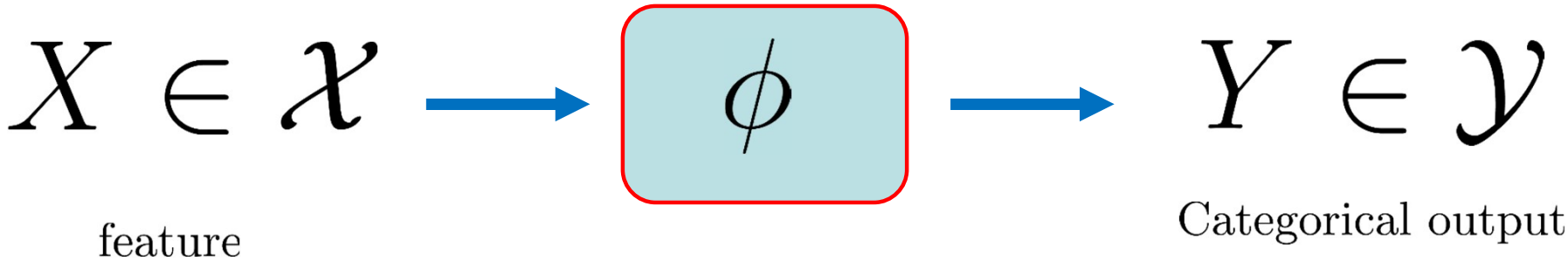
What is classification?

Classification problem: Given a feature vector (e.g., an image of fish) and a set of categories (e.g., { fish, cat, dog, airplane }), we want to assign a class label taken from the categories to the feature vector according to which class it belongs to.

We want to map a feature vector to a class label.

In machine learning, we want to learn a function $\phi : \mathcal{X} \to \mathcal{Y}$, which takes as input a feature vector $X \in \mathcal{X}$, and returns a categorical variable $\phi(X) \in \mathcal{Y}$ which is also known as a label.



Examples

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"

$$X \in \mathcal{X} \longrightarrow \phi \longrightarrow Y \in \mathcal{Y}$$

feature

Categorical output

Examples

Example 2: (Image classification) A biologist wants to automatically classify images of fish according to which species they belong to.

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

feature

Categorical output

Examples

Example 3: (Automated medical diagnosis) A medical doctor wants to establish an automated procedure for classifying retinal blood vessels as normal or abnormal.

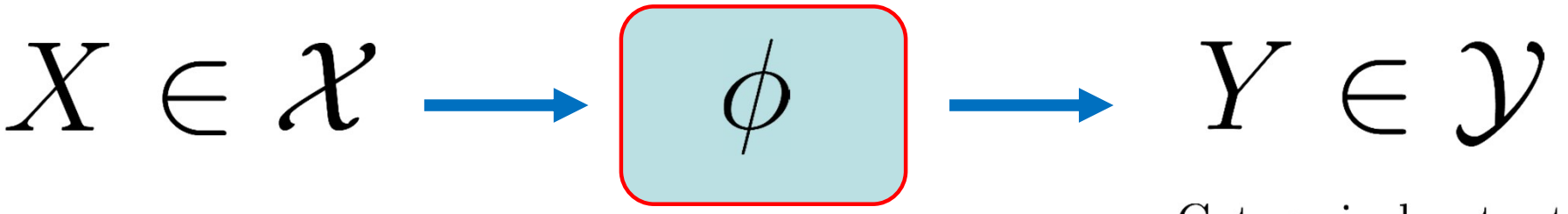
This is a binary classification problem because it has just two possible outcomes ("normal" vs. "abnormal")

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

feature

Categorical output

Classification rule



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

The core problem here is to find the classification rule. We want to implement the rule within computers.

How can we find that?

How can we create a classification rule (I)

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

feature — Categorical output

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

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

The rule-based approach:

We could attempt to program a detailed set of rules

- e.g., "The rainbow shark has two large eyes"…

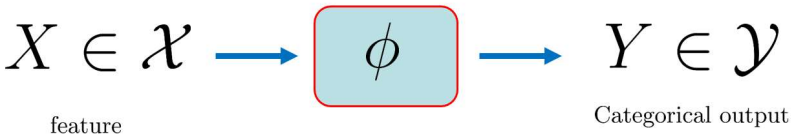
Problems:

This approach would be incredibly labor-intensive.

Based on a set of fixed rules and hence not flexible: New problems would require new rules.

Performs poorly in practice: Brittle e.g. what if we can't see both eyes etc.?

How can we create a classification rule (II)



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

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

The statistical learning approach: 统计学习方法：

Instead of setting a set of fixed rules, we design learning algorithms so that the machine can learn to classify from data.

- e.g., instead of giving the machine rules, we can give the machine a few examples of fish, and ask it to learn how to classify fish from the examples

Machine learning proverb:

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

The supervised learning pipeline

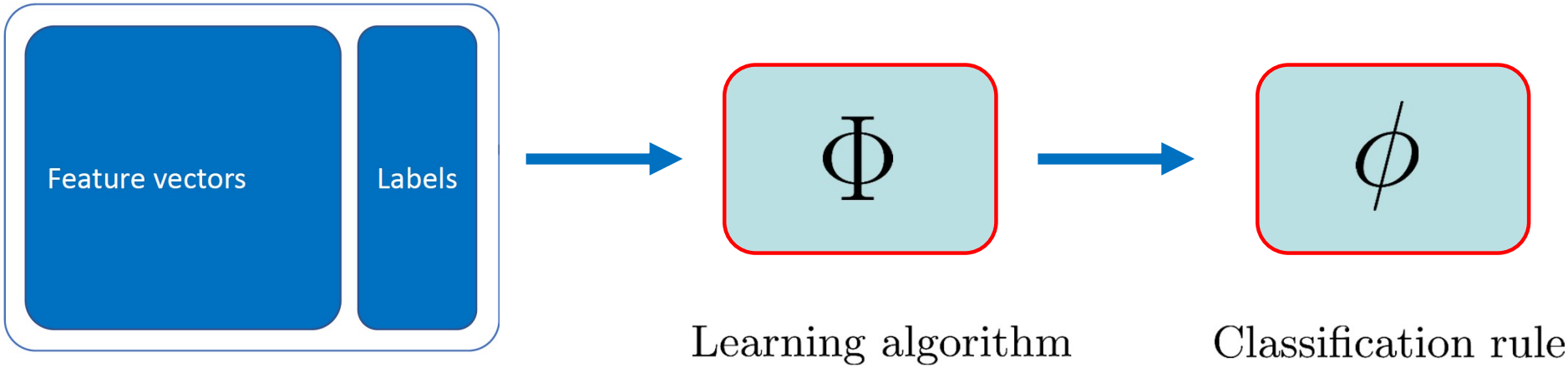
Supervised learning

Data: In supervised learning, we assume that a set of training data $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$ is given, where $X_i \in \mathcal{X}$ is a feature vector, and $Y_i \in \mathcal{Y}$ is the label associated with X_i.

Example: X_i is an image of a particular fish, and Y_i is a label corresponding to the species of the fish.

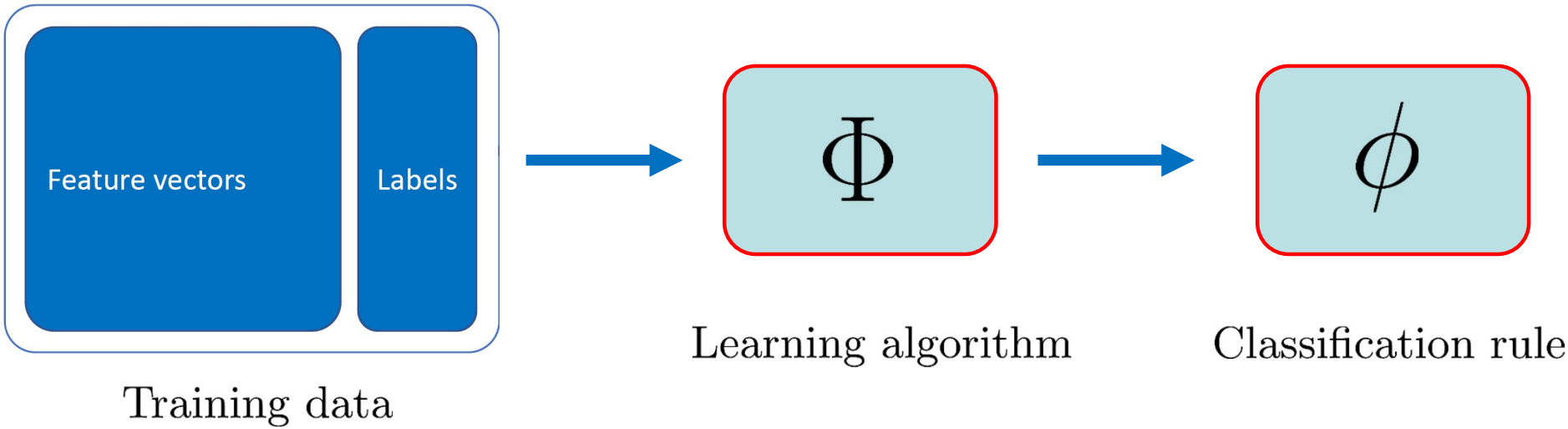
Goal: our goal is to learn a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ from the data $\mathcal{D}$.

Algorithm: The training data is then passed to a learning algorithm. The output of the learning algorithm is the classification rule that we want.



Supervised learning

Algorithm: The training data is then passed to a learning algorithm. The output of the learning algorithm is the classification rule that we want.



A classification rule is also known as a classifier.

A learned classifier should reflect the structure of the training data $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$.

The learning algorithm selects the classifier based on a certain criterion, for example, $\phi(X_i)$ must be close to the label Y_i, i.e., fitting $\phi(X_i)$ to Y_i.

Learning vs. memorization

Goal: our goal is to learn a classification rule $\phi : \mathcal{X} \to \mathcal{Y}$ from the data $\mathcal{D}$.

Algorithm: The training data is then passed to a learning algorithm. The output of the learning algorithm is the classification rule that we want.

A good classifier ϕ should map a feature $X \in \mathcal{X}$ to its associated label $Y \in \mathcal{Y}$.

Key point: The classification rule should perform well on unseen feature vectors $X \in \mathcal{X}$, i.e., the feature vectors that are not in $\mathcal{D}$.

- Knowing ϕ performs well on $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$ is not sufficient.

Example: classifying fish

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.

Learning vs. memorization

Key point: The classification rule should perform well on unseen feature vectors $X \in \mathcal{X}$, i.e., the feature vectors that are not in $\mathcal{D}$.

- Knowing ϕ performs well on $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$ is not sufficient.

This is the crucial difference between learning and memorization.

If the learning algorithm is not designed properly, then the learned classifier might remember the dataset instead of learning the useful rules.

- The classifier remembers each pair of (X_i, Y_i). Now given an input $X \in \{X_1, \cdots, X_n\}$, it compares it with each of the X_i and then outputs Y_i.
- As a result, the classifier performs poorly given an unseen X.

Therefore, we need a test dataset to check the performance of the classifier on data that was not seen during training.

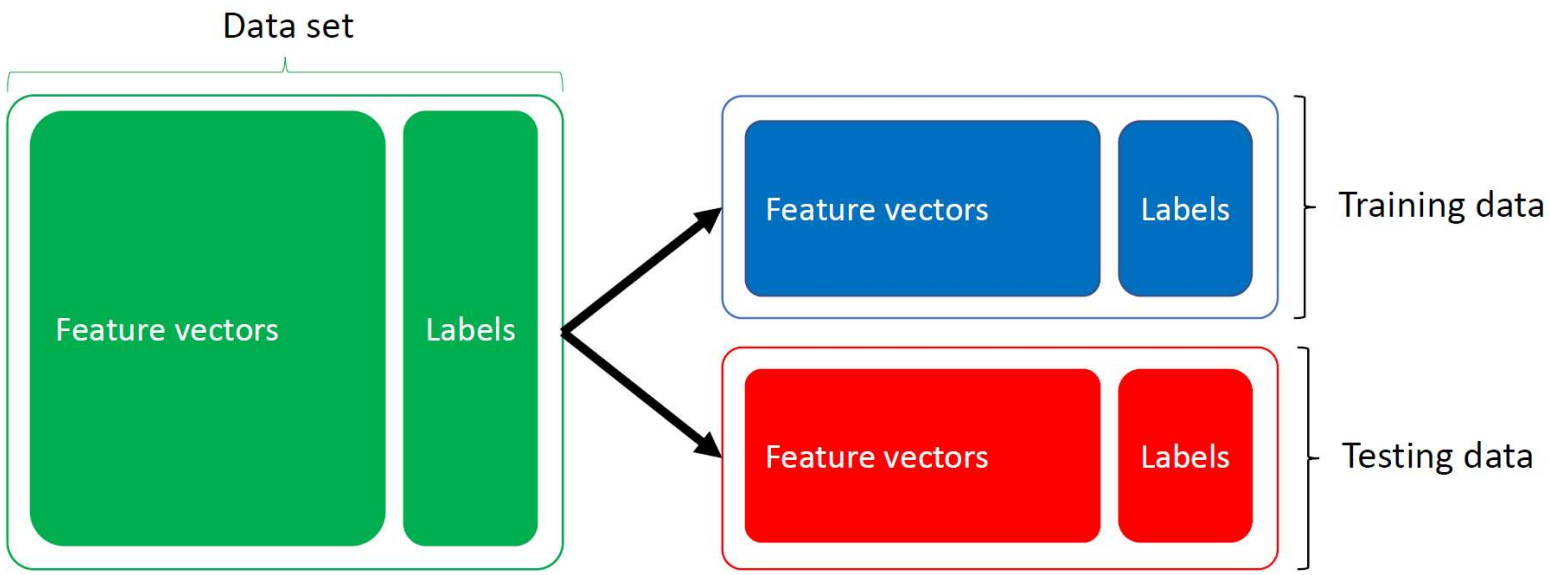
Train-test split

- Creating a dataset for checking the performance of your classifier on unseen data

The train-test split

We need a test dataset to check the performance of the classifier on data that was not seen during training.

Given a dataset, we can split it into a training dataset and a test dataset.

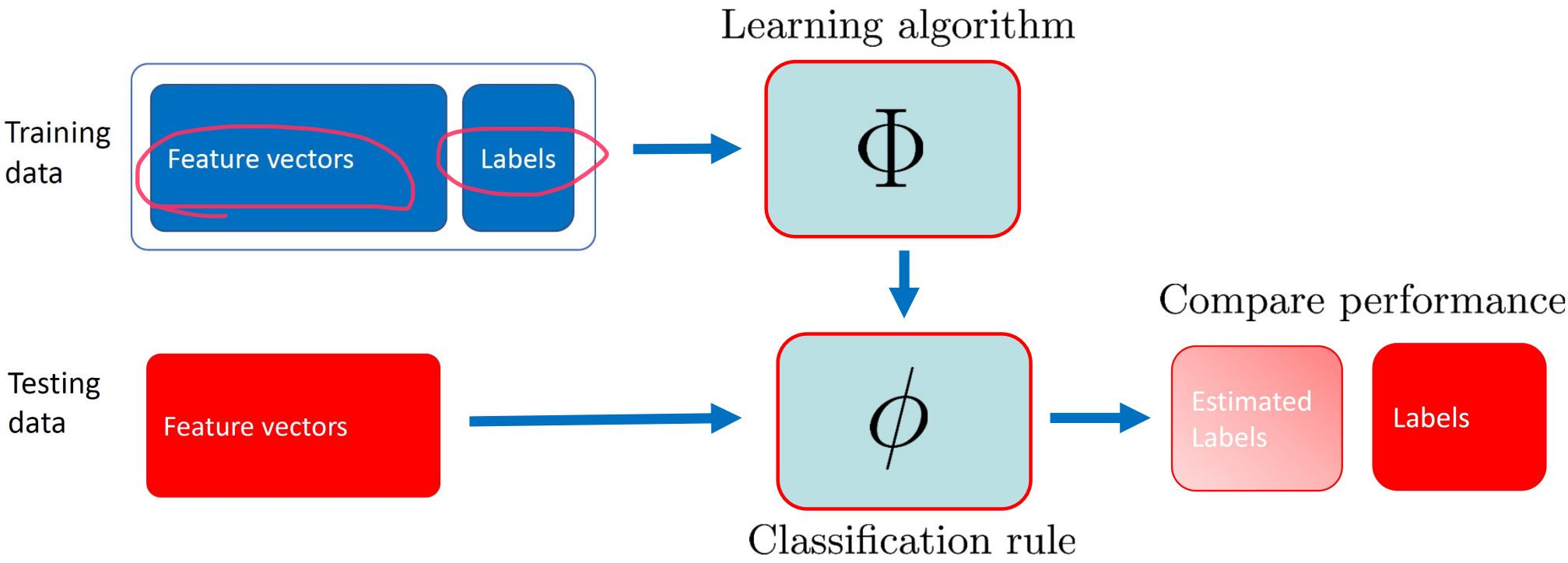


We use the test dataset to assess our learning algorithms

Given a dataset, we can split it into a training dataset and a test dataset.

We use the training dataset to train our classifier

We use the test dataset to assess our classifier.



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

Example: Penguin classification

Example: 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.

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

X^1 = the weight of the penguin (grams)

X^2 = the flipper length of the penguin (mm)

Labels: $Y \in \mathcal{Y} := \{0, 1\}$

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

Example: Penguin classification

Example: 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.

```
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

Example: 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.

X 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
```

Features:

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

X^1 = the weight of the penguin (grams)

X^2 = the flipper length of the penguin (mm)

Labels:

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

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

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
```

We can also separate out the feature vectors and labels (i.e., separate out X and Y into two different data frames).

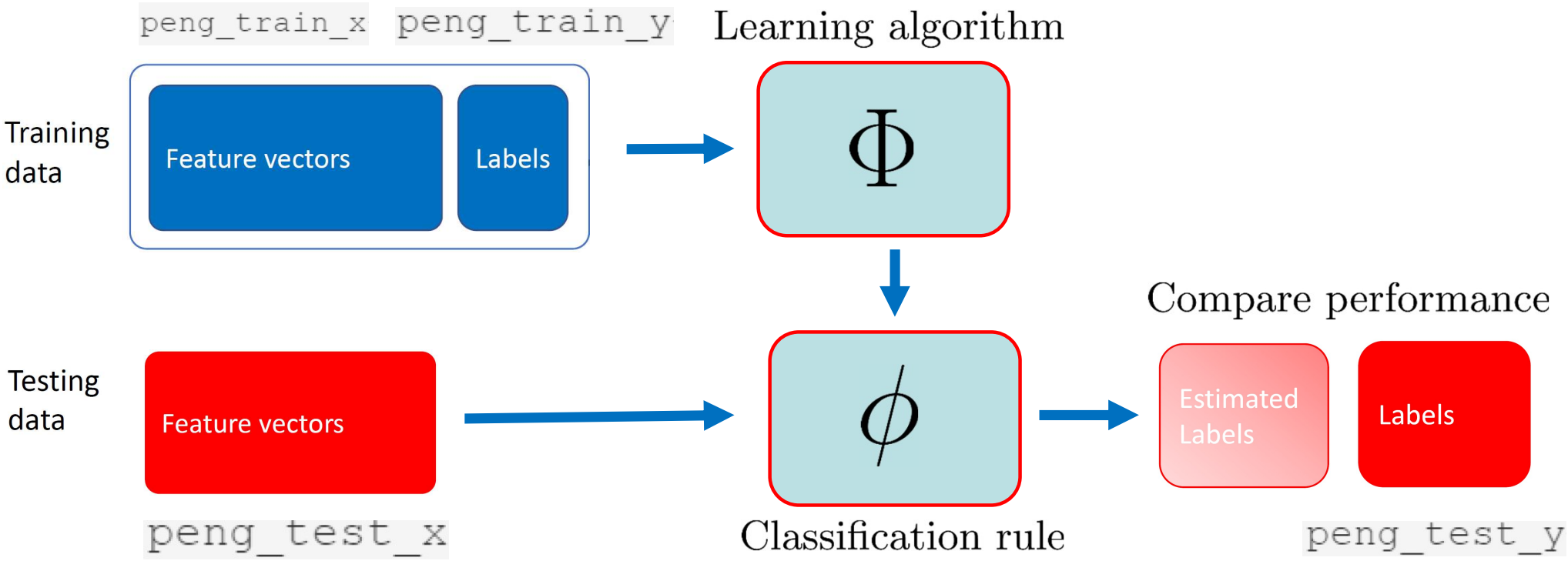
```
peng_train_x<-peng_train%>%select(-species) # train feature vectors
peng_train_y<-peng_train%>%pull(species) # train labels

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

We use the test dataset to assess our learning algorithms

Given a dataset, we can split it into a training dataset and a test dataset.

We use the training dataset to train our classifier

We use the test dataset to assess our classifier.



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

Expected test error and Bayes classifier

- quantifying the error of your classifiers

A probabilistic model

In machine learning, we want to learn a function $\phi : \mathcal{X} \to \mathcal{Y}$, which takes as input a feature vector $X \in \mathcal{X}$, and returns a categorical variable $\phi(X) \in \mathcal{Y}$ which is also known as a label.

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 $\mathcal{Y} = \{0, 1\}$.

Moreover, if we view a feature vector X as a random variable (vector) and Y as a random variable, then X and Y are dependent (the class label is dependent on the feature vector).

Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

- Then $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$ is a sample drawn from $\mathbf{P}$.
- We often assume $(X_1, Y_1), \cdots, (X_n, Y_n)$ are i.i.d.

With the distribution $\mathbf{P}$, we can describe the error of the classifier (see next slide).

Expected test error

In machine learning, we want to learn a function $\phi : \mathcal{X} \to \mathcal{Y}$, which takes as input a feature vector $X \in \mathcal{X}$, and returns a categorical variable $\phi(X) \in \mathcal{Y}$ which is also known as a label.

Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

We can quantify the classifier error by the probability of the classifier making a mistake ($\phi(X) \neq Y$), i.e.,

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

$\mathcal{R}(\phi)$ is called the expected test error. Intuitively, this corresponds to the average number of mistakes made by the classifier.

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

We want our classifier to have a low expected test error.

The Bayes classifier

In machine learning, we want to learn a function $\phi : \mathcal{X} \to \mathcal{Y}$, which takes as input a feature vector $X \in \mathcal{X}$, and returns a categorical variable $\phi(X) \in \mathcal{Y}$ which is also known as a label.

Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

Expected test error: $\mathcal{R}(\phi) := \mathbb{P}(\phi(X) \neq Y)$.

We want our classifier to have a low expected test error.

The classifier with the lowest expected test error is known as the Bayes classifier ϕ^* which satisfies:

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

So ϕ^* is a minimiser of the expected test error (over all possible classifiers).

The Bayes classifier

Expected test error: $\mathcal{R}(\phi) := \mathbb{P}(\phi(X) \neq Y)$.

ϕ^* is a minimiser of the expected test error (over all possible classifiers).

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

In the binary case where $\mathcal{Y} := \{0, 1\}$, the Bayes classifier is given by:

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

So ϕ^* outputs the label (either 0 or 1) with the biggest conditional probability given the feature $X = x$.

Unfortunately, we often do NOT know the distribution of (X, Y) in practice, so we can not compute ϕ^* based on the above formula.

We will learn about the distribution from the data (using learning algorithms)!

Training error vs test error

Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

Expected test error: $\mathcal{R}(\phi) := \mathbb{P}(\phi(X) \neq Y)$.

However, the training error is given by $\hat{R}_n(\phi) := \frac{1}{n} \sum_{i=1}^{n} \mathbb{1}\{\phi(X_i) \neq Y_i\}$.

So the training error is the average number of mistakes of the classifier on the training set $\mathcal{D} := \{(X_1, Y_1), \cdots, (X_n, Y_n)\}$.

Example: Classifying images.

Training error: the average number of misclassified images in the training data

Test error: the average number of misclassified images in the whole population

So having low training error does not imply low expected test error!

Estimating the expected test error

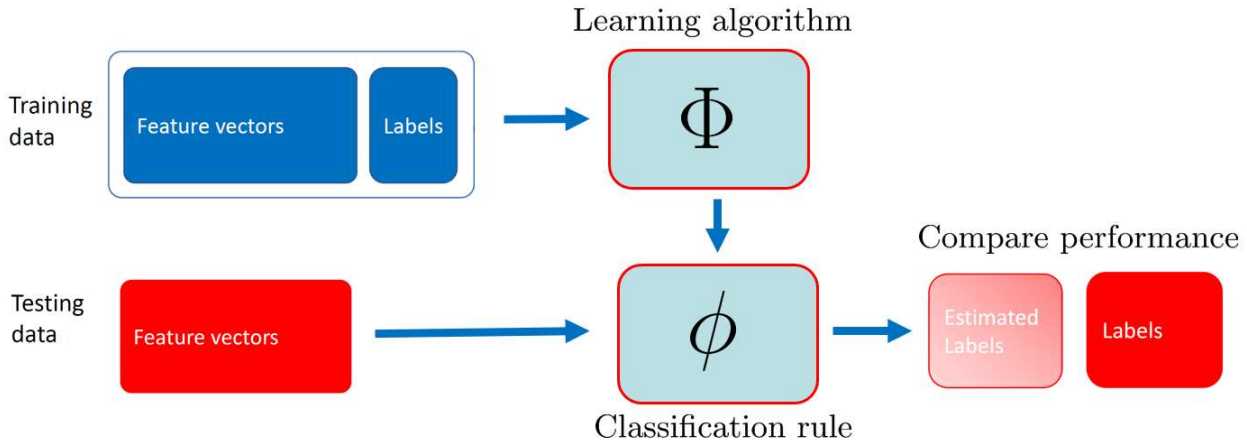
Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

Our goal is to do well on unseen data, and hence looking at the training error is not enough.

We can use the test data to estimate the expected test error:

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

This is not the same as the training error: The average number of mistakes on the training data.



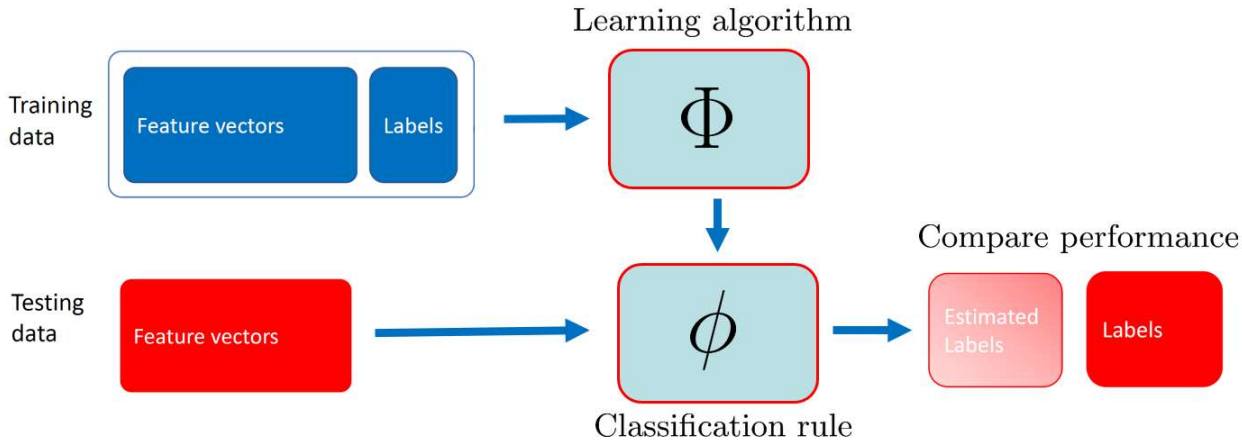
Estimating the expected test error

Assume that $(X, Y) \sim \mathbf{P}$ with joint distribution $\mathbf{P}$.

Our goal is to do well on unseen data, and hence looking at the training error is not enough.

We can use the test data to estimate the expected test error:

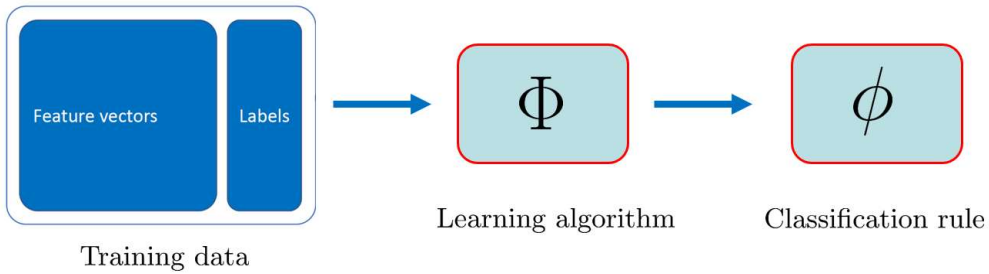
$$\mathcal{R}(\phi) := \mathbb{P}(\phi(X) \neq Y) \approx \text{Average number of mistakes on the test data.}$$



Once we have understood the classification problem and the performance measurement, we can start talking about how to solve the problem: Learning algorithms.

Learning algorithms

Learning algorithms are a set of procedures for converting training data into classifiers.



Examples:

Linear Discriminant Analysis, Logistic Regression, Nearest Neighbour classifiers, Random Forests, Boosting, Neural networks, SVMS.