• CHAPTER 3 BASICS OF LEARNING THEORY

LEARNING THEORY

- Learning is a process by which one can acquire knowledge and construct new ideas or concepts based on the experiences.
- The standard definition of learning proposed by Tom Mitchell is that a program can learn from E for the task T, and P improves with experience E.
- There are two kinds of problems – well-posed and ill-posed.
- Computers can solve only well- posed problems, as these have well-defined specifications and have the following components inherent to it.

LEARNING THEORY

1. Class of learning tasks (T)

2. A measure of performance (P)

3. A source of experience (E) Let x- input, χ-input space, Y –is the output space. Which is the set of all possible outputs, that is yes/no,

- Let D –dataset for n inputs.Consider, target function be: χ-> Y , that maps input to output.

- **Objective: To pick a function, g: χ-> Y to appropriate hypothesis f.**

INTRODUCTION TO LEARNING AND ITS TYPES

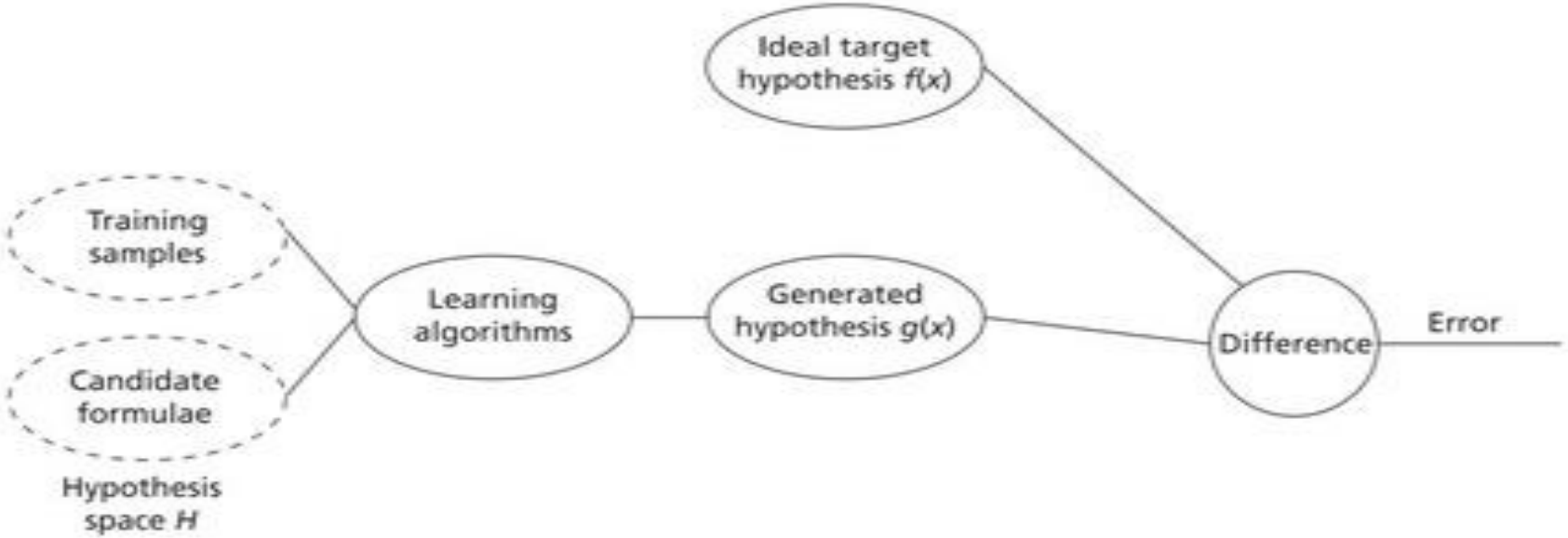


Fig: Learning Environment

Learning model= Hypothesis set + Learning algorithm

Let us assume a problem of predicting a label for a given input data. Let D be the input dataset with both positive and negative examples. Let y be the output with class 0 or 1. The simple learning model can be given as:

$$\sum_{i=1}^{D} x_i w_i > Threshold, \text{ belongs to class 1 and}$$

$$\sum_{i=1}^{D} x_i w_i < Threshold, \text{ belongs to another class}$$

This can be put into a single equation as follows:

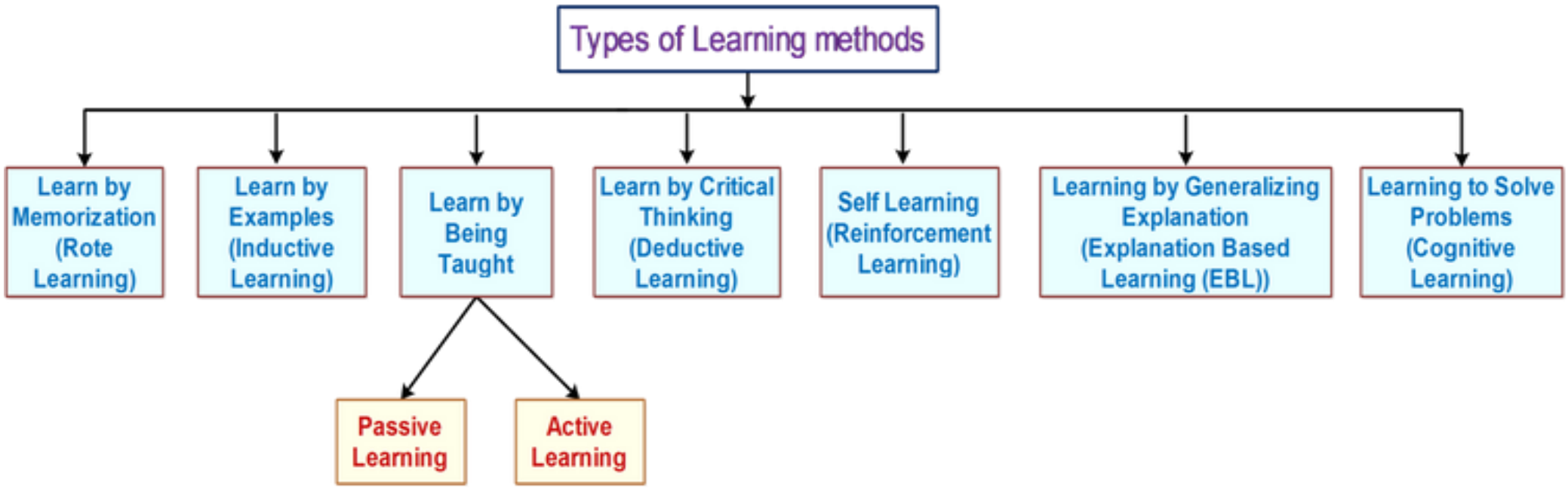
$$h(x) = sign\left(\left(\sum_{i=1}^{D} x_i w_i\right) + b\right)$$

where, $x_1, x_2, \cdots, x_d$ are the components of the input vector, $w_1, w_2, \ldots, w_d$ are the weights and +1 and −1 represent the class. This simple model is called perception model. One can simplify this by making $w_0 = b$ and fixing it as 1, then the model can further be simplified as:

$$h(x) = sign(w^T x).$$

Classical and Adaptive ML systems.

1. **Classic** machines examine data inputs according to a predetermined set of rules, finding patterns and relationships that can be used to generate predictions or choices.
2. Support vector machines, decision trees, and logistic regression are some of the most used classical machine learning techniques.
3. **Adaptive** or deep learning, is created to automatically learn from data inputs without being explicitly programmed.
4. By learning hierarchical representations of the input, these algorithms are able to handle more complex and unstructured data, such as photos, videos, and natural language.
5. Adaptive ML is the next generation of traditional ML – the new, the improved, the better.
6. Even though traditional ML witnessed significant progress.


Types of Learning methods
Learn by Memorization (Rote Learning)
Learn by Examples (Inductive Learning)
Learn by Being Taught
Learn by Critical Thinking (Deductive Learning)
Self Learning (Reinforcement Learning)
Learning by Generalizing Explanation (Explanation Based Learning (EBL))
Learning to Solve Problems (Cognitive Learning)
Passive Learning
Active Learning

3.2 INTRODUCTION TO COMPUTATION LEARNING THEORY

There are many questions that have been raised by mathematicians and logicians over the time taken by computers to learn. Some of the questions are as follows:

1. How can a learning system predict an unseen instance?
2. How do the hypothesis *h* is close to *f*, when hypothesis *f* itself is unknown?
3. How many samples are required?
4. Can we measure the performance of a learning system?
5. Is the solution obtained local or global?

These questions are the basis of a field called ‘Computational Learning Theory’ or in short (COLT).

3.3 DESIGN OF A LEARNING SYSTEM

A system that is built around a learning algorithm is called a learning system. The design of systems focuses on these steps:

1. Choosing a training experience
2. Choosing a target function
3. Representation of a target function
4. Function approximation

Determine the Target Function Representation

The representation of knowledge may be a table, collection of rules or a neural network. The linear combination of these factors can be coined as:

$$V = w_0 + w_1 x_1 + w_2 x_2 + w_3 x_3$$

where, x_1, x_2 and x_3 represent different board features and w_0, w_1, w_2 and w_3 represent weights.

Choosing an Approximation Algorithm for the Target Function

The focus is to choose weights and fit the given training samples effectively. The aim is to reduce the error given as:

$$E \equiv \sum_{Training} \left[V_{train}(b) - \hat{V}(b) \right]^2$$

Here, b is the sample and $\hat{V}(b)$ is the predicted hypothesis. The approximation is carried out as:

- Computing the error as the difference between trained and expected hypothesis. Let error be $error(b)$.
- Then, for every board feature x_i, the weights are updated as:

$$w_i = w_i + \mu \times error(b) \times x_i$$

Here, μ is the constant that moderates the size of the weight update.

Thus, the learning system has the following components:

- A Performance system to allow the game to play against itself.
- A Critic system to generate the samples.
- A Generalizer system to generate a hypothesis based on samples.
- An Experimenter system to generate a new system based on the currently learnt function. This is sent as input to the performance system.

INTRODUCTION TO CONCEPT LEARNING

Concept learning is a learning strategy of acquiring abstract knowledge or inferring a general concept or deriving a category from the given training samples. It is a process of abstraction and generalization from the data.

Concept learning helps to classify an object that has a set of common, relevant features. Thus, it helps a learner compare and contrast categories based on the similarity and association of positive and negative instances in the training data to classify an object. The learner tries to simplify by observing the common features from the training samples and then apply this simplified model to the future samples. This task is also known as learning from experience.

Table 3.1: Sample Training Instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

3.4 INTRODUCTION TO CONCEPT LEARNING

Concept learning requires three things:

1. Input – Training dataset which is a set of training instances, each labeled with the name of a concept or category to which it belongs. Use this past experience to train and build the model.
2. Output – Target concept or Target function f. It is a mapping function $f(x)$ from input x to output y. It is to determine the specific features or common features to identify an object. In other words, it is to find the hypothesis to determine the target concept. For e.g., the specific set of features to identify an elephant from all animals.
3. Test – New instances to test the learned model.

Table 3.1: Sample Training Instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

Here, in this set of training instances, the independent attributes considered are 'Horns', 'Tail', 'Tusks', 'Paws', 'Fur', 'Color', 'Hooves' and 'Size'. The dependent attribute is 'Elephant'. The target concept is to identify the animal to be an Elephant.

Let us now take this example and understand further the concept of hypothesis.

Target Concept: Predict the type of animal - For example –'Elephant'.

3.4.1 Representation of a Hypothesis

A *hypothesis* '*h*' approximates a target function '*f*' to represent the relationship between the independent attributes and the dependent attribute of the training instances. The hypothesis is the predicted approximate model that best maps the inputs to outputs. Each hypothesis is represented as a conjunction of attribute conditions in the antecedent part.

3.4.1 Representation of a Hypothesis

A *hypothesis* '*h*' approximates a target function '*f*' to represent the relationship between the independent attributes and the dependent attribute of the training instances. The hypothesis is the predicted approximate model that best maps the inputs to outputs. Each hypothesis is represented as a conjunction of attribute conditions in the antecedent part.

For example, (Tail = Short) ∧ (Color = Black)....

The set of hypothesis in the search space is called as hypotheses. Hypotheses are the plural form of hypothesis. Generally '*H*' is used to represent the hypotheses and '*h*' is used to represent a candidate hypothesis.

Each attribute condition is the constraint on the attribute which is represented as attribute-value pair. In the antecedent of an attribute condition of a hypothesis, each attribute can take value as either '?' or 'φ' or can hold a single value.

- "?" denotes that the attribute can take any value [e.g., Color = ?]
- "φ" denotes that the attribute cannot take any value, i.e., it represents a null value [e.g., Horns = φ]
- Single value denotes a specific single value from acceptable values of the attribute, i.e., the attribute 'Tail' can take a value as 'short' [e.g., Tail = Short]

For example, a hypothesis '*h*' will look like,

	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size
h =	<No	?	Yes	?	?	Black	No	Medium>

Given a test instance x, we say $h(x) = 1$, if the test instance x satisfies this hypothesis h.

Table 3.1: Sample Training Instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size
h =	<No	?	Yes	?	?	Black	No	Medium>

(or)

	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size
h =	<No	?	Yes	?	?	Black	No	Big>

It is represented as:

<?, ?, ?, ?, ?, ?, ?, ?>. This hypothesis indicates that any animal can be an elephant.

The most specific hypothesis will not allow any value for each of the attribute <φ, φ, φ, φ, φ, φ, φ, φ>. This hypothesis indicates that no animal can be an elephant.

Example 3.1: Explain Concept Learning Task of an Elephant from the dataset given in Table 3.1.

Given,

Input: 5 instances each with 8 attributes

Target concept/function 'c': Elephant $\rightarrow$ {Yes, No}

Hypotheses H: Set of hypothesis each with conjunctions of literals as propositions [i.e., each literal is represented as an attribute-value pair]

Solution: The hypothesis 'h' for the concept learning task of an Elephant is given as:

$h =$ <No Short Yes ? ? Black No ?>

This hypothesis h is expressed in propositional logic form as below:

(Horns = No) $\wedge$ (Tail = Short) $\wedge$ (Tusks = Yes) $\wedge$ (Paws = ?) $\wedge$ (Fur = ?) $\wedge$ (Color = Black) $\wedge$ (Hooves = No) $\wedge$ (Size = ?)

Output: Learn the hypothesis 'h' to predict an 'Elephant' such that for a given test instance x,

$$h(x) = c(x)$$

This hypothesis produced is also called as concept description which is a model that can be used to classify subsequent instances.

3.4.2 Hypothesis Space

Hypothesis space is the set of all possible hypotheses that approximates the target function *f*. In other words, the set of all possible approximations of the target function can be defined as hypothesis space. From this set of hypotheses in the hypothesis space, a machine learning algorithm

The subset of hypothesis space that is consistent with all-observed training instances is called as **Version Space.** Version space represents the only hypotheses that are used for the classification.

For example, each of the attribute given in the Table 3.1 has the following possible set of values.

Horns - Yes, No
Tail - Long, Short
Tusks - Yes, No
Paws - Yes, No
Fur - Yes, No
Color - Brown, Black, White
Hooves - Yes, No
Size - Medium, Big

Considering these values for each of the attribute, there are $(2 \times 2 \times 2 \times 2 \times 2 \times 3 \times 2 \times 2) = 384$ distinct instances covering all the 5 instances in the training dataset.

So, we can generate $(4 \times 4 \times 4 \times 4 \times 4 \times 5 \times 4 \times 4) = 81{,}920$ distinct hypotheses when including two more values [?, φ] for each of the attribute. However, any hypothesis containing one or more φ symbols represents the empty set of instances; that is, it classifies every instance as negative instance. Therefore, there will be $(3 \times 3 \times 3 \times 3 \times 3 \times 4 \times 3 \times 3 + 1) = 8{,}749$ distinct hypotheses by including only '?' for each of the attribute and one hypothesis representing the empty set of instances. Thus, the hypothesis space is much larger and hence we need efficient learning algorithms to search for the best hypothesis from the set of hypotheses.

3.4.4 Generalization and Specialization

In order to understand about how we construct this concept hierarchy, let us apply this general principle of generalization/specialization relation. By generalization of the most specific hypothesis and by specialization of the most general hypothesis, the hypothesis space can be searched for an approximate hypothesis that matches all positive instances but does not match any negative instance.

Searching the Hypothesis Space

There are two ways of learning the hypothesis, consistent with all training instances from the large hypothesis space.

1. Specialization – General to Specific learning
2. Generalization – Specific to General learning

Table 3.1: Sample Training Instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

Generalization – Specific to General Learning This learning methodology will search through the hypothesis space for an approximate hypothesis by generalizing the most specific hypothesis.

Example 3.2: Consider the training instances shown in Table 3.1 and illustrate Specific to General Learning.

Solution: We will start from all false or the most specific hypothesis to determine the most restrictive specialization. Consider only the positive instances and generalize the most specific hypothesis. Ignore the negative instances.

This learning is illustrated as follows:

The most specific hypothesis is taken now, which will not classify any instance to true.

$h = \langle \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \rangle$

Read the first instance $I1$, to generalize the hypothesis h so that this positive instance can be classified by the hypothesis $h1$.

$I1$: No Short Yes No No Black No Big **Yes (Positive instance)**

$h1 =$ <No Short Yes No No Black No Big>

Table 3.1: Sample training instances

S.No.	Horns	Tail	Tusks	Paws	Fur	Color	Hooves	Size	Elephant
1.	No	Short	Yes	No	No	Black	No	Big	Yes
2.	Yes	Short	No	No	No	Brown	Yes	Medium	No
3.	No	Short	Yes	No	No	Black	No	Medium	Yes
4.	No	Long	No	Yes	Yes	White	No	Medium	No
5.	No	Short	Yes	Yes	Yes	Black	No	Big	Yes

When reading the second instance $I2$, it is a negative instance, so ignore it.

$I2$: Yes Short No No No Brown Yes Medium **No (Negative instance)**

$h2$ = <No Short Yes No No Black No Big>

Similarly, when reading the third instance $I3$, it is a positive instance so generalize $h2$ to $h3$ to accommodate it. The resulting $h3$ is generalized.

$I3$: No Short Yes No No Black No Medium **Yes (Positive instance)**

$h3$ = <No Short Yes No No Black No ?>

Ignore $I4$ since it is a negative instance.

$I4$: No Long No Yes Yes White No Medium **No (Negative instance)**

$h4$ = <No Short Yes No No Black No ?>

When reading the fifth instance $I5$, $h4$ is further generalized to $h5$.

$I5$: No Short Yes Yes Yes Black No Big **Yes (Positive instance)**

$h5$ = <No Short Yes ? ? Black No ?>

Now, after observing all the positive instances, an approximate hypothesis $h5$ is generated which can now classify any subsequent positive instance to true.

Example 3.3: Illustrate learning by Specialization – General to Specific Learning for the data instances shown in Table 3.1.

Solution: Start from the most general hypothesis which will make true all positive and negative instances.

Initially,

$h = <?$? ? ? ? ? ? ?>

h is more general to classify all instances to true.

$I1$:	No	Short	Yes	No	No	Black	No	Big	**Yes (Positive instance)**
$h1 = <?$	?	?	?	?	?	?	?	?>	
$I2$:	Yes	Short	No	No	No	Brown	Yes	Medium	**No (Negative instance)**
$h2 = <$No	?	?	?	?	?	?	?	?>	
	<?	?	Yes	?	?	?	?	?>	
	<?	?	?	?	?	Black	?	?>	
	<?	?	?	?	?	?	No	?>	
	<?	?	?	?	?	?	?	Big>	

$h2$ imposes constraints so that it will not classify a negative instance to true.

$I3$:	No	Short	Yes	No	No	Black	No	Medium	**Yes (Positive instance)**
$h3 = <$No	?	?	?	?	?	?	?	?>	
	<?	?	Yes	?	?	?	?	?>	
	<?	?	?	?	?	Black	?	?>	
	<?	?	?	?	?	?	No	?>	
	<?	?	?	?	?	?	?	Big>	
$I4$:	No	Long	No	Yes	Yes	White	No	Medium	**No (Negative instance)**
$h4 = <?$	?	Yes	?	?	?	?	?	?>	
	<?	?	?	?	?	Black	?	?>	
	<?	?	?	?	?	?	?	Big>	

Remove any hypothesis inconsistent with this negative instance.

$I5$:	No	Short	Yes	Yes	Yes	Black	No	Big	**Yes (Positive instance)**
$h5 = <?$	?	Yes	?	?	?	?	?	?>	
	<?	?	?	?	?	Black	?	?>	
	<?	?	?	?	?	?	?	Big>	

Thus, $h5$ is the hypothesis space generated which will classify the positive instances to true and negative instances to false.

Algorithm 3.1: Find-S

Input: Positive instances in the Training dataset

Output: Hypothesis *'h'*

1. Initialize *'h'* to the most specific hypothesis.

   $h = <\varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \ldots..>$

2. Generalize the initial hypothesis for the first positive instance [Since *'h'* is more specific].
3. For each subsequent instances:

   If it is a positive instance,

   Check for each attribute value in the instance with the hypothesis *'h'*.

   If the attribute value is the same as the hypothesis value, then do nothing,

   Else if the attribute value is different than the hypothesis value, change it to '?' in *'h'*.

   Else if it is a negative instance,

   Ignore it.

Table 3.2: Training Dataset

CGPA	Interactiveness	Practical Knowledge	Communication Skills	Logical Thinking	Interest	Job Offer
≥9	Yes	Excellent	Good	Fast	Yes	Yes
≥9	Yes	Good	Good	Fast	Yes	Yes
≥8	No	Good	Good	Fast	No	No
≥9	Yes	Good	Good	Slow	No	Yes

Solution:

Step 1: Initialize 'h' to the most specific hypothesis. There are 6 attributes, so for each attribute, we initially fill 'φ' in the initial hypothesis 'h'.

$h = <\varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi \quad \varphi>$

Step 2: Generalize the initial hypothesis for the first positive instance. $I1$ is a positive instance, so generalize the most specific hypothesis 'h' to include this positive instance. Hence,

$I1$: ≥9 Yes Excellent Good Fast Yes **Positive instance**

h = <≥9 Yes Excellent Good Fast Yes>

Step 3: Scan the next instance $I2$, since $I2$ is a positive instance. Generalize 'h' to include positive instance $I2$. For each of the non-matching attribute value in 'h' put a '?' to include this positive instance. The third attribute value is mismatching in 'h' with $I2$, so put a '?'.

$I2$: ≥9 Yes Good Good Fast Yes **Positive instance**

h = <≥9 Yes ? Good Fast Yes>

Now,scan $I3$. Since it is a negative instance, ignore it. Hence, the hypothesis remains the same without any change after scanning $I3$.

I3: ≥8 No Good Good Fast No **Negative instance**

h = <≥9 Yes ? Good Fast Yes>

Now scan $I4$. Since it is a positive instance, check for mismatch in the hypothesis 'h' with $I4$. The 5th and 6th attribute value are mismatching, so add '?' to those attributes in 'h'.

I4: ≥9 Yes Good Good Slow No **Positive instance**

h = <≥9 Yes ? Good ? ?>

Now, the final hypothesis generated with Find-S algorithm is:

h = <≥9 Yes ? Good ? ?>

It includes all positive instances and obviously ignores any negative instance.

Limitations of Find-S Algorithm

1. Find-S algorithm tries to find a hypothesis that is consistent with positive instances, ignoring all negative instances. As long as the training dataset is consistent, the hypothesis found by this algorithm may be consistent.
2. The algorithm finds only one unique hypothesis, wherein there may be many other hypotheses that are consistent with the training dataset.
3. Many times, the training dataset may contain some errors; hence such inconsistent data instances can mislead this algorithm in determining the consistent hypothesis since it ignores negative instances.

List-Then-Eliminate

Algorithm 3.2: List-Then-Eliminate

Input: Version Space – a list of all hypotheses

Output: Set of consistent hypotheses

1. Initialize the version space with a list of hypotheses.
2. For each training instance,
   - remove from version space any hypothesis that is inconsistent.

Version Spaces and the Candidate Elimination Algorithm

Version space learning is to generate all consistent hypotheses around. This algorithm computes the version space by the combination of the two cases namely,

- Specific to General learning – Generalize *S* to include the positive example
- General to Specific learning – Specialize *G* to exclude the negative example

Candidate Elimination Algorithm

Algorithm 3.3: Candidate Elimination

Input: Set of instances in the Training dataset

Output: Hypothesis *G* and *S*

1. Initialize *G*, to the maximally general hypotheses.
2. Initialize *S*, to the maximally specific hypotheses.
   - Generalize the initial hypothesis for the first positive instance.
3. For each subsequent new training instance,
   - If the instance is **positive**,
     - Generalize *S* to include the positive instance,
       - Check the attribute value of the positive instance and *S*,
         - If the attribute value of positive instance and *S* are different, fill that field value with '?'.
         - If the attribute value of positive instance and *S* are same, then do no change.
     - Prune *G* to exclude all inconsistent hypotheses in *G* with the positive instance.
   - If the instance is **negative,**
     - Specialize *G* to exclude the negative instance,
       - Add to *G* all minimal specializations to exclude the negative example and be consistent with *S*.
         - If the attribute value of *S* and the negative instance are different, then fill that attribute value with *S* value.
         - If the attribute value of *S* and negative instance are same, no need to update '*G*' and fill that attribute value with '?'.
     - Remove from *S* all inconsistent hypotheses with the negative instance.

Example 3.4: Consider the same set of instances from the training dataset shown in Table 3.3 and generate version space as consistent hypothesis.

Solution:

Step 1: Initialize '*G*' boundary to the maximally general hypotheses,

$G = <?$? ? ? ? ?>

Step 2: Initialize '*S*' boundary to the maximally specific hypothesis. There are 6 attributes, so for each attribute, we initially fill 'φ' in the hypothesis '*S*'.

$S = <\varphi$ φ φ φ φ $\varphi>$

Generalize the initial hypothesis for the first positive instance. *I*1 is a positive instance; so generalize the most specific hypothesis '*S*' to include this positive instance. Hence,

*I*1:	≥9	Yes	Excellent	Good	Fast	Yes	**Positive instance**
*S*1 = <	≥9	Yes	Excellent	Good	Fast	Yes>	
*G*1 = <?		?	?	?	?	?>	

Step 3:

Iteration 1

Scan the next instance *I*2. Since *I*2 is a positive instance, generalize '*S*1' to include positive instance *I*2. For each of the non-matching attribute value in '*S*1', put a '?' to include this positive instance. The third attribute value is mismatching in '*S*1' with *I*2, so put a '?'.

*I*2:	≥9	Yes	Good	Good	Fast	Yes	**Positive instance**
*S*2 = <	≥9	Yes	?	Good	Fast	Yes>	

Prune *G*1 to exclude all inconsistent hypotheses with the positive instance. Since *G*1 is consistent with this positive instance, there is no change. The resulting *G*2 is,

*G*2 = <? ? ? ? ? ?>

Iteration 2

Now Scan *I*3,

*I*3:	≥8	No	Good	Good	Fast	No **Negative instance**

Since it is a negative instance, specialize *G*2 to exclude the negative example but stay consistent with *S*2. Generate hypothesis for each of the non-matching attribute value in *S*2 and fill with the attribute value of *S*2. In those generated hypotheses, for all matching attribute values, put a '?'. The first, second and 6th attribute values do not match, hence '3' hypotheses are generated in *G*3.

There is no inconsistent hypothesis in *S*2 with the negative instance, hence *S*3 remains the same.

*G*3 = < ≥9	?	?	?	?	?>
<?	Yes	?	?	?	?>
<?	?	?	?	?	Yes>
*S*3 = < ≥9	Yes	?	Good	Fast	Yes>

Iteration 3

Now Scan *I*4. Since it is a positive instance, check for mismatch in the hypothesis '*S*3' with *I*4. The 5th and 6th attribute value are mismatching, so add '?' to those attributes in '*S*4'.

*I*4:	≥9	Yes	Good	Good	Slow	No **Positive instance**
*S*4 = < ≥9	Yes	?	Good	?	?>	

Prune *G*3 to exclude all inconsistent hypotheses with the positive instance *I*4.

*G*3 = < ≥9	?	?	?	?	?>	
<?	Yes	?	?	?	?>	
<?	?	?	?	?	Yes>	**Inconsistent**

Since the third hypothesis in *G*3 is inconsistent with this positive instance, remove the third one. The resulting *G*4 is,

*G*4 = < ≥9	?	?	?	?	?>
<?	Yes	?	?	?	?>

Using the two boundary sets, *S*4 and *G*4, the version space is converged to contain the set of consistent hypotheses.

The final version space is,

< ≥9	Yes	?	?	?	?>
< ≥9	?	?	Good	?	?>
<?	Yes	?	Good	?	?>

Thus, the algorithm finds the version space to contain only those hypotheses that are most general and most specific.

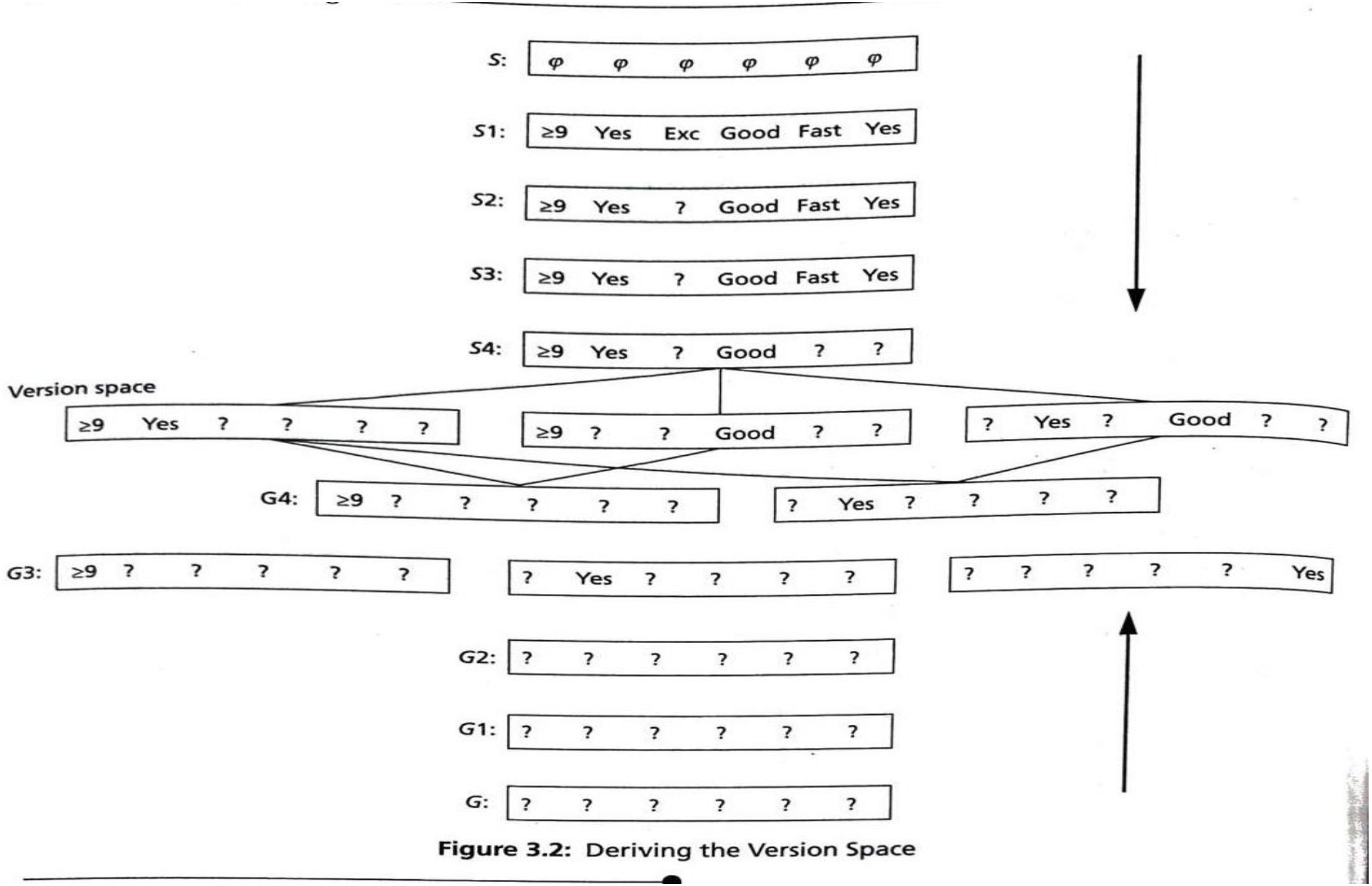


Figure 3.2: Deriving the Version Space

```
#LAB3 CANDIDATE ELIMINATION
import csv
with open("trainingexamples.csv") as f:
    csv_file = csv.reader(f)
    data = list(csv_file)
    specific = data[0][:-1]
    general = [['?' for i in range(len(specific))] for j in range(len(specific))]
    step=1 #for printing purpose
    for i in data:
        if i[-1] == "Y":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    specific[j] = "?"
                    general[j][j] = "?"
        elif i[-1] == "N":
            for j in range(len(specific)):
                if i[j] != specific[j]:
                    general[j][j] = specific[j]
                else:
                    general[j][j] = "?"
        print("\nStep {} of candidate elimination algo".format(step))
        step+=1
        print(specific)
        print(general)
```

```
gh = [] # gh = general Hypothesis
    for i in general:
        for j in i:
            if j != '?':
                gh.append(i)
                break
    print("\nFinal Specific hypothesis:\n",
specific)
    print("\nFinal General hypothesis:\n", gh)
```

```
Step 1 of candidate elimination algo
['sunny', 'warm', 'normal', 'strong', 'warm', 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]
Step 2 of candidate elimination algo
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?']]
Step 3 of candidate elimination algo
['sunny', 'warm', '?', 'strong', 'warm', 'same']
[['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?', '?', '?', '?'], ['?', '?',
'?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'],
['?', '?', '?', '?', '?', 'same']]
Step 4 of candidate elimination algo
['sunny', 'warm', '?', 'strong', '?', '?'] [['sunny', '?', '?', '?', '?', '?'], ['?',
'warm', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?',
'?', '?'], ['?', '?', '?', '?', '?', '?'], ['?', '?', '?', '?', '?', '?']]
Final Specific hypothesis:
['sunny', 'warm', '?', 'strong', '?', '?']
Final General hypothesis: [['sunny', '?', '?', '?', '?', '?'], ['?', 'warm', '?',
'?', '?', '?']]
```

CHAPTER 4 SIMILARITY BASED LEARNING

Similarity-based classifiers use similarity measures to locate the nearest neighbors and classify a test instance which works in contrast with other learning mechanisms such as decision trees or neural networks. Similarity-based learning is also called as Instance-based learning/Just-in time learning since it does not build an abstract model of the training instances and performs lazy learning when classifying a new instance. This learning mechanism simply stores all data and uses it only when it needs to classify an unseen instance. The advantage of using this learning is that processing occurs only when a request to classify a new instance is given. This methodology is particularly useful when the whole dataset is not available in the beginning but collected in an incremental manner.

Table 4.1: Differences between Instance-based Learning and Model-based Learning

Instance-based Learning	Model-based Learning
Lazy Learners	Eager Learners
Processing of training instances is done only during testing phase	Processing of training instances is done during training phase

Instance-based Learning	Model-based Learning
No model is built with the training instances before it receives a test instance	Generalizes a model with the training instances before it receives a test instance
Predicts the class of the test instance directly from the training data	Predicts the class of the test instance from the model built
Slow in testing phase	Fast in testing phase
Learns by making many local approximations	Learns by creating global approximation

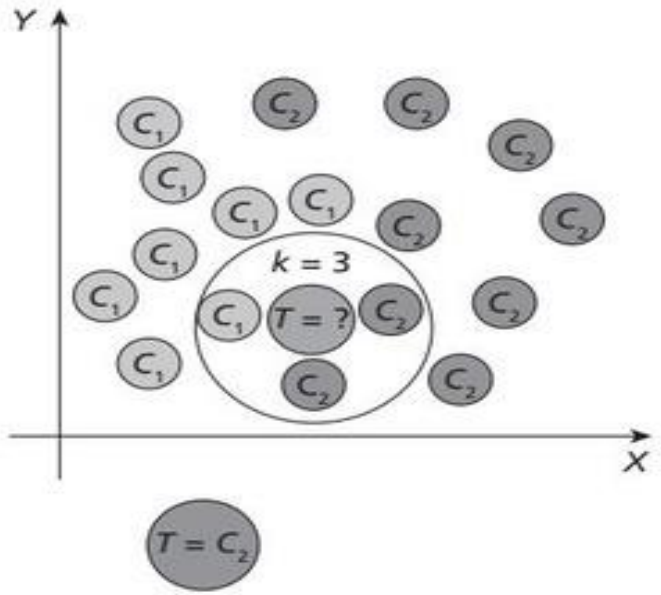
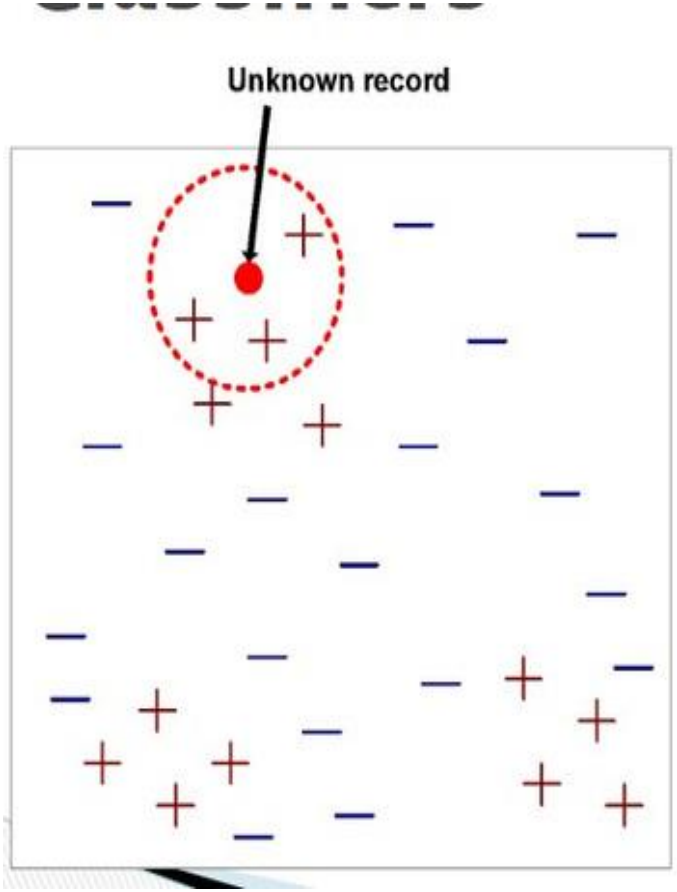


Figure 4.1: Visual Representation of *k*-Nearest Neighbor Learning



- Requires three things
  - The set of stored records
  - Distance Metric to compute distance between records
  - The value of *k*, the number of nearest neighbors to retrieve

- To classify an unknown record:
  1. Compute distance to other training records
  2. Identify *k* nearest neighbors
  3. Use class labels of nearest neighbors to determine the class label of unknown record (e.g., by taking majority vote)

Algorithm 4.1: k-NN

Algorithm 4.1: *k*-NN

Inputs: Training dataset T, distance metric d, Test instance t, the number of nearest neighbors k

Output: Predicted class or category

Prediction: For test instance t,

1. For each instance i in T, compute the distance between the test instance t and every other instance i in the training dataset using a distance metric (Euclidean distance).
   [Continuous attributes - Euclidean distance between two points in the plane with coordinates (x_1, y_1) and (x_2, y_2) is given as dist $((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$]
   [Categorical attributes (Binary) - Hamming Distance: If the value of the two instances is same, the distance d will be equal to 0 otherwise $d = 1$.]
2. Sort the distances in an ascending order and select the first k nearest training data instances to the test instance.
3. Predict the class of the test instance by majority voting (if target attribute is discrete valued) or mean (if target attribute is continuous valued) of the k selected nearest instances.

Example 4.1: Consider the student performance training dataset of 8 data instances shown in Table 4.2 which describes the performance of individual students in a course and their CGPA obtained in the previous semesters. The independent attributes are CGPA, Assessment and Project. The target variable is 'Result' which is a discrete valued variable that takes two values 'Pass' or 'Fail'. Based on the performance of a student, classify whether a student will pass or fail in that course.

Table 4.2: Training Dataset *T*

S.No.	CGPA	Assessment	Project Submitted	Result
1.	9.2	85	8	Pass
2.	8	80	7	Pass
3.	8.5	81	8	Pass
4.	6	45	5	Fail
5.	6.5	50	4	Fail
6.	8.2	72	7	Pass
7.	5.8	38	5	Fail
8.	8.9	91	9	Pass

Solution: Given a test instance (6.1, 40, 5) and a set of categories {Pass, Fail} also called as classes, we need to use the training set to classify the test instance using Euclidean distance.

The task of classification is to assign a category or class to an arbitrary instance.

Assign $k = 3$.

Step 1: Calculate the Euclidean distance between the test instance (6.1, 40, and 5) and each of the training instances as shown in Table 4.3.

Table 4.3: Euclidean Distance

S.No.	CGPA	Assessment	Project Submitted	Result	Euclidean Distance
1.	9.2	85	8	Pass	$\sqrt{(9.2-6.1)^2+(85-40)^2+(8-5)^2}$ $= 45.2063$
2.	8	80	7	Pass	$\sqrt{(8-6.1)^2+(80-40)^2+(7-5)^2}$ $= 40.09501$
3.	8.5	81	8	Pass	$\sqrt{(8.5-6.1)^2+(81-40)^2+(8-5)^2}$ $= 41.17961$
4.	6	45	5	Fail	$\sqrt{(6-6.1)^2+(45-40)^2+(5-5)^2}$ $= 5.001$
5.	6.5	50	4	Fail	$\sqrt{(6.5-6.1)^2+(50-40)^2+(4-5)^2}$ $= 10.05783$
6.	8.2	72	7	Pass	$\sqrt{(8.2-6.1)^2+(72-40)^2+(7-5)^2}$ $= 32.13114$
7.	5.8	38	5	Fail	$\sqrt{(5.8-6.1)^2+(38-40)^2+(5-5)^2}$ $= 2.022375$
8.	8.9	91	9	Pass	$\sqrt{(8.9-6.1)^2+(91-40)^2+(9-5)^2}$ $= 51.23319$

Step 2: Sort the distances in the ascending order and select the first 3 nearest training data instances to the test instance. The selected nearest neighbors are shown in Table 4.4.

Table 4.4: Nearest Neighbors

Instance	Euclidean Distance	Class
4	5.001	Fail
5	10.05783	Fail
7	2.022375	Fail

Here, we take the 3 nearest neighbors as instances 4, 5 and 7 with smallest distances.

Step 3: Predict the class of the test instance by majority voting.

The class for the test instance is predicted as 'Fail'.

k-NN classifier performance is strictly affected by three factors such as the number of nearest neighbors (i.e., selection of *k*), distance metric and decision rule.

4.3 Weighted k-Nearest-Neighbor Algorithm

Algorithm 4.2: Weighted *k*-NN

Inputs: Training dataset '*T*', Distance metric '*d*', Weighting function $w(i)$, Test instance '*t*', the number of nearest neighbors '*k*'

Output: Predicted class or category

Prediction: For test instance *t*,

1. For each instance '*i*' in Training dataset *T*, compute the distance between the test instance *t* and every other instance '*i*' using a distance metric (Euclidean distance).
   [Continuous attributes - Euclidean distance between two points in the plane with coordinates (x_1, y_1) and (x_2, y_2) is given as dist $((x_1, y_1), (x_2, y_2)) = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2}$]
   [Categorical attributes (Binary) - Hamming Distance: If the values of two instances are the same, the distance *d* will be equal to 0. Otherwise $d = 1$.]
2. Sort the distances in the ascending order and select the first '*k*' nearest training data instances to the test instance.
3. Predict the class of the test instance by weighted voting technique (Weighting function $w(i)$) for the *k* selected nearest instances:
   - Compute the inverse of each distance of the '*k*' selected nearest instances.
   - Find the sum of the inverses.
   - Compute the weight by dividing each inverse distance by the sum. (Each weight is a vote for its associated class).
   - Add the weights of the same class.
   - Predict the class by choosing the class with the maximum vote.

4.4 Nearest Centroid Classifier

Algorithm 4.3: Nearest Centroid Classifier

Inputs: Training dataset T, Distance metric d, Test instance t

Output: Predicted class or category

1. Compute the mean/centroid of each class.
2. Compute the distance between the test instance and mean/centroid of each class (Euclidean Distance).
3. Predict the class by choosing the class with the smaller distance.

4.5 Locally Weighted Regression (LWR)

Locally Weighted Regression (LWR) is a non-parametric supervised learning algorithm that performs local regression by combining regression model with nearest neighbor's model. LWR is also referred to as a memory-based method as it requires training data while prediction but uses only the training data instances locally around the point of interest. Using nearest neighbors algorithm, we find the instances that are closest to a test instance and fit linear function to each of those 'k' nearest instances in the local regression model. The key idea is that we need to approximate the linear functions of all 'k' neighbors that minimize the error such that the prediction line is no more linear but rather it is a curve.

Ordinary linear regression finds out a linear relationship between the input x and the output y.

Given training dataset T,

Hypothesis function $h_\beta(x)$, the predicted target output is a linear function where β_0 is the intercept and β_1 is the coefficient of x.

It is given in Eq. (4.1) as,

$$h_\beta(x) = \beta_0 + \beta_1 x \tag{4.1}$$

The cost function is such that it minimizes the error difference between the predicted value $h_\beta(x)$ and true value 'y' and it is given as in Eq. (4.2).

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{m}\left(h_\beta(x_i) - y_i\right)^2 \tag{4.2}$$

where 'm' is the number of instances in the training dataset.

Now the cost function is modified for locally weighted linear regression including the weights only for the nearest neighbor points. Hence, the cost function is given as in Eq. (4.3).

$$J(\beta) = \frac{1}{2}\sum_{i=1}^{m} w_i\left(h_\beta(x_i) - y_i\right)^2 \tag{4.3}$$

where w_i is the weight associated with each x_i.

The weight function used is a Gaussian kernel that gives a higher value for instances that are close to the test instance, and for instances far away, it tends to zero but never equals to zero.

w_i is computed in Eq. (4.4) as,

$$w_i = e^{\frac{-(x_i - x)^2}{2\tau^2}} \tag{4.4}$$

Chapter -5 Regression Analysis

Regression analysis is a supervised learning method for predicting continuous variables. The difference between classification and regression analysis is that regression methods are used to predict qualitative variables or continuous numbers unlike categorical variables or labels. It is used to predict linear or non-linear relationships among variables of the given dataset. This chapter deals with an introduction of regression and its various types.

Regression analysis is the premier method of supervised learning. This is one of the most popular and oldest supervised learning technique. Given a training dataset D containing N training points (x_i, y_i), where $i = 1...N$, regression analysis is used to model the relationship between one or more independent variables x_i and a dependent variable y_i. The relationship between the dependent and independent variables can be represented as a function as follows:

$$y = f(x) \tag{5.1}$$

Here, the feature variable x is also known as an explanatory variable, exploratory variable, a predictor variable, an independent variable, a covariate, or a domain point. y is a dependent variable. Dependent variables are also called as labels, target variables, or response variables.

Regression is used to predict continuous variables or quantitative variables such as price and revenue. Thus, the primary concern of regression analysis is to find answer to questions such as:

1. What is the relationship between the variables?
2. What is the strength of the relationships?
3. What is the nature of the relationship such as linear or non-linear?
4. What is the relevance of the attributes?
5. What is the contribution of each attribute?

There are many applications of regression analysis. Some of the applications of regressions include predicting:

1. Sales of a goods or services
2. Value of bonds in portfolio management
3. Premium on insurance companies
4. Yield of crops in agriculture
5. Prices of real estate

Introduction to Linearity ,Correlation, Causation

Correlation among two variables can be done effectively using a Scatter plot, which is a plot between explanatory variables and response variables. It is a 2D graph showing the relationship between two variables. The x-axis of the scatter plot is independent, or input or predictor variables and y-axis of the scatter plot is output or dependent or predicted variables.

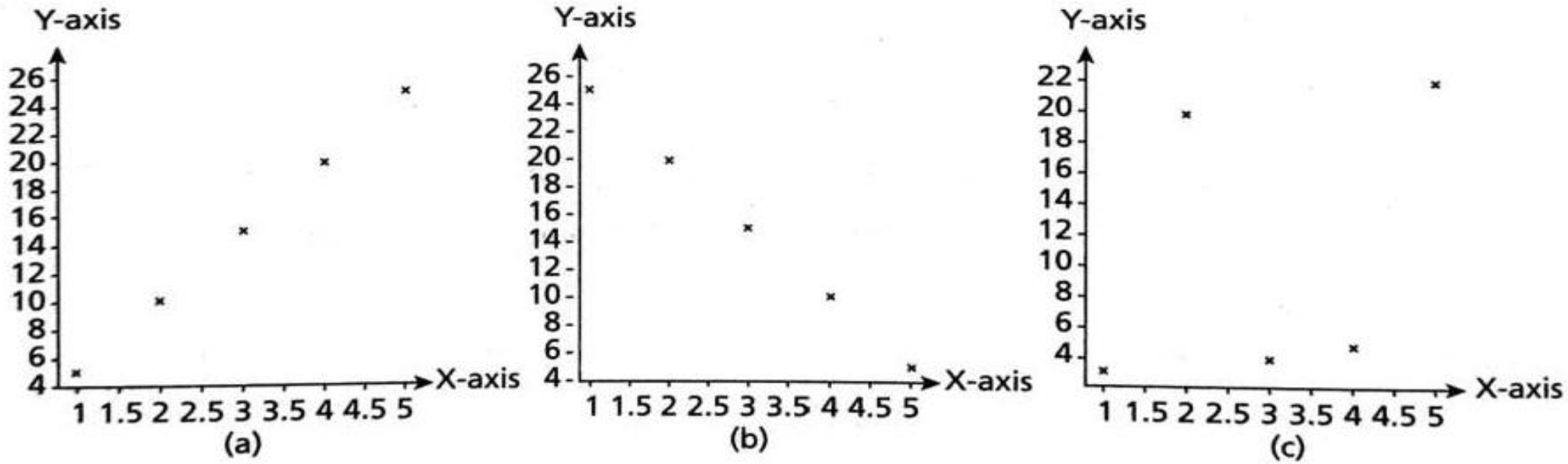


Figure 5.1: Examples of (a) Positive Correlation (b) Negative Correlation (c) Random Points with No Correlation

Linearity and Non-linearity Relationships

The linearity relationship between the variables means the relationship between the dependent and independent variables can be visualized as a straight line. The line of the form, $y = ax + b$ can be fitted to the data points that indicate the relationship between x and y. By linearity, it is meant that as one variable increases, the corresponding variable also increases in a linear manner. A linear relationship is shown in Figure 5.2 (a). A non-linear relationship exists in functions such as exponential function and power function and it is shown in Figures 5.2 (b) and 5.2 (c). Here, x-axis is given by x data and y-axis is given by y data.

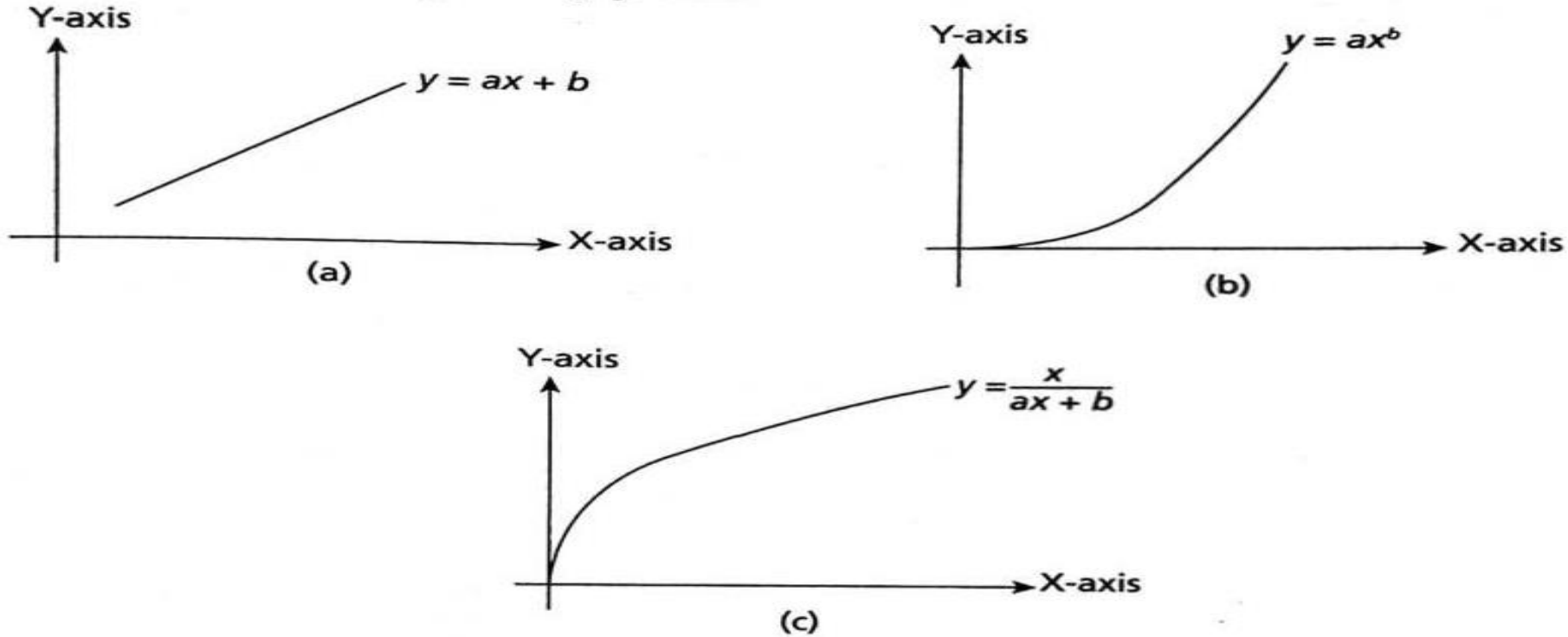


Figure 5.2: (a) Example of Linear Relationship of the Form $y = ax + b$ (b) Example of a Non-linear Relationship of the Form $y = ax^b$ (c) Examples of a Non-linear Relationship $y = \frac{x}{ax + b}$

The functions like exponential function ($y = ax^b$) and power function $\left(y = \frac{x}{ax + b}\right)$ are non-linear relationships between the dependent and independent variables that cannot be fitted in a line. This is shown in Figures 5.2 (b) and (c).

Types of Regression Methods

The classification of regression methods is shown in Figure 5.3.

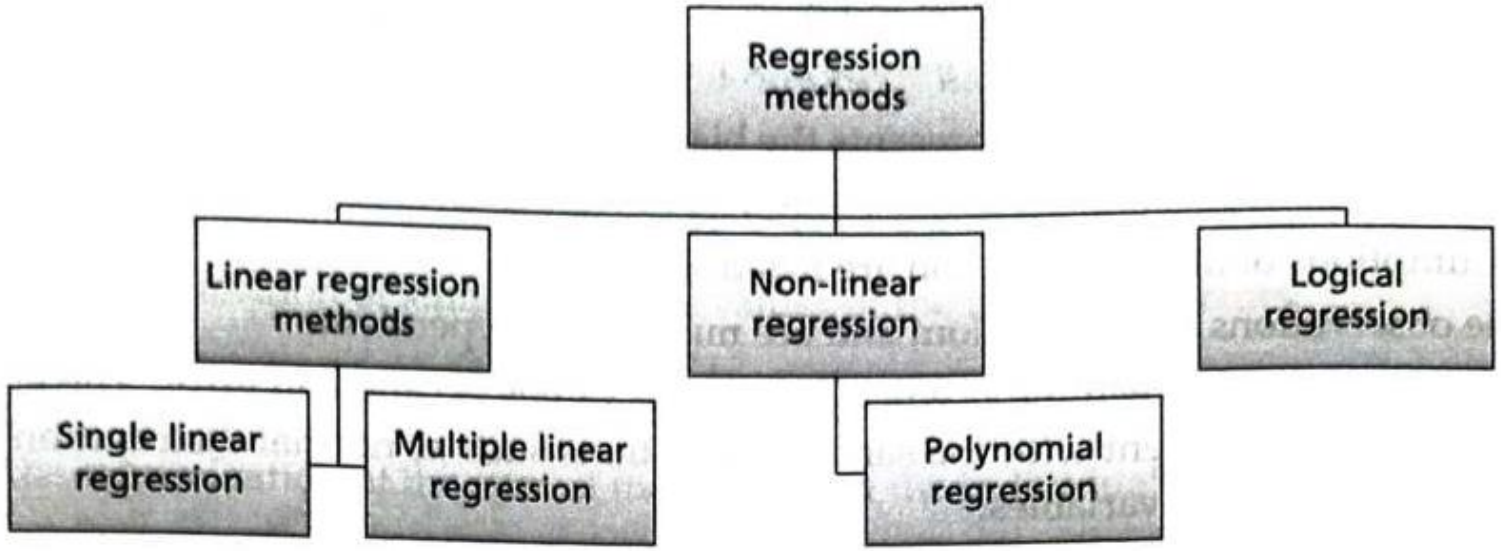


Figure 5.3: Types of Regression Methods

Linear Regression It is a type of regression where a line is fitted upon given data for finding the linear relationship between one independent variable and one dependent variable to describe relationships.

Multiple Regression It is a type of regression where a line is fitted for finding the linear relationship between two or more independent variables and one dependent variable to describe relationships among variables.

Polynomial Regression It is a type of non-linear regression method of describing relationships among variables where N^{th} degree polynomial is used to model the relationship between one independent variable and one dependent variable. Polynomial multiple regression is used to model two or more independent variables and one dependant variable.

Logistic Regression It is used for predicting categorical variables that involve one or more independent variables and one dependent variable. This is also known as a binary classifier.

Lasso and Ridge Regression Methods These are special variants of regression method where regularization methods are used to limit the number and size of coefficients of the independent variables.

Limitations of Regression Method

1. Outliers – Outliers are abnormal data. It can bias the outcome of the regression model, as outliers push the regression line towards it.
2. Number of cases – The ratio of independent and dependent variables should be at least 20 : 1. For every explanatory variable, there should be at least 20 samples. Atleast five samples are required in extreme cases.
3. Missing data – Missing data in training data can make the model unfit for the sampled data.
4. Multicollinearity – If exploratory variables are highly correlated (0.9 and above), the regression is vulnerable to bias. Singularity leads to perfect correlation of 1. The remedy is to remove exploratory variables that exhibit correlation more than 1. If there is a tie, then the tolerance (1 – *R* squared) is used to eliminate variables that have the greatest value.

5.3 INTRODUCTION TO LINEAR REGRESSION

In the simplest form, the linear regression model can be created by fitting a line among the scattered data points. The line is of the form given in Eq. (5.2).

$$y = a_0 + a_1 \times x + e \tag{5.2}$$

Here, a_0 is the intercept which represents the bias and a_1 represents the slope of the line. These are called regression coefficients. e is the error in prediction.

The assumptions of linear regression are listed as follows:

1. The observations (y) are random and are mutually independent.
2. The difference between the predicted and true values is called an error. The error is also mutually independent with the same distributions such as normal distribution with zero mean and constant variables.
3. The distribution of the error term is independent of the joint distribution of explanatory variables.
4. The unknown parameters of the regression models are constants.

The idea of linear regression is based on Ordinary Least Square (OLS) approach. This method is also known as ordinary least squares method. In this method, the data points are modelled using a straight line. Any arbitrarily drawn line is not an optimal line. In Figure 5.4, three data points and their errors (e_1, e_2, e_3) are shown. The vertical distance between each point and the line (predicted by the approximate line equation $y = a_0 + a_1x$) is called an error. These individual errors are added to compute the total error of the predicted line. This is called sum of residuals. The squares of the individual errors can also be computed and added to give a sum of squared error. The line with the lowest sum of squared error is called line of best fit.

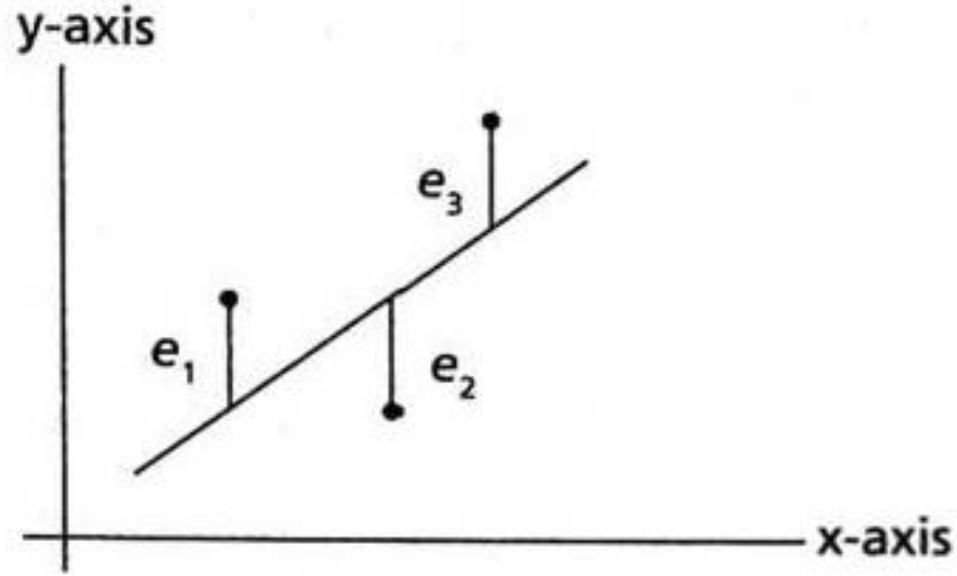


Figure 5.4: Data Points and their Errors

Mathematically, based on Eq. (5.2), the line equations for points $(x_1, x_2, \ldots, x_n)$ are:

$$y_1 = (a_0 + a_1 x_1) + e_1$$
$$y_2 = (a_0 + a_1 x_2) + e_2$$
$$\vdots$$
$$y_n = (a_0 + a_1 x_n) + e_n$$

In general, the error is given as: $e_i = y_i - (a_0 + a_1 x_i)$

This can be extended into the set of equations as shown in Eq. (5.3).

A regression line is the line of best fit for which the sum of the squares of residuals is minimum. The minimization can be done as minimization of individual errors by finding the parameters a_0 and a_1 such that:

$$E = \sum_{i=1}^{n} e_i = \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i)) \tag{5.5}$$

Or as the minimization of sum of absolute values of the individual errors:

$$E = \sum_{i=1}^{n} |e_i| = \sum_{i=1}^{n} |(y_i - (a_0 + a_1 x_i))| \tag{5.6}$$

Or as the minimization of the sum of the squares of the individual errors:

$$E = \sum_{i=1}^{n} (e_i)^2 = \sum_{i=1}^{n} (y_i - (a_0 + a_1 x_i))^2 \tag{5.7}$$

Sum of the squares of the individual errors, often preferred as individual errors (positive and negative errors), do not get cancelled out and are always positive, and sum of squares results in a large increase even for a small change in the error. Therefore, this is preferred for linear regression.

Therefore, linear regression is modelled as a minimization function as follows:

$$\begin{aligned} J(a_1, a_0) &= \sum_{i=1}^{n} [y_i - f(x_i)]^2 \\ &= \sum_{i=1}^{n} [y_i - (a_0 + a_1 x_i)]^2 \end{aligned} \tag{5.8}$$

Here, $J(a_0, a_1)$ is the criterion function of parameters a_0 and a_1. This needs to be minimized. This is done by differentiating and substituting to zero. This yields the coefficient values of a_0 and a_1. The values of estimates of a_0 and a_1 are given as follows:

$$a_1 = \frac{(\overline{xy}) - (\overline{x})(\overline{y})}{(\overline{x_i^2}) - (\overline{x})^2} \tag{5.9}$$

And the value of a_0 is given as follows:

$$a_0 = (\overline{y}) - a_1 \times \overline{x} \tag{5.10}$$

Example 5.1: Let us consider an example where the five weeks' sales data (in Thousands) is given as shown below in Table 5.1. Apply linear regression technique to predict the 7^{th} and 9^{th} month sales.

Table 5.1: Sample Data

x_i (Week)	y_i (Sales in Thousands)
1	1.2
2	1.8
3	2.6
4	3.2
5	3.8

Solution: Here, there are 5 items, i.e., $i = 1, 2, 3, 4, 5$. The computation table is shown below (Table 5.2). Here, there are five samples, so i ranges from 1 to 5.

Table 5.2: Computation Table

x_i	y_i	$(x_i)^2$	$x_i \times y_i$
1	1.2	1	1.2
2	1.8	4	3.6
3	2.6	9	7.8
4	3.2	16	12.8
5	3.8	25	19
Sum = 15 Average of (x_i) $= \bar{x} = \frac{15}{5}$ $= 3$	Sum = 12.6 Average of (y_i) $= \bar{y} = \frac{12.6}{5}$ $= 2.52$	Sum = 55 Average of (x_i^2) $= \overline{x_i^2} = \frac{55}{5}$ $= 11$	Sum = 44.4 Average of $(x_i \times y_i)$ $= \overline{xy} = \frac{44.4}{5}$ $= 8.88$

Let us compute the slope and intercept now using Eq. (5.9) as:

$$a_1 = \frac{8.88 - 3(2.52)}{11 - 3^2} = 0.66$$

$$a_0 = 2.52 - 0.66 \times 3 = 0.54$$

The fitted line is shown in Figure 5.5.

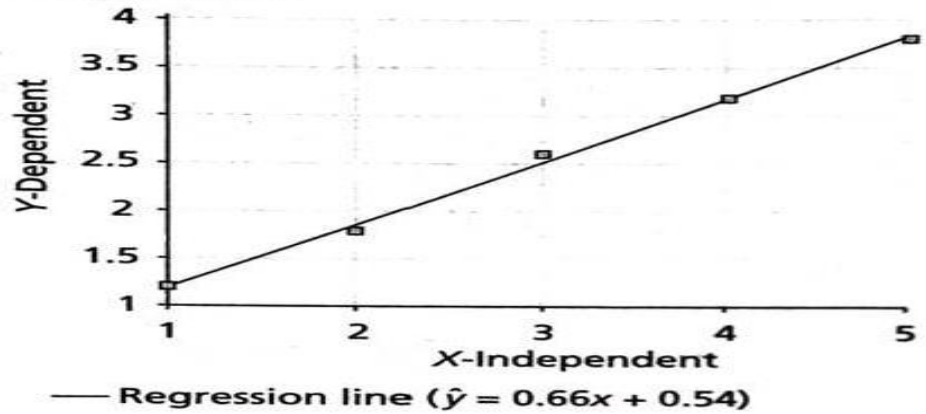


Figure 5.5: Linear Regression Model Constructed

Let us model the relationship as $y = a_0 + a_1 \times x$. Therefore, the fitted line for the above data is: $y = 0.54 + 0.66 \times x$.

The predicted 7th week sale would be (when $x = 7$), $y = 0.54 + 0.66 \times 7 = 5.16$ and the 12th month, $y = 0.54 + 0.66 \times 12 = 8.46$. All sales are in thousands.

Linear Regression in Matrix Form

Matrix notations can be used for representing the values of independent and dependent variables. This is illustrated through Example 5.2.

The Eq. (5.3) can be written in the form of matrix as follows:

$$\begin{pmatrix} y_1 \\ y_2 \\ \vdots \\ y_n \end{pmatrix} = \begin{pmatrix} 1 & x_1 \\ 1 & x_2 \\ 1 & \vdots \\ 1 & x_n \end{pmatrix} \begin{pmatrix} a_0 \\ a_1 \end{pmatrix} + \begin{pmatrix} e_1 \\ e_2 \\ \vdots \\ e_n \end{pmatrix} \tag{5.11}$$

This can be written as:

$Y = Xa + e$, where X is an $n \times 2$ matrix, Y is an $n \times 1$ vector, a is a 2×1 column vector and e is an $n \times 1$ column vector.

Example 5.2: Find linear regression of the data of week and product sales (in Thousands) given in Table 5.3. Use linear regression in matrix form.

Table 5.3: Sample Data for Regression

x_i (Week)	y_i (Product Sales in Thousands)
1	1
2	3
3	4
4	8

Solution: Here, the dependent variable X is be given as:

$$x^T = [1\ 2\ 3\ 4]$$

And the independent variable is given as follows:

$$y^T = [1\ 3\ 4\ 8]$$

The data can be given in matrix form as follows:

$X = \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix}$. The first column can be used for setting bias.

and $Y = \begin{pmatrix} 1 \\ 3 \\ 4 \\ 8 \end{pmatrix}$

The regression is given as:

$$a = ((X^T X)^{-1} X^T) Y$$

1. Computation of $(X^TX) = \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} \times \begin{pmatrix} 1 & 1 \\ 1 & 2 \\ 1 & 3 \\ 1 & 4 \end{pmatrix} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}$

2. Computation of matrix inverse of $(X^TX)^{-1} = \begin{pmatrix} 4 & 10 \\ 10 & 30 \end{pmatrix}^{-1} = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix}$

3. Computation of $((X^TX)^{-1}X^T) = \begin{pmatrix} 1.5 & -0.5 \\ -0.5 & 0.2 \end{pmatrix} \times \begin{pmatrix} 1 & 1 & 1 & 1 \\ 1 & 2 & 3 & 4 \end{pmatrix} = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{pmatrix}$

4. Finally, $((X^TX)^{-1}X^T)Y = \begin{pmatrix} 1 & 0.5 & 0 & -0.5 \\ -0.3 & -0.1 & 0.1 & 0.3 \end{pmatrix} \times \begin{pmatrix} 1 \\ 3 \\ 4 \\ 8 \end{pmatrix} = \begin{pmatrix} -1.5 \\ 2.2 \end{pmatrix} \begin{pmatrix} Intercept \\ slope \end{pmatrix}$

Thus, the substitution of values in Eq. (5.11) using the previous steps yields the fitted line as $2.2\,x - 1.5$.

5.4 VALIDATION OF REGRESSION METHODS

The regression model should be evaluated using some metrics for checking the correctness. The following metrics are used to validate the results of regression.

Standard Error

Residuals or error is the difference between the actual (y) and predicted value ($\hat{y}$).

If the residuals have normal distribution, then the mean is zero and hence it is desirable. This is a measure of variability in finding the coefficients. It is preferable that the error be less than the coefficient estimate. The standard deviation of residuals is called residual standard error. If it is zero, then it means that the model fits the data correctly.

Mean Absolute Error (MAE)

MAE is the mean of residuals. It is the difference between estimated or predicted target value and actual target incomes. It can be mathematically defined as follows:

$$\text{MAE} = \frac{1}{n}\sum_{i=0}^{n-1}\left|y_i - \hat{y}_i\right| \tag{5.12}$$

Here, $\hat{y}$ is the estimated or predicted target output and y is the actual target output, and n is the number of samples used for regression analysis.

Mean Squared Error (MSE)

It is the sum of square of residuals. This value is always positive and closer to 0. This is given mathematically as:

$$\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2 \tag{5.13}$$

Root Mean Square Error (RMSE)

The square root of the MSE is called RMSE. This is given as:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{\frac{1}{n}\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2} \tag{5.14}$$

Relative MSE

Relative MSE is the ratio of the prediction ability of the $\hat{y}$ to the average of the trivial population. The value of zero indicates that the model is perfect and its value ranges between 0 and 1. If the value is more than 1, then the created model is not a good one. This is given as follows:

$$\text{RelMSE} = \frac{\sum_{i=0}^{n-1}(y_i - \hat{y}_i)^2}{\sum_{i=0}^{n-1}(y_i - \overline{y}_i)^2} \tag{5.15}$$

Coefficient of Variation

Coefficient of variation is unit less and is given as:

$$\text{CV} = \frac{\text{RMSE}}{\overline{y}} \tag{5.16}$$

Example 5.3: Consider the following training set Table 5.4 for predicting the sales of the items.

Table 5.4: Training Item Table

Items x_i	Actual Sales (In Thousands) y_i
I_1	80
I_2	90
I_3	100
I_4	110
I_5	120

Consider two fresh items I_6 and I_7, whose actual values are 80 and 75, respectively. A regression model predicts the values of the items I_6 and I_7 as 75 and 85, respectively. Find MAE, MSE, RMSE, RelMSE and CV.

Solution: The test items' actual and prediction is given in Table 5.5 as:

Table 5.5: Test Item Table

Test Items	Actual Value y_i	Predicted Value $\overline{y}_i$
I_6	80	75
I_7	75	85

Mean Absolute Error (MAE) using Eq. (5.12) is given as:

$$\text{MAE} = \frac{1}{2} \times |80 - 75| + |75 - 85| = \frac{15}{2} = 7.5$$

Mean Squared Error (MSE) using Eq. (5.13) is given as:

$$\text{MSE} = \frac{1}{2} \times |80 - 75|^2 + |75 - 85|^2 = \frac{125}{2} = 62.5$$

Root Mean Square error using Eq. (5.14) is given as:

$$\text{RMSE} = \sqrt{\text{MSE}} = \sqrt{62.5} = 7.91$$

For finding RelMSE and CV, the training table should be used to find the average of y. The average of y is $\frac{80 + 90 + 100 + 110 + 120}{5} = \frac{500}{5} = 100.$

RelMSE using Eq. (5.15) can be computed as:

$$\text{RelMSE} = \frac{(80 - 75)^2 + (75 - 85)^2}{(80 - 100)^2 + (75 - 100)^2} = \frac{125}{1025} = 0.1219$$

CV can be computed using Eq. (5.16) as $\frac{\sqrt{62.5}}{100} = 0.08.$

Coefficient of Determination

To understand the coefficient of determination, one needs to understand the total variation of coefficients in regression analysis. The sum of the squares of the differences between the y-value of the data pair and the average of y is called total variation. Thus, the following variations can be defined.

The explained variation is given as:

$$= \Sigma(\hat{y}_i - \overline{y}_i)^2 \tag{5.17}$$

The unexplained variation is given as:

$$= \Sigma(y_i - \hat{y}_i)^2 \tag{5.18}$$

Thus, the total variation is equal to the explained variation and the unexplained variation.

The coefficient of determination r^2 is the ratio of the explained and total variations.

$$r^2 = \frac{\text{Explained variation}}{\text{Total variation}} \tag{5.19}$$

It is a measure of how many future samples are likely to be predicted by the regression model. Its value ranges from 1 to $-\infty$, where 1 is the most optimum value. It also signifies the proportion of variance. Here, r is the correlation coefficient. If $r = 0.95$, then r^2 is given as $0.95 \times 0.95 = 0.9025$. This means that 90% of the model can be explained by the relationship between x and y. The rest 10% is unexplained and that may be due to various reasons such as noise, chance, or error.

Standard Error Estimate

Standard error estimate is another useful measure of regression. It is the standard deviation of the observed values to the predicted values. This is given as:

$$s_e = \sqrt{\frac{\Sigma(y_i - \hat{y}_i)}{n-2}} \tag{5.20}$$

Here, as usual, y_i is the observed value and $\hat{y}_i$ is the predicted value. Here, n is the number of samples.

Example 5.4: Let us consider the data given in the Table 5.3 with actual and predicted values. Find standard error estimate.

Solution: The observed value or the predicted value is given below in Table 5.6.

Table 5.6: Sample Data

x_i	y_i	Predicted Value	$(y - \hat{y})^2$
1	1.5	1.46	$(1.5 - 1.46)^2 = 0.0016$
2	2.9	2.02	$(2.9 - 2.02)^2 = 0.7744$
3	2.7	2.58	$(2.7 - 2.58)^2 = 0.0144$
4	3.1	3.14	$(3.1 - 3.14)^2 = 0.0016$

The sum of $(y - \hat{y})^2$ for all $i = 1, 2, 3$ and 4 (i.e., number of samples $n = 4$) is 0.792. The standard deviation error estimate as given in Eq. (5.20) is:

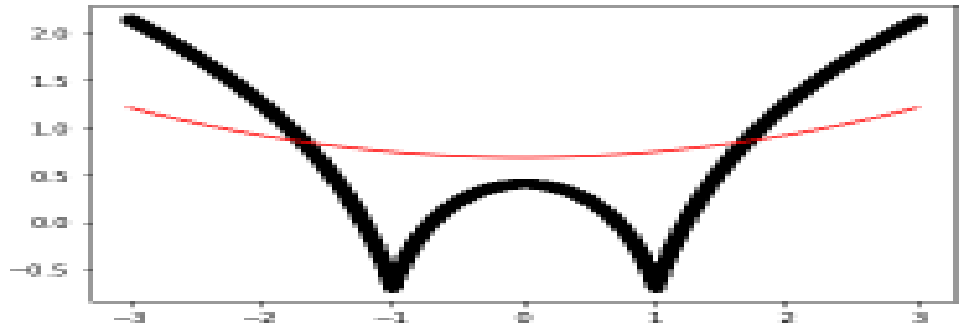
$$\sqrt{\frac{0.792}{4-2}} = \sqrt{0.396} = 0.629$$

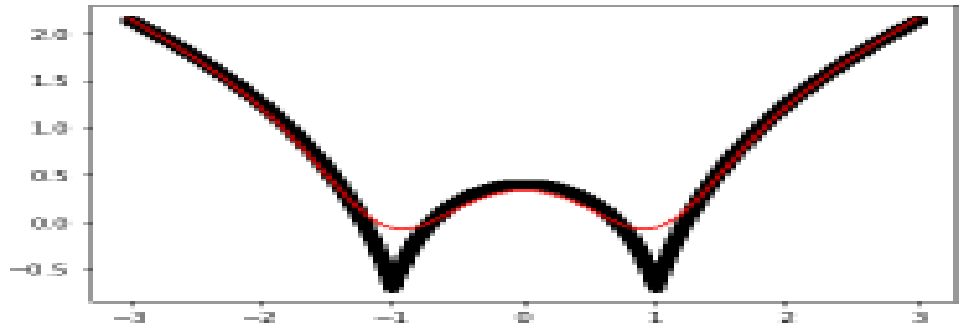
3]:

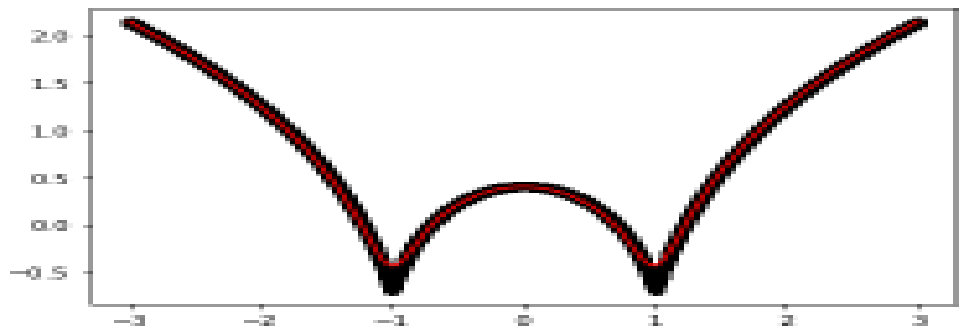
```python
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import datasets
iris=datasets.load_iris()
print("Iris Data set loaded...")
x_train, x_test, y_train, y_test = train_test_split(iris.data,iris.target,test_size=0.1)
#random_state=0
for i in range(len(iris.target_names)):
    print("Label", i , "-",str(iris.target_names[i]))
classifier = KNeighborsClassifier(n_neighbors=3)
classifier.fit(x_train, y_train)
y_pred=classifier.predict(x_test)
print("Results of Classification using K-nn with K=3 ")
for r in range(0,len(x_test)):
    print(" Sample:", str(x_test[r]), " Actual-label:", str(y_test[r])," Predicted-label:", str(y_pred[r]))

    print("Classification Accuracy :" , classifier.score(x_test,y_test));
```

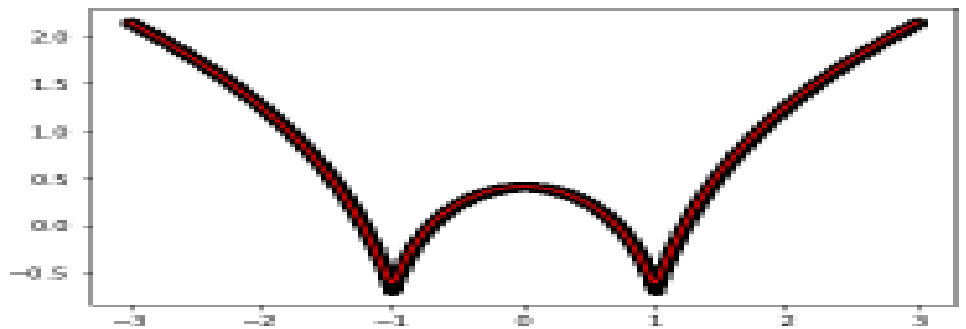
```
Iris Data set loaded...
Label 0 - setosa
Label 1 - versicolor
Label 2 - virginica
Results of Classification using K-nn with K=3
 Sample: [5.7 3.  4.2 1.2]  Actual-label: 1  Predicted-label: 1
Classification Accuracy : 0.9333333333333333
 Sample: [6.5 3.  5.5 1.8]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
 Sample: [7.7 2.8 6.7 2. ]  Actual-label: 2  Predicted-label: 2
Classification Accuracy : 0.9333333333333333
```

```
import numpy as np
import matplotlib.pyplot as plt
def local_regression(x0, X, Y, tau):
    x0 = [1, x0]
    X = [[1, i] for i in X]
    X = np.asarray(X)
    xw = (X.T) * np.exp(np.sum((X - x0) ** 2, axis=1) / (-2 * tau**2))
    beta = np.linalg.pinv(xw @ X) @ xw @ Y
    beta=beta @ x0
    return beta
def draw(tau):
    prediction = [local_regression(x0, X, Y, tau) for x0 in domain]
    plt.plot(X, Y, 'o', color='black')
    plt.plot(domain, prediction, color='red')
    plt.show()
X = np.linspace(-3, 3, num=1000)
domain = X
Y = np.log(np.abs(X ** 2 - 1) + .5)
draw(10)
draw(0.1)
draw(0.01)
draw(0.001)
```


2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3


2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3


2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3


2.0
1.5
1.0
0.5
0.0
−0.5
−3
−2
−1
0
1
2
3