GUDLAVALLERU ENGINEERING COLLEGE

(An Autonomous Institute with Permanent Affiliation to JNTUK, Kakinada)

Seshadri Rao Knowledge Village, Gudlavalleru – 521 356.

Department of Computer Science and Engineering

HANDOUT

on

MACHINE LEARNING

Vision

To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.

Mission

- To impart quality education through well-designed curriculum in tune with the growing software needs of the industry.
- To be a Centre of Excellence in computer science and engineering education and training to meet the challenging needs of the industry and society.
- To serve our students by inculcating in them problem solving, leadership, teamwork skills and the value of commitment to quality, ethical behavior & respect for others.
- To foster industry-academia relationship for mutual benefit and growth.

Program Educational Objectives

- Identify, analyze, formulate and solve Computer Science and Engineering problems both independently and in a team environment by using the appropriate modern tools.

- Manage software projects with significant technical, legal, ethical, social, environmental and economic considerations.

- Demonstrate commitment and progress in lifelong learning, professional development, leadership and Communicate effectively with professional clients and the public.

HANDOUT ON MACHINE LEARNING

Class & Sem : IV B.Tech – II Semester Year : 2019-20

Branch : CSE Credits : 3

1. Brief History & Scope of the Subject

A machine that is intellectually capable as much as humans has always fired the imagination of writers and also the early computer scientist who were excited about artificial intelligence and machine learning, but the first machine learning system was developed in the 1950s. In 1952, Arthur Samuel was at IBM. He developed a program for playing Checkers. In 1957, Rosenblatt proposed the Perceptron. However, the work along these lines suffered a setback when Minsky in 1969 came up with the limitations of perceptron. In 1986, J.R.Quinlan came up with decision tree learning, specifically the ID3 algorithm.

In the 90s, machine learning embraced statistics to a large extent. It was during this time, that support vector machines were proposed. It was a machine learning breakthrough and the support vector machines was proposed by Vapnik and Cortesin 1995 and S.V. Hemhad very strong theoretical standing and empirical results. Another strong machine learning model was proposed by Freund and Schapirein 1997, which was part of what we called ensembles or boosting and they came up with an algorithm called Adaboost by which they could create a strong classifier from an ensemble of weak classifiers. During 2001, Bayes net learning was also proposed. The rise of neural network began roughly in 2005 with the conjunction of many different discoveries for people by Hinton, LeCun, Bengio, Andrew and other researchers.

Some of the applications of machine learning are : In 1994, the first self driving car made a road test; in 1997, Deep Blue beat the world champion Gary Kasparov in the game of chess; in 2009 we have Google building self driving cars; in 2011,Watson, again from IBM, won the popular game of Jeopardy;

2014, human vision surpassed by ML systems. In 2014-15, machine translation systems driven by neural networks are very good and they are better than the other statistical machine translation systems where certain concepts and certain technology.

Now, in machine learning we have GPU's, which are enabling the use of machine learning and deep neural networks. There is the cloud, there is availability of big data and the field of machine learning is very exciting now.

2. Prerequisites

- Mathematics & statistics – calculus, differential equations, probability theory, graph theory
- Programming Experience – Python, R

3. Course Objectives

- To familiarize with supervised and unsupervised learning.
- To get acquainted with various machine learning algorithms.

4. Course Outcomes: Students will be able to

CO1: Define the basic concepts and applications of Machine learning

CO2:Demonstrate Version Space Find-S and Candidate elimination algorithms.

CO3: Design a decision tree

CO4: Apply appropriate supervised machine learning algorithm for an application.

CO5: Apply appropriate unsupervised machine learning algorithm for an application.

CO6: Compare supervised learning algorithms with unsupervised machine learning algorithms.

Program Outcomes:

Computer Science and Engineering Graduates will be able to:

1. **Engineering knowledge:** Apply the knowledge of mathematics, science, engineering fundamentals, and an engineering specialization to the solution of complex engineering problems.
2. **Problem analysis:** Identify, formulate, review research literature, and analyze complex engineering problems reaching substantiated conclusions using first principles of mathematics, natural sciences, and engineering sciences.
3. **Design/development of solutions:** Design solutions for complex engineering problems and design system components or processes that meet the specified needs with appropriate consideration for the public health and safety, and the cultural, societal, and environmental considerations.
4. **Conduct investigations of complex problems**: Use research-based knowledge and research methods including design of experiments, analysis and interpretation of data, and synthesis of the information to provide valid conclusions.
5. **Modern tool usage:** Create, select, and apply appropriate techniques, resources, and modern engineering and IT tools including prediction and modelling to complex engineering activities with an understanding of the limitations.
6. **The engineer and society:** Apply reasoning informed by the contextual knowledge to assess societal, health, safety, legal and cultural issues and the consequent responsibilities relevant to the professional engineering practice.
7. **Environment and sustainability:** Understand the impact of the professional engineering solutions in societal and environmental contexts, and demonstrate the knowledge of, and need for sustainable development.

8. **Ethics:** Apply ethical principles and commit to professional ethics and responsibilities and norms of the engineering practice.
9. **Individual and team work:** Function effectively as an individual, and as a member or leader in diverse teams, and in multidisciplinary settings.
10. **Communication:** Communicate effectively on complex engineering activities with the engineering community and with society at large, such as, being able to comprehend and write effective reports and design documentation, make effective presentations, and give and receive clear instructions.
11. **Project management and finance:** Demonstrate knowledge and understanding of the engineering and management principles and apply these to one's own work, as a member and leader in a team, to manage projects and in multidisciplinary environments.
12. **Life-long learning:** Recognize the need for, and have the preparation and ability to engage in independent and life-long learning in the broadest context of technological change.

5. **Mapping of Course Outcomes with Program Outcomes:**

	a	b	c	d	e	f	g	h	i	j	k	l
CO1	H											M
CO2			H									
CO3												
CO4					M		M					
CO5					M		M					
CO6	M											

6. Prescribed Text Books:

1. Tom M. Mitchell, Machine Learning, Mc Graw Hill Education.

7. Reference Books:

1. Ethem Alpaydin, Introduction to machine learning, 2nd edition, PHI.

8. URLs and Other E-Learning Resources

1. https://www.coursera.org/learn/machine-learning

9. Digital Learning Materials:

1. http://nptel.ac.in/courses/106106139/
2. http://nptel.ac.in/courses/106105152/

10. Lecture Schedule / Lesson Plan:

Topic	No. of Periods	
	Theory	Tutorial
UNIT I : Introduction		
Well- posed learning problems	3	1
Designing a learning system,	3	
Perspectives and issues in machine learning.	1	
UNIT - II: Concept Learning		
Introduction, A concept learning task,	1	1
Concept learning as search, Find-s: finding a maximally specific hypothesis,	2	
Version spaces and the candidate elimination algorithm	2	1
Remarks on version spaces and candidate elimination	1	
Inductive bias	1	
UNIT - III: Decision Tree Learning		
Decision tree representation	1	1
Appropriate problems for decision tree learning	1	
The basic decision tree learning algorithm	4	
Hypothesis space search in decision tree learning	2	1
Inductive bias in decision tree learning	2	
Issues in decision tree learning.	2	
UNIT - IV: Bayesian learning		
Bayes theorem	1	1
Byes theorem and concept learning	3	
Maximum likelihood and least squared error hypotheses	1	
Maximum likelihood hypotheses for predicting probabilities	2	1
Bayes optimal classifier	1	
An example learning to classify text	1	
Bayesian belief networks	2	
UNIT - V: Computational learning theory - 1		
Probability learning an approximately correct hypothesis	2	1
Sample complexity for infinite Hypothesis spaces	3	
The mistake bound model of learning	3	
Instance Based learning- Introduction.	1	
UNIT - VI: Computational learning theory - 2		

K- Nearest Neighbour Learning	3	1
Locally Weighted Regression	2	
Radial Basis Functions	2	
Case-Based Reasoning	2	
Remarks on Lazy and Eager Learning	1	
Total No. of Periods	56	9

UNIT- I

Machine Learning is the field of study that gives computers the ability to learn without being explicitly programmed.

In other words, Machine learning is a set of tools which allows us to "teach" computers how to perform tasks by providing examples of how they should be done.

For **example**, suppose we wish to write a program to distinguish between valid email messages and unwanted spam. We could try to write a set of simple rules, for example, flagging messages that contain certain features (such as the word "GOOD NEWS" or obviously-fake headers). However, writing rules to accurately distinguish which text is valid can actually be quite difficult to do well, resulting either in many missed spam messages, or, worse, many lost emails. Worse, the spammers will actively adjust the way they send spam in order to trick these strategies (e.g., writing "G00D NEW$"). Writing effective rules - and keeping them up-to-date - quickly becomes an insurmountable task. Fortunately, machine learning has provided a solution. Modern spam filters are "learned" from examples: we provide the learning algorithm with example emails which we have manually labelled as "ham" (valid email) or "spam" (unwanted email), and the algorithms learn to distinguish between them automatically.

1.1 WELL-POSED LEARNING PROBLEMS

A computer program is said to ***learn*** from **experience E** with respect to some class of **tasks *T*** and **performance measure P**, if its performance at tasks in T, as measured by P, improves with experience E.

A computer program that learns **to play checkers** might improve its performance as ***measured by its ability to win*** at the class of tasks involving ***playing checkers games,*** through experience ***obtained*** by ***playing games against itself.***

In general, to have a well-defined learning problem, we must identity these three features:

- The class of tasks, T
- The measure of performance, P to be improved, and
- The source of experience, E

***Example1*: The Checkers learning problem:**

- **Task T:** playing checkers
- **Performance measure P:** percent of games won against opponents
- **Training experience E**: playing practice games against itself

***Example 2:* A handwriting recognition learning problem:**

- **Task T**: recognizing and classifying handwritten words within images
- **Performance measure *P:*** percent of words correctly classified
- **Training experience E**: a database of handwritten words with given classifications

***Example 3*: A robot driving learning problem:**

- **Task T**: driving on public four-lane highways using vision sensors
- **Performance measure P**: average distance travelled before an error (as judged by human overseer)
- **Training experience E**: a sequence of images and steering commands recorded while observing a human driver

Basic Terminology:

Training data: A set of examples used for learning.

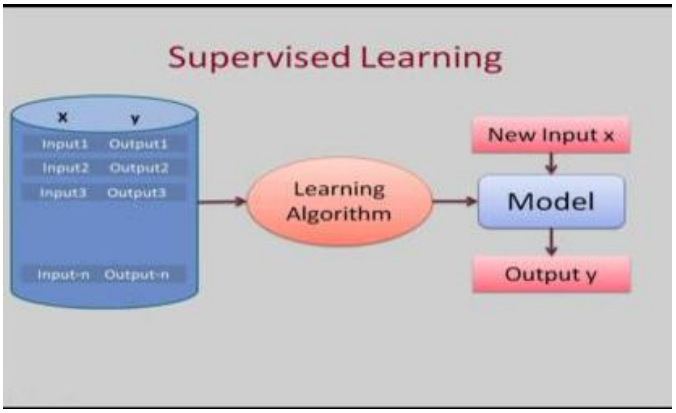
Test data: A set of examples used only to assess the performance of a fully-trained classifier.

Labelled and unlabeled data: If we consider data in the form of <x,y> (where x is input and y is output) then labelled data consists of both x and y where as unlabeled data has only x.

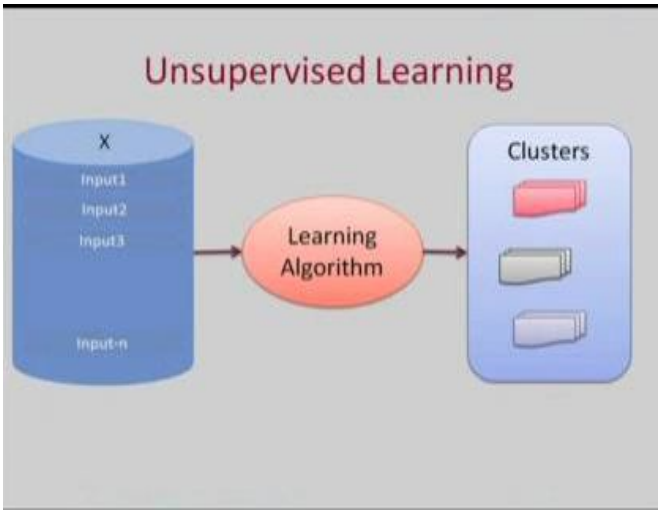
Types of Machine Learning paradigms:

Some of the main types of machine learning are:

- Supervised Learning
- Unsupervised Learning
- Reinforcement Learning



Supervised Learning: In Supervised Learning, the training data is labelled with the correct answers, for example whether the mail is "spam" or "ham". The two most common types of supervised learning are classification (where the outputs are discrete labels, as in spam filtering) and regression (where the outputs are continuous/real-valued). A simple demonstration of supervised learning is shown in the figure:



Unsupervised Learning: In Unsupervised learning, a collection of unlabeled data is given, which need to be analyzed and patterns discovered. The most important example of unsupervised learning is clustering. A simple

demonstration of unsupervised learning is shown in the figure:

Reinforcement learning: In Reinforcement learning, an agent (e.g., a robot or controller) seeks to learn the optimal actions to take based the outcomes of past actions.

Some Applications of Machine Learning:

Machine learning tools can be used in many domains like - medicine (eg: diagnose a disease), vision, robot control (eg: design autonomous mobile robots), etc. Some of the successful applications of machine learning include the following:

- **Learning to recognize spoken words**: For example, the SPHINX system learns speaker-specific strategies for recognizing the primitive sounds (phonemes) and words from the observed speech signal.
- **Learning to drive an autonomous vehicle:** For example, the ALVINN system has used its learned strategies to drive unassisted at 70 miles per hour for 90 miles on public highways among other cars.
- **Learning to classify new astronomical structures:** For example, decision tree learning algorithms have been used by NASA to learn how to classify celestial objects from the second Palomar Observatory Sky Survey. This system is now used to automatically classify all objects in the Sky Survey, which consists of three terabytes of image data.
- **Learning to play world-class backgammon**: For example, the world's top computer program for backgammon, TD-GAMMON learned its strategy by playing over one million practice games against itself. It now plays at a level competitive with the human world champion.

1.2 DESIGNING A LEARNING SYSTEM

The performance measure for playing checkers is the percent of games it wins in the world tournament. Designing a program to learn to play checkers, with

the goal of entering it in the world checkers tournament is briefly explained below:

The major choices in designing a learning system are:

- Choosing the Training Experience
- Choosing the Target Function
- Choosing a representation for the Target Function
- Choosing a Function approximation algorithm

1.2.1 Choosing the Training Experience:

- The type of training experience available can have a significant impact on success or failure of the learner.
- The three important attributes of Training experience are
    a. Direct/Indirect feedback
    b. The degree to which the learner controls the sequence of training examples
    c. The distribution of examples

Direct/Indirect feedback regarding the choices made by the performance system:

- One of the key attributes of Training experience are whether the training experience has direct/indirect feedback regarding the choices made by the performance system.

    Example: In learning to play checkers, the system might learn from ***direct*** training examples consisting of individual checkersboard states and the correct move for each. Alternatively, it might have available only ***indirect*** information consisting of the move sequences and final outcomes of various games played.
- In Indirect feedback there is a problem of **credit assignment** - determining the degree to which each move in the sequence deserves credit or blame for the final outcome. Hence, learning from direct training feedback is typically easier than learning from indirect feedback.

The degree to which the learner controls the sequence of training examples:

- For example, in learning to play checkers, the learner might rely on the teacher to select informative board states and to provide the correct move for each.
- Alternatively, the learner might itself propose board states that it finds particularly confusing and ask the teacher for the correct move or the learner may have complete control over both the board states and (indirect) training classifications, as it does when it learns by playing against itself with no teacher present.

The distribution of examples:

- Learning is most reliable when the training examples follow a distribution similar to that of future test examples.
- In checkers learning scenario, the performance metric P is the percent of games the system wins in the world tournament. If its training experience E consists only of games played against itself, there is no obvious danger that this training experience might not be fully representative of the distribution of situations over which it will later be tested.
- For example, the learner might never encounter certain crucial board states that are very likely to be played by the human checkers champion. In practice, it is often necessary to learn from a distribution of examples that is somewhat different from those on which the final system will be evaluated (e.g., the world checkers champion might not be interested in teaching the program!).

A checkers learning problem:

- Task **T**: playing checkers
- Performance measure ***P:*** percent of games won in the world tournament
- Training experience **E**: games played against itself

In order to complete the design of the learning system, we must now choose

1. the exact type of knowledge to be learned
2. a representation for this target knowledge
3. a learning mechanism

1.2.2 Choosing the Target Function:

- This design choice determines exactly what type of knowledge will be learned and how this will be used by the performance program.
- Checkers-playing program must be able to generate the ***legal*** moves from any board state. The program needs only to learn how to choose the ***best*** move from among these legal moves.
- Let us call this function ***ChooseMove*** and use the notation ***ChooseMove*: B→M** to indicate that this function accepts as input any board from the set of legal board states *B* and produces as output some move from the set of legal moves ***M.***
- We can view our task of improving performance P at task T to the problem of learning a target function such as *ChooseMove*. Hence the choice of target function is the key design choice.
- An alternative target function and one that will turn out to be easier to learn in checkers-playing program is an evaluation function that assigns a numerical score to any given board state. Let us call this target function ***V*** and again use the notation ***V*** : B→ ℝ to denote that ***V*** maps any legal board state from the set B to some real value .
- What exactly should be the value of the target function ***V*** for any given board state? Of course any evaluation function that assigns higher scores to better board states will do. Nevertheless, we will find it useful to define one particular target function ***V*** among the many that produce optimal play. As we shall see, this will make it easier to design a training algorithm. Let us therefore define the target value ***V(b)*** for an arbitrary board state ***b*** in *B*, as follows:

1. if b is a final board state that is won, then V(b) = 100
2. if b is a final board state that is lost, then V(b) = -100
3. if b is a final board state that is drawn, then V(b) = 0
4. if b is a not a final state in the game, then V(b) = V(b'), where b' is the best final board state that can be achieved starting from b and playing optimally until the end of the game (assuming the opponent plays optimally, as well).

- The goal of learning in this case is to discover an ***operational*** description of ***V;*** that is, a description that can be used by the checkers-playing program to evaluate states and select moves within realistic time bounds.
- Now our learning task has become as the problem of determining "an operational description of ideal target function" which is very difficult to learn so we try to learn an "approximation to target function" denoted by

1.2.3 Choosing a Representation for the Target Function:

- We must choose a representation that the learning program will use to describe the function that it will learn.
- can be represented by using a large table or collection of rules or a quadratic polynomial function or an ANN (Artificial Neural Network).
- In general, this choice of representation involves a crucial tradeoff.
- On one hand, we wish to pick a very expressive representation to allow representing as close an approximation as possible to the ideal target function V.
- On the other hand, the more expressive the representation, the more training data the program will require in order to choose among the alternative hypotheses it can represent.

Example: For Checkers-playing program, a simple representation for the function □ will be calculated as a linear combination of the following board features:

X_1: the number of black pieces on the board

X_2: the number of red pieces on the board

X_3: the number of black kings on the board

X_4: the number of red kings on the board

X_5: the number of black pieces threatened by red (i.e., which can be captured on red's next turn)

X_6: the number of red pieces threatened by black

Thus, our learning program will represent □(b) as a linear function of the form

$$□(b) = W_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + W_5X_5 + W_6X_6$$

where W_0 through W6 are numerical coefficients, or weights, to be chosen by the learning algorithm.

To summarize our design choices thus far, we have elaborated the original formulation of the learning problem by choosing a type of training experience, a target function to be learned, and a representation for this target function. Our elaborated learning task is now

Partial design of a checkers learning program:

- Task T: playing checkers
- Performance measure ***P:*** percent of games won in the world tournament
- Training experience E: games played against itself
- Target function: V:Board + ℝ
- Target function representation

  $$□(b) = W_0 + W_1X_1 + W_2X_2 + W_3X_3 + W_4X_4 + W_5X_5 + W_6X_6$$

The first three items above correspond to the specification of the learning task, whereas the final two items constitute design choices for the implementation of the learning program. Notice the net effect of this set of design choices is to

reduce the problem of learning a checkers strategy to the problem of learning values for the coefficients W_0 through W_6 in the target function representation.

1.2.4 Choosing a Function Approximation Algorithm:

In order to learn the target function □ we require a set of training examples, each describing a specific board state b and the training value $V_{train(b)}$ for b. In other words, each training example is an ordered pair of the form <b, $V_{train(b)}$>.

- **Estimating Training Values:** It is ambiguous to assign training values to intermediate states of a learning problem. But, one simple approach has been found to be surprisingly successful for this. The approach is to assign the training value of $V_{train(b)}$ for any intermediate board state *b* to be □*(Successor(b))* where □ is the learner's current approximation to *V* and *Successor(b)* denotes the next board state following *b* for which it is again the program's turn to move (i.e., the board state following the program's move and the opponent's response). This rule for estimating training values can be summarized as

  $$\mathbf{V_{train(b)} \leftarrow \square(Successor(b))}$$

- **Adjusting the Weights:** Now we have to choose the weights Wi to best fit the set of training examples <b, $V_{train(b)}$>. One common approach is to define set of weights, as that which minimizes the squared error E between the training values and the values predicted by the hypothesis □*:*

  $$\mathbf{E \equiv \sum_{<b,\ Vtrain(b)> \in\ Training\ Examples} (V_{train(b)} - \square(b))^2}$$

  Thus, we seek the weights, or equivalently the □, that minimize E for the observed training examples.

  LMS (Least Mean Square) Algorithm: LMS is a weight adjustment algorithm, which will incrementally refine the weights as new training examples become available and that will be robust to errors in the estimated training values. The LMS algorithm is defined as follows:

For each training example <b, $V_{train(b)}$>

- **Use the current weights to calculate □(b)**
- **For each weight W_i, update it as**

$$W_i \leftarrow W_i + \square\ (V_{train(b)} - \square(b))\ X_i$$

Here □ is a small constant (e.g., 0.1) that moderates the size of the weight update.

1.2.5 Final Design of a Learning System:

In general, the central components in any learning problem are the following:

- Performance System
- Critic
- Generalizer
- Experiment Generator

The **Performance System** is the module that must solve the given performance task, in this case playing checkers, by using the learned target function(s). It takes an instance of a new problem (new game) as input and produces a trace of its solution (game history) as output. In our case, the strategy used by the Performance System to select its next move at each step is determined by the learned □ evaluation function. Therefore, we expect its performance to improve as this evaluation function becomes increasingly accurate.

The **Critic** takes as input the history or trace of the game and produces as output a set of training examples of the target function. As shown in the diagram, each training example in this case corresponds to some game state in the trace, along with an estimate ***Vtrai*n,** if the target function value for this example.

The **Generalizer** takes as input the training examples and produces an output hypothesis that is its estimate of the target function. It generalizes from the specific training examples, hypothesizing a general function that covers these examples and other cases beyond the training examples. In our example, the

Generalizer corresponds to the LMS algorithm, and the output hypothesis is the function described by the learned weights ***wo,*** . . . , ***W6.***

The **Experiment Generator** takes as input the current hypothesis (currently learned function) and outputs a new problem (i.e., initial board state) for the Performance System to explore. Its role is to pick new practice problems that will maximize the learning rate of the overall system. In our example, the Experiment Generator follows a very simple strategy: It always proposes the same initial game board to begin a new game. More sophisticated strategies could involve creating board positions designed to explore particular regions of the state space.

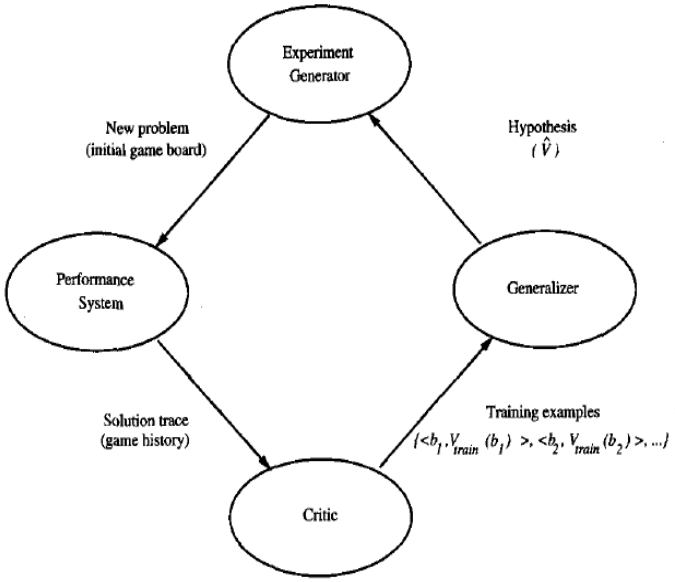


Figure: Summary of choices in designing the checkers learning program.

1.3 Perspectives in Machine Learning:

- One useful perspective on machine learning is that it involves searching a very large space of possible hypotheses to determine one that best fits the observed data and any prior knowledge held by the learner.
- For example, consider the space of hypotheses that could in principle be output by the above checkers learner. This hypothesis space consists of all evaluation functions that can be represented by some choice of values for the weights W_0 through W_6**.** The learner's task is thus to search through this vast space to locate the hypothesis that is most consistent with the available training examples.
- The LMS algorithm for fitting weights achieves this goal by iteratively tuning the weights, adding a correction to each weight each time the hypothesized evaluation function predicts a value that differs from the training value. This algorithm works well when the hypothesis representation considered by the learner defines a continuously parameterized space of potential hypotheses.

Issues in Machine Learning:

The checkers example raises a number of generic questions about machine learning and they are given below:

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?

- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

UNIT-I

Assignment-Cum-Tutorial Questions

SECTION-A

Objective Questions

1. Machine learning is [ ]

A. The autonomous acquisition of knowledge through the use of computer programs

B. The autonomous acquisition of knowledge through the use of manual programs

C. The selective acquisition of knowledge through the use of computer programs

D. The selective acquisition of knowledge through the use of manual programs

2. Factors which affect the performance of learner system does not include [ ]

A. Representation scheme used B. Training scenario

C. Type of feedback D. Good data structures

3. What types of Machine Learning, if any, best describe the following three scenarios: [ ]

(i) A coin classification system is created for a vending machine. The developers obtain exact coin specifications from the U.S. Mint and derive a statistical model of the size, weight, and denomination, which the vending machine then uses to classify coins.

(ii) Instead of calling the U.S. Mint to obtain coin information, an algorithm is presented with a large set of labeled coins. The algorithm uses this data to infer decision boundaries which the vending machine then uses to classify its coins.

(iii) A computer develops a strategy for playing Tic-Tac-Toe by playing repeatedly and adjusting its strategy by penalizing moves that eventually lead to losing.

A. (i) Supervised Learning, (ii) Unsupervised Learning, (iii) Reinforcement Learning

B. (i) Supervised Learning, (ii) Not learning, (iii) Unsupervised Learning

C. (i) Not learning, (ii) Reinforcement Learning, (iii) Supervised Learning

D. (i) Not learning, (ii) Supervised Learning, (iii) Reinforcement Learning

E. (i) Supervised Learning, (ii) Reinforcement Learning, (iii) Unsupervised Learning

4. Which of the following problems are best suited for Machine Learning? [ ]

(i) Classifying numbers into primes and non-primes.

(ii) Detecting potential fraud in credit card charges.

(iii) Determining the time it would take a falling object to hit the ground.

(iv) Determining the optimal cycle for traffic lights in a busy intersection.

A. (ii) and (iv) B. (i) and (ii)

C. (i), (ii), and (iii) D. (iii) E. (i) and (iii)

5. I am the marketing consultant of a leading e-commerce website. I have been given a task of making a system that recommends products to users based on their activity on Facebook. I realize that user-interests could be highly variable. Hence I decide to

i. First, cluster the users into communities of like-minded people and

ii. Second, train separate models for each community to predict which product category (e.g. electronic gadgets, cosmetics, etc) would be the most relevant to that community.

The first task is a/an _____ learning problem while the second is a/an _____ problem. [ ]

Choose from the options:

A. Supervised and unsupervised B. Unsupervised and supervised

C. Supervised and supervised D. Unsupervised and unsupervised

6. Which ONE of the following are regression tasks? [ ]

A. Predict the age of a person

B. Predict the country from where the person comes from

C. Predict whether the price of petroleum will increase tomorrow

D. Predict whether a document is related to science

7. Which of the following are classification tasks? (Mark all that apply) [ ]

A. Find the gender of a person by analyzing his writing style

B. Predict the price of a house based on floor area, number of rooms etc.

C. Predict whether there will be abnormally heavy rainfall next year

D. Predict the number of copies of a book that will be sold this month

8. Which of the following are examples of unsupervised learning? [ ]

A. Group news articles based on text similarity

B. Make clusters of books on similar topics in a library

C. Filter out spam emails

D. Segment online customers into two classes based on their age group – below 25 or above 25

SECTION-B

Descriptive Questions

1. Define machine learning.
2. What do you mean by a well –posed learning problem? Explain the important features that are required to well –define a learning problem.
3. Explain well posed learning problem for the following:
   i) A checkers learning problem.
   ii) A handwritten recognition learning problem.
   iii) A robot driving learning problem.
4. Discuss different applications of machine Learning.
5. Explain the phases in designing a learning system.

6. State LMS weight update rule.

7. Discuss perspectives and issues in machine learning.

8. Define target function in machine learning with an example.

9. List the objectives of machine learning.

10. Devise a simple machine learning solution to solve Checkers game problem.

11. Explain various design choices of machine learning strategies.

UNIT-II

CONCEPT LEARNING AND THE GENERAL-TO-SPECIFIC ORDERING

2.1 INTRODUCTION

- Inducing general functions from specific training examples is a main issue of machine learning.
- Concept Learning: Acquiring the definition of a general category from given sample positive and negative training examples of the category.
- Concept Learning can be seen as a problem of searching through a predefined space of potential hypotheses for the hypothesis that best fits the training examples.
- The hypothesis space has a *general-to-specific ordering* of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure over the hypothesis space.
- An example for concept-learning is the learning of bird-concept from the given examples of birds (positive examples) and non-birds (negative examples).
- We are trying to learn the definition of a concept from given examples.

A Formal Definition for Concept Learning: Inferring a boolean-valued function from training examples of its input and output.

2.2 A CONCEPT LEARNING TASK

To understand concept learning, consider the example of "EnjoySport". The task is to "learn to predict the value of EnjoySport for an arbitrary day", based on the values of its attribute values. Below table describes a set of example days, each represented by a set of ***attributes.***

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	**YES**
2	Sunny	Warm	High	Strong	Warm	Same	**YES**
3	Rainy	Cold	High	Strong	Warm	Change	**NO**
4	Sunny	Warm	High	Strong	Warm	Change	**YES**

ATTRIBUTES **CONCEPT**

2.2.1 Hypotheses representation of EnjoySport Example:

- In a simple representation of the hypothesis, each hypothesis consists of a conjunction of constraints on the instance attributes.
- For EnjoySport Example, each hypothesis will be a vector of six constraints, specifying the values of the six attributes:
  - (Sky, AirTemp, Humidity, Wind, Water, and Forecast)
- Each attribute will be:

  ? - indicating any value is acceptable for the attribute **(don't care)**

  Single value – specifying a single required value (ex. Warm) **(specific)**

  0 - indicating no value is acceptable for the attribute **(no value)**
- Example: The hypothesis that X enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes) is represented by the expression

  <?, Cold, High, ?, ?, ?>
- The most general hypothesis – that every day is a positive example – is represented by

  <?,?,?,?,?,?>
- The most specific hypothesis – that no day is a positive example – is represented by

  <ø, ø, ø, ø, ø, ø>

2.2.2 Basic Terminology:

Instances: The set of items over which the concept is defined is called the set of instances, which we denote by X.

In our example EnjoySport, X is the set of all possible days, each represented by the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.

Target Concept: The concept or function to be learned is called the target concept, which is denoted by c. In general, c can be any Boolean-valued function defined over the instances X; i.e., $c : X \rightarrow \{0, 1\}$.

In our current example, the target concept corresponds to the value of the attribute EnjoySport (i.e., c(x) = 1 if EnjoySport = Yes, and c(x) = 0 if EnjoySport = No).

Training Example: Each training example <x, c(x)>, consists of an instance x from X, along with its target concept value c(x).

Positive Examples: Instances for which c(x) = 1 are called positive examples, or members of the target concept.

Negative Examples: Instances for which c(x) = 0 are called negative examples, or non-members of the target concept.

Hypotheses: H denotes the set of all possible hypotheses that the learner may consider regarding the identity of the target concept c. In general, each hypothesis h in H represents a Boolean-valued function defined over X; that is, $h : X \rightarrow \{0, 1\}$. The goal of the learner is to find a hypothesis h such that h(x) = c(x) for all x in X.

The EnjoySport Concept Learning Task:

- Given:
  - Instances X: Possible days, each described by the attributes
    - Sky (with possible values Sunny, Cloudy, and Rainy),
    - AirTemp (with values Warm and Cold),
    - Humidity (with values Normal and High),
    - Wind (with values Strong and Weak),
    - Water (with values Warm and Cool), and
    - Forecast (with values Same and Change).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes Sky, AirTemp, Humidity, Wind, Water, and Forecast.
  - Target concept c: EnjoySport : $X \rightarrow \{0,1\}$
  - Training examples D: Positive and negative examples of the target function
- Determine: A hypothesis h in H such that h(x) = c(x) for all x in X.

2.2.3 The Inductive Learning Hypothesis

Although the learning task is to determine a hypothesis h identical to the target concept cover the entire set of instances X, the only information available about c is its value over the training examples.

– Inductive learning algorithms can at best guarantee that the output hypothesis fits the target concept over the training data.

– Lacking any further information, our assumption is that the best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. This is the fundamental assumption of inductive learning.

The Inductive Learning Hypothesis: Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.

2.3 CONCEPT LEARNING AS SEARCH

Concept learning can be viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.

• The goal of this search is to find the hypothesis that best fits the training examples.

• By selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.

• For example consider **EnjoySport** learning task:

Sky has 3 possible values, and other 5 attributes have 2 possible values.

• There are 96 (= 3.2.2.2.2.2) distinct instances in X.

• There are 5120 (=5.4.4.4.4.4) *syntactically distinct hypotheses* in H.

– Two more values for attributes: ? and 0

• Every hypothesis containing one or more 0 symbols represents the empty set of instances; that is, it classifies every instance as negative.

• There are 973 (= 1 + 4.3.3.3.3.3) *semantically distinct hypotheses* in H.

– Only one more value for attributes: ?, and one hypothesis representing empty set of instances.

• Although EnjoySport has small, finite hypothesis space, most learning tasks have much larger (even infinite) hypothesis spaces.

– We need efficient search algorithms on the hypothesis spaces.

2.3.1 General-to-Specific Ordering of Hypotheses:

- Many algorithms for concept learning organize the search through the hypothesis in a **general-to-specific ordering of hypotheses**.
- By using this ordering of hypotheses, the learning algorithms can exhaustively search even infinite hypothesis spaces without explicitly enumerating every hypothesis.
- Consider two hypotheses

  h1 = (Sunny, ?, ?, Strong, ?, ?)

  h2 = (Sunny, ?, ?, ?, ?, ?)

  • Now consider the sets of instances that are classified positive by hl and by h2.

  – Because h2 imposes fewer constraints on the instance, it classifies more instances as positive.

  – In fact, any instance classified positive by hl will also be classified positive by h2.

– Therefore, we say that h2 is *more general than* hl.

More-General-Than Relation

• For any instance x in X and hypothesis h in H, we say that x satisfies h if and only if h(x) = 1.

• **More-General-Than-Or-Equal Relation:**

Let h1 and h2 be two boolean-valued functions defined over X.

Then h1 is ***more-general-than-or-equal-to*** h2 (written h1 ≥ h2) if and only if any instance that satisfies h2 also satisfies h1.

- h1 is ***more-general-than*** h2 (h1 > h2) if and only if h1≥h2 is true and h2≥h1 is false. We also say h2 is ***more-specific-than*** h1.

Definition: Let h_j and h_k ***be*** boolean-valued functions defined over X. Then h_j is **more_general_than_or_equal_to** h_k (written $h_j \geq_g h_k$) if and only if

$$(\forall x \in X)[(h_k(x) = 1) \rightarrow (h_j(x) = 1)]$$

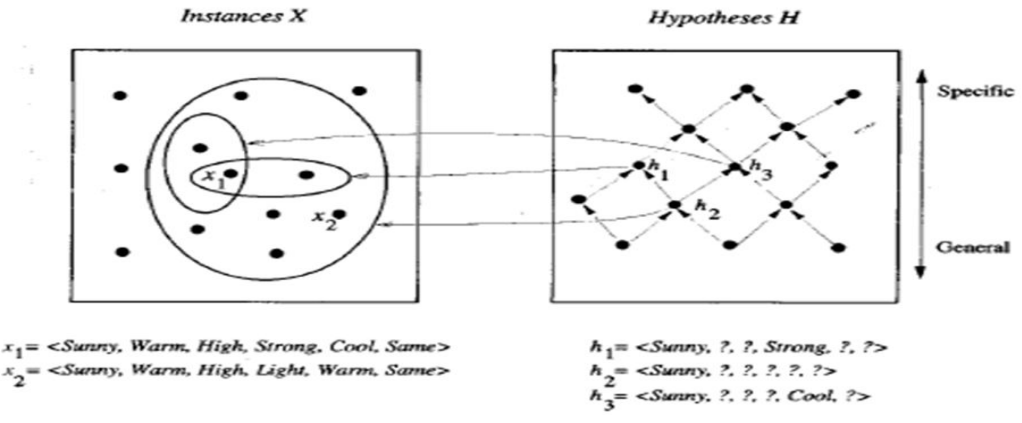
- For example, consider the two hypotheses:

  h_1 = <Sunny, ?, ?, Strong, ?, ?>

  h_2 = <Sunny, ?, ?, ?, ?, ?>

  Any instance classified positive by h_1 will also be classified positive by h_2. Therefore, we say that h_2 is more general than h_1.

- The $\geq_g$, relation defines a partial order over the hypothesis space H (the relation is reflexive, anti-symmetric and transitive).



Based on the above figure:

- h2 > h1 and h2 > h3

- But there is no more-general relation between h1 and h3

2.4 FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

FIND-S Algorithm starts from the most specific hypothesis and generalize it by considering only positive examples.

• FIND-S algorithm ignores negative examples.

– As long as the hypothesis space contains a hypothesis that describes the true target concept, and the training data contains no errors, ignoring negative examples does not cause to any problem.

• FIND-S algorithm finds the most specific hypothesis within H that is consistent with the positive training examples.

– The final hypothesis will also be consistent with negative examples if the correct target concept is in H, and the training examples are correct.

- This algorithm uses more-general-than partial ordering to organize the search in the hypotheses space.

- **FIND-S Algorithm:**
  1. Initialize h to the most specific hypothesis in H
  2. For each positive training instance ***x***
     - For each attribute constraint a_i in h
       If the constraint a_i is satisfied by ***x***
       Then do nothing
       Else replace a_i in h by the next more general constraint that is satisfied by ***x***
  3. Output hypothesis h

- **Example: Using Find-S algorithm for EnjoySport task**

  Step 1: The first step of FIND-S is to initialize *h* to the most specific hypothesis in H.

  $$h \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset, \emptyset, \emptyset \rangle$$

  Step 2: For the First training example, hypothesis h is changed as initial h is too specific.

  $$h \leftarrow \langle Sunny, Warm, Normal, Strong, Warm, Same \rangle$$

  Step 3: For the second training example (also positive in this case) forces the algorithm to further generalize *h,* this time substituting a "?" in place of any attribute value in h that is not satisfied by the new example. The refined hypothesis in this case is

  $$h \leftarrow \langle Sunny, Warm, ?, Strong, Warm, Same \rangle$$

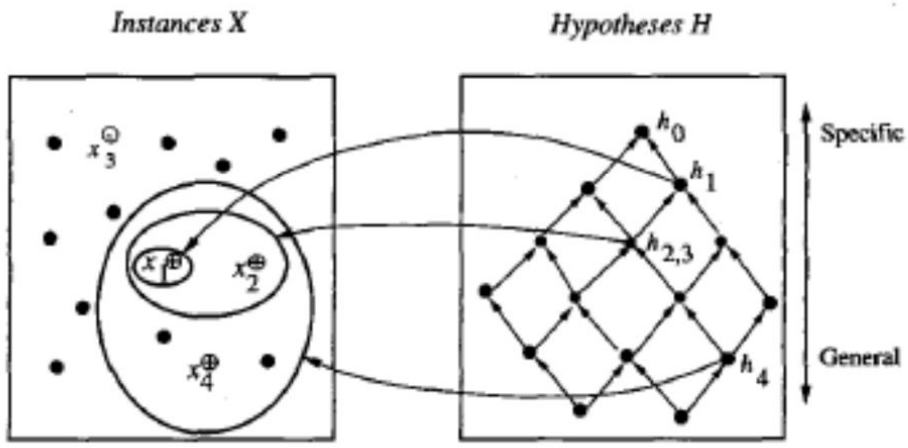
Step 4: With 3rd training example, which is negative in this case the algorithm makes no change to h.

Step 5: With 4th training example, h changes as below:

$h \leftarrow$ <*Sunny, Warm,* ?, *Strong,* ?, ?>

Key property of FIND-S:

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Also the final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.



x_1 = <*Sunny Warm Normal Strong Warm Same*>, +
x_2 = <*Sunny Warm High Strong Warm Same*>, +
x_3 = <*Rainy Cold High Strong Warm Change*>, -
x_4 = <*Sunny Warm High Strong Cool Change*>, +

h_0 = <∅, ∅, ∅, ∅, ∅, ∅>
h_1 = <*Sunny Warm Normal Strong Warm Same*>
h_2 = <*Sunny Warm ? Strong Warm Same*>
h_3 = <*Sunny Warm ? Strong Warm Same*>
h_4 = <*Sunny Warm ? Strong ? ?* >

Limitations of FIND-S:

1. Has FIND-S converged to the correct target concept?

– Although FIND-S will find a hypothesis consistent with the training data, it has no way to determine whether it has found the only hypothesis in H consistent with the data (i.e., the correct target concept), or whether there are many other consistent hypotheses as well.

– We would prefer a learning algorithm that could determine whether it had converged and, if not, at least characterize its uncertainty regarding the true identity of the target concept.

2. Why prefer the most specific hypothesis?

– In case there are multiple hypotheses consistent with the training examples, FIND-S will find the most specific.

– It is unclear whether we should prefer this hypothesis over, say, the most general, or some other hypothesis of intermediate generality.

3. Are the training examples consistent?

– In most practical learning problems there is some chance that the training examples will contain at least some errors or noise.

– Such inconsistent sets of training examples can severely mislead FIND-S, given the fact that it ignores negative examples.

– We would prefer an algorithm that could at least detect when the training data is inconsistent and, preferably, accommodate such errors.

4.What if there are several maximally specific consistent hypotheses?

– In the hypothesis language H for the EnjoySport task, there is always a unique, most specific hypothesis consistent with any set of positive examples.

– However, for other hypothesis spaces there can be several maximally specific hypotheses consistent with the data.

– In this case, FIND-S must be extended to allow it to backtrack on its choices of how to generalize the hypothesis, to accommodate the possibility that the target concept lies along a different branch of the partial ordering than the branch it has selected.

2.5 VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

FIND-S outputs a hypothesis from H, that is consistent with the training examples, this is just one of many hypotheses from H that might fit the training data equally well.

• The key idea in the Candidate-Elimination algorithm is to output a description of the set of all hypotheses consistent with the training examples.

2.5.1 Representation:

– Candidate-Elimination algorithm computes the description of this set without explicitly enumerating all of its members.

– This is accomplished by using the more-general-than partial ordering and maintaining a compact representation of the set of consistent hypotheses.

Consistent hypothesis:

- A hypothesis h is **consistent** with a set of training examples *D* if and only if *h(x) = c(x)* for each example <x, *c(x)*> in *D*.

$$Consistent(h, D) \equiv (\forall \langle x, c(x) \rangle \in D)\ h(x) = c(x)$$

The key difference between this definition of ***consistent*** and ***satisfies.***

• An example *x* is said to ***satisfy*** hypothesis ***h*** when ***h(x)*** = 1, regardless of whether x is a positive or negative example of the target concept.

• However, whether such an example is ***consistent*** with ***h*** depends on the target concept, and in particular, whether ***h(x)*** = ***c(x).***

Version Spaces

• The Candidate-Elimination algorithm represents the set of all hypotheses consistent with the observed training examples.

• This subset of all hypotheses is called the ***version space*** with respect to the hypothesis space H and the training examples D, because it contains all plausible versions of the target concept.

Definition: The **version space,** denoted $VS_{H,D}$ with respect to hypothesis space ***H*** and training examples D, is the subset of hypotheses from ***H*** consistent with the training examples in D.

$$VS_{H,D} \equiv \{h \in H | Consistent(h, D)\}$$

2.5.2 List-Then-Eliminate Algorithm:

• List-Then-Eliminate algorithm initializes the version space to contain all hypotheses in H, then eliminates any hypothesis found inconsistent with any training example.

• The version space of candidate hypotheses thus shrinks as more examples are observed, until ideally just one hypothesis remains that is consistent with all the observed examples.

– Presumably, this is the desired target concept.

– If insufficient data is available to narrow the version space to a single hypothesis, then the algorithm can output the entire set of hypotheses consistent with the observed data.

• List-Then-Eliminate algorithm can be applied whenever the hypothesis space H is finite.

– It has many advantages, including the fact that it is guaranteed to output all hypotheses consistent with the training data.

– Unfortunately, it requires exhaustively enumerating all hypotheses in H - an unrealistic requirement for all but the most trivial hypothesis spaces.

- **The List-Then-Eliminate Algorithm:**
  - ✓ ***VersionSpace*** ← *a* list containing every hypothesis in H
  - ✓ For each training example, <***x, c(x)***>
    - ✓ remove from ***VersionSpace*** any hypothesis ***h*** for which ***h(x) ≠c(x)***
  - ✓ Output the list of hypotheses in ***VersionSpace***

2.5.3 A More Compact Representation for Version Spaces:

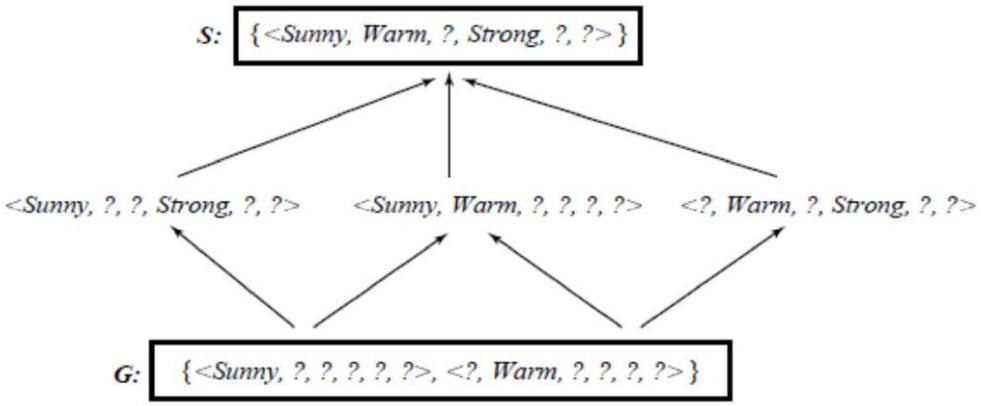
- A version space can be represented with its *general* and *specific boundary sets*.
- The Candidate-Elimination algorithm represents the version space by storing only its most general members G and its most specific members S.
- The **general boundary** G, with respect to hypothesis space H and training data *D,* is the set of maximally general members of *H* consistent with *D.*

$$G \equiv \{g \in H | Consistent(g, D) \wedge (\neg \exists g' \in H)[(g' >_g g) \wedge Consistent(g', D)]\}$$

- The **specific boundary** S, with respect to hypothesis space *H* and training data *D,* is the set of minimally general (i.e., maximally specific) members of *H* consistent with *D.*

$$S \equiv \{s \in H | Consistent(s, D) \wedge (\neg \exists s' \in H)[(s >_g s') \wedge Consistent(s', D)]\}$$

Example of version space:



- A version space with its general and specific boundary sets.

• The version space includes all six hypotheses shown here,
but can be represented more simply by S and ***G.***

- **Version space representation theorem:** Let X be an arbitrary set of instances and let H be a set of Boolean-valued hypotheses defined over X. Let c : $X \rightarrow \{0,1\}$ be an arbitrary target concept defined over X, and let D be an arbitrary set of training examples <x, c(x)>. For all X, H, c, and D such that *S* and *G* are well defined,

$$VS_{H,D} = \{h \in H | (\exists s \in S)(\exists g \in G)(g \geq_g h \geq_g s)\}$$

2.5.4 Candidate-Elimination Learning Algorithm:

The Candidate-Elimination algorithm computes the version space containing all hypotheses from H that are consistent with an observed sequence of training examples.

• It begins by initializing the version space to the set of all hypotheses in H; that is, by initializing the G boundary set to contain the most general hypothesis in H

$G_0 \leftarrow \{ \langle ?, ?, ?, ?, ?, ? \rangle \}$

and initializing the S boundary set to contain the most specific hypothesis

$S_0 \leftarrow \{ \langle 0, 0, 0, 0, 0, 0 \rangle \}$

• These two boundary sets delimit the entire hypothesis space, because every other
hypothesis in H is both more general than S0 and more specific than G0.

• As each training example is considered, the S and G boundary sets are generalized and specialized, respectively, to eliminate from the version space any hypotheses found inconsistent with the new training example.

• After all examples have been processed, the computed version space contains all the hypotheses consistent with these examples and only these hypotheses.

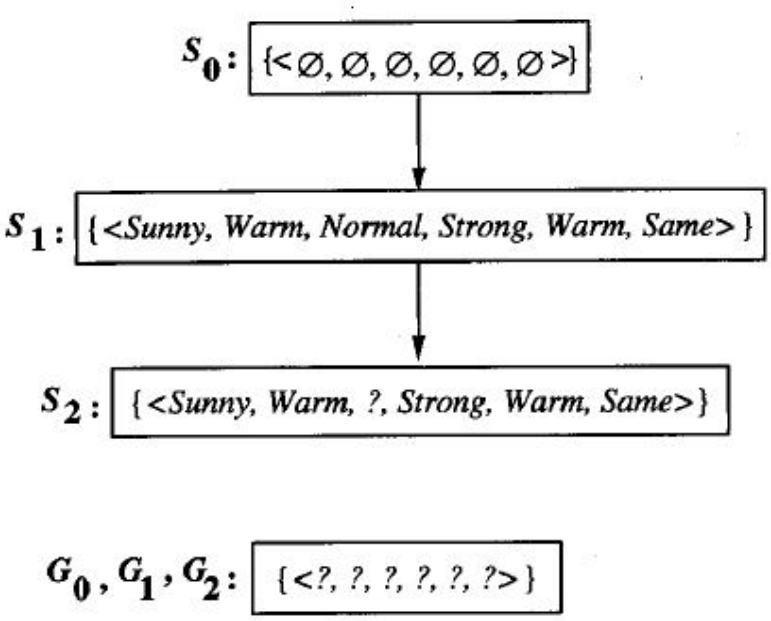
Candidate-elimination algorithm using version spaces:

Initialize G to the set of maximally general hypotheses in H
Initialize S to the set of maximally specific hypotheses in H
For each training example d, do

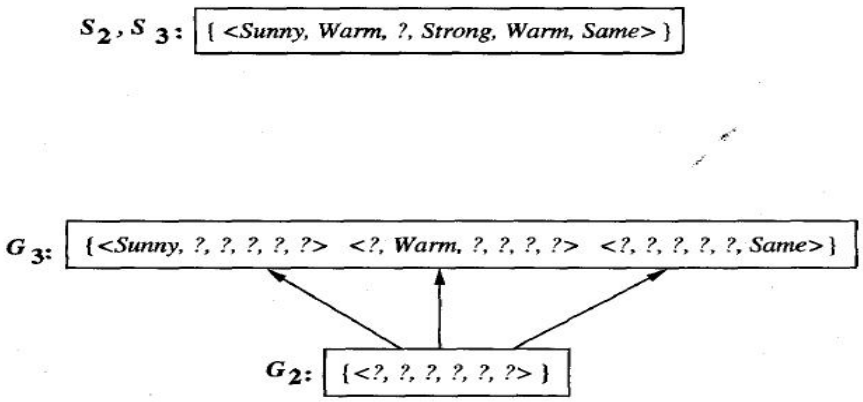
- If d is a positive example
  a. Remove from G any hypothesis inconsistent with d ,
  b. For each hypothesis **s** in S that is not consistent with d ,
     i. Remove **s** from S
     ii. Add to S all minimal generalizations h of **s** such that
        ✓ h is consistent with d, and some member of G is more general than h
  c. Remove from S any hypothesis that is more general than another hypothesis in S
- If d is a negative example
  a. Remove from S any hypothesis inconsistent with d
  b. For each hypothesis g in G that is not consistent with d
     i. Remove g from G
     ii. Add to G all minimal specializations h of g such that
        ✓ h is consistent with d, and some member of S is more specific than h
  c. Remove from G any hypothesis that is less general than another hypothesis in G

Example of Candidate-elimination algorithm using EnjoySport Task:

Trace 1: *S0* and *G0* are the initial boundary sets corresponding to the most specific and most general hypotheses. Training examples 1 and 2 force the *S* boundary to become more general. They have no effect on the *G* boundary. After observing the 1st and 2nd training examples, the S and G sets are as below:



Trace 2: With 3rd training example, the S and G boundaries are as below:



The hypothesis in G incorrectly predicts that 3rd training example is a positive example. The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example. The members of G3 are shown in the above figure. With the six attributes of EnjoySport task, only three are consistent hypothesis w.r.t S2. For example, the hypothesis *h* = <?, ?, *Normal,* ?, ?, ?> is a minimal specialization of G2 that correctly labels 3rd example as a negative example, but it is not included in G3 as it is inconsistent with S2.

Trace 3: With the 4th training example, the S and G boundaries are:

Given that there are six attributes that could be specified to specialize **G2,** why are there only three new hypotheses in **G3?**

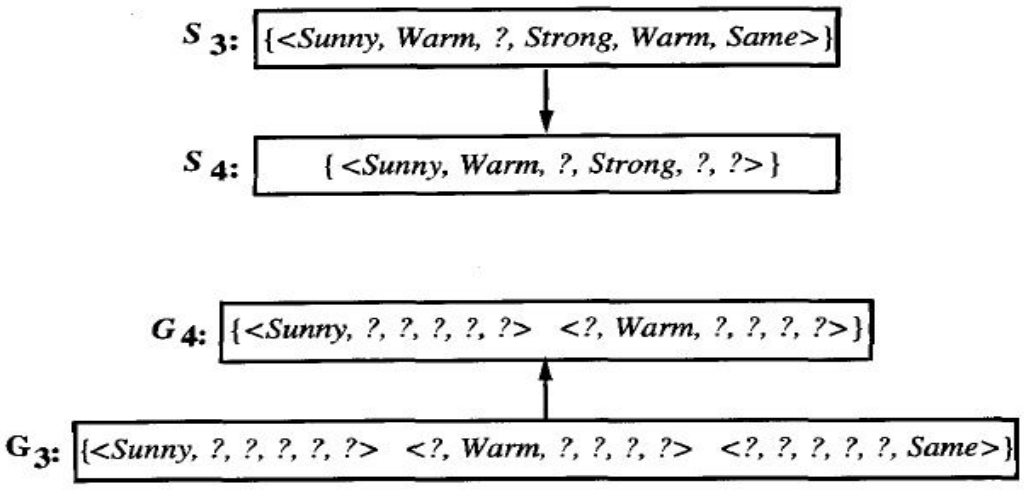
• For example, the hypothesis ***h*** = <?, ?, ***Normal,*** ?, ?, ?> is a minimal specialization of **G2** that correctly labels the new example as a negative example, but it is not included in **G3.**

– The reason this hypothesis is excluded is that it is inconsistent with S2.

– The algorithm determines this simply by noting that ***h*** is not more general than the current specific boundary, **S2.**

• In fact, the **S** boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples.

• The G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples



With the 4th example one member of the G boundary is removed as that member fails to cover the new positive example.

The fourth training example further generalizes the S boundary of the version space.

• It also results in removing one member of the G boundary, because this

member fails to cover the new positive example.

– To understand the rationale for this step, it is useful to consider why the offending hypothesis must be removed from G.

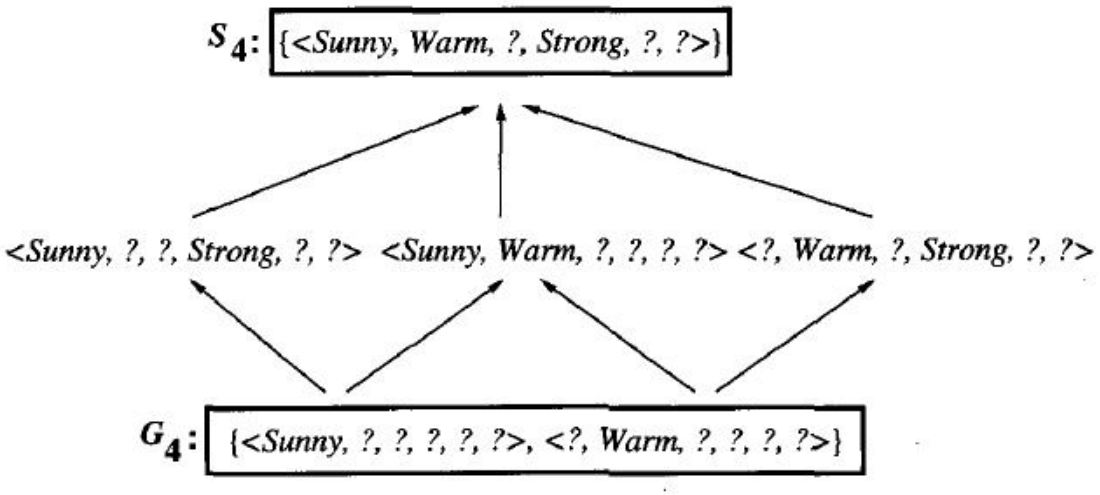
– Notice it cannot be specialized, because specializing it would not make it cover the new example.

– It also cannot be generalized, because by the definition of G, any more general hypothesis will cover at least one negative training example.

– Therefore, the hypothesis must be dropped from the G boundary, thereby removing an entire branch of the partial ordering from the version space of hypotheses remaining under consideration

The final version space of EnjoySport task:

After processing the four examples, the boundary sets **S4** and **G4** delimit the version space of all hypotheses consistent with the set of incrementally observed training examples. The entire version space, including those hypotheses bounded by **S4** and **G4,** is shown in below:



Key points about version space & Candidate-elimination algorithm:

- Positive training examples may force the **S** boundary of the version space to become increasingly general and Negative training examples play the complimentary role of forcing the **G** boundary to become increasingly specific.

- The **S** boundary of the version space forms a summary of the previously encountered positive examples that can be used to determine whether any given hypothesis is consistent with these examples.
- The G boundary summarizes the information from previously encountered negative examples. Any hypothesis more specific than G is assured to be consistent with past negative examples.
- The learned version space is independent of the sequence in which the training examples are presented.
- With more training data, the S and G boundaries will move monotonically closer to each other, delimiting a smaller and smaller version space of candidate hypotheses.

2.6 REMARKS ON VERSION SPACES AND CANDIDATE-ELIMINATION ALGORITHM

a. Will the Candidate-Elimination algorithm converge to the correct Hypothesis?

The version space learned by the Candidate-Elimination Algorithm will converge toward the hypothesis that correctly describes the target concept, provided

– There are no errors in the training examples, and

– there is some hypothesis in H that correctly describes the target concept.

• What will happen if the training data contains errors?

– The algorithm removes the correct target concept from the version space.

– S and G boundary sets eventually converge to an empty version space if sufficient additional training data is available.

– Such an empty version space indicates that there is no hypothesis in H consistent with all observed training examples.

• A similar symptom will appear when the training examples are correct, but the target concept cannot be described in the hypothesis representation.

– e.g., if the target concept is a disjunction of feature attributes and the hypothesis space supports only conjunctive descriptions

b. What Training Example Should the Learner Request Next?

We have assumed that training examples are provided to the learner by some external teacher.

• Suppose instead that the learner is allowed to conduct experiments in which it chooses the next instance, then obtains the correct classification for this instance from an external oracle (e.g., nature or a teacher).

– This scenario covers situations in which the learner may conduct experiments in nature or in which a teacher is available to provide the correct classification.

– We use the term query to refer to such instances constructed by the learner, which are then classified by an external oracle.

• Considering the version space learned from the four training examples of the EnjoySport concept.

– What would be a good query for the learner to pose at this point?

– What is a good query strategy in general?

-The learner should attempt to discriminate among the alternative competing hypotheses in its current version space.

– Therefore, it should choose an instance that would be classified positive by some of these hypotheses, but negative by others.

– One such instance is <Sunny, Warm, Normal, Light, Warm, Same>

– This instance satisfies three of the six hypotheses in the current version space.

– If the trainer classifies this instance as a positive example, the S boundary of the version space can then be generalized.

– Alternatively, if the trainer indicates that this is a negative example, the G boundary can then be specialized.

• In general, the optimal query strategy for a concept learner is to generate instances that satisfy exactly half the hypotheses in the current version space.

Instance	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
A	Sunny	Warm	Normal	Strong	Cool	Change	?
B	Rainy	Cold	Normal	Light	Warm	Same	?
C	Sunny	Warm	Normal	Light	Warm	Same	?
D	Sunny	Cold	Normal	Strong	Warm	Same	?

• When this is possible, the size of the version space is reduced by half with each new example, and the correct target concept can therefore be found with only $\log_2 |VS|$ experiments.

c. How can Partially Learned Concepts be used?

Even though the learned version space still contains multiple hypotheses, indicating that the target concept has not yet been fully learned, it is possible to classify certain examples with the same degree of confidence as if the target concept had been uniquely identified.

Let us assume that the followings are new instances to be classified:

Instance A was is classified as a positive instance by every hypothesis in the current version space.

• Because the hypotheses in the version space unanimously agree that this is a positive instance, the learner can classify instance A as positive with the same confidence it

would have if it had already converged to the single, correct target concept.

• Regardless of which hypothesis in the version space is eventually found to be the correct target concept, it is already clear that it will classify instance A as a positive example.

• Notice furthermore that we need not enumerate every hypothesis in the version space in order to test whether each classifies the instance as positive.

– This condition will be met if and only if the instance satisfies every member of S.

– The reason is that every other hypothesis in the version space is at least as general as some member of S.

– By our definition of more-general-than, if the new instance satisfies all members of S it must also satisfy each of these more general hypotheses.

Instance B is classified as a negative instance by every hypothesis in the version space.

– This instance can therefore be safely classified as negative, given the partially learned concept.

– An efficient test for this condition is that the instance satisfies none of the members of G.

• Half of the version space hypotheses classify ***instance C*** as positive and half classify it as negative.

– Thus, the learner cannot classify this example with confidence until further training examples are available.

• ***Instance D*** is classified as positive by two of the version space hypotheses and negative by the other four hypotheses.

– In this case we have less confidence in the classification than in the unambiguous cases of instances A and B.

– Still, the vote is in favor of a negative classification, and one approach we could take would be to output the majority vote, perhaps with a confidence rating indicating how close the vote was.

2.7 INDUCTIVE BIAS

The **Candidate-Elimination Algorithm** will converge toward the true target concept provided it is given accurate training examples and provided its initial hypothesis space contains the target concept.

Fundamental Questions of Inductive Inference:

- ✓ What if the target concept is not contained in the hypothesis space?
- ✓ Can we avoid this difficulty by using a hypothesis space that includes every possible hypothesis?
- ✓ How does the size of this hypothesis space influence the ability of the algorithm to generalize to unobserved instances?
- ✓ How does the size of the hypothesis space influence the number of training examples that must be observed?

2.7.1 A Biased Hypothesis Space:

- ✓ Suppose we wish to assure that the hypothesis space contains the unknown target concept. The obvious solution is to enrich the hypothesis space to include ***every possible*** hypothesis.
- ✓ To illustrate, consider again the **EnjoySport** example in which we restricted the hypothesis space to include only conjunctions of attribute values.

Example	*Sky*	*AirTemp*	*Humidity*	*Wind*	*Water*	*Forecast*	*EnjoySport*
1	Sunny	Warm	Normal	Strong	Cool	Change	Yes
2	Cloudy	Warm	Normal	Strong	Cool	Change	Yes
3	Rainy	Warm	Normal	Strong	Cool	Change	No

- ✓ Because of this restriction, the hypothesis space is unable to represent even simple disjunctive target concepts such as "***Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy".***
- ✓ In fact, given the three training examples (as in above table) of this disjunctive hypothesis, our algorithm would find that there are zero hypotheses in the version space.
- ✓ To see why there are no hypotheses consistent with these three examples, note that the most specific hypothesis consistent with the first two examples and representable in the given hypothesis space H is

S2 : <?, ***Warm, Normal, Strong, Cool, Change>***

- ✓ This hypothesis, although it is the maximally specific hypothesis from H that is consistent with the first two examples, is already overly general: it incorrectly covers the third (negative) training example.
- ✓ The problem is that we have biased the learner to consider only conjunctive hypotheses. In this case we require a more expressive hypothesis space.

2.7.2 An Unbiased Learner:

- The obvious solution to the problem of assuring that the target concept is in the hypothesis space H is to provide a hypothesis space

capable of representing ***every teachable concept;*** that is, it is capable of representing every possible subset of the instances X. In general, the set of all subsets of a set X is called the ***power set*** of X.

- In our initial ***EnjoySport*** learning task the size of the instance space X described by the six available attributes is 96. So, there are 2^{96} , or approximately 10^{28} distinct target concepts that could be defined over this instance space and that our learner might be called upon to learn.
- But the conjunctive hypothesis space is able to represent only 973 of these. Hence conjunctive hypothesis representation is very biased hypothesis space.
- To make our hypotheses space bias-free, consider the power set of instance space as hypotheses space.
- Then any Hypothesis can be represented with disjunctions, conjunctions, and negations of our earlier hypotheses.
- For instance, the target concept ***"Sky*** = ***Sunny*** or ***Sky*** = ***Cloudy"*** could then be described as

<*Sunny,* ?, ?, ?, ?, ?> v **<*Cloudy,*** ?, ?, ?, ?, ?>

- This way of representing hypothesis space eliminates any problems of expressibility but it unfortunately raises a new, equally difficult problem - our concept learning algorithm is now completely unable to generalize beyond the observed examples.
- For instance, consider three positive examples (xl, x2, x3) and two negative examples (x4, x5) to the learner. Then S : { x1 V x2 V x3 } and G : { ┐(x4 V x5) }
- Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.
- The classification of all the unobserved instances by taking a vote in this power set representation of hypotheses space will be futile as each unobserved instance will be classified positive by precisely half

the hypotheses in the version space and will be classified negative by the other half.

2.7.3 The Futility of Bias-Free Learning:

- The Fundamental property of inductive inference is: "*a learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances".*
- Inductive learning requires some form of prior assumptions, or inductive bias.
- General setting of an arbitrary learning problem:

  A learning algorithm L is provided by an arbitrary set of training data D_c = {<x, c(x)>} of some arbitrary target concept c. After training, L is asked to classify a new instance x_i. Let $L(x_i, D_c)$ denote the classification (e.g., positive or negative) that L assigns to x_i after learning from the training data D_c. We can describe this inductive inference step performed by L as follows:

$$(D_c \wedge x_i) \succ L(x_i, D_c)$$

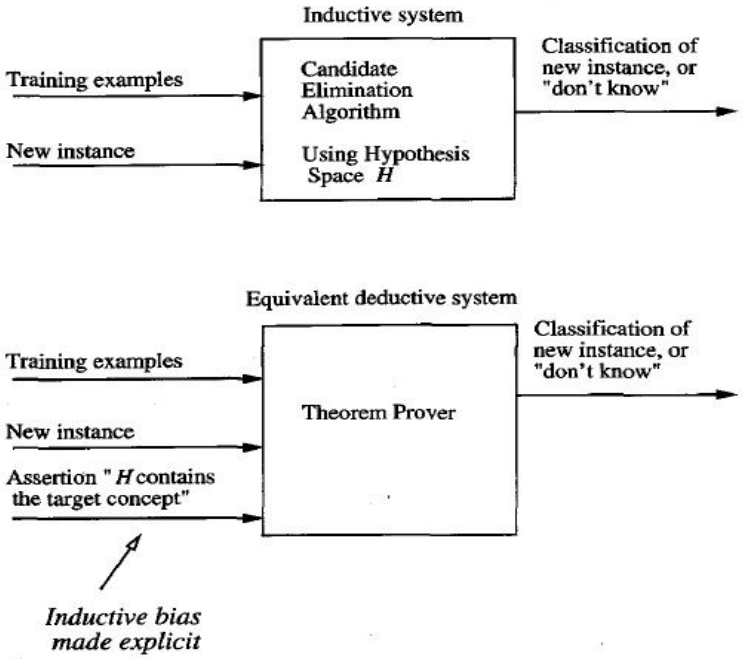
where the notation $y \succ z$ indicates that z is inductively inferred from y .

- **Inductive Bias:**

  Consider a concept learning algorithm ***L*** for the set of instances X. Let c be an arbitrary concept defined over X, and let $\boldsymbol{D_c}$ = {< ***x , c(x)***>} be an arbitrary set of training examples of c. Let ***L(xi, D_c)*** denote the classification assigned to the instance $\boldsymbol{x_i}$ by ***L*** after training on the data $\boldsymbol{D_c}$. The **inductive bias** of ***L*** is any minimal set of assertions ***B*** such that for any target concept c and corresponding training examples $\boldsymbol{D_c}$

$$(\forall x_i \in X)[(B \wedge D_c \wedge x_i) \vdash L(x_i, D_c)]$$

- **Inductive bias of Candidate-Elimination algorithm:** The target concept c is contained in the given hypothesis space H.
- **Below figure shows modelling of Inductive systems by equivalent deductive systems:**



Advantages of viewing inductive inference systems in terms of inductive Bias:

- It provides a nonprocedural means of characterizing inductive inference systems policy for generalizing beyond the observed data.
- It allows comparison of different learners according to the strength of the inductive bias the inference systems employ.

Inductive Bias of Three learning Algorithms (weakest to strongest bias):

- **Rote-Learner:** Learning corresponds simply to storing each observed training example in memory. Subsequent instances are classified by looking them up in memory. If the instance is found in memory, the stored classification is returned. Otherwise, the system refuses to

classify the new instance. The Rote-Learner has no inductive bias and no additional assumptions required for classification.

- **Candidate-Elimination algorithm**: New instances are classified only in the case where all members of the current version space agree on the classification. Otherwise, the system refuses to classify the new instance. The **Candidate-Elimination algorithm** has a stronger inductive bias.
- **FIND-S**: This algorithm, finds the most specific hypothesis consistent with the training examples. It then uses this hypothesis to classify all subsequent instances. The FIND-S algorithm has an even stronger inductive bias. The inductive bias is that the target concept can be represented in its hypothesis space, and all instances are negative instances unless the opposite is entailed by its other know1edge.

UNIT-II
SECTION-A

Objective Questions

1. Consider the following instances and hypothesis. identify the correct statement [ ]

x1= <Sunny, Warm, High, Strong, Cool, Same>
x2= <Sunny, Warm, High, Light, Warm, Same>

hl= <Sunny, ?, ?, Strong, ?, ?>
h2= <Sunny, ?, ?, ?, ?, ?>
h 3 = <Sunny, ?, ?, ?, Cool, ?>

A. h3 is more general than h1
B. h1 is more general than h2
C. h2 is more general than h1
D. All of the above

2. Which of the following is a negative example of EnjoySport learning task? [ ]

A. h=<Sunny, Warm, Normal, Strong, Warm, Same>
B. h=<Sunny, Warm, ?, Strong, Warm, Same>
C. h=<Sunny, Warm,?, Strong, Warm, Same>
D. h=<Sunny, Warm, ?,Strong,?,?>

3. Which of the following is a generalization of h with respect to FIND-S algorithm? [ ]

A. h=<Sunny, Warm, Normal, Strong, Warm, Same>
B. h=<Sunny, Warm, ?, Strong, Warm, Same>
C. h=<Sunny, Warm,?, Strong, Warm, Same>
D.. h=<Sunny, Warm, ?,Strong,?,?>

4.Identify the incorrect statement for FIND-S algorithm. [ ]

A. Can't tell whether it has learned concept.
B. Can't tell when training data consistent.
C. Picks a maximally specific h

D. depending on H, there are several h.

5. Pick the correct hypothesis representation for the task: [ ]
Consider the set of all pairs of people where the first is a tall male(of any nationality and hair color) and the second is a Japanese female(of any hair color and height)

A.<<male ? tall ?><female ? ? Japanese>>

B. <<? ? tall ?><female ? ? Japanese>>

C.<< ? ? tall ?><female ? ? ?>>

D.<<male ? tall ? >< female ? ? ?>>

6. The size of hypothesis space in the given learning task is: Sex(male,female),Clor(black,brown,blonde),Height(tall,medium,short),Nationality(US,French,German,Trish,Indian,Japanese,Portuguese) [ ]

A. 384 B.385 C. 383 D. 386

7. Which of the following learning algorithm has no inductive bias [ ]

A. Rote Learner B. Candidate-Elimination C. FIND-S D. None of the above.

8. Which of the following statement is true about Candidate-elimination algorithm? [ ]

A. Candidate-elimination algorithm uses general-to-specific ordering to find version space.

B. Candidate-elimination algorithm works well on noisy data.

C. Candidate-elimination algorithm searches the hypothesis space completely.

D. Both A and C

SECTION-B

Descriptive Questions

1. What is Concept Learning?

2. Elaborate on Find-S algorithm.

3. Write the steps for LIST-THEN-ELIMINATE algorithm.

4. Write the steps for CANDIDATE_ELIMINATION algorithm using version spaces.

5. Write the expression for defining version spaces.

6. Depict the version space for EnjoySport task with its general and specific boundary sets.

7. Explain why the size of the hypothesis space in the EnjoySport learning task is 973.

8. How would the number of possible instances and possible hypotheses increase with the addition of the attribute WaterCurrent, which can take on the values Light, Moderate or Strong ?

9. How the most specific hypothesis is represented? Explain with an example.

Unit – III
DECISION TREE LEARNING

3.1 INTRODUCTION

- Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.
- Learned trees can also be re-represented as sets of if-then rules to improve human readability. These learning methods are among the most popular of inductive inference algorithms and have been successfully applied to a broad range of tasks from learning to diagnose medical cases to learning to assess credit risk of loan applicants.

3.2 DECISION TREE REPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance. Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node. Figure *1* illustrates a typical learned decision tree. This decision tree classifies Saturday mornings according to whether they are suitable for playing tennis. For example, the instance

(Outlook = ***Sunny, Temperature*** = ***Hot, Humidity*** = ***High, Wind*** = ***Strong)***

would be sorted down the leftmost branch of this decision tree and would therefore be classified as a negative instance (i.e., the tree predicts that ***PlayTennis*** = ***no).***

- In general, decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances. Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions.
- For example, the decision tree shown in Figure **1** corresponds to the expression

(Outlook = ***Sunny*** □ ***Humidity*** = ***Normal)***
V ***(Outlook*** = ***Overcast)***
v ***(Outlook*** = ***Rain*** □ ***Wind*** = ***Weak)***

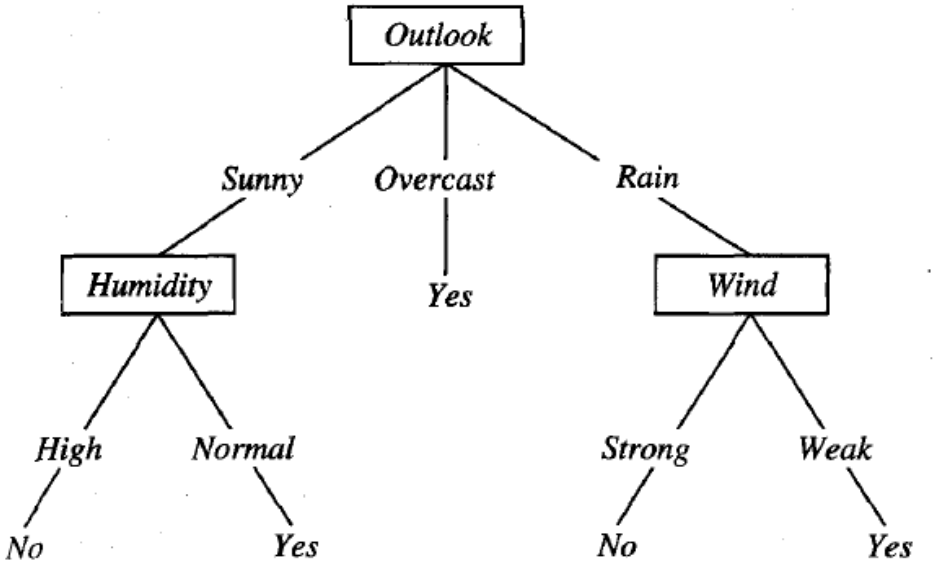


Fig 1: A decision tree for the concept *PlayTennis*.

3.3 APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

Decision tree learning is generally best suited to problems with the following characteristics:

a. Instances are represented by attribute-value pairs. Instances are described by a fixed set of attributes (e.g., ***Temperature)*** and their values (e.g., ***Hot).*** The easiest situation for decision tree learning is when each attribute takes on a small number of disjoint possible values (e.g., ***Hot, Mild, Cold).***

b. The targetfunction has discrete output values. The decision tree in figure 1 assigns a boolean classification (e.g., ***yes*** or ***no)*** to each example. Decision tree methods easily extend to learning functions with more than two possible output values. A more substantial extension allows learning target functions with real-valued outputs, though the application of decision trees in this setting is less common.

c. Disjunctive descriptions may be required. As noted above, decision trees naturally represent disjunctive expressions.

d. The training data may contain errors. Decision tree learning methods are robust to errors, both errors in classifications of the training examples and errors in the attribute values that describe these examples.

e. The training data may contain missing attribute values. Decision tree methods can be used even when some training examples have unknown values (e.g., if the ***Humidity*** of the day is known for only some of the training examples).

3.4 THE BASIC DECISION TREE LEARNING ALGORITHM

- Many algorithms have been developed for constructing the decision trees.
- The basic decision tree learning algorithm is ID3 which follows the top-down approach for constructing the decision tree.

3.1 Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- For this, ID3 algorithm uses information gain measure. The attribute with the highest information gain is chosen for testing at a node.
- ***Information gain*** measures how well a given attribute separates the training examples according to the target classification.
- In order to define information gain precisely, a measure called ***entropy*** is used.
- Entropy measures the impurity of an arbitrary collection of samples. (i.e. it measures the homogeneity of samples).
- Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

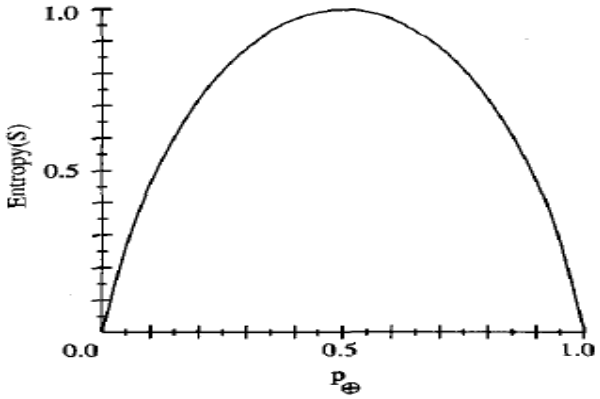
$$Entropy(S) \equiv -p_{\oplus} \log_2 p_{\oplus} - p_{\ominus} \log_2 p_{\ominus}$$

where $p_{\oplus}$, is the proportion of positive examples in S and $p_{\ominus}$, is the proportion of negative examples in S.

- To illustrate, suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this Boolean classification is

$$Entropy([9+, 5-]) = -(9/14) \log_2(9/14) - (5/14) \log_2(5/14)$$
$$= 0.940$$

- Notice that the **entropy is 0** if all members of S belong to the same class. Also note that the **entropy is 1** when the collection contains an equal number of positive and negative examples.
- If the collection contains unequal numbers of positive and negative examples, the entropy is between **0** and 1. The following figure shows this.



- If the target classification has c classes, then the entropy of ***S*** relative to this c-wise classification is defined as

$$Entropy(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i$$

- where p_i is the proportion of ***S*** belonging to class ***i.***
- Having been defined entropy, now we can define the information gain, ***Gain(S, A)*** of an attribute A, relative to a collection of samples ***S*** as

$$Gain(S, A) - Entropy(S) - Entropy_A(S)$$

Where $Entropy_A(S)$ is given as

$$\sum_{v \in Values(A)} \frac{|S_v|}{|S|} Entropy(S_v)$$

where ***Values(A)*** is the set of all possible values for attribute A, and ***S,*** is the subset of S for which attribute A has value v.

- As an example, let us construct the decision tree for the following data which shows training examples for the target concept ***PlayTennis***.

Day	*Outlook*	*Temperature*	*Humidity*	*Wind*	*PlayTennis*
D1	Sunny	Hot	High	Weak	No
D2	Sunny	Hot	High	Strong	No
D3	Overcast	Hot	High	Weak	Yes
D4	Rain	Mild	High	Weak	Yes
D5	Rain	Cool	Normal	Weak	Yes
D6	Rain	Cool	Normal	Strong	No
D7	Overcast	Cool	Normal	Strong	Yes
D8	Sunny	Mild	High	Weak	No
D9	Sunny	Cool	Normal	Weak	Yes
D10	Rain	Mild	Normal	Weak	Yes
D11	Sunny	Mild	Normal	Strong	Yes
D12	Overcast	Mild	High	Strong	Yes
D13	Overcast	Hot	Normal	Weak	Yes
D14	Rain	Mild	High	Strong	No

First, compute ***Entropy(S)*** [S is the given data set. There are two classes yes = 9, No = 5].

$$Entropy([9+, 5-]) = -(9/14)\log_2(9/14) - (5/14)\log_2(5/14)$$
$$= 0.940$$

Next, compute the information gain of each attribute in the data set.
$Entropy_{Outlook}(S)$ is given as

$$\sum_{v \in Values(Outlook)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{5}{14}\left(-\frac{3}{5}log\frac{3}{5} - \frac{2}{5}log\frac{2}{5}\right) + \frac{4}{14}\left(-\frac{4}{4}log\frac{4}{4}\right) + \frac{5}{14}\left(-\frac{3}{5}log\frac{3}{5} - \frac{2}{5}log\frac{2}{5}\right)$$

$= 0.694$

Therefore

$$Gain(S, Outlook) = Entropy(S) - Entropy_{Outlook}(S)$$
$$= 0.940 - 0.694$$
$$= 0.246$$

Similarly,
$Entropy_{Temperature}(S)$ is given as

$$\sum_{v \in Values(Temperature)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{4}{14}\left(-\frac{2}{4}log\frac{2}{4}-\frac{2}{4}log\frac{2}{4}\right)+\frac{6}{14}\left(-\frac{4}{6}log\frac{4}{6}-\frac{2}{6}log\frac{2}{6}\right)+\frac{4}{14}\left(-\frac{3}{4}log\frac{3}{4}-\frac{1}{4}log\frac{1}{4}\right)$$

$= 0.911$

Therefore

$$Gain(S, Temperature) = Entropy(S) - Entropy_{Temperature}(S)$$

$$= 0.940 - 0.911$$

$$= 0.029$$

Similarly,

$Entropy_{Humidity}(S)$ is given as

$$\sum_{v \in Values(Humidity)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$-\frac{7}{14}\left(-\frac{3}{7}log\frac{3}{7}-\frac{4}{7}log\frac{4}{7}\right)+\frac{7}{14}\left(-\frac{6}{7}log\frac{6}{7}-\frac{1}{7}log\frac{1}{7}\right)$$

$= 0.789$

Therefore

$$Gain(S, Humidity) - Entropy(S) - Entropy_{Humidity}(S)$$

$$= 0.940 - 0.789$$

$$= 0.151$$

Similarly,

$Entropy_{Wind}(S)$ is given as

$$\sum_{v \in Values(Wind)} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= \frac{8}{14}\left(-\frac{6}{8}log\frac{6}{8}-\frac{2}{8}log\frac{2}{8}\right)+\frac{6}{14}\left(-\frac{3}{6}log\frac{3}{6}-\frac{3}{6}log\frac{3}{6}\right)$$

$= 0.892$

Therefore

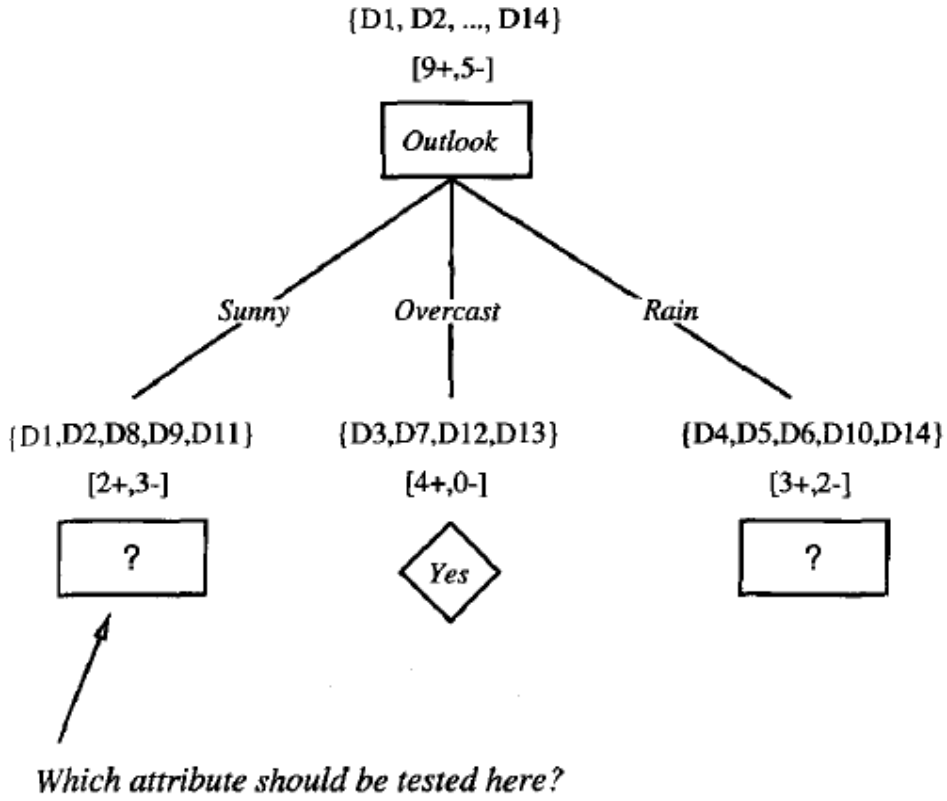
$$Gain(S, Wind) = Entropy(S) - Entropy_{Wind}(S)$$

$$= 0.940 - 0.892$$

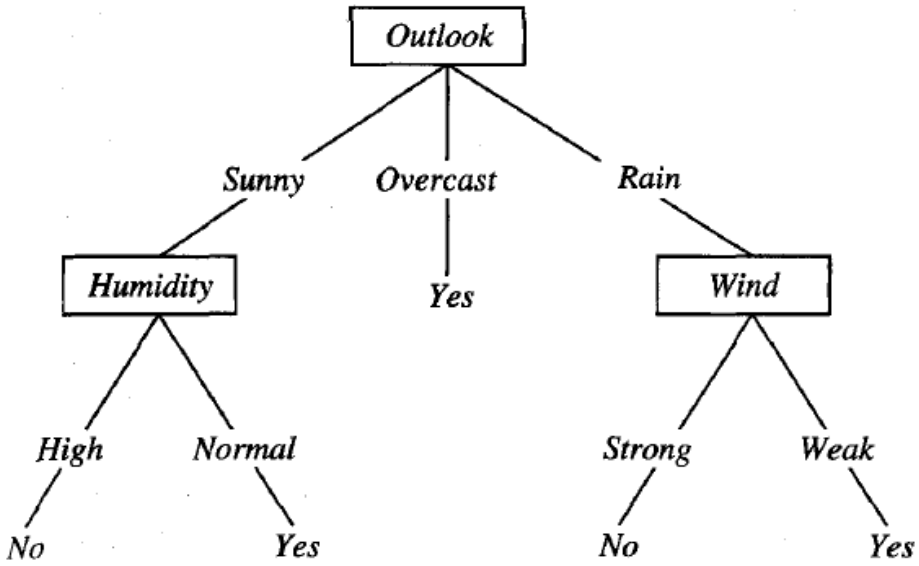
$$= 0.048$$

Since, *Gain(S, Outlook)* is high so, the attribute *Outlook* is tested first and becomes the root for the decision tree. It then classifies S into three

partitions [Let us say S1:leftmost subtree, S2:middle subtree, S3:rightmost subtree] (*because it has three different values: Sunny, Overcast, Rainy*) as shown below.



Now, we need to apply the same procedure to decide the root node for the leftmost subtree and for the rightmost subtree. However, observe that all the samples of middle tree are related to the same class (*Yes class*). So, we can create a leaf node here and label it with ***Yes.*** The final decision tree for the given data set, S, is given below.



After understanding this procedure, now we are ready to design the algorithm for the basic decision tree learning algorithm (ID3). This is given below.

ID3(*(Examples, Targetattribute, Attributes)*

- Create a *Root* node for the tree
- If all *Examples* are positive, Return the single-node tree *Root*, with label = +
- If all *Examples* are negative, Return the single-node tree *Root*, with label = −
- If *Attributes* is empty, Return the single-node tree *Root*, with label = most common value of *Target_attribute* in *Examples*
- Otherwise Begin
  - *A* ← the attribute from *Attributes* that best* classifies *Examples*
  - The decision attribute for *Root* ← *A*
  - For each possible value, v_i, of *A*,
    - Add a new tree branch below *Root*, corresponding to the test $A = v_i$
    - Let $Examples_{v_i}$ be the subset of *Examples* that have value v_i for *A*
    - If $Examples_{v_i}$ is empty
      - Then below this new branch add a leaf node with label = most common value of *Target_attribute* in *Examples*
      - Else below this new branch add the subtree ID3($Examples_{v_i}$, *Target_attribute*, *Attributes* − {*A*}))
- End
- Return *Root*

3.5. HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.

- The hypothesis space searched by ID3 is the set of possible decision trees. ID3 performs a simple-to complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data. The evaluation function that guides this hill-climbing search is the information gain measure.
- This search is depicted in the figure.

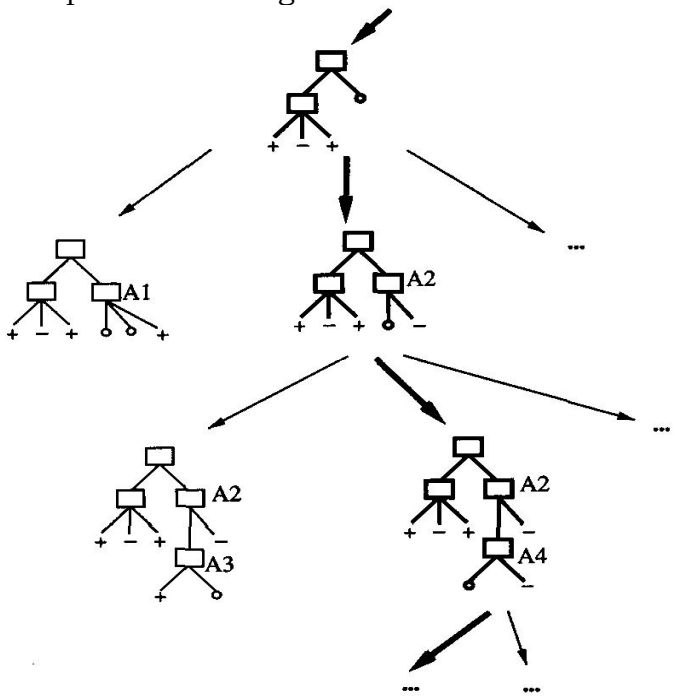


Fig: Hypothesis space search in decision tree learning

- In the above diagram, the search is denoted by dark lines.
- The search strategy has the following capabilities and limitations.
  - ID3's hypothesis space of all decision trees is a ***complete*** space relative to the available attributes.
  - ID3 maintains only a single current hypothesis as it searches through the space of decision trees.
  - ID3 in its pure form performs no backtracking in its search. Once it, selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice. Therefore, it is susceptible to the usual risks locally optimal solutions that are not globally optimal.
  - ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis.

3.6. INDUCTIVE BIAS IN DECISION TREE LEARNING

- Given a collection of training examples, there are typically many decision trees consistent with these examples.
- Describing the inductive bias of ID3 therefore consists of describing the basis by which it chooses one of these consistent hypotheses over the others. Which of these decision trees does ID3 choose?
- It chooses the first acceptable tree it encounters in its search through the space of possible trees.
- Roughly speaking, then, the ID3 search strategy (a) selects in favour of shorter trees over longer ones, and (b) selects trees that place the attributes with highest information gain closest to the root.
- However, we can approximately characterize its bias as a preference for short decision trees over complex trees.

Approximate inductive bias of ID3: Shorter trees are preferred over larger trees.

- In particular, it does not always find the shortest consistent tree, and it is biased to favor trees that place attributes with high information gain closest to the root.

A closer approximation to the inductive bias of ID3: Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

3.6.1 Restriction Biases and Preference Biases

Restriction Bias – is a bias where there are restrictions on the selection of hypothesis among all possible hypotheses. For example, Candidate-Elimination algorithm has restriction bias.

Preference Bias - is a bias where there are no restrictions on the selection of hypothesis among all possible hypotheses. For example, ID3 has preference bias.

The difference between the hypothesis space search in these two approaches:

ID3 searches a complete hypothesis space (i.e., one capable of expressing any finite discrete-valued function). It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met (e.g., until it finds a hypothesis consistent with the data). Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy. Its hypothesis space introduces no additional bias.

The version space CANDIDATE-ELIMINATION algorithm searches an incomplete hypothesis space (i.e., one that can express only a subset of the potentially teachable concepts). It searches this space completely, finding every hypothesis consistent with the training data. Its inductive bias is solely a consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias.

3.6.2 Why Prefer Short Hypotheses?

- Is ID3's inductive bias favouring shorter decision trees a sound basis for generalizing beyond the training data? Occam discussed this question and it is called as Occam's razor.

Occam's razor: Prefer the simplest hypothesis that fits the data.

- Then why prefer shorter hypothesis? Because, for the given training data, there are fewer short hypotheses whereas there are a large number of complex hypotheses that fit the training data.

There are many more 500-node decision trees than 5-node decision trees. Given a small set of 20 training examples, we might expect to be able to find many 500-node decision trees consistent with these, whereas we would be more surprised if a 5-node decision tree could perfectly fit this data. We might therefore believe the 5-node tree is less likely to be a statistical coincidence and prefer this hypothesis over the 500-node hypothesis.
A second problem with the above argument for Occam's razor is that the size of a hypothesis is determined by the particular representation used ***internally*** by the learner. Two learners using different internal representations could therefore anive at different hypotheses, both justifying their contradictory conclusions by Occam's razor!.

3.7 ISSUES IN DECISION TREE LEARNING

Practical issues in learning decision trees include determining how deeply to grow the decision tree, handling continuous attributes, choosing an appropriate attribute selection measure, handling training data with missing attribute values, handling attributes with differing costs, and improving computational efficiency. The issues are listed below:

3.7.1 Avoiding Overfitting the Data

The algorithm described in ID3 grows each branch of the tree just deeply enough to perfectly classify the training examples. While this is ometimes a reasonable strategy, in fact it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. In either of these cases, this simple algorithm can produce trees that ***overfit*** the training examples.

We will say that a hypothesis overfits the training examples if some other hypothesis that fits the training examples less well actually performs better over the entire distribution of instances (i.e., including instances beyond the training set).

Definition: **Given a hypothesis space H, a hypothesis *h E* H is said to overfit the training data if there exists some alternative hypothesis *h'* E H, such that *h* has smaller error than *h'* over the training examples, but *h'* has a smaller error than *h* over the entire distribution of instances.**

Figure illustrates the impact of overfitting in a typical application of decision tree learning. In this case, the ID3 algorithm is applied to the task of learning which medical patients have a form of diabetes. The horizontal axis of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree.

The solid line shows the accuracy of the decision tree over the training examples, whereas the broken line shows accuracy measured over an independent set of test examples (not included in the training set). Predictably, the accuracy of the tree over the training examples increases monotonically as the tree is grown. However, the accuracy measured over the independent test examples first increases, then decreases.

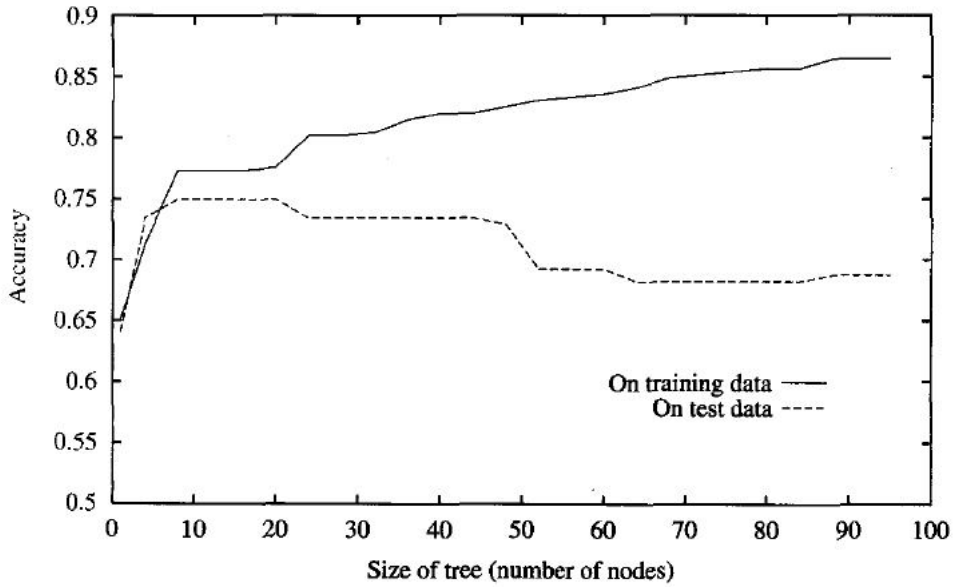


Fig: Overfitting in decision tree learning

There are several approaches to avoiding overfitting in decision tree learning.

These can be grouped into two classes:

- approaches that stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data,
- approaches that allow the tree to overfit the data, and then post-prune the tree.

3.7.1.1 REDUCED ERROR PRUNING

Reduced-error pruning (Quinlan 1987), considers each of the decision nodes in the tree to be candidates for pruning. Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node.

Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set. This has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set.

Nodes are pruned iteratively, always choosing the node whose removal most increases the decision tree accuracy over the validation set. Pruning of nodes continues until further pruning is harmful.

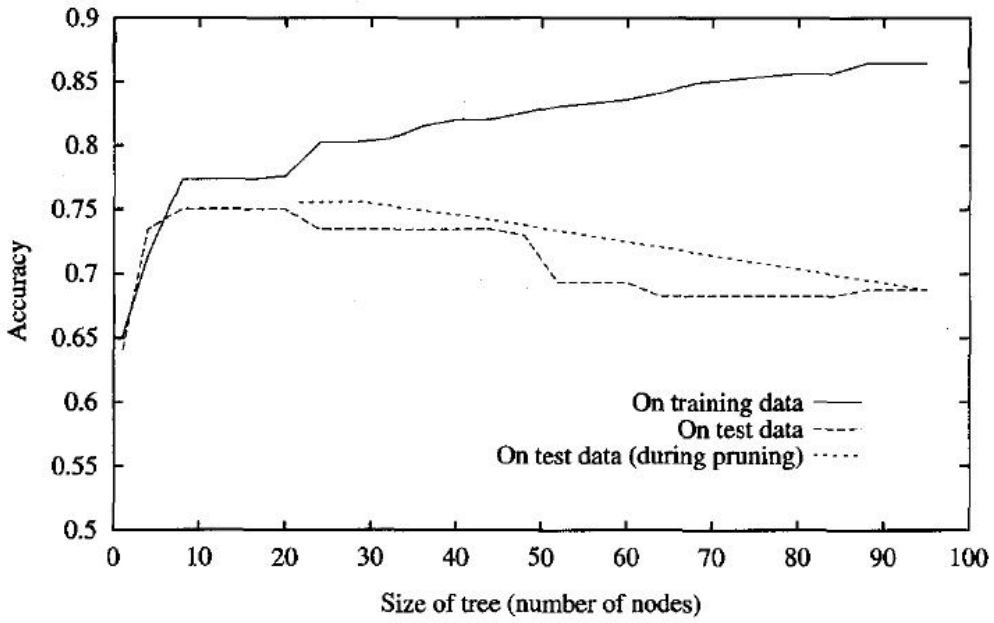


Fig: Effect of reduced-error pruning in decision tree learning

3.7.1.2RULE POST-PRUNING

Rule post-pruning involves the following steps:

a. Infer the decision tree from the training set, growing the tree until the training
data is fit as well as possible and allowing overfitting to occur.

b. Convert the learned tree into an equivalent set of rules by creating one rule
for each path from the root node to a leaf node.

c. Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.

d. Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Why convert the decision tree to rules before pruning? There are three main advantages.

- Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path. In contrast, if the tree itself were pruned, the only two choices would be to remove the decision node completely, or to retain it in its original form.
- Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, we avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.

- Converting to rules improves readability. Rules are often easier for to understand.

3.7.2 Incorporating Continuous-Valued Attributes

Our initial definition of ID3 is restricted to attributes that take on a discrete set of values.

First, the target attribute whose value is predicted by the learned tree must be discrete valued. Second, the attributes tested in the decision nodes of the tree must also be discrete valued. This second restriction can easily be removed so that continuous-valued decision attributes can be incorporated into the learned tree. This can be accomplished by dynamically defining new discretevalued attributes that partition the continuous attribute value into a discrete set of intervals. In particular, for an attribute A that is continuous-valued, the algorithm can dynamically create a new boolean attribute A, that is true if A < c and false otherwise. The only question is how to select the best value for the threshold c.

As an example, suppose we wish to include the continuous-valued attribute ***Temperature*** in describing the training example days in the learning task of play tennis.

Suppose further that the training examples associated with a particular node in the decision tree have the following values for ***Temperature*** and the target attribute ***PlayTennis.***

Temperature:	40	48	60	72	80	90
PlayTennis:	No	No	Yes	Yes	Yes	No

What threshold-based boolean attribute should be defined based on Temperature?

Clearly, we would like to pick a threshold, ***c,*** that produces the greatest information gain. By sorting the examples according to the continuous attribute **A,** then identifying adjacent examples that differ in their target classification, we can generate a set of candidate thresholds midway between the corresponding values of ***A.***

UNIT-III
SECTION-A

Objective Questions

1. Decision Tree is a display of an algorithm. [ ]

A. True B. False

2. Decision Trees can be used for classification tasks. [ ]

A. True B. False

3. When a decision tree is grown to full depth, it is more likely to fit the noise in the data. [ ]

A. True B. False

4. For which of the following hyper parameters, higher value is better for decision tree algorithm? [ ]

1. Number of samples used for split
2. Depth of tree
3. Samples for leaf

A.1 and 2 B. 2 and 3 C. 1 and 3 D. 1, 2 and 3

4. Below are some assumptions that we made while using decision tree. Identify the wrong statement.

[ ]

A. At the beginning, we consider the whole training set as the child node
B. Feature values are preferred to be categorical. If the values are continuous then they are discretized prior to building the model.
C. On the basis of attribute values records are distributed recursively.
D. We use statistical methods for ordering attributes as root or the internal node.

6. In which of the following scenario a gain ratio is preferred over Information Gain? [ ]

A. When a categorical variable has very large number of category
B. When a categorical variable has very small number of category
C. Number of categories is the not the reason
D. None of these

7. Below are the 8 actual values of target variable in the train file:[0,0,0,1,1,1,1,1].What is the entropy of the target variable? [ ]

A. -(5/8 log(5/8) + 3/8 log(3/8)) B. 5/8 log(5/8) + 3/8 log(3/8)

C. 3/8 log(5/8) + 5/8 log(3/8) D. 5/8 log(3/8) – 3/8 log(5/8)

8. Decision Tree is [ ]

A. Flow-Chart

B. Structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label

C. Flow-Chart & Structure in which internal node represents test on an attribute, each branch represents outcome of test and each leaf node represents class label

D. None of the mentioned

9. Suppose S is a collection of 14 examples of some Boolean concept, including 9 positive and 5 negative examples (we adopt the notation [9+, 5-] to summarize such a sample of data). Then the entropy of S relative to this boolean classification is [ ]

A.0.940 B.0.80 C.0.70 D.0.89

10. Suppose *S* is a collection of training-example days described by attributes including *Wind,* which can have the values *Weak* or *Strong.* Also consider, *S* is a collection containing *14* examples, [9+, 5-]. Of these 14 examples, suppose 6 of the positive and 2 of the negative examples have *Wind = Weak,* and the remainder have *Wind = Strong. What is the* information gain due to sorting the original *14* examples by the attribute *Wind ?* [ ]

A.0.045 B.0. C.0.038 D.0.035

11. Indentify wrong statement in the issues of decision tree learning [ ]

A. reduced error pruning

B. rule pre-pruning

C. incorporating continuous-valued attributes

D. handling training examples with missing attribute values

SECTION-B

Descriptive Questions

1. Explain basic decision tree algorithm.
2. Explain how hypothesis space search is carried in decision tree learning.
3. Discuss any three issues in decision tree learning.
4. What do you mean by Gain and Entropy? How is it used to build the Decision tree in algorithm? Illustrate using an example
5. What is the procedure of building decision tree using ID3 algorithm with gain and entropy. Illustrate with example.
6. Explain inductive bias in decision tree learning.
7. Explain various attributes selection measures for constructing a decision tree.
8. How is a decision tree pruned?
9. Consider the following set of training example:

Instance	Classification	a1	a2
1	+	T	T
2	+	T	T
3	-	T	F
4	+	F	F
5	-	F	T
6	-	F	T

   a. what is the entropy of this collection of training example with respect to target function classification.

   b. What is information gain of a2 relative to above training example.

10. What is Occam's razor?

UNIT-IV

BAYESIAN LEARNING

1. INTRODUCTION

Bayesian learning provides the basis for learning algorithms that directly manipulate probabilities.

Features of Bayesian learning methods

- Each observed training example can incrementally decrease or increase the estimated probability of the correct hypothesis.
- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions (e.g., hypotheses such as "this pneumonia patient has a 93% chance of complete recovery").
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- Even in cases where Bayesian methods prove computationally intractable, they can provide a standard of optimal decision making against which other practical methods can be measured.

2. BAYES THEOREM

Bayes theorem states that

$$P(h|D) = \frac{P(D|h)P(h)}{P(D)}$$

Where $P(h|D)$ denotes the posteriori probability of h given D $P(D|h)$ denotes the posteriori probability of D given h $P(h)$ denotes the prior probability of h and $P(D)$ denotes the prior probability of D

- In many learning scenarios, the learner considers some set of candidate hypotheses **H** and is interested in finding the most probable hypothesis ***h ϵ H*** given the observed data ***D.***
- The maximally probable hypothesis is called a ***maximum*** a ***posteriori*** (MAP) hypothesis. We can determine the MAP hypotheses (***hMAP)*** by using Bayes theorem as follows:

$$h_{MAP} \equiv \underset{h \in H}{\operatorname{argmax}} P(h|D)$$

$$= \underset{h \in H}{\operatorname{argmax}} \frac{P(D|h)P(h)}{P(D)}$$

$$= \underset{h \in H}{\operatorname{argmax}} P(D|h)P(h)$$

Notice in the final step we dropped the term ***P(D)*** because it is a constant for all ***h.***

In some cases, we will assume that every hypothesis in H is equally probable. In this case we can further simplify Equation (2) and need only consider the term ***P(D|h)*** to find the most probable hypothesis. ***P(D|h)*** is often called the ***likelihood*** of the data ***D*** given ***h,*** and any hypothesis that maximizes ***P(D|h)*** is called a ***maximum likelihood*** (ML) hypothesis, h_{ML}.

$$h_{ML} \equiv \underset{h \in H}{\operatorname{argmax}} P(D|h)$$

An Example:

To illustrate Bayes theorm, consider a medical diagnosis problem in which there are two alternative hypotheses: (1) the patient has cancer and (2) the patient has no cancer. The avaiiable data is from a particular laborator test with two possible outcomes: (positive) and (negative). We have prior knowledge that over the entire population of people only 0.008 have this disease. Furthermore, the test returns a correct positive result in only 98% of the cases in which the disease is actually present and a correct negative

result in only 97% of the cases in which the disease is not present. In other cases, the test returns the opposite result. Suppose we now observe a new patient for whom the lab test returns a positive result. Should we diagnose the patient as having cancer or not? **Sol:** The above situation can be summarized by the following probabilities:

$$P(cancer) = .008, \quad P(\neg cancer) = .992$$
$$P(\oplus|cancer) = .98, \quad P(\ominus|cancer) = .02$$
$$P(\oplus|\neg cancer) = .03, \quad P(\ominus|\neg cancer) = .97$$

The maximum a posteriori hypothesis can be found using Equation (2):

$$P(\oplus|cancer)P(cancer) = (.98).008 = .0078$$
$$P(\oplus|\neg cancer)P(\neg cancer) = (.03).992 = .0298$$

$$\text{Thus, } h_{MAP} = \neg cancer.$$

This means that, the patient has no cancer.

3. MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

In the previous section we saw the maximum likelihood hypothesis which is given as:

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}}\, p(D|h)$$

Here, we assume that the probability distribution is a normal distribution. A Normal distribution is a smooth, bell-shaped distribution that can be completely characterized by its mean μ and its standard deviation σ^2.

We assume a fixed set of training instances ***(x1 . . . xm)*** and therefore consider the data D to be the corresponding sequence of target values ***D = (d1 . . .dm).*** Then we can write ***P(D|h)*** as the product of the various ***p(di|hi)***

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} p(d_i|h)$$

Since, we are assuming the normal distribution for the probabilities, the above equation can be written as

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-\mu)^2}$$

$$= \underset{h \in H}{\operatorname{argmax}} \prod_{i=1}^{m} \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i-h(x_i))^2}$$

Applying logarithm to the above equation, we get

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} \ln \frac{1}{\sqrt{2\pi\sigma^2}} - \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

The first term in this expression is a constant independent of ***h,*** and can therefore be discarded, yielding

$$h_{ML} = \underset{h \in H}{\operatorname{argmax}} \sum_{i=1}^{m} -\frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Maximizing this negative quantity is equivalent to minimizing the corresponding
positive quantity. Hence, above equation becomes

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} \frac{1}{2\sigma^2}(d_i - h(x_i))^2$$

Finally, we can again discard constants that are independent of ***h*** to get

$$h_{ML} = \underset{h \in H}{\operatorname{argmin}} \sum_{i=1}^{m} (d_i - h(x_i))^2$$

Thus, the above equation shows that the maximum likelihood hypothesis ***hML*** the one that minimizes the sum of the squared errors between the observed training values ***di*** and the hypothesis predictions ***h(xi).***

4. BAYES OPTIMAL CLASSIFIER

- Before defining the Bayes optimal classifier, let us consider a hypothesis space containing three hypotheses, hl, h2, and h3. Suppose that the posterior probabilities of these hypotheses given the training data are **0.**4, **0.3,** and **0.3** respectively.
- Thus, hl is the MAP hypothesis.
- Suppose a new instance x is encountered, which is classified positive by ***h1,*** but negative by ***h2*** and ***h3***.
- Taking all hypotheses into account, the probability that x is positive is 0.4 (the probability associated with ***h1)***, and the probability that it is negative is therefore 0.6. The most probable classification (negative) in this case is different from the classification generated by the MAP hypothesis.
- In general, the most probable classification of the new instance is obtained by combining the predictions of all hypotheses, weighted by their posterior probabilities.
- If the possible classification of the new example can take on any value ***vj*** from some set V, then the probability P(vj | D) is given as

$$P(v_j|D) = \sum_{h_i \in H} P(v_j|h_i)P(h_i|D)$$

The optimal classification of the new instance is the value ***vj,*** for which P (vj | D) is maximum.

Thus Bayes optimal classifier is defined as

$$\underset{v_j \in V}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i) P(h_i|D)$$

To illustrate in terms of the above example, the set of possible classifications of the new instance is

$$V = \{\oplus, \ominus\}$$

and

$$P(h_1|D) = .4, \quad P(\ominus|h_1) = 0, \quad P(\oplus|h_1) = 1$$

$$P(h_2|D) = .3, \quad P(\ominus|h_2) = 1, \quad P(\oplus|h_2) = 0$$

$$P(h_3|D) = .3, \quad P(\ominus|h_3) = 1, \quad P(\oplus|h_3) = 0$$

Therefore

$$\sum_{h_i \in H} P(\oplus|h_i) P(h_i|D) = .4$$

$$\sum_{h_i \in H} P(\ominus|h_i) P(h_i|D) = .6$$

And

$$\underset{v_j \in \{\oplus, \ominus\}}{\operatorname{argmax}} \sum_{h_i \in H} P(v_j|h_i) P(h_i|D) = \ominus$$

Any system that classifies new instances according to Equation (5.1) is called a ***Bayes*** optimal ***classifier,*** or Bayes optimal learner.

5. BAYESIAN BELIEF NETWORKS (BBN)

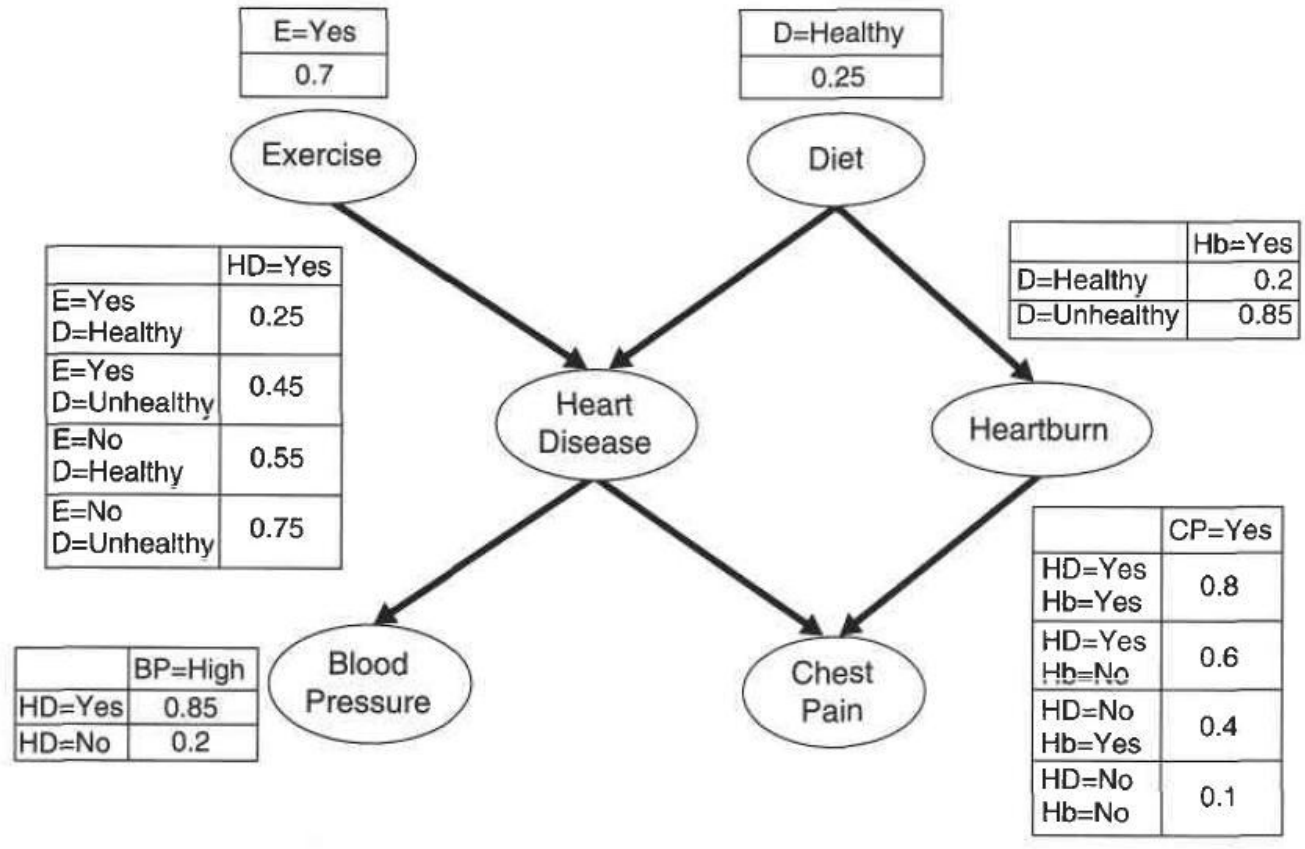
- The conditional independence assumption made by naive Bayes classifiers may seem too rigid, especially for classification problems in which the attributes are somewhat correlated (i.e. depended).
- Instead of requiring all the attributes to be conditionally independent given the class, this approach allows us to specify which pair of attributes is conditionally independent and which pair of attributes is conditionally dependent.

A Bayesian belief network (Bayesian network for short) represents the joint probability distribution for a set of variables.

Model Representation

- A Bayesian belief network (BBN), or simply, Bayesian network, provides a graphical representation of the probabilistic relationships among a set of random variables. There are two key elements of a Bayesian network:

1. A directed acyclic graph (dag) encoding the dependence relationships among a set of variables. 2. A conditional probability table (CPT) associating each variable in the network.



For example, the above figure shows the Bayesian belief network for detecting HeartDisease and HeartBurn in patients. Each variable in the diagram is assumed to be binary-valued. The parent nodes for heart disease (HD) correspond to risk factors that may affect the disease, such as exercise (E) and diet (D). The child nodes for heart disease correspond to symptoms of the disease, such as chest pain (CP) and high blood pressure (BP). For example, the diagram shows that heartburn (Hb) may result from an unhealthy diet and may lead to chest pain.

Model Building

- Model building in Bayesian networks involves two steps: (1) creating the structure of the network, and (2) estimating the probability values in the tables associated with each node. The network topology can be obtained by encoding the subjective knowledge of domain experts.

Example of Inferencing Using BBN

- Suppose we are interested in using the BBN shown above to diagnose whether a person has heart disease. The following cases illustrate how the diagnosis can be made under different scenarios.

Case 1: No Prior Information Without any prior information, we can determine whether the person is likely to have heart disease by computing the prior probabilities P(HD = Yes) and P(HD = No). To simplify the notation, let α Є{Yes, No} denote the binary values of Exercise and β Є {Healthy, Unhealthy} denote the binary value of Diet.

$$
\begin{aligned}
P(\mathtt{HD} = \mathtt{Yes}) &= \sum_{\alpha}\sum_{\beta} P(\mathtt{HD} = \mathtt{Yes}|E = \alpha, D = \beta)P(E = \alpha, D = \beta) \\
&= \sum_{\alpha}\sum_{\beta} P(\mathtt{HD} = \mathtt{Yes}|E = \alpha, D = \beta)P(E = \alpha)P(D = \beta) \\
&= 0.25 \times 0.7 \times 0.25 + 0.45 \times 0.7 \times 0.75 + 0.55 \times 0.3 \times 0.25 \\
&\quad + 0.75 \times 0.3 \times 0.75 \\
&= 0.49.
\end{aligned}
$$

Since P(HD = No) = 1 - P(HD = Yes) = 0.51, the person has a slightly higher chance of not getting the disease. **Case 2: High Blood Pressure** If the person has high blood pressure) we can make a diagnosis about heart disease by comparing the posterior probabilities, P(HD = Yes | BP = High) against P(HD = No | BP = High). To do this, we must compute P(BP = High):

$$P(\texttt{BP} = \texttt{High}) = \sum_{\gamma} P(\texttt{BP} = \texttt{High}|\texttt{HD} = \gamma)P(\texttt{HD} = \gamma)$$
$$= 0.85 \times 0.49 + 0.2 \times 0.51 = 0.5185.$$

where $\gamma \in \{\texttt{Yes}, \texttt{No}\}$. Therefore, the posterior probability the person has heart disease is

$$P(\texttt{HD} = \texttt{Yes}|\texttt{BP} = \texttt{High}) = \frac{P(\texttt{BP} = \texttt{High}|\texttt{HD} = \texttt{Yes})P(\texttt{HD} = \texttt{Yes})}{P(\texttt{BP} = \texttt{High})}$$
$$= \frac{0.85 \times 0.49}{0.5185} = 0.8033.$$

Similarly, $P(\texttt{HD} = \texttt{No}|\texttt{BP} = \texttt{High}) = 1 - 0.8033 = 0.1967$. Therefore, when a person has high blood pressure, it increases the risk of heart disease.

Characteristics of BBN Following are some of the general characteristics of the BBN method:

1. BBN provides an approach for capturing the prior knowledge of a particular domain using a graphical model. The network can also be used to encode causal dependencies among variables.

2. Constructing the network can be time consuming and requires a large amount of effort. However, once the structure of the network has been determined, adding a new variable is quite straightforward.

3. Bayesian networks are well suited to dealing with incomplete data. Instances with missing attributes can be handled by summing or integrating the probabilities over all possible values of the attribute.

4. Because the data is combined probabilistically with prior knowledge, the method is quite robust to model overfitting.

UNIT-IV
SECTION-A

Objective Questions

1. Bayes error is the_____bound of probability of classification error. [ ]
A. Lower B. Upper

2. Bayes decision rule is the theoretically …….. classifier that minimize probability of classification error. [ ]
A. Best B. Worst C. Average

3. In Bayes Theorem, unconditional probability is called as [ ]
A. Evidence b. Likelihood c. Prior d. Posterior

4. In Bayes Theorem, Class conditional probability is called as [ ]
A. Evidence B. Likelihood c. Prior d. Posterior

5. Bayesian reasoning provides a _____approach to inference. [ ]
A. deterministic B. Probabilistic C. both A and B D. none of the above

6. Previous probabilities in Bayes Theorem that are changed with help of new available information are classified as [ ]
A. independent probabilities B. posterior probabilities
C. independent probabilities D. dependent probabilities

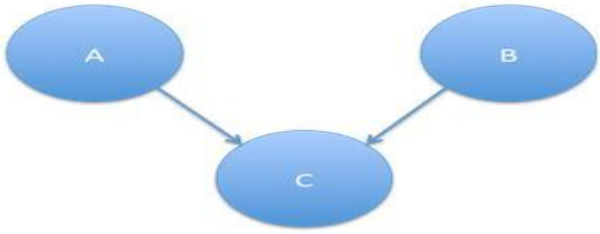
7. Three components of Bayes decision rule are class prior, likelihood and[ ]
A. Evidence B. Instance C. Confidence D. Salience

8. $P(\vec{X})P(wi/\vec{X})$ = [ ]

A. $P(1-\vec{X})P(wi/\vec{X})$ B. $P(\vec{X})P(1-wi/\vec{X})$ C. $P(\vec{X}|wi)P(wi)$ D. $P(\vec{X}-wi)P(wi/\vec{X})$

9. A and B are Boolean random variables. Given: P(A=True) = 0.3, P(A=False) = 0.7, P(B=True|A=True) = 0.4, P(B=False|A=True) = 0.6, P(B=True|A=False) = 0.6, P(B=False|A=False) = 0.4 . Calculate P(A=True|B=False) by Bayes rule.
[ ]
A.0.49 B. 0.39 C. 0.37 D.0.28

10. In the following Bayesian network A, B and C are Boolean random variables taking values in {True, False}.



Which of the following statements is true? [ ]

A. The value of C is not given. If the value of B changes from True to False, the conditional probability of A, P(A|B) changes.

B. The value of C is given to be True. If the value of B changes from True to False, the conditional probability of A, P(A|B) changes.

C. Neither A nor B D. Both A and B

11. Diabetic Retinopathy is a disease that affects 80% people who have diabetes for more than 10 years. 5% of the Indian population has been suffering from diabetes for more than 10 years. Answer the following questions. What is the joint probability of finding an Indian suffering from Diabetes for more than 10 years and also has Diabetic Retinopathy? [ ]

A. 0.024 B. 0.040 C. 0.076 D. 0.005

12. Which of the following properties is false in the case of a Bayesian Network: [ ]

A. The edges are directed
B. Contains cycles
C. Represents conditional independence relations among random variables
D. All of the above

SECTION-B

Descriptive Questions

1. Define the concept of Conditional Independence.
2. What is Bayes theorem? Explain how this is used in computing MAP and Maximum likelihood hypothesis?
3. Write the features of Bayesian learning methods.
3. Explain Naive Bayes Classifier with example.
4. Write a short note on Bayesian Belief Networks.
5. How is Naive Bayes algorithm useful for learning and classifying text?
6. Describe maximum likelihood and least-squared error hypotheses

7. Explain minimum description length principle.
8. How the gradient search can be performed to maximize likelihood in a neural net.
9. Illustrate the steps for Brute-force MAP learning algorithm
10. Explain about posteriori probability in Bayes theorem

UNIT-V

COMPUTATIONAL LEARNING THEORY

5.1 INTRODUCTION

Computational learning theory provides answers to the following questions within particular problem settings.

- Is it possible to identify learning problems as difficult or easy?
- Is it possible to identify learning problems that are independent of the learning algorithm?
- Number of training examples necessary or sufficient for successful learning?
- How is this number affected if the learner is allowed to pose queries to the trainer, versus observing a random sample of training examples?
- Number of mistakes that a learner will make before learning the target function?
- What is the Computational complexity of classes of learning problems?

5.2 PROBABLY LEARNING AN APPROXIMATELY CORRECT HYPOTHESIS

we consider a particular setting for the learning problem, called the probably approximately correct (PAC) learning model.

5.2.1 The Problem Setting:

Computational Learning Theory considers below problem setting to answer the listed questions.

Let X refer to the set of all possible instances.

Let C refer to some set of target concepts that our learner might be called upon to learn. Each target concept c in C corresponds to some subset of X, or equivalently to some Boolean valued function $c : X \rightarrow \{0, 1\}$.

We assume instances are generated at random from X according to some probability distribution **D**.

Training examples are generated by drawing an instance x at random according to D, then presenting x along with its target value, **c(x)**, to the learner.

The learner L considers some set H of possible hypotheses when attempting to learn the target concept. After observing a sequence of training examples of the target concept c, L must output some hypothesis h from H, which is its estimate of c.

To be fair, we evaluate the success of L by the performance of h over new instances drawn randomly from X according to D, the same probability distribution used to generate the training data. Within this setting, we characterize the performance of various learners.

5.2.2 Error of a Hypothesis:

The true error of **h** is just the error rate we expect when applying h to future instances drawn according to the probability distribution D.

Definition: The true error (denoted errorD(h)) of hypothesis h with respect to target concept c and distribution D is the probability that h will misclassify an instance drawn at random according to D.

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

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

Figure shows this definition of error in graphical form. The concepts c and ***h*** are depicted by the sets of instances within X that they label as positive. The error of ***h*** with respect to c is the probability that a randomly drawn instance will fall into the region where ***h*** and c disagree (i.e., their set difference). Note we have chosen to define error over the ***entire distribution*** of instances-not simply over the training examples-because this is the true error we expect to encounter when actually using the learned hypothesis ***h*** on subsequent instances drawn from *D*.

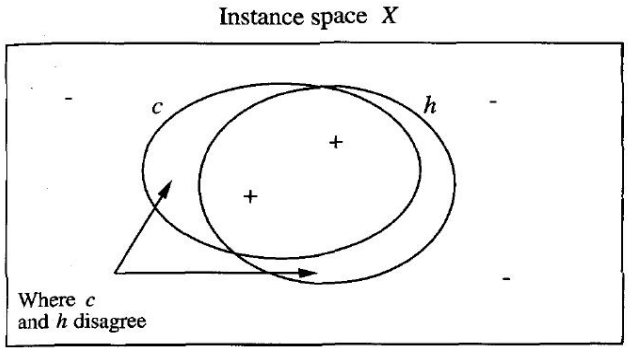


Figure:

The error of hypothesis h with respect to target concept c. The error of h with respect to c is the probability that a randomly drawn instance will fall into the region where h and c disagree on its classification. The + and − points indicate positive and negative training examples. Note h has a nonzero error with respect to c despite the fact that h and c agree on all five training examples observed thus far.

5.2.3 PAC Learnability

Our aim is to characterize classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation.

- What kinds of statements about learnability should we guess hold true?
- We might try to characterize the number of training examples needed to learn a hypothesis h for which errorD(h) = 0.

To accommodate these two difficulties, we weaken our demands on the learner in two ways. First, we will not require that the learner output a zero error hypothesis-we will require only that its error be bounded by some constant, c, that can be made arbitrarily small. Second, we will not require that the learner succeed for every sequence of randomly drawn training examples-we will require only that its probability of failure be bounded by some constant, , that can be made arbitrarily small. In short, we require only that the learner probably learn a hypothesis that is approximately correct-hence the term probably approximately correct learning, or PAC learning for short.

Consider some class C of possible target concepts and a learner L using hypothesis space H. Loosely speaking, we will say that the concept class C is PAC-learnable by L using H if, for any target concept c in C, L will with probability (1 - δ) output a hypothesis h with $error_D(h) < €$, after observing a reasonable number of training examples and performing a reasonable amount of computation. More precisely,

Definition: Consider a concept class C defined over a set of instances X of length ***n*** and a learner L using hypothesis space ***H.*** C is PAC-learnable by L using H if for all $c \in C$, distributions ***D*** over X, $€$ such that $0 < \delta < 1/2$, and δ such that $0 < \delta < 1/2$, learner L will with probability at least (1 - δ) output a hypothesis $h \in H$ such that $error_D(h)$ *5* $€$, in time that is polynomial in $1/€$, $1/\delta$, n, and size(c).

This definition requires two things from L.

i) L must, with arbitrarily high probability (1 - δ*)*, output a hypothesis having arbitrarily low error ***(€).***

ii)it must do so efficiently-in time that grows at most polynomially with $1/€$ and $1/\delta$, which define the strength of our demands on the output hypothesis, and with *n* and size(c) that define the inherent complexity of the underlying instance space X and concept class C. Here, *n* is the size of instances in X.

5.3 SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACES

PAC-learnability is largely determined by the number of training examples required by the learner.

The growth in the number of required training examples with problem size, called the ***sample complexity*** of the learning problem, is the characteristic that is usually of greatest interest. The reason is that in most practical settings the factor that most limits success of the learner is the limited availability of training data.

Here we present a general bound on the sample complexity for a very broad class of learners, called ***consistent learners.***

A learner is ***consistent*** if it outputs hypotheses that perfectly fit the training data, whenever possible. It is quite reasonable to ask that a learning algorithm be consistent, given that we typically prefer a hypothesis that fits the training data over one that does not.

The version space can derive a bound on the number of training examples required by ***any*** consistent learner, independent of the specific algorithm that generates consistent hypothesis.
The version space is defined as ***VSH,D,*** to be the set of all hypotheses ***h E H*** that correctly classify the training examples ***D.***

$$VS_{H,D} = \{h \in H | (\forall \langle x, c(x) \rangle \in D)\ (h(x) = c(x))\}$$

Definition: Consider a hypothesis space H, target concept *c,* instance distribution *D,* and set of training examples D of *c.* The version space $VS_{H,D}$, is said to be €-exhausted with respect to *c* and *D,*if every hypothesis h in $VS_{H,D}$, has error less than € with respect to *c* and *D.*

$$(\forall h \in VS_{H,D})\ error_D(h) < \epsilon$$

This definition is illustrated in below figure. The version space is €-exhausted just in the case that all the hypotheses consistent with the observed training examples (i.e., those with zero training error) happen to have true error less than € .

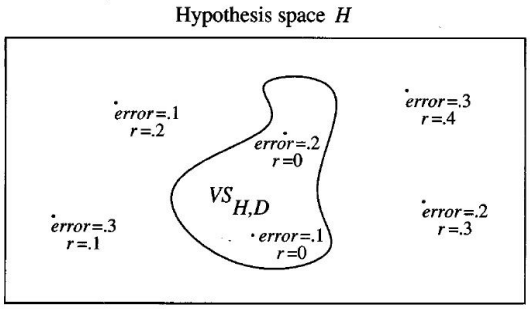


Figure :Exhausting the version space. The version space $VS_{H,D}$ is the subset of hypotheses $h \in H$, which have zero training error (denoted by $r = 0$ in the figure). Of course the true $error_D(h)$ (denoted by *error* in the figure) may be nonzero, even for hypotheses that commit zero errors over the training data. The version space is said to be €-exhausted when all hypotheses h remaining in $VS_{H,D}$, have $error_D(h) <$ €.

5.4 SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

we consider a measure of the complexity of H, called the Vapnik-Chervonenkis dimension of H (VC dimension, or VC(H), for short). As we shall see, we can state bounds on sample complexity that use VC(H) rather than |H|.

5.4.1 Shattering a Set of Instances

The VC dimension measures the complexity of the hypothesis space H, not by the number of distinct hypotheses |H|, but instead by the number of distinct instances from X that can be completely discriminated using H.

First define the notion of shattering a set of instances. Consider some subset of instances S ⊆ X. For example, below figure shows a subset of three instances from X. Each hypothesis h from H imposes some dichotomy on S; that is, h partitions S into the two subsets {x ∈ S / h(x) = 1} and {x ∈ S/h(x) = 0}. Given some instance set S, there are $2^{|S|}$ possible dichotomies, though H may be unable to represent some of these. We say that H shatters S if every possible dichotomy of S can be represented by some hypothesis from H.

Definition: A set of instances ***S*** is shattered by hypothesis space H if and only if for every dichotomy of ***S*** there exists some hypothesis in H consistent with this dichotomy.

Figure illustrates a set S of three instances that is shattered by the hypothesis space. Notice that each of the 2^3 dichotomies of these three instances is covered by some hypothesis.

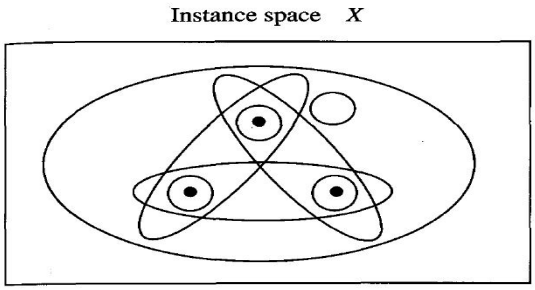


Figure :A set of three instances shattered by eight hypotheses. For every possible dichotomy of the instances, there exists a corresponding hypothesis.

7.4.2 The Vapnik-Chervonenkis Dimension

The ability to shatter a set of instances is closely related to the inductive bias of a hypothesis space.

An unbiased hypothesis space H is one that shatters the instance space X. What if H cannot shatter X, but can shatter some large subset S of X? Intuitively, it seems reasonable to say that the larger the subset of X that can be shattered, the more expressive H. The VC dimension of H is precisely this measure.

Definition: The Vapnik-Chervonenkis dimension, *VC(H),* of hypothesis space *H* defined over instance space *X* is the size of the largest finite subset of X shattered by *H.* If arbitrarily large finite sets of X can be shattered by H, then *VC(H)* = ∞.

- For any finite H,VC(H)≤ log2 |H

Example1 for finding VC(H):

- Suppose the instance space X is the set of real numbers X = ℝ (e.g., describing the height of people), and H is the set of hypotheses of the form a < ***x*** < ***b,*** where a and b may be any real constants.
- To find VC(H), we must find the largest subset of X that can be shattered by H.

- Consider a particular subset containing two distinct instances, say S = {3.1,5.7}. Here S is shattered by H.
- For example, consider the four hypotheses (1 < ***x*** < 2), (1 < ***x*** < 4), ***(4*** < ***x*** < 7), and (1 < ***x*** < 7). These hypotheses together represent each of the four dichotomies over S, covering neither instance, either one of the instances, and both of the instances, respectively.
- Since we have found a set of size two that can be shattered by H, the VC dimension of H is at least two.
- There is no subset S of size three that can be shattered by H, and VC(H) = 2.
- Here H is infinite, but VC(H) finite.

Example2 for finding VC(H):

- Consider the set X of instances corresponding to points on the ***x,*** y plane
- Let H be the set of all linear decision surfaces in the plane.
- Now we will find the VC dimension of this H?
- Any two distinct points in the plane can be shattered by H, because we can find four linear surfaces that include neither, either, or both points.
- For a sets of three points, if the points are not colinear, then there will be 2^3 linear surfaces that shatter the points.
- VC(H) in this case is three as no sets of size four can be shattered.
- **The VC dimension of linear decision surfaces in an r dimensional space is r + 1.**

Example3 for finding VC(H):

- Let each instance in X is described by the conjunction of exactly three Boolean literals.
- Also let each hypothesis in H is described by the conjunction of up to three boolean literals.
- To find its VC(H), lets represent each instance by a 3-bit string corresponding to the values of each of its three literals l1, l2, and l3.
- Consider the following set of three instances:
  $instance_1$:100

instance2 :010

instance3 :001

- This set of three instances can be shattered by H, because a hypothesis can be constructed for any desired dichotomy as follows: If the dichotomy is to exclude instancei, add the literal ¬li to the hypothesis.
- For example, if we wish to include instance2, but exclude instance1 and instance3. Then we use the hypothesis ¬l1 ¬l3.
- The VC dimension for conjunctions of n boolean literals is at least n.

7.4.3 Sample Complexity and the VC Dimension

Upper bound on sample complexity:

Based on VC(H) as a measure for the complexity of H, the upper bound on the number of training examples sufficient to probably approximately learn any target concept in C, for any desired **ϵ** and δ is given below:

$$m \geq \frac{1}{\epsilon}(4\log_2(2/\delta) + 8VC(H)\log_2(13/\epsilon))$$

Lower bound on sample complexity: Consider any concept class ***C*** such that ***VC(C)*** ≥ 2, any learner L, and any *0* < ***ϵ*** < 1/8, and 0 < δ < 1/100. Then there exists a distribution D and target concept in ***C*** such that if L observes fewer examples than

$$\max\left[\frac{1}{\epsilon}\log(1/\delta), \frac{VC(C)-1}{32\epsilon}\right]$$

then with probability at least δ, ***L*** outputs a hypothesis ***h*** having ***errorD(h)*** > ϵ.

5.5 THE MISTAKE BOUND MODEL OF LEARNING

In the mistake bound model of learning, the learner is evaluated by the total number of mistakes it makes before it converges to the correct hypothesis.

- Assume the learner receives a sequence of training examples. For each example x, the learner must predict the target value c(x), before it is shown the correct target value by the trainer.
- Here we consider the no. of mistakes that the learner make in its predictions before it learns the target concept.
- This mistake bound learning problem can be studied in various specific settings. For example, we might count the number of mistakes made before PAC learning the target concept.

5.5.1 Mistake Bound for the FIND-S Algorithm

- Let us see the bound on the total number of mistakes that FIND-S will make before exactly learning the target concept c.
- If c Є H (consisting of conjunction of n boolean literals), then FIND-S can never mistakenly classify a negative example as positive.
- To calculate the number of mistakes it will make, we need to count the number of mistakes it will make misclassifying truly positive examples as negative.
- How many such mistakes can occur before FIND-S learns c exactly?
- Consider the first positive example encountered by FIND-S. The learner will certainly make a mistake classifying this example, because its initial hypothesis labels every instance negative.
- The result will be that half of the $2n$ terms in its initial hypothesis will be eliminated, leaving only n terms.
- For each subsequent positive example that is mistakenly classified by the current hypothesis, at least one more of the remaining n terms must be eliminated from the hypothesis.

- Therefore, the total number of mistakes can be at most $n + 1$. This number of mistakes will be required in the worst case, corresponding to learning the most general possible target concept (∀x)c(x) = 1.

FIND-S:

- Initialize h to the most specific hypothesis $l_1 \wedge \neg l_1 \wedge l_2 \wedge \neg l_2 \ldots l_n \wedge \neg l_n$
- For each positive training instance x
  - Remove from h any literal that is not satisfied by x
- Output hypothesis h.

FIND-S converges in the limit to a hypothesis that makes no errors, provided C⊆ H and provided the training data is noise-free. FIND-S begins with the most specific hypothesis (which classifies every instance a negative example), then incrementally generalizes this hypothesis as needed to cover observed positive training examples.

5.5.2 Mistake Bound for the HALVING Algorithm

- Let us see the bound on the total number of mistakes that List-then-Eliminate and Candidate-elimination algorithm will make before exactly learning the target concept c.
- Let us assume this prediction is made by taking a majority vote among the hypotheses in the current version space. This combination of learning the version space, together with using a majority vote to make subsequent predictions, is called the HALVING algorithm.
- The HALVING algorithm can make a mistake when the majority of hypotheses in its current version space incorrectly classify the new example.
- Once the correct classification is revealed to the learner, the version space will be reduced to at most half its current size.
- Given that each mistake reduces the size of the version space by at least half, and given that the initial version space contains only |H| members, the maximum number of (worst case) mistakes possible before the version space contains just one member is $\log_2 |H|$.

5.5.3 Optimal Mistake Bounds

- Let us see optimal mistake bound, i.e. the lowest worst-case mistake bound over all possible learning algorithms.
- Assume for any learning algorithm ***A*** and any target concept c, let $M_A(c)$ denote the maximum over all possible sequences of training examples of the number of mistakes made by ***A*** to exactly learn c.

- **Definition:** Let ***C*** be an arbitrary nonempty concept class. The optimal mistake bound for ***C,*** denoted ***Opt(C),*** is the minimum over all possible learning algorithms A of $M_A(C)$.

$$Opt(C) \equiv \min_{A \in learning\ algorithms} M_A(C)$$

- Opt(C) is the number of mistakes made for the hardest target concept in C, using the hardest training sequence, by the best algorithm.
- For any concept class C, there is an interesting relationship among the optimal mistake bound for C, the bound of the HALVING algorithm, and the VC dimension of C.

$$VC(C) \leq Opt(C) \leq M_{Halving}(C) \leq log_2(|C|)$$

5.5.4 WEIGHTED-MAJORITY algorithm

- The Weighted-Majority algorithm makes predictions by taking a weighted vote among a pool of prediction algorithms and learns by altering the weight associated with each prediction algorithm.

- One of the property of Weighted-Majority algorithm is able to accommodate inconsistent training data. This is because it does not eliminate a hypothesis that is found to be inconsistent with some training example, but rather reduces its weight.

- Another interesting property is that we can bound the number of mistakes made by Weighted-Majority in terms of the number of mistakes committed by the best of the pool of prediction algorithms.
- Weighted-Majority algorithm is given below:

---

a_i denotes the i^{th} prediction algorithm in the pool A of algorithms. w_i denotes the weight associated with a_i.

- For all i initialize $w_i \leftarrow 1$
- For each training example $\langle x, c(x) \rangle$
  - Initialize q_0 and q_1 to 0
  - For each prediction algorithm a_i
    - If $a_i(x) = 0$ then $q_0 \leftarrow q_0 + w_i$
      If $a_i(x) = 1$ then $q_1 \leftarrow q_1 + w_i$
- If $q_1 > q_0$ then predict $c(x) = 1$
  If $q_0 > q_1$ then predict $c(x) = 0$
  If $q_1 = q_0$ then predict 0 or 1 at random for $c(x)$
- For each prediction algorithm a_i in A do
  If $a_i(x) \neq c(x)$ then $w_i \leftarrow \beta w_i$

UNIT-V
SECTION-A

Objective Questions

1. Suppose you have a PAC learning algorithm A for a concept class C such that with probability at least 0.5, the algorithm will output an approximately correct hypothesis. Suppose that for deployment purposes, you need an algorithm which can output the approximately correct hypothesis with probability at least 0.998. Is it possible to make use of algorithm A for this purpose? If so, how? [ ]

A. No, we cannot make use of A
B. Yes, repeat A three times and choose the best hypothesis
C. Yes, repeat A _ve times and choose the best hypothesis
D. Yes, repeat A nine times and choose the best hypothesis

2. To say that the VC-dimension of a class is at least k, is it necessary for the class to be able to shatter any configuration of k points? [ ]
A. Yes B. No

3. What is the VC-dimension of the class of axis-parallel rectangles? [ ]
A. 3 B. 4 C. 5 D. 6

4. The VC dimension of hypothesis space H1 is larger than the VC dimension of hypothesis space H2. Which of the following can be inferred from this?[ ]
A. The number of examples required for learning a hypothesis in H1 is larger than the number of examples required for H2.
B. The number of examples required for learning a hypothesis in H1 is smaller than the number of examples required for H2.
C. The number of examples required for learning a hypothesis in H1 is equal to the number of examples required for H2.
D. No relation to number of samples required for PAC learning

5. For a particular learning task, if the requirement of error parameter □ changes from 0.1 to 0.01. How many more samples will be required for PAC learning? [ ]

A. same B. 2 times C. 10 times D. 100 times

6. Suppose the VC dimension of a hypothesis space is 4. Which of the following are true? [ ]
A. No sets of 4 points can be shattered by the hypothesis space.

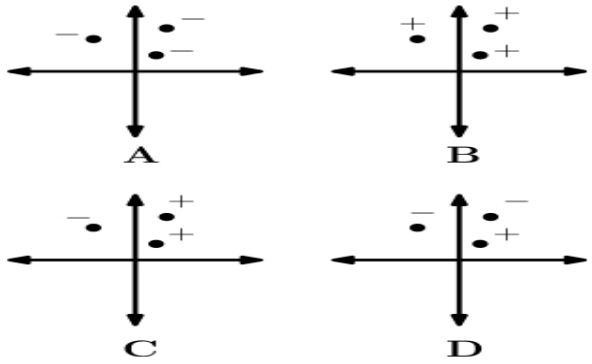
B. Atleast one set of 4 points can be shattered by the hypothesis space.
C. All sets of 4 points can be shattered by the hypothesis space.
D. No set of 5 points can be shattered by the hypothesis space.

7. Consider a circle in 2D whose center is at the origin. What is its VC dimension? [ ]

A. 1 B.2 C.3 D.4

8. Under a binary classification setting, which of the following sets of three labeled points cannot be shattered by a circle centered at the origin?

[ ]



A. A B. B C. C D. D

SECTION-B

Descriptive Questions

1. Define true error.
2. Describe these terms in brief (I) PAC Hypothesis (II) Mistake bound model of learning
3. Explain mistake bounds for Find-S algorithm and halving algorithm.
4. Describe the probability learning an approximately correct hypothesis.
5. Explain sample complexity for finite hypothesis space.
6. Explain sample complexity for infinite hypothesis spaces.
7. Describe optimal mistake bounds.
8. Explain the significance of VC Dimension measure with suitable example.

9.Discuss weighted majority algorithm.

10.How to calculate sample complexity for infinite hypothesis space?