Module 5

1. **Evaluating Hypothesis**: Motivation, **Estimating hypothesis accuracy**, Basics of sampling theorem, General approach for deriving confidence intervals, Difference in error of two hypothesis, Comparing learning algorithms.
2. **Instance Based Learning**: Introduction, k-nearest neighbor learning, locally weighted regression, radial basis function, cased-based reasoning,
3. **Reinforcement Learning**: Introduction, Learning Task, Q Learning .

5.1 : Evaluating Hypothesis

- Motivation,
- **Estimating hypothesis accuracy**,
- Basics of sampling theorem,
- General approach for deriving confidence intervals,
- Difference in error of two hypothesis,
- Comparing learning algorithms.

5.1.0 Introduction

- Empirically evaluating the accuracy of hypotheses is fundamental to machine learning.
- This chapter presents an introduction to statistical methods for estimating hypothesis accuracy, **focusing on three questions.**
  1. Given the observed accuracy of a hypothesis over a limited sample of data, how well does this ***estimate its accuracy over additional examples?***
  2. Given that ***one hypothesis outperforms another over some sample of data***, how probable is it that this hypothesis is more accurate in general?
  3. When ***data is limited what is the best way to use this data*** to both learn a hypothesis and estimate its accuracy?

5.1.1 : MOTIVATION

- In many cases it is important to evaluate the performance of learned hypotheses as precisely as possible.

1. One reason is simply to understand whether to use the hypothesis.
2. A second reason is that evaluating hypotheses is an integral component of many learning methods.

Estimating the accuracy of a hypothesis is relatively straightforward when data is plentiful. However, when we must learn a hypothesis and estimate its future accuracy given only a limited set of data, two key difficulties arise:

1. Bias in the estimate
2. Variance in the estimate

5.1.2 ESTIMATING HYPOTHESIS ACCURACY

Notation & Two Questions

A Hypothesis is an approximation of f(x), the target function

Notation

- X: the space of instances
- D: the probability distribution of encountering instances from X
- f: the target function
- H: the hypothesis space
- h: a particular hypothesis in H
- (x, f(x)): a training instance
- S: all training instances

Two Questions

- Given h constructed from n examples drawn randomly from D, what is the best estimate of h over future instances drawn from D?
- What is the probable error in this accuracy estimate?

5.1.2.1 : Sample Error

- **Sample Error** : is the error rate of the hypothesis over the sample of data that is available.

Definition: The **sample error** (denoted $error_S(h)$) of hypothesis h with respect to target function f and data sample S is

$$error_S(h) \equiv \frac{1}{n} \sum_{x \in S} \delta(f(x), h(x))$$

Where n is the number of examples in S, and the quantity $\delta(f(x), h(x))$ is 1 if $f(x) \neq h(x)$, and 0 otherwise.

5.1.2.1 : True Error

- **True Error** : is the error rate of the hypothesis over the entire unknown distribution **D** of examples. The true error of a hypothesis is the probability that it will misclassify a single randomly drawn instance from the distribution **D**.

Definition: The **true error** (denoted $error_{\mathcal{D}}(h)$) of hypothesis h with respect to target function f and distribution $\mathcal{D}$, is the probability that h will misclassify an instance drawn at random according to $\mathcal{D}$.

$$error_{\mathcal{D}}(h) \equiv \Pr_{x \in \mathcal{D}}[f(x) \neq h(x)]$$

Here the notation $\Pr_{x \in \mathcal{D}}$ denotes that the probability is taken over the instance distribution $\mathcal{D}$.

5.1.2.2 Confidence Intervals for Discrete-Valued Hypotheses

- Here we give an answer to the question "**How good an estimate of $error_D$(h) is provided by $error_S$(h)**?
- More specifically, suppose we wish to estimate the ***true error for some discrete- valued hypothesis h***, based on its observed sample error over a sample S, where
  - the sample S contains **n** examples drawn independent of one another, and independent of h, according to the probability distribution D
  - **n>=30**
  - hypothesis **h** commits **r** errors over these n examples (i.e., $error_S$**(h) = r/n**).

5.1.2.2 Confidence Intervals for Discrete-Valued Hypotheses

Under these conditions, statistical theory allows us to make the following assertions:

1. Given no other information, the most probable value of $error_{\mathcal{D}}(h)$ is $error_S(h)$
2. With approximately 95% probability, the true error $error_{\mathcal{D}}(h)$ lies in the interval

$$error_S(h) \pm 1.96\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

The above expression for the 95% confidence interval can be generalized to any desired confidence level. The constant 1.96 is used in case we desire a 95% confidence interval. A different constant, z_N, is used to calculate the $N\%$ confidence interval. The general expression for approximate $N\%$ confidence intervals for $error_{\mathcal{D}}(h)$ is

$$error_S(h) \pm z_N \sqrt{\frac{error_S(h)(1 - error_S(h))}{n}}$$

where the constant z_N is chosen depending on the desired confidence level, using the values of z_N given in Table 5.1.

Confidence level $N\%$:	50%	68%	80%	90%	95%	98%	99%
Constant z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

TABLE 5.1
Values of z_N for two-sided $N\%$ confidence intervals.

Reading Assignment

1. Two sided and one sided bound in Normal Distribution [page 141]
2. Hypothesis Testing [page 144 -145]
3. Comparing Learning Algorithms [145 – 148]
4. Paired t Tests [148-149]
5. Practical Considerations [149-150]

5.1.3.1 BASICS OF SAMPLING THEORY

- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- The *expected value*, or *mean*, of a random variable Y is $E[Y] = \sum_i y_i \Pr(Y = y_i)$. The symbol μ_Y is commonly used to represent E[Y].
- The *variance* of a random variable is $Var(Y) = E[(Y - \mu_Y)^2]$. The variance characterizes the width or dispersion of the distribution about its mean.
- The *standard deviation* of Y is $\sqrt{Var(Y)}$. The symbol σ_Y is often used used to represent the standard deviation of Y.

- The *Binomial distribution* gives the probability of observing r heads in a series of n independent coin tosses, if the probability of heads in a single toss is p.
- The *Normal distribution* is a bell-shaped probability distribution that covers many natural phenomena.
- The *Central Limit Theorem* is a theorem stating that the sum of a large number of independent, identically distributed random variables approximately follows a Normal distribution.
- An *estimator* is a random variable Y used to estimate some parameter p of an underlying population.
- The *estimation bias* of Y as an estimator for p is the quantity $(E[Y] - p)$. An unbiased estimator is one for which the bias is zero.
- A *N% confidence interval* estimate for parameter p is an interval that includes p with probability $N\%$.

Error Estimation and Estimating Binomial Proportions

A *Binomial distribution* gives the probability of observing r heads in a sample of n independent coin tosses, when the probability of heads on a single coin toss is p. It is defined by the probability function

$$P(r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r}$$

If the random variable X follows a Binomial distribution, then:

- The probability $\Pr(X = r)$ that X will take on the value r is given by $P(r)$
- The expected, or mean value of X, $E[X]$, is

$$E[X] = np$$

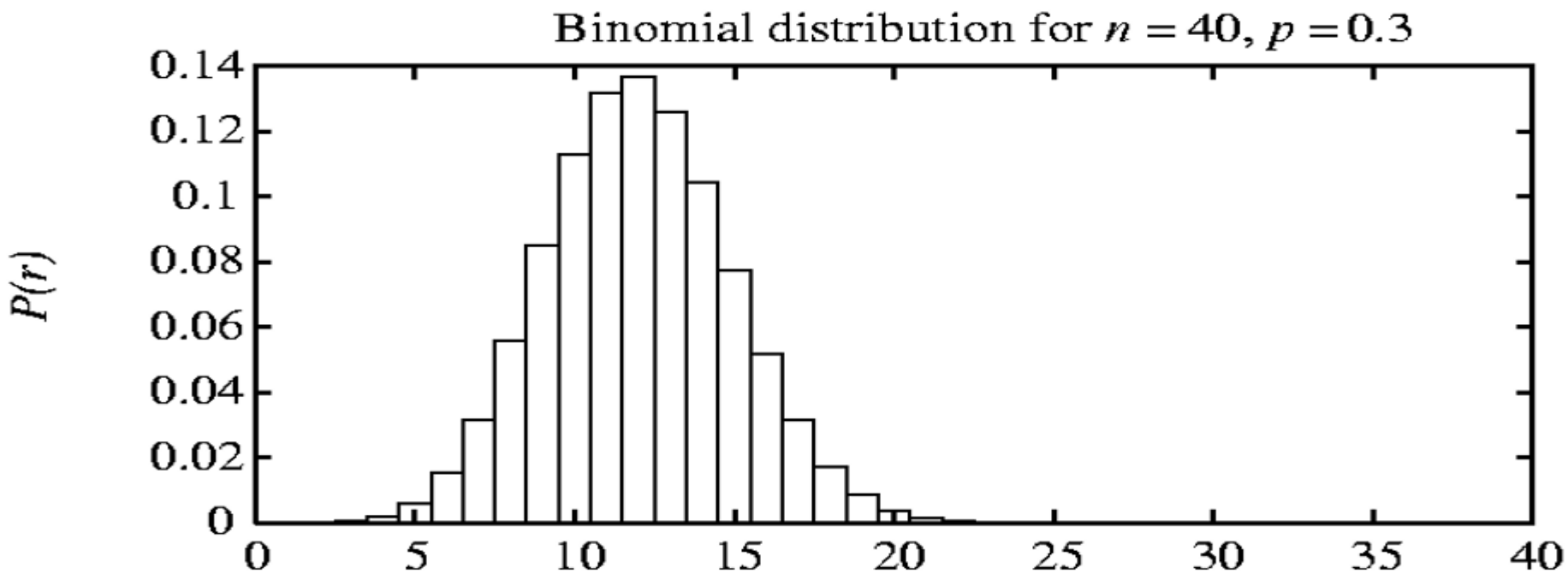
- The variance of X, $Var(X)$, is

$$Var(X) = np(1-p)$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sqrt{np(1-p)}$$

Binomial Probability Distribution



5.1.3.3 Mean and Variance

- Two properties of a random variable that are often of interest are its ***expected value (also called its mean value) and its variance***. The expected value is the average of the values taken on by repeatedly sampling the random variable.

Definition: Consider a random variable Y that takes on the possible values $y_1, \ldots y_n$. The **expected value** of Y, $E[Y]$, is

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i)$$

For example, if Y takes on the value 1 with probability .7 and the value 2 with probability .3, then its expected value is $(1 \cdot 0.7 + 2 \cdot 0.3 = 1.3)$. In case the random variable Y is governed by a Binomial distribution, then it can be shown that

$$E[Y] = np$$

where n and p are the parameters of the Binomial distribution

A second property, the variance, captures the "width" or "spread" of the probability distribution; that is, it captures how far the random variable is expected to vary from its mean value.

Definition: The **variance** of a random variable Y, $Var[Y]$, is

$$Var[Y] \equiv E[(Y - E[Y])^2]$$

The variance describes the expected squared error in using a single observation of Y to estimate its mean $E[Y]$. The square root of the variance is called the *standard deviation* of Y, denoted σ_Y.

Definition: The **standard deviation** of a random variable Y, σ_Y, is

$$\sigma_Y \equiv \sqrt{E[(Y - E[Y])^2]}$$

In case the random variable Y is governed by a Binomial distribution, then the variance and standard deviation are given by

$$Var[Y] = np(1-p)$$

$$\sigma_Y = \sqrt{np(1-p)}$$

5.1.3.4 Estimators, Bias and Variance

- Since $\boldsymbol{error_S(h)}$ (an ***estimator*** for the true error) obeys a Binomial distribution

  we have: $\boldsymbol{error_S(h) = r/n}$ and $\boldsymbol{error_D(h) = p}$

  where ***n*** is the number of instances in the sample ***S***, ***r*** is the number of instances from ***S*** misclassified by ***h***, and ***p*** is the probability of misclassifying a single instance drawn from *D*.

- **Definition:** The ***estimation bias*** ($\neq$ from the inductive bias) of an estimator *Y* for an arbitrary parameter *p* is $\boldsymbol{E[Y] - p}$

- The ***standard deviation*** for $error_S(h)$ is given by

$$\boldsymbol{\sqrt{p(1-p)/n} \approx \sqrt{error_S(h)(1-error_S(h))/n}}$$

In general, given r errors in a sample of n independently drawn test examples, the standard deviation for $error_S(h)$ is given by

$$\sigma_{error_S(h)} = \frac{\sigma_r}{n} = \sqrt{\frac{p(1-p)}{n}}$$

which can be approximated by substituting $r/n = error_S(h)$ for p

$$\sigma_{error_S(h)} \approx \sqrt{\frac{error_S(h)(1-error_S(h))}{n}}$$

Bias and Variance

- *Bias:* If S is training set, $error_S(h)$ is optimistically biased

$$bias \equiv E[error_S(h)] - error_{\mathcal{D}}(h)$$

If there is no bias, then

$$E[error_S(h)] = error_{\mathcal{D}}(h)$$

It will be the case when S is chosen independent of h.

- *Variance:* Even with unbiased S, $error_S(h)$ may still *vary* from $error_{\mathcal{D}}(h)$

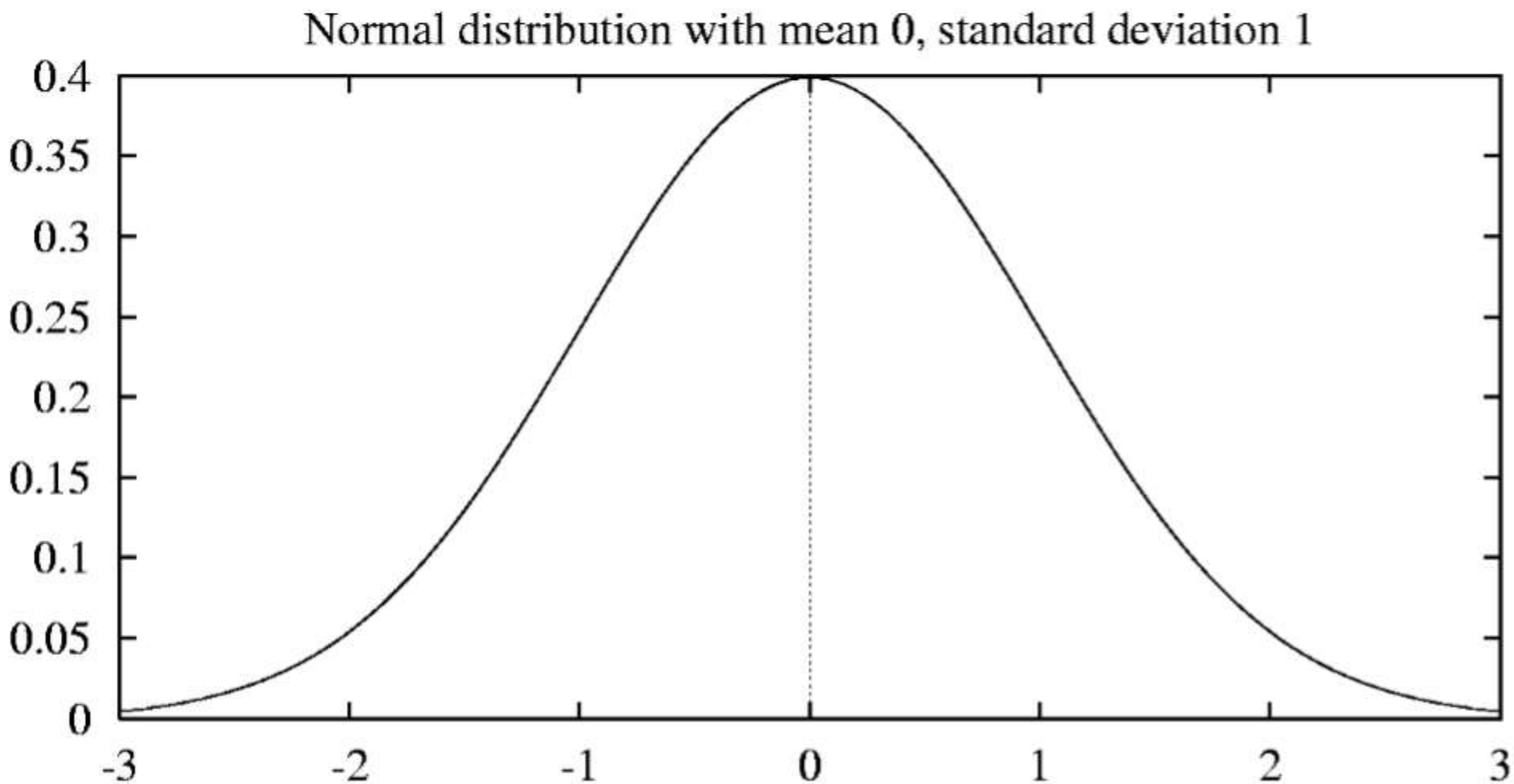
5.1.3.5 Confidence Intervals

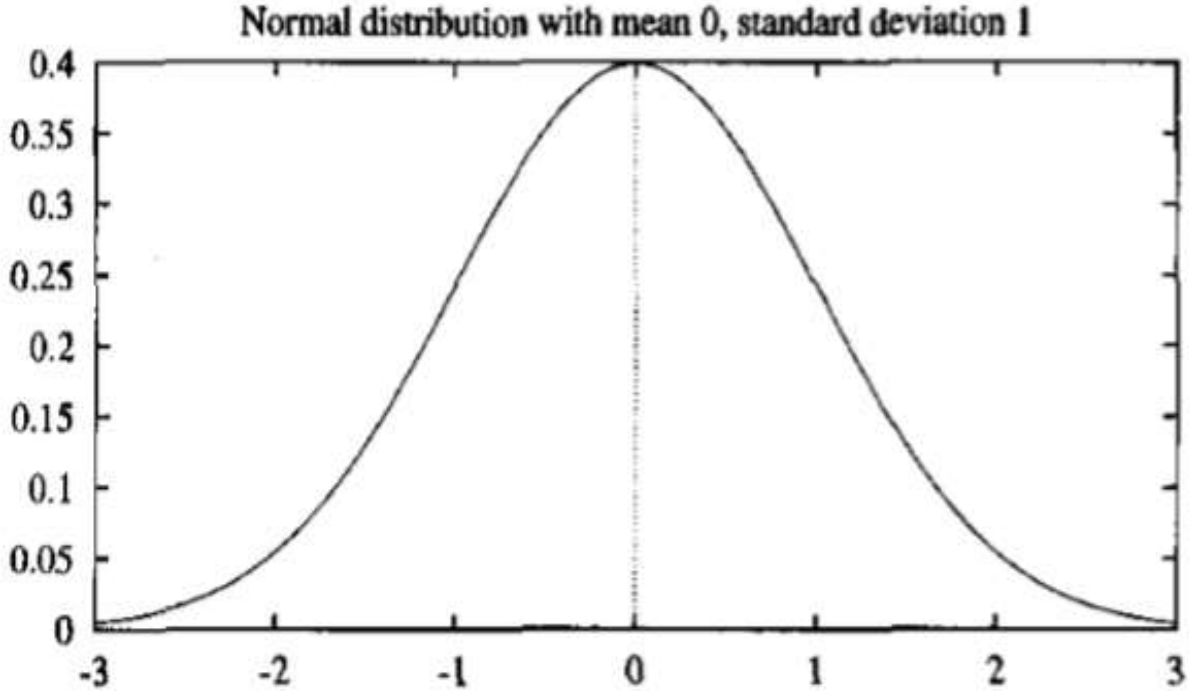
One common way to describe the uncertainty associated with an estimate is to give an interval within which the true value is expected to fall, along with the probability with which it is expected to fall into this interval. Such estimates are called *confidence interval* estimates.

> ***Definition:*** An $N\%$ **confidence interval** for some parameter p is an interval that is expected with probability $N\%$ to contain p.

For example, if we observe $r = 12$ errors in a sample of $n = 40$ independently drawn examples, we can say with approximately 95% probability that the interval 0.30 ± 0.14 contains the true error $error_{\mathcal{D}}(h)$.

Normal Probability Distribution





A Normal distribution (also called a Gaussian distribution) is a bell-shaped distribution defined by the probability density function

$$p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2}(\frac{x-\mu}{\sigma})^2}$$

A Normal distribution is fully determined by two parameters in the above formula: μ and σ.

If the random variable X follows a normal distribution, then:

- The probability that X will fall into the interval (a, b) is given by

$$\int_a^b p(x)dx$$

- The expected, or mean value of X, $E[X]$, is

$$E[X] = \mu$$

- The variance of X, $Var(X)$, is

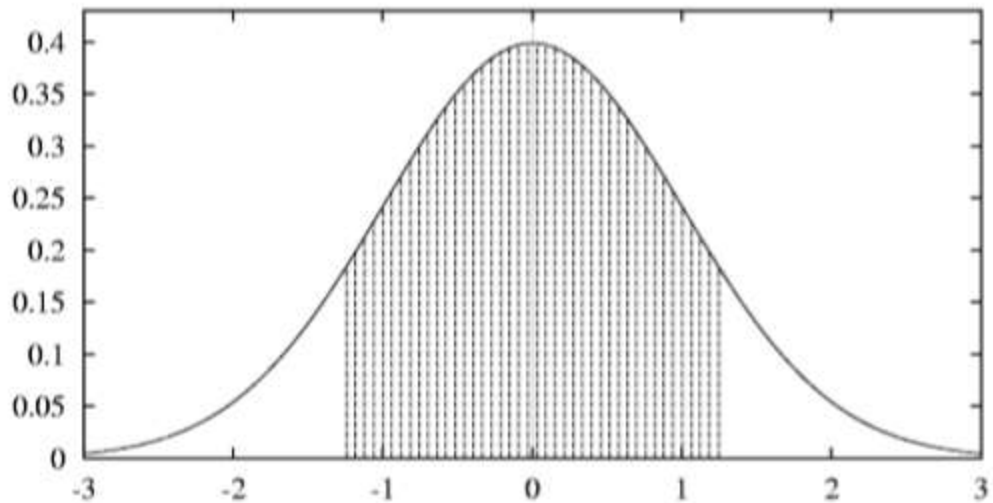
$$Var(X) = \sigma^2$$

- The standard deviation of X, σ_X, is

$$\sigma_X = \sigma$$

The Central Limit Theorem (Section 5.4.1) states that the sum of a large number of independent, identically distributed random variables follows a distribution that is approximately Normal.

Normal Probability Distribution



- 80% of area (probability) lies in $\mu \pm 1.28\sigma$
- N% of area (probability) lies in $\mu \pm z_N \sigma$

$N\%$:	50%	68%	80%	90%	95%	98%	99%
z_N:	0.67	1.00	1.28	1.64	1.96	2.33	2.58

A GENERAL APPROACH FOR DERIVING CONFIDENCE INTERVALS

- The general process includes the following steps:

1. Identify the underlying population parameter p to be estimated, for example, $error_{\mathcal{D}}(h)$.
2. Define the estimator Y (e.g., $error_S(h)$). It is desirable to choose a minimum-variance, unbiased estimator.
3. Determine the probability distribution $\mathcal{D}_Y$ that governs the estimator Y, including its mean and variance.
4. Determine the $N\%$ confidence interval by finding thresholds L and U such that $N\%$ of the mass in the probability distribution $\mathcal{D}_Y$ falls between L and U.

Estimating Confidence Intervals In General

1. Pick parameter p to estimate
   - $error_{\mathcal{D}}(h)$
2. Choose an estimator
   - $error_S(h)$
3. Determine probability distribution that governs estimator
   - $error_S(h)$ governed by Binomial distribution, approximated by Normal when $n \geq 30$
4. Find interval (L, U) such that N% of probability mass falls in the interval
   - Use table of z_N values

Central Limit Theorem

Theorem 5.1. Central Limit Theorem. Consider a set of independent, identically distributed random variables $Y_1 \ldots Y_n$ governed by an arbitrary probability distribution with mean μ and finite variance σ^2. Define the sample mean, $\bar{Y}_n \equiv \frac{1}{n} \sum_{i=1}^{n} Y_i$.

Then as $n \to \infty$, the distribution governing

$$\frac{\bar{Y}_n - \mu}{\frac{\sigma}{\sqrt{n}}}$$

approaches a Normal distribution, with zero mean and standard deviation equal to 1.

DIFFERENCE IN ERROR OF TWO HYPOTHESES

- Difference in True error

$$d \equiv error_{\mathcal{D}}(h_1) - error_{\mathcal{D}}(h_2)$$

- Difference Sample error

$$\hat{d} \equiv error_{S_1}(h_1) - error_{S_2}(h_2)$$

- Approximate Variance

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}$$

- Approximate N% confidence interval estimate for d is

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1 - error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1 - error_{S_2}(h_2))}{n_2}}$$

Comparing Learning Algorithms

- How can we tell if one algorithm can learn better than another?

1. Design an experiment to measure the accuracy of the two algorithms
2. Run multiple trials
3. Compare the samples not just their means . Do a statistically sound test of the two samples.
4. Is any observed difference significant? Is it due to true difference between algorithms or natural variations in the measurements?

Comparing Learning Algorithms

- Which of L_A and L_B is the better learning method on average for learning some particular target function f ?
- To answer this question, we wish to estimate the expected value of the difference in their errors:

$$E_{S \subset D}[error_D(L_A(S)) - error_D(L_B(S))]$$

- Of course, since we have only a limited sample D_0 we estimate this quantity by dividing D_0 into a ***training set*** S_0 and a ***testing set*** T_0 and measure:

$$error_{T0}(L_A(S_0)) - error_{T0}(L_B(S_0))$$

A procedure to estimate the difference in error between two learning methods **LA** and **LB**.

1. Partition the available data D_0 into k disjoint subsets $T_1, T_2, \ldots, T_k$ of equal size, where this size is at least 30.
2. For i from 1 to k, do

   use T_i for the test set, and the remaining data for training set S_i

   - $S_i \leftarrow \{D_0 - T_i\}$
   - $h_A \leftarrow L_A(S_i)$
   - $h_B \leftarrow L_B(S_i)$
   - $\delta_i \leftarrow error_{T_i}(h_A) - error_{T_i}(h_B)$
3. Return the value $\bar{\delta}$, where

$$\bar{\delta} \equiv \frac{1}{k} \sum_{i=1}^{k} \delta_i$$

Reading Assignment

1. Two sided and one sided bound in Normal Distribution [page 141]
2. Hypothesis Testing [page 144 -145]
3. Comparing Learning Algorithms [145 – 148]
4. Paired t Tests [148-149]
5. Practical Considerations [149-150]

5.2Instance Based Learning : Introuction

- Introduction, k-nearest neighbor learning, locally weighted regression, radial basis function, cased-based reasoning,

5.2Instance Based Learning : Introduction

- All learning methods presented so far construct a general explicit description of the target function when examples are provided.
- IBL methods simply store the training examples instead of learning explicit description of the target function.
- Generalizing the examples is postponed until a new instance must be classified .
- When a new instance is encountered , its relationship to the stored examples is examined in order to assign a target function value for the new instance.

5.2Instance Based Learning

- **Memory-based learning** (also called **instance-based learning**) is a type of learning algorithm that compares new test data with training data in order to solve the given machine learning problem. Such algorithms **search for the training data that are most similar to the test data** and make predictions based on these similarities.
- Learning in these algorithms consists of simply storing the presented training data. When a new query instance is encountered a set of similar related instances is retrieved from memory and used to classify the new query instance
- **Examples :** *k-nearest neighbor learning , locally weighted regression,* ***Radial Basis function (RBF),*** *kernel machines and Case based Reasoning .*

Key Advantage

Instead of estimating the target function once for the entire data set (which can lead to complex and not necessarily accurate functions) IBL can estimate the required function locally and differently for each new instance to be classified.

Advantages

- Instead of estimating for the whole instance space, local approximations to the target function are possible.
- Especially if target functions is complex but still decomposable

Disadvantages

- Classification costs are high
- Typically all attributes are considered when attempting to retrieve similar training examples.

k-NEAREST NEIGHBOR LEARNING

- The most basic instance-based method is the k-NEAREST NEIGHBOR algorithm. This algorithm assumes all instances correspond to points in the n-dimensional space R^n.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.

More precisely let an arbitrary instance x be described by the feature vector (set of attributes) as follows:

$$< a_1(x), a_2(x), ... a_n(x) >$$

where *ar(x)* denotes the value of the rth attribute of instance *x*. Then the Euclidean distance between two instances xi and xj is defined to be $d(x_i, x_j)$ where

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

The K-NN algorithm for approximating a discrete valued function

Consider, the case of learning a discrete-valued target function of the form $f : \Re^n \rightarrow V$, where V is the finite set $\{v_1, \ldots v_s\}$

Training Algorithm :

- For each training example $<x , f(x)>$, add the example to the list of *training_examples*

Classification Algorithm

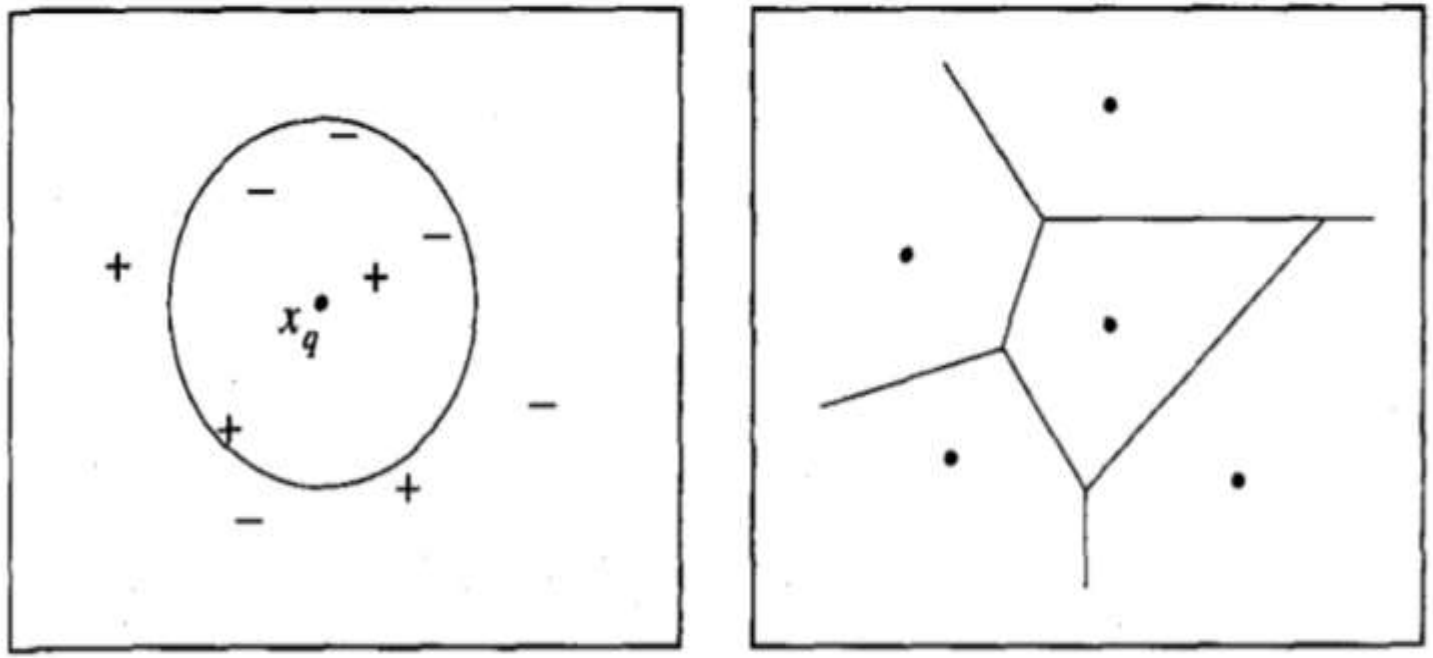
- Given a query instance x_q to be classified
  - Let $x_1 \ldots x_k$ denote the k instances from *training_examples* that are nearest to x_q
  - Return $$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} \delta(v, f(x_i))$$

where $\delta(a, b) = 1$ if $a = b$

$= 0$ otherwise

where argmax $f(x)$ returns the value of x which maximises $f(x)$,

e.g. $\underset{x \in \{1,2,-3\}}{\operatorname{argmax}}(x^2) = -3$



FIGURE

k-NEAREST NEIGHBOR. A set of positive and negative training examples is shown on the left, along with a query instance x_q to be classified. The 1-NEAREST NEIGHBOR algorithm classifies x_q positive, whereas 5-NEAREST NEIGHBOR classifies it as negative. On the right is the decision surface induced by the 1-NEAREST NEIGHBOR algorithm for a typical set of training examples. The convex polygon surrounding each training example indicates the region of instance space closest to that point (i.e., the instances for which the 1-NEAREST NEIGHBOR algorithm will assign the classification belonging to that training example).

The decision surface is a combination of convex polyhedra surrounding each of the training examples. For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example. Query points outside the polyhedron are closer to some other training example. This kind of diagram is often called the *Voronoi diagram* of the set of training examples

Continuous vs Discrete valued functions (classes)

K-NN works well for discrete-valued target functions. Furthermore, the idea can be extended f or continuos (real) valued functions. In this case we can take mean of the f values of k nearest neighbors:

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

Distance-weighted k-NN

Might want to weigh nearer neighbors more heavily

$$\hat{f}(x_q) \leftarrow \underset{v \in V}{\operatorname{argmax}} \sum_{i=1}^{k} w_i \delta(v, f(x_i)) \qquad \text{where } w_i = \frac{1}{d(x_q, x_i)^2}$$

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

For continuous functions:

$$\hat{f}(x_q) \leftarrow \frac{\sum_{i=1}^{k} w_i f(x_i)}{\sum_{i=1}^{k} w_i} \qquad \text{where } w_i = \frac{1}{d(x_q, x_i)^2}$$

Note now it may make more sense to use all training examples instead of just k.

Curse of Dimensionality : Remarks on KNN

Imagine instances described by 20 attributes, but only 2 are relevant to target function: Instances that have identical values for the two relevant attributes may nevertheless be distant from one another
in the 20-dimensional space.

Curse of dimensionality: nearest neighbor is easily misled when high-dimensional X. (Compare to decision trees).

One approach: Weight each attribute differently (Use training)

1) Stretch j^{th} axis by weight z_j , where $z_1, \ldots, z_n$ chosen to minimize prediction error
2) Use cross-validation to automatically choose weights $z_1, \ldots, z_n$
3) Note setting z_j to zero eliminates dimension i altogether

When To Consider Nearest Neighbor ?

- Instances map to points in $\Re^n$
- Average number of attributes (e.g. Less than 20 attributes per instance)
- Lots of training data
- When target function is complex but can be approximated by separate local simple approximations

Advantages:	**Disadvantages:**
Training is very fast	Slow at query time
Learn complex target functions	Easily fooled by irrelevant attributes

Note that efficient methods do exist to allow fast querying (kd-trees)

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)

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

KNN Performance

The performance of the KNN Algorithm is influenced by three main factors :

1. The distance function or distance metric used to determine the nearest neighbors
2. The decision rule used to derive a classification from the K nearest neighbors
3. The number of neighbors used to classify the new example.

Advantages

- The KNN algorithm is very easy to implement
- Nearly optimal in the large sample limit
- Uses local information which can yield highly adaptive behavior
- Lends itself very easily to parallel implementation

Disadvantages

- Large storage requirements
- Computationally intensive recall
- Highly susceptible to the curse of dimensionality

A Note on Terminology

Much of the literature on nearest-neighbor methods and weighted local regression uses a terminology that has arisen from the field of statistical pattern recognition. In reading that literature, it is useful to know the following terms:

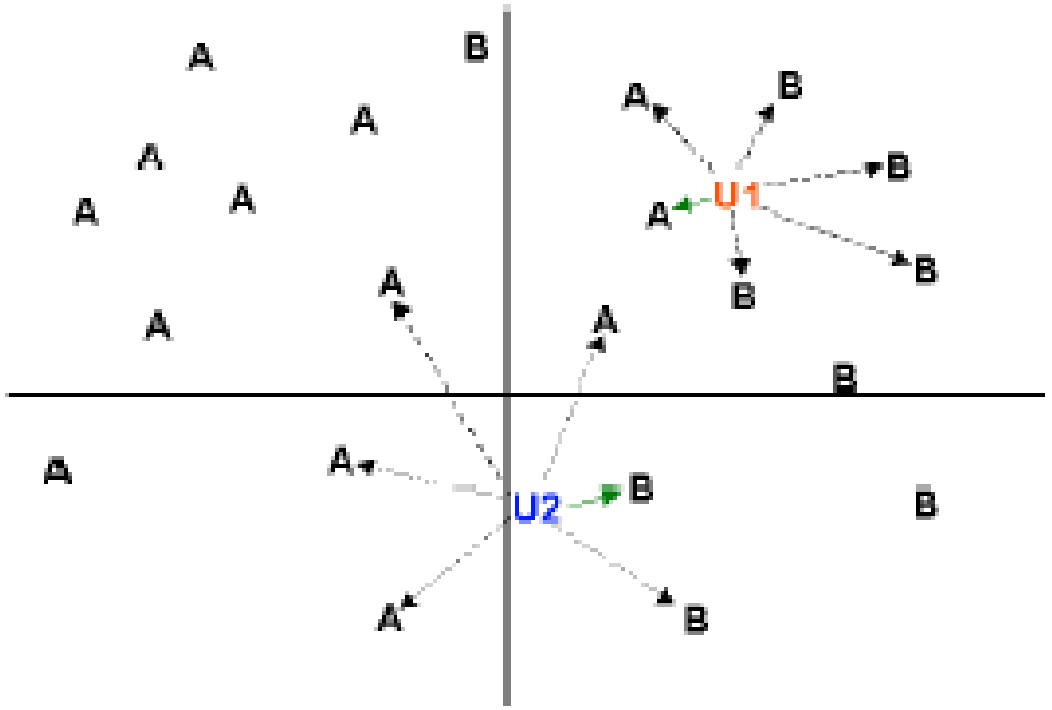
- *Regression* means approximating a real-valued target function.
- *Residual* is the error $\hat{f}(x) - f(x)$ in approximating the target function.
- *Kernel function* is the function of distance that is used to determine the weight of each training example. In other words, the kernel function is the function K such that $w_i = K(d(x_i, x_q))$.

Lab Program

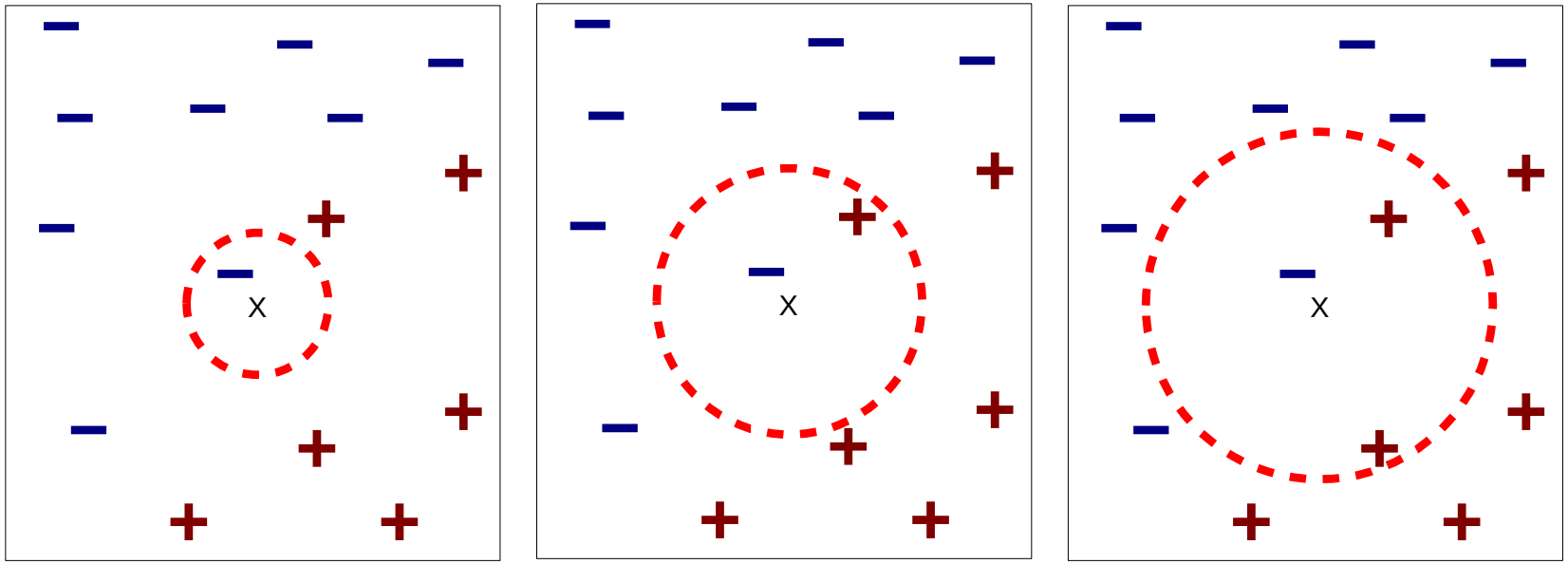
- Write a program to implement k-Nearest Neighbour algorithm to classify the iris data set. Print both correct and wrong predictions. Python ML library classes can be used for this problem.

K-Nearest-Neighbor Algorithm

- **Principle:** points (documents) that are close in the space belong to the same class



Definition of Nearest Neighbor



(a) 1-nearest neighbor (b) 2-nearest neighbor (c) 3-nearest neighbor

K-nearest neighbors of a record x are data points that have the k smallest distance to x

Nearest Neighbor Classification

- Compute distance between two points:
  - Euclidean distance

$$d(p,q) = \sqrt{\sum_i (p_i - q_i)^2}$$

Distance Metrics

Minkowsky:

$$D(\boldsymbol{x},\boldsymbol{y}) = \left(\sum_{i=1}^{m} |x_i - y_i|^r \right)^{1/r}$$

Euclidean:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sqrt{\sum_{i=1}^{m} (x_i - y_i)^2}$$

Manhattan / city-block:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} |x_i - y_i|$$

Camberra:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{|x_i - y_i|}{|x_i + y_i|}$$

Chebychev:

$$D(\boldsymbol{x},\boldsymbol{y}) = \max_{i=1}^{m} |x_i - y_i|$$

Quadratic:

$$D(\boldsymbol{x},\boldsymbol{y}) = (\boldsymbol{x} - \boldsymbol{y})^T Q(\boldsymbol{x} - \boldsymbol{y}) = \sum_{j=1}^{m} \left(\sum_{i=1}^{m} (x_i - y_i) q_{ji} \right) (x_j - y_j)$$

Q is a problem-specific positive definite $m \times m$ weight matrix

Mahalanobis:

$$D(\boldsymbol{x},\boldsymbol{y}) = [\det V]^{1/m} (\boldsymbol{x} - \boldsymbol{y})^{\mathrm{T}} V^{-1} (\boldsymbol{x} - \boldsymbol{y})$$

V is the covariance matrix of $A_1..A_m$, and A_j is the vector of values for attribute j occuring in the training set instances $1..n$.

Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = \frac{\sum_{i=1}^{m} (x_i - \overline{x_i})(y_i - \overline{y_i})}{\sqrt{\sum_{i=1}^{m} (x_i - \overline{x_i})^2 \sum_{i=1}^{m} (y_i - \overline{y_i})^2}}$$

$\overline{x_i} = \overline{y_i}$ and is the average value for attribute i occuring in the training set.

Chi-square:

$$D(\boldsymbol{x},\boldsymbol{y}) = \sum_{i=1}^{m} \frac{1}{sum_i} \left(\frac{x_i}{size_{\boldsymbol{x}}} - \frac{y_i}{size_{\boldsymbol{y}}} \right)^2$$

sum_i is the sum of all values for attribute i occuring in the training set, and $size_{\boldsymbol{x}}$ is the sum of all values in the vector $\boldsymbol{x}$.

Kendall's Rank Correlation:

$$D(\boldsymbol{x},\boldsymbol{y}) = 1 - \frac{2}{n(n-1)} \sum_{i=1}^{m} \sum_{j=1}^{i-1} \mathrm{sign}(x_i - x_j)\,\mathrm{sign}(y_i - y_j)$$

sign(x)=-1, 0 or 1 if $x < 0$, $x = 0$, or $x > 0$, respectively.

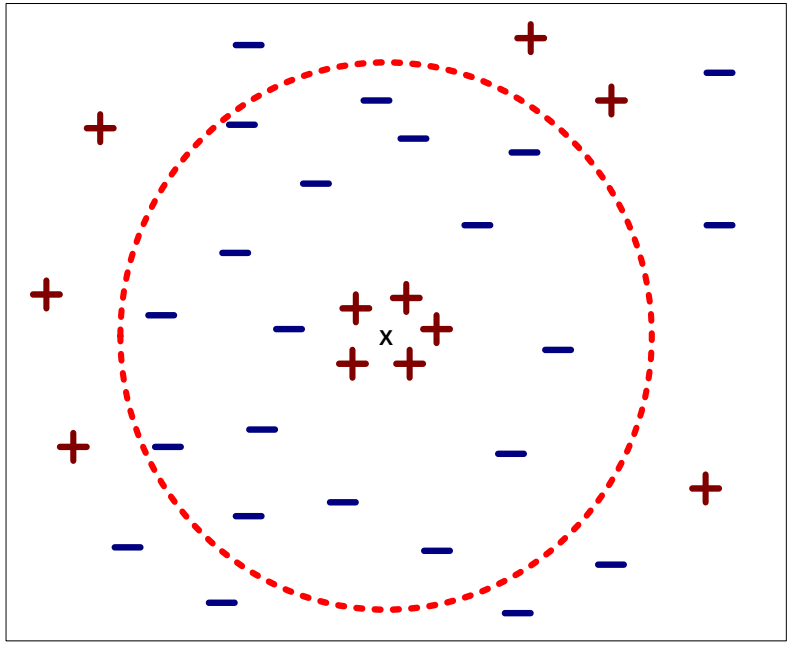
Figure 1. Equations of selected distance functions. ($\boldsymbol{x}$ and $\boldsymbol{y}$ are vectors of m attribute values).

Selection of Distance Metrics

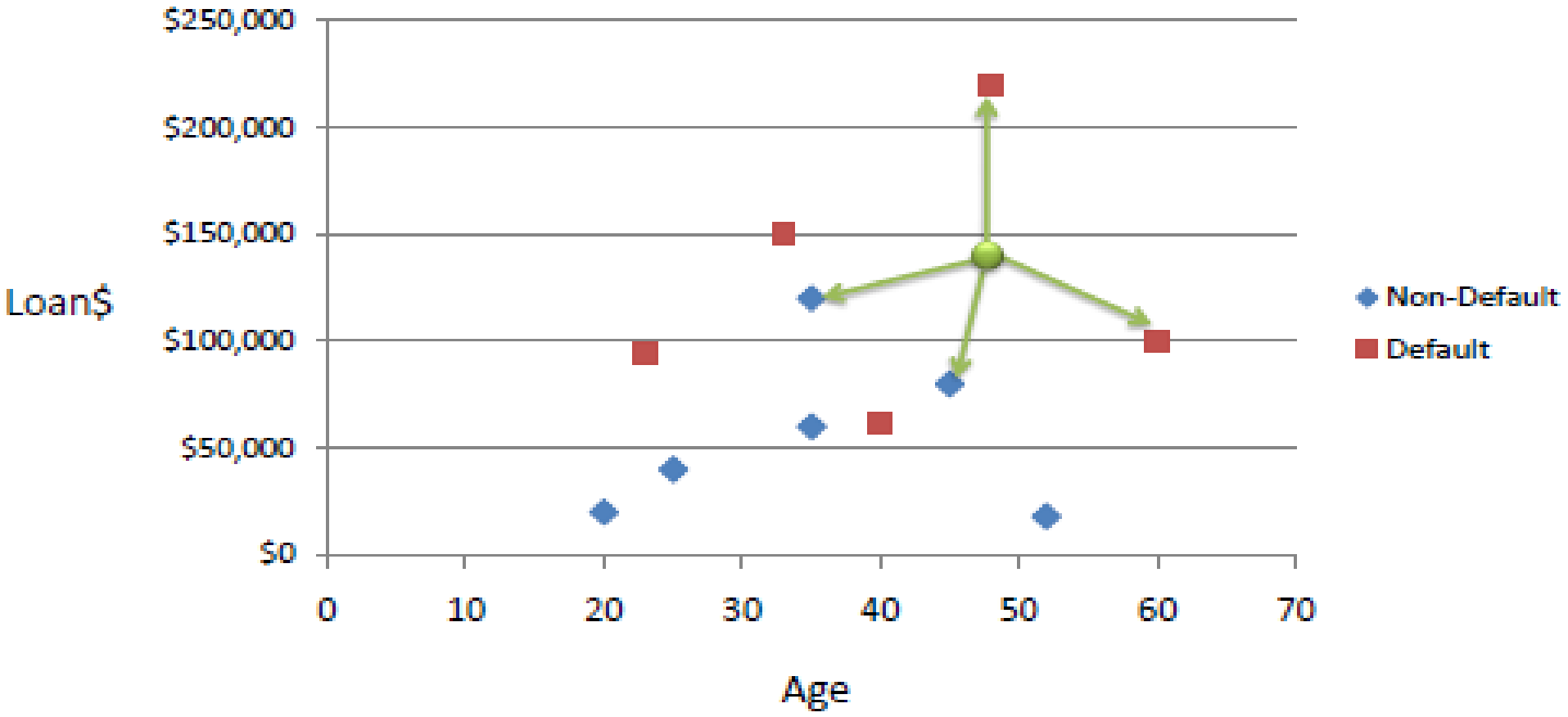
- You can choose the best distance metric based on the properties of your data. *If you are unsure, you can experiment with different distance metrics and different values of K together and see which mix results in the most accurate models*.
- Euclidean is a good distance measure to use if ***the input variables are similar in type (e.g. all measured widths and heights***).
- Manhattan distance is a good measure to use if the input variables are not similar in type (such as age, gender, height, etc.).

Nearest Neighbor Classification…

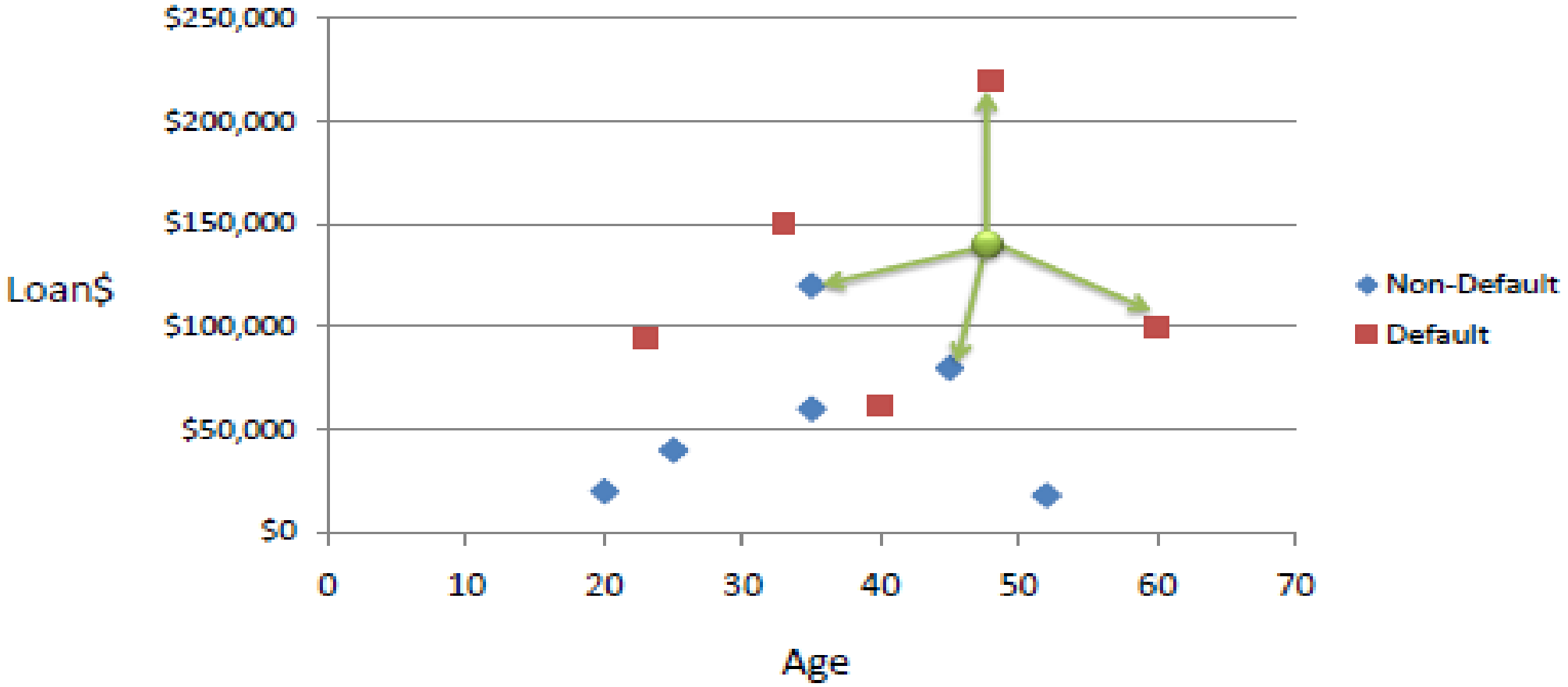
- **Choosing the value of k:**
  - If k is too small, sensitive to noise points
  - If k is too large, neighborhood may include points from other classes



Example: Consider the following data concerning credit default. Age and Loan are two numerical variables (predictors) and Default is the target.



Example: We can now use the training set to classify an unknown case (Age=48 and Loan=$142,000) using Euclidean distance. If K=1 then the nearest neighbor is the last case in the training set with Default=Y.



D = sqrt[(48-33)^2 + (142000-150000)^2] = 8000.01 >> Default=Y

Age	Loan	Default	Distance	
25	$40,000	N	102000	
35	$60,000	N	82000	
45	$80,000	N	62000	
20	$20,000	N	122000	
35	$120,000	N	22000	2
52	$18,000	N	124000	
23	$95,000	Y	47000	
40	$62,000	Y	80000	
60	$100,000	Y	42000	3
48	$220,000	Y	78000	
33	$150,000	Y	8000	1
48	**$142,000**	**?**		

Euclidean Distance

$$D = \sqrt{(x_1 - y_1)^2 + (x_2 - y_2)^2}$$

With K=3, there are two Default=Y and one Default=N out of three closest neighbors. The prediction for the unknown case is again Default=Y.

1.2 Locally-weighted Regression

Basic idea: k-NN forms local approximation to f for each query point x_q
Why not form an explicit approximation f (x) for region surrounding x_q

- Fit linear function to k nearest neighbors
- Fit quadratic, ...
- Thus producing ``**piecewise approximation**" to f

1.2 Locally-weighted Regression

- **Let us consider the case of locally weighted regression in which the target function f is approximated near x_q, using a linear function of the form**

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

As before, $a_i(x)$ denotes the value of the i^{th} attribute of the instance x.

- **There are three error criterion $E(x_q)$ to emphasize fitting the local training examples.**
- **Three possible criteria are given below :**

1. Minimize the squared error over just the k nearest neighbors:

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

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

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

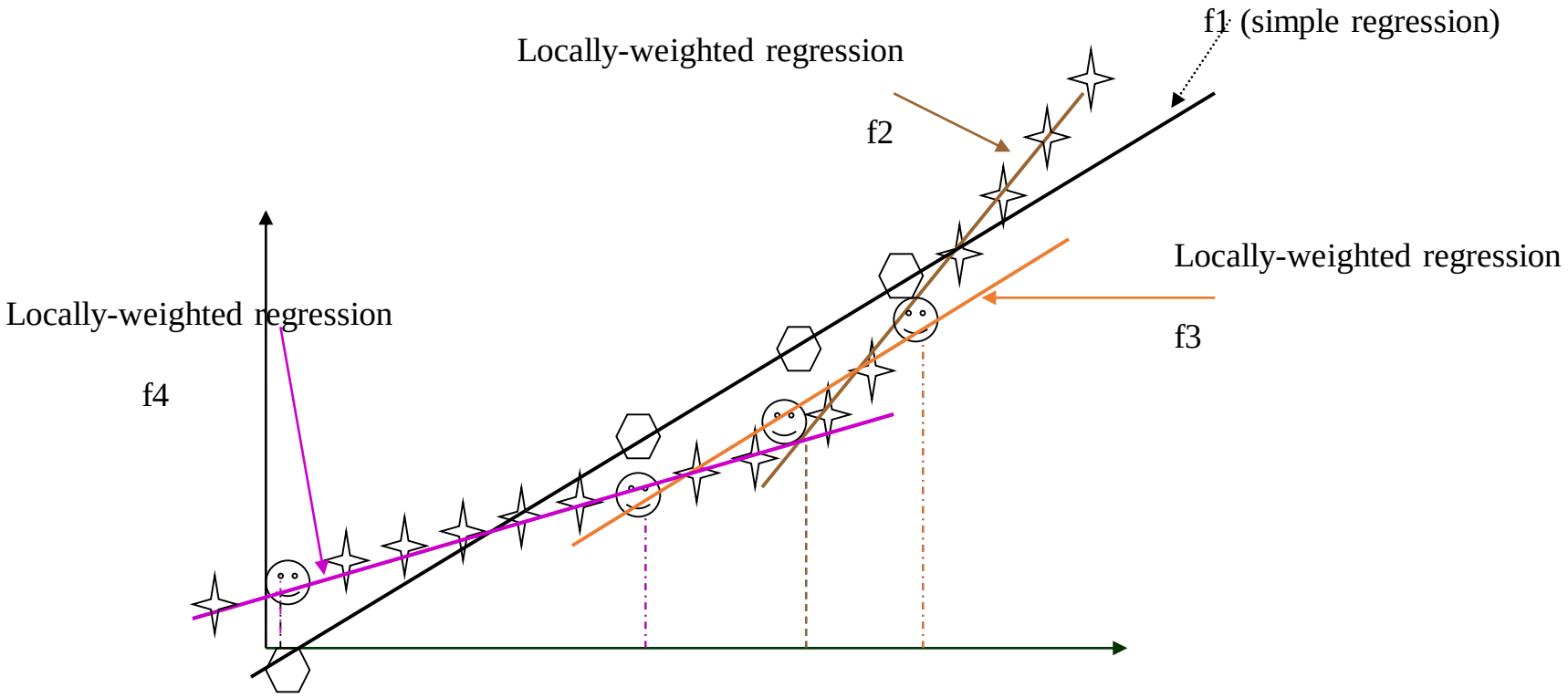
3. Combine 1 and 2:

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

If we choose criterion three above and rederive the gradient descent rule using the same style of argument as in Chapter 4, we obtain the following training rule (see Exercise 8.1):

$$\Delta w_j = \eta \sum_{x \in k \text{ nearest nbrs of } x_q} K(d(x_q, x)) \; (f(x) - \hat{f}(x)) \; a_j(x) \qquad (8.7)$$

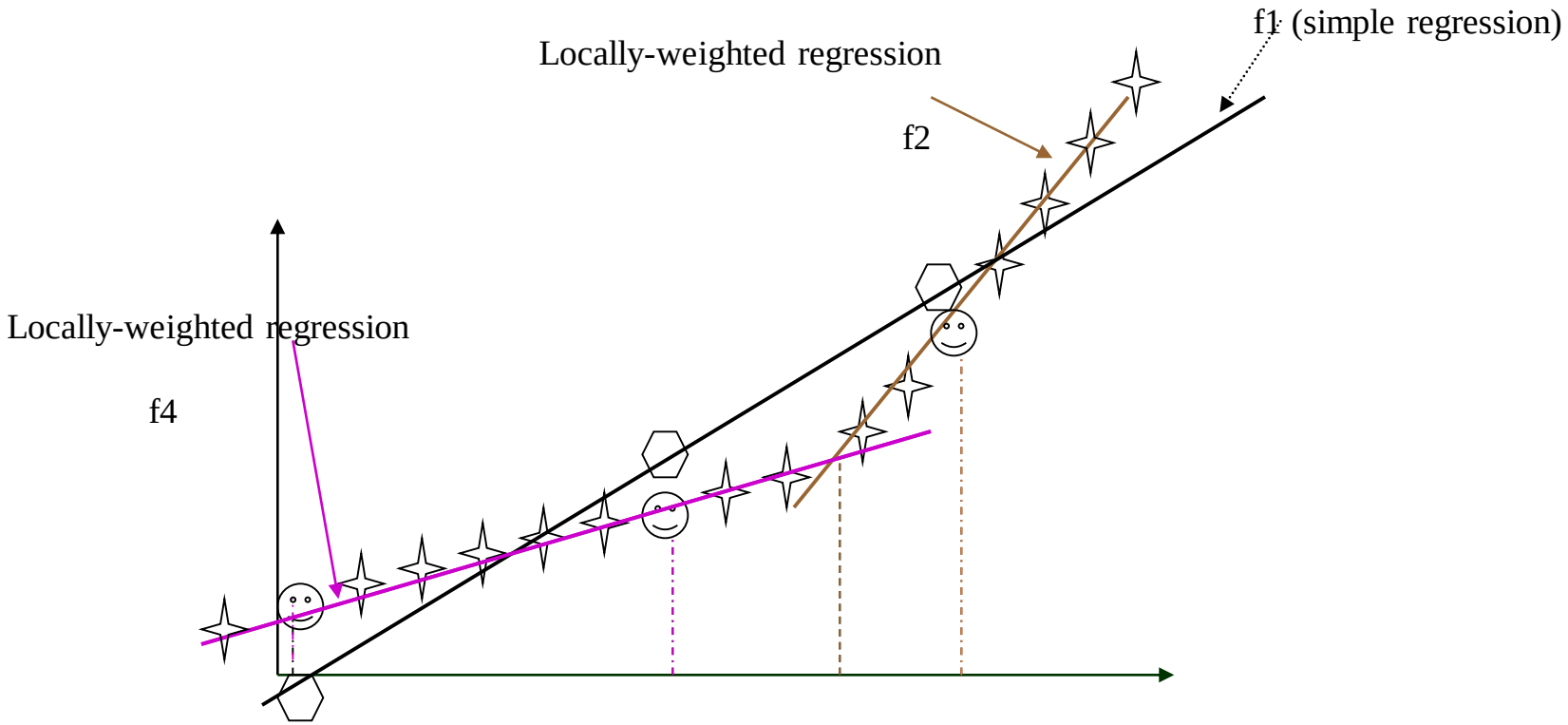
Notice the only differences between this new rule and the rule given by Equation (8.6) are that the contribution of instance x to the weight update is now multiplied by the distance penalty $K(d(x_q, x))$, and that the error is summed over only the k nearest training examples. In fact, if we are fitting a linear function to a fixed set of training examples, then methods much more efficient than gradient descent are available to directly solve for the desired coefficients $w_0 \ldots w_n$. Atkeson et al. (1997a) and Bishop (1995) survey several such methods.



Training data

Predicted value using simple regression

Predicted value using locally weighted (piece-wise) regression



Several choices of error to minimize:

e.g Squared error over k nearest neighbors

or Distance-weighted square error over all neighbors

or …..

Remarks on Locally Weighted Regression

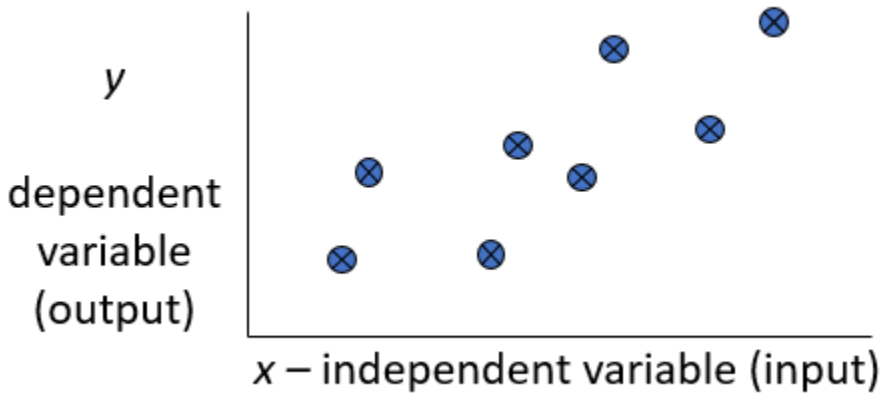
- The literature on locally weighted regression contains a broad range of alternative methods for distance weighting the training examples, and a range of methods for locally approximating the target function.
- In most cases, the target function is approximated by a constant, linear, or quadratic function.
- More complex functional forms are not often found because
  - (1) the cost of fitting more complex functions for each query instance is prohibitively high, and
  - (2) these simple approximations model the target function quite well over a sufficiently small subregion of the instance space.

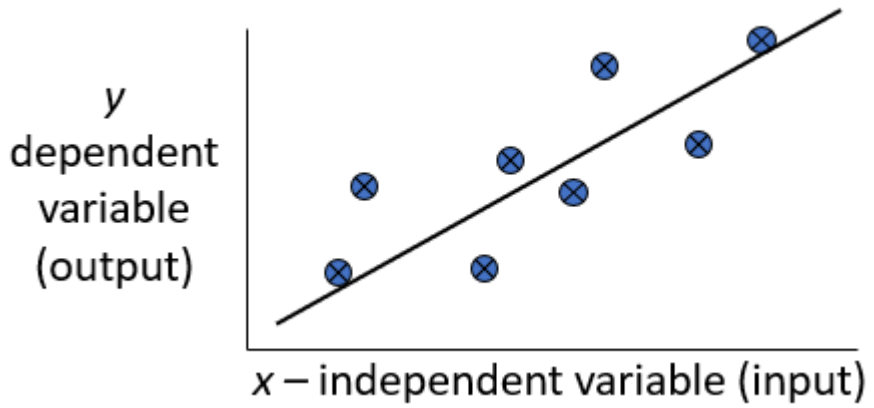
Lab Program 8

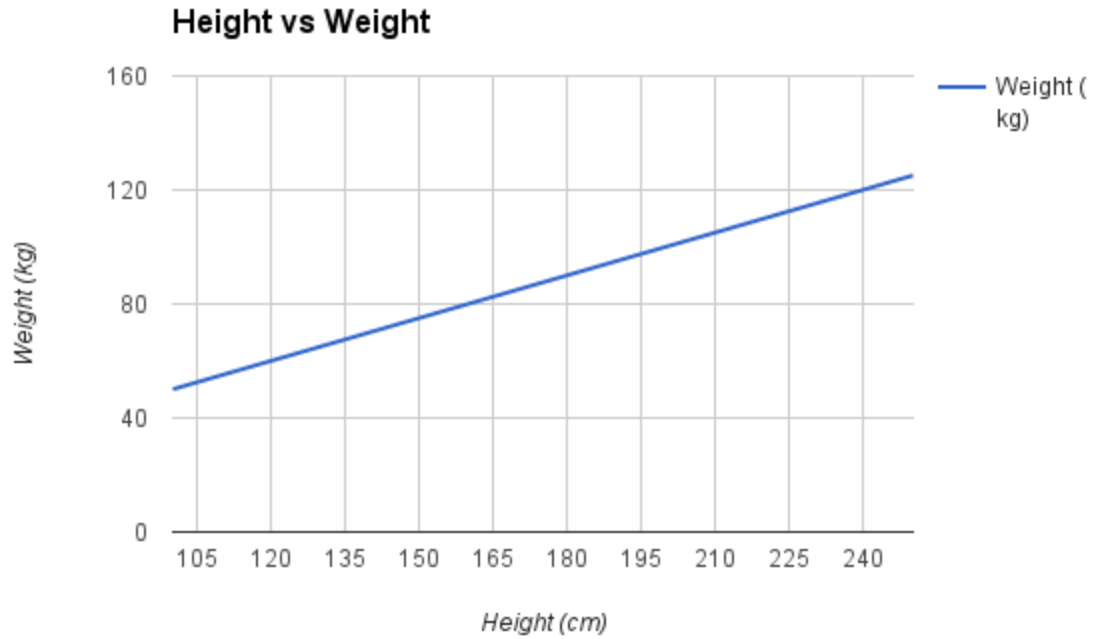
- **Implement the non-parametric Locally Weighted Regression (LOWESS) algorithm in order to fit data points. Select appropriate data set for your experiment and draw graphs.**

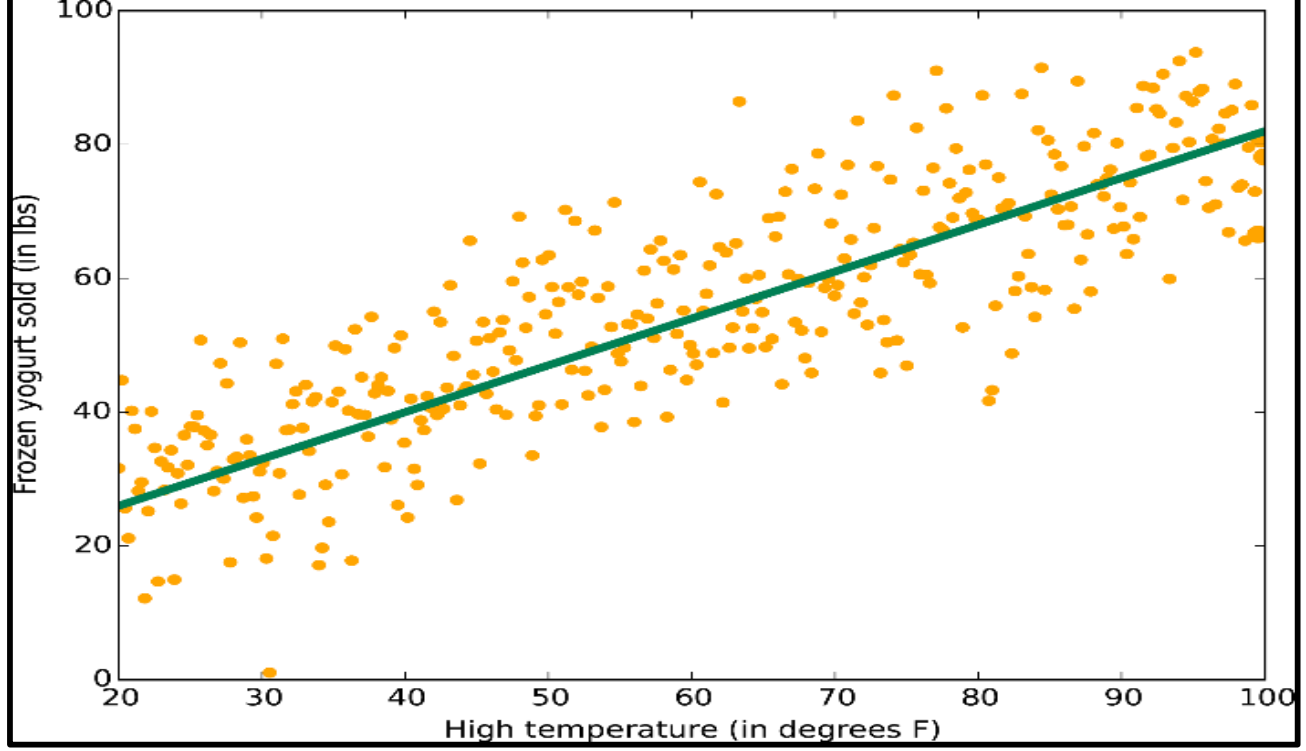
Regression

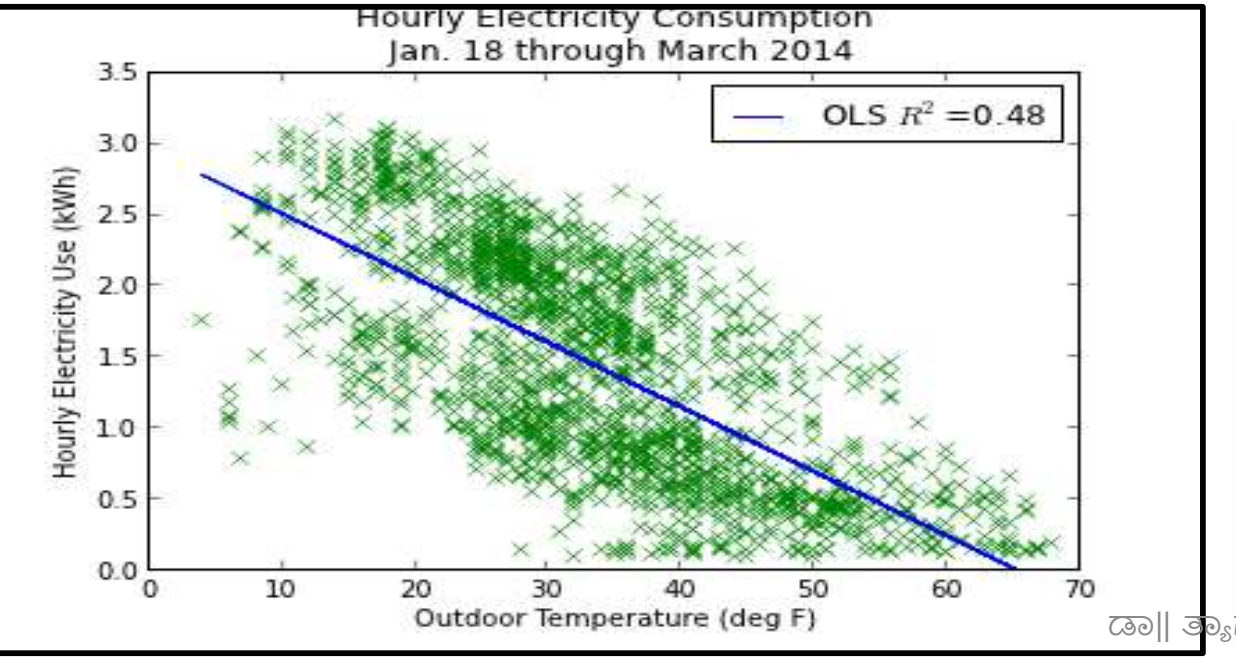
- Regression is a technique from statistics that is used to predict values of a desired target quantity when the target quantity is continuous .
- In regression, we seek to identify (or estimate) a continuous variable y associated with a given input vector x.
  - y is called the dependent variable.
  - x is called the independent variable.

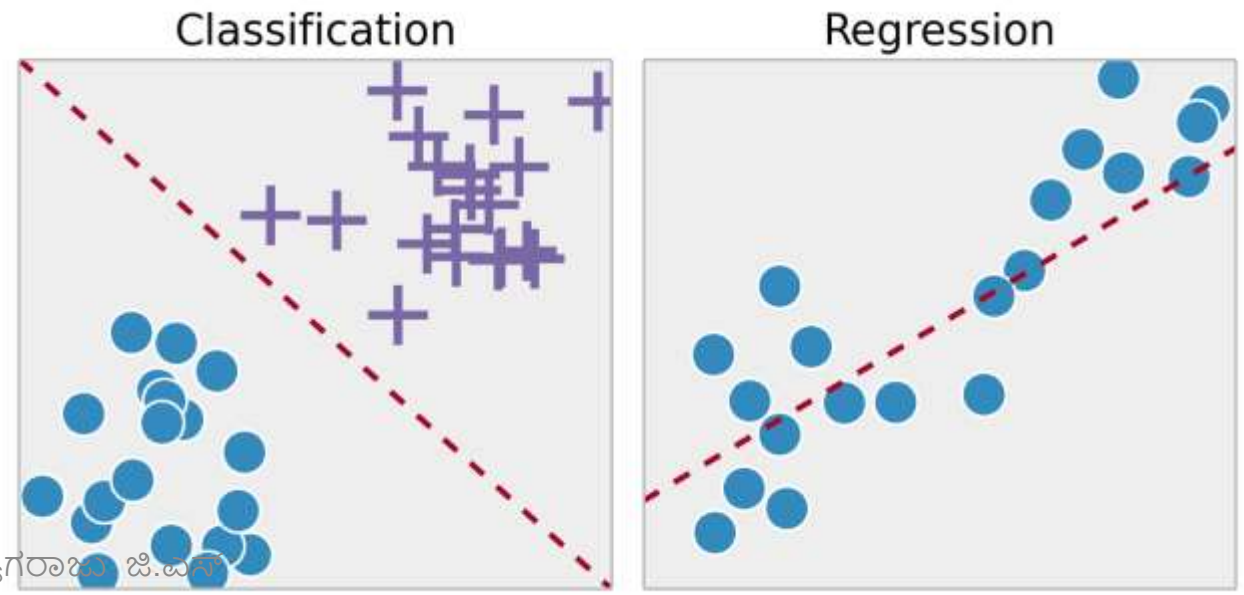




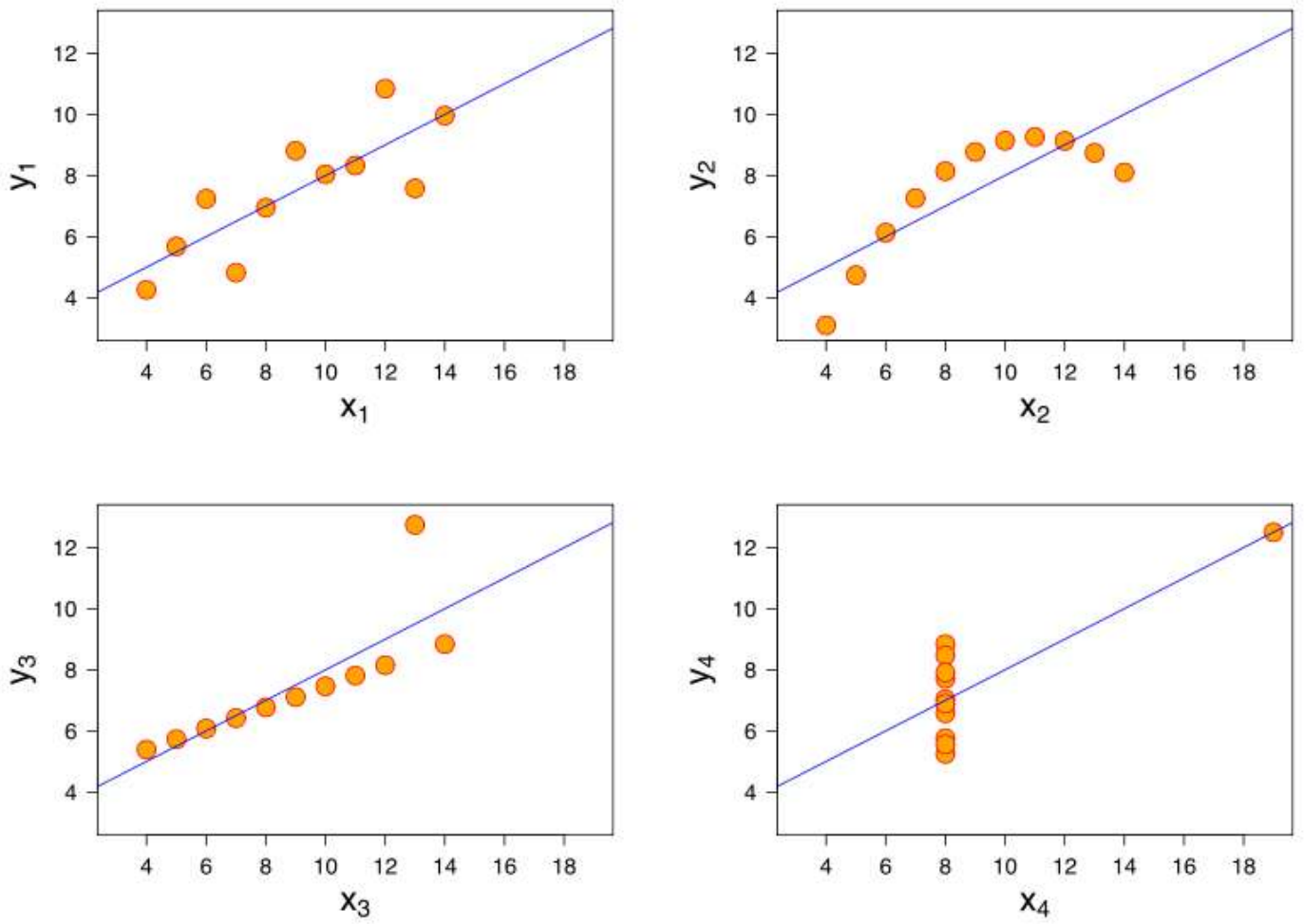

Height vs Weight
160
120
80
40
0
105 120 135 150 165 180 195 210 225 240
Weight (kg)
Height (cm)
Weight (kg)


100
80
60
40
20
0
20 30 40 50 60 70 80 90 100
Frozen yogurt sold (in lbs)
High temperature (in degrees F)


Hourly Electricity Consumption
Jan. 18 through March 2014
OLS R^2 =0.48
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
0 10 20 30 40 50 60 70
Hourly Electricity Use (kWh)
Outdoor Temperature (deg F)

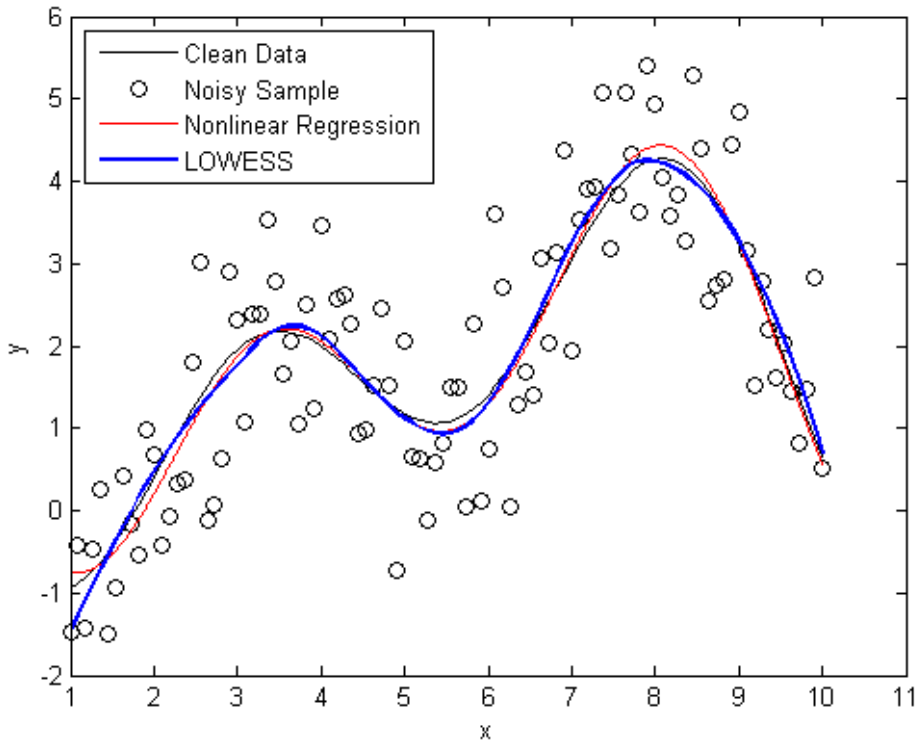

Classification
Regression

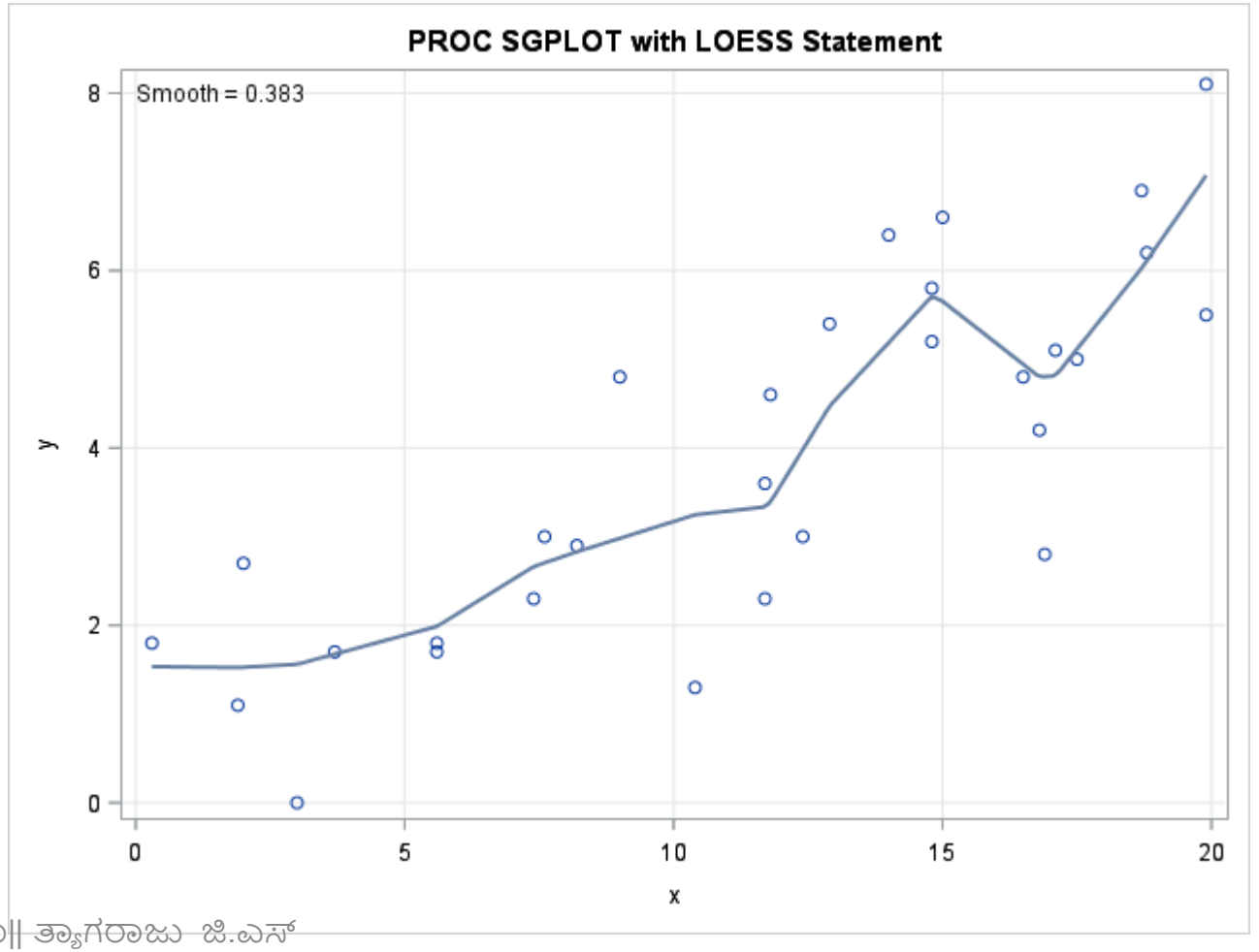
What lines "*really*" best fit each case?



Loess/Lowess Regression

- Loess regression is a nonparametric technique that uses ***local weighted*** regression to fit a **smooth curve** through points in a scatter plot.





Lowess Algorithm

- Locally weighted regression is a very powerful non-parametric model used in statistical learning .Given a *dataset* **X**, **y**, we attempt to find a *model* parameter β**(x)** that minimizes *residual sum of* ***weighted*** *squared errors*. The weights are given by a *kernel function(k or w)* which can be chosen arbitrarily .

Algorithm

1. Read the Given data Sample to **X** and the curve (linear or non linear) to **Y**
2. Set the value for Smoothening parameter or Free parameter say **τ**
3. Set the bias /Point of interest set **X0** which is a subset of **X**
4. Determine the weight matrix using :

$$w(x, x_o) = e^{-\frac{(x-x_o)^2}{2\tau^2}}$$

5. Determine the value of model term parameter β using :

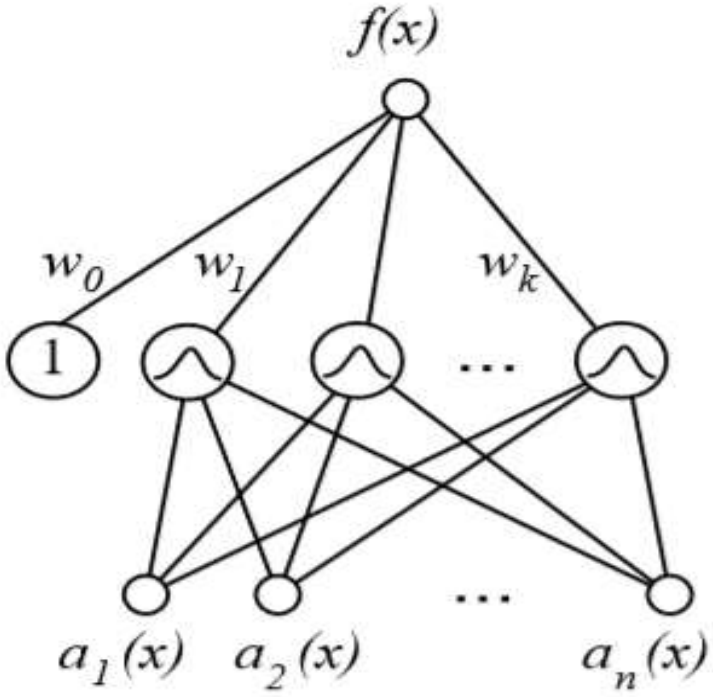
$$\hat{\beta}(x_o) = (X^T W X)^{-1} X^T W y$$

6. Prediction = $X_0 * \beta$

1.3 Radial basis Function Networks

- One approach to function approximation that is closely related to distance *weighted regression and also to artificial neural network its learning with radial basis functions .It is eager instead of lazy*
- *Global approximation to target function in terms of linear combination of local approximations*
- *Used e.g. for image classification*
- In this approach the learned hypothesis is a function of the form

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



where $a_i(x)$ are the attributes describing instance x, and

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

One common choice for $K_u(d(x_u, x))$ is

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

Training of RBF network

- Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process.
- First, the number k of hidden units is determined and each hidden unit u is defined by choosing the values of x_u and σ_u^2: that define its kernel function K,(d(x,, x)).
- Second, the weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by Equation

$$E \equiv \frac{1}{2} \sum_{x \in D} (f(x) - \hat{f}(x))^2$$

- Because the kernel functions are held fixed during this second stage, the linear weight values w, can be trained very efficiently.

1.4 Case Based Reasoning

- Instance-based methods such as k-NEAREST NEIGHBOUR and locally weighted regression share three key properties.
  - First, they are lazy learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed.
  - Second, they classify new query instances by ana- lyzing similar instances while ignoring instances that are very different from the query.
  - Third, they represent instances as real-valued points in an n-dimensional Euclidean space
- **Case-based reasoning (CBR)** is a learning paradigm based on the first two of these principles, but not the third

Case Based Reasoning

- In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate.
- CBR has been applied to problems such as conceptual design of mechanical devices based on a *stored library of previous designs* (Sycara et al. 1992), *reasoning about new legal cases based on previous rulings* (Ashley 1990), and *solving planning* and *scheduling problems by reusing and combining portions* of previous solutions to similar problems (Veloso 1992).

Basic Steps of CBR

1. Identify the problem/case
2. Look for a similar , previously experienced case
3. Predict a solution, possibly different from past experiences
4. Evaluate the solution
5. Update the system with the results

Case Based Reasoning using Symbolic Logic Descriptions

Case-Based Reasoning is instance-based learning applied to instances with symbolic logic descriptions

```
((user-complaint error53-on-shutdown)
 (cpu-model PowerPC)
 (operating-system Windows)
 (network-connection PCIA)
 (memory 48meg)
 (installed-applications Excel Netscape VirusScan)
 (disk 1gig)
 (likely-cause ???))
```

Case Based Reasoning in CADET

- **CADET** is a **C**ase-based **D**esign **T**ool. **CADET** is a system that aids conceptual design of electro-mechanical devices and is based on the paradigm of Case-based Reasoning
- The CADET system (Sycara et al. 1992) employs case- based reasoning to assist in the conceptual design of ***simple mechanical devices such as water faucets.***

- It uses a library containing approximately ***75 previous designs and design fragments*** to suggest conceptual designs to meet the ***specifications of new design problems.***

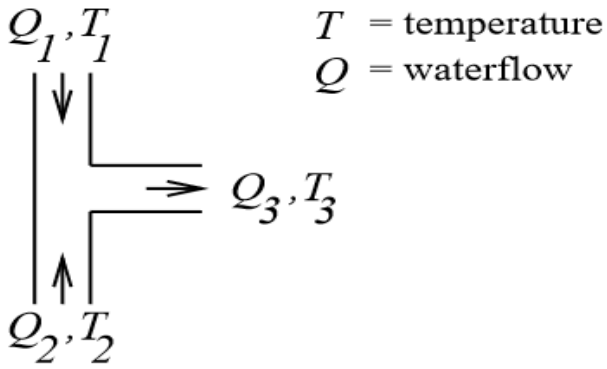
- Each instance stored in memory (e.g., a water pipe) is represented by describing both its structure and its qualitative function.

- New design problems are then presented by specifying the desired function and requesting the corresponding structure.

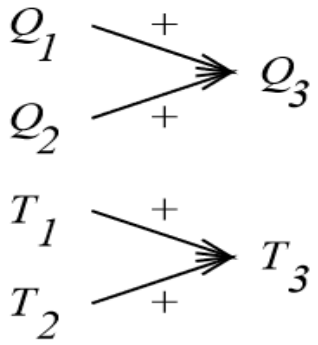
Case Based Reasoning in CADET

A stored case: T–junction pipe

Structure:



Function:

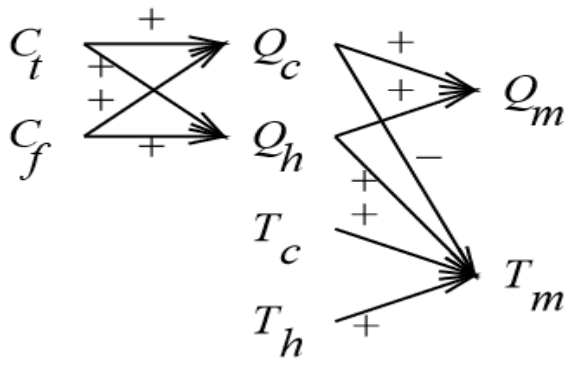


A problem specification: Water faucet

Structure:

?

Function:



- The top half of the figure shows the description of a typical stored case called a T-junction pipe. Its function is represented in terms of the qualitative relationships among the waterflow levels and temperatures at its inputs and outputs.
- In the functional description at its right, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail.
- For example, the output waterflow Q3 increases with increasing input waterflow Ql.
- Similarly a "-" label indicates that the variable at the head decreases with the variable at the tail.

- The bottom half of this figure depicts a new design problem described by its desired function. This particular function describes the required behavior of one type of water faucet.
- Here Qc, refers to the flow of cold water into the faucet, Qh to the input flow of hot water, and Qm, to the single mixed flow out of the faucet.
- Similarly, Tc, Th, and Tm , refer to the temperatures of the cold water, hot water, and mixed water respectively.
- The variable Ct, denotes the control signal for temperature that is input to the faucet, and Cf denotes the control signal for waterflow. Note the description of the desired function specifies that these controls Ct, and Cf are to influence the water flows Qc, and Qh, thereby indirectly influencing the faucet output flow Qm, and temperature Tm.

- Given this functional specification for the new design problem, CADET searches its library for stored cases whose functional descriptions match the design problem.
- If an exact match is found, indicating that some stored case implements exactly the desired function, then this case can be returned as a suggested solution to the design problem.
- If no exact match occurs, CADET may find cases that match various subgraphs of the desired functional specification.

Generic properties of case-based reasoning system

- Instances or cases may be represented by rich symbolic descriptions, such as the function graphs used in CADET. This may require a similarity metric different from Euclidean distance, such as the size of the largest shared subgraph between two function graphs.
-

Generic properties of case-based reasoning system

- Multiple retrieved cases may be combined to form the solution to the new problem.
- This is similar to the k-NEAREST NEIGHBOR approach, in that multiple similar cases are used to construct a response for the new query.
- However, the process for combining these multiple retrieved cases can be very different, relying on knowledge-based reasoning rather than statistical methods.
-

Generic properties of case-based reasoning system

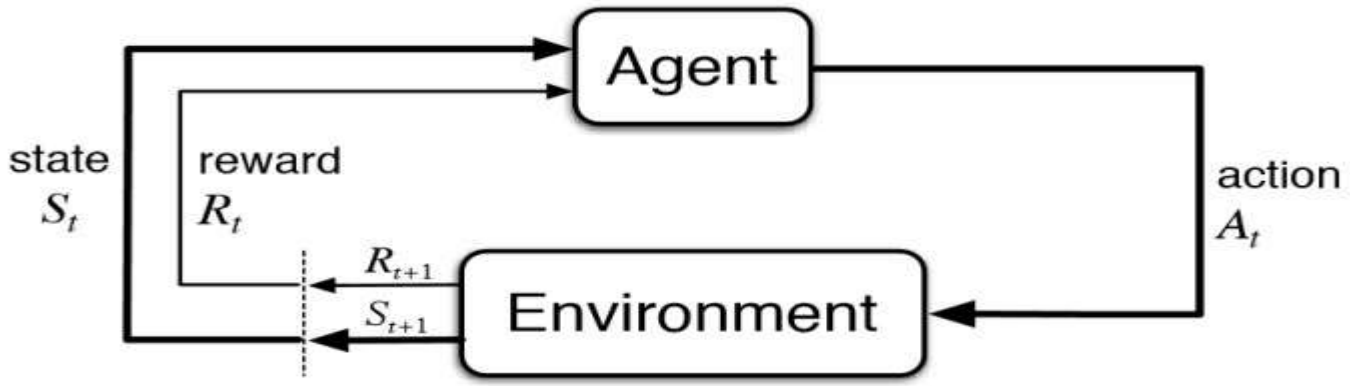
- There may be a tight coupling between case retrieval, knowledge-based reasoning, and problem solving.
- One simple example of this is found in CADET, which uses generic knowledge about influences to rewrite function graphs during its attempt to find matching cases.
- Other systems have been developed that more fully integrate case-based reasoning into general search- based problem-solving systems. Two examples are ANAPRON (Golding and Rosenbloom 199 1) and PRODIGY/ANALOGY (Veloso 1992).

Summary

- To summarize, case-based reasoning is an instance-based learning method in which instances (cases) may be rich relational descriptions and in which the retrieval and combination of cases to solve the current query may rely on knowledge- based reasoning and search-intensive problem-solving methods.
- One current re- search issue in case-based reasoning is to develop improved methods for indexing cases. The central issue here is that syntactic similarity measures (e.g., subgraph isomorphism between function graphs) provide only an approximate indication of the relevance of a particular case to a particular problem.

Reinforcement Learning

- User will get immediate feedback in supervised learning and no feedback from unsupervised learning . But in RL you will get delayed scalar feedback
- Figure below illustrates the concept of RL.

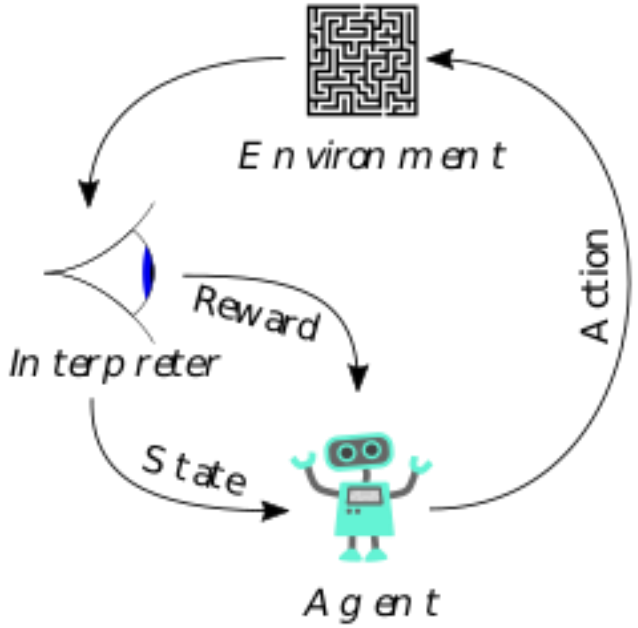


- Agent will perform actions , and based on the reward or feedback that it gets from the environment it improves its performance.

Elements of Reinforcement Learning

RL elements are as follows :

- a **policy**: is a mapping from perceived states of the environment to actions to be taken when in those states.
- a **reward function**: defines the goal in a reinforcement learning problem. iIt maps each perceived state (or state-action pair) of the environment to a single number, a reward, indicating the intrinsic desirability of that state.
- a **value function**: specifies what is good in the long run. The value of a state is the total amount of reward an agent can expect to accumulate over the future, starting from that state.
- a **model: This is something that mimics the behavior of the environment. For example, given a state and action, the model might predict the resultant next state and next reward. Models are used for planning, by which we mean any way of deciding on a course of action by considering possible future situations before they are actually experienced. The incorporation of models and planning into reinforcement learning systems is a relatively new development.**



3. Reinforced Learning

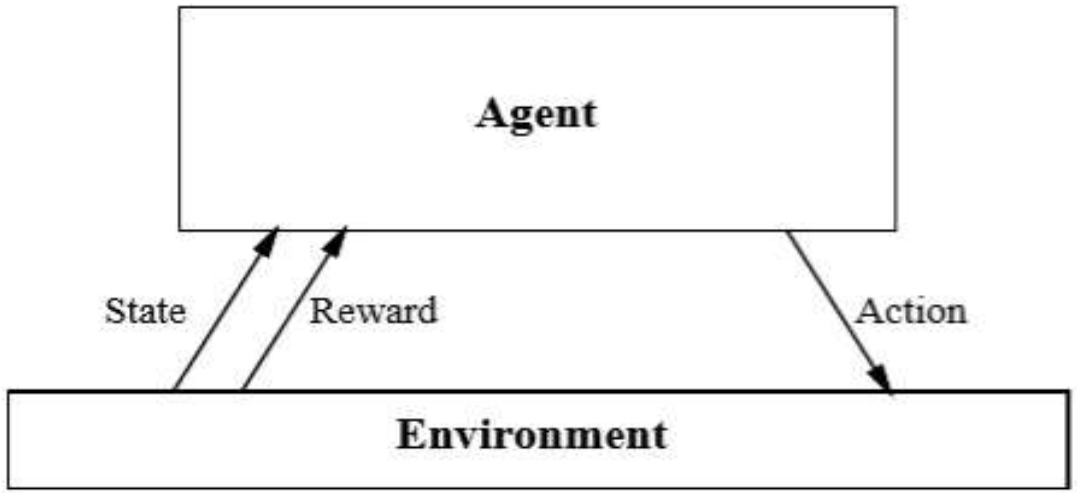
- Reinforcement learning addresses the question of how an autonomous agent that senses and acts in its environment can learn to choose optimal actions to achieve its goals.
- This very generic problem covers tasks such as learning to control a mobile robot, learning to optimize operations in factories, and learning to play board games. Each time the agent performs an action in its environment, a trainer may provide a reward or penalty to indicate the desirability of the resulting state.
  - *For example, when training an agent to play a game the trainer might provide a positive reward when the game is won, negative reward when it is lost, and zero reward in all other states. The task of the agent is to learn from this indirect, delayed reward, to choose sequences of actions that produce the greatest cumulative reward.*
- This chapter focuses on an algorithm called Q learning that can acquire optimal control strategies from delayed rewards, even when the agent has no prior knowledge of the effects of its actions on the environment. Reinforcement learning algorithms are related to dynamic programming algorithms frequently used to solve optimization problems.

3.1 Introduction :

Building a Learning Robot

- Consider building a learning robot. The robot, or agent, has a *set of sensors to observe the state of its environment, and a set of actions it can perform to alter this state.*
  - For example, a mobile robot may have sensors such as a *camera and sonars*, and actions such as "move forward" and "turn".
- Its task is to learn a control strategy, or policy, for choosing actions that achieve its goals.
  - For example, the robot may have a goal of docking onto its battery charger whenever its battery level is low.
- The problem of learning a control policy to maximize cumulative reward is very general and covers many problems beyond robot learning tasks. In general the problem is one of learning to control sequential processes.
- This includes, for example, manufacturing optimization problems in which a sequence of manufacturing actions must be chosen, and the reward to be maximized is the value of the goods produced minus the costs involved.

An agent interacting with its Environment



$$s_0 \xrightarrow[r_0]{a_0} s_1 \xrightarrow[r_1]{a_1} s_2 \xrightarrow[r_2]{a_2} \dots$$

Goal: Learn to choose actions that maximize

$$r_0 + \gamma r_1 + \gamma^2 r_2 + \dots \text{ , where } 0 \leq \gamma < 1$$

- An agent interacting with its environment. The agent exists in an environment described by some set of possible states S.
- It can perform any of a set of possible actions **A**. Each time it performs an action **a**, in some state $\mathbf{s}_t$ the agent receives a real-valued reward **r**, that indicates the immediate value of this state-action transition. This produces a sequence of states $\mathbf{s}_i$, actions $\mathbf{a}_i$, and immediate rewards $\mathbf{r}_i$ as shown in the figure.
- The agent's task is to learn a control policy, **π : S -> A,** that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay

The aspects which makes RL different from other

The **reinforcement learning problem** differs from other function approximation tasks in several important respects

- **Delayed reward.**
- **Exploration**
- **Partially observable states**
- **Life-long learning**

Delayed Reward

- The task of the agent is to learn a target function **π** that maps from the current state **s** to the optimal action **a = π**(s).
- Here training example is not of the form **(s, n(s)).**
- Instead, the trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions.
- The agent, therefore, faces the problem of temporal credit assignment: ***determining which of the actions in its sequence are to be credited with producing the eventual rewards***.

Exploration

- In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses.
- This raises the question of which experimentation strategy produces most effective learning.
- The learner faces a tradeoff in choosing whether to *favor exploration of unknown states and actions* (**to gather new information**), or *exploitation of states and actions that it has already learned will yield high reward* (**to maximize its cumulative reward**).

Partially observable states

- Although it is convenient to assume that the agent's sensors can perceive the entire state of the environment at each time step, in many practical situations sensors provide only partial information.
- For example, a robot with a forward-pointing camera cannot see what is behind it.
- In such cases, it may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.

Life-long learning

- Unlike isolated function approximation tasks, robot learning often requires that the robot learn several related tasks within the same environment, using the same sensors.
- For example, a mobile robot may need to learn how to dock on its battery charger, how to navigate through narrow corridors, and how to pick up output from laser printers.
- This setting raises the possibility of using previously obtained experience or knowledge to reduce sample complexity when learning new tasks.

3.2 THE LEARNING TASK

- In this section we formulate the problem of learning sequential control strategies more precisely. Note there are many ways to do so.
- For example, we might assume the agent's actions are deterministic or that they are nondeterministic.
- We might assume that the agent can predict the next state that will result from each action, or that it cannot.
- We might assume that the agent is trained by an expert who shows it examples of optimal action sequences, or that it must train itself by performing actions of its own choice.
- Here we define one quite general formulation of the problem, based on Markov decision processes.

Markov Decision Process

- In a Markov decision process (MDP) the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
- At each discrete time step t, the agent senses the current state s_t, chooses a current action a_t, and performs it.
- The environment responds by giving the agent a reward $r_t = r(s_t, a)$ and by producing the succeeding state $s_{t+1} = \delta(s_t, a)$. Here the functions δ and r are part of the environment and are not necessarily known to the agent.
- In an MDP, the functions $\delta(s_t, a,)$ and $r(s_t, a)$ depend only on the current state and action, and not on earlier states or actions.
- Here we consider only the case in which S and A are finite. In general, δ and r may be nondeterministic functions, but we begin by considering only the *deterministic case*

Agent's Learning Task

Execute actions in environment, observe results, and

- learn action policy $\pi : S \rightarrow A$ that maximizes

$$E[r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots]$$

  from any starting state in S
- here $0 \leq \gamma < 1$ is the discount factor for future rewards

Note something new:

- Target function is $\pi : S \rightarrow A$
- but we have no training examples of form $\langle s, a \rangle$
- training examples are of form $\langle \langle s, a \rangle, r \rangle$

Agents Learning Task (Value Function)

- The task of the agent is to learn a policy, **π : S -> A**, for selecting its next action **a**, based on the current observed state s_t; that is, **π(**s_t**) = a,**
- How shall we specify precisely which policy **π** we would like the agent to learn?
- One obvious approach is to require the policy that produces the greatest possible cumulative reward for the robot over time.
- To state this requirement more precisely, we define the cumulative value
- V^{π} (s_t) achieved by following an arbitrary policy **n** from an arbitrary initial state s_t as follows:

$$V^{\pi}(s_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \dots$$

$$\equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

Value Function

To begin, consider deterministic worlds...

For each possible policy π the agent might adopt, we can define an evaluation function over states

$$V^{\pi}(s) \equiv r_t + \gamma r_{t+1} + \gamma^2 r_{t+2} + \ldots \equiv \sum_{i=0}^{\infty} \gamma^i r_{t+i}$$

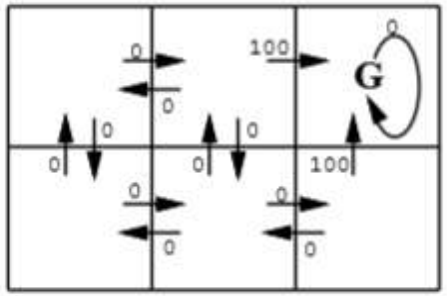
where $r_t, r_{t+1}, \ldots$ are generated by following policy π starting at state s

Restated, the task is to learn the optimal policy π^*

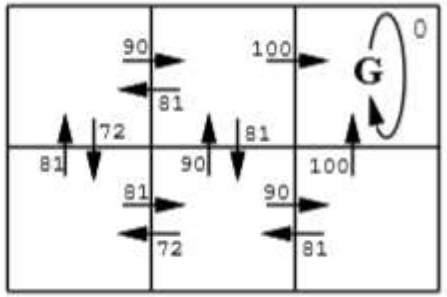
$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s)$$

The quantity $V^{\pi}(s_t)$ is often called the *discounted cumulative reward achieved by policy* **π.**

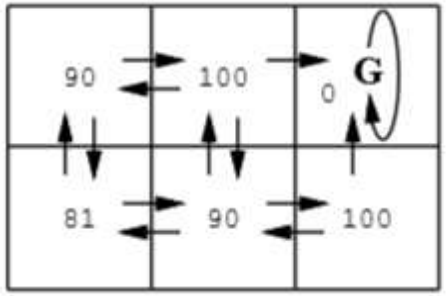
- The six grid squares in this diagram represent ***six possible states, or locations, for the agent***.
- Each arrow in the diagram represents a ***possible action the agent can take to move from one state to another***.
- The number associated *with each arrow represents the immediate reward* ***r(s,a)*** *the agent receives* if it executes the corresponding state-action transition.
- Note the immediate reward in this particular environment is defined to be zero for all state-action transitions except for those leading into **the state labeled G.**
- It is convenient to think ***of the state G as the goal state,*** because the only way the agent can receive reward, in this case, ***is by entering this state.***
- Note in this particular environment, the only action available to the agent once it enters the ***state G is to remain in this state***.
- For this reason, we call **G** an absorbing state.



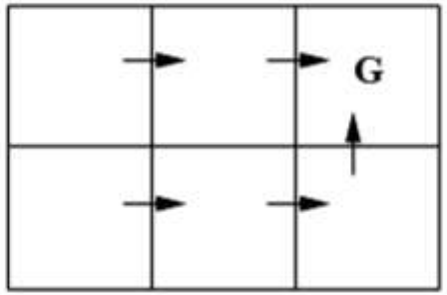
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values

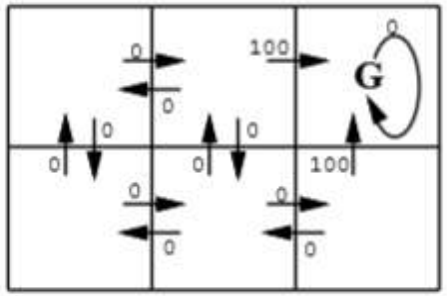


$V^*(s)$ values

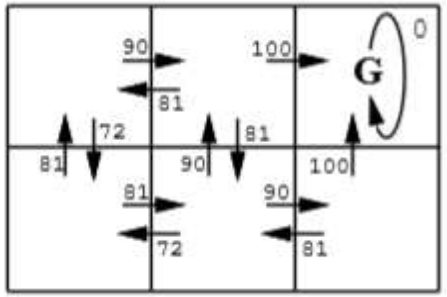


One optimal policy

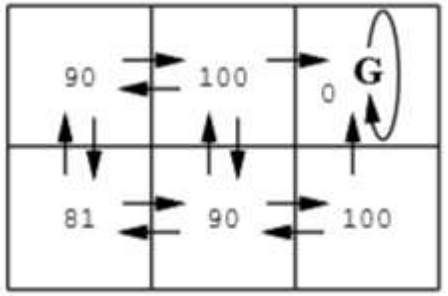
- A simple deterministic world to illustrate the basic concepts of ***Q-learning***. Each grid square represents a distinct state, each arrow a distinct action. The immediate reward function, **r(s,a**) gives reward 100 for actions entering the goal state G, and zero otherwise.
- Values of $V^{\pi}(s)$ and Q(s, a) follow from r(s, a), and the discount factor Y=0.9. An optimal policy, corresponding to actions with maximal Q values, is also shown.



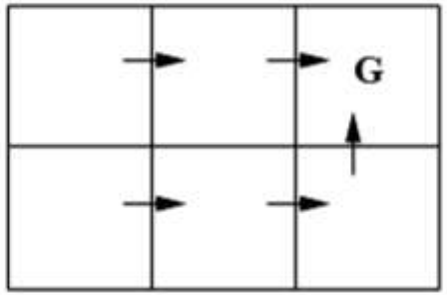
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values

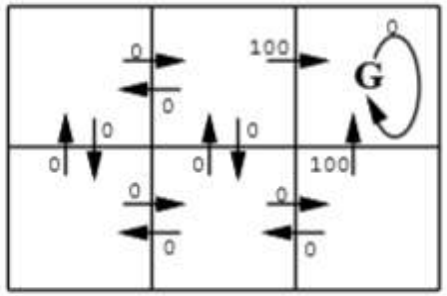


$V^*(s)$ values

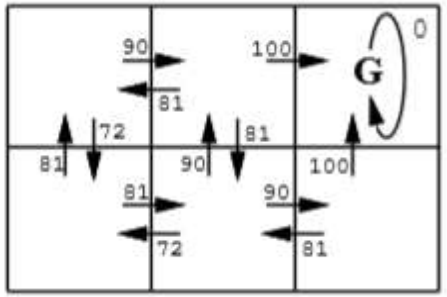


One optimal policy

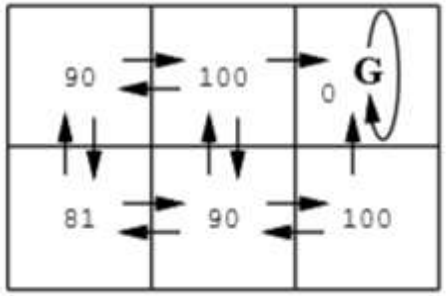
- The diagram at the right of Figure shows the values of V* for each state.
- For example, consider the bottom right state in this diagram. The value of V* for this state is 100 because the optimal policy in this state selects the "*move up*" action that receives immediate reward 100. Thereafter, the agent will remain in the absorbing state and receive no further rewards. Similarly, the value of V* for the bottom center state is 90. This is because the optimal policy will move the agent from this state to the right (generating an immediate reward of zero), then upward (generating an immediate reward of 100). Thus, the discounted future reward from the bottom center state is
- $0 + Y\ 100 + Y^2 0 + Y^3\ 0 + \ldots = 90$



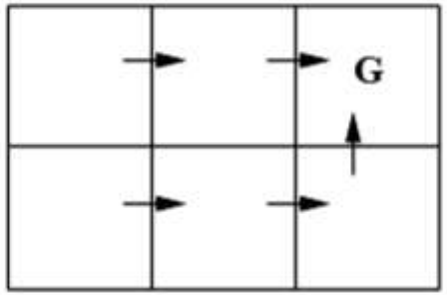
$r(s, a)$ (immediate reward) values



$Q(s, a)$ values



$V^*(s)$ values



One optimal policy

3.3 Q Learning

- Q Learning is a Reinforcement learning technique used in machine learning . The goal of Q learning is to learn a policy which tells an agent which action to take under which circumstances. It does not require a model of the environment and can handle problems with stochastic transitions and rewards, without requiring adaptations.
- In Q –learning an agent tries to learn the optimal policy from its history of interaction with the environment
- *Q*-learning finds a policy that is optimal in the sense that it maximizes the expected value of the total reward over all successive steps, starting from the current state.
- When an agent take action a_t in state s_t at time t, the predicted future rewards is defined as $Q(s_t,a_t)$

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

Q LEARNING

We might try to have agent learn the evaluation function V^{π^*} (which we write as V^*)

It could then do a lookahead search to choose best action from any state s because

$$\pi^*(s) = \operatorname{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

A problem:

- This works well if agent knows $\delta : S \times A \rightarrow S$, and $r : S \times A \rightarrow \Re$
- But when it doesn't, it can't choose actions this way

3.3.1 Q Function

Define new function very similar to V^*

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a))$$

If agent learns Q, it can choose optimal action even without knowing δ!

$$\pi^*(s) = \operatorname*{argmax}_a [r(s,a) + \gamma V^*(\delta(s,a))]$$

$$\pi^*(s) = \operatorname*{argmax}_a Q(s,a)$$

Q is the evaluation function the agent will learn

3.3.2 An Algorithm for Learning Q

Q learning algorithm

For each s, a initialize the table entry $\hat{Q}(s, a)$ to zero.
Observe the current state s
Do forever:

- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s, a)$ as follows:

$$\hat{Q}(s, a) \leftarrow r + \gamma \max_{a'} \hat{Q}(s', a')$$

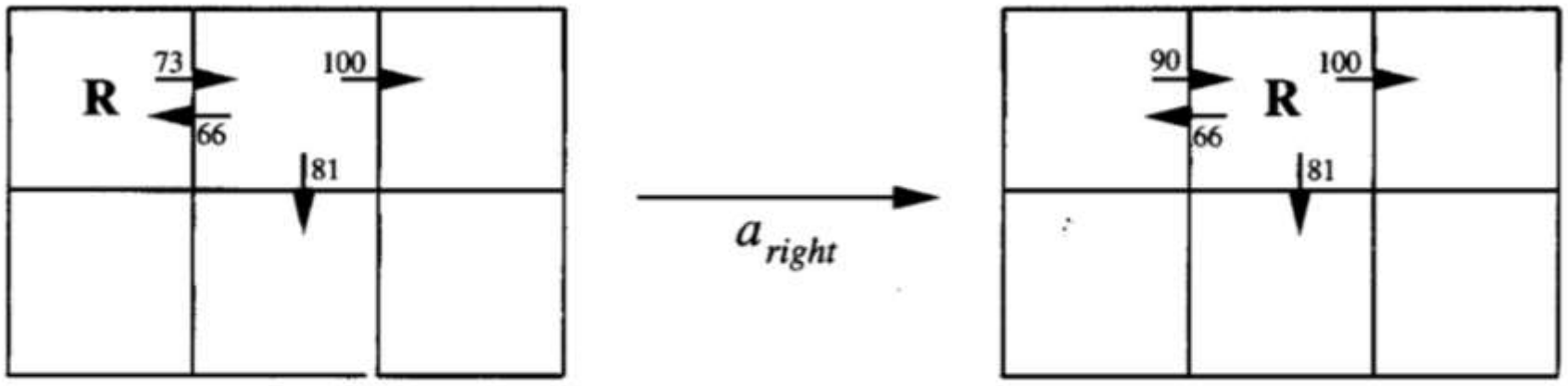
- $s \leftarrow s'$

TABLE 13.1
Q learning algorithm, assuming deterministic rewards and actions. The discount factor γ may be any constant such that $0 \leq \gamma < 1$.

3.3.3 : An Illustrative Example

- To illustrate the operation of the Q learning algorithm, consider a single action taken by an agent, and the corresponding refinement to $\hat{Q}$
- In this example, the agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition
- It then applies the training rule of to refine its estimate $\hat{Q}$ for the state-action transition it just executed.
- Training Rule :

$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$
$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$
$$\leftarrow 90$$



$$\hat{Q}(s_1, a_{right}) \leftarrow r + \gamma \max_{a'} \hat{Q}(s_2, a')$$

$$\leftarrow 0 + 0.9 \max\{66, 81, 100\}$$

$$\leftarrow 90$$

FIGURE 13.3
The update to $\hat{Q}$ after executing a single action. The diagram on the left shows the initial state s_1 of the robot (**R**) and several relevant $\hat{Q}$ values in its initial hypothesis. For example, the value $\hat{Q}(s_1, a_{right}) = 72.9$, where a_{right} refers to the action that moves **R** to its right. When the robot executes the action a_{right}, it receives immediate reward $r = 0$ and transitions to state s_2. It then updates its estimate $\hat{Q}(s_1, a_{right})$ based on its $\hat{Q}$ estimates for the new state s_2. Here $\gamma = 0.9$.

3.3.4 Convergence

Theorem 13.1. Convergence of Q learning for deterministic Markov decision processes. Consider a Q learning agent in a deterministic MDP with bounded rewards $(\forall s, a)|r(s, a)| \leq c$. The Q learning agent uses the training rule of Equation (13.7), initializes its table $\hat{Q}(s, a)$ to arbitrary finite values, and uses a discount factor γ such that $0 \leq \gamma < 1$. Let $\hat{Q}_n(s, a)$ denote the agent's hypothesis $\hat{Q}(s, a)$ following the nth update. If each state-action pair is visited infinitely often, then $\hat{Q}_n(s, a)$ converges to $Q(s, a)$ as $n \rightarrow \infty$, for all s, a.

Proof. Since each state-action transition occurs infinitely often, consider consecutive intervals during which each state-action transition occurs at least once. The proof consists of showing that the maximum error over all entries in the $\hat{Q}$ table is reduced by at least a factor of γ during each such interval. $\hat{Q}_n$ is the agent's table of estimated Q values after n updates. Let Δ_n be the maximum error in $\hat{Q}_n$; that is

$$\Delta_n \equiv \max_{s,a} |\hat{Q}_n(s, a) - Q(s, a)|$$

Below we use s' to denote $\delta(s, a)$. Now for any table entry $\hat{Q}_n(s, a)$ that is updated on iteration $n + 1$, the magnitude of the error in the revised estimate $\hat{Q}_{n+1}(s, a)$ is

$$\begin{aligned}
|\hat{Q}_{n+1}(s, a) - Q(s, a)| &= |(r + \gamma \max_{a'} \hat{Q}_n(s', a')) - (r + \gamma \max_{a'} Q(s', a'))| \\
&= \gamma |\max_{a'} \hat{Q}_n(s', a') - \max_{a'} Q(s', a')| \\
&\leq \gamma \max_{a'} |\hat{Q}_n(s', a') - Q(s', a')| \\
&\leq \gamma \max_{s'', a'} |\hat{Q}_n(s'', a') - Q(s'', a')| \\
|\hat{Q}_{n+1}(s, a) - Q(s, a)| &\leq \gamma \Delta_n
\end{aligned}$$

The third line above follows from the second line because for any two functions f_1 and f_2 the following inequality holds

$$|\max_a f_1(a) - \max_a f_2(a)| \leq \max_a |f_1(a) - f_2(a)|$$

In going from the third line to the fourth line above, note we introduce a new variable s'' over which the maximization is performed. This is legitimate because the maximum value will be at least as great when we allow this additional variable to vary. Note that by introducing this variable we obtain an expression that matches the definition of Δ_n.

Thus, the updated $\hat{Q}_{n+1}(s, a)$ for any s, a is at most γ times the maximum error in the $\hat{Q}_n$ table, Δ_n. The largest error in the initial table, Δ_0, is bounded because values of $\hat{Q}_0(s, a)$ and $Q(s, a)$ are bounded for all s, a. Now after the first interval

during which each s, a is visited, the largest error in the table will be at most $\gamma\Delta_0$. After k such intervals, the error will be at most $\gamma^k\Delta_0$. Since each state is visited infinitely often, the number of such intervals is infinite, and $\Delta_n \to 0$ as $n \to \infty$. This proves the theorem. □

3.3.5 Nondeterministic Case(Cont')

Q learning generalizes to nondeterministic worlds

Alter training rule to

$$\hat{Q}_n(s,a) \leftarrow (1-\alpha_n)\hat{Q}_{n-1}(s,a)+\alpha_n[r+\max_{a'}\hat{Q}_{n-1}(s',a')]$$

where

$$\alpha_n = \frac{1}{1+visits_n(s,a)}$$

Can still prove convergence of $\hat{Q}$ to Q [Watkins and Dayan, 1992]

3.3.6 Temporal Difference Learning

Q learning: reduce discrepancy between successive Q estimates

One step time difference:

$$Q^{(1)}(s_t, a_t) \equiv r_t + \gamma \max_a \hat{Q}(s_{t+1}, a)$$

Why not two steps?

$$Q^{(2)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \gamma^2 \max_a \hat{Q}(s_{t+2}, a)$$

Or n?

$$Q^{(n)}(s_t, a_t) \equiv r_t + \gamma r_{t+1} + \cdots + \gamma^{(n-1)} r_{t+n-1} + \gamma^n \max_a \hat{Q}(s_{t+n}, a)$$

Blend all of these:

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) \right.$$

Temporal Difference Learning(Cont')

$$Q^{\lambda}(s_t, a_t) \equiv (1-\lambda) \left[Q^{(1)}(s_t, a_t) + \lambda Q^{(2)}(s_t, a_t) + \lambda^2 Q^{(3)}(s_t, a_t) \right.$$

Equivalent expression:

$$Q^{\lambda}(s_t, a_t) = r_t + \gamma[(1-\lambda) \max_a \hat{Q}(s_t, a_t) + \lambda Q^{\lambda}(s_{t+1}, a_{t+1})]$$

TD(λ) algorithm uses above training rule

- Sometimes converges faster than Q learning
- converges for learning V^* for any $0 \leq \lambda \leq 1$ (Dayan, 1992)
- Tesauro's TD-Gammon uses this algorithm

3.3.7 Subtleties and Ongoing Research

- Replace $\hat{Q}$ table with neural net or other generalizer
- Handle case where state only partially observable
- Design optimal exploration strategies
- Extend to continuous action, state
- Learn and use $\hat{\delta} : S \times A \rightarrow S$
- Relationship to dynamic programming

List of Machine Learning Algorithms Discussed

1. Find S Algorithm
2. List then Elimination Algorithm
3. Candidate Elimination Algorithm
4. Rote Learner
5. The Basic Decision Tree Learning Algorithm (ID3)
6. Gradient Descent Algorithm
7. The Backpropagation Algorithm
8. Brute Force MAP Learning Algorithm
9. Gibbs Algorithm
10. Naïve Bayes Classifier for learning and classifying text
11. Bayesian Network Algorithm
12. The EM Algorithm
13. K- Means Algorithm
14. K-NN Algorithm
15. Non Parametric Locally Weighted Linear Regression Algorithm
16. Q – Learning assuming deterministic rewards and actions.

Notation Used

(a, b]: Brackets of the form [,], (, and) are used to represent intervals, where square brackets represent intervals including the boundary and round parentheses represent intervals excluding the boundary. For example, (1, 3] represents the interval $1 < x \leq 3$.

$\sum_{i=1}^{n} x_i$: The sum $x_1 + x_2 + \cdots + x_n$.

$\prod_{i=1}^{n} x_i$: The product $x_1 \cdot x_2 \cdots x_n$.

$\vdash$: The symbol for logical entailment. For example, $A \vdash B$ denotes that B follows deductively from A.

$>_g$: The symbol for the *more general than* relation. For example, $h_i >_g h_j$ denotes that hypothesis h_i is more general than h_j.

$\underset{x \in X}{\text{argmax}}\, f(x)$: The value of x that maximizes $f(x)$. For example,

$$\underset{x \in \{1,2,-3\}}{\text{argmax}}\, x^2 = -3$$

$\hat{f}(x)$: A function that approximates the function $f(x)$.

δ: In PAC-learning, a bound on the probability of failure. In artificial neural network learning, the error term associated with a single unit output.

ϵ: A bound on the error of a hypothesis (in PAC-learning).

η: The learning rate in neural network and related learning methods.

μ: The mean of a probability distribution.

σ: The standard deviation of a probability distribution.

$\nabla E(\vec{w})$: The gradient of E with respect to the vector $\vec{w}$.

C: Class of possible target functions.

D: The training data.

$\mathcal{D}$: A probability distribution over the instance space.

$E[x]$: The expected value of x.

$E(\vec{w})$: The sum of squared errors of an artifial neural network whose weights are given by the vector $\vec{w}$.

$Error$: The error in a discrete-valued hypothesis or prediction.

H: Hypothesis space.

$h(x)$: The prediction produced by hypothesis h for instance x.

$P(x)$: The probability (mass) of x.

$\Pr(x)$: The probability (mass) of the event x.

$p(x)$: The probability density of x.

$Q(s, a)$: The Q function from reinforcement learning.

$\Re$: The set of real numbers.

$VC(H)$: The Vapnik-Chervonenkis dimension of the hypothesis space H.

$VS_{H,D}$: The Version Space; that is, the set of hypotheses from H that are consistent with D.

w_{ji}: In artificial neural networks, the weight from node i to node j.

X: Instance space.

Greek Alphabets

NAME	UPPER CASE	LOWER CASE
Alpha	Α	α
Beta	Β	β
Gamma	Γ	γ
Delta	Δ	δ
Epsilon	Ε	ε
Zeta	Ζ	ζ
Eta	Η	η
Theta	Θ	θ
Iota	Ι	ι
Kappa	Κ	κ
Lambda	Λ	λ
Mu	Μ	μ

NAME	UPPER CASE	LOWER CASE
Nu	Ν	ν
Xi	Ξ	ξ
Omicron	Ο	ο
Pi	Π	π
Rho	Ρ	ρ
Sigma	Σ	σ
Tau	Τ	τ
Upsilon	Υ	υ
Phi	Φ	φ
Chi	Χ	χ
Psi	Ψ	ψ
Omega	Ω	ω

End of Module5