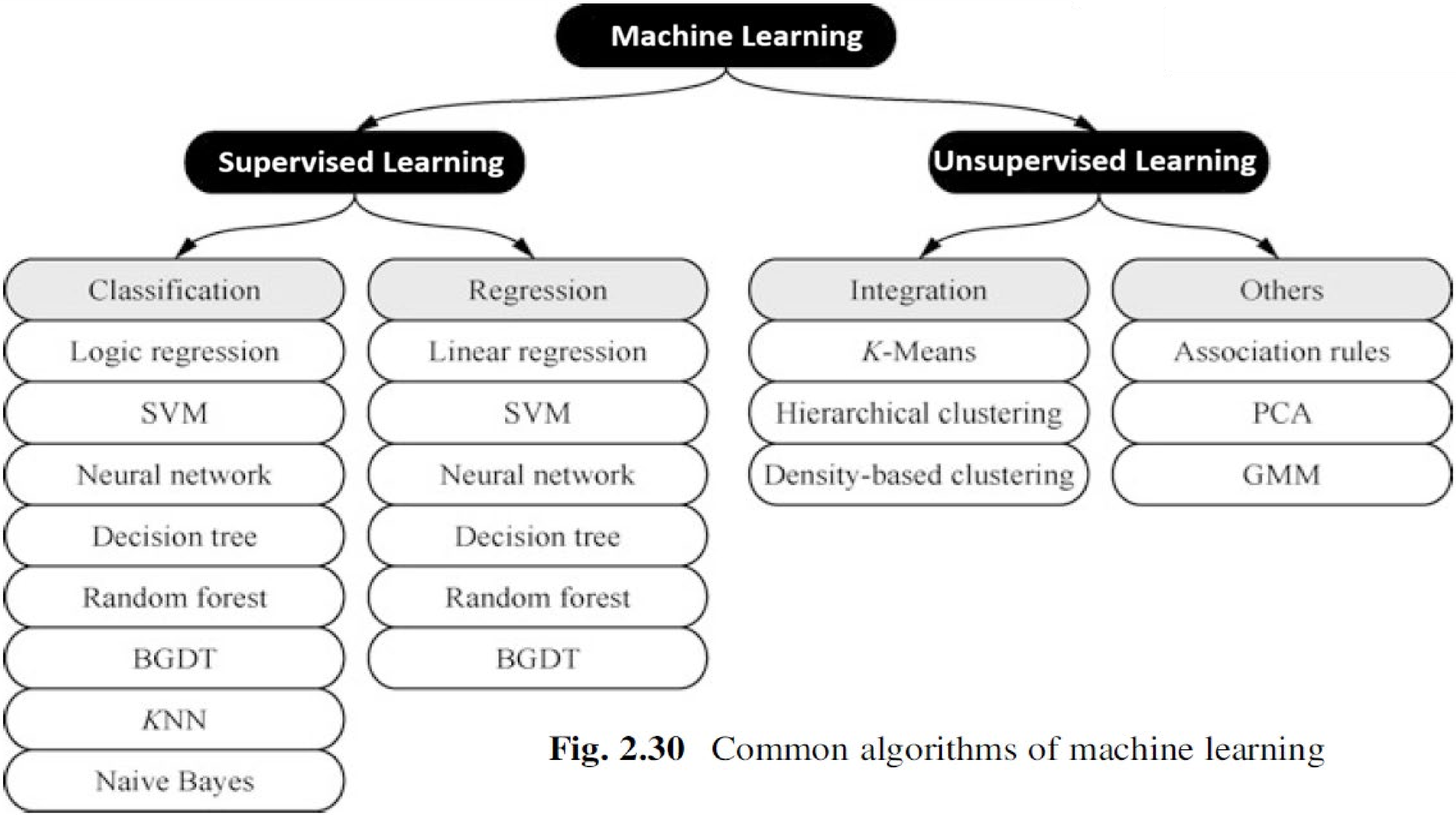
Lecture 2
Background of Supervised Machine Learning

Resources:

1. Artificial Intelligence Foundations of Computational Agents, 2nd Edition, David L. Poole and Alan K Mackworth, Cambridge University Press. 2019
2. https://doi.org/10.1007/978-981-19-2879-6
3. ChatGPT

Overview



- Machine Learning
  - Supervised Learning
    - Classification
      - Logic regression
      - SVM
      - Neural network
      - Decision tree
      - Random forest
      - BGDT
      - *K*NN
      - Naive Bayes
    - Regression
      - Linear regression
      - SVM
      - Neural network
      - Decision tree
      - Random forest
      - BGDT
  - Unsupervised Learning
    - Integration
      - *K*-Means
      - Hierarchical clustering
      - Density-based clustering
    - Others
      - Association rules
      - PCA
      - GMM

Fig. 2.30 Common algorithms of machine learning

Overview

- The ability to **learn** is essential to an **intelligent** agent.
- **Learning** is the ability of an agent to improve its behavior based on **experience**:
  - The *range of behaviors* is expanded; the agent can do more.
  - The *accuracy of tasks* is improved; the agent can do things better.
  - The *speed is improved*; the agent can do things faster.
- This section considers the problem of predicting **supervised learning**:
  - given a set of **training examples** of **input–output pairs**, predict the **output of a new example**.

Learning Issues

- The following components are part of any learning problem:
    - **Task:** The behavior or task that is being improved
    - **Data:** The experiences (in the form of a set of numerical values) that are used to improve performance in the task, in the form of a sequence of examples
    - **Measure of improvement***: How learning is measured?*
        - New skills that were not present initially, increasing accuracy in prediction.

Learning Issues - Task

- The most studied learning **task** is **supervised learning:**
  - Generate a ***set of training examples*** with ***input*** and ***target*** features and predict the value of *target features* for a **new example** (or unseen data which has only input features).
  - We called this a **classification** when the **target features** are **discrete**
    - **Discrete data** can take on only **two levels** of values
  - We called this a **regression** when the target features are **continuous**
    - For instance, the people with liver disease is **discrete**, but their weights are **continuous**

Learning Issues - Feedback

- Learning tasks can be characterized by the **feedback** given to the learner
- In **supervised learning**, what must be learned is specified for each example:
    - ***Supervised classification*** occurs when a trainer provides the classification for each example.
    - ***Supervised learning*** occurs when the agent is given immediate feedback about the value of each action.
- **Unsupervised learning** occurs when no classifications are given, and the learner must discover categories and regularities within the data.

Learning Issues - Online and offline

- In **offline learning**, all training examples are available to a **learning agent** before acting.
- In **online learning**, training examples arrive while the agent is acting
  - An agent that learns **online** requires some representation of its previous examples before seeing all its models.
  - As *new examples are observed, the agent must update its internal data representation*.
- **Active learning** is a form of **online learning** in which the agent acts to acquire valuable examples **from its learned data**

Learning Issues - Measuring success

- **Learning** is defined in terms of improving performance based on some **measures**.
- To know whether an agent has learned, we must define a **measure of success**.
  - The **measure of success** is usually not how well the agent performs on the **training dataset** (called **seen data**), but how well the agent acts for **new** or **test data** or **unseen data**.
- In **classification**, measuring success is not based on the classification of all training examples correctly.
  - But it is based on correctly classifying the **unseen data**.
  - *The* **unseen data** *should not be known during training.*

Learning Issues - Bias

- The tendency *to prefer one hypothesis over another* is called a **bias**
  - For example, consider two learning algorithms, ***X*** and ***Y,*** and it is said that a hypothesis is better than ***X'*s** or ***Y'*s** hypothesis
  - Both ***X*** and ***Y*** accurately predict all the data given, but it is something external to the data.
  - Without a **bias** value, a learning algorithm cannot predict **unseen** examples.

Learning Issues - Learning as search

- Given a **representation** and a **bias**, the problem of **learning** can be reduced to a **search:**
  - **Learning** is a **search** through the **space of possible representations**, trying to find the representation or representations that **best fit** the data based on the **bias** value.
  - Nearly all search techniques used in **machine learning** can be seen as forms of **local search** through a space of data representations.
  - The definition of **machine learning** includes the definition of the ***search space***, the ***evaluation function***, and the ***search method***.

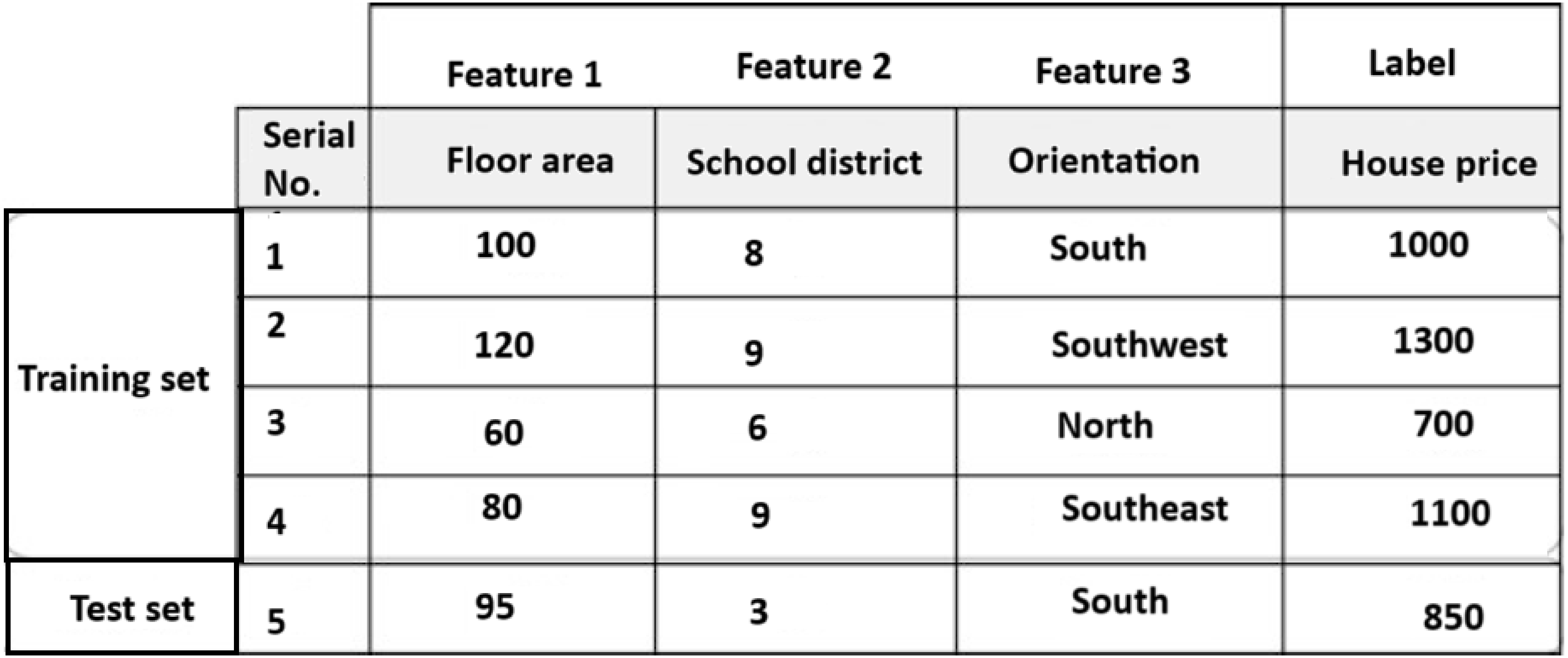
Supervised Learning

- **Supervised learning** is based on a set of training examples (a set of ***input*** and ***target*** features)
  - *Training aims to learn the values of the **target features** from the **input features**.*
- The learner is given training examples:
  - a set of **input features, $X_1, . . . , X_n$;**
  - a set of **target features, $Y_1, . . . , Y_k$;**
- The trained agent is tested with a set of **test examples,** which includes only the **input features** (it is the untrained data, called the **unseen data**).

Training Dataset

- The **training dataset** is a set of data used in a machine learning project, and each sample is called a training sample.
- The items or attributes that reflect the performance or nature of the sample in a particular aspect are called ***features***.
- ***Learning (training) is the process of learning a model from the training data.***
- The process of using the model to make predictions is called ***testing***, and the dataset used for testing is known as the **test data** (or **unseen data**).
  - Each sample in the **test data** is called a test sample.

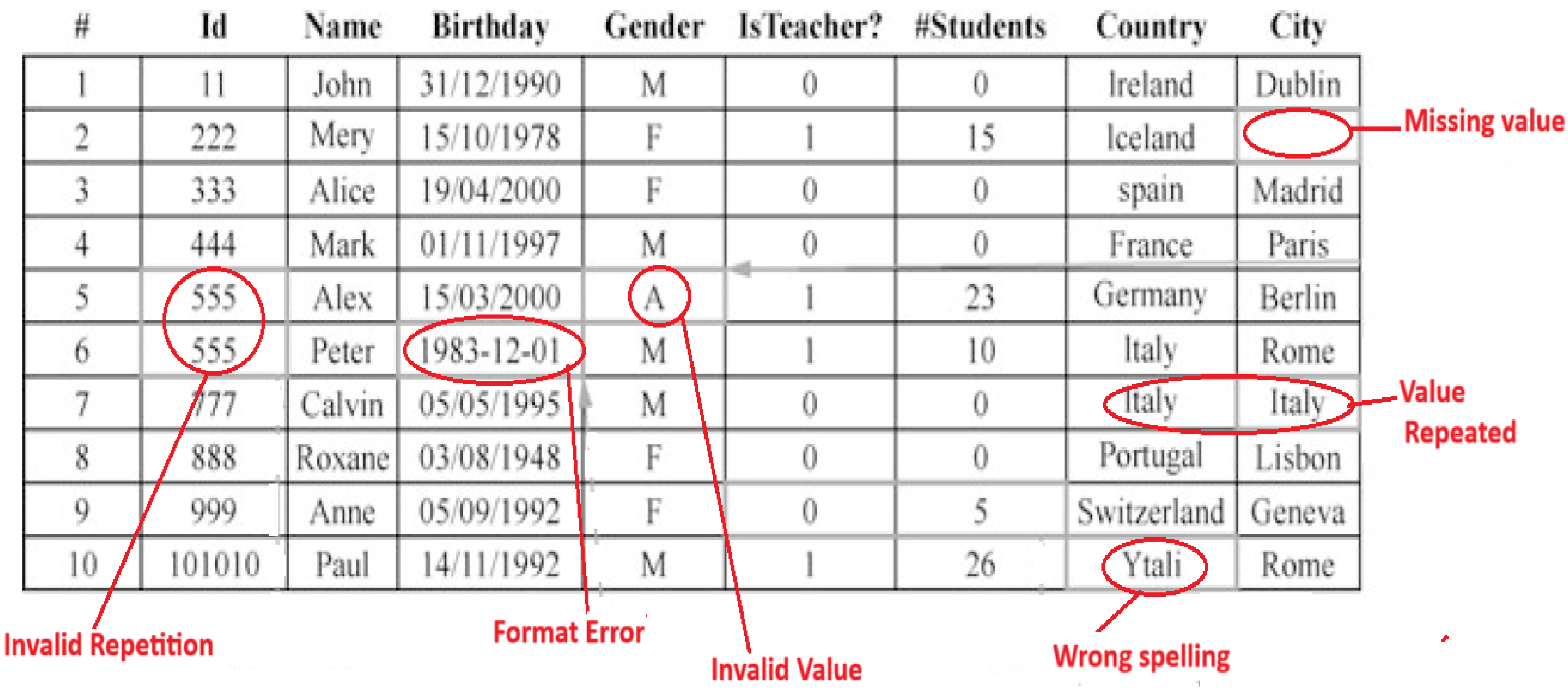
Training set and Test set



		Feature 1	Feature 2	Feature 3	Label
	Serial No.	Floor area	School district	Orientation	House price
Training set	1	100	8	South	1000
	2	120	9	Southwest	1300
	3	60	6	North	700
	4	80	9	Southeast	1100
Test set	5	95	3	South	850

Fig. 2.12 Sample dataset

Corrupted or “Dirty” Data



#	Id	Name	Birthday	Gender	IsTeacher?	#Students	Country	City
1	11	John	31/12/1990	M	0	0	Ireland	Dublin
2	222	Mery	15/10/1978	F	1	15	Iceland	
3	333	Alice	19/04/2000	F	0	0	spain	Madrid
4	444	Mark	01/11/1997	M	0	0	France	Paris
5	555	Alex	15/03/2000	A	1	23	Germany	Berlin
6	555	Peter	1983-12-01	M	1	10	Italy	Rome
7	777	Calvin	05/05/1995	M	0	0	Italy	Italy
8	888	Roxane	03/08/1948	F	0	0	Portugal	Lisbon
9	999	Anne	05/09/1992	F	0	5	Switzerland	Geneva
10	101010	Paul	14/11/1992	M	1	26	Ytali	Rome

Missing value

Value Repeated

Invalid Repetition

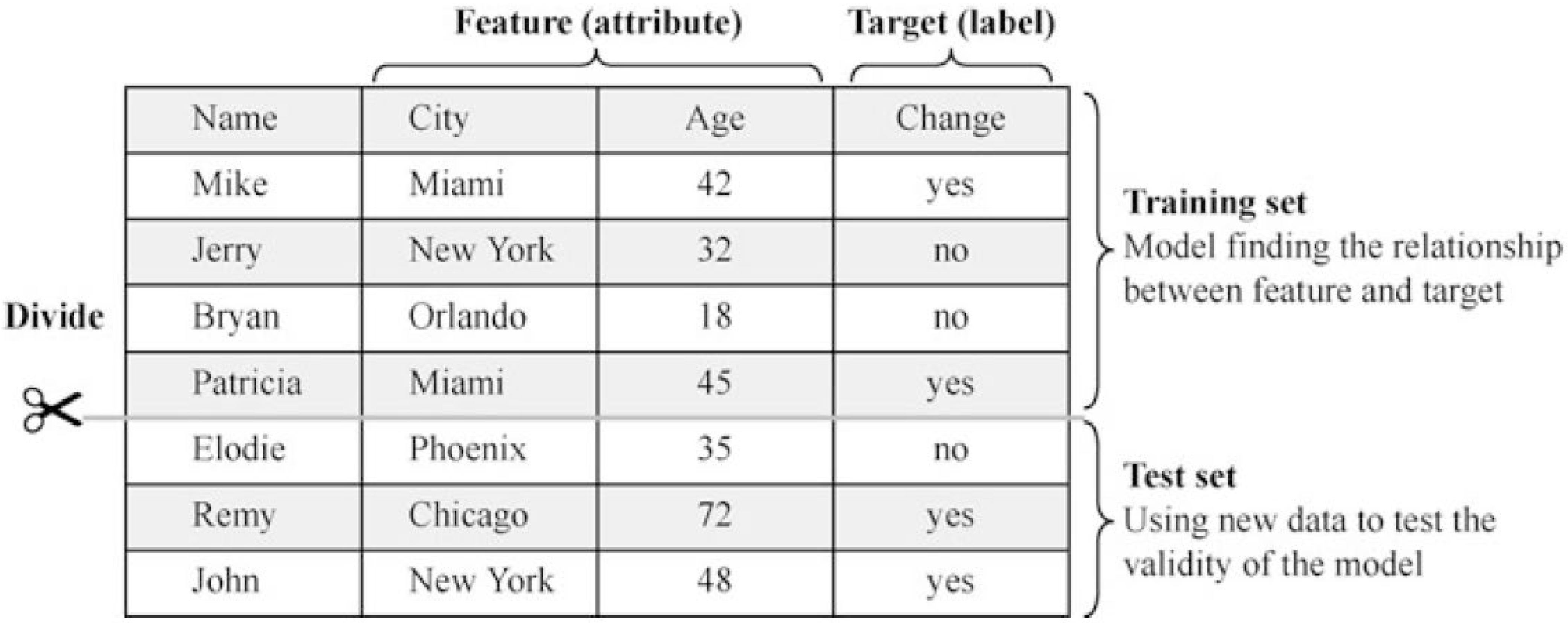
Format Error

Invalid Value

Wrong spelling

Fig. 2.13 “Dirty” data

Training set and Test set



Feature (attribute)

Target (label)

Name	City	Age	Change
Mike	Miami	42	yes
Jerry	New York	32	no
Bryan	Orlando	18	no
Patricia	Miami	45	yes
Elodie	Phoenix	35	no
Remy	Chicago	72	yes
John	New York	48	yes

Divide

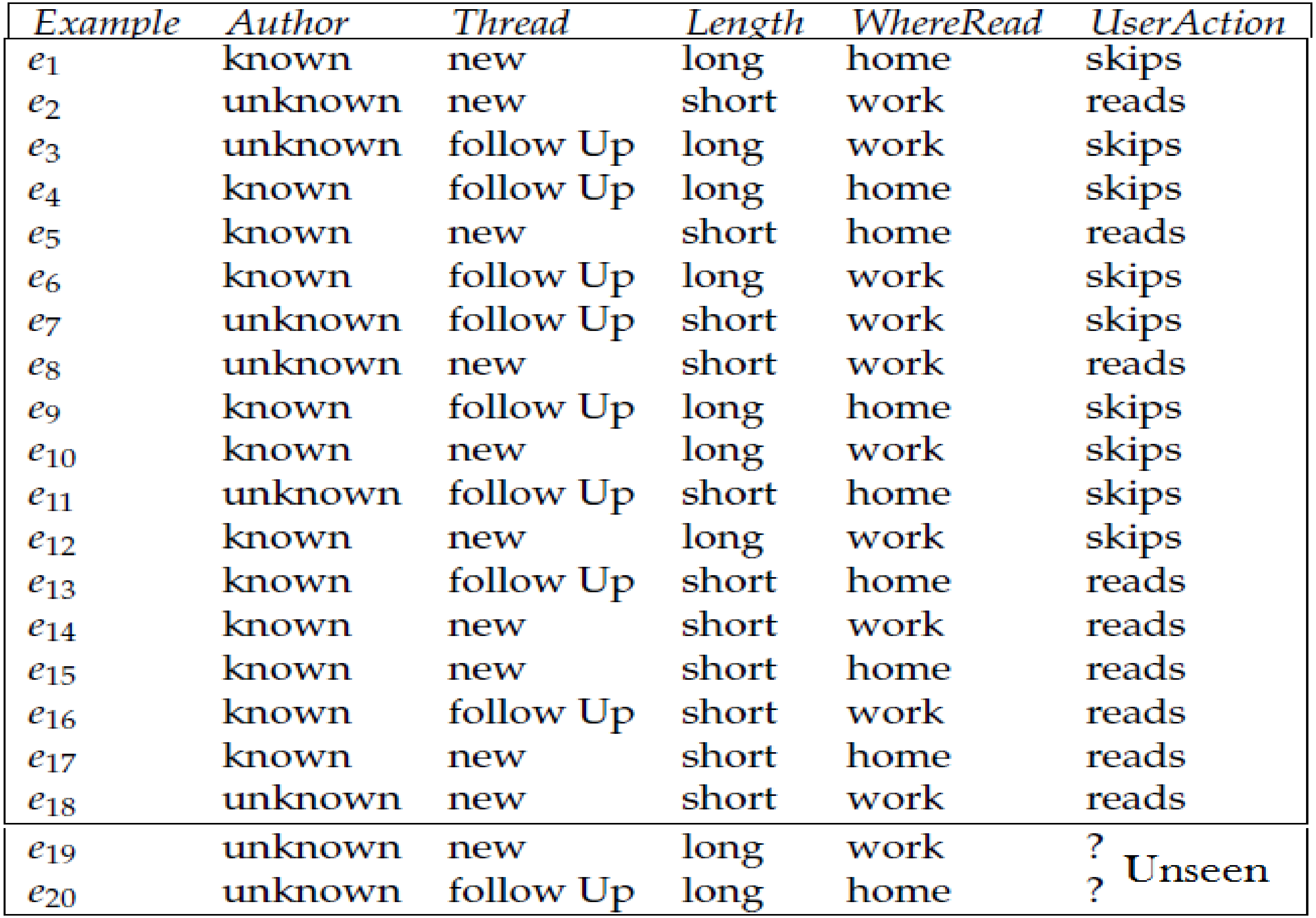
Training set
Model finding the relationship between feature and target

Test set
Using new data to test the validity of the model

Fig. 2.19 Training set and test set

Supervised Learning

- **Example 7.1:** **Figure 7.1** shows a non-numerical **training** and **test example** of a classification task. The aim is *to predict whether a person reads an article posted to a threaded discussion website or not*
    - The **input features** are *Author, Thread, Length, and WhereRead.* There is only one **target feature**, *UserAction.*
    - The domain of ***Author*** is **{***known*, *unknown***}**, the domain of ***Thread*** is **{***new*, *followup***}**, and so on.
  - There are **eighteen seen data** (**e_1**,….,**e_{18}**).
    - In this dataset, *Author*(e_{11})=*unknown, Thread(e_{11})= followUp,* and *UserAction*(e_{11}) = *skips.*
  - There are two **test examples**, **e_{19}** and **e_{20}**, where the ***userAction*** is unknown.
-



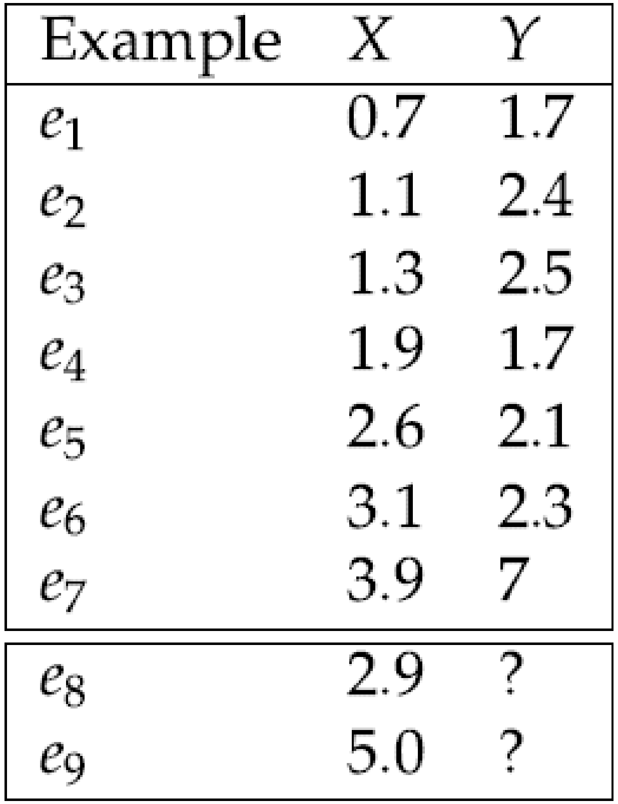
Example	*Author*	*Thread*	*Length*	*WhereRead*	*UserAction*	
e_1	known	new	long	home	skips	
e_2	unknown	new	short	work	reads	
e_3	unknown	follow Up	long	work	skips	
e_4	known	follow Up	long	home	skips	
e_5	known	new	short	home	reads	
e_6	known	follow Up	long	work	skips	
e_7	unknown	follow Up	short	work	skips	
e_8	unknown	new	short	work	reads	
e_9	known	follow Up	long	home	skips	
e_{10}	known	new	long	work	skips	
e_{11}	unknown	follow Up	short	home	skips	
e_{12}	known	new	long	work	skips	
e_{13}	known	follow Up	short	home	reads	
e_{14}	known	new	short	work	reads	
e_{15}	known	new	short	home	reads	
e_{16}	known	follow Up	short	work	reads	
e_{17}	known	new	short	home	reads	
e_{18}	unknown	new	short	work	reads	
e_{19}	unknown	new	long	work	?	Unseen
e_{20}	unknown	follow Up	long	home	?	

Figure 7.1: Examples of a user's preferences

Supervised Learning

- **Example 7.2:** **Figure 7.2** shows some data for a **linear regression task**, where the aim is to predict the value of feature ***Y*** on examples for which the value of feature ***X*** is provided
  - This is a **regression task** because ***Y*** is a **real-valued feature**.
  - Predicting a value of ***Y,*** for example, **$e_{8,}$** is an **interpolation problem**, *as its value for the* ***input feature*** *is* ***between the values*** *of the training examples*.
  - Predicting a value of ***Y*** for the example **e_9** is an **extrapolation problem** because its ***X*** value is **outside the range** of the **training examples**.

Supervised Learning



Example	X	Y
e_1	0.7	1.7
e_2	1.1	2.4
e_3	1.3	2.5
e_4	1.9	1.7
e_5	2.6	2.1
e_6	3.1	2.3
e_7	3.9	7
e_8	2.9	?
e_9	5.0	?

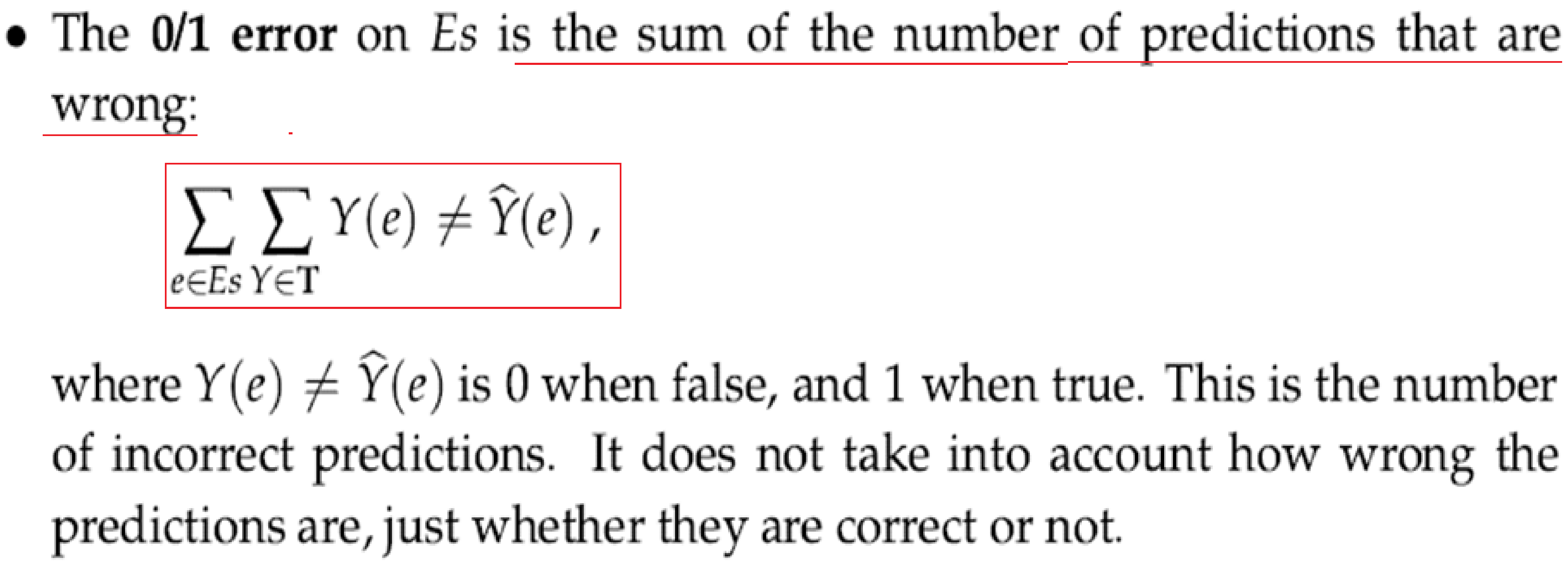
Figure 7.2: Training and test examples for a regression task

Evaluating Predictions

- Evaluating a **training prediction** in supervised learning is based on the **error parameter (ES)** generated for each training sample.
- A **point estimate** *for target feature* ***Y*** on example **e** is a prediction of the value of ***Y*(*e*)** called the ***desired output***.
- Let ***Y'*(*e*)** be the **predicted** or ***calculated output*** for the same target feature on the example ***e***.
- The **error** (***Es***) on this feature is a measure of how close they are to each other**,** ***Es*** = ***Y*(*e*)** - ***Y'*(*e*)**
  - For **regression**, when the target feature ***Y*** is **real-valued**, ***Y'*(*e*)** and ***Y*(*e*)** are **real numbers** that can be compared arithmetically.
  - For **classification**, when the target feature ***Y*** is a **discrete** value (*true* or *false*, *good* or *bad*, etc.), there are several alternatives.

Evaluating Predictions

- In a supervised learning system, the **error**, ***Es*** values, is an important feature to ***control/terminate*** the dataset training process.
- Some of the **error-based methods** that are used to control/terminate the training process of a learning system are described below:



- The **0/1 error** on *Es* is the sum of the number of predictions that are wrong:

$$\boxed{\sum_{e\in Es}\sum_{Y\in \mathrm{T}} Y(e) \neq \widehat{Y}(e)\,,}$$

where $Y(e) \neq \widehat{Y}(e)$ is 0 when false, and 1 when true. This is the number of incorrect predictions. It does not take into account how wrong the predictions are, just whether they are correct or not.

Evaluating Predictions

- The **absolute error** on *Es* is the sum of the absolute differences between the actual and predicted values on each example:

$$\boxed{\sum_{e\in Es}\sum_{Y\in \mathrm{T}}\left|Y(e)-\widehat{Y}(e)\right|.}$$

  This is always non-negative, and is only zero when all the predictions exactly fit the observed values. Unlike for the 0/1 error, close predictions are better than far-away predictions.

- The **sum-of-squares error** on *Es* is

$$\boxed{\sum_{e\in Es}\sum_{Y\in \mathrm{T}}(Y(e)-\widehat{Y}(e))^2.}$$

  This measure treats large errors as much worse than small errors.

Evaluating Predictions

- The **sum-of-squares error (SOSE)** example:

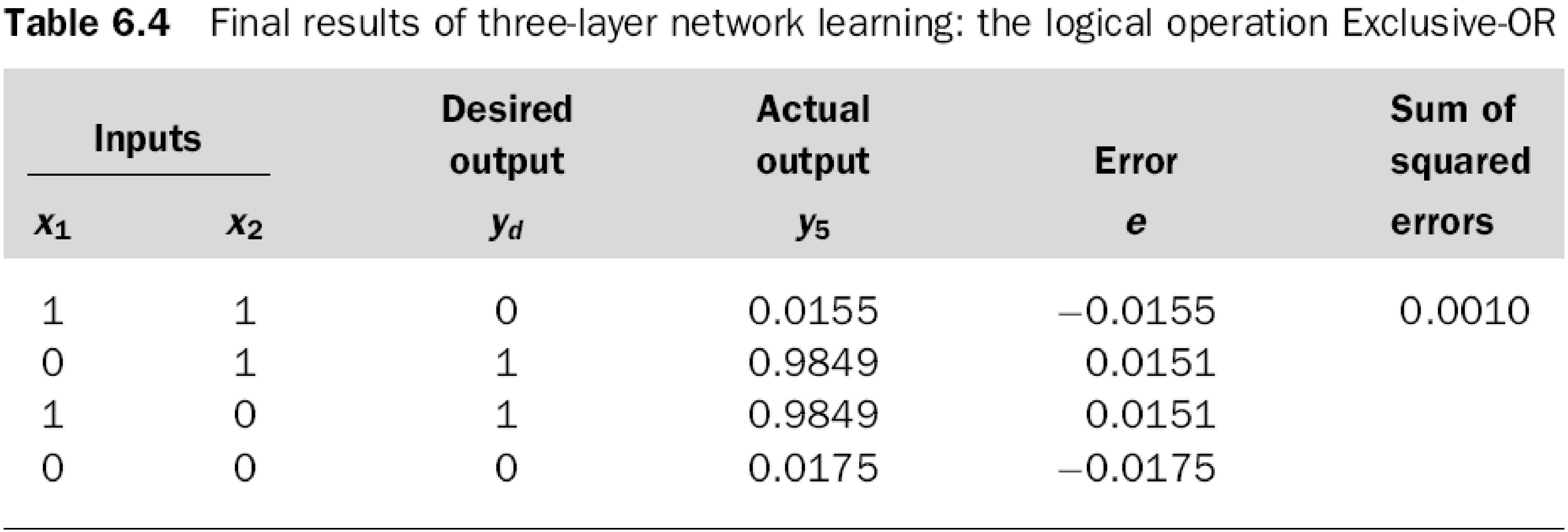


Table 6.4 Final results of three-layer network learning: the logical operation Exclusive-OR

Inputs		Desired output	Actual output	Error	Sum of squared errors
x_1	x_2	y_d	y_5	e	
1	1	0	0.0155	–0.0155	0.0010
0	1	1	0.9849	0.0151	
1	0	1	0.9849	0.0151	
0	0	0	0.0175	–0.0175	

Evaluating Predictions

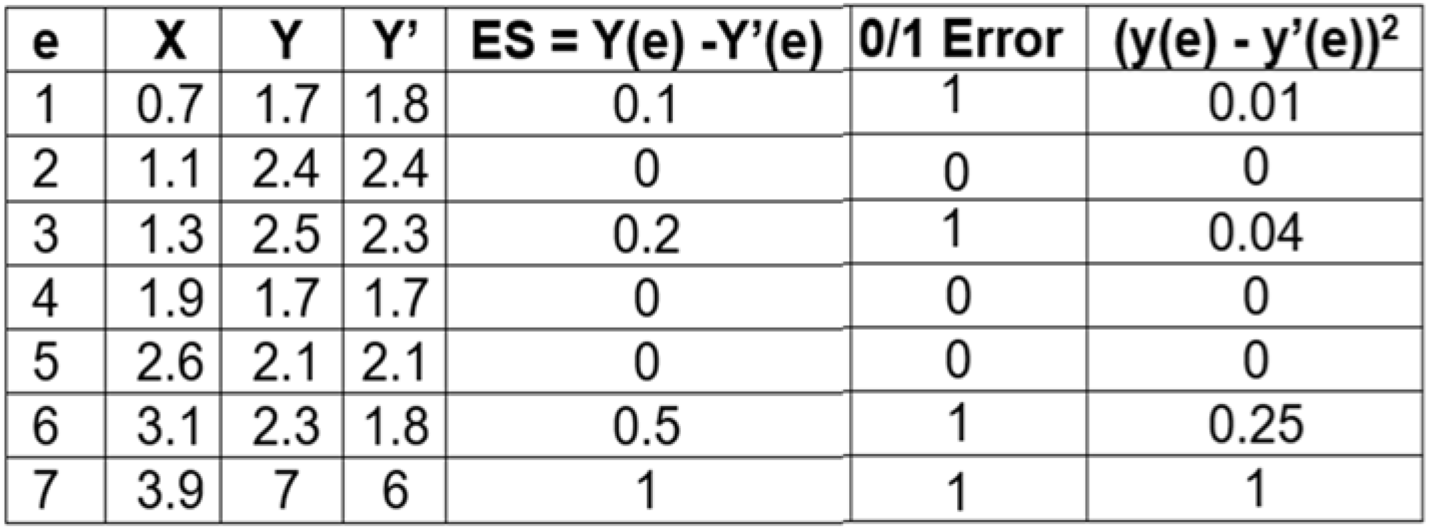
- Minimizing the **sum-of-squares error** (**SOSE**) is equivalent to minimizing the **Root-Mean-Square Error (RMSE)**, obtained by dividing by the number of examples and taking the square root:

$$RMSE = \sqrt{{}^{1}\!/_{N} \times \sum_{k=1}^{N} (y_k - y'_k)^2}$$

  - Where $\boldsymbol{y_k}$ is the actual output value, ***y*** is the calculated/predicted output value, and ***N*** is the **total number of data samples**

Evaluating Predictions

- The results of the various ***average error value*** calculations of the linear regression data in **Figure 7.2** are given below:
  - **0/1 Error = 4**
  - **Absolute Error = 1.8**
  - **Sum-of-Square Error = 1.3**
  - **Root Mean Square Error = 0.431**



e	X	Y	Y’	ES = Y(e) -Y’(e)	0/1 Error	(y(e) - y’(e))^2
1	0.7	1.7	1.8	0.1	1	0.01
2	1.1	2.4	2.4	0	0	0
3	1.3	2.5	2.3	0.2	1	0.04
4	1.9	1.7	1.7	0	0	0
5	2.6	2.1	2.1	0	0	0
6	3.1	2.3	1.8	0.5	1	0.25
7	3.9	7	6	1	1	1

Entropy

- The **entropy (Information)** of the data is the ***number of bits*** it will take to **encode** the data, given a code based on the predicted output ***Y'(e),*** which is treated as a probability.
- The **entropy** of ***Y'*(e)** is given as:

$$= -\sum_{e \in E} \sum_{Y \in T} [y(e) \log y'(e) + (1 - y(e) \log(1 - y'(e)))]$$

  - A better prediction is one with **<u>lower entropy</u>**.
  - *A prediction that **minimizes the entropy** is a prediction that maximizes the **likelihood*** (similarity).
  - Where ***Y*(e) is the actual output** of example **e***,* and ***Y'*(e)** is the **predicted output** of example ***e***.

Entropy

- To understand the notion of **Information (Entropy)**, consider the following statement;

  “*Whether a coin will come up heads*” ⇒The amount of **information** contained in the answer depends on **one’s prior knowledge**

  – The **information theory** measures information content in **bits**; ***one bit*** of information is enough to answer a **yes/no** question about the flip of a fair coin

  – In general, if the possible answers $\boldsymbol{v_i}$ have **probabilities $\boldsymbol{P(v_i)}$**, then the **information content I (**or **entropy)** is:

$$I(P(v_1),\ldots,P(v_n)) = -\sum_{i=1}^{n} P(v_i)\log_2 P(v_i)$$

Entropy

- For a **random variable *x*** with probability ***p*(*x*)**, the **entropy *H*** is the average (or expected) amount of **information** obtained by observing ***x***:

$$H(x) = -\Sigma_x \, p(x) \log_2 p(x)$$

- To check this equation, for the tossing of a **fair** coin (both **head** and **tail** have an equal probability of **0.5** each), we get:

$H(\text{head}) = \boldsymbol{I}(\text{p(head)} = 0.5,\ \text{p(tail)} = 0.5)$

$= -(0.5\log_2(0.5) + 0.5\log_2(0.5))$

$= \mathbf{1}$ $\quad\{\log_2(0.5) = -1$

  - That results from each coin toss delivering **one whole bit** of information.

Entropy

- However, if we know the **coin toss** is **not fair**, it comes up **heads** or **tails** with probabilities ***p*** and ***q***, where ***p* ≠ *q*.**
- Every time it is tossed, *one side is more likely to come up*.
- For example, if ***p* = 0.7**, then the entropy is:

  ***H*(head)** = ***I*(0.7, 0.3)** = $-(0.7\log_2(0.7) + 0.3\log_2(0.3))$

  $= -0.7 \times (-0.515) - 0.3 \times (-1.737)$

  = **0.8816** < 1 (on average, each toss delivers less than one full bit of information)

- **How to calculate $\log_2(Y)$ if *Y* is not a 2^n value?**

  $\log_2(Y) = \log(Y)/\log(2) = \log(Y)/0.30103$

  ; where log(2) = 0.30103

Entropy

- What is the **entropy** of getting **face six** from a **six-faced die**?

  H(6) = *I*(⅙, ⅙) = - (⅙ $\log_2$ ⅙ + ⅙ $\log_2$ ⅙) = **0.862** (each toss of the die delivers less than one full bit of information)

- *What is the **entropy** of a **four-sided** die*?

p(side1) =1/4 = 0.25, p(side2) = 0.25, p(side3) = 0.25 and p(side4) = 0.25

H(4-sided Die) = *I*(0.25, 0.25, 0.25, 0.25)

= -[0.25$\log_2$0.25 + 0.25$\log_2$0.25 + 0.25$\log_2$0.25 + 0.25$\log_2$0.25]

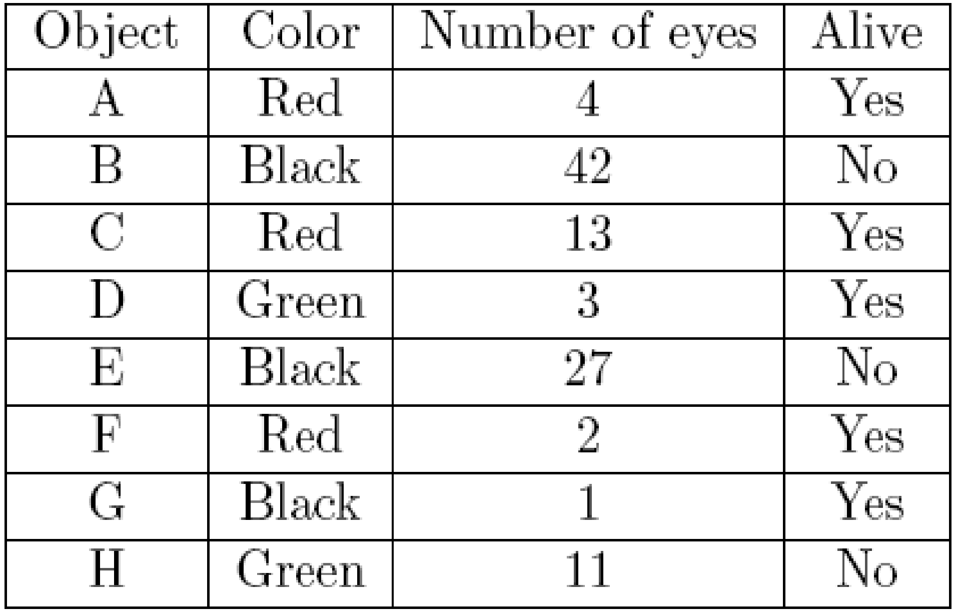
= -[(0.25 * -1.999)+(0.25 * -1.999) +(0.25 * -1.999) +(0.25 * -1.999)]

= 1.99999 = **2 bits**

- *What is the entropy of tossing a coin with a **99% head chance***?

Entropy Calculation

Your spaceship has just landed on an alien planet, and your crew has begun investigating the local wildlife. Unfortunately, most of your scientific equipment is broken, so all you can talk about a given object is what color it is, how many eyes it has, and whether it is alive. To make matters worse, none of you are biologists, so you will have to use a **decision tree** to classify objects near your landing site as either **alive** or **not alive**.



Object	Color	Number of eyes	Alive
A	Red	4	Yes
B	Black	42	No
C	Red	13	Yes
D	Green	3	Yes
E	Black	27	No
F	Red	2	Yes
G	Black	1	Yes
H	Green	11	No

Entropy Calculation

What is the entropy of Alive ?

$$H(\text{Alive}) = H(\tfrac{5}{8}, \tfrac{3}{8}) = -\tfrac{5}{8}log_2\tfrac{5}{8} - \tfrac{3}{8}log_2\tfrac{3}{8} = 0.9544$$

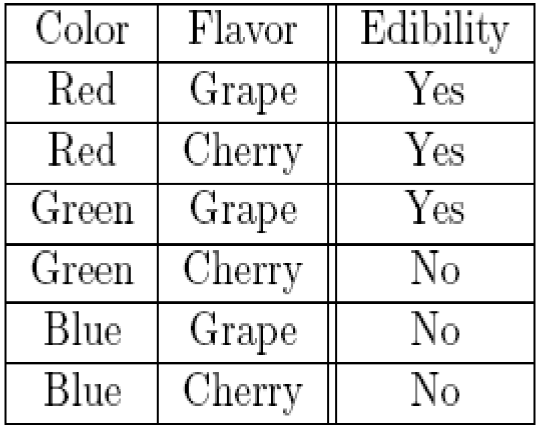
{ $\log_2 5/8 = \log_2 0.625 = -0.6781$

$\log_2 3/8 = \log_2 0.375 = -1.415037$}

2. A candy manufacturer interviews a customer on his willingness to eat a candy of a particular color or flavor. The following table shows the collected responses:

What is H(edibility)?

$$-\left(\tfrac{1}{2}log_2\tfrac{1}{2} + \tfrac{1}{2}log_2\tfrac{1}{2}\right) = 1$$



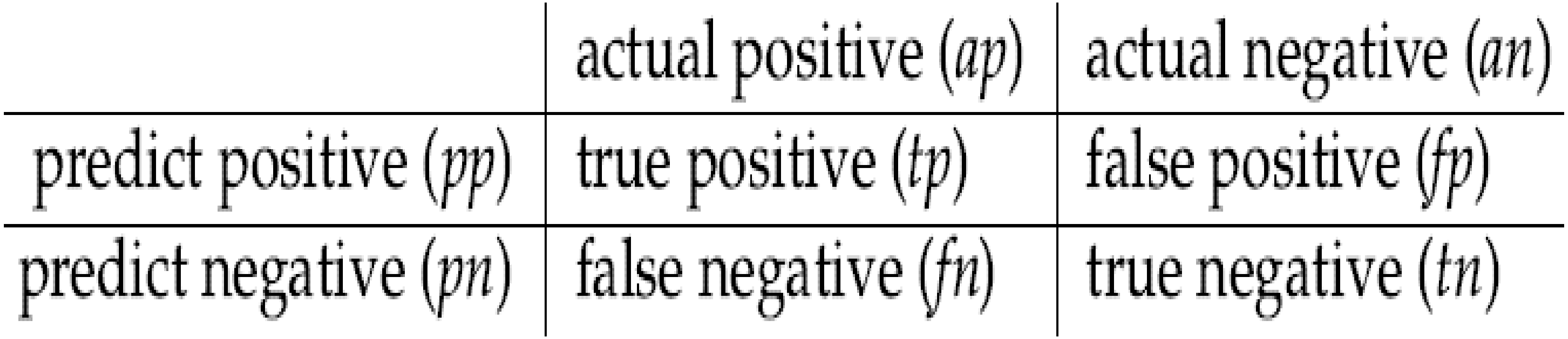
Color	Flavor	Edibility
Red	Grape	Yes
Red	Cherry	Yes
Green	Grape	Yes
Green	Cherry	No
Blue	Grape	No
Blue	Cherry	No

Types of Prediction Errors

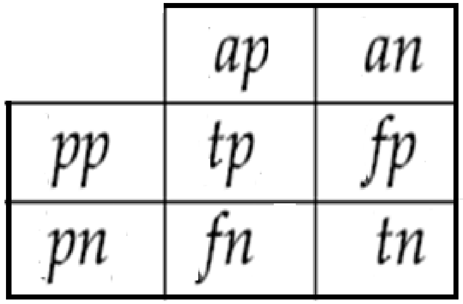
- Not all **errors** are equal; the consequences of some mistakes may be much worse than others.
  - For example, it may be ***much worse to predict that a patient does not have a disease; actually, the patient has a disease***, so the patient does not get appropriate treatment
  - Similarly, ***it is predicted that a patient has a disease; actually, the patient does not have it***, forcing the patient to undergo further unwanted tests.
- The machine learning agent should choose the **best prediction** according to the costs associated with the errors.

Types of Prediction Errors

- Consider a simple case where the domain of the ***target feature*** is **Boolean** (which we can consider as "***positive***" and "***negative***"), and the predictions are restricted to be **Boolean** (meaning **discrete**).
- One way to evaluate a prediction independently of the decision is to consider the **four cases** between the ***predicted value*** and the ***actual value***:



	actual positive (*ap*)	actual negative (*an*)
predict positive (*pp*)	true positive (*tp*)	false positive (*fp*)
predict negative (*pn*)	false negative (*fn*)	true negative (*tn*)



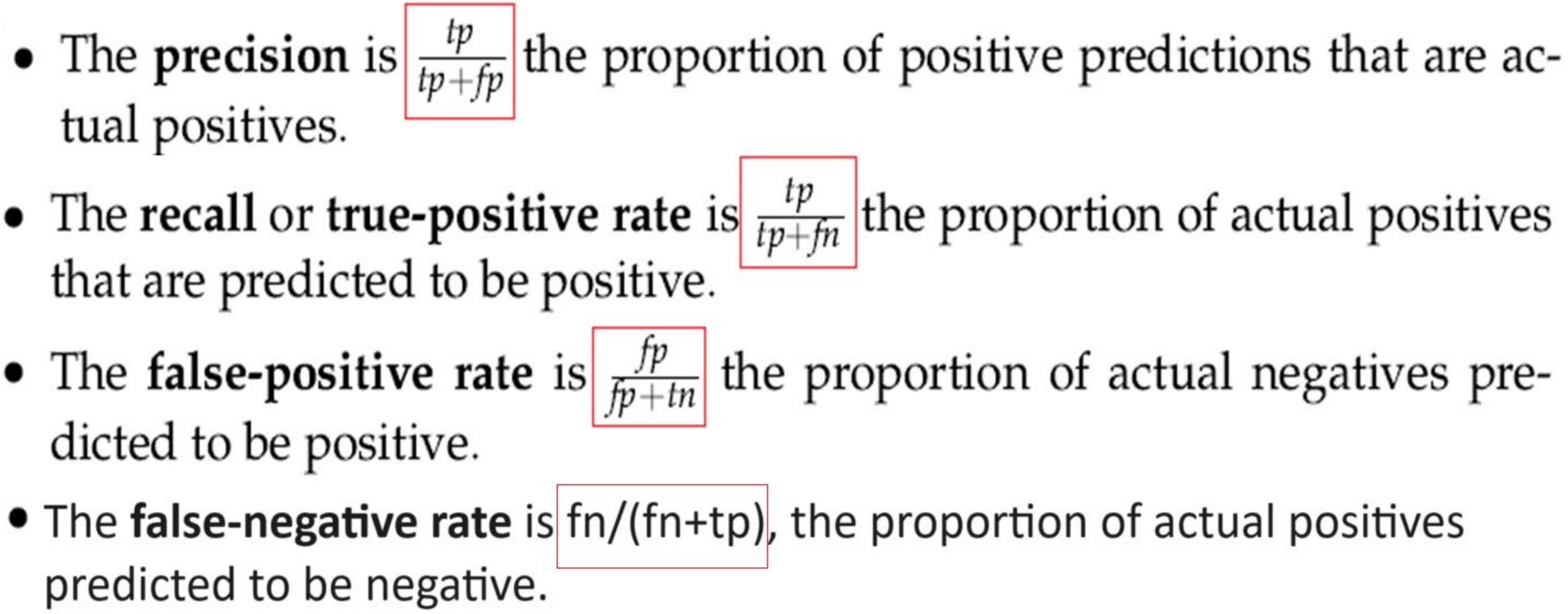
	ap	*an*
pp	*tp*	*fp*
pn	*fn*	*tn*

Types of Prediction Errors

- A **false-positive (f_p) error** or **type-I error** is a ***positive prediction that is wrong*** (i.e., the predicted value is ***true,*** but the actual value is ***false***).
- **false-negative (f_n) error** or **type-II error** is a ***negative prediction that is wrong*** (i.e., the predicted value is ***false,*** but the actual value is ***true***)
  - A **predictor** or **predicting agent** could claim a **positive prediction,** for example, when it is sure it is **positive.**
  - At the other extreme, it could claim a **positive prediction** for an example unless it is sure the example is **negative**.

Types of Prediction Errors

- For a given **predictor** for a given set of examples, suppose $\boldsymbol{t_p}$ is the number of ***true positives***, $\boldsymbol{f_p}$ is the number of ***false positives***, $\boldsymbol{f_n}$ is the number of ***false negatives***, and $\boldsymbol{t_n}$ is the number of ***true negatives***. <u>**The following measures are often used**</u>:



- The **precision** is $\boxed{\frac{tp}{tp+fp}}$ the proportion of positive predictions that are actual positives.
- The **recall** or **true-positive rate** is $\boxed{\frac{tp}{tp+fn}}$ the proportion of actual positives that are predicted to be positive.
- The **false-positive rate** is $\boxed{\frac{fp}{fp+tn}}$ the proportion of actual negatives predicted to be positive.
- The **false-negative rate** is $\boxed{fn/(fn+tp)}$, the proportion of actual positives predicted to be negative.

Types of Prediction Errors

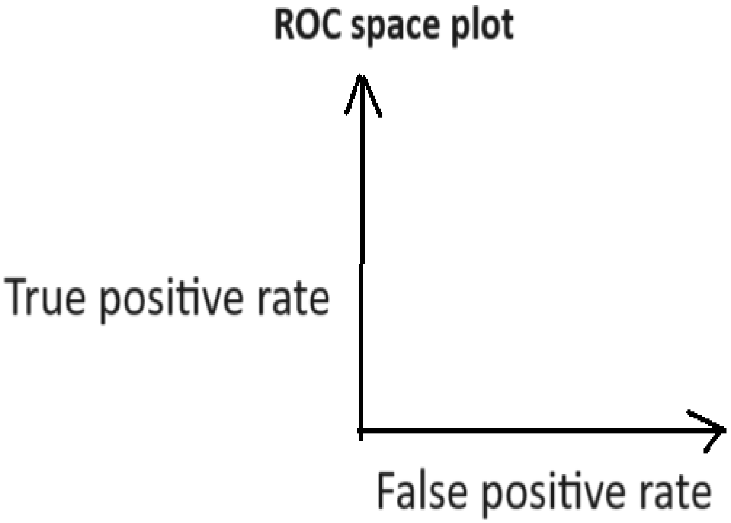
- A learning agent should try to maximize **precision** and **recall** and minimize the **false-positive rate** and **false-negative rate**
- An agent can **maximize precision** and minimize the **false-positive rate** by only making **positive predictions** that it is sure about.
- The following measurements are also used for **prediction evaluation**:

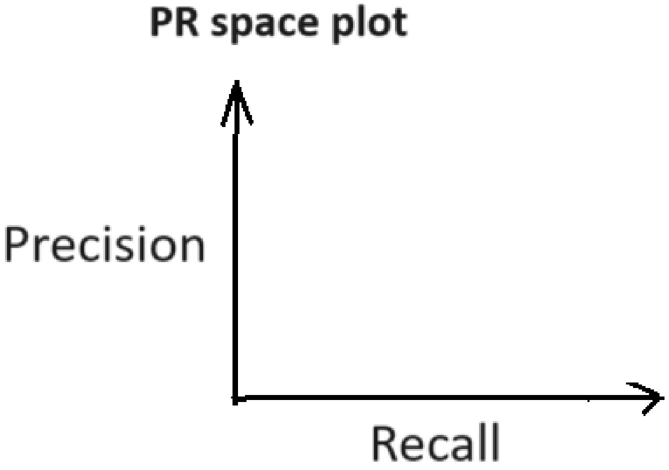
$$\boxed{\text{F1-score} = \frac{(2 \times \text{precision} \times \text{recall})}{(\text{precision} + \text{recall})}}$$

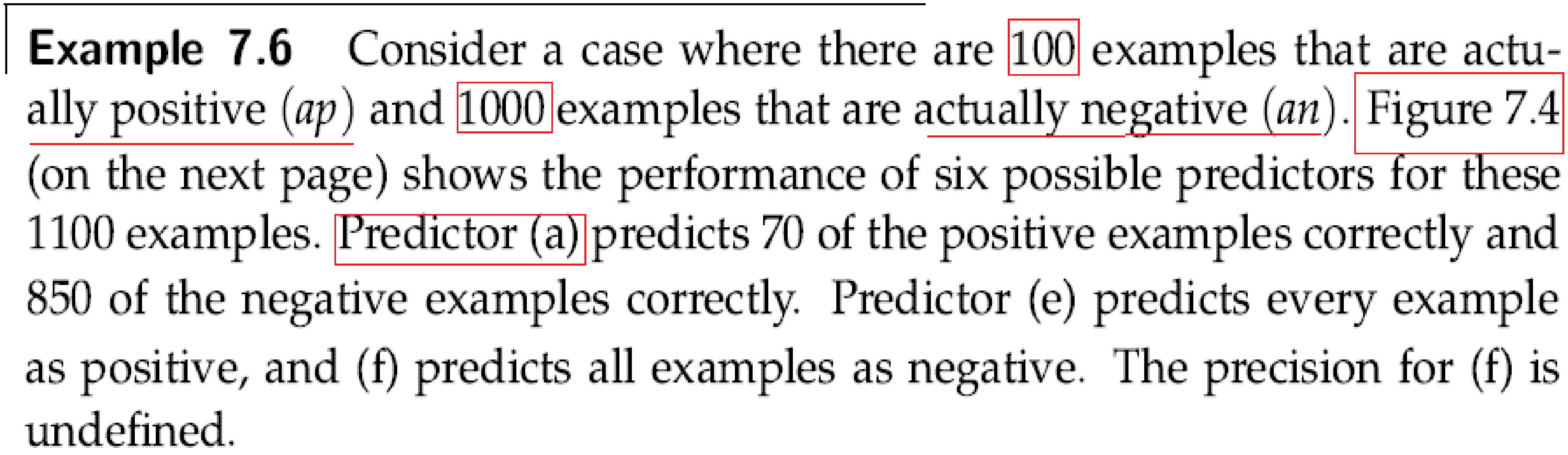
$$\boxed{\text{Accuracy} = \frac{tp + tn}{tp + tn + fp + fn}}$$

Types of Prediction Errors

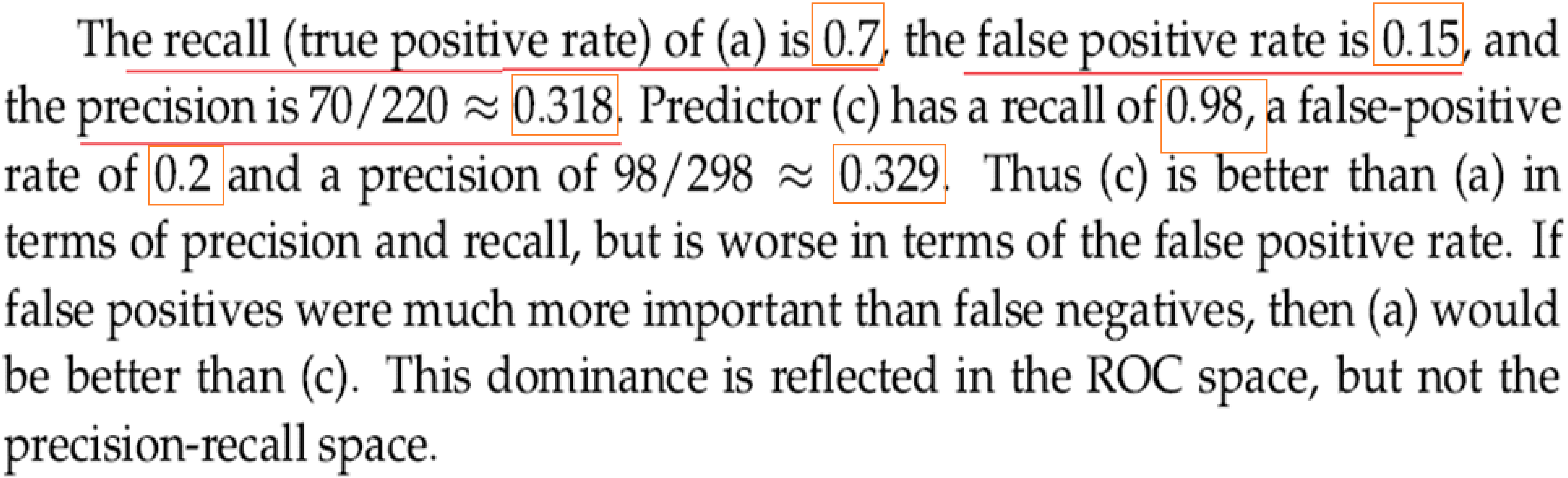
- **Receiver Operating Characteristic space plots** (**ROC space plots**) map the *false-positive rate (fp)* against the *true-positive rate (tp).*
  - Each predictor for these examples becomes a point in the space.
- A **Precision-Recall (PR) space plot** maps the *precision* against the *recall*.
  - Each approach may be used to compare **learning algorithms** independently of the actual costs of the **prediction errors**.





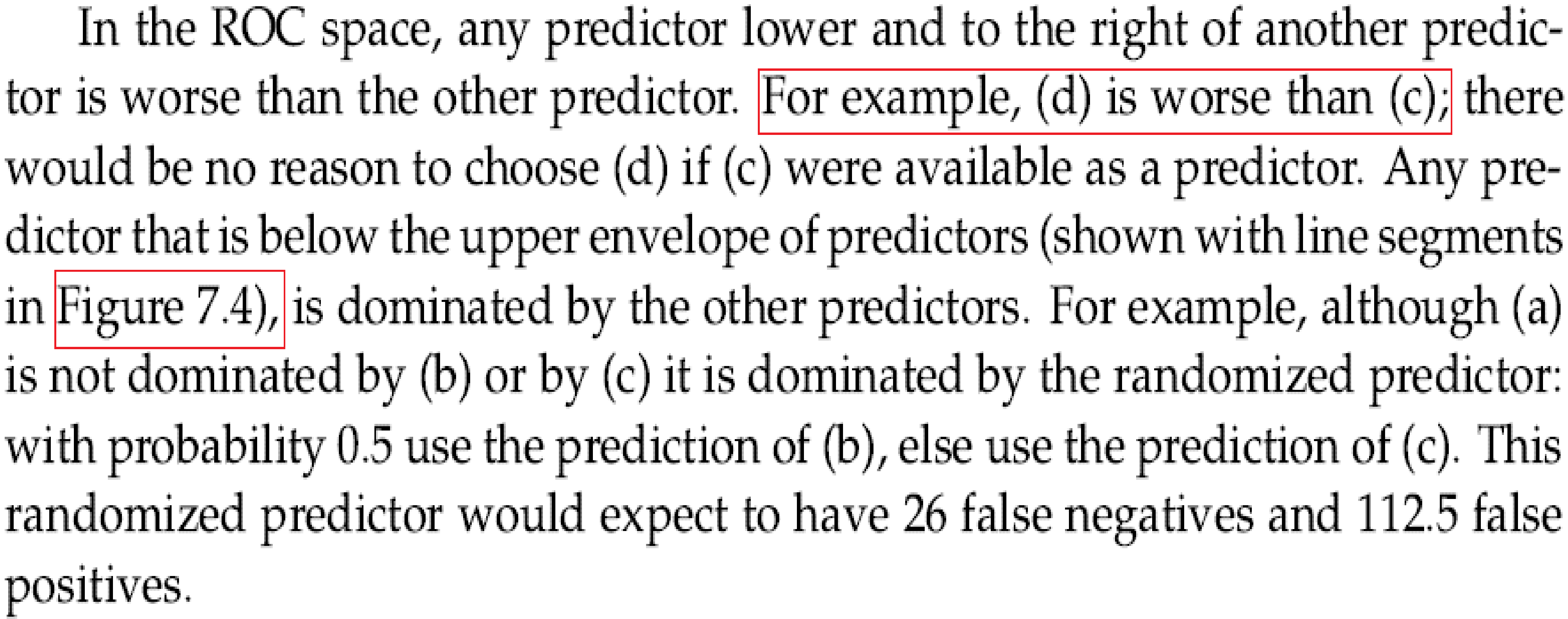


Example 7.6 Consider a case where there are 100 examples that are actually positive (*ap*) and 1000 examples that are actually negative (*an*). Figure 7.4 (on the next page) shows the performance of six possible predictors for these 1100 examples. Predictor (a) predicts 70 of the positive examples correctly and 850 of the negative examples correctly. Predictor (e) predicts every example as positive, and (f) predicts all examples as negative. The precision for (f) is undefined.

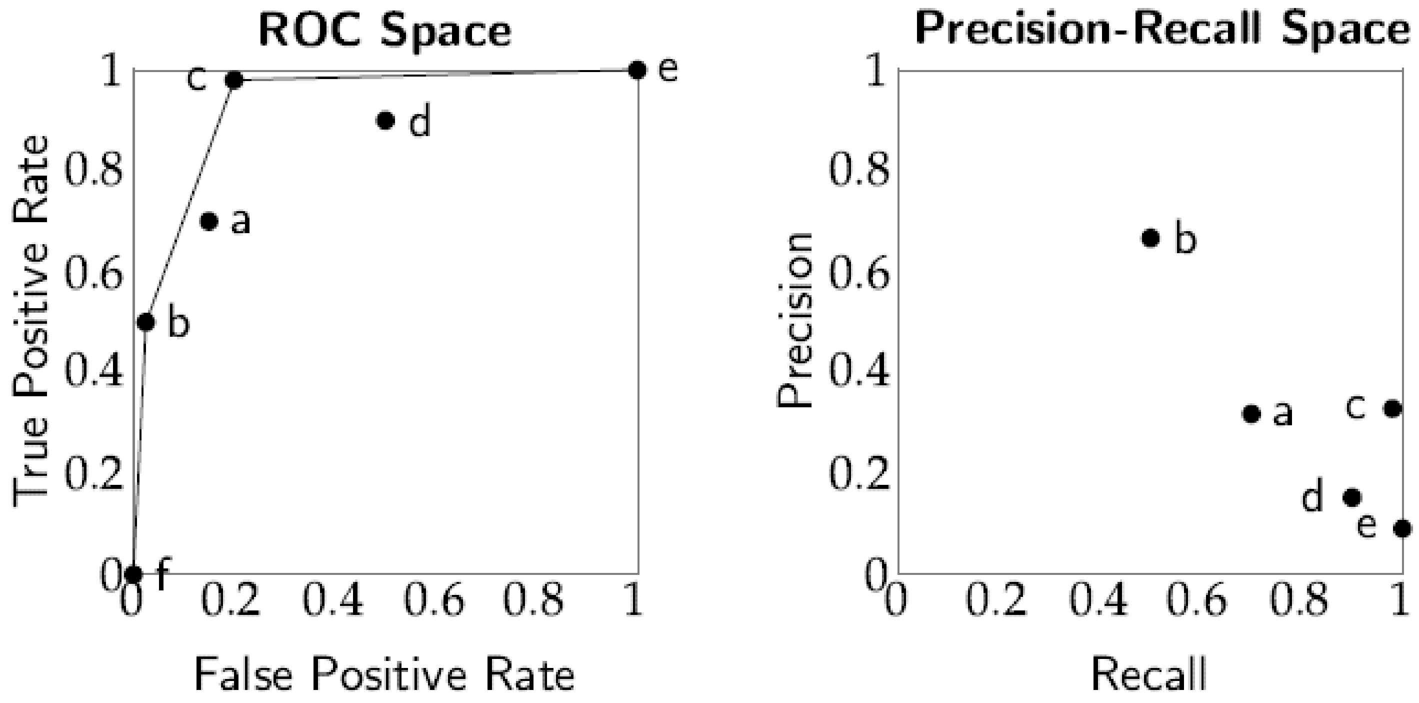


The recall (true positive rate) of (a) is 0.7, the false positive rate is 0.15, and the precision is $70/220 \approx 0.318$. Predictor (c) has a recall of 0.98, a false-positive rate of 0.2 and a precision of $98/298 \approx 0.329$. Thus (c) is better than (a) in terms of precision and recall, but is worse in terms of the false positive rate. If false positives were much more important than false negatives, then (a) would be better than (c). This dominance is reflected in the ROC space, but not the precision-recall space.

Types of Prediction Errors



In the ROC space, any predictor lower and to the right of another predictor is worse than the other predictor. For example, (d) is worse than (c); there would be no reason to choose (d) if (c) were available as a predictor. Any predictor that is below the upper envelope of predictors (shown with line segments in Figure 7.4), is dominated by the other predictors. For example, although (a) is not dominated by (b) or by (c) it is dominated by the randomized predictor: with probability 0.5 use the prediction of (b), else use the prediction of (c). This randomized predictor would expect to have 26 false negatives and 112.5 false positives.



(a)	*ap*	*an*
pp	70	150
pn	30	850

(b)	*ap*	*an*
pp	50	25
pn	50	975

(c)	*ap*	*an*
pp	98	200
pn	2	800

(d)	*ap*	*an*
pp	90	500
pn	10	500

(e)	*ap*	*an*
pp	100	1000
pn	0	0

(f)	*ap*	*an*
pp	0	0
pn	100	1000

Figure 7.4: Six predictors in the ROC space and the precision-recall spac

Learning Decision Trees

- A **decision tree** is a simple representation for **classifying examples**.
- **Decision tree learning** is one of the most straightforward techniques for **supervised classification learning**.
- Assume there is a single **discrete target feature** called the **classification**. Each element of the domain of the classification is called a **class**.
- A **decision tree** or a **classification tree** is a tree in which
  - each internal (non-leaf) node is labeled with a **condition**, a **Boolean function** of examples
  - each internal node has **two children;** one labeled with **true** and the other with **false**
  - each tree leaf is labeled with a point estimate on the class.
- A decision tree corresponds to a **nested if-else** structure in a programming language.

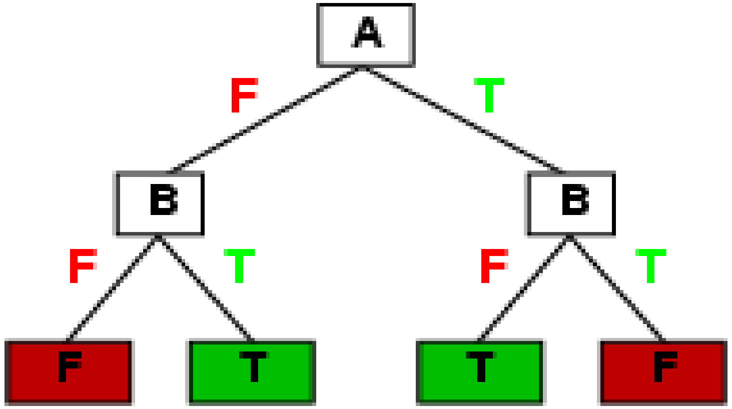
Learning Decision Trees

- To classify an example, filter it down the tree as follows:
  - The tree is constructed by recursively splitting the data into smaller subsets.
  - based on the feature that provides the most **information gain (entropy)**.
  - The result is a tree with ***decision nodes*** and ***leaf nodes***.
  - The **decision nodes** contain a question or a test on a feature.
  - The **leaf nodes** represent the ***final decision*** or the ***predicted outcome***.

Learning Decision Trees

- Decision trees can express any function of the input attributes.
- E.g., for Boolean functions, truth table row → path to leaf:

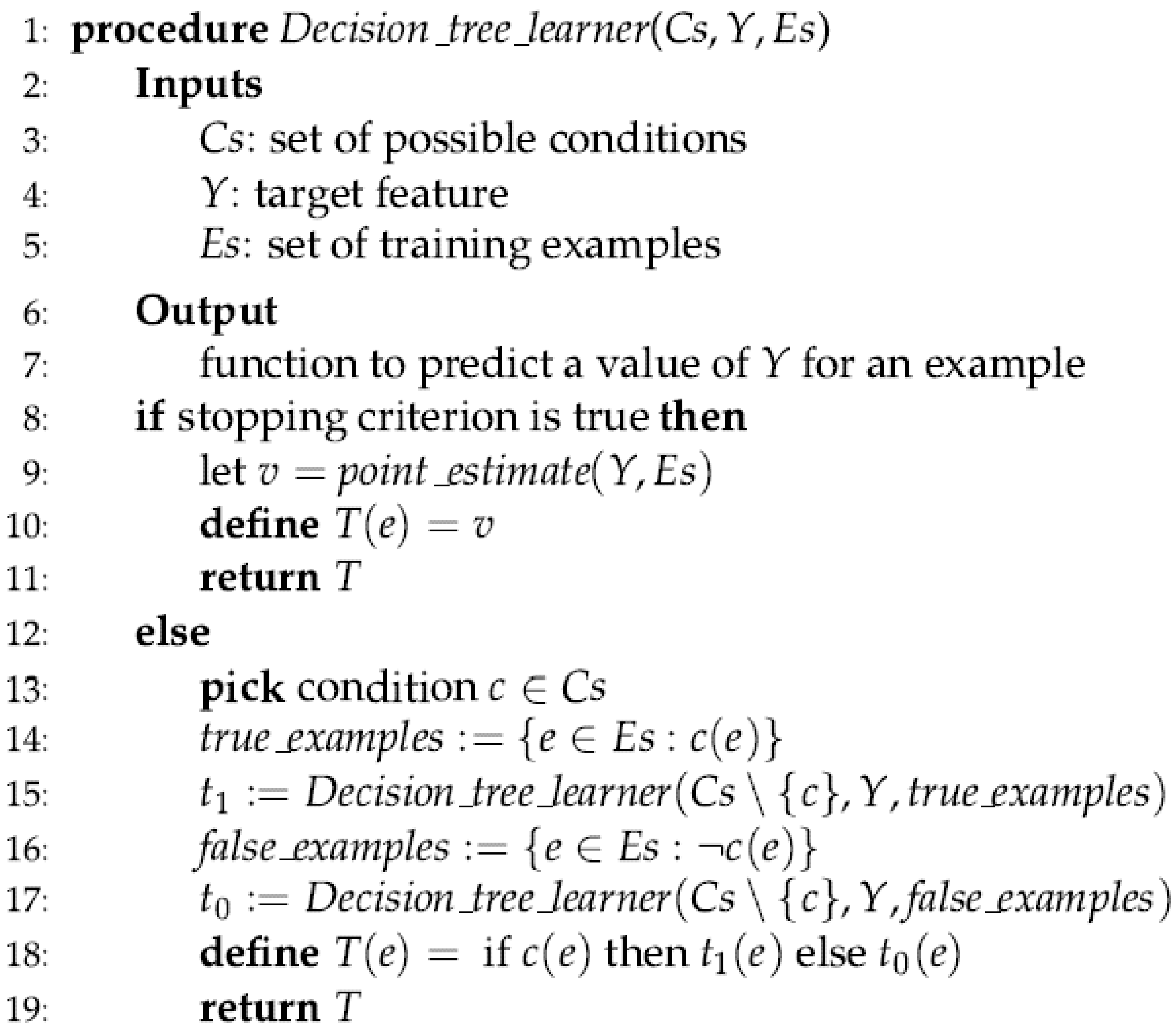
A	B	A xor B
F	F	F
F	T	T
T	F	T
T	T	F



- Trivially, there is a consistent decision tree for any training set with one path to leaf for each example, but it probably won't generalize to new examples
  - Prefer to find more compact decision trees

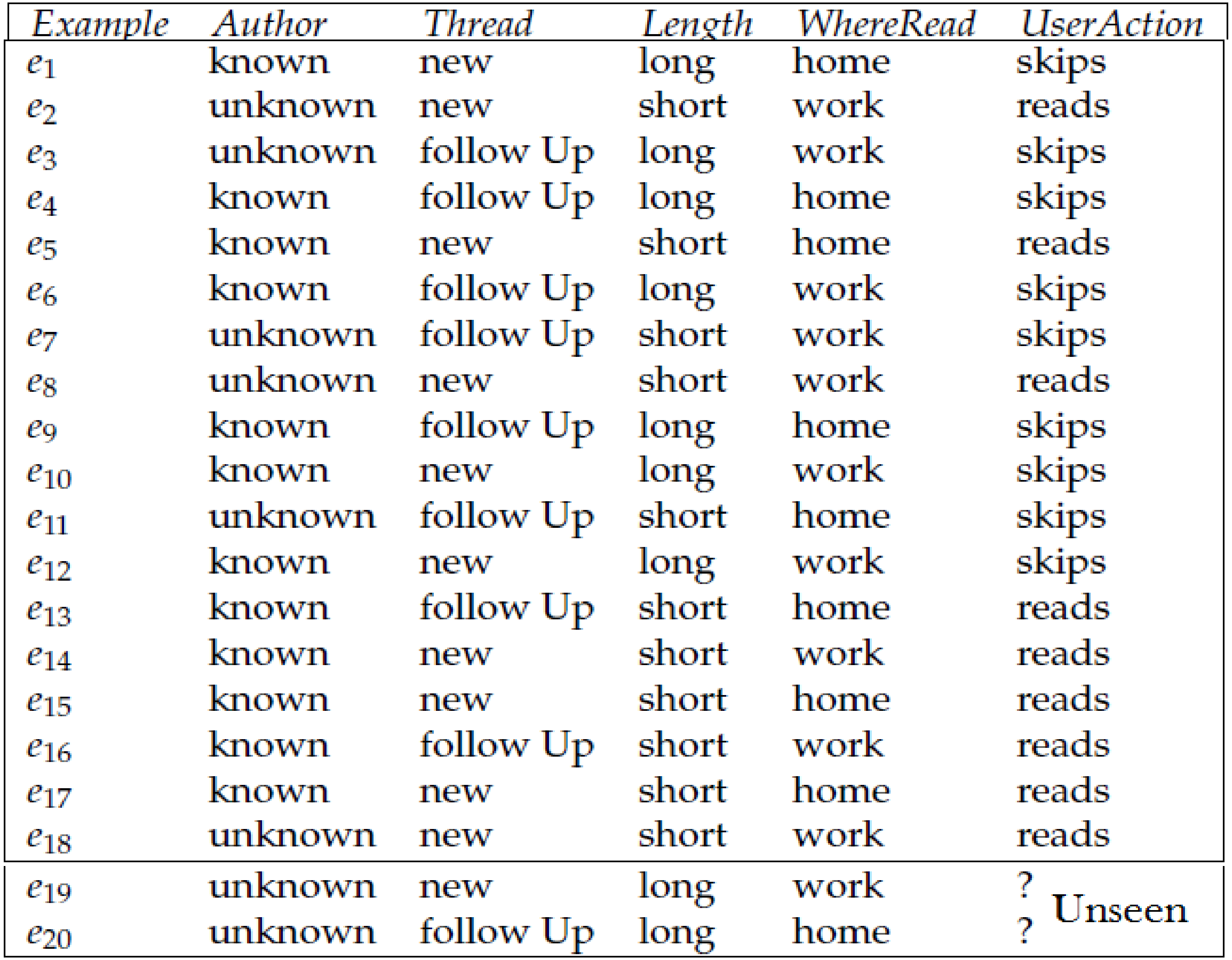
Learning Decision Trees

- The algorithm **Decision tree learner** of **Figure 7.7** builds a decision tree from the top down as follows:
  - The **input** to the algorithm is a set of **input conditions** (Boolean functions of examples that use only input features),
  - **target** feature, and **a set of training examples**.
  - If the input features are Boolean, they can be used directly as the conditions.



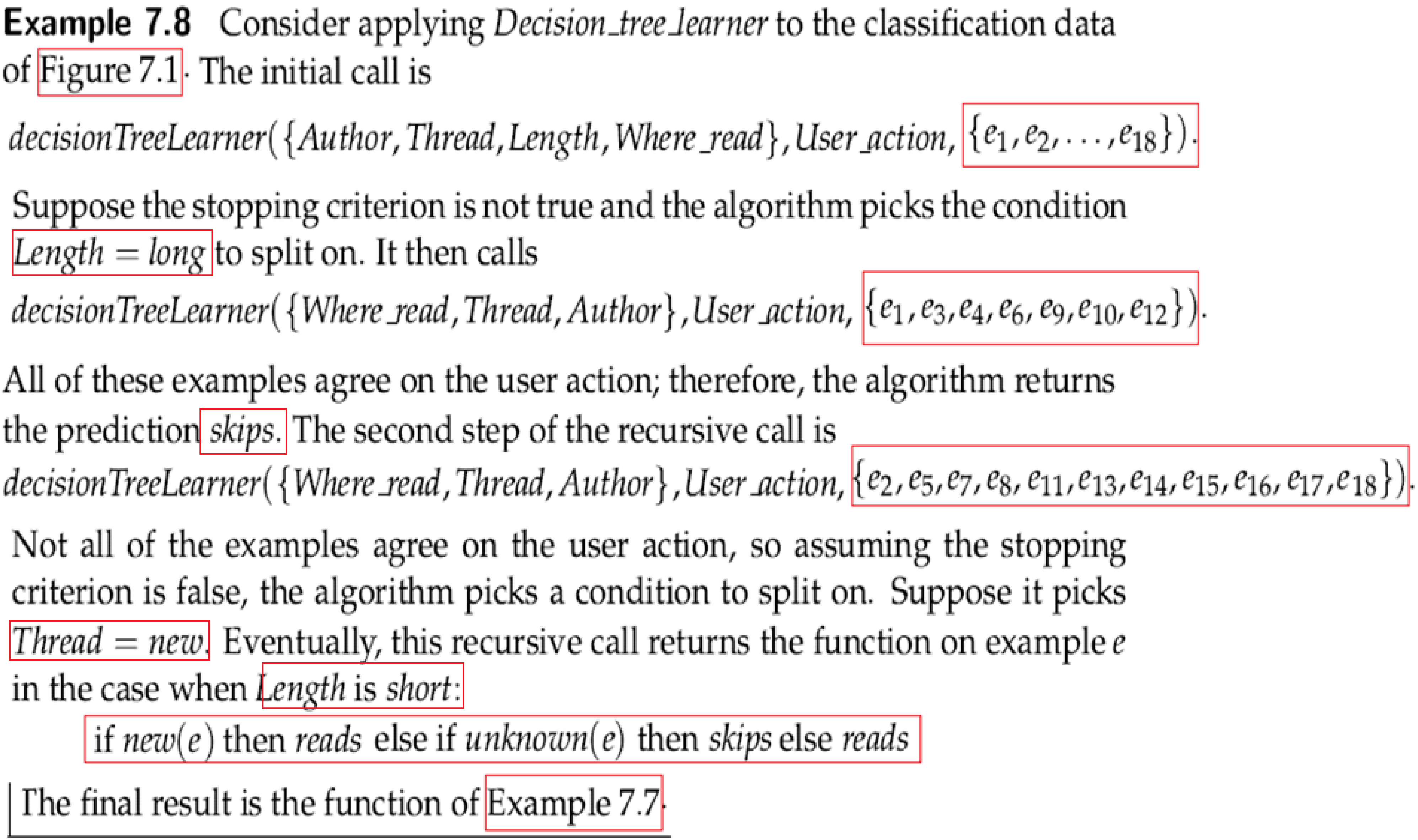
```
procedure $Decision\_tree\_learner(Cs, Y, Es)$
    Inputs
        $Cs$: set of possible conditions
        $Y$: target feature
        $Es$: set of training examples
    Output
        function to predict a value of $Y$ for an example
    if stopping criterion is true then
        let $v = point\_estimate(Y, Es)$
        define $T(e) = v$
        return $T$
    else
        pick condition $c \in Cs$
        $true\_examples := \{e \in Es : c(e)\}$
        $t_1 := Decision\_tree\_learner(Cs \setminus \{c\}, Y, true\_examples)$
        $false\_examples := \{e \in Es : \neg c(e)\}$
        $t_0 := Decision\_tree\_learner(Cs \setminus \{c\}, Y, false\_examples)$
        define $T(e) =$ if $c(e)$ then $t_1(e)$ else $t_0(e)$
        return $T$
```

Figure 7.7: Decision tree learner



Example	Author	Thread	Length	WhereRead	UserAction
e_1	known	new	long	home	skips
e_2	unknown	new	short	work	reads
e_3	unknown	follow Up	long	work	skips
e_4	known	follow Up	long	home	skips
e_5	known	new	short	home	reads
e_6	known	follow Up	long	work	skips
e_7	unknown	follow Up	short	work	skips
e_8	unknown	new	short	work	reads
e_9	known	follow Up	long	home	skips
e_{10}	known	new	long	work	skips
e_{11}	unknown	follow Up	short	home	skips
e_{12}	known	new	long	work	skips
e_{13}	known	follow Up	short	home	reads
e_{14}	known	new	short	work	reads
e_{15}	known	new	short	home	reads
e_{16}	known	follow Up	short	work	reads
e_{17}	known	new	short	home	reads
e_{18}	unknown	new	short	work	reads
e_{19}	unknown	new	long	work	? Unseen
e_{20}	unknown	follow Up	long	home	?

Figure 7.1: Examples of a user's preferences



Example 7.8 Consider applying *Decision_tree_learner* to the classification data of Figure 7.1. The initial call is

$$decisionTreeLearner(\{Author, Thread, Length, Where\_read\}, User\_action, \{e_1, e_2, \ldots, e_{18}\}).$$

Suppose the stopping criterion is not true and the algorithm picks the condition $Length = long$ to split on. It then calls

$$decisionTreeLearner(\{Where\_read, Thread, Author\}, User\_action, \{e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}\}).$$

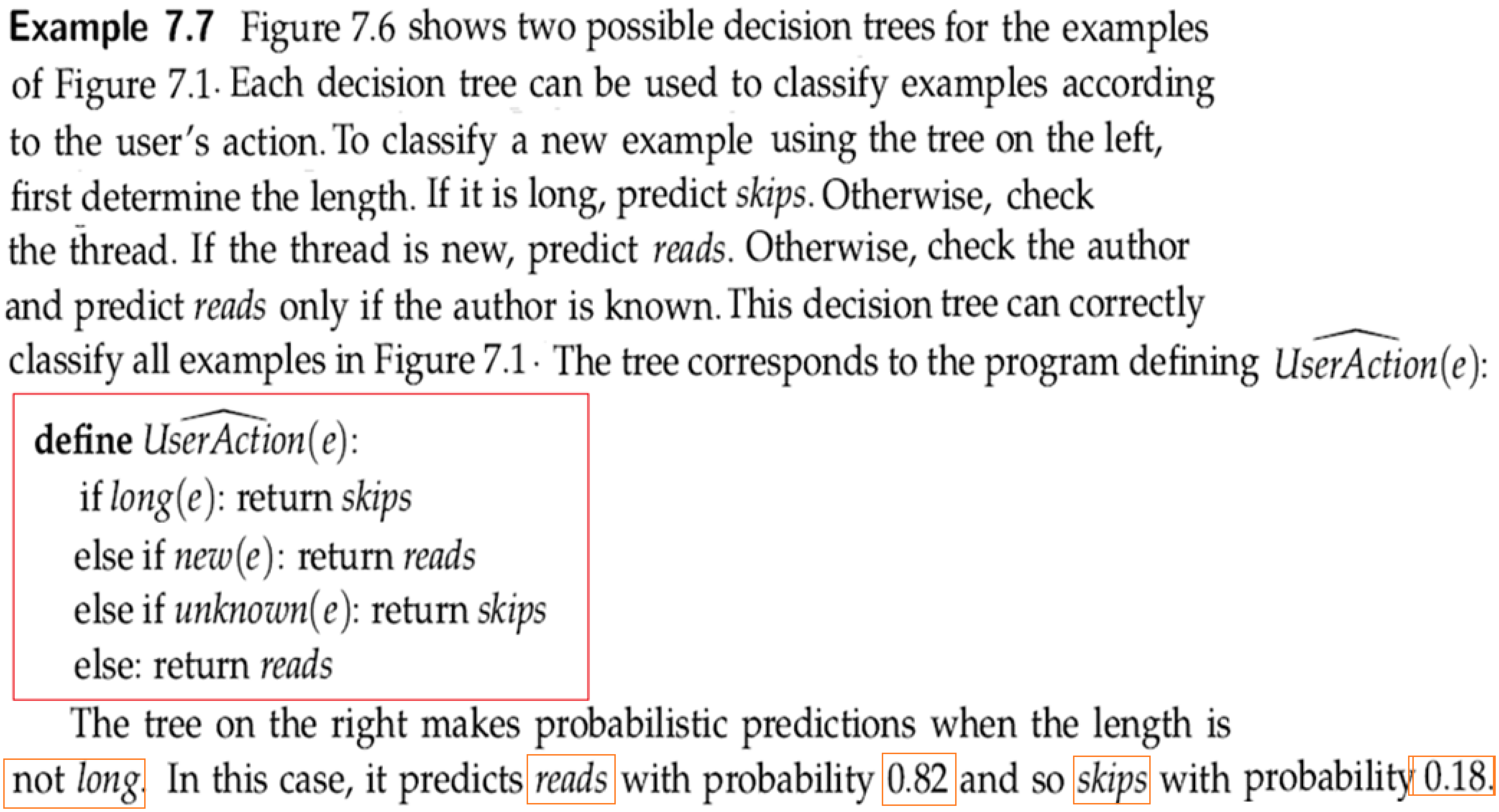
All of these examples agree on the user action; therefore, the algorithm returns the prediction *skips*. The second step of the recursive call is

$$decisionTreeLearner(\{Where\_read, Thread, Author\}, User\_action, \{e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}\}).$$

Not all of the examples agree on the user action, so assuming the stopping criterion is false, the algorithm picks a condition to split on. Suppose it picks $Thread = new$. Eventually, this recursive call returns the function on example e in the case when *Length* is *short*:

if $new(e)$ then *reads* else if $unknown(e)$ then *skips* else *reads*

The final result is the function of Example 7.7.



Example 7.7 Figure 7.6 shows two possible decision trees for the examples of Figure 7.1. Each decision tree can be used to classify examples according to the user's action. To classify a new example using the tree on the left, first determine the length. If it is long, predict *skips*. Otherwise, check the thread. If the thread is new, predict *reads*. Otherwise, check the author and predict *reads* only if the author is known. This decision tree can correctly classify all examples in Figure 7.1. The tree corresponds to the program defining $\widehat{UserAction}(e)$:

```
define $\widehat{UserAction}(e)$:
    if $long(e)$: return $skips$
    else if $new(e)$: return $reads$
    else if $unknown(e)$: return $skips$
    else: return $reads$
```

The tree on the right makes probabilistic predictions when the length is not *long*. In this case, it predicts *reads* with probability 0.82 and so *skips* with probability 0.18.

Learning Decision Trees

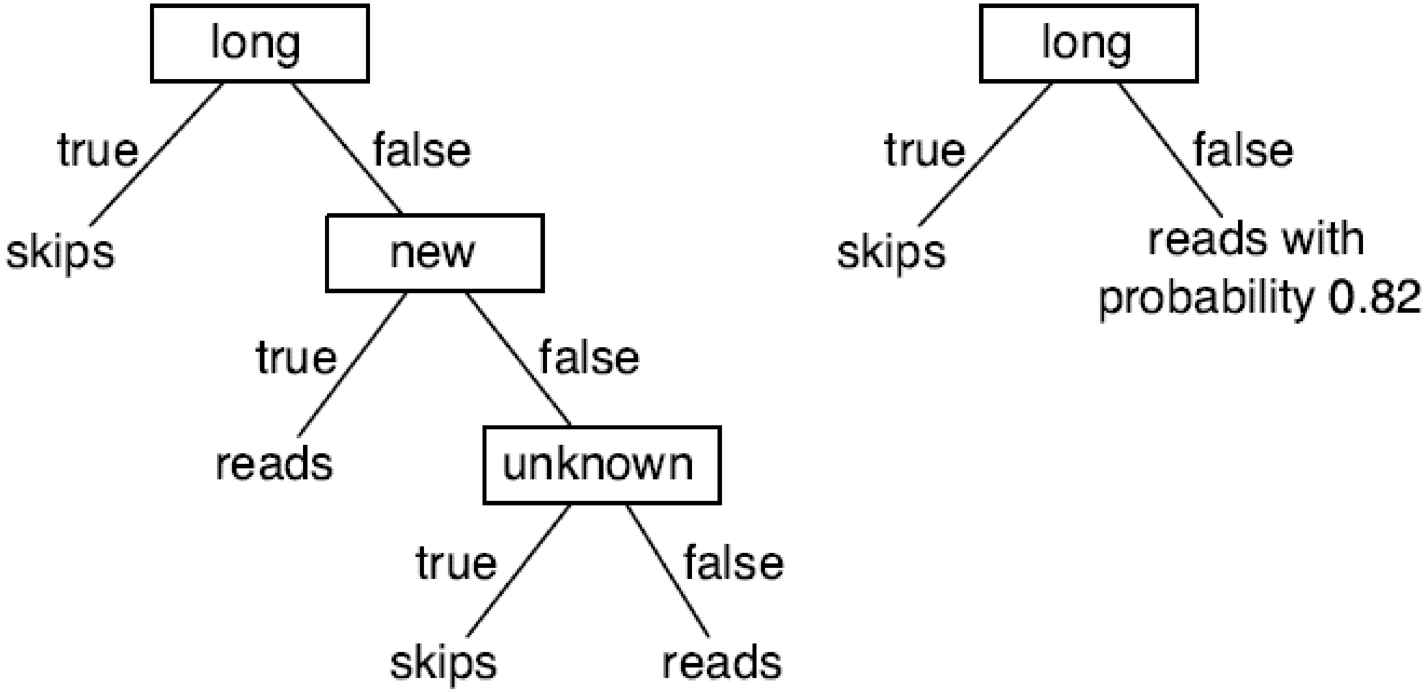
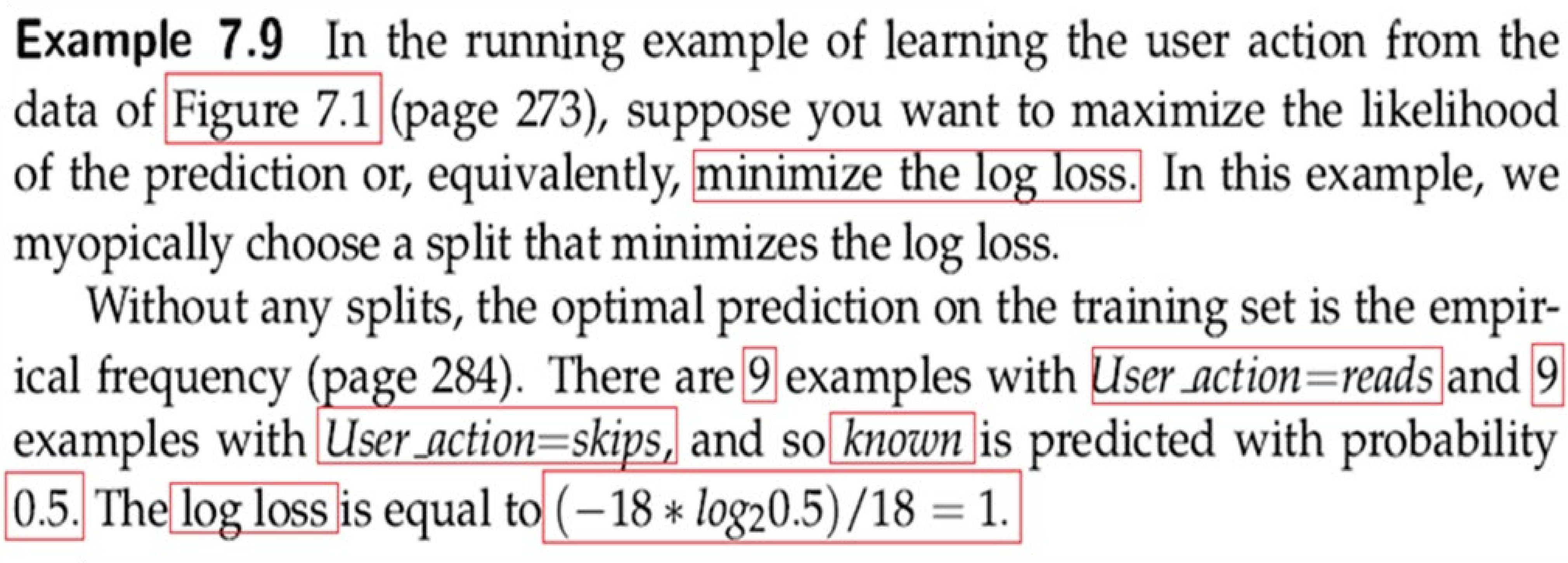


Figure 7.6: Two decision trees

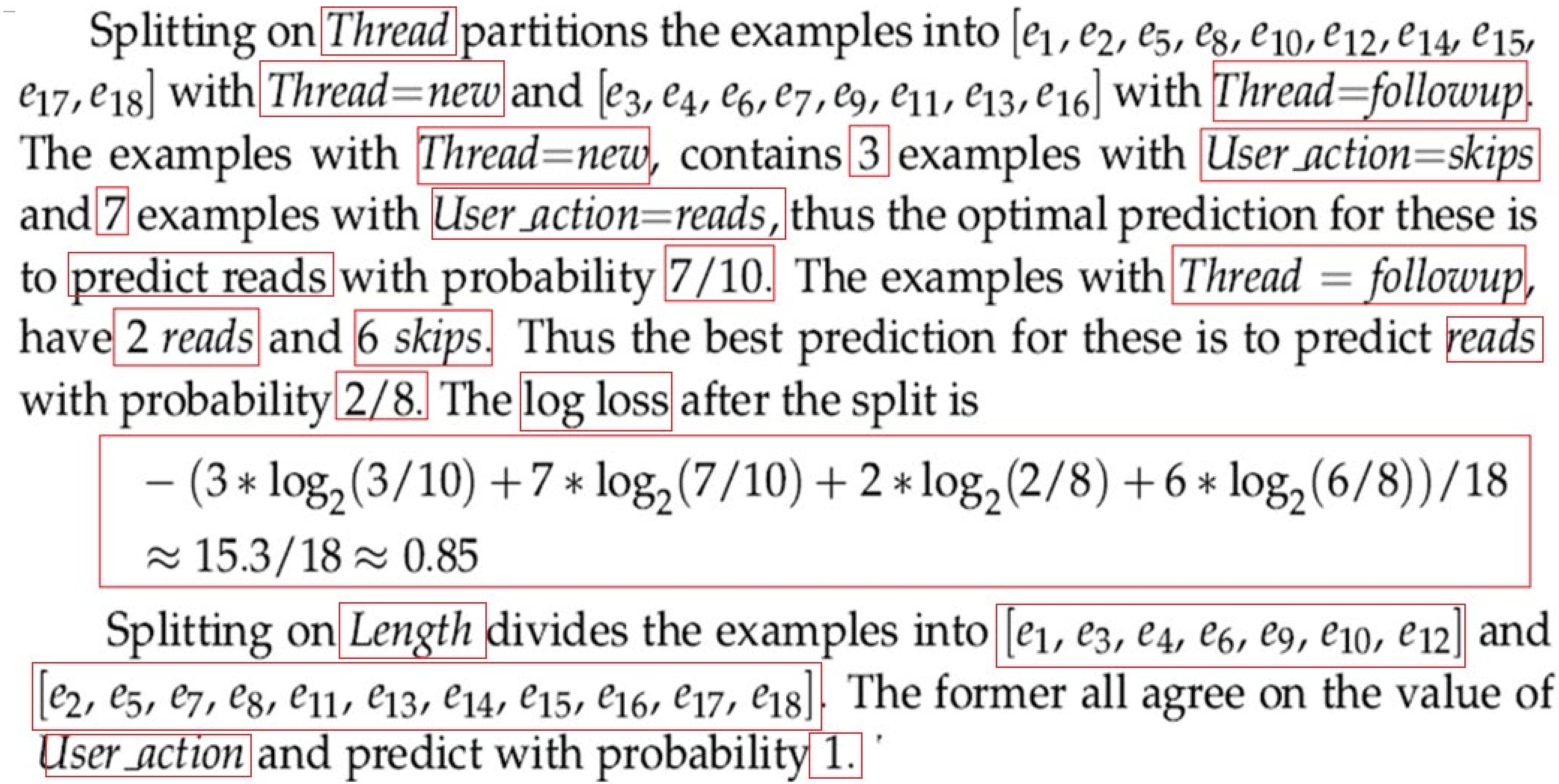
Learning Decision Trees



Example 7.9 In the running example of learning the user action from the data of Figure 7.1 (page 273), suppose you want to maximize the likelihood of the prediction or, equivalently, minimize the log loss. In this example, we myopically choose a split that minimizes the log loss.

Without any splits, the optimal prediction on the training set is the empirical frequency (page 284). There are 9 examples with *User_action=reads* and 9 examples with *User_action=skips*, and so *known* is predicted with probability 0.5. The log loss is equal to $(-18 * log_2 0.5)/18 = 1$.

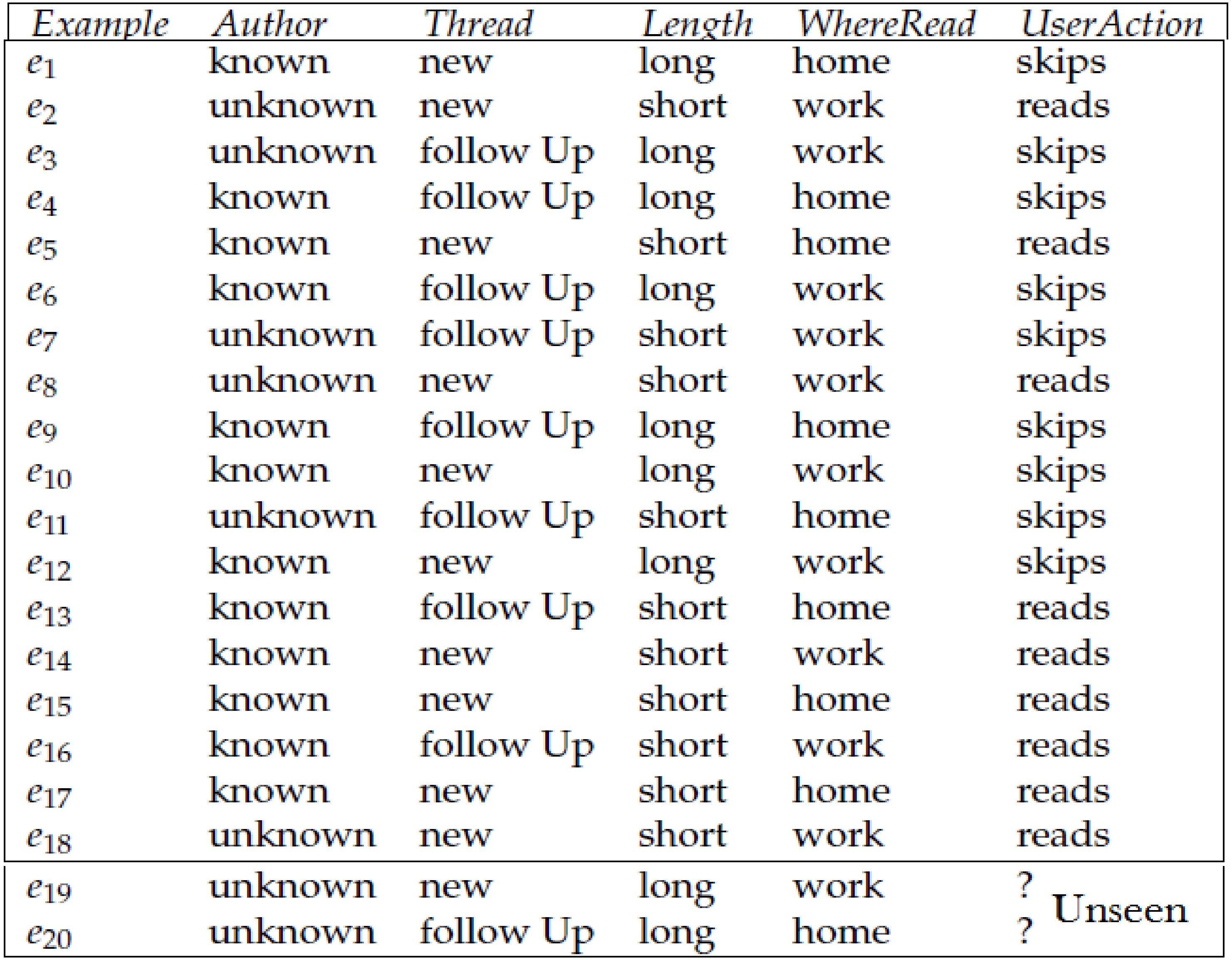
Learning Decision Trees



Splitting on *Thread* partitions the examples into $[e_1, e_2, e_5, e_8, e_{10}, e_{12}, e_{14}, e_{15}, e_{17}, e_{18}]$ with *Thread=new* and $[e_3, e_4, e_6, e_7, e_9, e_{11}, e_{13}, e_{16}]$ with *Thread=followup*. The examples with *Thread=new*, contains 3 examples with *User_action=skips* and 7 examples with *User_action=reads*, thus the optimal prediction for these is to predict reads with probability 7/10. The examples with *Thread = followup*, have 2 *reads* and 6 *skips*. Thus the best prediction for these is to predict *reads* with probability 2/8. The log loss after the split is

$$
\begin{aligned}
&-(3*\log_2(3/10)+7*\log_2(7/10)+2*\log_2(2/8)+6*\log_2(6/8))/18 \\
&\approx 15.3/18 \approx 0.85
\end{aligned}
$$

Splitting on *Length* divides the examples into $[e_1, e_3, e_4, e_6, e_9, e_{10}, e_{12}]$ and $[e_2, e_5, e_7, e_8, e_{11}, e_{13}, e_{14}, e_{15}, e_{16}, e_{17}, e_{18}]$. The former all agree on the value of *User_action* and predict with probability 1.



Example	Author	Thread	Length	WhereRead	UserAction
e_1	known	new	long	home	skips
e_2	unknown	new	short	work	reads
e_3	unknown	follow Up	long	work	skips
e_4	known	follow Up	long	home	skips
e_5	known	new	short	home	reads
e_6	known	follow Up	long	work	skips
e_7	unknown	follow Up	short	work	skips
e_8	unknown	new	short	work	reads
e_9	known	follow Up	long	home	skips
e_{10}	known	new	long	work	skips
e_{11}	unknown	follow Up	short	home	skips
e_{12}	known	new	long	work	skips
e_{13}	known	follow Up	short	home	reads
e_{14}	known	new	short	work	reads
e_{15}	known	new	short	home	reads
e_{16}	known	follow Up	short	work	reads
e_{17}	known	new	short	home	reads
e_{18}	unknown	new	short	work	reads
e_{19}	unknown	new	long	work	? Unseen
e_{20}	unknown	follow Up	long	home	?

Figure 7.1: Examples of a user’s preferences

Learning Decision Trees

- What is the **log loss** of splitting the input attribute ***WhereRead*** in ***UserAction***?
- **Ans:**
  - Splitting on ***WhereRead*** divides the examples in ***UserAction*** into ***home* → (4 *skips*) and (4 *reads***).
  - Similarly, ***work* → (5 *skips*) and (5 *reads*).**
  - Therefore, **4 combinations of entropy values** can be calculated.

Learning Decision Trees

(i). **4 x $\mathbf{log_2}$**(probability of ***home*** in ***UserAction*** attribute ***skips***)

= **4 x $\mathbf{log_2}$** (**4** ***skips***)**/(4** ***skips*** **+ 4** ***reads***) = **4x $\mathbf{log_2}$0.5**

(ii). **4 x $\mathbf{log_2}$** (probability of ***home*** in ***UserAction*** attribute ***reads***)

= **4 x $\mathbf{log_2}$ (4** ***reads***)**/(4** ***reads*** **+ 4** ***skips***) = **4x $\mathbf{log_2}$0.5**

(iii). **5 x $\mathbf{log_2}$** (probability of ***work*** in ***UserAction*** attribute ***skips***)

= **5 x $\mathbf{log_2}$ (5** ***skips***)**/(5** ***reads*****+ 5** ***skips***) = **5x $\mathbf{log_2}$0.5**

(iv). **5 x $\mathbf{log_2}$** (probability of ***work*** in ***UserAction*** attribute ***reads***)

= **5 x $\mathbf{log_2}$ (5** ***reads***)**/(5** ***reads*****+ 5** ***skips***) = **5x $\mathbf{log_2}$0.5**

So, the **log loss** = **-(4 x $\mathbf{log_2}$0.5 + 4 x $\mathbf{log_2}$0.5 + 5 x $\mathbf{log_2}$0.5 + 5 x $\mathbf{log_2}$0.5)/18**

Where **$\mathbf{log_2}$ 0.5 =-1**

So, the log loss = (4 + 4 + 5 + 5)/18 = 1

Expansion of Linear Regression for ML

- **Linear regression functions** provide the basis for **Machine Learning algorithms**
  - A **linear function** is a function whose graph is a ***straight line***
  - That is a polynomial function of degree **1** or **0**.
- When the **linear function** has only **one variable**, it is $\boldsymbol{f(x) = ax + b}$, where $\boldsymbol{b}$ is a constant.
- The function $\boldsymbol{f(x)}$ can be modified to handle a finite set of independent variables as $\boldsymbol{(x_1,\ldots, x_k)}$:

  $\boldsymbol{f(x) = f(x_1,\ldots, x_k) = a_1x_1 + a_2x_2 + \ldots\ldots\ldots\ldots + a_kx_k + b,}$ and the graph is a hyper-plane of dimension $\boldsymbol{k}$.
  - In geometry, a hyperplane is a subspace whose dimension is one less than that of its ambient space.

Linear Regression and Classification

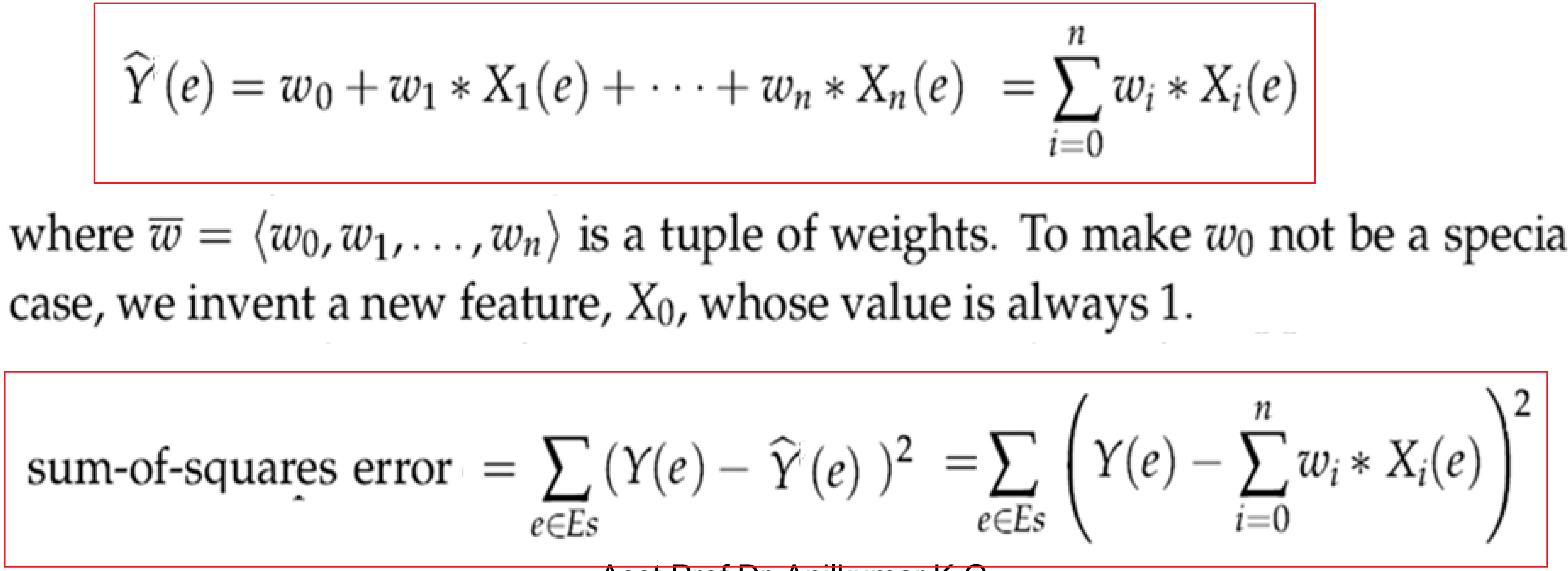
- **Gradient descent** is an iterative method to find the ***minimum*** of a function (*gradient descent will be very clear from the **perceptron learning** section on the next lecture slide*).
- **Gradient descent** starts with an initial set of ***weights***; in each step, the ***weight is updated*** in proportion to its partial derivative:

$$w_i := w_i + \eta \times \frac{\partial Error_E(\overline{w})}{\partial w_i} := w_i + \eta \times \delta \times val(e, X_i)$$

  - where ***η,*** or α*, is* the ***learning rate,*** and $\frac{\partial Error_E(\overline{w})}{\partial w_i} = \delta \times val(e, X_i)$
  - The **learning rate**, as well as the features and the data, are given as input to the learning algorithm. The **partial derivative specifies** how much a small change in the weight would change the error.

Linear Regression and Classification

- **Linear regression** is the problem of fitting a linear function to a set of **training examples**, in which the **input** and **target** features are numeric.
- Suppose the input features, $\boldsymbol{X_1}, \ldots, \boldsymbol{X_n}$, are all **numeric** and there is a single **target feature** $\boldsymbol{Y}$.
- A **linear function** of the input features is a function of the form that finds the predicted output $\vec{\boldsymbol{Y}}(\boldsymbol{e})$:



$$\widehat{Y}(e) = w_0 + w_1 * X_1(e) + \cdots + w_n * X_n(e) \quad = \sum_{i=0}^{n} w_i * X_i(e)$$

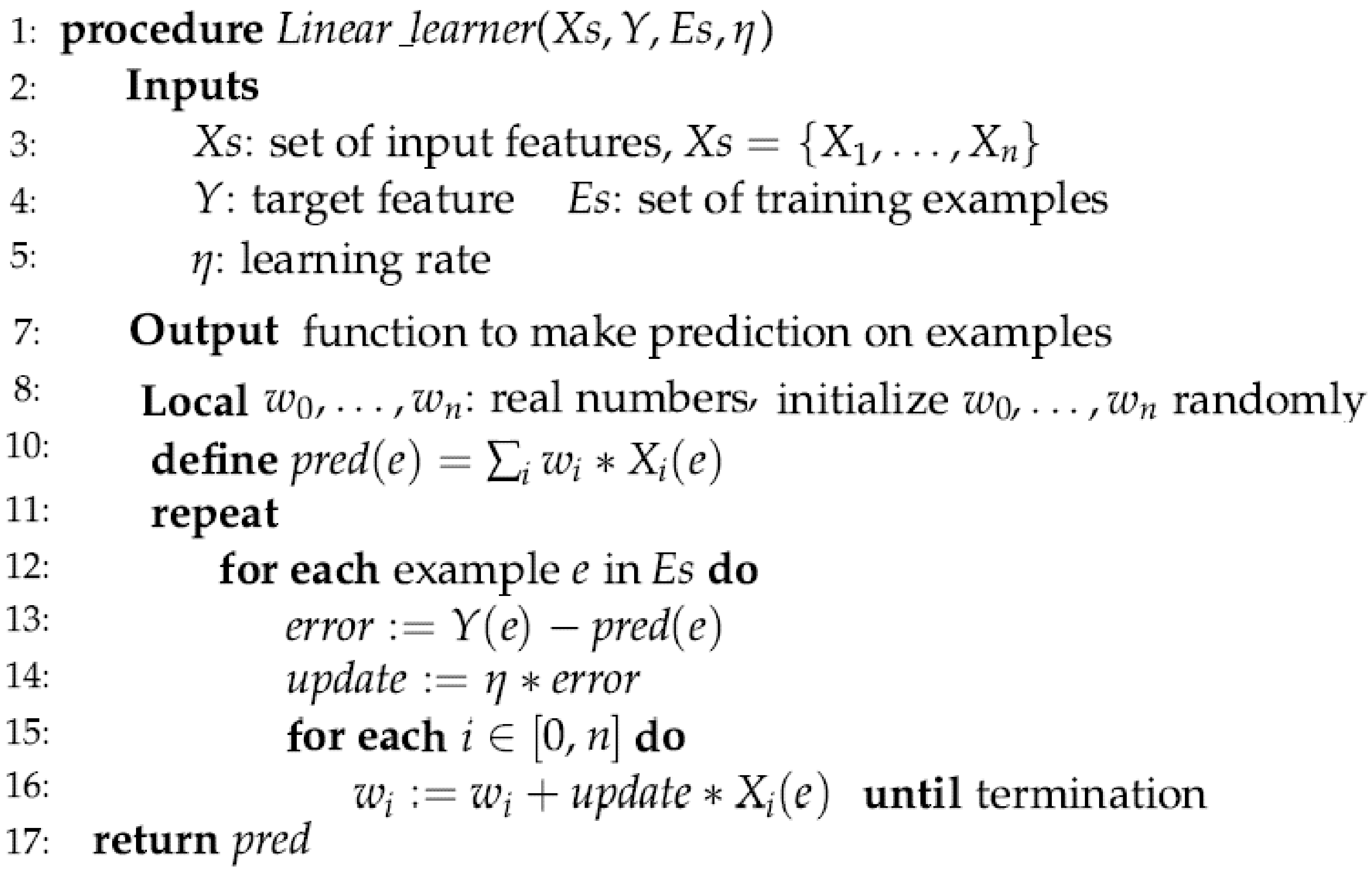
where $\overline{w} = \langle w_0, w_1, \ldots, w_n \rangle$ is a tuple of weights. To make w_0 not be a special case, we invent a new feature, X_0, whose value is always 1.

$$\text{sum-of-squares error} = \sum_{e \in Es} (Y(e) - \widehat{Y}(e))^2 = \sum_{e \in Es} \left(Y(e) - \sum_{i=0}^{n} w_i * X_i(e) \right)^2$$

Linear Regression and Backpropagation

- **Gradient descent** is an iterative method ***to find the minimum of a function***.
- **Gradient descent** for ***minimizing error*** starts with an initial set of **weights**; in each step, it decreases each **weight** in proportion to its **partial derivative.**
- The **backpropagation** learning is based on the **gradient descent** that keeps changing the neuron **weights** until there is the most significant error reduction by an amount known as the **learning rate** (α).
- **Learning rate** is a scalar parameter used to set the rate of adjustments to reduce the errors faster.
- It is used to adjust **weights** and **bias** in the backpropagation process.
- **The higher the learning rate,** the faster the algorithm will reduce the errors and the quicker the training process.

Linear Regression and Classification



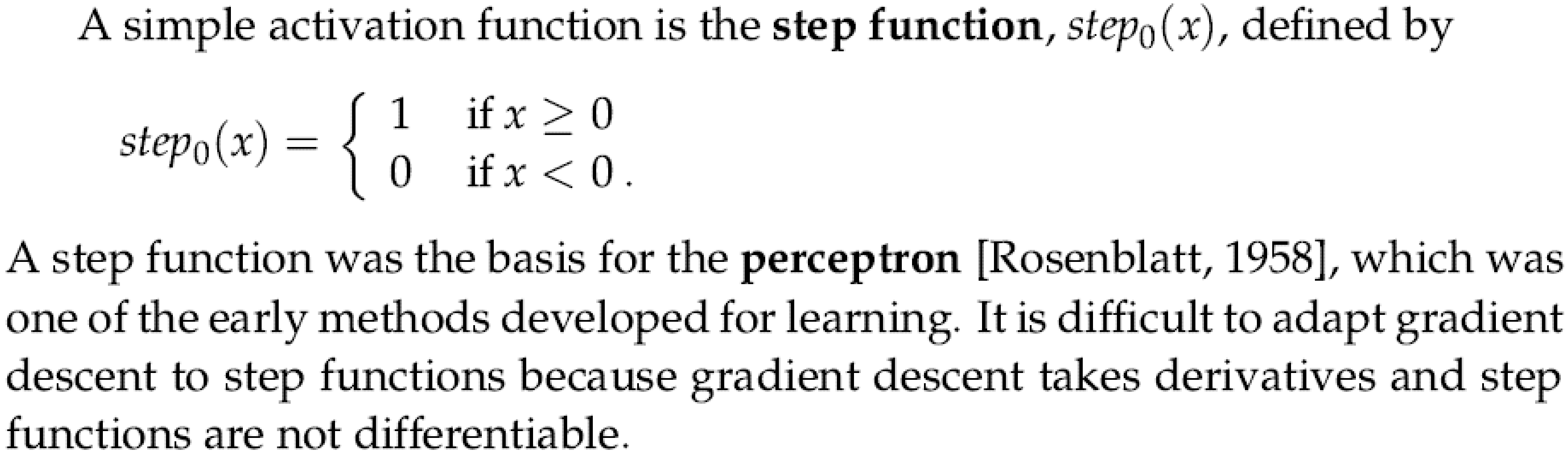
```
procedure $Linear\_learner(Xs, Y, Es, \eta)$
    Inputs
        $Xs$: set of input features, $Xs = \{X_1, \ldots, X_n\}$
        $Y$: target feature    $Es$: set of training examples
        $\eta$: learning rate
    Output function to make prediction on examples
     Local $w_0, \ldots, w_n$: real numbers, initialize $w_0, \ldots, w_n$ randomly
     define $pred(e) = \sum_i w_i * X_i(e)$
     repeat
         for each example $e$ in $Es$ do
             $error := Y(e) - pred(e)$
             $update := \eta * error$
             for each $i \in [0, n]$ do
                 $w_i := w_i + update * X_i(e)$   until termination
 return $pred$
```

Figure 7.8: Incremental gradient descent for learning a linear function

Linear Regression and Classification

- The algorithm presented in **Figure 7.8** is called **incremental gradient descent** because the weights change while iterates through the examples.
- If the training examples are selected at random, this is called **stochastic gradient descent (SGD)**.
  - These **incremental methods** have cheaper steps than gradient descent and typically become more accurate when saving all the changes to the end of the examples.
  - However, it is not guaranteed to converge as individual examples can move the weights away from the minimum.

Activation Functions



A simple activation function is the **step function**, $step_0(x)$, defined by

$$step_0(x) = \begin{cases} 1 & \text{if } x \geq 0 \\ 0 & \text{if } x < 0 \,. \end{cases}$$

A step function was the basis for the **perceptron** [Rosenblatt, 1958], which was one of the early methods developed for learning. It is difficult to adapt gradient descent to step functions because gradient descent takes derivatives and step functions are not differentiable.

- If the **activation** is (almost everywhere) differentiable, gradient descent can be used to update the weights.
- The step size might need to converge to zero to guarantee convergence.

Activation Functions

- One differentiable activation function is the **sigmoid** or **logistic function**: This function, depicted in **Figure 7.9**, squashes the real line into the interval (0, 1), which is appropriate for classification because we would never want to make a prediction of greater than 1 or less than 0.

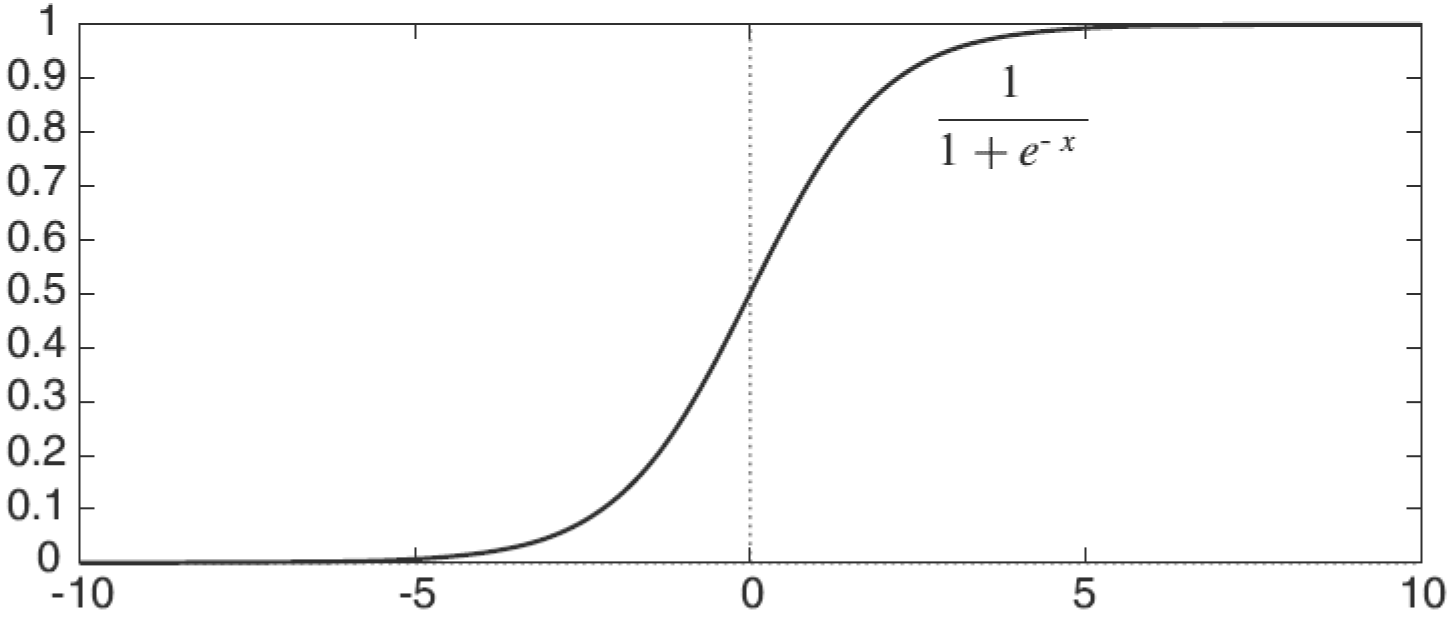


Figure 7.9: The sigmoid or logistic function

Underfitting and Overfitting

- **Overfitting** occurs when a model is trained too well on the training data and performs poorly on new, unseen data.
- The factors determining how well a machine learning algorithm will perform are its ability to
  1. **Make the training error small**
  2. **Make the gap between training and test error small**.
- These two factors correspond to the two central challenges in machine learning: **Underfitting** and **Overfitting**

Underfitting and Overfitting

– **Underfitting**: Underfitting occurs when a statistical model **cannot adequately capture** the underlying structure of the data.

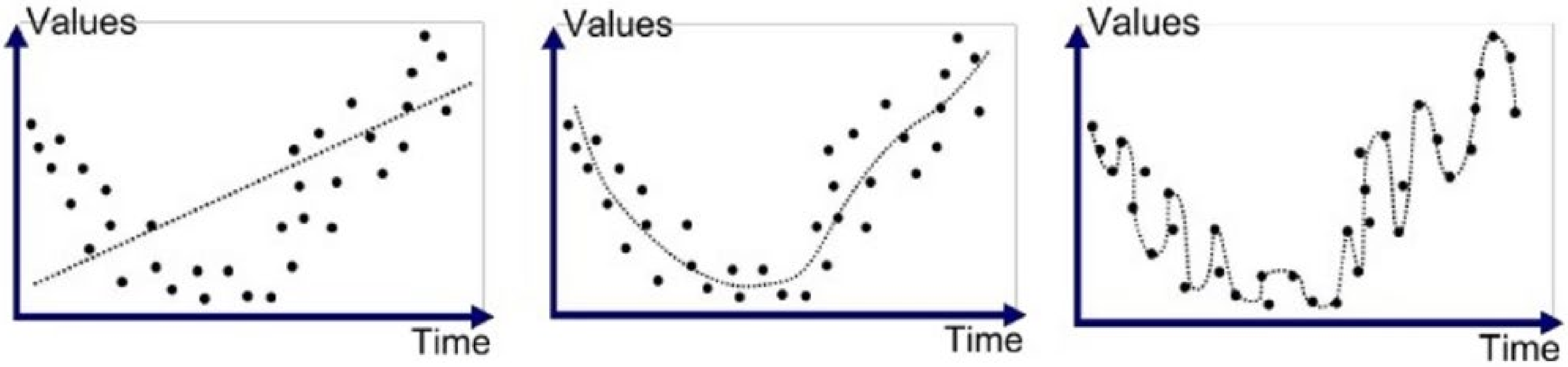
  - **Underfitting** refers to a model that neither trains the data nor generalizes to new data.
  - An **underfit machine learning model** is unsuitable and will be obvious as it will perform poorly on the **training data**.

– **Overfitting:** overfitting **occurs** if the model shows **low bias** but **high variance** (*variance = trained data – test data*)**:**

  - Such models are also responsible for predicting poor results due to their complexity.
  - Overfitting happens when a model learns the detail and noise in the **training data** to the extent that it negatively impacts the model's performance on **new data**. **Overfitting** is more likely with nonparametric and nonlinear models with more flexibility when learning a **target function**.

Underfitting and Overfitting

- **Underfitting** occurs when the model is not able to obtain a sufficiently ***low error value*** on the training set.
- **Overfitting** occurs when the gap between the **training** and **test** errors is too large.



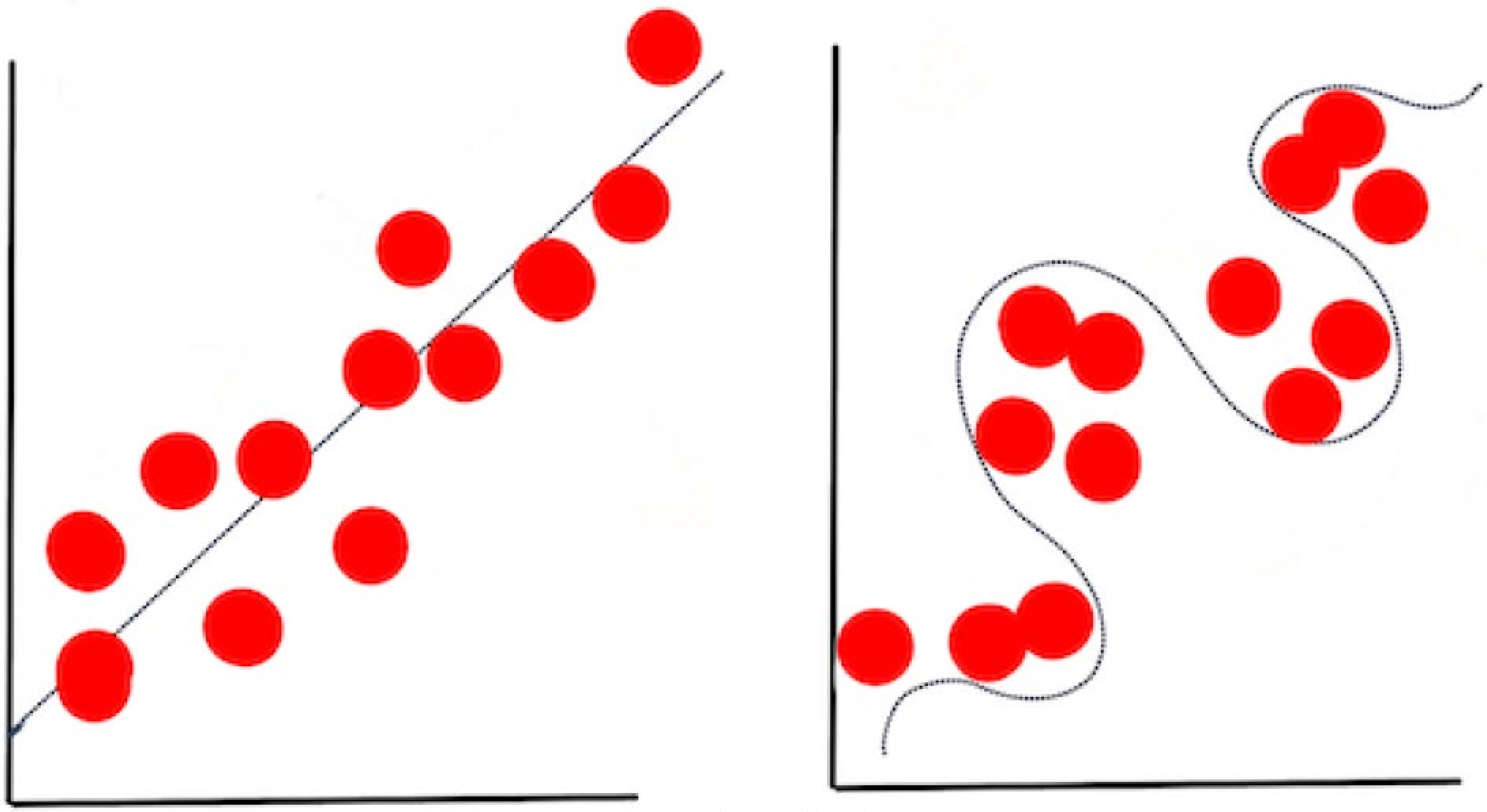
Underfitted

Good Fit/Robust

Overfitted

Figure 3.1 Underfitting, appropriate-fitting and overfitting events

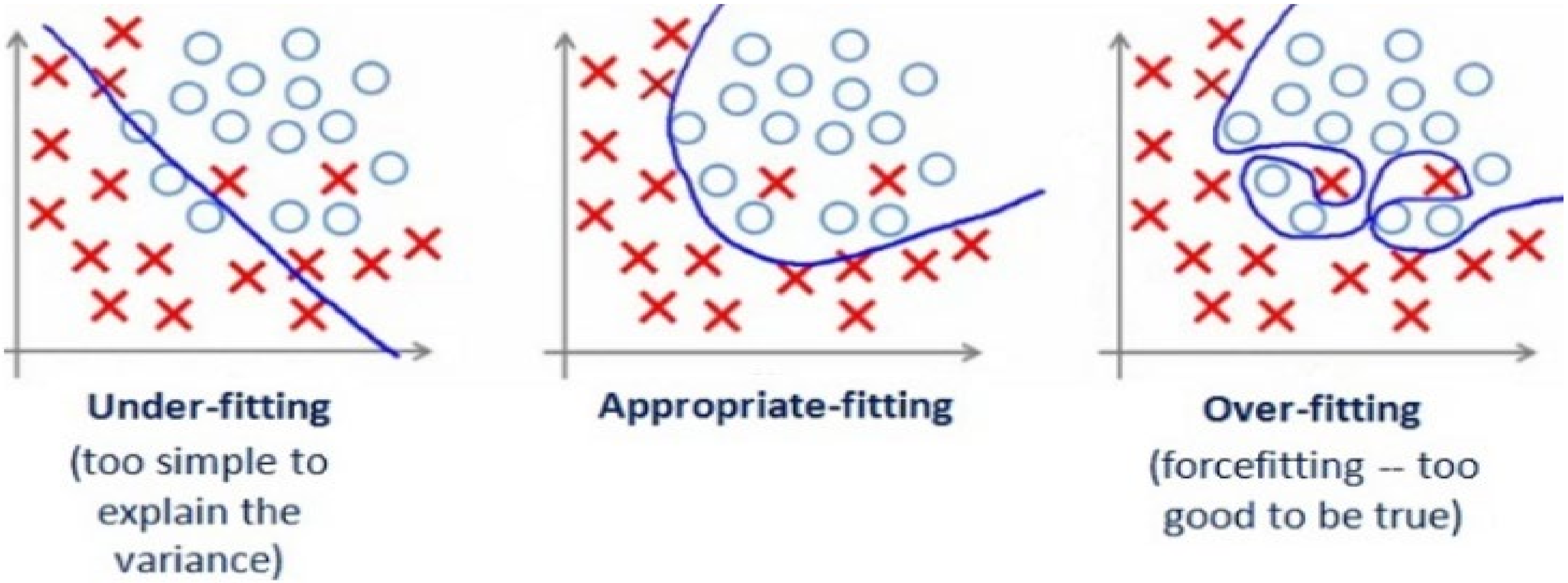
Underfitting and Overfitting



Correct vs overfit model

Underfitting and Overfitting

- Here comes the concept of **overfitting** and **underfitting** training issues. **Figure 3.1** shows overfitting, appropriate fitting and underfitting situations.



Under-fitting
(too simple to explain the variance)

Appropriate-fitting

Over-fitting
(forcefitting -- too good to be true)

Figure 3.2 Underfitting, appropriate-fitting and overfitting training events

Underfitting and Overfitting

- From **figure 3.2**, by looking at the **graph on the left side** we can predict that the line does not cover all the points. Such model tend to cause **underfitting** of data, it also called **High Bias**.
- Where as the **graph on right side**, shows the predicted line **covers all the points in graph**.
- Such model are also responsible to **predict poor result** due to its complexity. Such model tend to cause **overfitting** of data (it is also called **High Variance**).
  - In such condition you can also think that it's a good graph which cover all the points.
  - But that's not actually true, the predicted line into the graph covers all points which are noise and outlier.

Cross-validation

https://www.geeksforgeeks.org/cross-validation-machine-learning/

- **Cross-validation** is a machine learning technique to evaluate a model's performance on **unseen data (test data)**.
- It involves dividing the **training data** into multiple ***folds or subsets***, using one of these folds as a ***validation*** and ***training set***.
- This process is repeated using a different ***fold*** as the ***validation set***.
- Finally, the results from each ***validation step*** are averaged to produce a more robust estimate of the model's performance.
- The primary purpose of **cross-validation** is to prevent **overfitting**
  - it occurs *when a model is trained too well on the training data and performs poorly on new, unseen data*.

Methods of Cross-validation

https://www.geeksforgeeks.org/cross-validation-machine-learning/

- **Simple Validation:**
  - In this method, we train **50%** of the given data set; the rest, **50%, is used for testing purposes**.
  - The major drawback of this validation method is that we perform training on **50%** of the dataset;
    - It may be possible that the **remaining 50%** of the data contains some important information we left while training our model, i.e., ***higher bias***.

Methods of Cross-validation

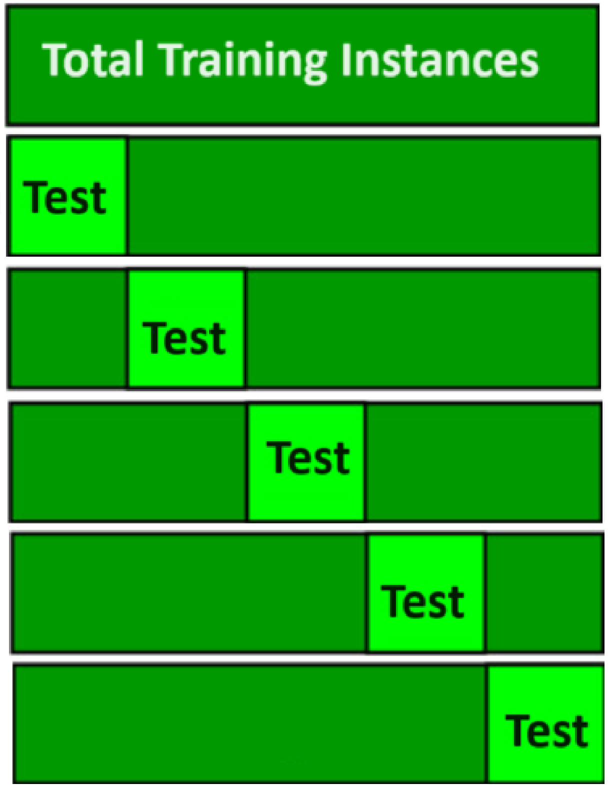
https://www.geeksforgeeks.org/cross-validation-machine-learning/

- **K-Fold Cross Validation:**
  - In this method, we split the **training data set** into ***k* subsets** (known as ***folds***), then we perform training on all the subsets but leave one(***k*-1**) subset to evaluate the trained model.
  - In this method, we iterate ***k times*** with a different subset reserved for testing purposes each time.

K-fold Cross-validation

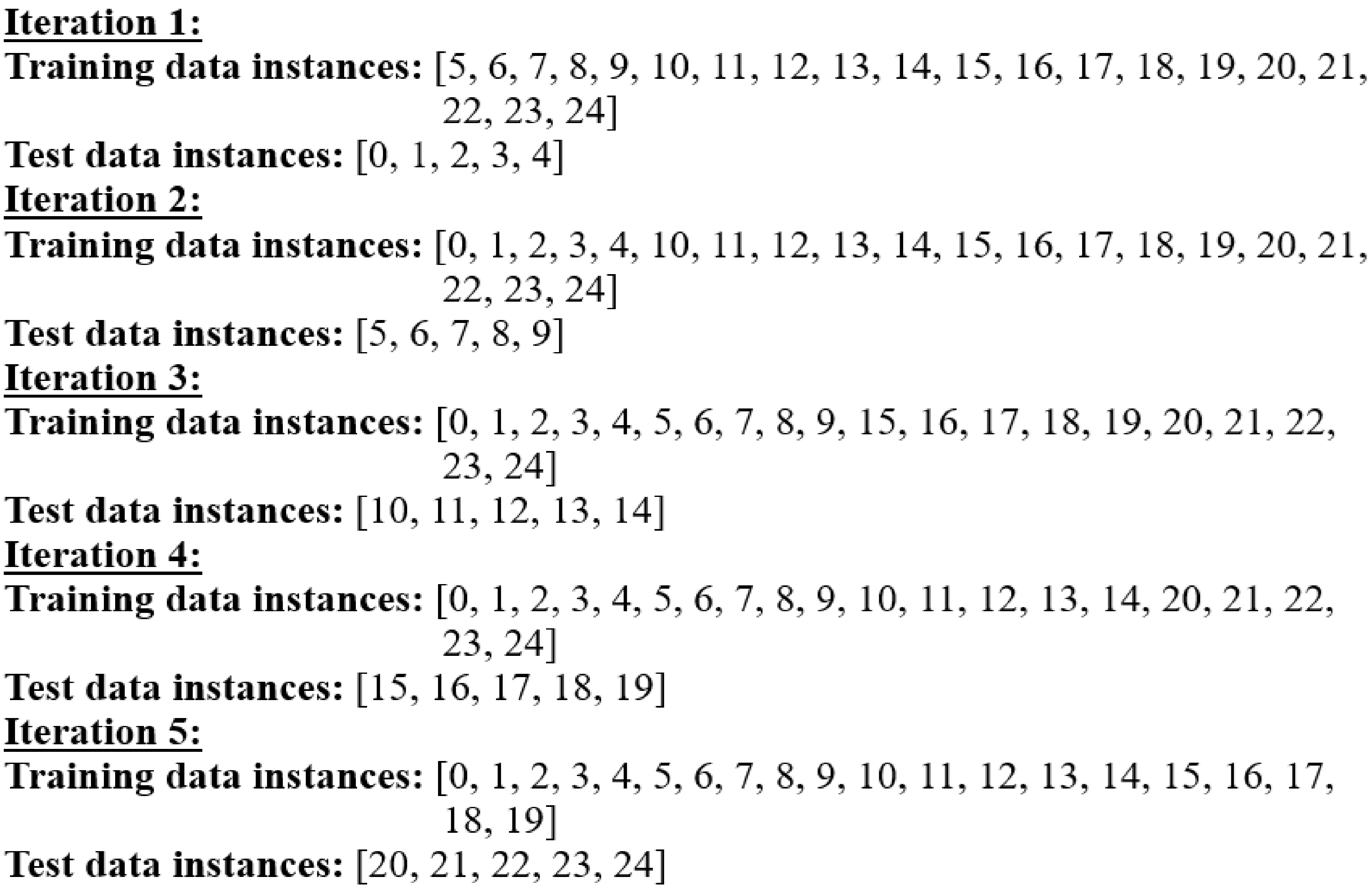
https://www.geeksforgeeks.org/cross-validation-machine-learning/

- The diagram below shows an example of the training and evaluation subsets generated in ***k*-fold cross-validation**
  - Here, we have **25 training instances**, and ***k* is 5**, which means there are **5 test instances** at each iteration.



- The sample **25 training instances** are as follows:

[0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]



Iteration 1:

Training data instances: [5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

Test data instances: [0, 1, 2, 3, 4]

Iteration 2:

Training data instances: [0, 1, 2, 3, 4, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

Test data instances: [5, 6, 7, 8, 9]

Iteration 3:

Training data instances: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 15, 16, 17, 18, 19, 20, 21, 22, 23, 24]

Test data instances: [10, 11, 12, 13, 14]

Iteration 4:

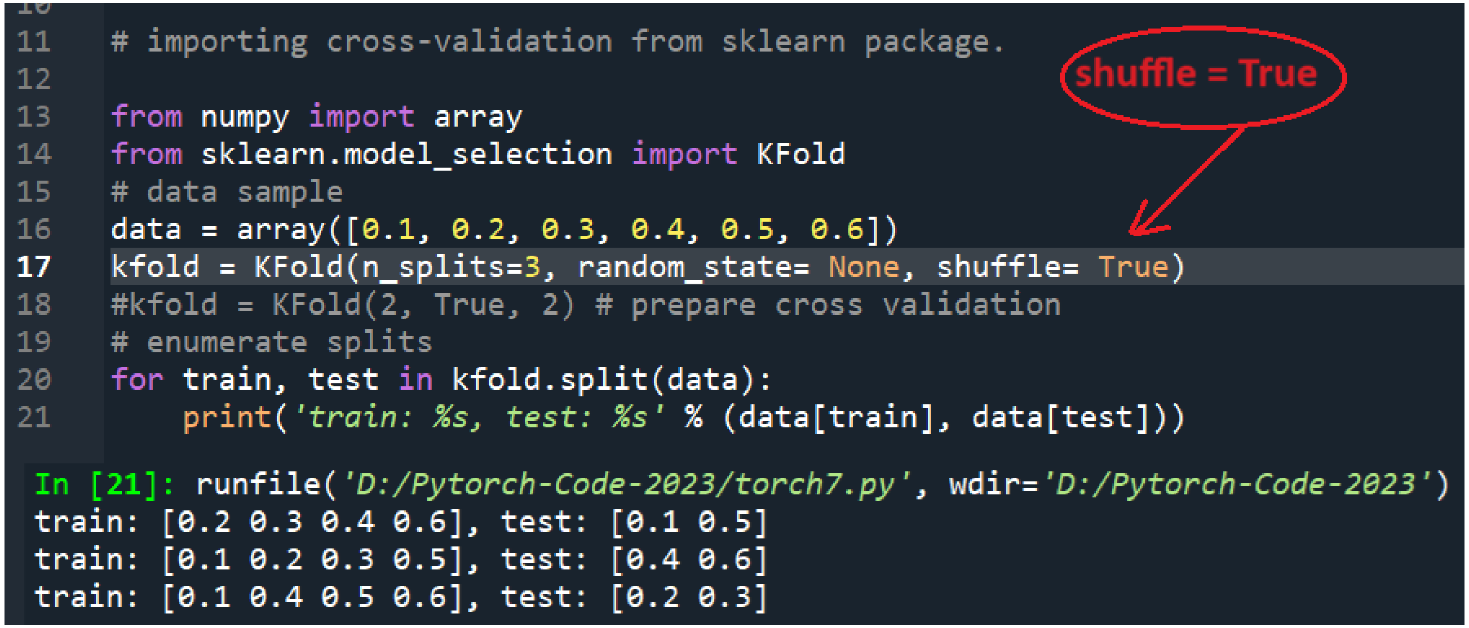
Training data instances: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 20, 21, 22, 23, 24]

Test data instances: [15, 16, 17, 18, 19]

Iteration 5:

Training data instances: [0, 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 18, 19]

Test data instances: [20, 21, 22, 23, 24]

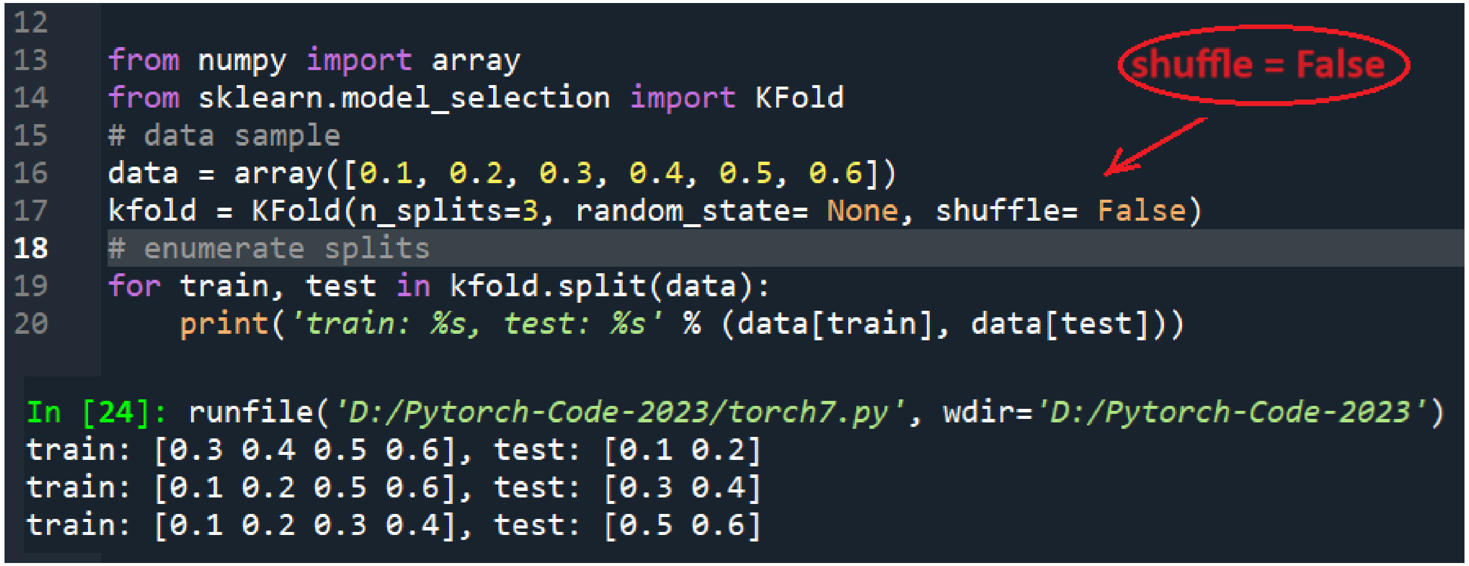


shuffle = True

```python
# importing cross-validation from sklearn package.

from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
kfold = KFold(n_splits=3, random_state= None, shuffle= True)
#kfold = KFold(2, True, 2) # prepare cross validation
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test]))
```

```
In [21]: runfile('D:/Pytorch-Code-2023/torch7.py', wdir='D:/Pytorch-Code-2023')
train: [0.2 0.3 0.4 0.6], test: [0.1 0.5]
train: [0.1 0.2 0.3 0.5], test: [0.4 0.6]
train: [0.1 0.4 0.5 0.6], test: [0.2 0.3]
```



shuffle = False

```python

from numpy import array
from sklearn.model_selection import KFold
# data sample
data = array([0.1, 0.2, 0.3, 0.4, 0.5, 0.6])
kfold = KFold(n_splits=3, random_state= None, shuffle= False)
# enumerate splits
for train, test in kfold.split(data):
    print('train: %s, test: %s' % (data[train], data[test]))
```

```
In [24]: runfile('D:/Pytorch-Code-2023/torch7.py', wdir='D:/Pytorch-Code-2023')
train: [0.3 0.4 0.5 0.6], test: [0.1 0.2]
train: [0.1 0.2 0.5 0.6], test: [0.3 0.4]
train: [0.1 0.2 0.3 0.4], test: [0.5 0.6]
```

https://machinelearningmastery.com/k-fold-cross-validation/

K-fold Cross-validation

https://www.geeksforgeeks.org/cross-validation-machine-learning/

- **Advantages of *k*-fold cross-validation:**
  - More accurate estimate of **out-of-sample accuracy**.
  - More "***efficient***" use of data, as every observation is used for ***training*** and ***testing***.
  - **Preventing Overfitting**: Cross-validation helps to prevent overfitting by providing a more robust estimate of the model's performance on unseen data.
  - **Model Selection**: Cross-validation can be used to compare different models and select the one that performs the best on average.
  - **Hyperparameter tuning:** Cross-validation can be used to optimize the hyperparameters of a model, such as the regularization parameter, by selecting the values that result in the best performance on the validation set.
  - **Data Efficient:** Cross-validation allows the use of all the available data for training and validation, making it a more data-efficient method than traditional validation techniques.

K-fold Cross-validation

https://www.geeksforgeeks.org/cross-validation-machine-learning/

- **Disadvantages of cross-validation:**
  1. **Computationally Expensive:** Cross-validation can be computationally expensive, especially when the number of folds is large, or the model is complex and requires a long time to train.
  2. **Bias-Variance Tradeoff:** The choice of the number of folds in cross-validation can impact the bias-variance tradeoff, i.e., too few folds may result in ***high variance***, while too many folds may result in ***high bias***.

Adam Optimizer

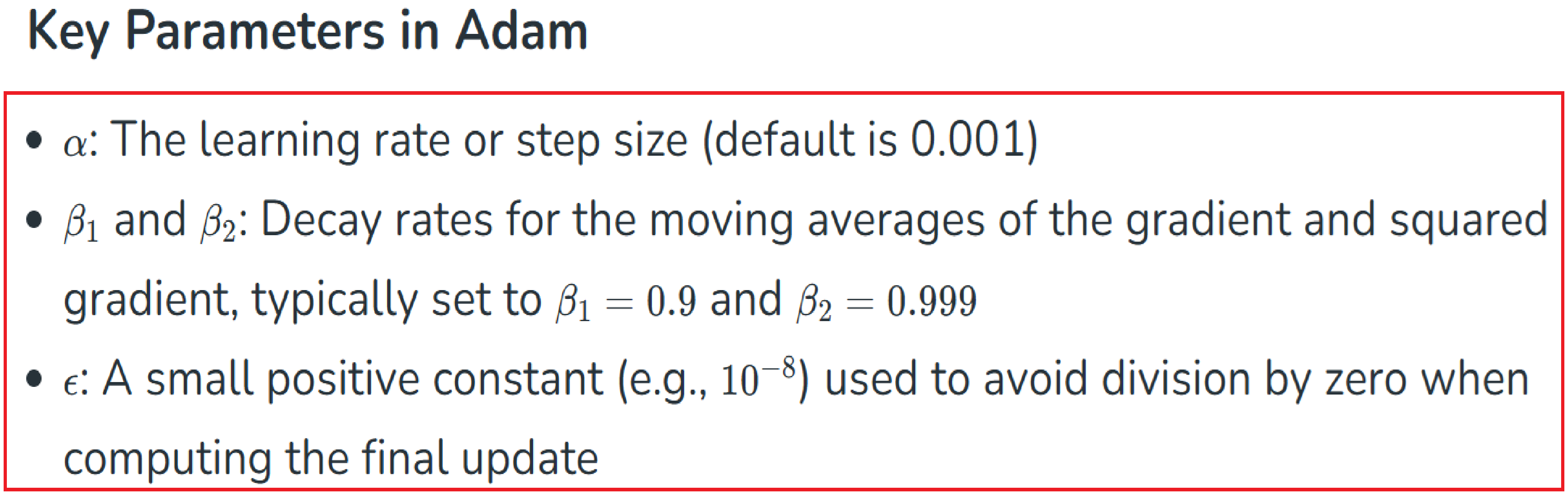
https://www.geeksforgeeks.org/adam-optimizer/

- The **Adam optimizer** (**Adaptive Moment Estimation)** is a popular algorithm used in machine learning, particularly in deep learning, to **update network weights during training**.
- It's known for its efficiency and effectiveness in converging quickly to optimal solutions.
- The Adam optimizer combines the advantages of the **Momentum** and **RMSprop** techniques to adjust learning rates during training.
- It works well with large datasets and complex models because it utilizes memory efficiently and automatically adjusts the learning rate for each parameter.

Adam Optimizer

https://www.geeksforgeeks.org/adam-optimizer/

- **Adam** addresses several challenges of ***gradient descent*** optimization:
- **Dynamic learning rates**: Each parameter has its adaptive learning rate based on past gradients and their magnitudes.
- This helps the optimizer avoid oscillations and get past local minima more effectively.



Key Parameters in Adam

- α: The learning rate or step size (default is 0.001)
- β_1 and β_2: Decay rates for the moving averages of the gradient and squared gradient, typically set to $\beta_1 = 0.9$ and $\beta_2 = 0.999$
- ϵ: A small positive constant (e.g., 10^{-8}) used to avoid division by zero when computing the final update

Adam Optimizer (ChatGPT)

- **Adam (Adaptive Moment Estimation)** is an optimization algorithm used to **update the weights** of a neural network during training.
- It combines ideas from ***momentum*** and ***adaptive learning rates*** to efficiently handle the weight updating in the presence of sparse or noisy data.
- **How does Adam work?**
  - It maintains moving averages of the **gradients** (first moments) and **squared gradients** (second moments).
  - It adjusts the ***learning rate*** for each parameter dynamically.
  - It corrects ***bias*** in these estimates during initial steps.

Adam Optimizer (ChatGPT)

- Consider the following linear dataset:

x (input)	**y (target)**
1.0	2.0
2.0	4.0
3.0	6.0

- **Goal**:
  - Find the ***weight*** that predicts ***y*** from ***x*** with a simple linear prediction ***y' = wx*** (where ***w*** is the *weight of the **x**,* and **x** is the input and **y'** is the predicted value of ***x***)
- **Initialize parameters**:
  - $\boldsymbol{w}$ = **0** (initial value of the weight)
  - $\boldsymbol{m_0}$ = **0** (first momentum value)
  - $\boldsymbol{v_0}$ = **0** (second momentum value)
- **Hyperparameters:** α **= 0.1,** β_1 **= 0.9,** β_2 **=0.999,** ε **= 10-8**

Adam Optimizer (ChatGPT)

Iteration1 (x = 1.0, y = 2.0)

Prediction, $y' = wx = 0 \times 1.0 = \mathbf{0}$

Gradient, $g = x \frac{\partial}{\partial w} \frac{1}{2} (y' - y)^2 = x(y' - y) = 1.0 * (0 - 2.0) = \mathbf{-2.0}$

- **Next is to update the momentum:**

New first momentum, $m_1 = \beta_1 m_0 + (1 - \beta_1) g^1$

$= (0.9 * 0) + (1 - 0.9) * -2.0 = 0 + 0.1 * -2.0$

$= \mathbf{-0.2}$

New second momentum, $v_1 = \beta_2 v_0 + (1 - \beta_2) g^2$

$= (0.999 * 0) + (1 - 0.999) * (-2.0)^2$

$= 0 + 0.001 * 4$

$= \mathbf{0.004}$

Adam Optimizer (ChatGPT)

- **Next is the *Bias* correction:**

$\boldsymbol{bm_1 = m_1 /(1\text{-} \beta_1)} = \boldsymbol{-}\ 0.2 / (1 - 0.9) = \boldsymbol{-2.0}$

$\boldsymbol{bv_1 = v_1 /(1\text{-} \beta_2)} = 0.004 / (1 - 0.999) = \boldsymbol{4.0}$

(where ***bm*₁** and ***bv*₁** are ***bias*** values from *momentums* and *decay rates*)

- **Next, update the *weight:***

New weight $\boldsymbol{w_1} = \boldsymbol{w_0} - (\alpha * \frac{bm_1}{\sqrt{bv_1}+\varepsilon})$

$= 0 - (0.1 * -2.0/(\sqrt{4.0} + 10^{-8}) = \mathbf{0.1}$

<u>Iteration2 (x = 2.0, y = 4.0)</u>

- **Prediction,** $\boldsymbol{y'} = \boldsymbol{w_1\, x} = 0.1 \times 2.0 = \mathbf{0.2}$
- **Gradient,** $\boldsymbol{g = x \frac{\partial}{\partial w} ½\,(y' - y)^2 = x(y' - y)} = 2.0 *(0.2 - 4.0) = \boldsymbol{-7.6}$

Adam Optimizer (ChatGPT)

- **Next is to update the momentum:**

New first momentum, $\boldsymbol{m_2 = \beta_1 m_1 + (1 - \beta_1) g^1}$

$= (0.9 * -0.2) + (1 - 0.9) * -7.6$

$= -0.18 + 0.76$

$= \mathbf{-0.94}$

New second momentum, $\boldsymbol{v_2 = \beta_2 v_1 + (1 - \beta_2) g^2}$

$= (0.999 * 0.004) + (1 - 0.999) * (-7.6)^2$

$= 0.003996 + 0.05776$

$= \mathbf{0.061756}$

Adam Optimizer (ChatGPT)

- **Next is the *Bias* correction:**

$$\boldsymbol{bm_2 = m_2 /(1 - \beta_1^2)} = \mathbf{-}\ 0.94\ /\ (1 - 0.9^2) = -0.94/0.19 = \mathbf{-4.9474}$$

$$\boldsymbol{bv_2 = v_2 /(1 - \beta_2^2)} = 0.061756\ /\ (1 - 0.999^2) = 0.06176/0.001999$$

$$= \mathbf{30.893}$$

(where ***bm*₂** and ***bv*₂** are updated ***bias*** values from *momentums* and *decay rates*)

- **Next, update the *weight:***

$$\textbf{New weight } \boldsymbol{w_2} = \boldsymbol{w_1} - (\alpha * \frac{bm_2}{\sqrt{bv_2}+\varepsilon})$$

$$= \ 0.1 - (0.1 * -4.9474/(\sqrt{30.893} + 10^{-8})$$

$$= 0.1 + (0.49474/5.55815)$$

$$= \mathbf{0.189}$$

Adam Optimizer (ChatGPT)

- **Iteration3 (x = 3.0, y = 6.0)**
- **Prediction**, $\boldsymbol{y' = w_2 x} = 0.189 \times 3.0 = \mathbf{0.567}$
- **Gradient,** $\boldsymbol{g = x \frac{\partial}{\partial w} ½ (y' - y)^2 = x(y' - y)} = 3.0 * (0.567 - 6.0) = \mathbf{-16.299}$
- **Next is to update the momentum:**

New first momentum, $\boldsymbol{m_3 = \beta_1 m_2 + (1 - \beta_1)g^1}$

$= (0.9 * -0.94) + (1 - 0.9) * -16.299$

$= -0.846 - 1.6299$

$= \mathbf{-2.476}$

New second momentum, $\boldsymbol{v_3 = \beta_2 v_2 + (1 - \beta_2)g^2}$

$= (0.999 * 0.061756) + (1 - 0.999) * (-16.299)^2$

$= 0.061694244 + 0.265657401$

$= \mathbf{0.32735}$

Adam Optimizer (ChatGPT)

- **Next is the *Bias* correction:**

$\boldsymbol{bm_3 = m_3 /(1 - \beta_1^3)}$ = –2.476 / (1 – 0.9^3) = –2.476 /(1 – 0.729) =

= –2.476/0.271

= **–9.137**

$\boldsymbol{bv_3 = v_3 /(1 - \beta_2^3)}$ = 0.32735 / (1 – 0.999^3)

= 0.32735/(1 – 0.997002999)

= 0.32735/0.002997001

= **109.226**

(where $\boldsymbol{bm_2}$ and $\boldsymbol{bv_2}$ are updated ***bias*** values from *momentums* and *decay rates*)

Adam Optimizer (ChatGPT)

- **Next, update the *weight:***

  New weight $\boldsymbol{w_3}$ = $\boldsymbol{w_2}$ – (α * $\frac{bm_3}{\sqrt{bv_3}+\varepsilon}$)

  = 0.189 – (0.1 * (–9.137/($\sqrt{109.226} + 10^{-8}$))

  = 0.189 – (0.1 * (–9.137/10.4511))

  = 0.189 – (0.1 * (–0.8741))

  = 0.189 – (–0.08741)

  = **0.2764**

- Repeat the sequence until to get the correct value of ***y*** for each ***x*** with the updated weight.

Adam Optimizer (ChatGPT)

- **Summary**
  - The weight is gradually updated to fit the data
  - Adam adapts the learning rate for each parameter based on the moments
  - The optimizer converges faster and more reliable than the standard gradient descent method