UNIT 2

EVALUATION METRICS FOR MACHINE LEARNING MODELS

Structure

2.1 INTRODUCTION

In the previous unit, you studied types of machine learning. Like me, you were also eagerly waiting to see how a machine can learn from data. In Sec. 2.3, you get a tiny flavour of that. But the actual objective of this unit is to explain some performance measures of evaluating a machine learning algorithm on the basis of which we can say that one algorithm is better than the other. To do so, first, we have to build at least two machine learning algorithms so that we can compare their performance measures. In machine learning terminology, performance measures are known as evaluation metrics. However, before doing all this, first we have to define training and testing datasets and the same is done in Sec. 2.2. After that some evaluation metrics are discussed in Secs. 2.3 and 2.4.

What we have discussed in this unit is summarised in Sec. 2.5. Self-Assessment Questions (SAQs) have been given in some sections which are generally based on the content discussed in that section. But to give you a good practice of what we have discussed in this unit, some more problems based on the entire unit are given in Sec. 2.6 under the heading Terminal Questions. Due to the reason mentioned in Sec. 1.1 of Unit 1 of this course, solutions of all the SAQs and Terminal Questions are given in Sec. 2.7.

In the next unit, you will study the OC curve and the area under the curve (AUC) in detail.

Expected Learning Outcomes

After completing this unit, you should be able to:

- ❖ explain training and testing datasets;
- ❖ obtain a confusion matrix of an algorithm and some other evaluation metrics such as sensitivity, specificity, positive predictive value, negative predictive value, overall accuracy, balanced accuracy, etc.

2.2 PARTITION OF A DATASET INTO TRAINING AND TESTING DATASETS

From Sec. 1.2 of the previous unit, recall that the term "machine learning" was coined by computer scientist Arthur Samuel in 1959, and he defined machine learning as follows:

"The field of study that gives computers the ability to learn without being explicitly programmed". … (2.1)

If you know the meaning of "explicitly programmed" used in this definition, then good, but if you do not know its meaning, very good. Here, by very good, I mean do not lose heart, you are reading IGNOU study learning material (SLM) which is known for its three features: self-explanatory, self-contained and teacher-built-in. So, if you go through the SLM sequentially, then you will get the answers to most of your queries related to the learning part of the topics discussed in SLM. … (2.2)

Now, come to the point of what the meaning of **explicitly programmed** is. It means you have a complete flow chart for every possible situation and clear-cut instructions for each situation on what to do. The job of a computer is just to follow a given set of rules and perform operations accordingly. The list of all rules and operations in a logical sequence is prepared by a traditional computer programmer. For example, if you want to obtain the mean of the numbers 3, 6, 7, 4 using R software, then it can be done as follows:

```
> x1<-c(3, 6, 7, 4)
> mean(x1)
[1] 5
```

Now, here, first of all, the four numbers 3, 6, 7, 4 are stored in a vector x1. After that, we have applied the built-in function mean(). The programming of mean() function is explicitly programmed in the R software, i.e., the system will perform the operation $\frac{\text{Sum of all observations}}{\text{Number of observations}}$. So, hope this simple example has explained the meaning of explicitly programmed. … (2.3)

In this definition, the second important thing that you should know is the meaning of "**ability to learn**". The learning part in machine learning algorithms is dictated by the dataset, and the more data we have, the more the machine learning algorithm learns. So, the algorithm's learning ability increases with more and more data we have to train the algorithm. So, the moral of the story

is we need a **good amount of data not only for training an algorithm**, but we also need a **good amount of data to test the trained algorithm**. … (2.4)

To define these two datasets (i) Training dataset and (ii) Testing dataset is the objective of this section. So, it is the right place to meet that objective. This classification of the original dataset into two datasets: training and testing datasets is visualised in Fig. 2.1.

Training Dataset: The original dataset is partitioned into two datasets generally in the ratio 80 : 20 or 75 : 25 or 70 : 30. That is, the larger dataset contains 70 to 80% of the original dataset. This larger dataset, which is a random subset of the original dataset, is used to train the machine learning algorithm and is known as the training dataset. A machine learning algorithm tries to find insights into the training dataset and discovers patterns in the training dataset. So, the more training data an algorithm has, the more learning will take place and therefore more accuracy will be there when we will apply this trained algorithm to an unseen dataset. … (2.5)

Testing Dataset: The original dataset is partitioned into two datasets generally in the ratio 80 : 20 or 75 : 25 or 70 : 30. That is, the smaller dataset contains 20 to 30% of the original dataset. This smaller dataset, which is a random subset of the original dataset, is used to test the trained machine learning algorithm and is known as the test dataset. The test dataset is used to determine the performance of the trained model on an unseen dataset. … (2.6)

Before closing this section, one important point that you should keep in mind is that the difference between the performance measures of an algorithm between the training dataset and testing dataset should be less. If the algorithm performs well in the training dataset but poorly in the testing dataset, it is an indication of the problem of **overfitting**. … (2.7)

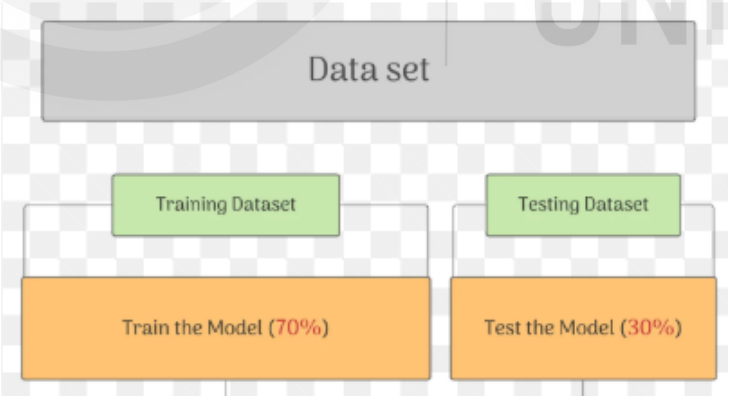

Fig. 2.1: Visualisation of the classification of the original dataset into two datasets

Now, you can try the following Self-Assessment Question.

SAQ 1

State whether the following statement is TRUE or FALSE. Give a reason in support of your answer.

Generally, a testing dataset is a random subset of the original dataset and contains 20 to 30% instances of the original dataset.

2.3 OVERALL ACCURACY OF AN ALGORITHM

As mentioned in Sec. 2.1 of this unit, the main objective of this unit is to explain some evaluation metrics of machine learning algorithms. We want to do it so that later on, we can use these metrics to compare two machine learning algorithms to determine which is better than the other with respect to a metric. So, first, we have to build two machine learning algorithms.

Let us consider the GaltonFamilies dataset in the HistData package of R. I have already installed this package in my system. So, I have to just load this package in my system and the same can be done as follows.

```
> library(HistData)
```

After that, we have to check how many rows and columns are in the GaltonFamilies dataset, which can be checked as follows.

```
> dim(GaltonFamilies)
[1] 934   8
```

… (2.8)

We see that there are 934 rows and 8 columns in this dataset. To see the first two rows of this dataset, we can run the following R code.

```
> head(GaltonFamilies, 2)
  family father mother midparentHeight children childNum gender childHeight
1    001   78.5     67          75.43        4        1   male       73.2
2    001   78.5     67          75.43        4        2 female       69.2
```

To give you a tiny flavour of a machine learning algorithm, we are going to build a machine learning algorithm where the sex of a child will be predicted on the basis of the given height of the child. So, we need only two variables, gender and childHeight, out of the 8 variables in this dataset. This can be done as follows.

```
> GaltonFamilies_2<-GaltonFamilies[, c(7, 8)]
```

Now, let us see the first two rows of this new data frame, which has only two columns.

```
  gender childHeight
1   male       73.2
2 female       69.2
```

Great, we need only two variables in this new dataset. Now, let us see how many male and female children there are. This can be obtained by running the following R code.

```
> table(GaltonFamilies_2$gender)

female   male
   453    481
```

… (2.10)

From a machine learning algorithm point of view, it is great that the ratio of male and female children is almost equal. In other words, we can say that this dataset is not biased from a gender point of view.

But dear learner, I want to explain an issue that you may face in your career as a data scientist. To explain that point, I have to start with a biased dataset

with respect to some variable. The technical word that is used to express this fact is prevalence, which you have already studied in Section 2.4 of Unit 2 of the course MST-019 (you may refer to page number 38). In this dataset prevalences of male and female children are given by … (2.11)

$$\text{Prevalence of male children in the study} = \frac{\text{Number of male children}}{\text{Total number of children}}$$

$$= \frac{481}{453+481} = \frac{481}{934} = 0.5149893$$

$$\text{Prevalence of female children in the study} = \frac{\text{Number of male children}}{\text{Total number of children}}$$

$$= \frac{453}{453+481} = \frac{453}{934} = 0.4850107$$

So, as mentioned earlier, the prevalence of male and female children is almost equal. This much difference is considered ok. But we want to make it biased with respect to the gender variable, so that as a data scientist, you should have an idea of the effect of prevalence in building a machine learning algorithm. So, to make it biased with respect to the gender variable, let us remove 300 rows of female children randomly from this dataset. This can be done as follows. … (2.12)

First of all, we have to set a seed for reproducibility purposes.

```
> set.seed(2006)    # for reproducibility
```

Next, we have to identify the positions of rows corresponding to female children. This can be done as follows.

```
> female_rows<- which(GaltonFamilies_2$gender == "female")
```

So, positions of rows of female children have been stored in the variable female_rows. If we want to see the first 10 positions of the female children, it can be seen by running the following R code.

```
> female_rows[1:10]
 [1]  2  3  4  7  8 10 13 14 15 19
```

Since the dataset GaltonFamilies_2 has 453 female children (you may refer to 2.10), so we expect that there should be 453 elements in the vector female_rows. Let us verify it.

```
> length(female_rows)
[1] 453
```

So, verification is done. Next, we have to select 300 row positions randomly out of 453, and it can be done using the sample() function as follows.

```
> set.seed(2006)
> rows_to_be_deleted<- sample(female_rows, 300)
```

Next, we have to remove rows corresponding to these randomly selected 300 rows from the data frame GaltonFamilies_2, and it can be done as follows.

```
> GaltonFamilies_300<-GaltonFamilies_2[- rows_to_be_deleted, ]
```

Now, we expect that the number of rows in GaltaonFamilies_300 should be 634 (= 934 – 300). Let us check it.

```
> dim(GaltonFamilies_300)
[1] 634   2
```
… (2.13)

Let us now check the number of male and female children in this dataset.

```
> table(GaltonFamilies_300$gender)

female   male
   153    481
```
… (2.14)

Finally, we now have a data frame GaltonFamilies_300, to work with.

In usual notations, we denote the feature variable by x and the target variable by y. Let us store values in x and y as follows.

```
> x<-GaltonFamilies_300$childHeight
> y<-GaltonFamilies_300$gender
```
… (2.15)

Next, to split the dataset GaltonFamilies_300 into train and test datasets, we will use createDataPartition() function of the caret package. I have already installed the caret package, so I have to just load it. Also, we will use the pipe operator, which lies in the package tidyverse. I have already installed the tidyverse package. So, I have to just load it. This time, if we want, we can use a different seed before running the createDataPartition() function. Let us use 2007.

```
> library(tidyverse)
> library(caret)
> set.seed(2007)
> test_index<-createDataPartition(y, times = 1, p = 0.5, list = FALSE)
```
… (2.16)

Let us explain the roles of the arguments of the createDataPartition() function.

- The times argument tells how many random samples of indices we want to create. Here, the times argument is 1, which means we need only one such sample. … (2.17)
- Argument p tells the proportion of the indices of the data. Here, p is 0.5 means we need 50% of the data. … (2.18)
- The argument list simply tells whether we need the output of the function as a list or not. If the list argument is set to TRUE, the output will be a list; here, we set it to FALSE, so the output will not be a list. … (2.19)

Now, using (2.16), we can obtain training and testing datasets using the following R code given as follows.

```
> test_dataset<-GaltonFamilies_300[test_index, ]
> train_dataset<-GaltonFamilies_300[- test_index, ]
```
… (2.20)

Let us first use the sample() function to decide sex of the child. We know that sample() function randomly selects a male or a female. So, sample() function is not using the feature childHeight in its prediction. It is a totally random selection. This can be done as follows.

```
> y_hat_1<-sample(c("male", "female"), length(test_index), replace = TRUE)%>%
+ factor(levels = levels(test_dataset$gender))
```

… (2.21)

Here, the target variable is gender, which is a categorical variable. We know that for a categorical variable, the suitable measure that can be used to check how many of them are predicted correctly is the proportion (you may refer to page number 30 of the course MST-019). This can be seen by running the following code. … (2.22)

```
> mean(y_hat_1 == test_dataset$gender)
[1] 0.4937107
```

… (2.23)

The result is close to 50% as expected, because we selected males and females randomly. This output may change run to run because we did not use any seed before running the R code (2.21). Note that here we have compared the values of the data object y_hat_1 with the values of the data object test_dataset$gender, not with train_dataset$gender, because evaluation of an algorithm is done using the test dataset, not the training dataset. The second point is that we created the y_hat_1 variable of length equal to the length of the variable test_index (you may refer to the code mentioned in 2.21).

But we can do better if we use the information available in the dataset GaltonFamilies_300. Let us obtain some information about this dataset using two statistical tools: the mean and standard deviation of male and female children's height separately. This can be obtained by running the following R code.

```
> GaltonFamilies_300%>%group_by(gender)%>%summarize(Mean_childHeight =
+ mean(childHeight), SD_of_childHeight = sd(childHeight))
# A tibble: 2 × 3
  gender Mean_childHeight SD_of_childHeight
  <fct>             <dbl>              <dbl>
1 female             64.2               2.41
2 male               69.2               2.62
```

… (2.24)

After getting this information, think for a while about how we can use this information to make a better prediction of the sex of the children.

Hope you have given some thought. One point that is striking in my mind is that the mean height of male children is 5 $(= 69.2 - 64.2)$ inches more than the mean height of female children. So, we should utilise this information in our prediction. If you have any other brilliant idea(s), then you should also give it(these) a try.

We know that the height of a population follows a normal distribution. We also know that in a normal distribution, 95% population lies within two standard deviations. But we take more than that, and consider all children having height greater than the mean minus two standard deviations as male.

So, $\text{mean} - 2(\text{Standard deviation}) = 69.2 - 2 \times 2.62 = 69.2 - 5.24 = 63.96$

Let us predict the gender of a child as male if the height is greater than 63.96 and female otherwise. This can be done by using the following R code.

```
> y_hat_2<-ifelse(x > 63.96, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
```

… (2.25)

Let us now check the overall accuracy.

```
> mean(y_hat_2 == y)
[1] 0.8359621
```
… (2.26)

Great! Overall accuracy has increased from 49.37% to 83.59%. It indicates that there may be some other cutoff value which can do better. We will try that. But at this point, one question that may arise in your mind is here (means in 2.26), we compared y_hat_2 with y, while in (2.23) we did a comparison of y_hat_1 with test_dataset$gender, not with y. Why did we do so? Good question. It shows your learning sincerity and hunger to dive deeper into the subject. The answer to this question is y_hat_2 is created using x (you may refer to 2.25), which is of length 634, and so the length of y_hat_2 is also 634. But the length of test_dataset$gender is 318. We can verify it using the following codes.

```
> length(x)
[1] 634
> length(y_hat_2)
[1] 634
> length(y)
[1] 634
> length(test_dataset$gender)
[1] 318
```

We know that in a normal distribution, within 3 standard deviations around the mean more than 99% values lie. So, let us obtain these values as follows.

$$\text{mean} \mp 3(\text{Standard deviation}) = 69.2 \mp 3 \times 2.62 = 69.2 \mp 7.86$$
$$= 61.34 \text{ and } 77.06.$$

So, we check for cutoff values 61, 62, 63, …, 77. First, we create a sequence of these numbers as follows and save it under the name of the cutoff variable.

```
> cutoff<- seq(from = 61, to = 77, by = 1)
```
… (2.27)

Now, we want to apply (2.25) for each value of the variable cutoff. That is, we want to apply a function to different values of an already created variable. Such a function that can perform this kind of job is the map_dbl() function of the purrr package. First, you have to install this package and then load it before applying the map_dbl() function. I have already installed it in my system. So, I only need to load it.

```
> library(purrr)
> overall_accuracy<- map_dbl(cutoff, function(x){
+ y_hat_3<-ifelse(train_dataset$childHeight > x, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
+ mean(y_hat_3 == train_dataset$gender)
+ })
```
… (2.28)

Let us see all values of overall accuracy just saved in the variable overall_accuracy.

```
> overall_accuracy
 [1] 0.7816456 0.8164557 0.8291139 0.8544304 0.8481013 0.8417722
 [7] 0.7911392 0.7151899 0.6234177 0.4968354 0.3924051 0.3196203
[13] 0.2753165 0.2531646 0.2500000 0.2500000 0.2468354
```

We see that the maximum value among these is 0.8544304. Let us find out at which value of cutoff it is obtained. This can be found out by running the

following code.

```
> best_cutoff<- cutoff[which.max(overall_accuracy)]
> best_cutoff
[1] 64
```
… (2.29)

We see that among the whole numbers 61, 62, 63, …, 77, the number 64 plays the role of the best cutoff. If we go in decimal numbers, then there may be some other number better than 64. If you are interested in getting that number, then you have to just replace the value of 'by' argument in (2.27) by 0.1 or 0.01 or 0.001, etc. and run the codes mentioned in (2.28) and (2.29), you will get the number of your interest.

Note that in (2.28), we have used the training data set, while, as mentioned in (2.4), we know that evaluation of a machine learning algorithm is done using a testing dataset instead of the training dataset. So, let us do that as follows.

```
> y_hat_4<-ifelse(test_dataset$childHeight > best_cutoff, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
```

Overall accuracy on the test dataset can be obtained as follows.

```
> mean(y_hat_4 == test_dataset$gender)
[1] 0.836478
```
… (2.30)

It is less than the overall accuracy (0.8544304) of the training dataset but much greater than the overall accuracy (0.4937107) of the guessing model.

Now, we move toward the learning side of the point that we discussed in (2.11) and (2.12). The point of concern with overall accuracy, especially when we have a biased dataset, is that it may hide the truth with respect to the actual prediction of each category (in the present case, individual accuracy of male and female children) of the variable of bias (in the present case gender is the variable of bias because we have more male than female children). So, our next goal is to obtain the accuracy of male and female children, individually. This point and some other evaluation metrics are discussed in the next section.
… (2.31)

Now, you can try the following Self-Assessment Question.

SAQ 2

Write any one line of R code where the pipe operator of the tidyverse package is used. After getting the output explain how pipe operator works. Objective of this SAQ is to tell you the silent power of pipe operator so that it can motivate you to apply pipe operator in your codes. Make a habit of applying pipe operator where possible It will make you smart in coding.

2.4 CONFUSION MATRIX AND EVALUATION METRICS

In this section, we have to continue the discussion of the previous section. Keeping our objective in view, we need four entries:

- Actually, female and machine learning algorithm also predicted female.
- Actually, female, but the machine learning algorithm predicted as male.

- Actually, male and machine learning algorithm also predicted male.
- Actually, male, but the machine learning algorithm predicted as female.

When these four entries are taken together in a 2 by 2 matrix form constitute what is known as a confusion matrix. So, the confusion matrix for the dataset discussed in the previous section can be obtained by running the following R code. The screenshot with output is shown as follows.

```
> table(Predicted = y_hat_4, Actual = test_dataset$gender)
         Actual
Predicted female male
   female     31    6
   male       46  235
```

(2.32)

Next, let us also obtain the margins sum of the above confusion matrix given by (2.32) by running the following R code.

```
> addmargins(table(Predicted = y_hat_4, Actual = test_dataset$gender))
         Actual
Predicted female male Sum
   female     31    6  37
   male       46  235 281
   Sum        77  241 318
```

… (2.33)

We see that actual male children are 241 and actual female children are 77. Also, the number of male children who are actually males and also predicted by the machine learning algorithm as male is 235. Similarly, the number of female children who are actually female and also predicted female is 31. So,

the accuracy of predicting male children correctly $\frac{235}{241} \simeq 0.9751$ and

the accuracy of predicting female children correctly $\frac{31}{77} \simeq 0.4026$

Using R code, we can obtain the same as follows.

```
> test_dataset%>%mutate(y_hat_4 = y_hat_4)%>%
+ group_by(gender)%>%
+ summarize(accuracy = mean(y_hat_4 == gender))
# A tibble: 2 × 2
  gender accuracy
  <fct>     <dbl>
1 female    0.403
2 male      0.975
```

You have noted the huge difference in the individual accuracy of male and female children 97.5% versus 40.3% respectively, while overall accuracy was 83.6% (you may refer to 2.30). This was the point I wanted to highlight: you should keep in mind when you compute overall accuracy, especially when you are working with a biased dataset. This happened because the prevalence of male children is much higher than the prevalence of female children in the GaltonFamilies_300 dataset. My objective in creating a biased dataset, GaltonFamilies_300, from the original dataset, GaltonFamilies, was to give

you an exposure to this important learning that you should have as a data scientist. … (2.34)

As promised in (2.30), in this section, other than the confusion matrix, we will also learn some other evaluation metrics which can also be used to notice the issues because of the presence of the problem of a higher prevalence of one category compared to the other category. Two such measures are sensitivity and specificity. Recall that you have already studied sensitivity and specificity in Unit 4 of the course MST-019. To refresh your memory, let us copy and paste Table 4.1 of the course MST-019 as follows.

Table 4.1 Result of Screening Test and Disease Status

		Disease Status		
		Yes (D^+)	**No (D^-)**	**Total**
Screening test result	T^+	a (True positive)	b (False positive)	a + b (Total number of participants who are tested positive by the test)
	T^-	c (False negative)	d (True negative)	c+ d (Total number of participants who are tested negative by the test)
	Total	a + c (Total screened participants who actually have disease)	b + d (Total screened participants who actually do not have disease)	a + b +c + d = N (Total number of screened participants)

Let us prepare a similar table by following the terminology of the present course on machine learning instead of epidemiology that was discussed in Block-1 of the Course MST-019. After doing that, Table 4.1 will look like this, and we give it the name Table 2.1.

Table 2.1 General Result of Algorithm Prediction and Actual Status

		Actual Status		
		Actual Positive	**Actual Negative**	**Total**
Algorithm Prediction	**Predicted Positive**	a True Positive (TP)	b False Positive (FP)	a + b The total of those who are predicted positive by the algorithm
	Predicted Negative	c False Negative (FN)	d True Negative (TN)	c + d The total of those who are predicted negative by the algorithm
	Total	a + c The total of those who are actually positive	b + d The total of those who are actually negative	a + b + c + d = N Grand Total

Now, we can define sensitivity and specificity as follows.

Sensitivity: The ability of an algorithm to correctly predict a positive outcome when it is actually positive. So, sensitivity is the probability, and in the terminology of the above table, it is defined as follows.

$$\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{a}{a + c} \quad \ldots (2.35)$$

Remark: Sensitivity is also known as the **True Positive Rate** (TPR). It has one more name **recall**.

Specificity: The ability of an algorithm to correctly predict a negative outcome when it is actually negative. So, specificity is the probability, and in the terminology of the above table, it is defined as follows.

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{d}{b + d} \quad \ldots (2.36)$$

Remark: Specificity is also known as the **True Negative Rate** (TNR).

Recall from Unit 4 of the course MST-019: in such a situation, we define two additional measures, the positive predictive value (PPV) and the negative predictive value (NPV), as follows.

Positive Predictive Value: The proportion of outcomes that are predicted positive by the algorithm that are actually positive. In the terminology of the above table, it is defined as follows.

$$\text{Positive predictive value} = \frac{TP}{TP + FP} = \frac{a}{a + b} \quad \ldots (2.37)$$

Remark: Positive predictive value is also known as **precision**.

Negative Predictive Value: The proportion of outcomes that are predicted negative by the algorithm that are actually negative. In the terminology of the above table, it is defined as follows.

$$\text{Negative predictive value} = \frac{TN}{TN + FN} = \frac{d}{c + d} \quad \ldots (2.38)$$

Let us add two more measures to our list of measures, known as balanced accuracy and F_1-score which are defined as follows.

Balanced accuracy is defined as the average of sensitivity and specificity, so

$$\text{Balanced accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2} \quad \ldots (2.39)$$

F_1-score is the harmonic mean of recall and precision, and so can be defined as follows.

$$F_1\text{-score} = 2\frac{(\text{Precision}) \times (\text{Recall})}{\text{Precision} + \text{Recall}} \quad \ldots (2.40)$$

Let us obtain these measures using the confusion matrix given by (2.33) as follows.

```
> addmargins(table(Predicted = y_hat_4, Actual = test_dataset$gender))
         Actual
Predicted female male Sum
   female     31    6  37
   male       46  235 281
   Sum        77  241 318
```

$$\text{Sensitivity (or Recall)} = \frac{TP}{TP+FN} = \frac{a}{a+c} = \frac{31}{77} \simeq 0.4026 = 40.26\%$$

$$\text{Specificity} = \frac{TN}{TN+FP} = \frac{d}{b+d} = \frac{235}{241} \simeq 0.9751 = 95.51\%$$

$$\text{Positive predictive value (or Precision)} = \frac{TP}{TP+FP} = \frac{a}{a+b} = \frac{31}{37} \simeq 0.8378$$

$$= 83.78\%$$

$$\text{Negative predictive value} = \frac{TN}{TN+FN} = \frac{d}{c+d} = \frac{235}{281} \simeq 0.8363 = 83.63\%$$

$$\text{Balanced accuracy} = \frac{0.4026 + 0.9751}{2} = \frac{1.3777}{2} = 0.68885 = 68.885\%$$

$$F_1\text{-score} = 2\frac{(\text{Precision}) \times (\text{Recall})}{\text{Precision} + \text{Recall}} = 2\frac{0.8378 \times 0.4026}{0.8378 + 0.4026} \simeq 0.5439$$

All these measures and some more can be obtained by using confusionMatrix() function of caret package as follows.

```
> confusionMatrix(data = y_hat_4, reference = test_dataset$gender)
Confusion Matrix and Statistics

          Reference
Prediction female male
    female     31    6
    male       46  235

               Accuracy : 0.8365
                 95% CI : (0.7912, 0.8754)
    No Information Rate : 0.7579
    P-Value [Acc > NIR] : 0.000429

                  Kappa : 0.4588

 Mcnemar's Test P-Value : 6.362e-08

            Sensitivity : 0.40260
            Specificity : 0.97510
         Pos Pred Value : 0.83784
         Neg Pred Value : 0.83630
             Prevalence : 0.24214
         Detection Rate : 0.09748
   Detection Prevalence : 0.11635
      Balanced Accuracy : 0.68885

       'Positive' Class : female
```

… (2.41)

Let us do one example.

Example 1: Consider the iris dataset in R. First, add a categorical column "Setosa" which takes the yes category if the Species column of that row has the entry 'setosa' and no otherwise and call this new dataset "iris_setosa". After that, split this new dataset randomly into 70:30, resulting in train_set and test_set. Train a logistic regression model with Setosa as a target variable while Sepal.Length and Sepal.Width as features. Construct a confusion matrix corresponding to this model. Hence, find sensitivity, specificity, overall accuracy, PPV, NPV, balanced accuracy and F_1-score.

Solution: We know that there are 150 rows and 5 columns in the iris dataset. This can be checked as follows.

```
> dim(iris)
[1] 150   5
```

The first two rows of the dataset iris can be seen as follows.

```
> head(iris, 2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
```

Let us first make a copy of the iris dataset and give it a new name iris_setosa. This can be done as follows.

```
> iris_setosa<-iris
```

Now, in this new dataset iris_setosa, let us create a new categorical column "Setosa" which takes the 'yes' category if the Species column of that row has the entry 'setosa' and 'no' otherwise. This can be done as follows.

```
> iris_setosa$Setosa <- ifelse(iris$Species == "setosa", "yes", "no")
```

Now, one column "Setosa" has been added to the original dataset iris. This can be seen as follows.

```
> dim(iris_setosa)
[1] 150   6
> head(iris_setosa, 2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Setosa
1          5.1         3.5          1.4         0.2  setosa    yes
2          4.9         3.0          1.4         0.2  setosa    yes
```

Note that now we have 6 columns instead of 5 in new dataset iris_Setosa_col.

Next, we have to select a random sample of 70% of row indices of this dataset, which can be done as follows. For reproducibility purposes, we will set 2006 as the seed.

```
> set.seed(2006)
> row_index<-sample(1: nrow(iris_setosa), 0.7*nrow(iris_setosa))
```

Now, we can obtain train and test sets as follows.

```
> train_set<-iris_setosa[row_index, ]
> test_set<-iris_setosa[- row_index, ]
```
… (2.42)

We know that 70% of 150 rows $= \frac{70}{100} \times 150 = 105$ rows and

30% of 150 rows $= \frac{30}{100} \times 150 = 45$ rows

So, we expect that the number of rows in the train_set and test_set datasets should be 105 and 45, respectively. This can be verified as follows.

```
> dim(train_set)
[1] 105   6
> dim(test_set)
[1] 45  6
```

Verified. Good.

To ensure consistent factor levels where yes remains first and no remains second, we have to run the following R code.

```
> train_set$Setosa <- factor(train_set$Setosa, levels = c("yes", "no"))
> test_set$Setosa  <- factor(test_set$Setosa,  levels = c("yes", "no"))
```
… (2.43)

We know that the logistic model predicts probabilities. We want that model to predict P(yes) = probability of yes. We also know that glm() function predicts the probability of the second level. Here, the second level is no. So, we have to relevel. This can be done as follows.

```
> train_set$Setosa <- relevel(train_set$Setosa, ref = "no")
```
… (2.44)

Now, we can train a logistic regression model with Setosa as a target variable while Sepal.Length and Sepal.Width as features as follows.

```
> log_model_1<-glm(Setosa ~ Sepal.Length + Sepal.Width,
+ data = train_set, family = binomial)
```
… (2.45)

You may get some warning messages. Ignore them.

Next, we need to obtain the model's predicted probabilities. Let us store these probabilities in the variable probabilities. It can be done as follows.

```
> probabilities <- predict(log_model_1, newdata = test_set, type = "response")
```
… (2.46)

To convert these probabilities into outcomes, we have to apply a classification function. This can be done as follows.

```
> predicted_classes <- ifelse(probabilities > 0.5, "yes", "no")
```
… (2.47)

Next, before obtaining the confusion matrix, we have to convert predicted_classes into a factor variable with levels yes and no. This can be done as follows.

```
> predicted_classes <- factor(predicted_classes, levels = c("yes", "no"))
```

… (2.48)

Finally, we can obtain a confusion matrix by running the following code. Let us save the output in the variable conf_matrix.

```
> conf_matrix <- table(Predicted = predicted_classes, Actual = test_set$Setosa)
```

… (2.49)

Now, the confusion matrix has been saved in the object conf_matrix, which can be seen by typing conf_matrix in the R console and hitting enter. The screenshot of the output is shown as follows.

```
> conf_matrix
         Actual
Predicted yes no
      yes  15  0
      no    0 30
```

… (2.50)

Now, if we compare the confusion matrix given by (2.50) with the general form of the confusion matrix given by (2.40), we have

$$TP = a = 15,\ FP = b = 0,\ FN = c = 0,\ TN = d = 30 \quad \ldots (2.51)$$

Now, required evaluation metrics are given by

$$\text{Sensitivity (or Recall)} = \frac{TP}{TP + FN} = \frac{a}{a + c} = \frac{15}{15 + 0} = \frac{15}{15} = 1 = 100\%$$

$$\text{Specificity} = \frac{TN}{TN + FP} = \frac{d}{b + d} = \frac{30}{0 + 30} = \frac{30}{30} = 1 = 100\%$$

$$\text{Overall accuracy} = \frac{TP + TN}{TP + FP + FN + TN} = \frac{a + d}{a + b + c + d} = \frac{15 + 30}{15 + 0 + 0 + 30}$$

$$= \frac{45}{45} = 1 = 100\%$$

$$\text{Positive predictive value (or Precision)} = \frac{TP}{TP + FP} = \frac{a}{a + b} = \frac{15}{15 + 0} = \frac{15}{15}$$

$$= 1 = 100\%$$

$$\text{Negative predictive value} = \frac{TN}{TN + FN} = \frac{d}{c + d} = \frac{30}{0 + 30} = \frac{30}{30} = 1 = 100\%$$

$$\text{Balanced accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2} = \frac{100 + 100}{2}\% = \frac{200}{2}\% = 100\%$$

$$F_1\text{-score} = 2\frac{(\text{Precision}) \times (\text{Recall})}{\text{Precision} + \text{Recall}} = 2\frac{100 \times 100}{100 + 100}\% = 100\%$$

All these measures and some more can be obtained by using confusionMatrix() function of caret package as follows.

```
> confusionMatrix(data = predicted_classes, reference = test_set$Setosa)
Confusion Matrix and Statistics

          Reference
Prediction yes no
       yes  15  0
       no    0 30

               Accuracy : 1
                 95% CI : (0.9213, 1)
    No Information Rate : 0.6667
    P-Value [Acc > NIR] : 1.191e-08

                  Kappa : 1

 Mcnemar's Test P-Value : NA

            Sensitivity : 1.0000
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 1.0000
             Prevalence : 0.3333
         Detection Rate : 0.3333
   Detection Prevalence : 0.3333
      Balanced Accuracy : 1.0000

       'Positive' Class : yes
```

Now, you can try the following Self-Assessment Question.

SAQ 3

Re do Example 1, just this time make an even split as train and test datasets. That is, make 50 : 50 split instead of 70 : 30 split. Do not obtain evaluation metrics manually like we did in Example 1. I assume that you can make a good practice of that from the TEE point of view as per your need. Like Example 1, just obtain those evaluation metrics using the confusionMatrix() function of the caret package. Comment on the output of the confusionMatrix() function by comparing the output with the output of Example 1.

2.5 SUMMARY

A brief summary of what we have covered in this unit is given as follows:

- **Training Dataset**: The original dataset is partitioned into two datasets generally in the ratio 80 : 20 or 75 : 25 or 70 : 30. That is, the larger dataset contains 70 to 80% of the original dataset. This larger dataset, which is a random subset of the original dataset, is used to train the machine learning algorithm and is known as training dataset.
- **Testing Dataset**: The original dataset is partitioned into two datasets generally in the ratio 80 : 20 or 75 : 25 or 70 : 30. That is, the smaller dataset contains 20 to 30% of the original dataset. This smaller dataset, which is a random subset of the original dataset, is used to test the trained machine learning algorithm and is known as the test dataset.

- If the algorithm performs well in the training dataset but poorly in the testing dataset, it is an indication of the problem of **overfitting**.
- **Sensitivity or True Positive Rate (TPR) or Recall**: The ability of an algorithm to correctly predict a positive outcome when it is actually positive. So, sensitivity is the probability that is defined as follows:
  $\text{Sensitivity} = \frac{TP}{TP + FN} = \frac{a}{a + c}$.
- **Specificity or True Negative Rate (TNR)**: The ability of an algorithm to correctly predict a negative outcome when it is actually negative. So, specificity is the probability that is defined as follows:
  $\text{Specificity} = \frac{TN}{TN + FP} = \frac{d}{b + d}$
- **Positive Predictive Value or Precision**: The proportion of outcomes that are predicted positive by the algorithm that are actually positive. So, positive predictive value is the probability and is defined as follows:
  $\text{Positive predictive value or Precision} = \frac{TP}{TP + FP} = \frac{a}{a + b}$
- **Negative Predictive Value**: The proportion of outcomes that are predicted as negative by the algorithm that are actually negative. So, negative predictive value is the probability and is defined as follows.
  $\text{Negative predictive value} = \frac{TN}{TN + FN} = \frac{d}{c + d}$
- **Balanced accuracy** is defined as the average of sensitivity and specificity, so $\text{Balanced accuracy} = \frac{\text{Sensitivity} + \text{Specificity}}{2}$.
- F_1-score is the harmonic mean of recall and precision, and so can be defined as follows: $F_1\text{-score} = 2\frac{(\text{Precision}) \times (\text{Recall})}{\text{Precision} + \text{Recall}}$.

2.6 TERMINAL QUESTIONS

1. Build an algorithm to predict the gender of a child using the childHeight variable as a feature of the GaltonFamilies dataset in the HistData package of R by removing 200 rows randomly of female children from this dataset. Find the overall accuracy of the algorithm.
2. Obtain female and male children's overall accuracy individually and comment on the results obtained by comparing these results with the results of Sec. 2.3 of the similar model, where we removed 300 random rows of female children instead of 200 rows.

2.7 SOLUTIONS/ANSWERS

Self-Assessment Questions (SAQs)

1. It is a false statement because the training dataset is a random subset of the original dataset is ok, but it contains 70 to 80% instances of the original dataset, not 20 to 30% instances.

2. A screenshot of one sample of a line of code of R with output where the pipe operator from the tidyverse package is used is given as follows.

```
> iris %>% subset(Sepal.Length > 5) %>% head(4)
   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1           5.1         3.5          1.4         0.2  setosa
6           5.4         3.9          1.7         0.4  setosa
11          5.4         3.7          1.5         0.2  setosa
15          5.8         4.0          1.2         0.2  setosa
```

Working of pipe operator is like this:

- Whatever we have before the first pipe operator it becomes the first argument of the function just written after the first pipe operator. For example, here iris is written before the pipe operator, so it becomes the first argument of the function subset() which is written just after the first pipe operator.

- Whatever is written before the second pipe operator it will be evaluated and its output will be the first argument of the second pipe operator R. Here first R will subset the dataset iris and select only those rows of the iris dataset where the variable Sepal.Length is greater than 5. This subset of iris dataset will be the first argument of the function head() which is written just after the second pipe operator in this line of code. Therefore final output will be the first 4 rows of the subseted iris dataset.

3. We have explained the meaning of all codes in Example 1, so here we are just running the codes and getting the required output. The screenshot of the code with output is shown as follows.

```
> library(dplyr)
> head(iris, 2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species
1          5.1         3.5          1.4         0.2  setosa
2          4.9         3.0          1.4         0.2  setosa
> iris_setosa<-iris
> iris_setosa$Setosa <- ifelse(iris$Species == "setosa", "yes", "no")
> dim(iris_setosa)
[1] 150   6
> head(iris_setosa, 2)
  Sepal.Length Sepal.Width Petal.Length Petal.Width Species Setosa
1          5.1         3.5          1.4         0.2  setosa    yes
2          4.9         3.0          1.4         0.2  setosa    yes
> set.seed(2006)
> row_index<-sample(1: nrow(iris_setosa), 0.5*nrow(iris_setosa))
> train_set<-iris_setosa[row_index, ]
> test_set<-iris_setosa[- row_index, ]
> train_set$Setosa <- factor(train_set$Setosa, levels = c("yes", "no"))
> test_set$Setosa  <- factor(test_set$Setosa,  levels = c("yes", "no"))
> dim(train_set)
[1] 75  6
> dim(test_set)
[1] 75  6
> train_set$Setosa <- relevel(train_set$Setosa, ref = "no")
> log_model_1<-glm(Setosa ~ Sepal.Length + Sepal.Width,
+ data = train_set, family = binomial)
```

```
> probabilities <- predict(log_model_1, newdata = test_set, type = "response")
> predicted_classes <- ifelse(probabilities > 0.5, "yes", "no")
> predicted_classes <- factor(predicted_classes, levels = c("yes", "no"))
> conf_matrix <- table(Predicted = predicted_classes, Actual = test_set$Setosa)
> conf_matrix
         Actual
Predicted yes no
      yes  23  0
      no    1 51
> library(caret)
> confusionMatrix(data = predicted_classes, reference = test_set$Setosa)
Confusion Matrix and Statistics

          Reference
Prediction yes no
       yes  23  0
       no    1 51

               Accuracy : 0.9867
                 95% CI : (0.9279, 0.9997)
    No Information Rate : 0.68
    P-Value [Acc > NIR] : 9.954e-12

                  Kappa : 0.969

 Mcnemar's Test P-Value : 1

            Sensitivity : 0.9583
            Specificity : 1.0000
         Pos Pred Value : 1.0000
         Neg Pred Value : 0.9808
             Prevalence : 0.3200
         Detection Rate : 0.3067
   Detection Prevalence : 0.3067
      Balanced Accuracy : 0.9792
```

Note that this time model incorrectly classifies one case of Setosa, while all non-Setosa cases are correctly classified. So, the specificity of the model is 100% while the sensitivity of the algorithm is 23/24 = 95.83%. What can we say about the reduction of sensitivity from 100% to 95.83% compared to Example 1? One possible answer is algorithm got less data for training compared to Example 1. In Example 1 training dataset has 105 instances, while this time we have only 75 instances in the training dataset. So, we can say that the more the dataset for training, the more learning takes place.

Terminal Questions

1. We have already explained all steps used in the process of building such an algorithm in Sec. 2.3. So, we are just running the final R codes on R console. Screenshot of R codes and their output is shown as follows.

```
> library(HistData)
> GaltonFamilies_2<-GaltonFamilies[, c(7, 8)]
> head(GaltonFamilies_2,2)
  gender childHeight
1   male        73.2
2 female        69.2
> table(GaltonFamilies_2$gender)

female   male
   453    481
```

```
> set.seed(2006)    # for reproducibility
> # to get row indices of female children
> set.seed(2006)    # for reproducibility
> female_rows<- which(GaltonFamilies_2$gender == "female")
> set.seed(2006)
> rows_to_be_deleted<- sample(female_rows, 200)
> GaltonFamilies_200<-GaltonFamilies_2[- rows_to_be_deleted, ]
> dim(GaltonFamilies_200)
[1] 734   2
> table(GaltonFamilies_200$gender)

female   male
   253    481
> x<-GaltonFamilies_200$childHeight
> y<-GaltonFamilies_200$gender

> library(caret)
> set.seed(2007)
> test_index<-createDataPartition(y, times = 1, p = 0.5, list = FALSE)
> test_dataset<-GaltonFamilies_200[test_index, ]
> train_dataset<-GaltonFamilies_200[- test_index, ]
> GaltonFamilies_200%>%group_by(gender)%>%summarize(Mean_childHeight =
+ mean(childHeight), SD_of_childHeight = sd(childHeight))
# A tibble: 2 × 3
  gender Mean_childHeight SD_of_childHeight
  <fct>             <dbl>              <dbl>
1 female             64.1               2.36
2 male               69.2               2.62
> y_hat_1<-ifelse(x > 63.96, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
> mean(y_hat_1 == y)
[1] 0.7915531
> cutoff<- seq(from = 61, to = 77, by = 1)
> library(purrr)
> overall_accuracy<- map_dbl(cutoff, function(x){
+ y_hat_2<-ifelse(train_dataset$childHeight > x, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
+ mean(y_hat_2 == train_dataset$gender)
+ })

> overall_accuracy
 [1] 0.7049180 0.7486339 0.7704918 0.8005464 0.8278689 0.8469945 0.8005464 0.7431694
 [9] 0.6721311 0.5573770 0.4699454 0.4180328 0.3770492 0.3579235 0.3524590 0.3524590
[17] 0.3497268

> best_cutoff<- cutoff[which.max(overall_accuracy)]
> best_cutoff
[1] 66
> y_hat_3<-ifelse(test_dataset$childHeight > best_cutoff, "male", "female")%>%
+ factor(levels = levels(test_dataset$gender))
> mean(y_hat_3 == test_dataset$gender)
[1] 0.8369565
> table(Predicted = y_hat_3, Actual = test_dataset$gender)
         Actual
Predicted female male
   female    101   34
   male       26  207
> addmargins(table(Predicted = y_hat_3, Actual = test_dataset$gender))
         Actual
Predicted female male Sum
   female    101   34 135
   male       26  207 233
   Sum       127  241 368
```

2. Female and male children's overall accuracy individually can be obtained by running the following code.

```
> test_dataset%>%mutate(y_hat_3 = y_hat_3)%>%
+ group_by(gender)%>%
+ summarize(accuracy = mean(y_hat_3 == gender))
# A tibble: 2 × 2
  gender accuracy
  <fct>     <dbl>
1 female    0.795
2 male      0.859
```

Note that this time also overall accuracy of male children is more than female children but they do not have as much huge difference as they have in Sec. 2.3 where we removed 300 rows of the female children instead of 200. It means if there is more gap between prevalence of female and male children then their overall accuracy will also has more gap.