INTRODUCTION

History of Machine Learning, Programs vs learning algorithms, Machine Learning definition, Components of a learning, Different Types of Learning, FIND-S and Candidate-Elimination algorithm, Linear regression, Logistic Regression.

History of Machine Learning

- 1950s:
  - Samuel's checker-playing program
- 1960s:
  - Neural network: Rosenblatt's perceptron
  - Minsky & Papert prove limitations of Perceptron
- 1970s:
  - Symbolic concept induction
  - Expert systems and knowledge acquisition bottleneck
  - Quinlan's ID3
  - Natural language processing (symbolic)

1980s:

- Advanced decision tree and rule learning
- Learning and planning and problem solving
- Resurgence of neural network
- Valiant's PAC learning theory
- Focus on experimental methodology

- 90's ML and Statistics
  - Data Mining
  - Adaptive agents and web applications
  - Text learning
  - Reinforcement learning
  - Ensembles
  - Bayes Net learning

- 1994: Self-driving carroad test
- 1997: Deep Blue beats Gary Kasparov

Popularity of this field in recent time and the reasonsbehind that

- New software/ algorithms
  - Neural networks
  - Deep learning
- New hardware
  - GPU's
- Cloud Enabled
- Availability of Big Data

- 2009: Google builds self driving car
- 2011: Watson winsJeopardy
- 2014: Human vision surpassed by ML systems

(History of Machine Learning) **https://www.youtube.com/watch?v=T3PsRW6wZSY**

1

Programs vs. learning algorithms

Traditional Programming refers to any manually created program that uses input data and runs on a computer to produce the output.

In Machine Learning, also known as augmented analytics, the input data and output are fed to an algorithm to create a program. This yields powerful insights that can be used to predict future outcomes.

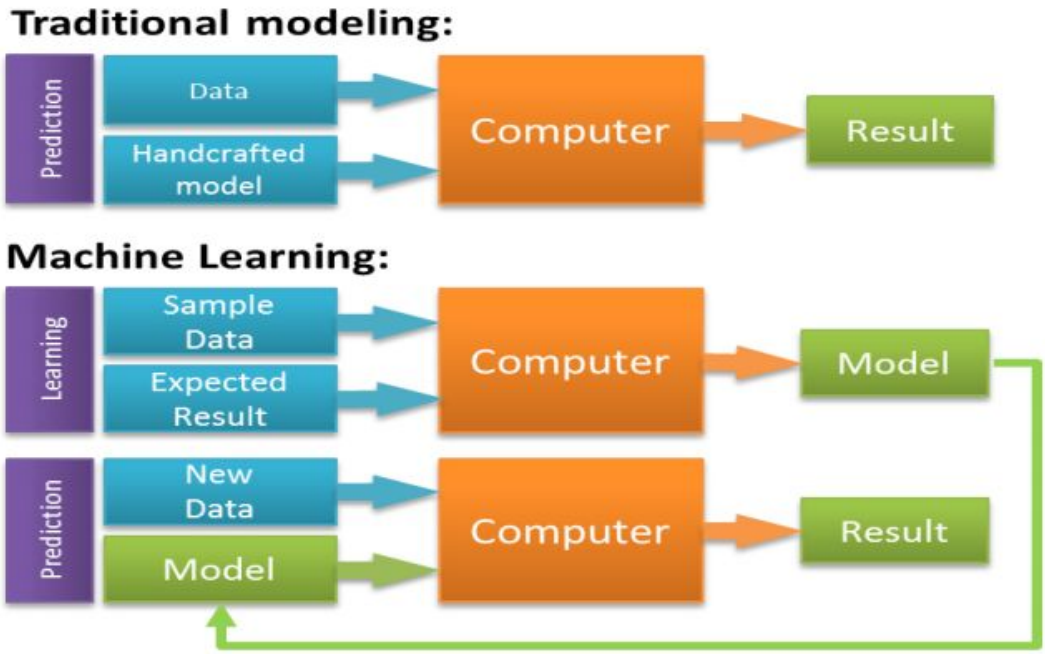


Fig: Programs vs. learning algorithms

Machine learning definition

- Arthur Samuel, a pioneer in the field of artificial intelligence and computer gaming, coined the term "Machine Learning".
- Computers learning from data are known as machine learning.
- "Field of study that gives computers the capability to learn without being explicitly programmed".
- In a very layman manner, Machine Learning (ML) can be explained as automating and improving the learning process of computers based on their experiences without being actually programmed i.e. without any human assistance.
- Learning denotes changes in a system that ... enables a system to do the same task … more efficiently the next time." - Herbert Simon
- "Learning is constructing or modifying representations of what is being experienced." - Ryszard Michalski
- According to SAS, "Machine learning is a method of data analysis that automates analytical model building. It is a branch of artificial intelligence based on the idea that systems can learn from data, identify patterns and make decisions with minimal human intervention".
- "Learning is making useful changes in our minds." - Marvin Minsky
- **Definition of Machine Learning (Mitchell 1997)** — "A **computer program** is said to learn from experience E with respect to some class of tasks T and performance measure P, if its performance at the tasks improves with the experiences".
- **"Machine learning refers to a system capable of the autonomous acquisition and integration of knowledge."**
- Machine Learning: can be defined as the practice of using algorithms to extract data, learn from it, and then forecast future trends for that topic.
- Programs that perform with better experience.
- Any computer program that improves its performance at some task through experience

Components of Learning Problem

Task: The behaviour or task being improved.

- For example: classification, acting in an environment

Data: The experiences that are being used toimprove performance in the task.

Measure of improvement:

- For example: increasing accuracy in prediction,acquiring new, improved speed and efficiency

Fruit Prediction Problem

- ✓ Task – forecasting different fruits for recognition
- ✓ Performance Measure – able to predict maximum variety of fruits
- ✓ Data or Experience – training machine with the largest datasets of fruits images

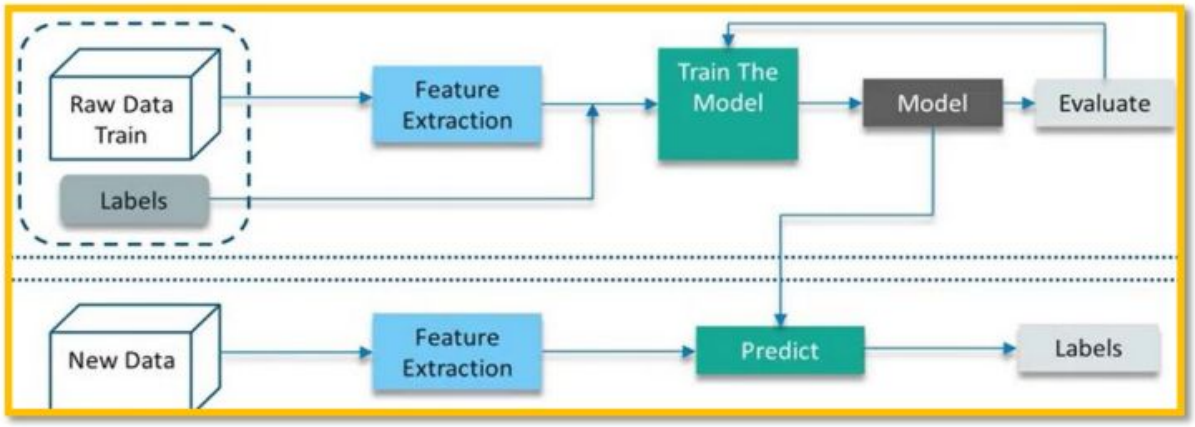
Face Recognition Problem

- ✓ Task – predicting different types of faces
- ✓ Performance Measure – able to predict maximum types of faces
- ✓ Data or Experience – training machine with maximum amount of datasets of different face images

Automatic Translation of documents

- ✓ Task – translating one type of language used in a document to other language
- ✓ Performance Measure – able to convert one language to other efficiently
- ✓ Data or Experience – training machine with a large dataset of different types of languages

Components of learning System



1) Feature Extraction + Domain knowledge

First and foremost we really need to understand what type of data we are dealing with and what eventually we want to get out of it. Essentially we need to understand how and what *features* need to be *extracted* from the data.

For instance assume we want to build software that distinguishes between male and female names. All the names in text can be thought of as our raw data while our features could be number of vowels in the name, length, first & last character, etc of the name.

2) Feature Selection

In many scenarios we end up with a lot of features at our disposal. We might want to *select* a *subset* of those based on the resources and computation power we have. In this step we

select a few of those influential features and separate them from the not-so-influential features. There are many ways to do this, information gain, gain ratio, correlation etc.

3) Choice of Algorithm

There are wide range of algorithms from which we can choose based on whether we are trying to do prediction, classification or clustering. We can also choose between linear and non-linear algorithms. Naive Bayes, Support Vector Machines, Decision Trees, k-Means Clustering are some common algorithms used.

4) Training

In this step we tune our algorithm based on the data we already have. This data is called training set as it is used to train our algorithm. This is the part where our machine or software learn and improve with experience.

5) Choice of Metrics/Evaluation Criteria

Here we decide our evaluation criteria for our algorithm. Essentially we come up with metrics to evaluate our results. Commonly used measures of performance are precision, recall, f1-measure, robustness, specificity-sensitivity, error rate etc.

6) Testing

Lastly, we test how our machine learning algorithm performs on an unseen set of test cases. One way to do this is to partition the data into training and testing set. The training set is used in step 4 while the test set is then used in this step. Techniques such as *cross-validation* and *leave-one-out* can be used to deal with scenarios where we do not have enough data.

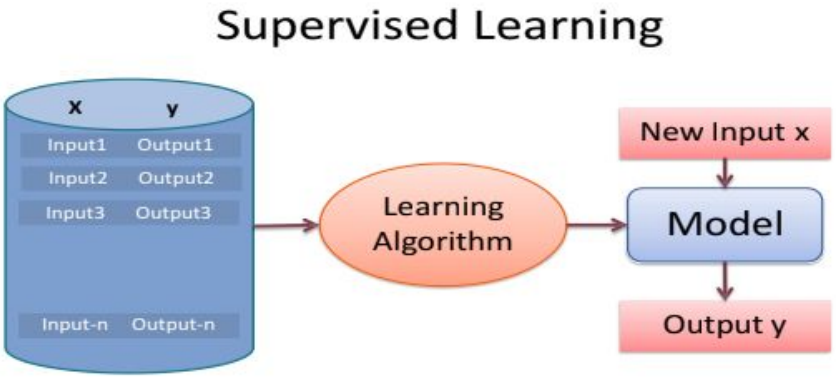
Different Types of learning (https://www.youtube.com/watch?v=T3PsRW6wZSY)

- **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.

Supervised Learning

A training set of examples with the correct responses (targets) is provided and, based on this training set, the algorithm generalises to respond correctly to all possible inputs. This is also called learning from exemplars. Supervised learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs.

In supervised learning, each example in the training set is a pair consisting of an input object (typically a vector) and an output value. A supervised learning algorithm analyzes the training data and produces a function, which can be used for mapping new examples. In the optimal case, the function will correctly determine the class labels for unseen instances. Both classification and regression problems are supervised learning problems. A wide range of supervised learning algorithms are available, each with its strengths and weaknesses. There is no single learning algorithm that works best on all supervised learning problems.



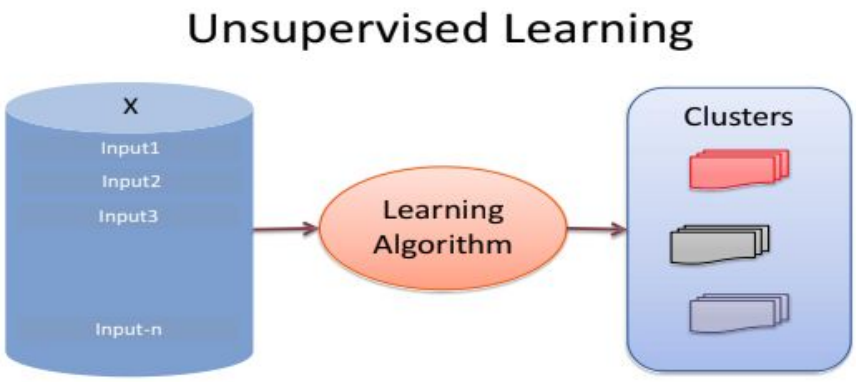
- **Predictive Modeling (Supervised Learning)**: Building a model for the target variable as a function of the explanatory variable.
  - **Classification:** Which is used for Discrete Target Variables

    Ex: Predicting whether a web user will make a purchase at an online book store (Target variable is binary valued).
  - **Regression:** Which is used for Continuous Target Variables.
  - Ex: Forecasting the future price of a stock (Price is a continuous-valued attribute)

Unsupervised Learning:

Correct responses are not provided, but instead the algorithm tries to identify similarities between the inputs so that inputs that have something in common are categorized together. The statistical approach to unsupervised learning is known as density estimation.

Unsupervised learning is a type of machine learning algorithm used to draw inferences from datasets consisting of input data without labeled responses. In unsupervised learning algorithms, a classification or categorization is not included in the observations. There are no output values and so there is no estimation of functions. Since the examples given to the learner are unlabeled, the accuracy of the structure that is output by the algorithm cannot be evaluated. The most common unsupervised learning method is cluster analysis, which is used for exploratory data analysis to find hidden patterns or grouping in data.



Cluster Analysis:

- Grouping of similar things is called cluster.
- The objects are **clustered** or **grouped** based on the principle of **maximizing the intra class similarity (Within a Cluster)** and **minimizing the interclass similarity(Cluster to Cluster)**.

Example:

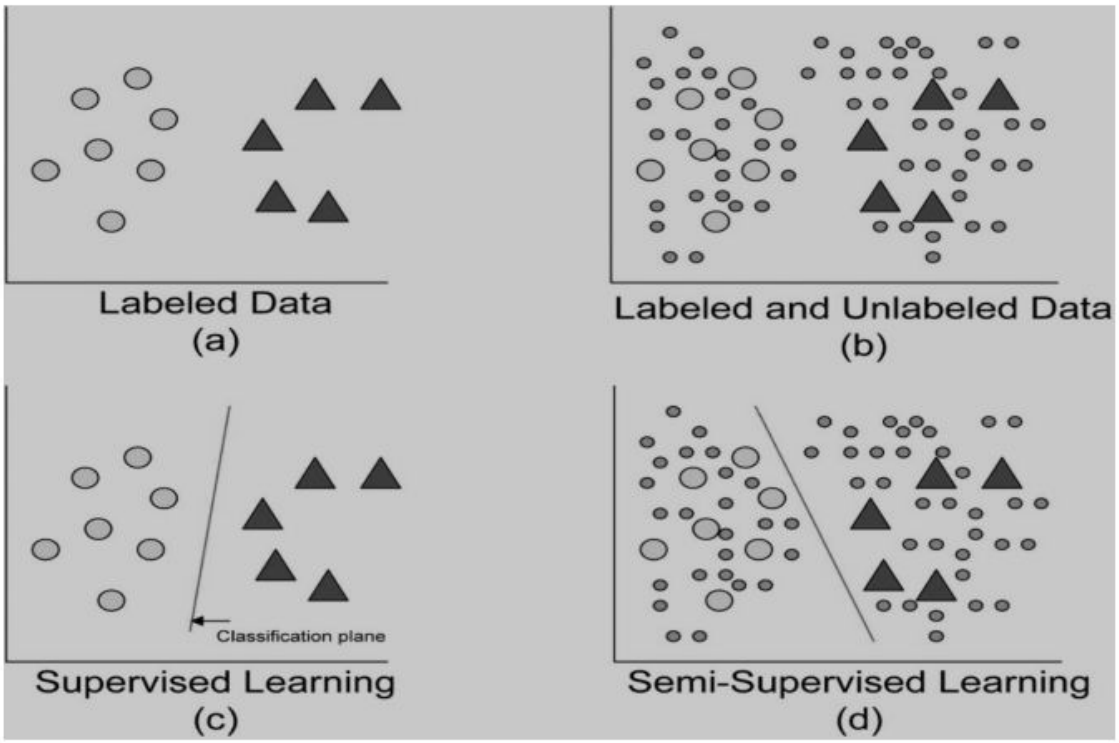
Article	Word
1	Dollar : 1, Industry : 4, Country : 2, Loan : 3, Deal : 2, Government : 2
2	Machinery : 2, Labor : 3, Market : 4, Industry : 2, Work : 3, Country : 1
3	Domestic: 4, Forecast : 2, Gain : 1, Market : 3, Country : 2, Index : 3
4	Patient : 4, Symptom : 2, Drug : 3, Health : 2, Clinic : 2, Doctor : 2
5	Death : 2, Cancer : 4, Drug : 3, Public : 4, Health : 3, Director : 2
6	Medical : 2, Cost : 3, Increase : 2, Patient : 2, Health : 3, Care : 1

Document Clustering

- Each Article is represented as a set of word frequency pairs (*w, c*), where w is a word and c is the number of times the word appears in the article.
- There are 2 natural clusters in the above dataset
- First Cluster consists of the first 3 articles (News about the Economy)
- Second cluster contain last 3 articles (News about the Heath Care)

Semi-supervised Learning

Semi-supervised learning is an approach to machine learning that combines a small amount of labeled data with a large amount of unlabeled data during training. Semi-supervised learning falls between unsupervised learning and supervised learning. It is a special instance of weak supervision.



- **Reinforcement Learning**

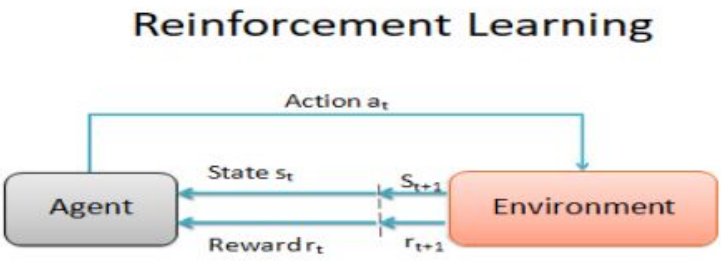
This is somewhere between supervised and unsupervised learning. The algorithm gets told when the answer is wrong, but does not get told how to correct it. It has to explore and try out different possibilities until it works out how to get the answer right. Reinforcement learning is sometime called learning with a critic because of this monitor that scores the answer, but does not suggest improvements.

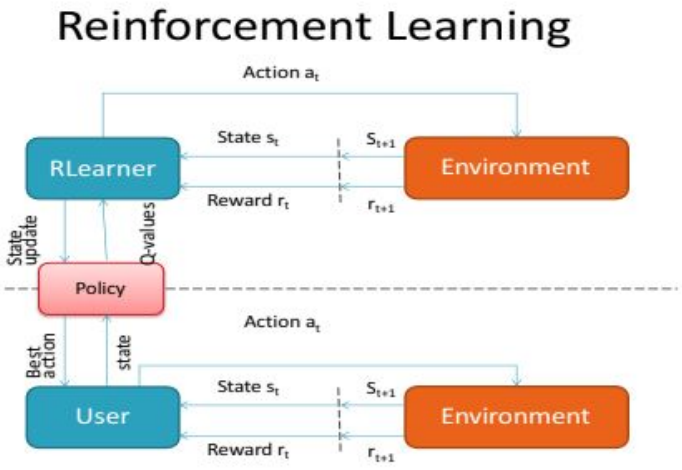
Reinforcement learning is the problem of getting an agent to act in the world so as to maximize its rewards. A learner (the program) is not told what actions to take as in most forms of machine learning, but instead must discover which actions yield the most reward by trying them. In the most interesting and challenging cases, actions may affect not only the immediate reward but also the next situations and, through that, all subsequent rewards.

Example

Consider teaching a dog a new trick: we cannot tell it what to do, but we can reward/punish it if it does the right/wrong thing. It has to find out what it did that made it get the reward/punishment. We can use a similar method to train computers to do many tasks, such as playing backgammon

or chess, scheduling jobs, and controlling robot limbs. Reinforcement learning is different from supervised learning. Supervised learning is learning from examples provided by a knowledgeable expert.





Why is Machine Learning Important?

- Some tasks cannot be defined well, except by examples (e.g., recognizing people).
- Relationships and correlations can be hidden within large amounts of data. Machine Learning/Data Mining may be able to find these relationships.
- Human designers often produce machines that do not work as well as desired in the environments in which they are used.
- The amount of knowledge available about certain tasks might be too large for explicit encoding by humans (e.g., medical diagnostic).
- Environments change over time.
- New knowledge about tasks is constantly being discovered by humans. It may be difficult to continuously re-design systems "by hand".

Well-Posed Learning Problems

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

To have a well-defined learning problem, three features needs to be identified:

1. The class of tasks
2. The measure of performance to be improved
3. The source of experience

Examples

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

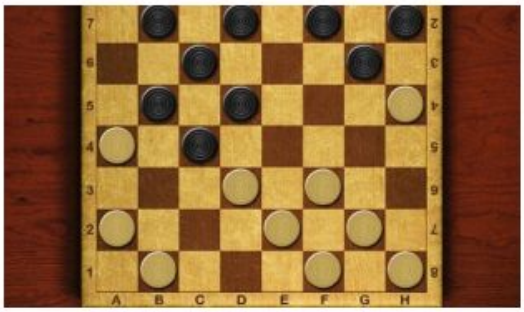


Fig: Checker game board

***A checkers learning problem*:**

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

2. ***A handwriting recognition learning problem:***
   - Task T: recognizing and classifying handwritten words within images
   - Performance measure P: percent of words correctly classified
   - Training experience E: a database of handwritten words with givenclassifications
3. ***A robot driving learning problem*:**
   - Task T: driving on public four-lane highways using vision sensors
   - Performance measure P: average distance travelled before an error (as judged by human overseer)
   - Training experience E: a sequence of images and steering commands recorded while observing a human driver

Issues in Machine Learning

The field of machine learning, and much of this book, is concerned with answering questions such as the following

- What algorithms exist for learning general target functions from specific training examples? In what settings will particular algorithms converge to the desired function, given sufficient training data? Which algorithms perform best for which types of problems and representations?
- How much training data is sufficient? What general bounds can be found to relate the confidence in learned hypotheses to the amount of training experience and the character of the learner's hypothesis space?
- When and how can prior knowledge held by the learner guide the process of generalizing from examples? Can prior knowledge be helpful even when it is only approximately correct?
- What is the best strategy for choosing a useful next training experience, and how does the choice of this strategy alter the complexity of the learning problem?
- What is the best way to reduce the learning task to one or more function approximation problems? Put another way, what specific functions should the system attempt to learn? Can this process itself be automated?
- How can the learner automatically alter its representation to improve its ability to represent and learn the target function?

CONCEPT LEARNING

- Learning involves acquiring general concepts from specific training examples. Example: People continually learn general concepts or categories such as "bird," "car," "situations in which I should study more in order to pass the exam," etc.
- Each such concept can be viewed as describing some subset of objects or events defined over a larger set
- Alternatively, each concept can be thought of as a Boolean-valued function defined over this larger set. (Example: A function defined over all animals, whose value is true for birds and false for other animals).

Definition: Concept learning **-** Inferring a Boolean-valued function from training examples of its input and output

A CONCEPT LEARNING TASK

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

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

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

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

What hypothesis representation is provided to the learner?

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

For each attribute, the hypothesis will either

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

If some instance ***x*** satisfies all the constraints of hypothesis ***h***, then ***h*** classifies ***x*** as a positive example (***h(x) = 1***).

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

(?, Cold, High, ?, ?, ?)

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

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

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

(Φ, Φ, Φ, Φ, Φ, Φ)

Notation

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

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

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

c: X→ {O, 1}

Example: The target concept corresponds to the value of the attribute ***EnjoySport*** (i.e., c(x) = 1 if ***EnjoySport*** = Yes, and c(x) = 0 if ***EnjoySport*** = No).

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

h: X→{O, 1}

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

- Given:
  - Instances X: Possible days, each described by the attributes
    - *Sky* (with possible values Sunny, Cloudy, and Rainy),
    - *AirTemp* (with values Warm and Cold),
    - *Humidity* (with values Normal and High),
    - *Wind* (with values Strong and Weak),
    - *Water* (with values Warm and Cool),
    - *Forecast* (with values Same and Change).
  - Hypotheses *H*: Each hypothesis is described by a conjunction of constraints on the attributes *Sky, AirTemp, Humidity, Wind, Water*, and *Forecast*. The constraints may be "?" (any value is acceptable), "Φ" (no value is acceptable), or a specific value.
  - Target concept *c*: ***EnjoySport*** : X → {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.

Table: The ***EnjoySport*** concept learning task.

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 unobservedexamples.

CONCEPT LEARNING AS SEARCH

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

Example:
Consider the instances X and hypotheses H in the *EnjoySport* learning task. The attribute Sky has three possible values, and *AirTemp, Humidity, Wind, Water, Forecast* each have two possible values, the instance space X contains exactly

3.2.2.2.2.2 = 96 distinct instances
5.4.4.4.4.4 = 5120 syntactically distinct hypotheses within H.

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

1 + (4.3.3.3.3.3) = 973. Semantically distinct hypotheses

General-to-Specific Ordering of Hypotheses

Consider the two hypotheses

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

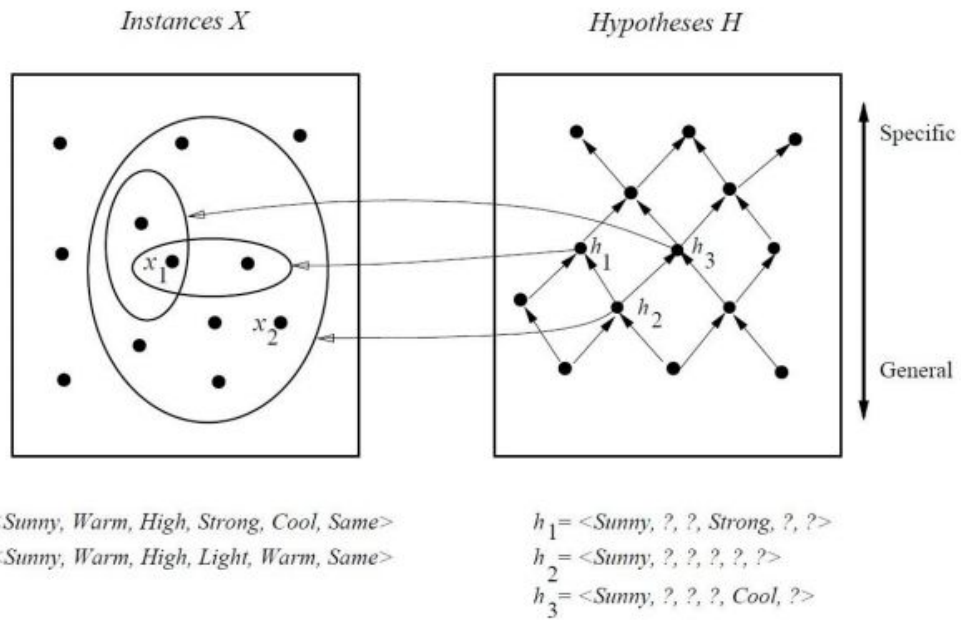
- Consider the sets of instances that are classified positive by h1 and by h2.
- h2 imposes fewer constraints on the instance, it classifies more instances as positive. So, any instance classified positive by h1 will also be classified positive by h2. Therefore, h2 is more general than h1.

Given hypotheses h_j and h_k, h_j is more-general-than or- equal do h_k if and only if any instancethat satisfies h_k also satisfies h_i

Definition: Let h_j and h_k be Boolean-valued functions defined over X. Then h_j is ***more general-than-or-equal-to*** h_k (written $h_j \geq h_k$) if and only if

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

15



- In the figure, the box on the left represents the set X of all instances, the box on the right the set H of all hypotheses.
- Each hypothesis corresponds to some subset of X-the subset of instances that it classifies positive.
- The arrows connecting hypotheses represent the more - general -than relation, with the arrow pointing toward the less general hypothesis.
- Note the subset of instances characterized by h2 subsumes the subset characterized by hl , hence h2 is more - general– than h1

FIND-S: FINDING A MAXIMALLY SPECIFIC HYPOTHESIS

FIND-S Algorithm

1. Initialize h to the most specific hypothesis in H
2. For each positive training instance x

   For each attribute constraint a_i in h

   If the constraint a_i is satisfied by x

   Then do nothing

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

To illustrate this algorithm, assume the learner is given the sequence of training examples from the ***EnjoySport*** task

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

- The first step of FIND-S is to initialize h to the most specific hypothesis in H

  h - (Ø, Ø, Ø, Ø, Ø, Ø)

- Consider the first training example

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

  Observing the first training example, it is clear that hypothesis *h* is too specific. None of the "Ø" constraints in h are satisfied by this example, so each is replaced by the next *more general constraint* that fits the example

  h1 = <Sunny Warm Normal Strong Warm Same>

- Consider the second training example

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

  The second training 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

  h2 = <Sunny Warm ? Strong Warm Same>

- Consider the third training example

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

  Upon encountering the third training the algorithm makes no change to h. The FIND-S algorithm simply ignores every negative example.

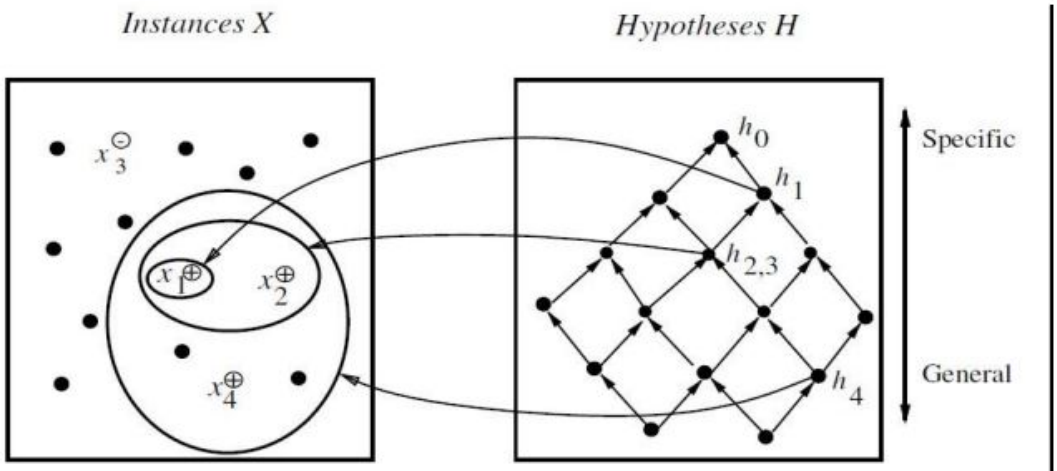
  h3 = < Sunny Warm ? Strong Warm Same>

- Consider the fourth training example

  x4 = <Sunny Warm High Strong Cool Change>, +

The fourth example leads to a further generalization of h

h4 = < Sunny Warm ? Strong ? ? >



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 San
h_2 = <Sunny Warm ? Strong Warm Same>
h_3 = <Sunny Warm ? Strong Warm Same>
h_4 = <Sunny Warm ? Strong ? ? >

The key property of the FIND-S algorithm

- FIND-S is guaranteed to output the most specific hypothesis within H that is consistent with the positive training examples
- FIND-S algorithm's 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.

Unanswered by FIND-S

1. Has the learner converged to the correct target concept?
2. Why prefer the most specific hypothesis?
3. Are the training examples consistent?
4. What if there are several maximally specific consistent hypotheses?

VERSION SPACES AND THE CANDIDATE-ELIMINATION ALGORITHM

The key idea in the CANDIDATE-ELIMINATION algorithm is to output a description of theset of all ***hypotheses consistent with the training examples***

Representation

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

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

Note difference between definitions of *consistent* and *satisfies*

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

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

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

The LIST-THEN-ELIMINATION algorithm

The LIST-THEN-ELIMINATE algorithm first initializes the version space to contain all hypotheses in H and then eliminates any hypothesis found inconsistent with any training example.

1. ***VersionSpace c*** a list containing every hypothesis in H
2. For each training example, (x, c(x))
   remove from ***VersionSpace*** any hypothesis h for which h(x) ≠ c(x)
3. Output the list of hypotheses in ***VersionSpace***

The LIST-THEN-ELIMINATE Algorithm

- List-Then-Eliminate works in principle, as long as version space is finite.
- However, since it requires exhaustive enumeration of all hypotheses in practice it is not feasible.

A More Compact Representation for Version Spaces

The version space is represented by its most general and least general members. These members form general and specific boundary sets that delimit the version space within the partially ordered hypothesis space.

Definition: The **general boundary** G, with respect to hypothesis space ***H*** and training data *D*, is the set of maximally general members of *H* consistent with *D*

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

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

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

CANDIDATE-ELIMINATION Learning Algorithm

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

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- ELIMINTION algorithm using version spaces

An Illustrative 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

CANDIDATE-ELIMINTION algorithm begins by initializing the version space to the set ofall hypotheses in H;

Initializing the G boundary set to contain the most general hypothesis in H

$G0 \langle ?, ?, ?, ?, ?, ? \rangle$

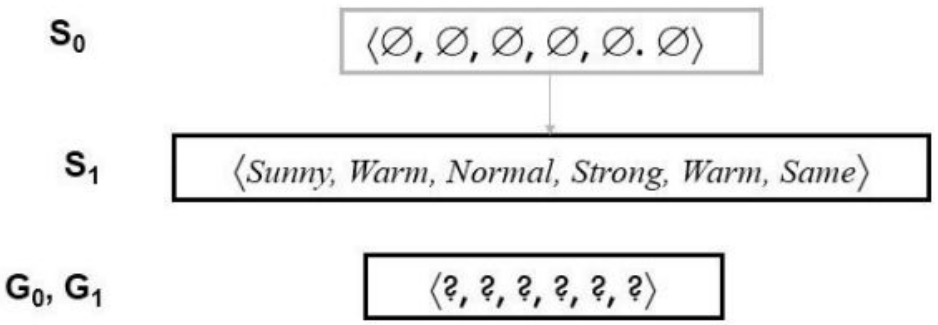
Initializing the S boundary set to contain the most specific (least general) hypothesis

$S0 \langle \varnothing, \varnothing, \varnothing, \varnothing, \varnothing, \varnothing \rangle$

- When the first training example is presented, the CANDIDATE-ELIMINTION algorithm checks the S boundary and finds that it is overly specific and it fails to cover the positive example.
- The boundary is therefore revised by moving it to the least more general hypothesis that covers this new example
- No update of the G boundary is needed in response to this training example because Go correctly covers this example

For training example d,

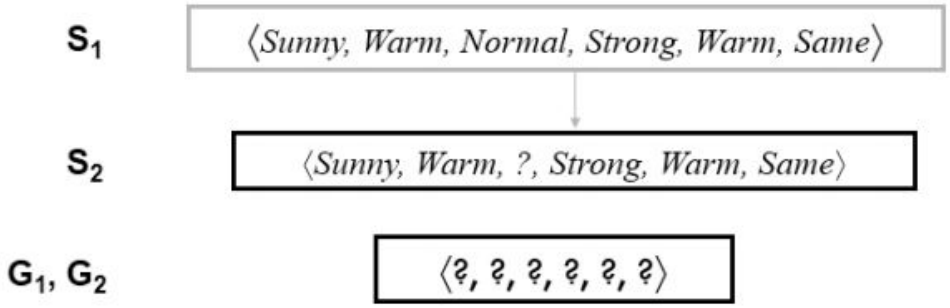
$\langle$Sunny, Warm, Normal, Strong, Warm, Same $\rangle$ +



- When the second training example is observed, it has a similar effect of generalizing S further to S_2, leaving G again unchanged i.e., $G_2 = G_1 = G_0$

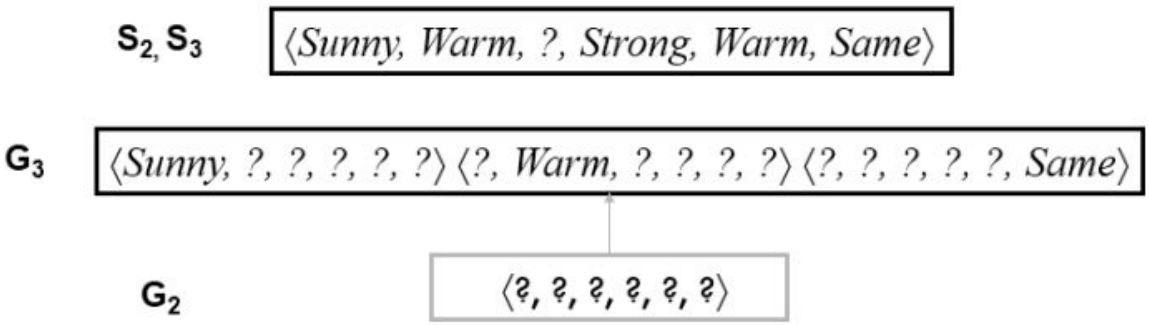
21

For training example d, ⟨Sunny, Warm, High, Strong, Warm, Same⟩ +



- Consider the third training example. This negative example reveals that the G boundary of the version space is overly general, that is, the hypothesis in G incorrectly predicts that this new example is a positive example.
- The hypothesis in the G boundary must therefore be specialized until it correctly classifies this new negative example

For training example d, ⟨Rainy, Cold, High, Strong, Warm, Change ⟩ –



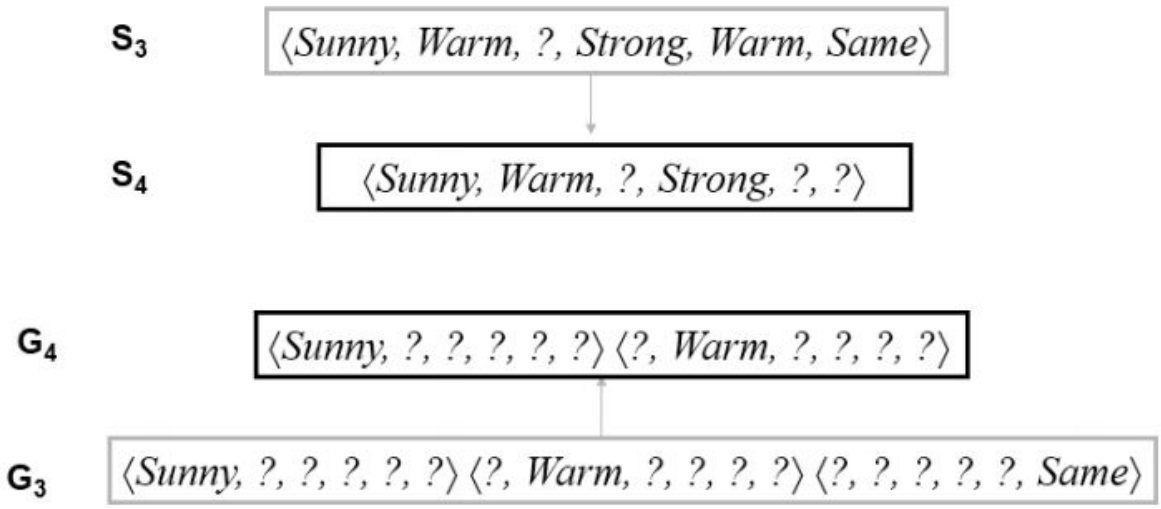
Given that there are six attributes that could be specified to specialize G2, why are there only three new hypotheses in G3?

For example, the hypothesis h = (?, ?, Normal, ?, ?, ?) is a minimal specialization of G2 that correctly labels the new example as a negative example, but it is not included in G3. The reason this hypothesis is excluded is that it is inconsistent with the previously encountered positive examples

- Consider the fourth training example.

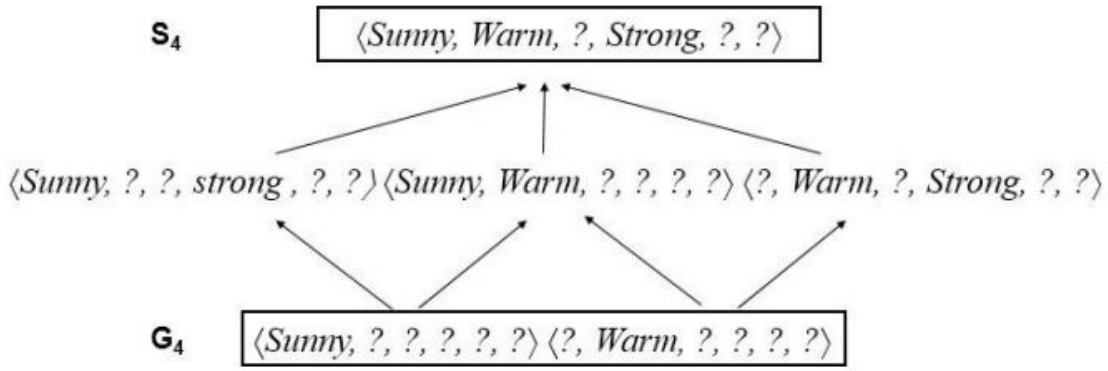
For training example d,

⟨Sunny, Warm, High, Strong, Cool Change ⟩ +



- This positive example further generalizes the S boundary of the version space. It also results in removing one member of the G boundary, because this member fails to cover the new positive example

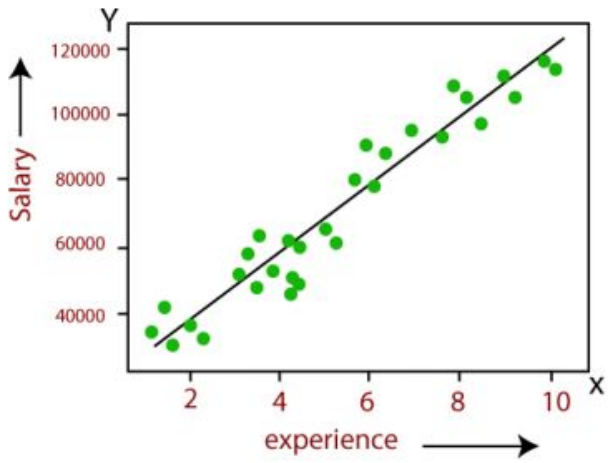
After processing these four examples, the boundary sets S4 and G4 delimit the version space of all hypotheses consistent with the set of incrementally observed training examples.



23

Linear Regression:

- Linear regression is a statistical regression method which is used for predictive analysis.
- It is one of the very simple and easy algorithms which works on regression and shows the relationship between the continuous variables.
- It is used for solving the regression problem in machine learning.
- Linear regression shows the linear relationship between the independent variable (X-axis) and the dependent variable (Y-axis), hence called linear regression.
- If there is only one input variable (x), then such linear regression is called **simple linear regression** and if there is more than one input variable, then such linear regression is called **multiple linear regression**.
- The relationship between variables in the linear regression model can be explained using the below image. Here we are predicting the salary of an employee on the basis of the year of experience.



Below is the mathematical equation for linear regression:

Y= aX+b, Here, Y = dependent variables (target variables), X= Independent variables (predictor variables), a and b are the linear coefficients

Some popular applications of linear regression are:

- Analyzing trends and sales estimates
- Salary forecasting
- Real estate prediction
- Arriving at ETAs in traffic

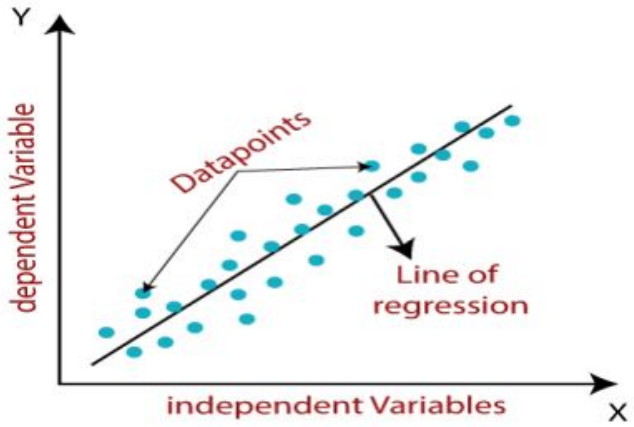
Or

Linear Regression:

Linear regression is one of the easiest and most popular Machine Learning algorithms. It is a statistical method that is used for predictive analysis. Linear regression makes predictions for continuous/real or numeric variables suchas **sales, salary, age, product price,** etc.

Linear regression algorithm shows a linear relationship between a dependent (y) and one or more independent (y) variables, hence called as linear regression. Since linear regression shows the linear relationship, which means it finds how the value of the dependent variable is changing according to the value of the independent variable.

The linear regression model provides a sloped straight line representing the relationship between the variables. Consider the below image:



Mathematically, we can represent a linear regression as: $y= a_0+a_1x+ \varepsilon$

Here,

Y= Dependent Variable (Target Variable)

X= Independent Variable (predictor Variable)

a0= intercept of the line (Gives an additional degree of freedom)

a1 = Linear regression coefficient (scale factor to each input value).

ε = random error

The values for x and y variables are training datasets for Linear Regression model representation.

Types of Linear Regression

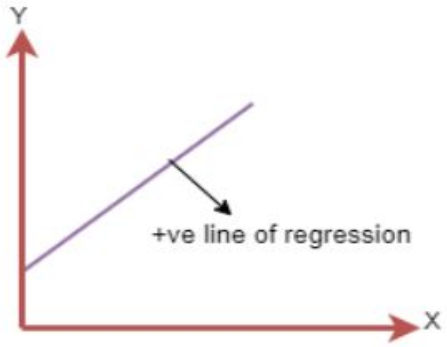
Linear regression can be further divided into two types of the algorithm:

- **Simple Linear Regression**: If a single independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Simple Linear Regression.
- **Multiple Linear Regression**: If more than one independent variable is used to predict the value of a numerical dependent variable, then such a Linear Regression algorithm is called Multiple Linear Regression.

  Linear Regression Line: A linear line showing the relationship between the dependent and independent variables is called a regression line.

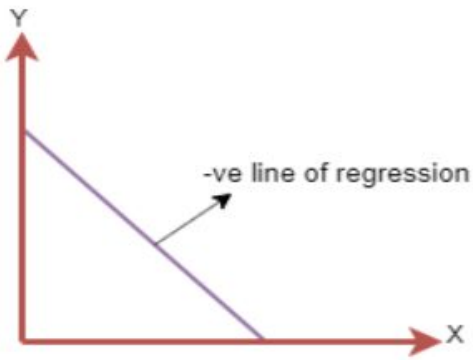
  A regression line can show two types of relationship:

  - **Positive Linear Relationship**: If the dependent variable increases on the Y-axis and independent variable increases on X-axis, then such a relationship is termed as a Positive linear relationship.



The line equation will be: $Y= a_0+a_1x$

  - **Negative Linear Relationship**: If the dependent variable decreases on the Y-axis and independent variable increases on the X-axis, then such a relationship is called a negative linear relationship.



The line of equation will be: $Y= -a_0+a_1x$

26

Finding the best fit line:

When working with linear regression, our main **goal is to find the best fit line** that means the error between **predicted values and actual values** should be minimized. The best fit line will have the least error.

The different values for weights or the coefficient of lines (a0, a1) gives a different line of regression, so we need to calculate the best values for a0 and a1 to find the best fit line, so to calculate this we use cost function.

Cost function

- The different values for weights or coefficient of lines (a0, a1) gives the different line of regression, and the cost function is used to estimate the values of the coefficient for the best fit line.
- Cost function optimizes the regression coefficients or weights. It measures how a linear regression model is performing.
- We can use the cost function to find the accuracy of the mapping function, which maps the input variable to the output variable. This mapping function is also known as Hypothesis function.

For Linear Regression, we use the **Mean Squared Error** (MSE) cost function, which is the average of squared error occurred ***between the predicted values and actual values***. It can be written as:

For the above linear equation, MSE can be calculated as:

$$MSE = 1\frac{1}{N}\sum_{i=1}^{n}(y_i - (a_1x_i + a_0))^2$$

Where, N=Total number of observation, Yi = Actual value, (a1xi+a0)= Predicted value.

Residuals: The distance between the actual value and predicted values is called residual. If the observed points are far from the regression line, then the residual will be high, and so cost function will high. If the scatter points are close to the regression line, then the residual will be small and hence the cost function.

Gradient Descent: o Gradient descent is used to minimize the MSE by calculating the gradient of the cost function.

- A regression model uses gradient descent to update the coefficients of the line by reducing the cost function.
- It is done by a random selection of values of coefficient and then iteratively updates the values to reach the minimum cost function.

27

Model Performance: The Goodness of fit determines how the line of regression fits the set of observations. The process of finding the best model out of various models is called ***optimization***. It can be achieved by below method:

28

Example:

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

Σx = 247, Σy = 486, Σxy = 20485, Σx2 = 11409, Σy2 = 40022. n is the sample size (6, in our case).

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

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

Σx = 247, Σy = 486, Σxy = 20485, Σx2 = 11409, Σy2 = 40022. n is the sample size (6, in our case).

Find a: ((486 × 11,409) – ((247 × 20,485))
/ 6 (11,409) – 247^2)

=484979 / 7445
=**65.14**

Find b:
(6(20,485) – (247 × 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

29

30

CHAPTER

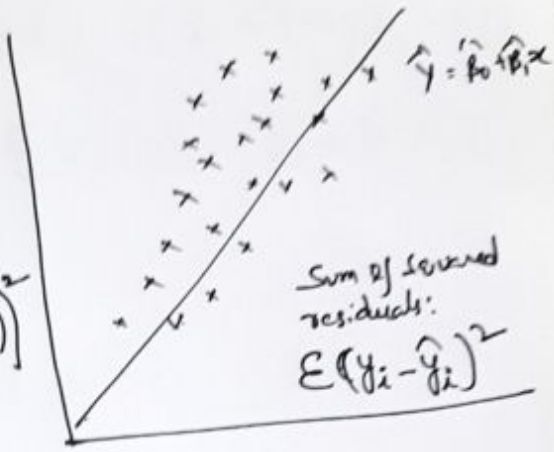
Linear Regression:-

→ The least squares method involves minimizing the sum of squared residuals.

→ Sum of squared residuals :

$$= \sum (y_i - \hat{y}_i)^2$$

→ $\Rightarrow \sum (y_i - \hat{y}_i)^2 = \sum (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$



→ How do we do that ?

1. Take the partial derivatives ~~equal to 0~~ w.r.t $\hat{\beta}_0$ and $\hat{\beta}_1$
2. Set the partial derivatives equal to 0
3. Solve ~~for~~ for $\hat{\beta}_0$ and $\hat{\beta}_1$

1. ⇒ Taking the partial derivative with respect to $\hat{\beta}_0$:

$$\frac{\partial}{\partial \hat{\beta}_0} \sum (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2 = \sum \frac{\partial}{\partial \hat{\beta}_0} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))^2$$

$$= \sum 2 (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)) (-1)$$

$$= -2 \sum (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i))$$

$\left(\because \frac{\partial}{\partial \hat{\beta}_0} (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)) = -1 \right)$

⇒ Now w.r.to $\hat{\beta}_1$:

$$\frac{\partial}{\partial \hat{\beta}_1} \Sigma (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))^2 = \Sigma \frac{\partial}{\partial \hat{\beta}_1} (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))^2$$

$$= \Sigma 2(Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))(-X_i)$$

$$= -2 \Sigma X_i (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i))$$

$$\because \frac{\partial}{\partial \hat{\beta}_1} (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)) = -X_i$$

② ⇒ Setting the partial derivatives equal to 0 :

$$-2 \Sigma (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)) = 0$$

$$-2 \Sigma X_i (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)) = 0$$

③ ⇒ Solving for $\hat{\beta}_0$:

$$\Sigma (Y_i - (\hat{\beta}_0 + \hat{\beta}_1 X_i)) = 0$$

$$\Sigma Y_i - \Sigma \hat{\beta}_0 - \Sigma \hat{\beta}_1 X_i = 0$$

$$\Sigma Y_i - n\hat{\beta}_0 - \hat{\beta}_1 \Sigma X_i = 0$$

$$n\hat{\beta}_0 = \frac{\Sigma Y_i}{n} - \frac{\hat{\beta}_1 \Sigma X_i}{n}$$

$$\hat{\beta}_0 = \bar{Y} - \hat{\beta}_1 \bar{X}$$

⇒ Solving for $\hat{\beta}_1$:

$$\sum x_i (y_i - (\hat{\beta}_0 + \hat{\beta}_1 x_i)) = 0$$

Substituting $\bar{y} - \hat{\beta}_1 \bar{x}$ for β_0

$$\sum x_i (y_i - (\bar{y} - \hat{\beta}_1 \bar{x} + \hat{\beta}_1 x_i)) = 0$$

$$\sum x_i (y_i - \bar{y} - \hat{\beta}_1 (x_i - \bar{x})) = 0$$

$$\sum x_i (y_i - \bar{y}) - \sum \hat{\beta}_1 x_i (x_i - \bar{x}) = 0$$

$$\sum x_i (y_i - \bar{y}) = \hat{\beta}_1 \sum x_i (x_i - \bar{x})$$

$$\hat{\beta}_1 = \frac{\sum x_i (y_i - \bar{y})}{\sum x_i (x_i - \bar{x})}$$

$$= \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

← Sum of products

← Sum of squares of x

$\because \sum (x_i - \bar{x})(y_i - \bar{y}) = \sum x_i (y_i - \bar{y}) - \sum \bar{x} (y_i - \bar{y})$

$= \sum x_i (y_i - \bar{y}) - \bar{x} \sum (y_i - \bar{y})$

$= \sum x_i (y_i - \bar{y})$

$$\sum (y_i - \bar{y}) = \sum y_i - n\bar{y} = 0$$

Note:

$$\sum (x_i - \bar{x})^2 = \sum (x_i - \bar{x})(x_i - \bar{x})$$

$$= \sum x_i (x_i - \bar{x}) - \sum \bar{x} (x_i - \bar{x})$$

$$= \sum x_i (x_i - \bar{x}) - \bar{x} \sum (x_i - \bar{x})$$

$$= \sum x_i (x_i - \bar{x})$$

$$\sum (y_i - \bar{y})^2 = \sum y_i (y_i - \bar{y})$$

$$\hat{\beta}_0 = \bar{y} - \hat{\beta}_1 \bar{x}$$

$$\hat{\beta}_1 = \frac{\sum (x_i - \bar{x})(y_i - \bar{y})}{\sum (x_i - \bar{x})^2}$$

Example:

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

Σx = 247, Σy = 486, Σxy = 20485, Σx2 = 11409, Σy2 = 40022. n is the sample size (6, in our case).

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

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

Σx = 247, Σy = 486, Σxy = 20485, Σx2 = 11409, Σy2 = 40022. n is the sample size (6, in our case).

Find a: ((486 × 11,409) – ((247 × 20,485))
/ 6 (11,409) – 247^2)

=484979 / 7445
=**65.14**

Find b:
(6(20,485) – (247 × 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

30

Decision Trees: Basic decision trees learning algorithm, inductive bias in decision tree learning, Random Forest algorithm, over fitting.

Decision tree learning is a method for approximating discrete-valued target functions, in which the learned function is represented by a decision tree.

DECISIONTREEREPRESENTATION

- Decision trees classify instances by sorting them down the tree from the root to some leaf node, which provides the classification of the instance.
- Each node in the tree specifies a test of some attribute of the instance, and each branch descending from that node corresponds to one of the possible values for this attribute.
- An instance is classified by starting at the root node of the tree, testing the attribute specified by this node, then moving down the tree branch corresponding to the value of the attribute in the given example. This process is then repeated for the subtree rooted at the new node.

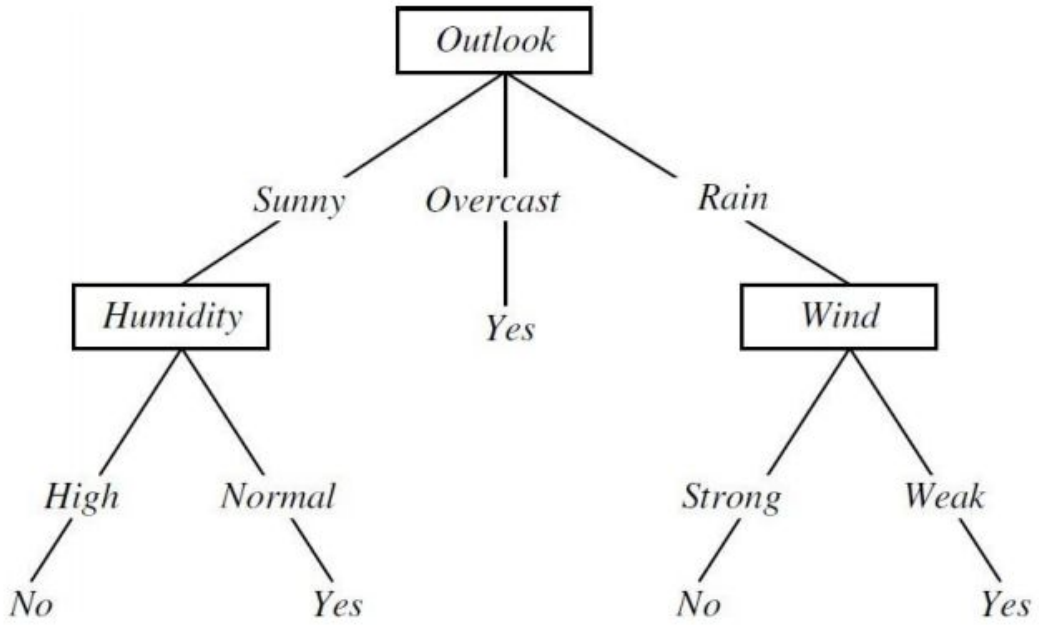


FIGURE: A decision tree for the concept ***Play Tennis***. An example is classified by sorting it through the tree to the appropriate leaf node, then returning the classification associated with this leaf

- Decision trees represent a disjunction of conjunctions of constraints on the attribute values of instances.
- Each path from the tree root to a leaf corresponds to a conjunction of attribute tests, and the tree itself to a disjunction of these conjunctions

For example, the decision tree shown in above figure corresponds to the expression

(Outlook = Sunny A Humidity = Normal)
V (Outlook = Overcast)
V (Outlook = Rain A Wind = Weak)

APPROPRIATE PROBLEMS FOR DECISION TREE LEARNING

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

1. ***Instances are represented by attribute-value pairs*** – Instances are described by a fixed set of attributes and their values

2. ***The target function has discrete output values*** – The decision tree assigns a Boolean classification (e.g., yes or no) to each example. Decision tree methods easily extend to learning functions with more than two possible output values.

3. ***Disjunctive descriptions may be required***

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

5. ***The training data may contain missing attribute values*** – Decision tree methods can be used even when some training examples have unknown values

2

THE BASIC DECISION TREE LEARNING ALGORITHM

The basic algorithm is ID3 which learns decision trees by constructing them top-down

---

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

---

* The best attribute is the one with highest information gain

TABLE: Summary of the ID3 algorithm specialized to learning Boolean-valued functions. ID3 is a greedy algorithm that grows the tree top-down, at each node selecting the attribute that best classifies the local training examples. This process continues until the tree perfectly classifies the training examples, or until all attributes have been used

Which Attribute Is the Best Classifier?

- The central choice in the ID3 algorithm is selecting which attribute to test at each node in the tree.
- A statistical property called ***information gain*** that measures how well a given attribute separates the training examples according to their target classification.
- ID3 uses ***information gain*** measure to select among the candidate attributes at each step while growing the tree.

ENTROPY MEASURES HOMOGENEITY OF EXAMPLES

To define information gain, we begin by defining a measure called entropy. *Entropy measures the impurity of a collection of examples.*

Given a collection S, containing positive and negative examples of some target concept, the entropy of S relative to this Boolean classification is

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

Where,

p_+ is the proportion of positive examples in S
$p-$ is the proportion of negative examples in S.

Example:
Suppose S is a collection of 14 examples of some boolean concept, including 9 positive and 5 negative examples. Then the entropy of S relative to this boolean classification is

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

- The entropy is 0 if all members of S belong to the same class
- The entropy is 1 when the collection contains an equal number of positive and negative examples
- If the collection contains unequal numbers of positive and negative examples, the entropy is between 0 and 1

INFORMATION GAIN MEASURES THE EXPECTED REDUCTION IN ENTROPY

- ***Information gain,*** is the expected reduction in entropy caused by partitioning the examples according to this attribute.
- The information gain, Gain(S, A) of an attribute A, relative to a collection of examples S, is defined as

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

Example: Information gain

Let, $Values(Wind) = \{Weak, Strong\}$

$S = [9+, 5-]$

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

$S_{Strong} = [3+, 3-]$

Information gain of attribute *Wind*:

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

$= 0.94 - (8/14)* 0.811 - (6/14) *1.00$

$= 0.048$

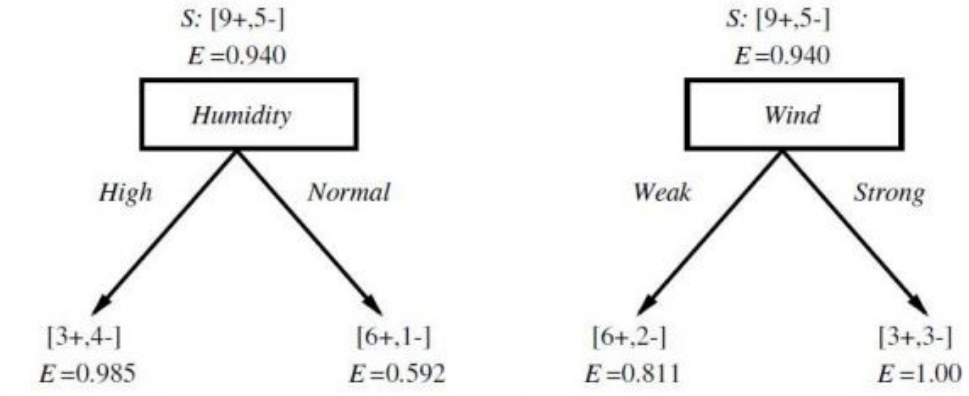
An Illustrative Example

- To illustrate the operation of ID3, consider the learning task represented by the training examples of below table.
- Here the target attribute ***PlayTennis***, which can have values ***yes*** or ***no*** for different days.
- Consider the first step through the algorithm, in which the topmost node of the decision tree is created.

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

- ID3 determines the information gain for each candidate attribute (i.e., Outlook, Temperature, Humidity, and Wind), then selects the one with highest information gain.

Which attribute is the best classifier?



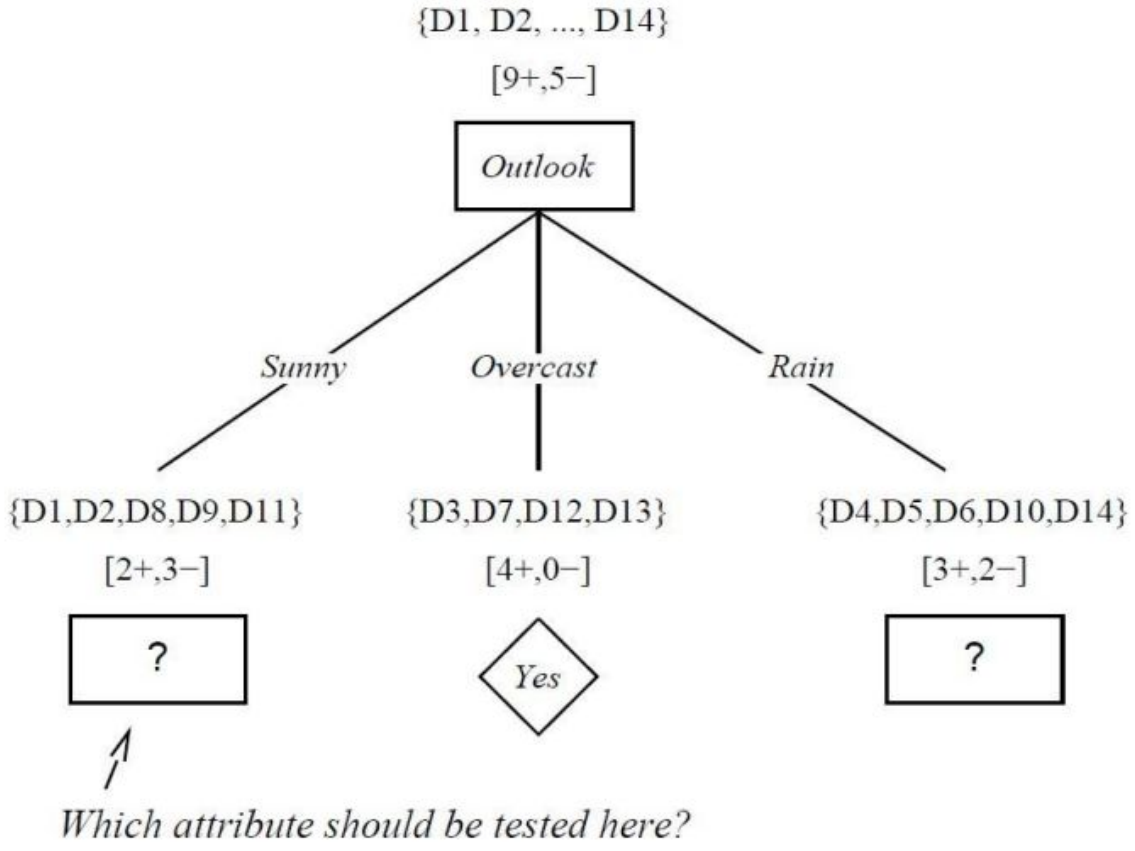
Gain (S, Humidity)
= .940 - (7/14).985 - (7/14).592
= .151

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

- The information gain values for all four attributes are

Gain(S, Outlook)	= 0.246
Gain(S, Humidity)	= 0.151
Gain(S, Wind)	= 0.048
Gain(S, Temperature)	= 0.029

- According to the information gain measure, the ***Outlook*** attribute provides the best prediction of the target attribute, ***PlayTennis***, over the training examples. Therefore, ***Outlook*** is selected as the decision attribute for the root node, and branches are created below the root for each of its possible values i.e., Sunny, Overcast, and Rain.



S_{sunny} = {D1,D2,D8,D9,D11}

$Gain\ (S_{sunny}, Humidity) = .970 - (3/5)\ 0.0 - (2/5)\ 0.0 = .970$

$Gain\ (S_{sunny}, Temperature) = .970 - (2/5)\ 0.0 - (2/5)\ 1.0 - (1/5)\ 0.0 = .570$

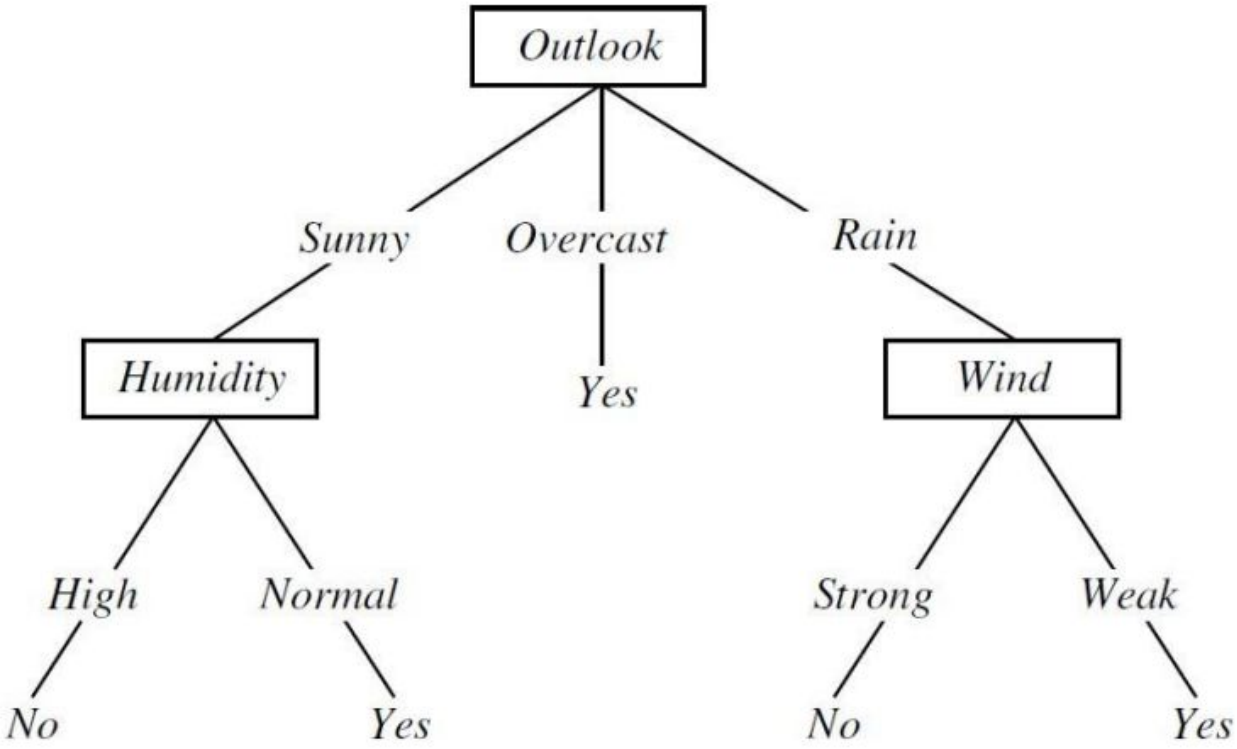
$Gain\ (S_{sunny}, Wind) = .970 - (2/5)\ 1.0 - (3/5)\ .918 = .019$

$SRain$ = { D4, D5, D6, D10, D14}

Gain (SRain , Humidity) = 0.970 – (2/5)1.0 – (3/5)0.917 = 0.019

Gain (SRain , Temperature) =0.970 – (0/5)0.0 – (3/5)0.918 – (2/5)1.0 = 0.019

Gain (SRain , Wind) =0.970 – (3/5)0.0 – (2/5)0.0 = 0.970



8

HYPOTHESIS SPACE SEARCH IN DECISION TREE LEARNING

- ID3 can be characterized as searching a space of hypotheses for one that fits the training examples.
- The hypothesis space searched by ID3 is the set of possible decision trees.
- ID3 performs a simple-to complex, hill-climbing search through this hypothesis space, beginning with the empty tree, then considering progressively more elaborate hypotheses in search of a decision tree that correctly classifies the training data

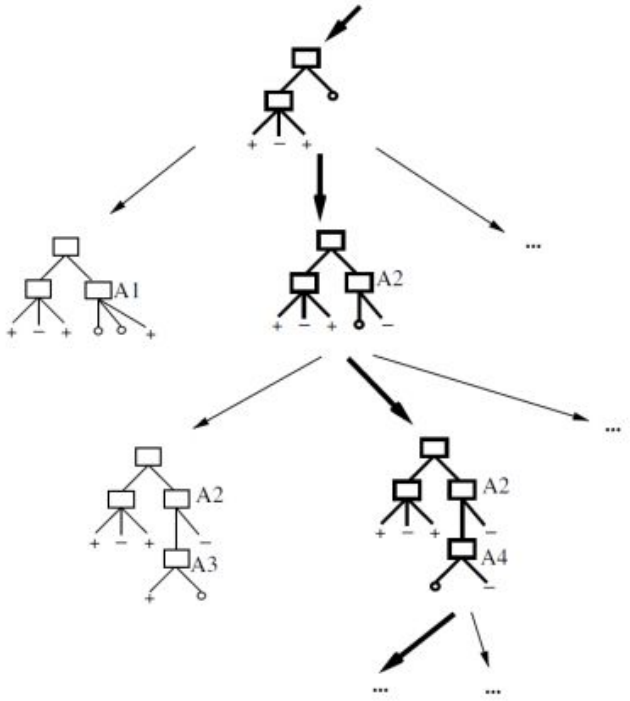


Figure: Hypothesis space search by ID3. ID3 searches through the space of possible decision trees from simplest to increasingly complex, guided by the information gain heuristic.

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

1. *ID3's hypothesis space of all decision trees is a complete space of finite discrete-valued functions, relative to the available attributes. Because every finite discrete-valued function can be represented by some decision tree*
   ID3 avoids one of the major risks of methods that search incomplete hypothesis spaces: that the hypothesis space might not contain the target function.

2. *ID3 maintains only a single current hypothesis as it searches through the space of decision trees.*
   For example, with the earlier version space candidate elimination method, which maintains the set of all hypotheses consistent with the available training examples.

   By determining only a single hypothesis, ID3 loses the capabilities that follow from explicitly representing all consistent hypotheses.
   For example, it does not have the ability to determine how many alternative decision trees are consistent with the available training data, or to pose new instance queries that optimally resolve among these competing hypotheses

3. *ID3 in its pure form performs no backtracking in its search. Once it selects an attribute to test at a particular level in the tree, it never backtracks to reconsider this choice.*
   In the case of ID3, a locally optimal solution corresponds to the decision tree it selects along the single search path it explores. However, this locally optimal solution may be less desirable than trees that would have been encountered along a different branch of the search.

4. *ID3 uses all training examples at each step in the search to make statistically based decisions regarding how to refine its current hypothesis.*
   One advantage of using statistical properties of all the examples is that the resulting search is much less sensitive to errors in individual training examples.
   ID3 can be easily extended to handle noisy training data by modifying its termination criterion to accept hypotheses that imperfectly fit the training data.

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

Given a collection of training examples, there are typically many decision trees consistent with these examples. Which of these decision trees does ID3 choose?

ID3 search strategy

- Selects in favour of shorter trees over longer ones
- Selects trees that place the attributes with highest information gain closest to the root.

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

- Consider an algorithm that begins with the empty tree and searches breadth first through progressively more complex trees.
- First considering all trees of depth 1, then all trees of depth 2, etc.
- Once it finds a decision tree consistent with the training data, it returns the smallest consistent tree at that search depth (e.g., the tree with the fewest nodes).
- Let us call this breadth-first search algorithm BFS-ID3.
- BFS-ID3 finds a shortest decision tree and thus exhibits the bias "shorter trees are preferred over longer trees.

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 can be viewed as an efficient approximation to BFS-ID3, using a greedy heuristic search to attempt to find the shortest tree without conducting the entire breadth-first search through the hypothesis space.
- Because ID3 uses the information gain heuristic and a hill climbing strategy, it exhibits a more complex bias than BFS-ID3.
- In particular, it does not always find the shortest consistent tree, and it is biased to favour trees that place attributes with high information gain closest to the root.

Restriction Biases and Preference Biases

Difference between the types of inductive bias exhibited by ID3 and by the CANDIDATE-ELIMINATION Algorithm.

ID3:

- ID3 searches a complete hypothesis space
- It searches incompletely through this space, from simple to complex hypotheses, until its termination condition is met
- Its inductive bias is solely a consequence of the ordering of hypotheses by its search strategy. Its hypothesis space introduces no additional bias

CANDIDATE-ELIMINATION Algorithm:

- The version space CANDIDATE-ELIMINATION Algorithm searches an incomplete hypothesis space
- It searches this space completely, finding every hypothesis consistent with the training data.
- Its inductive bias is solely a consequence of the expressive power of its hypothesis representation. Its search strategy introduces no additional bias

Preference bias – The inductive bias of ID3 is a preference for certain hypotheses over others (e.g., preference for shorter hypotheses over larger hypotheses), with no hard restriction on the hypotheses that can be eventually enumerated. This form of bias is called a preference bias or a search bias.

Restriction bias – The bias of the CANDIDATE ELIMINATION algorithm is in the form of a categorical restriction on the set of hypotheses considered. This form of bias is typically called a restriction bias or a language bias.

Which type of inductive bias is preferred in order to generalize beyond the training data, a preference bias or restriction bias?

- A preference bias is more desirable than a restriction bias, because it allows the learner to work within a complete hypothesis space that is assured to contain the unknown target function.
- In contrast, a restriction bias that strictly limits the set of potential hypotheses is generally less desirable, because it introduces the possibility of excluding the unknown target function altogether.

Why Prefer Short Hypotheses?

Occam's razor

- Occam's razor: is the problem-solving principle that the simplest solution tends to be the right one. When presented with competing hypotheses to solve a problem, one should select the solution with the fewest assumptions.

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

Argument in favour of Occam's razor:

- Fewer short hypotheses than long ones:
  - Short hypotheses fits the training data which are less likely to be coincident
  - Longer hypotheses fits the training data might be coincident.

- Many complex hypotheses that fit the current training data but fail to generalize correctly to subsequent data.

Argument opposed:

- There are few small trees, and our priori chance of finding one consistent with an arbitrary set of data is therefore small. The difficulty here is that there are very many small sets of hypotheses that one can define but understood by fewer learner.
- The size of a hypothesis is determined by the representation used internally by the learner. Occam's razor will produce two different hypotheses from the same training examples when it is applied by two learners, both justifying their contradictory conclusions by Occam's razor. On this basis we might be tempted to reject Occam's razor altogether.

ISSUES IN DECISION TREE LEARNING

Issues in learning decision trees include

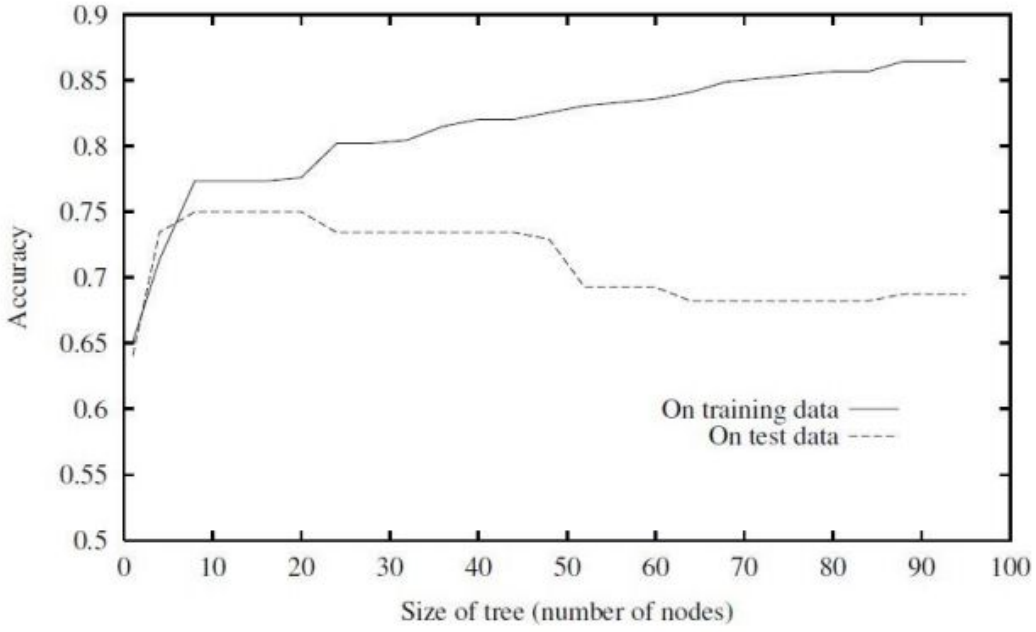
1. Avoiding Overfitting the Data
   Reduced error pruning
   Rule post-pruning
2. Incorporating Continuous-Valued Attributes
3. Alternative Measures for Selecting Attributes
4. Handling Training Examples with Missing Attribute Values
5. Handling Attributes with Differing Costs

1. Avoiding Overfitting the Data

- The ID3 algorithm grows each branch of the tree just deeply enough to perfectly classify the training examples but it can lead to difficulties when there is noise in the data, or when the number of training examples is too small to produce a representative sample of the true target function. This algorithm can produce trees that overfit the training examples.

- ***Definition - Overfit:*** 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.

The below figure illustrates the impact of overfitting in a typical application of decision tree learning.



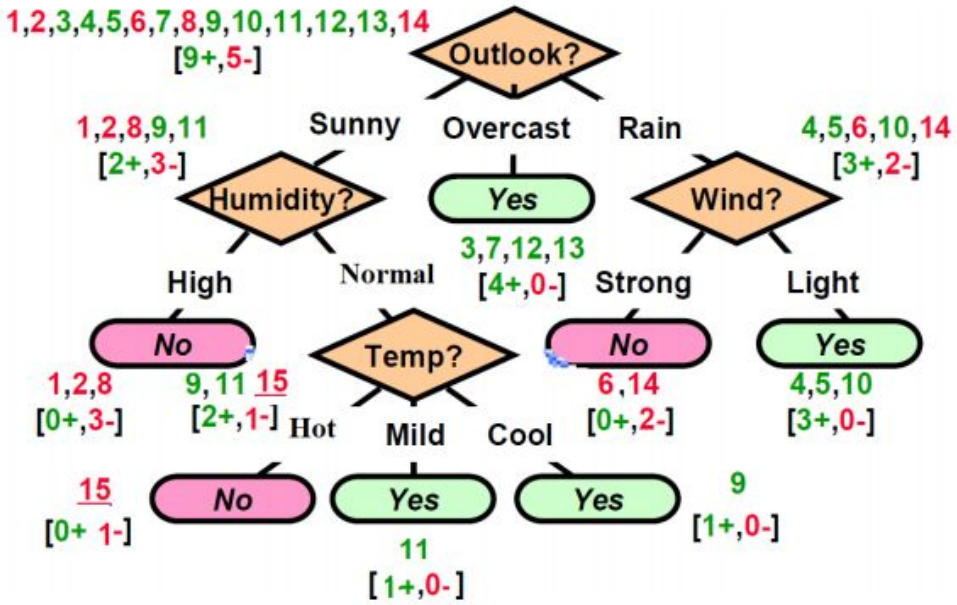
- *The horizontal axis* of this plot indicates the total number of nodes in the decision tree, as the tree is being constructed. The vertical axis indicates the accuracy of predictions made by the tree.
- *The solid line* shows the accuracy of the decision tree over the training examples. The broken line shows accuracy measured over an independent set of test example
- The accuracy of the tree over the training examples increases monotonically as the tree is grown. The accuracy measured over the independent test examples first increases, then decreases.

How can it be possible for tree h to fit the training examples better than h', but for it to perform more poorly over subsequent examples?

1. Overfitting can occur when the training examples contain random errors or noise
2. When small numbers of examples are associated with leaf nodes.

Noisy Training Example

- Example 15: <Sunny, Hot, Normal, Strong, ->
- Example is noisy because the correct label is +
- Previously constructed tree misclassifies it



Approaches to avoiding overfitting in decision tree learning

- Pre-pruning (avoidance): Stop growing the tree earlier, before it reaches the point where it perfectly classifies the training data
- Post-pruning (recovery): Allow the tree to overfit the data, and then post-prune the tree

Criterion used to determine the correct final tree size

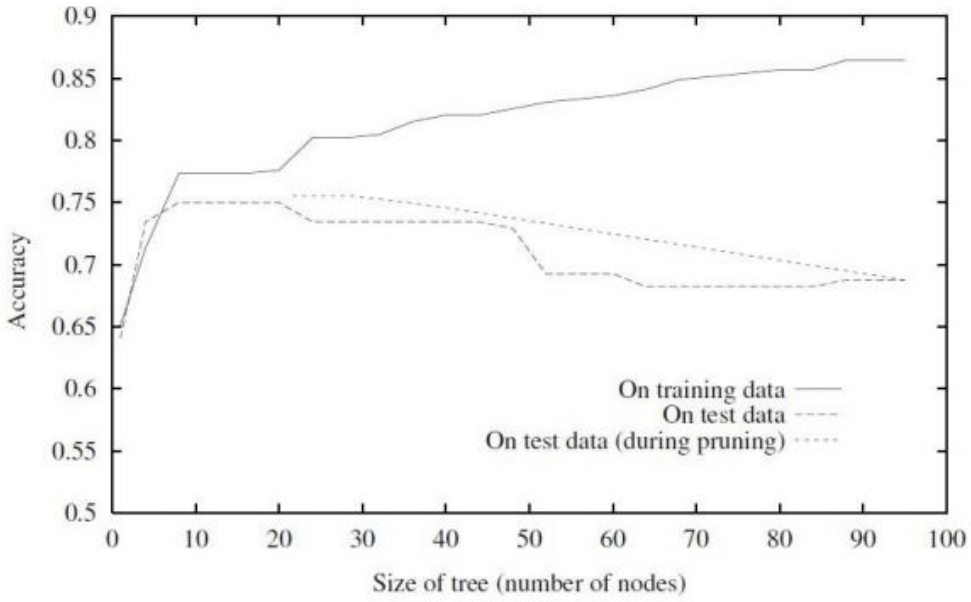
- Use a separate set of examples, distinct from the training examples, to evaluate the utility of post-pruning nodes from the tree
- Use all the available data for training, but apply a statistical test to estimate whether expanding (or pruning) a particular node is likely to produce an improvement beyond the training set
- Use measure of the complexity for encoding the training examples and the decision tree, halting growth of the tree when this encoding size is minimized. This approach is called the Minimum Description Length

MDL – Minimize : size(tree) + size (misclassifications(tree))

Reduced-Error Pruning

- Reduced-error pruning, is to consider each of the decision nodes in the tree to be candidates for pruning
- ***Pruning*** a decision node consists of removing the subtree rooted at that node, making it a leaf node, and assigning it the most common classification of the training examples affiliated with that node
- Nodes are removed only if the resulting pruned tree performs no worse than-the original over the validation set.
- Reduced error pruning has the effect that any leaf node added due to coincidental regularities in the training set is likely to be pruned because these same coincidences are unlikely to occur in the validation set

The impact of reduced-error pruning on the accuracy of the decision tree is illustrated in below figure



- The additional line in figure shows accuracy over the test examples as the tree is pruned. When pruning begins, the tree is at its maximum size and lowest accuracy over the test set. As pruning proceeds, the number of nodes is reduced and accuracy over the test set increases.
- The available data has been split into three subsets: the training examples, the validation examples used for pruning the tree, and a set of test examples used to provide an unbiased estimate of accuracy over future unseen examples. The plot shows accuracy over the training and test sets.

Pros and Cons

Pro: Produces smallest version of most accurate T (subtree of T)

Con: Uses less data to construct T

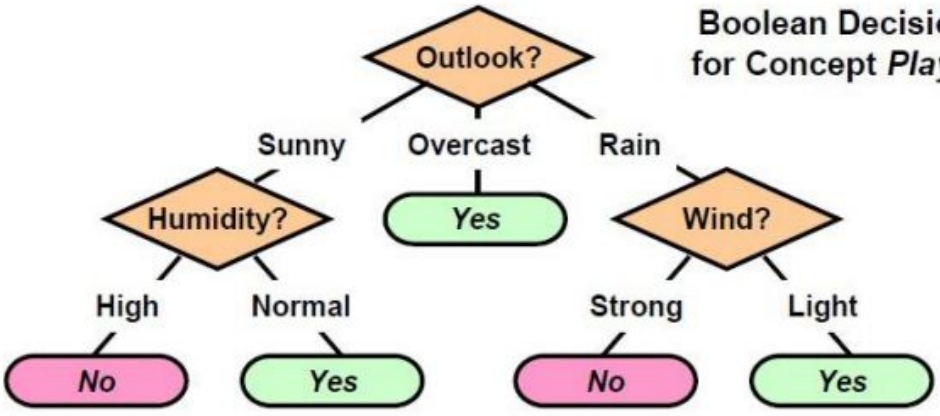
Can afford to hold out $D_{validation}$?. If not (data is too limited), may make error worse (insufficient D_{train})

Rule Post-Pruning

Rule post-pruning is successful method for finding high accuracy hypotheses

- Rule post-pruning involves the following steps:
- 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.
- 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.
- Prune (generalize) each rule by removing any preconditions that result in improving its estimated accuracy.
- Sort the pruned rules by their estimated accuracy, and consider them in this sequence when classifying subsequent instances.

Converting a Decision Tree into Rules



Example

- IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *High*) THEN *PlayTennis* = *No*
- IF (*Outlook* = *Sunny*) ∧ (*Humidity* = *Normal*) THEN *PlayTennis* = *Yes*
- …

17

For example, consider the decision tree. The leftmost path of the tree in below figure is translated into the rule.

IF (Outlook = Sunny) ^ (Humidity = High)
THEN PlayTennis = No

Given the above rule, rule post-pruning would consider removing the preconditions

(Outlook = Sunny) and (Humidity = High)

- It would select whichever of these pruning steps produced the greatest improvement in estimated rule accuracy, then consider pruning the second precondition as a further pruning step.
- No pruning step is performed if it reduces the estimated rule accuracy.

There are three main advantages by converting the decision tree to rules before pruning

1. Converting to rules allows distinguishing among the different contexts in which a decision node is used. Because each distinct path through the decision tree node produces a distinct rule, the pruning decision regarding that attribute test can be made differently for each path.
2. Converting to rules removes the distinction between attribute tests that occur near the root of the tree and those that occur near the leaves. Thus, it avoid messy bookkeeping issues such as how to reorganize the tree if the root node is pruned while retaining part of the subtree below this test.
3. Converting to rules improves readability. Rules are often easier for to understand.

2. Incorporating Continuous-Valued Attributes

Continuous-valued decision attributes can be incorporated into the learned tree.

There are two methods for Handling Continuous Attributes

1. Define new discrete valued attributes that partition the continuous attribute value into a discrete set of intervals.

E.g., {high ≡ Temp > 35° C, med ≡ 10° C < Temp ≤ 35° C, low ≡ Temp ≤ 10° C}

2. Using thresholds for splitting nodes

e.g., A ≤ a produces subsets A ≤ a and A > a

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

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

- Pick a threshold, c, that produces the greatest information gain
- In the current example, there are two candidate thresholds, corresponding to the values of Temperature at which the value of PlayTennis changes: (48 + 60)/2, and (80 + 90)/2.
- The information gain can then be computed for each of the candidate attributes, Temperature $_{>54}$, and Temperature $_{>85}$ and the best can be selected (Temperature $_{>54}$)

3. Alternative Measures for Selecting Attributes

- The problem is if attributes with many values, Gain will select it ?
- Example: consider the attribute Date, which has a very large number of possible values. (e.g., March 4, 1979).
- If this attribute is added to the PlayTennis data, it would have the highest information gain of any of the attributes. This is because Date alone perfectly predicts the target attribute over the training data. Thus, it would be selected as the decision attribute for the root node of the tree and lead to a tree of depth one, which perfectly classifies the training data.
- This decision tree with root node Date is not a useful predictor because it perfectly separates the training data, but poorly predict on subsequent examples.

One Approach: Use GainRatio instead of Gain

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

$$GainRatio(S,A) \equiv \frac{Gain(S,A)}{SplitInformation(S,A)}$$

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

Where, Si is subset of S, for which attribute A has value vi

4. Handling Training Examples with Missing Attribute Values

The data which is available may contain missing values for some attributes
Example: Medical diagnosis

- <Fever = true, Blood-Pressure = normal, …, Blood-Test = ?, …>
- Sometimes values truly unknown, sometimes low priority (or cost too high)

Strategies for dealing with the missing attribute value

- If node n test A, assign most common value of A among other training examples sorted to node n
- Assign most common value of A among other training examples with same target value
- Assign a probability pi to each of the possible values vi of A rather than simply assigning the most common value to A(x)

5. Handling Attributes with Differing Costs

- In some learning tasks the instance attributes may have associated costs.
- For example: In learning to classify medical diseases, the patients described in terms of attributes such as Temperature, BiopsyResult, Pulse, BloodTestResults, etc.
- These attributes vary significantly in their costs, both in terms of monetary cost and cost to patient comfort
- Decision trees use low-cost attributes where possible, depends only on high-cost attributes only when needed to produce reliable classifications

How to Learn A Consistent Tree with Low Expected Cost?

One approach is replace Gain by Cost-Normalized-Gain

Examples of normalization functions

- Tan and Schlimmer

$$\frac{Gain^2(S, A)}{Cost(A)}.$$

- Nunez

$$\frac{2^{Gain(S,A)} - 1}{(Cost(A) + 1)^w}$$

where $w \in [0, 1]$ determines importance of cost

Ensemble Learning

Ensemble learning usually produces more accurate solutions than a single model would.

Ensemble Learning is a technique that creates multiple models and then combines them to produce improved results. Ensemble learning usually produces more accurate solutions than a single model would. Ensemble learning methods is applied to regression as well as classification.

- Ensemble learning for regression creates multiple repressors i.e. multiple regression models such as linear, polynomial, etc.
- Ensemble learning for classification creates multiple classifiers i.e. multiple classification models such as logistic, decision tress, KNN, SVM, etc.

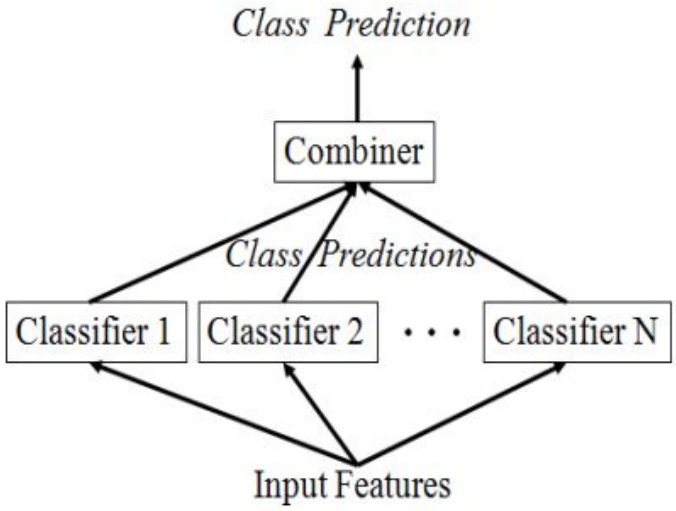


Figure 1: Ensemble learning view

Which components to combine?

• Different learning algorithms

• Same learning algorithm trained in different ways

• Same learning algorithm trained the same way

Note:

Multiples machine learning models were generated using ***same or different machine learning algorithm***. These are called "base models". The prediction performs on the basis of base models.

Techniques/Methods in ensemble learning:

- **Bagging**: Random Forest Trees,
- **Boosting**: Adaboost, Stacking.

1

Bagging:

Bootstrap aggregating, often abbreviated as bagging, and involves having each model in the ensemble vote with equal weight. In order to promote model variance, bagging trains each model in the ensemble using a randomly drawn subset of the training set. As an example, the random forest algorithm combines random decision trees with bagging to achieve very high classification accuracy.

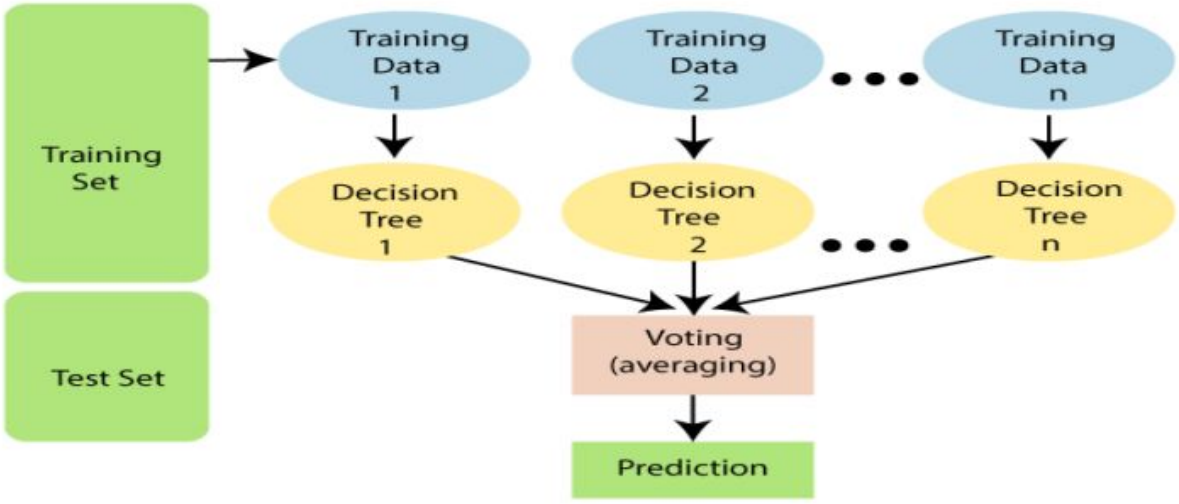
The simplest method of combining classifiers is known as **bagging**, which stands for **bootstrap aggregating**, the statistical description of the method. A bootstrap sample is a sample taken from the original dataset with replacement, so that we may get some data several times and others not at all. The bootstrap sample is the same size as the original, and lots and lots of these samples are taken.

Random Forest Algorithm

Random Forest is a popular machine learning algorithm that belongs to the supervised learning technique. It can be used for both Classification and Regression problems in ML. It is based on the concept of **ensemble learning,** which is a process of *combining multiple classifiers to solve a complex problem and to improve the performance of the model.*

As the name suggests, ***"Random Forest is a classifier that contains a number of decision trees on various subsets of the given dataset and takes the average to improve the predictive accuracy of that dataset."*** Instead of relying on one decision tree, the random forest takes the prediction from each tree and based on the majority votes of predictions, and it predicts the final output.

The greater number of trees in the forest leads to higher accuracy and prevents the problem of overfitting.



2

Assumptions for Random Forest

Since the random forest combines multiple trees to predict the class of the dataset, it is possible that some decision trees may predict the correct output, while others may not. But together, all the trees predict the correct output. Therefore, below are two assumptions for a better Random forest classifier:

- There should be some actual values in the feature variable of the dataset so that the classifier can predict accurate results rather than a guessed result.
- The predictions from each tree must have very low correlations.

Why use Random Forest?

- It takes less training time as compared to other algorithms.
- It predicts output with high accuracy, even for the large dataset it runs efficiently.
- It can also maintain accuracy when a large proportion of data is missing.

How does Random Forest algorithm work?

Random Forest works in two-phase first is to create the random forest by combining N decision tree, and second is to make predictions for each tree created in the first phase.

The Working process can be explained in the below steps and diagram:

Step-1: Select random K data points from the training set.

Step-2: Build the decision trees associated with the selected data points (Subsets).

Step-3: Choose the number N for decision trees that you want to build.

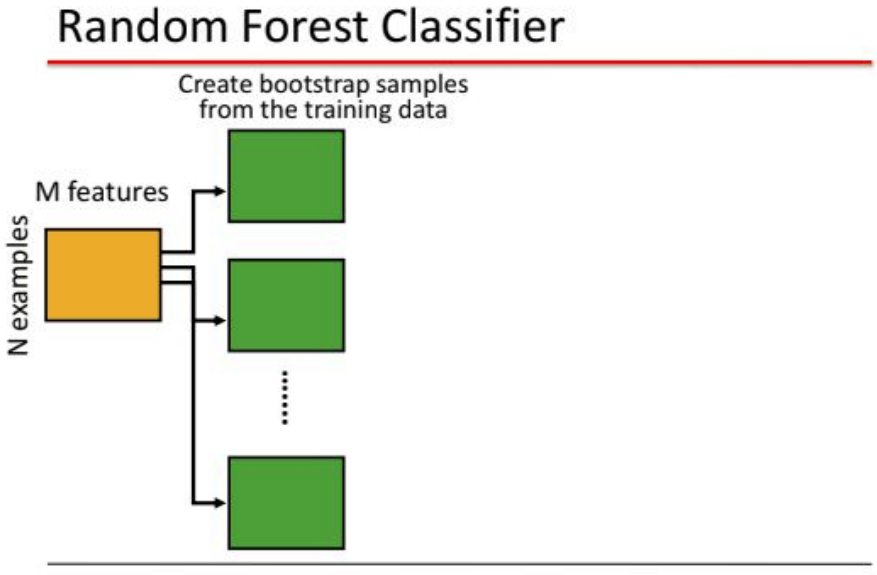
Step-4: Repeat Step 1 & 2.

Step-5: For new data points, find the predictions of each decision tree, and assign the new data points to the category that wins the majority votes

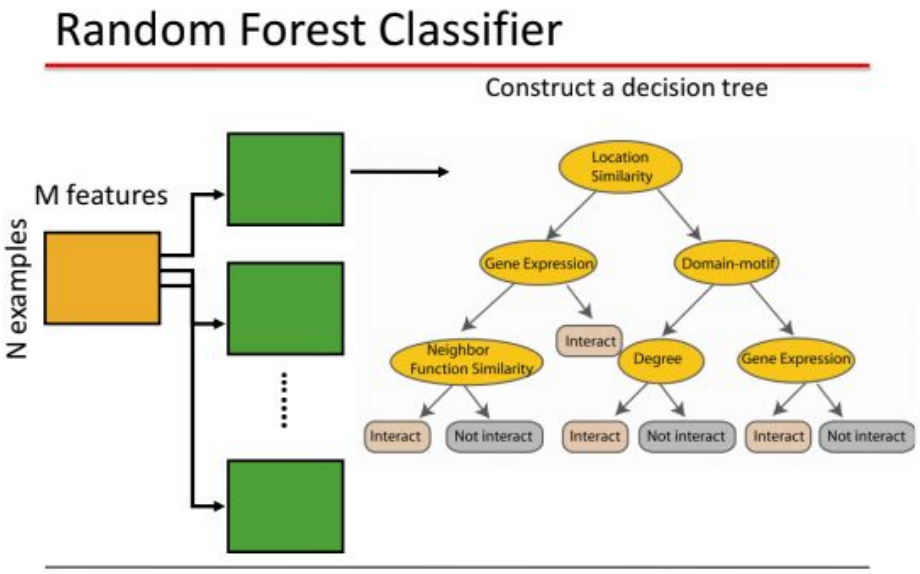
Example:

As the classifier of choice we adopt Random Forest Classifier, due to its robustness to heterogeneous and noisy feature.

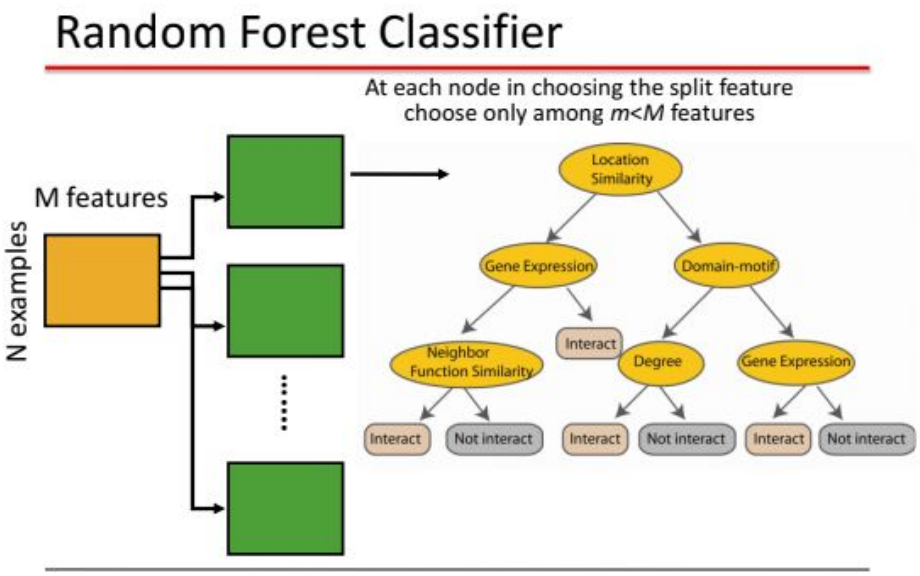
Random Forest is an ensemble classifier. Briefly, given N data and M features, bootsrap samples are created from the training data



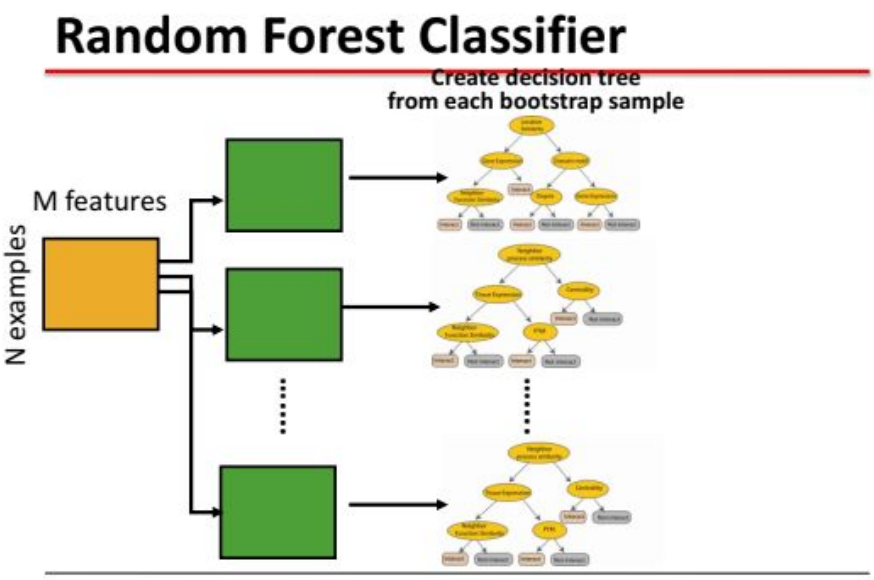
4

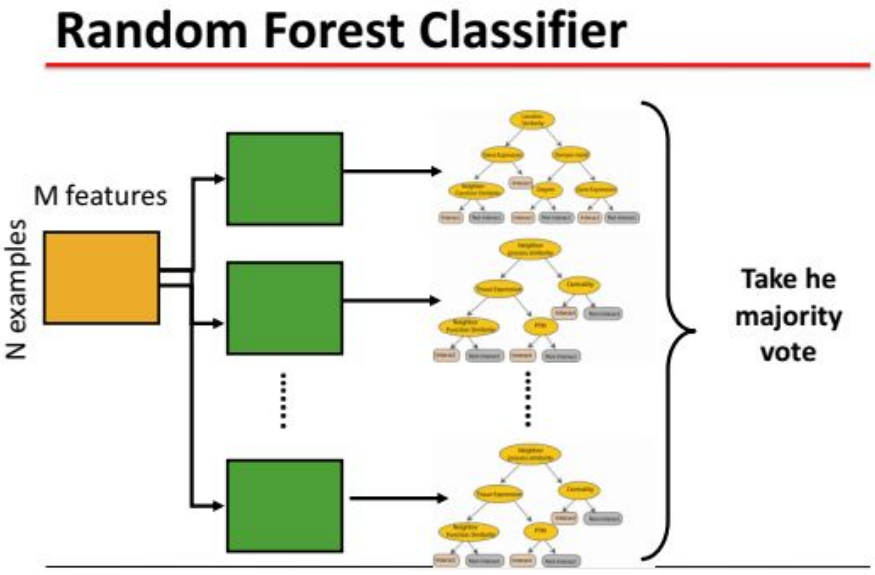


From eachd descision tree .. In spliting the nodes, the Gini Gain is employed



Different from the regular decision trees in random forest, when the splitting feature is chosen from only a subset of all features. The robustness of the classifier arises from bootstrapping of the training data and the random selection of features, the random choose of features.





Weaknesses

• A weakness of random forest algorithms is that when used for regression they cannot predict beyond the range in the training data, and that they may over-fit data sets that are particularly noisy.

• The sizes of the models created by random forests may be very large. It may take hundreds of Megabytes of memory and may be slow to evaluate.

• Random forest models are black boxes that are very hard to interpret.

https://www.analyticsvidhya.com/blog/2021/05/bagging-25-questions-to-test-your-skills-on-random-forest-algorithm/

6