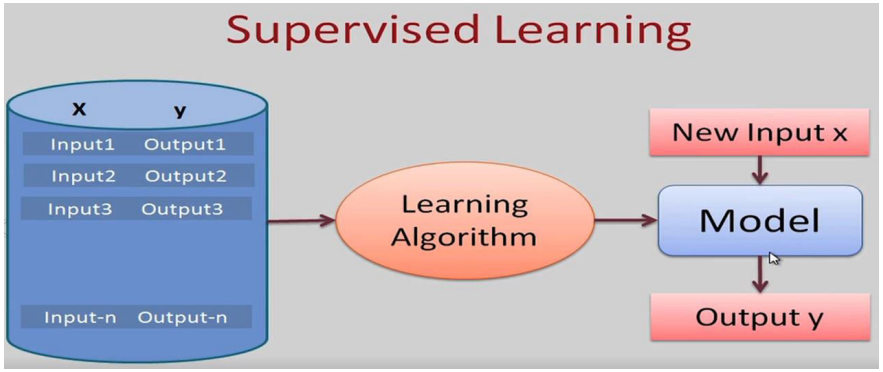
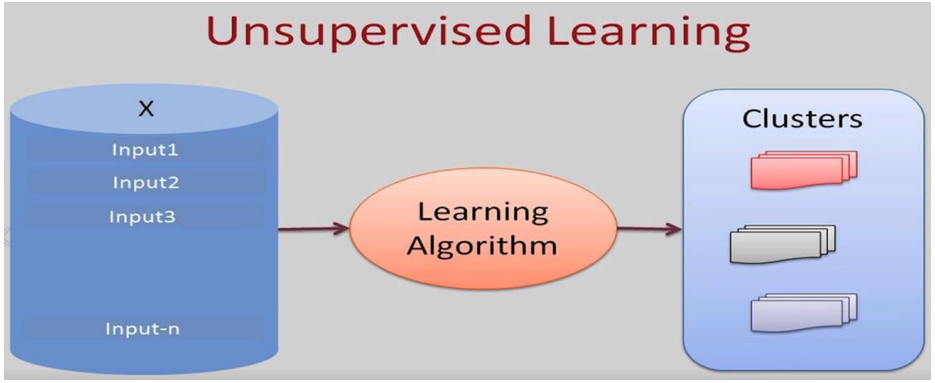
Learning Types

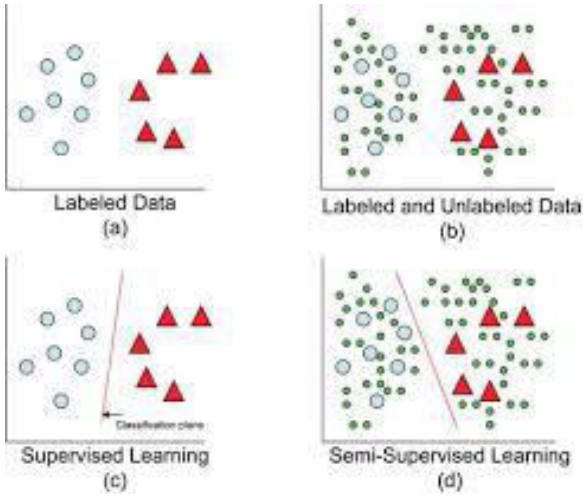
Broad types of machine learning

- Supervised Learning
  - X,y (pre-classified training examples)
  - Given an observation x, what is the best label for y?
- Unsupervised learning
  - X
  - Given a set of x's, cluster or summarize them
- Semi-supervised Learning
- Reinforcement Learning
  - Determine what to do based on rewards and punishments.

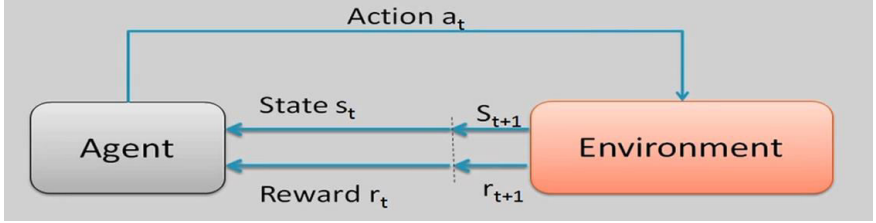




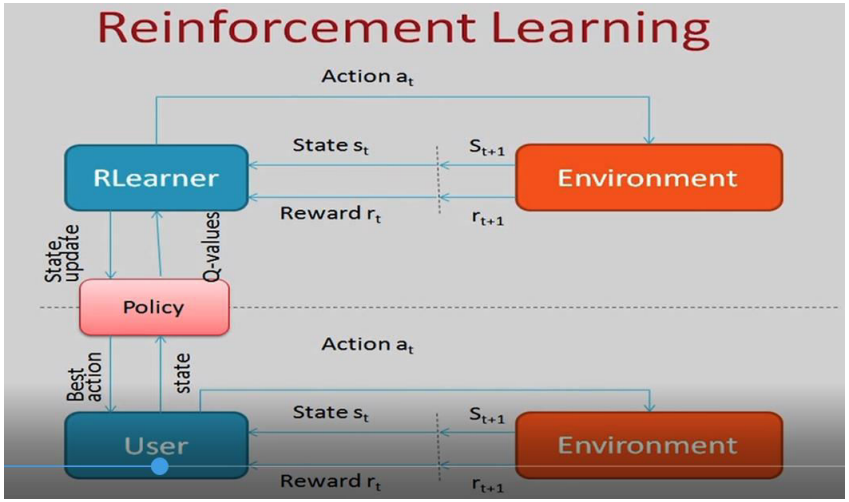
Semi-Supervised Learning



Reinforcement Learning



Reinforcement Learning



Supervised Learning

Given:

- a set of input features $X_1, \ldots, X_n$
- A target feature Y
- a set of training examples where the values for the input features and the target features are given for each example
- a new example, where only the values for the input features are given

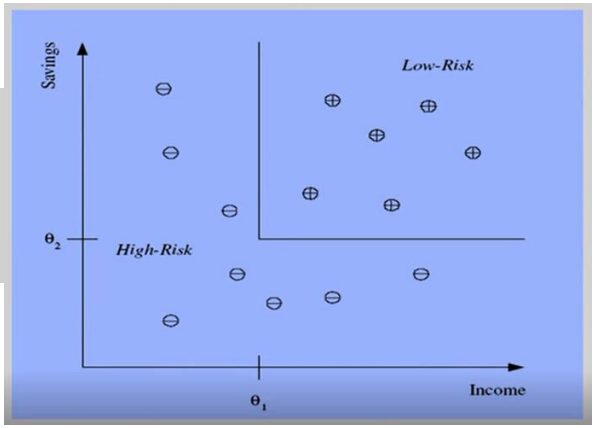
Predict the values for the target features for the new example.

- classification when Y is discrete
- regression when **Y** is continuous

Classification

Example: Credit scoring

Differentiating between low-risk and high-risk customers from their *income* and *savings*



Discriminant: IF *income* > θ_1 AND *savings* > θ_2

THEN low-risk ELSE high-risk

Regression

Example: Price of a used car

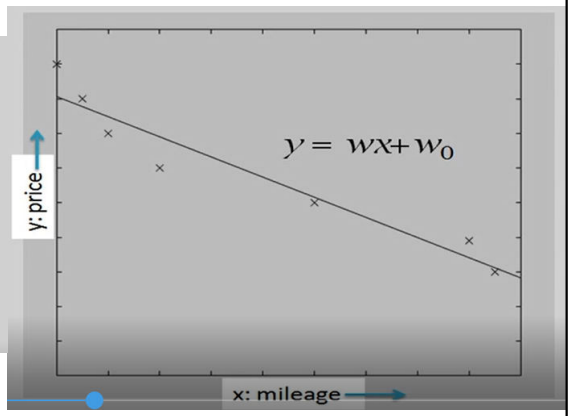
x : car attributes

y : price

$y = g(x, \theta)$

$g(\ )$ model,

θ parameters



Features

- Often, the individual observations are analyzed into a set of quantifiable properties which are called features. May be
  - categorical (e.g. "A", "B", "AB" or "O", for blood type)
  - ordinal (e.g. "large", "medium" or "small")
  - integer-valued (e.g. the number of words in a text)
  - real-valued (e.g. height)

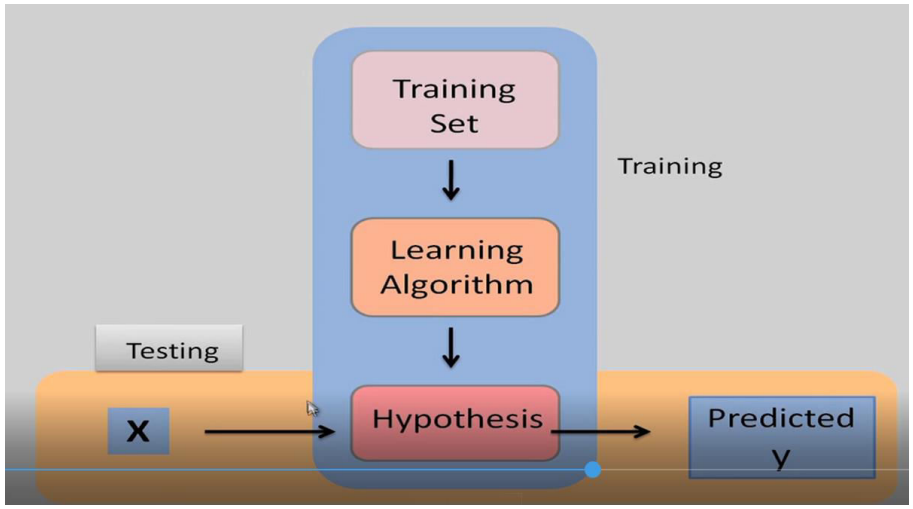
Example Data

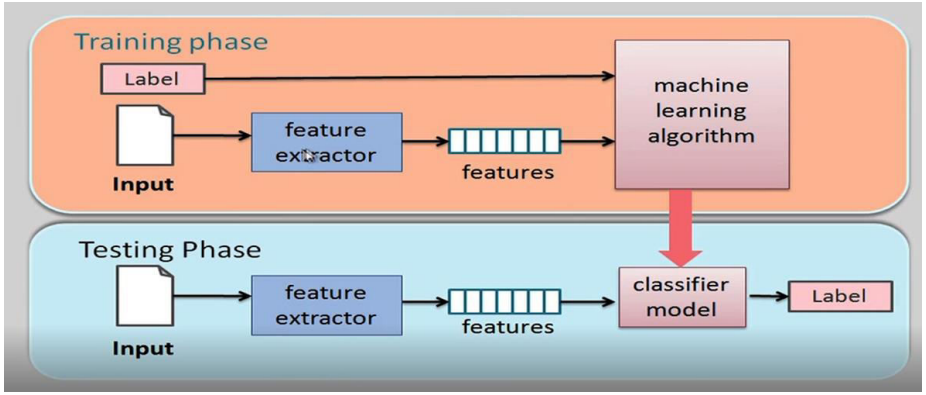
Training Examples:

	Action	Author	Thread	Length	Where
e1	skips	known	new	long	Home
e2	reads	unknown	new	short	Work
e3	skips	unknown	old	long	Work
e4	skips	known	old	long	home
e5	reads	known	new	short	home
e6	skips	known	old	long	work

New Examples:

e7	???	known	new	short	work
e8	???	unknown	new	short	work





Classification learning

- Task *T:*
  - input: a set of *instances* $d_1,\ldots,d_n$
    - an instance has a set of *features*
    - we can represent an instance as a vector $\mathbf{d}=\langle x_1,\ldots,x_n\rangle$
  - output: a set of *predictions* $\hat{y}_1,\ldots,\hat{y}_n$
    - one of a fixed set of constant values:
      - *{+1,-1}* or *{cancer, healthy},* or *{rose, hibiscus, jasmine, ...},* or ...
- Performance metric *P:*
- Experience *E:*

Classification Learning

Task	Instance	Labels
medical diagnosis	**patient record:** blood pressure diastolic, blood pressure systolic, age, sex (0 or 1), BMI, cholesterol	{-1,+1} = low, high risk of heart disease
finding company names in text	**a word in context:** capitalized (0,1), word-after-this-equals-Inc, bigram-before-this-equals-acquired-by, …	{first,later,outside} = first word in name, second or later word in name, not in a name
brain-human-interface	**brain state**: neural activity over the last 100ms of 96 neurons	{n,s,e,w,ne,se,nw,sw} = direction you intend to move the cursor

Classification learning

- Task *T:*
  - input: a set of *instances* $d_1,\ldots,d_n$
  - output: a set of *predictions* $\hat{y}_1,\ldots,\hat{y}_n$
- Performance metric *P:*
  - Prob (wrong prediction) on examples from *D*
- Experience *E:*
  - a set of *labeled examples (x,y)* where *y* is the true label for *x*
  - ideally, examples should be *sampled* from some fixed distribution *D*

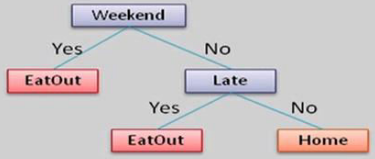
we care about performance on the *distribution*, not the *training data*

Classification Learning

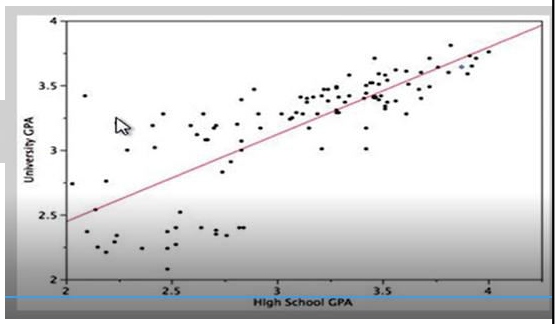
Task	Instance	Labels	Getting data
medical diagnosis	**patient record:** lab readings	risk of heart disease	wait and look for heart disease
finding company names in text	**a word in context:** capitalized, nearby words, ...	{first,later,outside}	text with company names highlighted (say by hand)
brain-human-interface	**brain state**: neural activity over the last 100ms of 96 neurons	{n,s,e,w,ne,se,nw,sw}	recordings of someone doing known tasks (so direction can be inferred)
image recognition	**image:** pixels	no house, house	hand-labeled images

Representations

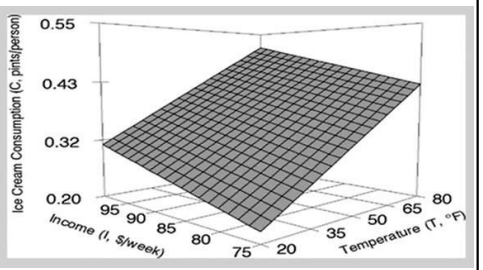
1. Decision Tree



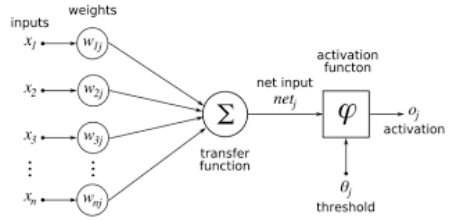
2. Linear function



3. Multivariate linear function



4. Single layer perceptron



Hypothesis Space

- One way to think about a supervised learning machine is as a device that explores a "hypothesis space".
  - Each setting of the parameters in the machine is a different hypothesis about the function that maps input vectors to output vectors.

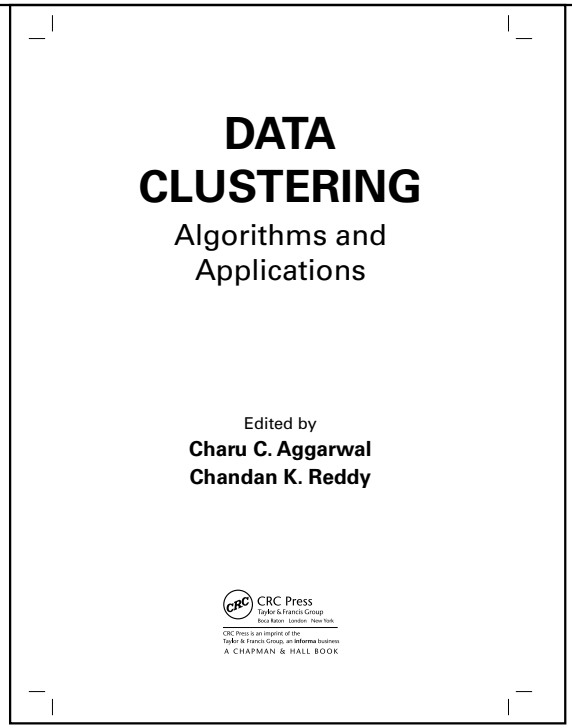
Terminology

- **Features:** The number of features or distinct traits that can be used to describe each item in a quantitative manner.
- **Feature vector:** n-dimensional vector of numerical features that represent some object
- **Instance Space X:** Set of all possible objects describable by features.
- **Example (x,y):** Instance x with label y=f(x).

Terminology

- **Concept c:** Subset of objects from X (c is unknown).
- **Target Function f:** Maps each instance $x \in X$ to target label $y \in Y$
- **Example (x,y):** Instance x with label y=f(x).
- **Training Data S:** Collection of examples observed by learning algorithm.

Used to discover potentially predictive relationships



CRC Press
Taylor & Francis Group
6000 Broken Sound Parkway NW, Suite 300
Boca Raton, FL 33487-2742

Printed on acid-free paper
Version Date: 20130508

International Standard Book Number-13: 978-1-4665-5821-2 (Hardback)

Library of Congress Cataloging-in-Publication Data

Data clustering : algorithms and applications / [edited by] Charu C. Aggarwal, Chandan K. Reddy.
pages cm. -- (Chapman & Hall/CRC data mining and knowledge discovery series)
Includes bibliographical references and index.
ISBN 978-1-4665-5821-2 (hardback)
1. Document clustering. 2. Cluster analysis. 3. Data mining. 4. Machine theory. 5. File organization (Computer science) I. Aggarwal, Charu C., editor of compilation. II. Reddy, Chandan K., 1980- editor of compilation.

QA278.D2294 2014
519.5'35--dc23 2013008698

Visit the Taylor & Francis Web site at
http://www.taylorandfrancis.com

and the CRC Press Web site at
http://www.crcpress.com

Contents

Preface

The problem of clustering is perhaps one of the most widely studied in the data mining and machine learning communities. This problem has been studied by researchers from several disciplines over five decades. Applications of clustering include a wide variety of problem domains such as text, multimedia, social networks, and biological data. Furthermore, the problem may be encountered in a number of different scenarios such as streaming or uncertain data. Clustering is a rather diverse topic, and the underlying algorithms depend greatly on the data domain and problem scenario.

Therefore, this book will focus on three primary aspects of data clustering. The first set of chapters will focus on the core methods for data clustering. These include methods such as probabilistic clustering, density-based clustering, grid-based clustering, and spectral clustering. The second set of chapters will focus on different problem domains and scenarios such as multimedia data, text data, biological data, categorical data, network data, data streams and uncertain data. The third set of chapters will focus on different detailed insights from the clustering process, because of the subjectivity of the clustering process, and the many different ways in which the same data set can be clustered. How do we know that a particular clustering is good or that it solves the needs of the application? There are numerous ways in which these issues can be explored. The exploration could be through interactive visualization and human interaction, external knowledge-based supervision, explicitly examining the multiple solutions in order to evaluate different possibilities, combining the multiple solutions in order to create more robust ensembles, or trying to judge the quality of different solutions with the use of different validation criteria.

The clustering problem has been addressed by a number of different communities such as pattern recognition, databases, data mining and machine learning. In some cases, the work by the different communities tends to be fragmented and has not been addressed in a unified way. This book will make a conscious effort to address the work of the different communities in a unified way. The book will start off with an overview of the basic methods in data clustering, and then discuss progressively more refined and complex methods for data clustering. Special attention will also be paid to more recent problem domains such as graphs and social networks.

The chapters in the book will be divided into three types:

- **Method Chapters:** These chapters discuss the *key techniques* which are commonly used for clustering such as feature selection, agglomerative clustering, partitional clustering, density-based clustering, probabilistic clustering, grid-based clustering, spectral clustering, and non-negative matrix factorization.
- **Domain Chapters:** These chapters discuss the specific methods used for different *domains* of data such as categorical data, text data, multimedia data, graph data, biological data, stream data, uncertain data, time series clustering, high-dimensional clustering, and big data. Many of these chapters can also be considered application chapters, because they explore the specific characteristics of the problem in a particular domain.
- **Variations and Insights:** These chapters discuss the *key variations* on the clustering process such as semi-supervised clustering, interactive clustering, multi-view clustering, cluster ensembles, and cluster validation. Such methods are typically used in order to obtain detailed insights from the clustering process, and also to explore different possibilities on the clustering process through either supervision, human intervention, or through automated generation

of alternative clusters. The methods for cluster validation also provide an idea of the quality of the underlying clusters.

This book is designed to be comprehensive in its coverage of the entire area of clustering, and it is hoped that it will serve as a knowledgeable compendium to students and researchers.

Editor Biographies

Charu C. Aggarwal is a Research Scientist at the IBM T. J. Watson Research Center in Yorktown Heights, New York. He completed his B.S. from IIT Kanpur in 1993 and his Ph.D. from Massachusetts Institute of Technology in 1996. His research interest during his Ph.D. years was in combinatorial optimization (network flow algorithms), and his thesis advisor was Professor James B. Orlin. He has since worked in the field of performance analysis, databases, and data mining. He has published over 200 papers in refereed conferences and journals, and has applied for or been granted over 80 patents. He is author or editor of nine books, including this one. Because of the commercial value of the above-mentioned patents, he has received several invention achievement awards and has thrice been designated a Master Inventor at IBM. He is a recipient of an IBM Corporate Award (2003) for his work on bioterrorist threat detection in data streams, a recipient of the IBM Outstanding Innovation Award (2008) for his scientific contributions to privacy technology, and a recipient of an IBM Research Division Award (2008) for his scientific contributions to data stream research. He has served on the program committees of most major database/data mining conferences, and served as program vice-chairs of the SIAM Conference on Data Mining (2007), the IEEE ICDM Conference (2007), the WWW Conference (2009), and the IEEE ICDM Conference (2009). He served as an associate editor of the *IEEE Transactions on Knowledge and Data Engineering Journal* from 2004 to 2008. He is an associate editor of the *ACM TKDD Journal*, an action editor of the *Data Mining and Knowledge Discovery Journal*, an associate editor of *ACM SIGKDD Explorations*, and an associate editor of the *Knowledge and Information Systems Journal*. He is a fellow of the IEEE for "*contributions to knowledge discovery and data mining techniques*", and a life-member of the ACM.

Chandan K. Reddy is an Assistant Professor in the Department of Computer Science at Wayne State University. He received his Ph.D. from Cornell University and M.S. from Michigan State University. His primary research interests are in the areas of data mining and machine learning with applications to healthcare, bioinformatics, and social network analysis. His research is funded by the National Science Foundation, the National Institutes of Health, Department of Transportation, and the Susan G. Komen for the Cure Foundation. He has published over 40 peer-reviewed articles in leading conferences and journals. He received the Best Application Paper Award at the ACM SIGKDD conference in 2010 and was a finalist of the INFORMS Franz Edelman Award Competition in 2011. He is a member of IEEE, ACM, and SIAM.

Contributors

Ayan Acharya
University of Texas
Austin, Texas

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, New York

Amrudin Agovic
Reliancy, LLC
Saint Louis Park, Minnesota

Mohammad Al Hasan
Indiana University - Purdue University
Indianapolis, Indiana

Salem Alelyani
Arizona State University
Tempe, Arizona

David C. Anastasiu
University of Minnesota
Minneapolis, Minnesota

Bill Andreopoulos
Lawrence Berkeley National Laboratory
Berkeley, California

James Bailey
The University of Melbourne
Melbourne, Australia

Arindam Banerjee
University of Minnesota
Minneapolis, Minnesota

Sandra Batista
Duke University
Durham, North Carolina

Shiyu Chang
University of Illinois at Urbana-Champaign
Urbana, Illinois

Wei Cheng
University of North Carolina
Chapel Hill, North Carolina

Hongbo Deng
University of Illinois at Urbana-Champaign
Urbana, Illinois

Cha-charis Ding
University of Texas
Arlington, Texas

Martin Ester
Simon Fraser University
British Columbia, Canada

S M Faisal
The Ohio State University
Columbus, Ohio

Joydeep Ghosh
University of Texas
Austin, Texas

Dimitrios Gunopulos
University of Athens
Athens, Greece

Jiawei Han
University of Illinois at Urbana-Champaign
Urbana, Illinois

Alexander Hinneburg
Martin-Luther University
Halle/Saale, Germany

Thomas S. Huang
University of Illinois at Urbana-Champaign
Urbana, Illinois

U Kang
KAIST
Seoul, Korea

George Karypis
University of Minnesota
Minneapolis, Minnesota

Dimitrios Kotsakos
University of Athens
Athens, Greece

Tao Li
Florida International University
Miami, Florida

Zhongmou Li
Rutgers University
New Brunswick, New Jersey

Huan Liu
Arizona State University
Tempe, Arizona

Jialu Liu
University of Illinois at Urbana-Champaign
Urbana, Illinois

Srinivasan Parthasarathy
The Ohio State University
Columbus, Ohio

Guo-Jun Qi
University of Illinois at Urbana-Champaign
Urbana, Illinois

Chandan K. Reddy
Wayne State University
Detroit, Michigan

Andrea Tagarelli
University of Calabria
Arcavacata di Rende, Italy

Jiliang Tang
Arizona State University
Tempe, Arizona

Hanghang Tong
IBM T. J. Watson Research Center
Yorktown Heights, New York

Goce Trajcevski
Northwestern University
Evanston, Illinois

Min-Hsuan Tsai
University of Illinois at Urbana-Champaign
Urbana, Illinois

Shen-Fu Tsai
Microsoft Inc.
Redmond, Washington

Bhanukiran Vinzamuri
Wayne State University
Detroit, Michigan

Wei Wang
University of California
Los Angeles, California

Hui Xiong
Rutgers University
New Brunswick, New Jersey

Mohammed J. Zaki
Rensselaer Polytechnic Institute
Troy, New York

Arthur Zimek
University of Alberta
Edmonton, Canada

Chapter 1

An Introduction to Cluster Analysis

Charu C. Aggarwal
IBM T. J. Watson Research Center
Yorktown Heights, NY
charu@us.ibm.com

1.1 Introduction

The problem of data clustering has been widely studied in the data mining and machine learning literature because of its numerous applications to summarization, learning, segmentation, and target marketing [46, 47, 52]. In the absence of specific labeled information, clustering can be considered a concise model of the data which can be interpreted in the sense of either a summary or a generative model. The basic problem of clustering may be stated as follows:

Given a set of data points, partition them into a set of groups which are as similar as possible.

Note that this is a very rough definition, and the variations in the problem definition can be significant, depending upon the specific model used. For example, a generative model may define similarity on the basis of a probabilistic generative mechanism, whereas a distance-based approach will use a traditional distance function for quantification. Furthermore, the specific data type also has a significant impact on the problem definition.

Some common application domains in which the clustering problem arises are as follows:

- **Intermediate Step for other fundamental data mining problems:** Since a clustering can be considered a form of data summarization, it often serves as a key intermediate step for many fundamental data mining problems such as classification or outlier analysis. A compact summary of the data is often useful for different kinds of application-specific insights.
- **Collaborative Filtering:** In collaborative filtering methods, the clustering provides a summarization of like-minded users. The ratings provided by the different users for each other are used in order to perform the collaborative filtering. This can be used to provide recommendations in a variety of applications.
- **Customer Segmentation:** This application is quite similar to collaborative filtering, since it creates groups of similar customers in the data. The major difference from collaborative filtering is that instead of using rating information, arbitrary attributes about the objects may be used for clustering purposes.
- **Data Summarization:** Many clustering methods are closely related to dimensionality reduction methods. Such methods can be considered a form of data summarization. Data summarization can be helpful in creating compact data representations, which are easier to process and interpret in a wide variety of applications.
- **Dynamic Trend Detection:** Many forms of dynamic and streaming algorithms can be used to perform trend detection in a wide variety of social networking applications. In such applications, the data is dynamically clustered in a streaming fashion and can be used in order to determine important patterns of changes. Examples of such streaming data could be multidimensional data, text streams, streaming time-series data, and trajectory data. Key trends and events in the data can be discovered with the use of clustering methods.
- **Multimedia Data Analysis:** A variety of different kinds of documents such as images, audio or video, fall in the general category of multimedia data. The determination of similar segments has numerous applications, such as the determination of similar snippets of music or similar photographs. In many cases, the data may be multimodal and may contain different types. In such cases, the problem becomes even more challenging.
- **Biological Data Analysis:** Biological data has become pervasive in the last few years, because of the success of the human genome effort and the increasing ability to collect different kinds of gene expression data. Biological data is usually structured either as sequences or as networks. Clustering algorithms provide good ideas of the key trends in the data, as well as the unusual sequences.
- **Social Network Analysis:** In these applications, the structure of a social network is used in order to determine the important communities in the underlying network. Community detection has important applications in social network analysis, because it provides an important understanding of the community structure in the network. Clustering also has applications to social network summarization, which is useful in a number of applications.

The aforementioned list of applications is not exhaustive by any means; nevertheless it represents a good cross-section of the wide diversity of problems which can be addressed with clustering algorithms.

The work in the data clustering area typically falls into a number of broad categories;

- **Technique-centered:** Since clustering is a rather popular problem, it is not surprising that numerous methods, such as probabilistic techniques, distance-based techniques, spectral techniques, density-based techniques, and dimensionality-reduction based techniques, are used for the clustering process. Each of these methods has its own advantages and disadvantages, and may work well in different scenarios and problem domains. Certain kinds of data types such as high dimensional data, big data, or streaming data have their own set of challenges and often require specialized techniques.
- **Data-Type Centered:** Different applications create different kinds of data types with different properties. For example, an ECG machine will produce time series data points which are highly correlated with one another, whereas a social network will generated a mixture of document and structural data. Some of the most common examples are categorical data, time series data, discrete sequences, network data, and probabilistic data. Clearly, the nature of the data greatly impacts the choice of methodology used for the clustering process. Furthermore, some data types are more difficult than others because of the separation between different kinds of attributes such as behavior or contextual attributes.
- **Additional Insights from Clustering Variations:** A number of insights have also been designed for different kinds of clustering variations. For example, visual analysis, supervised analysis, ensemble-analysis, or multiview analysis can be used in order to gain additional insights. Furthermore, the issue of cluster validation is also important from the perspective of gaining specific insights about the performance of the clustering.

This chapter will discuss each of these issues in detail, and will also discuss how the organization of the book relates to these different areas of clustering. The chapter is organized as follows. The next section discusses the common techniques which are used in cluster analysis. Section 1.3 explores the use of different data types in the clustering process. Section 1.4 discusses the use of different variations of data clustering. Section 1.5 offers the conclusions and summary.

1.2 Common Techniques Used in Cluster Analysis

The clustering problems can be addressed using a wide variation of methods. In addition, the data preprocessing phase requires dedicated techniques of its own. A number of good books and surveys discuss these issues [14, 20, 31, 37, 46, 47, 48, 52, 65, 80, 81]. The most common techniques which are used for clustering are discussed in this section.

1.2.1 Feature Selection Methods

The feature selection phase is an important preprocessing step which is needed in order to enhance the quality of the underlying clustering. Not all features are equally relevant to finding the clusters, since some may be more noisy than other. Therefore, it is often helpful to utilize a preprocessing phase in which the noisy and irrelevant features are pruned from contention. Feature selection and dimensionality reduction are closely related. In feature selection, original subsets of the features are selected. In dimensionality reduction, linear combinations of features may be used in techniques such as principal component analysis [50] in order to further enhance the feature selection effect. The advantage of the former is greater interpretability, whereas the advantage of the latter is that a lesser number of transformed directions is required for the representation process. Chapter 2 of this book will discuss such feature selection methods in detail. A comprehensive book on feature selection may be found in [61].

It should be noted that feature selection can also be integrated directly into the clustering algorithm to gain better locality specific insights. This is particularly useful, when different features are relevant to different localities of the data. The motivating factor for high dimensional subspace clustering algorithms is the failure of global feature selection algorithms. As noted in [9]: "*... in many real data examples, some points are correlated with respect to a given set of dimensions and others are correlated with respect to different dimensions. Thus, it may not always be feasible to prune off too many dimensions without at the same time incurring a substantial loss of information*" p.61. Therefore, the use of local feature selection, by integrating the feature selection process into the algorithm, is the best way of achieving this goal. Such local feature selections can also be extended to the dimensionality reduction problem [8, 19] and are sometimes referred to as *local dimensionality reduction*. Such methods are discussed in detail in Chapter 9. Furthermore, Chapter 2 also discusses the connections of these classes of methods to the problem of feature selection.

1.2.2 Probabilistic and Generative Models

In probabilistic models, the core idea is to model the data from a *generative process*. First, a specific form of the generative model (e.g., mixture of Gaussians) is assumed, and then the parameters of this model are estimated with the use of the Expectation Maximization (EM) algorithm [27]. The available data set is used to estimate the parameters in such as way that they have a *maximum likelihood fit* to the generative model. Given this model, we then estimate the generative probabilities (or fit probabilities) of the underlying data points. Data points which fit the distribution well will have high fit probabilities, whereas anomalies will have very low fit probabilities.

The broad principle of a mixture-based generative model is to *assume* that the data were generated from a mixture of k distributions with the probability distributions $\mathcal{G}_1 \ldots \mathcal{G}_k$ with the use of the following process:

- Pick a data distribution with prior probability α_i, where $i \in \{1 \ldots k\}$, in order to pick one of the k distributions. Let us assume that the rth one is picked.
- Generate a data point from $\mathcal{G}_r$.

The probability distribution $\mathcal{G}_r$ is picked from a host of different possibilities. Note that this generative process requires the determination of several parameters such as the prior probabilities and the model parameters for each distribution $\mathcal{G}_i$. Models with different levels of flexibility may be designed depending upon whether the prior probabilities are specified as part of the problem setting, or whether interattribute correlations are assumed within a component of the mixture. Note that the model parameters and the probability of assignment of data points to clusters are dependent on one another in a circular way. Iterative methods are therefore desirable in order to resolve this circularity. The generative models are typically solved with the use of an EM approach, which starts off with a random or heuristic initialization and then iteratively uses two steps to resolve the circularity in computation:

- (E-Step) Determine the expected probability of assignment of data points to clusters with the use of current model parameters.
- (M-Step) Determine the optimum model parameters of each mixture by using the assignment probabilities as weights.

One nice property of EM-models is that they can be generalized relatively easily to different kinds of data, as long as the generative model for each component is properly selected for the individual mixture component $\mathcal{G}_i$. Some examples are as follows:

- For numerical data, a Gaussian mixture model may be used in order to model each component $\mathcal{G}_r$. Such a model is discussed in detail in Chapter 3.
- For categorical data, a Bernoulli model may be used for $\mathcal{G}_r$ in order to model the generation of the discrete values.
- For sequence data, a Hidden Markov Model (HMM) may be used for $\mathcal{G}_r$ in order to model the generation of a sequence. Interestingly, an HMM is itself a special kind of mixture model in which the different components of the mixture are dependent on each other though transitions. Thus, the clustering of sequence data with a mixture of HMMs can be considered a two-level mixture model.

Generative models are among the most fundamental of all clustering methods, because they try to understand the underlying *process* through which a cluster is generated. A number of interesting connections exist between other clustering methods and generative models, by considering special cases in terms of prior probabilities or mixture parameters. For example, the special case in which each prior probability is fixed to the same value and all mixture components are assumed to have the same radius along all dimensions reduces to a soft version of the k-means algorithm. These connections will be discussed in detail in Chapter 3.

1.2.3 Distance-Based Algorithms

Many special forms of generative algorithms can be shown to reduce to distance-based algorithms. This is because the mixture components in generative models often use a distance function within the probability distribution. For example, the Gaussian distribution represents data generation probabilities in terms of the euclidian distance from the mean of the mixture. As a result, a generative model with the Gaussian distribution can be shown to have a very close relationship with the k-means algorithm. In fact, many distance-based algorithms can be shown to be reductions from or simplifications of different kinds of generative models.

Distance-based methods are often desirable because of their simplicity and ease of implementation in a wide variety of scenarios. Distance-based algorithms can be generally divided into two types:

- *Flat:* In this case, the data is divided into several clusters in one shot, typically with the use of partitioning representatives. The choice of the partitioning representative and distance function is crucial and regulates the behavior of the underlying algorithm. In each iteration, the data points are assigned to their closest partitioning representatives, and then the representative is adjusted according to the data points assigned to the cluster. It is instructive to compare this with the iterative nature of the EM algorithm, in which soft assignments are performed in the E-step, and model parameters (analogous to cluster representatives) are adjusted in the M-step. Some common methods for creating the partitions are as follows:
  - *k-Means:* In these methods, the partitioning representatives correspond to the mean of each cluster. Note that the partitioning representative is not drawn from the original data set, but is created as a function of the underlying data. The euclidian distance is used in order to compute distances. The k-Means method is considered one of the simplest and most classical methods for data clustering [46] and is also perhaps one of the most widely used methods in practical implementations because of its simplicity.
  - *k-Medians:* In these methods, the median along each dimension, instead of the mean, is used to create the partitioning representative. As in the case of the k-Means approach, the partitioning representatives are not drawn from the original data set. The k-Medians approach is more stable to noise and outliers, because the median of a set of values is usually less sensitive to extreme values in the data. It should also be noted that the term "k-Medians" is sometimes overloaded in the literature, since it is sometimes also used to refer to a k-Medoid approach (discussed below) in which the partitioning representatives are drawn from the original data. In spite of this overloading and confusion in the research literature, it should be noted that the k-Medians and k-Medoid methods should be considered as distinct techniques. Therefore, in several chapters of this book, the k-Medians approach discussed is really a k-Medoids approach, though we have chosen to be consistent within the specific research paper which is being described. Nevertheless, it would be useful to note the overloading of this term in order to avoid confusion.
  - *k-Medoids:* In these methods, the partitioning representative is sampled from the original data. Such techniques are particularly useful in cases, where the data points to be clustered are arbitrary objects, and it is often not meaningful to talk about functions of these objects. For example, for a set of network or discrete sequence objects, it may not be meaningful to talk about their mean or median. In such cases, partitioning representatives are drawn from the data, and iterative methods are used in order to improve the quality of these representatives. In each iteration, one of the representatives is replaced with a representative from the current data, in order to check if the quality of the clustering improves. Thus, this approach can be viewed as a kind of hill climbing method. These methods generally require many more iterations than k-Means and k-Medoids methods. However, unlike the previous two methods, they can be used in scenarios where it is not meaningful to talk about means or medians of data objects (eg. structural data objects).
- *Hierarchical:* In these methods, the clusters are represented hierarchically through a *dendogram*, at varying levels of granularity. Depending upon whether this hierarchical representation is created in top-down or bottom-up fashion, these representations may be considered either agglomerative or divisive.
  - *Agglomerative:* In these methods, a bottom-up approach is used, in which we start off with the individual data points and successively merge clusters in order to create a tree-like structure. A variety of choices are possible in terms of how these clusters may be merged, which provide different tradeoffs between quality and efficiency. Some examples of these choices are *single-linkage*, *all-pairs linkage*, *centroid-linkage*, and *sampled-linkage* clustering. In single-linkage clustering, the shortest distance between any pair of points in two clusters is used. In all-pairs linkage, the average over all pairs is used, whereas in sampled linkage, a sampling of the data points in the two clusters is used for calculating the average distance. In centroid-linkage, the distance between the centroids is used. Some variations of these methods have the disadvantage of *chaining*, in which larger clusters are naturally biased toward having closer distances to other points and will therefore attract a successively larger number of points. Single linkage clustering is particularly susceptible to this phenomenon. A number of data domains such as network clustering are also more susceptible to this behavior.
  - *Divisive:* In these methods, a top-down approach is used in order to successively partition the data points into a tree-like structure. Any flat clustering algorithm can be used in order to perform the partitioning at each step. Divisive partitioning allows greater flexibility in terms of both the hierarchical structure of the tree and the level of balance in the different clusters. It is not necessary to have a perfectly balanced tree in terms of the depths of the different nodes or a tree in which the degree of every branch is exactly two. This allows the construction of a tree structure which allows different tradeoffs in the balancing of the node depths and node weights (number of data points in the node). For example, in a top-down method, if the different branches of the tree are unbalanced in terms of node weights, then the largest cluster can be chosen preferentially for division at each level. Such an approach [53] is used in *METIS* in order to create well balanced clusters in large social networks, in which the problem of cluster imbalance is particularly severe. While *METIS* is not a distance-based algorithm, these general principles apply to distance-based algorithms as well.

Distance-based methods are very popular in the literature, because they can be used with almost any data type, as long as an appropriate distance function is created for that data type. Thus, the problem of clustering can be reduced to the problem of finding a distance function for that data type. Therefore, distance function design has itself become an important area of research for data mining in its own right [5, 82]. Dedicated methods also have often been designed for specific data domains such as categorical or time series data [32, 42]. Of course, in many domains, such as high-dimensional data, the quality of the distance functions reduces because of many irrelevant dimensions [43] and may show both errors and concentration effects, which reduce the statistical significance of data mining results. In such cases, one may use either the redundancy in larger portions of the pairwise distance matrix to abstract out the noise in the distance computations with spectral methods [19] or projections in order to directly find the clusters in relevant subsets of attributes [9]. A discussion of many hierarchical and partitioning algorithms is provided in Chapter 4 of this book.

1.2.4 Density- and Grid-Based Methods

Density- and grid-based methods are two closely related classes, which try to explore the data *space* at high levels of granularity. The density at any particular point in the data space is defined either in terms of the number of data points in a prespecified volume of its locality or in terms of a smoother kernel density estimate [74]. Typically, the data space is explored at a reasonably high level of granularity and a postprocessing phase is used in order to "put together" the dense regions of the data space into an arbitrary shape. Grid-based methods are a specific class of density-based methods in which the individual regions of the data space which are explored are formed into a grid-like structure. Grid-like structures are often particularly convenient because of greater ease in putting together the different dense blocks in the post-processing phase. Such grid-like methods can also be used in the context of high-dimensional methods, since the lower dimensional grids define clusters on subsets of dimensions [6].

A major advantage of these methods is that since they explore the data space at a high level of granularity, they can be used to reconstruct the entire shape of the data distribution. Two classical methods for density-based methods and grid-based methods are DBSCAN [34] and STING [83], respectively. The major challenge of density-based methods is that they are naturally defined on data points in a continuous space. Therefore, they often cannot be meaningfully used in a discrete or noneuclidian space, unless an embedding approach is used. Thus, many arbitrary data types such as time-series data are not quite as easy to use with density-based methods without specialized transformations. Another issue is that density computations becomes increasingly difficult to define

with greater dimensionality because of the greater number of cells in the underlying grid structure and the sparsity of the data in the underlying grid. A detailed discussion of density-based and grid-based methods is provided in Chapters 5 and 6 of this book.

1.2.5 Leveraging Dimensionality Reduction Methods

Dimensionality reduction methods are closely related to both feature selection and clustering, in that they attempt to use the closeness and correlations between dimensions to reduce the dimensionality of representation. Thus, dimensionality reduction methods can often be considered a vertical form of clustering, in which columns of the data are clustered with the use of either correlation or proximity analysis, as opposed to the rows. Therefore, a natural question arises, as to whether it is possible to perform these steps *simultaneously*, by clustering rows and columns of the data together. The idea is that simultaneous row and column clustering is likely to be more effective than performing either of these steps individually. This broader principle has led to numerous algorithms such as matrix factorization, spectral clustering, probabilistic latent semantic indexing, and co-clustering. Some of these methods such as spectral clustering are somewhat different but are nevertheless based on the same general concept. These methods are also closely related to projected clustering methods, which are commonly used for high dimensional data in the database literature. Some common models will be discussed below.

1.2.5.1 Generative Models for Dimensionality Reduction

In these models, a generative probability distribution is used to model the relationships between the data points and dimensions in an integrated way. For example, a generalized Gaussian distribution can be considered a mixture of arbitrarily correlated (oriented) clusters, whose parameters can be learned by the EM-algorithm. Of course, this is often not easy to do robustly with increasing dimensionality due to the larger number of parameters involved in the learning process. It is well known that methods such as EM are highly sensitive to overfitting, in which the number of parameters increases significantly. This is because EM methods try to retain *all* the information in terms of soft probabilities, rather than making the hard choices of point and dimension selection by non-parametric methods. Nevertheless, many special cases for different data types have been learned successfully with generative models.

A particularly common dimensionality reduction method is that of topic modeling in text data [45]. This method is also sometimes referred to as *Probabilistic Latent Semantic Indexing (PLSI)*. In topic modeling, a cluster is associated with a set of words and a set of documents simultaneously. The main parameters to be learned are the probabilities of assignments of words (dimensions) to topics (clusters) and those of the documents (data points) to topics (clusters). Thus, this naturally creates a soft clustering of the data from *both* a row and column perspective. These are then learned in an integrated way. These methods have found a lot of success in the text mining literature, and many methods, such as *Latent Dirichlet Allocation (LDA)*, which vary on this principle, with the use of more generalized priors have been proposed [22]. Many of these models are discussed briefly in Chapter 3.

1.2.5.2 Matrix Factorization and Co-Clustering

Matrix factorization and co-clustering methods are also commonly used classes of dimensionality reduction methods. These methods are usually applied to data which is represented as sparse nonnegative matrices, though it is possible in principle to generalize these methods to other kinds of matrices as well. However, the real attraction of this approach is the additional *interpretability* inherent in *nonnegative* matrix factorization methods, in which a data point can be expressed as a *nonnegative* linear combination of the concepts in the underlying data. Nonnegative matrix factorization methods are closely related to co-clustering, which clusters the rows and columns of a matrix simultaneously.

Let A be a nonnegative $n \times d$ matrix, which contains n data entries, each of which has a dimensionality of d. In most typical applications such as text data, the matrix A represents small nonnegative quantities such as word frequencies and is not only nonnegative, but also sparse. Then, the matrix A can be *approximately* factorized into two nonnegative low rank matrices U and V of sizes $n \times k$ and $k \times d$, respectively. As we will discuss later, these matrices are the representatives for the clusters on the rows and the columns, respectively, when exactly k clusters are used in order to represent both rows and columns. Therefore, we have

$$A \approx U \cdot V \tag{1.1}$$

The residual matrix R represents the noise in the underlying data:

$$R = A - U \cdot V \tag{1.2}$$

Clearly, it is desirable to determine the factorized matrices U and V, such that the sum of the squares of the residuals in R is minimized. This is equivalent to determining nonnegative matrices U and V, such that the Froebinius norm of $A - U \cdot V$ is minimized. This is a constrained optimization problem, in which the constraints correspond to the nonnegativity of the entries in U and V. Therefore, a Lagrangian method can be used to learn the parameters of this optimization problem. A detailed discussion of the iterative approach is provided in [55].

The nonnegative $n \times k$ matrix U represents the coordinates of each of the n data points into each of the k newly created dimensions. A high positive value of the entry (i, j) implies that data point i is closely related to the newly created dimension j. Therefore, a trivial way to perform the clustering would be to assign each data point to the newly created dimension for which it has the largest component in U. Alternatively, if data points are allowed to belong to multiple clusters, then for each of the k columns in V, the entries with value above a particular threshold correspond to the document clusters. Thus, the newly created set of dimensions can be made to be synonymous with the clustering of the data set. In practice, it is possible to do much better, by using k-means on the new representation. One nice characteristic of the nonnegative matrix factorization method is that the size of an entry in the matrix U tells us how much a particular data point is related to a particular concept (or newly created dimension). The price of this greater interpretability is that the newly created sets of dimensions are typically not orthogonal to one another. This brings us to a discussion of the physical interpretation of matrix V.

The $k \times d$ matrix V provides the actual representation of each of the k newly created dimensions in terms of the original d dimensions. Thus, each row in this matrix is one component of this newly created axis system, and the rows are not necessarily orthogonal to one another (unlike most other dimensionality reduction methods). A large positive entry implies that this newly created dimension is highly related to a particular dimension in the underlying data. For example, in a document clustering application, the entries with large positive values in each row represent the most common words in each cluster. A thresholding technique can be used to identify these words. Note the similarity of this approach with a projection-based technique or the softer PLSI technique. In the context of a document clustering application, these large positive entries provide the *word clusters* in the underlying data. In the context of text data, each document can therefore be approximately expressed (because of the factorization process) as a nonnegative linear combination of at most k word-cluster vectors. The specific weight of that component represents the importance of that component, which makes the decomposition highly interpretable. Note that this interpretability is highly dependent on nonnegativity. Conversely, consider the word-membership vector across the corpus. This can be expressed in terms of at most k document-cluster vectors. This provides an idea of how important each document cluster is to that word.

At this point, it should be evident that the matrices U and V *simultaneously* provide the clusters

on the rows (documents) and columns (words). This general principle, when applied to sparse nonnegative matrices, is also referred to as co-clustering. Of course, nonnegative matrix factorization is only one possible way to perform the co-clustering. A variety of graph-based spectral methods and other information theoretic methods can also be used in order to perform co-clustering [29, 30, 71]. Matrix factorization methods are discussed in Chapter 7. These techniques are also closely related to *spectral methods* for dimensionality reduction, as discussed below.

1.2.5.3 Spectral Methods

Spectral methods are an interesting technique for dimensionality reduction, which work with the similarity (or distance) matrix of the underlying data, instead of working with the original points and dimensions. This, of course, has its own set of advantages and disadvantages. The major advantage is that it is now possible to work with arbitrary objects for dimensionality reduction, rather than simply data points which are represented in a multi-dimensional space. In fact, spectral methods also perform the dual task of *embedding* these objects into a euclidian space, while performing the dimensionality reduction. Therefore, spectral methods are extremely popular for performing clustering on arbitrary objects such as node sets in a graph. The disadvantage of spectral methods is that since they work with an $n \times n$ similarity matrix, the time complexity for even creating the similarity matrix scales with the *square* of the number of *data points*. Furthermore, the process of determining the eigenvectors of this matrix can be extremely expensive, unless a very small number of these eigenvectors is required. Another disadvantage of spectral methods is that it is much more difficult to create lower dimensional representations for data points, unless they are part of the original sample from which the similarity matrix was created. For multidimensional data, the use of such a large similarity matrix is rather redundant, unless the data is extremely noisy and high dimensional.

Let $\mathcal{D}$ be a database containing n points. The first step is to create an $n \times n$ matrix W of weights, which represents the pairwise similarity between the different data points. This is done with the use of the *heat kernel*. For any pair of data points $\overline{X_i}$ and $\overline{X_j}$, the *heat kernel* defines a similarity matrix W of weights:

$$W_{ij} = exp(-||\overline{X_i} - \overline{X_j}||^2/t) \tag{1.3}$$

Here t is a user-defined parameter. Furthermore, the value of W_{ij} is set to 0 if the distance between $\overline{X_i}$ and $\overline{X_j}$ is greater than a given threshold. The similarity matrix may also be viewed as the adjacency matrix of a graph, in which each node corresponds to a data item, and the weight of an edge corresponds to the similarity between these data items. Therefore, spectral methods reduce the problem to that of finding optimal cuts in this graph, which correspond to partitions which are weakly connected by edges representing similarity. Hence, spectral methods can be considered a graph-based technique for clustering of any kinds of data, by converting the similarity matrix into a network structure. Many variants exist in terms of the different choices for constructing the similarity matrix W. Some simpler variants use the mutual k-nearest neighbor graph, or simply the binary graph in which the distances are less than a given threshold. The matrix W is symmetric.

It should be noted that even when distances are very noisy, the similarity matrix encodes a significant amount of information because of its exhaustive nature. It is here that spectral analysis is useful, since the noise in the similarity representation can be abstracted out with the use of eigenvector-analysis of this matrix. Thus, these methods are able to recover and sharpen the latent information in the similarity matrix, though at a rather high cost, which scales with the square of the number of data points.

First, we will discuss the problem of mapping the points onto a 1-dimensional space. The generalization to the k-dimensional case is relatively straightforward. We would like to map the data points in $\mathcal{D}$ into a set of points $y_1 \ldots y_n$ on a line, in which the similarity maps onto euclidian distances on this line. Therefore, it is undesirable for data points which are very similar to be mapped

onto distant points on this line. We would like to determine values of y_i which minimize the following objective function O:

$$O = \sum_{i=1}^{n} \sum_{j=1}^{n} W_{ij} \cdot (y_i - y_j)^2 \tag{1.4}$$

The objective function O can be rewritten in terms of the Laplacian matrix L of W. The Laplacian matrix is defined as $D - W$, where D is a diagonal matrix satisfying $D_{ii} = \sum_{j=1}^{n} W_{ij}$. Let $\overline{y} = (y_1 \ldots y_n)$. The objective function O can be rewritten as follows:

$$O = 2 \cdot \overline{y}^T \cdot L \cdot \overline{y} \tag{1.5}$$

We need to incorporate a scaling constraint in order to ensure that the trivial value of $y_i = 0$ for all i is not selected by the problem. A possible scaling constraint is as follows:

$$\overline{y}^T \cdot D \cdot \overline{y} = 1 \tag{1.6}$$

Note that the use of the matrix D provides different weights to the data items, because it is assumed that nodes with greater similarity to different data items are more involved in the clustering process. This optimization formulation is in generalized eigenvector format, and therefore the value of $\overline{y}$ is optimized by selecting the smallest eigenvalue for which the generalized eigenvector relationship $L \cdot \overline{y} = \lambda \cdot D \cdot \overline{y}$ is satisfied. In practice however, the smallest generalized eigenvalue corresponds to the trivial solution, where $\overline{y}$ is the (normalized) unit vector. This trivial eigenvector is noninformative. Therefore, it can be discarded, and it is not used in the analysis. The second-smallest eigenvalue then provides an optimal solution, which is more informative.

This model can be generalized to determining all the eigenvectors in ascending order of eigenvalue. Such directions correspond to successive orthogonal directions in the data, which result in the best mapping. This results in a set of n eigenvectors $\overline{e_1}, \overline{e_2} \ldots \overline{e_n}$ (of which the first is trivial), with corresponding eigenvalues $0 = \lambda_1 \leq \lambda_2 \ldots \leq \lambda_n$. Let the corresponding vector representation of the data points along each eigenvector be denoted by $\overline{y}^1 \ldots \overline{y}^n$.

What do the small and large magnitude eigenvectors intuitively represent in the new transformed space? By using the ordering of the data items along a small magnitude eigenvector to create a cut, the weight of the edges across the cut is likely to be small. Thus, this represents a cluster in the space of data items. At the same time, if the top k *largest* eigenvectors are picked, then the vector representations $\overline{y}^n \ldots \overline{y}^{n-k+1}$ provide a $n \times k$ matrix. This provides a k-dimensional embedded representation of the entire data set of n points, which preserves the maximum amount of information. Thus, spectral methods can be used in order to simultaneously perform clustering and dimensionality reduction. In fact, spectral methods can be used to recover the entire lower dimensional (possibly nonlinear) shape of the data, though arguably at a rather high cost [78]. The specific local dimensionality reduction technique of this section is discussed in detail in [19]. An excellent survey on spectral clustering methods may be found in [35, 63]. These methods are also discussed in detail in Chapter 8 of this book.

1.2.6 The High Dimensional Scenario

The high dimensional scenario is particularly challenging for cluster analysis because of the large variations in the behavior of the data attributes over different parts of the data. This leads to numerous challenges in many data mining problems such as clustering, nearest neighbor search, and outlier analysis. It should be noted that many of the algorithms for these problems are dependent upon the use of distances as an important subroutine. However, with increasing dimensionality, the distances seem to increasingly lose their effectiveness and statistical significance because of irrelevant attributes. The premise is that a successively smaller *fraction* of the attributes often remains relevant with increasing dimensionality, which leads to the blurring of the distances and increasing

concentration effects, because of the averaging behavior of the irrelevant attributes. *Concentration effects* refer to the fact that when many features are noisy and uncorrelated, their additive effects will lead to all pairwise distances between data points becoming similar. The noise and concentration effects are problematic in two ways for distance-based clustering (and many other) algorithms:

- An increasing noise from irrelevant attributes may cause errors in the distance representation, so that it no longer properly represents the *intrinsic distance* between data objects.
- The concentration effects from the irrelevant dimensions lead to a reduction in the statistical significance of the results from distance-based algorithms, if used directly with distances that have not been properly denoised.

The combination of these issues above leads to questions about whether full-dimensional distances are truly meaningful [8, 21, 43]. While the natural solution to such problems is to use feature selection, the problem is that different attributes may be relevant to different localities of the data. This problem is inherent in high-dimensional distance functions and nearest neighbor search. As stated in [43]: "*... One of the problems of the current notion of nearest neighbor search is that it tends to give equal treatment to all features (dimensions), which are however not of equal importance. Furthermore, the importance of a given dimension may not even be independent of the query point itself*" p. 506. These noise and concentration effects are therefore a problematic *symptom* of (locally) irrelevant, uncorrelated, or noisy attributes, which tend to impact the effectiveness and statistical significance of full-dimensional algorithms. This has lead to significant efforts to redesign many data mining algorithms such as clustering, which are dependent on the notion of distances [4]. In particular, it would seem odd that data mining algorithms should behave poorly with increasing dimensionality at least from a qualitative perspective when a larger number of dimensions clearly provides more information. The reason is that conventional distance measures were generally designed for many kinds of low-dimensional spatial applications which are not suitable for the high-dimensional case. While high-dimensional data contain more information, they are also more complex. Therefore, naive methods will do worse with increasing dimensionality because of the noise effects of locally irrelevant attributes. Carefully designed distance functions can leverage the greater information in these dimensions and show *improving* behavior with dimensionality [4, 11] at least for a few applications such as similarity search.

Projected clustering methods can be considered a form of *local feature selection*, or *local dimensionality reduction*, in which the feature selection or transformation is performed specific to different localities of the data. Some of the earliest methods even refer to these methods as local-dimensionality reduction [23] in order to emphasize the local feature selection effect. Thus, a projected cluster is defined as a set of clusters $C_1 \ldots C_k$, along with a corresponding set of subspaces $\mathcal{E}_1 \ldots \mathcal{E}_k$, such that the points in C_i cluster well in the subspace represented by $\mathcal{E}_i$. Thus, these methods are a form of local feature selection, which can be used to determine the relevant clusters in the underlying data. Note that the subspace $\mathcal{E}_i$ could represent a subset of the original attributes, or it could represent a transformed axis system in which the clusters are defined on a small set of orthogonal attributes. Some of the earliest projected clustering methods are discussed in [6, 9, 10]. The literature on this area has grown significantly since 2000, and surveys on the area may be found in [67, 54]. An overview of algorithms for high-dimensional data will be provided in Chapter 9.

It should also be pointed out that while pairwise distances are often too noisy or concentrated to be used meaningfully with off-the-shelf distance-based algorithms, larger portions of the *entire* distance matrix often retain a significant amount of latent structure due to the inherent redundancies in representing different pairwise distances. Therefore, even when there is significant noise and concentration of the distances, there will always be some level of consistent variations over the similarity matrix because of the impact of the consistent attributes. This redundancy can be leveraged in conjunction with spectral analysis [19, 78] to filter out the noise and concentration effects from full-dimensional distances and implicitly recover the local or global lower dimensional shape of the underlying data in terms of enhanced distance representations of newly embedded data in a lower dimensional space. In essence, this approach enhances the contrasts of the distances by carefully examining how the noise (due to the many irrelevant attributes) and the minute correlations (due to the smaller number of relevant attributes) relate to the different pairwise distances. The general principle of these techniques is that distances are measured differently in high-dimensional space, depending upon how the data is distributed. This in turn depends upon the relevance and relationships of different attributes in different localities. These results are strong evidence for the fact that proper distance function design is highly dependent on its ability to recognize the relevance and relationships of different attributes in different data localities.

Thus, while (naively designed) pairwise distances cannot be used meaningfully in high dimensional data with *off-the-shelf* algorithms because of noise and concentration effects, they often do retain sufficient *latent* information *collectively* when used carefully in conjunction with denoising methods such as spectral analysis. Of course, the price for this is rather high, since the size of the similarity matrix scales with the square of the number of data points. The projected clustering method is generally a more efficient way of achieving an approximately similar goal, since it works directly with the data representation, rather than building a much larger and more redundant intermediate representation such as the distance matrix.

1.2.7 Scalable Techniques for Cluster Analysis

With advances in software and hardware technology, data collection has become increasingly easy in a wide variety of scenarios. For example, in social sensing applications, users may carry mobile or wearable sensors, which may result in the continuous accumulation of data over time. This leads to numerous challenges when real-time analysis and insights are required. This is referred to as the *streaming* scenario, in which it is assumed that a single pass is allowed over the data stream, because the data are often too large to be collected within limited resource constraints. Even when the data are collected offline, this leads to numerous scalability issues, in terms of integrating with traditional database systems or in terms of using the large amounts of data in a distributed setting, with the big data framework. Thus, varying levels of challenges are possible, depending upon the nature of the underlying data. Each of these issues is discussed below.

1.2.7.1 I/O Issues in Database Management

The most fundamental issues of scalability arise when a data mining algorithm is coupled with a traditional database system. In such cases, it can be shown that the major bottleneck arises from the I/O times required for accessing the objects in the database. Therefore, algorithms which use sequential scans of the data, rather than methods which randomly access the data records, are often useful. A number of classical methods were proposed in the database literature in order to address these scalability issues.

The easiest algorithms to extend to this case are flat partitioning methods which use sequential scans over the database in order to assign data points to representatives. One of the first methods [66] in this direction was *CLARANS*, in which these representatives are determined with the use of a k-medoids approach. Note that the k-medoids approach can still be computationally quite intensive because each iteration requires trying out new partitioning representatives through an exchange process (from the current set). If a large number of iterations is required, this will increase the number of passes over the data set. Therefore, the *CLARANS* method uses sampling, by performing the iterative hill climbing over a smaller sample, in order to improve the efficiency of the approach. Another method in this direction is *BIRCH* [87], which generalizes a k-means approach to the clustering process. The *CURE* method proposed in [41] finds clusters of nonspherical shape by using more than one representative point per cluster. It combines partitioning and sampling to ensure a

more efficient clustering process. A number of scalable algorithms for clustering are discussed in Chapter 11.

1.2.7.2 Streaming Algorithms

The streaming scenario is particularly challenging for clustering algorithms due to the requirements of *real-time* analysis, and the *evolution* and *concept-drift* in the underlying data. While database-centric algorithms require a *limited* number of passes over the data, streaming algorithms require exactly one pass, since the data cannot be stored at all. In *addition* to this challenge, the analysis typically needs to be performed in real time, and the changing patterns in the data need to be properly accounted for in the analysis.

In order to achieve these goals, virtually all streaming methods use a summarization technique to create *intermediate representations*, which can be used for clustering. One of the first methods in this direction uses a *microclustering approach* [7] to create and maintain the clusters from the underlying data stream. Summary statistics are maintained for the microclusters to enable an effective clustering process. This is combined with a pyramidal time frame to capture the evolving aspects of the underlying data stream. Stream clustering can also be extended to other data types such as discrete data, massive-domain data, text data, and graph data. A number of unique challenges also arises in the distributed setting. A variety of stream clustering algorithms is discussed in detail in Chapter 10.

1.2.7.3 The Big Data Framework

While streaming algorithms work under the assumption that the data are too large to be stored explicitly, the big data framework leverages advances in storage technology in order to actually store the data and process them. However, as the subsequent discussion will show, even if the data can be explicitly stored, it is often not easy to process and extract insights from them. This is because an increasing size of the data implies that a distributed file system must be used in order to store the information, and distributed processing techniques are required in order to ensure sufficient scalability. The challenge here is that if large segments of the data are available on different machines, it is often too expensive to shuffle the data across different machines in order to extract integrated insights from them. Thus, as in all distributed infrastructures, it is desirable to exchange intermediate insights, so as to minimize communication costs. For an application programmer, this can sometimes create challenges in terms of keeping track of where different parts of the data are stored, and the precise ordering of communications in order to minimize the costs.

In this context, Google's *MapReduce* framework [28] provides an effective method for analysis of large amounts of data, especially when the nature of the computations involves linearly computable statistical functions over the elements of the data streams. One desirable aspect of this framework is that it abstracts out the precise details of where different parts of the data are stored to the application programmer. As stated in [28]: "*The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system*" p. 107. Many clustering algorithms such as k-means are naturally linear in terms of their scalability with the size of the data. A primer on the *MapReduce* framework implementation on *Apache Hadoop* may be found in [84]. The key idea here is to use a *Map* function to distribute the work across the different machines, and then provide an automated way to shuffle out much smaller data in (key,value) pairs containing intermediate results. The *Reduce* function is then applied to the aggregated results from the *Map* step to obtain the final results.

Google's original *MapReduce* framework was designed for analyzing large amounts of web logs and more specifically deriving linearly computable statistics from the logs. While the clustering process is not as simple as linearly computable statistics, it has nevertheless been shown [26] that many existing clustering algorithms can be generalized to the *MapReduce* framework. A proper choice of the algorithm to adapt to the *MapReduce* framework is crucial, since the framework is particularly effective for linear computations. It should be pointed out that the major attraction of the *MapReduce* framework is its ability to provide application programmers with a cleaner abstraction, which is independent of very specific run-time details of the distributed system. It should not, however, be assumed that such a system is somehow inherently superior to existing methods for distributed parallelization from an *effectiveness* or *flexibility* perspective, especially if an application programmer is willing to design such details from scratch. A detailed discussion of clustering algorithms for big data is provided in Chapter 11.

1.3 Data Types Studied in Cluster Analysis

The specific data type has a tremendous impact on the particular choice of the clustering algorithm. Most of the earliest clustering algorithms were designed under the implicit assumption that the data attributes were numerical. However, this is not true in most real scenarios, where the data could be drawn from any number of possible types such as discrete (categorical), temporal, or structural. This section discusses the impact of data types on the clustering process. A brief overview of the different data types is provided in this section.

1.3.1 Clustering Categorical Data

Categorical data is fairly common in real data sets. This is because many attributes in real data such as sex, race, or zip code are inherently discrete and do not take on a natural ordering. In many cases, the data sets may be *mixed*, in which some attributes such as salary are numerical, whereas other attributes such as sex or zip code are categorical. A special form of categorical data is market basket data, in which all attributes are binary.

Categorical data sets lead to numerous challenges for clustering algorithms:

- When the algorithms depends upon the use of a similarity or distance function, the standard L_k metrics can no longer be used. New similarity measures need to be defined for categorical data. A discussion of similarity measures for categorical data is provided in [32].
- Many clustering algorithms such as the k-means or k-medians methods construct clustering representatives as the means or medians of the data points in the clusters. In many cases, statistics such as the mean or median are naturally defined for numerical data but need to be appropriately modified for discrete data.

When the data is mixed, then the problem because more difficult because the different attributes now need to be treated in a heterogeneous way, and the similarity functions need to explicitly account for the underlying heterogeneity.

It should be noted that some models of clustering are more amenable to different data types than others. For example, some models depend only on the distance (or similarity) functions between records. Therefore, as long as an appropriate similarity function can be defined between records, cluster analysis methods can be used effectively. Spectral clustering is one class of methods which can be used with virtually any data type, as long as appropriate similarity functions are defined. The downside is that the the methods scale with the square of the similarity matrix size. Generative models can also be generalized easily to different data types, as long as an appropriate generative model can be defined for each component of the mixture. Some common algorithms for categorical

data clustering include *CACTUS* [38], *ROCK* [40], *STIRR* [39], and *LIMBO* [15]. A discussion of categorical data clustering algorithms is provided in Chapter 12.

1.3.2 Clustering Text Data

Text data is a particularly common form of data with the increasing popularity of the web and social networks. Text data is typically represented in vector space format, in which the specific ordering is abstracted out, and the data is therefore treated as a bag-of-words. It should be noted that the methods for clustering text data can also be used for clustering set-based attributes. Text data has a number of properties which should be taken into consideration:

- The data is extremely high-dimensional and sparse. This corresponds to the fact that the text lexicon is rather large, but each document contains only a small number of words. Thus, most attributes take on zero values.
- The attribute values correspond to word frequencies and are, therefore, typically nonnegative. This is important from the perspective of many co-clustering and matrix factorization methods, which leverage this nonnegativity.

The earliest methods for text clustering such as the scatter–gather method [25, 75] use distance-based methods. Specifically, a combination of k-means and agglomerative methods is used for the clustering process. Subsequently, the problem of text clustering has often been explored in the context of topic modeling, where a soft membership matrix of words and documents to clusters is created. The EM-framework is used in conjunction with these methods. Two common methods used for generative topic modeling are *PLSI* and *LDA* [22, 45]. These methods can be considered soft versions of methods such as co-clustering [29, 30, 71] and matrix factorization [55], which cluster the rows and columns together at the same time. Spectral methods [73] are often used to perform this partitioning by creating a bipartite similarity graph, which represents the membership relations of the rows (documents) and columns (words). This is not particularly surprising since matrix factorization methods and spectral clustering are known to be closely related [57], as discussed in Chapter 8 of this book. Surveys on text clustering may be found in [12, 88]. In recent years, the popularity of social media has also lead to an increasing importance of *short* text documents. For example, the posts and tweets in social media web sites are constrained in length, both by the platform and by user preferences. Therefore, specialized methods have also been proposed recently for performing the clustering in cases where the documents are relatively short. Methods for clustering different kinds of text data are discussed in Chapter 13.

1.3.3 Clustering Multimedia Data

With the increasing popularity of social media, many forms of multimedia data may be used in conjunction with clustering methods. These include image data, audio and video. Examples of social media sites which contain large amounts of such data are *Flickr* and *Youtube*. Even the web and conventional social networks typically contain a significant amount of multimedia data of different types. In many cases, such data may occur in conjunction with other more conventional forms of text data.

The clustering of multimedia data is often a challenging task, because of the varying and heterogeneous nature of the underlying content. In many cases, the data may be multimodal, or it may be contextual, containing both behavioral and contextual attributes. For example, image data are typically contextual, in which the position of a pixel represents its context, and the value on the pixel represents its behavior. Video and music data are also contextual, because the temporal ordering of the data records provides the necessary information for understanding. The heterogeneity and contextual nature of the data can only be addressed with proper data representations and analysis. In fact, data representation seems to be a key issue in all forms of multimedia analysis, which significantly impacts the final quality of results. A discussion of methods for clustering multimedia data is provided in Chapter 14.

1.3.4 Clustering Time-Series Data

Time-series data is quite common in all forms of sensor data, stock markets, or any other kinds of temporal tracking or forecasting applications. The major aspect of time series is that the data values are not independent of one another, but they are temporally dependent on one another. Specifically, the data contain a contextual attribute (time) and a behavioral attribute (data value). There is a significant diversity in problem definitions in the time-series scenario. The time-series data can be clustered in a variety of different ways, depending upon whether correlation-based online analysis is required or shape-based offline analysis is required.

- In correlation-based online analysis, the correlations among the different time-series data streams are tracked over time in order to create online clusters. Such methods are often useful for sensor selection and forecasting, especially when streams exhibit lag correlations within the clustered patterns. These methods are often used in stock market applications, where it is desirable to maintain groups of clustered stock tickers, in an online manner, based on their correlation patterns. Thus, the distance functions between different series need to be computed continuously, based on their mutual regression coefficients. Some examples of these methods include the MUSCLES approach [86] and a large scale time-series correlation monitoring method [90].
- In shape-based offline analysis, the time-series objects are analyzed in offline manner, and the specific details about when a particular time series was created is not important. For example, for a set of ECG time series collected from patients, the precise time of when a series was collected is not important, but the overall shape of the series is important for the purposes of clustering. In such cases, the distance function between two time series is important. This is important, because the different time series may not be drawn on the same range of data values and may also show time-warping effects, in which the shapes can be matched only by elongating or shrinking portions of the time-series in the temporal direction. As in the previous case, the design of the distance function [42] holds the key to the effective use of the approach.

A particular interesting case is that of multivariate time series, in which many series are simultaneously produced over time. A classical example of this is trajectory data, in which the different coordinate directions form the different components of the multivariate series. Therefore, trajectory analysis can be viewed as a special kind of time-series clustering. As in the case of univariate time series, it is possible to perform these steps using either online analysis (trajectories moving together in real time) or offline analysis (similar shape). An example of the former is discussed in [59], whereas an example of the latter is discussed in [68]. A survey on time-series data is found in [60], though this survey is largely focussed on the case of offline analysis. Chapter 15 discusses both the online and offline aspects of time series clustering.

1.3.5 Clustering Discrete Sequences

Many forms of data create discrete sequences instead of categorical ones. For example, web logs, the command sequences in computer systems, and biological data are all discrete sequences. The contextual attribute in this case often corresponds to placement (e.g., biological data), rather than time. Biological data is also one of the most common applications of sequence clustering.

As with the case of continuous sequences, a key challenge is the creation of similarity functions between different data objects. Numerous similarity functions such as the hamming distance, edit distance, and longest common subsequence are commonly used in this context. A discussion of the similarity functions which are commonly used for discrete sequences may be found in [58]. Another key problem which arises in the context of clustering discrete sequences is that the intermediate and summary representation of a set of sequences can be computationally intensive. Unlike numerical data, where averaging methods can be used, it is much more difficult to find such representations for discrete sequences. A common representation, which provides a relatively limited level of summarization is the suffix tree. Methods for using suffix trees for sequence clustering methods have been proposed in CLUSEQ [85].

Generative models can be utilized, both to model the distances between sequences and to create probabilistic models of cluster generation [76]. A particularly common approach is the use of mixtures of HMMs. A primer on Hidden Markov Models may be found in [70]. Hidden Markov Models can be considered a special kind of mixture model in which the different components of the mixture are dependent on one another. A second level of mixture modeling can be used in order to create clusters from these different HMMs. Much of the work on sequence clustering is performed in the context of biological data. Detailed surveys on the subject may be found in [33, 49, 64]. A discussion of sequence clustering algorithms is provided in Chapter 16, though this chapter also provides a survey of clustering algorithms for other kinds of biological data.

1.3.6 Clustering Network Data

Graphs and networks are among the most fundamental (and general) of all data representations. This is because virtually every data type can be represented as a similarity graph, with similarity values on the edges. In fact, the method of spectral clustering can be viewed as the most general connection between all other types of clustering and graph clustering. Thus, as long as a similarity function can be defined between arbitrary data objects, spectral clustering can be used in order to perform the analysis. Graph clustering has been studied extensively in the classical literature, especially in the context of the problem of 2-way and multi-way graph partitioning. The most classical of all multi-way partitioning algorithms is the Kernighan-Lin method [56]. Such methods can be used in conjunction with graph coarsening methods in order to provide efficient solutions. These methods are known as multilevel graph partitioning techniques. A particularly popular algorithm in this category is METIS [53].

A number of methods are commonly used in the literature in order to create partitions from graphs:

- *Generative Models:* As discussed earlier in this chapter, it is possible to define a generative model for practically any clustering problem, as long as an appropriate method exists for defining each component of the mixture as a probability distribution. An example of a generative model for network clustering is found in [77].
- *Classical Combinatorial Algorithms:* These methods use network flow [13] or other iterative combinatorial techniques in order to create partitions from the underlying graph. It should be pointed out that even edge sampling is often known to create good quality partitions, when it is repeated many times [51]. It is often desirable to determine cuts which are well balanced across different partitions, because the cut with the smallest absolute value often contains the large majority of the nodes in a single partition and a very small number of nodes in the remaining partitions. Different kinds of objective functions in terms of creating the cut, such as the unnormalized cut, normalized cut, and ratio cut, provide different tradeoffs between cluster balance and solution quality [73]. It should be pointed out that since graph cuts are a combinatorial optimization problem, they can be formulated as integer programs.

- *Spectral Methods:* Spectral methods can be viewed as *linear programming relaxations* to the integer programs representing the optimization of graph cuts. Different objective functions can be constructed for different kinds of cuts, such as the unnormalized cut, ratio cut, and normalized cut. Thus, the continuous solutions to these linear programs can be used to create a multidimensional embedding for the nodes, on which conventional k-means algorithms can be applied. These linear programs can be shown to take on a specially convenient form, in which the generalized eigenvectors of the graph Laplacian correspond to solutions of the optimization problem.
- *Nonnegative Matrix Factorization:* Since a graph can be represented as an adjacency matrix, nonnegative matrix factorization can be used in order to decompose it into two low rank matrices. It is possible to apply the matrix factorization methods either to the node–node adjacency matrix or the node–edge adjacency matrix to obtain different kinds of insights. It is also relatively easy to augment the matrix with content to create analytical methods, which can cluster with a combination of content and structure [69].

While the aforementioned methods represent a sampling of the important graph clustering methods, numerous other objective functions are possible for the construction of graph cuts such as the use of modularity-based objective functions [24]. Furthermore, the problem becomes even more challenging in the context of social networks, where content may be available at either the nodes [89] or edges [69]. Surveys on network clustering may be found in [36, 72]. Algorithms for network clustering are discussed in detail in Chapter 17.

1.3.7 Clustering Uncertain Data

Many forms of data either are of low fidelity or the quality of the data has been intentionally degraded in order to design different kinds of network mining algorithms. This has lead to the field of probabilistic databases. Probabilistic data can be represented either in the form of attribute-wise uncertainty or in the form of a *possible worlds* model, in which only particular subsets of attributes can be present in the data at a given time [3]. The key idea here is that the incorporation of probabilistic information can improve the quality of data mining algorithms. For example, if two attributes are equally desirable to use in an algorithm in the deterministic scenario, but one of them has greater uncertainty than the other, then the attribute with less uncertainty is clearly more desirable for use. Uncertain clustering algorithms have also been extended recently to the domain of streams and graphs. In the context of graphs, it is often desirable to determine the most reliable subgraphs in the underlying network. These are the subgraphs which are the most difficult to disconnect under edge uncertainty. A discussion of algorithms for reliable graph clustering may be found in [62]. Uncertain data clustering algorithms are discussed in Chapter 18.

1.4 Insights Gained from Different Variations of Cluster Analysis

While the aforementioned methods discuss the basics of the different clustering algorithms, it is often possible to obtain enhanced insights either by using more rigorous analysis or by incorporating additional techniques or data inputs. These methods are particularly important because of the subjectivity of the clustering process, and the many different ways in which the same data set can be clustered. How do we know that a particular clustering is good or that it solves the needs of the application? There are numerous ways in which these issues can be explored. The exploration could be through interactive visualization and human interaction, external knowledge-based supervision,

explicitly exploring the multiple solutions to evaluate different possibilities, combining the multiple solutions to create more robust ensembles, or trying to judge the quality of different solutions with the use of different validation criteria. For example, the presence of labels adds domain knowledge to the clustering process, which can be used in order to improve insights about the quality of the clustering. This approach is referred to as semisupervision. A different way of incorporating domain knowledge (and indirect supervision) would be for the human to explore the clusters interactively with the use of visual methods and interact with the clustering software in order to create more meaningful clusters. The following subsections will discuss these alternatives in detail.

1.4.1 Visual Insights

Visual analysis of multidimensional data is a very intuitive way to explore the structure of the underlying data, possibly incorporating human feedback into the process. Thus, this approach could be considered an informal type of supervision, when human feedback is incorporated into cluster generation. The major advantage of incorporating human interaction is that a human can often provide intuitive insights, which are not possible from an automated computer program of significant complexity. On the other hand, a computer is much faster at the detailed calculations required both for clustering and for "guessing" the most appropriate feature-specific views of high dimensional data. Thus, a combination of a human and a computer often provides clusters which are superior to those created by either.

One of the most well-known systems for visualization of high-dimensional clusters is the HD-Eye method [44], which explores different subspaces of the data in order to determine clusters in different feature-specific views of the data. Another well-known technique is the *IPCLUS* method [2]. The latter method generates feature-specific views in which the data is *well polarized*. A well-polarized view refers to a 2-dimensional subset of features in which the data clearly separates out into clusters. A kernel-density estimation method is used to determine the views in which the data is well polarized. The final clustering is determined by exploring different views of the data, and counting how the data separates out into clusters in these different views. This process is essentially an ensemble-based method, an approach which is used popularly in the clustering literature, and will be discussed in a later part of this section. Methods for both incorporating and extracting visual insights from the clustering process are discussed in Chapter 19.

1.4.2 Supervised Insights

The same data set may often be clustered in multiple ways, especially when the dimensionality of the data set is high and subspace methods are used. Different features may be more relevant to different kinds of applications and insights. Since clustering is often used as an intermediate step in many data mining algorithms, it then becomes difficult to choose a particular kind of clustering that may suit that application. The subjectiveness of clustering is highly recognized, and small changes in the underlying algorithm or data set may lead to significant changes in the underlying clusters. In many cases, this subjectiveness also implies that it is difficult to refer to one particular clustering as significantly better than another. It is here that supervision can often play an effective role, because it takes the specific goal of the analyst into consideration.

Consider a document clustering application, in which a web portal creator (analyst) wishes to segment the documents into a number of categories. In such cases, the analyst may already have an approximate idea of the categories in which he is interested, but he may not have *fully* settled on a particular set of categories. This is because the data may also contain as yet unknown categories in which the analyst is interested. In such cases, semisupervision is an appropriate way to approach the problem. A number of labeled examples are provided, which approximately represent the categories in which the analyst is interested. This is used as domain knowledge or prior knowledge,

which is used in order to supervise the clustering process. For example, a very simple form of supervision would be to use seeding, in which the documents of the appropriate categories are provided as (some of the) seeds to a representative clustering algorithm such as k-means. In recent years, spectral methods have also been heavily adapted for the problem of semisupervised clustering. In these methods, Laplacian smoothing is used on the labels to generate the clusters. This allows the learning of the lower dimensional data surface in a semisupervised way, as it relates to the underlying clusters. The area of semisupervised clustering is also sometimes referred to as constrained clustering. An excellent discussion on constrained clustering algorithms may be found in [18]. A number of interesting methods for semisupervised clustering are discussed in Chapter 20, with a special focus on the graph-based algorithms.

1.4.3 Multiview and Ensemble-Based Insights

As discussed above, one of the major issues in the clustering process is that different kinds of clusters are possible. When no supervision is available, the bewildering number of possibilities in the clustering process can sometimes be problematic for the analyst, especially from the perspective of interpretability. These are referred to as alternative clusters, and technically represent the behavior from different perspectives. In many cases, the ability to provide different clustering solutions that are significantly different provides insights to the analyst about the key clustering properties of the underlying data. This broader area is also referred to as multiview clustering.

The most naive method for multiview clustering is to simply run the clustering algorithm multiple times, and then examine the different clustering solutions to determine those which are different. A somewhat different approach is to use spectral methods in order to create approximately orthogonal clusters. Recall that the eigenvectors of the Laplacian matrix represent alternative cuts in the graph and that the small eigenvectors represent the best cuts. Thus, by applying a 2-means algorithm to the embedding on each eigenvector, it is possible to create a clustering which is very different from the clustering created by other (orthogonal) eigenvectors. The orthogonality of the eigenvectors is important, because it implies that the embedded representations are very different. Furthermore, the smallest eigenvectors represent the best clusterings, whereas clusterings derived from successively larger eigenvectors represent successively suboptimal solutions. Thus, this approach not only provides alternative clusterings which are quite different from one another, but also provides a ranking of the quality of the different solutions. A discussion of alternative clustering methods is provided in Chapter 21.

In many cases, the alternative clustering methods can be combined to create more robust solutions with the use of ensemble-based techniques. The idea here is that a combination of the output of the different clusterings provides a more robust picture of how the points are related to one another. Therefore, the outputs of the different alternatives can be used as input to a meta-algorithm which combines the results from the different algorithms. Such an approach provides a more robust clustering solution. A discussion of ensemble-based methods for clustering is provided in Chapter 22.

1.4.4 Validation-Based Insights

Given a particular clustering, how do we know what the quality of the clustering really is? While one possible approach is to use synthetic data to determine the matching between the input and output clusters, it is not fully satisfying to rely on only synthetic data. This is because the results on synthetic data may often be specific to a particular algorithm and may not be easily generalizable to arbitrary data sets.

Therefore, it is desirable to use validation criteria on the basis of real data sets. The problem in the context of the clustering problem is that the criteria for quality is not quite as crisp as many other data mining problems such as classification, where external validation criteria are available in

the form of labels. Therefore, the use of one or more criteria may inadvertently favor different algorithms. As the following discussion suggests, clustering is a problem in which precise quantification is often not possible because of its unsupervised nature. Nevertheless, many techniques provide a partial understanding of the underlying clusters. Some common techniques for cluster validation are as follows:

- A common method in the literature is to use case studies to illustrate the subjective quality of the clusters. While case studies provide good intuitive insights, they are not particularly effective for providing a more rigorous quantification of the quality. It is often difficult to compare two clustering methods from a quantitative perspective with the use of such an approach.
- Specific measurements of the clusters such as the cluster radius or density may be used in order to provide a measure of quality. The problem here is that these measures may favor different algorithms in a different way. For example, a k-means approach will typically be superior to a density-based clustering method in terms of average cluster radius, but a density-based method may be superior to a k-means algorithms in terms of the estimated density of the clusters. This is because there is a circularity in using a particular criterion to evaluate the algorithm, when the same criterion is used for clustering purposes. This results in a bias during the evaluation. On the other hand, it may sometimes be possible to reasonably compare two different algorithms of a very similar type (e.g., two variations of k-means) on the basis of a particular criterion.
- In many data sets, labels may be associated with the data points. In such cases, cluster quality can be measured in terms of the correlations of the clusters with the data labels. This provides an external validation criterion, if the labels have not been used in the clustering process. However, such an approach is not perfect, because the class labels may not always align with the natural clusters in the data. Nevertheless, the approach is still considered more "impartial" than the other two methods discussed above and is commonly used for cluster evaluation.

A detailed discussion of the validation methods commonly used in clustering algorithms is provided in Chapter 23.

1.5 Discussion and Conclusions

Clustering is one of the most fundamental data mining problems because of its numerous applications to customer segmentation, target marketing, and data summarization. Numerous classes of methods have been proposed in the literature, such as probabilistic methods, distance-based methods, density-based methods, grid-based methods, factorization techniques, and spectral methods. The problem of clustering has close relationships to the problems of dimensionality reduction, especially through projected clustering formulations. High-dimensional data is often challenging for analysis, because of the increasing sparsity of the data. Clustering methods can be viewed as an integration of feature selection/dimensionality reduction methods with clustering.

The increasing advances in hardware technology allow the collection of large amounts of data through in an ever-increasing number of ways. This requires dedicated methods for streaming and distributed processing. Streaming methods typically work with only one pass over the data, and explicitly account for temporal locality in the clustering methods. Big data methods develop distributed techniques for clustering, especially through the *MapReduce* framework. The diversity of different data types significantly adds to the richness of the clustering problems. Many variations and enhancements of clustering such as visual methods, ensemble methods, multiview methods, or

supervised methods can be used to improve the quality of the insights obtained from the clustering process.

Bibliography

[1] C. Aggarwal. *Data Streams: Models and Algorithms*, Springer, 2007.

[2] C. C. Aggarwal. A human-computer interactive system for projected clustering, *ACM KDD Conference*, 2001.

[3] C. Aggarwal. *Managing and Mining Uncertain Data*, Springer, 2009.

[4] C. Aggarwal. Re-designing distance functions and distance-based applications for high dimensional data, *ACM SIGMOD Record*, March 2001.

[5] C. Aggarwal. Towards systematic design of distance functions for data mining applications, *KDD Conference*, 2003.

[6] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. *ACM SIGMOD Conference*, 1998.

[7] C. Aggarwal, J. Han, J. Wang, and P. Yu. A framework for clustering evolving data streams. In *VLDB Conference*, 2003.

[8] C. Aggarwal, A. Hinneburg, and D. Keim. On the surprising behavior of distance metrics in high dimensional space, *ICDT Conference*, 2001.

[9] C. Aggarwal, C. Procopiuc, J. Wolf, P. Yu, and J.-S. Park. Fast algorithms for projected clustering. In *ACM SIGMOD Conference*, 1999.

[10] C. Aggarwal and P. Yu. Finding generalized projected clusters in high dimensional spaces, *ACM SIGMOD Conference*, 2000.

[11] C. Aggarwal and P. Yu. The Igrid index: Reversing the dimensionality curse for similarity indexing in high dimensional space, *KDD Conference*, 2000.

[12] C. Aggarwal and C. Zhai. A survey of text clustering algorithms, *Mining Text Data*, Springer, 2012.

[13] R. Ahuja, T. Magnanti, and J. Orlin. *Network Flows: Theory, Algorithms and Applications*, Prentice Hall, 1993.

[14] M. Anderberg. *Cluster Analysis for Applications*. Academic Press, 1973.

[15] P. Andritsos et al. LIMBO: Scalable clustering of categorical data. *EDBT Conference*, 2004.

[16] A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. I. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):778–785, 1999.

[17] A. Baraldi and P. Blonda. A survey of fuzzy clustering algorithms for pattern recognition. II. *IEEE Transactions on Systems, Man, and Cybernetics, Part B: Cybernetics*, 29(6):786–801, 1999.

[18] S. Basu, I. Davidson, and K. Wagstaff. *Constrained clustering: Advances in theory, Algorithms and applications*, Chapman and Hall, 2008.

[19] M. Belkin and P. Niyogi. Laplacian eigenmaps for dimensionality reduction and data representation, *Neural Computation*, 15(6), 2003.

[20] P. Berkhin. A survey of clustering data mining techniques. *Grouping Multidimensional Data*, pages 25–71, 2006.

[21] K. Beyer, J. Goldstein, U. Shaft, and R. Ramakrishnan. When is nearest neighbor meaningful? *ICDT Conference*, 2001.

[22] D. Blei, A. Ng, and M. Jordan. Latent Dirichlet allocation, *Journal of Machine Learning Research*, 3:993–1022, 2003.

[23] K. Chakrabarti and S. Mehrotra. Local dimensionality reduction: A new approach to indexing high dimensional spaces. *VLDB Conference Proceedings*, pages 89–100, 2000.

[24] A. Clauset, M. E. J. Newman, and C. Moore, Finding community structure in very large networks, *Physical Review E* 70:066111, 2004.

[25] D. Cutting, D. Karger, J. Pedersen, and J. Tukey. Scatter/gather: A cluster-based approach to browsing large document collections. *ACM SIGIR Conference*, pages 318–329,1992.

[26] R. Cordeiro, C. Traina Jr., A. Traina, J. López, U. Kang, and C. Faloutsos. Clustering very large multi-dimensional datasets with mapreduce. In *KDD*, pages 690–698, 2011.

[27] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the EM algorithm, *Journal of the Royal Statistical Society*, B, 39(1):1–38, 1977.

[28] J. Dean and S. Ghemawat. MapReduce: A flexible data processing tool, *Communication of the ACM*, 53: 72–77, 2010.

[29] I. Dhillon. Co-clustering documents and words using bipartite spectral graph partitioning, *ACM KDD Conference*, 2001.

[30] I. Dhillon, S. Mallela, and D. Modha. Information-theoretic co-clustering, *ACM KDD Conference*, 2003.

[31] B. Duran. *Cluster Analysis: A Survey*. Springer-Verlag, 1974.

[32] G. Das and H. Mannila. Context-based similarity measures for categorical databases. *PKDD Conference*, pages 201–210, 2000.

[33] M. B. Eisen, P. T. Spellman, P. O. Brown, and D. Botstein. Cluster analysis and display of genome-wide expression patterns. *Proceedings of the National Academy of Sciences*, 95(25):14863–14868, 1998. `http://rana.lbl.gov/EisenSoftware.htm`

[34] M. Ester, H.-P. Kriegel, J. Sander, and X. Xu. A density-based algorithm for discovering clusters in large spatial databases with noise. *ACM KDD Conference*, pages 226–231, 1996.

[35] M. Filippone, F. Camastra, F. Masulli, and S. Rovetta. A survey of kernel and spectral methods for clustering. *Pattern Recognition*, 41(1):176–190, 2008.

[36] S. Fortunato. Community detection in graphs, *Physics Reports*, 486(3–5):75–174, February 2010.

[37] G. Gan, C. Ma, and J. Wu. *Data Clustering: Theory, Algorithms, and Applications*. SIAM, Society for Industrial and Applied Mathematics, 2007.

[38] V. Ganti, J. Gehrke, and R. Ramakrishnan. CACTUS-clustering categorical data using summaries. *ACM KDD Conference*, 1999.

[39] D. Gibson, J. Kleiberg, and P. Raghavan. Clustering categorical data: An approach based on Dynamical Systems. *VLDB Conference*, 1998.

[40] S. Guha, R. Rastogi, and K. Shim. ROCK: A robust clustering algorithm for categorical attributes. *Information Systems*, 25(5):345–366, 2000.

[41] S. Guha, R. Rastogi, and K. Shim. CURE: An efficient clustering algorithm for large databases. In *ACM SIGMOD Conference*, 1998.

[42] D. Gunopulos and G. Das. Time-series similarity measures, and time series indexing, *ACM SIGMOD Conference*, 2001.

[43] A. Hinneburg, C. Aggarwal, and D. Keim. What is the nearest neighbor in high dimensional spaces, *VLDB Conference*, 2000.

[44] A. Hinneburg, D. Keim, and M. Wawryniuk. Hd-eye: Visual mining of high-dimensional data. *IEEE Computer Graphics and Applications*, 19:22–31, 1999.

[45] T. Hofmann. Probabilistic latent semantic indexing. *ACM SIGIR Conference*, 1999.

[46] A. Jain. Data clustering: 50 years beyond *k*-means. *Pattern Recognition Letters*, 31(8):651–666, 2010.

[47] A. Jain and R. Dubes. *Algorithms for Clustering Data*. Prentice Hall, 1988.

[48] A. Jain, M. Murty, and P. Flynn. Data clustering: A review. *ACM Computing Surveys (CSUR)*, 31(3):264–323, 1999.

[49] D. Jiang, C. Tang, and A. Zhang. Cluster analysis for gene expression data: A survey. *IEEE Transactions on Knowledge and Data Engineering*, 16(11):1370–1386, 2004.

[50] I. Jolliffe. *Principal Component Analysis*, Springer, 2002.

[51] D. R. Karger. Random sampling in cut, flow, and network design problems, *STOC*, pp. 648–657, 1994.

[52] L. Kaufman and P. Rousseeuw. *Finding Groups in Data: An Introduction to Cluster Analysis*. Wiley-Interscience, 2005.

[53] G. Karypis and V. Kumar. Multilevel k-way partitioning scheme for irregular graphs. *Journal of Parallel and Distributed Computing*, 48(1):96–129, 1998.

[54] H.-P. Kriegel, P. Kröger, and A. Zimek. Clustering high-dimensional data: A survey on subspace clustering, pattern-based clustering, and correlation clustering. *ACM Transactions on Knowledge Discovery from Data (TKDD)*, 3(1):1–58, 2009.

[55] D. D. Lee and H. S. Seung. Learning the parts of objects by non-negative matrix factorization, *Nature*, 401:788–791, 1999.

[56] B.W. Kernighan and S. Lin. An efficient heuristic procedure for partitioning graphs, *Bell System Tech. Journal*, 49:291–307, Feb. 1970.

[57] C. Ding, X. He, and H. Simon. On the equivalence of nonnegative matrix factorization and spectral clustering. *SDM Conference*, 2005.

[58] D. Gusfield. *Algorithms for Strings, Trees and Sequences*, Cambridge University Press, 1997.

[59] J.-G. Lee, J. Han, and K.-Y. Whang. Trajectory clustering: A partition-and-group framework. *SIGMOD Conference*, 593–604, 2007.

[60] T. Liao. Clustering of time series data—A survey. *Pattern Recognition*, 38(11):1857–1874, 2005.

[61] H. Liu and H. Motoda. *Computational Methods of Feature Selection*, Chapman and Hall (CRC Press), 2007.

[62] L. Liu, R. Jin, C. Aggarwal, and Y. Shen. Reliable clustering on uncertain graphs, *ICDM Conference*, 2012.

[63] U. von Luxberg. A tutorial on spectral clustering, *Statistics and Computing*, Springer, 2007.

[64] S. Madeira and A. Oliveira. Biclustering algorithms for biological data analysis: A survey. *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 1(1):24–45, 2004.

[65] F. Murtagh. A survey of recent advances in hierarchical clustering algorithms. *The Computer Journal*, 26(4):354–359, 1983.

[66] R. T. Ng and J. Han. CLARANS: A method for clustering objects for spatial data mining. *IEEE Trans. Knowl. Data Eng.*, 14(5):1003–1016, 2002

[67] L. Parsons, E. Haque, and H. Liu. Subspace clustering for high dimensional data: A review. *ACM SIGKDD Explorations*, 6(1):90–105, 2004.

[68] G. Qi, C. Aggarwal, and T. Huang. Online community detection in social sensing. *WSDM Conference*, 2013.

[69] G. Qi, C. Aggarwal, and T. Huang. Community detection with edge content in social media networks, *ICDE Coference*, 2013.

[70] L. R. Rabiner. A tutorial on hidden Markov models and selected applications in speech recognition, *Proceedings of the IEEE*, 77(2):257–285, Feb. 1989.

[71] M. Rege, M. Dong, and F. Fotouhi. Co-clustering documents and words using bipartite isoperimetric graph partitioning. *ICDM Conference*, pp. 532–541, 2006.

[72] S. Schaeffer. Graph clustering. *Computer Science Review*, 1(1):27–64, 2007.

[73] J. Shi and J. Malik. Normalized cuts and image segmentation. *IEEE Transaction on Pattern Analysis and Machine Intelligence*, 2000.

[74] B. W. Silverman. *Density Estimation for Statistics and Data Analysis*. Chapman and Hall, 1986.

[75] H. Schutze and C. Silverstein. Projections for efficient document clustering, *ACM SIGIR Conference*, 1997.

[76] P. Smyth. Clustering sequences with hidden Markov models, *Neural Information Processing*, 1997.

[77] Y. Sun, C. Aggarwal, and J. Han. Relation-strength aware clustering of heterogeneous information networks with incomplete attributes, *Journal of Proceedings of the VLDB Endowment*, 5(5):394–405, 2012.

[78] J. Tenenbaum, V. de Silva, and J. Langford. A global geometric framework for nonlinear dimensionality reduction. *Science*, 290(5500):2319–2323, 2000.

[79] P. Willett. Recent trends in hierarchical document clustering: A critical review. *Information Processing & Management*, 24(5):577–597, 1988.

[80] R. Xu and D. Wunsch. Survey of clustering algorithms. *IEEE Transactions on Neural Networks*, 16(3):645–678, 2005.

[81] R. Xu and D. Wunsch. *Clustering*. Wiley-IEEE Press, 2008.

[82] F. Wang and J. Sun. Distance metric learning in data mining, *SDM Conference (Tutorial)*, 2012.

[83] W. Wang, J. Yang, and R. Muntz. Sting: A statistical information grid approach to spatial data mining. *VLDB Conference*, 1997.

[84] T. White. Hadoop: The Definitive Guide. *Yahoo! Press*, 2011.

[85] J. Yang and W. Wang. CLUSEQ: Efficient and effective sequence clustering, *ICDE Conference*, 2003.

[86] B.-K. Yi, N. D. Sidiropoulos, T. Johnson, H. Jagadish, C. Faloutsos, and A. Biliris. Online data mining for co-evolving time sequences. *ICDE Conference*, 2000.

[87] T. Zhang, R. Ramakrishnan, and M. Livny. BIRCH: An efficient data clustering method for very large databases. In *ACM SIGMOD Conference*, 1996.

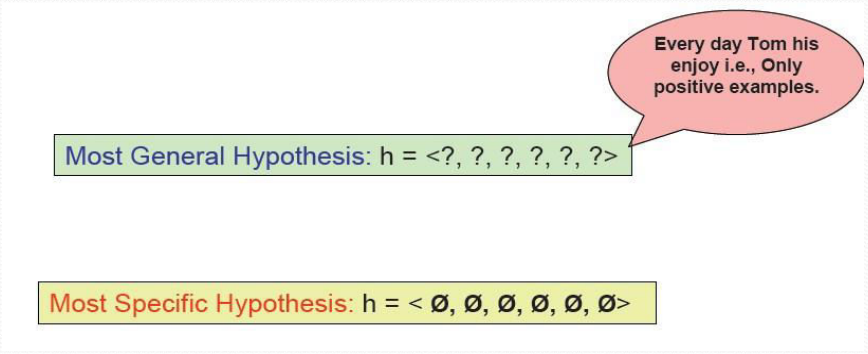
[88] Y. Zhao, G. Karypis, and U. Fayyad. Hierarchical clustering algorithms for document datasets. *Data Mining and Knowledge Discovery*, 10(2):141–168, 2005.

[89] Y. Zhou, H. Cheng, and J. X. Yu, Graph clustering based on structural/attribute similarities, *Proc. VLDB Endow.*, 2(1):718–729, 2009.

[90] Y. Zhu and D. Shasha. StatStream: Statistical monitoring of thousands of data streams in real time. *VLDB Conference*, pages 358–369, 2002.

Concept Learning

General-to-Specific Learning Most



Concept Learning as Search:

- Concept learning can be viewed as the task of searching through a large space of hypothesis implicitly defined by the hypothesis representation.
- The goal of the concept learning search is to find the hypothesis that best fits the training examples.

General-to-Specific Learning:

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

h1=(Sunny,?,?,Strong,?,?)
h2=(Sunny,?,?,?,?,?)

***h2 is more general than h1.**
***h2 imposes fewer constraints on the instance than h1.**

Version Space:

The set of all valid hypotheses provided by an algorithm is called version space (VS)with respect to the hypothesis space H and the given example set D.

Candidate-Elimination Algorithm:

* The Candidate-Elimination algorithm finds all describable hypotheses that are consistent with the observed training examples.
* Hypothesis is derived from examples regardless of whether x is positive or negative example

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

Candidate-Elimination Algorithm:

- It employs a much more compact representation of the version space.
- In this the version space is represented by its most general and least general members (Specific).
- These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

Example :

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	Yes
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

Initialization

G0 ← {<?, ?, ?, ?, ?, ?>}

S0 ← {<Ø, Ø, Ø, Ø, Ø, Ø >}

G0 ← {<?, ?, ?, ?, ?, ?>}

S0 ← {<Ø, Ø, Ø, Ø, Ø, Ø >}

Iteration 1

x1 = <Sunny, Warm, Normal, Strong, Warm, Same>

G1 ← {<?, ?, ?, ?, ?, ?>}

S1 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

Iteration 2

x2 = <Sunny, Warm, High, Strong, Warm, Same>

G2 ← {<?, ?, ?, ?, ?, ?>}

S2 ← {< Sunny, Warm, ?, Strong, Warm, Same >}

G2 ← {<?, ?, ?, ?, ?, ?>}

S2 ← {< Sunny, Warm, ?, Strong, Warm, Same >}

consistent

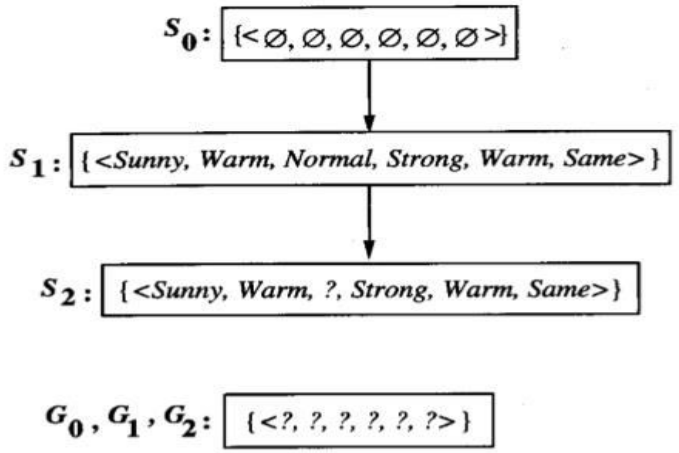
Iteration 3

x3 = <Rainy, Cold, High, Strong, Warm, Change>

S3 ← {< Sunny, Warm, ?, Strong, Warm, Same >}

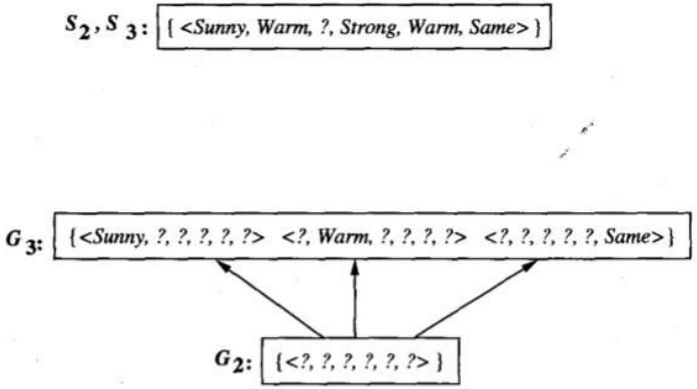
G3 ← {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>, <?, ?, ?, ?, Same>}

G2 ← {<?, ?, ?, ?, ?, ?>}



Training examples:

1. <Sunny, Warm, Normal, Strong, Warm, Same>, Enjoy Sport = Yes
2. <Sunny, Warm, High, Strong, Warm, Same>, Enjoy Sport = Yes



Training Example:

3. <Rainy, Cold, High, Strong, Warm, Change>, EnjoySport=No

CANDIDATE-ELIMINATION Trace 2. Training example 3 is a negative example that forces the G_2 boundary to be specialized to G_3. Note several alternative maximally general hypotheses are included in G_3.

S3 ← {< Sunny, Warm, ?, Strong, Warm, Same >}

G3 ← {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>, <?, ?, ?, ?, Same>}

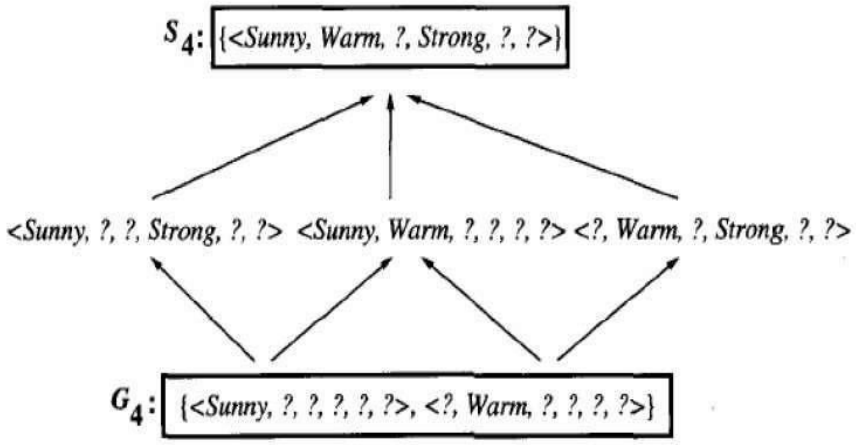
Iteration 4

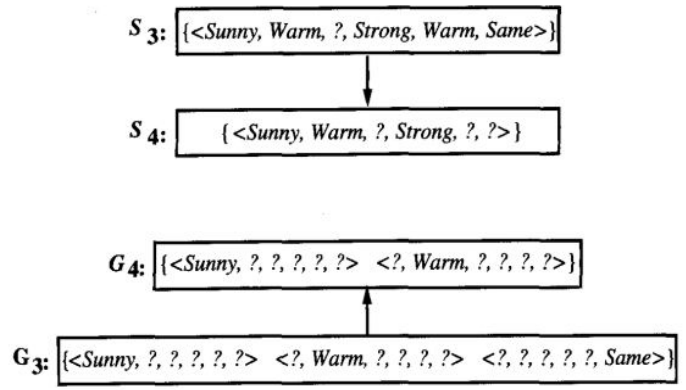
x4 = <Sunny, Warm, high, Strong, Cool, Change>

S4 ← {< Sunny, Warm, ?, Strong, ?, ? >}

G4 ← {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>}

G3 ← {<Sunny, ?, ?, ?, ?, ?>, <?, Warm, ?, ?, ?, ?>, <?, ?, ?, ?, Same>}





Training Example:

4. <*Sunny, Warm, High, Strong, Cool, Change*>, *EnjoySport* = *Yes*

CANDIDATE-ELIMINATION Trace 3. The positive training example generalizes the S boundary, from S_3 to S_4. One member of G_3 must also be deleted, because it is no longer more general than the S_4 boundary.

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

What will Happen if the Training Contains errors ?

Example	Sky	AirTemp	Humidity	Wind	Water	Forecast	EnjoySport
1	Sunny	Warm	Normal	Strong	Warm	Same	Yes
2	Sunny	Warm	High	Strong	Warm	Same	No
3	Rainy	Cold	High	Strong	Warm	Change	No
4	Sunny	Warm	High	Strong	Cool	Change	Yes

G0 ← {<?, ?, ?, ?, ?, ?>}

S0 ← {<Ø, Ø, Ø, Ø, Ø, Ø >}

Iteration 1

x1 = <Sunny, Warm, Normal, Strong, Warm, Same>

G1 ← {<?, ?, ?, ?, ?, ?>}

S1 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

Iteration 2

x2 = <Sunny, Warm, High, Strong, Warm, Same>

G2 ← {<?, ?, Normal, ?, ?, ?>}

S2 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

G2 ← {<?, ?, Normal, ?, ?, ?>}

S2 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

consistent

Iteration 3

x3 = <Rainy, Cold, High, Strong, Warm, Change>

S3 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

G3 ← {<?, ?, Normal, ?, ?, ?>}

S3 ← {< Sunny, Warm, Normal, Strong, Warm, Same >}

G3 ← {<?, ?, Normal, ?, ?, ?>}

Iteration 4

x4 = <Sunny, Warm, high, Strong, Cool, Change>

S4 ← { }

G4 ← { }

Empty

G3 ← {<?, ?, Normal, ?, ?, ?>}

Concept Learning

Concept Learning

- We incrementally learn general concepts from specific training examples.
- Each concept can be viewed as **describing some subset of objects or events defined over a larger set.**
- Concept : A boolean-valued function defined over a larger set.
  - E.g.: **A function defined over all animals, whose value is true for birds and false for other animals**.

Concept Learning

- **Concept learning** is the task of automatically inferring the general definition of some concept, given examples labelled as members or non members of the concept.
  - **Inferring a boolean-valued function from training examples of its input and output**.

Concept Learning - Task

- Example: Task of learning the target concept "**Days on which my friend Aldo enjoys his favorite water sport"**

Positive and negative training examples for the target concept "***EnjoySport***"

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

- The attribute ***EnjoySport*** indicates whether or not Aldo enjoys his favorite water sport on this day.
- **The task is to learn to predict the value of *EnjoySport* for an arbitrary day, based on the values of its other attributes.**

Hypothesis Representation

- Hypothesis : h, 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).***
- Each constraint can be
  - A specific value (e.g., Water = Warm)
  - Don't care (e.g., Water = ?)
  - No value allowed (e.g., Water = *"ϕ"*)
- Example: Sky, AirTemp, Humidity, Wind, Water, Forecast
  < Sunny , ? , High, ? , ? , ? >

Hypothesis Representation

- If some instance x satisfies all the constraints of hypothesis h, then **h classifies x as a positive example (h(x) = 1)**.
- Example: The hypothesis that Aldo **enjoys his favorite sport only on cold days with high humidity (independent of the values of the other attributes)** is represented by the expression

  < ?, Cold**,** High, ?, ?, ? >

Hypothesis Representation

- 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

  < *ϕ, ϕ, ϕ, ϕ, ϕ, ϕ* **>**

Hypothesis Representation

- In general, any concept learning task can be described by
  - the **set of instances** over which the target function is defined
  - the **target function**
  - the **set of candidate hypotheses** considered by the learner, and
  - the **set of available training examples**

Prototypical Concept Learning Task

- **Given:**
  - Instances X: Possible days, each described by the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, *Forecast*
  - Target function c: $EnjoySport : X \rightarrow \{0, 1\}$
  - Hypotheses H: Conjunctions of literals. E.g.

    $$\langle ?, Cold, High, ?, ?, ? \rangle.$$

  - Training examples D: Positive and negative examples of the target function

    $$\langle x_1, c(x_1) \rangle, \ldots \langle x_m, c(x_m) \rangle$$

- **Determine:** A hypothesis h in H such that $h(x) = c(x)$ for all x in D.

Concept Learning Task- EnjoySport Prototype

- **Given:**
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values *Sunny*, *Cloudy*, and *Rainy*),
    - *AirTemp* (with values *Warm* and *Cold*),
    - *Humidity* (with values *Normal* and *High*),
    - *Wind* (with values *Strong* and *Weak*),
    - *Water* (with values *Warm* and *Cool*), and
    - *Forecast* (with values *Same* and *Change*).
  - Hypotheses H: Each hypothesis is described by a conjunction of constraints on the attributes *Sky*, *AirTemp*, *Humidity*, *Wind*, *Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Ø" (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.

Concept Learning Task- EnjoySport Prototype

- The set of items over which the concept is defined is called the set of instances, which we denote 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 we denote by c.
  - In general, c can be any boolean-valued function defined over the instances X; that is, ***c : X -> {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)

Concept Learning Task- EnjoySport Prototype

- When learning the target concept, the learner is presented a set of **training examples,** each consisting of an instance **x** from X, along with its target concept value **c(x)**
- 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.
- Ordered pair **<x, c(x)> describes the training example** consisting of the instance **x** and its target concept value **c(x).**
- **D** denotes the set of available training examples.

Concept Learning Task- EnjoySport Prototype

- Given a set of training examples of the target concept **c, the problem faced by the learner is to hypothesize, or estimate, c.**
- H denotes the set of **all possible hypotheses** that the learner may consider regarding the identity of the target concept.
- In general, each hypothesis **h** in H represents a boolean-valued function defined over X; that is, **h** : X → {0, 1}.
- The goal of the learner is to find a hypothesis **h** such that **h(x)** = **c(x)** for all **x** in X.

The Inductive Learning Hypothesis

- Learning Task :**To determine a hypothesis h identical to the target concept *c* over 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.
- Assumption : The best hypothesis regarding unseen instances is the hypothesis that best fits the observed training data.
- The inductive learning hypothesis: **Any hypothesis found to approximate the target function well over a sufficiently large set of training examples will also approximate the target function well over other unobserved examples.**

CONCEPT LEARNING AS SEARCH

- Concept Learning : Viewed as the task of searching through a large space of hypotheses implicitly defined by the hypothesis representation.
- Goal of search: Find the hypothesis that best fits the training examples.
  - By selecting a hypothesis representation, the designer of the learning algorithm implicitly defines the space of all hypotheses that the program can ever represent and therefore can ever learn.
- Example: Consider the instance X and hypotheses H in the ***EnjoySport*** learning task.
  - Given that the attribute ***Sky*** has three possible values, and that ***AirTemp, Humidity, Wind, Water,*** and ***Forecast*** each have two possible values, the instance space X contains exactly 3 .2. 2 .2 .2 .2 = **96** distinct instances.
  - **Further,** there are **5.4.4.4.4.4** = 5120 syntactically distinct hypotheses within H.

CONCEPT LEARNING AS SEARCH

- Every hypothesis containing one or more ϕ symbols represents the empty set of instances (classified as negative)
- Therefore, the number of semantically distinct hypotheses is only **1+ (4.3.3.3.3.3)**
- EnjoySport example is a very simple learning task, with a relatively small, finite hypothesis space.
- Most practical learning tasks involve much larger, sometimes infinite hypothesis spaces.

FIND-S: FINDING A MAXIMALLY specific HYPOTHESIS

- Use the *more_general_than* partial ordering to organize the search for a hypothesis consistent with the observed training examples
  - Begin with the most specific possible hypothesis in H, then generalize this hypothesis each time it fails to cover an observed positive training example.
  - A hypothesis "covers" a positive example if it correctly classifies the example as positive
- FIND-S algorithm
  1. Initialize h to the most specific hypothesis in H
  2. For each positive training instance ***x***

     For each attribute constraint *a,* in h

     If the constraint *a,* is satisfied by ***x*** Then do nothing

     Else replace *a,* in h by the next more general constraint that is satisfied by ***x***
  3. Output hypothesis h

FIND-S: Example

- Illustration: *EnjoySport* task
  - The first step of FIND-S is to initialize *h* to the most specific hypothesis in H

    h ← < ϕ, ϕ, ϕ, ϕ, ϕ, ϕ >
  - Observing the first training example in *EnjoySport* , which is a positive example, it is clear that the hypothesis is too specific.
  - None of the " *ϕ* " constraints in h are satisfied by this example, so each is replaced by the next more general constraint that fits the example; namely, the attribute values for this training example.

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

FIND-S: Example

Hence, ***h ← (Sunny, Warm, Normal, Strong, Warm, Same)***

- This *h* is still very specific; it asserts that all instances are negative except for the single positive training example observed.
- Next, the second training example (positive example) forces the algorithm to further generalize *h,* this time **substituting a "?' in place of any attribute value in h that is not satisfied by the new example**.
- The refined hypothesis in this case is

  h ← (Sunny, Warm, ?, Strong, Warm, Same)
- The third training example (a negative example) the algorithm makes no change to *h.*
- FIND-S algorithm simply *ignores every negative example*, and hence no revision in h, is needed.

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

FIND-S: Example

- The fourth (positive) example leads to a further generalization of h.

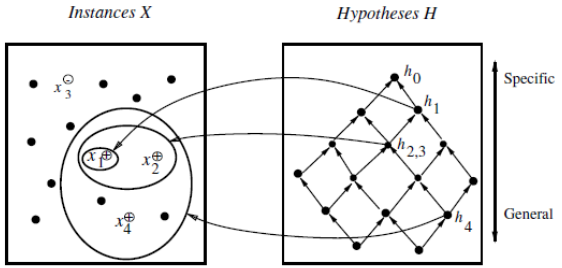
  h ← (Sunny, Warm, ?, Strong, ?, ?)
- The search moves from hypothesis to hypothesis, searching from the most specific to progressively more general hypotheses.

Sky	Temp	Humid	Wind	Water	Forecst	EnjoySpt
Sunny	Warm	Normal	Strong	Warm	Same	Yes
Sunny	Warm	High	Strong	Warm	Same	Yes
Rainy	Cold	High	Strong	Warm	Change	No
Sunny	Warm	High	Strong	Cool	Change	Yes

FIND-S: Example

Search in terms of the instance and hypothesis spaces:

- At each step, the hypothesis is generalized only as far as necessary to cover the new positive example.
- At each stage the hypothesis is the most specific hypothesis consistent with the training examples observed up to this point (hence the name FIND-S).



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

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

FIND-S: Example

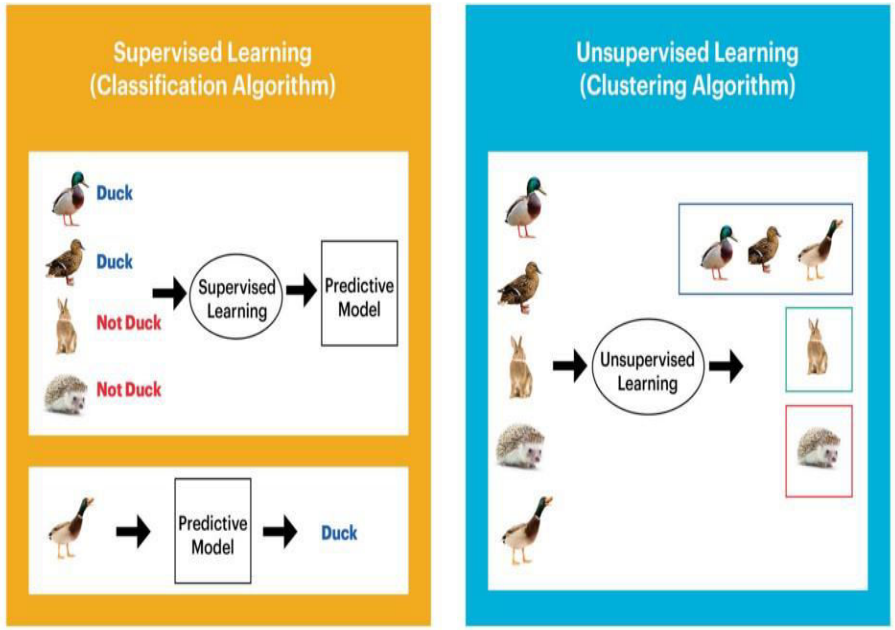
- The key property of the FIND-S algorithm is that for hypothesis spaces described by conjunctions of attribute constraints (such as H for the ***EnjoySport*** task),
  - FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples.
- Its final hypothesis will also be consistent with the negative examples provided the correct target concept is contained in H, and provided the training examples are correct.

Find-S Example

Eyes	Nose	Head	Fcolor	Hair?	Smile?
Round	Triangle	Round	Purple	Yes	Yes
Square	Square	Square	Green	Yes	No
Square	Triangle	Round	Yellow	Yes	Yes
Round	Triangle	Round	Green	No	No
Square	Square	Round	Yellow	Yes	Yes

Supervised vs. Unsupervised Learning

- **Supervised learning (classification)**
  - Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- **Unsupervised learning (clustering)**
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data



Classification vs. Prediction

- **Classification**
  - predicts categorical class labels (**discrete** or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- **Prediction**
  - models **continuous-valued** functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit approval
  - Target marketing
  - Medical diagnosis
  - Fraud detection

CLASSIFICATION

Classification: Definition

- Given a collection of records (*training set*)
  - Each record contains a set of *attributes*, one of the attributes is the *class*.
- Find a *model* for class attribute as a function of the values of other attributes.
- Goal: previously unseen records should be assigned a class as accurately as possible.
  - A *test set* is used to determine the accuracy of the model. Usually, the given data set is divided into training and test sets, with training set used to build the model and test set used to validate it.

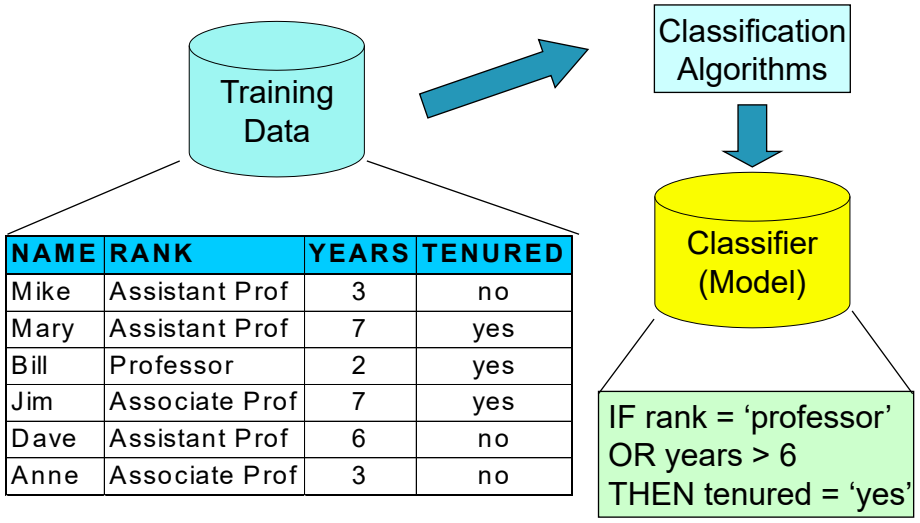
Classification—A Two-Step Process

- **Model construction:** describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as **classification rules, decision trees, or mathematical formulae**

Classification—A Two-Step Process

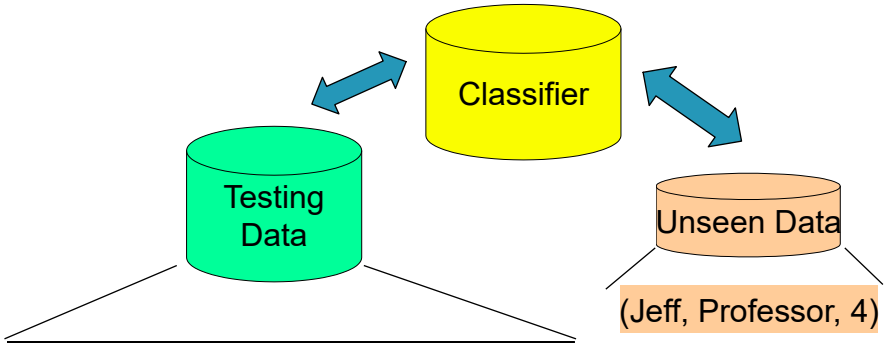
- **Model usage:** for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set, otherwise over-fitting will occur
  - If the accuracy is acceptable, use the model to classify data tuples whose class labels are not known

Classification Process (1): Model Construction



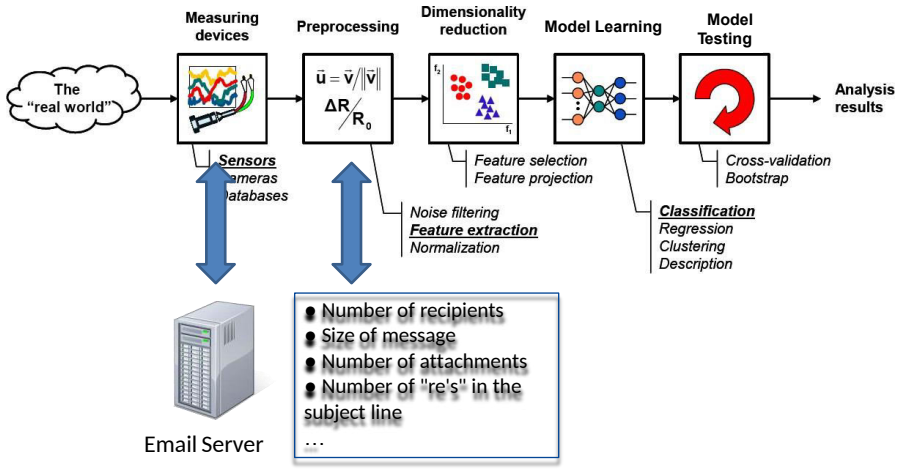
NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

Classification Process (2): Use the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

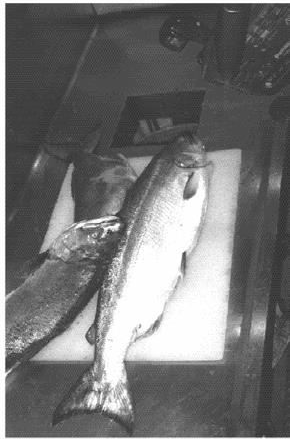
The Learning Process in spam mail Example



An Example

- A fish-packing plant wants to automate the process of sorting incoming fish according to species
- As a pilot project, it is decided to try to separate **sea bass** from **salmon** using optical sensing

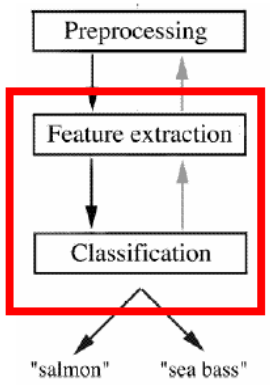
An Example (continued)



Features/attributes:

- Length
- Lightness
- Width
- Position of mouth

An Example (continued)



- **Preprocessing:** Images of different fishes are isolated from one another and from background;
- **Feature extraction:** The information of a single fish is then sent to a feature extractor, that measure certain "features" or "properties";
- **Classification:** The values of these features are passed to a classifier that evaluates the evidence presented, and build a model to discriminate between the two species

An Example (continued)

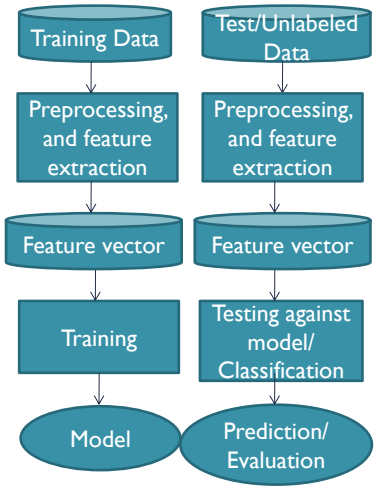
- Domain knowledge:
  - A sea bass is generally longer than a salmon
- Related feature: (or attribute)
  - Length
- Training the classifier:
  - Some examples are provided to the classifier in this form: <fish_length, fish_name>
  - These examples are called training examples
  - The classifier *learns* itself from the training examples, how to distinguish Salmon from Bass based on the *fish_length*

An Example (continued)

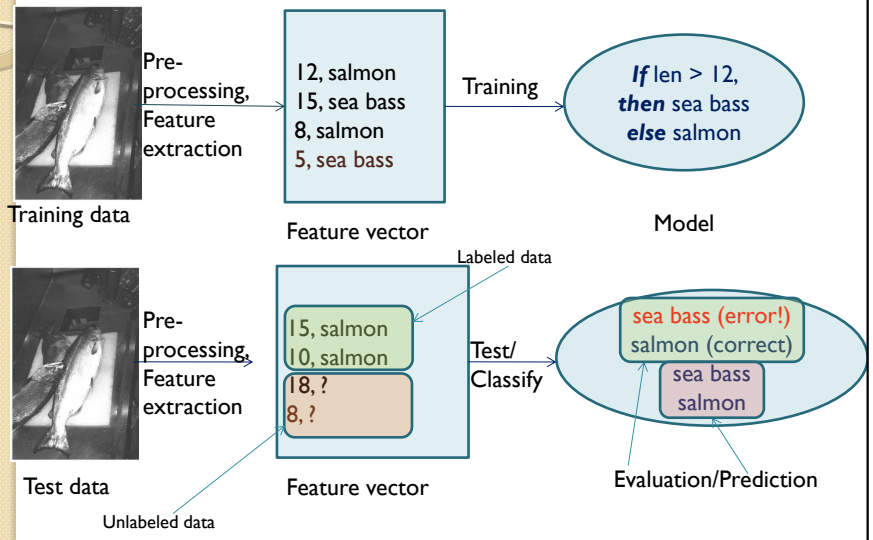
- Classification model (hypothesis):
  - The classifier generates a model from the training data to classify future examples (test examples)
  - An example of the model is a rule like this:
  - If *Length* >= *l* then sea bass* otherwise *salmon*
  - Here the value of *l** determined by the classifier
- Testing the model
  - Once we get a model out of the classifier, we may use the classifier to test future examples
  - The test data is provided in the form <fish_length>
  - The classifier outputs <fish_type> by checking *fish_length* against the model

An Example (continued)

- So the overall classification process goes like this →



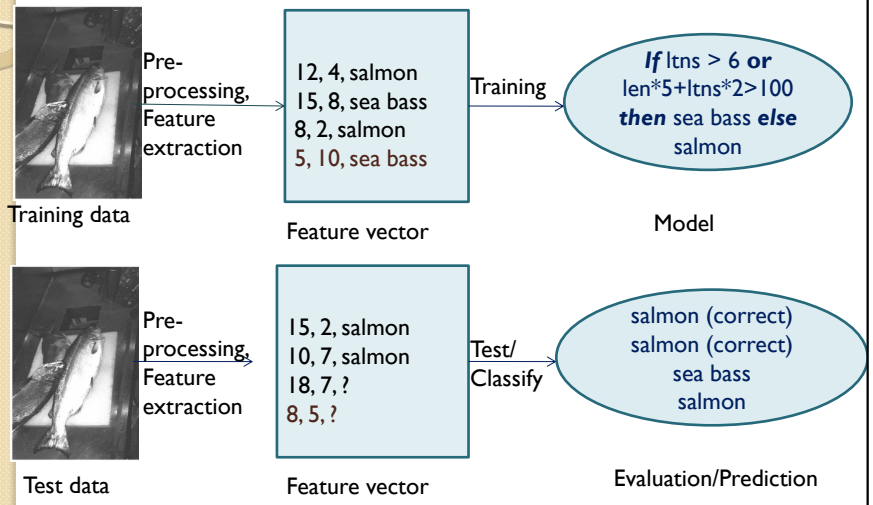
An Example (continued)



An Example (continued)

- Why error?
  - Insufficient training data
  - Too few features
  - Too many/irrelevant features
  - Overfitting / specialization

An Example (continued)



Linear, Non-linear, Multi-class and Multi-label classification

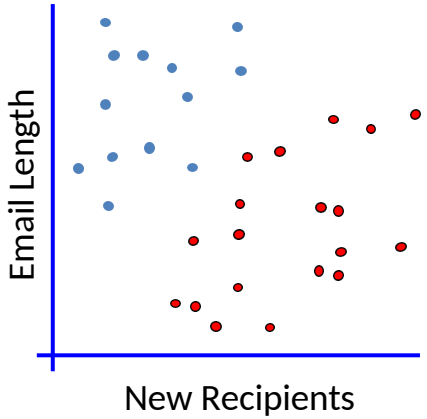
Linear Classification

- A linear classifier achieves this by making a classification decision based on the value of a linear combination of the characteristics.
- A classification algorithm (Classifier) that makes its classification based on a linear predictor function combining a set of weights with the feature vector

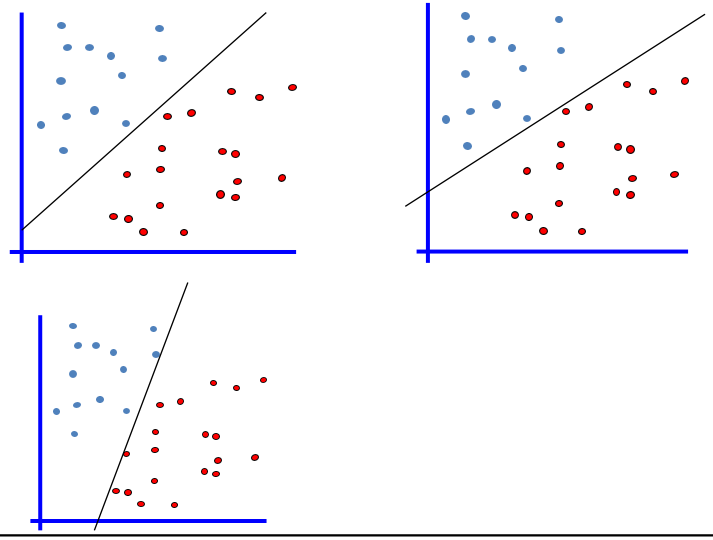
$$y = f(\vec{w} \cdot \vec{x}) = f\left(\sum_j w_j x_j\right),$$

- Decision boundaries is flat
  - Line, plane, ….
- May involve non-linear operations

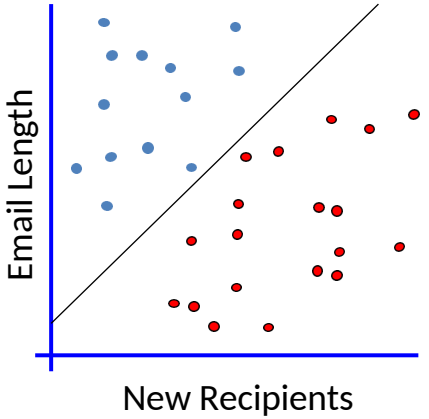
Linear Classifiers



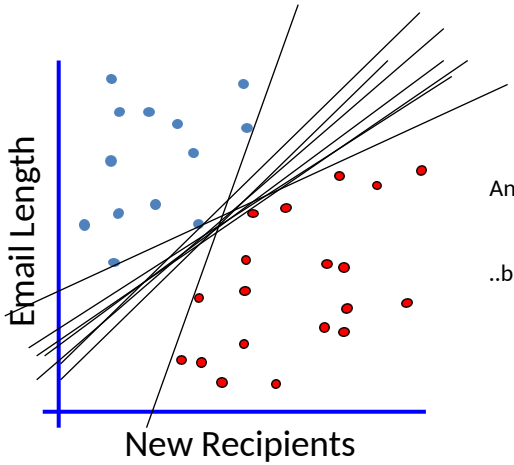
How would you classify this data?



Linear Classifiers



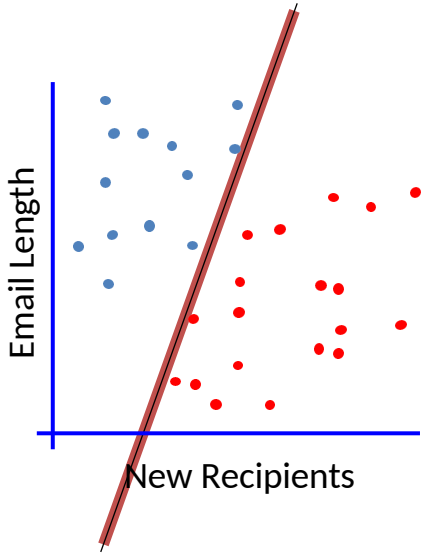
Linear Classifiers



Any of these would be fine..

..but which is best?

Classifier Margin



Define the margin of a linear classifier as the width that the boundary could be increased by before hitting a datapoint.

Acceptance at a University

Test

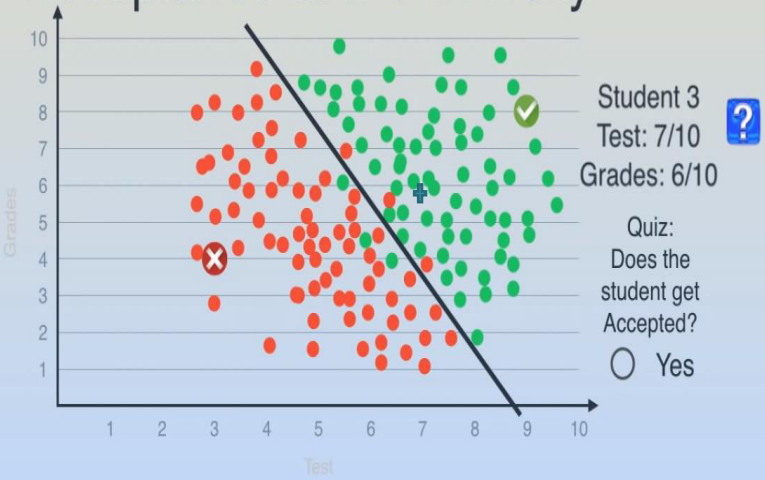
Grades

Student 1
Test: 9/10
Grades: 8/10

Student 2
Test: 3/10
Grades: 4/10

Student 3
Test: 7/10
Grades: 6/10

Acceptance at a University

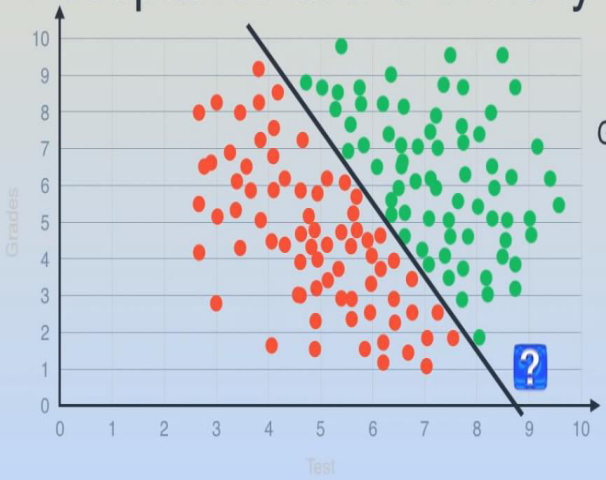


Student 3
Test: 7/10
Grades: 6/10

Quiz:
Does the student get Accepted?

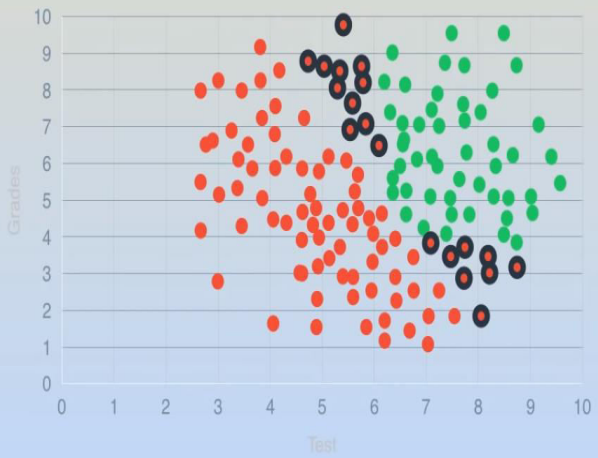
Yes

Acceptance at a University

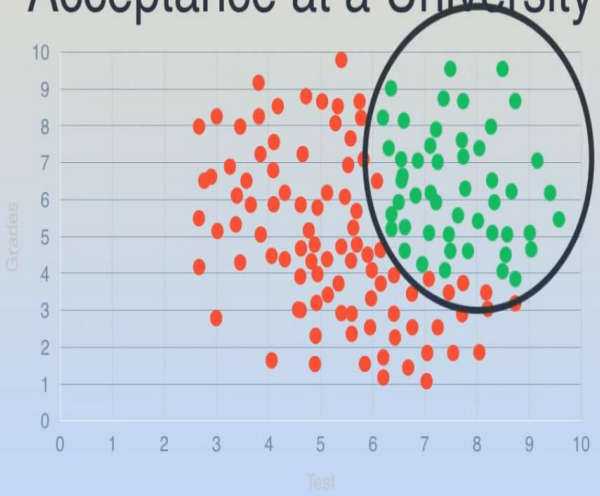


Student 4
Test: 9/10
Grades: 1/10

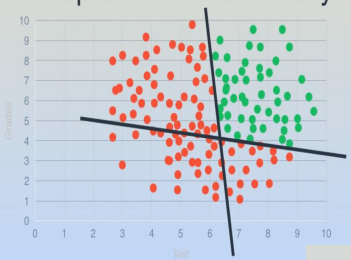
Acceptance at a University

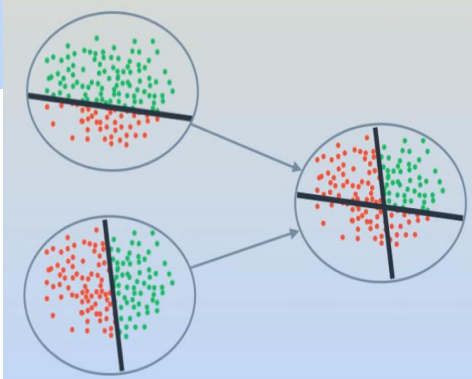


Acceptance at a University



Acceptance at a University





XOR & XNOR Gate: Truth Table & Symbol

XOR / XNOR Tables and Symbols

XOR

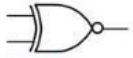
X	Y	X ⊕ Y
0	0	0
0	1	1
1	0	1
1	1	0

XNOR

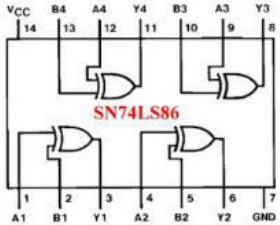
X	Y	$\overline{X \oplus Y}$
0	0	1
0	1	0
1	0	0
1	1	1

XOR Symbol

XNOR Symbol



- The XNOR is also denoted as equivalence

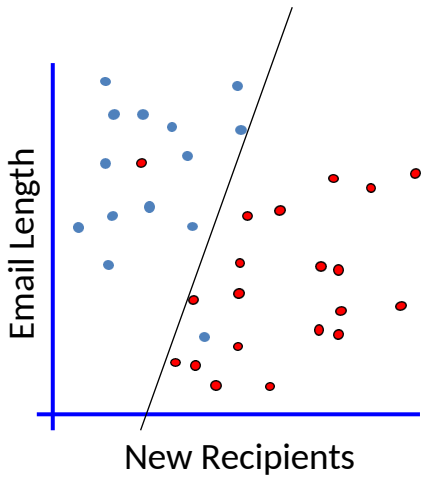


$Y = A \oplus B = \overline{A}B + A\overline{B}$

Inputs		Output
A	B	Y
L	L	L
L	H	H
H	L	H
H	H	L

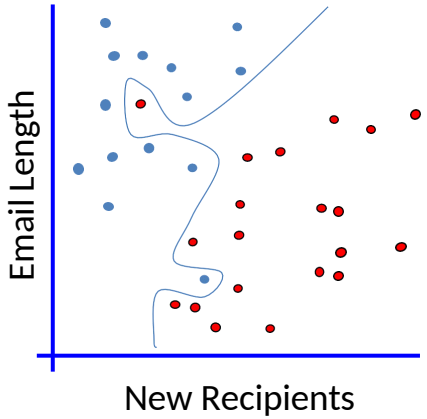
Electrical 4 U

No Linear Classifier can cover all instances



How would you classify this data?

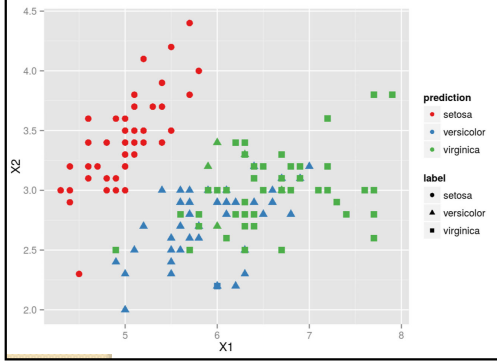
No Linear Classifier can cover all instances



- Ideally, the best decision boundary should be the one which provides an optimal performance such as in the following figure

What is multiclass

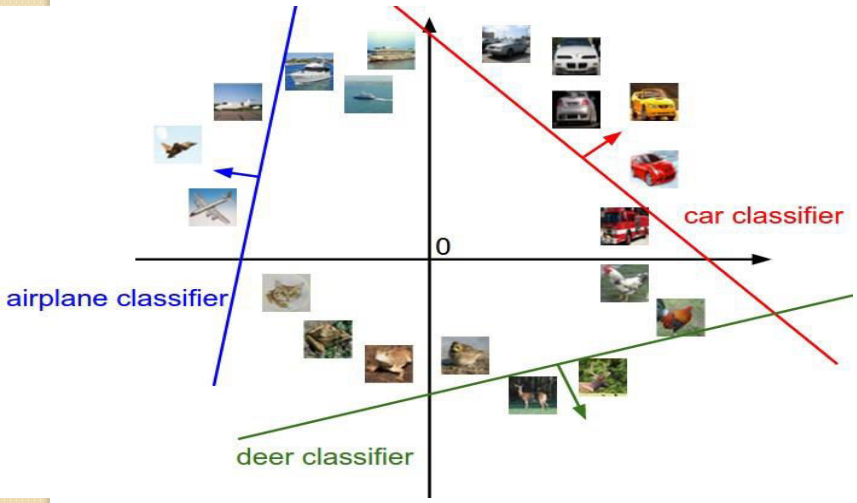
- Output $\in \{1,2,3,\ldots K\}$
  - In some cases, output space can be very large (i.e., K is very large)
- Each input belongs to exactly one class (c.f. in multilabel, input belongs to many classes)



SepalLengthCm	SepalWidthCm	PetalLengthCm	PetalWidthCm	Species
6.8	3.2	5.9	2.3	Iris-virginica
6.9	3.1	5.1	2.3	Iris-virginica
4.9	3.0	1.4	0.2	Iris-setosa
5.6	3.0	4.5	1.5	Iris-versicolor
4.8	3.1	1.6	0.2	Iris-setosa
5.8	2.8	5.1	2.4	Iris-virginica
7.2	3.6	6.1	2.5	Iris-virginica
5.1	3.5	1.4	0.3	Iris-setosa
4.7	3.2	1.6	0.2	Iris-setosa
6.6	3.0	4.4	1.4	Iris-versicolor

Multi-Classes Classification

- Multi-class classification is simply classifying objects into any one of multiple categories. Such as classifying just into either a dog or cat from the dataset.
- *1. When there are more than two categories in which the images can be classified, and*
- *2. An image does not belong to more than one class*
- *If both of the above conditions are satisfied, it is referred to as a multi-class image classification problem*

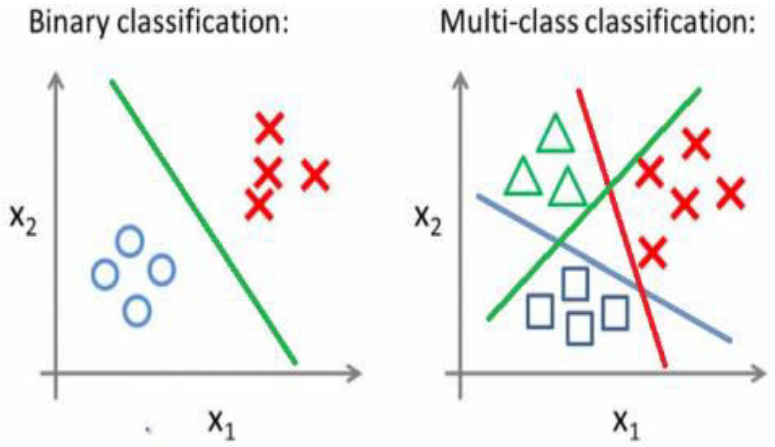


Multi-label classification

- *When we can classify an image into more than one class (as in the image beside), it is known as a multi-label image classification problem.*
- **Multi-label classification** is a type of **classification** in which an object can be categorized into more than one class.
- For example, In the image dataset, we will **classify** a picture as the **image** of a dog or cat and also **classify** the same **image** based on the breed of the dog or cat

These are all labels of the given images. **Each image here belongs to more than one class and hence it is a multi-label image classification problem.**

Binary Vs Multi-class



Three Type of Classification Tasks

YAHOO! JAPAN

Binary Classification	Multiclass Classification	Multi-label Classification
		
• Spam • Not spam	• Dog • Cat • Horse • Fish • Bird • ...	• Dog • Cat • Horse • Fish • Bird • ...

Decision Tree Induction (ID3/C4.5)

Multi class Vs multi label classification

Table 1

X	y
X_1	t_1
X_2	t_2
X_3	t_1
X_4	t_2
X_5	t_1

Binary Classification

Table 2

X	y
X_1	t_2
X_2	t_3
X_3	t_4
X_4	t_1
X_5	t_3

Multi-class Classification

Table 3

X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_2, t_3, t_4]$
X_3	$[t_3]$
X_3	$[t_2, t_4]$
X_3	$[t_1, t_3, t_4]$

Multi-label Classification

Supervised vs. Unsupervised Learning

- Supervised learning (classification)
  - Supervision: The training data (observations, measurements, etc.) are accompanied by **labels** indicating the class of the observations
  - New data is classified based on the training set
- Unsupervised learning (clustering)
  - The class labels of training data is unknown
  - Given a set of measurements, observations, etc. with the aim of establishing the existence of classes or clusters in the data

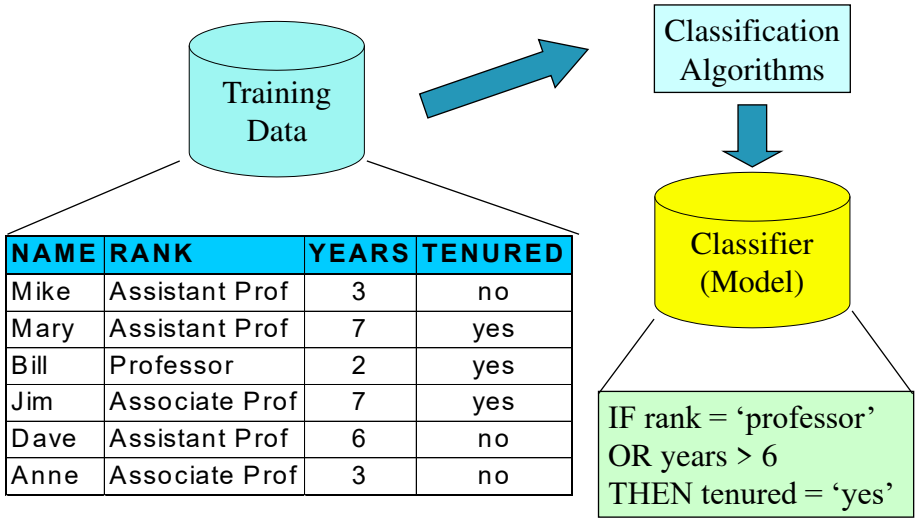
Prediction Problems: Classification vs. Numeric Prediction

- Classification
  - predicts categorical class labels (discrete or nominal)
  - classifies data (constructs a model) based on the training set and the values (class labels) in a classifying attribute and uses it in classifying new data
- Numeric Prediction
  - models continuous-valued functions, i.e., predicts unknown or missing values
- Typical applications
  - Credit/loan approval:
  - Medical diagnosis: if a tumor is cancerous or benign
  - Fraud detection: if a transaction is fraudulent
  - Web page categorization: which category it is

Classification—A Two-Step Process

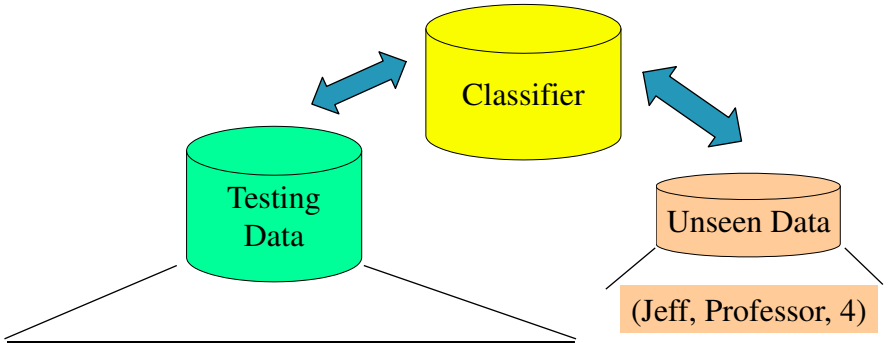
- Model construction: describing a set of predetermined classes
  - Each tuple/sample is assumed to belong to a predefined class, as determined by the class label attribute
  - The set of tuples used for model construction is training set
  - The model is represented as classification rules, decision trees, or mathematical formulae
- Model usage: for classifying future or unknown objects
  - Estimate accuracy of the model
    - The known label of test sample is compared with the classified result from the model
    - Accuracy rate is the percentage of test set samples that are correctly classified by the model
    - Test set is independent of training set (otherwise overfitting)
  - If the accuracy is acceptable, use the model to classify new data
- Note: If *the test set* is used to select models, it is called validation (test) set

Process (1): Model Construction



NAME	RANK	YEARS	TENURED
Mike	Assistant Prof	3	no
Mary	Assistant Prof	7	yes
Bill	Professor	2	yes
Jim	Associate Prof	7	yes
Dave	Assistant Prof	6	no
Anne	Associate Prof	3	no

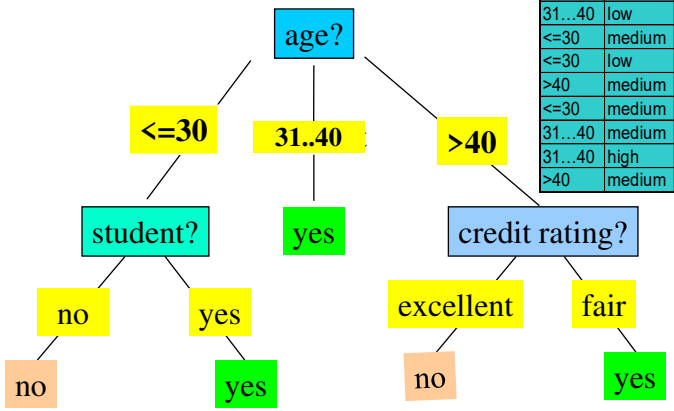
Process (2): Using the Model in Prediction



NAME	RANK	YEARS	TENURED
Tom	Assistant Prof	2	no
Merlisa	Associate Prof	7	no
George	Professor	5	yes
Joseph	Assistant Prof	7	yes

Decision Tree Induction: An Example

- Training data set: Buys_computer
- Resulting tree:



age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Algorithm for Decision Tree Induction

- Basic algorithm (a greedy algorithm)
  - Tree is constructed in a top-down recursive divide-and-conquer manner
  - At start, all the training examples are at the root
  - Attributes are categorical (if continuous-valued, they are discretized in advance)
  - Examples are partitioned recursively based on selected attributes
  - Test attributes are selected on the basis of a heuristic or statistical measure (e.g., information gain)
- Conditions for stopping partitioning
  - All samples for a given node belong to the same class
  - There are no remaining attributes for further partitioning – majority voting is employed for classifying the leaf
  - There are no samples left

Attribute Selection Measure: Information Gain (ID3/C4.5)

- Select the attribute with the highest information gain
- Let p_i be the probability that an arbitrary tuple in D belongs to class C_i, estimated by $|C_{i,D}|/|D|$
- Expected information (entropy) needed to classify a tuple in D:

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

- Information needed (after using A to split D into v partitions) to classify D:

$$Info_A(D) = \sum_{j=1}^{v} \frac{|D_j|}{|D|} \times Info(D_j)$$

- Information gained by branching on attribute A

$$Gain(A) = Info(D) - Info_A(D)$$

Attribute Selection: Information Gain

- Class P: buys_computer = "yes"
- Class N: buys_computer = "no"

$$Info(D) = -\sum_{i=1}^{m} p_i \log_2(p_i)$$

$$Info(D) = I(9,5) = -\frac{9}{14}\log_2(\frac{9}{14}) - \frac{5}{14}\log_2(\frac{5}{14}) = 0.940$$

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

age	p_i	n_i	$I(p_i, n_i)$
<=30	2	3	0.971
31…40	4	0	0
>40	3	2	0.971

age	income	student	credit_rating	buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

$$Info_{age}(D) = \frac{5}{14} I(2,3) + \frac{4}{14} I(4,0) + \frac{5}{14} I(3,2) = 0.694$$

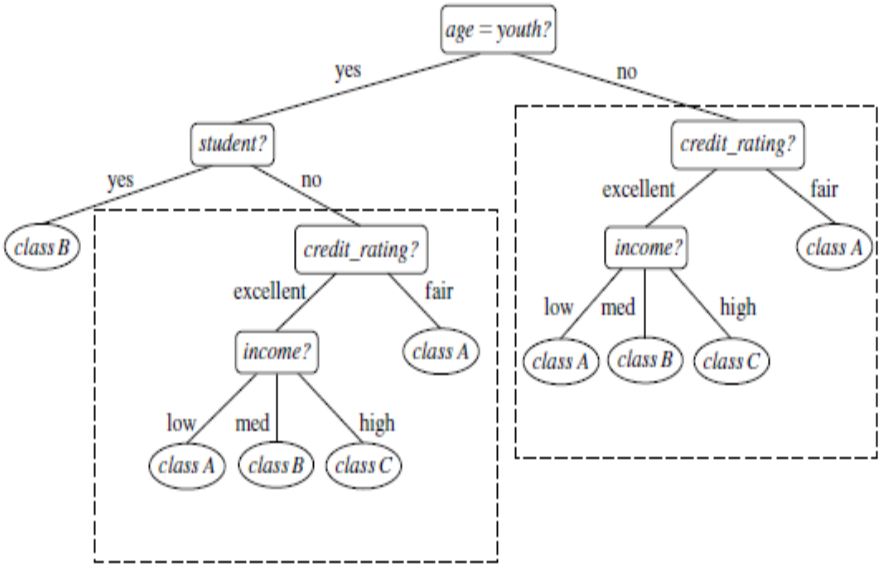
$\frac{5}{14} I(2,3)$ means "age <=30" has 5 out of 14 samples, with 2 yes'es and 3 no's. Hence

$$Gain(age) = Info(D) - Info_{age}(D) = 0.246$$

$Gain(age) = Info(D) - Info_{age}(D) = 0.940 - 0.694 = 0.246$ bits.

$$Info_{age}(D) = \frac{5}{14} \times \left(-\frac{2}{5}\log_2\frac{2}{5} - \frac{3}{5}\log_2\frac{3}{5}\right) + \frac{4}{14} \times \left(-\frac{4}{4}\log_2\frac{4}{4}\right) + \frac{5}{14} \times \left(-\frac{3}{5}\log_2\frac{3}{5} - \frac{2}{5}\log_2\frac{2}{5}\right) = 0.694 \text{ bits.}$$

$$Gain(income) = 0.029$$
$$Gain(student) = 0.151$$
$$Gain(credit\_rating) = 0.048$$



Create a decision tree for the data values given in the table using Entropy (Information Gain) based attribute finding.

Refund	Marital Status	Taxable Income	Cheat
Yes	Single	125K	No
No	Married	100K	No
No	Single	70K	No
Yes	Married	120K	No
No	Divorced	95K	Yes
No	Married	60K	No
Yes	Divorced	220K	No
No	Single	85K	Yes
No	Married	75K	No
No	Single	90K	Yes

Computing Information-Gain for Continuous-Valued Attributes

- Let attribute A be a continuous-valued attribute
- Must determine the *best split point* for A
  - Sort the value A in increasing order
  - Typically, the midpoint between each pair of adjacent values is considered as a possible *split point*
    - $(a_i+a_{i+1})/2$ is the midpoint between the values of a_i and a_{i+1}
  - The point with the *minimum expected information requirement* for A is selected as the split-point for A
- Split:
  - D1 is the set of tuples in D satisfying A ≤ split-point, and D2 is the set of tuples in D satisfying A > split-point

Model Evaluation and Selection

- Evaluation metrics: How can we measure accuracy? Other metrics to consider?
- Use **validation test set** of class-labeled tuples instead of training set when assessing accuracy
- Methods for estimating a classifier's accuracy:
  - Holdout method, random subsampling
  - Cross-validation
  - Bootstrap
- Comparing classifiers:
  - Confidence intervals
  - Cost-benefit analysis and ROC Curves

Comparing Attribute Selection Measures

- The three measures, in general, return good results but
  - **Information gain**:
    - biased towards multivalued attributes
  - **Gain ratio**:
    - tends to prefer unbalanced splits in which one partition is much smaller than the others
  - **Gini index**:
    - biased to multivalued attributes
    - has difficulty when # of classes is large
    - tends to favor tests that result in equal-sized partitions and purity in both partitions

Classifier Evaluation Metrics: Confusion Matrix

Confusion Matrix:

Actual class\Predicted class	C_1	¬ C_1
C_1	**True Positives (TP)**	**False Negatives (FN)**
¬ C_1	**False Positives (FP)**	**True Negatives (TN)**

Example of Confusion Matrix:

Actual class\Predicted class	buy_computer = yes	buy_computer = no	Total
buy_computer = yes	**6954**	**46**	7000
buy_computer = no	**412**	**2588**	3000
Total	7366	2634	10000

- Given *m* classes, an entry, $\boldsymbol{CM_{i,j}}$ in a **confusion matrix** indicates # of tuples in class *i* that were labeled by the classifier as class *j*
- May have extra rows/columns to provide totals

Classifier Evaluation Metrics: Accuracy, Error Rate, Sensitivity and Specificity

A\P	C	¬C	
C	**TP**	**FN**	**P**
¬C	**FP**	**TN**	**N**
	P'	**N'**	**All**

- **Classifier Accuracy,** or recognition rate: percentage of test set tuples that are correctly classified

  Accuracy = (TP + TN)/All
- **Error rate:** *1 – accuracy*, or

  Error rate = (FP + FN)/All
- **Class Imbalance Problem**:
  - One class may be *rare*, e.g. fraud, or HIV-positive
  - Significant *majority of the negative class* and minority of the positive class
  - **Sensitivity**: True Positive recognition rate
    - **Sensitivity = TP/P**
  - **Specificity**: True Negative recognition rate
    - **Specificity = TN/N**

Classifier Evaluation Metrics: Example

Actual Class\Predicted class	cancer = yes	cancer = no	Total	Recognition(%)
cancer = yes	**90**	**210**	300	30.00 (*sensitivity*)
cancer = no	**140**	**9560**	9700	98.56 (*specificity*)
Total	230	9770	10000	96.40 (*accuracy*)

- *Precision* = 90/230 = 39.13% *Recall* = 90/300 = 30.00%

Classifier Evaluation Metrics: Precision and Recall, and F-measures

- **Precision**: exactness – what % of tuples that the classifier labeled as positive are actually positive

$$precision = \frac{TP}{TP + FP}$$

- **Recall:** completeness – what % of positive tuples did the classifier label as positive?

$$recall = \frac{TP}{TP + FN}$$

- Perfect score is 1.0
- Inverse relationship between precision & recall
- ***F* measure (*F_1* or *F*-score)**: harmonic mean of precision and recall,

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

- ***F_β*:** weighted measure of precision and recall
  - assigns ß times as much weight to recall as to precision

$$F_\beta = \frac{(1 + \beta^2) \times precision \times recall}{\beta^2 \times precision + recall}$$

Evaluating Classifier Accuracy: Holdout & Cross-Validation Methods

- **Holdout method**
  - Given data is randomly partitioned into two independent sets
    - Training set (e.g., 2/3) for model construction
    - Test set (e.g., 1/3) for accuracy estimation
  - Random sampling: a variation of holdout
    - Repeat holdout k times, accuracy = avg. of the accuracies obtained
- **Cross-validation** (*k*-fold, where k = 10 is most popular)
  - Randomly partition the data into *k mutually exclusive* subsets, each approximately equal size
  - At *i*-th iteration, use D_i as test set and others as training set
  - Leave-one-out: *k* folds where *k* = # of tuples, for small sized data
  - ***Stratified cross-validation***: folds are stratified so that class dist. in each fold is approx. the same as that in the initial data

Evaluating Classifier Accuracy: Bootstrap

- **Bootstrap**
  - Works well with small data sets
  - Samples the given training tuples uniformly *with replacement*
    - i.e., each time a tuple is selected, it is equally likely to be selected again and re-added to the training set
- Several bootstrap methods, and a common one is **.632 boostrap**
  - A data set with *d* tuples is sampled *d* times, with replacement, resulting in a training set of *d* samples. The data tuples that did not make it into the training set end up forming the test set. About 63.2% of the original data end up in the bootstrap, and the remaining 36.8% form the test set (since $(1 - 1/d)^d \approx e^{-1} = 0.368$)
  - Repeat the sampling procedure *k* times, overall accuracy of the model:

$$Acc(M) = \frac{1}{k}\sum_{i=1}^{k}(0.632 \times Acc(M_i)_{test\_set} + 0.368 \times Acc(M_i)_{train\_set})$$

Estimating Confidence Intervals: Null Hypothesis

- Perform 10-fold cross-validation
- Assume samples follow a **t distribution** with *k–1* **degrees of freedom** (here, *k=10*)
- Use **t-test** (or **Student's t-test**)
- **Null Hypothesis**: M_1 & M_2 are the same
- If we can **reject** null hypothesis, then
  - we conclude that the difference between M_1 & M_2 is **statistically significant**
  - Chose model with lower error rate

Estimating Confidence Intervals: Classifier Models M_1 vs. M_2

- Suppose we have 2 classifiers, M_1 and M_2, which one is better?
- Use 10-fold cross-validation to obtain $\overline{err}(M_1)$ and $\overline{err}(M_2)$
- These mean error rates are just *estimates* of error on the true population of *future* data cases
- What if the difference between the 2 error rates is just attributed to *chance*?
  - Use a **test of statistical significance**
  - Obtain **confidence limits** for our error estimates

Estimating Confidence Intervals: t-test

- If only 1 test set available: **pairwise comparison**
  - For i^{th} round of 10-fold cross-validation, the same cross partitioning is used to obtain $err(M_1)_i$ and $err(M_2)_i$
  - Average over 10 rounds to get $\overline{err}(M_1)$ and $\overline{err}(M_2)$
  - **t-test** computes **t-statistic** with *k-1* **degrees of freedom:**

$$t = \frac{\overline{err}(M_1) - \overline{err}(M_2)}{\sqrt{var(M_1 - M_2)/k}} \quad \text{where}$$

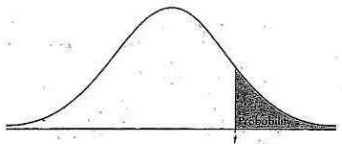
$$var(M_1 - M_2) = \frac{1}{k}\sum_{i=1}^{k}\left[err(M_1)_i - err(M_2)_i - (\overline{err}(M_1) - \overline{err}(M_2))\right]^2$$

- If two test sets available: use **non-paired t-test**

where $var(M_1 - M_2) = \sqrt{\frac{var(M_1)}{k_1} + \frac{var(M_2)}{k_2}}$,

where k_1 & k_2 are # of cross-validation samples used for M_1 & M_2, resp.

Estimating Confidence Intervals: Table for t-distribution



- Symmetric
- **Significance level**, e.g., *sig = 0.05* or *5%* means M_1 & M_2 are *significantly different* for 95% of population
- **Confidence limit**, *z = sig/2*

TABLE B: *t*-DISTRIBUTION CRITICAL VALUES

	Tail probability *p*											
df	.25	.20	.15	.10	.05	.025	.02	.01	.005	.0025	.001	.0005
1	1.000	1.376	1.963	3.078	6.314	12.71	15.89	31.82	63.66	127.3	318.3	636.6
2	.816	1.061	1.386	1.886	2.920	4.303	4.849	6.965	9.925	14.09	22.33	31.60
3	.765	.978	1.250	1.638	2.353	3.182	3.482	4.541	5.841	7.453	10.21	12.92
4	.741	.941	1.190	1.533	2.132	2.776	2.999	3.747	4.604	5.598	7.173	8.610
5	.727	.920	1.156	1.476	2.015	2.571	2.757	3.365	4.032	4.773	5.893	6.869
6	.718	.906	1.134	1.440	1.943	2.447	2.612	3.143	3.707	4.317	5.208	5.959
7	.711	.896	1.119	1.415	1.895	2.365	2.517	2.998	3.499	4.029	4.785	5.408
8	.706	.889	1.108	1.397	1.860	2.306	2.449	2.896	3.355	3.833	4.501	5.041
9	.703	.883	1.100	1.383	1.833	2.262	2.398	2.821	3.250	3.690	4.297	4.781
10	.700	.879	1.093	1.372	1.812	2.228	2.359	2.764	3.169	3.581	4.144	4.587
11	.697	.876	1.088	1.363	1.796	2.201	2.328	2.718	3.106	3.497	4.025	4.437
12	.695	.873	1.083	1.356	1.782	2.179	2.303	2.681	3.055	3.428	3.930	4.318
13	.694	.870	1.079	1.350	1.771	2.160	2.282	2.650	3.012	3.372	3.852	4.221
14	.692	.868	1.076	1.345	1.761	2.145	2.264	2.624	2.977	3.326	3.787	4.140
15	.691	.866	1.074	1.341	1.753	2.131	2.249	2.602	2.947	3.286	3.733	4.073
16	.690	.865	1.071	1.337	1.746	2.120	2.235	2.583	2.921	3.252	3.686	4.015
17	.689	.863	1.069	1.333	1.740	2.110	2.224	2.567	2.898	3.222	3.646	3.965
18	.688	.862	1.067	1.330	1.734	2.101	2.214	2.552	2.878	3.197	3.611	3.922
19	.688	.861	1.066	1.328	1.729	2.093	2.205	2.539	2.861	3.174	3.579	3.883
20	.687	.860	1.064	1.325	1.725	2.086	2.197	2.528	2.845	3.153	3.552	3.850
21	.686	.859	1.063	1.323	1.721	2.080	2.189	2.518	2.831	3.135	3.527	3.819
22	.686	.858	1.061	1.321	1.717	2.074	2.183	2.508	2.819	3.119	3.505	3.792
23	.685	.858	1.060	1.319	1.714	2.069	2.177	2.500	2.807	3.104	3.485	3.768
24	.685	.857	1.059	1.318	1.711	2.064	2.172	2.492	2.797	3.091	3.467	3.745
25	.684	.856	1.058	1.316	1.708	2.060	2.167	2.485	2.787	3.078	3.450	3.725
26	.684	.856	1.058	1.315	1.706	2.056	2.162	2.479	2.779	3.067	3.435	3.707
27	.684	.855	1.057	1.314	1.703	2.052	2.158	2.473	2.771	3.057	3.421	3.690
28	.683	.855	1.056	1.313	1.701	2.048	2.154	2.467	2.763	3.047	3.408	3.674
29	.683	.854	1.055	1.311	1.699	2.045	2.150	2.462	2.756	3.038	3.396	3.659
30	.683	.854	1.055	1.310	1.697	2.042	2.147	2.457	2.750	3.030	3.385	3.646
40	.681	.851	1.050	1.303	1.684	2.021	2.123	2.423	2.704	2.971	3.307	3.551
50	.679	.849	1.047	1.299	1.676	2.009	2.109	2.403	2.678	2.937	3.261	3.496
60	.679	.848	1.045	1.296	1.671	2.000	2.099	2.390	2.660	2.915	3.232	3.460
80	.678	.846	1.043	1.292	1.664	1.990	2.088	2.374	2.639	2.887	3.195	3.416
100	.677	.845	1.042	1.290	1.660	1.984	2.081	2.364	2.626	2.871	3.174	3.390
1000	.675	.842	1.037	1.282	1.646	1.962	2.056	2.330	2.581	2.813	3.098	3.300
∞	.674	.841	1.036	1.282	1.645	1.960	2.054	2.326	2.576	2.807	3.091	3.291
	50%	60%	70%	80%	90%	95%	96%	98%	99%	99.5%	99.8%	99.9%
	Confidence level *C*											

Estimating Confidence Intervals: Statistical Significance

- Are M_1 & M_2 **significantly different**?
  - Compute *t.* Select *significance level* (e.g. *sig = 5%)*
  - Consult table for t-distribution: Find *t value* corresponding to *k-1 degrees of freedom* (here, 9)
  - t-distribution is symmetric: typically upper % points of distribution shown → look up value for **confidence limit** *z=sig/2* (here, 0.025)
  - **If t > z or t < -z**, then t value lies in rejection region:
    - **Reject null hypothesis** that mean error rates of M_1 & M_2 are same
    - Conclude: statistically significant difference between M_1 & M_2
  - **Otherwise**, conclude that any difference is **chance**

Issues Affecting Model Selection

- **Accuracy**
  - classifier accuracy: predicting class label
- **Speed**
  - time to construct the model (training time)
  - time to use the model (classification/prediction time)
- **Robustness**: handling noise and missing values
- **Scalability**: efficiency in disk-resident databases
- **Interpretability**
  - understanding and insight provided by the model
- Other measures, e.g., goodness of rules, such as decision tree size or compactness of classification rules

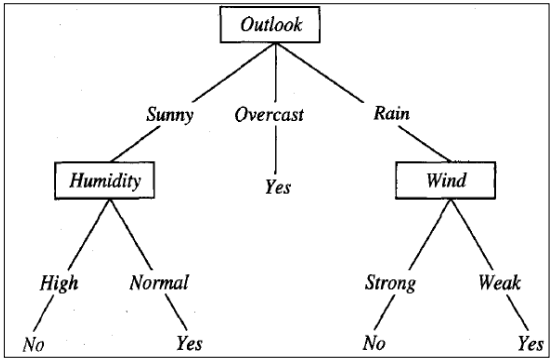
Predictor Error Measures

- Measure predictor accuracy: measure how far off the predicted value is from the actual known value
- **Loss function**: measures the error betw. y_i and the predicted value y_i'
  - Absolute error: $| y_i - y_i' |$
  - Squared error: $(y_i - y_i')^2$
- Test error (generalization error): the average loss over the test set
  - Mean absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{d}$ Mean squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{d}$
  - Relative absolute error: $\frac{\sum_{i=1}^{d} | y_i - y_i' |}{\sum_{i=1}^{d} | y_i - \bar{y} |}$ Relative squared error: $\frac{\sum_{i=1}^{d} (y_i - y_i')^2}{\sum_{i=1}^{d} (y_i - \bar{y})^2}$

The mean squared-error exaggerates the presence of outliers

Popularly use (square) root mean-square error, similarly, root relative squared error

Decision Tree learning: ID3



Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.

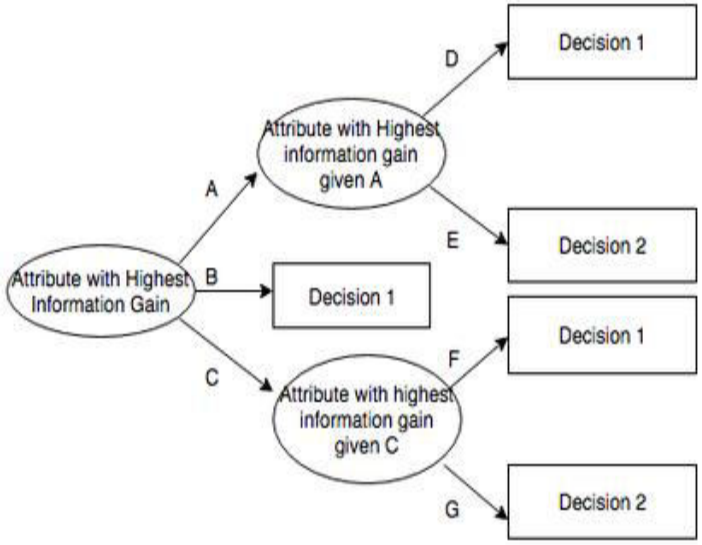
$$(Outlook = Sunny \wedge Humidity = Normal)$$
$$\vee \quad (Outlook = Overcast)$$
$$\vee \quad (Outlook = Rain \wedge Wind = Weak)$$

For "Yes"

- Decision tree representation
  - Objective
    - Classify the instances by sorting them down the tree from the root to some leaf node, which provides the class of the instance
      - Each node tests the attribute of an instance whereas each branch descending from that node specifies one possible value of this attribute
  - Testing
    - Start at the root node, test the attribute specified by this node and identify the appropriate branch then repeat the process
  - Example
    - Decision tree for classifying the Saturday mornings to test whether it is suitable for playing tennis based on the weather attributes

- **The characteristics of best suited problems for DT**
  - Instances that are represented by attribute-value pairs
    - Each attribute takes small number of disjoint possible values (e.g., ***Hot, Mild, Cold)*** → ***discrete***
    - Real-valued attributes like temperature → continuous
  - The target function has discrete output values
    - Binary, multi-class and real-valued outputs
  - Disjunctive descriptions may be required
  - The training data may contain errors
    - DT robust to both errors: errors in classification and attributes
  - The training data may contain missing attribute values

- Basics of DT learning - Iterative Dichotomiser 3 - ID3



Information gain

Information gain $IG(A)$ is the measure of the difference in entropy from before to after the set S is split on an attribute A. In other words, how much uncertainty in S was reduced after splitting set S on attribute A.

$$IG(A,S) = H(S) - \sum_{t \in T} p(t)H(t)$$

Where,

- $H(S)$ – Entropy of set S
- T – The subsets created from splitting set S by attribute A such that $S = \bigcup_{t \in T} t$
- $p(t)$ – The proportion of the number of elements in t to the number of elements in set S
- $H(t)$ – Entropy of subset t

In ID3, information gain can be calculated (instead of entropy) for each remaining attribute. The attribute with the **largest** information gain is used to split the set S on this iteration.

Entropy

Entropy $H(S)$ is a measure of the amount of uncertainty in the (data) set S (i.e. entropy characterizes the (data) set S).

$$H(S) = \sum_{x \in X} -p(x) \log_2 p(x)$$

Where,

- S – The current (data) set for which entropy is being calculated (changes every iteration of the ID3 algorithm)
- X – Set of classes in S
- $p(x)$ – The proportion of the number of elements in class x to the number of elements in set S

When $H(S) = 0$, the set S is perfectly classified (i.e. all elements in S are of the same class).

In ID3, entropy is calculated for each remaining attribute. The attribute with the **smallest** entropy is used to split the set S on this iteration. The higher the entropy, the higher the potential to improve the classification here.

- ID3 algorithm
  - ID3 algorithm begins with the original set S as the root node.
  - On each iteration of the algorithm, it iterates through every unused attribute of the set S and calculates the entropy H(S) (or information gain IG(S)) of that attribute.
  - It then selects the attribute which has the smallest entropy (or largest information gain) value.
    - The set S is then split by the selected attribute (e.g. age is less than 50, age is between 50 and 100, age is greater than 100) to produce subsets of the data.
  - The algorithm continues to recurs on each subset, considering only attributes never selected before.

- Recursion on a subset may stop, When
  - All the elements in the class belong to same class
  - All instances does not belong to same class but there is no attribute to select
  - There is no example in the subset
- Steps in ID3
  - Calculate the entropy of every attribute using the data set S
  - Split the set S into subsets using the attribute for which the resulting entropy (after splitting) is minimum (or, equivalently, information gain is maximum)
  - Make a decision tree node containing that attribute
  - Recurs on subsets using remaining attributes.

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

- Root - Significance
  - Root node
    - Which attribute should be tested at the root of the DT?
      - Decided based on the information gain or entropy
      - Significance
        - » The best attribute that classifies the training samples very well
        - » The attribute that should be tested in all the instances of the dataset
    - @root node
      - Information gain is more or entropy has the least value
    - Entropy measures the homogeneity or impurities in the instances
    - Information gain measures the expected reduction in entropy

DT by example

Target

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

Entropy using frequency table of independent attributes on single dependent attribute
(Target Variable here: PlayGolf)

Training examples: **9 yes / 5 no**

Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

Entropy using frequency table of single attribute on single attribute

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

Play Golf	
Yes	No
9	5

Entropy(PlayGolf) = Entropy (5,9)
= Entropy (0.36, 0.64)
= - (0.36 $\log_2$ 0.36) - (0.64 $\log_2$ 0.64)
= 0.94

5/14

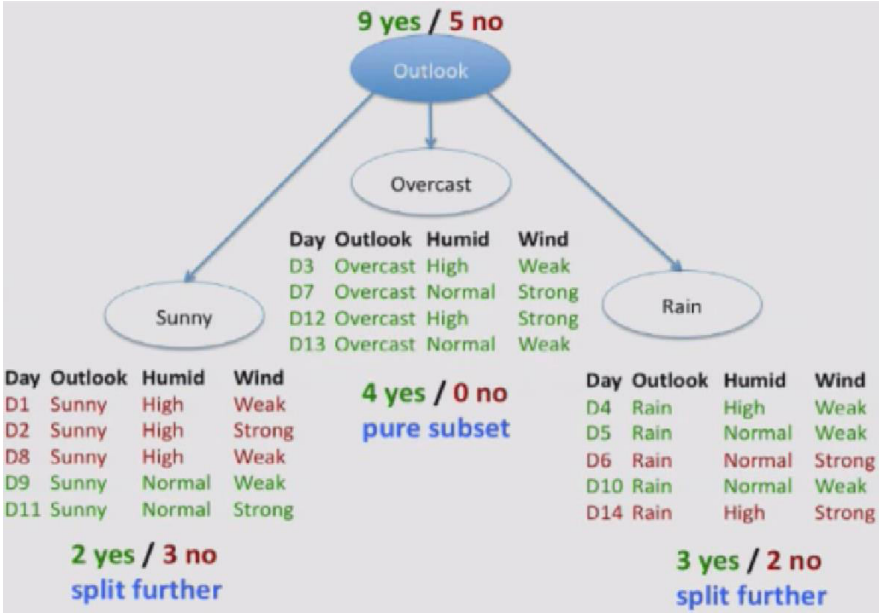
9/14

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

Entropy using frequency table of single attribute on two attribute

PlayGolf on Outlook
PlayGolf on Temperature
PlayGolf on Humidity
PlayGolf on Windy

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



Entropy using frequency table of single attribute on two attribute

$$E(T,X) = \sum_{c \in X} P(c)E(c)$$

= -3/5 $\log_2(3/5)$ – 2/5 $\log_2(2/5)$
=0.971

= -4/4 $\log_2(4/4)$ – 0/4 $\log_2(0/4)$
=0.0

		Play Golf		
		Yes	No	
Outlook	Sunny	3	2	5
	Overcast	4	0	4
	Rainy	2	3	5
				14

= -2/5 $\log_2(2/5)$ – 3/5 $\log_2(3/5)$
=0.971

E(PlayGolf, Outlook) = **P**(Sunny)***E**(3,2) + **P**(Overcast)***E**(4,0) + **P**(Rainy)***E**(2,3)

= (5/14)*0.971 + (4/14)*0.0 + (5/14)*0.971

= 0.693

Information gain calculation

		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

		Play Golf	
		Yes	No
Temp.	Hot	2	2
	Mild	4	2
	Cool	3	1
Gain = 0.029			

		Play Golf	
		Yes	No
Humidity	High	3	4
	Normal	6	1
Gain = 0.152			

		Play Golf	
		Yes	No
Windy	False	6	2
	True	3	3
Gain = 0.048			

$$Gain(T,X) = Entropy(T) - Entropy(T,X)$$

G(PlayGolf, Outlook) = **E**(PlayGolf) – **E**(PlayGolf, Outlook)

= 0.940 – 0.693 = 0.247

Calculation for Humidity, Temperature

Gain(PlayGolf , Humidity)=E(PlayGolf)-E(PlayGolf, Humidity)
= 0.94 - P(High) * E(3,4) - P(Normal) * E(6,1)
= 0.940 - 7/14 *E(3,4) - 7/14*E(6,1)
=0.940 – 0.5 * (3/7*log(3/7) + 4/7 log(4/7)) –
0.5* (1/7log(1/7) + 6/7 log(6/7))
= 0.94 - 0.5 * 0.985 – 0.5 * 0.592 = **0.152**

Gain(PlayGolf ,Temperature)=E(PlayGolf)-E(PlayGolf, Temperature)
=0.94 - P(Hot) * E(2,2) - P(Mild) * E(4,2) – P(Cold) E(3,1)
=0.940 - 4/14 *E(2,2) - 6/14*E(4,2) – 4/14 E(3,1)
=0.940 – 4/14 * (2/4*log(2/4) + 2/4 log(2/4)) – 6/14* (4/6log(4/6) + 2/6 log(2/6)) -
4/14 * (3/4*log(3/4) + 1/4 log(1/4))
= **0.029**

Calculation for windy

$$Values(Wind) = Weak, Strong$$

$$S = [9+, 5-]$$

$$S_{Weak} \leftarrow [6+, 2-]$$

$$S_{Strong} \leftarrow [3+, 3-]$$

$$Gain(S, Wind) = Entropy(S) - \sum_{v \in \{Weak, Strong\}} \frac{|S_v|}{|S|} Entropy(S_v)$$

$$= Entropy(S) - (8/14) Entropy(S_{Weak}) - (6/14) Entropy(S_{Strong})$$

$$= 0.940 - (8/14)0.811 - (6/14)1.00$$

$$= 0.048$$

WINNER is Outlook and chosen as Root node

Choose attribute with the largest information gain as the decision node, divide the dataset by its branches and repeat the same process on every branch.

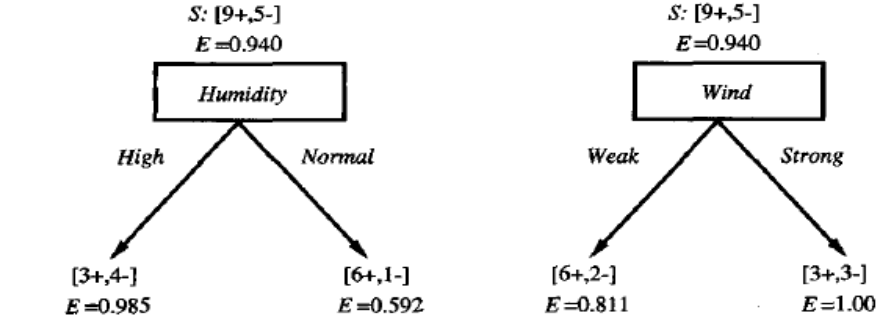
		Play Golf	
		Yes	No
Outlook	Sunny	3	2
	Overcast	4	0
	Rainy	2	3
Gain = 0.247			

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

Training examples: 9 yes / 5 no

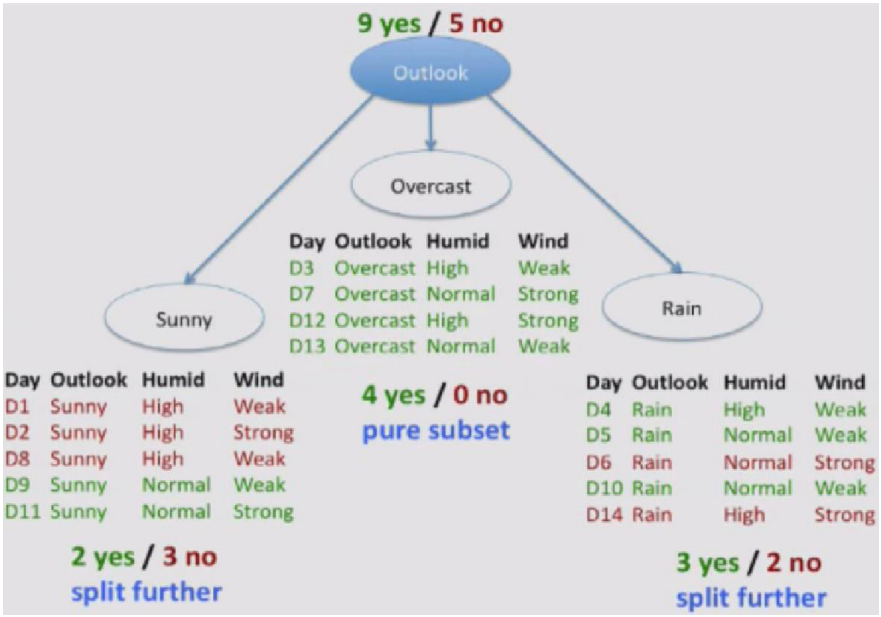
Day	Outlook	Humidity	Wind	Play
D1	Sunny	High	Weak	No
D2	Sunny	High	Strong	No
D3	Overcast	High	Weak	Yes
D4	Rain	High	Weak	Yes
D5	Rain	Normal	Weak	Yes
D6	Rain	Normal	Strong	No
D7	Overcast	Normal	Strong	Yes
D8	Sunny	High	Weak	No
D9	Sunny	Normal	Weak	Yes
D10	Rain	Normal	Weak	Yes
D11	Sunny	Normal	Strong	Yes
D12	Overcast	High	Strong	Yes
D13	Overcast	Normal	Weak	Yes
D14	Rain	High	Strong	No

Which attribute is the best classifier?



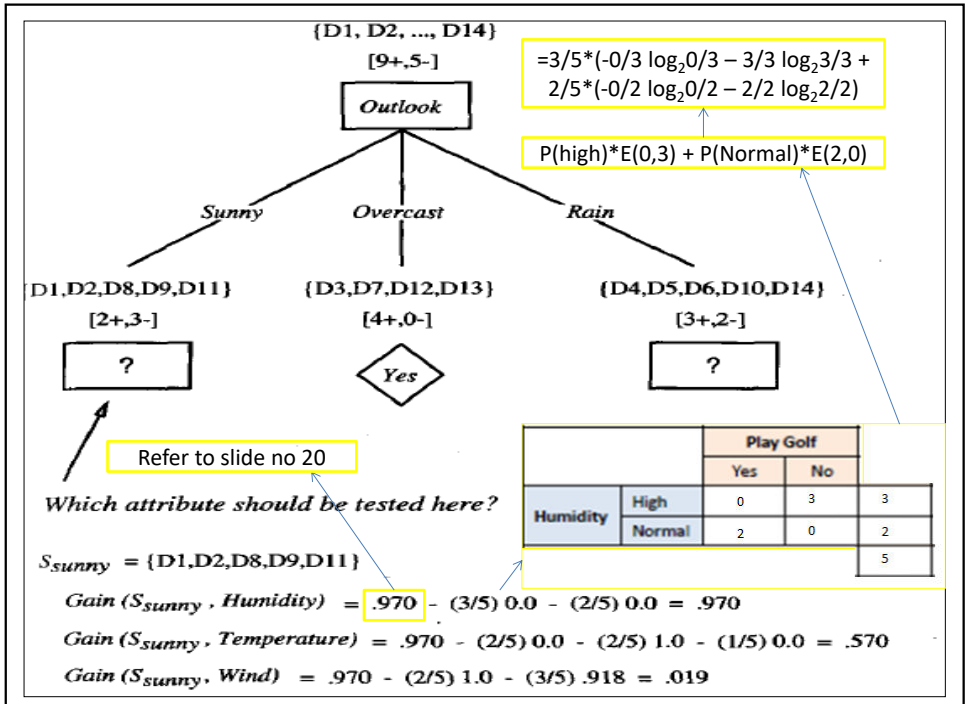
Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

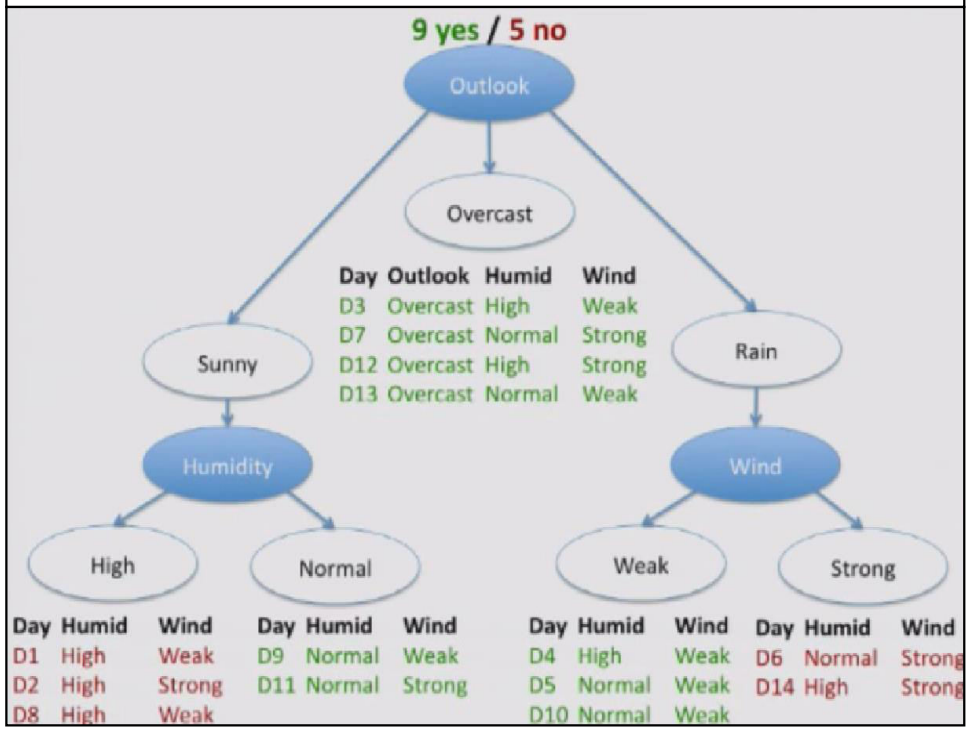
Gain (S, Wind)
= .940 - (8/14).811 - (6/14)1.0
= .048



Choose the other nodes below the root node

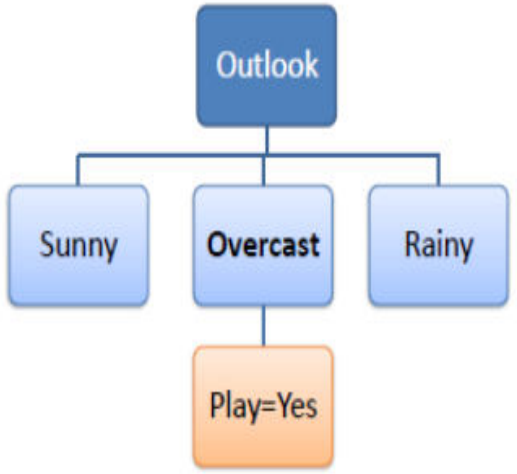
Iterate the same procedure for finding the root node





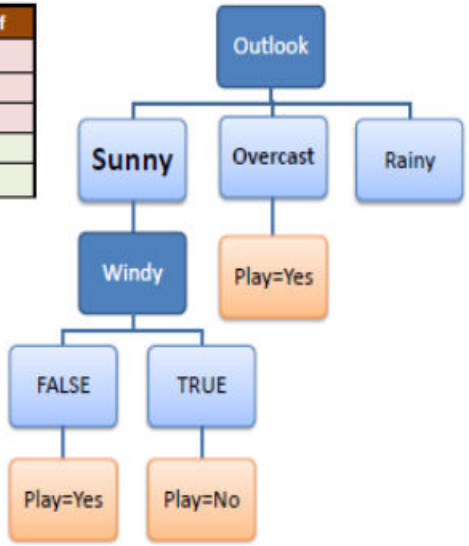
Branch with entropy of ‘0’ is a leaf node

Temp	Humidity	Windy	Play Golf
Hot	High	FALSE	Yes
Cool	Normal	TRUE	Yes
Mild	High	TRUE	Yes
Hot	Normal	FALSE	Yes



Branch with entropy more than '0' needs further splitting

Temp	Humidity	Windy	Play Golf
Mild	High	FALSE	Yes
Cool	Normal	FALSE	Yes
Mild	Normal	FALSE	Yes
Cool	Normal	TRUE	No
Mild	High	TRUE	No



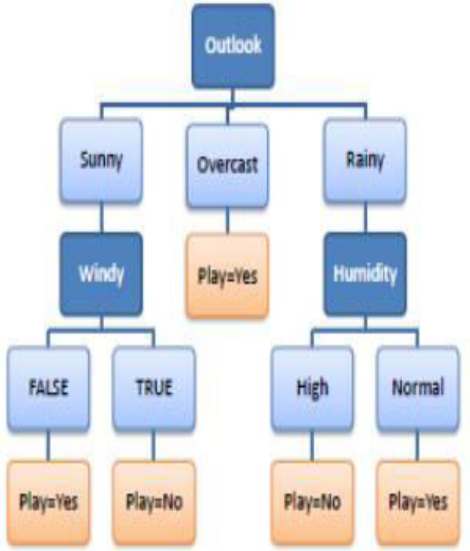
R_1: IF (Outlook=Sunny) AND (Windy=FALSE) THEN Play=Yes

R_2: IF (Outlook=Sunny) AND (Windy=TRUE) THEN Play=No

R_3: IF (Outlook=Overcast) THEN Play=Yes

R_4: IF (Outlook=Rainy) AND (Humidity=High) THEN Play=No

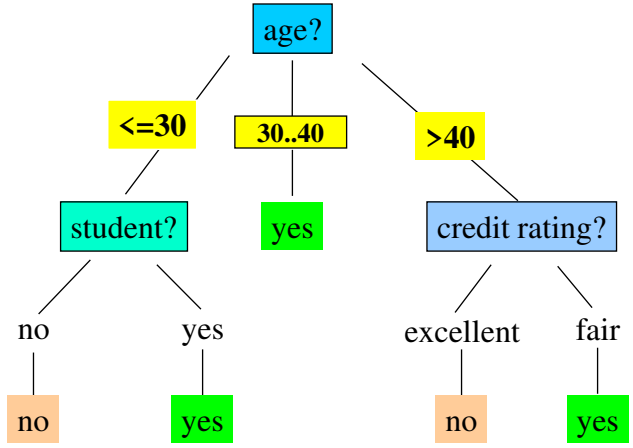
R_5: IF (Outlook=Rain) AND (Humidity=Normal) THEN Play=Yes



Try this

Age	Income	Student	Credit_rating	Buys_computer
<=30	high	no	fair	no
<=30	high	no	excellent	no
31…40	high	no	fair	yes
>40	medium	no	fair	yes
>40	low	yes	fair	yes
>40	low	yes	excellent	no
31…40	low	yes	excellent	yes
<=30	medium	no	fair	no
<=30	low	yes	fair	yes
>40	medium	yes	fair	yes
<=30	medium	yes	excellent	yes
31…40	medium	no	excellent	yes
31…40	high	yes	fair	yes
>40	medium	no	excellent	no

Output: A Decision Tree for *"buys_computer"*



- **HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING**
- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples
  - The hypothesis space searched by ID3 is the set of possible decision trees
    - 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

- **INDUCTIVE BIAS IN DECISION TREE LEARNING**
  - inductive bias is the set of assumptions that, together with the training data, deductively justify the classifications assigned by the learner to future instances
    - selects in favor of shorter trees over longer ones, and
    - selects trees that place the attributes with highest information gain closest to the root.
    - A closer approximation to the inductive bias of ID3: Shorter trees are preferred over longer trees. Trees that place high information gain attributes close to the root are preferred over those that do not.

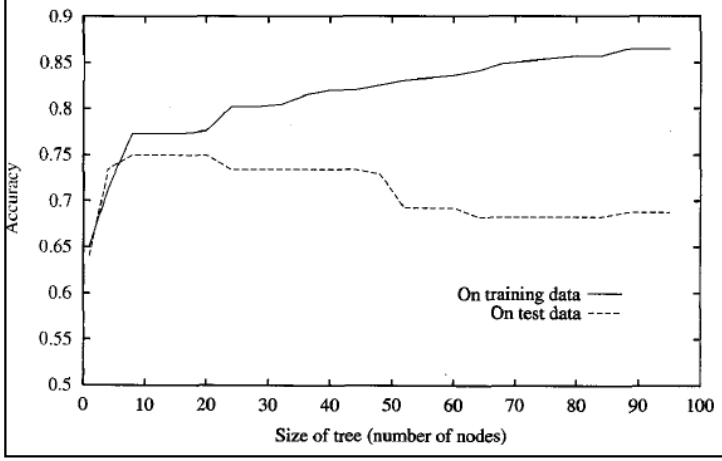
- ID3 capabilities and limitations by considering search space and search strategy
  - ID3 hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes
    - Adv – avoids search incomplete hypothesis space that might not contain target function
  - ID3 maintains only a single current hypothesis as it searches through the space of decision trees
    - In candidate-elimination algorithms set of all hypotheses consistent with the available training examples
      - it does not have the ability to determine how many alternative decision trees are consistent with the available training data
  - ID3 does not support backtracking leads to local optimum.
  - ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis

- Inductive bias ID3 versus CANDIDATE-ELIMINATION
  - ID3 → Preference bias
    - searches a complete hypothesis space
      - searches incompletely through this space, from simple to complex hypotheses, until it finds a hypothesis consistent with the data and it does not go beyond that.
      - Inductive bias: a consequence of the ordering of hypotheses by its search strategy and its hypothesis space introduces no additional bias
        - » inductive bias of ID3 follows from its search strategy thus a preference for certain hypotheses over others (e.g., for shorter hypotheses)
  - CANDIDATE-ELIMINATION → Restriction bias
    - searches an incomplete hypothesis space (i.e., one that can express only a subset of the potentially teachable concepts)
      - searches this space completely, finding every hypothesis consistent with the training data
      - Inductive bias: consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias.
        - » Solely relies on search space representation

- **Why Prefer Short Hypotheses?**
  - Occam's razor
    - Prefer the simplest hypothesis that fits the data
  - Claim
    - short hypothesis that coincidentally fits the training data
  - Critics
    - Not always true
    - Internal representation strategy of learners decides the size of the hypothesis.
      - Two learners arrive at different hypotheses, both justifying their contradictory conclusions by Occam's razor
    - Occam's razor will produce two different hypotheses from the same training examples

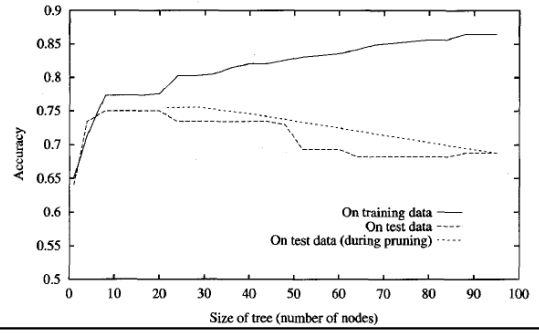
Criterion is to be used to determine the correct final tree size are:

- Assess the utility of post-pruning nodes using separate set of examples different from training
- Assess the improvement by applying expansion or pruning
- Apply explicit condition for tree growth



- **ISSUES IN DECISION TREE LEARNING**
  - Avoid Overfitting of data
    - ***Definition:*** Given a hypothesis space H, a hypothesis $h \in H$ is said to overfit the training data if there exists some alternative hypothesis $h' \in H$, such that h has smaller error than h' over the training examples, but h' has a smaller error than h over the entire distribution of instances.
    - Reasons for overfitting
      - when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function
    - Methods to avoid overfitting
      - Stop growing the tree earlier
      - Firstly allow to overfit and then post-prune the tree (more successful)

- **REDUCED ERROR PRUNING**
  - How exactly might we use a validation set to prevent overfitting?
    - Pruning a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node.
      - Condition: Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set



- **RULE POST-PRUNING**

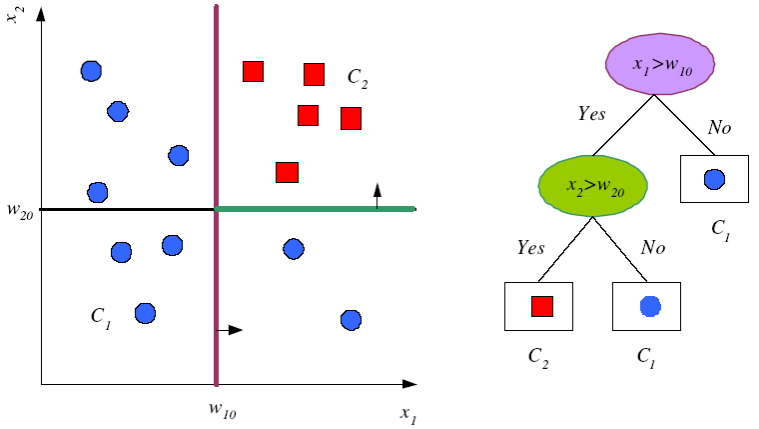
1. Infer the decision tree from the training set, growing the tree until the training data is fit as well as possible and allowing overfitting to occur.

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

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

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

Tree Uses Nodes, and Leaves



Divide and Conquer

- Internal decision nodes
  - Univariate: Uses a single attribute, x_i
    - Numeric x_i : Binary split : $x_i > w_m$
    - Discrete x_i : n-way split for n possible values
  - Multivariate: Uses all attributes, $\boldsymbol{x}$
- Leaves
  - Classification: Class labels, or proportions
  - Regression: Numeric; r average, or local fit
- Learning is greedy; find the best split recursively (Breiman et al, 1984; Quinlan, 1986, 1993)

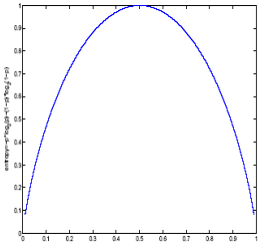
Classification Trees (ID3, CART, C4.5)

- For node m, N_m instances reach m, N^i_m belong to C_i

$$\hat{P}(C_i \mid \boldsymbol{x}, m) \equiv p^i_m = \frac{N^i_m}{N_m}$$

- Node m is pure if p^i_m is 0 or 1
- Measure of impurity is entropy

$$I_m = -\sum_{i=1}^{K} p^i_m \log_2 p^i_m$$



Best Split

- If node *m* is pure, generate a leaf and stop, otherwise split and continue recursively
- Impurity after split: N_{mj} of N_m take branch *j*. N^i_{mj} belong to C_i

$$\hat{P}(C_i \mid \boldsymbol{x}, m, j) \equiv p^i_{mj} = \frac{N^i_{mj}}{N_{mj}}$$

$$I'_m = -\sum_{j=1}^{n} \frac{N_{mj}}{N_m} \sum_{i=1}^{K} p^i_{mj} \log_2 p^i_{mj}$$

- Find the variable and split that min impurity (among all variables -- and split positions for numeric variables)


```
GenerateTree(X)
    If NodeEntropy(X)< θ_I /* eq. 9.3
        Create leaf labelled by majority class in X
        Return
    i ← SplitAttribute(X)
    For each branch of x_i
        Find X_i falling in branch
        GenerateTree(X_i)
SplitAttribute(X)
    MinEnt← MAX
    For all attributes i = 1,...,d
        If x_i is discrete with n values
            Split X into X_1,...,X_n by x_i
            e ← SplitEntropy(X_1,...,X_n) /* eq. 9.8 */
            If e<MinEnt MinEnt ← e; bestf ← i
        Else /* x_i is numeric */
            For all possible splits
                Split X into X_1, X_2 on x_i
                e←SplitEntropy(X_1, X_2)
                If e<MinEnt MinEnt ← e; bestf ← i
    Return bestf
```

– **Classification:** split the dataset based on homogeneity of data

– **Regression: (There is no class label)**

» fit a regression model to the target variable using each of the independent variables

» for each independent variable, the data is split at several split points

» At each split point, the "error" between the predicted value and the actual values is squared to get a "Sum of Squared Errors (SSE)".

» split point errors across the variables are compared and the variable /point yielding the lowest SSE is chosen as the root node/split point.

» This process is recursively continued

Regression Trees

- Error at node *m*:

$$b_m(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \mathcal{X}_m : \boldsymbol{x} \text{ reaches node } m \\ 0 & \text{otherwise} \end{cases}$$

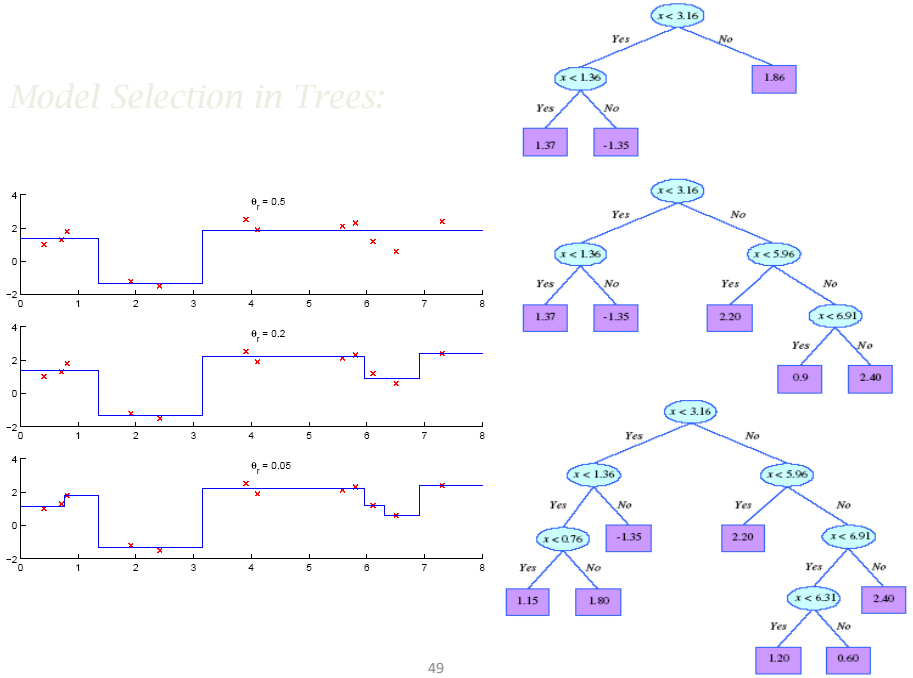
$$E_m = \frac{1}{N_m} \sum_t \left(r^t - g_m\right)^2 b_m\left(\boldsymbol{x}^t\right) \qquad g_m = \frac{\sum_t b_m\left(\boldsymbol{x}^t\right) r^t}{\sum_t b_m\left(\boldsymbol{x}^t\right)}$$

- After splitting:

$$b_{mj}(\boldsymbol{x}) = \begin{cases} 1 & \text{if } \boldsymbol{x} \in \mathcal{X}_{mj} : \boldsymbol{x} \text{ reaches node } m \text{ and branch } j \\ 0 & \text{otherwise} \end{cases}$$

$$E'_m = \frac{1}{N_m} \sum_j \sum_t \left(r^t - g_{mj}\right)^2 b_{mj}\left(\boldsymbol{x}^t\right) \qquad g_{mj} = \frac{\sum_t b_{mj}\left(\boldsymbol{x}^t\right) r^t}{\sum_t b_{mj}\left(\boldsymbol{x}^t\right)}$$

Model Selection in Trees:

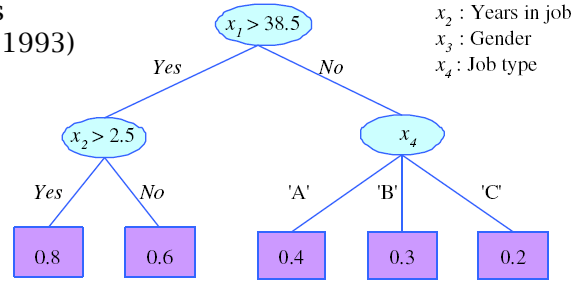


Pruning Trees

- Remove subtrees for better generalization (decrease variance)
  - Prepruning: Early stopping
  - Postpruning: Grow the whole tree then prune subtrees which overfit on the pruning set
- Prepruning is faster, postpruning is more accurate (requires a separate pruning set)

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



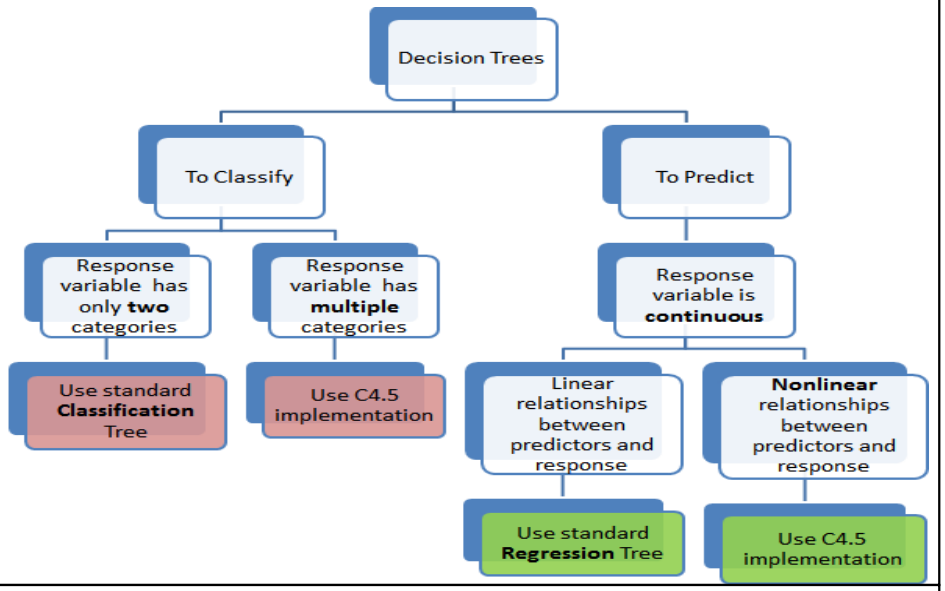
x_1 : Age
x_2 : Years in job
x_3 : Gender
x_4 : Job type

R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Learning Rules

- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered
- IREP (Fürnkrantz and Widmer, 1994), Ripper (Cohen, 1995)

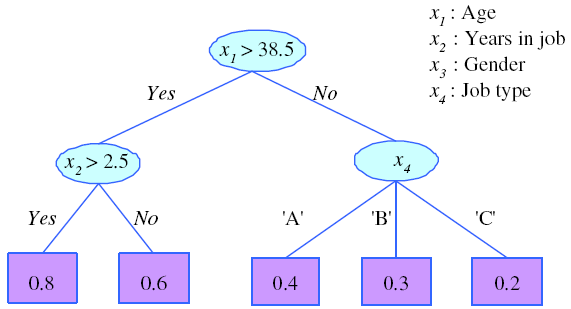
Classification and Regression Trees (CART)



Comparison between ID3, C4.5 and CART

	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible to outliers
CART	Towing Criteria	Handles both Categorical & Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical & Numeric value	Handle missing values.	Error Based pruning is used	Susceptible to outliers

Rule Extraction from Trees

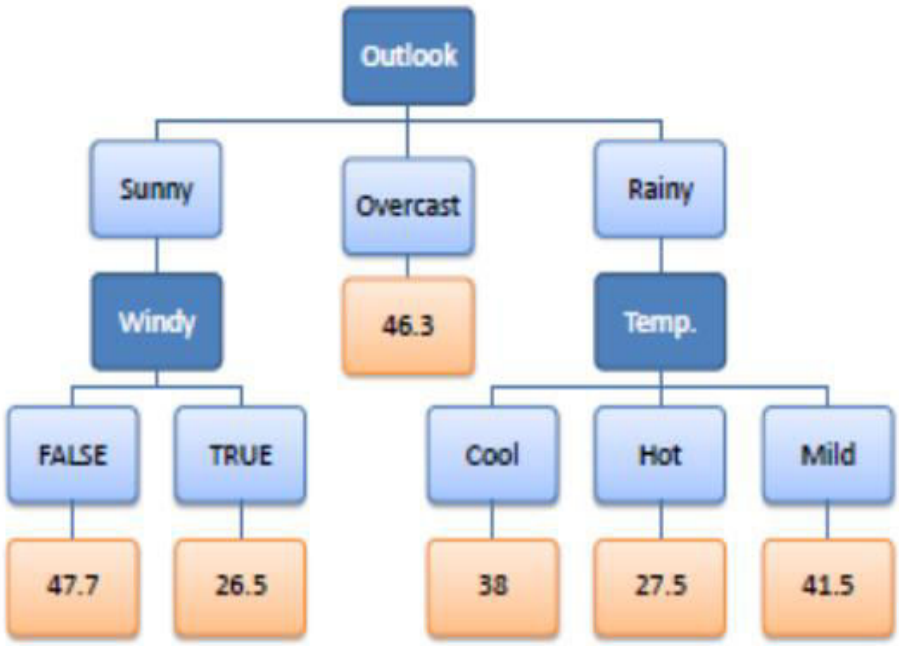


x_1 : Age
x_2 : Years in job
x_3 : Gender
x_4 : Job type

R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

Regression Tree

Predictors				Target
Outlook	Temp	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



- Regression Decision Tree
  - The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction
  - If the numerical sample is completely homogeneous its standard deviation is zero
  - Algorithm
    - *Step 1*: The standard deviation of the target is calculated.
    - *Step 2*: The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.
    - *Step 3*: The attribute with the largest standard deviation reduction is chosen as the decision node.
    - *Step 4a*: Dataset is divided based on the values of the selected attribute.
    - *Step 4b*: A branch set with standard deviation more than 0 needs further splitting.

Step.1: Standard deviation for one attribute

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$S = \sqrt{\frac{\sum (x-\mu)^2}{n}}$$

Standard Deviation

S = 9.32

Step.2: Standard deviation for two attributes

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)***S**(Sunny) + **P**(Overcast)***S**(Overcast) + **P**(Rainy)***S**(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

$$SDR(T,X) = S(T) - S(T,X)$$

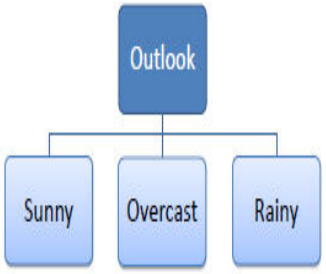
SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

Step3

★		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a



Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

Step 4b

SDR(Sunny, Humidity)=S(Sunny)-S(Sunny, Humidity)

= 10.87 – (2/5 (7.5)+ 3/5(12.5))

= 0.37

SDR(Sunny, Temperature)=S(Sunny)-S(Sunny, Temperature)

=10.87 – (3/5(7.3)+2/5(14.5))=2.69

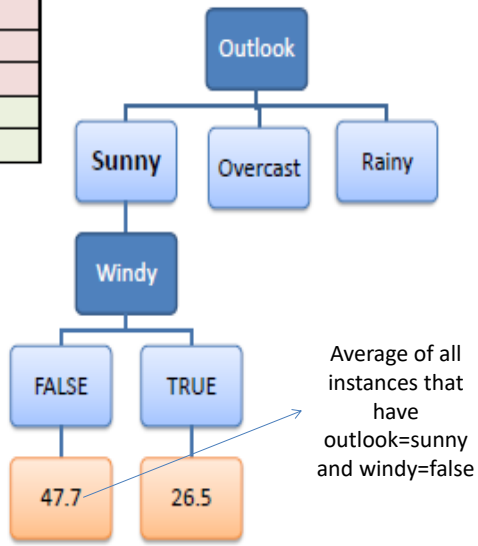
SDR(Sunny, Windy)=S(Sunny)-S(Sunny, Windy) = 10.87 – (3/5*3.09 +2/5 *3.5) = 7.62

Step 4b

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

★		Hours Played (StDev)
Windy	False	3.09
	True	3.50
SDR= 7.62		

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5)



Average of all instances that have outlook=sunny and windy=false

Step 4b

SDR(Rainy, Temperature)=S(Rainy)- S(Rainy, Temperature)
=7.78 – (2/5(2.5)+2/5(6.5)+1/5(0))=4.18

SDR(Rainy, Humidity)= S(Rainy)- S(Rainy, Humidity)
= 7.78 – (3/5(4.08)+ 2/5 (5))
=3.332

Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

SDR(Rainy, Windy)= S(Rainy)- S(Rainy, Windy) = 7.78 – (2/5(9) +3/5 (5.56)) = 7.78-(3.6+3.336) = 0.844

Step 4b

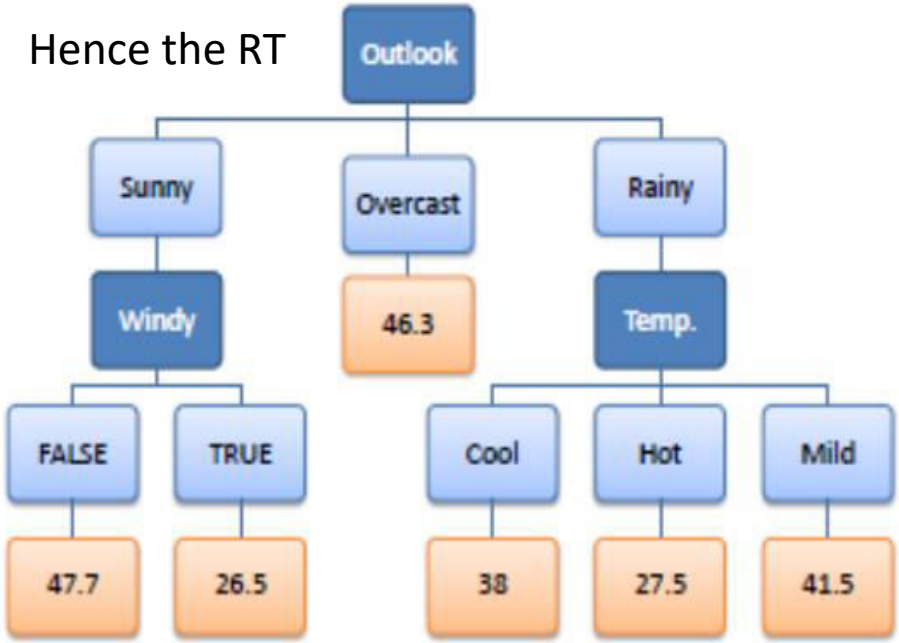
SDR(Overcast, Temperature)=S(Overcast)- S(Overcast, Temperature)
=3.49 – (2/4(1)+1/4(0)+1/4(0))=2.99

Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

SDR(Overcast, Humidity)= S(Overcast)- S(Overcast, Humidity)
= 3.49 – (2/4(3)+ 2/4 (0.5))
= 1.74

SDR(Overcast, Windy)= S(Overcast)- S(Overcast, Windy) = 3.49 – (2/4(4.5) +2/4 (1)) = 3.49-(2.25+0.5) = 0.74

Hence the RT



Decision Tree
Example

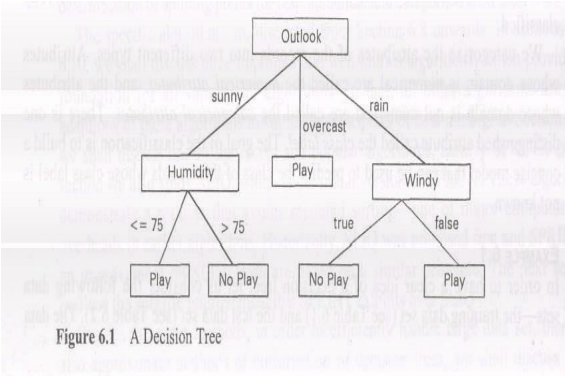


Figure 6.1 A Decision Tree

- We have five leaf nodes.
- In a decision tree, each leaf node represents a rule.
- We have the following rules corresponding to the tree given in Figure.
  - *RULE 1: If it is sunny and the humidity is not above 75%, then play.*
  - *RULE 2: If it is sunny and the humidity is above 75%, then do not play.*
  - *RULE 3: If it is overcast, then play.*
  - *RULE 4: If it is rainy and not windy, then play.*
  - *RULE 5: If it is rainy and windy, then don't play.*

- The *accuracy of the classifier* is determined by the percentage of the test data set that is correctly classified.
- We can see that for *Rule 1* there are two records of the test data set satisfying *outlook= sunny* and *humidity < 75,* and only one of these is correctly classified *as play.*
- Thus, the accuracy of this rule is 0.5 (or 50%). Similarly, the accuracy of Rule 2 is also 0.5 (or 50%). The accuracy of Rule 3 is 0.66.

RULE 1
If it is sunny and the humidity is not above 75%, then play.

Table 6.2 Test Data Set

OUTLOOK	TEMP(F)	HUMIDITY(%)	WINDY	CLASS
sunny	79	90	true	play
sunny	56	70	false	play
sunny	79	75	true	no play
sunny	60	90	true	no play
overcast	88	88	false	no play
overcast	63	75	true	play
overcast	88	95	false	play
rain	78	60	false	play
rain	66	70	false	no play
rain	68	60	true	play

Decision tree???


A

Not interested in learning

Class test ~50%
Test ~47%

Under-fit/ biased learning


B

Memorizing the lessons

Class test ~98%
Test ~69%

Over-fit/ Memorizing


C

Conceptual Learning

Class test ~92%
Test ~89%

Best-fit

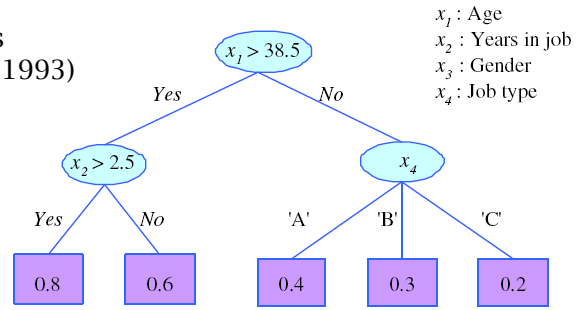
Avoid Overfitting in Decision Tree

- **Overfitting**: An induced tree may overfit the training data
  - Too many branches, some may reflect anomalies due to noise or outliers
  - Poor accuracy for unseen samples
- Two approaches to avoid overfitting
  - **Prepruning:** Halt tree construction early—do not split a node if this would result in the goodness measure falling below a threshold
    - Difficult to choose an appropriate threshold
  - **Postpruning:** Remove branches from a "fully grown" tree—get a sequence of progressively pruned trees
    - Use a set of data different from the training data to decide which is the "best pruned tree"

Decision Tree learning: CART

Rule Extraction from Trees

C4.5Rules
(Quinlan, 1993)



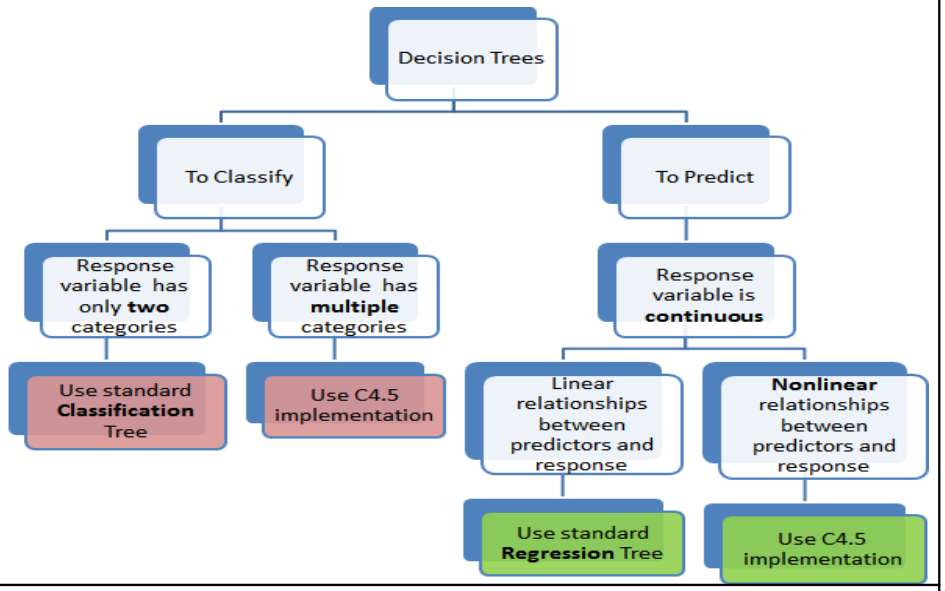
R1: IF (age>38.5) AND (years-in-job>2.5) THEN y =0.8
R2: IF (age>38.5) AND (years-in-job≤2.5) THEN y =0.6
R3: IF (age≤38.5) AND (job-type='A') THEN y =0.4
R4: IF (age≤38.5) AND (job-type='B') THEN y =0.3
R5: IF (age≤38.5) AND (job-type='C') THEN y =0.2

– **Classification:** split the dataset based on homogeneity of data
– **Regression: (There is no class label)**
» fit a regression model to the target variable using each of the independent variables
» for each independent variable, the data is split at several split points
» At each split point, the "error" between the predicted value and the actual values is squared to get a "Sum of Squared Errors (SSE)".
» split point errors across the variables are compared and the variable /point yielding the lowest SSE is chosen as the root node/split point.
» This process is recursively continued

Learning Rules

- Rule induction is similar to tree induction but
  - tree induction is breadth-first,
  - rule induction is depth-first; one rule at a time
- Rule set contains rules; rules are conjunctions of terms
- Rule covers an example if all terms of the rule evaluate to true for the example
- Sequential covering: Generate rules one at a time until all positive examples are covered
- IREP (Fürnkrantz and Widmer, 1994), Ripper (Cohen, 1995)

Classification and Regression Trees (CART)

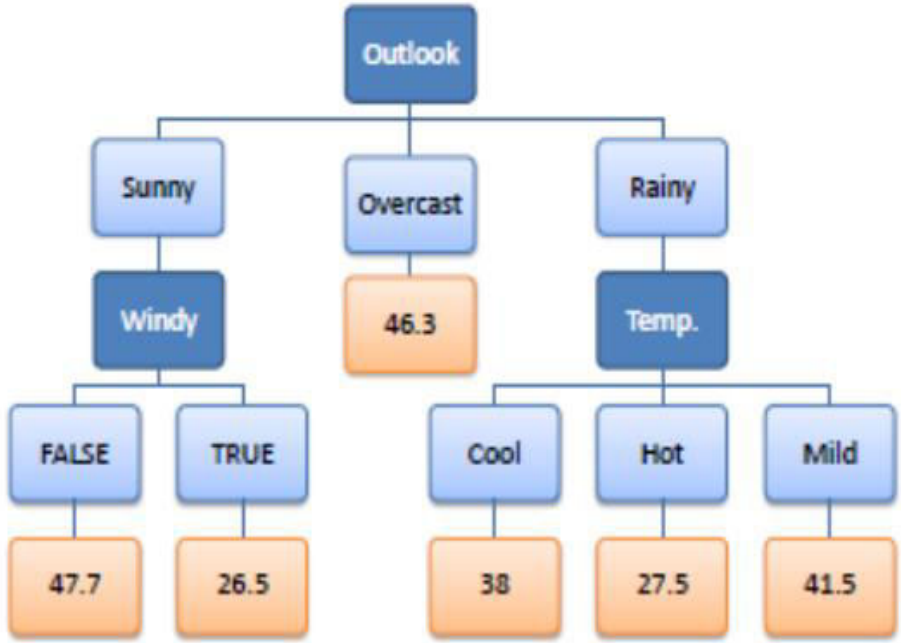


Comparison between ID3, C4.5 and CART

	Splitting Criteria	Attribute type	Missing values	Pruning Strategy	Outlier Detection
ID3	Information Gain	Handles only Categorical value	Do not handle missing values.	No pruning is done	Susceptible to outliers
CART	Towing Criteria	Handles both Categorical & Numeric value	Handle missing values.	Cost-Complexity pruning is used	Can handle Outliers
C4.5	Gain Ratio	Handles both Categorical & Numeric value	Handle missing values.	Error Based pruning is used	Susceptible to outliers

Regression Tree

Predictors				Target
Outlook	Temp.	Humidity	Windy	Hours Played
Rainy	Hot	High	False	25
Rainy	Hot	High	True	30
Overcast	Hot	High	False	48
Sunny	Mild	High	False	46
Sunny	Cool	Normal	False	62
Sunny	Cool	Normal	True	23
Overcast	Cool	Normal	True	43
Rainy	Mild	High	False	36
Rainy	Cool	Normal	False	38
Sunny	Mild	Normal	False	48
Rainy	Mild	Normal	True	48
Overcast	Mild	High	True	52
Overcast	Hot	Normal	False	44
Sunny	Mild	High	True	30



- Regression Decision Tree
  - The ID3 algorithm can be used to construct a decision tree for regression by replacing Information Gain with Standard Deviation Reduction
  - If the numerical sample is completely homogeneous its standard deviation is zero
  - Algorithm
    - *Step 1*: The standard deviation of the target is calculated.
    - *Step 2*: The dataset is then split on the different attributes. The standard deviation for each branch is calculated. The resulting standard deviation is subtracted from the standard deviation before the split. The result is the standard deviation reduction.
    - *Step 3*: The attribute with the largest standard deviation reduction is chosen as the decision node.
    - *Step 4a*: Dataset is divided based on the values of the selected attribute.
    - *Step 4b*: A branch set with standard deviation more than 0 needs further splitting.

Step.1: Standard deviation for one attribute

Hours Played
25
30
46
45
52
23
43
35
38
46
48
52
44
30

$$S = \sqrt{\frac{\sum (x-\mu)^2}{n}}$$

σ = population standard deviation

N = the size of the population

x_i = each value from the population

μ = the population mean

Standard Deviation

S = 9.32

Step.2: Standard deviation for two attributes

$$S(T,X) = \sum_{c \in X} P(c)S(c)$$

		Hours Played (StDev)	Count
Outlook	Overcast	3.49	4
	Rainy	7.78	5
	Sunny	10.87	5
			14

S(Hours, Outlook) = **P**(Sunny)*__**S**__(Sunny) + **P**(Overcast)*__**S**__(Overcast) + **P**(Rainy)*__**S**__(Rainy)

= (4/14)*3.49 + (5/14)*7.78 + (5/14)*10.87

= 7.66

		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

		Hours Played (StDev)
Temp.	Cool	10.51
	Hot	8.95
	Mild	7.65
SDR=0.17		

		Hours Played (StDev)
Humidity	High	9.36
	Normal	8.37
SDR=0.28		

		Hours Played (StDev)
Windy	False	7.87
	True	10.59
SDR=0.29		

$$SDR(T,X) = S(T) - S(T,X)$$

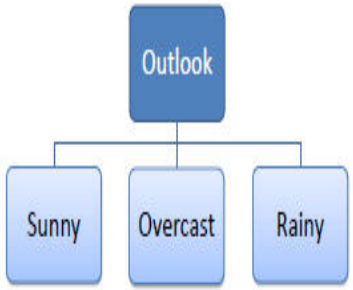
SDR(Hours , Outlook) = **S**(Hours) – **S**(Hours, Outlook)

= 9.32 – 7.66 = 1.66

Step3

★		Hours Played (StDev)
Outlook	Overcast	3.49
	Rainy	7.78
	Sunny	10.87
SDR=1.66		

Step 4a



Outlook	Temp	Humidity	Windy	Hours Played
Sunny	Mild	High	FALSE	45
Sunny	Cool	Normal	FALSE	52
Sunny	Cool	Normal	TRUE	23
Sunny	Mild	Normal	FALSE	46
Sunny	Mild	High	TRUE	30
Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48
Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

Step 4b

SDR(Sunny, Humidity)=S(Sunny)-S(Sunny, Humidity)

= 10.87 – (2/5 (7.5)+ 3/5(12.5))

= 0.37

SDR(Sunny, Temperature)=S(Sunny)-S(Sunny, Temperature)

=10.87 – (3/5(7.3)+2/5(14.5))=2.69

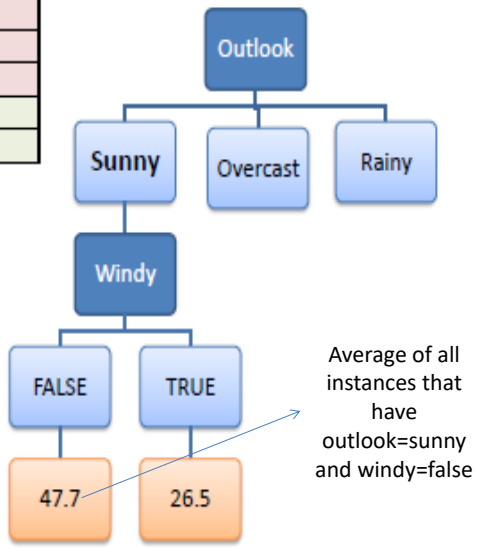
SDR(Sunny, Windy)=S(Sunny)-S(Sunny, Windy) = 10.87 – (3/5*3.09 +2/5 *3.5) = 7.62

Step 4b

Temp	Humidity	Windy	Hours Played
Mild	High	FALSE	45
Cool	Normal	FALSE	52
Mild	Normal	FALSE	46
Cool	Normal	TRUE	23
Mild	High	TRUE	30

★		Hours Played (StDev)
Windy	False	3.09
	True	3.50
SDR= 7.62		

SDR = 10.87-((3/5)*3.09 + (2/5)*3.5)



Step 4b

SDR(Rainy, Temperature)=S(Rainy)- S(Rainy, Temperature)
=7.78 – (2/5(2.5)+2/5(6.5)+1/5(0))=4.18

SDR(Rainy, Humidity)= S(Rainy)- S(Rainy, Humidity)
= 7.78 – (3/5(4.08)+ 2/5 (5))
=3.332

Rainy	Hot	High	FALSE	25
Rainy	Hot	High	TRUE	30
Rainy	Mild	High	FALSE	35
Rainy	Cool	Normal	FALSE	38
Rainy	Mild	Normal	TRUE	48

SDR(Rainy, Windy)= S(Rainy)- S(Rainy, Windy) = 7.78 – (2/5(9) +3/5 (5.56)) = 7.78-(3.6+3.336) = 0.844

Step 4b

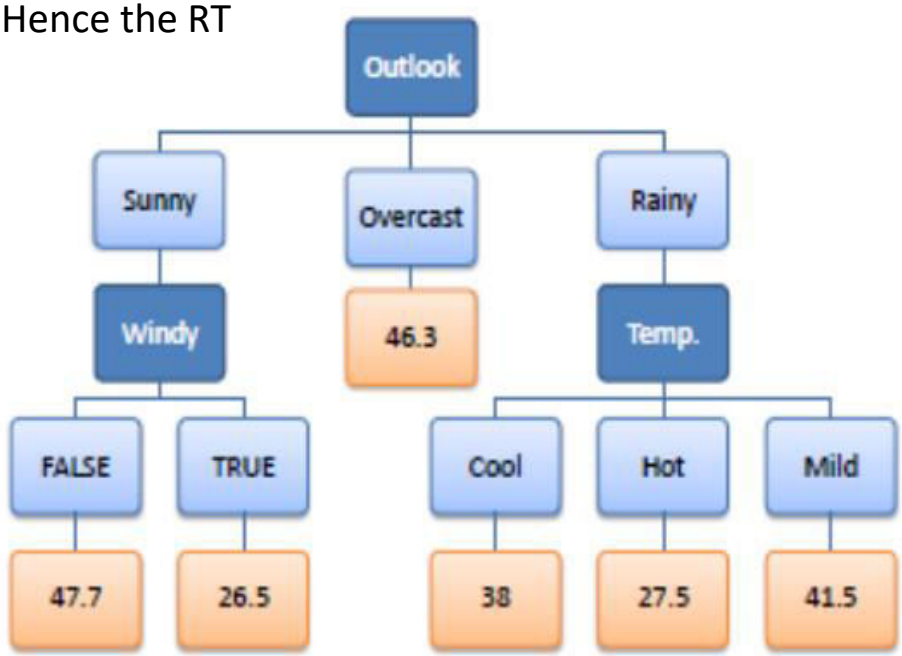
SDR(Overcast, Temperature)=S(Overcast)- S(Overcast, Temperature)
=3.49 – (2/4(1)+1/4(0)+1/4(0))=2.99

Overcast	Hot	High	FALSE	46
Overcast	Cool	Normal	TRUE	43
Overcast	Mild	High	TRUE	52
Overcast	Hot	Normal	FALSE	44

SDR(Overcast, Humidity)= S(Overcast)- S(Overcast, Humidity)
= 3.49 – (2/4(3)+ 2/4 (0.5))
= 1.74

SDR(Overcast, Windy)= S(Overcast)- S(Overcast, Windy) = 3.49 – (2/4(4.5) +2/4 (1)) = 3.49-(2.25+0.5) = 0.74

Hence the RT



Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
3	Overcast	Hot	High	Weak	46
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
10	Rain	Mild	Normal	Weak	46
11	Sunny	Mild	Normal	Strong	48
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44
14	Rain	Mild	High	Strong	30

Standard deviation

Golf players = {25, 30, 46, 45, 52, 23, 43, 35, 38, 46, 48, 52, 44, 30}

Average of golf players = (25 + 30 + 46 + 45 + 52 + 23 + 43 + 35 + 38 + 46 + 48 + 52 + 44 + 30)/14 = 39.78

Standard deviation of golf players = $\sqrt{[(}$ $(25 - 39.78)^2 + (30 - 39.78)^2 + (46 - 39.78)^2 + \ldots + (30 - 39.78)^2$)/14] = 9.32

Outlook

Outlook can be sunny, overcast and rain. We need to calculate standard deviation of golf players for all of these outlook candidates.

Sunny outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Golf players for sunny outlook = {25, 30, 35, 38, 48}

Average of golf players for sunny outlook = (25+30+35+38+48)/5 = 35.2

Standard deviation of golf players for sunny outlook = $\sqrt{(((25 - 35.2)^2 + (30 - 35.2)^2 + \ldots)/5)}$ = 7.78

Overcast outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
3	Overcast	Hot	High	Weak	46
7	Overcast	Cool	Normal	Strong	43
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44

Golf players for overcast outlook = {46, 43, 52, 44}

Average of golf players for overcast outlook = (46 + 43 + 52 + 44)/4 = 46.25

Standard deviation of golf players for overcast outlook = $\sqrt{(((46-46.25)^2+(43-46.25)^2+\ldots)}$= 3.49

Rainy outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
10	Rain	Mild	Normal	Weak	46
14	Rain	Mild	High	Strong	30

Golf players for overcast outlook = {45, 52, 23, 46, 30}

Average of golf players for overcast outlook = (45+52+23+46+30)/5 = 39.2

Standard deviation of golf players for rainy outlook = $\sqrt{(((45 - 39.2)^2+(52 - 39.2)^2+\ldots)/5)}$=10.87

Summarizing standard deviations for the outlook feature

Outlook	Stdev of Golf Players	Instances
Overcast	3.49	4
Rain	10.87	5
Sunny	7.78	5

Weighted standard deviation for outlook = (4/14)x3.49 + (5/14)x10.87 + (5/14)x7.78 = 7.66

You might remember that we have calculated the global standard deviation of golf players 9.32 in previous steps. Standard deviation reduction is difference of the global standard deviation and standard deviation for current feature. In this way, maximized standard deviation reduction will be the decision node.

Standard deviation reduction for outlook = 9.32 – 7.66 = 1.66

Temperature

Temperature can be hot, cool or mild. We will calculate standard deviations for those candidates.

Hot temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
3	Overcast	Hot	High	Weak	46
13	Overcast	Hot	Normal	Weak	44

Golf players for hot temperature = {25, 30, 46, 44}

Standard deviation of golf players for hot temperature = 8.95

Cool temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
9	Sunny	Cool	Normal	Weak	38

Golf players for cool temperature = {52, 23, 43, 38}

Standard deviation of golf players for cool temperature = 10.51

Mild temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
8	Sunny	Mild	High	Weak	35
10	Rain	Mild	Normal	Weak	46
11	Sunny	Mild	Normal	Strong	48
12	Overcast	Mild	High	Strong	52
14	Rain	Mild	High	Strong	30

Golf players for mild temperature = {45, 35, 46, 48, 52, 30}

Standard deviation of golf players for mild temperature = 7.65

Summarizing standard deviations for temperature feature

Temperature	Stdev of Golf Players	Instances
Hot	8.95	4
Cool	10.51	4
Mild	7.65	6

Weighted standard deviation for temperature = (4/14)x8.95 + (4/14)x10.51 + (6/14)x7.65 = 8.84

Standard deviation reduction for temperature = 9.32 – 8.84 = 0.47

Humidity

Humidity is a binary class. It can either be normal or high.

High humidity

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
3	Overcast	Hot	High	Weak	46
4	Rain	Mild	High	Weak	45
8	Sunny	Mild	High	Weak	35
12	Overcast	Mild	High	Strong	52
14	Rain	Mild	High	Strong	30

Golf players for high humidity = {25, 30, 46, 45, 35, 52, 30}

Standard deviation for golf players for high humidity = 9.36

Normal humidity

Day	Outlook	Temp.	Humidity	Wind	Golf Players
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
9	Sunny	Cool	Normal	Weak	38
10	Rain	Mild	Normal	Weak	46
11	Sunny	Mild	Normal	Strong	48
13	Overcast	Hot	Normal	Weak	44

Golf players for normal humidity = {52, 23, 43, 38, 46, 48, 44}

Standard deviation for golf players for normal humidity = 8.73

Summarizing standard deviations for humidity feature

Humidity	Stdev of Golf Player	Instances
High	9.36	7
Normal	8.73	7

Weighted standard deviation for humidity = (7/14)x9.36 + (7/14)x8.73 = 9.04

Standard deviation reduction for humidity = 9.32 – 9.04 = 0.27

Wind

Wind is a binary class, too. It can either be Strong or Weak.

Strong Wind

Day	Outlook	Temp.	Humidity	Wind	Golf Players
2	Sunny	Hot	High	Strong	30
6	Rain	Cool	Normal	Strong	23
7	Overcast	Cool	Normal	Strong	43
11	Sunny	Mild	Normal	Strong	48
12	Overcast	Mild	High	Strong	52
14	Rain	Mild	High	Strong	30

Golf players for strong wind= {30, 23, 43, 48, 52, 30}

Standard deviation for golf players for strong wind = 10.59

Weak Wind

1	Sunny	Hot	High	Weak	25
3	Overcast	Hot	High	Weak	46
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
10	Rain	Mild	Normal	Weak	46
13	Overcast	Hot	Normal	Weak	44

Golf players for weakk wind= {25, 46, 45, 52, 35, 38, 46, 44}

Standard deviation for golf players for weak wind = 7.87

Summarizing standard deviations for wind feature

Wind	Stdev of Golf Player	Instances
Strong	10.59	6
Weak	7.87	8

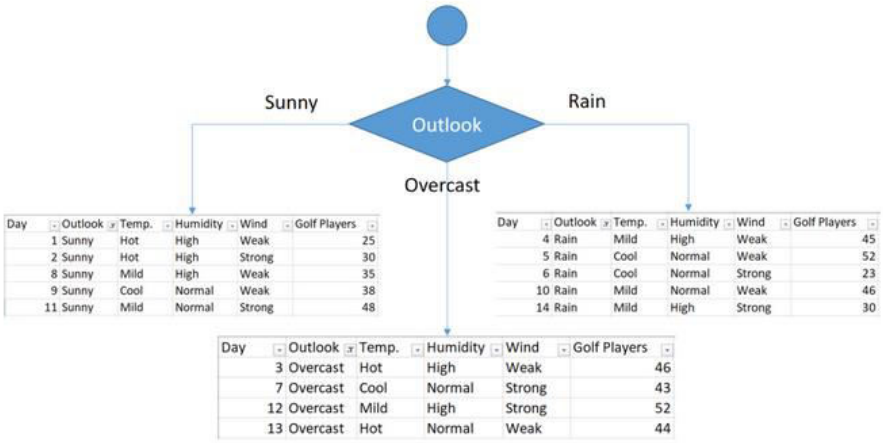
Weighted standard deviation for wind = (6/14)x10.59 + (8/14)x7.87 = 9.03

Standard deviation reduction for wind = 9.32 – 9.03 = 0.29

So, we've calculated standard deviation reduction values for all features. The winner is outlook because it has the highest score.

Feature	**Standard Deviation Reduction**
Outlook	1.66
Temperature	0.47
Humidity	0.27
Wind	0.29

We'll put outlook decision at the top of decision tree. Let's monitor the new sub data sets for the candidate branches of outlook feature.



Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
10	Rain	Mild	Normal	Weak	46
14	Rain	Mild	High	Strong	30

Day	Outlook	Temp.	Humidity	Wind	Golf Players
3	Overcast	Hot	High	Weak	46
7	Overcast	Cool	Normal	Strong	43
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44

Sunny Outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Golf players for sunny outlook = {25, 30, 35, 38, 48}

Standard deviation for sunny outlook = 7.78

Notice that we will use this standard deviation value as global standard deviation for this sub data set.

Sunny outlook and Hot Temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30

Standard deviation for sunny outlook and hot temperature = 2.5

Sunny outlook and Cool Temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
9	Sunny	Cool	Normal	Weak	38

Standard deviation for sunny outlook and cool temperature = 0

Sunny outlook and Mild Temperature

Day	Outlook	Temp.	Humidity	Wind	Golf Players
8	Sunny	Mild	High	Weak	35
11	Sunny	Mild	Normal	Strong	48

Summary of standard deviations for temperature feature when outlook is sunny

Temperature	Stdev for Golf Players	Instances
Hot	2.5	2
Cool	0	1
Mild	6.5	2

Weighted standard deviation for sunny outlook and temperature = (2/5)x2.5 + (1/5)x0 + (2/5)x6.5 = 3.6

Standard deviation reduction for sunny outlook and temperature = 7.78 – 3.6 = 4.18

Sunny outlook and high humidity

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
2	Sunny	Hot	High	Strong	30
8	Sunny	Mild	High	Weak	35

Standard deviation for sunny outlook and high humidity = 4.08

Sunny outlook and normal humidity

Day	Outlook	Temp.	Humidity	Wind	Golf Players
9	Sunny	Cool	Normal	Weak	38
11	Sunny	Mild	Normal	Strong	48

Standard deviation for sunny outlook and normal humidity = 5

Summarizing standard deviations for humidity feature when outlook is sunny

Humidity	Stdev for Golf Players	Instances
High	4.08	3
Normal	5.00	2

Weighted standard deviations for sunny outlook and humidity = (3/5)x4.08 + (2/5)x5 = 4.45

Standard deviation reduction for sunny outlook and humidity = 7.78 – 4.45 = 3.33

Sunny outlook and Strong Wind

Day	Outlook	Temp.	Humidity	Wind	Golf Players
2	Sunny	Hot	High	Strong	30
11	Sunny	Mild	Normal	Strong	48

Standard deviation for sunny outlook and strong wind = 9

Sunny outlook and Weak Wind

Day	Outlook	Temp.	Humidity	Wind	Golf Players
1	Sunny	Hot	High	Weak	25
8	Sunny	Mild	High	Weak	35
9	Sunny	Cool	Normal	Weak	38

Standard deviation for sunny outlook and weak wind = 5.56

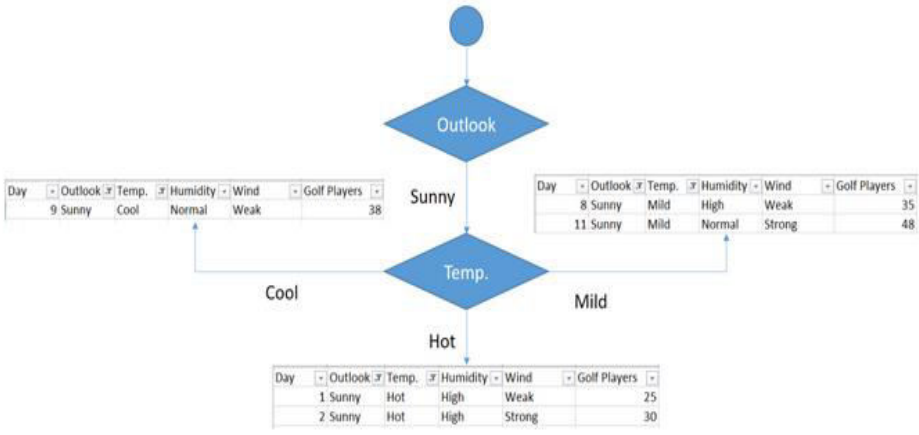
Wind	Stdev for Golf Players	Instances
Strong	9	2
Weak	5.56	3

Weighted standard deviations for sunny outlook and wind = (2/5)x9 + (3/5)x5.56 = 6.93

Standard deviation reduction for sunny outlook and wind = 7.78 – 6.93 = 0.85

We've calculated standard deviation reductions for sunny outlook. The winner is temperature.

Feature	**Standard Deviation Reduction**
Temperature	4.18
Humidity	3.33
Wind	0.85



Putting temperature decision at the bottom of sunny outlook

Pruning

Cool branch has one instance in its sub data set. We can say that if outlook is sunny and temperature is cool, then there would be 38 golf players. But what about hot branch? There are still 2 instances. Should we add another branch for weak wind and strong wind? No, we should not. Because this causes over-fitting. We should terminate building branches, for example if there are less than five instances in the sub data set. Or standard deviation of the sub data set can be less than 5% of the entire data set. I prefer to apply the first one. I will terminate the branch if there are less than 5 instances in the current sub data set. If this termination condition is satisfied, then I will calculate the average of the sub data set. This operation is called as pruning in decision tree trees.

Overcast outlook

Overcast outlook branch has already 4 instances in the sub data set. We can terminate building branches for this leaf. Final decision will be average of the following table for overcast outlook.

Day	Outlook	Temp.	Humidity	Wind	Golf Players
3	Overcast	Hot	High	Weak	46
7	Overcast	Cool	Normal	Strong	43
12	Overcast	Mild	High	Strong	52
13	Overcast	Hot	Normal	Weak	44

If outlook is overcast, then there would be (46+43+52+44)/4 = 46.25 golf players.

Rainy Outlook

Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
6	Rain	Cool	Normal	Strong	23
10	Rain	Mild	Normal	Weak	46
14	Rain	Mild	High	Strong	30

We need to find standard deviation reduction values for the rest of the features in same way for the sub data set above.

Standard deviation for rainy outlook = 10.87

Notice that we will use this value as global standard deviation for this branch in reduction step.

Rainy outlook and temperature

Temperature	Standard deviation for golf players	instances
Cool	14.50	2
Mild	7.32	3

Weighted standard deviation for rainy outlook and temperature = (2/5)x14.50 + (3/5)x7.32 = 10.19

Standard deviation reduction for rainy outlook and temperature = 10.87 – 10.19 = 0.67

Rainy outlook and humidity

Humidity	Standard deviation for golf players	instances
High	7.50	2
Normal	12.50	3

Weighted standard deviation for rainy outlook and humidity = (2/5)x7.50 + (3/5)x12.50 = 10.50

Standard deviation reduction for rainy outlook and humidity = 10.87 – 10.50 = 0.37

Rainy outlook and wind

Wind	Standard deviation for golf players	instances
Weak	3.09	3
Strong	3.5	2

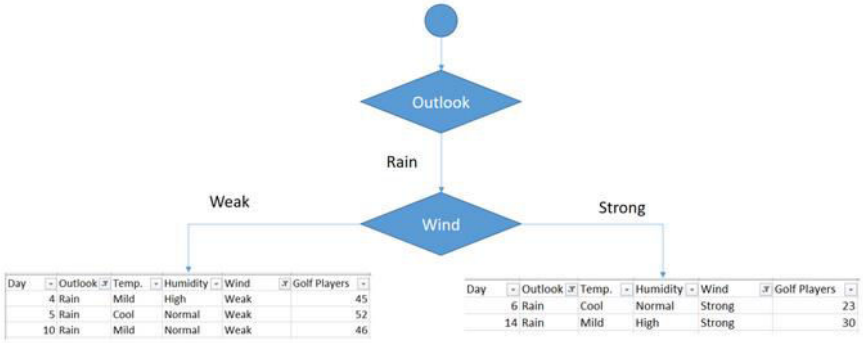
Weighted standard deviation for rainy outlook and wind = (3/5)x3.09 + (2/5)x3.5 = 3.25

Standard deviation reduction for rainy outlook and wind = 10.87 – 3.25 = 7.62

Summarizing rainy outlook

As illustrated below, the winner is wind feature.

Feature	**Standard deviation reduction**
Temperature	0.67
Humidity	0.37
Wind	7.62



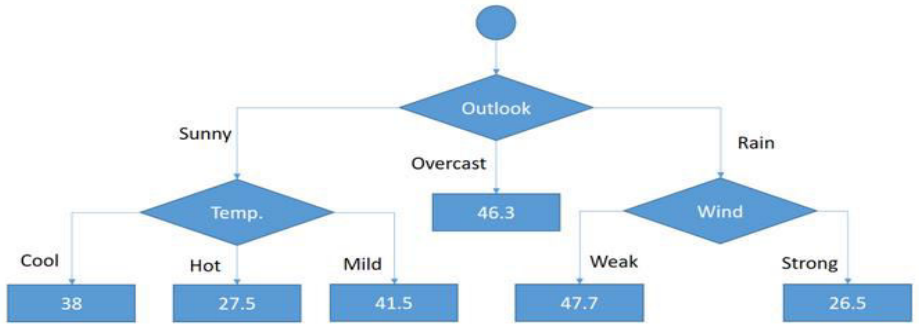
Day	Outlook	Temp.	Humidity	Wind	Golf Players
4	Rain	Mild	High	Weak	45
5	Rain	Cool	Normal	Weak	52
10	Rain	Mild	Normal	Weak	46

Day	Outlook	Temp.	Humidity	Wind	Golf Players
6	Rain	Cool	Normal	Strong	23
14	Rain	Mild	High	Strong	30

Sub data set for rainy outlook

As seen, both branches have items less than 5. Now, we can terminate these leafs based on the termination rule.

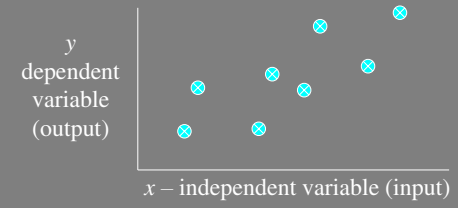
So, Final form of the decision tree is demonstrated below.



Final form of the regression tree

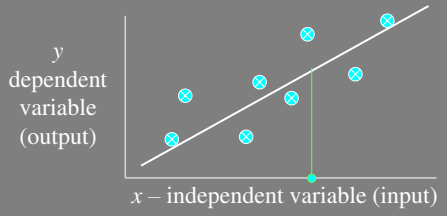
Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points



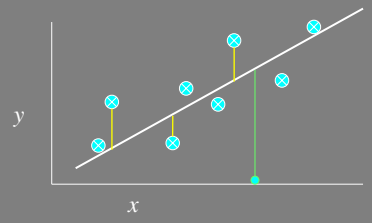
Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points



Regression

- For classification the output(s) is nominal
- In regression the output is continuous
  - Function Approximation
- Many models could be used – Simplest is linear regression
  - Fit data with the best hyper-plane which "goes through" the points
  - For each point the difference between the predicted point and the actual observation is the *residue*



# CORRECT (x)	ATTITUDE (y)
17	94
13	73
12	59
15	80
16	93
14	85
16	66
16	79
18	77
19	91

Simple Linear Regression

- For now, assume just one (input) independent variable x, and one (output) dependent variable y
  - Multiple linear regression assumes an input vector **x**
  - Multivariate linear regression assumes an output vector **y**
- We "fit" the points with a line (i.e. hyper-plane)
- Which line should we use?
  - Choose an objective function
  - For simple linear regression we choose sum squared error (SSE)
    - $\Sigma\ (predicted_i - actual_i)^2 = \Sigma\ (residue_i)^2$
  - Thus, find the line which minimizes the sum of the squared residues (e.g. least squares)
  - This mimics the case where data points were sampled from the actual hyperplane with Gaussian noise added

Linear Regression function

$$y = a + bx$$

Slope (b) of Regression Line

$$b = r \frac{S_y}{S_x}$$

y-Intercept (a) of Regression Line

$$a = \bar{y} - b\bar{x}$$

Pearson Correlation Coefficient (r)

$$r = \frac{\sum ((x-\bar{x})(y-\bar{y}))}{\sqrt{\sum (x-\bar{x})^2 \sum (y-\bar{y})^2}}$$

# CORRECT (x)	ATTITUDE (y)
17	94
13	73
12	59
15	80
16	93
14	85
16	66
16	79
18	77
19	91

# CORRECT (x)	ATTITUDE (y)	$(x-\bar{x})$	$(y-\bar{y})$	$(x-\bar{x})(y-\bar{y})$	$(x-\bar{x})^2$	$(y-\bar{y})^2$
17	94	1.4	14.3	20.02	1.96	204.49
13	73	-2.6	-6.7	17.42	6.76	44.89
12	59	-3.6	-20.7	74.52	12.96	428.49
15	80	-0.6	0.3	-0.18	0.36	0.09
16	93	0.4	13.3	5.32	0.16	176.89
14	85	-1.6	5.3	-8.48	2.56	28.09
16	66	0.4	-13.7	-5.48	0.16	187.69
16	79	0.4	-0.7	-0.28	0.16	0.49
18	77	2.4	-2.7	-6.48	5.76	7.29
19	91	3.4	11.3	38.42	11.56	127.69
$\bar{x} = 15.6$	$\bar{y} = 79.7$			$\Sigma = 134.8$	$\Sigma = 42.4$	$\Sigma = 1206.1$

$$r = \frac{134.8}{\sqrt{42.4 \times 1206.1}}$$

$$= 0.596$$

$$S_y = \sqrt{\frac{\Sigma (y-\bar{y})^2}{n-1}}$$

$$= \sqrt{\frac{1206.1}{9}} = 11.576$$

$$S_x = \sqrt{\frac{\Sigma (x-\bar{x})^2}{n-1}}$$

$$= \sqrt{\frac{42.4}{9}} = 2.171$$

$$b = r \times \frac{S_y}{S_x}$$

$$= 0.596 \times \frac{11.576}{2.171}$$

$$= 0.596 \times 5.332$$

$$= 3.178$$

$$a = \bar{y} - b\bar{x}$$

$$= 79.7 - 3.178 \times 15.6$$

$$= 79.7 - 49.577$$

$$= 30.123$$

$$y = a + bx$$

$$= 30.123 + 3.178x$$

SUBJECT	AGE X	GLUCOSE LEVEL Y	XY	X^2	Y^2
1	43	99	4257	1849	9801
2	21	65	1365	441	4225
3	25	79	1975	625	6241
4	42	75	3150	1764	5625
5	57	87	4959	3249	7569
6	59	81	4779	3481	6561
Σ	247	486	20485	11409	40022

SUBJECT	AGE X	GLUCOSE LEVEL Y
1	43	99
2	21	65
3	25	79
4	42	75
5	57	87
6	59	81
Σ	247	486

Step 2: Use the following equations to find a and b.

$$a = \frac{(\Sigma y)(\Sigma x^2) - (\Sigma x)(\Sigma xy)}{n(\Sigma x^2) - (\Sigma x)^2}$$

$$b = \frac{n(\Sigma xy) - (\Sigma x)(\Sigma y)}{n(\Sigma x^2) - (\Sigma x)^2}$$

Find a:

$((486 \times 11{,}409) - ((247 \times 20{,}485)) / 6\ (11{,}409) - 247^2)$
$484979 / 7445$
=65.14

Find b:

$(6(20{,}485) - (247 \times 486)) / (6\ (11409) - 247^2)$
$(122{,}910 - 120{,}042) / 68{,}454 - 247^2$
$2{,}868 / 7{,}445$
= .385225

Step 3: *Insert the values into the equation.*

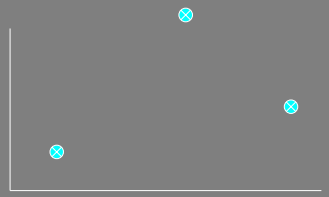
$y' = a + bx$

$y' = 65.14 + .385225x$

SSE and Linear Regression

- SSE chooses to square the difference of the predicted vs actual. Why square?
- Don't want residues to cancel each other
- Could use absolute or other distances to solve problem

  $\Sigma\,|predicted_i - actual_i|$: L1 vs L2
- SSE leads to a parabolic error surface which is good for gradient descent
- Which line would least squares choose?
  - There is always one "best" fit



Sum-squared Error (SSE)

$$SSE = \sum_{y} (y_{observed} - y_{predicted})^2$$

$$TSS = \sum_{y} (y_{observed} - \bar{y}_{observed})^2$$

$$R^2 = 1 - \frac{SSE}{TSS}$$

What is Best Fit?

- The smaller the SSE, the better the fit
- Hence,
  - Linear regression attempts to minimize SSE (or similarly to maximize R2)
- Assume 2 dimensions

$$Y = \beta_0 + \beta_1 X$$

Analytical Solution

$$\beta_0 = \frac{y - \beta_1 x}{n}$$

$$\beta_1 = \frac{n \sum xy - \sum x \sum y}{n \sum x^2 - \left(\sum x\right)^2}$$

Example (I)

x	y	x^2	xy
1.20	4.00	1.44	4.80
2.30	5.60	5.29	12.88
3.10	7.90	9.61	24.49
3.40	8.00	11.56	27.20
4.00	10.10	16.00	40.40
4.60	10.40	21.16	47.84
5.50	12.00	30.25	66.00
24.10	**58.00**	**95.31**	**223.61**

$$_1 = \frac{n \quad xy \quad x \quad y}{n \quad x^2 \quad \left(\quad x\right)^2}$$

$$= \frac{7 \quad 223.61 \quad 24.10 \quad 58.00}{7 \quad 95.31 \quad 24.10^2}$$

$$= \frac{1565.27 \quad 1397.80}{667.17 \quad 580.81}$$

$$= \frac{167.47}{86.36} = \underline{\underline{1.94}}$$

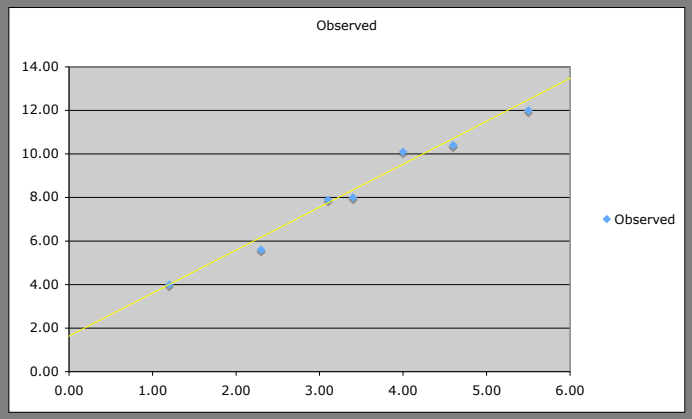
$$_0 = \frac{y \quad _1 \quad x}{n}$$

$$= \frac{58.00 \quad 1.94 \quad 24.10}{7}$$

$$= \frac{11.27}{7} = \underline{\underline{1.61}}$$

Target: $y=2x+1.5$

Example (II)



Example (III)

x	y (obs)	y (pred)	SSE	TSS
1.20	4.00	3.94	0.004	18.367
2.30	5.60	6.07	0.221	7.213
3.10	7.90	7.62	0.078	0.149
3.40	8.00	8.21	0.044	0.082
4.00	10.10	9.37	0.533	3.292
4.60	10.40	10.53	0.017	4.470
5.50	12.00	12.28	0.078	13.796
			0.975	**47.369**

$$R^2 = 1 \quad \frac{SSE}{TSS} = 1 \quad \frac{0.975}{47.369} = 0.98$$

let's say that we had data on the prices of homes on sale and the actual number of sales of homes:

Price(thousands of $)	Sales of new homes
x	y
160	126
180	103
200	82
220	75
240	82
260	40
280	20

Plot the data and check for the linear relationship between attributes if any? and find the least square regression line and compute SSE.