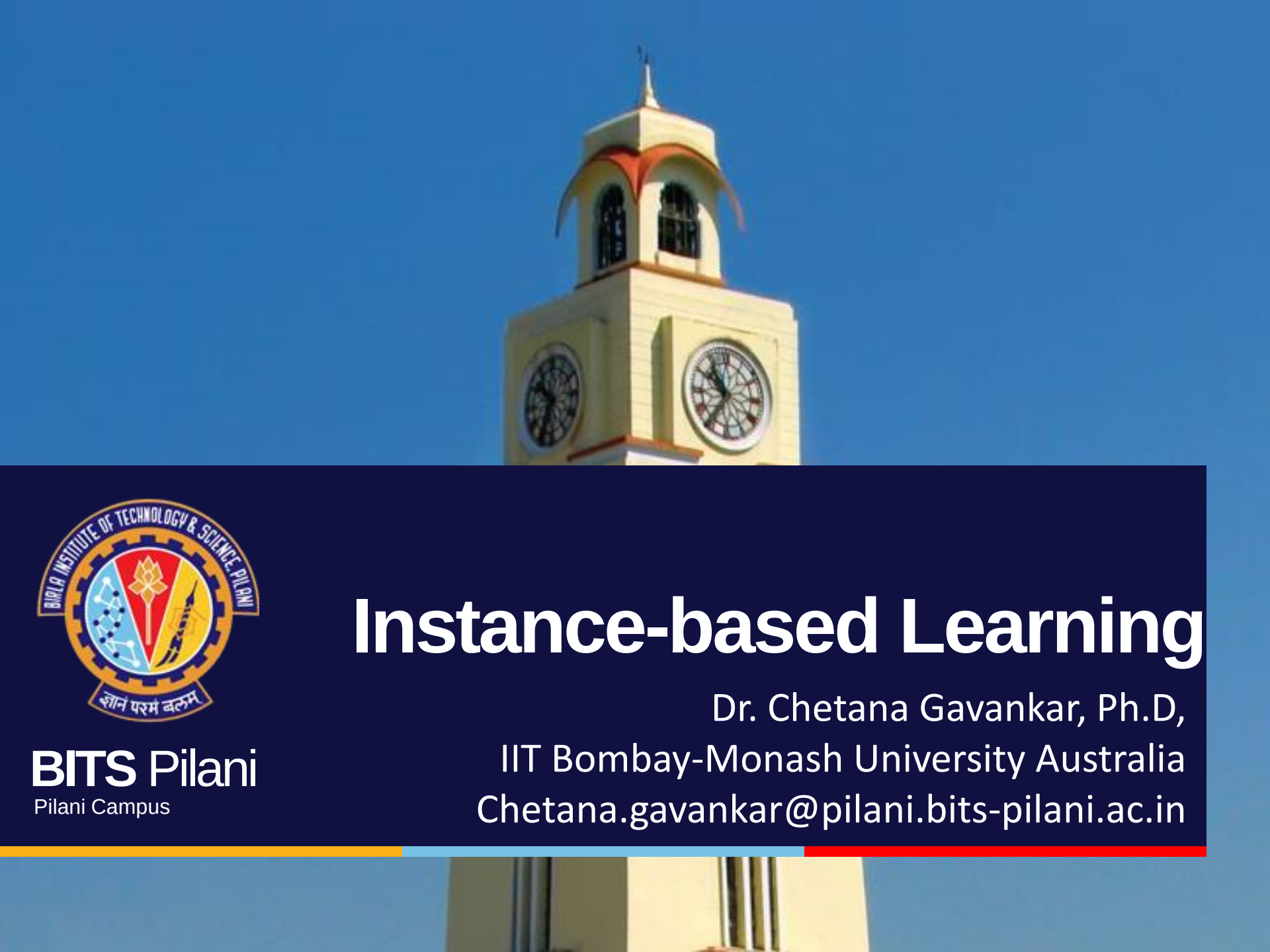


Instance-based Learning

Dr. Chetana Gavankar, Ph.D,
IIT Bombay-Monash University Australia
Chetana.gavankar@pilani.bits-pilani.ac.in

BITS Pilani
Pilani Campus

innovate achieve lead

BITS Pilani

Pilani Campus

Text Book(s)

T1	Christopher Bishop: Pattern Recognition and Machine Learning, Springer International Edition
T2	Tom M. Mitchell: Machine Learning, The McGraw-Hill Companies, Inc..

These slides are prepared by the instructor, with grateful acknowledgement of Prof. Tom Mitchell, Prof.. Burges, Prof. Andrew Moore and many others who made their course materials freely available online.

Topics to be covered

- Instance based learning
- K-Nearest Neighbour Learning
- Locally Weighted Regression (LWR) Learning
- Radial Basis Functions

k-Nearest Neighbor Classifier

- Nearest Neighbour classifier is an instance based classifier
- 'lazy learning', as learning is postponed until a new instance is encountered
- Constructs a local approximation to the target function, applicable in the neighbourhood of new instance
- Suitable in cases where target function is complex over the entire input space, but easily describable in local approximations
- Real world applications found in recommendation systems (amazon).
- Caveat is the high cost of classification, which happens at the time of processing rather than before hand (there's no training phase)

Instance Based Learning

Key idea: just store all training examples $<x_i, f(x_i)>$

Nearest neighbor:

- Given query instance x_q, first locate nearest training example x_n, then estimate $f^*(x_q)=f(x_n)$

K-nearest neighbor:

- Given x_q, take vote among its k nearest neighbors (if discrete-valued target function)
- Take mean of f values of k nearest neighbors (if real-valued) $f^*(x_q)=\sum_{i=1}^{k} f(x_i)/k$

When to Consider Nearest Neighbors

- Instances map to points in R^N
- Less than 20 attributes per instance
- Lots of training data

Advantages:

- Training is very fast
- Learn complex target functions
- Do not lose information

Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

k-Nearest Neighbor Classifier

- Considers all instances as members of n-dimensional space
- Nearest neighbours of an instance is determined based on Euclidean distance
- Distance between two n-dimensional instances x_i and x_j is given by:

$$d(x_i, x_j) \equiv \sqrt{\sum_{r=1}^{n} (a_r(x_i) - a_r(x_j))^2}$$

- For nearest neighbour classifier, target function can be discrete or continuous

Distance Measures : Special Cases of Minkowski

- $h = 1$: Manhattan (city block, L_1 norm) distance

$$d(i,j) = |x_{i_1} - x_{j_1}| + |x_{i_2} - x_{j_2}| + \ldots + |x_{i_p} - x_{j_p}|$$

- $h = 2$: (L_2 norm) Euclidean distance

$$d(i,j) = \sqrt{(|x_{i_1} - x_{j_1}|^2 + |x_{i_2} - x_{j_2}|^2 + \ldots + |x_{i_p} - x_{j_p}|^2)}$$

- $h \to \infty$. "supremum" (L_{max} norm, L_∞ norm) distance.
  - This is the maximum difference between any component (attribute) of the vectors

$$d(i,j) = \lim_{h \to \infty} \left(\sum_{f=1}^{p} |x_{if} - x_{jf}|^h \right)^{\frac{1}{h}} = \max_{f}^{p} |x_{if} - x_{jf}|$$

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Standardizing Numeric Data

- X: raw score to be standardized, μ: mean of the population, σ: standard deviation
- the distance between the raw score and the population mean in units of the standard deviation
- negative when the raw score is below the mean, "+" when above

Where

$$z = \frac{x - \mu}{\sigma}$$

BITS Pilani, Deemed to be University under Section 3 of UGC Act, 1956

Multiple Features

$$y_n = w_0 + w_1 f_{n1} + w_2 f_{n2} + w_3 f_{n3} + w_4 f_{n4}$$

Size (feet²) f1	Number of bedrooms f2	Number of floors f3	Age of home (years) f4	Price ($1000) y
2104	5	1	45	460
1416	3	2	40	232
1534	3	2	30	315
852	2	1	36	178
…	…	…	…	…

Feature scaling

Feature scaling of features x_i consists of rescaling the range of features to scale the range in [0, 1] or [−1, 1] (Do not apply to $x_0 = 1$)

E.g. $x_1 = \frac{size - 1000}{2000}$

Average value of x1

Maximum value of x1 – min value of x1

$$x_2 = \frac{\#bedrooms - 2}{5}$$

Discrete and Continuous-valued function

- **discrete-valued target function:**
  - $f : \Re^n \rightarrow V$ where V is the finite set $\{v_1, v_2, ..., v_s\}$
  - the target function value is the most common value among the k nearest training examples

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = (a == b)$

- **continuous-valued target function:**
  - algorithm has to calculate the mean value instead of the most common value
  - $f : \Re^n \rightarrow \Re$

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} f(x_i)}{k}$$

k-Nearest Neighbor Classifier

Training algorithm:

- For each training example $\langle x, f(x) \rangle$, add the example to the list *training_examples*

Classification algorithm:

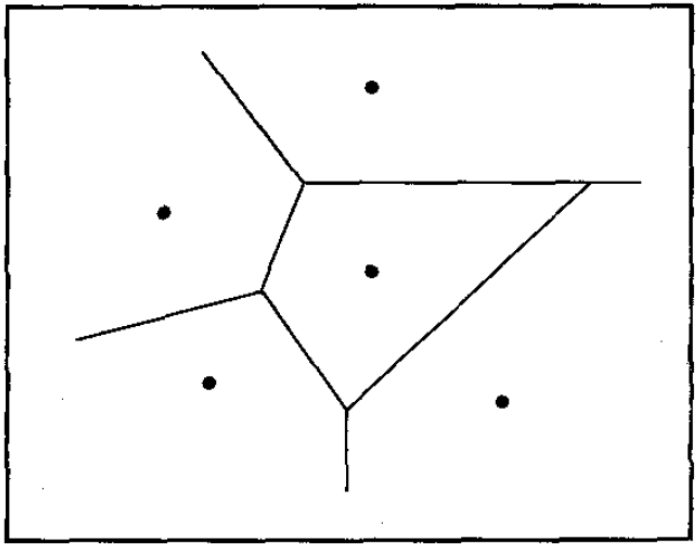
- Given a query instance x_q to be classified,
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

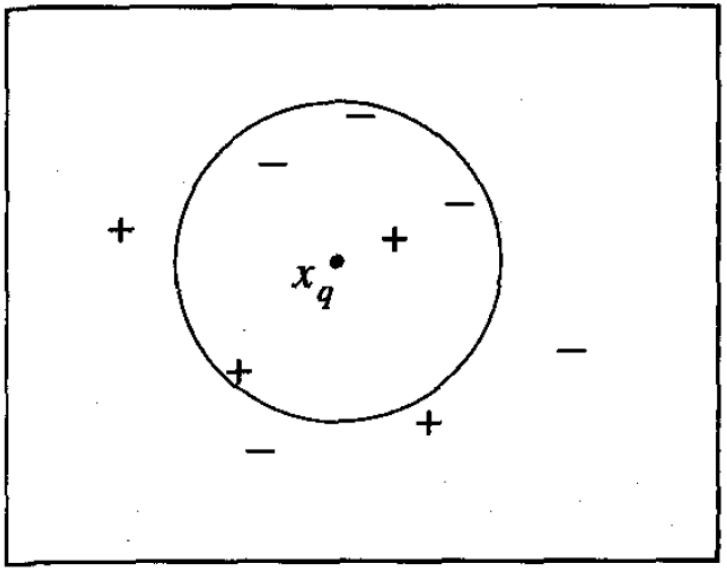
where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.

* It can be used for Regression as well.

k-NN examples



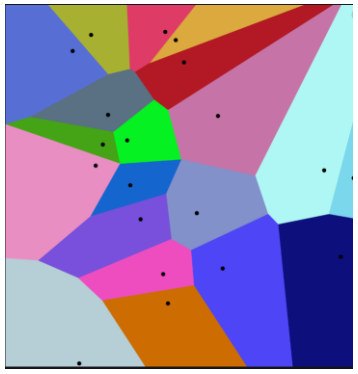
K=1



K=5

Voronoi Diagram

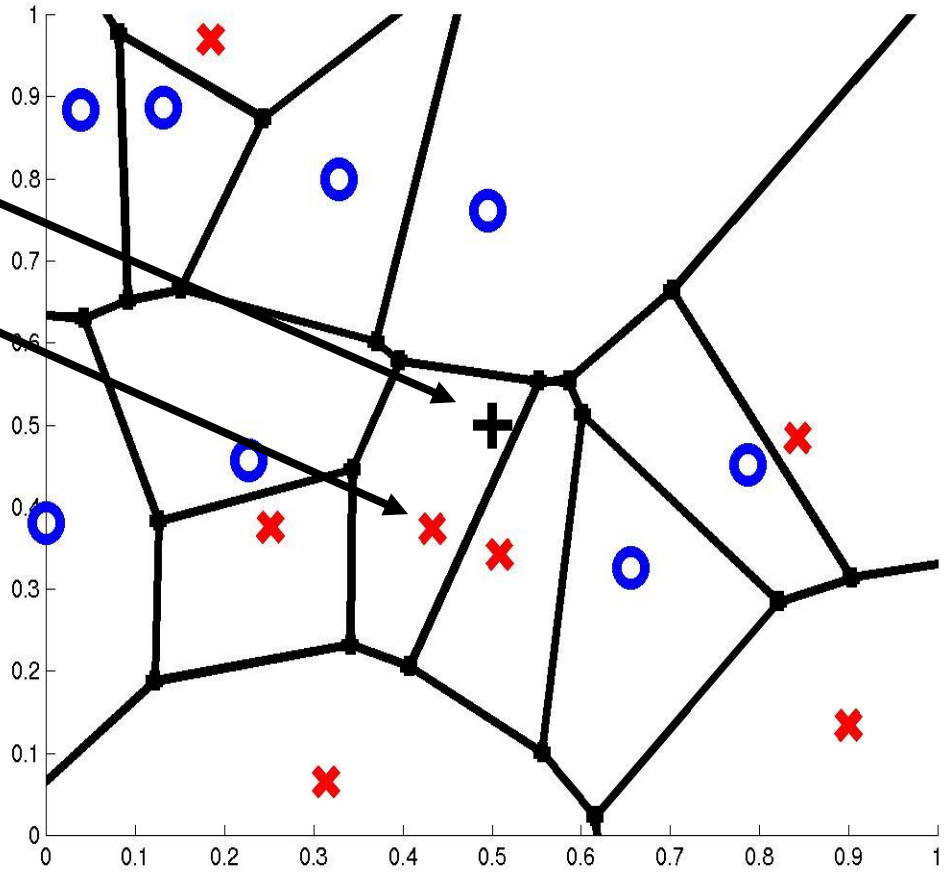
- It is a partition of a plane into regions close to each of a given set of objects.



Voronoi Diagram

query point q_f

nearest neighbor q_i

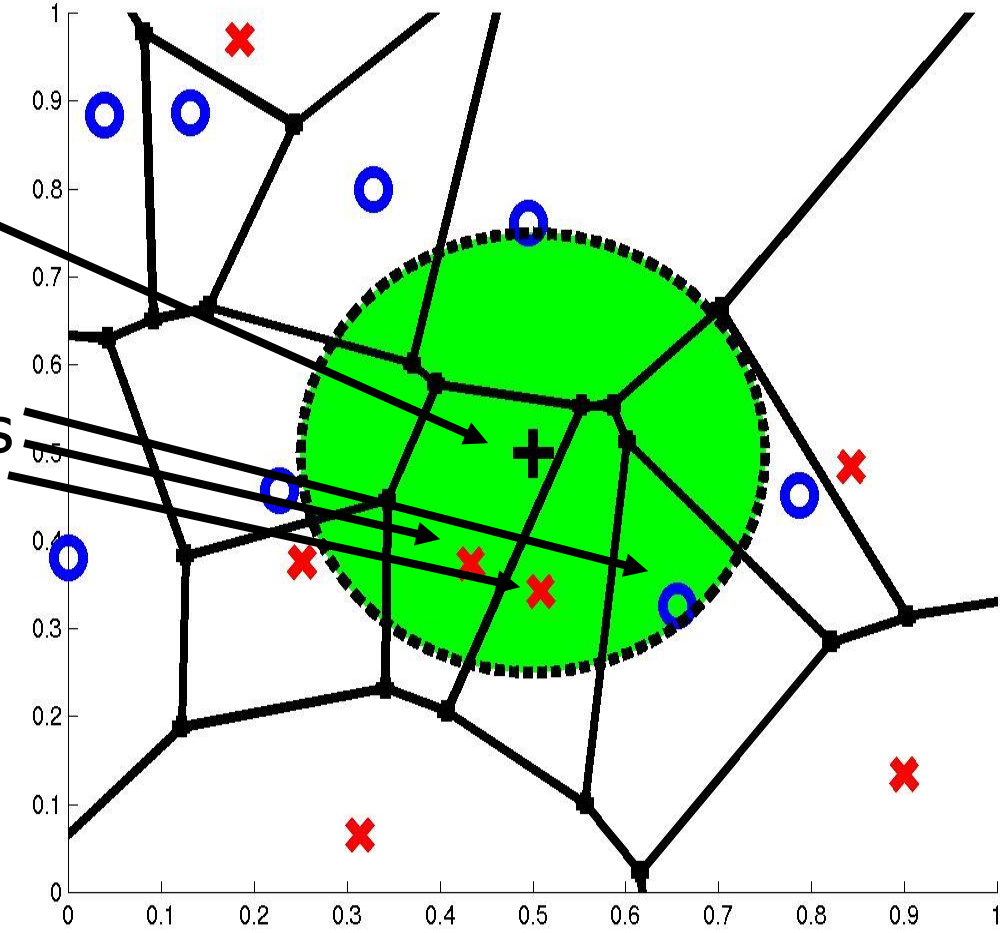


3-Nearest Neighbors

query point q_f

3 nearest neighbors

2x,1o

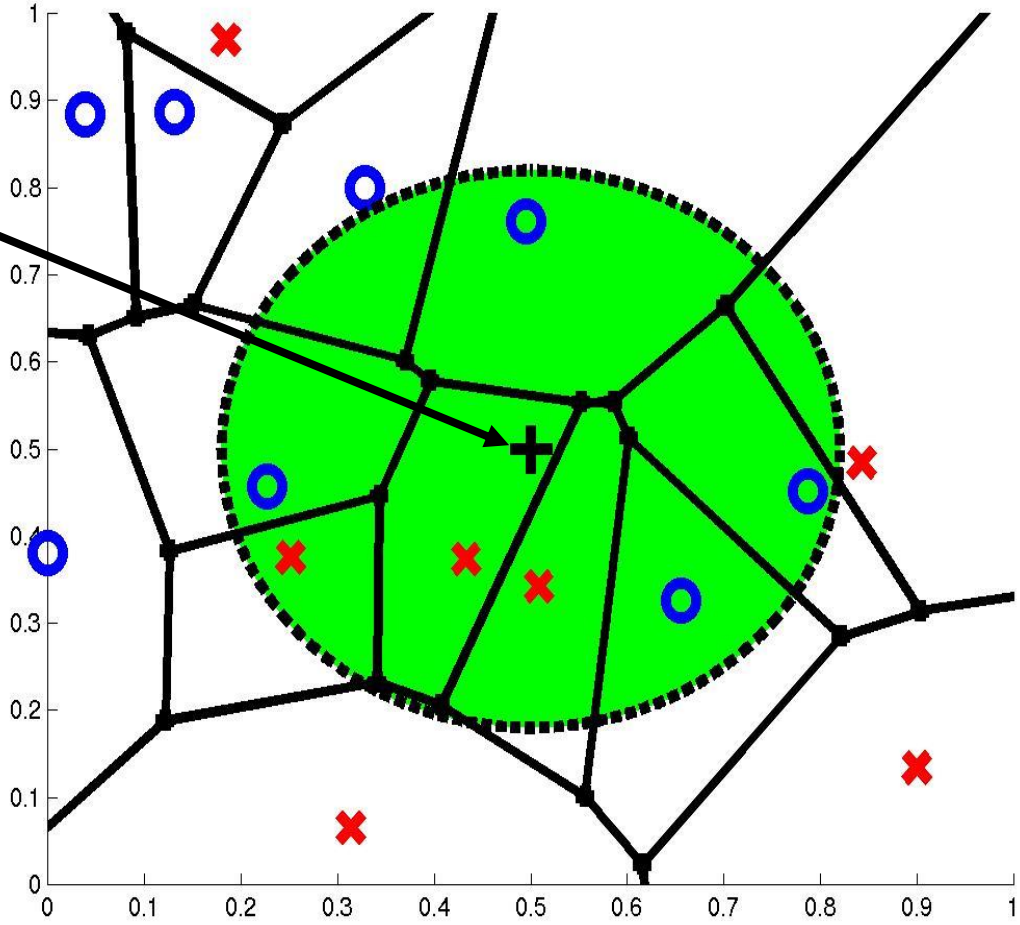


7-Nearest Neighbors

query point q_f

7 nearest neighbors

3x,4o

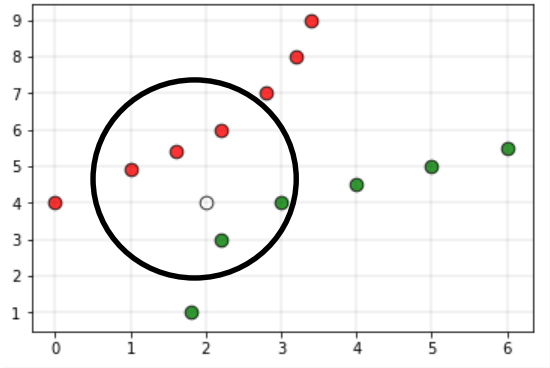


Various issues that affect the performance of kNN:

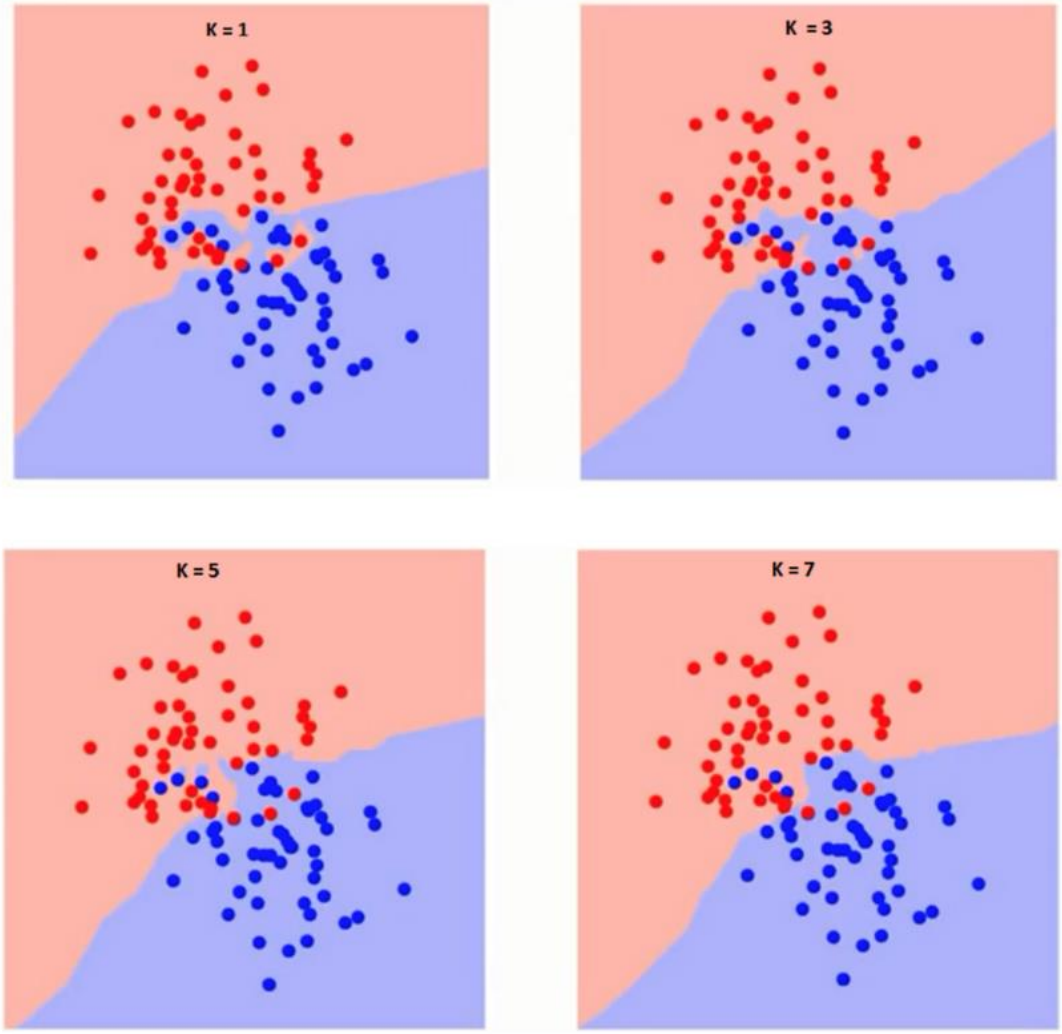
Performance of a classifier largely depends on the of the hyperparameter k

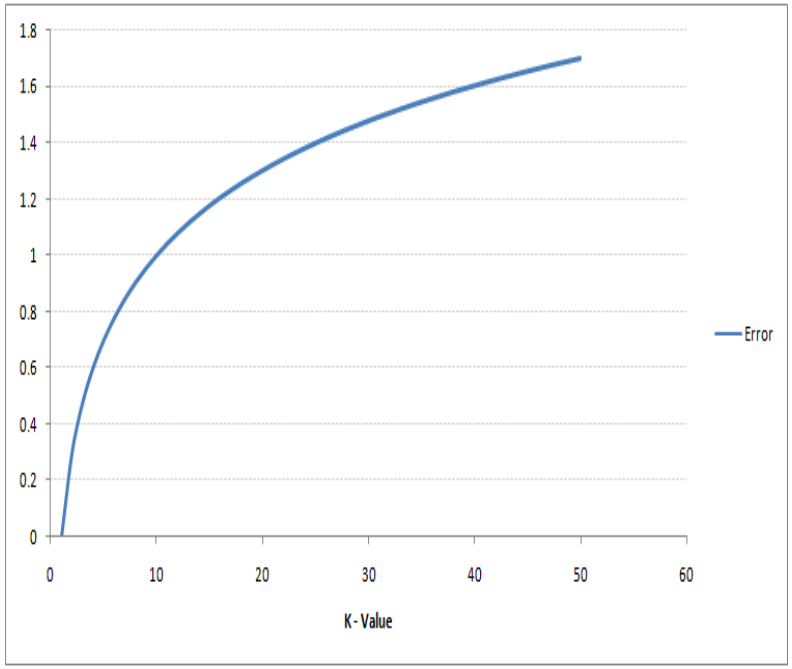
- Choosing smaller values for K, noise can have a higher influence on the result.
- Larger values of k are computationally expensive

Assigning the class labels can be tricky. For example, in the below case, for (k=5) the point is closer to ‘green’ classification, but gets classified as ‘red’ due to higher red votes/majority voting to ‘red’



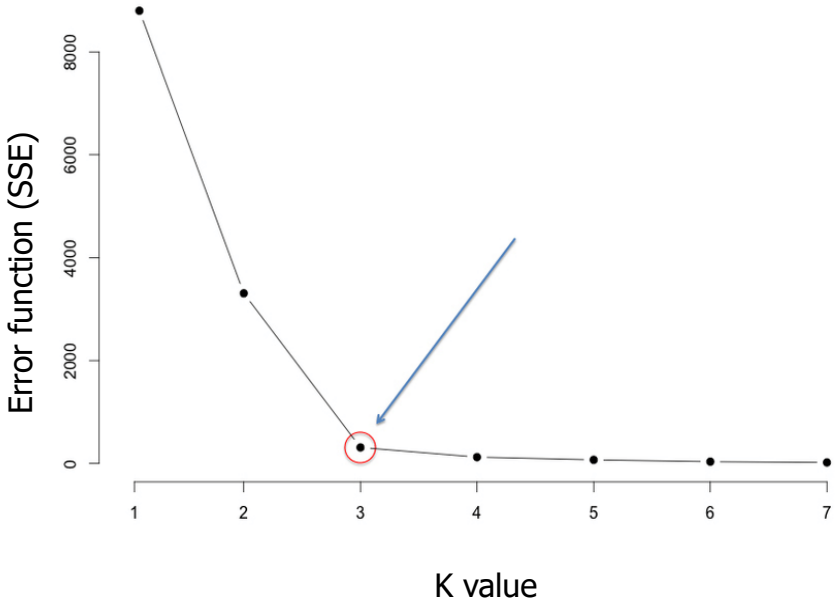
Finding optimal k for kNN classifiers





Finding K - Elbow method

- Compute sum of squares error (SSE) or any other error function for varying values of K (1 to a reasonable X) and plot against K
- In the plot, the elbow (see pic) gives the value of K beyond which the error function plot almost flattens
- As K approaches the total number of instances in the set, error function drops down to ‘0’, but beyond optimal K, the model becomes too generic



Distance weighted nearest neighbor

- contribution of each of the k nearest neighbors is weighted accorded to their distance to x_q
  - **discrete-valued target functions**

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} w_i \delta(v, f(x_i))$$

where $w_i \equiv \frac{1}{d(x_q, x_i)^2}$ and $\hat{f}(x_q) = f(x_i)$ if $x_q = x_i$

  - **continuous-valued target function:**

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i}$$

Distance Weighted k-NN

Give more weight to neighbors closer to the query point

– $f^*(x_q) = \Sigma_{i=1}^{k} w_i f(x_i) / \Sigma_{i=1}^{k} w_i$
; $w_i = K(d(x_q, x_i))$

and

– $d(x_q, x_i)$ is the distance between x_q and x_i

Variation: Instead of only k-nearest neighbors use all training examples (Shepard's method)

Distance Weighted Average

Weighting the data

- $f^*(x_q) = \Sigma_i f(x_i) K(d(x_i,x_q))/ \Sigma_i K(d(x_i,x_q))$

Relevance of a data point (xi,f(xi)) is measured by calculating the distance d(xi,xq) between the query xq and the input vector xi

Weighting the error criterion

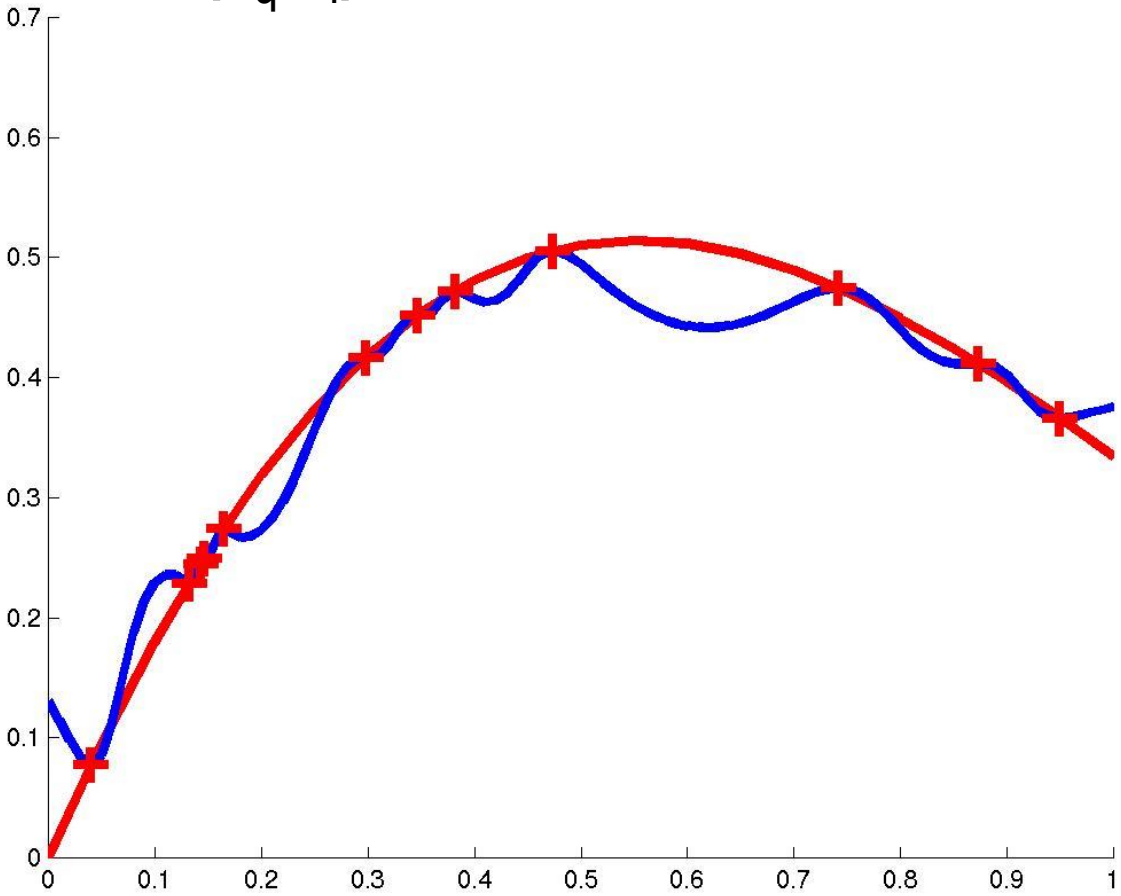
- $E(x_q) = \Sigma_i (f^*(x_q)-f(x_i))^2 K(d(x_i,x_q))$

Best estimate $f^*(x_q)$ will minimize the cost $E(x_q)$, therefore

$$\partial E(x_q)/\partial f^*(x_q)=0$$

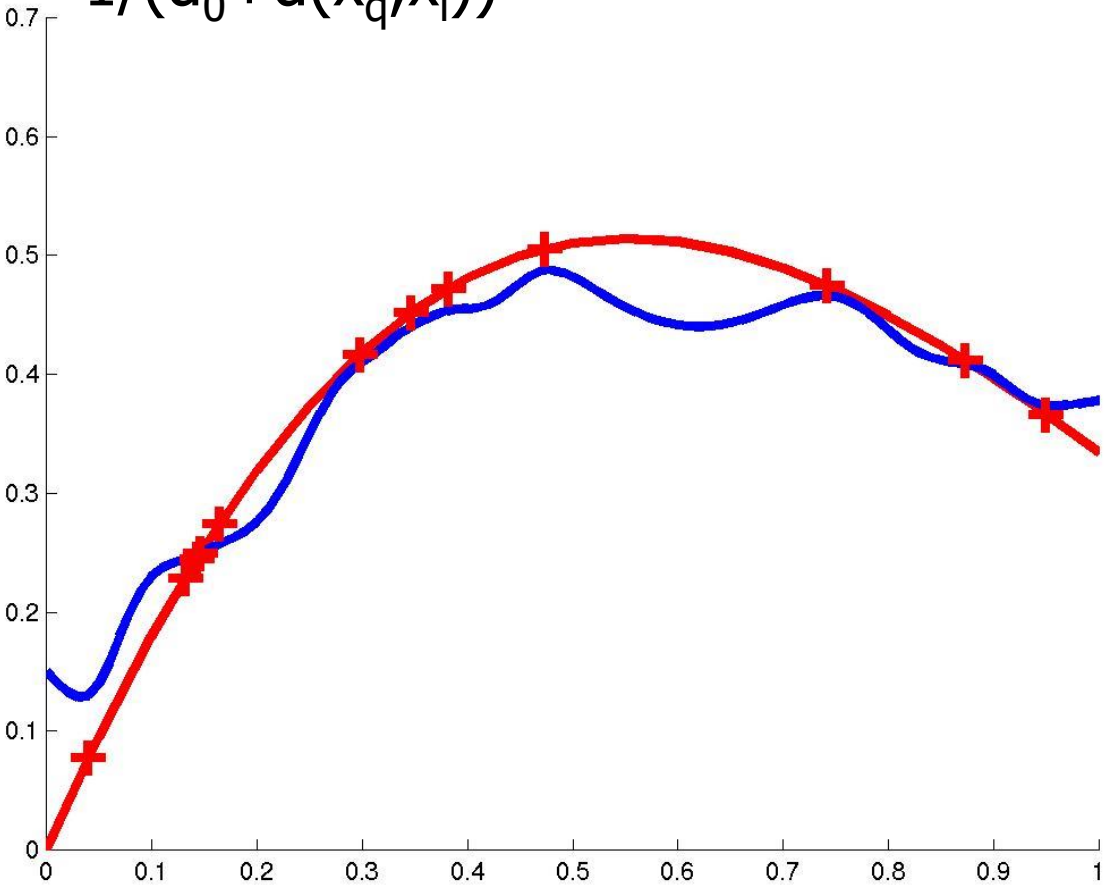
Distance Weighted NN

$K(d(x_q,x_i)) = 1/\ d(x_q,x_i)^2$



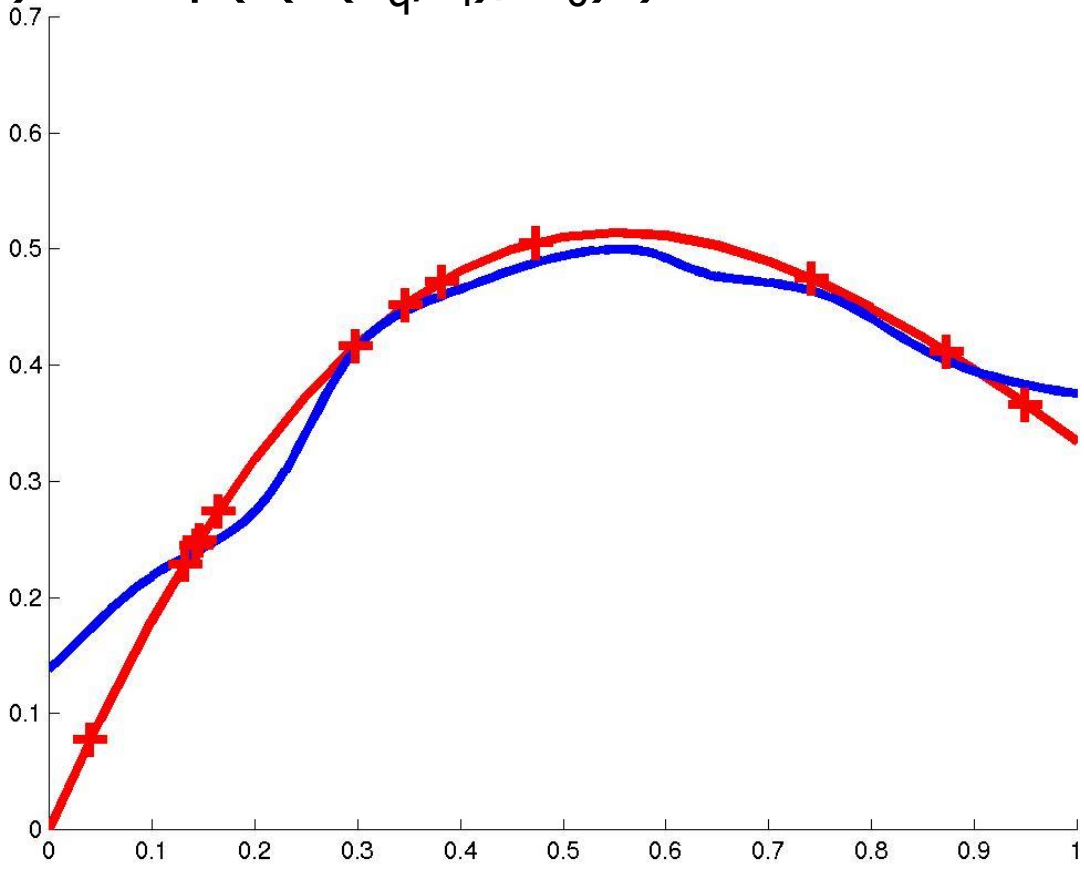
Distance Weighted NN

$$K(d(x_q,x_i)) = 1/(d_0+d(x_q,x_i))^2$$



Distance Weighted NN

$$K(d(x_q,x_i)) = \exp(-(d(x_q,x_i)/\sigma_0)^2)$$



Curse of Dimensionality

Imagine instances described by 20 attributes but only a few are relevant to target function

Curse of dimensionality: nearest neighbor is easily misled when instance space is high-dimensional

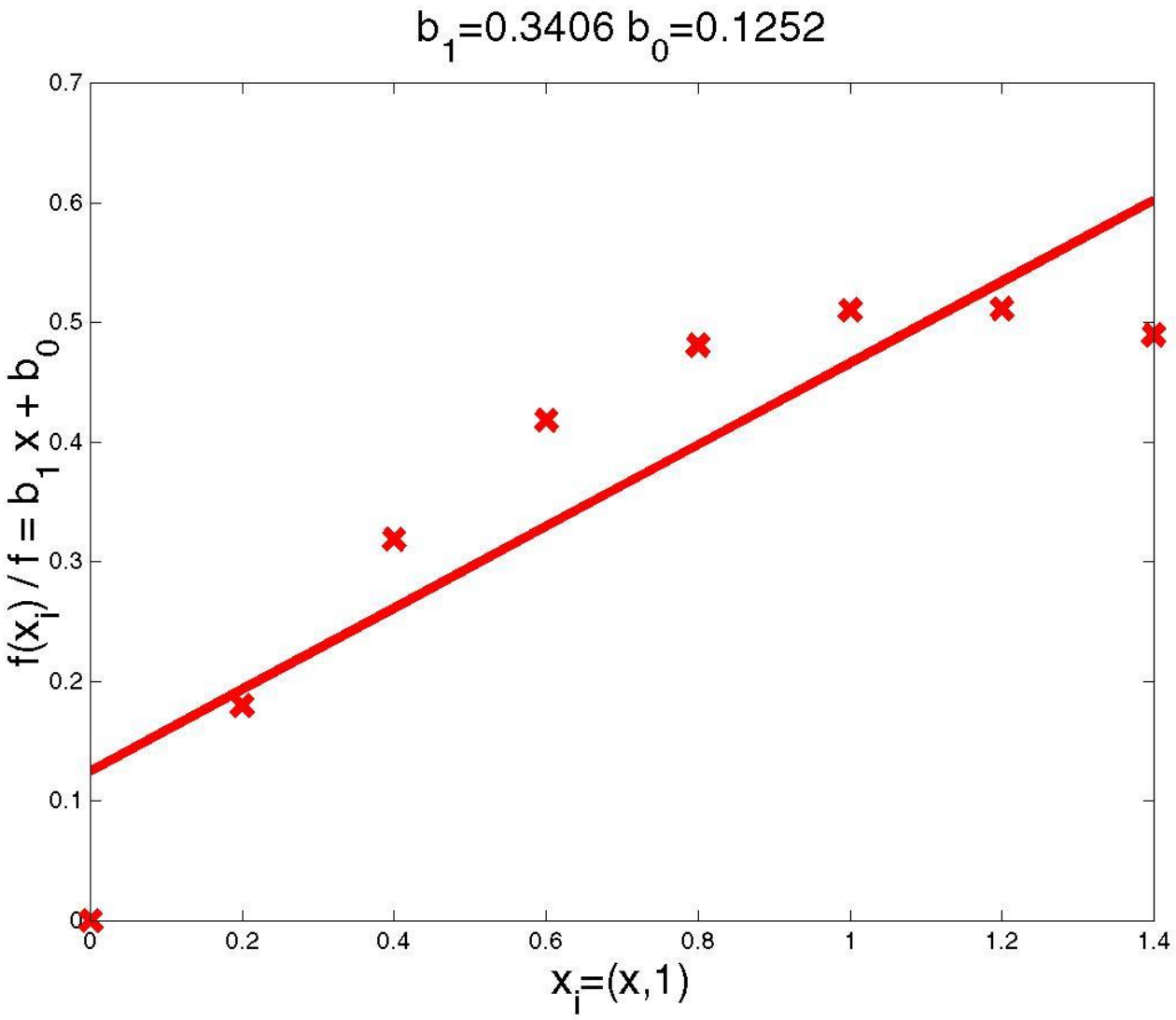
One approach:
Stretch j-th axis by weight z_j, where $z_1,\ldots,z_n$ chosen to minimize prediction error
Use cross-validation to automatically choose weights $z_1,\ldots,z_n$
Note setting z_j to zero eliminates this dimension alltogether (feature subset selection)

Locally Weighted Regression

Locally Weighted Regression

- Locally – Function approximated based on data near query point
- Weighted – Contribution by each training example is weighted by its distance from query point
- Regression- Approximates real-valued target function

Linear Regression Example



Locally weighted regression

- a note on terminology:
  - *Regression* means approximating a real-valued target function
  - *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function
  - *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$
- nearest neighbor approaches can be thought of as approximating the target function at the single query point x_q
- locally weighted regression is a generalization to this approach, because it constructs an explicit approximation of f over a local region surrounding x_q

Locally weighted regression

- target function is approximated using a **linear function**

$$\hat{f}(x) = w_0 + w_1 a_1(x) + ... + w_n a_n(x)$$

- methods like **gradient descent** can be used to calculate the coefficients $w_0, w_1, ..., w_n$ to minimize the error in fitting such linear functions

- ANNs require a global approximation to the target function
- here, just a local approximation is needed

$\Rightarrow$ the error function has to be redefined

Locally weighted regression

- possibilities to redefine the error criterion E
  1. Minimize the squared error over just the k nearest neighbors

$$E_1(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neigbors}} (f(x) - \hat{f}(x))^2$$

  2. Minimize the squared error over the entire set D, while weighting the error of each training example by some decreasing function K of its distance from x_q

$$E_2(x_q) \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

  3. Combine 1 and 2

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest neighbors}} (f(x) - \hat{f}(x))^2 \cdot K(d(x_q, x))$$

Locally weighted regression

- choice of the error criterion
  - E_2 is the most esthetically criterion, because it allows every training example to have impact on the classification of x_q
  - however, computational effort grows with the number of training examples
  - E_3 is a good approximation to E_2 with constant effort

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest neighbors}} K(d(x_q, x))(f(x) - \hat{f}(x))a_j$$

- Remarks on locally weighted linear regression:
  - in most cases, constant, linear or quadratic functions are used
  - costs for fitting more complex functions are prohibitively high
  - simple approximations are good enough over a sufficiently small subregion of X

Design Issues in Local Regression

- Local model order (constant, linear, quadratic)
- Distance function d
  - feature scaling: $d(x,q)=(\Sigma_{j=1}^{d} m_j(x_j-q_j)^2)^{1/2}$
  - irrelevant dimensions $m_j=0$
- kernel function K

See paper by Atkeson [1996] "Locally Weighted Learning"

Radial Basis Function

Radial Basis Function

- closely related to distance-weighted regression and to ANNs
- learned hypotheses have the form

$$\hat{f}(x) = w_0 + \sum_{u=1}^{k} w_u \cdot K_u(d(x_u, x))$$

where

- each x_u is an instance from X and
- $K_u(d(x_u, x))$ decreases as $d(x_u, x)$ increases and
- k is a user-provided constant

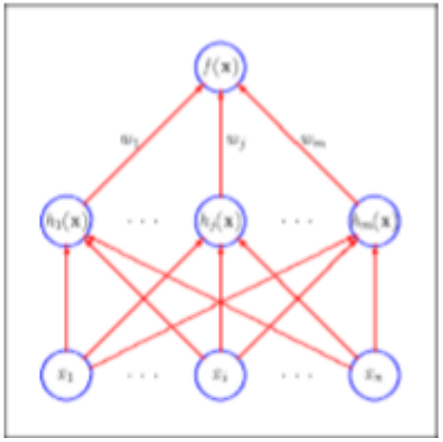
- though $\hat{f}(x)$ is a global approximation to $f(x)$, the contribution of each of the K_u terms is localized to a region nearby the point x_u

Radial Basis Function

- it is common to choose each function $K_u(d(x_u, x))$ to be a Gaussian function centered at x_u with some variance σ^2

$$K_u(d(x_u, x)) = e^{\frac{1}{2\sigma_u^2} d^2(x_u, x)}$$

- the function of $\hat{f}(x)$ can be viewed as describing a two-layer network where the first layer of units computes the various $K_u(d(x_u, x))$ values and the second layer a linear combination of the results



Training Radial Basis Function Networks

How to choose the center x_n for each Kernel function K_n?

- scatter uniformly across instance space
- use distribution of training instances (clustering)

How to train the weights?

- Choose mean x_n and variance σ_n for each K_n
  non-linear optimization or EM
- Hold K_n fixed and use local linear regression to compute the optimal weights w_n

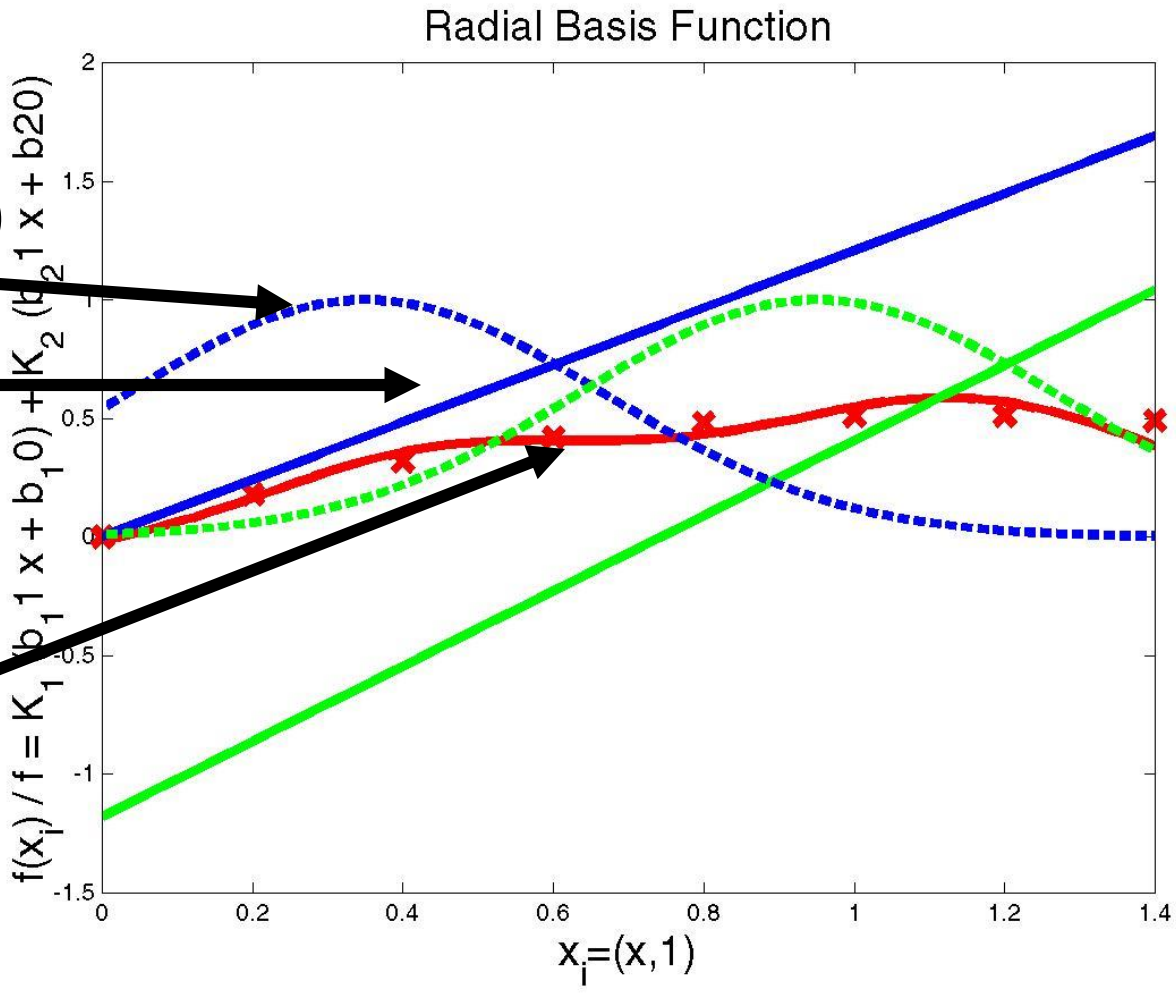
Radial Basis Network Example

$K_1(d(x_1,x))=$

$\exp(-1/2\ d(x_1,x)^2/\sigma^2)$

$w_1\ x+\ w_0$

$\hat{f}(x) = K_1\ (w_1\ x+\ w_0) + K_2\ (w_3\ x\ +\ w_2)$



Lazy and Eager Learning

Lazy: wait for query before generalizing

- k-nearest neighbors, weighted linear regression

Eager: generalize before seeing query

- Radial basis function networks, decision trees, back-propagation
- Eager learner must create global approximation

Lazy learner can create local approximations
If they use the same hypothesis space, lazy can represent more complex functions (H=linear functions)

Confusion Matrix

- **True Positive (TP):** It refers to the number of predictions where the classifier correctly predicts the positive class as positive.
- **True Negative (TN):** It refers to the number of predictions where the classifier correctly predicts the negative class as negative.
- **False Positive (FP):** It refers to the number of predictions where the classifier incorrectly predicts the negative class as positive.
- **False Negative (FN):** It refers to the number of predictions where the classifier incorrectly predicts the positive class as negative.

Predicted class ->	C_1	$\neg C_1$
Actual class⇓		
C_1	**True Positives (TP)**	**False Negatives (FN)**
$\neg C_1$	**False Positives (FP)**	**True Negatives (TN)**

Classifier Evaluation Metrics: Accuracy, Error Rate

innovate achieve lead

Classifier Accuracy, or recognition rate: percentage of test set tuples that are correctly classified

Accuracy = (TP + TN)/All

most effective when the class distribution is relatively balanced

Classification Error/ Misclassification rate

Misclassification rate =1 – accuracy,

= (FP + FN)/All

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

Metrics - Basics

True positive rate (TPR) or **sensitivity** is defined as the fraction of positive examples predicted correctly by the model

$$TPR = TP/(TP + FN)$$

True negative rate (TNR) or **specificity** is defined as the fraction of negative examples predicted correctly by the model

$$TNR = TN/(TN + FP)$$

False positive rate (FPR) is the fraction of negative examples predicted as a positive class

$$FPR = FP/(TN + FP)$$

False negative rate (F N R) is the fraction of positive examples predicted as a negative class

$$FNR = FN/(TP + FN)$$

Precision, Recall & F1

- Precision determines the fraction of records that actually turns out to be positive in the group the classifier has declared as a positive class
  - The higher the precision is, the lower the number of false positive errors committed by the classifier

$$\text{Precision, } p = \frac{TP}{TP + FP}$$

- Recall (same as TPR) measures the fraction of positive examples correctly predicted by the classifier

$$\text{Recall, } r = \frac{TP}{TP + FN}$$

  - Classifiers with large recall have very few positive examples misclassified as the negative class

- Harmonic between precision & recall is known as F_1 measure

$$F_1 = \frac{2}{\frac{1}{r} + \frac{1}{p}} \qquad F_1 = \frac{2rp}{r + p} = \frac{2 \times TP}{2 \times TP + FP + FN}$$

High value of F_1 measure ensures that both precision and recall are high

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

Holdout method

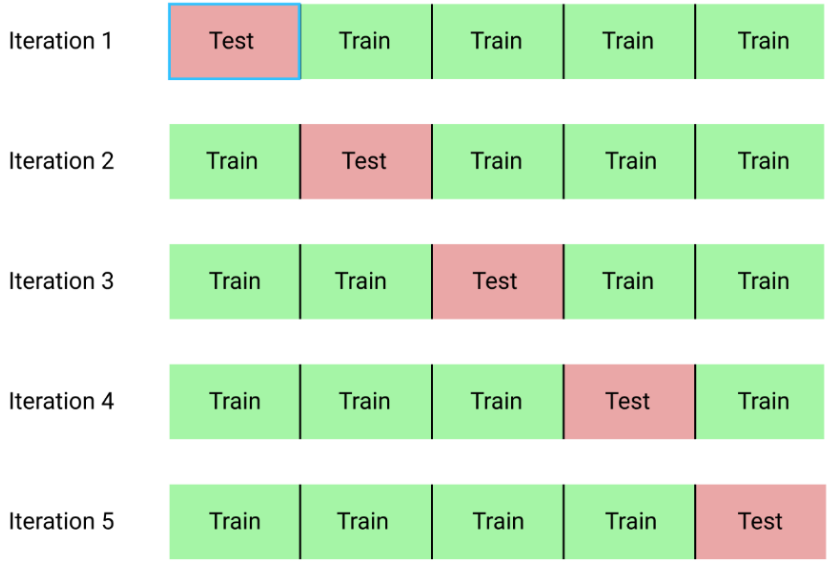
- Given data is randomly partitioned into two independent sets
  - Training set (e.g., 2/3) for model construction
  - Test set (e.g., 1/3) for accuracy estimation
- Random sampling: a variation of holdout
  - Repeat holdout k times, accuracy = avg. of the accuracies obtained

Cross-validation (*k*-fold, where k = 10 is most popular)

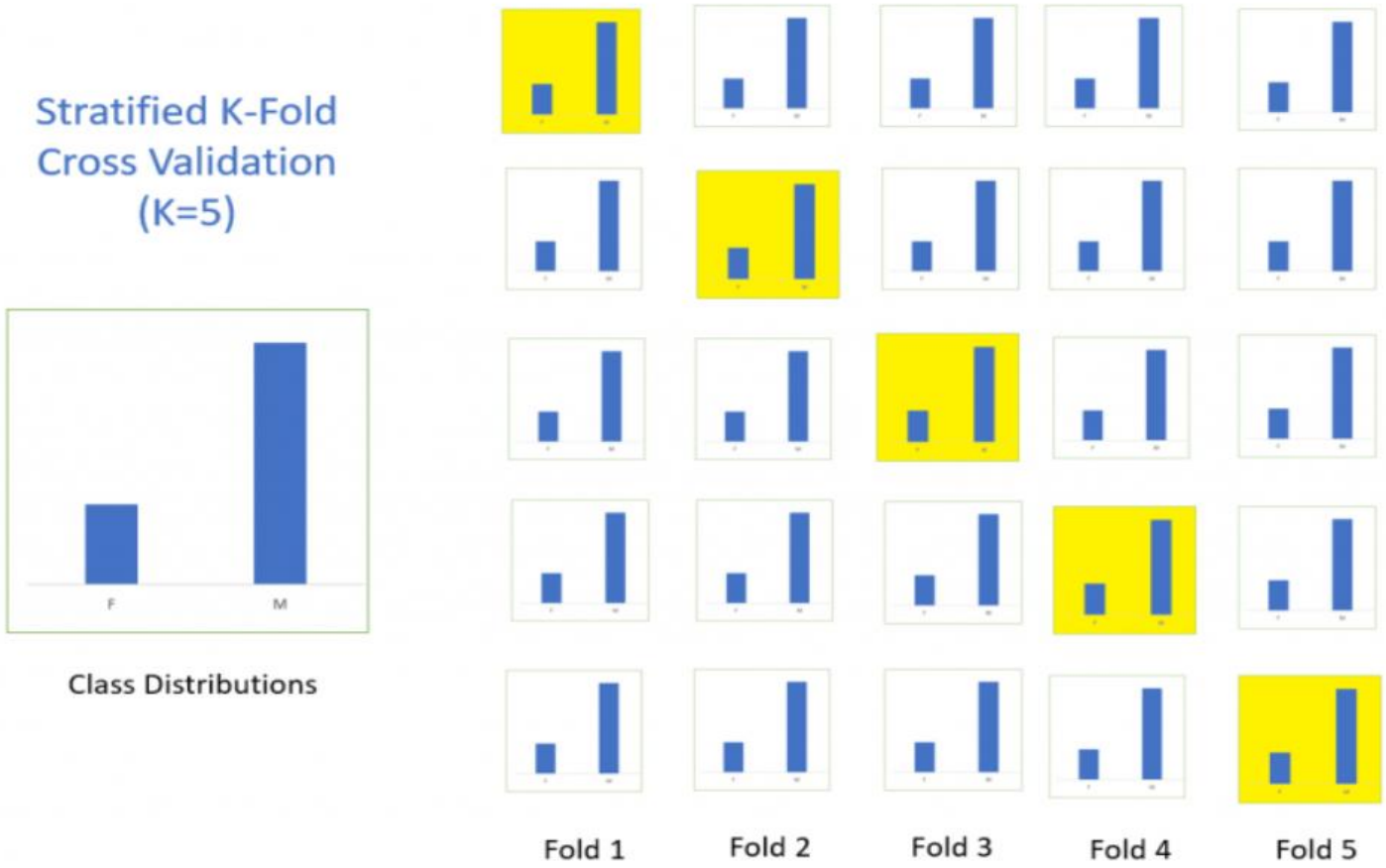
- Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
- At *i*-th iteration, use D_i as test set and others as training set
- The Accuracy of the model is the average of the accuracy of each fold.

Cross Validation

k-Fold Cross Validation:



Stratified Cross Validation



Literature & Software

- T. Mitchell, “Machine Learning”, chapter 8, “Instance-Based Learning”
- “Locally Weighted Learning”, Christopher Atkeson, Andrew Moore, Stefan Schaal
- R. Duda et al., “Pattern recognition”, chapter 4 “Non-Parametric Techniques”

Netlab toolbox

- k-nearest neighbor classification
- Radial basis function networks

Thank You