MODULE 5: EVALUATING HYPOTHESIS, INSTANCE BASED LEARNING, REINFORCEMENT LEARNING

References
1. Tom M. Mitchell, Machine Learning, India Edition 2013, McGraw Hill Education.

5.1 MOTIVATION

In many cases it is important to evaluate the performance of learned hypotheses as precisely as possible. One reason is simply to understand whether to use the hypothesis. For instance, when learning from a limited-size database indicating the effectiveness of different medical treatments, it is important to understand as precisely as possible the accuracy of the learned hypotheses. A second reason is that evaluating hypotheses is an integral component of many learning methods. For example, in post-pruning decision trees to avoid overfitting, we must evaluate the impact of possible pruning steps on the accuracy of the resulting decision tree.

Therefore it is important to understand the likely errors inherent in estimating the accuracy of the pruned and unpruned tree.

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:

- ***Bias in the estimate.*** First, the observed accuracy of the learned hypothesis over the training examples is often a poor estimator of its accuracy over future examples. Because the learned hypothesis was derived from these examples, they will typically provide an optimistically biased estimate of hypothesis accuracy over future examples. This is especially likely when the learner considers a very rich hypothesis space, enabling it to overfit the training examples. To obtain an unbiased estimate of future accuracy, we typically test the hypothesis on some set of test examples chosen independently of the training examples and the hypothesis.
- ***Variance in the estimate.*** Second, even if the hypothesis accuracy is measured over an unbiased set of test examples independent of the training examples, the measured accuracy can still vary from the true accuracy, depending on the makeup of the particular set of test examples. The smaller the set of test examples, the greater the expected variance.

5.2 ESTIMATING HYPOTHESIS ACCURACY

When evaluating a learned hypothesis we are most often interested in estimating the accuracy with which it will classify future instances. At the same time, we would like to know the probable error in this accuracy estimate (i.e., what error bars to associate with this estimate).

To illustrate, consider learning the target function "people who plan to purchase new skis this year," given a sample of training data collected by surveying people as they arrive at a ski resort. In this case the instance space X is the space of all people, who might be

described by attributes such as their age, occupation, how many times they skied last year, etc. The distribution D specifies for each person ***x*** the probability that x will be encountered as the next person arriving at the ski resort. The target function f: X -> **{0, 1}** classifies each person according to whether or not they plan to purchase skis this year. Within this general setting we are interested in the following two questions:

1. Given a hypothesis h and a data sample containing n examples drawn at random according to the distribution D, what is the best estimate of the accuracy of h over future instances drawn from the same distribution?
2. What is the probable error in this accuracy estimate?

5.2.1 Sample Error and True Error

To answer these questions, we need to distinguish carefully between two notions of accuracy or, equivalently, error. One is the error rate of the hypothesis over the sample of data that is available. The other is the error rate of the hypothesis over the entire unknown distribution ***D*** of examples. We will call these the ***sample error*** and the ***true error*** respectively. The ***sample error*** of a hypothesis with respect to some sample *S* of instances drawn from X is the fraction of *S* that it misclassifies:

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.

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 V.

5.2.2 Confidence Intervals for Discrete-Valued Hypothesis

Here we give an answer to the question "How good an estimate of $error_D(h)$ is provided by $error_s(h)$?" for the case in which ***h*** is a discrete-valued hypothesis. 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., $error_s(h) = r/n$).

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

1. Given no other information, the most probable value of $error_D(h)$ is $error_s(h)$

2. With approximately ***95%*** probability, the true error $error_D(h)$ lies in the interval

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

To illustrate, suppose the data sample ***S*** contains ***n*** = ***40*** examples and that hypothesis ***h*** commits ***r*** = ***12*** errors over this data. In this case, the sample error $error_s(h) = 12/40 = .30$. Given no other information, the best estimate of the true error $error_D(h)$ is the observed sample error ***.30.***

TABLE 5.1: Values of z_N for two-sided N% confidence intervals

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

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 $error_D(h)$ is

$$error_S(h) \pm z_N\sqrt{\frac{error_S(h)(1 - error_S(h))}{n}} \quad \text{--------- (5.1)}$$

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

5.3 BASICS OF SAMPLING THEORY

This section introduces basic notions from statistics and sampling theory, including probability distributions, expected value, variance, Binomial and Normal distributions, and two-sided and one-sided intervals. A basic familiarity with these concepts is important to understanding how to evaluate hypotheses and learning algorithms. Even more important,

these same notions provide an important conceptual framework for understanding machine learning issues such as overfitting and the relationship between successful generalization and the number of training examples considered. The key concepts introduced in this section are summarized in Table *5.2.*

TABLE 5.2: Basic definitions and facts from statistics

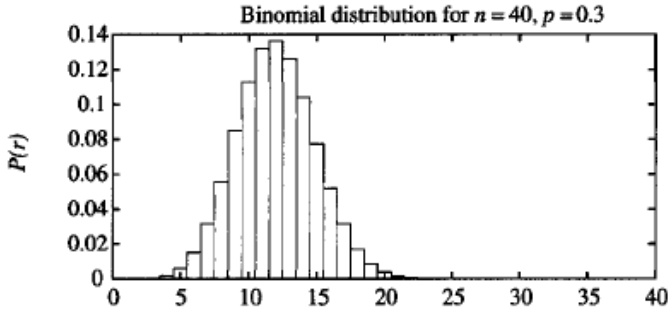
- A *random variable* can be viewed as the name of an experiment with a probabilistic outcome. Its value is the outcome of the experiment.
- A *probability distribution* for a random variable Y specifies the probability $\Pr(Y = y_i)$ that Y will take on the value y_i, for each possible value y_i.
- 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 $N\%$.

5.3.1 Error Estimation and Estimating Binomial Proportions

Precisely how does the deviation between sample error and true error depend on the size of the data sample? This question is an instance of a well-studied problem in statistics: the problem of estimating the proportion of a population that exhibits some property, given the observed proportion over some random sample of the population. In our case, the property of interest is that h misclassifies the example.

The key to answering this question is to note that when we measure the sample error we are performing an experiment with a random outcome. We first collect a random sample **S** of ***n*** independently drawn instances from the distribution ***D,*** and then measure the sample error $\boldsymbol{error_S(h)}$**.** As noted in the previous section, if we were to repeat this experiment many times, each time drawing a different random sample $\boldsymbol{S_i}$ of size n, we would expect to observe different values for the various errors,(h), depending on random differences in the makeup of the various $\boldsymbol{S_i}$.

The table 5.3 describes a particular probability distribution called the Binomial distribution.



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

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

For sufficiently large values of n the Binomial distribution is closely approximated by a Normal distribution (see Table 5.4) with the same mean and variance. Most statisticians recommend using the Normal approximation only when $np(1-p) \geq 5$.

5.3.2 The Binomial Distribution

A good way to understand the Binomial distribution is to consider the following problem. You are given a worn and bent coin and asked to estimate the probability that the coin will turn up heads when tossed. Let us call this unknown probability of heads p. You toss the coin n times and record the number of times r that it turns up heads. A reasonable estimate of p is r/n. The general setting to which the Binomial distribution applies is:

1. There is a base, or underlying, experiment (e.g., toss of the coin) whose outcome can be described by a random variable, say *Y*. The random variable *Y* can take on two possible values (e.g., *Y* = 1 if heads, Y = 0 if tails).
2. The probability that *Y* = 1 on any single trial of the underlying experiment is given by some constant p, independent of the outcome of any other experiment. The probability that *Y* = 0 is therefore (1 - p). Typically, p is not known in advance, and the problem is to estimate it.

3. A series of n independent trials of the underlying experiment is performed (e.g., n independent coin tosses), producing the sequence of independent, identically distributed random variables $Y_1, Y_2, \ldots, Yn$. Let R denote the number of trials for which $\boldsymbol{Y_i} = 1$ in this series of n experiments

$$R \equiv \sum_{i=1}^{n} Y_i$$

4. The probability that the random variable $\boldsymbol{R}$ will take on a specific value r (e.g., the probability of observing exactly r heads) is given by the Binomial distribution

$$\Pr(R = r) = \frac{n!}{r!(n-r)!} p^r (1-p)^{n-r} \qquad \text{------- (5.2)}$$

5.3.3 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. More precisely

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

$$E[Y] \equiv \sum_{i=1}^{n} y_i \Pr(Y = y_i) \qquad \text{------ (5.3)}$$

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*0.7 + 2*0.3 = 1.3). In case the random variable Y is governed by a Binomial distribution, then it can be shown that

$$E[Y] = np \qquad \text{------ (5.4)}$$

where n and p are the parameters of the Binomial distribution defined in Equation(5.2).

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

$$Var[Y] \equiv E[(Y - E[Y])^2] \qquad \text{---------- (5.5)}$$

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

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

------ (5.6)

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

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

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

-------- (5.7)

5.4 A GENERAL APPROACH FOR DERIVING CONFIDENCE INTERVALS

The previous section described in detail how to derive confidence interval estimates for one particular case: estimating $error_D(h)$ for a discrete-valued hypothesis ***h,*** based on a sample of ***n*** independently drawn instances. The approach described there illustrates a general approach followed in many estimation problems. In particular, we can see this as a problem of estimating the mean (expected value) of a population based on the mean of a randomly drawn sample of size ***n.***

The general process includes the following steps:

1. Identify the underlying population parameter p to be estimated, for example, $error_D(h)$.
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 ***Dy*** 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 ***Dy*** falls between L and U.

5.4.1 Central Limit Theorem

One essential fact that simplifies attempts to derive confidence intervals is the Central Limit Theorem. Consider again our general setting, in which we observe the values of ***n*** independently drawn random variables $Y_1 \ldots Y_n$ that obey the same unknown underlying probability distribution (e.g., ***n*** tosses of the same coin).

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.

5.5 DIFFERENCE IN ERROR OF TWO HYPOTHESES

Consider the case where we have two hypotheses $h_1$and h_2 for some discrete-valued target function. Hypothesis h_1 has been tested on a sample S_1 containing n_l randomly drawn examples, and h_2 has been tested on an independent sample S_2 containing n_2 examples drawn from the same distribution. Suppose we wish to estimate the difference d between the true errors of these two hypothesis.

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

We will use the generic four-step procedure described at the beginning of Section 5.4 to derive a confidence interval estimate for d. Having identified d as the parameter to be estimated, we next define an estimator. The obvious choice for an estimator in this case is the difference between the sample errors, which we denote by $\hat{d}$

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

Although we will not prove it here, it can be shown that $\hat{d}$ gives an unbiased estimate of d; that is $E[\hat{d}\,] = d$.

What is the probability distribution governing the random variable $\hat{d}$ ***?*** From earlier sections, we know that for large ***nl*** and ***n2*** (e.g., both >= ***30),*** both ***error***$_{s1}$ ***(h1)*** and ***error***$_{s2}$ ***(h2)*** follow distributions that are approximately Normal. Because the difference of two Normal distributions is also a Normal distribution, $\hat{d}$ will also follow a distribution that is approximately Normal, with mean d. It can also be shown that the variance of this distribution is the sum of the variances of ***error***$_{s1}$ ***(h1)*** and ***error***$_{s2}$ ***(h2)***. Using Equation ***(5.9)*** to obtain the approximate variance of each of these distributions, we have

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

$$\sigma_{\hat{d}}^2 \approx \frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2} \quad \text{--------- (5.12)}$$

Now that we have determined the probability distribution that governs the estimator $\hat{d}$, it is straightforward to derive confidence intervals that characterize the likely error in employing $\hat{d}$ to estimate d. For a random variable $\hat{d}$ obeying a Normal distribution with mean d and variance σ^2, the N% confidence interval estimate for d is $\hat{d} \pm z_N \sigma$. Using the approximate variance $\sigma_{\hat{d}}^2$ given above, this approximate ***N%*** confidence interval estimate for d is

$$\hat{d} \pm z_N \sqrt{\frac{error_{S_1}(h_1)(1-error_{S_1}(h_1))}{n_1} + \frac{error_{S_2}(h_2)(1-error_{S_2}(h_2))}{n_2}} \quad \text{--- (5.13)}$$

Although the above analysis considers the case in which $\mathbf{h_1}$ and $\mathbf{h_2}$ are tested on independent data samples, it is often acceptable to use the confidence interval seen in Equation ***(5.13)*** in the setting where $\mathbf{h_1}$ and $\mathbf{h_2}$ are tested on a single sample ***S*** (where ***S*** is still independent of h_1 and h_2**).** In this later case, we redefine $\hat{d}$ as

$$\hat{d} \equiv error_S(h_1) - error_S(h_2)$$

5.6 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 L_A and L_B is given in table below.

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

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

5.7 INTRODUCTION OF INSTANCE BASED LEARNING

Instance-based learning methods such as nearest neighbour and locally weighted regression are conceptually straightforward approaches to approximating real-valued or discrete-valued target functions. 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. Instance-based approaches can construct a different approximation to the target function for each distinct query instance that must be classified.

Advantages of Instance-based learning

1. Training is very fast

2. Learn complex target function

3. Don’t lose information

Disadvantages of Instance-based learning

1. The cost of classifying new instances can be high. This is due to the fact that nearly all computation takes place at classification time rather than when the training examples are first encountered.

2. In many instance-based approaches, especially nearest-neighbour approaches, is that they typically consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

5.8 k- NEAREST NEIGHBOR LEARNING

- The most basic instance-based method is the K- Nearest Neighbor Learning. This algorithm assumes all instances correspond to points in the n-dimensional space Rn.
- The nearest neighbors of an instance are defined in terms of the standard Euclidean distance.
- Let an arbitrary instance x be described by the feature vector

  ((a1(x), a2(x), ……, an(x))

  Where, ar(x) denotes the value of the r^{th} attribute of instance x.
- Then the distance between two instances xi and xj is defined to be d(xi , xj)

  Where,

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

- In nearest-neighbor learning the target function may be either discrete-valued or real-valued.

Let us first consider learning ***discrete-valued target functions*** of the form

$$f : \Re^n \rightarrow V.$$

Where, V is the finite set {v1, . . . vs }

The k- Nearest Neighbor algorithm for approximation a **discrete-valued target function** is given below:

Training algorithm:

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

Classification algorithm:

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

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

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

- The value f'(xq) returned by this algorithm as its estimate of f(xq) is just the most common value of f among the k training examples nearest to xq.
- If k = 1, then the 1- Nearest Neighbor algorithm assigns to f'(xq) the value f(xi). Where xi is the training instance nearest to xq.

- For larger values of k, the algorithm assigns the most common value among the k nearest training examples.
- Below figure illustrates the operation of the k-Nearest Neighbor algorithm for the case where the instances are points in a two-dimensional space and where the target function is Boolean valued.

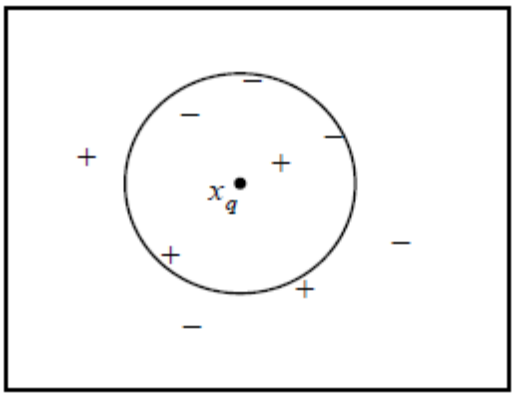


Figure 5.1: Operation of k-Nearest Neighbor algorithm

- The positive and negative training examples are shown by "+" and "-" respectively. A query point x_q is shown as well.
- The 1-Nearest Neighbor algorithm classifies x_q as a positive example in this figure, whereas the 5-Nearest Neighbor algorithm classifies it as a negative example.
- Below figure shows the shape of this **decision surface** induced by 1- Nearest Neighbor over the entire instance space. The decision surface is a combination of convex polyhedra surrounding each of the training examples.

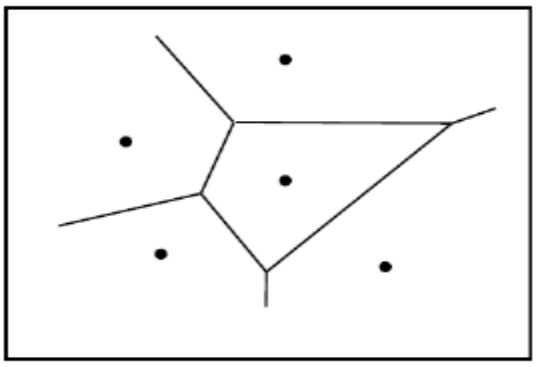


Figure 5.2: Decision surface induced by 1- Nearest Neighbor

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 example.

The K- Nearest Neighbor algorithm for approximation a **real-valued target function** is given below $f : \Re^n \rightarrow \Re$

Training algorithm:

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

Classification algorithm:

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

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

Distance-Weighted Nearest Neighbor Algorithm

- The refinement to the k-NEAREST NEIGHBOR Algorithm is to weight the contribution of each of the k neighbors according to their distance to the query point x_q, giving greater weight to closer neighbors.
- For example, in the k-Nearest Neighbor algorithm, which approximates discrete-valued target functions, we might weight the vote of each neighbor according to the inverse square of its distance from x_q.

Distance-Weighted Nearest Neighbor Algorithm for approximation a discrete-valued target functions is given below:

Training algorithm:

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

Classification algorithm:

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

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

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

Distance-Weighted Nearest Neighbor Algorithm for approximation a Real-valued target functions

---

Training algorithm:

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

Classification algorithm:

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

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

where

$$w_i \equiv \frac{1}{d(x_q, x_i)^2}$$

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

5.9 LOCALLY WEIGHTED REGRESSION

The phrase "**locally weighted regression**" is called **local** because the function is approximated based only on data near the query point, **weighted** because the contribution of each training example is weighted by its distance from the query point, and **regression** because this is the term used widely in the statistical learning community for the problem of approximating real-valued functions. Given a new query instance x_q, the general approach in locally weighted regression is to construct an approximation $\hat{f}$ that fits the training examples in the neighborhood surrounding x_q. This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.

Locally Weighted Linear Regression

Consider locally weighted regression in which the target function ***f*** is approximated near xq using a linear function of the form

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

Where, ai(x) denotes the value of the i^{th} attribute of the instance x

Derived methods are used to choose weights that minimize the squared error summed over the set D of training examples using gradient descent

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

Which led us to the gradient descent training rule

$$\Delta w_j = \eta \sum_{x \in D}(f(x) - \hat{f}(x))a_j(x)$$

Where, η is a constant learning rate

Need to modify this procedure to derive a local approximation rather than a global one. The simple way is to redefine the error criterion E to emphasize fitting the local training examples. Three possible criteria are given below.

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

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

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 xq:

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

3. Combine 1 and 2:

$$E_3(x_q) \equiv \frac{1}{2} \sum_{x \in k \text{ nearest nbrs of } x_q} (f(x) - \hat{f}(x))^2 \; K(d(x_q, x))$$ ------- **(3)**

If we choose criterion three and re-derive the gradient descent rule, we obtain the following training rule

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

The differences between this new rule and the rule given by Equation (3) are that the contribution of instance x to the weight update is now multiplied by the distance penalty **K(d(xq, x)),** and that the error is summed over only the k nearest training examples.

5.10 RADIAL BASIS FUNCTIONS

One approach to function approximation that is closely related to distance-weighted regression and also to artificial neural networks is learning with radial basis functions. In this approach, the learned hypothesis is a function of the form

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

- Where, each x_u is an instance from X and where the kernel function $K_u(d(x_u, x))$ is defined so that it decreases as the distance $d(x_u, x)$ increases.
- Here k is a user provided constant that specifies the number of kernel functions to be included.
- $\hat{f}$ is a global approximation to f(x), the contribution from each of the $K_u(d(x_u, x))$ terms is localized to a region nearby the point x_u.

Choose each function $K_u(d(x_u, x))$ to be a Gaussian function centred at the point x_u with some variance σ_u^2

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

The functional form of equation (1) can approximate any function with arbitrarily small error, provided a sufficiently large number k of such Gaussian kernels and provided the width σ^2 of each kernel can be separately specified. The function given by equation (1) can be viewed as describing a two layer network where the first layer of units computes the values of the various $K_u(d(x_u, x))$ and where the second layer computes a linear combination of these first-layer unit values.

Example: Radial basis function (RBF) network

Given a set of training examples of the target function, RBF networks are typically trained in a two-stage process.

1. 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 $K_u(d(x_u, x))$

2. Second, the weights w, are trained to maximize the fit of the network to the training data, using the global error criterion given by

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

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

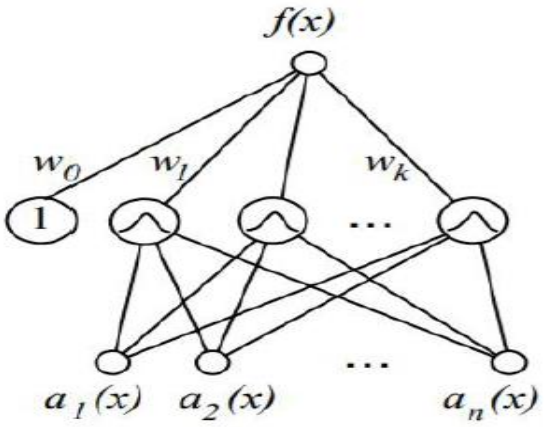


Figure 5.3: Radial basis function (RBF) network

Several alternative methods have been proposed for choosing an appropriate number of hidden units or, equivalently, kernel functions.

- One approach is to allocate a Gaussian kernel function for each training example (x_i, f (x_i)), centring this Gaussian at the point x_i.
- Each of these kernels may be assigned the same width σ^2. Given this approach, the RBF network learns a global approximation to the target function in which each training example (x_i, f (x_i)) can influence the value of ***f*** only in the neighbourhood of x_i.
- A second approach is to choose a set of kernel functions that is smaller than the number of training examples. This approach can be much more efficient than the first approach, especially when the number of training examples is large.

5.11 CASE-BASED REASONING

Case-based reasoning (CBR) is a learning paradigm based on lazy learning methods and they classify new query instances by analysing similar instances while ignoring instances that are very different from the query.

- In CBR represent instances are not represented as real-valued points, but instead, they use a ***rich symbolic*** representation.
- CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based

on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems.

A prototypical example of a case-based reasoning

- The CADET system 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.

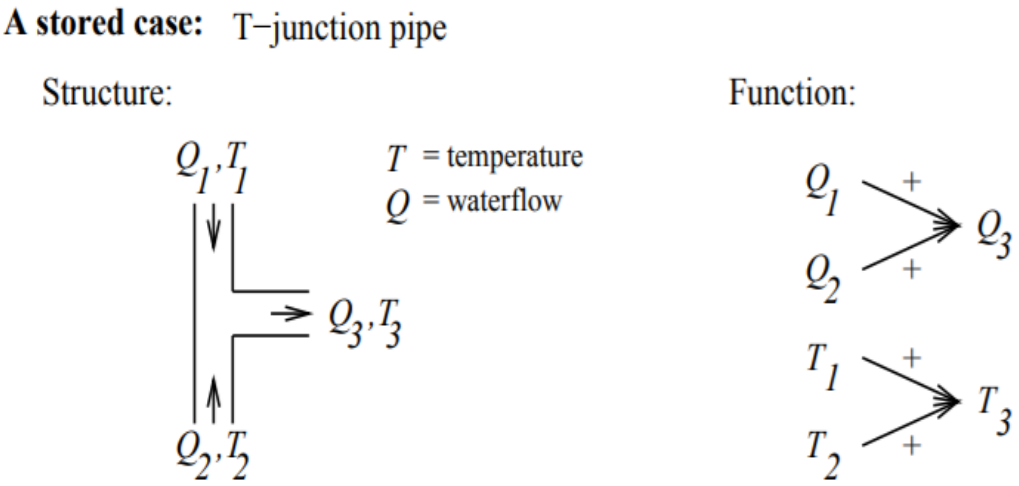


Figure 5.4: CADET system

- The function is represented in terms of the qualitative relationships among the water-flow levels and temperatures at its inputs and outputs.
- In the functional description, an arrow with a "+" label indicates that the variable at the arrowhead increases with the variable at its tail. A "-" label indicates that the variable at the head decreases with the variable at the tail.
- Here Qc refers to the flow of cold water into the faucet, Q_h to the input flow of hot water, and Qm to the single mixed flow out of the faucet.
- Tc, T_h, and Tm refer to the temperatures of the cold water, hot water, and mixed water respectively.
- The variable C_t denotes the control signal for temperature that is input to the faucet, and C_f denotes the control signal for water flow.

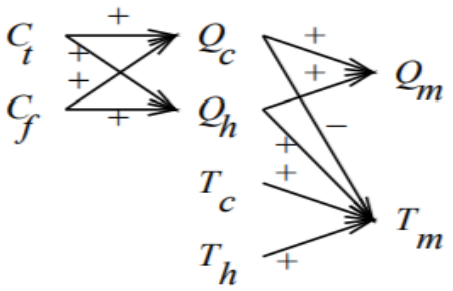
- The controls C_t and C_f are to influence the water flows Qc and Q_h, thereby indirectly influencing the faucet output flow Qm and temperature Tm.

A problem specification: Water faucet

Structure:

?

Function:



5.12 INTRODUCTION ON REINFORCEMENT 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.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. Its task is to learn a control strategy, or ***policy***, for choosing actions that achieve its goals. The goals of the agent can be defined by a ***reward function*** that assigns a numerical value to each distinct action the agent may take from each distinct state. This reward function may be built into the robot, or known only to an external teacher who provides the reward value for each action performed by the robot. The **task** of the robot is to perform sequences of actions, observe their consequences, and learn a control policy. The control policy is one that, from any initial state, chooses actions that maximize the reward accumulated over time by the agent.

Example:

A mobile robot may have sensors such as a camera and sonars, and actions such as "move forward" and "turn." The robot may have a goal of docking onto its battery charger whenever its battery level is low. The goal of docking to the battery charger can be captured by assigning a positive reward (Eg., +100) to state-action transitions that immediately result in a connection to the charger and a reward of zero to every other state-action transition.

Reinforcement Learning Problem

An agent interacting with its environment. The agent exists in an environment described by some set of possible states S. Agent perform any of a set of possible actions A. Each time it performs an action a, in some state 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 s_i,

actions a_i, and immediate rewards 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.

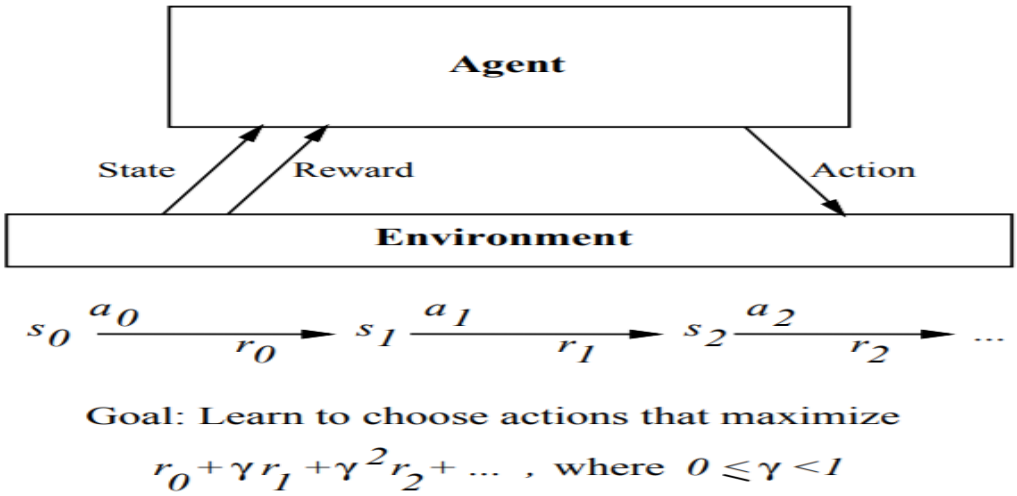


Figure 5.5: Reinforcement Learning Problem

5.13 THE LEARNING TASK

Consider Markov decision process (MDP) where 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 $r_t = r(s_t, a_t)$ and by producing the succeeding state $s_{t+1} = \delta(s_t, a_t)$. Here the functions $\delta(s_t, a_t)$ and $r(s_t, a_t)$ depend only on the current state and action, and not on earlier states or actions.

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, $\pi(s_t) = a_t$.

How shall we specify precisely which policy π we would like the agent to learn?

1. One approach is to require the policy that produces the greatest possible ***cumulative reward*** for the robot over time. To state this requirement more precisely, define the cumulative value $V^{\pi}(s_t)$ achieved by following an arbitrary policy π from an arbitrary initial state s_t as follows:

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

- ------- (1)

Where, the sequence of rewards r_{t+i} is generated by beginning at state s_t and by repeatedly using the policy π to select actions. Here $0 \leq \gamma \leq 1$ is a constant that determines the relative value of delayed versus immediate rewards. if we set $\gamma = 0$, only the immediate reward is considered. As we set γ closer to 1, future rewards are given greater emphasis relative to the immediate reward. The quantity $V^{\pi}(s_t)$ is called the ***discounted cumulative reward*** achieved by policy π from initial state ***s***. It is reasonable to discount future rewards relative to immediate rewards because, in many cases, we prefer to obtain the reward sooner rather than later.

2. Other definitions of total reward is ***finite horizon reward,***

$$\sum_{i=0}^{h} r_{t+i}$$

Considers the undiscounted sum of rewards over a finite number ***h*** of steps

3. Another approach is ***average reward***

$$\lim_{h\to\infty} \frac{1}{h} \sum_{i=0}^{h} r_{t+i}$$

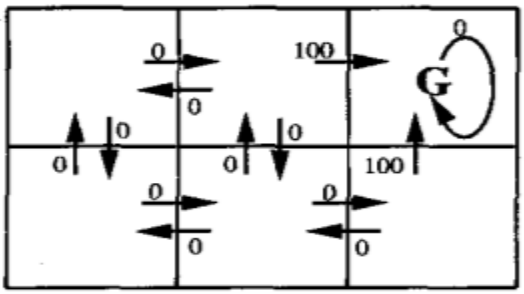
Considers the average reward per time step over the entire lifetime of the agent.

We require that the agent learn a policy π that maximizes $V^{\pi}(s_t)$ for all states s. such a policy is called an ***optimal policy*** and denote it by π^*

$$\pi^* \equiv \underset{\pi}{\operatorname{argmax}}\, V^{\pi}(s), (\forall s) \quad \text{--------- (2)}$$

Refer the value function $V^{\pi^*}(s)$ an optimal policy as $V^*(s)$. $V^*(s)$ gives the maximum discounted cumulative reward that the agent can obtain starting from state ***s***.

Example: A simple grid-world environment is depicted in the diagram

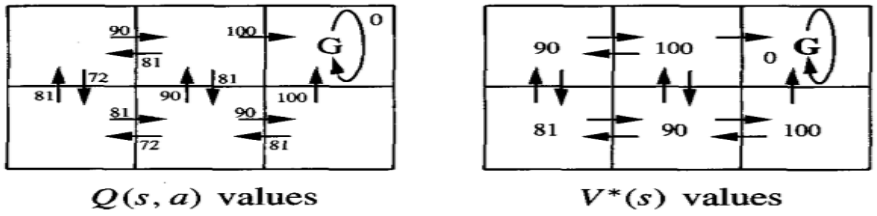


$r(s, a)$ (immediate reward) values

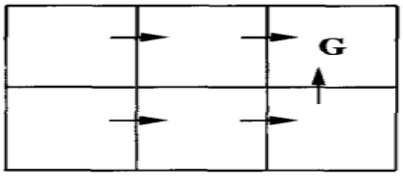
Figure 5.5: A simple grid-world environment

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. The immediate reward in this environment is defined to be zero for all state-action transitions except for those leading into the state labelled G. The state G as the goal state, and the agent can receive reward by entering this state. Once the states, actions, and immediate rewards are defined, choose a value for the discount factor γ, determine the optimal policy π* and its value function V*(s).

Let's choose γ = 0.9. The diagram at the bottom of the figure shows one optimal policy for this setting.



$Q(s, a)$ values $V^*(s)$ values



One optimal policy

Values of V*(s) and Q(s, a) follow from r(s, a), and the discount factor γ = 0.9. An optimal policy, corresponding to actions with maximal Q values, is also shown. The discounted future reward from the bottom centre state is

$$0+ \gamma\ 100+ \gamma^2\ 0+ \gamma^3\ 0+... = 90$$

5.14 Q LEARNING

1) How can an agent learn an optimal policy π * for an arbitrary environment?

The training information available to the learner is the sequence of immediate rewards $r(s_i,a_i)$ for i = 0,1,2, ... Given this kind of training information it is easier to learn a numerical evaluation function defined over states and actions, then implement the optimal policy in terms of this evaluation function.

2) What evaluation function should the agent attempt to learn?

One obvious choice is V*. The agent should prefer state s_1 over state s_2 whenever $V^*(s_1) > V^*(s_2)$, because the cumulative future reward will be greater from s_1. The optimal action in state ***s*** is the action ***a*** that maximizes the sum of the immediate reward r(s, a) plus the value V* of the immediate successor state, discounted by γ.

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}[r(s,a) + \gamma V^*(\delta(s,a))] \quad \text{------- (3)}$$

The *Q* Function

The value of Evaluation function ***Q(s, a)*** is the reward received immediately upon executing action a from state ***s***, plus the value (discounted by ***γ***) of following the optimal policy thereafter

$$Q(s,a) \equiv r(s,a) + \gamma V^*(\delta(s,a)) \quad \text{---------- (4)}$$

Rewrite Equation (3) in terms of ***Q(s, a)*** as

$$\pi^*(s) = \underset{a}{\operatorname{argmax}}\, Q(s,a) \quad \text{---------- (5)}$$

Equation (5) makes clear, it need only consider each available action ***a*** in its current state ***s*** and choose the action that maximizes ***Q(s, a)***.

Q learning algorithm

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

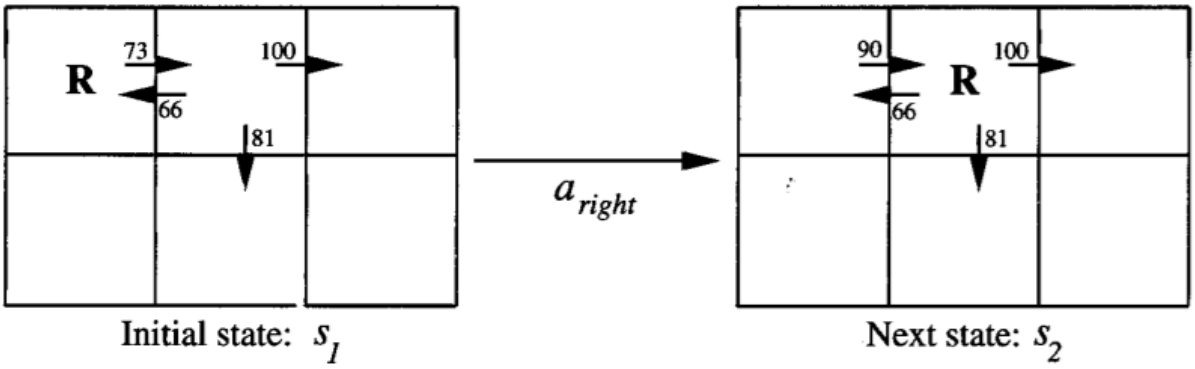
- Select an action a and execute it
- Receive immediate reward r
- Observe the new state s'
- Update the table entry for $\hat{Q}(s,a)$ as follows:

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

- $s \leftarrow s'$

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 Q shown in below figure



The agent moves one cell to the right in its grid world and receives an immediate reward of zero for this transition. Apply the training rule of Equation to refine its estimate Q for the state-action transition it just executed.

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

According to the training rule, the new Q estimate for this transition is the sum of the received reward (zero) and the highest Q value associated with the resulting state (100), discounted by γ (.9).

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

Acknowledgement

The diagrams and tables are taken from the textbooks specified in the references section.

Prepared by:

Rakshith M D

Department of CS&E

SDMIT, Ujire