

AUTHORS PROFILE



Dr.S.GOKULAKRISHNAN Completed his B.Tech (Information Technology) from Pallavan College of Engg ,Kanchipuram, Tamil Nadu, India and M. Tech (Information Technology) from Sathyabama University, Chennai, in the year 2005 and 2011 respectively. He also completed Ph.D in Sathyabama University, Chennai. His areas of interest include Cloud Computing and Big data Analytics. He has got around 16 years of teaching experience in various Institutions. Currently working as a Assistant Professor in CSE Department at Sri ChandrasekharendraSaraswathiViswaMahavidyalaya ,EnathurKanchipuram.



Dr.C.SUNITHA RAM received her Ph.D from Sri Chandrasekerandra Saraswathi Viswa Maha Vidyalaya Univeristy, Master of Engineering in Computer Science and Engineering from Sathyabama University and Bachelor of Engineering in Computer Science and Engineering from University of Madras. She has more than 15 years of experience in teaching profession. She is presently working as a Assistant professor in CSE, Sri Chandrasekerandra Viswa Maha Vidyalaya University. She has more theoretical and practical experience in teaching profession. Her area of interest is Principles of compiler design, Computer architecture, System software, Database Management Systems (DBMS), Machine learning and Neural Networks. Her area of specialization lies in Computational intelligence domain. She has been attended more than 20 FDP on various emerging topics, presented various papers in international conferences and also in Journals. She has been served as reviewer committee member for an international journal. She has been an active member in IAENG professional bodies.



Dr N.KUMARAN received his Ph.D. from National Institute of Technology, Tiruchirappalli., M.Tech (Information Technology) from Sathyabama University, Chennai and B.E (Computer Science and Engineering) from Coimbatore Institute of Technology, Coimbatore-14, Tamil Nadu, India, in the year 2007 and 1998 respectively. He is currently working as an Assistant Professor in the Department of Computer Science and Engineering, Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya (Deemed to be University) Kanchipuram. He has served as a reviewer committee for an international journal. He has been an active member of professional bodies, such as IAENG and Judge of Toycathon 2021(Ministry of Education and Innovative Cell). He also received best teacher award in the 2016 in his university.His areas of interest include Video Analysis, Machine Learning, Internet of Things and Computer Networks. He has got around 20 years of teaching experience various Institutions.



SHYAM MOHAN J S received the B.E (Computer Science and Engineering) from Anna University, Chennai in 2008 and the Masters Degree in Computer Science and Engineering from Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya SCSVMV (Deemed to be University), Enathur, Kanchipuram in 2012. He is currently working as an Assistant Professor in the Department of Computer Science and Engineering at Sri Chandrasekharendra Saraswathi Viswa Mahavidyalaya (SCSVMV), Kanchipuram, and Tamilnadu, India. He has published various Research papers from his research findings in many International Journals. His research areas include Big Data Analytics, Cloud Computing, Blockchain Technology and Internet of Things (IoT). He is reviewer in many International Journals. He is a member of various technical bodies including UACEE and IAENG.

CHARULATHA PUBLICATIONS

38/7, Rukmani Street, West Mambalam, Chennai - 600 033.

Mobile : 98404 28577

Email : charulathapublication@yahoo.com / info@charulathapublication.com

FOR ONLINE PURCHASE

web : www.charulathapublication.com

ISBN-13: 978-93-5577-058-5



Machine Learning

Machine Learning

Dr. S. Gokulakrishnan

Dr. C. Sunitha

Dr. N. Kumaran

Shyam Mohan J.S.



Dr. S. Gokulakrishnan, Dr. C. Sunitha
Dr. N. Kumaran, Shyam Mohan J.S.

CHARULATHA PUBLICATIONS



Raja Rammohun Roy National Agency for ISBN

Department of Higher Education, Ministry of Human Resource Development

Government of India



HOMEAbout UsHow To ApplyContact UsFAQUser ManualISBN Report

Recently Published Books

Book Title	<input type="text"/>	Email	<input type="text"/>						
Name of Author	<input type="text"/>	Mobile	<input type="text"/> (do not add '0' as starti						
Name of Publisher	<input type="text"/>	Publish Year	<input type="text"/>						
ISBN Number	<input type="text" value="978-93-5577-058-5"/>		Issue Date	<input type="text" value="From Date"/>		<input type="text" value="To Date"/>			
			<input type="button" value="Submit"/>						

SI No.	Book Title	Name of Publisher	Publishing Agency	Author	Product Composition	Product form	Language	ISBN No.
1.	MACHINE LEARNING	M.R.Bharathi	charulatha publications	Dr. S. Gokulakrishnan, Dr. C. Sunitha, Dr. N. Kumaran and Shyam Mohan J.S	Single-component retail product	Book	English	978-95577-058-5

Machine Learning

Dr. S. Gokulakrishnan

Assistant Professor

Department of Computer Science and Engineering

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

[SCSVMV Deemed to be University]

Enathur, Kanchipuram - 631 561.

Dr. C. Sunitha Ram

Assistant Professor

Department of Computer Science and Engineering

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

[SCSVMV Deemed to be University]

Enathur, Kanchipuram - 631 561.

Dr N. Kumaran

Assistant Professor

Department of Computer Science and Engineering

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

[SCSVMV Deemed to be University]

Enathur, Kanchipuram - 631 561.

Shyam Mohan J S

Assistant Professor

Department of Computer Science and Engineering

SRI CHANDRASEKHARENDRA SARASWATHI VISWA MAHAVIDYALAYA

[SCSVMV Deemed to be University]

Enathur, Kanchipuram - 631 561.

For online purchase

www.charulathapublications.com

January 2022

Price : **Rs.375/-**

ISBN No. : 978-93-5577-058-5

CHARULATHA	PUBLICATIONS
------------	--------------

38/7, Rukmani Street,
West Mambalam, Chennai - 600 033.
Phone : 044-24745589, 044-24746546
Email : charulathapublication@yahoo.com
info@charulathapublication.com
web : www.charulathapublication.com

CONTENTS

UNIT – 1	INTRODUCTION	1.1 – 1.40
1.1. MACHINE LEARNING: A BRIEF INTRODUCTION	1.1	
1.1.1. Classes of Machine Learning Algorithms	1.2	
1.1.2. Types of Machine Learning algorithms	1.4	
1.2. LEARNING PROBLEMS	1.8	
1.2.1. Designing a Learning System.....	1.10	
1.3. PRESPECTIVES AND ISSUES IN MACHINE LEARNING.....	1.13	
1.4. CONCEPT LEARNING	1.14	
1.5. VERSION SPACES AND CANDIDATE ELIMINATIONS	1.19	
1.5.1. Version Space	1.19	
1.5.2. Representation of version space (LIST-THEN-ELIMINATE).....	1.21	
1.5.3. Candidate Elimination Algorithm	1.23	
1.6. INDUCTIVE BIAS	1.26	
1.6.1. Biased Hypothesis Space.....	1.27	
1.6.2. An Unbiased Learner.....	1.27	
1.6.3. The Futility of Bias-Free Learning.....	1.28	
1.7. DECISION TREE LEARNING.....	1.31	
1.7.1. Representing Decision Trees.....	1.31	
1.7.2. Decision tree algorithm	1.32	
1.7.3. Heuristic Space Search in Decision Tree Learning	1.40	
UNIT – 2	NEURAL NETWORKS AND GENETIC ALGORITHMS	2.1 – 2.53
2.1. INTRODUCTION NERUAL NETWORKS.....	2.1	
2.2. REPRESENTATION OF NEURAL NETWORKS	2.3	
2.3. PROBLEMS SOLVED BY NEURAL NETWORKS	2.5	
2.4. PERCEPTRONS	2.5	
2.4.1. Perceptrons over Boolean functions	2.8	

2.4.2.	Training rules in Perceptron: Perceptron Rule	2.10
2.4.3.	Training rules in Perceptron: Gradient Descent and Delta Rule	2.11
2.4.4.	Visualizing the hypothesis space.....	2.12
2.4.5.	Stochastic Gradient Descent (SGD)	2.15
2.5.	MULTILAYER NETWORKS AND THE BACKPROPAGATION ALGORITHM	2.17
2.5.1.	Multi Layer Perceptron (MLP).....	2.17
2.5.2.	Back propagation Algorithm	2.20
2.5.3.	Adding Momentum	2.22
2.4.6.	Derivation of Back propagation rule	2.24
2.5.5.	Additional Information	2.26
2.6.	INTRODUCTION TO GENETIC ALGORITHMS.....	2.29
2.6.1.	Representation of hypothesis.....	2.33
2.6.2.	Genetic Operations	2.35
2.7.	HYPOTHESIS SPACE SEARCH	2.43
2.7.1.	Population Evolution and the Schema Theorem	2.44
2.8.	GENETIC PROGRAMMING	2.47
2.9.	MODELS OF EVALUATION AND LEARNING	2.52
2.9.1.	Lamarckian Evolution	2.52
2.9.2.	Baldwin Effect.....	2.53
<hr/> UNIT – 3 BATESIAN AND COMPUTATIONAL LEARNING		3.1 – 3.46
3.1.	INTRODUCTION TO BAYESIAN LEARNING	3.1
3.2.	BAYES THEOREM.....	3.2
3.3.	CONCEPT LEARNING AND BAYES THEOREM	3.5
3.3.1.	Fundamental Probability Rules	3.5
3.3.2.	Brute Force MAP Hypothesis Learner	3.6
3.3.3.	MAP Hypothesis and Consistent Learners.....	3.8
3.4.	MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES	3.9

3.5. MAXIMUM LIKELIHOOD HYPOTHESES	3.12
3.5.1. Gradient Search to Maximize Likelihood in a Neural Net.....	3.12
3.6. MINIMUM DESCRIPTION LENGTH (MDL) PRINCIPLE	3.14
3.7. OPTIMAL BAYES CLASSIFIER	3.15
3.8. GIBBS ALGORITHM.....	3.16
3.9. NAIVE BAYES ALGORITHM.....	3.17
3.10. BAYESIAN BELIEF NETWORK	3.22
3.10.1. Conditional Independence.....	3.22
3.10.2. Inferences from Bayesian Belief Network	3.24
3.10.3. Learning Bayesian Networks	3.25
3.10.4. Gradient Ascent Training of Bayesian Networks.....	3.25
3.10.5. Learning the Structure of Bayesian Networks.....	3.27
3.11. EXPECTATION MAXIMIZATION (EM) ALGORITHM.....	3.28
3.11.1. Estimating means of k Gaussians	3.29
3.12. PROBABILITY LEARNING	3.32
3.12.1. Error of the hypothesis	3.34
3.12.2. PAC Learnability.....	3.35
3.13. SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACE.....	3.37
3.13.1. Agnostic Learning and Inconsistent Hypotheses	3.39
3.13.2. Conjunctions of Boolean Literals Are PAC-Learnable.....	3.40
3.14. SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES	3.40
3.14.1. Vapnik-Chervonenkis (VC) Dimension	3.41
3.14.2. VC Dimension for Neural Networks.....	3.44
3.15. MISTAKE BOUND (MB) MODEL	3.45
3.15.1. Mistakes in Halving Algorithm	3.46
3.15.2. Mistake Bound for the FIND-S Algorithm	3.46
3.15.3. Optimal mistake bounds.....	3.47
3.15.4. Weighted Majority Algorithm.....	3.47

UNIT – 4	INSTANT BASED LEARNING	4.1 – 4.18
4.1.	INTRODUCTION TO INSTANCE BASED LEARNING.....	4.1
4.2.	K-NEAREST NEIGHBOUR (K-NN).....	4.2
4.2.1.	Distance Weighted Nearest Neighbour Algorithm.....	4.5
4.3.	LOCALLY WEIGHTED REGRESSION	4.8
4.3.1.	Locally Weighted Linear Regression	4.9
4.4.	RADIAL BASIS FUNCTIONS (RBF)	4.11
4.5.	CASE BASED REASONING (CBR).....	4.14
4.5.1.	Problem solving in CBR.....	4.15
4.5.2.	CBR Working Cycle	4.16
4.5.3.	Case representation.....	4.18
UNIT – 5	ADVANCED LEARNING	5.1 – 5.58
5.1.	LEARNING SETS OF RULES.....	5.1
5.2.	SEQUENTIAL COVERING ALGORITHM.....	5.2
5.2.1.	General to Specific Beam Search.....	5.4
5.3.	LEARNING RULE SETS	5.7
5.4.	LEARNING FIRST-ORDER RULES.....	5.10
5.4.1.	First-Order Horn Clauses	5.11
5.4.2.	Terminologies.....	5.12
5.5.	LEARNING SETS OF FIRST-ORDER RULES: FOIL	5.13
5.5.1.	Expanding candidate specializations in FOIL.....	5.16
5.5.2.	Guiding the Search (Foil_Gain)	5.17
5.5.3.	Learning Recursive Sets	5.19
5.6.	INDUCTION ON INVERTED DEDUCTION	5.19
5.7.	INVERTING RESOLUTIONS.....	5.22
5.7.1.	First Order Resolution.....	5.24
5.7.2.	Generalization, θ -Subsumption, and Entailment.....	5.26

5.7.3. PROGOL	5.27
5.8. ANALYTICAL LEARNING	5.28
5.9. PERFECT DOMAIN THEORY	5.30
5.9.1. PROLOG-EBG (Kedar-Cabelli and McCarty)	5.31
5.9.2. Analysis of the explanation	5.34
5.9.3. Refine the current hypothesis	5.36
5.10. EXPLANATION BASED LEARNING (EBL)	5.36
5.10.1. Discovering new features	5.39
5.10.2. Deductive Learning	5.40
5.10.3. Inductive Bias in Explanation-Based Learning	5.41
5.10.4. Knowledge Level Learning	5.42
5.11. USING PRIOR KNOWLEDGE TO AUGMENT SEARCH OPERATORS: FIRST ORDER COMBINED LEARNER (FOCL) ALGORITHM	5.42
5.11.1. First Order Inductive Learner (FOIL)	5.42
5.11.2. First Order Combined Learner (FOCL)	5.44
5.11.3. Hypothesis search in FOCL	5.45
5.12. REINFORCEMENT LEARNING	5.47
5.13. LEARNING TASK IN REINFORCMENT LEARNING	5.51
5.14. Q-LEARNING (QUALITY LEARNING)	5.53
5.14.1. Q-function	5.53
5.14.2. Q-learning Algorithm	5.54
5.14.3. Convergence	5.55
5.14.4. Sequence Update in Q Learning	5.56
5.15. TEMPORAL DIFFERENCE LEARNING	5.58

UNIT

1

INTRODUCTION

1.1. MACHINE LEARNING: A BRIEF INTRODUCTION

Ever since computers were invented, we have wondered whether they might be made to learn. If we could understand how to program them to learn-to improve automatically with experience-the impact would be dramatic.

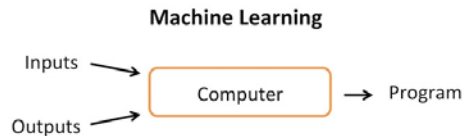
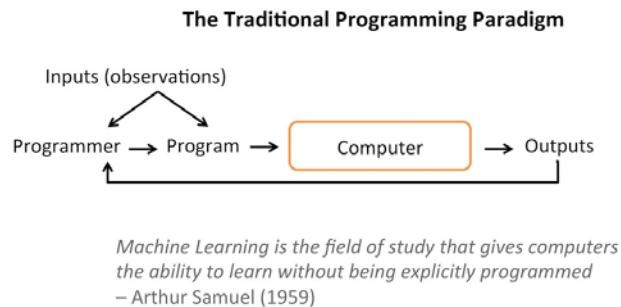
- Imagine computers learning from medical records which treatments are most effective for new diseases
- Houses learning from experience to optimize energy costs based on the particular usage patterns of their occupants.
- Personal software assistants learning the evolving interests of their users in order to highlight especially relevant stories from the online morning newspaper

A successful understanding of how to make computers learn, would open up many new uses of computers and new levels of competence and customization. Machine learning refers to a system that is capable of autonomous acquisition and integration of knowledge. This process of learning from experience through analytical observation results in continuously self-improved system to achieve high degree of efficiency. The direct implication of the machine learning is concerned about how to construct computer programs that automatically improve with experience without explicitly programming them. Tom Mitchell's famous definition of machine learning is stated below:

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 .

1.1.1. Classes of Machine Learning Algorithms

Machine learning is a set of methods that can automatically detect patterns in data in the form of text, images, numeric or videos and produces outputs, and then use the uncovered patterns to predict future data, or to take decisions with some degree of uncertainty. Machine learning is a subset of AI and can be seen as an implementation of AI.



Sebastian Raschka, 2016

Fig. 1.1. Programming Culture

Below are some classes of algorithms:

- Generalized linear models (e.g., logistic regression)
- Support vector machines (e.g., linear SVM, RBF-kernel SVM)
- Artificial neural networks (e.g., multi-layer perceptions)
- Tree - or rule-based models (e.g., decision trees)
- Graphical models (e.g., Bayesian networks)
- Ensembles (e.g., Random Forest)
- Instance-based learners (e.g., K-nearest neighbors)

Mathematical Interpretations

Before we know about the types we shall first understand the mathematical interpretation throughout his book. More formally, we define as the “hypothesis,” a function that we use to approximate some unknown function

$$h(x) = y$$

where x is a vector of input features associated with a training example or dataset instance (for example, the pixel values of an image) and y is the outcome we want to predict (e.g., what class of object we see in an image). In other words, $h(x)$ is a function that predicts y .

Given a training set or Dataset

$$D = \{\langle x^{[i]}, y^{[i]} \rangle\}, i = 1, \dots, n$$

We denote the i^{th} training example as $\langle x^{[i]}, y^{[i]} \rangle$. Please note that the superscript $[i]$ is unrelated to exponentiation. Note that a critical assumption for (most) machine learning theory is that the training examples are i.i.d. (independent and identically distributed).

In classification, we define the hypothesis function as

$$h : x \rightarrow y$$

Where $x \in \mathbb{R}^D$ and $y = \{1, \dots, k\}$ with class labels k . In the special case of binary classification, we have $y = \{0, 1\}$ (or $y = \{-1, 1\}$).

And in regression, the task is to learn a function $h: \mathbb{R}^m \rightarrow \mathbb{R}$ Hypothesis (model) class H : the set of classifier functions we will use. Ideally, the true class distribution C can be represented by a function in H (exactly, or with a small error).

x : A scalar denoting a single training example with 1 feature (e.g., the height of a person)

X : A training example with m features (e.g., with $m = 3$ we could represent the height, weight, and age of a person), represented as a column vector (i.e., a matrix with 1 column, $X \in \mathbb{R}^m$),

$$X = \begin{bmatrix} x_1 \\ x_2 \\ \vdots \\ x_m \end{bmatrix}$$

\mathfrak{X} : Design matrix, $\mathfrak{X} \in \mathbb{R}^{m \times n}$, which stores n training examples, where m is the number of features.

$$\mathfrak{X} = \begin{bmatrix} X_1^T \\ X_2^T \\ \vdots \\ X_n^T \end{bmatrix}$$

Note that in order to distinguish the feature index and the training example index, we will use a square-bracket superscript notation to refer to the i^{th} training example and a regular subscript notation to refer to the j^{th} feature:

$$\mathfrak{X} = \begin{bmatrix} x_1^{[1]} & x_2^{[1]} & \cdots & x_m^{[1]} \\ x_1^{[2]} & x_2^{[2]} & \ddots & x_m^{[2]} \\ \vdots & \vdots & \vdots & \vdots \\ x_1^{[n]} & x_2^{[n]} & \cdots & x_m^{[n]} \end{bmatrix}$$

1.1.2. Types of Machine Learning algorithms

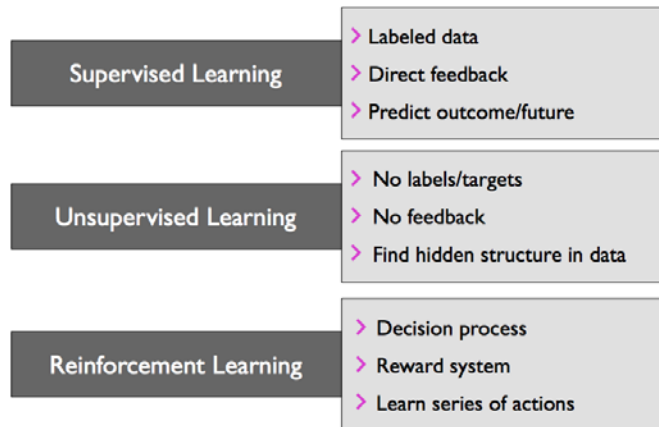


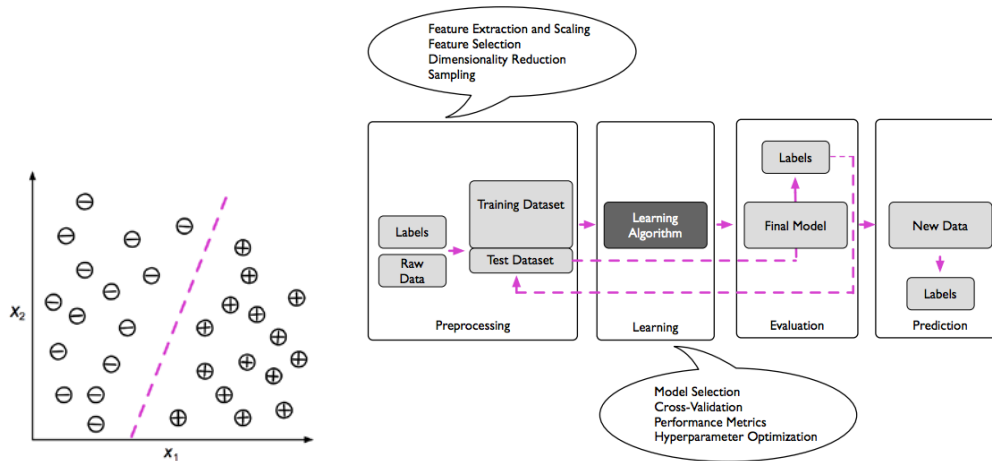
Fig. 1.2. Types of machine learning algorithms

There are four types of machine learning algorithms:

- Supervised learning
- Unsupervised learning
- Reinforcement learning
- Semi supervised learning

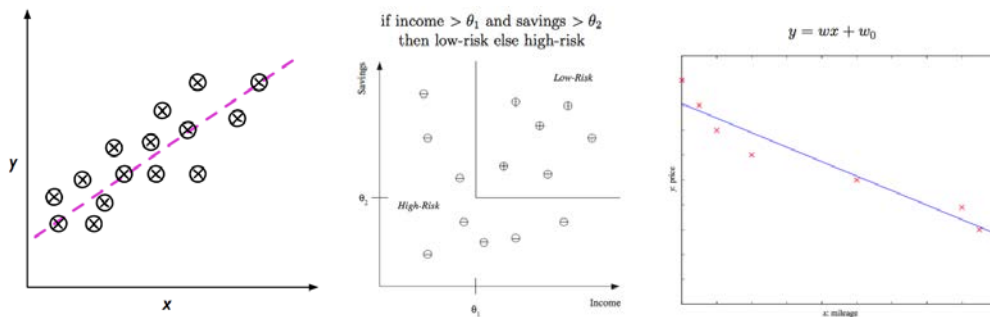
SUPERVISED LEARNING

Supervised learning is the subcategory of machine learning that focuses on learning a classification or regression model, that is, learning from labeled training data (i.e., inputs that also contain the desired outputs or targets; basically, “examples” of what we want to predict).



Binary Classification (Either +/- or 1/0 or Black/White)

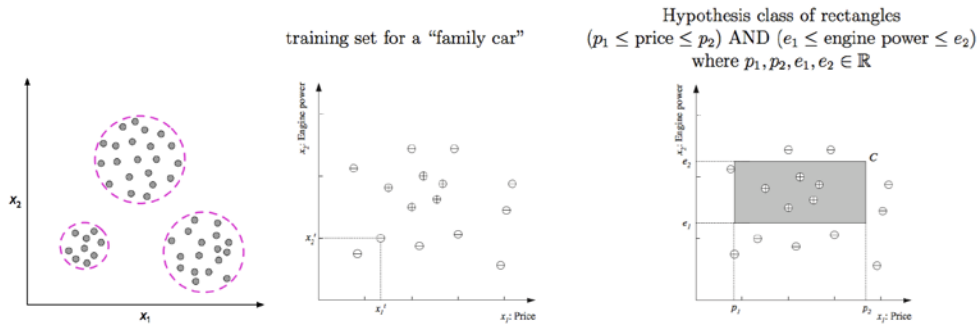
Input representation: we need to decide what attributes (features) to use to describe the input patterns (examples, instances). This implies ignoring other attributes as irrelevant.



The dashed-line indicates the functional form of the LINEAR REGRESSION model. CLASSIFICATION of the dataset by features e.g: Income and Savings in a home budget or a vehicle Mileage and Price incurred for a travel.

UNSUPERVISED LEARNING

In contrast to supervised learning, unsupervised learning is a branch of machine learning that is concerned with unlabeled data. Common tasks in unsupervised learning are clustering analysis (assigning group memberships) and dimensionality reduction (compressing data onto a lower-dimensional subspace or manifold).

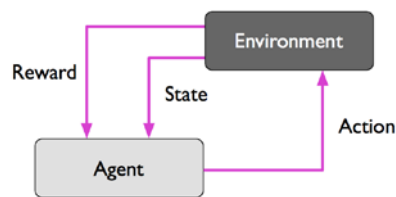


CLUSTERING, where the dashed lines indicate potential group membership assignments of unlabeled data points.

REINFORCEMENT LEARNING

Reinforcement is the process of learning from rewards while performing a series of actions. In reinforcement learning, we do not tell the learner or agent, for example, a robot, which action to take but merely assign a reward to each action and/or the overall outcome. Instead of having "correct/false" label for each step, the learner must discover or learn a behavior that maximizes the reward for a series of actions. In that sense, it is not a supervised setting and somewhat related to unsupervised learning; however, reinforcement learning really is its own category of machine learning.

Typical applications of reinforcement learning involve playing games (chess, Go, Atari video games) and some form of robots, e.g., drones, warehouse robots, and more recently self-driving cars.



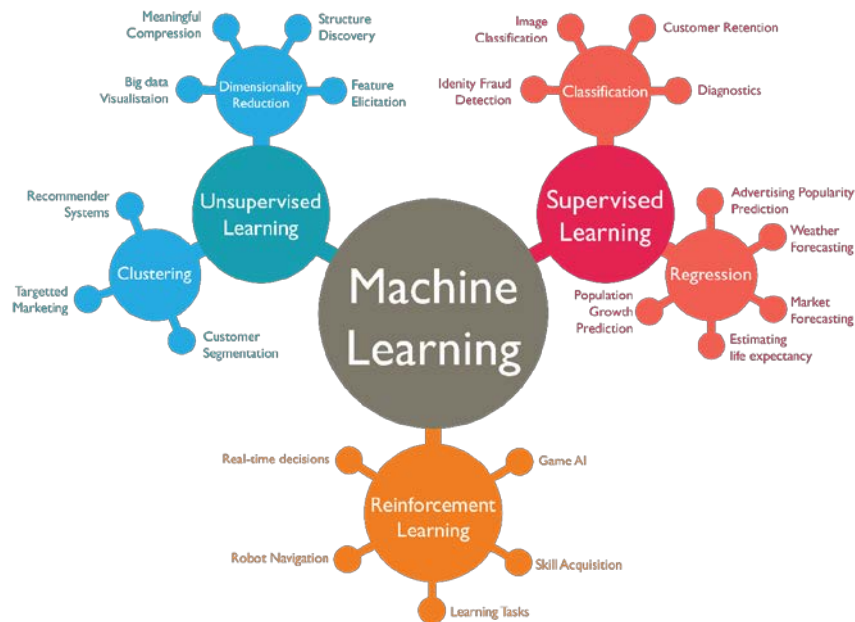
SEMI - SUPERVISED LEARNING

Roughly speaking, semi-supervised learning can be described as a mix between supervised and unsupervised learning. In semi-supervised learning tasks, some training examples contain outputs, but some do not. We then use the labeled training subset to label the unlabeled portion of the training set, which we then also utilize for model training.

Applications of Machine Learning

After the field of machine learning was “founded” more than a half a century ago, we can now find applications of machine learning in almost every aspect of our life. Popular applications of machine learning include the following:

- Email spam detection
- Face detection and matching (*e.g., iPhone X*)
- Web search (*e.g., DuckDuckGo, Bing, Google*)
- Sports predictions
- Post office (*e.g., sorting letters by zip codes*)
- ATMs (*e.g., reading checks*)
- Credit card fraud
- Stock predictions
- Smart assistants (*e.g., Apple Siri, Amazon Alexa, . . .*)
- Product recommendations (*e.g., Netflix, Amazon*)
- Self-driving cars (*e.g., Uber, Tesla*)
- Language translation (*Google translate*)
- Sentiment analysis
- Drug design
- Medical diagnoses
- ...



*Fig. 1.3. Types and Application of Machine Learning Algorithms**

It is a good exercise to think about how machine learning could be applied in these problem areas or tasks listed above:

- What is the desired outcome?
- What could the dataset look like?
- Is this a supervised or unsupervised problem, and what algorithms would you use?
- How would you measure success?
- What are potential challenges or pitfalls?

1.2. LEARNING PROBLEMS

A machine learning problem is said to be well-posed (well defined) if a unique solution exists, provided that the solution depends on the data / experience but it is not sensitive to (reasonably small) changes in the data / experience.

A problem is well defined if it possesses the following features:

- class of tasks (T)
- measure of performance to be improved (P)
- source of experience (E)

Examples of well posed problems:

1) A robot driving learning problem

- ✓ **Task T:** driving on public, 4-lane highway using vision sensors
- ✓ **Performance measure P:** average distance travelled before an error
- ✓ **Training experience E:** a sequence of images and steering commands recorded while observing a human driver

2) Checkers learning problem:

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

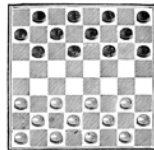


Fig. 1.4. Checker's Game

3) Spam Mail detection learning problem

- ✓ **Task T:** To recognize and classify mails into 'spam' or 'not spam'.
- ✓ **Performance measure P:** Total percentage of mails being correctly classified as 'spam' (or 'not spam') by the program.
- ✓ **Training experience E:** A set of mails with given labels ('spam' / 'not spam').

4) 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

1.2.1. Designing a Learning System

The factors that are very much vital for designing a learning system are:

- Choosing the training experience: determining the type of training
- Choosing the Target Function: exact type of knowledge to be learned
- Choosing a representation for the Target Function: representation for this target knowledge
- Choosing an approximation algorithm for the Target Function: learning mechanism / algorithm

Choosing the training experience

- Determining what type of training given to the system is a key factor in achieving desirable performance.
- Choice must be made between direct and indirect feedback given to the learning system. This feedback is the knowledge gaining from current training step that is fed into the next step.
- The learner or learning agent can decide on the next step either through knowledge gained from previous experience or make new choices to explore the space. This depends on the degree of learner's control over the board.
- The distribution of examples or data presented to the system to learn must also be studied before designing a learning system.

Choosing a target function

- Target function is determined based on the type of knowledge that is to be learned from the system.
- The operational and non- operational profiles of the learning agent accumulated over time will form the target function.
- These profiles are devised by evaluating the states and activities of the agent in its environment.
- It is not always possible to form the exact target function. Hence an approximation is applied to target function which is termed as **function approximation**.

Example: Consider the game of playing chess

- ✓ Task T: playing chess games
- ✓ Performance measure P: percentage of games won against opponents
- ✓ Training experience E: playing practice games against itself

The target function of chess game is to find the next move of an entity from a given board position: **Target function (V): Board \rightarrow Move**

Representation of target function

The way how the perceived target function is expressed depends on the choice of the problem and developer. Common representations include:

- ✓ Table with distinct entries for each state
- ✓ Rules that match features of the state
- ✓ Polynomial functions depicting the states
- ✓ Artificial neural network

The chosen target function V' should be close to the original function V . In case of chess game, the target function V' is:

$$V' = w_0 + w_1x_1 + w_2x_2 + w_3x_3 + w_4x_4 + w_5x_5 + w_6x_6$$

x_1 : count of black pieces on the board

x_2 : count of red pieces on the board

x_3 : count of black kings on the board

x_4 : count of red kings on the board

x_5 : count of black pieces threatened by red

x_6 : count of red pieces threatened by black

The variables w_0, w_1, \dots, w_6 are weights.

Choosing a function approximation algorithm

The training algorithm learns/approximate the coefficients w_0, w_1, \dots, w_6 with the help of the training examples by estimating and adjusting these weights. The target function is coined based on the set of training examples that best describes the specific state b and the training value $V_{\text{train}}(b)$ for b .

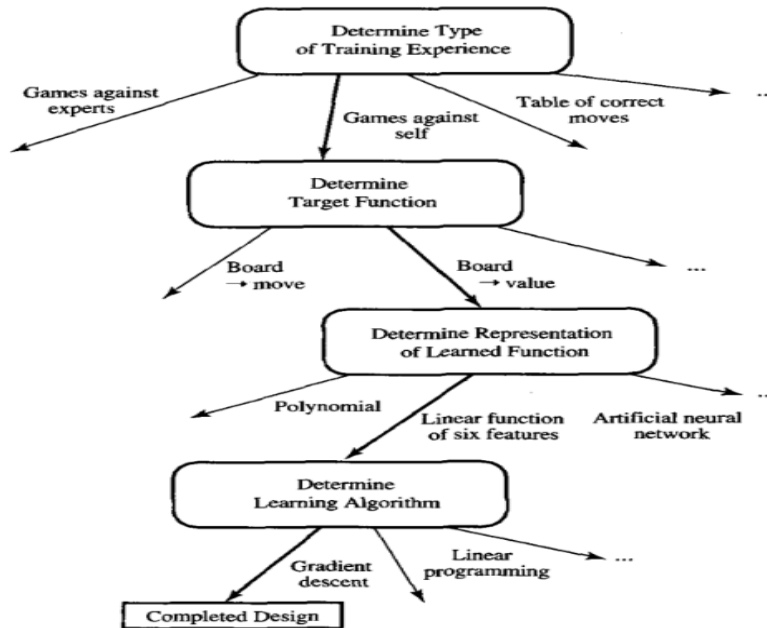


Fig. 1.5. Steps in designing a learning system

Final design

The final system design has the following four major components:

- ✓ **Performance System:** To select its next move at each step is determined by the learned p evaluation function. The performance to improve as this evaluation function becomes increasingly
- ✓ **Critic:** This takes as input the history or trace of the game and produces output a set of training examples of the target function.
- ✓ **Generalizer:** This 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.
- ✓ **Experiment Generator:** This 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.

1.3. PRESPECTIVES AND ISSUES IN MACHINE LEARNING

Machine learning has many insights and viewpoints:

- It is involved with larger space and capable of handling “Big data”.
- It requires merger prior knowledge about the learning problem.
- It looks on the underlying structures and patterns of the data.
- The model can be trained on the available data and later can be generalized to work on unseen examples called test data.

Issues in Machine Learning

The following are some of the issues in machine learning:

- What algorithms exist for learning general target functions from the 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? P
- 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?

Apart from answering the above mentioned questions, there are other out of box issues like:

- ✓ Ethics and legal issues
- ✓ Lack of data. In other words it requires large amounts of hand-crafted, structured training data
- ✓ Installation of sensors that would adversely affect the environment
- ✓ Chances of misapplication
- ✓ Degree of interpretability and interoperability
- ✓ Each narrow application needs to be specially trained
- ✓ Learning must generally be supervised: Training data must be tagged
- ✓ Require lengthy offline/ batch training
- ✓ Do not learn incrementally or interactively, in real-time
- ✓ Poor transfer learning ability, reusability of modules, and integration
- ✓ Systems are opaque, making them very hard to debug
- ✓ Performance cannot be audited or guaranteed at the long run
- ✓ They encode correlation, not causation or ontological relationships
- ✓ Do not encode entities or spatial relationships between entities
- ✓ Only handle very narrow aspects of natural language
- ✓ Not well suited for high-level, symbolic reasoning or planning.

1.4. CONCEPT LEARNING

Concept learning can be formulated as a problem of searching through a predefined space of potential hypotheses for the hypothesis. It can also be through as the process of evolving a Boolean or approximated Boolean valued function from the trained input and output of the problems.

This is the first step towards building machine learning algorithms. It is acquiring the definition of a general category from given sample positive and negative training examples of the category. The hypothesis or search space has a general-to-specific ordering of hypotheses, and the search can be efficiently organized by taking advantage of a naturally occurring structure or patterns over the data in the hypothesis space.

Example:

Consider the example task of learning the target concept "Days on which *Aldo* enjoys his favorite water sport"

Example	Sky	Air Temp	Humidity	Wind	Water	Forecast	Enjoy Sport
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	Cool	Change	Yes

Table: Positive and negative training examples for the target concept *Enjoy Sport*.

The task is to learn to predict the value of *Enjoy Sport* for an arbitrary day, based on the values of its other attributes?

What hypothesis representation is provided to the learner?

- Let's consider a simple representation in which each hypothesis consists of a conjunction of constraints on the instance attributes.
- Let each hypothesis be a vector of six constraints, specifying the values of the six attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*.

For each attribute, the hypothesis will either

- Indicate by a "?" that any value is acceptable for this attribute,
- Specify a single required value (e.g., Warm) for the attribute, or
- Indicate by a " Φ " that no value is acceptable

If some instance \mathbf{x} satisfies all the constraints of hypothesis \mathbf{h} , then \mathbf{h} classifies \mathbf{x} as a positive example ($\mathbf{h}(\mathbf{x}) = 1$).

The hypothesis that **PERSON** enjoys his favorite sport only on cold days with high humidity is represented by the expression

$$(? , \text{Cold}, \text{High}, ? , ? , ?)$$

The most general hypothesis-that every day is a positive example-is represented by

$$(? , ? , ? , ? , ? , ?)$$

The most specific possible hypothesis-that no day is a positive example-is represented by

$$(\Phi , \Phi , \Phi , \Phi , \Phi , \Phi)$$

Notation

- The set of items over which the concept is defined is called the *set of instances*, which is denoted by x

Example: x is the set of all possible days, each represented by the attributes: Sky, AirTemp, Humidity, Wind, Water, and Forecast

- The concept or function to be learned is called the *target concept*, which is denoted by c . c can be any Boolean valued function defined over the instances x

$$c: x \rightarrow \{0, 1\}$$

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).

- Instances for which $c(x) = 1$ are called *positive examples*, or members of the target concept.
- Instances for which $c(x) = 0$ are called *negative examples*, or non-members of the target concept.
- The ordered pair $(x, c(x))$ to describe the training example consisting of the instance x and its target concept *value* $c(x)$.
- D to denote the set of available training examples
- The symbol H to denote the set of all possible hypotheses that the learner may consider regarding the identity of the target concept. Each hypothesis h in H represents a Boolean-valued function defined over x .

The goal of the learner is to find a hypothesis h such that $h(x) = c(x)$ for all x in x .

- 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),
 - *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*. The constraints may be "?" (any value is acceptable), " Φ " (no value is acceptable), or a specific value.
 - 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 .

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

$h_1 = (\text{Sunny}, ?, ?, \text{Strong}, ?, ?)$

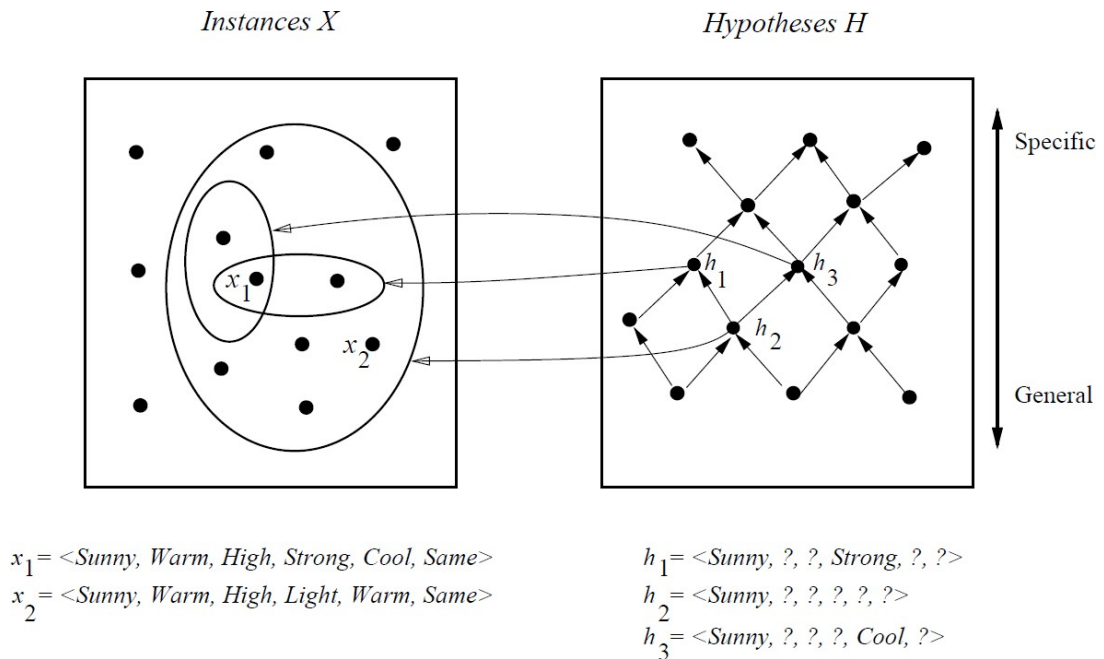
$h_2 = (\text{Sunny}, ?, ?, ?, ?, ?)$

- Consider the sets of instances that are classified positive by h_1 and by h_2 .
- h_2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h_1 will also be classified positive by h_2 . Therefore, h_2 is more general than h_1 .

Given hypotheses h_j and h_k , h_j is *more-general-than or-equal-to* h_k if and only if any instance that satisfies h_k also satisfies h_j .

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

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



- In the figure above, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X —the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the *more-general-than* relation, with the arrow pointing toward the less general hypothesis.
- Note the subset of instances characterized by h_2 subsumes the subset characterized by h_1 , hence h_2 is *more-general-than* h_1 .

Inductive Learning Hypothesis

This involves the process of learning by example where a system tries to induce a general rule from a set of observed instances. This is different from conceptual learning where 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 guarantee that the output hypothesis fits the target concept over the training data. The best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data. This is **inductive learning**.

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. A computer program is said to learn from experience E with respect.

1.5. VERSION SPACES AND CANDIDATE ELIMINATIONS

1.5.1. Version Space

A version space is a hierarchical representation of knowledge that enables you to keep track of all the useful information supplied by a sequence of learning examples without remembering any of the examples.

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.

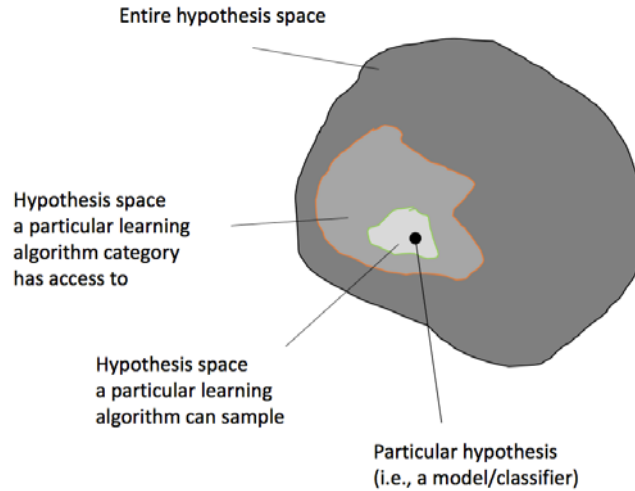
T, as measured by P, improves with experience E.

T, as measured by P, improves with experience E.

The 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 probable versions of the target concept. Given a set of training examples, any concept consistent with them must include every positive and every negative instance.

Consider a binary classification problem. Let D be a set of training examples and H a hypothesis space for the problem. The *version space* for the problem with respect to the set D and the space H is the set of hypotheses from H consistent with D; that is, it is the set

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



Selecting a hypothesis from a hypothesis space (from Entire hypotheses, H called General (G) to Particular hypothesis h named Specific (S)).

Assume we are given a dataset with 4 features and 3 class labels, the class label ($y \in \{\text{Setosa, Versicolor, Virginica}\}$). Also, assume all features are binary. Given 4 features with binary values (True, False), we have $2^4 = 16$ different feature combinations (see table below). Now, of the 16 rules, we have three classes to consider (Setosa, Versicolor, Virginica). Hence, we have $3^{16} = 43,046,721$ potential combinations of 16 rules that we can evaluate (this is the size hypothesis space, $|H| = 43,046,721$)!

sepal length <5 cm	sepal width <5 cm	petal length <5 cm	petal width <5 cm	Class Label
True	True	True	True	Setosa
True	True	True	False	Versicolor
True	True	False	True	Setosa
...	

Example of decision rules for the Iris flower data dataset.

Version space method involves identifying all concepts consistent with a set of training examples. This can be implemented incrementally, one example at a time.

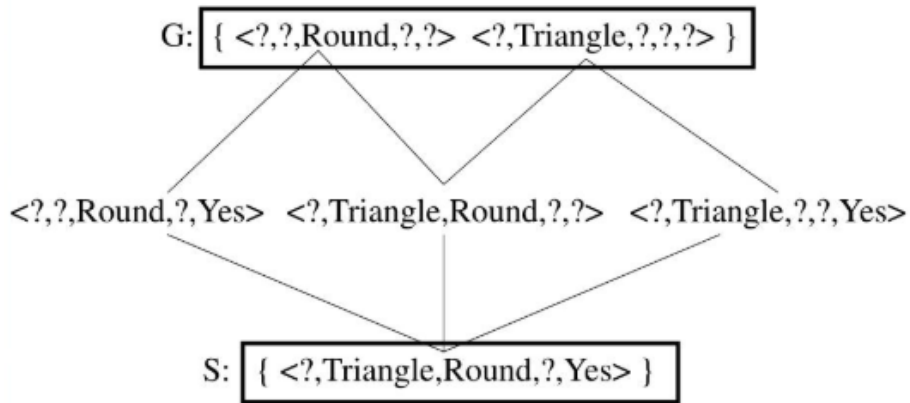
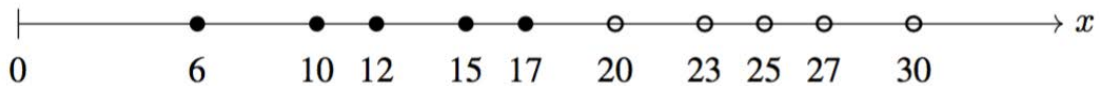


Fig. 1.6. Example Version Space

Example:

Consider the set of observations of a variable 'x' with the associated 'class' labels



From Figure above we can easily see that the hypothesis space with respect this dataset D and hypothesis space H is as given below:

Version space, $VS_{D,H} = \{h_m: 17 < m \leq 20\}$

1.5.2. Representation of version space (LIST-THEN-ELIMINATE)

- Simple way to represent the version space is to list all the members in the version space.
- An algorithm to do so is termed as LIST-THEN-ELIMINATE learning algorithm.

x	27	15	23	20	25	17	12	30	6	10
Class	1	0	1	1	1	0	0	1	0	0

- This algorithm initializes the version space to contain the entire hypothesis H. It then eliminated all the inconsistent hypothesis by comparing it with the given training examples.

- Iteratively, this procedure will shrink the version space and will finally remain with one single hypothesis that is consistent with the training examples. This is the **target hypothesis or target concept**.
- When there are no sufficient training examples, then this algorithm will return either entire hypothesis space or set or hypothesis as target.
- The merit of this method is that the algorithm is guaranteed to converge to a result.
- The drawbacks includes: exhaustive searching of all the hypothesis and can be applied to a finite space.

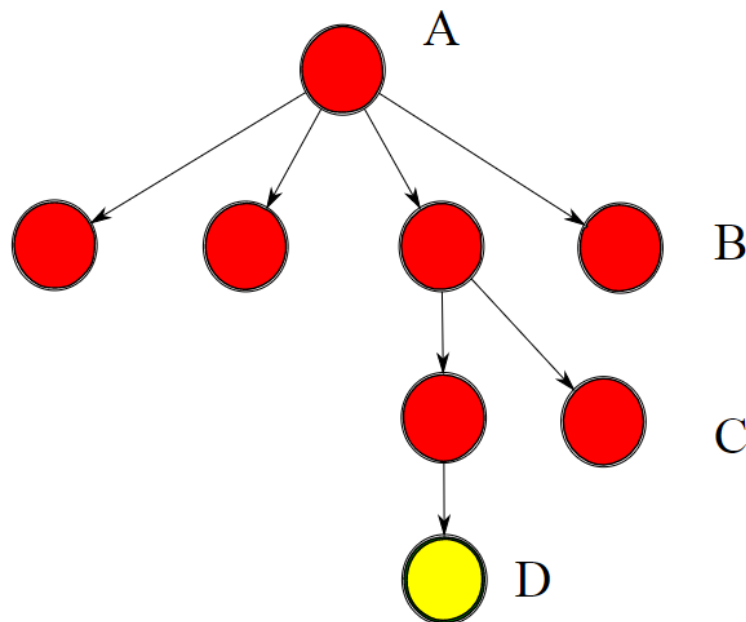


Fig. 1.7. The problem A is enumerating many hypothesis in the levels B and C.

The LIST-THEN-ELIMINATE algorithm, returns D as target hypothesis

Algorithm for LIST-THEN-ELIMINATE

1. Version_Space \leftarrow list containing every hypothesis in H
2. For each training example, (x, C(x))
Remove from Version_Space any hypothesis h, where $h(c) \neq c(x)$
3. Return the list of hypothesis in Version_Space.

1.5.3. Candidate Elimination Algorithm

- This algorithm returns the consistent hypothesis for the given training examples.
- The candidate elimination algorithm incrementally builds the version space given a hypothesis space H and a set E of examples.
- The examples are added one by one; each example possibly shrinks the version space by removing the hypotheses that are inconsistent with the example.
- The candidate elimination algorithm does this by updating the general and specific boundary for each new example.
- It begins by initializing the version space to the set of all hypotheses in H . This process is otherwise termed initializing the **G (General Boundary)** boundary set that contains the most general hypothesis in H .

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

- On the other hand, a **specific boundary set (S)** is initialized to contain the most specific or least general hypothesis. Initially S will look like:

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

- These two boundary sets delimit the entire hypothesis space, because all the hypothesis listed in H will be more general than G_0 but more specific than S_0 .
- For every training example, S and G set will be generalised and specialized, thus eliminating the inconsistent hypothesis from H .
- Finally, the set H will contain only the hypothesis that are consistent with training examples.

Algorithm for Candidate_Elimination

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

Remove from G any hypothesis inconsistent with d ,

For each hypothesis s in S that is not consistent with d ,-

Remove s from S

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

Remove from S any hypothesis that is more general than another

hypothesis in S

If d is a negative example

Remove from S any hypothesis inconsistent with d

For each hypothesis g in G that is not consistent with d

Remove g from G

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

Remove from G any hypothesis that is less general than another

hypothesis in G

- The version space learned by the Candidate Elimination Algorithm will converge toward the hypothesis that correctly describes the target concept provided:
 - (1) There are no errors in the training examples
 - (2) There is some hypothesis in H that correctly describes the target concept.
- Convergence can be speeded up by presenting the data in a strategic order.
- The best examples are those that satisfy exactly half of the hypotheses in the current version space.
- Version-Spaces can be used to assign certainty scores to the classification of new examples

Example of Candidate elimination algorithm

To learn the concept of “Japanese Economy Car” from the features < Country of Origin, Manufacturer, Color, Decade, Type >

Origin	Manufacturer	Color	Decade	Type	Example Type
Japan	Honda	Blue	1980	Economy	Positive
Japan	Toyota	Green	1970	Sports	Negative
Japan	Toyota	Blue	1990	Economy	Positive
USA	Chrysler	Red	1980	Economy	Negative
Japan	Honda	White	1980	Economy	Positive
Japan	Toyota	Green	1980	Economy	Positive
Japan	Honda	Red	1990	Economy	Negative

1. Positive Example 1: <Japan, Honda, Blue, 1980, Economy>

Initialize G to a singleton set that includes everything.

$$G = \{ (?, ?, ?, ?, ?) \}$$

Initialize S to a singleton set that includes the first positive example.

$$S = \{ (\text{Japan}, \text{Honda}, \text{Blue}, 1980, \text{Economy}) \}$$

2. Negative Example 2: <Japan, Toyota, Green, 1970, Sports>

Specialize G to exclude the negative example.

$$G = \{ (?, \text{Honda}, ?, ?, ?), (?, ?, \text{Blue}, ?, ?), (?, ?, ?, 1980, ?), (?, ?, ?, ?, \text{Economy}) \} \\ S = \{ (\text{Japan}, \text{Honda}, \text{Blue}, 1980, \text{Economy}) \}$$

3. Positive Example 3: <Japan, Toyota, Blue, 1990, Economy>

Prune G to exclude descriptions inconsistent with the positive example.

$$G = \{ (?, ?, \text{Blue}, ?, ?), (?, ?, ?, ?, \text{Economy}) \}$$

Generalize S to include the positive example.

$$S = \{ (\text{Japan}, ?, \text{Blue}, ?, \text{Economy}) \}$$

4. Negative Example <USA, Chrysler, Red, 1980, Economy>

Specialize G to exclude the negative example (but stay consistent with S)

$$G = \{ (?, ?, \text{Blue}, ?, ?), (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

$$S = \{ (\text{Japan}, ?, \text{Blue}, ?, \text{Economy}) \}$$

5. Positive Example: <Japan, Honda, White, 1980, Economy>

Prune G to exclude descriptions inconsistent with positive example.

$$G = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

Generalize S to include positive example.

$$S = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

6. Positive Example: <Japan, Toyota, Green, 1980, Economy>

New example is consistent with version-space, so no change is made.

$$G = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

$$S = \{ (\text{Japan}, ?, ?, ?, \text{Economy}) \}$$

7. Negative Example: <Japan, Honda, Red, 1990, Economy>

Example is inconsistent with the version-space.

G cannot be specialized.

S cannot be generalized.

The version space collapses. No conjunctive hypothesis is consistent with the data set.

1.6. INDUCTIVE BIAS

Inductive bias is the set of assumptions a learner uses to predict results given inputs it has not yet encountered.

- Inductive reasoning is the process of learning general principles on the basis of specific instances.
- This is done by all the machine learning algorithms to produce predictions for any unseen test instance based on the knowledge it had obtained from the finite number of training instances.
- Inductive bias describes the tendency for a system to prefer a certain set of generalizations over others that are equally consistent with the observed data.

Steps in Inductive Learning:

Inducing a general function from training examples:

1. Construct hypothesis from the training example.
2. A hypothesis is consistent if it agrees with all training examples.
3. A hypothesis said to generalize well if it correctly predicts the value of y for novel example.

This is inductive learning.

Lesson on Machine Learning interpretability

- **Rashomon effect:** The multiplicity of good models. Often we have multiple good models that fit the data well. If we have different models that all fit the data well, which one should we pick?

- **Occam's razor:** While we prefer favoring simple models, there is usually a conflict between accuracy and simplicity to varying degree.
- **Bellman and the “curse of dimensionality”** Usually, having more data is considered a good thing (i.e., more information). However, more data can be harmful to a model and make it more prone to over fitting (fitting the training data too closely and not generalizing well to new data that was not seen during training; fitting noise). Note that the curse of dimensionality refers to an increasing number of feature variables given a fixed number of training examples. Some models have smart workarounds for dealing with large feature sets. E.g., Breiman's random forest algorithm, which partitions the feature space to fit individual decision trees that are then later joined to a decision tree ensemble (the particular algorithm is called random forest).

Also note that there's a “*No Free Lunch*” theorem for machine learning, meaning that there's no single best algorithm that works well across different problem domains.

1.6.1. Biased Hypothesis Space

- The version space of a problem contains all possible hypothesis. Only the hypothesis that agree with the training examples will qualify as target hypothesis.
- The target hypothesis is the generalization of the learning process.
- When a novel, unknown training example is presented to the target hypothesis, it should give right predictions.
- But when a more specific, negative example is presented to the target hypothesis, it cannot predict the result, since it has not been trained over that kind of examples. This phenomenon is called **biased hypothesis**.
- This necessitates the formulation of more expressive hypothesis.
- A biased hypothesis leads to biased learner.

1.6.2. 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

- The target hypothesis should be capable of representing every possible subset of the instances X . The set of all subsets of a set X is called the **powerset of X** .
- The Enjoysport learning task can be reformulated in an unbiased way by defining a new hypothesis space H' that can represent every subset of instances.
- Let H' correspond to the power set of X . This can be formed by including arbitrary disjunctions, conjunctions, and negations of the hypotheses.
- For example, the target concept "Sky = Sunny or Sky = Cloudy" could then be described as: (Sunny, ?, ?, ?, ?, ?) \vee (Cloudy, ?, ?, ?, ?, ?).
- In this way, the more learning can happen. But the intuitive problem is the learning algorithm cannot learn beyond the examples or in other words it cannot generalise beyond the observed examples.
- The S boundary (Specific boundary) of the version space will contain the hypothesis which is the disjunction of the positive examples.
- The G boundary (General boundary) will consist of the hypothesis that rules out only the observed negative examples.
- The problem here is that with this very expressive hypothesis representation, the S boundary will always be simply the disjunction of the observed positive examples, while the G boundary will always be the negated disjunction of the observed negative examples.
- Therefore, the only examples that will be unambiguously classified by S and G are the observed training examples themselves.
- Inorder to converge to a single, final target concept, we will have to present every single instance in X as a training example, which is very difficult.

1.6.3. The Futility of Bias-Free Learning

- The important inference obtained is that learner that makes no a priori assumptions regarding the identity of the target concept has no rational basis for classifying any unseen instances.
- The Candidate Learning can generalize beyond training examples since it was biased by the implicit assumption that the target concept could be represented by a conjunction of attribute values.

- In cases where this assumption is correct its classification of new instances will also be correct. If this assumption is incorrect, the algorithms will misclassify at least some instances from X .
- The key idea of inductive learning is that the policy by which the learner generalizes beyond the observed training data, to infer the classification of new instances.

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

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

- 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.
- 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 $\log_2 |\text{VersionSpace}|$ experiments.
- 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.

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	?

- 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.

- 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 .
 - 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.
- From these theory we infer the formal definition of inductive bias as:

Consider a concept learning algorithm L for the set of instances X . Let c be an arbitrary concept defined over X , and let $D_c = \{\langle x, c(x) \rangle\}$ be an arbitrary set of training examples of c . Let $L(x_i, D_c)$ denote the classification assigned to the instance x_i by L after training on the data 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 D_c the following formula holds

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

- Reformulating the 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.

Inductive Bias: the target concept can be represented in its hypothesis space.

1.7. DECISION TREE LEARNING

It is one of the widely adopted inductive learning method. It is a method for approximating discrete-valued functions that is robust to noisy data and capable of learning disjunctive expressions. Learned trees can also be represented as sets of if-then rules to improve human readability.

1.7.1. Representing Decision Trees

A decision tree consists of:

- ✓ **Nodes:** test for the value of a certain attribute
- ✓ **Edges:** correspond to the outcome of a test connect to the next node or leaf
- ✓ **Leaves:** terminal node that predict the outcome

- 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.
- Decision trees can be constructed for both classification as well as regression based problems.
- A decision tree is drawn upside down with its root at the top.
- The condition/internal node is the location where the tree splits into branches/edges.
- The end of the branch that doesn't split anymore is the decision/leaf.

- **Recursive Binary Splitting:** All the features are considered and different split points are tried and tested using a cost function. The split with the best cost (or lowest cost) is selected. This is the strategy followed in construction of decision tree.

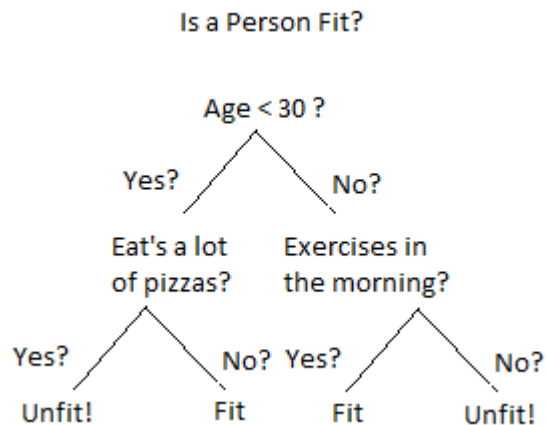


Fig. 1.8. A decision tree for finding the fitness of a person

- The following are the rules that could be inferred from decision tree given in Fig 1.8:
 - ✓ $\langle \text{Age} < 30 \wedge \text{Eats lots of pizzas} \rangle$ is unfit
 - ✓ $\langle \text{Age} < 30 \wedge \text{Do not Eats lots of pizzas} \rangle$ is fit
 - ✓ $\langle \text{Age} > 30 \wedge \text{Exercises in the morning} \rangle$ is fit
 - ✓ $\langle \text{Age} > 30 \wedge \text{Do not exercise in the morning} \rangle$ is unfit
- Decision trees will be a good option if the following conditions are met:
 - ✓ Instances are represented by attribute-value pairs
 - ✓ Disjunctive descriptions may be required.
 - ✓ The training data may contain errors
 - ✓ The training data may contain missing attribute values.

1.7.2. Decision tree algorithm

The ID3 (Iterative Dichotomiser 3) is a primitive decision tree algorithm. It deploys enhanced greedy search algorithm that implement heuristic function. The probability is used as comparison metric. This tree do not support backtracking.

- The construction of tree starts with deciding which attribute should be tested at the root.
- Each instance attribute is evaluated using a statistical test to determine how well it alone classifies the training examples. The best attribute is selected and used as the test at the root node of the tree.
- A descendant of the root node is then created for each possible value of this attribute, and the training examples are sorted to the appropriate descendant node.
- The entire process is then repeated using the training examples associated with each descendant node to select the best attribute to test at that point in the tree.
- This forms a greedy search for an acceptable decision tree, in which the algorithm never backtracks to reconsider earlier choices.

Algorithm of ID3 Decision Tree

ID3(Examples, Target attribute, Attributes)

```
// Examples are the training examples.
// Target attribute is the attribute whose value is to be predicted by the tree.
//Attributes is a list of other attributes that may be tested by the learned decision
Tree.
//Returns a decision tree that correctly classifies the given Examples.
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
    At 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_{vi}, Targetattribute, Attributes – {A}))

End

Return Root

Choosing best classifier

ID3 algorithm for generating decision trees, uses the notion of **information gain** to choose the best classifier. Information gain is defined in terms of **entropy**, the fundamental quantity in information theory.

Information gain measures how well a given attribute separates the training examples according to their target classification. ID3 uses this information gain measure to select among the candidate attributes at each step while growing the tree.

Entropy characterizes the purity or impurity of an arbitrary collection of examples. Given a collection H, containing positive and negative examples of some target concept, the entropy (H(x)) of S relative to this boolean classification is:

$$H(x) = -p_+ \log_2 p_+ - p_- \log_2 p_-$$

Where p_+ is the proportion of positive examples in H and p_- is the proportion of negative examples in H.

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

Here p_i is the fraction of examples in the given class. Entropy controls how a Decision Tree decides to split the data. It actually effects how a decision tree draws its boundaries.

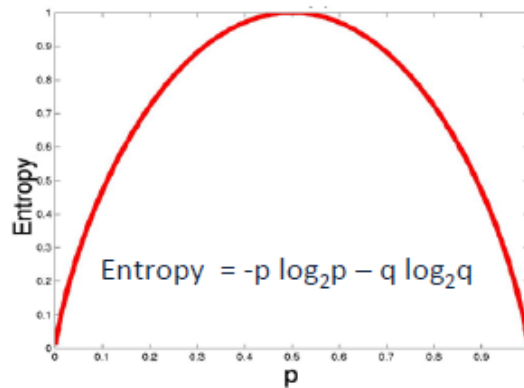


Fig. 1.9. Entropy function for Boolean classification (p -positive examples, q -negative examples)

Information gain and entropy

- The information gain is based on the decrease in entropy after a data-set is split on an attribute.
- Information gain measures how much information a feature gives us about the class. Information gain is the main key that is used by Decision Tree Algorithms to construct a Decision Tree.
- Decision Trees algorithm will always tries to maximize Information gain.
- An attribute with highest Information gain will tested/split first.
- The information gain (G) of an attribute y , relative to a collection of examples x is:

$$G(x, y) = H(x) - \sum_{i \in \text{vaule}(y)} \frac{|\Delta y_i|}{|\Delta y|} H(y_i)$$

Where $H(y_i)$ is the entropy.

Example:

Use ID3 algorithm to construct a decision tree for the data in Table

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

Solution: Note that, in the given data, there are four features but only two class labels (or, target variables), namely, “yes” and “no”.

Step 1: We first create a root node for the tree.

Step 2: Note that not all examples are positive (class label “yes”) and not all examples are negative (class label “no”). Also the number of features is not zero.

Step 3: We have to decide which feature is to be placed at the root node. For this, we have to calculate the information gains corresponding to each of the four features. The computations are shown below.

(i) *Calculation of Entropy (S)*

$$\begin{aligned}\text{Entropy (S)} &= -p_{\text{yes}} \log_2(p_{\text{yes}}) - p_{\text{no}} \log_2(p_{\text{no}}) \\ &= -(9/14) \times \log_2(9/14) - (5/14) \times \log_2(5/14)\end{aligned}$$

(ii) *Calculation of Gain (S, outlook)*

The values of the attribute “outlook” are “sunny”, “overcast” and “rain”. We have to calculate Entropy () for $v = \text{sunny}$, $v = \text{overcast}$ and $v = \text{rain}$.

$$\text{Entropy (S}_{\text{sunny}}) = -(3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5) = 0.9710$$

$$\text{Entropy (S}_{\text{overcast}}) = -(4/4) \times \log_2(4/4) = 0$$

$$\text{Entropy (S}_{\text{rain}}) = -(3/5) \times \log_2(3/5) - (2/5) \times \log_2(2/5) = 0.9710$$

$$\begin{aligned}
\text{Gain}(S, \text{outlook}) &= \text{Entropy}(S) - \left\{ \frac{|S_{\text{sunny}}|}{|S|} \times \text{Entropy}(S_{\text{sunny}}) \right\} \\
&\quad - \left\{ \frac{|S_{\text{overcast}}|}{|S|} \times \text{Entropy}(S_{\text{overcast}}) \right\} - \left\{ \frac{|S_{\text{rain}}|}{|S|} \times \text{Entropy}(S_{\text{rain}}) \right\} \\
&= 0.9405 - (5/14) \times 0.9710 \\
&= (4/14) \times 0 - (5/14) \times 0.9710 \\
&= 0.2469
\end{aligned}$$

(iii) *Calculation of Gain (S, temperature)*

The values of the attribute “temperature” are “hot”, “mild” and “cool”. We have to calculate Entropy (S_v) for $v = \text{hot}$, $v = \text{mild}$ and $v = \text{cool}$.

$$\text{Entropy}(S_{\text{hot}}) = -(2/4) \times \log_2(2/4) - (2/4) \times \log_2(2/4) = 1.0000$$

$$\text{Entropy}(S_{\text{mild}}) = -(4/6) \times \log_2(4/6) - (2/6) \times \log_2(2/6) = 0.9186$$

$$\text{Entropy}(S_{\text{cool}}) = -(3/4) \times \log_2(3/4) - (1/4) \times \log_2(1/4) = 0.8113$$

$$\begin{aligned}
\text{Gain}(S, \text{temperature}) &= \text{Entropy}(S) \left\{ \frac{|S_{\text{hot}}|}{|S|} \times \text{Entropy}(S_{\text{hot}}) \right\} \\
&\quad - \left\{ \frac{|S_{\text{mild}}|}{|S|} \times \text{Entropy}(S_{\text{mild}}) \right\} - \left\{ \frac{|S_{\text{cool}}|}{|S|} \times \text{Entropy}(S_{\text{cool}}) \right\} \\
&= 0.9405 - (4/14) \times 1.0000 - (6/14) \times 0.9186 - (4/14) \times 0.8113 \\
&= 0.0293
\end{aligned}$$

(iv) *Calculation of Gain (S, humidity) and Gain (S, wind)*

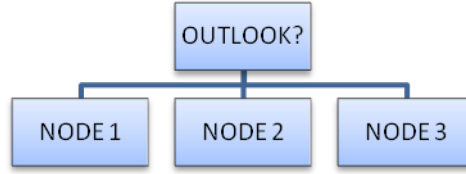
The following information gains can be calculated in a similar way:

$$\text{Gain}(S, \text{humidity}) = 0.151 \quad \text{Gain}(S, \text{wind}) = 0.048$$

Step 4: We find the highest information gain which is the maximum among $\text{Gain}(S, \text{outlook})$, $\text{Gain}(S, \text{temperature})$, $\text{Gain}(S, \text{humidity})$ and $\text{Gain}(S, \text{wind})$. Therefore, we have:

$$\begin{aligned}
\text{highest information gain} &= \max\{0.2469, 0.0293, 0.151, 0.048\} \\
&= 0.2469
\end{aligned}$$

This corresponds to the feature “outlook”. Therefore, we place “outlook” at the root node. We now split the root node into three branches according to the values of the feature “outlook” as in Figure below



Step 5: Let $S^{(1)} = S_{\text{outlook=sunny}}$. We have $|S^{(1)}| = 5$. The examples in $S^{(1)}$ are shown in figure below

Day	outlook	temperature	humidity	wind	playtennis
D1	sunny	hot	high	weak	no
D2	sunny	hot	high	strong	no
D8	sunny	mild	high	weak	no
D9	sunny	cool	normal	weak	yes
D11	sunny	mild	normal	strong	yes

Fig. 1.10. Training examples with outlook = “sunny”

$$\begin{aligned}
 \text{Gain}(S^{(1)}, \text{temperature}) &= \text{Entropy}(S) - \left\{ \frac{|S_{\text{temp=hot}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{temp=hot}}^{(1)}) \right\} \\
 &\quad - \left\{ \frac{|S_{\text{temp=mild}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{temp=mild}}^{(1)}) \right\} \\
 &\quad - \left\{ \frac{|S_{\text{temp=cool}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{temp=cool}}^{(1)}) \right\} \\
 &= [-(2/5) \log_2(2/5) - (3/5) \log_2(3/5)] - (2/5) \\
 &\quad \times [-(2/2) \log_2(2/2)] - (2/5) \times [-(1/2) \log_2(1/2) \\
 &\quad - (1/2) \log_2(1/2)] - (1/5) \times [-(1/1) \log_2(1/1)] \\
 &= 0.5709
 \end{aligned}$$

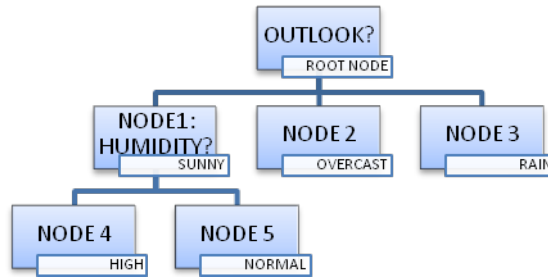
Similarly we find for humidity and wind

$$\begin{aligned}
 \text{Gain}(S^{(1)}, \text{humidity}) &= \text{Entropy}(S) - \left\{ \frac{|S_{\text{hum=high}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{hum=high}}^{(1)}) \right\} \\
 &\quad - \left\{ \frac{|S_{\text{hum=normal}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{hum=normal}}^{(1)}) \right\} \\
 &= [-(2/5) \log_2(2/5) - (3/5) \log_2(3/5)] - (3/5) \\
 &\quad \times [-(3/3) \log_2(3/3)]
 \end{aligned}$$

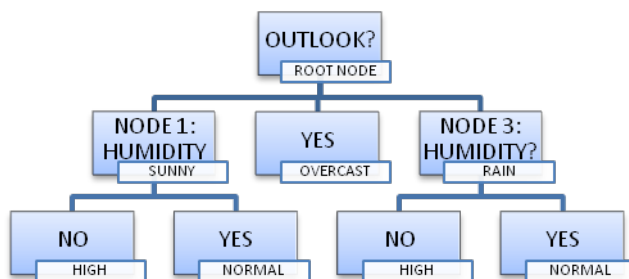
$$= 0.9709$$

$$\begin{aligned} \text{Gain}(S^{(1)}, \text{wind}) &= \text{Entropy}(S) - \left\{ \frac{|S_{\text{wind}=\text{weak}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{wind}=\text{weak}}^{(1)}) \right\} \\ &\quad - \left\{ \frac{|S_{\text{wind}=\text{strong}}^{(1)}|}{|S^{(1)}|} \times \text{Entropy}(S_{\text{wind}=\text{strong}}^{(1)}) \right\} \\ &= \left[-(2/5) \log_2(2/5) - (3/5) \log_2(3/5) \right] - (3/5) \\ &\quad \times \left[-(2/3) \log_2(2/3) - (1/3) \log_2(1/3) \right] \\ &= 0.0110 \end{aligned}$$

The maximum of $\text{Gain}(S(1), \text{temp})$, $\text{Gain}(S(1), \text{hum})$ and $\text{Gain}(S(1), \text{wind})$ is $\text{Gain}(S(1), \text{hum})$. Hence we place “humidity” at Node 1 and split this node into two branches according to the values of the feature “humidity” to get the tree in



Step 6: It can be seen that all the examples in the data set corresponding to Node 4 have the same class label “no” and all the examples corresponding to Node 5 have the same class label “yes”. So we represent Node 4 as a leaf node with value “no” and Node 5 as a leaf node with value “yes”. Similarly, all the examples corresponding to Node 2 have the same class label “yes”. So we convert Node 2 as a leaf node with value “yes”. Finally, let $S^{(2)} = S_{\text{outlook} = \text{rain}}$. The highest information gain for this data set is $\text{Gain}(S^{(2)}, \text{humidity})$. The branches resulting from splitting this node corresponding to the values “high” and “normal” of “humidity” lead to leaf nodes with class labels “no” and “yes”. With these changes, we get the tree in Figure below.



1.7.3. Heuristic 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

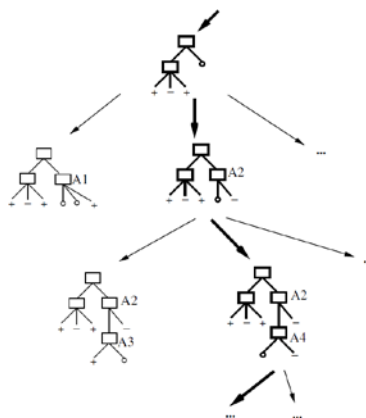


Fig. 1.11. Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic

By viewing ID3 in terms of its search space and search strategy, there are some insight into its capabilities and limitations as discussed below:

1. *ID3's hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree*

- ◆ ID3 avoids one of the major risks of methods that search incomplete hypothesis spaces: that the hypothesis space might not contain the target function.
2. ID3 maintains only a single current hypothesis as it searches through the space of decision trees.
 - a. For example, with the earlier version space candidate elimination method, which maintains the set of all hypotheses consistent with the available training examples.

By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.

- b. For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries that optimally resolve among these competing hypotheses
3. 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.
 - ◆ In the case of ID3, a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.
4. ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis.
 - ◆ One advantage of using statistical properties of all the examples is that the resulting search is much less sensitive to errors in individual training examples.
 - ◆ ID3 can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

Alternative Measures for Selecting Attributes

- ◆ The problem is if attributes with many values, Gain will select it ?
- ◆ Example: consider the attribute Date, which has a very large number of possible values. (e.g., March 4, 1979).

- ◆ If this attribute is added to the PlayTennis data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data.
- ◆ This decision tree with root node Date is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

One Approach: Use GainRatio instead of Gain

The gain ratio measure penalizes attributes by incorporating a split information, that is sensitive to how broadly and uniformly the attribute splits the data

$$\text{Gain Ratio (S, A)} \equiv \frac{\text{Gain (S, A)}}{\text{Split Information (S, A)}}$$

$$\text{Split Information (S, A)} \equiv - \sum_{i=1}^c \frac{|S_i|}{|S|} \log_2 \frac{|S_i|}{|S|}$$

Where, S_i is subset of S , for which attribute A has value v_i

Glossary:

Machine learning borrows concepts from many other fields and redefines what has been known in other fields under different names. Below is a small glossary of machine learning- specific terms along with some key concepts to help navigate the machine learning literature.

- Training example D : A row in the table representing the dataset. Synonymous to an observation, training record, training instance, training sample (in some contexts, sample refers to a collection of training examples).
- Training: model fitting, for parametric models similar to parameter estimation.
- Feature, x : a column in the table representing the dataset. Synonymous to predictor, variable, input, attribute.
- Target, y : Synonymous to outcome, output, response variable, dependent variable, (class) label, ground truth.
- Predicted output, \hat{y} : use this to distinguish from targets; here, means output from the model.

- **Loss function:** Often used synonymously with cost function; sometimes also called error function. In some contexts the loss for a single data point, whereas the cost function refers to the overall (average or summed) loss over the entire dataset.
- **Hypothesis, H:** A hypothesis is a certain function that we believe (or hope) is similar to the true function, the target function that we want to model. In context of spam classification, it would be a classification rule we came up with that allows us to separate spam from non-spam emails.
- **Model:** In the machine learning field, the terms hypothesis and model are often used interchangeably. In other sciences, they can have different meanings: A hypothesis could be the “educated guess” by the scientist, and the model would be the manifestation of this guess to test this hypothesis.
- **Learning algorithm:** Again, our goal is to find or approximate the target function, and the learning algorithm is a set of instructions that tries to model the target function using our training dataset. A learning algorithm comes with a hypothesis space, the set of possible hypotheses it explores to model the unknown target function by formulating the final hypothesis.
- **Classifier:** A classifier is a special case of a hypothesis (nowadays, often learned by a machine learning algorithm). A classifier is a hypothesis or discrete-valued function that is used to assign (categorical) class labels to particular data points. In an email classification example, this classifier could be a hypothesis for labeling emails as spam or non-spam. Yet, a hypothesis must not necessarily be synonymous to the term classifier. In a different application, our hypothesis could be a function for mapping study time and educational backgrounds of students to their future, continuous-valued, SAT scores – a continuous target variable, suited for regression analysis.
- **Hyper parameters:** Hyper parameters are the tuning parameters of a machine learning algorithm – for example, the regularization strength of an L2 penalty in the mean squared error cost function of linear regression, or a value for setting the maximum depth of a decision tree. In contrast, model parameters are the parameters that a learning algorithm fits to the training data – the parameters of the model itself. For example, the weight coefficients (or slope) of a linear regression line and its bias (or y-axis intercept) term are model parameters.

UNIT

2

NEURAL NETWORKS AND GENETIC ALGORITHMS

2.1. INTRODUCTION NERUAL NETWORKS

Bio inspired computing is gaining popularity because of its ability to solve many difficult problems. Neural network is one of the prominent brain child of bio inspired algorithms. Neural networks are designed with an aim of artificially imitating the working of human brain to solve the real world problems. They are otherwise known an **Artificial Neural Network (ANN)**.

- ANN is artificial representation of human brain that attempts to simulate the learning process.
- ANNs are constructed by interconnecting artificial neurons that shares the properties of biological neurons.

ANN is an interconnected group of artificial neurons that acts as a mathematical or computational model for information processing.

Definitions of neural network

- A neural network is a system composed of many simple processing elements working in parallel whose function is determined by network structure, connection strengths, and the processing performed at computing elements or nodes.
- A neural network is a massively parallel distributed processor that has a natural propensity for storing experiential knowledge and making it available for use. It resembles the brain in two respects:

- Knowledge is acquired by the network through a learning process.
- Interneuron connection strengths known as synaptic weights are used to store the knowledge
- A neural network is a circuit composed of a very large number of simple processing elements that are neurally based. Each element operates only on local information asynchronously.
- Artificial neural systems, or neural networks, are physical cellular systems which can acquire, store and utilize experiential knowledge.
- ANNs are an attempt to imitate the highly parallel processing capabilities of brain in handling distributed representations.
- Neural networks are a form of multiprocessor systems with the following properties:
 - Simple processing elements
 - High degree of interconnection
 - Simple messaging formats
 - Adaptive interaction between the elements.

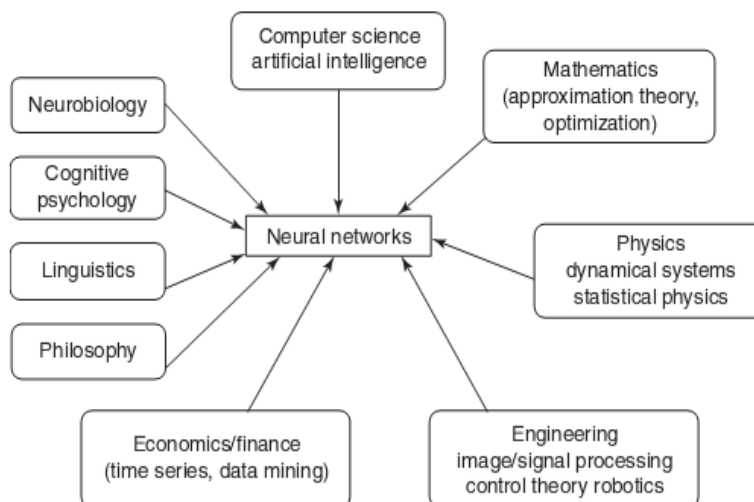


Fig. 2.1. Application areas of ANN

Advantages of Neural networks

The following are some of the advantages of neural networks:

- ✓ Massive parallelism
- ✓ Distributed representation
- ✓ Highly distributed computation
- ✓ Excellent Learning ability
- ✓ More generalized
- ✓ Highly adaptive nature
- ✓ Inherent contextual information processing
- ✓ Fault tolerance
- ✓ Low energy consumption

2.2. REPRESENTATION OF NEURAL NETWORKS

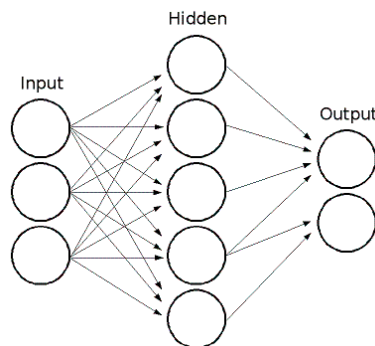


Fig. 2.2. Representation of neural network

There are many versions of neural network representation. The most common one is Multi Layer Perceptron (MLP) model, which has three layers. Each layer takes input from the previous layer and summed with the weights before passing it to next layer. The following are the layers:

- **Input Layer:** The initial data that is to be processed by the nodes or neurons. Each node will be connected to every other node in the next layer.
- **Hidden Layer:** Computational layer that lies in between input and output layer neurons.
- **Output Layer:** The final layer that is responsible for producing output.

Mathematical Representation of Neural Networks

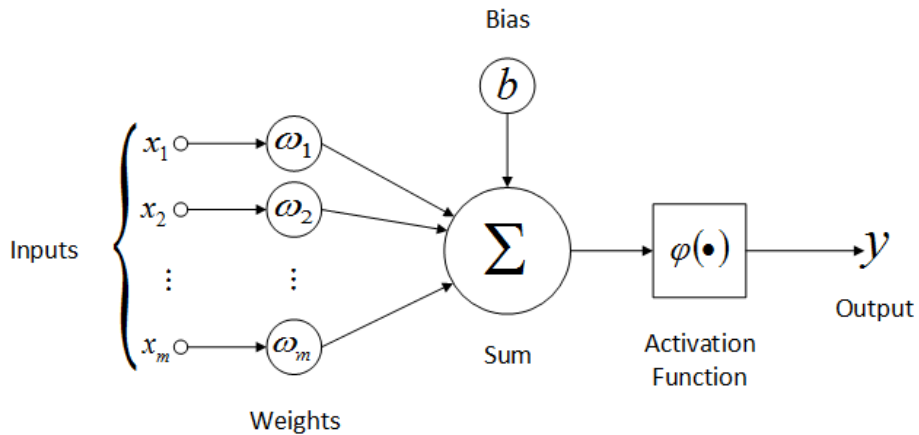


Fig. 2.3. Mathematical model of neural network

The input units (x_1, x_2, \dots, x_m) are connected with next layer by multiplying the weight vectors ($\omega_1, \omega_2, \dots, \omega_m$) and then summing them. The resultant is added with the bias value. Bias value is added to make the neurons adapt to realistic situations. Activation function is applied over this value which will yield the output value (y).

$$Y = \text{activation_function} \left(\sum_{i=0}^m x_i \omega_i + b \right)$$

$$Y = \text{activation} \left(\sum (\text{weight} * \text{input}) + \text{bias} \right)$$

Role of activation functions

- Activation functions are complicated and form Non-linear complex functional mappings between the inputs and response variable.
- They introduce non-linear properties to neural network.
- Their main purpose is to convert input signal of a node in an ANN to an output signal. That output signal now is used as input in the next layer in the stack.
- Specifically in ANN we do the sum of products of inputs(X) and their corresponding Weights (W) and apply a Activation function $f(x)$ to it to get the output of that layer and feed it as an input to the next layer.
- They basically decide whether a neuron should be activated or not. Whether the information that the neuron is receiving is relevant for the given information or should it be ignored.

- The activation function is the non-linear transformation done over the input signal. This transformed output is then send to the next layer of neurons as input.
- It is used to determine the output of neural network like yes or no. It maps the resulting values in between 0 to 1 or -1 to 1 etc. (depending upon the function).

2.3. PROBLEMS SOLVED BY NEURAL NETWORKS

The ability of humans to understand the learned target function is not important. The weights learned by neural networks are often difficult for humans to interpret. Learned neural networks are less easily communicated to humans than learned rules. The BACKPROPAGATION algorithm is the most commonly used ANN learning technique. It is appropriate for problems with the following characteristics:

- Instances are represented by many attribute-value pairs: The target function to be learned is defined over instances that can be described by a vector of predefined features.
- The target function output may be discrete-valued, real-valued, or a vector of several real- or discrete-valued attributes.
- The training examples may contain errors.
- ANN learning methods are quite robust to noise in the training data.
- Long training times are acceptable.
- Fast evaluation of the learned target function may be required.

2.4. PERCEPTRONS

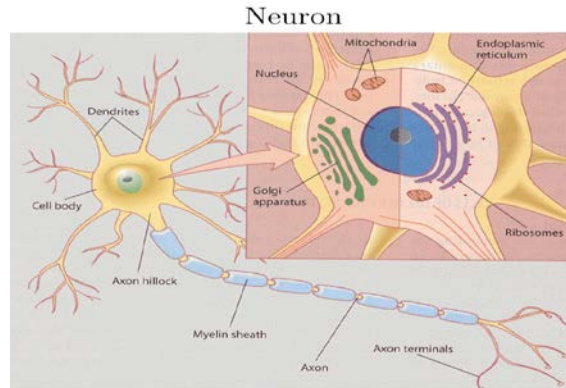
The perceptron was first proposed by Rosenblatt (1958) is a simple neuron that is used to classify its input into one of two categories. A perceptron is a single processing unit of a neural network. This is a good learning tool. This model follows perceptron training rule and it could operate well with linearly separable patterns.

Linear separability is the separation of the input space into regions is based on whether the network response is positive or negative.

A perceptron uses a step function that returns +1 if the weighted sum of its input (v) is greater than or equal to 0 else it returns -1.

$$\Phi(v) = \begin{cases} +1 & \text{if } v \geq 0 \\ -1 & \text{if } v < 0 \end{cases}$$

Working of perceptrons



Representation of a biological neuron

- In the biological neurons, the dendrite receives the electrical signals from the axons of other neurons. The signals are modulated in various amounts before further transmission.
- The signals are transmitted to other neurons only if the modulated signal exceeds the threshold value. The same principle is applied in perceptron model.
- In the perceptron, the input received is always represented as numerical values. These values are multiplied by the weights.
- The total strength of the input is calculated as the weighted sum of the inputs. A step function (activation function) is applied to determine its output.
- This output is fed to the other perceptrons if it exceeds the threshold value.

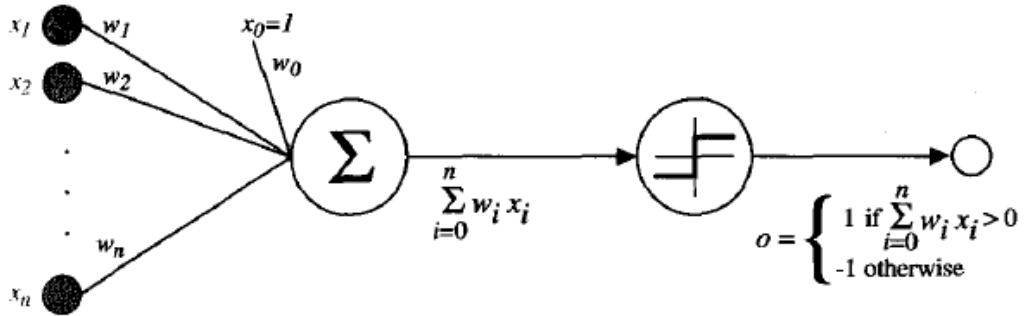
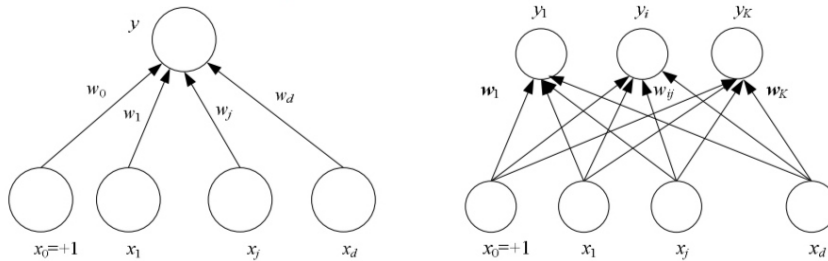


Fig. 2.4. Perceptron Model (In this model $x_0 = 1$, which is the bias)

- Linear perceptron: computes a linear function $y = \sum_{d=1}^D w_d x_d + w_0 \in \mathbb{R}$ of an input vector $x \in \mathbb{R}^D$ with weight vector $w \in \mathbb{R}^D$ (or $y = w^T x$ with $w \in \mathbb{R}^{D+1}$ and we augment x with a 0th component of value 1). To have K outputs: $y = w_x$ with $W \in \mathbb{R}^{K \times (D+1)}$



- For classification, we can use:
 - Two classes: $y = \sigma(w^T x) = \frac{1}{1 + \exp(-w^T x)} \in (0, 1)$ (logistic).
 - $K > 2$ classes: $y_k = \frac{\exp(w_k^T x)}{\sum_{j=1}^K \exp(w_j^T x)} \in (0, 1)$, $k = 1, \dots, K$ with $\sum_{k=1}^K y_k = 1$ (soft max)

Training a perceptron

- Apply stochastic gradient descent: to minimize the error $E(w) = \sum_{n=1}^K e(W; X_n, y_n)$, repeatedly update the weights $w \leftarrow w + \Delta$ with $\Delta w = -\eta \nabla e(W; X_n, y_n)$ and $n = 1, \dots, N$ (one epoch).

Regression by least-squares error:

$$E(W; X_n, y_n) = (y_n - w^T x_n)^2 \Rightarrow \Delta W = \eta(y_n - w^T x_n)x_n$$

Classification by maximum likelihood, or equivalently cross-entropy:

- ❖ For two classes: $y_n \in \{0, 1\}$ and $e(W; X_n, y_n) = -y_n \log \theta_n - (1 - y_n) \log (1 - \theta_n)$

$$\text{where, } \theta_n = \sigma(w^T x_n) \Rightarrow \Delta w = \eta(y_n - \theta_n)x_n$$

- ❖ For $K > 2$ classes: $y_n \in \{0, 1\}^K$ coded as 1-of-K and $e(W; X_n, y_n) = -\sum_{k=1}^K y_{kn} \log \theta_{kn}$

$$\text{where, } \theta_{kn} = \frac{\exp(w_k^T x)}{\sum_{j=1}^K \exp(w_j^T x)} \Rightarrow \Delta w_{kn} = \eta(y_{kn} - \theta_{kn}) x_{dn} \text{ for } d = 0 \dots D \text{ and}$$

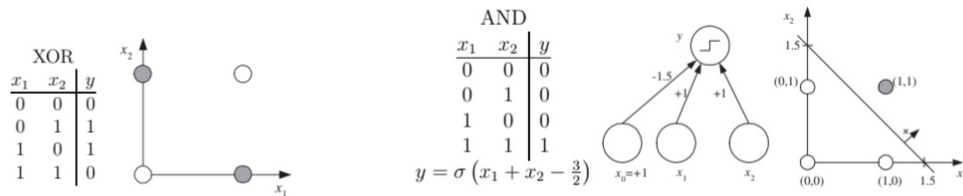
$$k = 0, \dots, K$$

The original perceptron algorithm was a variation of stochastic gradient descent. For linearly separable problems, it converges in a finite (possibly large) number of iterations. For problems that are not linearly separable problems, it never converges.

2.4.1. Perceptrons over Boolean functions

- The ability of perceptrons to represent AND, OR, NAND, and NOR is important because every boolean function can be represented by some network of interconnected units based on these primitives.
- Every boolean function can be represented by some network of perceptrons only two levels deep, in which the inputs are fed to multiple units, and the outputs of these units are then input to a second, final stage.
- One way is to represent the boolean function in disjunctive normal form. The input to an AND perceptron can be negated simply by changing the sign of the corresponding input weight.
- Because networks of threshold units can represent a rich variety of functions and because single units alone cannot, we will generally be interested in learning multilayer networks of threshold units.

- Boolean function: $\{0, 1\}^D \rightarrow \{0, 1\}$. Maps a vector of D bits to a single bit (truth value).
- Can be seen as a binary classification problem where the input instances are binary vectors.
- Since a perceptron can learn linearly separable problems, it can learn AND, OR but not XOR.



Construction of AND

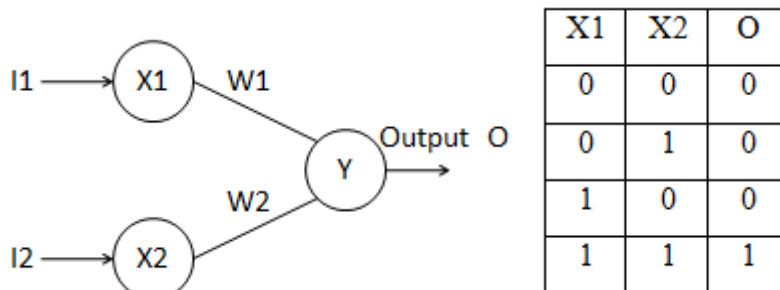


Fig. 2.5. AND function using perceptron

- The AND function has two inputs (I1 and I2) with one output (O) and two hidden units (X1 and X2).
- The following equalities could be inferred from the above table:
 - $W_1(0) + W_2(0) < \theta$
 - $W_1(0) + W_2(1) < \theta$
 - $W_1(1) + W_2(0) < \theta$
 - $W_1(1) + W_2(1) > \theta$
- Many solutions could be framed for the above equations. One possible solution is fixing the threshold value as 1.5.
- If $w_1=1$ and $w_2=1$, then the equations can be written as:

- a) $(0) + (1) (0) < 1.5$
- b) $(1) (0) + (1) (1) < 1.5$
- c) $(1) (1) + (1) (0) < 1.5$
- d) $(1) (1) + (1) (1) > 1.5$
- The output of the network is determined by calculating a weighted sum of its two input and comparing this value with a threshold value.
- If the net input (net) is greater than the threshold value, then the output is 1, else it is 0.
- To summarize, the output of the network is calculated as:

$$\text{Net} = W_1 \cdot I_1 + W_2 \cdot I_2$$
- If $\text{Net} > \text{threshold value}$, then output = 1, else output = 0.

2.4.2. Training rules in Perceptron: Perceptron Rule

- Training in perceptron occurs by adjusting the weights.
- The precise learning problem is to determine a weight vector that causes the perceptron to produce the correct ± 1 output for each of the given training examples.
- Several algorithms are known to solve this learning problem. The most common ones are: the perceptron rule and the delta rule, since they are guaranteed to converge.
- These rules form the basis for learning networks of many units.
- One way to learn an acceptable weight vector is to begin with **random weights**, then iteratively apply the perceptron to each training example, modifying the perceptron weights whenever it misclassifies an example. This process is repeated, iterating through the training examples as many times as needed until the perceptron classifies all training examples correctly.
- Weights are modified at each step according to the perceptron training rule, which revises the weight w_i associated with input x_i according to the rule.

$$W_i \leftarrow w_i + \Delta w_i$$

Where,

$$w_i \leftarrow w_i + \Delta w_i$$

- Here, t is the target output for the current training example, o is the output generated by the perceptron, and q is a positive constant called the learning rate.
- The role of the learning rate is to moderate the degrees to which weights are changed at each step.
- It is usually set to some small value (e.g., 0.1) and is sometimes made to decay as the number of weight-tuning iterations increases.
- The convergence is not assured for non-linearly separable problems.

2.4.3. Training rules in Perceptron: Gradient Descent and Delta Rule

- The perceptron rule fails to converge if the examples are not linearly separable. Delta rule is designed to overcome this difficulty.
- If the training examples are not linearly separable, the delta rule converges toward a best-fit approximation to the target concept.
- The key idea behind the delta rule is to use gradient descent to search the hypothesis space of possible weight vectors to find the weights that best fit the training examples.
- Gradient descent searches the hypothesis space of possible weight vectors to find the best one. The search hypothesis space contains many different types of continuously parameterized hypotheses.

“Gradient descent is an iterative algorithm, that starts from a random point on a function and travels down its slope in steps until it reaches the lowest point.”

- To find a local minimum of a function using gradient descent, one takes steps proportional to the negative of the gradient (or of the approximate gradient) of the function from the current point.

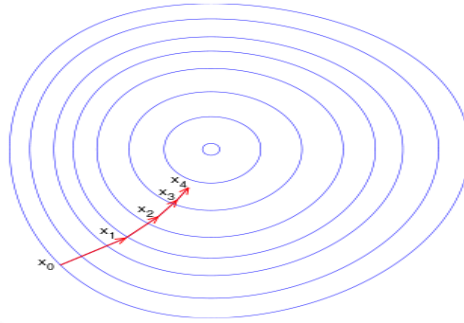


Fig. 2.6. Steps taken in gradient descent

- The delta training rule is best understood by considering the task of training an unthresholded perceptron which is a linear unit for which the output o is given by:

$$o(\vec{x}) = \vec{w} \cdot \vec{x}$$

- A linear unit corresponds to the first stage of a perceptron, without the threshold.
- In order to derive a weight learning rule for linear units, specify a measure for the training error of a hypothesis (weight vector), relative to the training examples.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2$$

Where D is the set of training examples, t_d is the target output (actual output) for training example d , and o_d (predicted output) is the output of the linear unit for training example d .

- $E(\vec{w})$ is half the squared difference between the target output and the output, summed over all training examples. This is the deviation between actual and target output. In other words it is the error in prediction.
- The best machine learning model will try to minimise this error value through continuous learning.

2.4.4. Visualizing the hypothesis space

- The hypothesis space of gradient descent is best represented using contour plot.
- Contour plots are topographical maps drawn from three-dimensional data.

- One variable is represented on the horizontal axis and a second variable is represented on the vertical axis. The third variable is represented by a Colour gradient and isolines (lines of constant value).
- These plots are helpful in searching for minimums and maximum values in a set of trivariate data.
- Let w_0, w_1 be the plane that represent the entire hypothesis space. Start with arbitrary initial weight vector, then repeatedly modify it in small steps in the direction that produces the steepest descent.

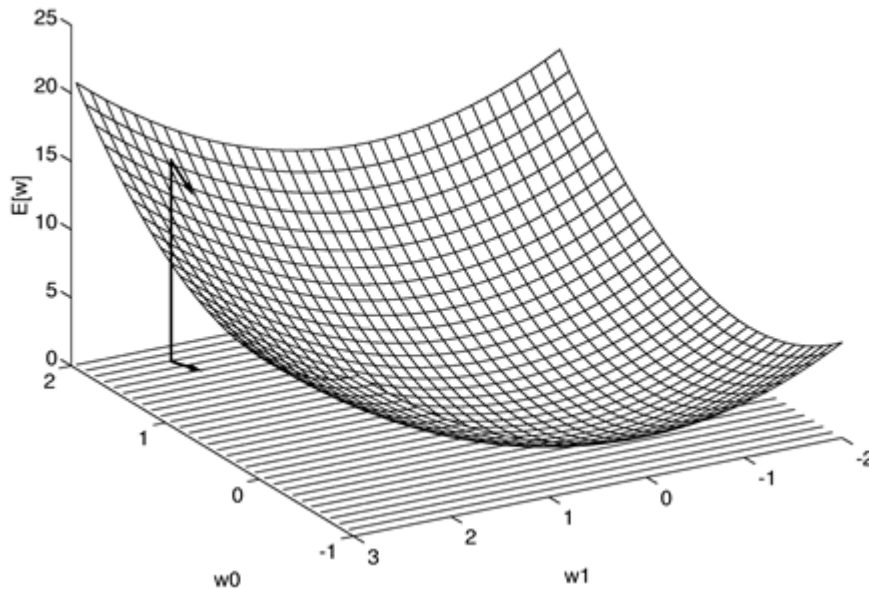


Fig. 2.7. Visualization of gradient descent using Contour plot

- The direction of steepest descent is along the error surface. Gradient descent algorithm will try to fix the values of w_1 and w_2 (refer Fig 2.7) at which the error $E[w]$ is minimum.
- Derivative of E with respect to each component of the vector \vec{w} is expressed as $\nabla E(\vec{w})$.

$$\nabla E(\vec{w}) = \left[\frac{\partial E}{\partial w_0}, \frac{\partial E}{\partial w_1}, \dots, \frac{\partial E}{\partial w_n} \right]$$

- The gradient specifies the direction that produces the steepest increase in E .

- $\nabla E(\vec{w})$ is a vector, whose components are the partial derivatives of E with respect to each of the w_i .
- When interpreted as a vector in weight space, the gradient specifies the direction that produces the steepest increase in E. The negative of this vector therefore gives the direction of steepest decrease.
- Since the gradient specifies the direction of steepest increase of E, the training Rule for gradient descent is

$$\vec{w} \leftarrow \vec{w} + \Delta \vec{w}$$

Where

$$\Delta \vec{w} = -\eta \nabla E(\vec{w})$$

- η is the learning rate, a positive constant, which determines the step size in the gradient descent search.
- The negative sign is present because the weight vector in the direction that decreases E.
- The gradient descent rule states

$$\vec{w}_i \leftarrow \vec{w}_i + \Delta \vec{w}_i$$

$$\Delta \vec{w}_i = -\eta \nabla E(\vec{w}_i)$$

- The vector of $\frac{\partial E}{\partial w_i}$ derivatives that form the gradient can be obtained by partially differentiating E with respect to w.
- Calculation of gradient is given by:

$$\begin{aligned} \frac{\partial E}{\partial w_i} &= \frac{\partial}{\partial w_i} \frac{1}{2} \sum_{d \in D} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} \frac{\partial}{\partial w_i} (t_d - o_d)^2 \\ &= \frac{1}{2} \sum_{d \in D} 2(t_d - o_d) \frac{\partial}{\partial w_i} (t_d - o_d) \\ &= \sum_{d \in D} (t_d - o_d) \frac{\partial}{\partial w_i} (t_d - \vec{w} \cdot \vec{x}_d) \end{aligned}$$

$$= \sum_{d \in D} (t_d - o_d) (-x_{id})$$

$$\Delta w_i = \eta \sum_{d \in D} (t_d - o_d) x_{id}$$

Algorithm for Gradient Descent**Gradient_Descent(training example, η)**

Each training example is a pair of the form (\vec{x}, t) , where \vec{x} is the vector of input values, and

T is the target output value. η is the learning rate.

Initialize each w_i to some small random value

Until the termination condition is met, Do

Initialize each Δw_i to zero.

For each (\vec{x}, t) in training examples, Do

Input the instance \vec{x} to the unit and compute the output o

For each linear unit weight w_i , Do

$$-\Delta w_i \leftarrow \Delta w_i + \eta(t - o) x_i$$

For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \Delta w_i$$

Limitations of gradient descent

- Converging to a local minimum can sometimes be quite slow
- No guarantee to find the global minimum
- Huge computations for large amount of data

2.4.5. Stochastic Gradient Descent (SGD)

- This is an improvisation done over gradient descent to reduce the computations.
- SGD or **incremental gradient descent** randomly picks one data point from the whole data set at each iteration to reduce the computations enormously.
- SGD approximates the gradient descent search by updating weights incrementally, following the calculation of the error for each individual example.

- The modified training rule of SGD is:

$$\Delta w_i \leftarrow \eta(t - o) x_i$$

- This provides a reasonable approximation to descending the gradient with respect to our original error function
- Also, by making the η sufficiently small, it can be made to approximate rule gradient descent arbitrarily closely.
- The distinct error function is:

$$E_d(\vec{w}) = \frac{1}{2} (t_d - o_d)^2$$

Algorithm for Stochastic Gradient Descent

Stochastic Gradient Descent (training example, η)

Each training example is a pair of the form (\vec{x}, t) , where \vec{x} is the vector of input values, and

t is the target output value. η is the learning rate.

Initialize each w_i to some small random value

Until the termination condition is met, Do

Initialize each Δw_i to zero.

For each (\vec{x}, t) in training examples, Do

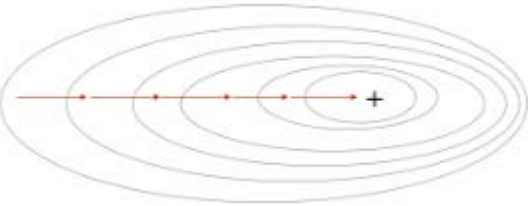
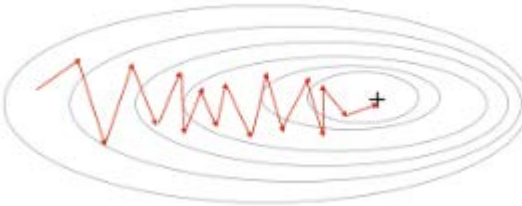
Input the instance \vec{x} to the unit and compute the output o

For each linear unit weight w_i , Do

$$w_i \leftarrow w_i + \eta(t - o) x_i$$

Differences between Gradient Descent and Stochastic Gradient Descent

Gradient Descent	Stochastic Gradient Descent
The error is summed over all the examples before updating the weights.	The weights are updated after each training example.
This requires more computations.	Relatively fewer computations are needed.
Summing over multiple examples requires larger step size per weight update.	This requires lower step size per weight update.

Relatively more prone to be stuck with local minima.	Do not get stuck with local minima.
	

Limitations of SGD

The SGD performs frequent updates with high variance that causes the function to fluctuate more.

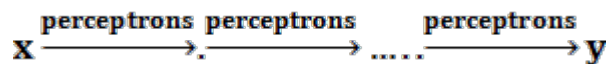
2.5. MULTILAYER NETWORKS AND THE BACKPROPAGATION ALGORITHM

Single perceptrons can only express linear decision surfaces. But a multilayer network learned by the Back propagation algorithm is capable of expressing a rich variety of nonlinear decisions.

2.5.1. Multi Layer Perceptron (MLP)

The MultiLayer Perceptron (MLPs) classifies datasets which are not linearly separable. This is made possible by their more robust and complex architecture.

- Multilayer perceptron (or feed forward neural net): nested sequence of perceptrons, with an input layer, an output layer and zero or more hidden layers:



- It can represent nonlinear discriminants (for classification) or functions (for regression).
- Architecture of the MLP: each layer has several units. Each unit h takes as input the output z of the previous layer's units and applies to it a linear function $w_h^T z$ (using a weight vector w_h , including a bias term) followed by a nonlinearity $s(t)$: output of unit $h = s(w_h^T z)$.

$$\frac{d\sigma(y)}{dy} = \sigma(y) \cdot (1 - \sigma(y))$$

- The function is **monotonic** but function's derivative is not.

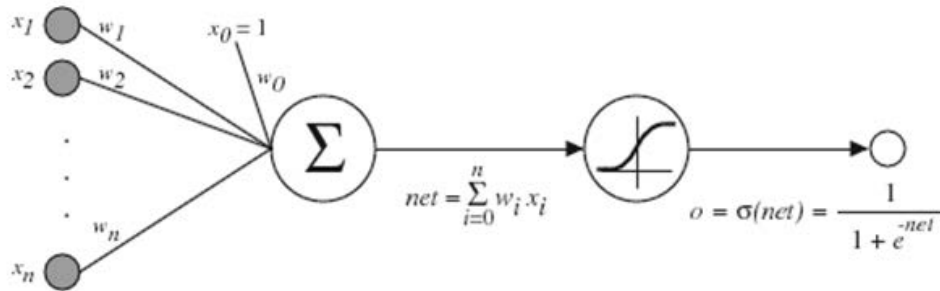


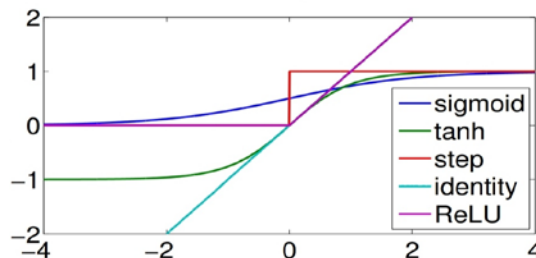
Fig. 2.8. Sigmoid Threshold Unit

- The sigmoid unit first computes a linear combination of its inputs, then applies a threshold to the result. The threshold output is a continuous function of its input.
- Sigmoid unit computes its output o by

$$O = \sigma(\vec{w} \cdot \vec{x})$$

$$\sigma(y) = \frac{1}{1 + e^{-y}}$$

- The other name for sigmoid function is **squashing function**, since it maps a very large input domain to a small range of outputs.
- To summarize, Sigmoid function:
 - ✓ Logistic function
 - ✓ Output ranges between 0 to 1
 - ✓ Increasing with its input (monotonic)
 - ✓ Its derivative is easily expressed in terms of its output (differentiable)
- Typical nonlinearity functions $s(t)$ used:
 - Logistic function: $s(t) = \frac{1}{1 + e^{-t}}$ (or softmax)



- Hyperbolic tangent: $s(t) = \tanh t = \frac{e^t - e^{-t}}{e^t + e^{-t}}$.
- Rectified linear unit (Re LU): $s(t) = \max(0, t)$.
- Step function: $s(t) = 0$ if $t < 0$, else 1 .
- Identity function: $s(t) = t$ (no nonlinearity).

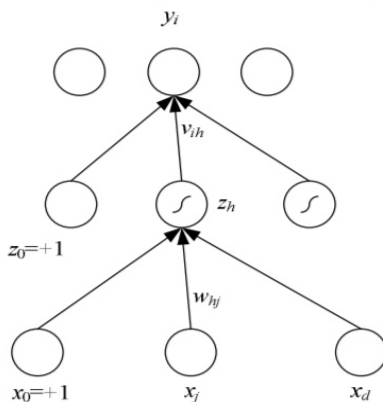
The output layer uses as nonlinearity:

- For regression: the identity (so the outputs can take any real value).
- For classification: the sigmoid and a single unit ($K = 2$ classes), or the soft max and K units ($K > 2$ classes).

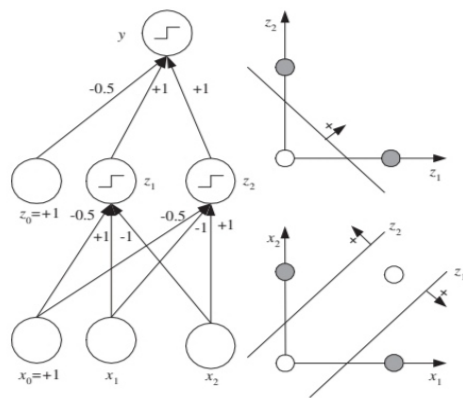
All nonlinearities (sigmoid, etc.) have a similar universal approximation ability, but some (e.g. Re LU) are easier to optimize than others.

- If all the layers are linear $s(t) = t$ for all units in each layer) the MLP is overall a linear function, which is not useful, so we must have some nonlinear layers.
- Ex: an MLP with one hidden layer having H units, with inputs $x \in \mathbb{R}^D$ and outputs $y \in \mathbb{R}^{D'}$, where the hidden units are sigmoidal and the output units are linear:

An MLP with a single nonlinear hidden layer...



...solves the XOR problem



MLP as a universal approximator

- A Boolean function can always be written as a disjunction of conjunctions, and this can be implemented by an MLP with one hidden layer. Each conjunction (AND) is implemented by one hidden unit. Ex: $x_1 \text{ XOR } x_2 = (x_1 \text{ AND } \bar{x}_2) \text{ OR}$

(\bar{x}_1 AND x_2) The disjunction (OR) is implemented by the output unit. This existence proof generates very large MLPs (up to 2D hidden units with D inputs). Practically, MLPs are much smaller.

- Universal approximation: any continuous function (satisfying mild assumptions) from R^D to $R^{D'}$ can be approximated by an MLP with a single hidden layer with an error as small as desired (by using sufficiently many hidden units).

2.5.2. Back propagation Algorithm

This algorithm is used to compute the gradients in the delta rule, to minimize the squared error between the actual and the predicted values. It learns the weights for a multilayer network, given a network with a fixed set of units and interconnections. The MLPs have multiple output units rather than single unit, hence they can have multiple local minima. The gradient descent is guaranteed only to converge toward some local minimum. So redefine $E(\text{error})$ to sum the errors over the entire network output units.

$$E(\vec{w}) = \frac{1}{2} \sum_{d \in D} \sum_{k \in \text{outputs}} (t_{kd} - o_{kd})^2$$

t_{kd} and o_{kd} target and output values respectively for the k^{th} output and training example d .

Stochastic gradient descent version of the Back propagation for Feed forward networks containing two layers of sigmoid units.

```
//Back propagation(training _example,  $\eta$ ,  $\eta_{\text{in}}$ ,  $\eta_{\text{out}}$ ,  $\eta_{\text{hidden}}$ )
```

//The input from unit i into unit j is denoted x_{ij} , and the weight from unit i to unit j is denoted

w_{ij} , Each training example is a pair of the form (\vec{x}, \vec{t}) which is the vector of network input

\vec{t} values, and is the vector of target network output values.

Create a feed-forward network with input (η_{in}), output (η_{out}), hidden (η_{hidden}) units

Initialize all network weights to small random number (between -0.05 and 0.05).

Until the termination condition is met, Do

For each (\vec{x}, \vec{t}) in training_examples, Do

Propagate the input forward through the network:

1. input the instance \vec{x} to the network and compute the output o_u of every unit

Propagate the errors backward through the network:

2. For each network output unit k , calculate its error term

$$\delta(k) \leftarrow o_k (1 - o_k) (t_k - o_k)$$

3. For each hidden unit h , calculate its error term

$$\delta_h = o_h (1 - o_h) \sum_{k \in \text{outputs}} \delta_k W_{kh}$$

4. Update each network weight

$$W_{ji} \leftarrow W_{ji} + \Delta W_{ji},$$

Where

$$\Delta W_{ji} = \eta \delta_j x_j$$

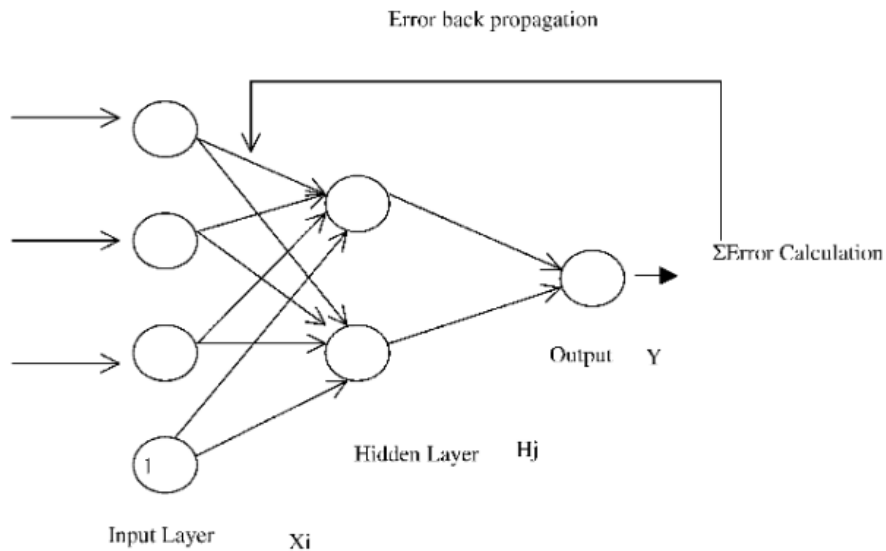


Fig. 2.9. Back propagation

- η_n denotes the error term associated with unit n which is the difference between actual and predicted output.
- The neural network described in back propagation algorithm starts with the desired number of hidden and output units and initializing all network weights to small random values.
- With this fixed network structure, the main loop of the algorithm then repeatedly iterates over the training examples.

- For each training example, it applies the network to the example, calculates the error of the network output for this example, computes the gradient with respect to the error on this example, then updates all weights in the network.
- This gradient descent step is iterated until the network performs acceptably well.
- The error term δ_k is computed for each unit k , by $(t_k - o_k)$ which is the difference between target and output values.
- This difference is then multiplied by $o_k (1 - o_k)$, which is the derivative of sigmoid squashing function.
- Error estimation for hidden unit:
 - ✓ No target values are directly available to indicate the error of hidden units' values
 - ✓ The error terms for hidden unit h is calculated by summing the errors δ_k for each output unit influenced by h , weighting each of the δ_k by w_{kh} which is the weight from hidden unit h to output unit k .
- Updating weights incrementally, following the presentation of each training example. This corresponds to a stochastic approximation to gradient descent
- To obtain the true gradient of E , one would sum the $\delta_j x_{ji}$ values over all training examples before altering weight values.
- This is iterated thousands of times in a typical application.
- The following are the termination condition can be used to halt the procedure
 - ✓ Choose to halt after certain iteration
 - ✓ Error on training examples falls below some threshold
 - ✓ Error on a separate validation set of examples meets some criterion

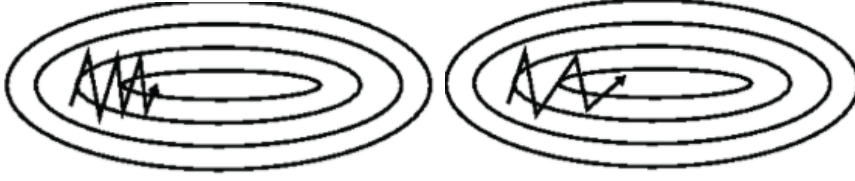
2.5.3. Adding Momentum

- The Back propagation algorithm only requires that the weight changes be proportional to the derivative of the error.
- The larger the learning rate the larger the weight changes on each epoch, and the quicker the network learns.

- However, the size of the learning rate can also influence whether the network achieves a stable solution.
- If the learning rate gets too large, then the weight changes no longer approximate a gradient descent procedure. Oscillation of the weights is often the result.
- So it's natural to use the largest learning rate possible without triggering oscillation. This would offer the most rapid learning and the least amount of time spent waiting at the computer for the network to train.
- One method that has been proposed is a slight modification of the backpropagation algorithm so that it includes a momentum term.
- The concept of momentum is that previous changes in the weights should influence the current direction of movement in weight space. This concept is implemented by the revised weight-update rule:

$$\Delta W_{ji}(n) = \eta \delta_j x_{ji} + \alpha \Delta W_{ji}(n-1)$$

- With momentum, once the weights start moving in a particular direction in weight space, they tend to continue moving in that direction.
- $\alpha \Delta W_{ji}(n-1)$ is the momentum term that the gradient descent search trajectory is analogous to that of a ball rolling down the error surface.
- The effect of α is to add momentum that tends to keep the ball rolling in the same direction from one iteration to the next.
- This can sometimes have the effect of keeping the ball rolling through small local minima in the error surface, or along flat regions in the surface where the ball would stop if there were no momentum.
- It also has the effect of gradually increasing the step size of the search in regions where the gradient is unchanging, thereby speeding convergence.



Learning rate in SGD without Momentum Learning rate in SGD with momentum

Fig. 2.10. Role of momentum in boosting the learning rate

2.5.4. Derivation of Back propagation rule

The stochastic gradient descent involves iterating through the training examples one at a time, for each training example d descending the gradient of the error E_d with respect to this single example.

$$\Delta W_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}}$$

$$E_d(\mathbf{w}) = \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

The weighted sum of inputs for unit j is given by

$$\text{Net}_j = \sum_i W_{ji} x_{ji}$$

σ is the sigmoid function.

$$\frac{\partial E_d}{\partial w_{ji}} = \frac{\partial E_d}{\partial \text{net}_j} \frac{\partial \text{net}_j}{\partial w_{ji}} = \frac{\partial E_d}{\partial \text{net}_j} x_{ji}$$

Chaining Rule for Output Unit Weights

Just as w_{ji} can influence the rest of the network only through net_j , net_i can influence the network only through o_j .

$$\frac{\partial E_d}{\partial \text{net}_j} = \frac{\partial E_d}{\partial o_j} \frac{\partial o_j}{\partial \text{net}_j}$$

To begin, consider just the first term in Equation

$$\frac{\partial E_d}{\partial o_j} = \frac{\partial}{\partial o_j} \frac{1}{2} \sum_{k \in \text{outputs}} (t_k - o_k)^2$$

$$\begin{aligned}
&= \frac{\partial}{\partial o_j} \frac{1}{2} (t_j - o_j) \\
&= \frac{1}{2} 2(t_j - o_j) \frac{\partial(t_j - o_j)}{\partial o_j} \\
&= (t_j - o_j)
\end{aligned}$$

Consider the second term,

$$\frac{\partial o_j}{\partial net_j} = \frac{\partial \sigma(net_j)}{\partial net_j} = o_j (1 - o_j)$$

After substitution of the results of first two terms

$$\frac{\partial E_d}{\partial net_j} = -(t_j - o_j) o_j (1 - o_j)$$

After application of SGD,

$$\Delta w_{ji} = -\eta \frac{\partial E_d}{\partial w_{ji}} = \eta (t_j - o_j) o_j (1 - o_j) x_{ji} = \eta \delta_j x_{ji}$$

Estimation of weights in hidden unit

$$\begin{aligned}
&\frac{\partial E_d}{\partial net_j} \sum_{K \in D \text{ ownstream}(j)} \frac{\partial E_d}{\partial net_k} \frac{\partial net_k}{\partial net_j} \\
&= \sum_{K \in D \text{ ownstream}(j)} -\delta_k \frac{\partial net_k}{\partial net_j} \\
&= \sum_{K \in D \text{ ownstream}(j)} -\delta_k \frac{\partial net_k}{\partial o_j} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{K \in D \text{ ownstream}(j)} -\delta_k w_{kj} \frac{\partial o_j}{\partial net_j} \\
&= \sum_{K \in D \text{ ownstream}(j)} -\delta_k w_{kj} o_j (1 - o_j) \\
&= -o_j (1 - o_j) \sum_{K \in D \text{ ownstream}(j)} \delta_k w_{kj} \\
\Delta w_{ji} &= -\eta x_{ji} o_j (1 - o_j) \sum_{K \in D \text{ ownstream}(j)} \delta_k w_{kj}
\end{aligned}$$

2.5.5. Additional Information

Convergence and local minima

- ✓ Back propagation with SGD can guarantee to converge toward some local minimum E and not necessarily to the global minimum error.
- ✓ Back Propagation is a highly effective function approximation method in practice

Underfitting and overfitting

- ✓ The cause of poor performance in machine learning is either over fitting or under fitting the data.
- ✓ Supervised machine learning is best understood as approximating a target function (f) that maps input variables (X) to an output variable (Y). $Y = f(X)$.
- ✓ This characterization describes the range of classification and prediction problems and the machine algorithms that can be used to address them.
- ✓ An important consideration in learning the target function from the training data is how well the model generalizes to new data. Generalization is important because the data we collect is only a sample, it is incomplete and noisy.
- ✓ **Generalization** refers to how well the concepts learned by a machine learning model apply to specific examples not seen by the model when it was learning.
- ✓ The goal of a good machine learning model is to generalize well from the training data to any data from the problem domain. This allows us to make predictions in the future on data the model has never seen.
- ✓ Overfitting and underfitting are the two biggest causes for poor performance of machine learning algorithms.

Overfitting in Machine Learning

- ✓ Overfitting refers to a model that models the training data too well.
- ✓ Overfitting happens when a model learns the detail and noise in the training data to the extent that it negatively impacts the performance of the model on new data.

- ✓ This means that the noise or random fluctuations in the training data is picked up and learned as concepts by the model.
- ✓ The problem is that these concepts do not apply to new data and negatively impact the models ability to generalize.
- ✓ Overfitting is more likely with nonparametric and nonlinear models that have more flexibility when learning a target function.
- ✓ As such, many nonparametric machine learning algorithms also include parameters or techniques to limit and constrain how much detail the model learns.
- ✓ A statistical model is said to be overfitted, when we train it with a lot of data.
- ✓ When a model gets trained with so much of data, it starts learning from the noise and inaccurate data entries in our data set.
- ✓ Then the model does not categorize the data correctly, because of too much of details and noise.
- ✓ The causes of overfitting are the non-parametric and non-linear methods because these types of machine learning algorithms have more freedom in building the model based on the dataset and therefore they can really build unrealistic models.
- ✓ Some weights begin to grow in order to reduce the error over the training data, and the complexity of the learned decision surface increase.
- ✓ Given enough iterations, Backpropagation will often be able to create overly complex decision surfaces that fit noise in the training data or unrepresentative characteristics of the particular training sampletrees.

Avoiding Overfitting

Cross- Validation: A standard way to find out-of-sample prediction error is to use 5-fold cross validation.

Early Stopping: Its rules provide us the guidance as to how many iterations can be run before learner begins to over-fit.

Pruning: Pruning is extensively used while building related models. It simply removes the nodes which add little predictive power for the problem in hand.

Regularization: It introduces a cost term for bringing in more features with the objective function. Hence it tries to push the coefficients for many variables to zero and hence reduce cost term.

Underfitting in Machine Learning

- ✓ Underfitting refers to a model that can neither model the training data nor generalize to new data.
- ✓ An underfit machine learning model is not a suitable model and will be obvious as it will have poor performance on the training data.
- ✓ Underfitting is often not discussed as it is easy to detect given a good performance metric. The remedy is to move on and try alternate machine learning algorithms. Nevertheless, it does provide a good contrast to the problem of overfitting.
- ✓ A statistical model or a machine learning algorithm is said to have underfitting when it cannot capture the underlying trend of the data.
- ✓ Underfitting destroys the accuracy of our machine learning model. Its occurrence simply means that our model or the algorithm does not fit the data well enough. It usually happens when we have less data to build an accurate model and also when we try to build a linear model with a non-linear data.
- ✓ In such cases the rules of the machine learning model are too easy and flexible to be applied on such a minimal data and therefore the model will probably make a lot of wrong predictions.
- ✓ Underfitting can be avoided by using more data and also reducing the features by feature selection.

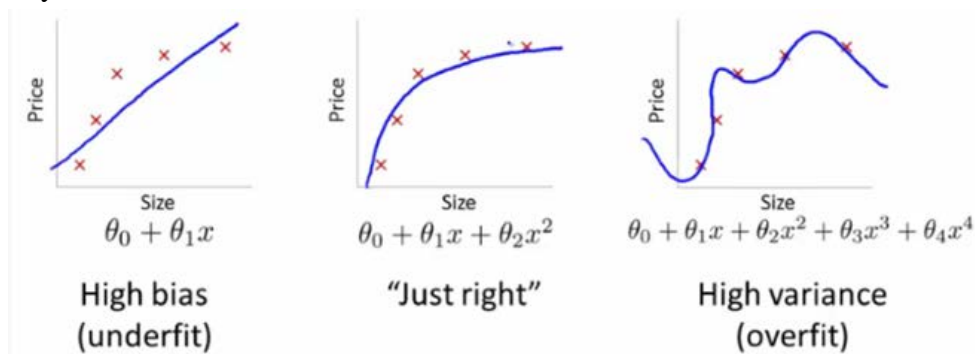


Fig. 2.11. Underfitting, Just Fit and Overfitting

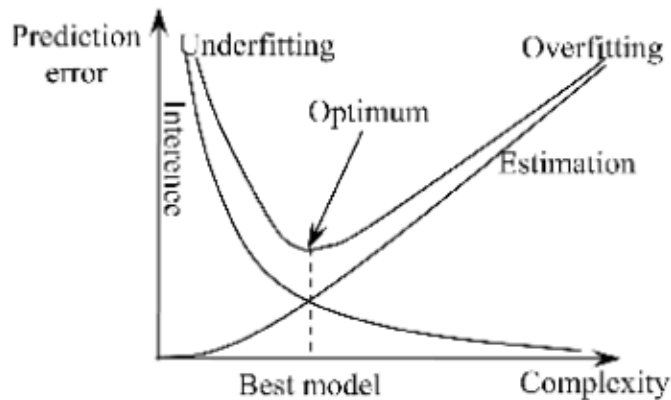


Fig. 2.12. Tradeoff between underfitting and overfitting

2.6. INTRODUCTION TO GENETIC ALGORITHMS

Genetic Algorithm (GA) is a search-based optimization technique based on the principles of Genetics and Natural Selection. It is commonly used to find optimal or near-optimal solutions to difficult problems which otherwise would take a lifetime to solve (optimization problems). They use adaptive heuristic search based on the evolutionary ideas of natural selection and genetics. They are inspired by the biological evolution.

A Genetic Algorithm (GA) is a problem solving method that uses genetics as model of problem solving. This technique finds approximate solutions to optimization and search problems.

Genetic algorithms are a particular class of evolutionary algorithms that use techniques inspired by evolutionary biology such as inheritance, mutation, selection, and crossover. The genetic algorithms are seen as exploitation of random search to solve optimization problems using some historic information. The nature's "Survival of the fittest" is the key deciding factor in the development of genetic algorithms.

A genetic algorithm maintains a population of potential candidate solutions for the given problem, and makes it evolve by iteratively applying a set of stochastic operators.

Stochastic Operators

The following are the common stochastic operators:

- ✓ **Selection:** This replicates the successful solutions found in a population based on their quality.
- ✓ **Recombination:** This decomposes two distinct solutions and randomly mixes their parts to form novel solutions.
- ✓ **Mutation:** This randomly forms a candidate solution.

Basic Terminologies in GA:

- Individual - Any possible solution got in the progress of the algorithm.
- Population - Group of all individuals participating in the evolution.
- Search Space - All possible solutions to the problem.
- Chromosome - Blueprint for an individual.
- Trait – Possible features of an individual
- Allele - Possible settings or characteristics of trait.
- Locus - The position of a gene on the chromosome.
- Genome - Collection of all chromosomes for an individual.

Generalized Genetic Algorithm

GA(Fitness, Fitness_threshold, p, r, m)

Fitness: A function that assigns an evaluation score, given a hypothesis.

Fitnessthreshold: A threshold specifying the termination criterion.

P: The number of hypotheses to be included in the population.

r: The fraction of the population to be replaced by Crossover at each step.

m: The mutation rate.

Initialize population: $P \leftarrow$ Generate p hypotheses at random

Evaluate: For each h in P, compute Fitness(h)

While [max Fitness(h)] < Fitness threshold do

Create a new generation, Ps:

1. **Select:** Probabilistically select $(1 - r)p$ members of P to add to Ps. The probability $\text{Pr}(h_i)$ of selecting hypothesis h_i from P is given by

$$p_r(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^p \text{Fitness}(h_j)}$$

2. Crossover: Probabilistically select $(r.p)/2$, pairs of hypotheses from P , according to $P_r(h_i)$ given above. For each pair (h_1, h_2) , produce two offspring by applying the Crossover operator. Add all offspring to P_s .

3. Mutate: Choose m percent of the members of P_s with uniform probability. For each, invert one randomly selected bit in its representation.

4. Update: $P \leftarrow P_s$.

5. Evaluate: for each h in P , compute $\text{Fitness}(h)$

Return the hypothesis from P that has the highest fitness.

Steps in Genetic Algorithm

- A population containing p hypotheses is maintained.
- On each iteration, the successor population P_s is formed by probabilistically selecting current hypotheses according to their fitness and by adding new hypotheses.
- New hypotheses are created by applying a crossover operator to pairs of most fit hypotheses and by creating single point mutations in the resulting generation of hypotheses.
- This process is iterated until sufficiently fit hypotheses are discovered.
- The inputs to this algorithm include the fitness function for ranking candidate hypotheses, a threshold defining an acceptable level of fitness for terminating the algorithm, the size of the population to be maintained, and parameters that determine how successor populations are to be generated: the fraction of the population to be replaced at each generation and the mutation rate.
- Each iteration through the main loop produces a new generation of hypotheses based on the current population.
- A certain number of hypotheses from the current population are selected for inclusion in the next generation.
- The probability at which hypothesis selection is made is given by

$$p_r(h_i) = \frac{\text{Fitness}(h_i)}{\sum_{j=1}^P \text{Fitness}(h_j)}$$

- The probability that a hypothesis will be selected is proportional to its own fitness and is inversely proportional to the fitness of the other competing hypotheses in the current population.
- Once these members of the current generation have been selected for inclusion in the next generation population, additional members are generated using a crossover operation.
- **Cross over**, takes two parent hypotheses from the current generation and creates two offspring hypotheses by recombining portions of both parents.
- The parent hypotheses are chosen probabilistically from the current population, again using the probability function.
- After new members have been created by this crossover operation, the new generation population now contains the desired number of members.
- At this point, a certain fraction m of these members are chosen at random, and random mutations are performed to alter these members.
- This GA algorithm thus performs a randomized, parallel beam search for hypotheses that perform well according to the fitness function.

Advantages of GA

- They do not require any derivative information.
- It is faster and more efficient as compared to the traditional methods.
- It has good parallel processing capabilities.
- It is capable of optimizing both continuous and discrete functions and also multi-objective problems.
- This provides a list of potential solutions from which optimized one could be isolated.
- GA always converges to a solution, which gets better over the time.
- This will be of immense use when the search space is very large and there are a large number of parameters involved.

Limitations of GA:

- GAs are not suited for simple problems.
- The calculation of fitness value is computationally expensive.
- There is no guarantee for the algorithm to find the optimal solution.
- Improper implementation may not converge to the optimal solution.

2.6.1. Representation of hypothesis

The space of all feasible solutions called search

- The problem solving process in GA is looking for the best solution in a specific subset of solutions or search space or state space.
- Every point in the search space represents a possible solution.
- So every point has a fitness value, depending on the problem definition.
- In this search space, there lies a point or a set of points which gives the optimal solution.
- The aim of optimization is to find that point or set of points in the search space.
- The difficulties faced in finding best solution are the local minima problem and the starting point of the search. One does not know where to look for the solution and where to start.

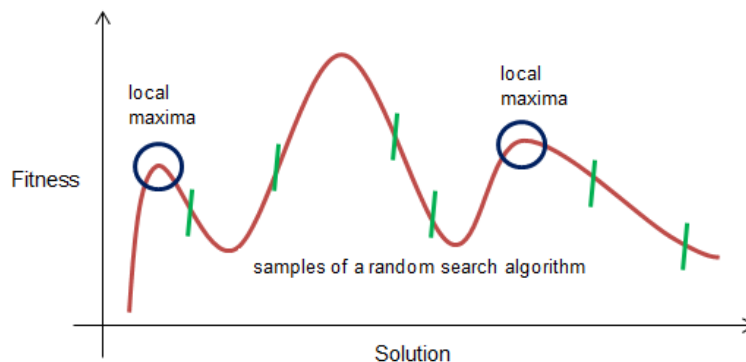


Fig. 2.13. Local maxima in search space

- For the genetic algorithm, the population encompasses a range of possible outcomes.
- Solutions are identified purely on a fitness level, and therefore local optima or local maxima are not distinguished from other equally fit individuals.
- Those solutions closer to the global optimum will thus have higher fitness values.
- Successive generations improve the fitness of individuals in the population until the optimisation convergence criterion is met. Due to this probabilistic nature GA tends to the global optimum.
- Hypotheses in GAs are often represented by bit strings, so that they can be easily manipulated by genetic operators such as mutation and crossover.
- The hypotheses represented by these bit strings can be quite complex. Sets of if-then rules can easily be represented in this way, by choosing an encoding of rules that allocates specific substrings for each rule precondition and postcondition.
- Placing a 1 in some position indicates that the attribute is allowed to take on the corresponding value.
- Given this method for representing constraints on a single attribute, conjunctions of constraints on multiple attributes can easily be represented by concatenating the corresponding bit strings.

Example: Consider the attribute Outlook which can take one among the three values <sunny, overcast, rainy>.

- ✓ Encoding Outlook = 001 means that the attribute outlook takes the third value in the list or Outlook = rainy.
- ✓ Encoding Outlook = 011 means that the attribute outlook takes the third value in the list or Outlook = overcast V rainy.

Outlook	Wind
011	10

Rule with precondition: (**Outlook = Overcast V Rain**) A (**Wind = Strong**) can be encoded as

Rule with postcondition: IF Wind = Strong THEN **PlayTennis = yes**

Outlook	Wind	PlayTennis
111	10	10

- The bit string representing the rule contains a substring for each attribute in the hypothesis space, even if that attribute is not constrained by the rule preconditions.
- This yields a fixed length bit-string representation for rules, in which substrings at specific locations describe constraints on specific attributes.
- Given this representation for single rules, we can represent sets of rules by similarly concatenating the bit string representations of the individual rules.
- In designing a bit string encoding for some hypothesis space, it is useful to arrange for every syntactically legal bit string to represent a well-defined hypothesis.

2.6.2. Genetic Operations

After an initial population is randomly generated, the algorithm evolves the through three operators:

1. **Selection:** Survival of the fittest
2. **Crossover:** Mating between individuals
3. **Mutation:** Random modifications.

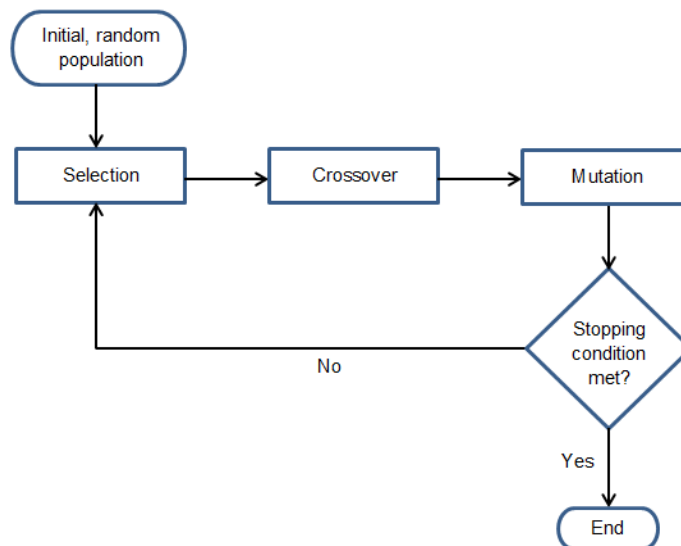


Fig. 2.14. Operators in GA

Selection Operator (Parents selection)

- This gives preference to better individuals (survival of the fittest), allowing them to pass on their genes to the next generation.
- The quality of each individual depends on its fitness function.
- Fitness function may be an objective function or a subjective judgement.

Selection is the process of selecting parents (candidates) from the search space which mate and recombine to create off-springs for the next generation.

- Some of the strategies followed in implementing the selection operation are:
 - ✓ Fitness Proportionate Selection
 - ✓ Tournament Selection
 - ✓ Rank Selection
 - ✓ Random Selection

Fitness Proportionate Selection

- ✓ This strategy offers chance for every individual in the space to become a parent with a probability which is based on the fitness.
- ✓ The fitter individual stands a higher chance of getting selected to mate and propagate their features to the next generation.
- ✓ This strategy evolved better individuals over time.
- ✓ This does not go well with negative fitness value.
- ✓ There are two implementations of this strategy:

a) Roulette Wheel Selection

- Here a circular wheel is divided into n pies, where n is the number of individuals in the population.
- Each individual gets a portion of the circle which is proportional to its fitness value. It is clear that a fitter individual has a greater pie on the wheel and therefore a greater chance of landing in front of the fixed point when the wheel is rotated.

- A fixed point is chosen on the wheel circumference and the wheel is rotated.
- The region of the wheel which comes in front of the fixed point is chosen as the parent.
- The process is repeated for selecting the next parent.

Steps in Roulette Wheel Selection

1. Calculate S = the sum of a fitnesses.
2. Generate a random number between 0 and S .
3. Starting from the top of the population, keep adding the fitnesses to the partial sum P , till $P < S$.
4. The individual for which P exceeds S is the chosen individual.

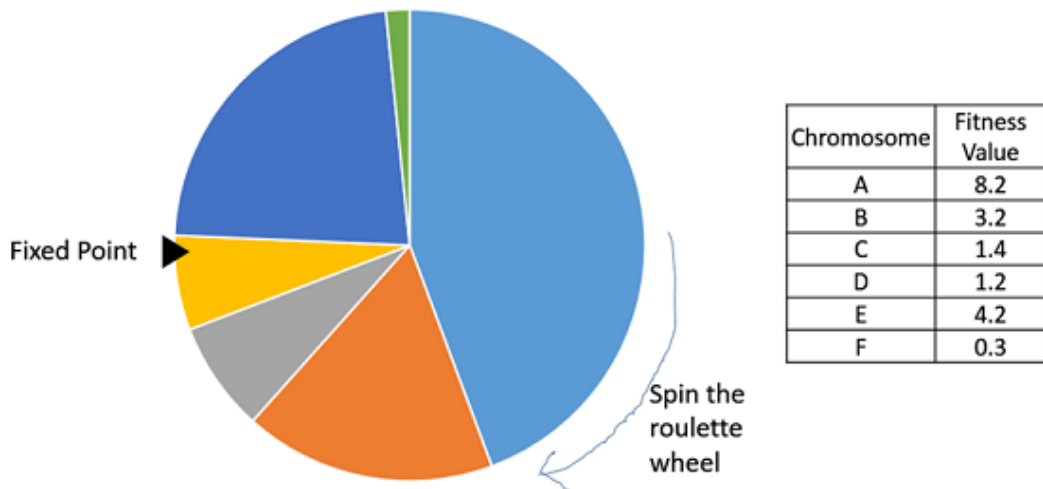


Fig. 2.15. Roulette Wheel Selection

b) Stochastic Universal Sampling (SUS)

- This is similar to Roulette wheel selection, but instead of having just one fixed point, multiple fixed points are used.
- All the parents are chosen in just one spin of the wheel.
- This setup encourages the highly fit individuals to be chosen at least once.

Tournament Selection

- ✓ In K-Way tournament selection, K individuals are selected from the population randomly and the best is selected as the parent.
- ✓ The same process is repeated for selecting the next parent.
- ✓ It could be applied to problems with negative fitness values.

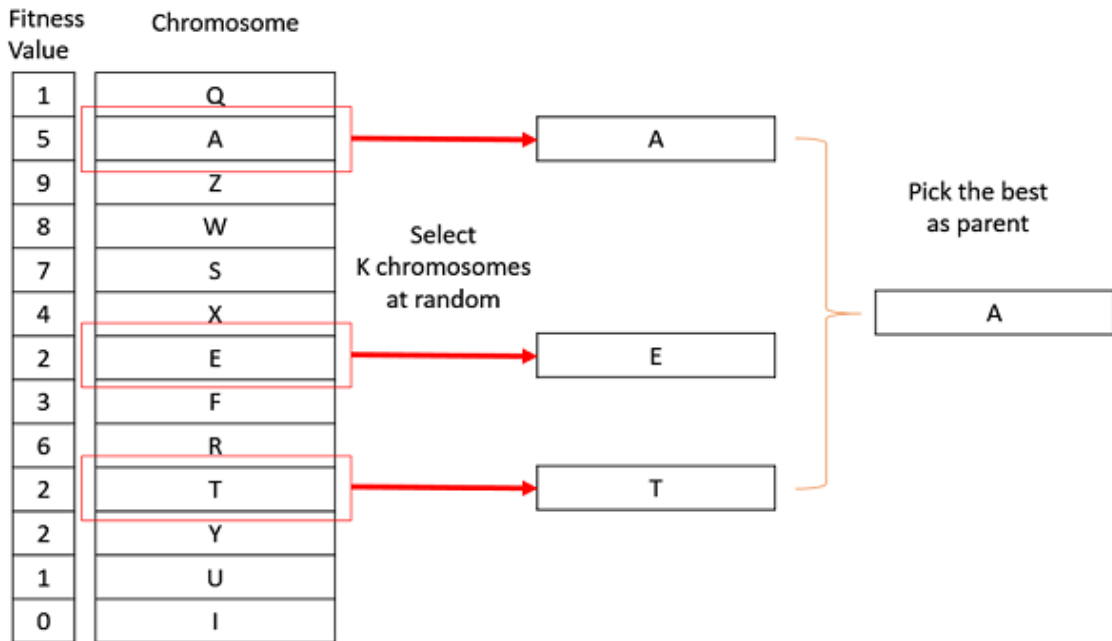


Fig. 2.16. Tournament Selection

Rank Selection

- ✓ This method is commonly used when the individuals in the population have very close fitness values.
- ✓ This works well with negative fitness value.
- ✓ Here, all the individual possess equal share of the pie.
- ✓ Each individual have same probability of getting selected as a parent.
- ✓ This leads to poor parent selections in such situations.
- ✓ Every individual in the population is ranked according to their fitness.
- ✓ The selection of the parents depends on the rank of each individual and not the fitness.

- ✓ The higher ranked individuals are preferred more than the lower ranked ones.
- ✓ The chromosomes are arranged in the table according to their rank.

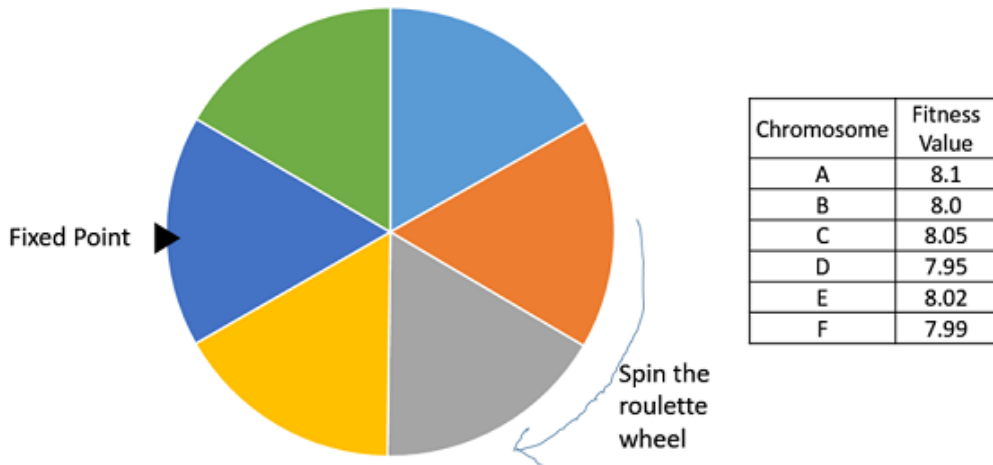


Fig. 2.17. Rank Selection

Random Selection

- ✓ Here the parents are randomly selected from the existing population.
- ✓ There is no selection pressure towards fitter individuals and therefore this strategy is usually avoided.

Crossover Operator

- This is the prime distinguished factor of GA from other optimization techniques.
- Any two individuals are chosen from the population using the selection operator.
- A crossover site along the bit strings is randomly chosen.
- The values of the two strings are exchanged up to the cross over point.
- **Example:** If $S1=000000$ and $s2=111111$ and the crossover point is 2 then $S1'=110000$ and $s2'=001111$.
- The two new offspring created from this mating are put into the next generation of the population.
- By recombining portions of good individuals, this process is likely to create even better individuals.

- Some of the types of crossovers are discussed here.
- Single Point cross over:** In this one-point crossover, a random crossover point is selected and the tails of its two parents are swapped to get new off-springs.

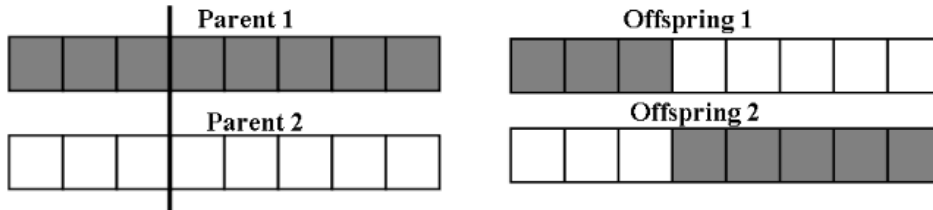


Fig. 2.18. Single point crossover

- Multipoint cross over:** Multi point crossover is a generalization of the one-point **crossover** wherein alternating segments are swapped to get new off-springs.

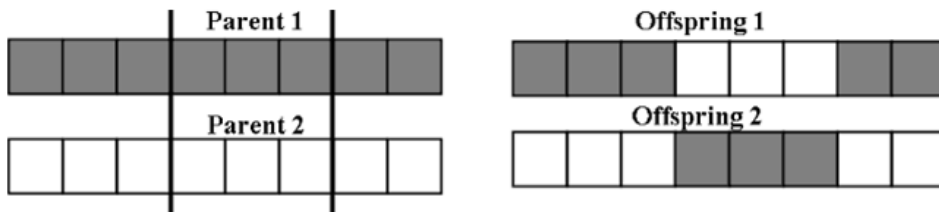


Fig. 2.19. Multi point cross over

- Uniform cross over:** Each gene is treated independently. There is no partitioning the chromo some into segments. Any one of the gene can be selected for cross over based on random selection.

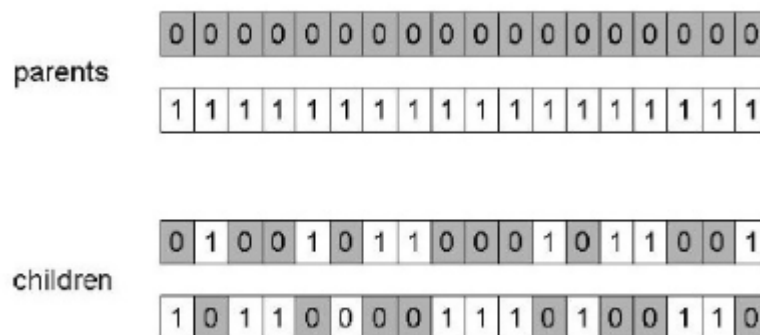


Fig. 2.20. Uniform Cross over

- **Whole arithmetic recombination**

- ✓ This is commonly used for integer representations.
- ✓ This takes the weighted average of the two parents,

$$\text{Child 1} = \alpha \cdot x + (1 - \alpha) \cdot y$$

$$\text{Child 2} = \alpha \cdot y + (1 - \alpha) \cdot x$$

- ✓ If $\alpha = 0.5$, then both the children will be identical.

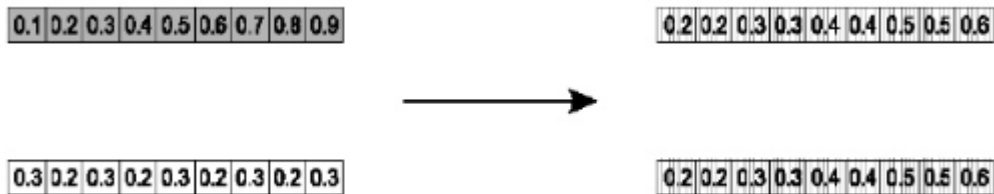


Fig. 2.21. Whole arithmetic cross over

- **Davis' Order Crossover (0 X 1)**

- ✓ This is used for permutation based crossovers with the intention of transmitting information about relative ordering to the off-springs.
- ✓ Create two random crossover points in the parent and copy the segment between them from the first parent to the first offspring.
- ✓ Starting from the second crossover point in the second parent, copy the remaining unused numbers from the second parent to the first child, wrapping around the list.
- ✓ Repeat for the second child with the parent's role reversed.

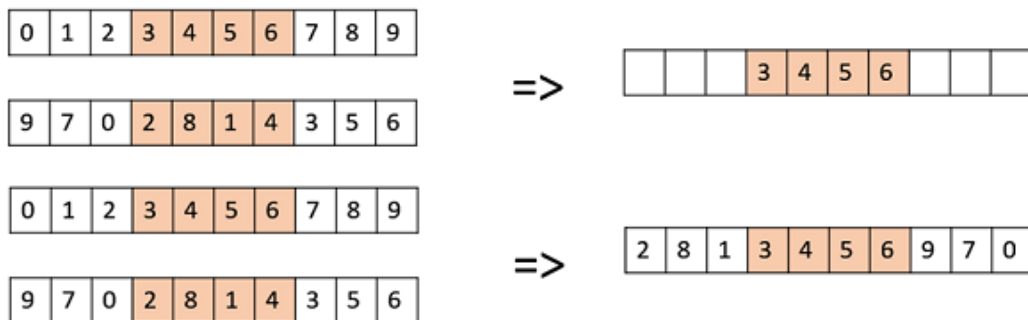


Fig. 2.22. Davis Order crossover

Mutation Operator

- A portion of the new individuals will have some of their bits flipped.
- The main objective is to maintain diversity within the population and inhibit premature convergence.
- Mutation induces a random walk through the search space.
- It is used to maintain and introduce diversity in the genetic population and is usually applied with a low probability.

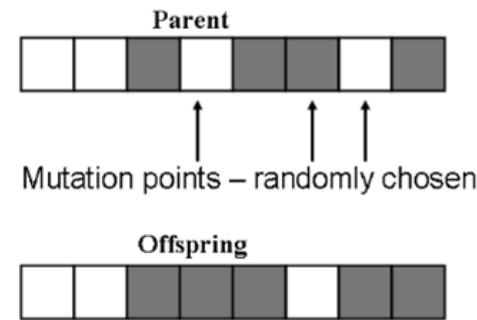


Fig. 2.23. Mutation operator

- Some of the common mutation operator types are discussed below.
- **Bit flip mutation:** Here one or more random bits are selected and flipped (changed). This is used for binary encoded GAs.

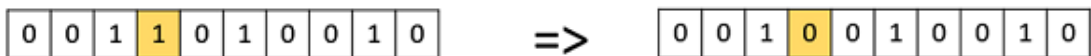


Fig. 2.24. Bit flip mutation

- **Random Resetting:** This can be seen as an extension of the bit flip for the integer representation. Here a random value from the set of permissible values is assigned to a randomly chosen gene.
- **Swap Mutation:** Here two positions on the chromosome are selected at random, and the values are interchanged. This is common in permutation based encodings.

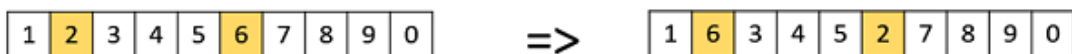


Fig. 2.25. Swap Mutation

- **Scramble Mutation:** A subset of genes is chosen and their values are scrambled or shuffled randomly.

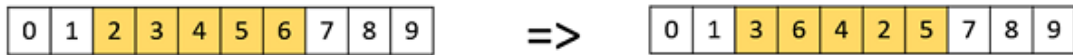


Fig. 2.26. Scramble Mutation

- **Inversion Mutation:** A subset of genes is chosen as in scramble mutation, but instead of shuffling the subset, invert the entire string in the subset.

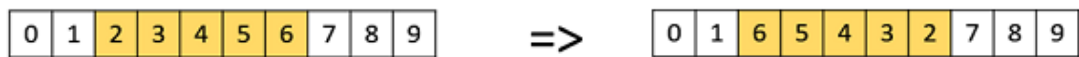


Fig. 2.27. Inversion Mutation

Effects of Genetic Operators

- When only selection operator is used, the entire population will be the copies of the best individual.
- When both selection and crossover operators are combined, they will make the algorithms to converge on a good but sub-optimal solution (local optima).
- When only mutation operation is done, then it is similar to random search, which may not result in good solution.
- Combining selection and mutation creates a parallel, noise-tolerant, hill climbing algorithm.

2.7. HYPOTHESIS SPACE SEARCH

- GA employ a randomized beam search method to seek a maximally fit hypothesis. The GA search can move much more abruptly, than the searches occurring in other algorithms like back propagation.
- The update in search space in GA is replacing a parent hypothesis by an offspring that may be totally different from the parent.
- This GA search is less likely to fall into the same kind of local minima that can plague gradient descent methods.
- The major hurdle in GAs is the problem of crowding.

- **Crowding** is a phenomenon in which some individual that is more highly fit than others in the population quickly reproduces, so that copies of this individual and very similar individuals take over a large fraction of the population.
- The negative impact of crowding is that it reduces the diversity of the population, thereby slowing further progress by the GA. The variance among the population is confined.
- One of the promising way to mitigate crowding effect is to alter the selection function, using tournament selection or rank selection in place of fitness proportionate roulette wheel selection.
- Alternatively, fitness sharing can be done in which the measured fitness of an individual is reduced by the presence of other, similar individualism the population.
- A third approach is to restrict the kinds of individuals allowed to recombine to form offspring. For example, by allowing only the most similar individuals to recombine, we can encourage the formation of clusters of similar individuals, or multiple subspecies within the population.
- A related approach is to spatially distribute individuals and allow only nearby individuals to recombine .many of these techniques are inspired by the analogy to biological evolution.

2.7.1. Population Evolution and the Schema Theorem

- The objective of the Schema theorem is to provide a formal model for the effectiveness of the GA search process. This theorem is formalised by Holland and popularized by Goldberg.
- This concentrates on providing a model for the expectation of schema survival, where this naturally represents a limitation in itself.
- An individual bit string can be viewed as a representative of each of the different schemas that it matches.
- The bit string 0010 can be thought of as a representative of 2^4 distinct schemas including 00**, 0* 10, ****, etc.

- A population of bit strings can be viewed in terms of the set of schemas that it represents and the number of individuals associated with each of these schema.
- The schema theorem characterizes the evolution of the population within a GA in terms of the number of instances representing each schema.
- The schema $H = [0\ 1\ *\ 1\ *]$ identifies the chromosome set

0	1	0	1	0
0	1	0	1	1
0	1	1	1	0
0	1	1	1	1

Let $m(s, t)$ denote the number of instances of schemas in the population at time t (i.e., during the t^{th} generation). The schema theorem describes the expected value of $m(s, t + 1)$ in terms of $m(s, t)$ and other properties of the schema, population, and GA algorithm parameters.

- The evolution of the population in the GA depends on the selection step, the recombination step, and the mutation step.
- Let $f(h)$ denote the fitness of the individual bit string h and $\bar{f}(\vec{h})$ denote the average fitness of all individuals in the population at time t .
- Let n be the total number of individuals in the population. Let $h \in s \cap p_t$, indicate that the individual h is both a representative of schemas and a member of the population at time t .
- Let $\hat{u}(s, t)$ denotes the average fitness of instances of schemas in the population at time t .
- $E[m(s, t+1)]$ is the using the probability distribution for selection

$$P_r(h) = \frac{f(h)}{\sum_{i=1}^n f(h_i)}$$

$$\frac{f(h)}{n \bar{f}(t)}$$

- If n member is selected for the new population according to this probability Distribution, then the probability that we will select a representative of schemas is

$$\begin{aligned} P_r(h \in s) &= \frac{\sum_{h \in s \cap p_t} f(h)}{n \bar{f}(t)} \\ &= \frac{\hat{u}(s, t)}{n \bar{f}(t)} m(s, t) \end{aligned}$$

Since
$$\hat{u}(s, t) = \frac{\sum_{h \in s \cap p} f(h)}{m(s, t)}$$

- The above equation gives the probability that a single hypothesis selected by the GA will be an instance of schemas.

$$E(m(s, t + 1)) = \frac{\hat{u}(s, t)}{n \bar{f}(t)} m(s, t)$$

- The schemas with above average fitness to be represented with increasing frequency on successive generations.
- If we view the GA as performing a virtual parallel search through the space of possible schemas at the same time it performs its explicit parallel search through the space of individuals.
- The schema theorem considers only the possible negative influence of the genetic operators.
- The full schema theorem thus provides a lower bound on the expected frequency of schemas, as follows:

$$E(m(s, t + 1)) \geq \frac{\hat{u}(s, t)}{n \bar{f}(t)} m(s, t) \left(1 - p_c \frac{d(s)}{l - 1} \right) (1 - p_m)^{o(s)}$$

$m(s, t)$ = instances of schema s in pop at time t

$\bar{f}(t)$ = average fitness of pop, at time t

$\hat{u}(s, t)$ = ave. fitness of instances of s at time t

p_c = probability of single point crossover operator

p_m = probability of mutation operator

l = length of single bit strings

$o(s)$ = number of defined (non “*”) bits in s

$d(s)$ = distance between leftmost, rightmost defined bits in s

2.8. GENETIC PROGRAMMING

Genetic programming (GP) is a form of evolutionary computation in which the individualism the evolving population are computer programs rather than bit strings.

Genetic programming can automatically create a general solution to a problem in the form of a parameterized topology. Genetic programming starts from a high-level statement of what needs to be done and automatically creates a computer program to solve the problem. Genetic Programming delivers High-Return Human-Competitive Machine Intelligence. Often genetic programming is confused with the term genetic algorithms. The following table states their differences:

Genetic Algorithms (GA)	Genetic Programming (GP)
GAs deal with binary strings or real strings.	GP involves construction of tree structure.
Posts processing of results are needed.	No post processing of results are needed.
The length of the coded string is static.	The length of the tree can be dynamic.

The solution or program designed through genetic programming has two basic nodes:

Terminals: The terminal set contains nodes that provide an input to the GP system.

Functions: Function set contains nodes that process values already in the system.

Basic steps in GP

The five basic preparatory steps in GP are listed below:

- (1) Deciding the set of terminals (i.e.) the independent variables and constant of the problem,
- (2) Deciding the set of primitive functions for each branch of the to-be-evolved computer program
- (3) Specifying the fitness measure
- (4) Specifying the parameters for controlling the running of the program
- (5) Specifying a termination criterion and method for designating the result of the run.

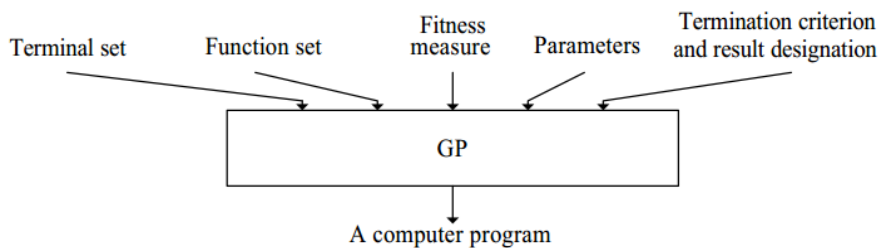


Fig. 2.28. Steps in Genetic Programming

Representing Programs

Programs manipulated by a GP are typically represented by trees corresponding to the parse tree of the program. Each function call is represented by a node in the tree, and the arguments to the function are given by its descendant nodes.

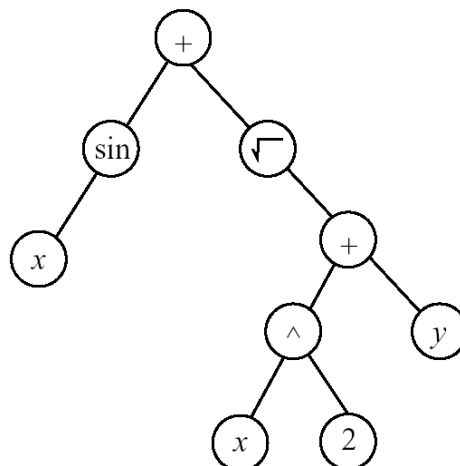
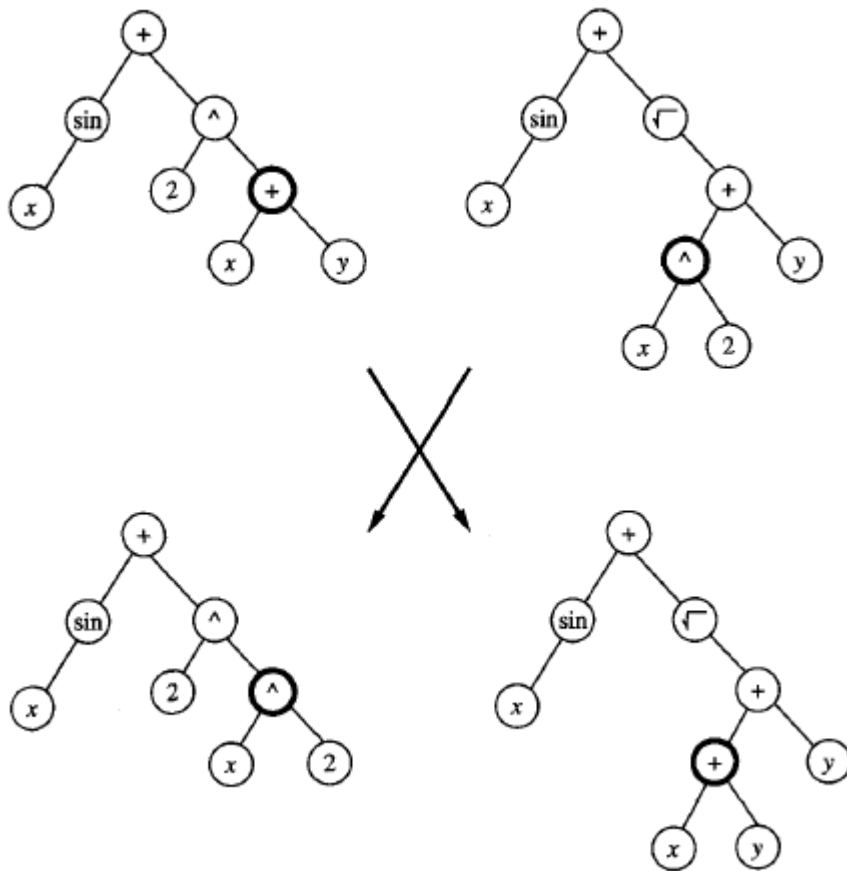


Fig. 2.29. Tree for $\sin x + \sqrt{x^2 + y}$

Crossover Operation

- The prototypical genetic programming algorithm maintains a population of individuals.
- On each iteration, it produces a new generation of individuals using selection, crossover, and mutation.
- The fitness of a given individual program in the population is typically determined by executing the program on a set of training data.
- Crossover operations are performed by replacing a randomly chosen subtree of one parent program by a subtree from the other parent program.

*Fig. 2.30. Crossover Operation*

Example:

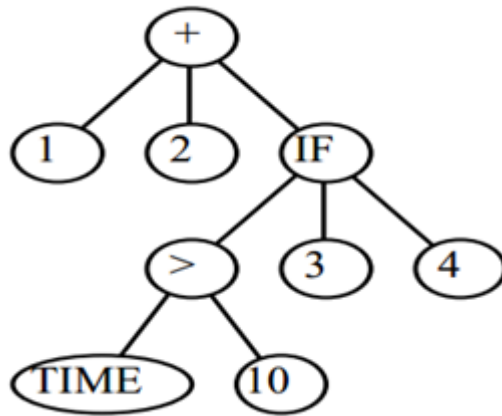
Construct a parse tree for the prefix programming statement

(+ 1 2 (IF (> TIME 10) 3 4)).

Create the terminal set and function set for the give statement.

Terminal set $T = \{1, 2, 10, 3, 4, \text{TIME}\}$

Function set $F = \{+, \text{IF}, >\}$



Perform mutation over the given parental tree.

Steps:

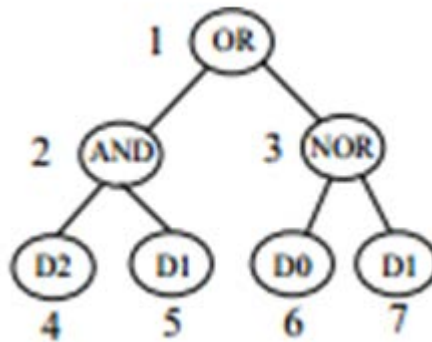
Select parent probabilistically based on fitness

Pick point from 1 to NUMBER-OF-POINTS

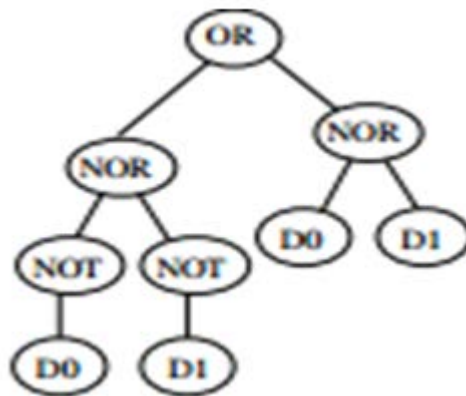
Delete subtree at the picked point

Grow new subtree at the mutation point in same way as generated trees for initial random population (generation 0).

The result is a syntactically valid executable program.



After mutation,



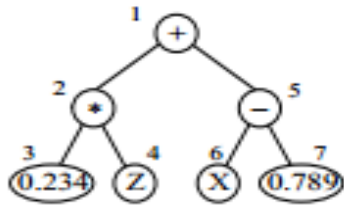
Perform crossover for the given programs (given as trees).

Steps:

- Select two parents probabilistically based on fitness
- Randomly pick a number from 1 to NUMBER-OF-POINTS independently for each of the two parental programs
- Identify the two subtrees rooted at the two picked points

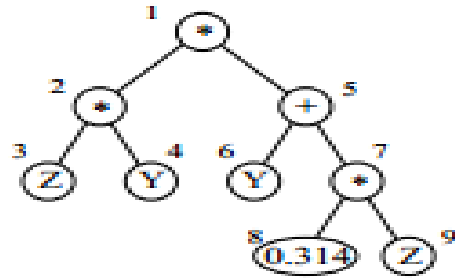
Parent 1: (+ (* 0.234 Z) (- X 0.789))

Parent 2: (* (* Z Y) (+ Y (* 0.314 Z)))



$$0.234Z + X - 0.789$$

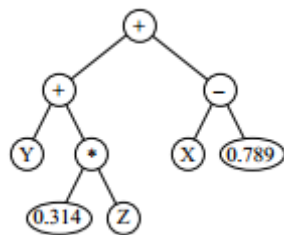
Parent 1



$$ZY(Y + 0.314Z)$$

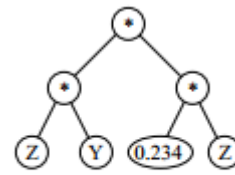
Parent 2

Offsprings:



$$Y + 0.314Z + X - 0.789$$

Offspring 1



$$0.234Z^2Y$$

Offspring 2

2.9. MODELS OF EVALUATION AND LEARNING

2.9.1. Lamarckian Evolution

- Lamarck proposed that evolution over many generations was directly influenced by the experiences of individual organisms during their lifetime.
- The experiences of a single organism directly affected the genetic makeup of their offspring: If an individual learned during its lifetime to avoid some toxic food, it could pass this trait on genetically to its offspring, which therefore would not need to learn the trait.
- This is an attractive conjecture, because it would presumably allow for more efficient evolutionary progress than a generate-and-test process that ignores the experience gained during an individual's lifetime.

- Despite the attractiveness of this theory, current scientific evidence overwhelmingly contradicts Lamarck's model.
- The currently accepted view is that the genetic makeup of an individual is, in fact, unaffected by the lifetime experience of one's biological parents.
- Despite this apparent biological fact, recent computer studies have shown that Lamarckian processes can sometimes improve the effectiveness of computerized genetic.

2.9.2. Baldwin Effect

- The Lamarckian evolution is not an accepted model of biological evolution, other mechanisms have been suggested by which individual learning can alter the course of evolution.
- The Baldwin effect is based on the following observations:
- If a species is evolving in a changing environment, there will be evolutionary pressure to favour individuals with the capability to learn during their lifetime.
- Those individuals who are able to learn many traits will rely less strongly on their genetic code to hard-wire traits. As a result, these individuals can support a more diverse gene pool, relying on individual learning to overcome the "missing" or "not quite optimized" traits in the genetic code.
- The Baldwin effect provides an indirect mechanism for individual learning to positively impact the rate of evolutionary progress.
- By increasing survivability and genetic diversity of the species, individual learning supports more rapid evolutionary progress, thereby increasing the chance that the species will evolve genetic, non learned traits that better fit the new environment.
- There have been several attempts to develop computational models to study the Baldwin effect.
- The genetic makeup of the individual determined which weights were trainable and which were fixed. In their experiments, when no individual learning was allowed, the population failed to improve its fitness over time.

- However, when individual learning was allowed, the population quickly improved its fitness.
- During early generations of evolution the population contained a greater proportion of individuals with many trainable weights. However, as evolution proceeded, the number of fixed, correct network weights tended to increase, as the population evolved toward genetically given weight values and toward less dependence on individual learning of weights.

UNIT

3

BAYESIAN AND COMPUTATIONAL LEARNING

3.1. INTRODUCTION TO BAYESIAN LEARNING

Bayesian reasoning provides a probabilistic approach to drawing inferences. It is based on the assumption that the quantities of interest are governed by probability distributions and that optimal decisions can be made by reasoning about these probabilities together with observed data. This provides a quantitative approach for weighing the evidence supporting alternative hypotheses. Bayesian reasoning is a platform for learning algorithms that directly manipulate probabilities and to analyse the operation of other algorithms that do not explicitly manipulate probabilities. Bayesian machine learning is a particular set of approaches **to probabilistic machine learning that treats model parameters as random variables.**

Bayesian learning outperforms other learning because:

- It provides practical learning algorithms such as Naive Bayes, Bayesian belief network learning that combine prior knowledge (prior probabilities) with observed data. This method requires prior probabilities
- It provides useful conceptual framework for evaluating other learning algorithms

Features of Bayesian Learning

- Each observed training example can incrementally decrease or increase the estimated probability that a hypothesis is correct. This provides a more flexible approach to learning than algorithms that completely eliminate a hypothesis if it is found to be inconsistent with any single example.

- Prior knowledge can be combined with observed data to determine the final probability of a hypothesis. In Bayesian learning, prior knowledge is provided by asserting
 - ✓ a prior probability for each candidate hypothesis
 - ✓ a probability distribution over observed data for each possible hypothesis.
- Bayesian methods can accommodate hypotheses that make probabilistic predictions.
- New instances can be classified by combining the predictions of multiple hypotheses, weighted by their probabilities.
- They are computationally intractable, that is they can provide a standard of optimal decision making against which other practical methods can be measured.

Limitations of Bayesian Learning

- ✓ Require initial knowledge of many probabilities. When these probabilities are not known in advance they are often estimated based on background knowledge, previously available data, and assumptions about the form of the underlying distributions.
- ✓ It demands significant computational cost required to determine the Bayes optimal hypothesis in the general case.

3.2. BAYES THEOREM

- Bayes theorem describes how the conditional probability of an event or a hypothesis can be computed using evidence and prior knowledge.

Bayes theorem provides a way to calculate the probability of a hypothesis based on its prior probability, the probabilities of observing various data given the hypothesis, and the observed data itself

- Mathematically, Bayes theorem is given by

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

$\Rightarrow h$ is called the proposition and D is called the evidence.

$$P(\text{rain} | \text{cloud}) = \frac{P(\text{cloud} | \text{rain}) P(\text{rain})}{P(\text{cloud})}$$

$\Rightarrow P(h)$ is called the *prior* probability of *proposition* [P(rain) is true means its raining] and $P(D)$ is called the *prior* probability of *evidence* [P(cloud) is true, means its cloudy].

$\Rightarrow P(h | D)$ is called the posterior probability of h given D. [Probability of raining is true given its Cloudy]

$\Rightarrow P(D | h)$ is called the likelihood of D given h . [Probability of Cloudy is true given its Raining]

Example: Three factories A, B, C of an electric bulb manufacturing company produce respectively 35%, 35% and 30% of the total output. Approximately 1.5%, 1% and 2% of the bulbs produced by these factories are known to be defective. If a randomly selected bulb manufactured by the company was found to be defective, what is the probability that the bulb was manufactures in factory A?

Solution:

Let A, B, C denote the events that a randomly selected bulb was manufactured in factory A, B, C respectively. Let D denote the event that a bulb is defective. We have the following data:

$$P(A) = 0.35, P(B) = 0.35, P(C) = 0.30 \text{ and}$$

$$P(D | A) = 0.015, P(D | B) = 0.010, P(D | C) = 0.020$$

We are required to find $P(A | D)$. By the generalisation of the Bayes' theorem we have:

$$\begin{aligned} P(A | D) &= \frac{P(D | A) P(A)}{P(D | A) P(A) + P(D | B) P(A) + P(D | C) P(C)} \\ &= \frac{0.015 \times 0.35}{0.015 \times 0.35 + 0.010 \times 0.35 + 0.020 \times 0.30} = 0.356 \end{aligned}$$

Hence, 35.6% probability that the bulb manufactured by the company A was found to be defective.

Maximum A Posterior (MAP)

- The learner considers some set of candidate hypotheses H and is interested in finding the most probable hypothesis $h \in H$ given the observed data D .
- Any such maximally probable hypothesis is called a maximum a posterior (MAP) hypothesis. This can be found by Bayes theorem to calculate the posterior probability of each candidate hypothesis.
- This is an estimate of an unknown quantity, that equals the mode of the posterior distribution.
- The MAP can be used to obtain a point estimate of an unobserved quantity on the basis of empirical data.

Maximum a posterior hypothesis h_{MAP}

$$\begin{aligned}
 h_{\text{MAP}} &= \operatorname{argmax}_{h \in H} P(h|D) \\
 &= \operatorname{argmax}_{h \in H} \frac{P(D|h) P(h)}{P(D)} \\
 &= \operatorname{argmax}_{h \in H} P(D|h) P(h)
 \end{aligned}$$

Maximum Likelihood (ML) Hypothesis

- Sometimes it is assumed that every hypothesis is equally probable a priori. In this case, the equation above can be simplified because $P(D|h)$ is often called the likelihood of D given h , any hypothesis that maximizes $P(D|h)$ is called maximum likelihood (ML) hypothesis
- When $P(h_i) = P(h_j)$, it can be further simplified, and become Maximum likelihood (ML) hypothesis

$$h_{\text{ML}} = \operatorname{argmax}_{h_i \in H} P(D|h_i)$$

Differences between ML and MAP

Sl. No.	Maximum A Posterior	Maximum Likelihood
1	Maximum a posterior (MAP) estimation is the value of the parameter that maximizes the entire posterior	ML of a parameter is the value of the parameter that maximizes the likelihood, where the likelihood is a

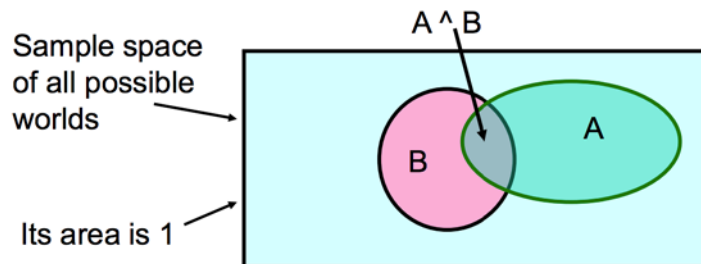
	distribution (which is calculated using the likelihood). A MAP estimate is the mode of the posterior distribution.	function of the parameter and is actually equal to the probability of the data conditioning on that parameter.
2	Suppose x is the quantity we are trying to estimate, and that some operation on x has yielded y . what is the best possible value of x given the fact that the observation has a value y .	This is about what value of x best explains the observed value, y .
3	MAP estimation produces the value of x that maximizes $p(y x) * p(x)$.	ML estimation attempts to find the value of x that maximizes $p(y x)$

3.3. CONCEPT LEARNING AND BAYES THEOREM

Bayes theorem provides a way to calculate the posterior probability of each hypothesis given the training data, it can be used as the basis for a straightforward learning algorithm that calculates the probability for each possible hypothesis, and then outputs the most probable.

3.3.1. Fundamental Probability Rules

- **Product Rule (Joint probability):** The probability $P(A \wedge B)$ of a conjunction of two events A and B :



$$P(A \cap B) = P(A|B) P(B) = P(B|A) P(A)$$

- **Sum Rule (Union Rule):** The probability of a disjunction of two events A and B :

$$P(A \cup B) = P(A) + P(B) - P(A \cap B)$$

If the events A and B are mutually exclusive, then

$$P(A \cup B) = P(A) + P(B)$$

- **Theorem of total probability:** if events A_1, \dots, A_n are mutually exclusive with, then

$$\sum_{i=1}^n P(A_i) = 1$$

$$P(B) = \sum_{i=1}^n P(B|A_i) P(A_i)$$

Conditional Probability: The conditional probability of event A, given that event B is true, as follows:

$$P(A|B) = \frac{P(A, B)}{P(B)} \text{ if } p(B) > 0$$

3.3.2. Brute Force MAP Hypothesis Learner

A learner is some finite hypothesis space H defined over instance space X whose task is to learn some target concept $c: X \rightarrow [0, 1]$. The learner is set of training examples of the form (x_i, d_i) , where x is some instance from training set and d is the target value.

Brute Force MAP learning algorithm

1. For each hypothesis h in H , calculate the posterior probability

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

2. Output the hypothesis h_{MAP} with the highest posterior probability

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

- This algorithm requires significant computation, because it applies Bayes theorem to each hypothesis in H to calculate $P(h|D)$.
- In order solve a learning problem using the brute force learning algorithm, the values of $P(h)$ and $P(D|h)$ must be specified.
- The $P(h)$ and $P(D|h)$ must adhere to the following assumptions:
 - ✓ The training data D is noise free. (i.e., $d_i = c(x_i)$)
 - ✓ The target concept c is contained in the hypothesis space H
 $(\exists h \in H) [(\forall x \in X) [h(x) = c(x)]]$
 - ✓ There is no reason to believe that any hypothesis is more probable than any other.

$$P(h) = \frac{1}{|H|} \text{ for all } h \in H$$

$$P(D|h) = \begin{cases} 1 & \text{if } d_i = h(x_i) \text{ for all } d_i \in D \\ 0 & \text{otherwise} \end{cases}$$

Let $\langle x_1, x_2, x_3, \dots, x_m \rangle$ be the fixed set of instances with targets $D = \langle c(x_1), c(x_2), c(x_3), \dots, c(x_m) \rangle$. Then choose $P(D|h)$ such that

- $P(D|h) = 1$ if h consistent with D
- $P(D|h) = 0$ otherwise

Choose $P(h)$ to be uniform distribution:

$$P(h) = 1/|H| \text{ for all } h \text{ in } H$$

Then,

$$P(h|D) = \begin{cases} \frac{1}{|VS_{H,D}|} & \text{if } h \text{ is consistent with } D \\ 0 & \text{otherwise} \end{cases}$$

The probability of data D given hypothesis h is 1 if D is consistent with h , and 0 otherwise. As training data accumulates, the posterior probability for inconsistent hypotheses becomes zero while the total probability summing to one is shared equally among the remaining consistent hypotheses.

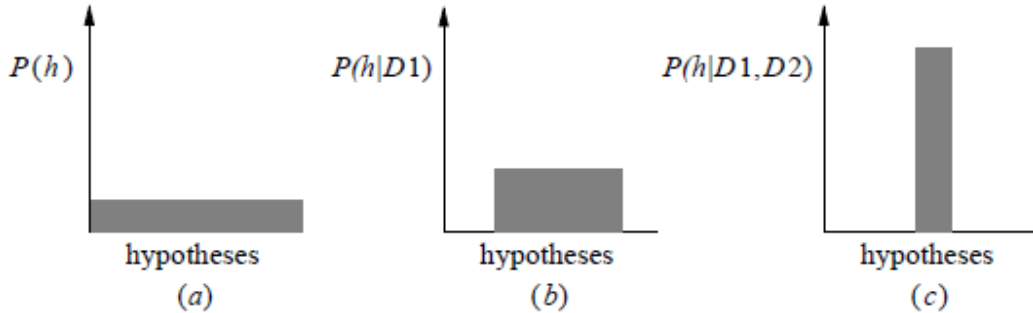


Fig. 3.1. Evolution of Posterior probabilities

In the Fig 3.1 a) all the hypotheses have the same probability, whereas in (b) + (c) as training data accumulates, the posterior probability builds on. In case of inconsistent hypotheses becomes zero while the total probability summing to 1 is shared equally among the remaining consistent hypotheses.

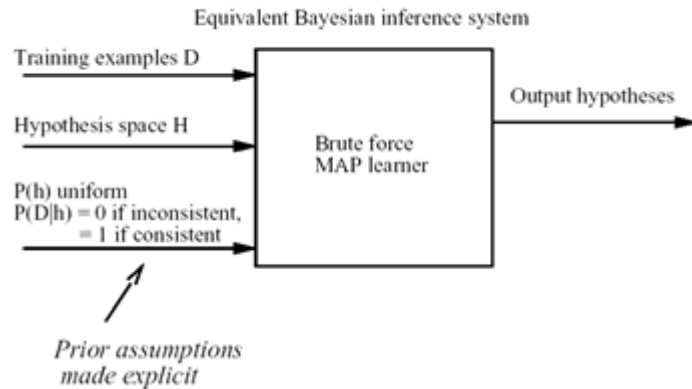


Fig. 3.2. Overview of Brute force MAP learning

3.3.3. MAP Hypothesis and Consistent Learners

Every consistent learner outputs a MAP hypothesis, on uniform prior probability distribution over H (i.e., $P(h_i) = P(h_j)$ for all (i,j)), and on deterministic, noise free training data (i.e., $P(D|h) = 1$ if D and h are consistent, and 0 otherwise).

- A learning algorithm is a consistent learner when it outputs a hypothesis that commits zero errors over the training examples.
- The Bayesian framework allows one way to characterize the behaviour of learning algorithms, even when the learning algorithm does not explicitly manipulate probabilities.
- By identifying probability distributions $P(h)$ and $P(D|h)$ under which the algorithm outputs optimal hypotheses, which characterizes the implicit assumptions, under which this algorithm behaves optimally.
- Here, instead of modeling the inductive inference method by an equivalent deductive System, we model it by an equivalent probabilistic reasoning system based on Bayes theorem.
- The implicit assumptions that we attribute to the learner are assumptions of the form "the prior probabilities over H are given by the distribution $P(h)$, and the strength of data in rejecting or accepting a hypothesis is given by $P(D|h)$."

3.4. MAXIMUM LIKELIHOOD AND LEAST-SQUARED ERROR HYPOTHESES

- In statistics, maximum likelihood estimation (MLE) is a method of estimating the parameters of a probability distribution by maximizing a likelihood function, so that under the assumed statistical model the observed data is most probable.
- Maximum likelihood estimation is a method that determines values for the parameters of a model.
- The parameter values are found such that they maximise the likelihood that the process described by the model produced the data that were actually observed.

Under certain assumptions any learning algorithm that minimizes the squared error between the output hypothesis predictions and the training data will output a maximum likelihood hypothesis.

- This infers that the Bayesian justification for many neural network and other curve fitting methods that attempt to minimize the sum of squared errors over the training data.
- Consider Learner L considers an instance space X and a hypothesis space H consisting of some class of real-valued functions defined over X.
- The problem faced by L is to learn an unknown target function $f: X \rightarrow Y$ drawn from H.
- A set of m training examples is provided, where the target value of each example is corrupted by random noise drawn according to a Normal probability distribution.
- More precisely, each training example is a pair of the form (x_i, d_i) where $d_i = f(x_i) + e_i$.
- The task of the learner is to output a maximum likelihood hypothesis, or, equivalently, a MAP hypothesis assuming all hypotheses are equally probable a priori.

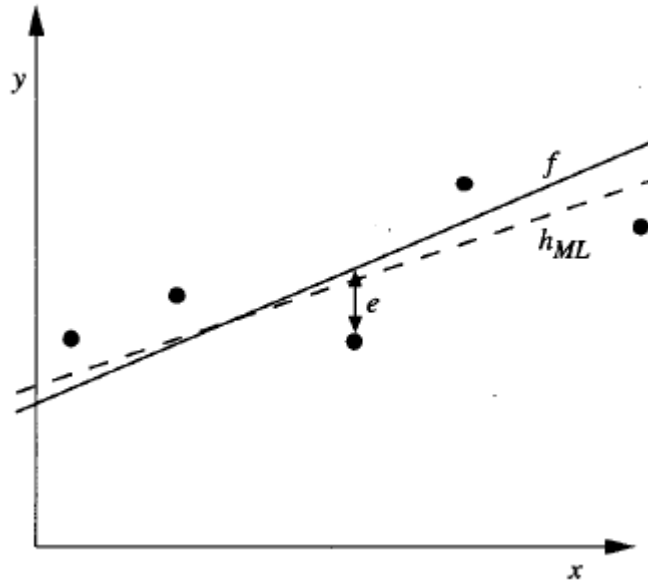


Fig. 3.3. Learning a real valued Function

- In Fig 3.3, the target function f corresponds to the solid line.
- The training examples (x_i, d_i) are assumed to have Normally distributed noise e_i with zero mean added to the true target value $f(x_i)$.
- The dashed line corresponds to the linear function that minimizes the sum of squared errors.

Probability Density Function (PDF)

A probability density function (PDF) is a mathematical function that describes the probability of each member of a discrete set or a continuous range of outcomes or possible values of a variable.

$$p(x_0) = \lim_{\epsilon \rightarrow 0} \frac{1}{\epsilon} P(x_0 \leq x < x_0 + \epsilon)$$

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 . The noise variable follows normal distribution.

The maximum likelihood hypothesis h is the one that minimizes the sum of squared errors:

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

For a fixed set of training instances (x_1, \dots, x_m) and therefore consider the data D to be the corresponding sequence of target values $D = (d_1 \dots d_m)$. Here $d_i = f(x_i) + e_i$. Assuming the training examples are mutually independent given h ,

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

The noise e_i obeys a Normal distribution with zero mean and unknown variance, each d_i must also obey a Normal distribution with variance σ^2 centered around the true target value $f(x_i)$ rather than zero.

$$\begin{aligned} h_{\text{ML}} &= \underset{h \in H}{\operatorname{argmin}} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - \mu)^2} \\ &= \underset{h \in H}{\operatorname{argmin}} \prod_{i=1}^m \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{1}{2\sigma^2}(d_i - h(x_i))^2} \end{aligned}$$

Applying transformation that is common in maximum likelihood calculations: Rather than maximizing the above complicated expression we shall choose to maximize its logarithm. This is justified because $\ln p$ is a monotonic function of p . Therefore maximizing $\ln p$ also maximizes p .

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

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

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

Maximizing this negative quantity is equivalent to minimizing the corresponding positive quantity.

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

Discard constants that are independent of h .

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

Thus the maximum likelihood hypothesis minimizes the sum of the squared errors between the observed training values and the hypothesis predictions.

3.5. MAXIMUM LIKELIHOOD HYPOTHESES

The maximum likelihood hypothesis is the one that minimizes the sum of squared errors over the training examples. To maximize the likelihood find $P(D|h)$ as:

$$P(D|h) = \prod_{i=1}^m P(x_i, d_i|h)$$

where x_i and d_i as random variables, and assuming that each training example is drawn independently. Also, the probability of encountering any particular instance x_i is independent of the hypothesis h . When x is independent of h , rewrite as:

$$\begin{aligned} P(D|h) &= \prod_{i=1}^m P(x_i, d_i|h) = \prod_{i=1}^m P(d_i|h, x_i) P(x_i) \\ P(d_i|h, x_i) &= \begin{cases} h(x_i) & \text{if } d_i = 1 \\ (1 - h(x_i)) & \text{if } d_i = 0 \end{cases} \\ P(d_i|h, x_i) &= h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} \end{aligned}$$

When $d_i = 1$, the second term in the previous equation becomes 1. Hence

$$\begin{aligned} P(d_i = 1|h, x_i) &= h(x_i) \\ P(D|h) &= \prod_{i=1}^m h(x_i)^{d_i} (1 - h(x_i))^{1-d_i} P(x_i) \end{aligned}$$

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

The last term is independent of h . So it can be dropped

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

3.5.1. Gradient Search to Maximize Likelihood in a Neural Net

The gradient of $G(h, D)$ is given by the vector of partial derivatives of $G(h, D)$ with respect to the various network weights that define the hypothesis h represented by the

learned network. The partial derivative of $G(h, D)$ with respect to weight w_{jk} from input k to unit j is:

$$\begin{aligned}\frac{\partial G(h, D)}{\partial w_{jk}} &= \sum_{i=1}^m \frac{\partial G(h, D)}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^m \frac{\partial (d_i \ln h(x_i) + (1 - d_i) \ln (1 - h(x_i)))}{\partial h(x_i)} \frac{\partial h(x_i)}{\partial w_{jk}} \\ &= \sum_{i=1}^m \frac{d_i - h(x_i)}{h(x_i) (1 - h(x_i))} \frac{\partial h(x_i)}{\partial w_{jk}}\end{aligned}$$

The neural network consists of single layered sigmoid units:

$$\frac{\partial h(x_i)}{\partial w_{jk}} = \sigma'(x_i) x_{ijk} = h(x_i) (1 - h(x_i)) x_{ijk}$$

The expression for the derivatives that constitute the gradient

$$\frac{\partial G(h, D)}{\partial w_{jk}} = \sum_{i=1}^m (d_i - h(x_i)) x_{ijk}$$

Since the aim is to maximize not to minimize $P(D|h)$, gradient ascent is performed rather than gradient descent search.

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where

$$\Delta w_{jk} = \eta \sum_{i=1}^m (d_i - h(x_i)) x_{ijk}$$

The above equation is also seen as update rule for output unit weights. So rewriting,

$$w_{jk} \leftarrow w_{jk} + \Delta w_{jk}$$

Where

$$\Delta w_{jk} = \eta \sum_{i=1}^m h(x_i) (1 - h(x_i)) (d_i - h(x_i)) x_{ijk}$$

In short, the two weight update rules infer:

- The rule that minimizes sum of squared error seeks the maximum likelihood hypothesis under the assumption that the training data can be modeled by Normally distributed noise added to the target function value.

- The rule that minimizes cross entropy seeks the maximum likelihood hypothesis under the assumption that the observed boolean value is a probabilistic function of the input instance.

3.6. MINIMUM DESCRIPTION LENGTH (MDL) PRINCIPLE

MDL is based on the following insight: any regularity in the data can be used to compress the data, i.e. to describe it using fewer symbols than the number of symbols needed to describe the data literally.

The principle of MDL is from information theory. Consider the problem of designing a code C to transmit messages drawn at random probability of encountering message i is p_i . The compacted code is represented by C . It has been found by Shannon that number of bits to form optimal code is $-\log_2 p_i$.

MDL methods are particularly well-suited for dealing with model selection, prediction, and estimation problems in situations where the models under consideration can be arbitrarily complex, and overfitting the data is a serious concern.

The h_{MAP} is estimated using,

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

$$h_{\text{MAP}} = \underset{h \in H}{\operatorname{argmax}} \log_2 P(D|h) P(h) + \log_2 P(h)$$

$$h_{\text{MAP}} = \underset{h \in H}{\operatorname{argmin}} -\log_2 P(D|h) - \log_2 P(h)$$

The last equation minimizes the negative of this quantity.

Applying Shannon's method of optimizing number of bits to MDL,

- $L_{\text{CH}}(h) = -\log_2 P(h)$, where C_H is the optimal code for hypothesis space H .

$$L_{\text{C}_{D|h}}(D|h) = -\log_2 P(D|h)$$

Where $C_{D|h}$ is the optimal code for describing data D assuming that both the sender and receiver know hypothesis h

Rewriting h_{MAP} as that minimizes the sum given by the description length of the hypothesis plus the description length of the data given the hypothesis:

$$h_{\text{MAP}} = \underset{h \in H}{\operatorname{argmin}} L_{C_H}(h) + L_{C_D|h}(D|h)$$

The Minimum Description Length (MDL) principle recommends choosing the hypothesis that minimizes the sum of these two description lengths. To apply this principle in practice chooses specific encodings or representations appropriate for the given learning task. Assuming to use the codes C_1 and C_2 to represent the hypothesis and the data given the hypothesis, we can state the MDL principle as

Minimum Description Length: Choose h_{MDL} where

$$h_{\text{MDL}} = \underset{h \in H}{\operatorname{argmin}} L_{C_1}(h) + L_{C_2}(D|h)$$

- The MDL principle as recommending the shortest method for re-encoding the training data, where we count both the size of the hypothesis and any additional cost of encoding the data given this hypothesis
- Thus the MDL principle provides a way of trading off hypothesis complexity for the number of errors committed by the hypothesis. It might select a shorter hypothesis that makes a few errors over a longer hypothesis that perfectly classifies the training data.
- Viewed in this light, it provides one method for dealing with the issue of overfitting the data.

3.7. OPTIMAL BAYES CLASSIFIER

Bayes Optimal Classifier is a probabilistic model that finds the most probable prediction using the training data and space of hypotheses to make a prediction for a new data instance.

- The Bayes Theorem provides a principled way for calculating a conditional probability.
- It is also closely related to the Maximum a Posterior which is a probabilistic framework referred to as MAP that finds the most probable hypothesis for a training dataset.
- But the Bayes Optimal Classifier is computationally expensive.

- If the possible classification of the new example can take on any value v_j from some set V , then the probability $P(v_j|D)$ that the correct classification for the new instance is v_j :

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

Example: Consider the set of hypothesis $H = \{h_1, h_2, h_3\}$ where $P(h_1) = 0.4$; $P(h_2) = 0.3$; $P(h_3) = 0.3$; A new instance x is classified positive by h_1 and negative by h_2 and h_3 .

$$\begin{array}{lll} P(h_1, D) = 4 & P(\ominus, h_1) = 0 & P(\oplus, h_1) = 1 \\ P(h_2, D) = 3 & P(\ominus, h_2) = 1 & P(\oplus, h_2) = 0 \\ P(h_3, D) = 3 & P(\ominus, h_3) = 1 & P(\oplus, h_3) = 0 \end{array}$$

Where \ominus denotes negative class and \oplus denotes positive class.

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

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

Hence, according to optimal Bayes classifier,

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

This method maximizes the probability that the new instance is classified correctly, given the available data, hypothesis space, and prior probabilities over the hypotheses.

3.8. GIBBS ALGORITHM

Though Bayes classifier yields accurate results, it is computationally expensive since it has to estimate the prior probabilities. Gibb's algorithm reduces the computations.

Generalised Gibb's Algorithm

1. Choose a hypothesis h from H at random, according to the posterior probability distribution over H .
2. Use h to predict the classification of the next instance x .

- Given a new instance to classify, the Gibbs algorithm simply applies a hypothesis drawn at random according to the current posterior probability distribution.

- The expected value is taken over target concepts drawn at random according to the prior probability distribution assumed by the learner.
- Under this condition, the expected value of the error of the Gibbs algorithm is at worst twice the expected value of the error of the Bayes optimal classifier.

$$E[\text{error}_{\text{Gibbs}}] \leq 2E[\text{error}_{\text{Bayes Optimal}}]$$

- In particular, it implies that if the learner assumes a uniform prior over H , and if target concepts are in fact drawn from such a distribution when presented to the learner, then classifying the next instance according to a hypothesis drawn at random from the current version space will have expected error at most twice that of the Bayes optimal classifier.

3.9. NAIVE BAYES ALGORITHM

Naive Bayes algorithm applies to learning tasks where each instance x is described by a conjunction of attribute values and where the target function $f(x)$ can take on any value from some finite set V .

When a set of training examples of the target function is provided, and a new instance is presented, described by the tuple of attribute values $\langle a_1, a_2, \dots, a_n \rangle$, the learner will be asked to predict the target value, or classification, for this new instance. The Bayesian approach would be

$$\begin{aligned} u_{\text{MAP}} &= \underset{v_j \in V}{\operatorname{argmax}} P(v_j | a_1, a_2, \dots, a_n) \\ &= \underset{v_j \in V}{\operatorname{argmax}} \frac{P(a_1, a_2, \dots, a_n | v_j) P(v_j)}{P(a_1, a_2, \dots, a_n)} \\ &= \underset{v_j \in V}{\operatorname{argmax}} P(a_1, a_2, \dots, a_n | v_j) P(v_j) \end{aligned}$$

To solve the above equation we need two terms

- ✓ $P(v_j)$: Obtained by counting the frequency with which each target value v_j occurs in the training data.
- ✓ $P(a_1, a_2, \dots, a_n | v_j)$: Obtaining this term is very difficult. The problem is that the number of these terms is equal to the number of possible instances times the

number of possible target values. Therefore, we need to see every instance in the instance space many times in order to obtain reliable estimates.

To simplify this, Naïve Bayes classifier is used. Here, the attribute values are conditionally independent.

$$P(a_1, a_2, \dots, a_n | v_j) = \prod_i P(a_i | v_j)$$

Hence, number terms is $|\text{distinct attributes}| \cdot |\text{distinct target values}| + |\text{distinct target values}|$

The final Naïve Bayes classifier is given as:

$$u_{NB} = \underset{v_j \in V}{\operatorname{argmax}} \prod_i P(a_i | v_j)$$

Example: Use Naïve Bayes theorem to find out the likelihood of playing tennis for a given set weather attributes.

$f(x) \in v = (\text{yes}, \text{no})$ i.e. $v = (\text{yes we will play tennis}, \text{no we will not play tennis})$

The attribute values are $a_0 \dots a_3 = (\text{Outlook}, \text{Temperature}, \text{Humidity}, \text{and Wind})$.

TRAINING EXAMPLES

Day	Outlook	Temperature	Humidity	Wind	Play Tennis
1	Sunny	Hot	High	Weak	No
2	Sunny	Hot	High	Strong	No
3	Overcast	Hot	High	Weak	Yes
4	Rain	Mild	High	Weak	Yes
5	Rain	Cool	Normal	Weak	Yes
6	Rain	Cool	Normal	Strong	No
7	Overcast	Cool	Normal	Strong	Yes
8	Sunny	Mild	High	Weak	No
9	Sunny	Cool	Normal	Weak	Yes
10	Rain	Mild	Normal	Weak	Yes
11	Sunny	Mild	Normal	Strong	Yes
12	Overcast	Mild	High	Strong	Yes
13	Overcast	Hot	Normal	Weak	Yes
14	Rain	Mild	High	Strong	No

According to Bayes theorem,

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

$$P(\text{Play Tennis}|\text{Attributes}) = \frac{P(\text{Attributes}|\text{Play Tennis}) * P(\text{Play Tennis})}{P(\text{Attributes})}$$

Or

$$P(v/a) = \frac{P(a|v) * P(v)}{P(a)}$$

$$P(a|v) = P(a_0 \dots a_3 | v_{0,1})$$

$$P(a|v) = P(\text{Outlook, Temperature, Humidity, Wind} | \text{Play tennis, Don't Play tennis})$$

According to Naïve Bayes,

$$P(a_0 \dots a_3 | v_{j=0,1}) \approx P(a_0|v_0) * P(a_1|v_0) * P(a_n|v_0) \approx P(a_0|v_1) * P(a_1|v_1) * P(a_n|v_1)$$

or

$$P(a_0 \dots a_n | v_j) \approx \prod_i P(a_i | v_j)$$

$P(\text{outlook} = \text{sunny, temperature} = \text{cool, humidity} = \text{normal, wind} = \text{strong} | \text{Play tennis})$

$$\approx P(\text{outlook} = \text{sunny} | \text{Play tennis}) * P(\text{temperature} = \text{cool} | \text{Play tennis}) *$$

$$P(\text{humidity} = \text{normal} | \text{Play tennis}) * P(\text{wind} = \text{strong} | \text{Play tennis})$$

The probability of observing $P(a_0 \dots a_n | v_j)$ is equal the product of probabilities of observing the individual attributes. Using the table of 14 examples we can calculate our overall probabilities and conditional probabilities.

Probability of playing tennis:

- $P(\text{Play Tennis} = \text{Yes}) = 9/14 = .64$
- $P(\text{Play Tennis} = \text{No}) = 5/14 = .36$

Outlook:

Sunny

- $P(\text{Outlook} = \text{Sunny} | \text{Play Tennis} = \text{Yes}) = 2/9 = .22$
- $P(\text{Outlook} = \text{Sunny} | \text{Play Tennis} = \text{No}) = 3/5 = .6$

Overcast

- $P(\text{Outlook} = \text{Overcast} | \text{Play Tennis} = \text{Yes}) = 4/9 = .44$
- $P(\text{Outlook} = \text{Overcast} | \text{Play Tennis} = \text{No}) = 0/5 = 0$

Rain

- $P(\text{Outlook} = \text{Rain} \mid \text{Play Tennis} = \text{Yes}) = 3/9 = .33$
- $P(\text{Outlook} = \text{Rain} \mid \text{Play Tennis} = \text{No}) = 2/5 = .4$

Temperature**Hot**

- $P(\text{Temperature} = \text{Hot} \mid \text{Play Tennis} = \text{Yes}) = 2/9 = .22$
- $P(\text{Temperature} = \text{Hot} \mid \text{Play Tennis} = \text{No}) = 2/5 = .40$

Mild

- $P(\text{Temperature} = \text{Mild} \mid \text{Play Tennis} = \text{Yes}) = 4/9 = .44$
- $P(\text{Temperature} = \text{Mild} \mid \text{Play Tennis} = \text{No}) = 2/5 = .40$

Cool

- $P(\text{Temperature} = \text{Cool} \mid \text{Play Tennis} = \text{Yes}) = 3/9 = .33$
- $P(\text{Temperature} = \text{Cool} \mid \text{Play Tennis} = \text{No}) = 1/5 = .20$

Humidity**High**

- $P(\text{Humidity} = \text{Hi} \mid \text{Play Tennis} = \text{Yes}) = 3/9 = .33$
- $P(\text{Humidity} = \text{Hi} \mid \text{Play Tennis} = \text{No}) = 4/5 = .80$

Normal

- $P(\text{Humidity} = \text{Normal} \mid \text{Play Tennis} = \text{Yes}) = 6/9 = .66$
- $P(\text{Humidity} = \text{Normal} \mid \text{Play Tennis} = \text{No}) = 1/5 = .20$

Wind**Weak**

- $P(\text{Wind} = \text{Weak} \mid \text{Play Tennis} = \text{Yes}) = 6/9 = .66$
- $P(\text{Wind} = \text{Weak} \mid \text{Play Tennis} = \text{No}) = 2/5 = .40$

Strong

- $P(\text{Wind} = \text{Strong} \mid \text{Play Tennis} = \text{Yes}) = 3/9 = .33$
- $P(\text{Wind} = \text{Strong} \mid \text{Play Tennis} = \text{No}) = 3/5 = .60$

$a = (\text{Outlook} = \text{sunny}, \text{Temperature} = \text{cool}, \text{Humidity} = \text{high}, \text{Wind} = \text{strong})$

What would our Naïve Bayes classifier predict in terms of playing tennis on a day like this?

- i) $P(\text{Playtennis} = \text{Yes} \mid (\text{Outlook} = \text{sunny}, \text{Temperature} = \text{cool}, \text{Humidity} = \text{high}, \text{Wind} = \text{strong}))$

OR

- ii) $P(\text{Playtennis} = \text{No} \mid (\text{Outlook} = \text{sunny}, \text{Temperature} = \text{cool}, \text{Humidity} = \text{high}, \text{Wind} = \text{strong}))$

$$\begin{aligned}
 \text{i) } P(\text{Yes} \mid (\text{sunny}, \text{cool}, \text{high}, \text{strong})) &= \frac{P(\text{sunny}, \text{cool}, \text{high}, \text{strong}) \mid (\text{yes}) * P(\text{yes})}{P(\text{sunny}, \text{cool}, \text{high}, \text{strong})} \\
 &= \frac{P(\text{sunny} \mid \text{yes}) * P(\text{cool} \mid \text{yes}) * P(\text{high} \mid \text{yes}) * P(\text{strong} \mid \text{yes}) * P(\text{yes})}{P((\text{sunny}, \text{cool}, \text{high}, \text{strong}) \mid \text{Yes}) + P((\text{sunny}, \text{cool}, \text{high}, \text{strong}) \mid \text{No})} \\
 &= \frac{(0.22 * 0.33 * 0.33 * 0.33) * 0.64}{(0.22 * 0.33 * 0.33 * 0.33) * 0.64 + (0.6 * 0.2 * 0.8 * 0.6) * 0.36} \\
 &= 0.0051 / (0.0051 + 0.0207) \\
 &= 0.1977
 \end{aligned}$$

$$\begin{aligned}
 \text{ii) } P(\text{No} \mid (\text{sunny}, \text{cool}, \text{high}, \text{strong})) &= \frac{P(\text{sunny}, \text{cool}, \text{high}, \text{strong}) \mid (\text{No}) * P(\text{No})}{P(\text{sunny}, \text{cool}, \text{high}, \text{strong})} \\
 &= \frac{0.0207}{0.0051 + 0.0207} \\
 &= 0.8023
 \end{aligned}$$

Naive Bayes is best when:

- Moderate or large training set available
- Attributes that describe instances are conditionally independent given classification

Bayes vs Naive Bayes Classifier

Bayes Classifier	Naive Bayes Classifier
A Bayesian network is a graphical model that represents a set of variables and their conditional dependencies.	Naive Bayes classifier is a technique to assign class labels to the samples from the available set of labels.
There need not be any correlation between the features.	This method assumes each feature's value as independent and will not consider any correlation or relationship between the features

Bayes Classifier	Naive Bayes Classifier
It is very difficult to estimate the probability of all the features.	Probability estimation is done only to dependent features.

3.10. BAYESIAN BELIEF NETWORK

A Bayesian belief network describes the probability distribution governing a set of variables by specifying a set of conditional independence assumptions along with a set of conditional probabilities.

Bayesian belief networks provide an intermediate approach that is less constraining than the global assumption of conditional independence made by the naive Bayes classifier. This is more tractable than avoiding conditional independence assumptions all together. Bayesian networks are a type of probabilistic graphical model comprised of nodes and directed edges. Bayesian network models capture both conditionally dependent and conditionally independent relationships between random variables. Models can be prepared by experts or learned from data, then used for inference to estimate the probabilities for causal or subsequent events.

3.10.1. Conditional Independence

Bayesian belief network is a probabilistic graphical model where each node represents a random variable or group of random variables, and the links express probabilistic relationships between these variables. The prerequisites for developing a Bayesian Belief network are:

- ✓ Random Variables
- ✓ Conditional Relationships
- ✓ Probability Distributions

Consider a problem with three random variables: A, B, and C. A is dependent upon B, and C is dependent upon B.

- A is conditionally dependent upon B, e.g. $P(A|B)$
- C is conditionally dependent upon B, e.g. $P(C|B)$

It clearly infers that C and A have no effect on each other. The conditional independencies can be rewritten as follows:

- A is conditionally independent from C: $P(A|B, C)$
- C is conditionally independent from A: $P(C|B, A)$

The conditional dependence is stated in the presence of the conditional independence. That is, A is conditionally independent of C, or A is conditionally dependent upon B in the presence of C. Also, the conditional independence of A given C as the conditional dependence of A given B, as A is unaffected by C and can be calculated from A given B alone.

$$P(A/C, B) = P(A/B)$$

B is unaffected by A and C and has no parents; So simply state that the conditional independence of B from A and C as:

$$P(B, P(A/B), P(C/B)) \text{ or } P(B)$$

The joint probability of A and C given B or conditioned on B as the product of two conditional probabilities:

$$P(A, C/B) = P(A/B) * P(C/B)$$

The model summarizes the joint probability of $P(A, B, C)$ calculated as:

$$P(A, B, C) = P(A/B) * P(C/B) * P(B)$$

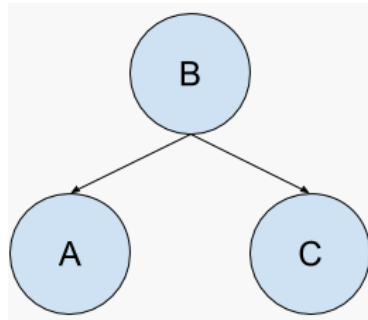
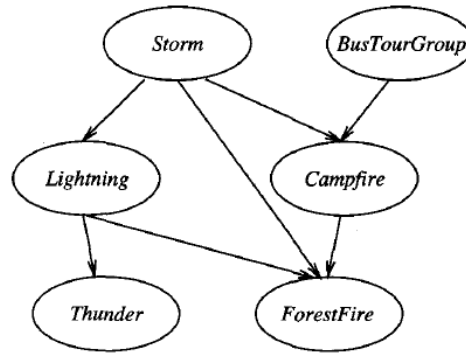


Fig. 3.4. Bayesian Belief Network for random variables A, B and C

The graph in Fig 3.4 is a Bayesian belief network of the random variables A, B and C. The variables are each assigned a node, and the conditional probabilities are given as directed connections between the nodes. The graph must not form a cycle.

Example**Fig. 3.5. Bayesian Belief Network**

- The network in Fig 3.5 represents a set of conditional independence assumptions.
- Each node is asserted to be conditionally independent of its non descendants, given its immediate parents.
- The conditional probability table specifies the conditional distribution for the variable given its immediate parents in the graph.
- Here C denotes campfire, B denotes BusTourGroup and S denotes Storm.
- The network nodes and arcs represent the assertion that Campfire is conditionally independent of its nondescendants Lightning and Thunder, given its immediate parents Storm and BusTourGroup.
- This means that once the values of the variables Storm and BusTourGroup is known, then the variables Lightning and Thunder provide no additional information about Campfire.

	S, B	S, ~B	~S, B	~S, ~B
C	0.4	0.1	0.8	0.2
~C	0.6	0.9	0.2	0.8

- This table provides only the conditional probabilities of Campfire given its parent variables Storm and BusTourGroup.
- The set of local conditional probability tables for all the variables, together with the set of conditional independence assumptions described by the network, describe the full joint probability distribution for the network.

3.10.2. Inferences from Bayesian Belief Network

- The probability distribution for the target variable, which specifies the probability that it will take on each of its possible values given the observed values of the other variables
- This inference step can be straightforward if values for all of the other variables in the network are known exactly.
- To infer the probability distribution for some variable given observed values for only a subset of the other variables is possible in belief networks.
- A Bayesian network can be used to compute the probability distribution for any subset of network variables given the values or distributions for any subset of the remaining variables.
- Exact inference of probabilities in general for an arbitrary Bayesian network is known to be NP-hard.
- Numerous methods have been proposed for probabilistic inference in Bayesian networks, including exact inference methods and approximate inference methods that sacrifice precision to gain efficiency.

3.10.3. Learning Bayesian Networks

- Learning from Bayesian Networks is very straight forward if:
 - ✓ Network structure is known in advance, or can be inferred from the training data.
 - ✓ All the network variables might be directly observable in each training example, or some might be unobservable.
- The learning problem becomes complicated is the network structure is given but only some of the variable values are observable in the training data.
- This problem is same as learning the weights for the hidden units in an artificial neural network, where the input and output node values are given but the hidden unit values are left unspecified by the training examples.
- The gradient ascent procedure can be used that learns the entries in the conditional probability tables.
- This gradient ascent procedure searches through a space of hypotheses that corresponds to the set of all possible entries for the conditional probability tables.

- The objective function that is maximized during gradient ascent is the probability $P(D|h)$ of the observed training data D given the hypothesis h .
- By definition, this corresponds to searching for the maximum likelihood hypothesis for the table entries.

3.10.4. Gradient Ascent Training of Bayesian Networks

The gradient ascent rule maximizes $P(D|h)$ by following the gradient of $\ln P(D|h)$ with respect to the parameters that define the conditional probability tables of the Bayesian network.

- Let w_{ijk} denote the conditional probability that the network variable Y_i will take on the value y_i , given that its immediate parents U_i take on the values given by u_{ik} . The gradient is calculated by

$$\frac{\partial \ln P_h(D)}{\partial w_{ij}} = \sum_{d \in D} \frac{P(Y_i = y_{ij}, U_i = u_{ik} | d)}{w_{ijk}}$$

- When some of the variables are unobservable for the training example d , the required probability can be calculated from the observed variables in d using standard Bayesian network inference.
- These required quantities are easily derived from the calculations performed during most Bayesian network inference, so learning can be performed at little additional cost whenever the
- Bayesian network is used for inference and new evidence is subsequently obtained.

$$\begin{aligned} \frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \frac{\partial}{\partial w_{ijk}} \prod_{d \in D} P_h(d) \\ &= \sum_{d \in D} \frac{\partial \ln P_h(D)}{\partial w_{ijk}} \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial \ln P_h(D)}{\partial w_{ijk}} \end{aligned}$$

After introducing parents,

$$\frac{\partial \ln P_h(D)}{\partial w_{ijk}} = \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d | y_{ij}, u_{ik}) P_h(y_{ij} | u_{ik}) P_h(u_{ik})$$

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}', u_{ik}') P_h(y_{ij}', u_{ik}') \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}', u_{ik}') P_h(y_{ij}'|u_{ik}') P_h(u_{ik}')$$

$w_{ijk} = P_h(y_{ij}'|u_{ik}')$ the only term in this sum for which and is nonzero is the term for which $j' = j$ and $i' = i$.

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) P_h(y_{ij}|u_{ik}) P_h(u_{ik}) \\ &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{\partial}{\partial w_{ijk}} P_h(d|y_{ij}, u_{ik}) w_{ijk} P_h(u_{ik}) \\ &= \sum_{d \in D} \frac{1}{P_h(d)} P_h(d|y_{ij}, u_{ik}) P_h(u_{ik})\end{aligned}$$

Applying Bayes theorem,

$$\begin{aligned}\frac{\partial \ln P_h(D)}{\partial w_{ijk}} &= \sum_{d \in D} \frac{1}{P_h(d)} \frac{P_h(y_{ij}, u_{ik}|d) P_h(d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d) P_h(u_{ik})}{P_h(y_{ij}, u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{P_h(y_{ij}|u_{ik})} \\ &= \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}\end{aligned}$$

Updating each value of w_{ijk} , is done by,

$$w_{ijk} \leftarrow w_{ijk} + \eta \sum_{d \in D} \frac{P_h(y_{ij}, u_{ik}|d)}{w_{ijk}}$$

3.10.5. Learning the Structure of Bayesian Networks

- Learning Bayesian networks when the network structure is not known in advance is also difficult.
- A Bayesian scoring metric for choosing among alternative networks and a heuristic search algorithm called K2 for learning network structure when the data is fully observable.

- K2 performs a greedy search that trades off network complexity for accuracy over the training data.
- Starting from an initial BN structure, the K2 algorithm searches the BN structure space and selects the structure maximizing the K2 metric.
- K2 algorithm considers the dependence on an initial topological ordering, search procedure, and score.
- By ranking the domain features based on their degree of separability, K2 algorithm establish an expert-free initial ordering that is both data-driven and classification-oriented.

3.11. EXPECTATION MAXIMIZATION (EM) ALGORITHM

The EM is used in maximum likelihood estimation where the problem involves two sets of random variables of which one, X , is observable and the other, Z , is hidden.

- Through EM algorithm it is easier to add extra variables that are not actually known (called hidden or latent variables) and then to maximise the function over those variables.
- This might seem to be making a problem much more complicated than it needs to be, but it turns out for many problems that it makes finding the solution significantly easier.
- Expectation maximization provides an iterative solution to maximum likelihood estimation with latent variables.
- Gaussian mixture models are an approach to density estimation (maximum likelihood is a method of density estimation) where the parameters of the distributions are fit using the expectation-maximization algorithm.
- Density estimation involves selecting a probability distribution function and the parameters of that distribution that best explain the joint probability distribution of the observed data.

- Maximum Likelihood Estimation involves treating the problem as an optimization or search problem, where we seek a set of parameters that results in the best fit for the joint probability of the data sample.
- A limitation of maximum likelihood estimation is that it assumes that the dataset is complete, or fully observed. This does not mean that the model has access to all data; instead, it assumes that all variables that are relevant to the problem are present.
- The EM algorithm is an iterative approach that cycle between two modes.
 - ✓ **First mode:** attempts to estimate the missing or latent variables, called the estimation-step or E-step.
 - ✓ **Second mode:** attempts to optimize the parameters of the model to best explain the data, called the maximization-step or M-step.
- **E-Step:** Estimate the missing variables in the dataset.
- **M-Step:** Maximize the parameters of the model in the presence of the data.

3.11.1. Estimating means of k Gaussians

- A **mixture model** is a model comprised of an unspecified combination of multiple probability distribution functions.

The Gaussian Mixture Model (GMM), is a mixture model that uses a combination of Gaussian (Normal) probability distributions and requires the estimation of the mean and standard deviation parameters for each.

- Consider the case where a dataset is comprised of many points that happen to be generated by two different processes.
- The points for each process have a Gaussian probability distribution, but the data is combined and the distributions are similar enough that it is not obvious to which distribution a given point may belong.
- The processes used to generate the data point represents a latent variable, e.g. process 0 and process 1. It influences the data but is not observable.
- As such, the EM algorithm is an appropriate approach to use to estimate the parameters of the distributions.

- In the EM algorithm, the estimation-step would estimate a value for the process latent variable for each data point, and the maximization step would optimize the parameters of the probability distributions in an attempt to best capture the density of the data.
- The process is repeated until a good set of latent values and a maximum likelihood is achieved that fits the data.

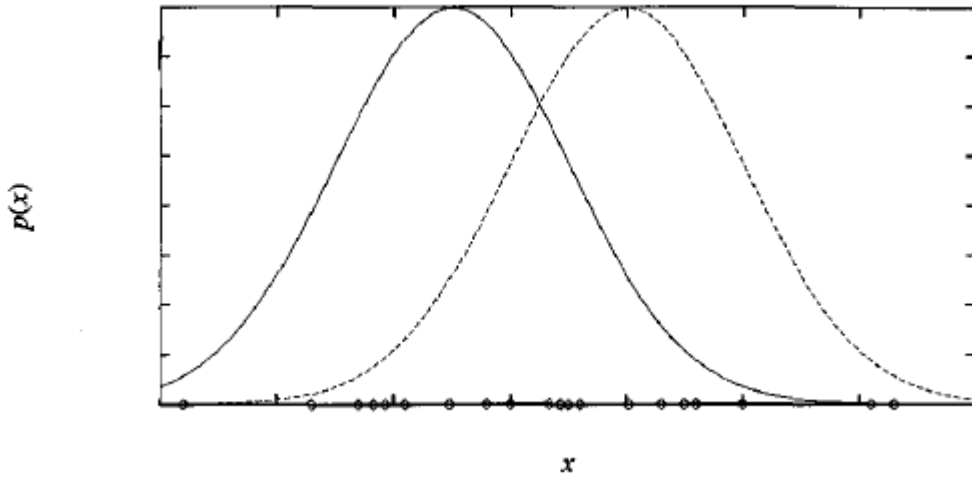


Fig. 3.6. GMM of two distributions

Maximum likelihood hypothesis for the mean of a single Normal distribution given the observed data instances $\langle x_1, x_2, \dots, x_m \rangle$ drawn from this single distribution.

$$\mu_{\text{ML}} = \underset{\mu}{\operatorname{argmin}} \sum_{i=1}^m (x_i - \mu)^2$$

The sum of squared errors is minimized by the sample mean.

$$\mu_{\text{ML}} = \frac{1}{m} \sum_{i=1}^m (x_i - \mu)^2$$

- The triple $\langle x_i, z_{i1}, z_{i2} \rangle$ where x_i is the observed value of the i^{th} instance and where z_{i1} and z_{i2} indicate which of the two Normal distributions was used to generate the value x_i .
- In particular, z_{ij} has the value 1 if x_i was created by the j^{th} Normal distribution and 0 otherwise.

- Here x_i is the observed variable in the description of the instance, and z_{i1} , z_{i2} are hidden variables.
- The EM algorithm first initializes the hypothesis to $h = (\mu_1, \mu_2)$, where μ_1 and μ_2 are arbitrary initial values.
- It then iteratively re-estimates h by repeating the following two steps until the procedure converges to a stationary value for h .

Step 1: Calculate the expected value $E[z_{ij}]$ of each hidden variable z_{ij} , assuming the current hypothesis $h = (\mu_1, \mu_2)$, holds

Step 2: Calculate a new maximum likelihood hypothesis $h' = (\mu_1', \mu_2')$, holds assuming the value taken on by each hidden variable z_{ij} is its expected value $E[z_{ij}]$ calculated in Step 1. Then replace the hypothesis $h = (\mu_1, \mu_2)$, by the replace the hypothesis $h' = (\mu_1', \mu_2')$ and iterate.

EM algorithm can be applied in the following situations

- It can be used to fill the missing data in a sample.
- It can be used as the basis of unsupervised learning of clusters.
- It can be used for the purpose of estimating the parameters of Hidden Markov Model (HMM).
- It can be used for discovering the values of latent variables.

Advantages of EM algorithm

- It is always guaranteed that likelihood will increase with each iteration.
- The E-step and M-step are often pretty easy for many problems in terms of implementation.
- Solutions to the M-steps often exist in the closed form.

Disadvantages of EM algorithm

- It has slow convergence.
- It makes convergence to the local optima only.
- It requires both the probabilities, forward and backward (numerical optimization requires only forward probability).

3.12. PROBABILITY LEARNING

Though there are many machine learning algorithms, it is very difficult to set quantitative bound on these measures, depending on attributes of the learning problem such as:

- the size or complexity of the hypothesis space considered by the learner
- the accuracy to which the target concept must be approximated
- the probability that the learner will output a successful hypothesis
- the manner in which training examples are presented to the learner

Learning Models

The models that dominate the field of machine learning are:

- **Probably Approximately Correct Model (PAC):** In this, the data comes from some fixed probability distribution over the instance space, labelled by an unknown target function. Assume training data is drawn from this distribution, this model is based on how much data do the model need to see so that if it performs well over it, then it can do well on new points also. The new points must also be drawn from the same distribution as the training data.
- **Mistake Bound (MB) model:** Algorithm A has mistake-bound M for learning class C if A makes at most M mistakes on any sequence that is consistent with a function in C.
- **Consistency model:** If class C is learnable in the Mistake Bound model by an algorithm A that uses hypotheses in C, then C is learnable in the consistency model. The consistency model has the problem that there was nothing in it about being able to predict well on new data.

The learning algorithm must generalize to some extent to arrive the

- **Sample complexity:** The number of training examples needed for the learner to converge to a successful hypothesis.
- **Computational complexity:** The computational effort for a learner to converge to a successful hypothesis.

- **Mistake bound:** The number of training examples can the learner allowed to misclassify before converging to a successful hypothesis.
- Given a continuous stream of examples where the learner predicts whether each one is a member of the concept or not and is then is told the correct answer, does the learner eventually converge to a correct concept and never make a mistake again.
- No limit on the number of examples required or computational demands, but must eventually learn the concept exactly, although do not need to explicitly recognize this convergence point.
- By simple enumeration, concepts from any known finite hypothesis space are learnable in the limit, although typically requires an exponential number of examples and time.
- Class of total recursive functions is not learnable in the limit.
- 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 are interested in characterizing the performance of various learners L using various hypothesis spaces H , when learning individual target concepts drawn from various classes C .
- Because we demand that L be general enough to learn any target concept from C regardless of the distribution of training examples, we will often be interested in worst-case analyses over all possible target concepts from C and all possible instance distributions D .

The probably approximately correct (PAC) learning is a framework for mathematical analysis of machine learning. The learner receives samples and must select a generalization function or the hypothesis from a certain class of possible functions. The goal is that, with high probability the selected function will have low generalization error. The learner must be able to learn the concept given any arbitrary approximation ratio, probability of success, or distribution of the samples.

3.12.1. Error of the hypothesis

- The true error of h is just the error rate expected when applying h to future instances drawn according to the probability distribution.

The true error 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 .

$$\text{error}_D(h) = \Pr_{x \in D} [c(x) \neq h(x)]$$

- 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).
- 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 .
- The error depends strongly on the unknown probability distribution.

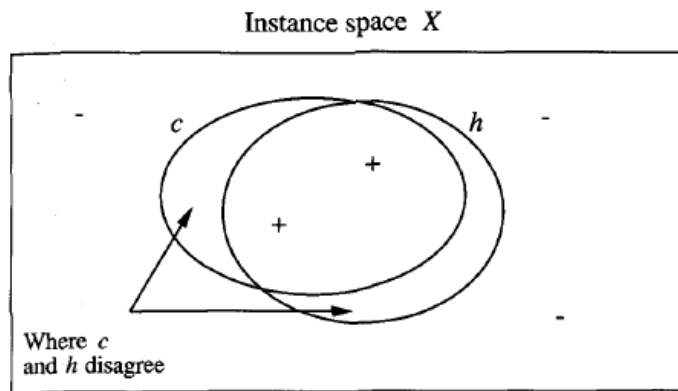


Fig. 3.7. Relation of target concept and hypothesis

- 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.
- The 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.
- The same h and c will have a much higher error if D happens to assign very high probability to instances for which h and c disagree.
- In the extreme, if V happens to assign zero probability to the instances for which $h(x) = c(x)$ then the error for the h will be 1.
- The error of h with respect to c is not directly observable to the learner. L can only observe the performance of h over the training examples, and it must choose its output hypothesis on this basis only.
- Training error refers to the fraction of training examples misclassified by h .

3.12.2. PAC Learnability

- PAC characterizes classes of target concepts that can be reliably learned from a reasonable number of randomly drawn training examples and a reasonable amount of computation.
- This characterization is done based on the number of training examples needed to learn a hypothesis h for which error is 0.
- This is infeasible because:

- ✓ There may be multiple hypotheses consistent with the provided training examples, and the learner cannot be certain to pick the one corresponding to the target concept.
- ✓ Given that the training examples are drawn randomly, there will always be some nonzero probability that the training examples encountered by the learner will be misleading.
- To overcome these difficulties:
 - ✓ The learner need not output a zero error hypothesis but with the error be bounded by some constant, ϵ , that can be made arbitrarily small.
 - ✓ It is not required that the learner succeed for every sequence of randomly drawn training examples but will require only that its probability of failure be bounded by some constant, that can be made arbitrarily small.
- The direct implication is that the learner is required to learn a probable 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 output a hypothesis h with $\text{error}(h) < \epsilon$, after observing a reasonable number of training examples and performing a reasonable amount of computation.
- PAC is concerned only with the computational resources required for learning, the number of training examples required.
- If L , a learner requires some minimum processing time per training example, then for C to be PAC-learnable by L , L must learn from a polynomial number of training examples.
- To show that some class C of target concepts is PAC-learnable, first show that each target concept in C can be learned from a polynomial number of training example sand then show that the processing time per example is also polynomial bounded.

3.13. SAMPLE COMPLEXITY FOR FINITE HYPOTHESIS SPACE

- 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.
- A general bound on the sample complexity for a very broad class of learners is called consistent learners.
- A learner is consistent if it outputs hypotheses that perfectly fit the training data, whenever possible.
- A learning algorithm is said to be consistent, if there is a hypothesis that fits the training data.
- A bound on the number of training examples required by any consistent learner, independent of the specific algorithm it uses to derive a consistent hypothesis is given by:

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

Where $VS_{H,D}$ to be the set of all hypotheses $h \in H$ that correctly classify the training examples D .

- The significance of the version space here is that every consistent learner outputs a hypothesis belonging to the version space, regardless of the instance space X , hypothesis space H , or training data D .
- The version space contains no unacceptable hypotheses.

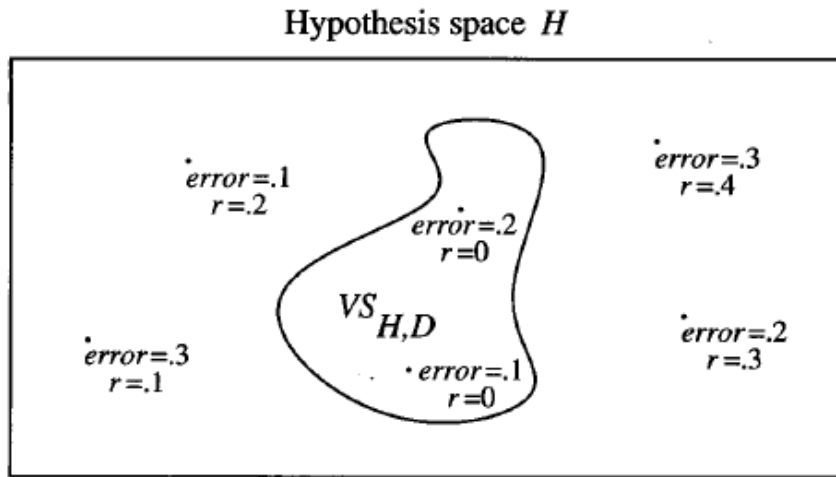


Fig. 3.8. Exhausting the version space

ϵ -exhausting the version space: If the hypothesis space H is finite and D is a sequence of $m \geq 1$ independently random samples of some target concept c , then for any $0 \leq \epsilon \leq 1$, the probability that the version space $VS_{H,D}$ is not ϵ -exhausted is less than or equal to

$$|H| e^{-\epsilon m}$$

Where e is the error.

- A subset of the version space has zero training error. But the real error returned may be higher even for the hypothesis that commits zero errors for training data.
- The number of training examples required to reduce this probability of failure below some desired level δ is given by:

$$|H| e^{-\epsilon m} \leq \delta$$

Where m is the number of training examples Rearranging,

$$m \geq \frac{1}{\epsilon} (\ln |H| + \ln(1/\delta))$$

- This equation provides a general bound on the number of training examples (m) sufficient for any consistent learner to successfully learn any target concept in H , for any desired values of ϵ and δ .
- m grows linearly in $1/\epsilon$ and logarithmically in $1/\delta$.

3.13.1. Agnostic Learning and Inconsistent Hypotheses

- Agnostic learning makes virtually no assumptions on the target function.
- This deals with how many training examples suffice to ensure (with probability $(1 - \delta)$) that every hypothesis in H having zero training error will have a true error of at most ϵ .
- If H does not contain the target concept c , then a zero-error hypothesis cannot always be found.
- The task of our learner is to output the hypothesis from H that has the minimum error over the training examples.
- A learner that makes no assumption that the target concept is representable by H and that simply finds the hypothesis with minimum training error, is called an agnostic learner, because it makes no prior commitment about whether or not $C \subseteq H$.
- Let $\text{error}_D(h)$ denote the training error of hypothesis h .
- Training error is defined as the fraction of the training examples in D that are misclassified by h .
- It is evident that the training error over the particular sample of training data D may differ from the true error over the entire probability distribution.
- h_{best} denote the hypothesis from H having lowest training error over the training examples.
- The Hoeffding bounds characterize the deviation between the true probability of some event and its observed frequency over m independent trials. This states that if the training error, $\text{error}_D(h)$ is measured over the set D containing m randomly drawn examples, then

$$\Pr[\text{error}_D(h) > \text{error}_D(h) + \epsilon] \leq |H| e^{-2m\epsilon^2}$$

- This gives us a bound on the probability that an arbitrarily chosen single hypothesis has a very misleading training error. To assure that the best

hypothesis found by L has an error bounded in this way, consider the probability that any one of the $|H|$ hypotheses could have a large error

$$\Pr[(\exists h \in H) (\text{error}_B(h) > \text{error}_D(h) + \epsilon)] \leq |H| e^{-2m\epsilon^2}$$

- When the probability δ , and ask how many examples m suffice to hold δ to some desired value

$$m \geq \frac{1}{2\epsilon^2} \left(\ln |H| + \ln \left(\frac{1}{\delta} \right) \right)$$

3.13.2. Conjunctions of Boolean Literals Are PAC-Learnable

The class C of target concepts can be described by conjunctions of Boolean literals. A boolean literal is any boolean variable or its negation. The sample complexity of learning conjunctions of up to n boolean literals.

$$m \geq \frac{1}{\epsilon} (n \ln 3 + \ln(1/\delta))$$

m grows linearly in the number of literals n , linearly in $1/\epsilon$, and logarithmically in $1/\delta$.

3.14. SAMPLE COMPLEXITY FOR INFINITE HYPOTHESIS SPACES

There are two limitations in expressing the complexity in term of H :

- ✓ It can lead to quite weak bounds.
- ✓ This is not applicable to infinite hypothesis

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. If a set of instances is not shattered by a hypothesis space, then there must be some concept (dichotomy) that can be defined over the instances, but that cannot be represented by the hypothesis space. The ability of H to shatter a set of instances is thus a measure of its capacity to represent target concepts defined over these instances.

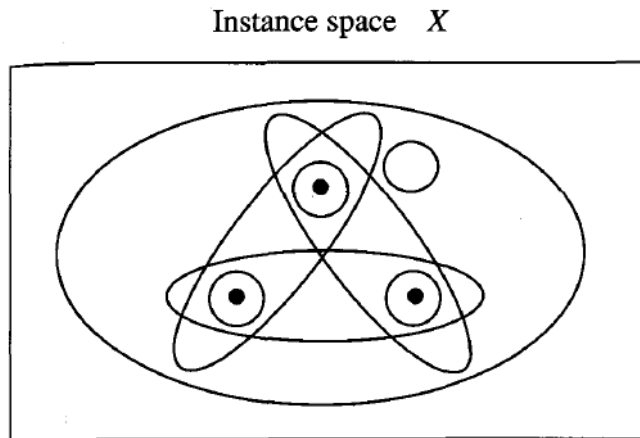


Fig. 3.9. A set of three instances shattered by eight hypotheses

A subset S of instances of a set X is shattered by a collection of function F if $\forall S' \subseteq S$ there is a function $f \in F$ such data:

$$f(x) = \begin{cases} 1 & x \in S' \\ 0 & x \in S - S' \end{cases}$$

Shattering sets of points

- A configuration of N points on the plane is just any placement of N points.
- In order to have a VC dimension of at least N , a classifier must be able to shatter a single configuration of N points.
- In order to shatter a configuration of points, the classifier must be able to, for every possible assignment of positive and negative for the points, perfectly partition the plane such that the positive points are separated from the negative points.
- For a configuration of N points, there are 2^N possible assignments of positive or negative, so the classifier must be able to properly separate the points in each of these.

3.14.1. Vapnik-Chervonenkis (VC) Dimension

- This is derived from computational learning theory that is used to formally quantify the power of a classification algorithm.

- The VC dimension of a classifier is defined by Vapnik and Chervonenkis to be the cardinality (size) of the largest set of points that the classification algorithm can shatter.

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 $VC(H) = \infty$

For any finite H , $VC(H) \leq \log_2 |H|$

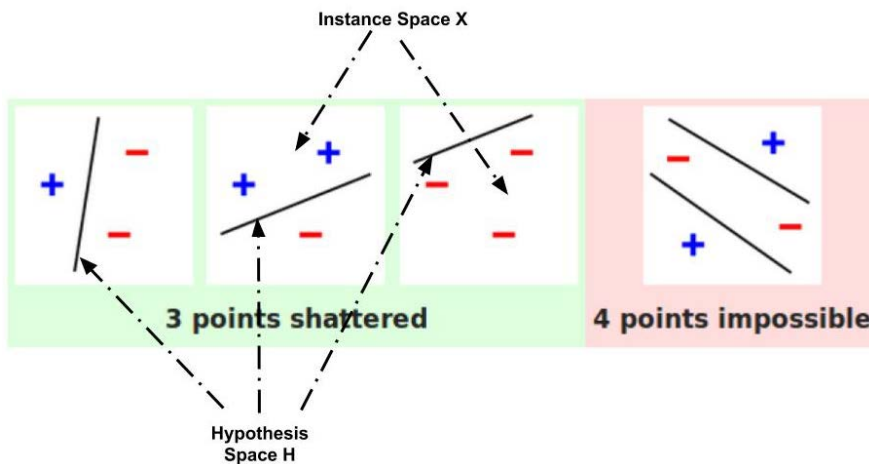


Fig. 3.10. a) $VC(H)$ depicting the X and H for shattering



Fig 3.10 b) Set of points that can be shattered



Fig 3.10 c) Set of points that cannot be shattered

Upper bound on sample complexity

Earlier we considered the question “How many randomly drawn training examples suffice to probably approximately learn any target concept t in C ?”(i.e., how many

examples suffice to ϵ -exhaust the version space with probability $(1 - \delta)$?. Using $VC(H)$ as a measure for the complexity of H , it is possible to derive an alternative answer to this question. This new bound (see Blumer et al. 1989) is

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

Then with probability $(1 - \delta)$,

$$\text{error}_D(h) \leq \epsilon$$

Lower bound on complexity

Consider any concept class C such that $VC(C) \geq 2$, any learner L , and any $0 < \epsilon < \frac{1}{\delta}$ and $0 < \delta < \frac{1}{100}$. Then there exists a distribution D and target concept in C such that L observes fewer examples than

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

Then with probability at least δ , L outputs a hypothesis h having $\text{error}_D(h) \leq \epsilon$

Example:

Choose 4-point set, which can be shattered in all possible ways. Given such 4 points, we assign them the $\{+, -\}$ labels, in all possible ways. For each labelling it must exist a rectangle which produces such assignment, i.e. such classification.

Classifier: Points inside the rectangle is positive and outside is negative examples

Given 4 points (linearly independent),

- All points are "+" \Rightarrow use a rectangle that includes them
- All points are "-" \Rightarrow use a empty rectangle
- 3 points "-" and 1 "+" \Rightarrow use a rectangle centered on the "+" points
- 3 points "+" and one "-" \Rightarrow we can always find a rectangle which exclude the "-" points
- 2 points "+" and 2 points "-" \Rightarrow we can define a rectangle which includes the 2 "+" and excludes the 2 "-".

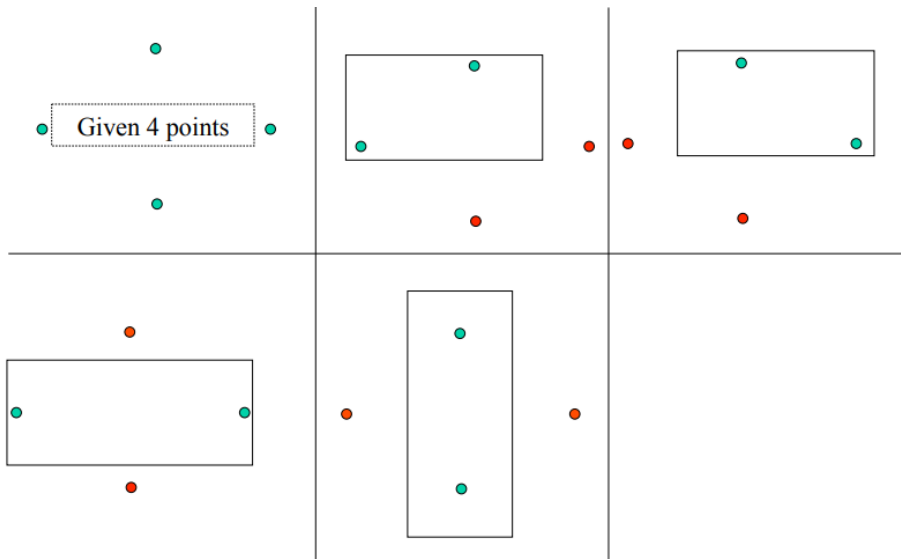


Fig. 3.11. Illustration of the example

- For any 5-point set, we can define a rectangle which has the most extern points as vertices.
- If we assign to such vertices the “+” label and to the internal point the “-” label, there will not be any rectangle which reproduces such assignment.

3.14.2. VC Dimension for Neural Networks

- The VC dimension of layered acyclic network is decided based on the structure of the network and the VC dimension of its individual units.
- This VC dimension can then be used to bound the number of training examples sufficient to probably approximately correctly learn a feed forward network to desired values of ϵ and δ .
- Neural networks are considered as Direct Acyclic Graphs. A directed acyclic graph is one for which the edges have a direction, and in which there are no directed cycles.
- A layered graph is one whose nodes can be partitioned into layers such that all directed edges from nodes at layer n go to nodes at layer $n + 1$.
- To generalise the VC dimension, consider the neural network with n number of nodes and the neural network with only one output node is represented using the direct acyclic graph, G .

- Let N_i be the internal units of G with almost r inputs and implements Boolean function that takes only two values $\{0, 1\}$ for some function class C .
- The G -composition of C is the class of all functions that can be implemented by the network G . The individual units of G follow the function class C .
- In other words, The G - composition of C is the hypothesis space of the network G .

Let G be the layered directed acyclic graph (neural networks) with n input nodes and $s \geq 2$ internal nodes, each having almost r inputs. Let C be a concept class of VC dimension d and C_G be the G -composition of C that are the set of functions represented by G . Then the VC dimension of G is given by: $VC(CG) \leq 2ds \log(e)$ where e is the base of the natural logarithm.

The complexity of the network G grows linearly with the VC dimension d of its individual units and log times linear in s which is the number of threshold units in the network. To bound the VC dimension of acyclic layered networks containing s perceptrons each with r inputs:

$$VC(C_G^{\text{perceptrons}}) \leq 2(r+1)s \log(e)$$

3.15. MISTAKE BOUND (MB) MODEL

In the MB model, learning is in stages. In each stage:

1. The learner gets an unlabeled example.
2. The learner predicts its classification.
3. The learner is told the correct label.

The goal of Mistake bound model is the total number of mistakes that the model is allowed to make.

Algorithm A has mistake-bound M for learning class C if A makes at most M mistakes on any sequence that is consistent with a function in C .

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

3.15.1. Mistakes in Halving Algorithm

- **Halving Algorithm:** This predicts using majority vote over all concepts in C consistent with past data
- Each mistake of halving algorithm cuts down the number of available concepts in half (or more). So, it makes at most $\log|C|$ mistakes.
- The steps in halving algorithm are:
 1. Initialize V to C . (V is called the version space.)
 2. Given example x , predict according to the majority of the concepts in V .
 3. Remove from V all the concepts in C that predicted wrongly.
 4. Return to step 2.
- Since with each mistake at least half of the version space V is removed, the number of mistakes is bounded by $\log|C|$.
- Storing V and predicting according to the majority of the concepts in V is likely to be hard.
- 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 mistakes possible before the version space contains just one member is $\log_2|H|$.

3.15.2. Mistake Bound for the FIND-S Algorithm

Consider the learning task where the:

- training instances are represented by n Boolean features
- target concept is conjunction of up to n Boolean (negated) literals

FIND-S for Boolean literals

Initialize h to the most specific hypothesis

$$x_1 \wedge \neg x_1 \wedge x_2 \wedge \neg x_2 \wedge \dots \wedge x_n \wedge \neg x_n$$

for each positive training instance x

remove from h any literal that is not satisfied by x

output hypothesis h

- The Find-S converges to a hypothesis that does not make any errors.

- The Find-S begins its course by choosing the most specific hypothesis and then incrementally generalizes the hypothesis to train more positive examples.
- FIND-S can never mistakenly classify a negative example as positive.
- The reason is that its current hypothesis h is always at least as specific as the target concept. So count the number of mistakes it will make misclassifying truly positive examples as negative.
- The total number of mistakes are bound can be at most $n + 1$.

3.15.3. Optimal mistake bounds

- Optimal mistake bound is the lowest worst-case mistake bound over all possible learning algorithms.
- Let $M_A(C)$ be the maximum number of mistakes done by the training algorithm A to learn a target concept C .

$$M_A(C) \equiv \max_{c \in C} M_A(C)$$

The optimal mistake bound for C is the minimum overall possible learning algorithms A of $M_A(C)$. $Opt(C) \equiv \min_{A \in \text{learning algorithms}} M_A(C)$

- In case of Halving algorithm, this becomes

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

- For Find-S algorithm the optimal mistake bound is:

$$M_{\text{Find-S}}(C) = n + 1$$

- The comparative relationship among the optimal mistake bound for C for various algorithms is given by:

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

3.15.4. Weighted Majority Algorithm

- The weighted majority algorithm is an ensemble method: way to combine the advice from several other algorithms or hypotheses called as experts.
- This is a generalization of halving algorithm.

- The predictions made by weighted majority is by taking a weighted vote among a pool of prediction algorithms and learns by altering the weight associated with each prediction algorithm.
- These prediction algorithms can be taken to be the alternative hypotheses in H , or they can be taken to be alternative learning algorithms that themselves vary over time.
- Two important properties of weighted majority algorithm are:
 - ✓ it can 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.
 - ✓ The mistakes can be bounded by the terms of the number of mistakes committed by the best of the pool of prediction algorithms.
- The weighted majority algorithm initially assigns an equal weight of 1 to all experts.
- On each round, it ask all the experts for their predictions, and sum up the weights for each of the two possible predictions, "positive" or "negative".
- Then it outputs the prediction that has the higher weight.

Weighted Majority Algorithm

```
//  $a_i$  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  $(x, c(x))$ 
  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  $q_1 < q_0$  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  $w_i \leftarrow \beta w_i$ 
```

- Whenever a prediction algorithm misclassifies a new training example its weight is decreased by multiplying it by some number β , where $0 < \beta < 1$
- When $\beta = 0$, then this algorithm is same as halving algorithm.
- When β is given some value, then algorithm will never be eliminated completely.
- If an algorithm misclassifies a training example, it will simply receive a smaller vote in the future.

The number of mistakes over the D (sequence of training examples) made by weighted majority algorithm when $\beta = 0.5$ is almost $2.4(k + \log_2 n)$. This is the relative mistake bound for weighted majority algorithm.

UNIT

4

INSTANT BASED LEARNING

4.1. INTRODUCTION TO INSTANCE BASED LEARNING

Memory based learning or Instance based learning is a supervised classification learning algorithm that performs operation after comparing the current instances with the previously trained instances, which have been stored in memory. It creates assumption from the training data instances. The time complexity of Instance based learning algorithm depends upon the size of training data.

The stored training instances themselves represent the knowledge. Each time a new query instance is encountered, its relationship to the previously stored examples is examined in order to assign a target function value for the new instance. This is also known as **lazy learning**.

The distinguishing feature of instance-based approach is that it can construct a different approximation to the target function for each distinct query instance that must be classified.

Differences between instance based and model based learning

Instance based learning	Model based learning
No parameter tuning.	Parameters can be tuned.
The system is normally hard coded with priors in form of fixed weights	These parameters with optimal settings are supposed to model the problem as accurately as possible thus learning is not simply about memorization but rather more about searching for those optimal parameters.
Instance-based learning is just about storing the training data instances. Though the training data itself can be	The model infers which actions to take in that environment that lead to desired outcomes in order to achieve a given goal.

preprocessed in many ways and stored in memory.	
The cost of classifying new instances can be high.	The cost of classifying new instances is relatively low.

Advantages of instance based learning:

- Lazy approach is suitable when the examples are not all available from the beginning but are collected on-line.
- Lazy learning does not suffer from data interference. That is, acquiring examples about an operating regime does not degrade modelling performance about others.

Disadvantages of Instance based algorithms:

- The cost of classifying new instances can be high. This is because nearly all computation takes place at classification time rather than when the training examples are first encountered. Therefore, techniques for efficiently indexing training examples are a significant practical issue in reducing the computation required at query time.
- They consider all attributes of the instances when attempting to retrieve similar training examples from memory. If the target concept depends on only a few of the many available attributes, then the instances that are truly most "similar" may well be a large distance apart.

4.2. K-NEAREST NEIGHBOUR (K-NN)

K-NN is a non parametric, unsupervised learning algorithm, that tries to learn a function that allows to make predictions with some new unlabeled data. An unsupervised learning strategy tries to learn the basic structure of the data by gaining more insight into the data. The KNN algorithm assumes that similar things exist in close proximity. It works on the assumption that similar things are generally near to each other.

The main advantages of k-NN are:

- ✓ Ease to interpret the output
- ✓ Less calculation time
- ✓ More Predictive Power

- ✓ Highly effective inductive inference method for many practical problems.
- ✓ Smooth out the impact of isolated noisy training examples.
- K-NN or k-NN algorithm assumes the similarity between the new case/data and available cases and put the new case into the category that is most similar to the available categories.
- K-NN algorithm stores all the available data and classifies a new data point based on the similarity. This means when new data appears then it can be easily classified into a well suite category by using K- NN algorithm.
- K-NN algorithm can be used for Regression as well as for Classification but mostly it is used for the Classification problems.
- K-NN is a **non-parametric algorithm**, which means it does not make any assumption on underlying data.
- It is also called a **lazy learner algorithm** because it does not learn from the training set immediately instead it stores the dataset and at the time of classification, it performs an action on the dataset.
- KNN algorithm at the training phase just stores the dataset and when it gets new data, then it classifies that data into a category that is much similar to the new data.
- An arbitrary instance x be described by the feature vector $\langle a_1(x), a_2(x), \dots a_n(x) \rangle$

Where $a_r(x)$ denotes the value of the r^{th} attribute of instance x .

- Then the distance between two instances x_i and x_j is defined to be $d(x_i, x_j)$, where

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

- The target function of k-NN may be real or discrete valued.

Algorithm for K-Nearest Neighbours

Training algorithm:

For each training example $(x, f(x))$ add the example to the list training_examples

Classification algorithm:

Given a query instance x_q to be classified,

Let $x_1, x_2, x_3, \dots, x_k$ denotes the k instances from training_examples that are nearest

to x_q

return

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

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

- In the above algorithm, the instances are points in a two-dimensional space and where the target function is boolean valued.
- The positive and negative training examples are shown by "+" and "-" respectively.

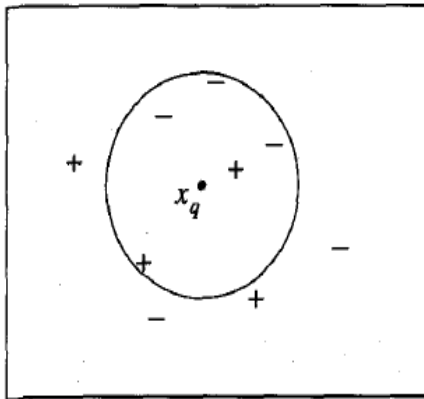


Fig. 4.1. a) Examples for k-NN

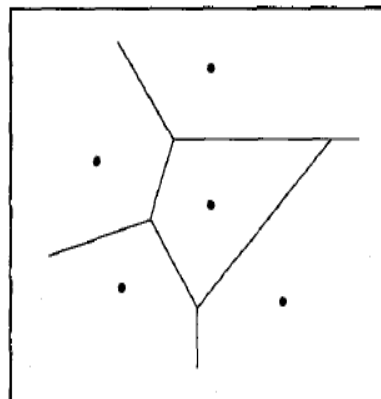


Fig 4.1b) Decision surface for Fig 4.1a)

- The fig 4.1 shows a set of positive and negative training examples is shown on the left, along with a query instance x_q that is to be classified.
- On the right is the decision surface induced by the 1-NN ($k = 1$) for a typical set of training examples.
- The convex polygon surrounding each training example indicates the region of instance space closest to that point.
- The decision surface is a combination of convex polyhedra surrounding each of the training examples.

- For every training example, the polyhedron indicates the set of query points whose classification will be completely determined by that training example.
- Query points outside the polyhedron are closer to some other training example. This kind of diagram is called the Voronoi *diagram* of the set of training examples.

The partitioning of a plane with n points into convex polygons such that each polygon contains exactly one generating point and every point in a given polygon is closer to its generating point than to any other. This is called Voronoi diagram.

The k-NN never forms an explicit general hypothesis f regarding the target function.

- It simply computes the classification of each new query instance as and when needed.
- The given algorithm computes discrete valued target. To compute real valued target, change the last line of algorithm as:

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

4.2.1. Distance Weighted Nearest Neighbour Algorithm

Another variation of k-NN is to weight the contribution of each of the k neighbors according to their distance to the query point x_q , giving greater weight to closer neighbors. The last line of the algorithm can be modified as:

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

Where,

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

To computer the instances for real-valued target functions in a similar fashion, replacing the final line of the algorithm in this case by:

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

- All the above variants of the k-NN consider only the k nearest neighbors to classify the query point.
- Once new distance weighting is added, there is really no harm in allowing all training examples to have an influence on the classification of the x_q , because very distant examples will have very little effect on $f(x_q)$.
- The disadvantage of considering all examples is that the classifier will run more slowly.
- If all training examples are considered when classifying a new query instance, then the algorithm follows a **global method**.
- Since only the nearest training examples are considered, K-NN follows local method.

Limitations of K-NN:

- The distance between neighbors will be dominated by the large number of irrelevant attributes.
- Algorithm delays all processing until a new query is received, significant computation can be required to process each new query. Hence memory indexing is needed.
- **Does not work well with large dataset:** In large datasets, the cost of calculating the distance between the new point and each existing points is huge which degrades the performance of the algorithm.
- **Does not work well with high dimensions:** The KNN algorithm doesn't work well with high dimensional data because with large number of dimensions, it becomes difficult for the algorithm to calculate the distance in each dimension.
- **Need feature scaling:** Standardization and normalization is needed before applying KNN algorithm to any dataset.
- Sensitive to noisy data, missing values and outliers

Example:

The steps to classify the target using k-NN are:

1. Determine the parameter k (number of nearest neighbors).
2. Calculate the distance between the query instance and all the training samples.
3. Sort the distance and find the nearest neighbors based on the k^{th} minimum distance.
4. Collect the category of nearest neighbors
5. Use majority of the category of the nearest neighbors as the prediction value of the query instance.

Find the quality of the paper as good or bad based on the features = {acid durability, strength}.

X1 = Acid durability (seconds)	X2 = Strength (kg/ square meter)	Y = classification result
7	7	Bad
7	4	Bad
3	4	Good
1	4	good

Test whether the paper with features = {3, 7} as good/ bad using k-NN.

1. Let us assume $k = 3$
2. Calculate the distance between the query instance and all the training samples.

$$X_q = \{3, 7\}$$

X1 = Acid durability (seconds)	X2 = Strength (kg/ square meter)	Squared distance to x_q
7	7	$(7-3)^2 + (7-7)^2 = 16$
7	4	25
3	4	9
1	4	13

3. Sort the distance and find the nearest neighbors based on minimum distance. If the rank or position after sorting is greater than k value, then exclude the

example from further processing. Hence the example in second row is not considered since its rank > 3.

X1 = Acid durability (seconds)	X2 = Strength (kg/ square meter)	Squared distance to x_q	Sort the minimum distance	Is it included in 3-NN
7	7	$(7-3)^2 + (7-7)^2 = 16$	3	Yes
7	4	25	4	No
3	4	9	1	Yes
1	4	13	2	Yes

4. Collect the category of nearest neighbors

X1 = Acid durability (seconds)	X2 = Strength (kg/ square meter)	Squared distance to x_q	Sort the minimum distance	Is it included in 3-NN	Y-Category of Nearest neighbor
7	7	$(7-3)^2 + (7-7)^2 = 16$	3	Yes	Bad
7	4	25	4	No	-
3	4	9	1	Yes	Good
1	4	13	2	Yes	Good

5. Use majority of the category of the nearest neighbors as the prediction value of the query instance.

From the table, we have two results as good and one as bad. Hence the paper represented by x_q ($X1 = 3$, $X2 = 7$) can be included in Good category.

4.3. LOCALLY WEIGHTED REGRESSION

Locally weighted regression constructs an explicit approximation to f over a local region surrounding x_q . Locally weight edregression uses nearby or distance-weighted training examples to form this local approximation to f .

- It is a non-parametric regression method that combine multiple regression models in a k-nearest-neighbor-based meta-model.
- They address situations in which the classical procedures do not perform well or cannot be effectively applied.
- This combines the simplicity of linear least squares regression with the flexibility of nonlinear regression.
- It does this by fitting simple models to localized subsets of the data to build up a function that describes the variation in the data, point by point.
- The dissemination of the name "locally weighted regression" infers loads of information:
 - ✓ Local - the function is approximated based only on data near the query point, weighted because the contribution of each training example is weighted by its distance from the query point
 - ✓ Regression - the term used widely in the statistical learning community for the problem of approximating real-valued functions.
- Locally weighted regression uses nearby or distance-weighted training examples to form this local approximation to f .
- Given a new query instance x_q , the general approach in locally weighted regression is to construct an approximation \hat{f} that fits the training examples in the neighborhood surrounding x_q .
- This approximation is then used to calculate the value $\hat{f}(x_q)$, which is output as the estimated target value for the query instance.
- The description of \hat{f} may then be deleted, because a different local approximation will be calculated for each distinct query instance.

4.3.1. Locally Weighted Linear Regression

The target function of the locally weighted regression is of the form:

$$\hat{f}(x) = \omega_0 + \omega_1 a_1(x) + \dots + \omega_n a_n(x)$$

Gradient descent uses coefficients to minimize the squared error function over training examples D .

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

This leads to the gradient descent rule:

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

This rule has to be modified to achieve local error at query point x_q . Three ways of modifications are currently in practise:

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

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

3. Combine both methods.

The above mention strategies minimize the errors. These do not involve complex functional transformations since:

- Cost of fitting more complex functions for each query instance is prohibitively high,
- The simple approximations model the target function quite well over a sufficiently small sub region of the instance space.

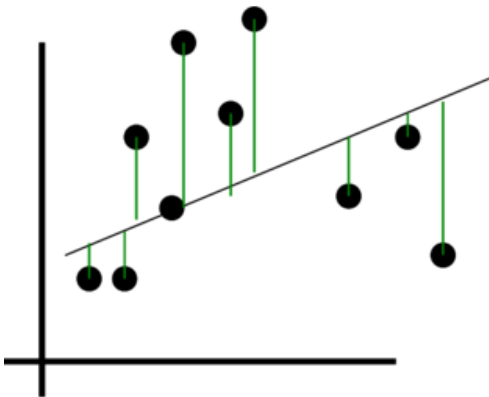


Fig. 4.2. a) Line fitting Linear regression

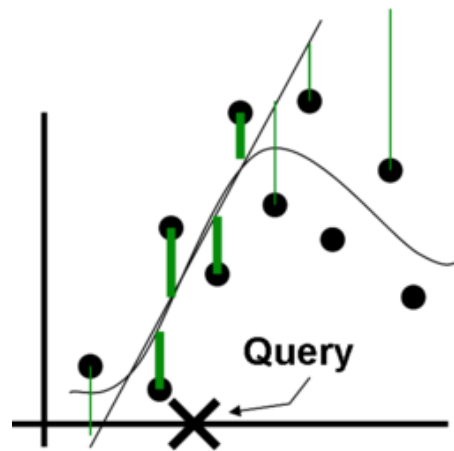


Fig 4.2 b) Locally weighted linear regression

- The aim of locally weighted linear regression is that the Y values of neighbouring X values are the best indicators of what the Y value should be at a given X value.
- Two factors must be taken into consideration:
 - ✓ decide on the number of X values to be considered
 - ✓ Assigning weights to Y values that correspond to neighbouring X values
- Fit a line to a bivariate scatter of points in a series of iterations using the following procedure:
 - ✓ Decides how smooth the fitted relationship should be by deciding on the number of adjacent points (q) to be used in the estimation procedure; the greater the number, the smoother will be the fitted line.
 - ✓ Each point to be used is then given a neighbourhood weight in relation to its distance from the focal point (x_i).
 - ✓ A simple linear regression is then fitted to each of the q values for a given focal point by weighted least squares, and an estimate i is computed for the focal point.
 - ✓ The procedure is repeated until all n points have estimated (fitted) Y values.
 - ✓ Ordinary residuals are then calculated from the difference between observed and fitted values, and robustness weights calculated.
 - ✓ A further weighted least squares regression is then run using the product of the neighbourhood and robustness weights.
 - ✓ The procedure is repeated until there is little or no change in the final fitted line.

4.4. RADIAL BASIS FUNCTIONS (RBF)

Radial basis functions are means to approximate multivariable functions by linear combinations of terms based on a single univariate function (the radial basis function).

- These functions are radialised so that in can be used in more than one dimension.
- They are usually applied to approximate functions or data which are only known at a finite number of points so that then evaluations of the approximating function can take place often and efficiently
- They uses a series of basis functions that are symmetric and centered at each sampling point.
- The main feature of these functions is that their response decreases, or increases, monotonically with distance from a central point.
- They transforms the input signal into another form, which can be then feed into the network to get linear separability.
- Radial basis function networks provide a global approximation to the target function, represented by a linear combination of many local kernel functions.
- The value for any given kernel function is non-negligible only when the input x falls into the region defined by its particular center and width.
- Thus, the network can be viewed as a smooth linear combination of many local approximations to the target function.

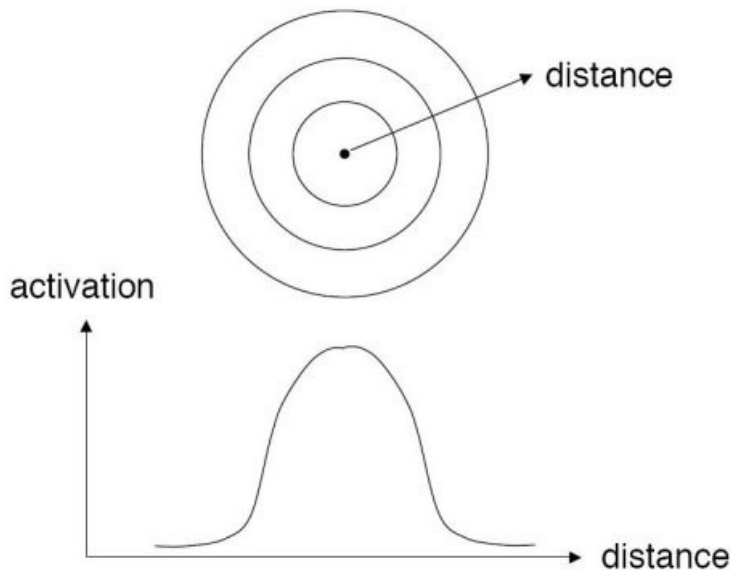


Fig. 4.3.Radial distance and radial basis function

- The general function is of the form

$$\hat{f}(x) = \omega_0 + \sum_{u=1}^k \omega_u K_u(d(x_u, x))$$

x_u - instance from X

$K_u(d(x_u, x))$ - kernel function

$d(x_u, x)$ - distance between two points x_u and x

- The kernel function decreases as the distance between two points increases.
- The kernel function is localised to a region nearby the point x_u .
- The kernel function is given as:

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

- This is a Gaussian kernel with σ_u^2 as the variance.
- The RBF networks are trained in two stages:
 - ✓ Determine the value of k , the number of hidden units. Each hidden unit u is defined by choosing the values x_u and σ_u^2 accordance with the kernel function.
 - ✓ Maximize the weights w_u to fit the network to the training data using global error criterion (E)

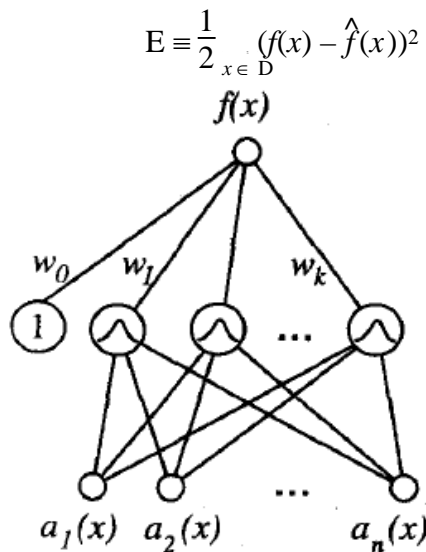


Fig. 4.4. Radial Basis Network

- Each hidden unit in the radial basis function shown in Fig 4.4 produces an activation determined by a Gaussian function centered at some instance x_u .
- Its activation will be close to zero unless the input x is near x_u .
- The output unit produces a linear combination of the hidden unit activations.
- Although the network shown here has just one output, multiple output units can also be included.
- Choosing the number of hidden units or proper kernel function is crucial.
- Two approaches are widely adopted for selection:
 - ✓ Allocate a Gaussian kernel function for each training example $(x_i, f(x_i))$ centered at x_i . Each kernel is assigned same width σ^2 . The RBF network learns the global approximation to the target function in which the training example can influence the value of \hat{f} only in the neighbourhood of x_i . The main advantage of this approach is that it allows RBF network to fit the training data exactly.
 - ✓ Choose set of kernel functions that is smaller than the number of training examples. These functions can be distributed at centres with uniform spacing. Non uniform distribution of centers may also be considered.

4.5. CASE BASED REASONING (CBR)

- In case-based reasoning, the training examples, the cases, are stored and accessed to solve a new problem.
- To get a prediction for a new example, those cases that are similar, or close to, the new example are used to predict the value of the target features of the new example.

Case-Based Reasoning (CBR) solves new problems by adapting previously successful solutions to similar problems.

- CBR draws attention because of the following features:
 - ✓ CBR does not require an explicit domain model and so elicitation becomes a task of gathering case histories.

- ✓ Implementation is reduced to identifying significant features that describe a case, an easier task than creating an explicit model.
- ✓ CBR systems can learn by acquiring new knowledge as cases.
- ✓ This and the application of database techniques makes the maintenance of large volumes of information easier.

Similarities and differences between other learning methods

Instance-based methods have three key properties.

- ✓ They are lazy learning methods in that they defer the decision of how to generalize beyond the training data until a new query instance is observed.
- ✓ They classify new query instances by analysing similar instances while ignoring instances that are very different from the query.
- ✓ They represent instances as real-valued points in an n-dimensional Euclidean space.

The case based reasoning approaches follow first two properties but differ in third property. In CBR, instances are typically represented using more rich symbolic descriptions, and the methods used to retrieve similar instances are correspondingly more elaborate. CBR has been applied to problems such as conceptual design of mechanical devices based on a stored library of previous designs, reasoning about new legal cases based on previous rulings, and solving planning and scheduling problems by reusing and combining portions of previous solutions to similar problems.

4.5.1. Problem solving in CBR

CBR is described by reasoning by remembering: previously solved problems (cases) are used to suggest solutions for novel but similar problems. The four assumptions about the world around us that represent the basis of the CBR approach:

1. Regularity: the same actions executed under the same conditions will tend to have the same or similar outcomes.
2. Typicality: experiences tend to repeat themselves.
3. Consistency: small changes in the situation require merely small changes in the interpretation and in the solution.

4. Adaptability: when things repeat, the differences tend to be small, and the small differences are easy to compensate for.

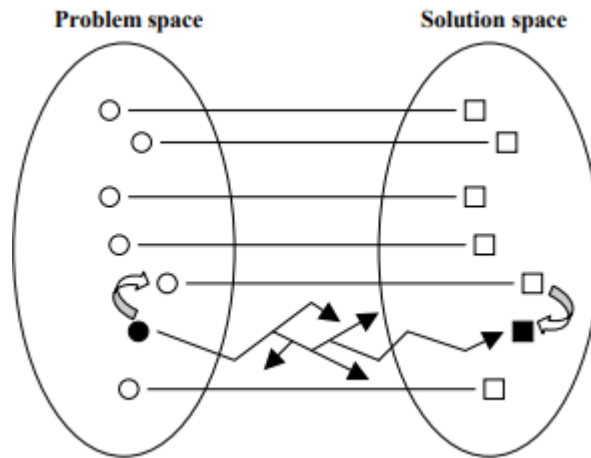


Fig. 4.5. Problem Solving using CBR

- Once the currently encountered problem is described in terms of previously solved problems, the most similar solved problem can be found.
- The solution to this problem might be directly applicable to the current problem but, usually, some adaptation is required.
- The adaptation will be based upon the differences between the current problem and the problem that served to retrieve the solution.
- Once the solution to the new problem has been verified as correct, a link between it and the description of the problem will be created and this additional problem solution pair (case) will be used to solve new problems in the future.
- Adding of new cases will improve results of a CBR system by filling the problem space more densely.

4.5.2. CBR Working Cycle

The CBR working cycle can be described best in terms of four processing stages:

1. Case retrieval: after the problem situation has been assessed, the best matching case is searched in the case base and an approximate solution is retrieved.
2. Case adaptation: the retrieved solution is adapted to fit better the new problem.

3. Solution evaluation: the adapted solution can be evaluated either before the solution is applied to the problem or after the solution has been applied. In any case, if the accomplished result is not satisfactory, the retrieved solution must be adapted again or more cases should be retrieved.
4. Case-base updating: If the solution was verified as correct, the new case may be added to the case base.

A variant of the above mentioned cycle is given as:

1. RETRIEVE the most similar case(s);
2. REUSE the case(s) to attempt to solve the current problem;
3. REVISE the proposed solution if necessary;
4. RETAIN the new solution as a part of a new case.

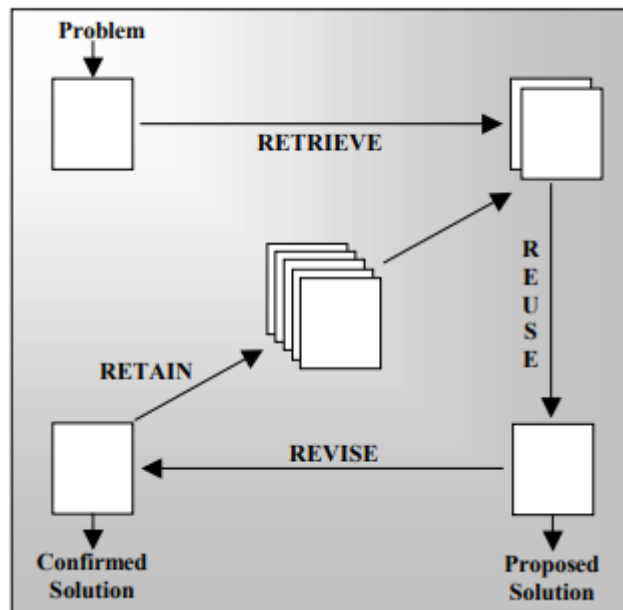


Fig. 4.6. CBR Working Cycle

- A new problem is matched against the cases furnishing the case base and one or more similar cases are retrieved.
- A solution suggested by the matching cases is then reused.

- Unless the retrieved case is a close match, the solution will probably have to be revised (adapted) and tested (evaluated) for success, producing a new case that can be retained ensuing, consequently, update of the case base.

4.5.3. Case representation

A case is a contextualized piece of knowledge representing an experience. It contains the past lesson that is the content of the case and the context in which the lesson can be used. A case contains:

- **Problem description:** depicts the state of the world when the case occurred
- **Problem solution:** states the derived solution to that problem
- **Outcome:** describes the state of the world after the case occurred.
- The problem description essentially contains as much data about the problem and its context as necessary for an efficient and accurate case retrieval.
- The problem solution can be either atomic or compound.
- Atomic solutions are typical for CBR systems used for diagnosis or for classification in general.
- Compound solutions can be found for instance in CBR systems utilised for planning or design.
- A compound solution may be composed of a sequence of actions, an arrangement of components, etc.

Advantages of Case based learning

The benefits of CBR as a lazy problem-solving method are:

- **Ease of knowledge elicitation:** Lazy methods use the available cases or problem instances instead of rules that are difficult to extract. So, classical knowledge engineering is replaced by case acquisition and structuring.
- **Absence of problem-solving bias:** The cases are stored in a raw form, they can be used for multiple problem-solving purposes. This in contrast to eager methods, which can be used merely for the purpose for which the knowledge has already been compiled or preprocessed.
- **Incremental learning:** A CBR system can be put into operation with a minimal set of solved cases serving as case base. The case base will be filled with new

cases as the system is used, increasing the system's problem-solving ability. When new cases are augmented, new indexes and clusters/categories can be created and the existing ones can be changed. Hence dynamic on-line adaptation to a non-rigid environment is possible.

- Suitability for complex and not-fully formalised solution spaces: CBR systems can be applied to an incomplete model of problem domain; implementation involves both to identify relevant case features and to furnish, possibly a partial case base, with propercases.
- Suitability for sequential problem solving: Sequential tasks, like these encountered in reinforcement learning problems, benefit from the storage of history in the form of a sequence of states or procedures. Such storage is facilitated by lazy approaches.
- Ease of explanation: The results of a CBR system can be justified based upon the similarity of the current problem to the retrieved case(s). Because solutions generated by CBR are easily traceable to precedent cases, it is also easier to analyse failures of the system.
- Ease of maintenance: This is particularly due to the fact that CBR systems can adapt to many changes in the problem domain and the pertinent environment, merely by acquiring new cases. This eliminates some need for maintenance; only the case base needs to be maintained.

Limitations of CBR

Major disadvantages of lazy problem solvers are their memory requirements and time consuming execution due the processing necessary to answer the queries. The limitations of CBR can be summarised as follows:

- Handling large case bases
- Dynamic problem domains
- Handling noisy data
- Fully automatic operation

UNIT

5

ADVANCED LEARNING

5.1. LEARNING SETS OF RULES

- Learning the target function from the given training examples can be effectively represented as if-then rules.
- These rules are easy to understand.

This type of rule based learning happens in decision trees and genetic algorithms.

- Alternate ways of learning through rules is by framing first order rules and sequential covering algorithms.
- PROLOG is a programming language that is specifically designed to frame first order rules for given examples.

PROLOG (PROgramming in LOGic)

Prolog is a logic programming language. Unlike many other programming languages, Prolog is intended primarily as a declarative programming language. In prolog, logic is expressed as relations (called as Facts and Rules). Formulation or Computation is carried out by running a query over these relations. Prolog is a Programming Language for symbolic, non-numeric computation. Prolog is the major example of a fourth generation programming language. The distinguishing features of PROLOG includes:

- **Unification:** The basic idea is, can the given terms be made to represent the same structure.
- **Backtracking:** When a task fails, prolog traces backwards and tries to satisfy previous task.
- **Recursion:** Recursion is the basis for any search in program.

Prolog is based on 'Horn Clauses' or 'clauses' (Rules, Facts and Queries.)Horn Clauses are a subset of Predicate Logic. Predicate logic is a way of simply defining how reasoning gets done in logic terms. Predicate Logic is a syntax for easily reading and writing Logical ideas.

To transform an English sentence to Predicate Logic, remove unnecessary terms. This leaves only the relationship and the entities involved, known as **arguments**.

Example: An elephant is bigger than a horse is equivalent to: bigger (elephant, horse).

The relation is 'bigger', the relation's arguments are 'elephant and horse'.

In Prolog, the relation's name (e.g. "bigger") is the '**Functor**'. A relation may include many arguments after the functor.

A Prolog Program consists of clauses and each clause terminates with a full stop.

- bigger(elephant, horse).
- bigger(horse, donkey).
- bigger(donkey, dog).
- bigger(donkey, monkey).

Advantages:

- Easy to build database.
- Do not need a lot of programming effort.
- Pattern matching is easy. Search is recursion based.
- It has built in list handling. Makes it easier to play with any algorithm involving lists.

Disadvantages:

- LISP dominates over prolog with respect to I/O features.
- Sometimes input and output is not easy.

5.2. SEQUENTIAL COVERING ALGORITHM

Sequential covering is a general procedure that repeatedly learns a single rule to create a decision list or set that covers the entire dataset rule by rule.

The idea in a sequential covering algorithm is to learn one rule, remove the data it covers, then repeat.

- A covering algorithm develops a cover for the set of positive examples that is, a set of hypotheses that account for all the positive examples but none of the negative examples.
- This is called **sequential covering** because it learn one rule at a time and repeat this process to gradually cover the full set of positive examples. The most effective approach to Learn-One-Rule is beam search.
- The input rules have high accuracy, but not necessarily high coverage.
- By accepting low coverage, it need not make predictions for every training example.

Steps in Sequential Covering algorithm

1. Start with an empty Cover
2. Using Learn-One-Rule to find the best hypothesis.
3. If the Just-Learnt-Rule satisfies the threshold then
 - Put Just-Learnt-Rule to the Cover.
 - Remove examples covered by Just-Learnt-Rule.
 - Go to step 2.
4. Sort the Cover according to its performance over examples.
5. Return: Cover.

Algorithm for Sequential Coverage

```
Sequential_covering (Target_attribute, Attributes, Examples, Threshold) :  
    Learned_rules = { }  
    Rule = Learn-One-Rule(Target_attribute, Attributes, Examples)  
    while Performance(Rule, Examples) > Threshold :  
        Learned_rules = Learned_rules + Rule  
        Examples = Examples - {examples correctly classified by Rule}  
        Rule = Learn-One-Rule(Target_attribute, Attributes, Examples)  
    Learned_rules = sort Learned_rules according to performance over Examples  
    return Learned_rules
```

- The sequential covering algorithm for learning a disjunctive set of rules.
- The learn-one-rule return a single rule that covers at least some of the examples.
- Performance is a user-provided subroutine to evaluate rule quality. This covering algorithm learns rules until it can no longer learn a rule whose performance is above the given Threshold.

5.2.1. General to Specific Beam Search

Learn-one-rule

- ✓ This follows the principle of general-to-Specific Search.
- ✓ Consider the most general rule (hypothesis) which matches every instance in the training set.
- ✓ Repeatedly add the attribute that most improves rule performance measured over the training set until the hypothesis reaches an acceptable level of performance.

Algorithm for Learn-one-rule

```

// returns a single rule that covers some of the examples. Conducts a
general_to_specific

// greedy beam search for the best rule, guided by performance metric.
Initialize best_hypothesis to the most general hypothesis  $\phi$ 
Initialize Candidate_hypothesis to the set {best_hypothesis}
While candidate_hypothesis is not empty, Do
  1. Generate the next more specific candidate_hypothesis
      All_constraints  $\leftarrow$  set of all constraints of the form (a=v), a-member of
attributes,
      v-value of a that occurs in the current set of examples
      New_candidate_hypothesis  $\leftarrow$ 
        For each h in Candidate_hypothesis
          For each c in All_constraints
            Create a specialization of h by adding the constraint c

```

Remove from new_candidate_hypothesis any hypothesis that are duplicates,
Inconsistent or not maximally specific

2. Update best_hypothesis

For all h in new_candidate_hypothesis do

 If (performance(h, examples, Target_attribute)

 >(performance(best_hypothesis, examples, Target_attribute)

 Then best_hypothesis \leftarrow h

3. Update candidate_hypothesis

Candidate_hypothesis \leftarrow the k best members of new_candidate_hypothesis
according to the performance measure.

Return a rule of the form

 “IF best_hypothesis THEN prediction” where prediction is the most
frequent

 Value of target_attribute among those examples that match
best_hypothesis

Performance(h, examples, target_attribute)

h_examples \leftarrow subset of examples that match attributes h

return-entropy(h-examples) where entropy is with respect to target attribute.

The steps are:

Discretize the continuous features by choosing appropriate intervals.

For each feature:

 Create a cross table between the feature values and the (categorical)
outcome.

 For each value of the feature, create a rule which predicts the most frequent
class of the instances that have this particular feature value (can
be read from the cross table)

 Calculate the total error of the rules for the feature.

Select the feature with the smallest total error

General-to-Specific Beam Search:

- ✓ This is same as learn-one-rule but with a difference that rather than considering a single candidate at each search step, keep track of the k best candidates.
- ✓ The search begins by considering the most general rule precondition possible, then greedily adding the attribute test that most improves rule performance measured over the training examples.
- ✓ Once this test has been added, the process is repeated by greedily adding a second attribute test, and so on.
- ✓ This process grows the hypothesis by greedily adding new attribute tests until the hypothesis reaches an acceptable level of performance.
- ✓ This approach to implementing Learn-to-one forms a general-to specific search through the space of possible rules in search of a rule with high accuracy, though perhaps incomplete coverage of the data.
- ✓ As in decision tree learning, there are many ways to define a measure to select the best descendant.
- ✓ The general-to-specific search is a greedy algorithm with depth-first search with no backtracking.

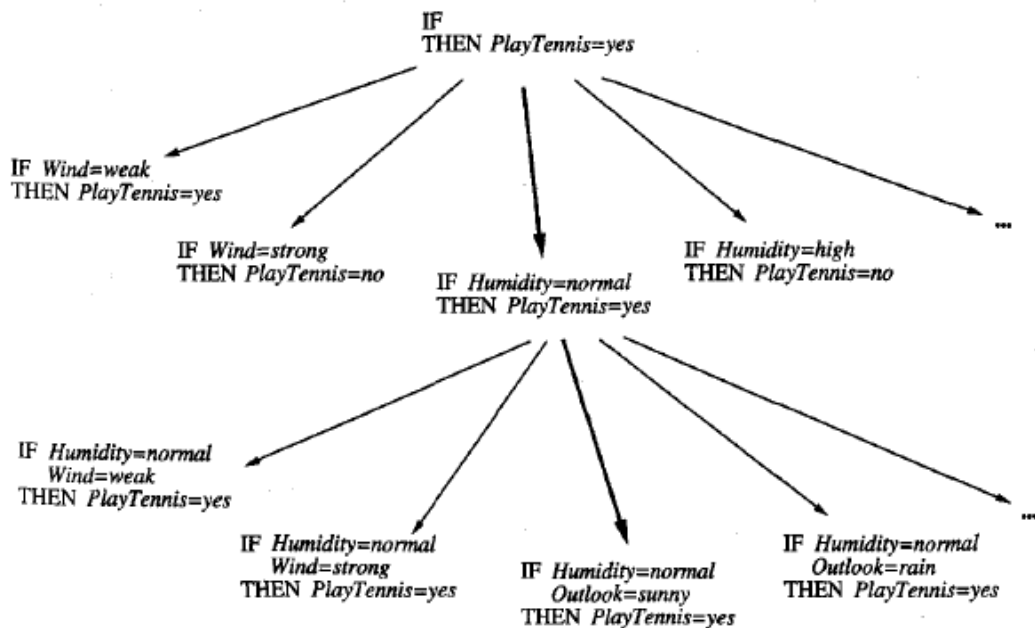


Fig. 5.1. Beam search of width = 1

-
- ✓ The search for rule preconditions as learn to one proceeds from general to specific.
 - ✓ At each step, the preconditions of the best rule are specialized in all possible ways.
 - ✓ Rule postconditions are determined by the examples found to satisfy the preconditions.
 - ✓ There is a danger that a suboptimal choice will be made at any step.
 - ✓ To reduce this risk, extend the algorithm to perform a beam search; that is, a search in which the algorithm maintains a list of the k best candidates at each step, rather than a single best candidate.
 - ✓ On each search step, descendants are generated for each of these k best candidates, and the resulting set is again reduced to the k most promising members.
 - ✓ Beam search keeps track of the most promising alternatives to the current top-rated hypothesis, so that all of their successors can be considered at each search step.
 - ✓ Each hypothesis considered in the main loop of the algorithm is a conjunction of attribute-value constraints.
 - ✓ Each of these conjunctive hypotheses corresponds to a candidate set of preconditions for the rule to be learned and is evaluated by the entropy of the examples it covers.
 - ✓ The search considers increasingly specific candidate hypotheses until it reaches a maximally specific hypothesis that contains all available attributes.
 - ✓ The postcondition for the output rule is chosen only in the final step of the algorithm, after its precondition has been determined.
 - ✓ The algorithm constructs the rule postcondition to predict the value of the target attribute that is most common among the examples covered by the rule precondition.

5.3. LEARNING RULE SETS

The implications of learning by rule strategy are discussed briefly in this section.

Relation between ID3 and Sequential Covering algorithm

The decision tree algorithm (ID3) and sequential covering algorithm learns the hypothesis in a similar manner.

Sequential Covering Algorithm	Simultaneous Covering algorithms
This learn one rule at a time, removing the covered examples and repeating the process on the remaining examples.	This learns the entire set of disjuncts simultaneously as part of the single search for an acceptable decision tree.
This chooses among alternative attribute-value pairs, by comparing the subsets of data they cover.	ID3 chooses among alternative attributes by comparing the partitions of the data they generate.
To learn a set of n rules, each containing k attribute-value tests in their preconditions, sequential covering algorithms will perform $n \cdot k$ primitive search steps, making an independent decision to select each precondition of each rule.	If the decision node tests an attribute that has m possible values, the choice of the decision node corresponds to choosing a precondition for each of the m corresponding rules
Sequential covering algorithms make a larger number of independent choices than simultaneous covering algorithms.	Simultaneous covering algorithms will make many fewer independent choices, because each choice of a decision node in the decision tree corresponds to choosing the precondition for the multiple rules associated with that node.
If data is more, then it may support the larger number of independent decisions required by the sequential covering algorithm,	If data is scarce, the sharing of decisions regarding preconditions of different rules may be more effective.
These algorithms different rules test the same attributes.	Testing an attribute using different rules is not possible.

Specific-to-general and General-to-specific

There are lot of controversies in drawing a hypothesis using Specific-to-general and General-to-specific approaches.

- In general to specific search there is a single maximally general hypothesis from which to begin the search, whereas there are very many specific hypotheses in most hypothesis spaces.
- The selection of starting search point is the major challenge in general to specific search.
- But the specific-to-general search, addresses this issue by choosing several positive examples at random to initialize and to guide the search.
- The best hypothesis obtained through multiple random choices is then selected.

Training through examples vs Generate and test approach

- The individual training examples constrain the generation of hypotheses.
- The generation or revision of hypotheses is driven by the analysis of an individual training example, and the result is a revised hypothesis designed to correct performance for this single example.
- The training data is considered only after these candidate hypotheses are generated and is used to choose among the candidates based on their performance over the entire collection of training examples.
- The generate and test approach is that each choice in the search is based on the hypothesis performance over many examples, so that the impact of noisy data is minimized.
- The example-driven algorithms that refine the hypothesis based on individual examples are more easily misled by a single noisy training example and are therefore less robust to errors in the training data.

Post Pruning the rules

The rules formed after forming the tree perform well on training data but not on new data. So post-pruning the rules is a good option to generalize the rules. Preconditions can be removed from the rule whenever this leads to improved performance over a set of pruning examples distinct from the training examples.

Evaluation metrics

The evaluation metric is very important to test the quality of the rules.

- **Relative frequency:**

Let n denote the number of examples the rule matches and let n_c denote the number of these that it classifies correctly. The relative frequency estimate of rule performance is n_c/n .

- **m-estimate of accuracy:**

This accuracy estimate is biased toward the default accuracy expected of the rule. It is often preferred when data is scarce and the rule must be evaluated based on few examples. As above, let n and n_c denote the number of examples matched and correctly predicted by the rule. The m-estimate of rule accuracy is:

$$\frac{n_c + mp}{n + m}$$

Where p be the prior probability and m is the weight.

- **Entropy:**

Entropy measures the uniformity of the target function values for this set of examples.

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

where c is the number of distinct values the target function may take on, and where p_i is the proportion of examples from S for which the target function takes on the i^{th} value.

5.4. LEARNING FIRST-ORDER RULES

Propositional Logic

- It is the study of propositions, where a proposition is a statement that is either true or false.
- Propositional logic may be used to encode simple arguments that are expressed in natural language, and to determine their validity.
- The validity of an argument may be determined from truth tables, or using inference rules such as modus ponens to establish the conclusion via deductive steps.

- This is not sufficient to represent the complex sentences or natural language statements. The propositional logic has very limited expressive power.

Predicate Logic

- Predicate logic allows complex facts about the world to be represented, and new facts may be determined via deductive reasoning.
- Predicate calculus includes predicates, variables and quantifiers, and a predicate is a characteristic or property that the subject of a statement can have.
- The universal quantifier is used to express a statement such as that all members of the domain of discourse have property P, and the existential quantifier states that there is at least one value of x has property P.

First Order Logic (FOL)

- Propositional logic is the foundation of first-order logic.
- First-order logic uses quantified variables over non-logical objects and allows the use of sentences that contain variables.
- The propositional logic does not use quantifiers or relations.
- First-order logic is another way of knowledge representation in artificial intelligence. It is an extension to propositional logic.
- FOL is sufficiently expressive to represent the natural language statements in a concise way.
- First-order logic is also known as Predicate logic or First-order predicate logic. First-order logic is a powerful language that develops information about the objects in a more easy way and can also express the relationship between those objects.
- First-order logic does not only assume that the world contains facts like propositional logic.

5.4.1. First-Order Horn Clauses

*A clause is called a Horn clause if it contains at most one positive literal.
Horn clauses express a subset of statements of first-order logic.*

The problem is that propositional representations offer no general way to describe the essential relations among the values of the attributes. Consider the following propositional logic which states that Sharon is the daughter of Bob.

If (Father₁ = Bob) \wedge (Name₂ = Bob) \wedge (Female₁ = true) THEN Daughter_{1,2} = True

These rules are very specific. Hence we cannot derive generic rules from them. Now consider the following FOL:

IF Father (y, x) \wedge Female(y), THEN Daughter (x, y)

This is a generic rule. Father (y, x) returns true if y is the father of x else false. Same goes for daughter (x, y) also. First-order Horn clauses may also refer to variables in the preconditions that do not occur in the postconditions. The previous rule may be extended to granddaughter also.

IF Father (y, z) \wedge Mother (z, x) \wedge Female (y) THEN GrandDaughter(x, y)

The variable z in this rule, which refers to the father of y, is not present in the rule postconditions. Whenever such a variable occurs only in the preconditions, it is assumed to be existentially quantified; that is, the rule preconditions are satisfied as long as there exists at least one binding of the variable that satisfies the corresponding literal. It is also possible to use the same predicates in the rule postconditions and preconditions, enabling the description of recursive rules.

5.4.2. Terminologies

- Every well-formed expression is composed of constants, variables, predicates and functions.
- Term - it is a constant, variable or function applied to any term.
- Literal- Any predicate or its negation applied to any set of terms.
- A ground literal is a literal that does not contain any variables.
- A negative literal is a literal containing a negated predicate.
- A positive literal is a literal with no negation sign.
- A clause is any disjunction of literals $M1 \vee \dots \vee Mn$ whose variables are universally quantified.
- A Horn clause is an expression of the form

$$H \leftarrow (L_1 \wedge \dots \wedge L_n)$$

H-head of the clause; $L_1 \dots L_n$ -Antecedents of the clause

- A substitution is any function that replaces variables by terms.
- A unifying substitution for two literals L_1 and L_2 is any substitution θ such that $L_1\theta = L_2\theta$.

5.5. LEARNING SETS OF FIRST-ORDER RULES: FOIL

The Foil algorithm is a supervised learning algorithm that produces rules in first-order logic. The FOIL (First Order Inductive Logic) algorithm is an extension of sequential covering and learn one rule algorithm. The output of this algorithm is single rules as in learn one rule. But it has two distinct features:

- **More restricted:** literals are not permitted to contain function symbols
- **More expressive:** literals in the body can be negated

FOIL is a top-down algorithm that starts out with a general rule and explores the search space by greedily specializing the current rule. The main modification is that search can also specialize on predicates with variables. The resulting rules differ from Horn clauses in two ways: negated symbols are allowed within the body, and FOIL's rules will not include function symbols.

Example:

Consider the following statement,

IF Father(y, z) ^ Father(z, x) ^ Female(x) THEN GrandDaughter (x, y).

- The algorithm starts with the most general rule.
- In this example the population has 1,000 individuals, 50 of which are granddaughters to someone in the data.
- As FOIL adds literals to the rule it will greedily choose the rule that increases the proportion of POSitives to NEGatives (tick marks).

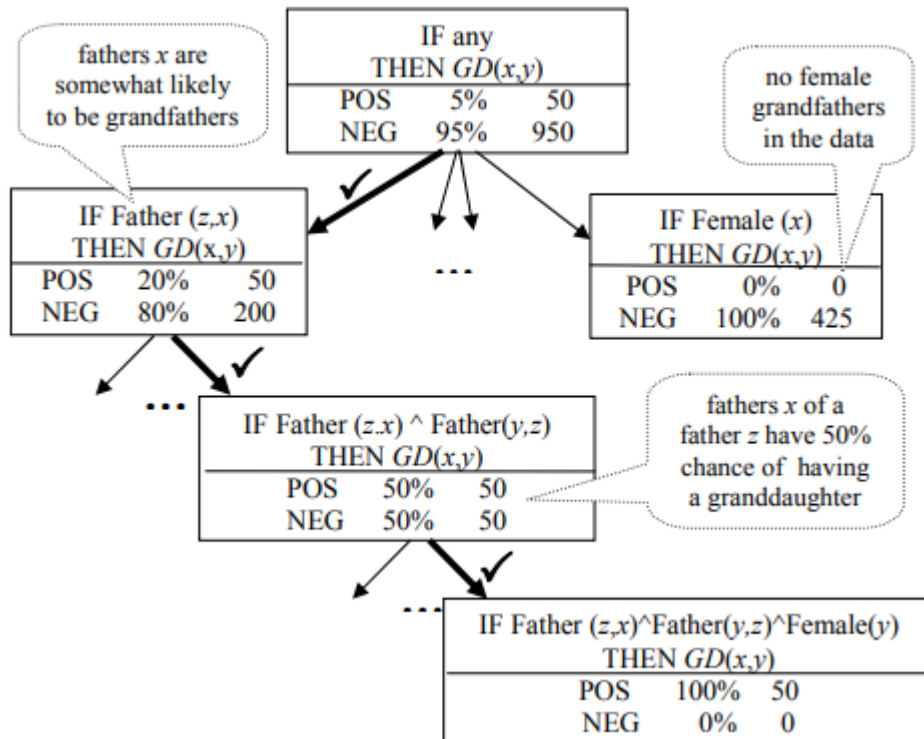


Fig. 5.2. Example of FOIL algorithm

FOIL algorithm

FOIL(Target-predicate, Predicates, Examples)

Pos \leftarrow those Examples for which the Target-predicate is True

Neg \leftarrow those Examples for which the Target-predicate is False

Learned_rules $\leftarrow \{ \}$

while Pos, do

Learn a NewRule

 New Rule \leftarrow the rule that predicts Target-predicate with no preconditions

 NewRuleNeg \leftarrow Neg

while NewRuleNeg, do

Add a new literal to specialize New Rule

 Candidate literals \leftarrow generate candidate new literals for NewRule,
based on

Predicates

Bestliteral $\leftarrow \text{argmax}_{L \in \text{candidate_literals}} \text{Foil-Gain}(L, \text{NewRule})$

add Bestliteral to preconditions of NewRule

NewRuleNeg \leftarrow subset of NewRuleNeg that satisfies NewRule preconditions

Learned_rules \leftarrow Learned-rules + NewRule

Pos \leftarrow Pos - {members of Pos covered by NewRule}

Return Learned-rules

- The outer loop corresponds to a variant of the sequential covering algorithm. It learns new rules one at a time, removing the positive examples covered by the latest rule before attempting to learn the next rule.
- The inner loop corresponds to a variant learn one rule algorithm which is extended to accommodate first-order rules.
- FOIL seeks only rules that predict when the target literal is True in contrast to other algorithms which seek both True and False rules.
- FOIL performs a simple hill climbing search rather than a beam search.
- The hypothesis space search performed by FOIL is best understood by viewing it hierarchically.
- Each iteration through FOIL'S outer loop adds a new rule to its disjunctive hypothesis, learned rules.
- The effect of each new rule is to generalize the current disjunctive hypothesis (i.e., to increase the number of instances it classifies as positive), by adding a, new disjunct.
- The search is a specific-to-general search through the space of hypotheses, beginning with the most specific empty disjunction and terminating when the hypothesis is sufficiently general to cover all positive training examples.
- The inner loop of FOIL performs a finer-grained search to determine the exact definition of each new rule.

- This inner loop searches a second hypothesis space, consisting of conjunctions of literals, to find a conjunction that will form the preconditions for the new rule.
- Within this hypothesis space, it conducts a general-to-specific, hill-climbing search, beginning with the most general preconditions possible (the empty precondition), then adding literals one at a time to specialize the rule until it avoids all negative examples.

Distinguishing features of FOIL against sequential covering and learn one rule

- In its general-to-specific search to 'learn each new rule, FOIL employs different detailed steps to generate candidate specializations of the rule. This difference follows from the need to accommodate variables in the rule preconditions.
- FOIL employs a PERFORMANCE measure, Foil-Gain, that differs from the entropy measure as in learn one rule. This difference follows from the need to distinguish between different bindings of the rule variables and from the fact that FOIL seeks only rules that cover positive examples.

5.5.1. Expanding candidate specializations in FOIL

FOIL expands its search space by specializing rules through the addition of literals to the rule body. If the current rule has rule head $P(x_1, x_2, \dots, x_k)$ and body literals of $L_1, L_2 \dots L_n$ the following three type of literal additions can be done:

- $Q(v_i, \dots, v_r)$ where Q is a valid predicate and at least one of variable v_1 is already in the rule body.
- $\text{Equal}(x_j, x_k)$, where variables x_j, x_k are already in the rule.

The negation of the above: $\neg Q(v_i, \dots, v_r)$ or $\neg \text{Equal}(x_j, x_k)$

Example:

- Consider learning rules to predict the target literal $\text{GrandDaughter}(x, y)$, where the other predicates used to describe examples are Father and Female .
- The general-to-specific search in FOIL begins with the most general rule

$\text{GrandDaughter}(x, y) \leftarrow$

which asserts that $\text{GrandDaughter}(x, y)$ is true of any x and y .

- To specialize this initial rule, the above procedure generates the following literals as candidate additions to the rule preconditions:
Equal (x,y) , Female(x), Female(y), Father(x, y), Father(y, x), Father(x, z), Father(z, x), Father(y, z), Father(z, y), and the negations of each of these literals (e.g., -Equal(x, y)).
- FOIL greedily selects Father (y, z) as the most promising, leading to the more specific rule

Grand Daughter(x, y) \leftarrow Father (y, z)

- In generating candidate literals to further specialize this rule, FOIL will now consider all of the literals mentioned in the previous step, plus the additional literals Female(z), Equal(z, x), Equal(z, y), Father(z, w), Father(w, z), and their negations.
- These new literals are considered at this point because the variable z was added to the rule in the previous step. Because of this, FOIL now considers an additional new variable w.
- If FOIL at this point were to select the literal Father(z, x) and on then next iteration select the literal Female(y), this would lead to the following rule, which covers only positive examples and hence terminates the search for further specializations of the rule.

GrandDaughter(x, y) \leftarrow Father (y, z) \wedge Father (z, x) \wedge female (y)

- At this point, FOIL will remove all positive examples covered by this new rule.
- If additional positive examples remain to be covered, then it will begin yet another general-to-specific search for an additional rule.

5.5.2. Guiding the Search (Foil_Gain)

- FOIL uses a version of the gain algorithm to determine which newly specialized rule to favour.
- Each rule's utility is estimated by the number of bits required to encode all of the positive bindings.
- To select the most promising literal from the candidates generated at each step, FOIL considers the performance of the rule over the training data.

- In doing this, it considers all possible bindings of each variable in the current rule.
- The $\text{Foil_Gain}(L, R)$ is estimated from

$$\text{Foil_Gain}(L, R) \equiv t(\log_2 \frac{p_1}{p_1 + n_1} - \log_2 \frac{p_0}{p_0 + n_0})$$

Where

L is the candidate literal to add to rule R

p_0 = number of positive bindings of R

n_0 = number of negative bindings of R

p_1 = number of positive bindings of $R + L$

n_1 = number of negative bindings of $R + L$

t is the number of positive bindings of R also covered by $R + L$

Here R -rule and L -literal.

Example

- Assume the training data includes the following simple set of assertions, where we use the convention that $P(x, y)$ can be read as "The P of x is y ."
- The clauses for learning $\text{GrandDaughter}(x, y)$ is given as:
 $\text{GrandDaughter}(\text{Victor}, \text{Sharon})$ $\text{Father}(\text{Sharon}, \text{Bob})$ $\text{Father}(\text{Tom}, \text{Bob})$
 $\text{Female}(\text{Sharon})$ $\text{Father}(\text{Bob}, \text{Victor})$
- The clauses that are not mentioned above are assumed to be false.
- To select the best specialization of the current rule, FOIL considers each distinct way in which the rule variables can bind to constants in the training examples.

$\text{GrandDaughter}(x, y) \leftarrow$

- The rule variables x and y are not constrained by any preconditions and may therefore bind in any combination to the four constants Victor, Sharon, Bob, and Tom.
- At each stage, the rule is evaluated based on these sets of positive and negative variable bindings, with preference given to rules that possess more positive bindings and fewer negative bindings.

- As new literals are added to the rule, the sets of bindings will change. Note if a literal is added that introduces a new variable, then the bindings for the rule will grow in length.
- If the new variable can bind to several different constants, then the number of bindings fitting the extended rule can be greater than the number associated with the original rule.
- $-\log_2 \frac{p_0}{p_0 + n_0}$ is the number of optimal bits to indicate class of positive bindings.

5.5.3. Learning Recursive Sets

- A more sophisticated form of rule specialization involves the addition of a literal that contains the target predicate.
- If we include the target predicate in the input list of predicates, then FOIL will consider it as well when generating candidate literals.
- This will allow it to form recursive rules-rules that use the same predicate in the body and the head of the rule.
- This specialization is necessary to discover rules such as

IF Parents(x, z)^Ancestor(z, y) THEN Ancestor(x, y)

- The Ancestor (z, y) literal initiates a recursive description.
- Rules must not cause infinite recursion.

5.6. INDUCTION ON INVERTED DEDUCTION

Induction is the reverse of deduction.

Induction	Deduction
Inductive reasoning consists in constructing the axioms from the observation of supposed consequences of these axioms.	Deductive reasoning consists in combining logical statements according to certain agreed upon rules in order to obtain new statements.
This is what scientists like physicists for example do: observing natural phenomena, they postulate the laws of Nature.	This is how mathematicians prove theorems from axioms. Proving a theorem is nothing but combining a small set of axioms with certain rules.
Inductive reasoning involves making a	Deductive reasoning uses available

generalization from specific facts, and observations.	facts, information, or knowledge to deduce a valid conclusion.
Inductive reasoning uses a bottom-up approach.	Deductive reasoning uses a top-down approach.
Inductive reasoning moves from specific observation to a generalization.	Deductive reasoning moves from generalized statement to a valid conclusion.
In inductive reasoning, the conclusions are probabilistic.	In deductive reasoning, the conclusions are certain.
The limitation is that it is impossible to prove that an inductive statement is correct. At most can one empirically observe that the deductions that can be made from this statement are not in contradiction with experiments. But one can never be sure that no future observation will contradict the statement.	The limitation is for a rich enough set of axioms, one can produce statements that can be neither proved nor disproved.
Inductive argument can be strong or weak, which means conclusion may be false even if premises are true.	Deductive arguments can be valid or invalid, which means if premises are true, the conclusion must be true.

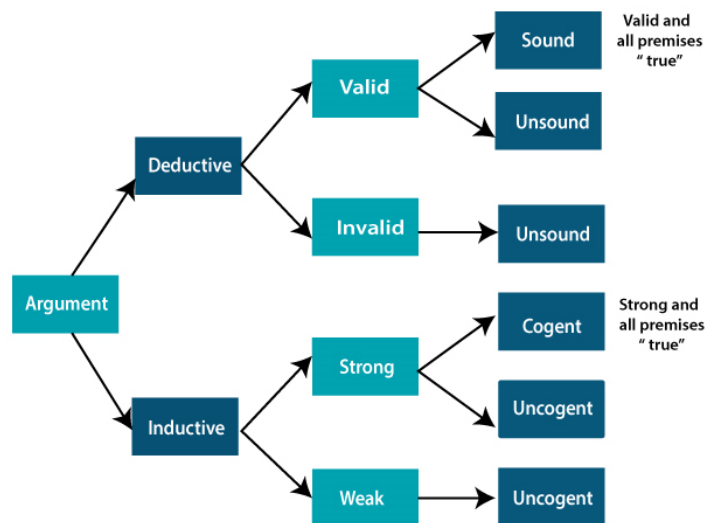


Fig. 5.3. Inductive vs Deductive learning

Induction is finding hypothesis h such that,

$$(\forall \langle x_i, f(x_i) \rangle \in D) B \wedge h \wedge x_i \vdash f(x_i)$$

Where D is the training data, x_i is the i^{th} training instance, $f(x_i)$ is the target function value for x_i and B is the back ground knowledge. For each training data instance, the instance's target classification is logically entailed by the background knowledge, together with hypothesis and the instance itself.

Example: Induction as inverted Deduction

Concept: Pairs of people, $\langle u, v \rangle$ such that child of u is v .

Consider that there exist one training example

- ✓ describing two people Tim and Sharon in terms of their gender and their relation in terms of the predicate father.
- ✓ giving the target predicate $\text{Child}(\text{Tim}, \text{Sharon})$ for these two people

The background knowledge is that if u is the father of v then u is a parent of v . Then,

$f(x_i)$: $\text{child}(\text{Time}, \text{Sharon})$

x_i : Male (Tim), Female (Sharon), Father (Sharon, Tim)

B : $\text{parent}(u, v) \leftarrow \text{father}(u, v)$.

The hypothesis h satisfies

$$(\forall \langle x_i, f(x_i) \rangle \in D) B \wedge h \wedge x_i \vdash f(x_i)$$

h_1 : $\text{Child}(u, v) \leftarrow \text{Father}(v, u)$

h_2 : $\text{Child}(u, v) \leftarrow \text{Parent}(v, u)$

$h_1 \wedge \text{Father}(\text{Sharon}, \text{Tim}) \vdash \text{Child}(\text{Tim}, \text{Sharon})$

$\text{Parent}(u, v) \leftarrow \text{father}(u, v) \wedge h_2 \wedge \text{father}(\text{Sharon}, \text{Tim}) \vdash \text{Child}(\text{Tim}, \text{Sharon})$

This is learning a hypothesis using inverse deduction.

Advantages:

- Supports earlier idea of finding h that fits training data.
- Domain theory B helps define meaning of fit the data
- Suggests algorithms that search H guided by B

Disadvantages:

- Noise can result in inconsistent constraints in h and most logical frameworks break down when given inconsistent sets of assertions. Hence this is not good with noisy data.
- First order logic gives a huge hypothesis space H with overfitting and intractability of calculating all acceptable hypothesis
- While using background knowledge B should help constrain hypothesis search, for many ILP systems hypothesis space search increases as B is increased.

5.7. INVERTING RESOLUTIONS

- The automated deduction is obtained by the resolution.
- The resolution rule is a sound and complete rule for deductive inference in first-order logic.
- The resolution rule can be inverted to form an inverse entailment operator.
- It is easiest to introduce the resolution rule in propositional form, though it is readily extended to first-order representations.
- Let L be an arbitrary propositional literal, and let P and R be arbitrary propositional clauses. The resolution rule is:

$$\begin{array}{l}
 P \vee L \\
 \hline
 \neg L \vee R \\
 \hline
 P \vee R
 \end{array}$$

- Given the two clauses above the line, conclude the clause below the line.
- Intuitively, the resolution rule is quite sensible. Given the two assertions $P \vee L$ and $\neg L \vee R$, it is obvious that either L or $\neg L$ must be false. Therefore, either P or R must be true. Thus, the conclusion $P \vee R$ of the resolution rule is intuitively satisfying.
- Given two clauses $C1$ and $C2$, the resolution operator first identifies a literal L that occurs as a positive literal in one of these two clauses and as a negative literal in the other. It then draws the conclusion given by the above formula.

$$\begin{array}{ll}
 C_1 & P \vee L \\
 C_2 & \underline{\neg L \vee R} \\
 \text{Resolvent:} & P \vee R
 \end{array}$$

- Treating clauses as sets of literals (i.e. implicit disjunction) resolution is defined as follows:
 - ✓ Given initial clauses C_1 and C_2 , find a literal L from clause C_1 such that $\neg L$ occurs in C_2
 - ✓ Form the resolvent C by including all literals from C_1 and C_2 , except for L and $\neg L$. More precisely, the set of literals occurring in the conclusion C is $C = (C_1 - \{L\}) \cup (C_2 - \{\neg L\})$

Example:

Consider,

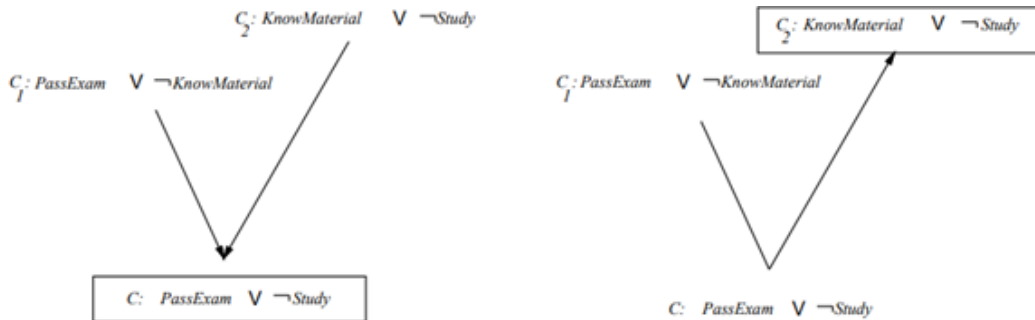
C_2 Study \rightarrow KnowMaterial

\neg Study \vee KnowMaterial

C_1 KnowMaterial \rightarrow PassExam

$\equiv \neg$ KnowMaterial \vee PassExam

C Study \rightarrow PassExam \neg Study \vee PassExam



So, to invert we need to go from C and C_1 to C_2 . A general operation for doing this is (inverted resolution):

1. Given initial clauses C_1 and C , find a literal L that occurs in clause C_1 , but not in clause C .
2. Form the second clause C_2 by including the following literals.

$$C_2 = (C - (C_1 - \{L\})) \cup \{\neg L\}$$

5.7.1. First Order Resolution

First order resolution takes two clauses C1 and C2 as input and yields a third C as output. Unlike propositional resolution the C1 and C2 must be related not by sharing a literal and its negation, but by sharing a literal and negated literal that can be matched by a unifying substitution.

Formally, first order resolution is defined as follows:

1. Find a literal L1 from clause C1, literal L2 from clause C2, and substitution q such that $L1q = \neg L2q$
2. Form the resolvent C by including all literals from C1q and C2q, except for L1q and $\neg L2q$.

More precisely, the set of literals occurring in the conclusion C is

$$C = (C_1 - \{L_1\}) \theta \cup (C_2 - \{L_2\})\theta$$

Example:

C1	Swan(X) \rightarrow White(X)	\neg Swan(X) \vee White(X)
C2	Swan(fred)	\equiv Swan(fred)
C	White(fred)	White(fred)

Setting $\theta = \{X/\text{Fred}\}$,

$$C = (C_1 - \{L_1\}) \theta \cup (C_2 - \{L_2\})\theta$$

$$= (\{\neg\text{Swan}(X), \text{White}(X)\} - \{\neg\text{Swan}(X)\})\{X/\text{fred}\} \cup$$

$$(\{\text{Swan}(\text{frd})\} - \{\text{Swan}(\text{fred})\})\{X/\text{fred}\} = \text{white}(\text{fred})$$

An inverse first order resolution operator can be derived by algebraic manipulation of the equation expressin the definition of the first order resolvent:

$$C = (C_1 - \{L_1\}) \theta \cup (C_2 - \{L_2\})\theta$$

The substitution q can be factored into two substitutions θ_1 and θ_2 such that:

- $\theta = \theta_1\theta_2$
- θ_1 contains all and only variable bindings involving variables in C1
- θ_2 contains all and only variable bindings involving variables in C2

Since C1 and C2 are universally quantified they can be rewritten, if necessary, to contain novariables in common. Hence the above definition can be rewritten as:

$$C = (C_1 - \{L_1\}) \theta_1 \cup (C_2 - \{L_2\}) \theta_2$$

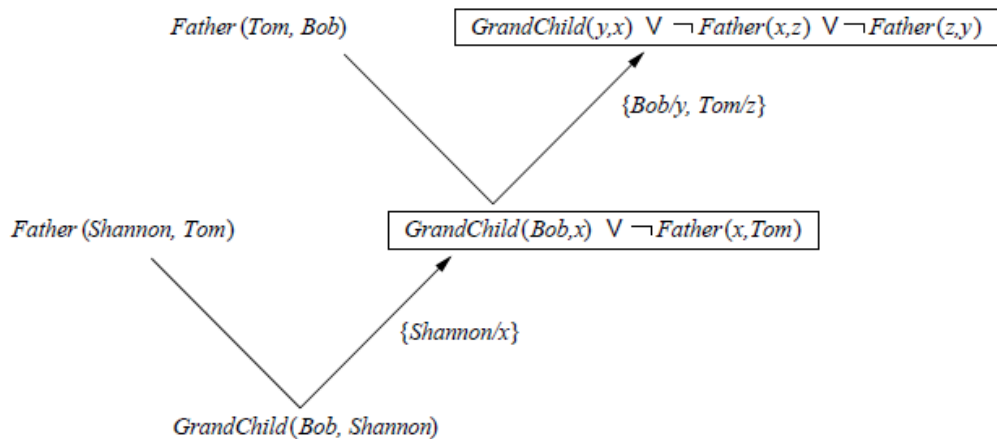
If we restrict inverse resolution to infer clauses C_2 that contain no literals in common with C_1 then re-write above as:

$$C = (C_1 - \{L_1\}) \theta_1 \cup (C_2 - \{L_2\}) \theta_2$$

Note that $L_2 = \neg L_1 \theta_1 \theta_2^{-1}$,

$$C_2 = (C - (C_1 - \{L_1\}) \theta_1) \theta_2^{-1} \cup (\neg L_1 \theta_1 \theta_2^{-1})$$

Example:



To learn rules for target predicate *GrandChild*(y, x) given

- training data $D = GrandChild(Bob, Shannon)$
- background information $B = \{Father(Shannon, Tom), Father(Tom, Bob)\}$

Proceed as follows:

1. Set $C = GrandChild(Bob, Shannon)$
2. Select $C_1 = Father(Shannon, Tom)$ from B
3. For inverse resolution L_1 must be *Father*(Shannon, Tom). Selecting $\theta_2^{-1} = \{Shannon/x\}$,

$$C_2 = (C - (C_1 - \{L_1\}) \theta_1) \theta_2^{-1} \cup \{\neg L_1 \theta_1 \theta_2^{-1}\}$$

$$= GrandChild(Bob, x) \cup \{\neg father(x, Tom)\}$$
4. Appropriate choices for L_1 and θ_2^{-1} yields,

$$GrandChild(y, x) \leftarrow Father(x, z) \wedge Father(z, y)$$

Summary

- The inverse resolution operation is non-deterministic.
- In general, for a given target predicate C
 - ✓ there are many ways to pick C1 and L1
 - ✓ many ways to pick the unifying substitutions θ_1 and θ_2
- Rule learning algorithms based on inverse resolution have been developed.
- For example CIGOL uses sequential covering to iteratively learn a set of Horn clauses that covers positive examples
 - ✓ on each iteration a training example $(x_i, f(x_i))$ not yet covered by rules is selected
 - ✓ inverse resolution is used to generate a candidate hypothesis h that satisfies $B \wedge h \wedge x_i \vdash f(x_i)$ where B is background knowledge plus clauses learned already.
 - ✓ note that this is an example-driven search, though if multiple hypotheses cover the example, then the one with highest accuracy over further examples can be preferred .
 - ✓ this contrasts with FOIL which uses a generate-then-test approach.

5.7.2. Generalization, θ -Subsumption, and Entailment

- **more-general-than:** Given two boolean-valued functions $h_j(x)$ and $h_k(x)$, we say that $h_j \geq h_k$ if and only if $(\forall x) h_k(x) \rightarrow h_j(x)$

This relation is used by many learning algorithms to guide search through the hypothesis space.

- **θ subsumption:** Consider two clauses C_j and C_k , both of the form $H \vee L_1 \vee \dots \vee L_n$, where H is a positive literal, and the L_i are arbitrary literals. Clause C_j is said to θ -subsume clause C_k if and only if there exists a substitution θ such that $C_j \theta \subseteq C_k$

- **Entailment:** Consider two clauses C_j and C_k . Clause C_j is said to entail clause C_k if and only if C_k follows deductively from C_j .

Consider a boolean-valued hypothesis $h(x)$ for some target concept $c(x)$, where $h(x)$ is expressed by a conjunction of literals, then we can re-express the hypothesis as the clause

$$c(x) \leftarrow h(x)$$

x is classified a negative example if it cannot be proven to be a positive example. If

$$h_1 \geq h_2$$

then the clause $C1: c(x) \leftarrow h_1(x)$ θ -subsumes the clause $C2: c(x) \leftarrow h_2(x)$.

θ -subsumption is a special case of entailment. That is, if clause A θ -subsumes clause B , then $A \models B$. However, we can find clauses A and B such that $A \models B$, but where A does not θ -subsume B .

5.7.3. PROGOL

Progol is implementation of inductive logic programming used in computer science that combines Inverse Entailment with general-to-specific search through a refinement graph.

Inverse Entailment is used with mode declarations to derive the most-specific clause within the mode language which entails a given example. This clause is used to guide a refinement-graph search. Inverse resolution is one way to invert deduction to derive inductive generalisations. But, can easily lead to combinatorial explosion of candidate hypotheses due to multiple choices for:

- ✓ input clauses/literals for inverse resolution
- ✓ unifying substitutions for inverse resolution

PROGOL reduces this combinatorial explosion by using an alternative approach, called **mode directed inverse entailment (MDIE)**.

- ✓ Use inverse entailment to generate most specific h that together with background information entails observed data
- ✓ Then perform general-to-specific search through a hypothesis space H bounded by the most specific hypothesis and constrained by user-specified predicates.

PROGOL algorithm:

- The user specifies a restricted language of first-order expressions to be used as the hypothesis space H . Restrictions are stated using mode declarations, which enable the user to specify the predicate and function symbols to be considered, and the types and formats of arguments for each.
- PROGOL uses a sequential covering algorithm to learn a set of expressions from H that cover the data.
- PROGOL then performs a general-to-specific search of the hypothesis space bounded by the most general possible hypothesis and by the specific bound h_i calculated in step 2.

5.8. ANALYTICAL LEARNING

Analytical learning uses prior knowledge and deductive reasoning to augment the information provided by the training examples. This is different from inductive learning that require a certain number of training examples to achieve a given level of generalization accuracy.

Differences between analytical and inductive learning

Analytical Learning	Inductive Learning
Logical reasoning is used to identify features.	Statistical reasoning is used to identify features.
Works well even with scarce training examples.	Fundamental bounds on accuracy depend on number of training examples.
<div style="text-align: center;">deduction</div> <div style="display: flex; justify-content: space-between;"> <div>Examples + Prior Knowledge</div> <div style="text-align: center;">></div> <div>hypothesis (generalization)</div> </div>	<div style="text-align: center;">induction</div> <div style="display: flex; justify-content: space-between;"> <div>Examples [+ Prior Knowledge]</div> <div style="text-align: center;">-----></div> <div>hypothesis (generalization)</div> </div>
Hypothesis fits domain theory.	Hypothesis fits data.

- In **inductive learning**, the learner is given a hypothesis space H from which it must select an output hypothesis, and a set of training examples $D = \{(x_1, f(x_1)), \dots, (x_n, f(x_n))\}$ where $f(x_i)$ is the target value for the instance x_i . The desired output of the learner is a hypothesis h from H that is consistent with these training examples.

- In analytical learning, the input to the learner includes the same hypothesis space H and training examples D as for inductive learning. In addition, the learner is provided an additional input: A domain theory B consisting of background knowledge that can be used to explain observed training examples. The desired output of the learner is a hypothesis h from H that is consistent with both the training examples D and the domain theory B .

The analytical learning problem must provide a domain theory sufficient to explain why observed positive examples satisfy the target concept. This is the distinguishing factor of analytical learning

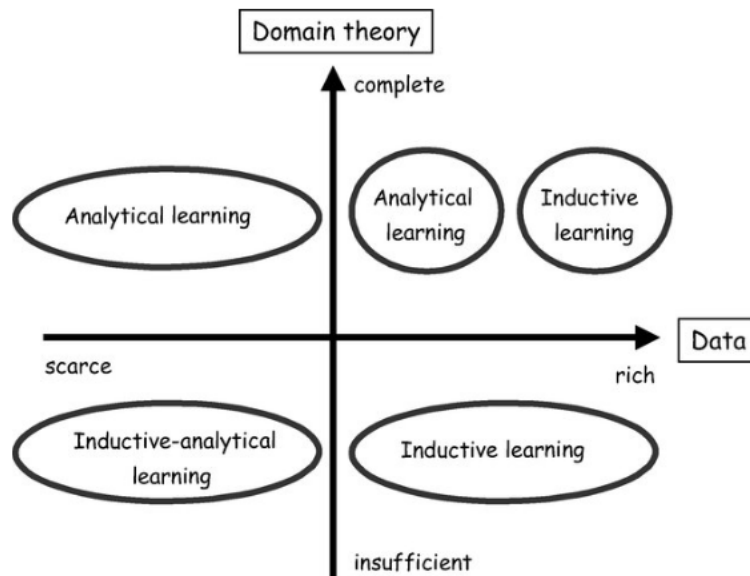


Fig. 5.4. Choice between Inductive and analytical learning

Example:

X: Instance Space: Each instance is a pair of objects describe by: Type, Color, Volume, Owner, Material, Density, and On (can be stacked on another object).

H: Hypothesis space: Each one is a set of Horn clauses.

Target Concept: SafeToStack(x,y)

D: SafeToStack(o1,o2), On(o1,o2), Type(o1,Box), Type(o2, endtable), Color(o1,red), Color(o2,blue), Volume(o1, 2), Owner(o1, Fred), Owner(o2, Louis), Density(o1, 0.3), Material(o1, Cardboard), Material(o2, Wood)

B: Domain Theory

$\text{SafeToStack}(x,y) \leftarrow \neg \text{Fragile}(y)$

$\text{SafeToStack}(x,y) \leftarrow \text{Lighter}(x,y)$

$\text{Lighter}(x,y) \leftarrow \text{Weight}(x,w_x) \wedge \text{Weight}(y,w_y) \wedge \text{LessThan}(w_x,w_y)$

$\text{Weight}(x,w) \leftarrow \text{Volume}(x,v) \wedge \text{Density}(x,d) \wedge \text{Equal}(w,\text{times}(v,d))$

$\text{Weight}(x,5) \leftarrow \text{Type}(x, \text{endtable})$

$\text{Fragile}(x) \leftarrow \text{Material}(x, \text{Glass})$

- The crux of analytical learning is finding h that is consistent with training examples and domain theory.
- The domain theory must explain why certain pairs of objects can be safely stacked on one another.
- The domain theory includes assertions such as "it is safe to stack x on y if y is not Fragile," and "an object x is Fragile if the Material from which x is made is Glass."
- Like the learned hypothesis, the domain theory is described by a collection of Horn clauses, enabling the system in principle to incorporate any learned hypotheses into subsequent domain theories.
- The domain theory is sufficient to prove that the positive example shown there satisfies the target concept SafeToStack.

5.9. PERFECT DOMAIN THEORY

- A perfect domain theory is correct and complete.

A domain theory is correct if each of its assertions is a truthful statement about the world.

- A domain theory is complete with respect to target concept and X , if it covers every positive example in the instance space.
- Domain theory is complete if every instance that satisfies the target concept can be proven by the domain theory to satisfy it.
- Completeness does not require that the domain theory be able to prove that negative examples do not satisfy the target concept.
- Perfect domain theories are often unrealistic, but, learning in them is a first step before learning with imperfect theories.

Learning through domain theory:

The need to learn the problem even after having the domain theory is due to two reasons:

- There are cases in which it is feasible to provide a perfect domain theory. For example, in chess problem in which the legal moves of chess form a perfect domain theory from which the optimal chess playing strategy can (in principle) be inferred. Optimal chess playing rules are extremely difficult to be framed. Here, provide the domain theory to the learner and rely on the learner to formulate a useful description of the target concept by examining and generalizing from specific training examples.
- It is unreasonable to assume that a perfect domain theory is available. It is difficult to write a perfectly correct and complete theory even simple problem.

5.9.1. PROLOG-EBG (Kedar-Cabelli and McCarty)

PROLOG-EBG is an illustration to perfect domain theory, which is a representative of several explanation-based learning algorithms. It is a method that combines the power of inductive and analytical learning.

PROLOG-EBG is a sequential covering algorithm that operates by learning a single Horn clause rule, removing the positive training examples covered by this rule, then iterating this process on the remaining positive examples until no further positive examples remain uncovered.

- When given a complete and correct domain theory, PROLOG-EBG is guaranteed to output a hypothesis that is itself correct and that covers the observed positive training examples.
- For any set of training examples, the hypothesis output by PROLOG-EBG constitutes a set of logically sufficient conditions for the target concept, according to the domain theory.

//Target function: the function to be learned.

//Domain theory: prior knowledge capable of predicting/explaining the output of the target function given its input

//Training instances: examples of input, output pairs of the target function

```
//Operation criterion: constraints on how the learned function must be described
//Determine: An operational representation of the target function that best fits both the
observed training instances and the given domain theory
Prolog-EGB(TargetConcept, TrainingExamples, DomainTheory)
LearnedRules = { }
Pos = the positive examples from TrainingExamples.
for each PositiveExample in Pos that is not covered by LearnedRules do
1. Explain
Explanation = an explanation in terms of DomainTheory that Pos satisfies the
TargetConcept
2. Analyse
SufficientConditions = the most general set of features of PositiveExample sufficient
to satisfy the TargetConcept according to the Explanation
3. Refine
LearnedRules = LearnedRules + newHornClause where newHornClause is of the
form Targetconcept ← sufficient conditions
return LearnedRules
```

- The new Horn clause is created by
 1. Explain *i*, by showing how the domain theory predicts the value of the target function for input *i*
 2. Analyze this explanation to determine the relevance of different features of *i* with respect to the target function
 3. Refine the current representation of the target function to take into account this new training example and the information about feature relevance extracted from the explanation
- There may be multiple possible explanations for the domain.
- In such cases, any or all of the explanations may be used.
- While each may give rise to a somewhat different generalization of the training example, all will be justified by the given domain theory.

Example:

Concept: SafeToStack(o1,o2) can be explained by using the domain theory, as such:

1. $\text{Volume}(o1,2) \wedge \text{Density}(o1,0.3) \wedge \text{Equal}(0.6, 2*0.3) \rightarrow \text{Weight}(o1,0.6)$
2. $\text{Type}(o2,\text{endtable}) \rightarrow \text{Weight}(o2,5)$
3. $\text{Weight}(o1, 0.6) \wedge \text{LessThan}(0.6, 5) \wedge \text{Weight}(o2,5) \rightarrow \text{Lighter}(o1,o2)$
4. $\text{Lighter}(o1, o2) \rightarrow \text{SafeToStack}(o1,o2)$

In Prolog-EGB this explanation is generated using backward chaining search, as done by Prolog. Like Prolog, it halts when it finds a proof.

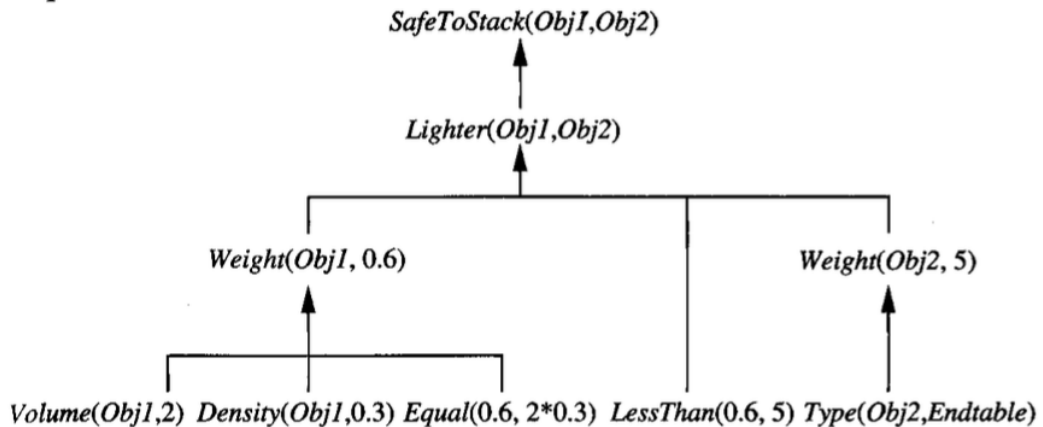
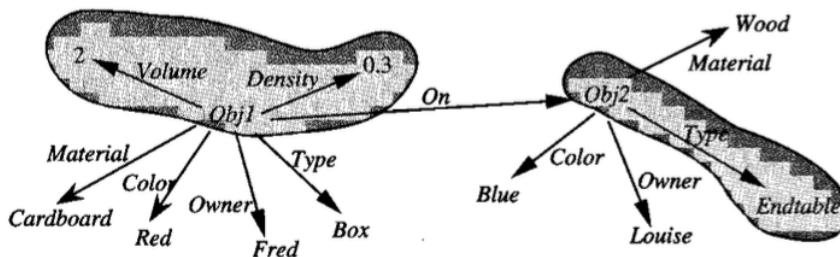
Explanation:**Training Example:**

Fig. 5.5. Explanation of training example

5.9.2. Analysis of the explanation

- It is important to find the fact that out the many features that happen to be true of the current training example, which ones are generally relevant to the target concept.
- By including $\text{Density}(x, 0.3)$, but not $\text{Owner}(x, \text{Fred})$, the hypothesis for $\text{SafeToStack}(x,y)$ becomes

$$\text{Volume}(x,2) \wedge \text{Density}(x,0.3) \wedge \text{Type}(y,\text{endtable}) \rightarrow \text{SafeToStack}(x,y)$$

- This is obtained by substituting variables x and y for Obj1 and Obj2 . The explanation of the training example forms a proof for the correctness of this rule.
- Although this explanation was formed to cover the observed training example, the same explanation will apply to any instance that matches this general rule.
- The above rule constitutes a significant generalization of the training example, because it omits many properties of the example that are irrelevant to the target concept.
- However, an even more general rule can be obtained by more careful analysis of the explanation.
- PROLOG-EBG computes the most general rule that can be justified by the explanation, by computing the weakest preimage of the explanation.

The weakest preimage of a conclusion C with respect to a proof P is the most general set of assertions A , such that A entails C according to P .

- Prolog-EGB computes the most general rule that can be justified by the explanation by computing the weakest preimage.
- PROLOG-EBG computes the weakest preimage of the target concept with respect to the explanation, using a general procedure called regression.

Regression

- The regression procedure operates on a domain theory represented by an arbitrary set of Horn clauses.

- It works iteratively backward through the explanation, first computing the weakest preimage of the target concept with respect to the final proof step in the explanation, then computing the weakest preimage of the resulting expressions with respect to the preceding step, and so on.
- The procedure terminates when it has iterated over all steps in the explanation, yielding the weakest precondition of the target concept with respect to the literals at the leaf nodes of the explanation.

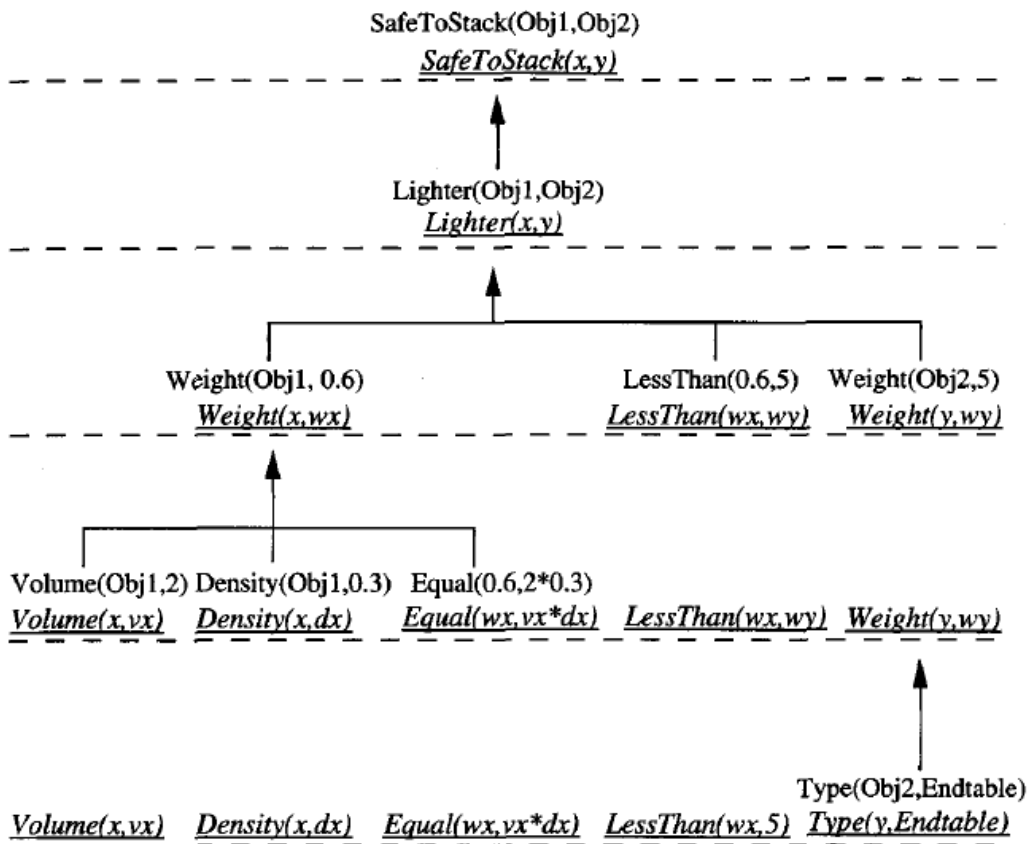


Fig. 5.6. Computation of weakest preimage

Learning the concept $\text{SafeToStack}(x, y)$ through regression

- The target concept is regressed from the root (conclusion) of the explanation, down to the leaves.

- At each step (indicated by the dashed lines) the current frontier set of literals (underlined in *italics*) is regressed backward over one rule in the explanation.
 - When this process is completed, the conjunction of resulting literals constitutes the weakest preimage of the target concept with respect to the explanation.
4. $\text{Lighter}(\text{o1}, \text{o2}) \rightarrow \text{SafeToStack}(\text{o1}, \text{o2})$

Lighter(x, y)

3. $\text{Weight}(\text{o1}, 0.6) \wedge \text{LessThan}(0.6, 5) \wedge \text{Weight}(\text{o2}, 5) \rightarrow \text{Lighter}(\text{o1}, \text{o2})$

Weight(x,wx), LessThan(wx, wy), Weight(y,wy)

2. $\text{Type}(\text{o2}, \text{endtable}) \rightarrow \text{Weight}(\text{o2}, 5)$

Weight(x,wx), LessThan(wx,5), Type(y,endtable)

1. $\text{Volume}(\text{o1}, 2) \wedge \text{Density}(\text{o1}, 0.3) \wedge \text{Equal}(0.6, 2 * 0.3) \rightarrow \text{Weight}(\text{o1}, 0.6)$

Volume(x,vx), Density(x,dx), Equal(wx, vx*dx), LessThan(wx,5), Type(y,endtable)

The final Horn clause has a body that corresponds to the weakest preconditions, and the head is the concept:

SafeToStack(x,y) \leftarrow Volume(x, vx) \wedge Density(x,dx) \wedge Equal(wx, vx*dx) \wedge LessThan(wx,5) \wedge Type(y,endtable)

5.9.3. Refine the current hypothesis

The current hypothesis at each stage consists of the set of Horn clauses learned

- thus far.
- At each stage, the sequential covering algorithm picks a new positive example that is not yet covered by the current Horn clauses, explains this new example, and formulates a new rule according to the procedure described above.
- A new instance is classified as negative if the current rules fail to predict that it is positive.
- This is in keeping with the standard negation-as-failure approach used in Horn clause inference systems such as PROLOG.

5.10. EXPLANATION BASED LEARNING (EBL)

- Explanation-Based Learning (EBL) is a principled method for exploiting available domain knowledge to improve supervised learning.

- Improvement can be in one or more dimension such as speed of learning, confidence of learning, accuracy of the learned concept, or a combination of these.
- The domain theory represents an expert's approximate knowledge of complex systematic world behavior. It may be imperfect and incomplete.
- In EBL, the domain theory is required to be much stronger; inferred properties are guaranteed.
- The interaction between domain knowledge and labeled training examples afforded by explanations.

A detailed analysis of individual training examples to determine how best to generalize from the specific.

The key properties are:

- Unlike inductive methods, this method produces justified general hypotheses by using prior knowledge to analyze individual examples.
- The explanation of how the example satisfies the target concept determines which example attributes are relevant: those mentioned by the explanation.
- The further analysis of the explanation, regressing the target concept to determine its weakest preimage with respect to the explanation, allows deriving more general constraints on the values of the relevant features.
- Each learned Horn clause corresponds to a sufficient condition for satisfying the target concept.
- The set of learned Horn clauses covers the positive training examples encountered by the learner, as well as other instances that share the same explanations.
- The generality of the learned Horn clauses will depend on the formulation of the domain theory and on the sequence in which training examples are considered.

PROLOG-EBG is one of the instance of explanation based learning. It has some disadvantages also:

- PROLOG-EBG assumes that the domain theory is correct and complete. If the domain theory is incorrect or incomplete, the resulting learned concept may also be incorrect.

Capabilities and limitations of EBL

EBL is seen from different perspectives:

EBL as theory-guided generalization of examples:

EBL uses its given domain theory to generalize rationally from examples, distinguishing the relevant example attributes from the irrelevant, allowing it to avoid the bounds on sample complexity that apply to purely inductive learning.

EBL as example-guided reformulation of theories:

The PROLOG-EBG algorithm can be viewed as a method for reformulating the domain theory into a more operational form. The original domain theory is reformulated by creating rules that

- ✓ follow deductively from the domain theory
- ✓ classify the observed training examples in a single inference step.

The learned rules can be seen as a reformulation of the domain theory into a set of special-case rules capable of classifying instances of the target concept in a single inference step.

EBL as restating what the learner already knows:

The learner may sometimes begin with full knowledge of the concept. If its initial domain theory is sufficient to explain any observed training examples, then it is also sufficient to predict their classification in advance. The difference between what one knows in principle and what one can efficiently compute in practice may be great. This kind of knowledge reformulation can be an important form of learning in such cases. This is also known as knowledge compilation, indicating that the transformation is an efficiency improving one that does not alter the correctness of the system's knowledge.

In general, given the

- ✓ goal in the form of some predicate calculus statement
- ✓ Situation Description or facts

- ✓ Domain Theory or inference rules
- ✓ Operability Criterion

Use problem solver to justify, using the rules, the goal in terms of the facts. Generalize the justification as much as possible. The operability criterion states which other terms can appear in the generalized result. This is EBL

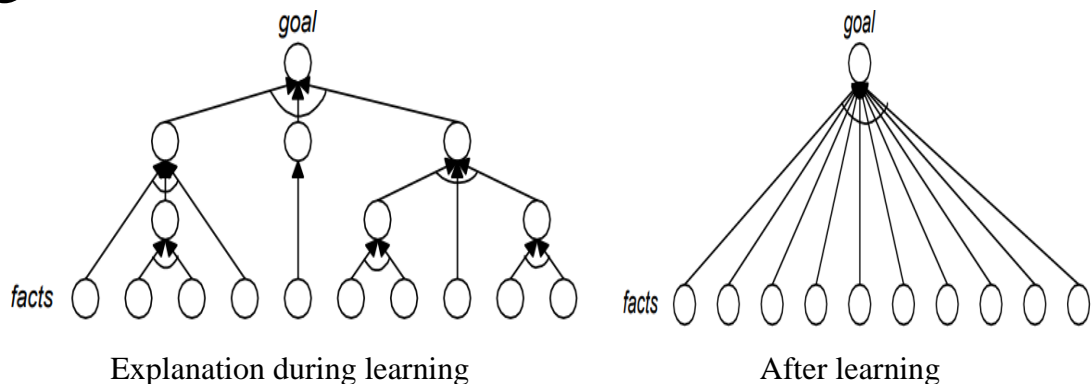


Fig. 5.7. Explanation Based Learning

- ✓ An explanation is an inter-connected collection of pieces of knowledge in terms of inference rules, rewrite rules, etc.
- ✓ These rules are connected using unification, as in Prolog.
- ✓ The generalization task is to compute the most general unifier that allows the knowledge pieces to be connected together as generally as possible.

5.10.1. Discovering new features

- PROLOG-EBG can formulate new features that are not explicit in the description of the training examples.
- These features may be needed to describe the general rule underlying the training example.
- This aspect of learning the features is similar in kind to the types of features represented by the hidden units of neural networks.

- But the PROLOG-EBG employs an analytical process to derive new features based on analysis of single training examples whereas neural networks use statistical learning approaches.
- The issue of automatically learning useful features to augment the instance representation is an important issue for machine learning.
- The analytical derivation of new features in explanation-based learning and the inductive derivation of new features in the hidden layer of neural networks provide two distinct approaches.
- Because they rely on different sources of information, it may be useful to explore new methods that combine both sources.

5.10.2. Deductive Learning

- PROLOG-EBG is deductive, rather than inductive learning process.
- It proceeds by calculating the weakest preimage of the explanation it produces a hypothesis h that follows deductively from the domain theory B , while covering the training data D .
- PROLOG-EBG outputs a hypothesis h that satisfies the following two constraints:

$$(\forall \langle x_i, f(x_i) \rangle \in D) (h \wedge x_i) \vdash f(x_i)$$

$$D \wedge B \vdash h$$

Where the training data D consists of a set of training examples in which x_i is the i^{th} training instance and $f(x_i)$ is its target value.

- The first constraint states that the hypothesis h correctly predicts the target value $f(x_i)$ for each instance x_i in the training data.
- The second constraint reduces the ambiguity faced by the learner when it must choose a hypothesis.
- The impact of the domain theory is to reduce the effective size of the hypothesis space and hence reduce the sample complexity of learning.
- In particular, PROLOG-EBG assumes the domain theory B entails the classifications of the instances in the training data:

$$(\forall \langle x_i, f(x_i) \rangle \in D) (B \wedge x_i) \vdash f(x_i)$$

- This constraint on the domain theory B assures that an explanation can be constructed for each positive example.

5.10.3. Inductive Bias in Explanation-Based Learning

- Inductive bias characterizes how the learner generalizes beyond the observed training examples.
- In PROLOG-EBG the output hypothesis follows deductively from $D \wedge B$.
- The domain theory B is a set of assertions which, together with the training examples, entail the output hypothesis.
- Given that predictions of the learner follow from this hypothesis h , it appears that the inductive bias is simply the domain theory B input to the learner.
- There can be more number of Horn clauses entailed by the domain theory.
- The PROLOG-EBG algorithm employs a sequential covering algorithm that continues to formulate additional Horn clauses until all positive training examples have been covered.
- Also, each individual Horn clause is the most general clause (weakest preimage) licensed by the explanation of the current training example.
- Among the sets of Horn clauses entailed by the domain theory, we can characterize the bias of PROLOG-EBG as a preference for small sets of maximally general Horn clauses.
- The greedy version of PROLOG-EBG is only a heuristic approximation to the exhaustive search algorithm that would be required to find the truly shortest set of maximally general Horn clauses.

Approximate inductive bias of PROLOG-EBG comprises of the domain theory B , along with a preference for small sets of maximally general Horn clauses.

- Any attempt to develop a general-purpose learning method must at minimum allow the inductive bias to vary with the learning problem at hand.
- To do this efficiently impart domain specific knowledge to the learner to generalize beyond the training data.

5.10.4. Knowledge Level Learning

- The hypothesis h output by PROLOG-EBG follows deductively from the domain theory B and training data D .
- By examining the PROLOG-EBG algorithm it is easy to see that h follows directly from B alone, independent of D .
- Lemma-Enumerator algorithm does this job.
- The algorithm enumerates all proof trees that conclude the target concept based on assertions in the domain theory B .
- It estimates the weakest preimage and constructs a Horn clause
- Unlike PROLOG-EBG, this algorithm considers the training data and enumerates all proof trees.
- The output will be a superset of the Horn clauses.
- The knowledge-level learning refers to the type of learning, in which the learned hypothesis entails predictions that go beyond those entailed by the domain theory.
- The set of all predictions entailed by a set of assertions Y is called as deductive closure of Y .
- The key distinction here is that in knowledge-level learning the deductive closure of B is a proper subset of the deductive closure of $B + h$.
- Determinations is another perspective of knowledge level learning.
- Determinations assert that some attribute of the instance is fully determined by certain other attributes, without specifying the exact nature of the dependence.

**5.11. USING PRIOR KNOWLEDGE TO AUGMENT SEARCH OPERATORS:
FIRST ORDER COMBINED LEARNER (FOCL) ALGORITHM**

FOCL algorithm is seen as an extension of FOIL algorithm.

5.11.1. First Order Inductive Learner (FOIL)

FOIL learns function-free Horn clauses, a subset of first-order predicate calculus.

Given positive and negative examples of some concept and a set of background-knowledge predicates, FOIL inductively generates a logical concept definition or rule for the concept. The induced rule allows negated predicates but it must not involve any constants or function symbols.

FOIL can be extended to use a variety of types of background knowledge to increase the class of problems that can be solved, to decrease the hypothesis space explored, and to increase the accuracy of learned rules. FOIL inductively generates a logical concept definition or rule for the concept.

Example:

Concept: grandfather(X, Y)

Given: father(X, Y) and parent (X, Y)

Current clause: Grandfather(X, Y) ← Parent(X, Z)

This clause can be extended by conjoining the body with any of the literals father(X, X), father(Y, Z), father(U, Y), parent(Y, Z), parent(Y, Y)

Algorithm for FOIL

Let Pred be the predicate to be learned

Let Pos be the positive examples

Until Pos is empty do:

 Let Neg be the negative examples

 Set Body to empty

 Call LearnClauseBody

 Add Pred ← Body to the rule

 Remove from Pos all examples that satisfy the Body

Procedure LearnClauseBody

 Until Neg is empty do:

 Choose a literal L

 Conjoin L to Body

 Remove from Neg examples that do not satisfy L

5.11.2. First Order Combined Learner (FOCL)

FOCL extends FOIL in a variety of ways. Each of these extensions affects only how FOCL selects literals to test while extending a clause under construction. These extensions allow FOCL to use domain knowledge to guide the learning process. The following are the extensions:

- FOCL to use constraints to limit the search space.
- FOCL to use defined predicates (i.e., predicates defined by a rule instead of a collection of examples) in a manner similar to the extensionally defined predicates in FOIL. A collection of intensionally defined predicates is analogous to the domain theory of EBL.
- FOCL to accept as input a partial, possibly incorrect rule that is an initial approximation of the predicate to be learned. If this rule is defined in terms of extensionally defined predicates, it is analogous to a partial concept definition constructed by an incremental inductive learning system. If this rule is defined in terms of intensionally defined predicates, it is analogous to the target concept of EBL.

Similarities between FOIL and FOCL:

- FOCL is an extension of the purely inductive FOIL system.
- Both FOIL and FOCL learn a set of first-order Horn clauses to cover the observed training examples.
- Both systems employ a sequential covering algorithm that learns a single Horn clause, removes the positive examples covered by this new Horn clause, and then iterates this procedure over the remaining training examples.
- In both systems, each new Horn clause is created by performing a general-to-specific search, beginning with the most general possible Horn clause. Several candidate specializations of the current clause are then generated, and the specialization with greatest information gain relative to the training examples is chosen. This process is iterated, generating further candidate specializations and selecting the best, until a Hornclause with satisfactory performance is obtained.

Differences between FOIL and FOCL algorithm

FOIL	FOCL
FOIL generates each candidate specialization by adding a single new literal to the clause preconditions.	FOCL uses this same method for producing candidate specializations, but also generates additional specializations based on the domain theory.

5.11.3. Hypothesis search in FOCL

- To learn a single rule, FOCL searches from general to increasingly specific hypotheses.
- Two types of operators generate specializations of the current hypothesis.
 - ✓ One type adds a single new literal.
 - ✓ Next type of operator specializes the rule by adding a set of literals that constitute logically sufficient conditions for the target concept, according to the domain theory.
- FOCL selects among all these candidate specializations, based on their performance over the data.
- Therefore, imperfect domain theories will impact the hypothesis only if the evidence supports the theory.

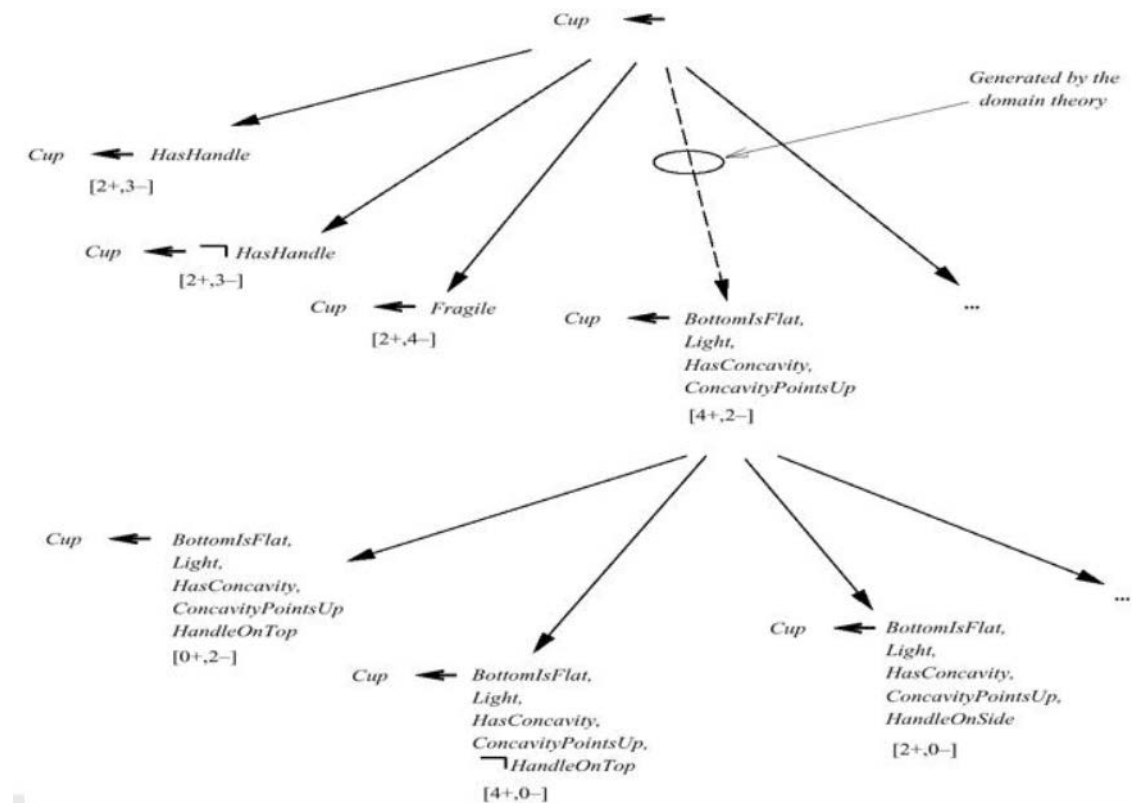


Fig. 5.8. Hypothesis search space in FOCL

Example

The Fig 5.8 refers to 12 attributes that describe the training examples (e.g., HasHandle, HandleOnTop). Literals based on these 12 attributes are thus considered operational.

FOCL expands its current hypothesis h using the following two operators:

1. For each operational literal that is not part of h , create a specialization of h by adding this single literal to the preconditions.
 2. Create an operational, logically sufficient condition for the target concept according to the domain theory. Add this set of literals to the current preconditions of h . Finally, prune the preconditions of h by removing any literals that are unnecessary according to the training data.
- FOCL selects one of the domain theory clauses whose head (postcondition) matches the target concept.
 - If there are several such clauses, it selects the clause whose body (preconditions) have the highest information gain relative to the training examples of the target concept.
 - There is only one such clause:

Cup \leftarrow Stable, Lifiable, OpenVessel

- The preconditions of the selected clause form a logically sufficient condition for the target concept.
- Each non operational literal in these sufficient conditions is now replaced, again using the domain theory and substituting clause preconditions for clause post conditions.
- This process of "unfolding" the domain theory continues until the sufficient conditions have been restated in terms of operational literals.
- If there are several alternative domain theory clauses that produce different results, then the one with the greatest information gain is greedily selected at each step of the unfolding process.
- The final operational sufficient condition given the data and domain theory for the above example is

BottomIsFlat , HasHandle, Light, HasConcavity , ConcavityPointsUp

- For each literal in the expression, the literal is removed unless its removal reduces classification accuracy over the training examples.
- The final pruned, operational, sufficient conditions are

BottomIsFlat , Light, HasConcavity , ConcavityPointsUp

- Once candidate specializations of the current hypothesis have been generated, using both of the two operations above, the candidate with highest information gain is selected.
- The search then proceeds by considering further specializations of the theory-suggested preconditions, thereby allowing the inductive component of learning to refine the preconditions derived from the domain theory.
- FOCL algorithm learns the Horn clauses of the form,

$$c \leftarrow o_i \wedge o_b \wedge o_f$$

c- target concept

o_i - initial conjunction of operational literals

o_b - conjunction of operational literals added in a single step based on the domain theory

o_f - final conjunction of operational literals added one at a time by the first syntactic operator.

- Any of these three sets of literals may be empty.

5.12. REINFORCEMENT LEARNING

In reinforcement learning, the learner is a decision-making agent that takes actions in an environment and receives reward for its actions in trying to solve a problem. After a set of trial-and error runs, it should learn the best policy, which is the sequence of actions that maximize the total reward.

- Reinforcement learning is the training of machine learning models to make a sequence of decisions.
- The agent learns to achieve a goal in an uncertain, potentially complex environment.
- In reinforcement learning, an artificial intelligence faces a game-like situation.
- The computer employs trial and error to come up with a solution to the problem.
- To get the machine to do what the programmer wants, the artificial intelligence gets either rewards or penalties for the actions it performs.
- Its goal is to maximize the total reward.
- Although the designer sets the reward policy, the rules of the game, do not offer any hint to the model or suggestions for how to solve the game.
- The model has to figure out how to perform the task to maximize the reward, starting from totally random trials and finishing with sophisticated tactics and superhuman skills.
- By leveraging the power of search and many trials, reinforcement learning is currently the most effective way to hint machine's creativity.
- In contrast to human beings, artificial intelligence can gather experience from thousands of parallel gameplays if a reinforcement learning algorithm is run on a sufficiently powerful computer infrastructure.
- The key distinguishing factor of reinforcement learning is how the agent is trained. Instead of inspecting the data provided, the model interacts with the environment, seeking ways to maximize the reward. In the case of deep reinforcement learning, a neural network is in charge of storing the experiences and thus improves the way the task is performed.

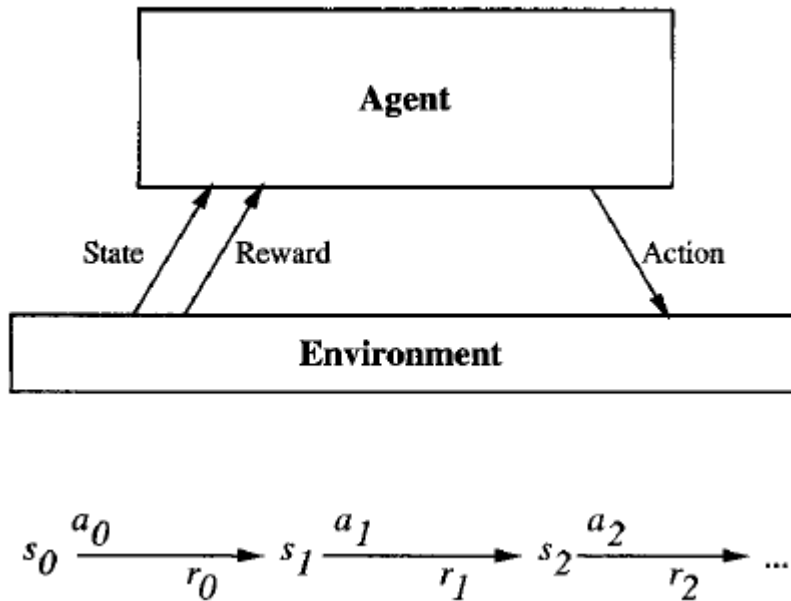


Fig. 5.9. Agent interacting with the environment

- The agent exists in an environment described by some set of possible states S .
- It can perform any of a set of possible actions A .
- Each time it performs an action a , in some state s_t the agent receives a real-valued reward r , that indicates the immediate value of this state-action transition. This produces a sequence of states s_i , actions a_i , and immediate rewards r_i .

- The agent's task is to learn a control policy,

$$\pi : S \rightarrow A$$

that maximizes the expected sum of these rewards, with future rewards discounted exponentially by their delay.

Approximation tasks vs Reinforcement learning

The following aspects of reinforcement learning makes it distinct from approximation algorithms:

- **Delayed reward:**
 - ✓ The task of the agent is to learn a target function n that maps from the current state s to the optimal action $a = \pi(s)$.
 - ✓ Training information is not available in reinforcement learning.
 - ✓ The trainer provides only a sequence of immediate reward values as the agent executes its sequence of actions.
 - ✓ The agent, therefore, faces the problem of temporal credit assignment: determining which of the actions in its sequence are to be credited with producing the eventual rewards.
- **Exploration:**
 - ✓ In reinforcement learning, the agent influences the distribution of training examples by the action sequence it chooses.
 - ✓ The learner faces a tradeoff in choosing whether to favor exploration of unknown states and actions or exploitation of states and actions that it has already learned will yield high reward.
- **Partially observable states:**
 - ✓ Deploying sensors to observe the environment provide only partial information.
 - ✓ It may be necessary for the agent to consider its previous observations together with its current sensor data when choosing actions, and the best policy may be one that chooses actions specifically to improve the observability of the environment.
- **Life-long learning:**
 - ✓ Unlike isolated function approximation tasks, robot or reinforcement learning often requires that the robot learn several related tasks within the same environment, using the same sensors.

Challenges with reinforcement learning

- Preparing the simulation environment, which is highly dependant on the task to be performed.

- Transferring the model out of the training environment and into to the real world is where things get tricky.
- Scaling and tweaking the neural network controlling the agent is another challenge.
- There is no way to communicate with the network other than through the system of rewards and penalties.
- This in particular may lead to catastrophic forgetting, where acquiring new knowledge causes some of the old to be erased from the network.
- The agent performs the task as it is, but not in the optimal or required way.
- There are agents that will optimize the prize without performing the task it was designed for.

5.13. LEARNING TASK IN REINFORCEMENT LEARNING

The reinforcement learning problems can be formalised as Markov Decision Process (MDP). MDPs are a straightforward framing of the problem of learning from interaction to achieve a goal.

In MDP, the agent and the environment interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent.

- In a Markov decision process (MDP) the agent can perceive a set S of distinct states of its environment and has a set A of actions that it can perform.
- The mathematical framework for defining a solution in reinforcement learning scenario is called Markov Decision Process. This can be designed as:
 - Set of states, S
 - Set of actions, A
 - Reward function, R
 - Policy, π
 - Value, V

- At each discrete time step t , the agent senses the current state s_t , chooses a current action a_t , and performs it.
- The rewards are notated as $r(s_t, a_t)$ and the next state as $s_{t+1} = \delta(s_t, a_t)$. The δ and r values will not be known to the agent.
- The set of actions we took define our policy (π) and the rewards we get in return defines our value (V). Our task here is to maximize our rewards by choosing the correct policy.

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

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

- The cumulative value $V^\pi(s_t)$ which is achieved by following an arbitrary policy π from an arbitrary initial state is given in the above equation. This is also known as discounted cumulative reward achieved by policy π from initial state s .
- A Markov Reward Process or an MRP is a Markov process with value judgment, saying how much reward accumulated through some particular sequence that are sampled.
- The following are some of the types of rewards:

1. **Discounted cumulative reward**

2. **Finite horizon reward** which considers the undiscounted sum of rewards over a finite number h of steps.

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

3. **Average reward** which considers the average reward per time step over the entire lifetime of the agent.

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

Optimal Policy:

It is the best action to take at each state, for maximum rewards over time. Two factors are essential for determining optimal policy:

- ✓ A way to determine the value of a state in MDP.
- ✓ An estimated value of an action taken at a particular state.

The optimal policy denoted by π^* , maximises $V^\pi(S)$ for all states S .

$$\pi^* \cong \operatorname{argmax} V^\pi(S), (\forall s)$$

5.14. Q-LEARNING (QUALITY LEARNING)

Q-learning is an off policy reinforcement learning algorithm that seeks to find the best action to take given the current state.

Off-policy algorithm means the learning function learns from actions that are outside the current policy, like taking random actions, where a policy or set of rules are not available. Q-Learning learns the optimal policy even when actions are selected according to a more exploratory or even random policy. The optimal policy is learnt from the following equation:

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

V^* - sum of discounted future rewards over the infinite future

$r(s, a)$ - reward of state s over an action a

$\delta(s, a)$ - resulting state after applying action a to state s

γ - discount rate

The agent can acquire the optimal policy by learning V^* , with knowledge of the immediate reward function r and the state transition function δ . After the agent know the functions r and δ used by the environment to respond to its actions, it can calculate the optimal policy $\pi^*(s)$.

In many practical problems, it is difficult for the agent or its human programmer to predict in advance the exact outcome of applying an arbitrary action to an arbitrary state. Q-function is a natural solution in such situations.

5.14.1. Q-function

- The Q function states what the value of a state s and an action a under the policy π is.

- It is denoted as $Q(s, a)$ whose value is the maximum discounted cumulative reward that can be achieved starting from state s and applying action a as the first action.
- The value of Q is the reward received immediately upon executing action a from state s , plus the value (discounted by γ) of following the optimal policy.

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

- This equation can be rewritten as,

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

since the RHS on the optimal policy $\pi^*(s)$ is the maximized reward at state s .

- This substitution gains significance since, the Q -function will be able to select optimal actions without knowing the values of r and δ .
- The agent can choose the optimal action without ever conducting a look ahead search to explicitly consider what state results from the action.
- The value of Q for the current state and action summarizes in a single number all the information needed to determine the discounted cumulative reward that will be gained in the future if action a is selected in state s .

5.14.2. Q-learning Algorithm

- Learning the Q function corresponds to learning the optimal policy.
- Finding a way to estimate training values for Q , given only a sequence of immediate rewards r spread out over time is done through iterative approximation.

- The relationship between Q and V^* is given as:

$$V^*(s) = \max_{a'} Q(s, a')$$

- This makes the Q -function to be written as,

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

- After applying the Q function recursively, update to the Q -function ($\hat{Q}(s, a)$) is ruled by,

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

$\hat{Q}(s, a)$ - estimate of actual Q function

$\hat{Q}(s', a')$ - Q function value for new state (s') and action (a')

Q-Table is a lookup table used to calculate the maximum expected future rewards for action at each state. This table will help in achieving the best action at each state.

- The Q-table can be initially filled with random values.
- The agent repeatedly observes its current state s , chooses some action a , executes this action, then observes the resulting reward $r = r(s, a)$ and the new state $s' = \delta(s, a)$.
- It then updates the table entry for $\hat{Q}(s, a)$ following each such transition, according to the rule

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

Algorithm for Q-Learning

For each s , initialize the table entry $\hat{Q}(s, a)$ to zero

Observe the current state s

Do

Select an action a and execute it

Receive an immediate reward (r)

Observe the new state s'

Update the entry $\hat{Q}(s, a)$ according to the rule

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

$s \leftarrow s'$

5.14.3. Convergence

- The system for Q-learning is a deterministic MDP.
- The immediate reward values are bounded. There is an upper limit.

- The agent selects actions in such a fashion that it visits every possible state-action pair infinitely often.
- The conditions are also restrictive in that they require the agent to visit every distinct state-action transition infinitely often.
- The table entry with the largest error must have its error reduced by a factor of γ whenever it is updated.
- The reason is that its new value depends only in part on error-prone Q estimates, with the remainder depending on the error-free observed immediate reward r .
- The agent in state s to select the action a that maximizes $\hat{Q}(s, a)$ hereby exploiting its current approximation Q.
- This has a chance to over commit to actions that are found during early training to have high Q values, while failing to explore other actions that have even higher values. Thus the update may sometimes be stuck in local maxima.
- So, a probabilistic approach is applied: Actions with higher Q values are assigned higher probabilities, but every action is assigned a nonzero probability.

$$P(a_i | s) = \frac{k \hat{Q}(s, a_i)}{\sum_j k \hat{Q}(s, a_j)}$$

- Larger values of k will assign higher probabilities to actions with above average Q, causing the agent to exploit what it has learned and seek actions it believes will maximize its reward.
- Smaller values of k will allow higher probabilities for other actions, leading the agent to explore actions that do not currently have high Q values.
- Sometimes, k is varied with the number of iterations so that the agent favors exploration during early stages of learning, then gradually shifts toward a strategy of exploitation.

5.14.4. Sequence Update in Q Learning

- Q learning need not train on optimal action sequences in order to converge to the optimal policy.

- It can learn the Q function and the optimal policy while training from actions chosen completely at random at each step, as long as the resulting training sequence visits every state-action transition infinitely often.
- Changing the sequence of training example transitions in order to improve training efficiency without endangering final convergence.
- For each training episode, the agent is placed in a random initial state and is allowed to perform actions and to update its Q table until it reaches the absorbing goal state.
- A new training episode begins by removing the agent from the goal state and placing it at a new random initial state.
- The initial values of the table from zero will be changed to the final transition into the goal state.
- If the agent happens to follow the same sequence of actions from the same random initial state in its second full episode, then a second table entry would be made nonzero, and so on.
- When identical episodes are run in this fashion, the frontier of nonzero Q values will creep backward from the goal state at the rate of one new state-action transition per episode.
- Considering reverse chronological order for each episode, then each updates will also appear in reverse order.
- This training process will clearly converge in fewer iterations, although it requires that the agent use more memory to store the entire episode before beginning the training for that episode.
- To improve the convergence rate, store the past state-action transitions, along with the immediate reward that was received, and retrain on them periodically.
- The degree of replaying old transitions versus obtain new ones from the environment depends on the relative costs of these two operations in the specific problem domain.
- This difference can be very significant given that Q learning can often require thousands of training iterations to converge.

5.15. TEMPORAL DIFFERENCE LEARNING

Temporal difference learning is an approach to learning how to predict a quantity that depends on future values of a given signal. It uses changes, or differences, in predictions over successive time steps to drive the learning process.

- The prediction at any given time step is updated to bring it closer to the prediction of the same quantity at the next time step.
- It is a supervised learning process in which the training signal for a prediction is a future prediction.
- These algorithms are used in reinforcement learning to predict a measure of the total amount of reward expected over the future, but they can be used to predict other quantities as well.
- Q learning can be considered as a special case of a general class of temporal difference algorithms that learn by reducing discrepancies between estimates made by the agent at different times.

- The rule that assures convergence in Q-learning is:

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

Where

$$\alpha_n = \frac{1}{1 + \text{visits}_n(s, a)}$$

Visits(s, a)- total number of times this state-action pair has been visited up to and including the nth iteration.

α -learning rate that takes any non zero value between 0 and 1.

- The above equation reduces the difference between the estimated Q values of a state and its immediate successor. Same could be used to reduce discrepancies between this state and more distant descendants or ancestors.
- Consider the notion of calculating training values of $\hat{Q}(s_t, a_t)$ in terms of $\hat{Q}(s_{t+1}, a_{t+1})$. $Q^{(1)}(s_t, a_t)$ be the training value found with one look ahead step.

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

- Alternatively, the same value can also be computed two steps ahead.

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

- Extending this,

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

- When a constant value for γ is used, $0 \leq \gamma < 1$, the equation becomes,

$$Q^\lambda(s_p, a_t) \equiv (1 - \lambda) [Q^{(1)}(s_p, a_t) + \lambda Q^{(2)}(s_p, a_t) + \lambda^2 Q^{(3)}(s_p, a_t) + \dots]$$

- The recursive version of the same is given as,

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

SUMMARY OF NOTATION

(a ,b]: Brackets are the form [,] , (,and) are used to represent intervals, where square brackets represent intervals including the boundary and round parentheses represent intervals excluding the boundary.

For example, (1,3] represent the interval $1 < x \leq 3$.

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

$\prod_{i=1}^n$: The product $x_1, x_2 \dots x_n$

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

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

$\operatorname{argmax}_{x \in (1, 2, -3)} f(x)$: The value of x that maximizes $f(x)$. For example,

$\operatorname{argmax}_{x \in (1, 2, -3)} x^2 = -3$

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

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

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

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

μ : The mean of a probability distribution.

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

C: Class of possible target function.

D: The training data.

D: A probability distribution over the instance space.

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

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

Error: The error in the discrete valued hypothesis or prediction.

H : Hypothesis space.

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

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

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

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

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

\mathbb{R} : The set of real numbers.

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

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

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

X : Instance space.