

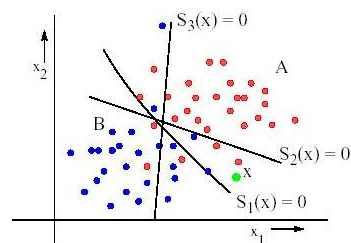
BAYESIAN LEARNING, cont.

Danica Kragic



November 18, 2005

Coping with "bad features"



Bagging and Boosting

- Bagging and Boosting aggregate multiple hypotheses generated by the same learning algorithm invoked over different distributions of training data [Breiman 1996, Freund and Schapire 1996]
- Bagging and Boosting generate a classifier with a smaller error on the training data as it combines multiple hypotheses which individually have a large error

Bagging and Boosting

- Bagging replicates training sets by sampling with replacement from the training instances.
- Boosting uses all instances but weights them and therefore produces different classifiers.
- Classifiers are then combined by voting to create a composite classifier.
- Bagging: classifiers have same votes;
- Boosting: vote dependant on the classifiers' accuracy

Boosting

- Boosting maintains a weight w_i for each instance $\langle x_i, \dots, c_i \rangle$ in the training set
- The higher the weight w_i , the more the instance x_i influences the next hypotheses learned
- At each trial, the weights are adjusted to reflect the performance of the previously learned hypothesis, with the result that the weight of correctly classified instances is decreased and the weight of incorrectly classified instances is increased
- The final hypothesis c_{BO} aggregates the individual hypotheses C^j by weighted voting

$$c_{BO}(x) = \operatorname{argmax}_{c_j \in C} \sum_{i=1}^T \alpha_i \delta(c_j, c_i(x))$$

- Each hypothesis vote is a function of its accuracy

Boosting

- The error of the hypothesis C^j is measured with respect to the weights

$$\epsilon_j = \sum_{\forall i \text{ such that } C^j(x_i) \neq c_i} \frac{w_i}{\sum_i w_i}$$

$$\alpha_j = \frac{1}{2} \ln((1 - \epsilon_j) / \epsilon_j)$$

- Update the weights w_i^j of *correctly* and *incorrectly* classified instances by

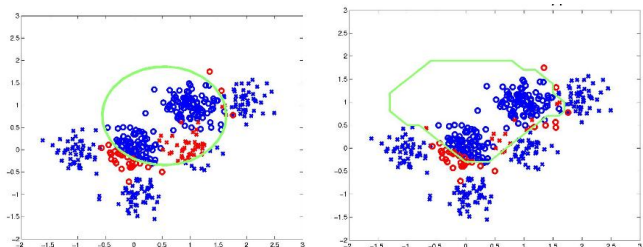
$$w_i^{j+1} = w_i^j e^{-\alpha_j} \text{ if } C^j(x_i) = c_i$$

$$w_i^{j+1} = w_i^j e^{\alpha_j} \text{ if } C^j(x_i) \neq c_i$$

- Afterwards normalize the w_i^{j+1} such that they form a proper distribution $\sum_i w_i^{j+1} = 1$

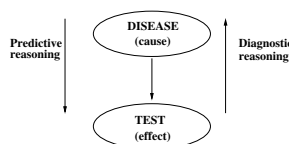
Bayes MAP Hypothesis

- Bayes MAP hypothesis for two classes x and o
- red: incorrect classified instances



Remember this?

Priors $P(h)$: $P(\text{disease}) = 0.008$ $P(\neg \text{disease}) = 0.992$
 Likelihoods $P(D|h)$: $P(+ | \text{disease}) = 0.98$ $P(- | \text{disease}) = 0.02$
 $P(+ | \neg \text{disease}) = 0.03$ $P(- | \neg \text{disease}) = 0.97$



Joint Probability table

| | d | $\neg d$ |
|---|------|----------|
| + | 0.98 | 0.03 |
| - | 0.02 | 0.97 |

Joint probability distribution $P(\text{test}, \text{disease})$ (number of parameters ?)

Joint probability distribution

- assigning probabilities to all possible combinations of values of random variables for a domain
- n -variables and k -values table size is k^n
- given JPD we can compute *any* probabilistic statement about the domain
- Bayesian Belief Networks: space efficient data structure for encoding all of the information in the full joint probability distribution for a set of random variables
- Intuitively, to construct a BN draw arcs from cause to immediate effects

Bayesian Belief Networks (Bayes Nets)

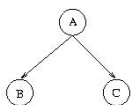
- Fact: many real-world problems include conditionally independent variables
- Bayesian Belief networks describe conditional independence among *subsets* of variables
- A BN is an augmented directed acyclic graph, represented by a pair V, E where
 - V is a set of vertices
 - E is a set of directed edges joining vertices. **No loops of any kind are allowed.**
- Each vertex contains:
 - The name of the random variable
 - A probability distribution table indicating how the probability of this variable's values depends on all possible combinations of its parent's values.

Bayesian Belief Nets

- The occurrence of A provides support for B: predictive support
- Backward influence: diagnostic or evidential support for A due to the occurrence of B
- Each node A in the net is conditionally independent of any subset of nodes that are not descendants of A given the parents of A
- Below: $P(C|A, B) = P(C|A)$ and $P(B|A, C) = P(B|A)$
- In general, the net is constructed so that, given its parents, a node is conditionally independent of the rest of the variables, or:

$$P(X_1 = x_1, \dots, X_N = x_n) = \prod_i P(x_i | \text{Parents}(X_i))$$

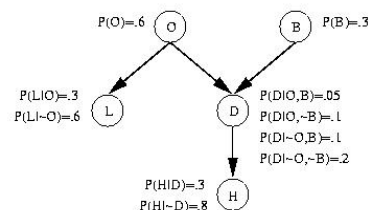
- Hence - we do not need the full joint distribution, only conditionals relative to the parent variables



Bayesian Belief Network - Example

Consider the problem domain in which when I go home I want to know if someone in my family is home before I go in. I know the following information: (1) When nobody is home, the outside light is sometimes on. (It is sometimes on when we expect guests, too.) (2) When nobody is home, the dog is often left outside. (3) If the dog has tummy-troubles, it is also often left outside. (4) If the dog is outside, I will probably hear it barking (though it might not bark, or I might hear a different dog barking and think it's my dog). Given this information, we define the following five Boolean random variables:

- O: Everyone is Out of the house
- L: The Light is on
- D: The Dog is outside
- B: The dog has tummy troubles
- H: I can Hear the dog barking



Inference in Bayesian Networks

- Computing joint probabilities from a Bayesian Net
 - Computing conditional probabilities
1. Causal/Predictive (Top-Down) Inference
 2. Diagnostic/Evidential (Bottom-Up) Inference

Computing joint probabilities from a Bayesian Net

Task: compute $P(B, \neg O, D, \neg L, H)$

$$\begin{aligned}
 P(B, \neg O, D, \neg L, H) &= P(H, \neg L, D, \neg O, B) \\
 &= P(H|\neg L, D, \neg O, B)P(\neg L, D, \neg O, B) && \text{by Product Rule} \\
 &= P(H|D)P(\neg L, D, \neg O, B) && \text{by Conditional Indep. of H and L, O, and B given D} \\
 &= P(H|D)P(\neg L|D, \neg O, B)P(D, \neg O, B) && \text{by Product Rule} \\
 &= P(H|D)P(\neg L|\neg O)P(D, \neg O, B) && \text{by Conditional Indep. of L and D, and L and B, given O} \\
 &= P(H|D)P(\neg L|\neg O)P(D|\neg O, B)P(\neg O, B) && \text{by Product Rule} \\
 &= P(H|D)P(\neg L|\neg O)P(D|\neg O, B)P(\neg O|B)P(B) && \text{by Product Rule} \\
 &= P(H|D)P(\neg L|\neg O)P(D|\neg O, B)P(\neg O)P(B) && \text{by Independence of O and B} \\
 &= (.3)(1 - .6)(.1)(1 - .6)(.3) \\
 &= 0.00144
 \end{aligned}$$

Computing Conditional Probabilities from a Bayesian Net

Causal (Top-Down) Inference

- difficult in general, but easy when the query involves nodes that are directly connected to each other
- Example: we consider problems of the form $P(Q|E)$ - from evidence E to query Q
- This case is called causal inference because we are reasoning in the same direction as the causal arc.

Consider computing $P(D|B)$: What is the probability that my dog is outside when it has tummy troubles?

1. Apply the Product Rule and Marginalization

$$\begin{aligned}
 P(D|B) &= P(D, B)/P(B) && \text{by the Product Rule} \\
 &= (P(D, B, O) + P(D, B, \neg O))/P(B) && \text{by marginalizing } P(D, B) \\
 &= P(D, B, O)/P(B) + P(D, B, \neg O)/P(B) \\
 &= P(D, O|B) + P(D, \neg O|B)
 \end{aligned}$$

Computing Conditional Probabilities from a Bayesian Net, Cont.

2. Apply the conditionalized version of the chain rule, i.e., $P(A, B|C) = P(A|B, C)P(B|C)$, to obtain $P(D|B) = P(D|O, B)P(O|B) + P(D|\neg O, B)P(\neg O|B)$
3. Since O and B are independent by the network, we know $P(O|B) = P(O)$ and $P(\neg O|B) = P(\neg O)$. This means we now have

$$\begin{aligned}
 P(D|B) &= P(D|O, B)P(O) + P(D|\neg O, B)P(\neg O) \\
 &= (.05)(.6) + (.1)(1 - .6) \\
 &= 0.07
 \end{aligned}$$

Computing Conditional Probabilities from a Bayesian Net

Diagnostic (Bottom-Up) Inference

- simplest case of diagnostic inference: compute $P(Q|E)$ assuming that there is an arc from query Q to evidence E
- using a symptom to infer a cause
- consider computing $P(\neg B|\neg D)$: If the dog is not outside, what is the probability that the dog has tummy troubles?

1. First, Bayes Rule: $P(B|\neg D) = P(\neg D|\neg B)P(\neg B)/P(\neg D)$

- We look up in the BN the value of $P(\neg B) = 1 - 0.3 = 0.7$. Next, compute $P(\neg D|\neg B)$ using the causal inference as in the Top-Down method.

$$\begin{aligned}
 P(\neg L|\neg B) &= P(\neg D, O|\neg B) + P(\neg D, \neg O|\neg B) \\
 &= P(\neg D|O, \neg B)P(O|\neg B) + P(\neg D|\neg O, \neg B)P(\neg O|\neg B) \\
 &= P(\neg D|O, \neg B)P(O) + P(\neg D|\neg O, \neg B)P(\neg O) \\
 &= (.9)(.6) + (.8)(.4) = 0.86
 \end{aligned}$$

$$\text{So, } P(\neg B|\neg D) = (.86)(.7)/P(\neg D) = .602/P(\neg D)$$

- Continue

Computing Conditional Probabilities from a Bayesian Net, Cont.

3. To avoid computing the prior probability of symptom D, $P(\neg D)$, we can use normalization, which requires computing $P(B|\neg D)$.

Then by Bayes: $P(B|\neg D) = P(\neg D|B)P(B)/P(\neg D)$, and

$P(B) = .3$ from the BN

- Now compute $P(\neg D|B)$ as follows:

$$\begin{aligned}
 P(\neg D|B) &= P(\neg D, O|B) + P(\neg D, \neg O|B) \\
 &= P(\neg D|O, B)P(O|B) + P(\neg D|\neg O, B)P(\neg O|B) \\
 &= P(\neg D|O, B)P(O) + P(\neg D|\neg O, B)P(\neg O) \\
 &= (.95)(.6) + (.9)(.4) = 0.93
 \end{aligned}$$

- So: $P(B|\neg D) = (.93)(.3)/P(\neg D) = .279/P(\neg D)$

- Since $P(\neg B|\neg D) + P(B|\neg D) = 1$ we have $.602/P(\neg D) + .279/P(\neg D) = 1$ and so $P(\neg D) = .881$

- Thus, $P(\neg B|\neg D) = .602/.881 = .683$

Learning of Bayesian Networks

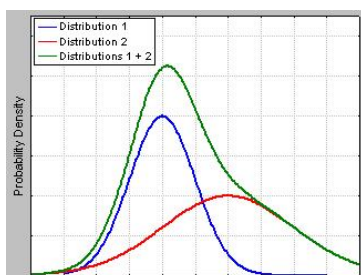
- 1 The network structure is given in advance and all the variables are fully observable in the training examples. ==> Trivial Case: just estimate the conditional probabilities.
- 2 The network structure is given in advance but only some of the variables are observable in the training data. ==> Similar to learning the weights for the hidden units of a Neural Net: Gradient Ascent Procedure
- 3 The network structure is not known in advance. ==> Use a heuristic search or constraint-based technique to search through potential structures.

Data clustering: K-means

- Suppose that you already have hypotheses concerning the number of clusters in your training set.
- You may want to form exactly 3 clusters that are to be as distinct as possible.
- In general, the *k-means* method will produce exactly *k* different clusters of greatest possible distinction.
- You start with *k* random clusters and then move objects between those clusters with the goal to 1) minimize variability within clusters and 2) maximize variability between clusters.

EM

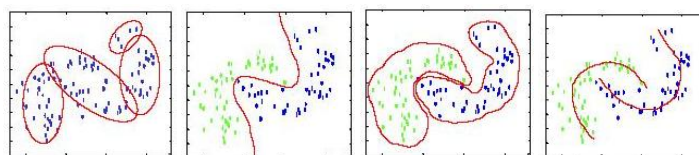
Given are two normal distributions with different means and stds and their sum. Only the mixture (sum) is assumed known. EM estimates the means and stds for each cluster so as to maximize the likelihood of the observed data (distribution). Or, the EM algorithm attempts to approximate the observed distributions of values based on mixtures of different distributions in different clusters.



Pattern recognition

From left to right:

- **Clustering:** find natural groups of samples in unlabeled data
- **Classification:** find functions separating the classes - learning discrete target function
- **Density estimation:** make a statistical model of the data
- **Regression:** fit lines or other functions to data - learning continuous functions



Expectation Maximization (EM)

Learning with unobservable relevant variables.

- Example: Assume that data points have been uniformly generated from *k* distinct Gaussian with the same known variance. The problem is to output a hypothesis $h = \langle \mu_1, \mu_2, \dots, \mu_k \rangle$ that describes the means of each of the *k* distributions. In particular, we are looking for a maximum likelihood hypothesis for these means.
- We extend the problem description as follows: for each point x_i , there are *k* hidden variables z_{i1}, \dots, z_{ik} such that $z_{il} = 1$ if x_i was generated by normal distribution *l* and $z_{iq} = 0$ for all $q \neq l$.

Expectation Maximization

Usage:

- Train Bayesian Belief Networks
- Unsupervised clustering
- Learning Hidden Markov Models

EM

Given:

- Instances from X generated by mixture of k Gaussian distributions
- Unknown means $\langle \mu_1, \dots, \mu_k \rangle$ of the k Gaussians (known and equal variances)
- Don't know which instance x_i was generated by which Gaussian

Determine:

- Maximum likelihood estimates of $\langle \mu_1, \dots, \mu_k \rangle$

Think of full description of each instance as $y_i = \langle x_i, z_{i1}, z_{i2} \rangle$, where

- z_{ij} is 1 if x_i generated by j th Gaussian
- x_i observable
- z_{ij} unobservable

EM

EM Algorithm: Pick random initial $h = \langle \mu_1, \mu_2 \rangle$, then iterate

E step:

Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij} , assuming the current hypothesis $h = \langle \mu_1, \mu_2 \rangle$ holds.

$$E[z_{ij}] = \frac{P(x = x_i | \mu = \mu_j)}{\sum_{n=1}^2 P(x = x_i | \mu = \mu_n)} = \frac{e^{-\frac{1}{2\sigma^2}(x_i - \mu_j)^2}}{\sum_{n=1}^2 e^{-\frac{1}{2\sigma^2}(x_i - \mu_n)^2}}$$

M step:

Calculate a new maximum likelihood hypothesis $h' = \langle \mu'_1, \mu'_2 \rangle$, assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated above. Replace $h = \langle \mu_1, \mu_2 \rangle$ by $h' = \langle \mu'_1, \mu'_2 \rangle$.

$$\mu_j \leftarrow \frac{\sum_{i=1}^m E[z_{ij}] x_i}{\sum_{i=1}^m E[z_{ij}]}$$

INSTANCE BASED LEARNING

EM

The EM algorithm searches for a ML hypothesis by repeatedly estimating the expected values of the hidden variables z_{ij} given its current hypothesis $\langle \mu_1, \dots, \mu_k \rangle$, then recalculating the ML hypothesis using these expected values for the hidden variables.

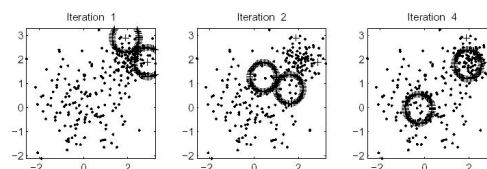


Figure 1: The progress of the K-Means algorithm with $k = 2$ and random initialization on the two-Gaussian data set (note: some data points omitted for clarity).

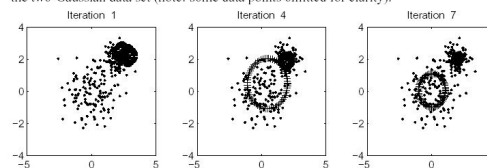


Figure 2: The progress of the EM algorithm with $k = 2$ and random initialization on the two-Gaussian data set (note: some data points omitted for clarity). The radius of the circle around each Gaussian is set to its variance.

Instance-Based Learning

- no explicit description of the target function
- generalization postponed until new instance has to be classified
- "lazy learning"
- local approximation of the target function
- High cost: training vs. classification!!!
- Case-base reasoning: more complex, symbolic representations for instances

Instance Based Learning

- k -Nearest Neighbor
- Locally weighted regression
- Radial basis functions
- Local linear models
- Case-based reasoning

Instance-Based Learning

Key idea: just store all training examples $\langle x_i, f(x_i) \rangle$

Nearest neighbor:

- Given query instance x_q , first locate nearest training example x_n , then estimate $\hat{f}(x_q) \leftarrow f(x_n)$

k -Nearest neighbor:

- Given x_q , take vote among its k nearest nbrs (if discrete-valued target function)
- take mean of f values of k nearest nbrs (if real-valued)

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

Nearest neighbor

- Instances map to points in n -dimensional space \mathfrak{R}^n
- Nearest neighbors defined by standard Euclidean distance given their feature vectors

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

When To Consider Nearest Neighbor

- Less than 20 attributes per instance
- Lots of training data

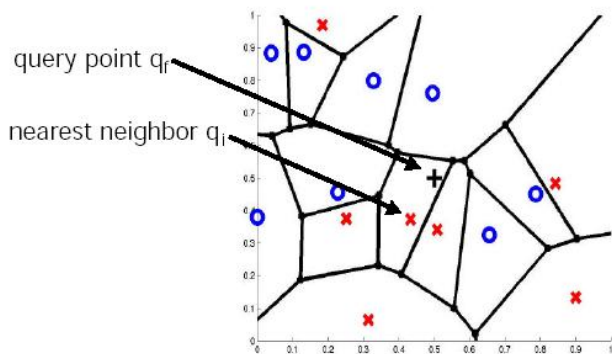
Advantages:

- Training is very fast
- Learn complex target functions
- Don't lose information

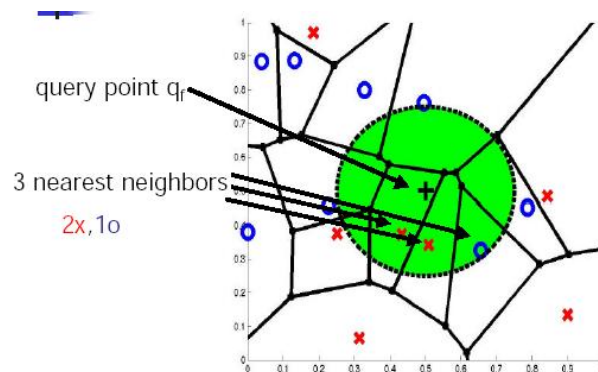
Disadvantages:

- Slow at query time
- Easily fooled by irrelevant attributes

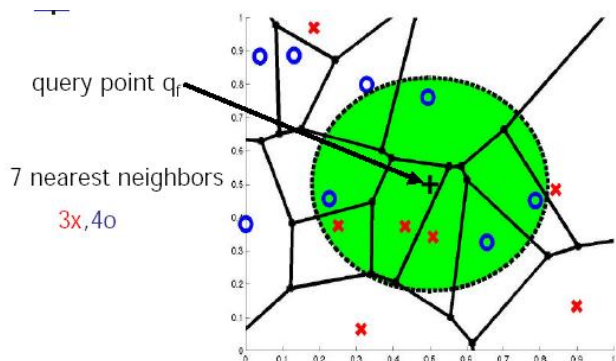
Voronoi Diagram



3-Nearest Neighbors



7-Nearest neighbors



Behavior in the Limit

Consider $p(x)$ defines probability that instance x will be labeled 1 (positive) versus 0 (negative).

Nearest neighbor:

- As number of training examples $\rightarrow \infty$, approaches Gibbs Algorithm
- Gibbs: with probability $p(x)$ predict 1, else 0

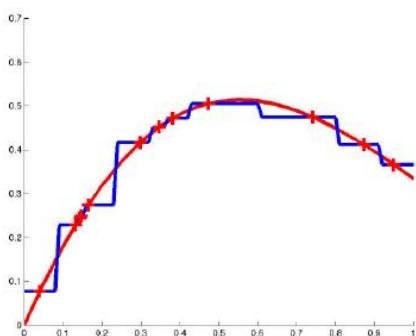
k -Nearest neighbor:

- As number of training examples $\rightarrow \infty$ and k gets large, approaches Bayes optimal
- Bayes optimal: if $p(x) > .5$ then predict 1, else 0

Note Gibbs has at most twice the expected error of Bayes optimal

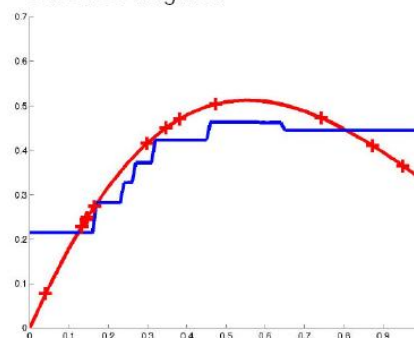
Nearest Neighbors (continuous)

1-nearest neighbor



Nearest Neighbors (continuous)

5-nearest neighbor



Locally Weighted Regression (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 $w_i = K(d(x_i, x_q))$

Locally Weighted Regression

Note k NN forms local approximation to f for each query point x_q

Why not form an explicit approximation $\hat{f}(x)$ for region surrounding x_q

- Fit linear function to k nearest neighbors
- Fit quadratic, ...
- Produces "piecewise approximation" to f

Several choices of error to minimize:

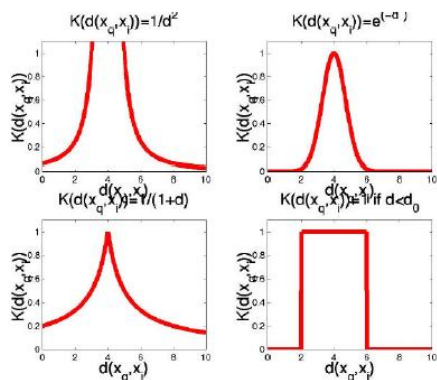
- Squared error over k nearest neighbors

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

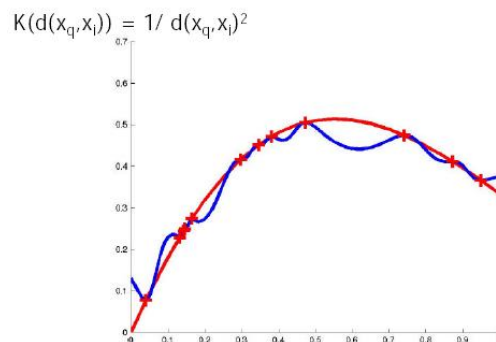
- Distance-weighted squared error over all nbrs

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

Kernel Functions



Distance Weighted NN



Distance Weighted NN

