BITS Pilani
Pilani Campus

Machine Learning DSECL ZG565

- Dr. Monali Mavani

These slides are prepared by the instructor, with grateful acknowledgement of Tom Mitchell, Andrew Ng< Prof. Chetana, Prof. Vimal 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

Tom Mitchell – Ch 8

Model-based learning techniques

Use the input data

$$\begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m,0} & x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \text{ and } \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}$$

$\Downarrow$

To learn a set of parameters

$$\begin{bmatrix} \theta_0 & \theta_1 & \cdots & \theta_n \end{bmatrix}$$

$\Downarrow$

Which yield a **generalized** function

$$f(x) = \theta_0 x_0 + \theta_1 x_1 + \theta_2 x_2 + \dots + \theta_n x_n$$

$\Downarrow$

Capable of predicting values or classes on new input data

$$f(x_i) = 39$$
$$f(x_j) = 1$$

Instance-based learning techniques

Store the input data

$$\begin{bmatrix} x_{1,0} & x_{1,1} & \cdots & x_{1,n} \\ x_{2,0} & x_{2,1} & \cdots & x_{2,n} \\ \vdots & \vdots & \vdots & \vdots \\ x_{m,0} & x_{m,1} & \cdots & x_{m,n} \end{bmatrix} \text{ and } \begin{bmatrix} y_1 \\ y_2 \\ \dots \\ y_m \end{bmatrix}$$

$\Downarrow$

When asked to predict a new value (a query)

$$y_i = ?$$

$\Downarrow$

Search for similar data points previously stored

$$\begin{bmatrix} x_{4,1} & x_{4,2} & \cdots & x_{4,n} \\ x_{9,1} & x_{9,1} & \cdots & x_{9,n} \\ x_{15,1} & x_{15,1} & \cdots & x_{15,n} \end{bmatrix} \text{ and } \begin{bmatrix} y_4 \\ y_9 \\ y_{15} \end{bmatrix}$$

$\Downarrow$

And use them to generate your prediction

$$y_i = \frac{y_4 + y_9 + y_{15}}{3}$$

Instance based learning

- Can approximate real-valued or discrete-valued target functions.
- ‘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**
- A key advantage of this kind of delayed, or lazy, learning is that instead of estimating the target function once for the entire instance space, these methods can estimate it ***locally and differently*** for each new instance to be classified. E.g Nearest Neighbour and locally weighted regression
- Real world applications found in recommendation systems (amazon).

Eager: generalize before seeing query

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

Disadvantages-Instance based approaches

- High cost of classifying new instances
  - nearly all computation takes place at classification time rather than when the training examples are first encountered.
  - Need techniques for efficiently indexing training examples to reduce computation required at query time.

Nearest neighbor approaches

- Key idea: just store all training examples $<x_i, f(x_i)>$
- Given query instance x_q, first locate nearest training example x_n, then estimate $f^*(x_q)=f(x_n)$

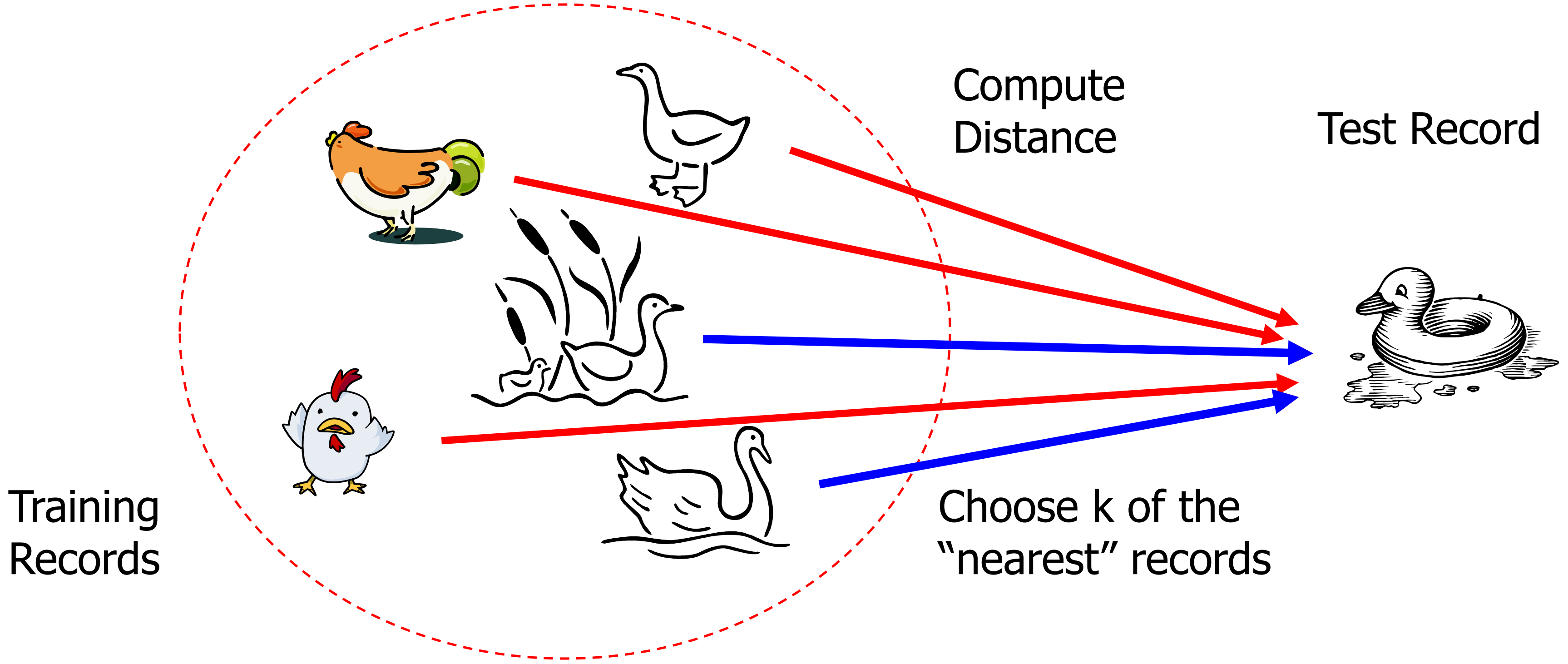
K-nearest neighbor:

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

Nearest Neighbor Classifiers

Basic idea:

– If it walks like a duck, quacks like a duck, then it's probably a duck



k-Nearest Neighbor

- 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:

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

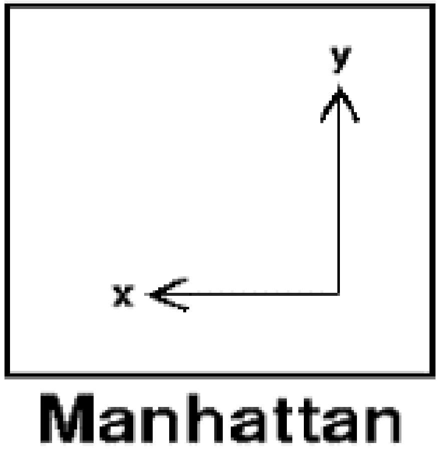
Distances

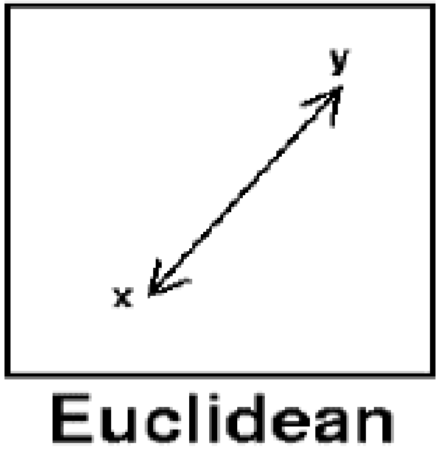
- Manhattan Distance

|X1-X2| + |Y1-Y2|

- Euclidean Distance

$$\sqrt{(x1 - x2)^2} + \sqrt{(y1 - y2)^2}$$





- works if the points are arranged in the form of a grid
- if the input variables are not similar in type (such as age, gender, height, etc.)

- Euclidean is commonly used on dense, continuous variables.
- There every dimension matters, and a 20 dimensional space can be challenging
- if the input variables are of similar in type (such as width, height, depth etc.)

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, **Chebyshev 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, Pilani Campus

Distance Measure: Scale Effects

- Different features may have different measurement scales
  - E.g., patient weight in kg (range [50,200]) vs. blood protein values in ng/dL (range [-3,3])
- Consequences
  - Patient weight will have a much greater influence on the distance between samples
  - May bias the performance of the classifier

Feature scaling - - Standardization

- 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

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

- The Standard Scaler assumes data is normally distributed within each feature and scales them such that the distribution centered around 0, with a standard deviation of 1.

BITS Pilani, Pilani Campus

Feature scaling - Normalization

Min-Max scaler/Normalization

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)

$$x_1 = \frac{size-1000}{2000}$$

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

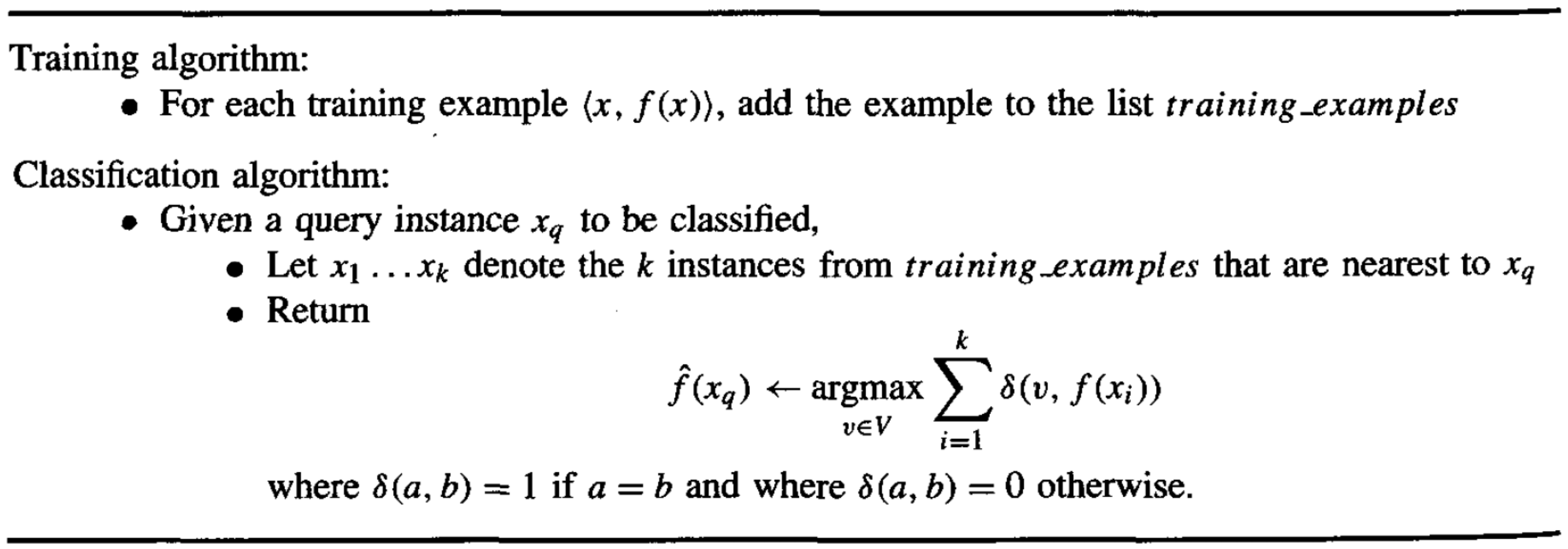
$$\hat{f}(x_q) \leftarrow \underset{v \in V}{argmax} \sum_{i=1}^{k} \delta(v, f(x_i))$$

  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

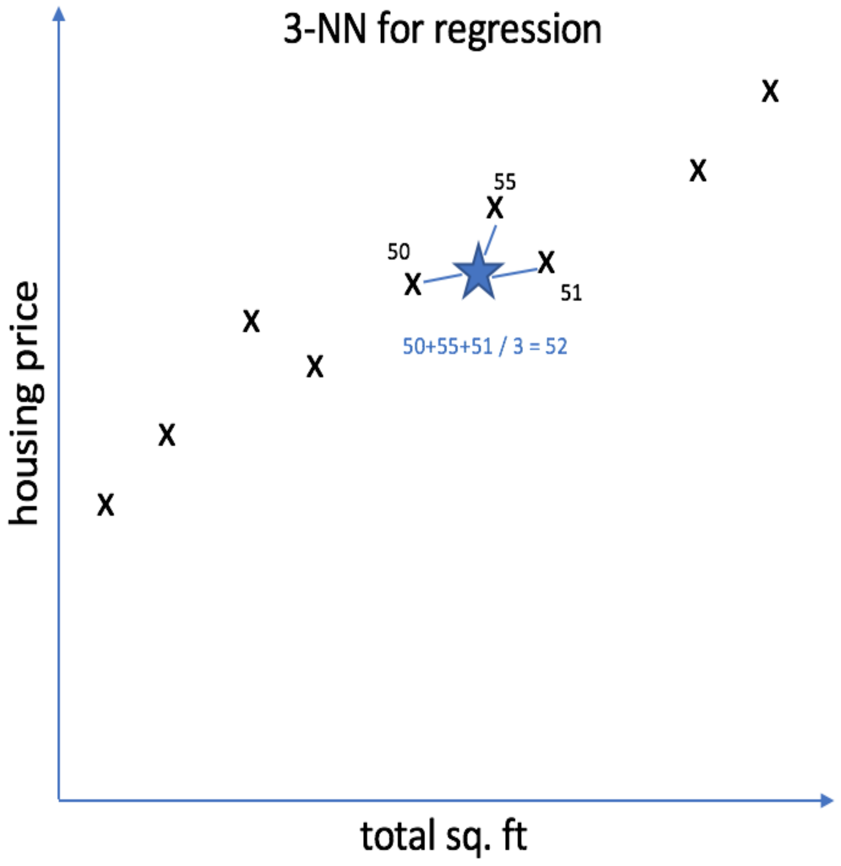


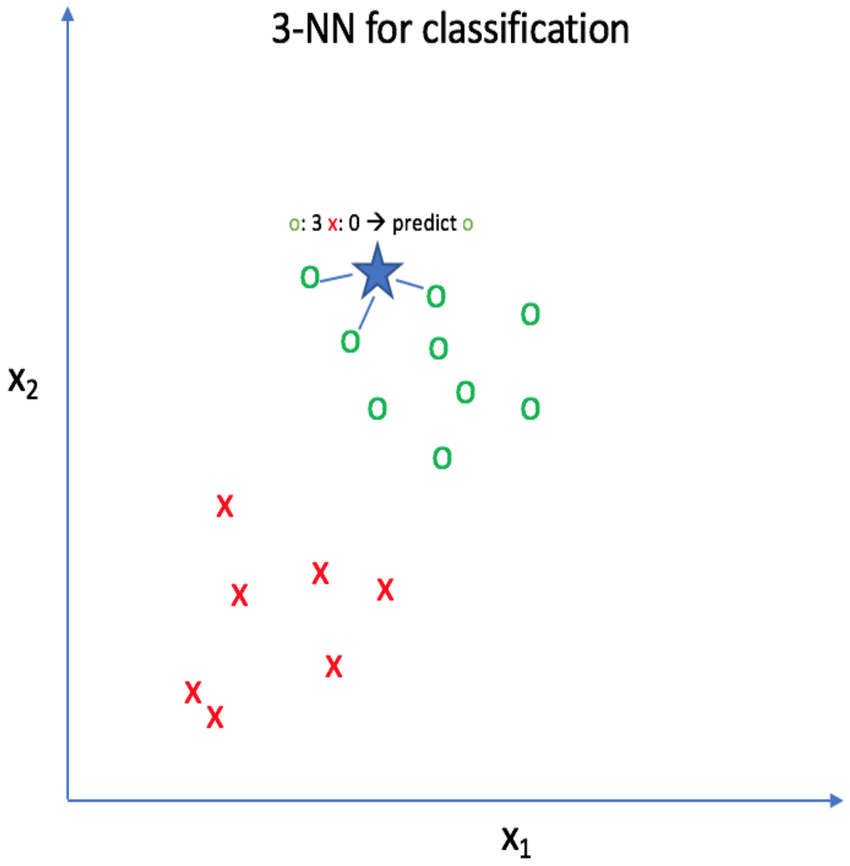
```
Training algorithm:
    • For each training example $\langle x, f(x) \rangle$, add the example to the list $training\_examples$

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
              $\hat{f}(x_q) \leftarrow \operatorname*{argmax}_{v \in V} \sum_{i=1}^{k} \delta(v, f(x_i))$
          where $\delta(a, b) = 1$ if $a = b$ and where $\delta(a, b) = 0$ otherwise.
```

* It can be used for Regression as well.

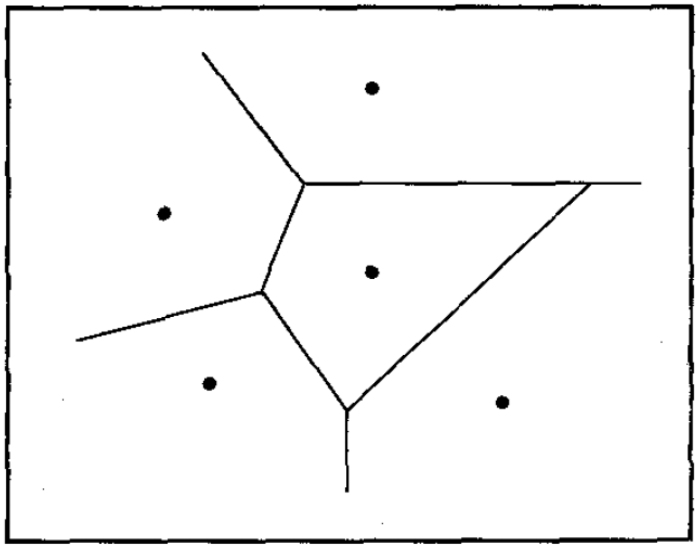
KNN for regression and Classification

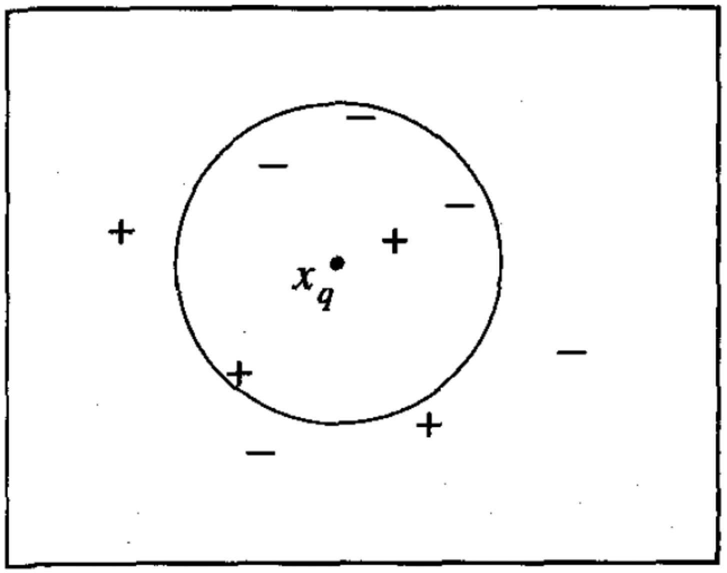




https://www.jeremyjordan.me/k-nearest-neighbors/

k-NN examples


K=1

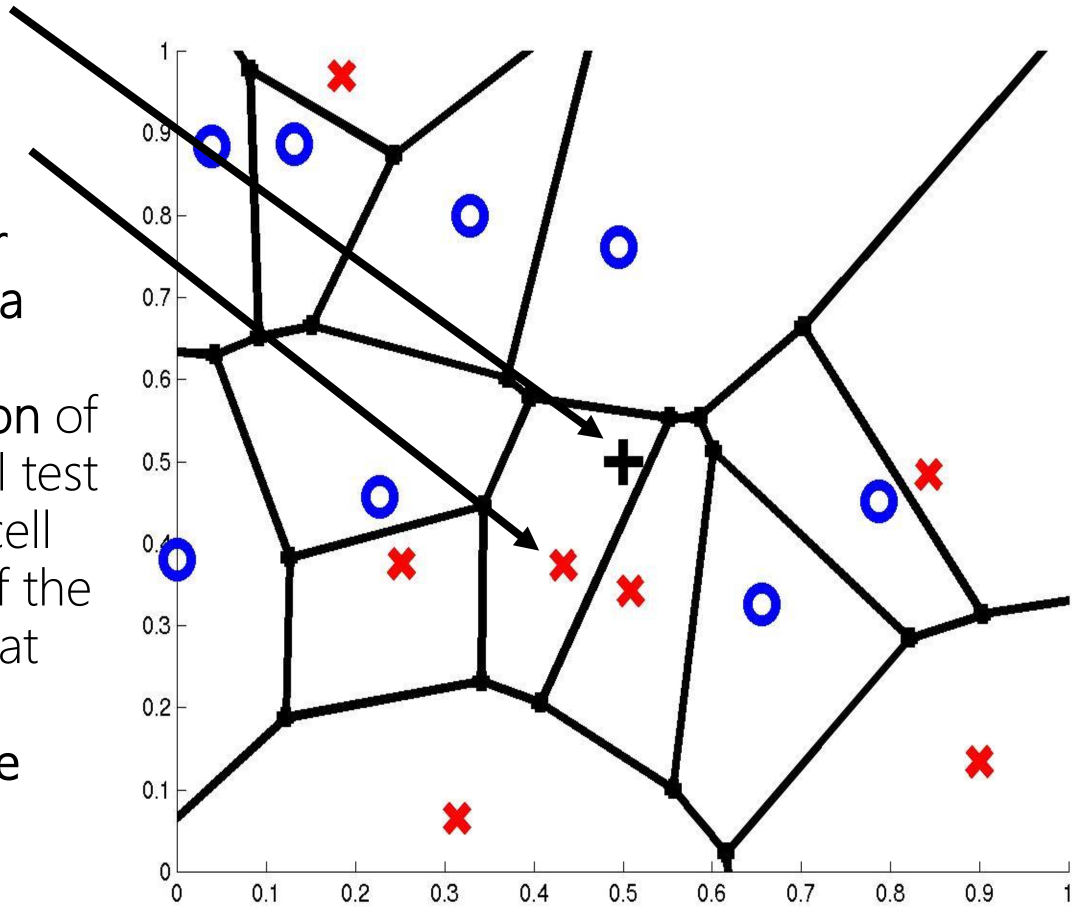


K=5

Voronoi Diagram

query point q_f

nearest neighbor q_i



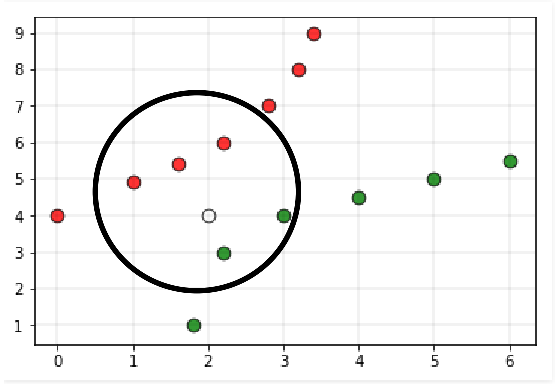
- Nearest neighbour approach induces a Voronoi tessellation/partition of the input space (all test points falling in a cell will get the label of the training input in that cell)
- For any sample, the nearest sample is determined by the closest Voronoi cell edge

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

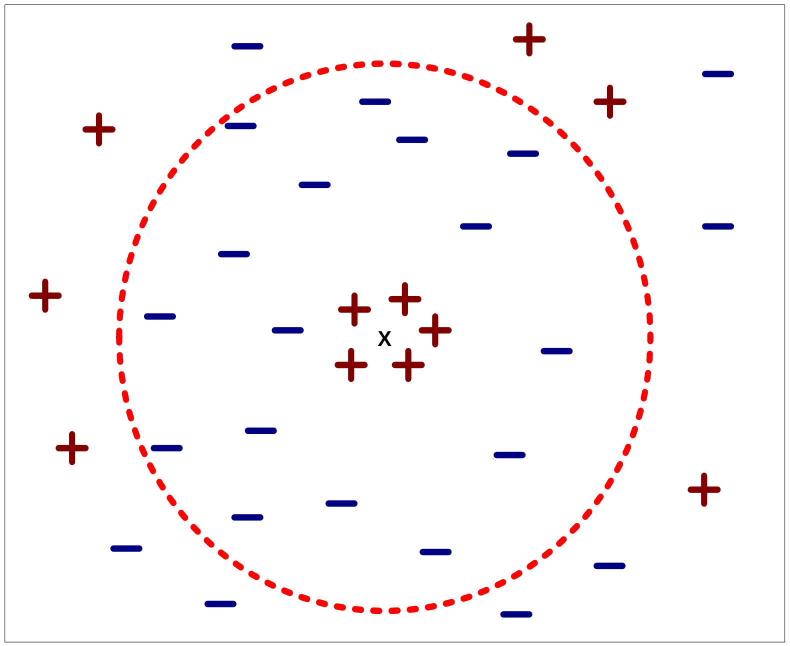
Assigning the class labels can be tricky. For example, in the below case, for (k=5) the point is closer to 'green' classification, but gets classified as 'red' due to higher red votes/majority voting to 'red'



Value of K

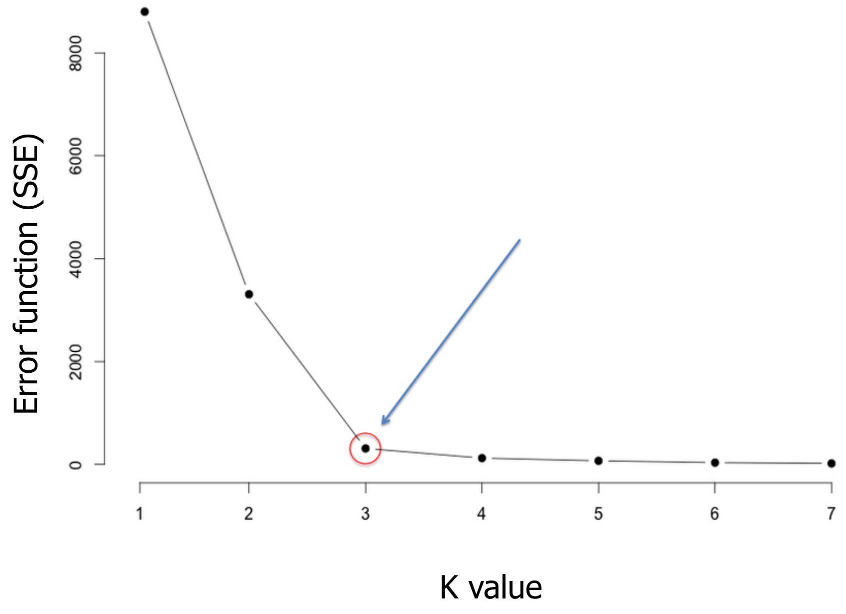
- Choosing the value of k:
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes

Rule of thumb:
K = sqrt(N)
N: number of training points



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',



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
  - $w_i = K(d(x_q, x_i))$
  - $K(d(x_q, x_i)) = 1/\, d(x_q, x_i)^2$
  - $K(d(x_q, x_i)) = 1/(d_0 + d(x_q, x_i))^2$
  - $K(d(x_q, x_i)) = \exp(-(d(x_q, x_i)/\sigma_0)^2)$
  - $d(x_q, x_i)$ is the distance between x_q and x_i
- Variation: Instead of only k-nearest neighbors use all training examples (global method - Shepard’s method)

Distance Weighted Average

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

Curse of Dimensionality

- Imagine instances described by 20 attributes but only a few(2) are relevant to target function
- *Curse of dimensionality*: nearest neighbor is easily misled when instance space is high-dimensional
- First approach:
  - completely eliminate the least relevant attributes from the instance space. This is equivalent to setting some of the z_i scaling factors to zero
  - Use of cross-validation methods for selecting relevant subsets of the attributes

Second approach

- weight each attribute differently when calculating the distance between two instances i.e
  - stretching the axes in the Euclidean space, shortening the axes that correspond to less relevant attributes, and lengthening the axes that correspond to more relevant attributes.
- The amount by which each axis should be stretched can be determined automatically using a cross-validation approach
- To stretch (multiply) the jth axis by some factor z_j, where the values $z_1 \ldots z_n$, are chosen to minimize the true classification error of the learning algorithm

When to Consider Nearest Neighbors

- Suitable for Low dimensional datasets
- Lots of training data (distance-weighted KNN)
  - Training is very fast
- Learn complex target functions
  - Do not lose information
- Noisy training data (distance-weighted KNN)
  - by taking the weighted average of the k neighbors nearest to the query point, it can smooth out the impact of isolated noisy training examples.

Nearest-Neighbor Classifiers: Issues

- The value of *k*, the number of nearest neighbors to retrieve
- Choice of Distance Metric to compute distance between records
- They typically consider ***all*** attributes of the instances when attempting to retrieve similar training examples from memory.
  - If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

Nearest Neighbours issues

- Expensive, Slow at query time
  - To determine the nearest neighbour of a query point *q*, must compute the distance to all *N* training examples
    - + Pre-sort training examples into fast data structures (kd-trees)
    - + Remove redundant data (condensing)
- Storage Requirements
  - Must store all training data **P**
    - + Remove redundant data (condensing)
    - - Pre-sorting often increases the storage requirements
- High Dimensional Data
  - "Curse of Dimensionality"
    - Required amount of training data increases exponentially with dimension
    - Computational cost also increases dramatically

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
- ***Residual*** is the error in approximating the target function.

$$\hat{f}(x) - f(x)$$

- ***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

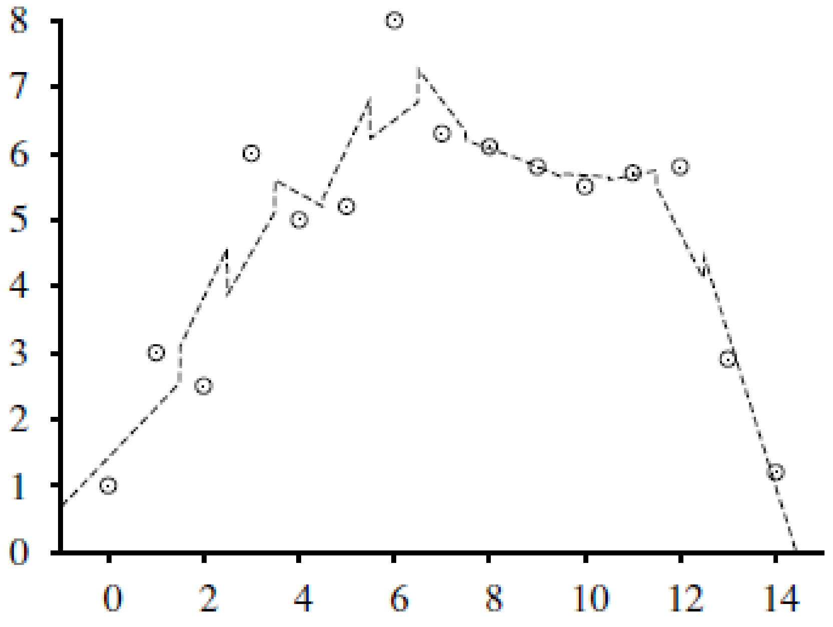
$$w_i = K(d(x_i, x_q))$$

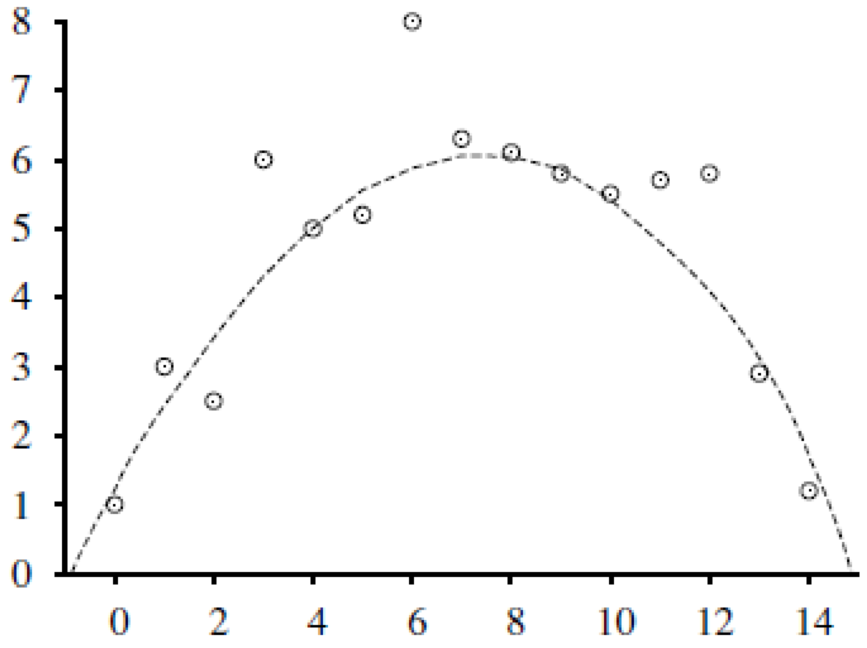
Locally Weighted Regression

- The nearest-neighbor approaches described in the previous section can be thought of as approximating the target function f ***(x)*** at the single query point $\boldsymbol{x} = \boldsymbol{x_q}$.
- Locally weighted regression is a generalization of this approach. It constructs an explicit approximation to f over a local region surrounding $\boldsymbol{x_q}$.

- Uses nearby or distance-weighted training examples to form this local approximation to f.
- Approximate the target function in the neighborhood surrounding x, using a linear function, a quadratic function, a multilayer neural network, or some other functional form.
  - More complex functional forms are not often found the cost of fitting more complex functions for each query instance is prohibitively high,
  - these simple approximations model the target function quite well over a sufficiently small subregion of the instance space

Example





3-nearest-neighbors linear regression

locally weighted regression

Locally weighted linear 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 linear 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 linear regression regression

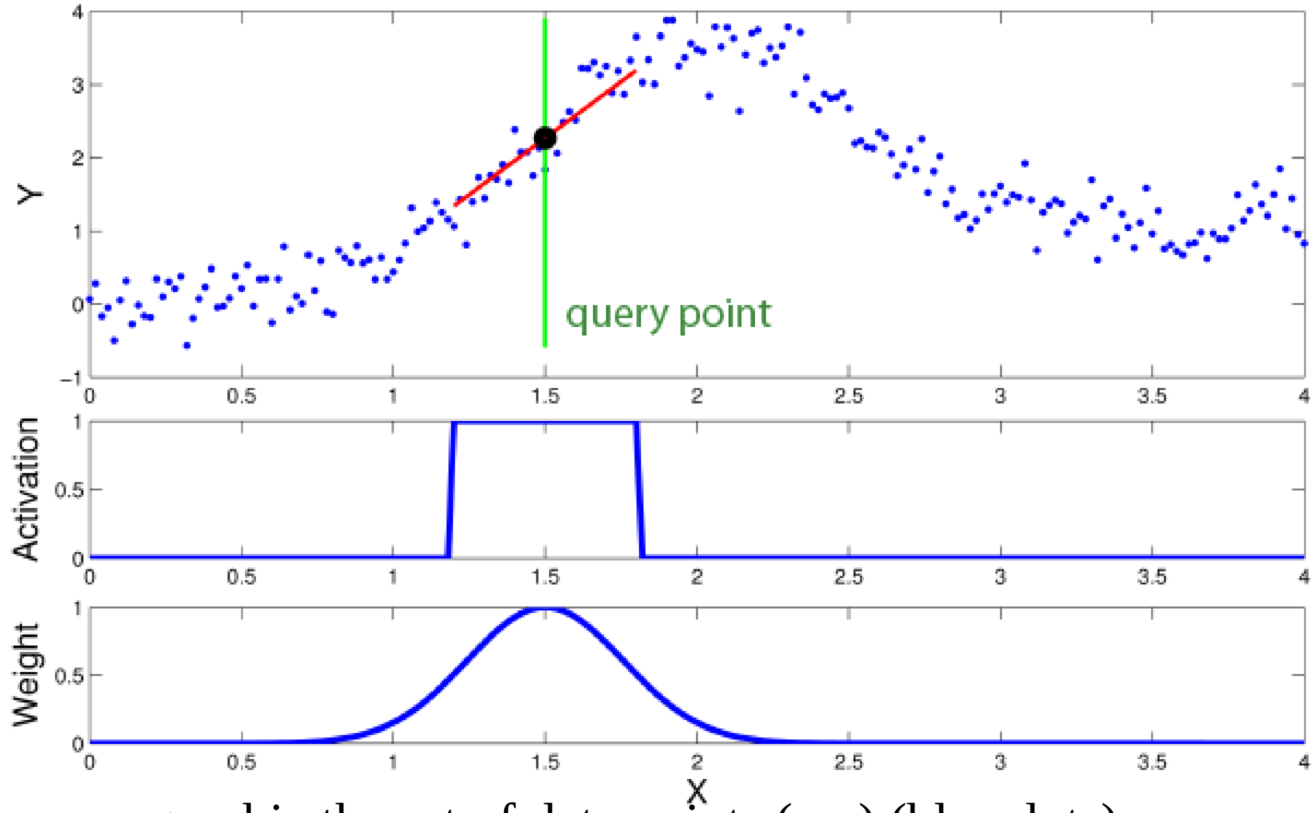
- For a given query point $\mathbf{x}_q$ we solve the following weighted regression problem using gradient descent:

$$\mathbf{w}^* = \underset{\mathbf{w}}{\operatorname{argmin}} \sum_j \mathcal{K}(Distance(\mathbf{x}_q, \mathbf{x}_j))\,(y_j - \mathbf{w}\cdot\mathbf{x}_j)^2$$

$$h(\mathbf{x}_q) = \mathbf{w}^* \cdot \mathbf{x}_q.$$

- Note that we need to solve a new regression problem for *every* query point—that's what it means to be *local*.
- In ordinary linear regression, we solved the regression problem once, globally, and then used the same h_w for any query point.

Example



- The upper graphic the set of data points (x,y) (blue dots), query point (green line), local linear model (red line) and prediction (black dot).
- The graphic in the middle shows the activation area of the model.
- The corresponding weighting kernel (receptive field) is shown in the bottom graphic.

Thank You

7 August 2022