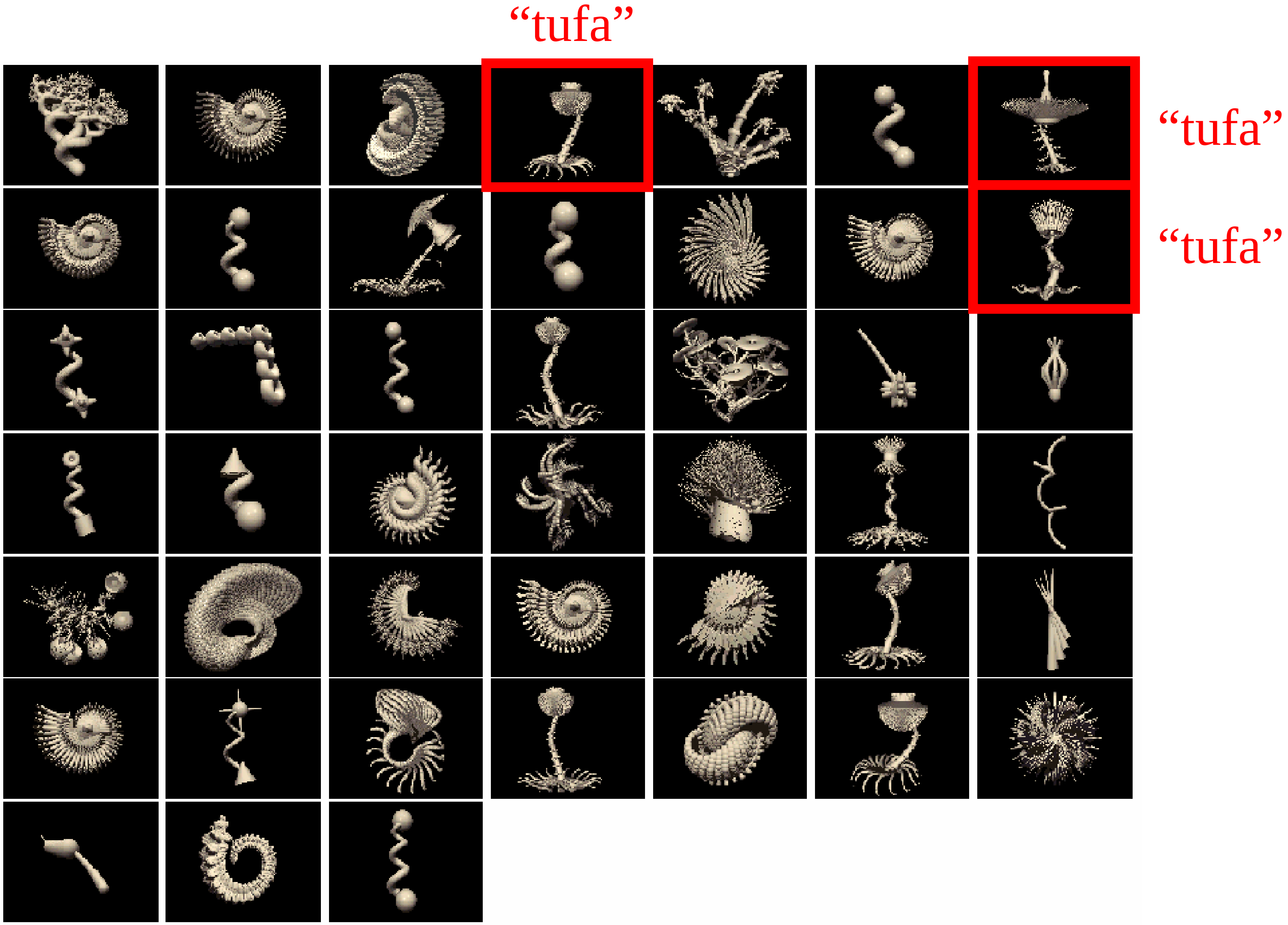
Not on exam

Reducing space requirements of kNN

- Various heuristic algorithms have been proposed to prune/ edit/ condense “irrelevant” points that are far from the decision boundaries
- Later we will study sparse kernel machines that give a more principled solution to this problem

Not on exam

Similarity is hard to define

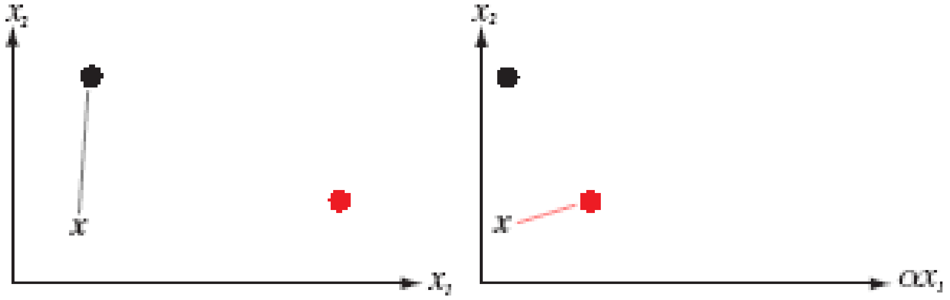


Euclidean distance

- For real-valued feature vectors, we can use Euclidean distance

$$D(u,v)^2 = ||u-v||^2 = (u-v)^T(u-v) = \sum_{i=1}^{d}(u_i - v_i)^2$$

- If we scale x1 by 1/3, NN changes!



Mahalanobis distance

- Mahalanobis distance lets us put different weights on different comparisons

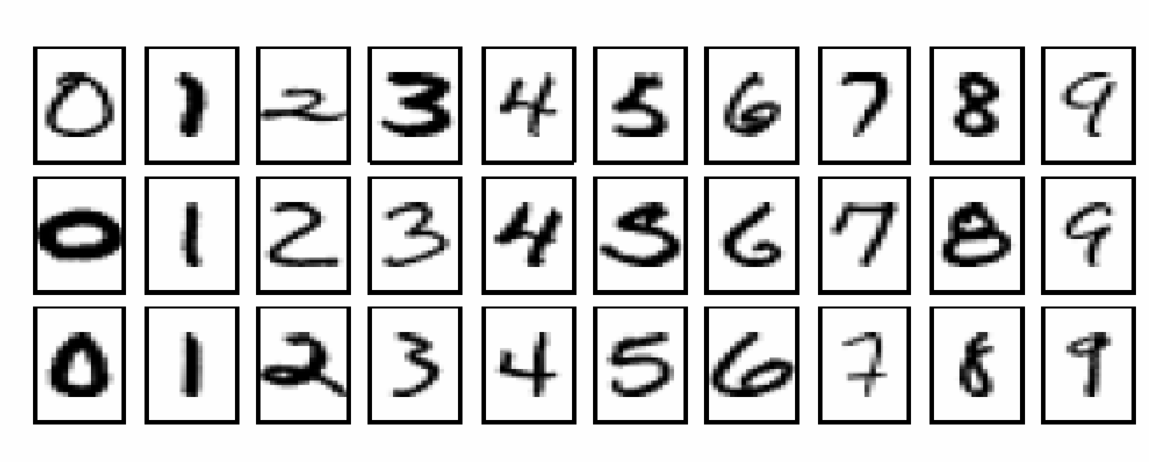
$$\begin{aligned} D(u,v)^2 &= (u-v)^T \Sigma (u-v) \\ &= \sum_i \sum_j (u_i - v_i) \Sigma_{ij} (u_j - v_j) \end{aligned}$$

where Σ is a symmetric positive definite matrix

- Euclidean distance is Σ=I

Error rates on USPS digit recognition

- 7291 train, 2007 test
- Neural net: 0.049
- 1-NN/Euclidean distance: 0.055
- 1-NN/tangent distance: 0.026
- In practice, use neural net, since KNN too slow (*lazy learning*) at test time



HTF 13.9

Problems with kNN

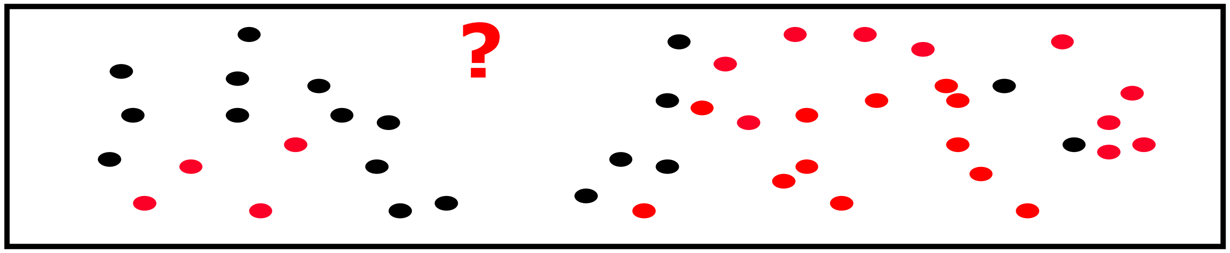
- Can be slow to find nearest nbr in high dim space

$$n^* = \arg\min_{n \in D} dist(x, x_n)$$

- Need to store all the training data, so takes a lot of memory
- Need to specify the distance function
- Does not give probabilistic output

Why is probabilistic output useful?

- A classification function returns a single best guess given an input $\hat{y}(x, \theta) \in \mathcal{Y}$
- A probabilistic classifier returns a probability distribution over outputs given an input $p(y|x, \theta) \in [0, 1]$
- If p(y|x) is near 0.5 (very uncertain), the system may choose not to classify as 0/1 and instead ask for human help



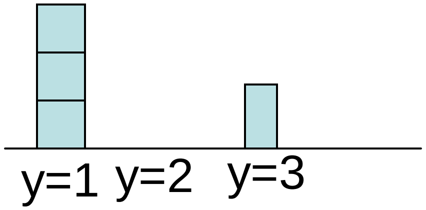
- If we want to combine different predictions p(y|x), we need a measure of confidence
- p(y|x) lets us use likelihood as a measure of fit

Probabilistic kNN

- We can compute the empirical distribution over labels in the K-neighborhood
- However, this will often predict 0 probability due to sparse data

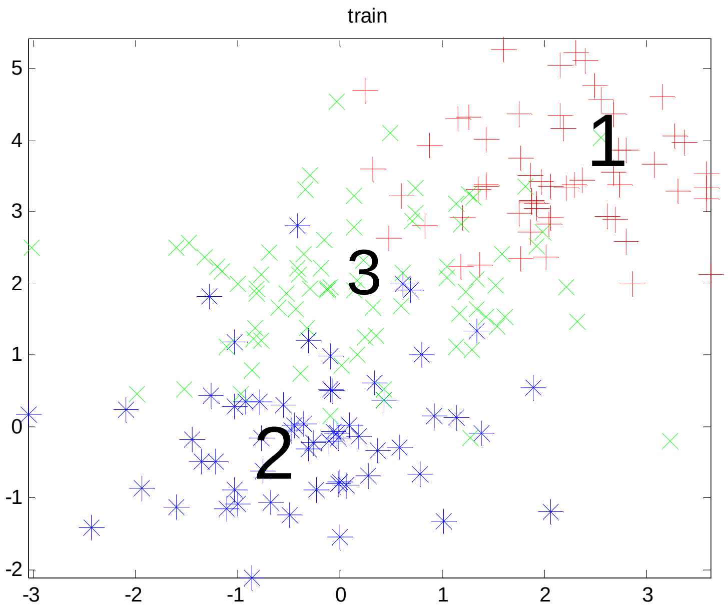
$$p(y|x, D) = \frac{1}{K} \sum_{j \in nbr(x,K,D)} I(y = y_j)$$

K=4, C=3

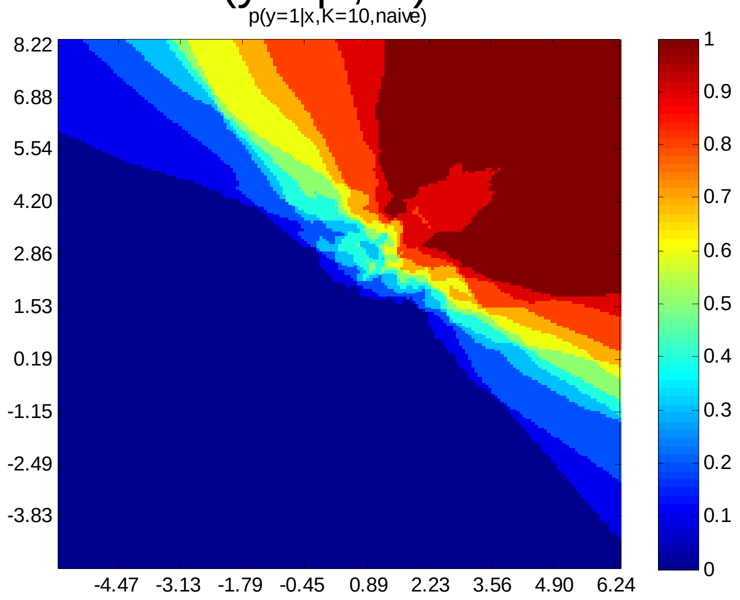


P = [3/4, 0, 1/4]

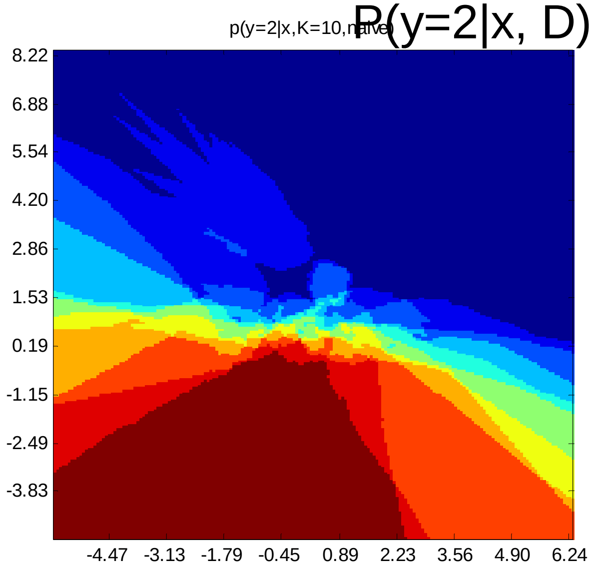
Probabilistic kNN



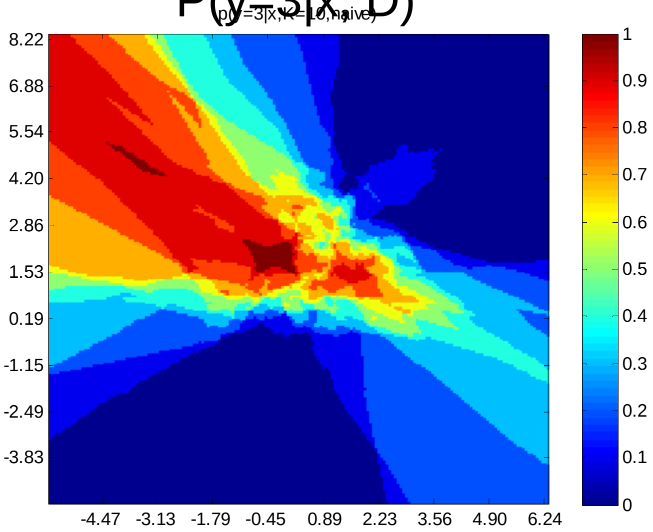
P(y=1|x, D)



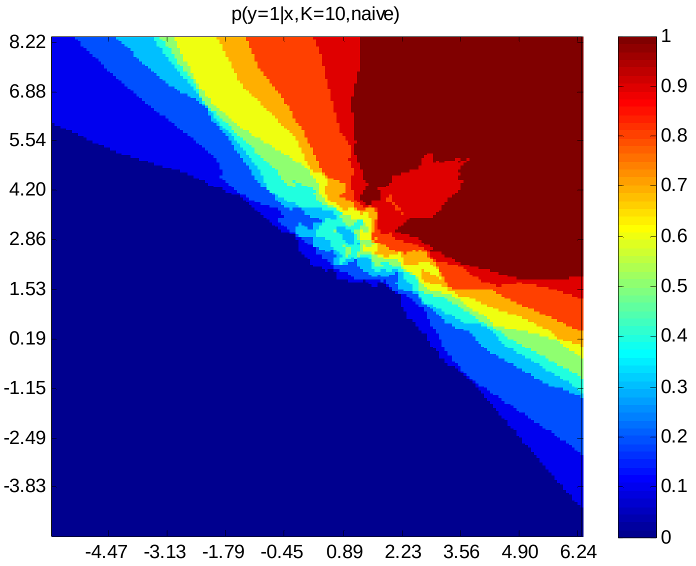
P(y=2|x, D)



P(y=3|x, D)

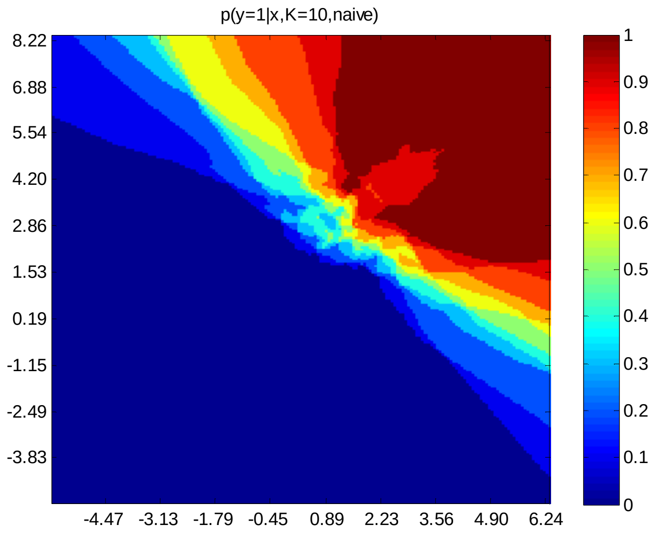


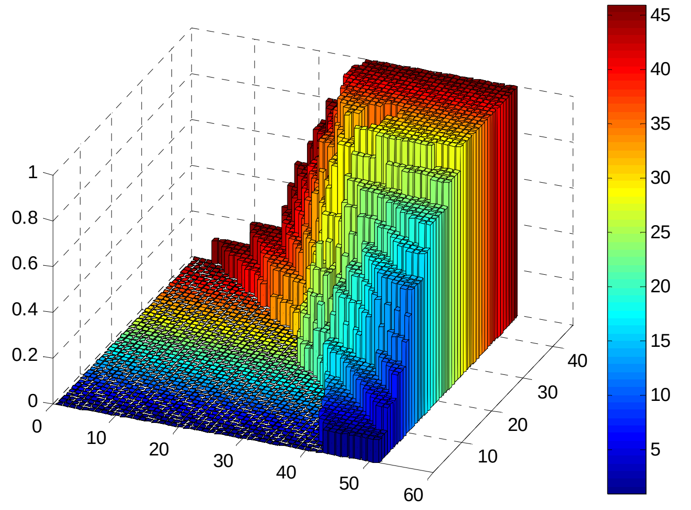
Heatmap of p(y|x,D) for a 2D grid

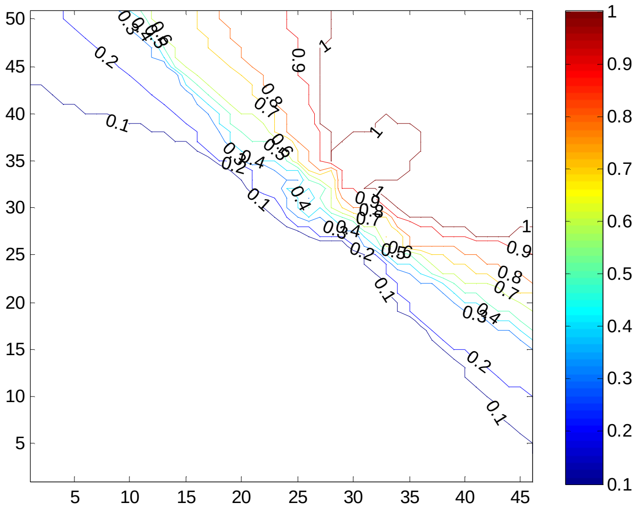


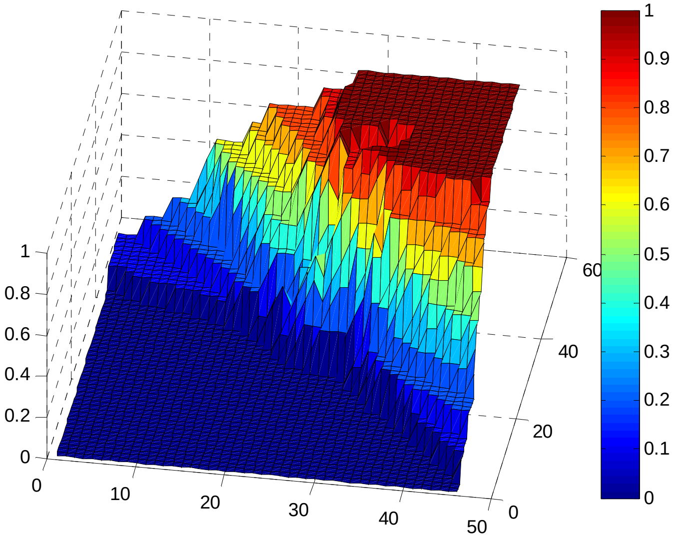
```
xrange = -4.5:0.1:6.25; yrange = -3.85:0.1:8.25;
[X Y] = meshgrid(xrange, yrange); XtestGrid = [X(:) Y(:)];
%[XtestGrid, xrange, yrange] = makeGrid2d(Xtrain, 0.4);
[ypredGrid, yprobGrid]  = knnClassify(Xtrain, ytrain, XtestGrid, K);
HH = reshape(yprobGrid(:,1), [length(yrange) length(xrange)]);
figure(3);clf
imagesc(HH); axis xy; colorbar
```

imagesc, bar3, surf, contour





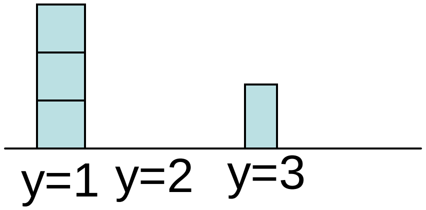




Smoothing empirical frequencies

- The empirical distribution will often predict 0 probability due to sparse data
- We can add *pseudo counts* to the data and then normalize

K=4, C=3



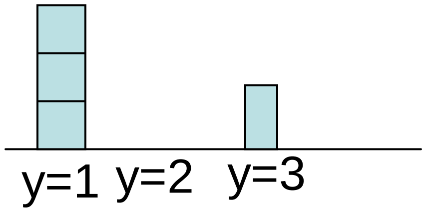
P = [3 + 1, 0 + 1, 1 + 1] / 7 = [4/7, 1/7, 2/7]

Softmax (multinomial logit) function

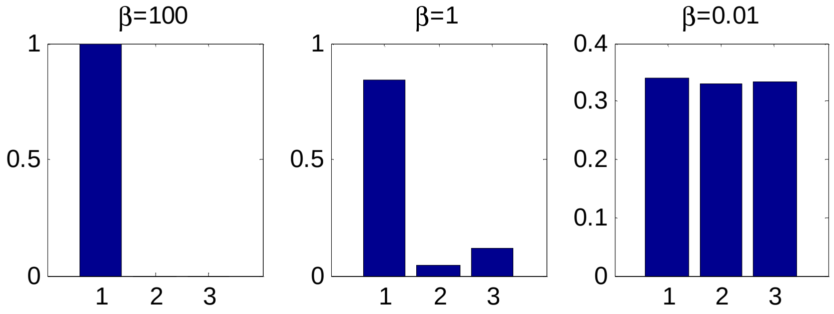
- We can “soften” the empirical distribution so it spreads its probability mass over unseen classes
- Define the softmax with inverse temperature β

$$S(x,\beta)_i = \frac{\exp(\beta x_i)}{\sum_j \exp(\beta x_j)}$$

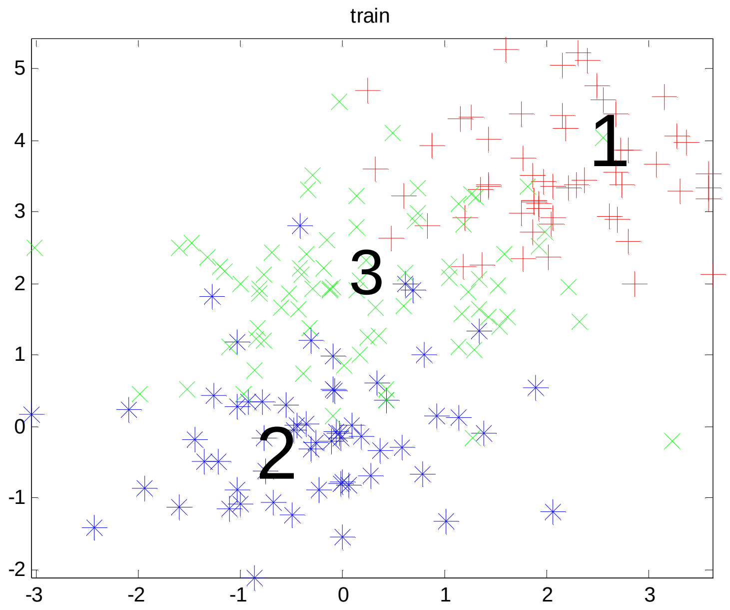
- Big beta = cool temp = spiky distribution
- Small beta = high temp = uniform distribution



X = [3 0 1]



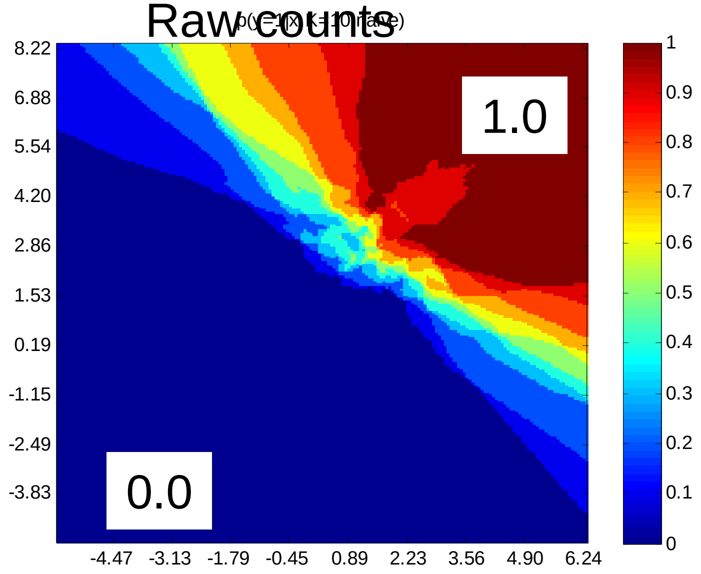
Softened Probabilistic kNN



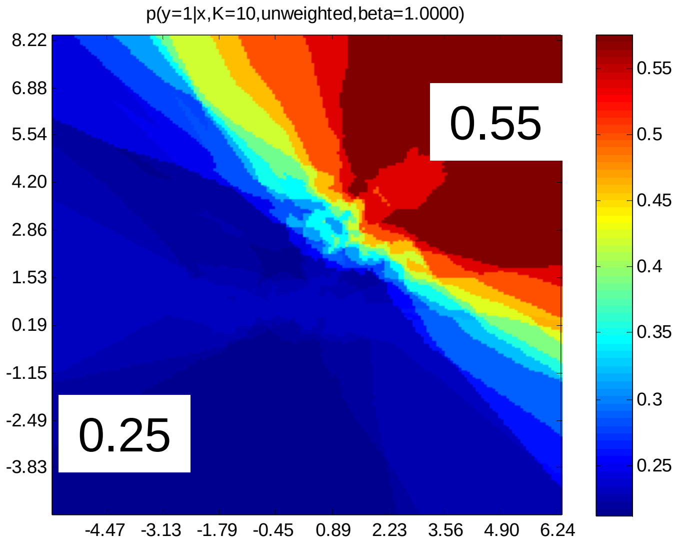
Sum over Knn

$$p(y|x, D, K, \beta) = \frac{\exp[(\beta/K)\sum_{j\sim x} I(y = y_j)]}{\sum_{y'} \exp[(\beta/K)\sum_{j\sim x} I(y' = y_j)]}$$

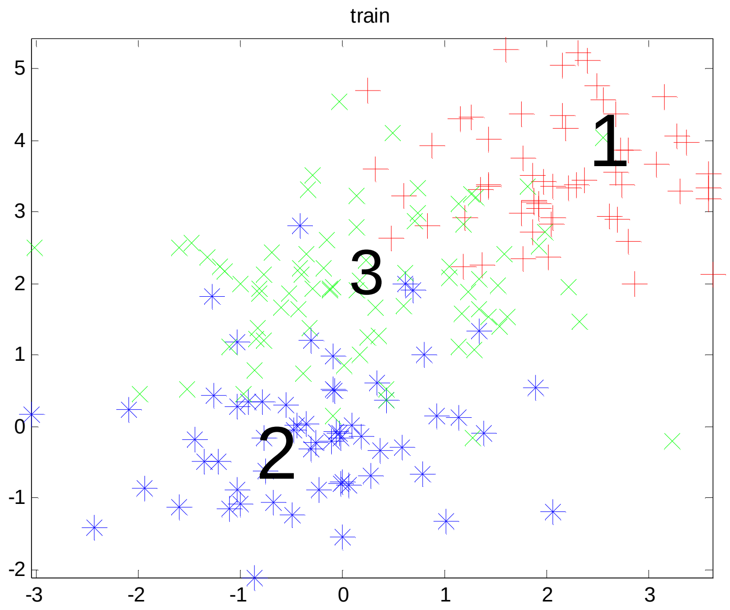
Raw counts



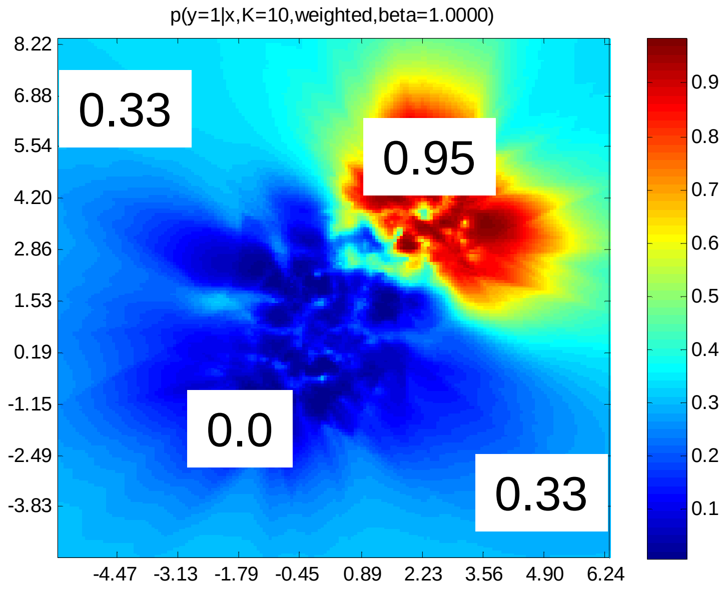
Softmax



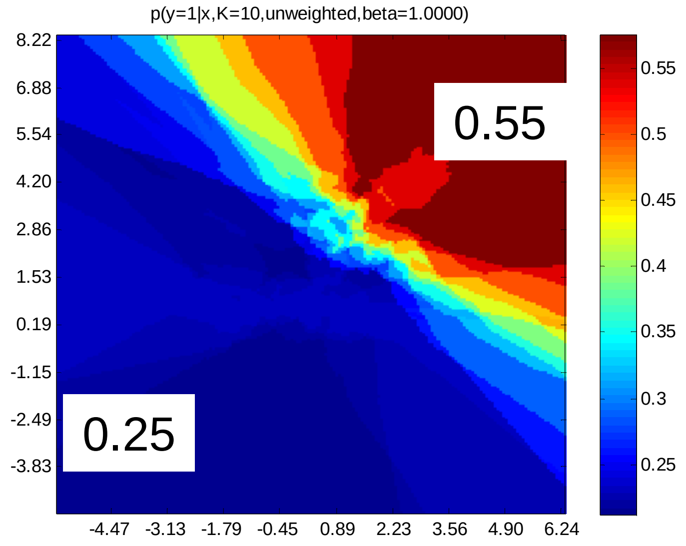
Weighted Probabilistic kNN



Weighted



Softmax



Weighted sum over Knn

$$p(y|x, D, K, \beta) = \frac{\exp[(\beta/K) \sum_{j \sim x} w(x, x_j) I(y = y_j)]}{\sum_{y'} \exp[(\beta/K) \sum_{j \sim x} w(x, x_j) I(y' = y_j)]}$$

Local kernel function

Kernel functions

Any smooth function K such that

$$K(x) \geq 0, \int K(x)dx = 1, \int xK(x)dx = 0 \text{ and } \int x^2 K(x)dx > 0$$

- Epanechnikov quadratic kernel

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right) \quad D(t) = \begin{cases} \frac{3}{4}(1-t^2) & \text{if } |t| \leq 1; \\ 0 & \text{otherwise.} \end{cases}$$

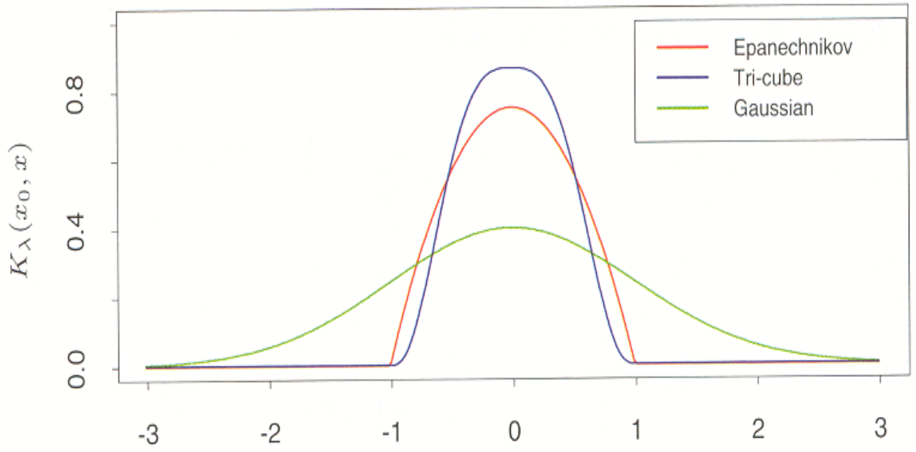
λ = bandwidth

- tri-cube kernel

$$K_\lambda(x_0, x) = D\left(\frac{|x - x_0|}{\lambda}\right) \quad D(t) = \begin{cases} (1-|t|^3)^3 & \text{if } |t| \leq 1; \\ 0 & \text{otherw} \end{cases}$$

- Gaussian kernel

$$K_\lambda(x_0, x) = \frac{1}{\sqrt{2\pi}\lambda} \exp\left(-\frac{(x - x_0)^2}{2\lambda^2}\right)$$



Kernel characteristics

Compact support – vanishes beyond a finite range (Epanechnikov, tri-cube)
Everywhere differentiable (Gaussian, tri-cube)

HTF 6.2

Kernel functions on structured objects

- Rather than defining a feature vector x, and computing Euclidean distance D(x, x'), sometimes we can directly compute distance between two *structured objects*
- Eg string/graph matching using dynamic programming