MACHINE LEARNING – INTRODUCTORY CONCEPTS

We can generally classify features as either quantitative or qualitative. *Quantitative* features possess "numerical quantity", such as height, age, number of births, etc., and can either be *continuous* or *discrete*. Continuous quantitative features take values in a continuous range of possible values, such as height, voltage, or crop yield; such features capture the idea that measurements can always be made more precisely. Discrete quantitative features have a countable number of possibilities, such as a count.

In contrast, *qualitative* features do not have a numerical meaning, but their possible values can be divided into a fixed number of categories, such as {M,F} for gender or {blue, black, brown, green} for eye color. For this reason such features are also called *categorical*. A simple rule of thumb is: if it does not make sense to average the data, it is categorical. For example, it does not make sense to average eye colors. Of course it is still possible to represent categorical data with numbers, such as 1 = blue, 2 = black, 3 = brown, but such numbers carry no quantitative meaning. Categorical features are often called *factors*.

When manipulating, summarizing, and displaying data, it is important to correctly specify the type of the variables (features).

DATA SETS FOR MACHINE LEARNING

There are many data sets available from the Internet and in software packages. A well known repository of data sets is the Machine Learning Repository maintained by the University of California at Irvine (UCI), found at https://archive.ics.uci.edu/.

Another useful repository of over 1000 data sets from various packages in the R programming language, collected by Vincent Arel-Bundock, can be found at: https://vincentarelbundock.github.io/Rdatasets/datasets.html.

For example, to read Fisher's famous `iris` data set from R's `datasets` package into Python, type:

```
urlprefix = 'https://vincentarelbundock.github.io/Rdatasets/csv/'
dataname = 'datasets/iris.csv'
iris = pd.read_csv(urlprefix + dataname)
```

1.2 Structuring Features According to Type

QUANTITATIVE

We can generally classify features as either quantitative or qualitative. *Quantitative* features possess "numerical quantity", such as height, age, number of births, etc., and can either be *continuous* or *discrete*. Continuous quantitative features take values in a continuous range of possible values, such as height, voltage, or crop yield; such features capture the idea that measurements can always be made more precisely. Discrete quantitative features have a countable number of possibilities, such as a count.

QUALITATIVE

CATEGORICAL

In contrast, *qualitative* features do not have a numerical meaning, but their possible values can be divided into a fixed number of categories, such as {M,F} for gender or {blue, black, brown, green} for eye color. For this reason such features are also called *categorical*. A simple rule of thumb is: if it does not make sense to average the data, it is categorical. For example, it does not make sense to average eye colors. Of course it is still possible to represent categorical data with numbers, such as 1 = blue, 2 = black, 3 = brown, but such numbers carry no quantitative meaning. Categorical features are often called *factors*.

FACTORS

When manipulating, summarizing, and displaying data, it is important to correctly specify the type of the variables (features). We illustrate this using the `nutrition_elderly` data set from [73], which contains the results of a study involving nutritional measurements of thirteen features (columns) for 226 elderly individuals (rows). The data set can be obtained from:

`http://www.biostatisticien.eu/springeR/nutrition_elderly.xls`.

Excel files can be read directly into **`pandas`** via the `read_excel` method:

```
xls = 'http://www.biostatisticien.eu/springeR/nutrition_elderly.xls'
nutri = pd.read_excel(xls)
```

This creates a DataFrame object **nutri**. The first three rows are as follows:

```
pd.set_option('display.max_columns', 8) # to fit display
nutri.head(3)
```

```
   gender  situation  tea ...   cooked_fruit_veg  chocol  fat
0       2          1    0 ...                  4       5    6
1       2          1    1 ...                  5       1    4
2       2          1    0 ...                  2       5    4

[3 rows x 13 columns]
```

You can check the type (or structure) of the variables via the **info** method of **nutri**.

```
nutri.info()
```

```
 <class 'pandas.core.frame.DataFrame'>
RangeIndex: 226 entries, 0 to 225
Data columns (total 13 columns):
gender              226 non-null int64
situation           226 non-null int64
tea                 226 non-null int64
coffee              226 non-null int64
height              226 non-null int64
weight              226 non-null int64
age                 226 non-null int64
meat                226 non-null int64
fish                226 non-null int64
raw_fruit           226 non-null int64
cooked_fruit_veg    226 non-null int64
chocol              226 non-null int64
fat                 226 non-null int64
dtypes: int64(13)
memory usage: 23.0 KB
```

All 13 features in **nutri** are (at the moment) interpreted by Python as *quantitative* variables, indeed as integers, simply because they have been entered as whole numbers. The *meaning* of these numbers becomes clear when we consider the description of the features, given in Table 1.2. Table 1.1 shows how the variable types should be classified.

Table 1.1: The feature types for the data frame `nutri`.

Qualitative	`gender`, `situation`, `fat`
	`meat`, `fish`, `raw_fruit`, `cooked_fruit_veg`, `chocol`
Discrete quantitative	`tea`, `coffee`
Continuous quantitative	`height`, `weight`, `age`

Table 1.2: Description of the variables in the nutritional study [73].

Feature	**Description**	**Unit** or **Coding**
`gender`	Gender	1=Male; 2=Female
`situation`	Family status	1=Single 2=Living with spouse 3=Living with family 4=Living with someone else
`tea`	Daily consumption of tea	Number of cups
`coffee`	Daily consumption of coffee	Number of cups
`height`	Height	cm
`weight`	Weight (actually: mass)	kg
`age`	Age at date of interview	Years
`meat`	Consumption of meat	0=Never 1=Less than once a week 2=Once a week 3=2–3 times a week 4=4–6 times a week 5=Every day
`fish`	Consumption of fish	As in `meat`
`raw_fruit`	Consumption of raw fruits	As in `meat`
`cooked_fruit_veg`	Consumption of cooked fruits and vegetables	As in `meat`
`chocol`	Consumption of chocolate	As in `meat`
`fat`	Type of fat used for cooking	1=Butter 2=Margarine 3=Peanut oil 4=Sunflower oil 5=Olive oil 6=Mix of vegetable oils (e.g., Isio4) 7=Colza oil 8=Duck or goose fat

contrast to qualitative features without order, which are called *nominal*. We will not make such a distinction in this book.

We can modify the Python value and type for each categorical feature, using the **replace** and **astype** methods. For categorical features, such as `gender`, we can replace the value 1 with `'Male'` and 2 with `'Female'`, and change the type to `'category'` as follows.

```
DICT = {1:'Male', 2:'Female'} # dictionary specifies replacement
nutri['gender'] = nutri['gender'].replace(DICT).astype('category')
```

The structure of the other categorical-type features can be changed in a similar way. Continuous features such as `height` should have type `float`:

```
nutri['height'] = nutri['height'].astype(float)
```

1.3 Summary Tables

It is often useful to summarize a large spreadsheet of data in a more condensed form. A table of counts or a table of frequencies makes it easier to gain insight into the underlying distribution of a variable, especially if the data are qualitative. Such tables can be obtained with the methods **describe** and **value_counts**.

As a first example, we load the **nutri** `DataFrame`, which we restructured and saved (see previous section) as `'nutri.csv'`, and then construct a summary for the feature (column) `'fat'`.

```
nutri = pd.read_csv('nutri.csv')
nutri['fat'].describe()
```

```
count            226
unique             8
top        sunflower
freq              68
Name: fat, dtype: object
```

We see that there are 8 different types of fat used and that sunflower has the highest count, with 68 out of 226 individuals using this type of cooking fat. The method **value_counts** gives the counts for the different fat types.

```
nutri['fat'].value_counts()
```

```
sunflower    68
peanut       48
olive        40
margarine    27
Isio4        23
butter       15
duck          4
colza         1
Name: fat, dtype: int64
```

Column labels are also attributes of a `DataFrame`, and `nutri.fat`, for example, is exactly the same object as `nutri['fat']`.

It is also possible to use **crosstab** to *cross tabulate* between two or more variables, giving a *contingency table*:

CROSS TABULATE

```
pd.crosstab(nutri.gender, nutri.situation)
```

```
situation  Couple  Family  Single
gender
Female         56       7      78
Male           63       2      20
```

We see, for example, that the proportion of single men is substantially smaller than the proportion of single women in the data set of elderly people. To add row and column totals to a table, use `margins=True`.

```
pd.crosstab(nutri.gender, nutri.situation, margins=True)
```

```
situation  Couple  Family  Single  All
gender
Female         56       7      78  141
Male           63       2      20   85
All           119       9      98  226
```

1.4 Summary Statistics

In the following, $\boldsymbol{x} = [x_1, \ldots, x_n]^\mathsf{T}$ is a column vector of n numbers. For our **nutri** data, the vector $\boldsymbol{x}$ could, for example, correspond to the heights of the $n = 226$ individuals.

SAMPLE MEAN

The *sample mean* of $\boldsymbol{x}$, denoted by $\bar{x}$, is simply the average of the data values:

$$\bar{x} = \frac{1}{n} \sum_{i=1}^{n} x_i.$$

Using the **mean** method in Python for the **nutri** data, we have, for instance:

```
nutri['height'].mean()
```

```
163.96017699115043
```

SAMPLE QUANTILE

SAMPLE MEDIAN

QUARTILES

The *p-sample quantile* ($0 < p < 1$) of $\boldsymbol{x}$ is a value x such that at least a fraction p of the data is less than or equal to x and at least a fraction $1 - p$ of the data is greater than or equal to x. The *sample median* is the sample 0.5-quantile. The p-sample quantile is also called the $100 \times p$ *percentile*. The 25, 50, and 75 sample percentiles are called the first, second, and third *quartiles* of the data. For the **nutri** data they are obtained as follows.

```
nutri['height'].quantile(q=[0.25,0.5,0.75])
```

```
0.25    157.0
0.50    163.0
0.75    170.0
```

SAMPLE RANGE

SAMPLE VARIANCE

The sample mean and median give information about the *location* of the data, while the distance between sample quantiles (say the 0.1 and 0.9 quantiles) gives some indication of the *dispersion* (spread) of the data. Other measures for dispersion are the *sample range*, $\max_i x_i - \min_i x_i$, the *sample variance*

$$s^2 = \frac{1}{n-1} \sum_{i=1}^{n} (x_i - \bar{x})^2, \tag{1.1}$$

SAMPLE STANDARD DEVIATION

☞ 455

and the *sample standard deviation* $s = \sqrt{s^2}$. For the **nutri** data, the range (in cm) is:

```
nutri['height'].max() - nutri['height'].min()
```

```
48.0
```

The variance (in cm^2) is:

```
round(nutri['height'].var(), 2)  # round to two decimal places
```

```
81.06
```

And the standard deviation can be found via:

```
round(nutri['height'].std(), 2)
```

```
9.0
```

We already encountered the **describe** method in the previous section for summarizing qualitative features, via the most frequent count and the number of unique elements. When applied to a *quantitative* feature, it returns instead the minimum, maximum, mean, and the three quartiles. For example, the `'height'` feature in the **nutri** data has the following summary statistics.

```
nutri['height'].describe()
```

```
count    226.000000
mean     163.960177
std        9.003368
min      140.000000
25\%      157.000000
50\%      163.000000
75\%      170.000000
max      188.000000
Name: height, dtype: float64
```

1.5 Visualizing Data

In this section we describe various methods for visualizing data. The main point we would like to make is that the way in which variables are visualized should always be adapted to the variable types; for example, qualitative data should be plotted differently from quantitative data.

For the rest of this section, it is assumed that **matplotlib.pyplot**, **pandas**, and **numpy**, have been imported in the Python code as follows.

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
```

1.5.1 Plotting Qualitative Variables

Suppose we wish to display graphically how many elderly people are living by themselves, as a couple, with family, or other. Recall that the data are given in the `situation` column of our **`nutri`** data. Assuming that we already *restructured the data*, as in Section 1.2, we can make a *barplot* of the number of people in each category via the **`plt.bar`** function of the standard **`matplotlib`** plotting library. The inputs are the x-axis positions, heights, and widths of each bar respectively.

```
width = 0.35 # the width of the bars
x = [0, 0.8, 1.6] # the bar positions on x-axis
situation_counts=nutri['situation'].value_counts()
plt.bar(x, situation_counts, width, edgecolor = 'black')
plt.xticks(x, situation_counts.index)
plt.show()
```

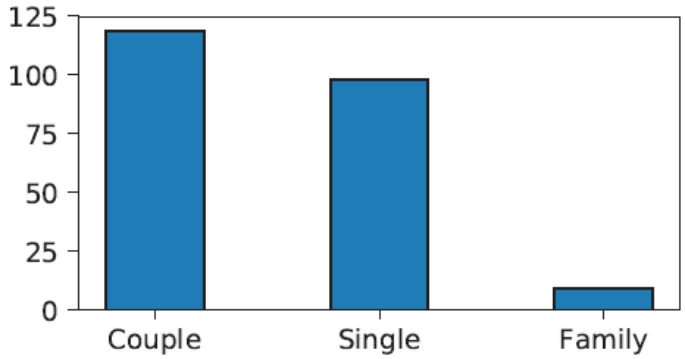

Figure 1.1: Barplot for the qualitative variable `'situation'`.

1.5.2 Plotting Quantitative Variables

We now present a few useful methods for visualizing quantitative data, again using the **`nutri`** data set. We will first focus on continuous features (e.g., `'age'`) and then add some specific graphs related to discrete features (e.g., `'tea'`). The aim is to describe the variability present in a single feature. This typically involves a central tendency, where observations tend to gather around, with fewer observations further away. The main aspects of the distribution are the *location* (or center) of the variability, the *spread* of the variability (how far the values extend from the center), and the *shape* of the variability; e.g., whether or not values are spread symmetrically on either side of the center.

1.5.2.1 Boxplot

A *boxplot* can be viewed as a graphical representation of the five-number summary of the data consisting of the minimum, maximum, and the first, second, and third quartiles. Figure 1.2 gives a boxplot for the `'age'` feature of the **nutri** data.

```
plt.boxplot(nutri['age'],widths=width,vert=False)
plt.xlabel('age')
plt.show()
```

The `widths` parameter determines the width of the boxplot, which is by default plotted vertically. Setting `vert=False` plots the boxplot horizontally, as in Figure 1.2.

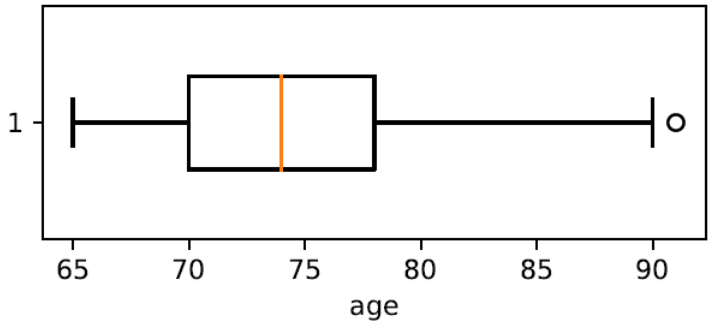

Figure 1.2: Boxplot for `'age'`.

The box is drawn from the first quartile (Q_1) to the third quartile (Q_3). The vertical line inside the box signifies the location of the median. So-called "whiskers" extend to either side of the box. The size of the box is called the *interquartile range*: IQR = $Q_3 - Q_1$. The left whisker extends to the largest of (a) the minimum of the data and (b) $Q_1 - 1.5\,\text{IQR}$. Similarly, the right whisker extends to the smallest of (a) the maximum of the data and (b) $Q_3 + 1.5\,\text{IQR}$. Any data point outside the whiskers is indicated by a small hollow dot, indicating a suspicious or deviant point (outlier). Note that a boxplot may also be used for discrete quantitative features.

1.5.2.2 Histogram

A *histogram* is a common graphical representation of the distribution of a quantitative feature. We start by breaking the range of the values into a number of *bins* or *classes*. We tally the counts of the values falling in each bin and then make the plot by drawing rectangles whose bases are the bin intervals and whose heights are the counts. In Python we can use the function **plt.hist**. For example, Figure 1.3 shows a histogram of the 226 ages in **nutri**, constructed via the following Python code.

```
weights = np.ones_like(nutri.age)/nutri.age.count()
plt.hist(nutri.age,bins=9,weights=weights,facecolor='cyan',
         edgecolor='black', linewidth=1)
plt.xlabel('age')
plt.ylabel('Proportion of Total')
plt.show()
```

Here 9 bins were used. Rather than using raw counts (the default), the vertical axis here gives the percentage in each class, defined by $\frac{\text{count}}{\text{total}}$. This is achieved by choosing the "weights" parameter to be equal to the vector with entries $1/266$, with length 226. Various plotting parameters have also been changed.

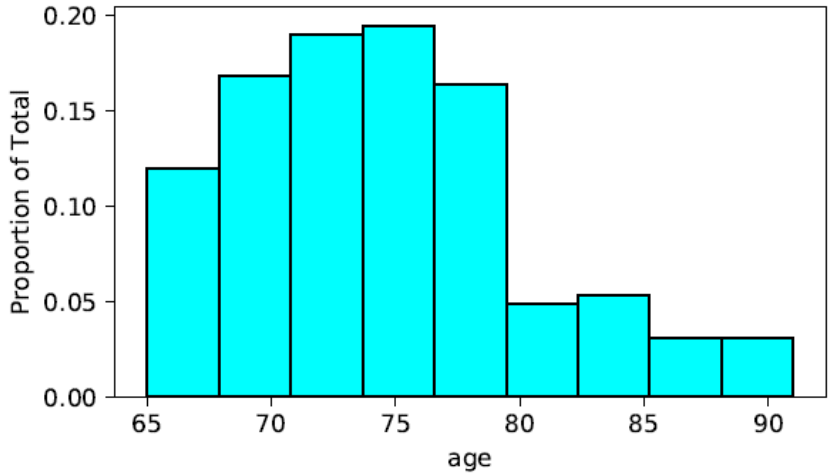

Figure 1.3: Histogram of `'age'`.

Histograms can also be used for discrete features, although it may be necessary to explicitly specify the bins and placement of the ticks on the axes.

1.5.3 Data Visualization in a Bivariate Setting

In this section, we present a few useful visual aids to explore relationships between two features. The graphical representation will depend on the type of the two features.

1.5.3.1 Two-way Plots for Two Categorical Variables

Comparing barplots for two categorical variables involves introducing subplots to the figure. Figure 1.5 visualizes the contingency table of Section 1.3, which cross-tabulates the family status (situation) with the gender of the elderly people. It simply shows two barplots next to each other in the same figure.

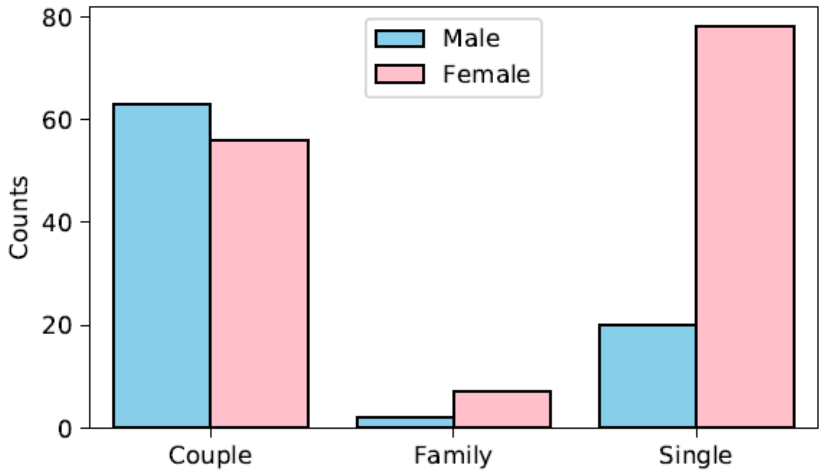

Figure 1.5: Barplot for two categorical variables.

The figure was made using the **seaborn** package, which was specifically designed to simplify statistical visualization tasks.

```
import seaborn as sns
sns.countplot(x='situation', hue = 'gender', data=nutri,
     hue_order = ['Male', 'Female'], palette = ['SkyBlue','Pink'],
      saturation = 1, edgecolor='black')
plt.legend(loc='upper center')
plt.xlabel('')
plt.ylabel('Counts')
plt.show()
```

1.5.3.2 Plots for Two Quantitative Variables

We can visualize patterns between two quantitative features using a *scatterplot*. This can be done with `plt.scatter`. The following code produces a scatterplot of `'weight'` against `'height'` for the `nutri` data.

```
plt.scatter(nutri.height, nutri.weight, s=12, marker='o')
plt.xlabel('height')
plt.ylabel('weight')
plt.show()
```

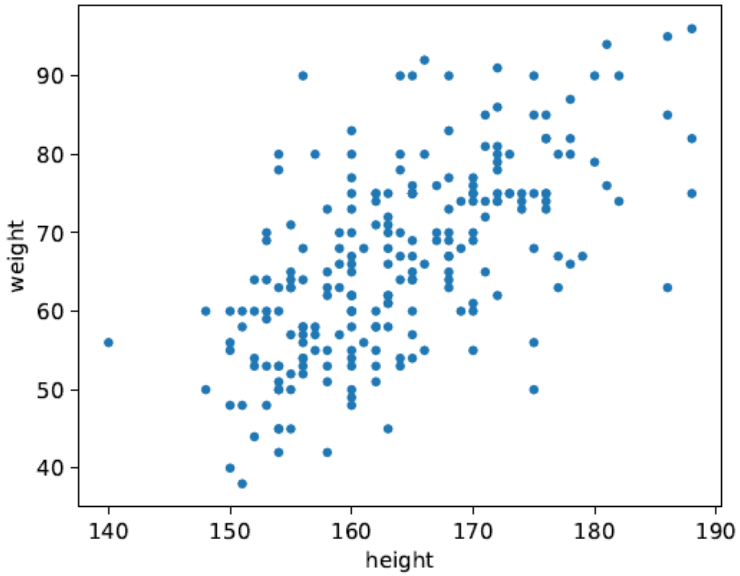

Figure 1.6: Scatterplot of `'weight'` against `'height'`.

The next Python code illustrates that it is possible to produce highly sophisticated scatter plots, such as in Figure 1.7. The figure shows the birth weights (mass) of babies whose mothers smoked (blue triangles) or not (red circles). In addition, straight lines were fitted to the two groups, suggesting that birth weight decreases with age when the mother smokes, but increases when the mother does not smoke! The question is whether these trends are statistically significant or due to chance.

```
urlprefix = 'https://vincentarelbundock.github.io/Rdatasets/csv/'
dataname = 'MASS/birthwt.csv'
bwt = pd.read_csv(urlprefix + dataname)
bwt = bwt.drop('Unnamed: 0',1)   #drop unnamed column
styles = {0: ['o','red'], 1: ['^','blue']}
for k in styles:
    grp = bwt[bwt.smoke==k]
    m,b = np.polyfit(grp.age, grp.bwt, 1) # fit a straight line
    plt.scatter(grp.age, grp.bwt, c=styles[k][1], s=15, linewidth=0,
        marker = styles[k][0])
    plt.plot(grp.age, m*grp.age + b, '-', color=styles[k][1])

plt.xlabel('age')
plt.ylabel('birth weight (g)')
plt.legend(['non-smokers','smokers'],prop={'size':8},
            loc=(0.5,0.8))
plt.show()
```

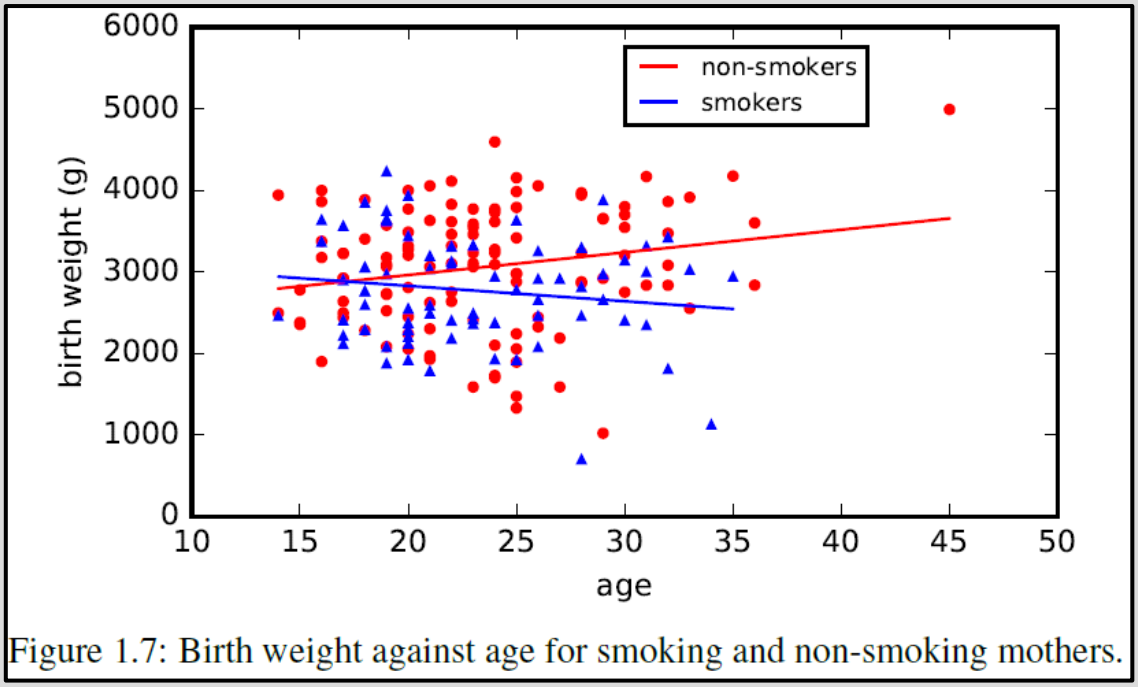

Figure 1.7: Birth weight against age for smoking and non-smoking mothers.

1.5.3.3 Plots for One Qualitative and One Quantitative Variable

In this setting, it is interesting to draw boxplots of the quantitative feature for each level of the categorical feature. Assuming the variables are structured correctly, the function **`plt.boxplot`** can be used to produce Figure 1.8, using the following code:

```
males = nutri[nutri.gender == 'Male']
females = nutri[nutri.gender == 'Female']
plt.boxplot([males.coffee,females.coffee],notch=True,widths
   =(0.5,0.5))
plt.xlabel('gender')
plt.ylabel('coffee')
plt.xticks([1,2],['Male','Female'])
plt.show()
```

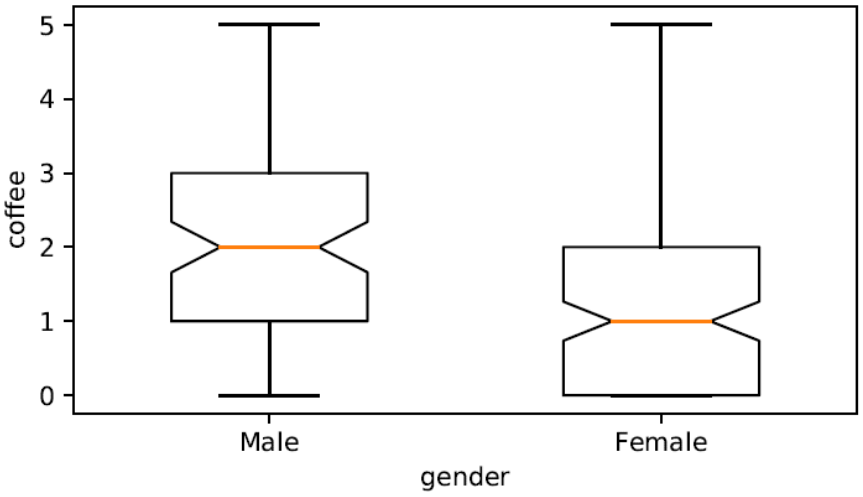

Figure 1.8: Boxplots of a quantitative feature `'coffee'` as a function of the levels of a categorical feature `'gender'`. Note that we used a different, "notched", style boxplot this time.

STATISTICAL LEARNING

Introduction

Although structuring and visualizing data are important aspects of data science, the main challenge lies in the mathematical analysis of the data.

When the goal is to interpret the model and quantify the uncertainty in the data, this analysis is usually referred to as **Statistical Learning**.

In contrast, when the emphasis is on making predictions using large-scale statistical data, then it is common to speak about machine learning or data mining.

There are **two major goals for Modeling data**:

- To accurately predict some future quantity of interest, given some observed data, and
- To discover unusual or interesting patterns in the data.

To achieve these goals, one must rely on knowledge from three important pillars of the mathematical sciences.

Function approximation. Building a mathematical model for data usually means understanding how one data variable depends on another data variable. The most natural way to represent the relationship between variables is via a mathematical function or map. We usually assume that this mathematical function is not completely known, but can be approximated well given enough computing power and data. Thus, data scientists have to understand how best to approximate and represent functions using the least amount of computer processing and memory.

Optimization. Given a class of mathematical models, we wish to find the best possible model in that class. This requires some kind of efficient search or optimization procedure. The optimization step can be viewed as a process of fitting or calibrating a function to observed data. This step usually requires knowledge of optimization algorithms and efficient computer coding or programming.

Probability and Statistics. In general, the data used to fit the model is viewed as a realization of a random process or numerical vector, whose probability law determines the accuracy with which we can predict future observations. Thus, in order to quantify the uncertainty inherent in making predictions about the future, and the sources of error in the model, data scientists need a firm grasp of probability theory and statistical inference.

Supervised and Unsupervised Learning

Given an input or *feature* vector $\boldsymbol{x}$, one of the main goals of machine learning is to predict an output or *response* variable y. For example, $\boldsymbol{x}$ could be a digitized signature and y a binary variable that indicates whether the signature is genuine or false. Another example is where $\boldsymbol{x}$ represents the weight and smoking habits of an expecting mother and y the birth weight of the baby. The data science attempt at this prediction is encoded in a mathematical function g, called the *prediction function*, which takes as an input $\boldsymbol{x}$ and outputs a guess $g(\boldsymbol{x})$ for y (denoted by $\widehat{y}$, for example). In a sense, g encompasses all the information about the relationship between the variables $\boldsymbol{x}$ and y, excluding the effects of chance and randomness in nature.

In *regression* problems, the response variable y can take any real value. In contrast, when y can only lie in a finite set, say $y \in \{0, \ldots, c-1\}$, then predicting y is conceptually the same as classifying the input $\boldsymbol{x}$ into one of c categories, and so prediction becomes a *classification* problem.

We can measure the accuracy of a prediction $\widehat{y}$ with respect to a given response y by using some *loss function* $\text{Loss}(y, \widehat{y})$. In a regression setting the usual choice is the squared-error loss $(y-\widehat{y})^2$. In the case of classification, the zero–one (also written 0–1) loss function $\text{Loss}(y, \widehat{y}) = \mathbb{1}\{y \neq \widehat{y}\}$ is often used, which incurs a loss of 1 whenever the predicted class $\widehat{y}$ is not equal to the class y.

The word *error* is often used as a measure of distance between a "true" object y and some approximation $\widehat{y}$ thereof. If y is real-valued, the absolute error $|y-\widehat{y}|$ and the squared error $(y-\widehat{y})^2$ are both well-established error concepts, as are the norm $\|\boldsymbol{y}-\widehat{\boldsymbol{y}}\|$ and squared norm $\|\boldsymbol{y}-\widehat{\boldsymbol{y}}\|^2$ for vectors. The squared error $(y-\widehat{y})^2$ is just one example of a loss function.

It is unlikely that any mathematical function g will be able to make accurate predictions for all possible pairs $(\boldsymbol{x}, y)$ one may encounter in Nature. One reason for this is that, even with the same input $\boldsymbol{x}$, the output y may be different, depending on chance circumstances or randomness. For this reason, we adopt a probabilistic approach and assume that each pair $(\boldsymbol{x}, y)$ is the outcome of a random pair $(\boldsymbol{X}, Y)$ that has some joint probability density $f(\boldsymbol{x}, y)$. We then assess the predictive performance via the expected loss, usually called the *risk*, for g:

$$\ell(g) = \mathbb{E}\,\text{Loss}(Y, g(\boldsymbol{X})).$$

For example, in the classification case with zero–one loss function the risk is equal to the probability of incorrect classification: $\ell(g) = \mathbb{P}[Y \neq g(\boldsymbol{X})]$. In this context, the prediction function g is called a *classifier*. Given the distribution of $(\boldsymbol{X}, Y)$ and any loss function, we can in principle find the best possible $g^* := \text{argmin}_g\, \mathbb{E}\,\text{Loss}(Y, g(\boldsymbol{X}))$ that yields the smallest risk $\ell^* := \ell(g^*)$. We will see in Chapter 7 that in the classification case with $y \in \{0, \ldots, c-1\}$ and $\ell(g) = \mathbb{P}[Y \neq g(\boldsymbol{X})]$, we have

$$g^*(\boldsymbol{x}) = \underset{y \in \{0,\ldots,c-1\}}{\text{argmax}}\; f(y \,|\, \boldsymbol{x}),$$

where $f(y \mid \boldsymbol{x}) = \mathbb{P}[Y = y \mid \boldsymbol{X} = \boldsymbol{x}]$ is the conditional probability of $Y = y$ given $\boldsymbol{X} = \boldsymbol{x}$. As already mentioned, for regression the most widely-used loss function is the squared-error loss. In this setting, the optimal prediction function g^* is often called the *regression function*. The following theorem specifies its exact form.

Theorem 2.1: Optimal Prediction Function for Squared-Error Loss

For the squared-error loss $\text{Loss}(y, \widehat{y}) = (y - \widehat{y})^2$, the optimal prediction function g^* is equal to the conditional expectation of Y given $\boldsymbol{X} = \boldsymbol{x}$:

$$g^*(\boldsymbol{x}) = \mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}].$$

Modeling Data

The first step in any data analysis is to *model* the data in one form or another. For example, in an *unsupervised* learning setting with data represented by a vector $\boldsymbol{x} = [x_1, \ldots, x_p]^\top$, a very general model is to assume that $\boldsymbol{x}$ is the outcome of a random vector $\boldsymbol{X} = [X_1, \ldots, X_p]^\top$ with some unknown pdf f. The model can then be refined by assuming a specific form of f.

When given a sequence of such data vectors $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$, one of the simplest models is to assume that the corresponding random vectors $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ are *independent and identically distributed (iid)*. We write

$$\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \overset{\text{iid}}{\sim} f \quad \text{or} \quad \boldsymbol{X}_1, \ldots, \boldsymbol{X}_n \overset{\text{iid}}{\sim} \mathsf{Dist},$$

to indicate that the random vectors form an iid sample from a sampling pdf f or sampling distribution Dist. This model formalizes the notion that the knowledge about one variable does not provide extra information about another variable. The main theoretical use of independent data models is that the joint density of the random vectors $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ is simply the *product* of the marginal ones; see Theorem C.1. Specifically,

$$f_{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = f(\boldsymbol{x}_1) \cdots f(\boldsymbol{x}_n).$$

In most models of this kind, our approximation or model for the sampling distribution is specified up to a small number of parameters. That is, $g(\boldsymbol{x})$ is of the form $g(\boldsymbol{x} \mid \boldsymbol{\beta})$ which is known up to some parameter vector $\boldsymbol{\beta}$. Examples for the one-dimensional case ($p = 1$) include the $\mathcal{N}(\mu, \sigma^2)$, $\mathsf{Bin}(n, p)$, and $\mathsf{Exp}(\lambda)$ distributions.

Typically, the parameters are unknown and must be estimated from the data. In a non-parametric setting the whole sampling distribution would be unknown. To visualize the underlying sampling distribution from outcomes $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n$ one can use graphical representations such as histograms, density plots, and empirical cumulative distribution functions.

If the order in which the data were collected (or their labeling) is not informative or relevant, then the joint pdf of $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ satisfies the symmetry:

$$f_{X_1,\ldots,X_n}(\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n) = f_{X_{\pi_1},\ldots,X_{\pi_n}}(\boldsymbol{x}_{\pi_1}, \ldots, \boldsymbol{x}_{\pi_n}) \tag{2.26}$$

for any permutation $\pi_1, \ldots, \pi_n$ of the integers $1, \ldots, n$. We say that the infinite sequence $\boldsymbol{X}_1, \boldsymbol{X}_2, \ldots$ is *exchangeable* if this permutational invariance (2.26) holds for any finite subset of the sequence.

Figure 2.12 illustrates the modeling tradeoffs. The keywords within the triangle represent various modeling paradigms. A few keywords have been highlighted, symbolizing their importance in modeling. The specific meaning of the keywords does not concern us here, but the point is there are many models to choose from, depending on what assumptions are made about the data.

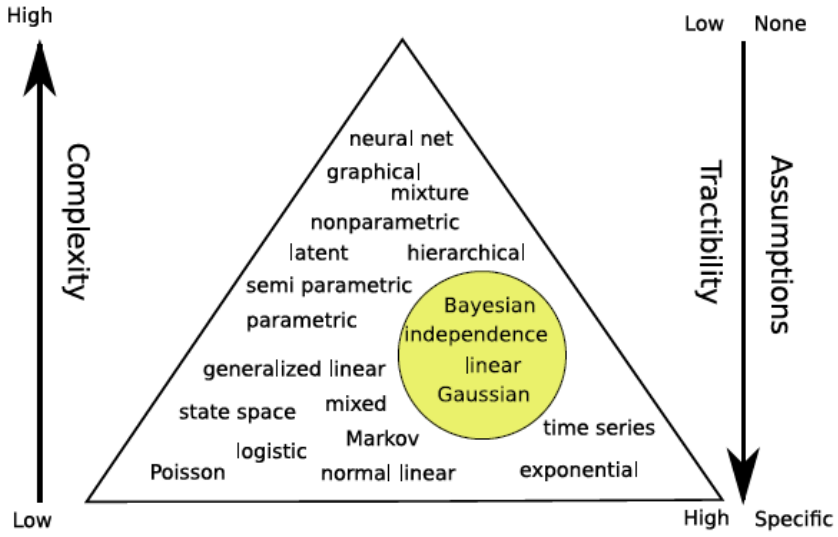

Figure 2.12: Illustration of the modeling dilemma. Complex models are more generally applicable, but may be difficult to analyze. Simple models may be highly tractable, but may not describe the data accurately. The triangular shape signifies that there are a great many specific models but not so many generic ones.

On the one hand, models that make few assumptions are more widely applicable, but at the same time may not be very mathematically tractable or provide insight into the nature of the data. On the other hand, very specific models may be easy to handle and interpret, but may not match the data very well. This tradeoff between the tractability and applicability of the model is very similar to the approximation–estimation tradeoff described in Section 2.4.

In the typical *unsupervised* setting we have a training set $\tau = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ that is viewed as the outcome of n iid random variables $\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n$ from some unknown pdf f. The objective is then to learn or estimate f from the finite training data. To put the learning in a similar framework as for supervised learning discussed in the preceding Sections 2.3–2.5, we begin by specifying a class of probability density functions $\mathcal{G}_p := \{g(\cdot \,|\, \boldsymbol{\theta}), \boldsymbol{\theta} \in \boldsymbol{\Theta}\}$, where $\boldsymbol{\theta}$ is a parameter in some subset $\boldsymbol{\Theta}$ of $\mathbb{R}^p$. We now seek the best g in $\mathcal{G}_p$ to minimize some risk. Note that $\mathcal{G}_p$ may not necessarily contain the true f even for very large p.

> We stress that our notation $g(\boldsymbol{x})$ has a different meaning in the supervised and unsupervised case. In the supervised case, g is interpreted as a prediction function for a response y; in the unsupervised setting, g is an approximation of a density f.

For each $\boldsymbol{x}$ we measure the discrepancy between the true model $f(\boldsymbol{x})$ and the hypothesized model $g(\boldsymbol{x} \,|\, \boldsymbol{\theta})$ using the loss function

$$\text{Loss}(f(\boldsymbol{x}), g(\boldsymbol{x} \,|\, \boldsymbol{\theta})) = \ln \frac{f(\boldsymbol{x})}{g(\boldsymbol{x} \,|\, \boldsymbol{\theta})} = \ln f(\boldsymbol{x}) - \ln g(\boldsymbol{x} \,|\, \boldsymbol{\theta}).$$

The expected value of this loss (that is, the risk) is thus

$$\ell(g) = \mathbb{E} \ln \frac{f(\boldsymbol{X})}{g(\boldsymbol{X} \,|\, \boldsymbol{\theta})} = \int f(\boldsymbol{x}) \ln \frac{f(\boldsymbol{x})}{g(\boldsymbol{x} \,|\, \boldsymbol{\theta})} \, \mathrm{d}\boldsymbol{x}. \tag{2.27}$$

The integral in (2.27) provides a fundamental way to measure the distance between two densities and is called the *Kullback–Leibler (KL) divergence*[2] between f and $g(\cdot \,|\, \boldsymbol{\theta})$. Note that the KL divergence is not symmetric in f and $g(\cdot \,|\, \boldsymbol{\theta})$. Moreover, it is always greater than or equal to 0 (see Exercise 15) and equal to 0 when $f = g(\cdot \,|\, \boldsymbol{\theta})$.

Linear Model

In a *supervised* setting, where the data is represented by a vector $\boldsymbol{x}$ of explanatory variables and a response y, the general model is that $(\boldsymbol{x}, y)$ is an outcome of $(\boldsymbol{X}, Y) \sim f$ for some unknown f. And for a training sequence $(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_n, y_n)$ the default model assumption is that $(\boldsymbol{X}_1, Y_1), \ldots, (\boldsymbol{X}_n, Y_n) \sim_{\text{iid}} f$. As explained in Section 2.2, the analysis primarily involves the conditional pdf $f(y \mid \boldsymbol{x})$ and in particular (when using the squared-error loss) the conditional expectation $g^*(\boldsymbol{x}) = \mathbb{E}[Y \mid \boldsymbol{X} = \boldsymbol{x}]$. The resulting representation (2.2) allows us to then write the response at $\boldsymbol{X} = \boldsymbol{x}$ as a function of the feature $\boldsymbol{x}$ plus an error term: $Y = g^*(\boldsymbol{x}) + \varepsilon(\boldsymbol{x})$.

This leads to the simplest and most important model for supervised learning, where we choose a *linear* class $\mathcal{G}$ of prediction or guess functions and assume that it is rich enough to contain the true g^*. If we further assume that, conditional on $\boldsymbol{X} = \boldsymbol{x}$, the error term ε does not depend on $\boldsymbol{x}$, that is, $\mathbb{E}\,\varepsilon = 0$ and $\mathbb{V}\text{ar}\,\varepsilon = \sigma^2$, then we obtain the following model.

Definition 2.1: Linear Model

In a *linear model* the response Y depends on a p-dimensional explanatory variable $\boldsymbol{x} = [x_1, \ldots, x_p]^\top$ via the linear relationship

$$Y = \boldsymbol{x}^\top \boldsymbol{\beta} + \varepsilon, \tag{2.29}$$

where $\mathbb{E}\,\varepsilon = 0$ and $\mathbb{V}\text{ar}\,\varepsilon = \sigma^2$.

Note that (2.29) is a model for a single pair $(\boldsymbol{x}, Y)$. The model for the training set $\{(\boldsymbol{x}_i, Y_i)\}$ is simply that each Y_i satisfies (2.29) (with $\boldsymbol{x} = \boldsymbol{x}_i$) and that the $\{Y_i\}$ are independent. Gathering all responses in the vector $\boldsymbol{Y} = [Y_1, \ldots, Y_n]^\top$, we can write

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2.30}$$

where $\boldsymbol{\varepsilon} = [\varepsilon_1, \ldots, \varepsilon_n]^\top$ is a vector of iid copies of ε and $\mathbf{X}$ is the so-called *model matrix*, with rows $\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top$. Linear models are fundamental building blocks of statistical learning algorithms. For this reason, a large part of Chapter 5 is devoted to linear regression models.

Learning Model

- *Any* model for data is likely to be *wrong*. For example, real data (as opposed to computer-generated data) are often assumed to come from a normal distribution, which is never exactly true. However, an important advantage of using a normal distribution is that it has many nice mathematical properties, as we will see in Section 2.7.
- Most data models depend on a number of unknown parameters, which need to be estimated from the observed data.
- Any model for real-life data needs to be *checked* for suitability. An important criterion is that data simulated from the model should resemble the observed data, at least for a certain choice of model parameters.

Here are some guidelines for choosing a model.

- First establish the type of the features (quantitative, qualitative, discrete, continuous, etc.).
- Assess whether the data can be assumed to be independent across rows or columns.
- Decide on the level of generality of the model. For example, should we use a simple model with a few unknown parameters or a more generic model that has a large number of parameters? Simple specific models are easier to fit to the data (low estimation error) than more general models, but the fit itself may not be accurate (high approximation error).
- Decide on using a classical (frequentist) or Bayesian model.

Multivariate Normal Models

A standard model for numerical observations $x_1, \ldots, x_n$ (forming, e.g., a column in a spreadsheet or data frame) is that they are the outcomes of iid normal random variables

$$X_1, \ldots, X_n \overset{\text{iid}}{\sim} \mathcal{N}(\mu, \sigma^2).$$

It is helpful to view a normally distributed random variable as a simple transformation of a standard normal random variable. To wit, if Z has a standard normal distribution, then $X = \mu + \sigma Z$ has a $\mathcal{N}(\mu, \sigma^2)$ distribution. The generalization to n dimensions is discussed in Appendix C.7. We summarize the main points: Let $Z_1, \ldots, Z_n \overset{\text{iid}}{\sim} \mathcal{N}(0, 1)$. The pdf of $\boldsymbol{Z} = [Z_1, \ldots, Z_n]^\top$ (that is, the joint pdf of $Z_1, \ldots, Z_n$) is given by

$$f_{\boldsymbol{Z}}(\boldsymbol{z}) = \prod_{i=1}^{n} \frac{1}{\sqrt{2\pi}} \mathrm{e}^{-\frac{1}{2} z_i^2} = (2\pi)^{-\frac{n}{2}} \mathrm{e}^{-\frac{1}{2} \boldsymbol{z}^\top \boldsymbol{z}}, \quad \boldsymbol{z} \in \mathbb{R}^n. \tag{2.31}$$

Normal Linear Models

Normal linear models combine the simplicity of the linear model with the tractability of the Gaussian distribution. They are the principal model for traditional statistics, and include the classic linear regression and analysis of variance models.

Definition 2.3: Normal Linear Model

In a *normal linear model* the response Y depends on a p-dimensional explanatory variable $\boldsymbol{x} = [x_1, \ldots, x_p]^\top$, via the linear relationship

$$Y = \boldsymbol{x}^\top \boldsymbol{\beta} + \varepsilon, \tag{2.34}$$

where $\varepsilon \sim \mathcal{N}(0, \sigma^2)$.

Thus, a normal linear model is a linear model (in the sense of Definition 2.1) with normal error terms. Similar to (2.30), the corresponding normal linear model for the whole training set $\{(\boldsymbol{x}_i, Y_i)\}$ has the form

$$\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \boldsymbol{\varepsilon}, \tag{2.35}$$

where $\mathbf{X}$ is the model matrix comprised of rows $\boldsymbol{x}_1^\top, \ldots, \boldsymbol{x}_n^\top$ and $\boldsymbol{\varepsilon} \sim \mathcal{N}(\mathbf{0}, \sigma^2 \mathbf{I}_n)$. Consequently, $\boldsymbol{Y}$ can be written as $\boldsymbol{Y} = \mathbf{X}\boldsymbol{\beta} + \sigma \boldsymbol{Z}$, where $\boldsymbol{Z} \sim \mathcal{N}(\mathbf{0}, \mathbf{I}_n)$, so that $\boldsymbol{Y} \sim \mathcal{N}(\mathbf{X}\boldsymbol{\beta}, \sigma^2 \mathbf{I}_n)$. It follows from (2.33) that its joint density is given by

$$g(\boldsymbol{y} \,|\, \boldsymbol{\beta}, \sigma^2, \mathbf{X}) = (2\pi\sigma^2)^{-\frac{n}{2}} \, \mathrm{e}^{-\frac{1}{2\sigma^2}\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|^2}. \tag{2.36}$$

Estimation of the parameter $\boldsymbol{\beta}$ can be performed via the least-squares method.

An estimate can also be obtained via the maximum likelihood method.

This simply means finding the parameters σ^2 and $\boldsymbol{\beta}$ that maximize the likelihood of the outcome $\boldsymbol{y}$, given by the right-hand side of (2.36). It is clear that for every value of σ^2 the likelihood is maximal when $\|\boldsymbol{y} - \mathbf{X}\boldsymbol{\beta}\|^2$ is minimal. As a consequence, the maximum likelihood estimate for $\boldsymbol{\beta}$ is the same as the least-squares estimate

Unsupervised Learning

Definition: When there is no distinction between response and explanatory variables, unsupervised methods are required to learn the structure of the data.

Various unsupervised learning techniques: **Density Estimation, Clustering, And Principal Component Analysis**.

Important tools in unsupervised learning: **Cross-Entropy Training Loss, Mixture Models, The Expectation–Maximization (EM) Algorithm, and the Singular Value Decomposition (SVD).**

Introduction

In contrast to supervised learning, where an "output" (response) variable y is explained by an "input" (explanatory) vector $\boldsymbol{x}$, in unsupervised learning there is no response variable and the overall goal is to extract useful information and patterns from the data, e.g., in the form $\tau = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ or as a matrix $\mathbf{X}^\top = [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$. In essence, the objective of unsupervised learning is to learn about the underlying probability distribution of the data.

If the data forms an Independent and Identically Distributed **(iid)** sample from some unknown distribution, the "empirical distribution" of the data provides valuable information about the unknown distribution.

Most unsupervised learning techniques focus on identifying certain traits of the underlying distribution, such as its local maximizers.

A related idea is to partition the data into clusters of points that are in some sense "similar" to each other.

We formulate the clustering problem in terms of a mixture model.

In particular, the data are assumed to come from a mixture of (usually Gaussian) distributions, and the objective is to recover the parameters of the mixture distributions from the data.

The principal tool for parameter estimation in mixture models is the EM algorithm.

The unsupervised learning technique called Principal Component Analysis (PCA), is an important tool for reducing the dimensionality of the data.

PCA can be used for Variable Selection* and *Dimensionality Reduction, to make models easier to train and increase their predictive power.

Risk and Loss in Unsupervised Learning

In unsupervised learning, the training data $\mathcal{T} := \{\boldsymbol{X}_1, \ldots, \boldsymbol{X}_n\}$ only consists of (what are usually assumed to be) independent copies of a feature vector $\boldsymbol{X}$; there is no response data. Suppose our objective is to learn the unknown pdf f of $\boldsymbol{X}$ based on an outcome $\tau = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\}$ of the training data $\mathcal{T}$.