University of BRISTOL

# An introduction to classification

Learning functions which map feature vectors to class labels

Henry W J Reeve

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.

Statistical Computing & Empirical Methods (EMATM0061)

MSc in Data Science, Teaching block 1, 2021.

# What will we cover today?

- We will begin by introducing the concept of classification.
- We will emphasize the importance of predictive performance on unseen data.
- We will consider the supervised learning pipeline and the role of the test-train split.
- We will use probabilistic ideas to understand the classification problem.
- We will emphasize the difference the difference between train and test error.
- We will also discuss the fundamental concept of a Bayes classifier.

# What is classification?

Learning a function $\phi : \mathcal{X} \to \mathcal{Y}$

which takes as input a feature vector $X \in \mathcal{X}$

and returns a categorical variable $\phi(X) \in \mathcal{Y}$ also known as a label.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# What is classification?

Example 1: Sentiment analysis

A company wants to automatically classify social media posts as being either “positive” or “negative” in sentiment.

“The food was fantastic! ”

“The service was disappointing”

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# What is classification?

<u>Example 2</u>: Image classification

A biologist wants to automatically classify images of fish according to which species they belong to.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# What is classification?

<u>Example 3</u>: Digit recognition

The postal service needs an automatic system for converting hand-written addresses into digitized addresses.


$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# What is classification?

<u>Example 3</u>: Automated medical diagnosis

A medical doctor wants to establish an automated procedure for classifying retinal blood vessels as normal or abnormal.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features — Classification rule — Categorical output

# What is classification?

Learning a function $\phi : \mathcal{X} \to \mathcal{Y}$ known as a **classification rule**.

which takes as input a feature vector $X \in \mathcal{X}$

and returns a categorical variable $\phi(X) \in \mathcal{Y}$

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# What is binary classification?

Learning a function $\phi : \mathcal{X} \to \mathcal{Y}$ known as a **classification rule**.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y} = \{0, 1\}$$

Features | Classification rule | Categorical output (label)

A binary classification problem is one with just two possible outcomes

<u>Example</u>: In automated medical diagnosis "normal" vs. "abnormal".

# How can we create a classification rule?

Let’s suppose we want **classification rule** implemented within a computer.

<u>The rule-based approach</u>

We could attempt to program a detailed set of rules:

e.g. “The rainbow shark has two large eyes”...

**Problems**:

This approach would be incredibly labor intensive.

New problems would require new rules.

Performs poorly in practice.

Brittle e.g. what if we can’t see both eyes etc.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features — Classification rule — Categorical output

# How can we create a classification rule?

Let's suppose we want **classification rule** implemented within a computer.

<u>The rule-based approach</u>

We could attempt to program a detailed set of rules:

**Problem**: Highly labor intensive

<u>The statistical learning approach</u>

We set instead we design learning algorithms so that the machine can learn to classify from data.

$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features | Classification rule | Categorical output

# How can we create a classification rule?

The statistical learning approach is to program our machine to learn tasks from data.

<u>ML proverb</u>

Why teach a computer to classify fish, when you can teach a computer to learn how to classify fish?

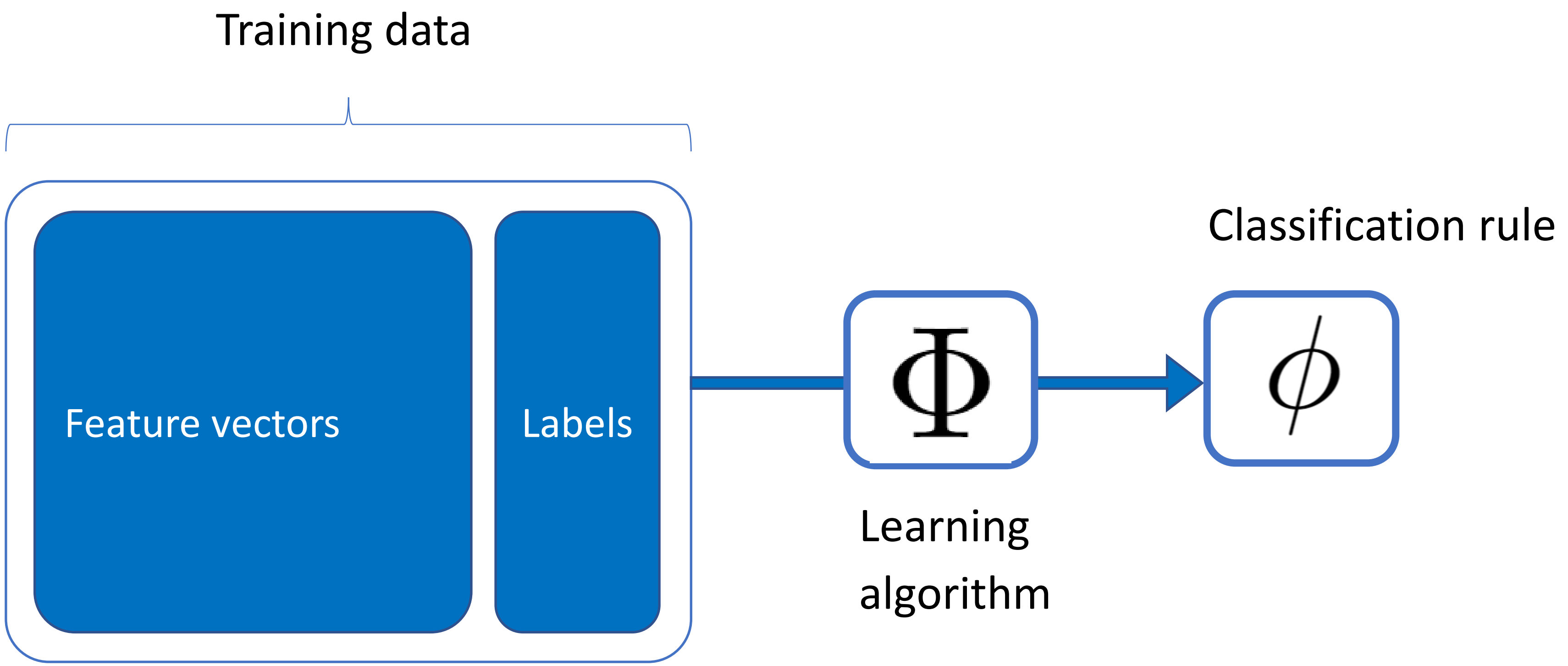
$$X \in \mathcal{X} \longrightarrow \boxed{\phi} \longrightarrow \phi(X) \in \mathcal{Y}$$

Features — Classification rule — Categorical output

# Supervised learning

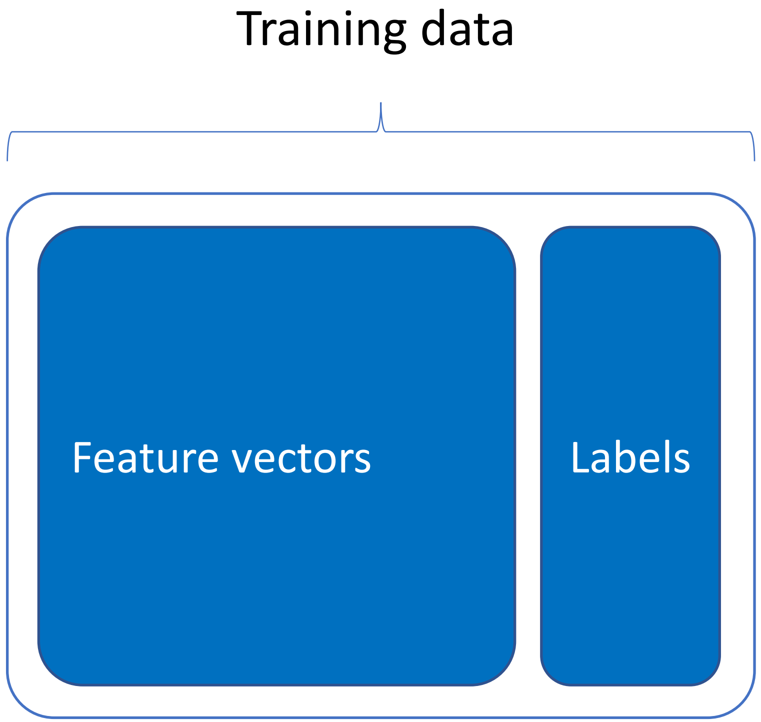
We learn a **classification rule** based on a set of training data $\mathcal{D}$ .

**Training data** is passed to a **learning algorithm** which outputs a classification rule.



# Supervised learning

We learn a set of **classification rule** based on a set of training data $\mathcal{D}$ .




**Training data** consists of a set of labelled data

$$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$$

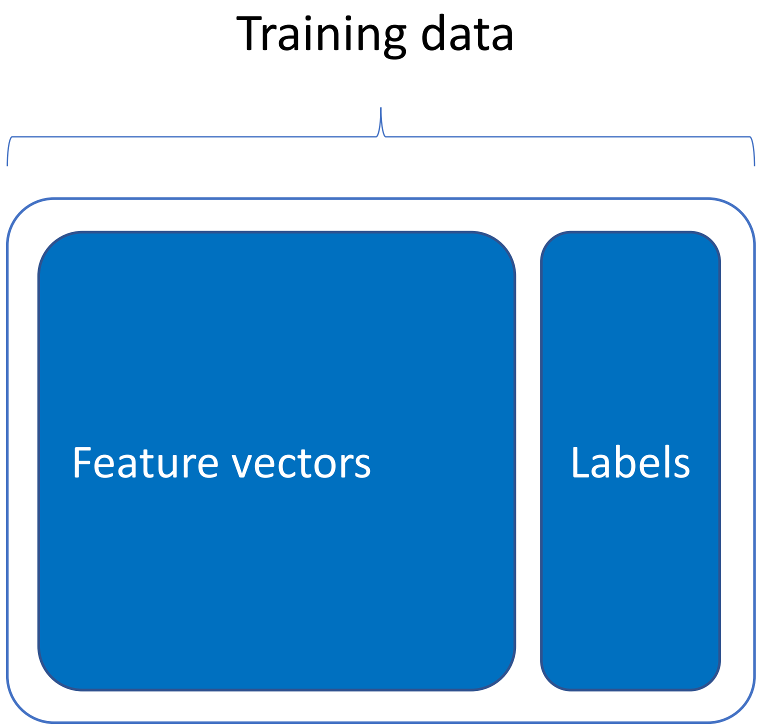
A sequence of ordered pairs $(X_i, Y_i)$ .

$X_i$ is a feature vector.

$Y_i$ is a label associated with $X_i$ .

# Supervised learning

We learn a set of **classification rule** based on a set of training data $\mathcal{D}$ .




**Training data** consists of a set of labelled data

$$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$$

<u>Example</u>

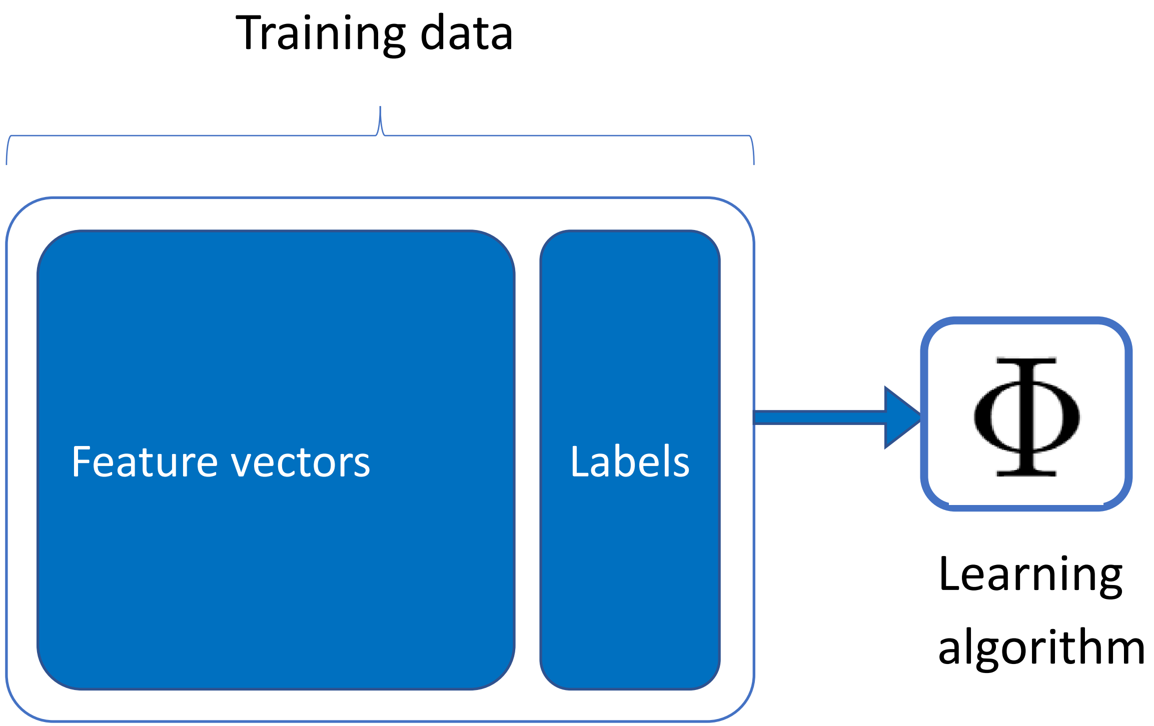
$X_i$ is an image of a particular fish.

$Y_i$ is a label corresponding to the species of the fish.

# Supervised learning

We learn a **classification rule** based on a set of training data $\mathcal{D}$ .

**Training data** is passed to a **learning algorithm.**




The learning algorithm

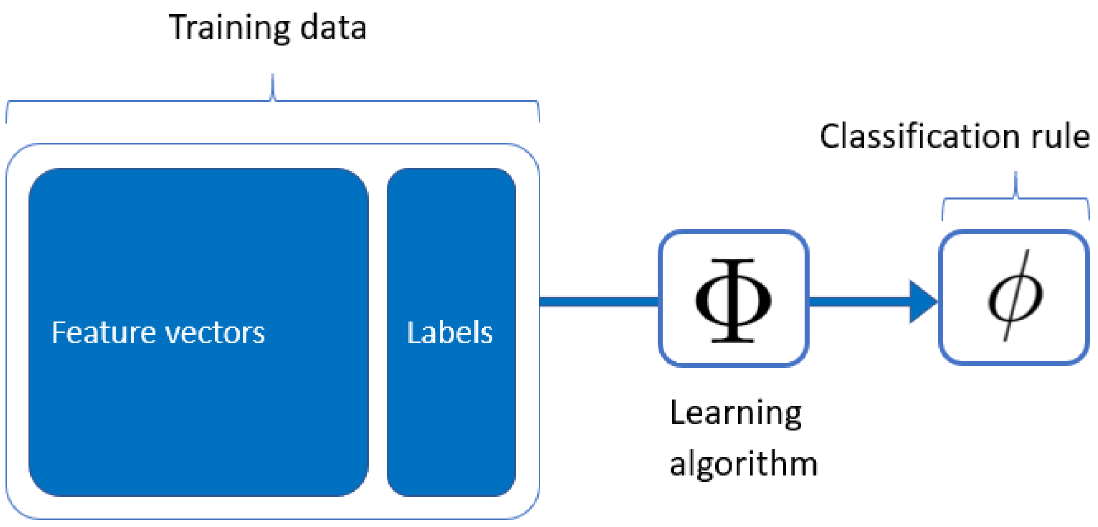
automatically identifies patterns

within the training data $\mathcal{D}$ .

# Supervised learning

We learn a set of **classification rule** based on a set of training data $\mathcal{D}$ .

**Training data** is passed to a **learning algorithm** which outputs a classification rule.




The classification rule is a mapping:

$$\phi : X \mapsto Y$$

A classification rule is also known as a classifier.

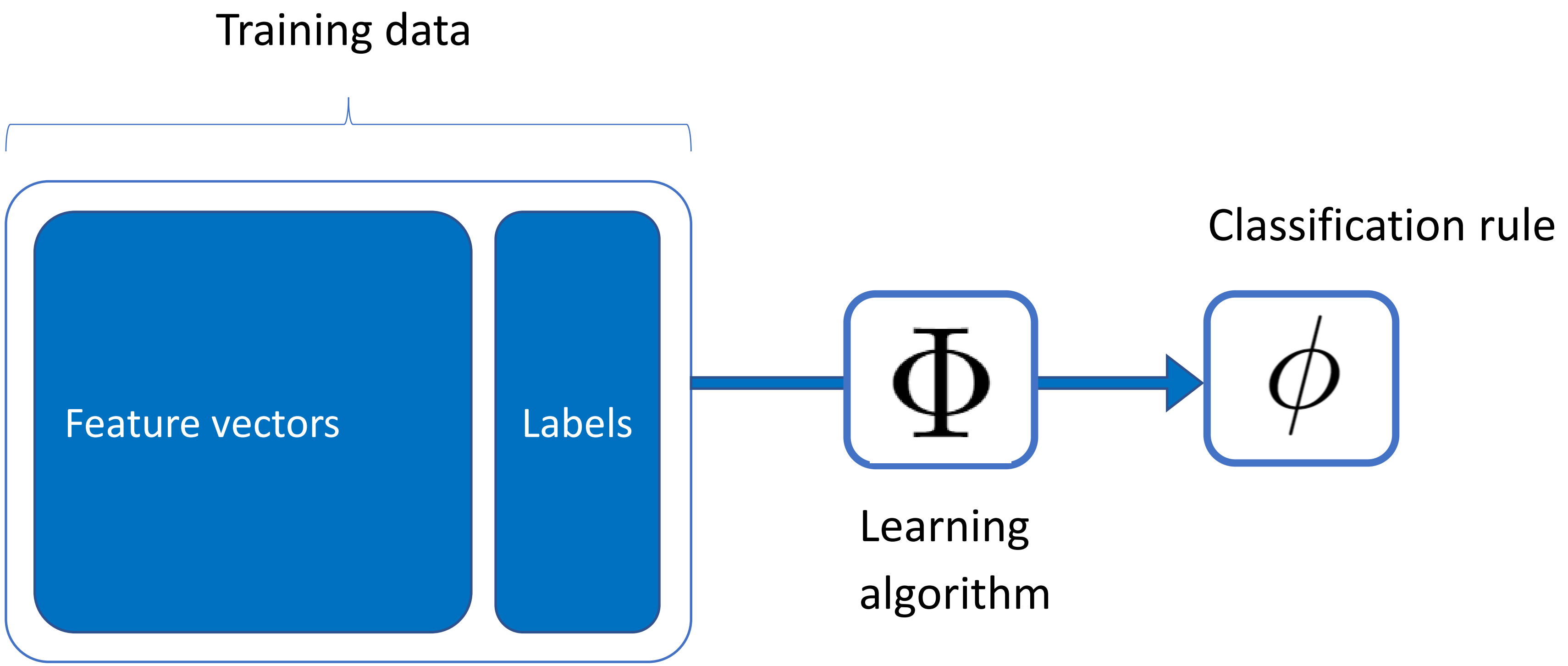
Should reflect the structure of the training data:

$$\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$$

# Supervised learning

We learn a **classification rule** based on a set of training data $\mathcal{D}$ .

**Training data** is passed to a **learning algorithm** which outputs a classification rule.



# Now take a break!

# Learning vs. memorization

Our goal is to learn a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ .

The classification rule $\phi$ should map feature vectors $X \in \mathcal{X}$ to labels $\phi(X) \in \mathcal{Y}$ .

**Key point:** The classification rule should perform well on unseen feature vectors $X \in \mathcal{X}$ .

Not just on the training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ .

<u>Example</u>:

We want the fish classification rule to correctly determine the fish species for new images…

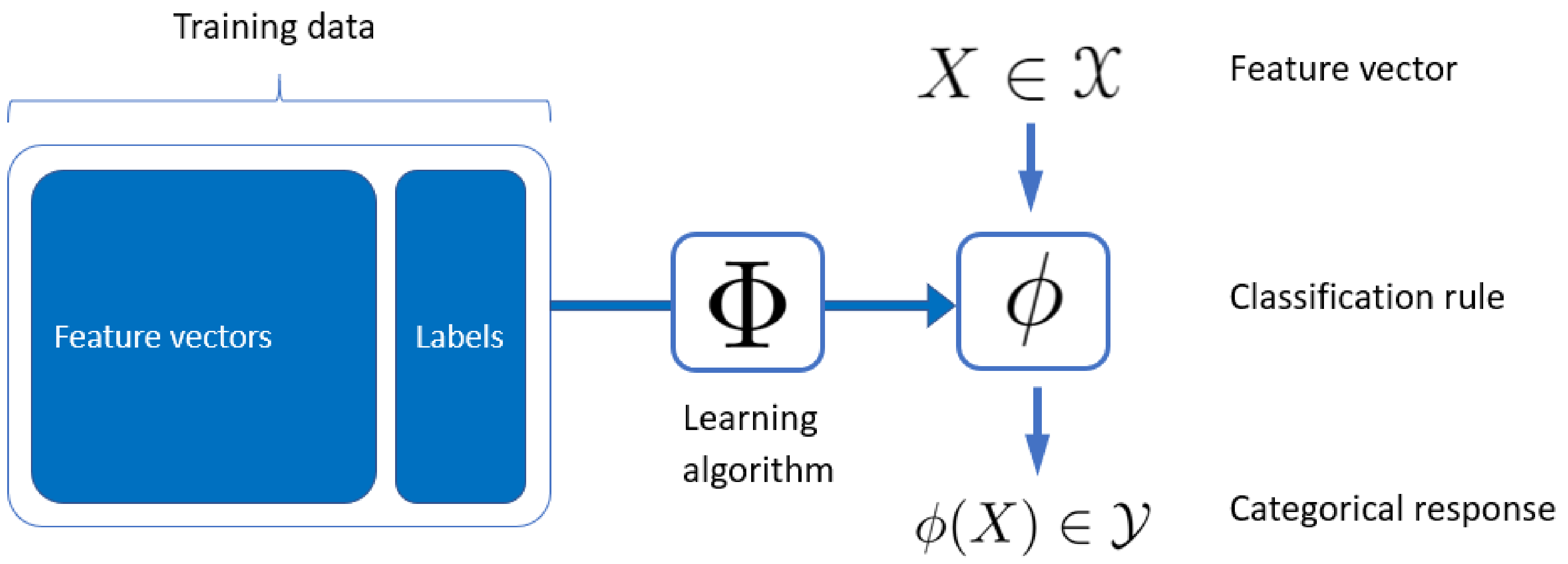
… not just the images within the training data.

This is the crucial difference between learning and memorization.

# Supervised learning

We learn a set of **classification rule** based on a set of training data $\mathcal{D}$ .

**Training data** is passed to a **learning algorithm** which outputs a classification rule.

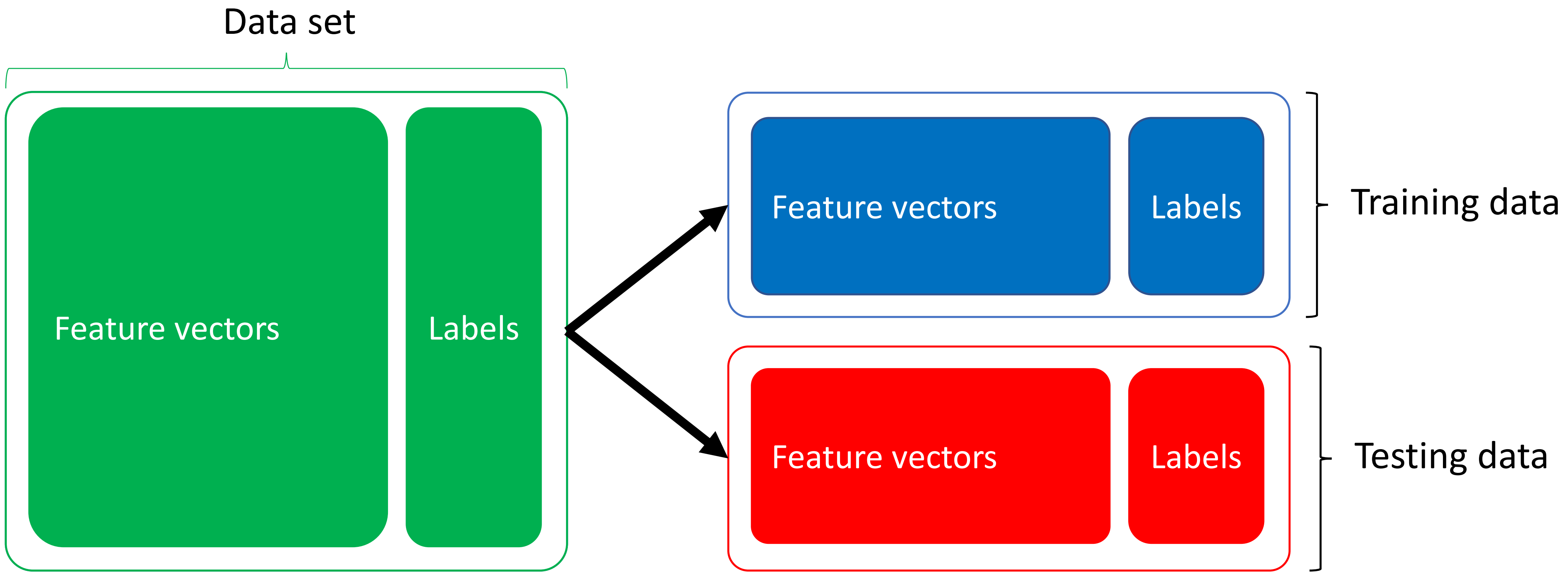



For the classification rule to be successful it must perform well on **unseen** data.

# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

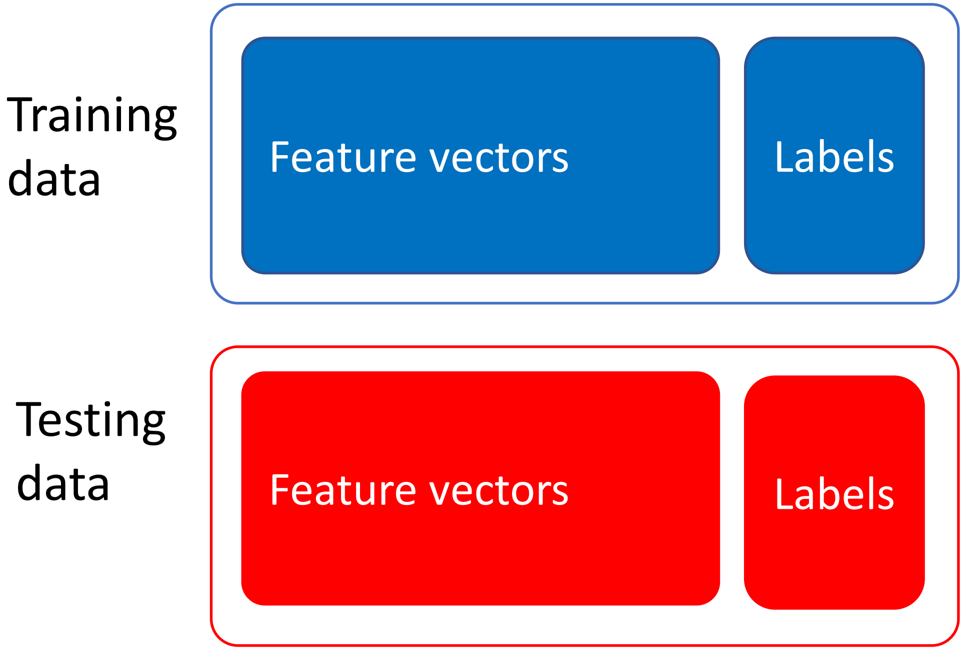
In order to investigate learning algorithms, we always need to do a test train split.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

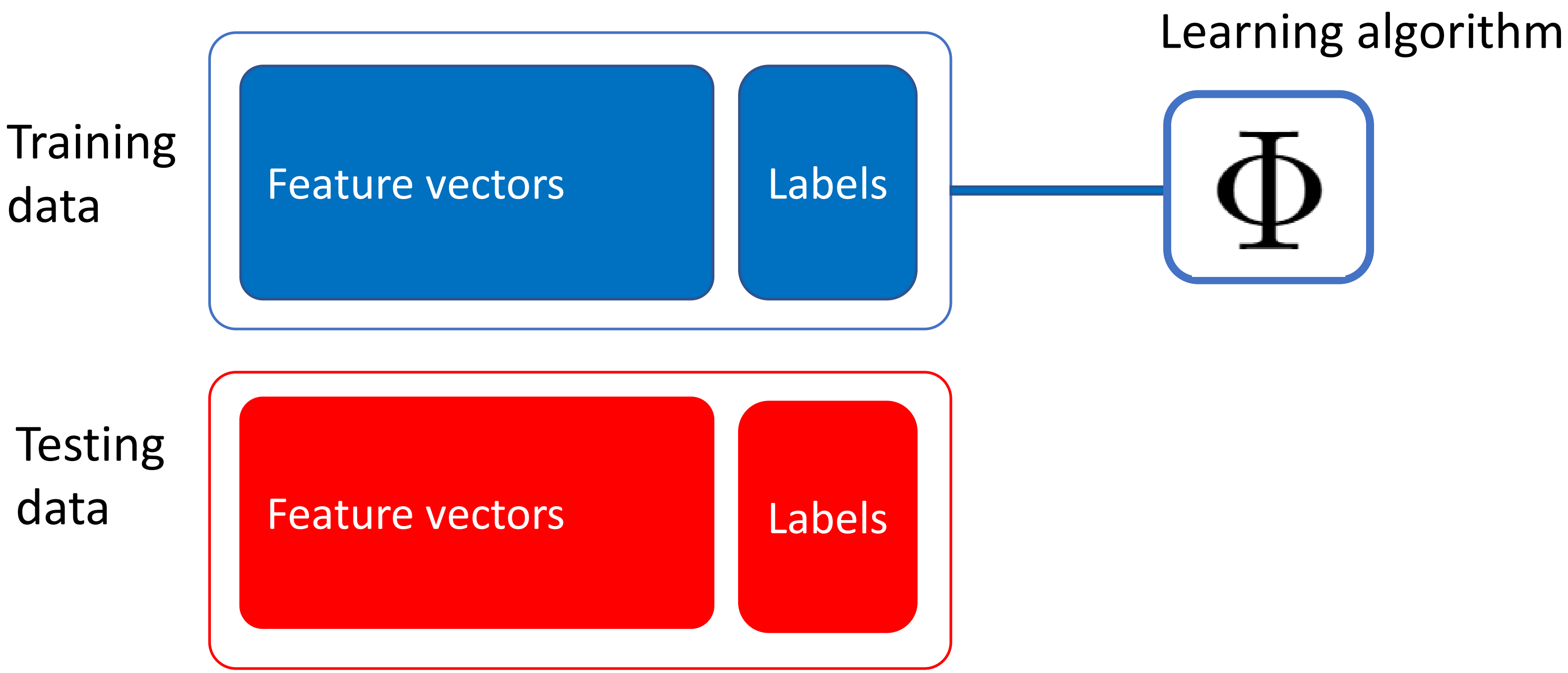
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

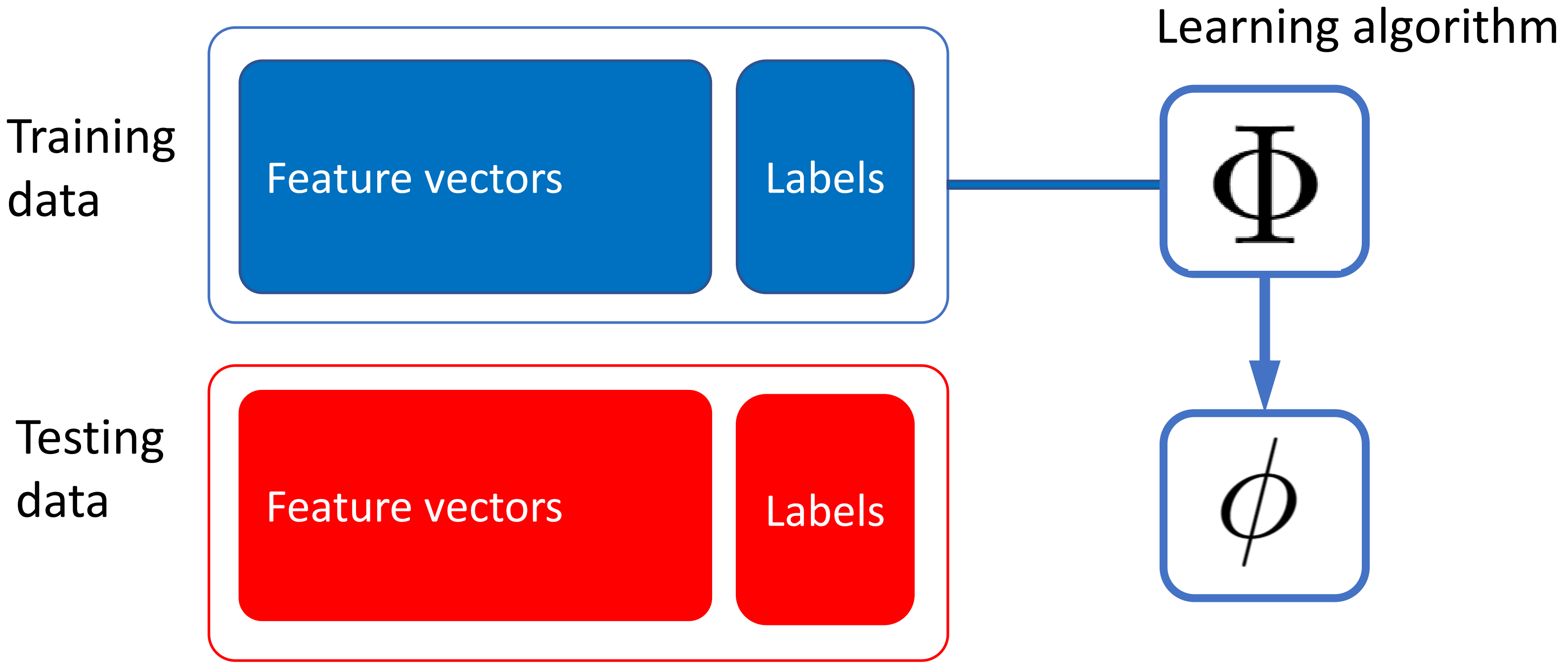
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

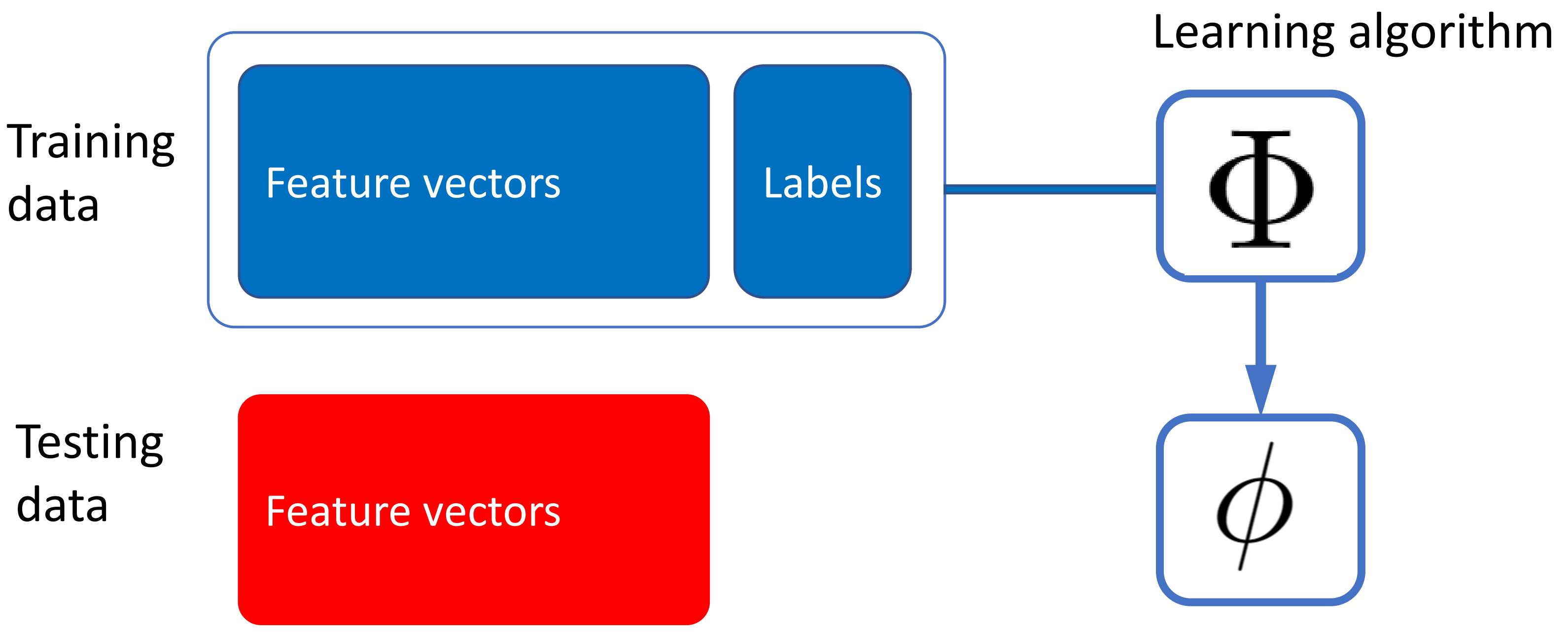
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

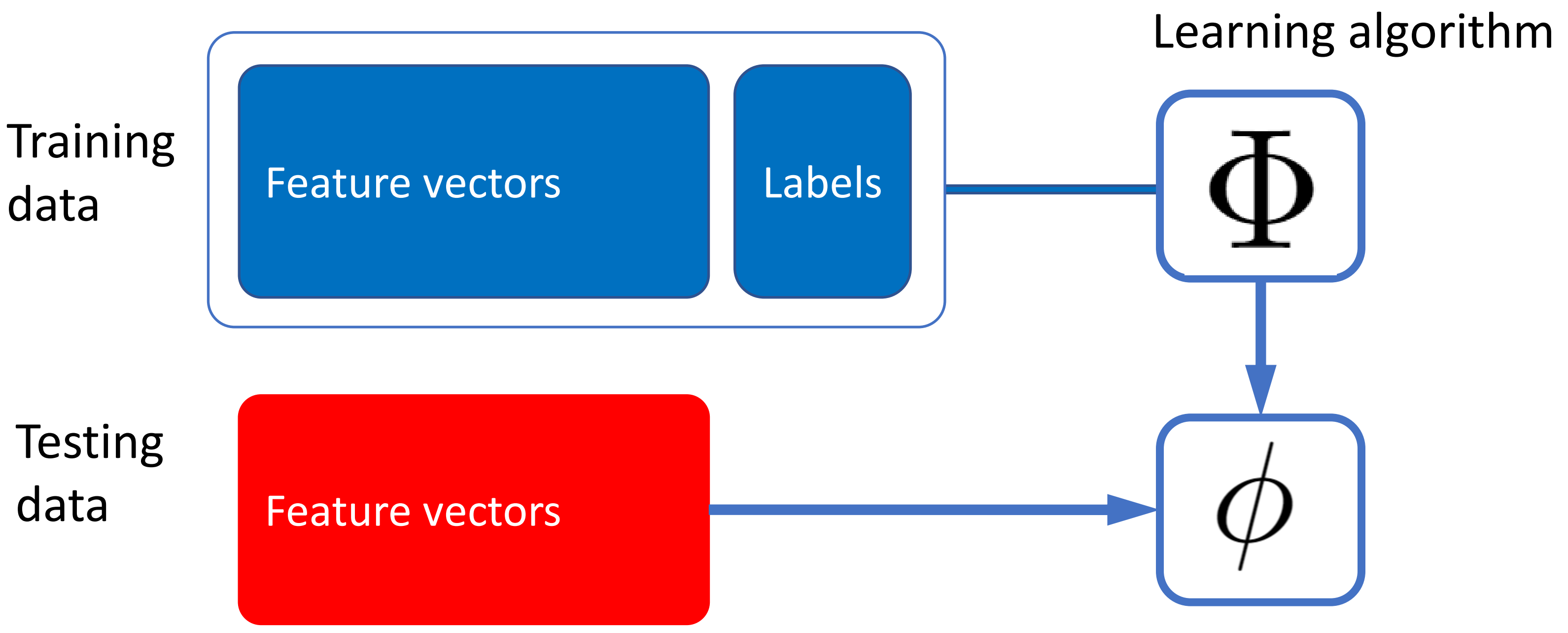
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

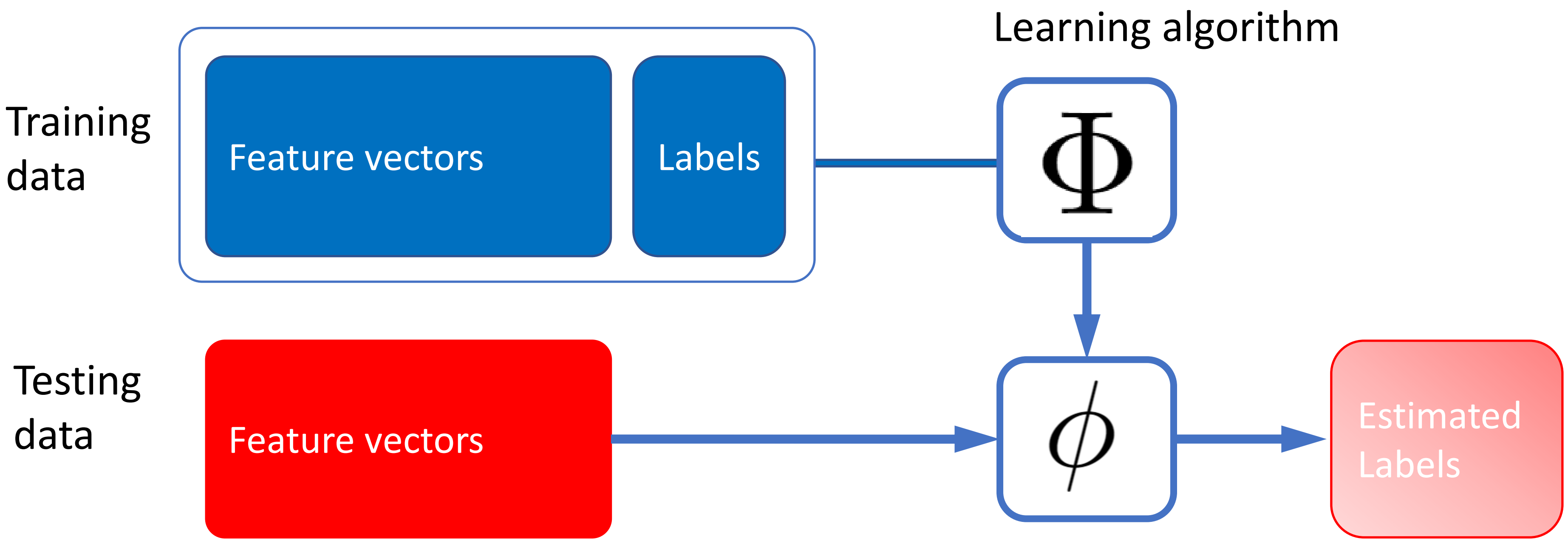
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

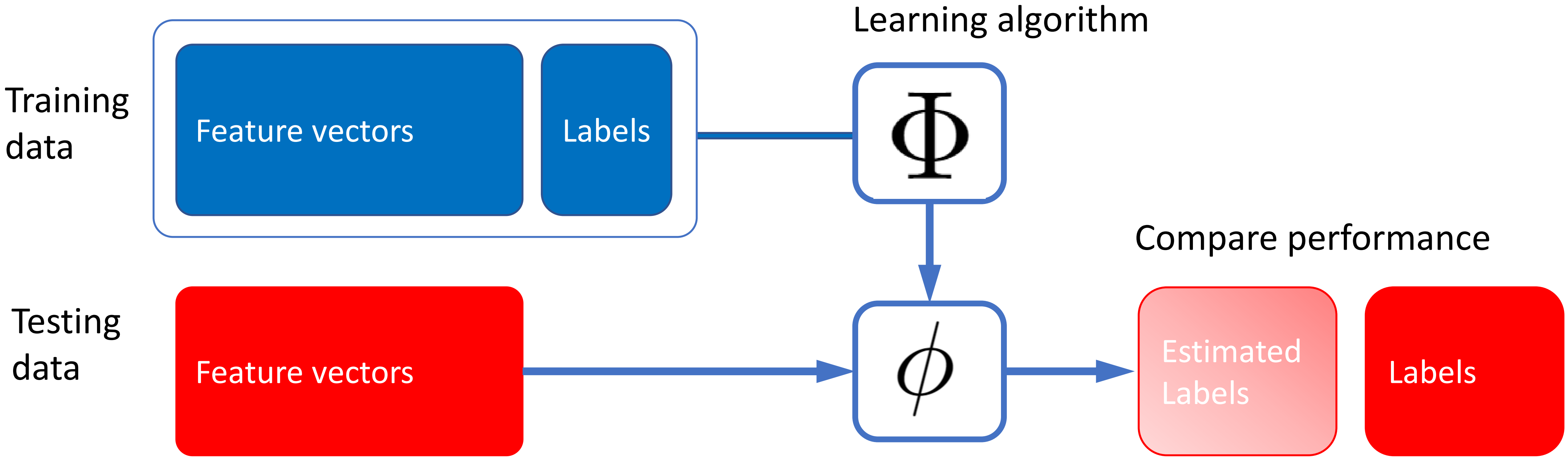
We use the train test split to assess our learning algorithms.



# The train test split

For the classification rule to be successful it must perform well on **unseen** data.

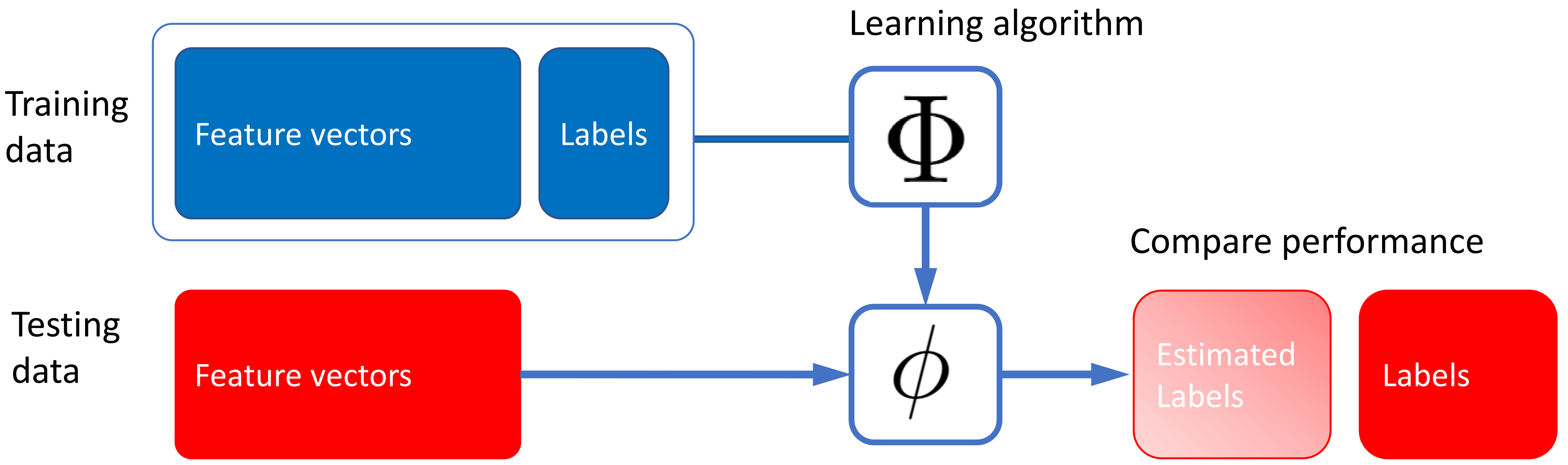
We use the train test split to assess our learning algorithms.



# The train test split

**Key point:** Never use your test data to learn your classifier!

The goal of the test data is to see how well the classification rule does on unseen data.



# Example: Penguin classification

Suppose we want to learn a classifier $\phi : \mathcal{X} \to \mathcal{Y}$ which takes a feature vector of morphological features and predicts whether a penguin belongs to either the Adelie species or the Chinstrap species.

<u>Features</u>: $$X = (X^1, X^2) \in \mathcal{X} = \mathbb{R}^2$$

$$X^1 = \text{ the weight of the penguin (grams).}$$

$$X^2 = \text{ the flipper length of the penguin (mm).}$$

<u>Labels</u>: $$Y \in \mathcal{Y} = \{0, 1\}$$

$$Y = \begin{cases} 1 \text{ if the penguin is an Adelie} \\ 0 \text{ if the penguin is a Chinstrap.} \end{cases}$$

# Example: Penguin classification

Suppose we want to learn a classifier $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ which takes a feature vector of morphological features and predicts whether a penguin belongs to either the Adelie species or the Chinstrap species.

```
library(tidyverse)
library(palmerpenguins)

peng_total<-penguins%>% # prepare our data
  select(body_mass_g,flipper_length_mm,species)%>%
  filter(species!="Gentoo")%>%
  drop_na()%>%
  mutate(species=as.numeric(species=="Adelie"))
```

# Example: Penguin classification

Suppose we want to learn a classifier $\phi : \mathcal{X} \rightarrow \mathcal{Y}$ which takes a feature vector of morphological features and predicts whether a penguin belongs to either the Adelie species or the Chinstrap species.

```
library(tidyverse)
library(palmerpenguins)

peng_total<-penguins%>% # prepare our data
  select(body_mass_g,flipper_length_mm,species)%>%
  filter(species!="Gentoo")%>%
  drop_na()%>%
  mutate(species=as.numeric(species=="Adelie"))
```

```
peng_total
```

```
## # A tibble: 219 x 3
##    body_mass_g flipper_length_mm species
##          <int>             <int>   <dbl>
##  1        3750               181       1
##  2        3800               186       1
##  3        3250               195       1
##  4        3450               193       1
##  5        3650               190       1
##  6        3625               181       1
##  7        4675               195       1
##  8        3475               193       1
##  9        4250               190       1
## 10        3300               186       1
## # ... with 209 more rows
```

# Example: Penguin classification

Suppose we want to learn a classifier $\phi : \mathcal{X} \to \mathcal{Y}$ which takes a feature vector of morphological features and predicts whether a penguin belongs to either the Adelie species or the Chinstrap species

$$\overbrace{\text{body\_mass\_g} \quad \text{flipper\_length\_mm}}^{X} \quad \overbrace{\text{species}}^{Y}$$

```
## # A tibble: 219 x 3
##    body_mass_g flipper_length_mm species
##          <int>             <int>   <dbl>
##  1        3750               181       1
##  2        3800               186       1
##  3        3250               195       1
##  4        3450               193       1
##  5        3650               190       1
```

Feature vector $X = (X^1, X^2) \in \mathcal{X} = \mathbb{R}^2$

$$X^1 = \text{ the weight of the penguin (grams).}$$

$$X^2 = \text{ the flipper length of the penguin (mm).}$$

Label $Y \in \mathcal{Y} = \{0, 1\}$

$$Y = \begin{cases} 1 \text{ if the penguin is an Adelie} \\ 0 \text{ if the penguin is a Chinstrap.} \end{cases}$$

# Example: Penguin classification

Now let’s carry out a train test split.

```
num_total<-peng_total%>%nrow() # number of penguin data
num_train<-floor(num_total*0.75) # number of train examples
num_test<-num_total-num_train # number of test samples

set.seed(1) # set random seed for reproducibility
test_inds<-sample(seq(num_total),num_test) # random sample of test indicies
train_inds<-setdiff(seq(num_total),test_inds) # training data indicies

peng_train<-peng_total%>%filter(row_number() %in% train_inds) # train data
peng_test<-peng_total%>%filter(row_number() %in% test_inds) # test data
```

Remember to set a random seed for reproducibility.

# Example: Penguin classification

Now let's carry out a train test split.

```
num_total<-peng_total%>%nrow() # number of penguin data
num_train<-floor(num_total*0.75) # number of train examples
num_test<-num_total-num_train # number of test samples

set.seed(1) # set random seed for reproducibility
test_inds<-sample(seq(num_total),num_test) # random sample of test indicies
train_inds<-setdiff(seq(num_total),test_inds) # training data indicies

peng_train<-peng_total%>%filter(row_number() %in% train_inds) # train data
peng_test<-peng_total%>%filter(row_number() %in% test_inds) # test data
```

We can also separate out the feature vectors and labels.

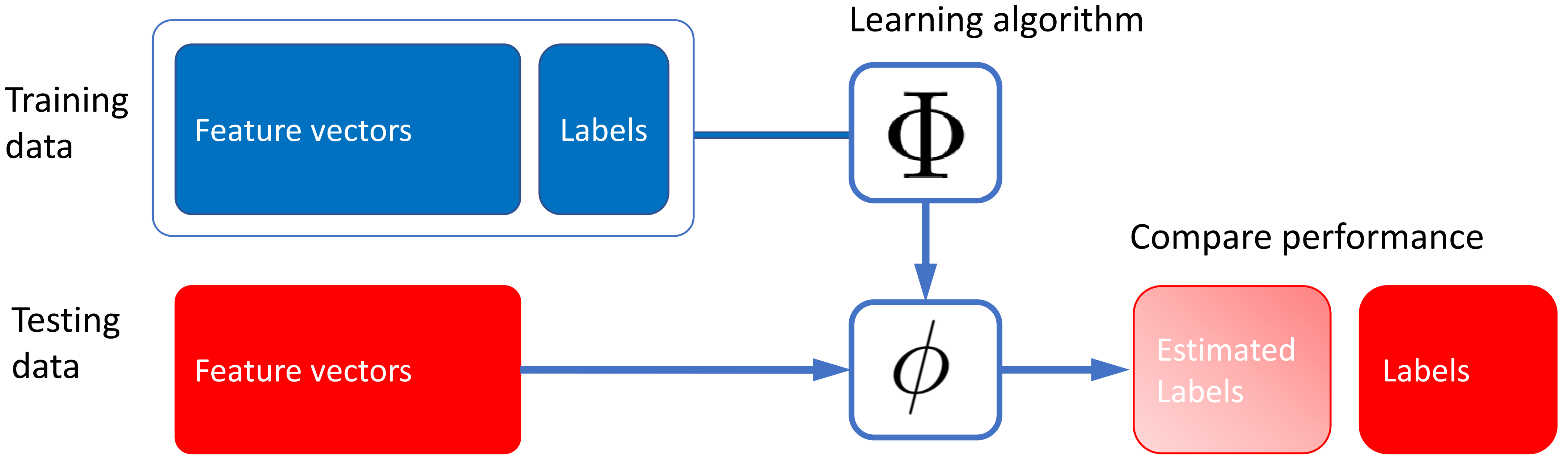
```
peng_train_x<-peng_train%>%select(-species) # train feature vectors
peng_train_y<-peng_train%>%pull(species) # train labels

peng_test_x<-peng_test%>%select(-species) # test feature vectors
peng_test_y<-peng_test%>%pull(species) # test labels
```

# The train test split

**Key point:** Never use your test data to learn your classifier!

The goal of the test data is to see how well the classification rule does on unseen data.



# Now take a break!

# A probabilistic model for classification

We begin with a feature space $\mathcal{X}$. In the simplest case this will be $\mathcal{X} = \mathbb{R}^d$.

We have a finite set of categories $\mathcal{Y}$. In the simplest case this will be $\mathcal{Y} = \{0, 1\}$.

We then have random variables $(X, Y)$

$X$ is a feature vector which takes values in $\mathcal{X}$.

$Y$ is a label which takes values in $\mathcal{Y}$.

The random variables $(X, Y) \sim \mathrm{P}$ have joint distribution $\mathrm{P}$.

# A probabilistic model for classification

We begin with a feature space $\mathcal{X}$. In the simplest case this will be $\mathcal{X} = \mathbb{R}^d$.

We have a finite set of categories $\mathcal{Y}$. In the simplest case this will be $\mathcal{Y} = \{0, 1\}$.

We then have random variables $(X, Y) \sim \mathrm{P}$ with joint distribution $\mathrm{P}$.

In addition we have some training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$.

We assume that examples $(X_i, Y_i) \sim \mathrm{P}$ are independent and identically distributed.

This will let us learn properties about the underlying distribution of $(X, Y) \sim \mathrm{P}$.

# Measuring performance

We have random variables $(X, Y) \sim \mathrm{P}$ with joint distribution $\mathrm{P}$.

Our goal of the learn a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ , also known as classifier,

such that $\phi(X) \approx Y$ for typical $(X, Y) \sim \mathrm{P}$.

We quantify our performance with the **expected test error**

$$\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right).$$

A good classifier $\phi : \mathcal{X} \to \mathcal{Y}$ is one with a low expected test error.

# Measuring performance

We quantify our performance with the **test error**

$$\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right).$$

This is the average number of mistakes on **unseen** data.

A good classifier $\phi : \mathcal{X} \to \mathcal{Y}$ is one with a low test error.

This weights all types of errors equally. Sometimes we weight different types of errors differently.

In practice we are also interested in computational issues e.g. Training/testing time and memory.

# The Bayes classifier

We quantify our performance with the **test error**

$$\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right).$$

The **Bayes classifier** or Bayes classification rule is the **best possible** classifier from an error perspective,

$$\mathcal{R}(\phi^*) = \min\left\{\mathcal{R}(\phi) : \quad \phi : \mathcal{X} \to \mathcal{Y} \text{ is a classifier}\right\}.$$

That is, $\phi^* : \mathcal{X} \to \mathcal{Y}$ is a classifier which **minimizes the test error** over all possible classifiers.

# The Bayes classifier

The **Bayes classifier** $\phi^* : \mathcal{X} \to \mathcal{Y}$ minimizes the test error over all possible classifiers

$$\mathcal{R}(\phi^*) = \min \{\mathbb{P}(\phi(X) \neq Y) : \quad \phi : \mathcal{X} \to \mathcal{Y} \text{ is a classifier}\} .$$

Let's think about the binary case where $\mathcal{Y} = \{0, 1\}$ .

We can define the Bayes classifier in terms of probability as follows,

$$\phi^*(x) := \begin{cases} 1 & \text{if} \quad \mathbb{P}(Y = 1|X = x) \geq \mathbb{P}(Y = 0|X = x) \\ 0 & \text{if} \quad \mathbb{P}(Y = 0|X = x) > \mathbb{P}(Y = 1|X = x). \end{cases}$$

# Now take a break!

# Learning from data

The **Bayes classifier** $\phi^* : \mathcal{X} \to \mathcal{Y}$ minimizes the test error over all possible classifiers

$$\mathcal{R}(\phi^*) = \min\left\{\mathbb{P}(\phi(X) \neq Y) : \quad \phi : \mathcal{X} \to \mathcal{Y} \text{ is a classifier}\right\}.$$

In an ideal world our computer would already know a lot about the distribution of $(X, Y) \sim \mathrm{P}$

If this were the case, we could just use mimic the Bayes classifier $\phi^* : \mathcal{X} \to \mathcal{Y}$.

Unfortunately, the computer doesn't have prior knowledge of $(X, Y) \sim \mathrm{P}$.

Instead we rely upon learning algorithms to learn information about the underlying distribution from the training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ i.i.d. with $(X_i, Y_i) \sim \mathrm{P}$.

# Test error vs. train error

The true goal is to find a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ with a low test error,

$$\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right) \text{ with } (X, Y) \sim \mathrm{P}.$$

Unfortunately, we can't directly observe the test error $\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right)$ .

However, we can use the training data $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$

to compute the **train error**

$$\hat{\mathcal{R}}_n(\phi) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left\{\phi(X_i) \neq Y_i\right\}.$$

# Test error vs. train error

Given a classifier $\phi : \mathcal{X} \to \mathcal{Y}$ we can't directly compute the **test error**

$$\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right) \approx \text{ Average number of mistakes on unseen data.}$$

We can use $\mathcal{D} = ((X_1, Y_1), \cdots, (X_n, Y_n))$ to compute the **train error**

$$\hat{\mathcal{R}}_n(\phi) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\left\{\phi(X_i) \neq Y_i\right\} \approx \text{ Average number of mistakes on training data.}$$

**Key point:** The train error and the test error are <u>not the same</u>.

However, we can use information about the training data to achieve a low-test error.

# Test error vs. train error

Given $\phi : \mathcal{X} \to \mathcal{Y}$ we can't directly compute the **test error** $\mathcal{R}(\phi) := \mathbb{P}\left(\phi(X) \neq Y\right)$

We can use $\mathcal{D}$ to compute the **train error** $\hat{\mathcal{R}}_n(\phi) := \frac{1}{n}\sum_{i=1}^{n} \mathbb{1}\left\{\phi(X_i) \neq Y_i\right\}.$

<u>Example</u>: Image classification

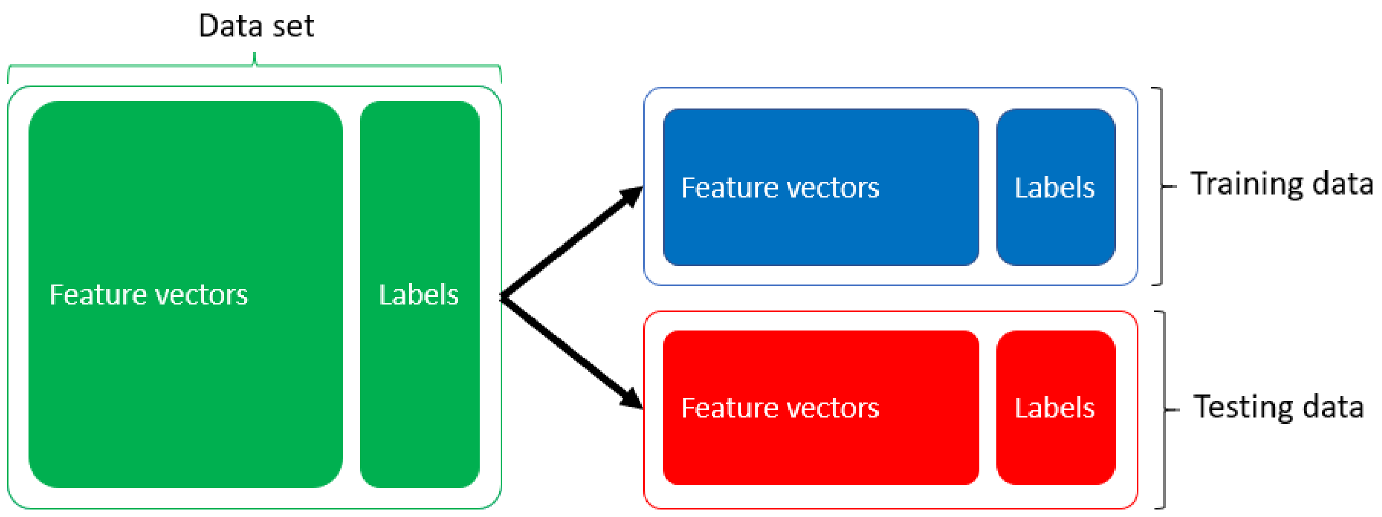
Train error - average number of misclassified images in the training data.

Test error – average number of misclassified future as yet unseen images.

Our real goal is to do well on the unseen data!

# The train test split

For the classification rule to be successful it must perform well on **unseen** data.



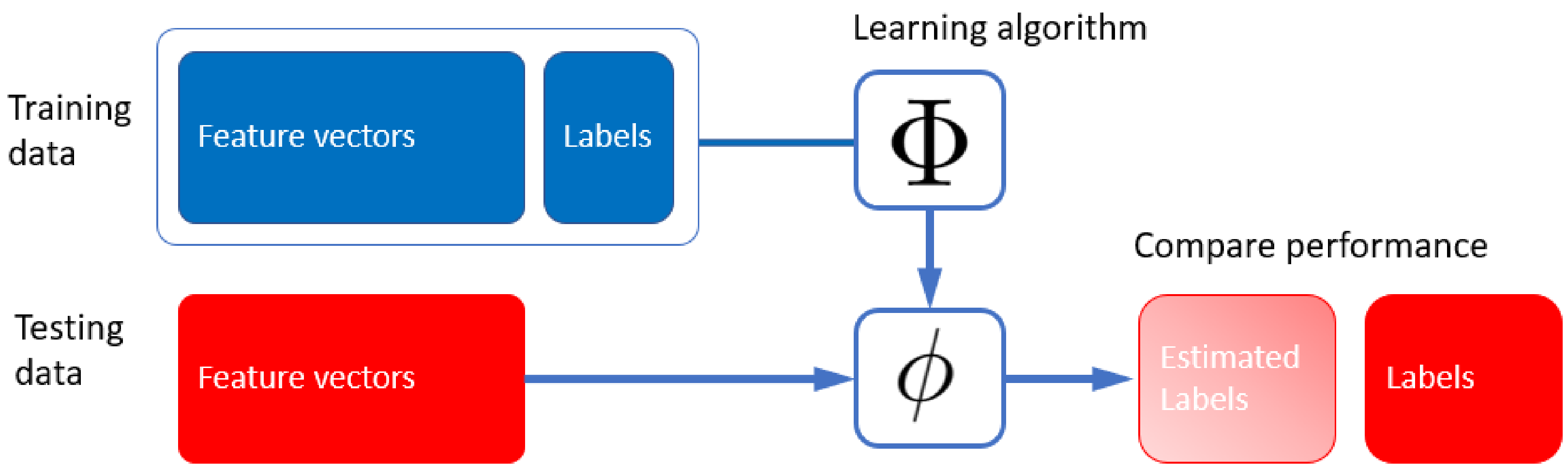

We will use the test data to estimate the test error:

$$\mathcal{R}(\phi) = \mathbb{P}\left(\phi(X) \neq Y\right) \approx \text{ Average number of mistakes on unseen data.}$$

Not the same as the train error: The average number of mistakes on the training data.

# The train test split

For the classification rule to be successful it must perform well on **unseen** data.



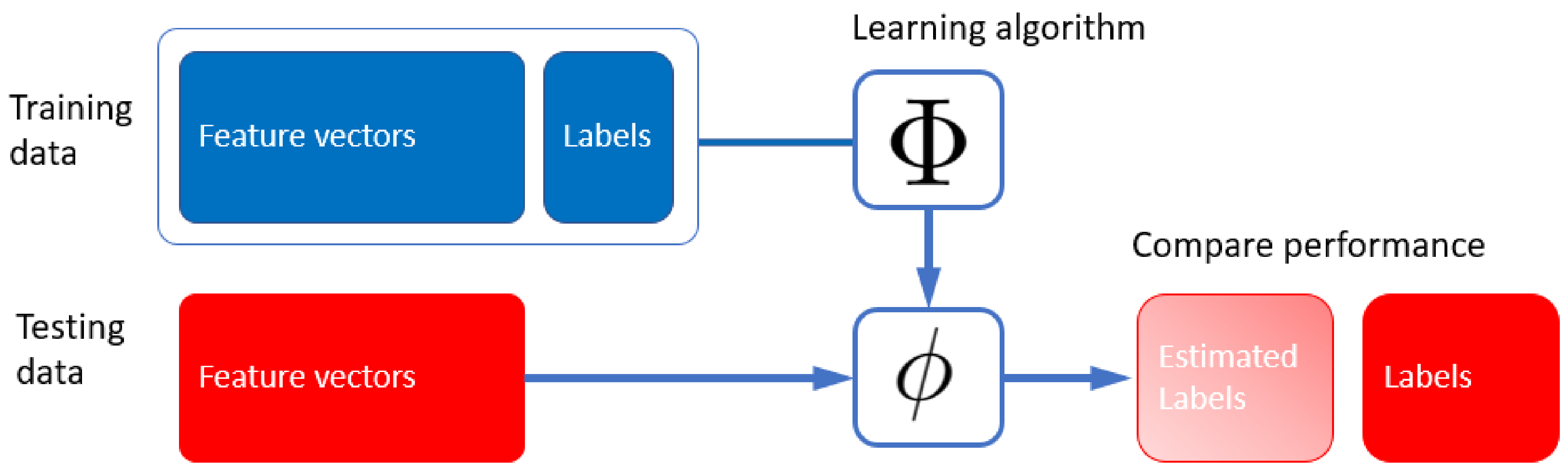

We will use the test data to estimate the test error:

$$\mathcal{R}(\phi) = \mathbb{P}\left(\phi(X) \neq Y\right) \approx \text{ Average number of mistakes on unseen data.}$$

Not the same as the train error: The average number of mistakes on the training data.

# Learning algorithms

For the classification rule to be successful it must perform well on **unseen** data.

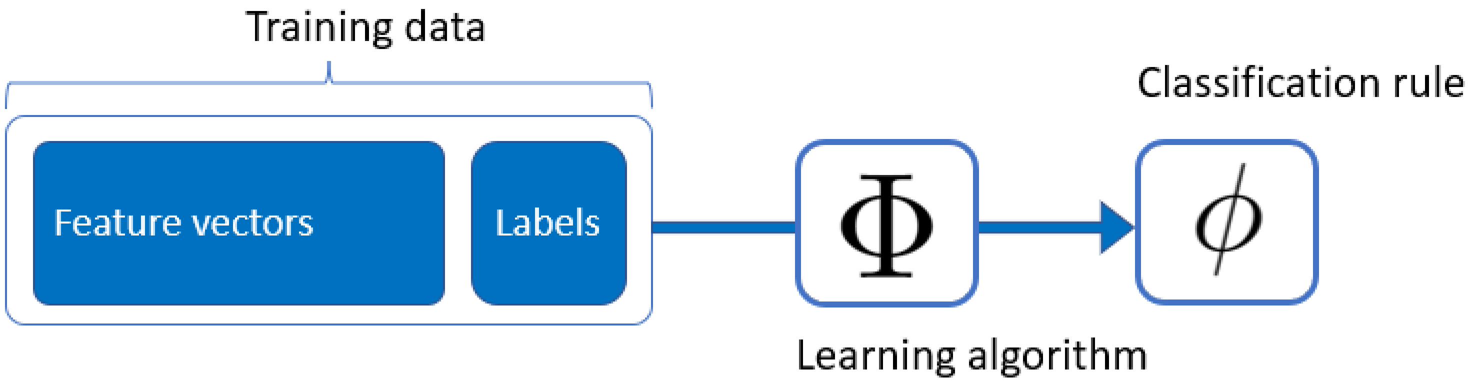


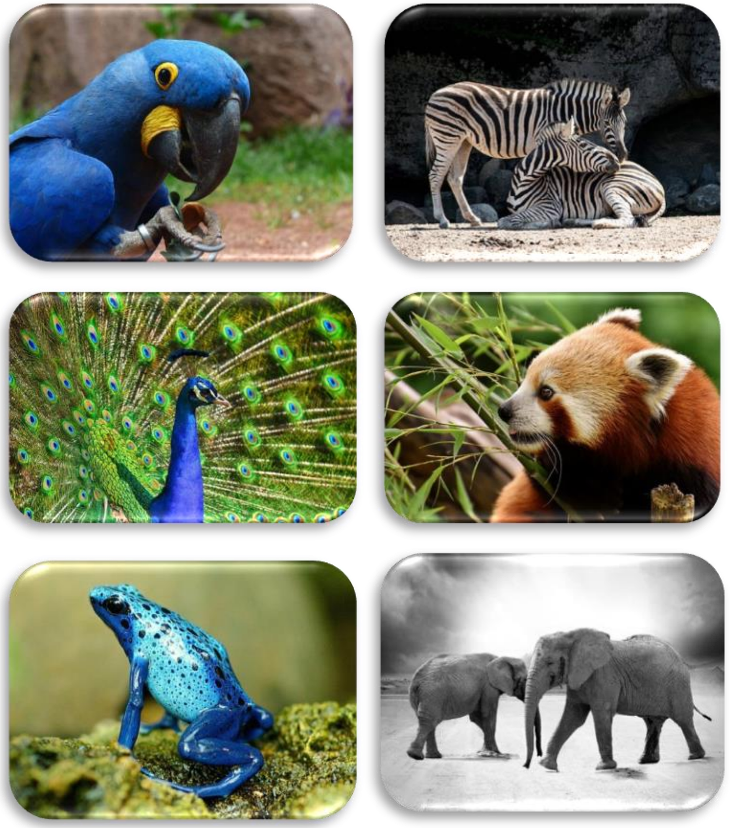

Once we have understood the classification problem.

We can start talking about how to solve the problem: Learning algorithms.

# Learning algorithms

Learning algorithms are the rules for converting training data into classifiers.



Examples:

Linear Discriminant Analysis, Logistic Regression, Nearest Neighbor

classifiers, Random Forests, Boosting, Neural networks, SVMS.

# What have we covered today?

- We began by introducing the concept of classification with some examples.
- We emphasized the importance of predictive performance on unseen data.
- We used probabilistic ideas to understand the classification problem.
- We emphasized the difference the difference between train and test error.
- We discussed the fundamental concept of a Bayes classifier.
- We also considered the supervised learning pipeline and the role of the test-train split.

University of BRISTOL

# Thanks for listening!

henry.reeve@bristol.ac.uk

Include EMATM0061 in the subject of your email.

# What have we covered today?