Statistics and Machine Learning 1

Lecture 3A: Multivariate Exploratory Data Analysis

Mark Muldoon
Department of Mathematics, Alan Turing Building
University of Manchester

Week 3

Motivation

- In real applications, we almost almost always have multiple features of different things measured, and are so in a *multivariate* rather than *univariate* situation.
- In some cases, the multivariate analysis simply generalises a univaraite technique.
- In other cases, there is no univariate equivalent of the multivariate technique.
- Here we will consider both generalisations of univariate techniques and purely multivariate techniques.

Professional Skills

- Last lecture I mentioned assessment of data quality; but that is really part of a general professionalism that is important for all data scientists and which interacts with, but is not limited to, the technical aspects.
- The Royal Statistical Society (RSS) publishes a *Code of Conduct* at

  http://www.rss.org.uk/RSS/Join_the_RSS/Code_of_conduct/RSS/Join_the_RSS/Code_of_conduct.aspx

It says that fellows of the RSS should:

- Act in the public interest
- Fulfill their obligations to employers and clients
- Fulfill their obligations to the profession and the Society
- At all time show professional competence and integrity

Be clear about what you have (and haven't) done

An important point is from 5(a) of this code of conduct:

> *Fellows should ensure that any statistical analysis attributed to them by an employer, client or colleagues is amplified, if necessary, by a description of the way the data were selected, and the way any apparently erroneous data were corrected or rejected. Explicit statements will generally be needed about the assumptions made when selecting a method of analysis and any consequences of the selected approach should be identified. Views or opinions based on general knowledge or belief should be clearly distinguished from views or opinions derived from the statistical analyses being reported.*

Data types

https://en.wikibooks.org/wiki/Statistics/Different_Types_of_Data

Data can be:

- Nominal or *categorical* (e.g. colours, car names): not ordered; cannot be added or compared; can be relabelled.
- *Ordinal* (e.g. small/medium/large): sometimes represented by numbers; can be ordered, but differences or ratios are not meaningful.
- *Measurement*: meaningful numbers, on which (some) operations make sense. They can be:
  - *Discrete* (e.g. publication year, number of cylinders): typically integer.
  - *Continuous* (e.g. height): precision limited only by measurement accuracy.

  Measurements can be in an interval scale (e.g. temperature in degrees Celsius), ratio scale (say, weights in kg), or circular scale (time of day on the 24 hr clock), depending on the 0 value and on which operations yield meaningful results.

This week's data

We will use the 'auto' dataset from the UCI Machine Learning Repository. The first 16 of its 398 lines looks like:

```
18.0   8   307.0    130.0    3504.    12.0   70  1 "chevrolet chevelle malibu"
15.0   8   350.0    165.0    3693.    11.5   70  1 "buick skylark 320"
18.0   8   318.0    150.0    3436.    11.0   70  1 "plymouth satellite"
16.0   8   304.0    150.0    3433.    12.0   70  1 "amc rebel sst"
17.0   8   302.0    140.0    3449.    10.5   70  1 "ford torino"
15.0   8   429.0    198.0    4341.    10.0   70  1 "ford galaxie 500"
14.0   8   454.0    220.0    4354.     9.0   70  1 "chevrolet impala"
14.0   8   440.0    215.0    4312.     8.5   70  1 "plymouth fury iii"
14.0   8   455.0    225.0    4425.    10.0   70  1 "pontiac catalina"
15.0   8   390.0    190.0    3850.     8.5   70  1 "amc ambassador dpl"
15.0   8   383.0    170.0    3563.    10.0   70  1 "dodge challenger se"
14.0   8   340.0    160.0    3609.     8.0   70  1 "plymouth 'cuda 340"
15.0   8   400.0    150.0    3761.     9.5   70  1 "chevrolet monte carlo"
14.0   8   455.0    225.0    3086.    10.0   70  1 "buick estate wagon (sw)"
24.0   4   113.0    95.00    2372.    15.0   70  3 "toyota corona mark ii"
22.0   6   198.0    95.00    2833.    15.5   70  1 "plymouth duster"
```

Metadata

- The data are features of car models from 1983.
- The *metadata* file contains information on the columns:

```
Attribute Information:

1. mpg:           continuous
2. cylinders:     multi-valued discrete
3. displacement:  continuous
4. horsepower:    continuous
5. weight:        continuous
6. acceleration:  continuous
7. model year:    multi-valued discrete
8. origin:        multi-valued discrete
9. car name:      string (unique for each instance)
```

Importing into Python

- So far, we have mainly used `pandas`, but it is possible to import numerical data directly into a `numpy` array:

```
import numpy as np
import scipy.stats as st
import matplotlib.pyplot as plt
f = open('./auto.txt', 'r')
mpg, cyl, dis, hp, wgt, acc, yr, org = np.loadtxt(
  f, unpack=True, usecols = (0,1,2,3,4,5,6,7))
```

- Note that here we are ignoring the list of names in the final column. Where the data is a general mix of text, integers, floats, dates etc. then pandas or similar will normally be best!

Importing into R

Getting the into data to R is also straightforward,

```
auto.data <- read.table("auto.txt", header=FALSE, sep="\t")
auto.data
```

and yields a data frame with 398 rows that begins with:

	V1	V2	V3	V4	V5	V6	V7	V8	V9
1	18	8	307.0	130	3504	12.0	70	1	chevrolet chevelle malibu
2	15	8	350.0	165	3693	11.5	70	1	buick skylark 320
3	18	8	318.0	150	3436	11.0	70	1	plymouth satellite
4	16	8	304.0	150	3433	12.0	70	1	amc rebel sst
5	17	8	302.0	140	3449	10.5	70	1	ford torino
6	15	8	429.0	198	4341	10.0	70	1	ford galaxie 500
7	14	8	454.0	220	4354	9.0	70	1	chevrolet impala
8	14	8	440.0	215	4312	8.5	70	1	plymouth fury iii
9	14	8	455.0	225	4425	10.0	70	1	pontiac catalina

Renaming variables in R

A simple example of the kind of data wrangling we might need to do:

```
library(plyr)

auto.data <- rename(auto.data, c(
  "V1"="MPG",
  "V2"="Cylinders",
  "V3"="Displacement",
  "V4"="Horsepower",
  "V5"="Weight",
  "V6"="Acceleration",
  "V7"="ModelYear",
  "V8"="Origin",
  "V9"="CarName"
  ))
```

A quick overview of the data: R

It's easy to get summary statistics for all the columns in an R data frame.

```
summary( auto.data )
```

```
      MPG          Cylinders      Displacement     Horsepower        Weight      Acceleration     ModelYear   
 Min.   : 9.00   Min.   :3.000   Min.   : 68.0   Min.   : 46.0   Min.   :1613   Min.   : 8.00   Min.   :70.00  
 1st Qu.:17.50   1st Qu.:4.000   1st Qu.:104.2   1st Qu.: 75.0   1st Qu.:2224   1st Qu.:13.82   1st Qu.:73.00  
 Median :23.00   Median :4.000   Median :148.5   Median : 93.5   Median :2804   Median :15.50   Median :76.00  
 Mean   :23.51   Mean   :5.455   Mean   :193.4   Mean   :104.5   Mean   :2970   Mean   :15.57   Mean   :76.01  
 3rd Qu.:29.00   3rd Qu.:8.000   3rd Qu.:262.0   3rd Qu.:126.0   3rd Qu.:3608   3rd Qu.:17.18   3rd Qu.:79.00  
 Max.   :46.60   Max.   :8.000   Max.   :455.0   Max.   :230.0   Max.   :5140   Max.   :24.80   Max.   :82.00  
                                                 NA's   :6                                                     
     Origin          CarName         
 Min.   :1.000   Length:398         
 1st Qu.:1.000   Class :character   
 Median :1.000   Mode  :character   
 Mean   :1.573                      
 3rd Qu.:2.000                      
 Max.   :3.000                      
```

A quick overview of the data: Python

To get a similar summary in Python one needs to use `pandas` rather than `numpy`:

```
import pandas as pd

colNames = [
    "MPG", "Cylinders", "Displacement", "Horsepower",
    "Weight", "Acceleration", "ModelYear", "Origin", "CarName"
]
auto_df = pd.read_table('./auto.txt', sep='\t', names=colNames )
auto_df.describe()
```

	MPG	Cylinders	Displacement	Horsepower	Weight	Acceleration	ModelYear	Origin
count	398.000000	398.000000	398.000000	392.000000	398.000000	398.000000	398.000000	398.000000
mean	23.514573	5.454774	193.425879	104.469388	2970.424623	15.568090	76.010050	1.572864
std	7.815984	1.701004	104.269838	38.491160	846.841774	2.757689	3.697627	0.802055
min	9.000000	3.000000	68.000000	46.000000	1613.000000	8.000000	70.000000	1.000000
25%	17.500000	4.000000	104.250000	75.000000	2223.750000	13.825000	73.000000	1.000000
50%	23.000000	4.000000	148.500000	93.500000	2803.500000	15.500000	76.000000	1.000000
75%	29.000000	8.000000	262.000000	126.000000	3608.000000	17.175000	79.000000	2.000000
max	46.600000	8.000000	455.000000	230.000000	5140.000000	24.800000	82.000000	3.000000

Representation of data

- We will represent the data as an $n \times p$ matrix $\boldsymbol{X} = [x_{ia}]_{i=1,\ldots,n}^{a=1,\ldots,p}$ with which in Python we will explicitly save as an array

```
X = [mpg, dis, wgt, acc]
```

  so here, in X, $p = 4$ and $n = 398$.

- This can be written in terms of rows and columns,

$$\boldsymbol{X} = [\mathbf{x}_i^\top]_{i=1}^n = \begin{bmatrix} \mathbf{x}_1^\top \\ \mathbf{x}_2^\top \\ \vdots \\ \mathbf{x}_n^\top \end{bmatrix} = [\mathbf{y}_a]_{a=1}^p = \begin{bmatrix} \mathbf{y}_1 & \mathbf{y}_2 & \cdots & \mathbf{y}_p \end{bmatrix},$$

  where each $\mathbf{y}$ is essentially a univariate dataset and each $\mathbf{x}^\top$ is a set of p observations of an individual.

- Therefore, $\mathbf{x}_i$ will normally be thought of as a random *vector*—we'll discuss this further in part C of the lecture.

Sample Mean

- We write the sample mean as

$$\langle \mathbf{x} \rangle := \frac{1}{n} \sum_{i=1}^{n} \mathbf{x}_i. \tag{1}$$

- *Note:* We are thinking of the n samples as coming from a population of size $N \gg n$, so that each $\mathbf{x}_i$ is a copy of a random vector of length-p, $\mathbf{X}$, which has its own mean $\mathbb{E}[\mathbf{X}]$. In particular, $\mathbb{E}[\mathbf{X}] = \mathbb{E}[\langle \mathbf{x} \rangle]$, but generally $\mathbb{E}[\mathbf{X}] \neq \langle \mathbf{x} \rangle$.

```
colMeans(auto.data[,1:6], na.rm = TRUE)
```

```
       MPG    Cylinders Displacement   Horsepower       Weight Acceleration
 23.514573     5.454774   193.425879   104.469388  2970.424623    15.568090
```

```
xbar = np.mean(X,1)
print(xbar)
```

```
[ 23.51457286   193.4258794   2970.42462312    15.56809045 ]
```

Sample Covariance Matrix

The sample covariance (which is biased; divide by $n-1$ for the unbiased estimate) is

$$\boldsymbol{S} = [S_{ab}], \qquad \text{where} \quad S_{ab} = \frac{1}{n}\sum_{i=1}^{n}(x_{ia} - \langle x_a \rangle)(x_{ib} - \langle x_b \rangle),$$

with $a = 1, \ldots, p$ and $b = 1, \ldots, p$

Defaults to unbiased:

```
cov(auto.data[,1:6], use = "na.or.complete")
```

```
                     MPG   Cylinders Displacement  Horsepower      Weight Acceleration
MPG            60.918142  -10.352928    -657.5852  -233.85793  -5517.4407     9.115514
Cylinders     -10.352928    2.909696     169.7219    55.34824   1300.4244    -2.375052
Displacement -657.585207  169.721949   10950.3676  3614.03374  82929.1001  -156.994435
Horsepower    -233.857926   55.348244    3614.0337  1481.56939  28265.6202   -73.186967
Weight       -5517.440704 1300.424363   82929.1001 28265.62023 721484.7090  -976.815253
Acceleration     9.115514   -2.375052    -156.9944   -73.18697   -976.8153     7.611331
```

Defaults to biased:

```
S = np.cov(X)
print(S)
```

```
[[  6.10896108e+01  -6.55402318e+02  -5.50521175e+03   9.05892966e+00]
 [ -6.55402318e+02   1.08721992e+04   8.23684232e+04  -1.56332976e+02]
 [ -5.50521175e+03   8.23684232e+04   7.17140991e+05  -9.74899011e+02]
 [  9.05892966e+00  -1.56332976e+02  -9.74899011e+02   7.60484823e+00]]
```

Correlation Matrix

The sample correlation matrix has elements that correspond to the slopes of the (linear) regression lines between variables:

$$\boldsymbol{R} = [R_{ab}], \text{ where } \quad R_{ab} = \frac{S_{ab}}{\sqrt{S_{aa} S_{bb}}} \, .$$

```r
cor(auto.data[,1:6], use = "na.or.complete")
```

```
                    MPG  Cylinders Displacement Horsepower     Weight Acceleration
MPG           1.0000000 -0.7776175   -0.8051269 -0.7784268 -0.8322442    0.4233285
Cylinders    -0.7776175  1.0000000    0.9508233  0.8429834  0.8975273   -0.5046834
Displacement -0.8051269  0.9508233    1.0000000  0.8972570  0.9329944   -0.5438005
Horsepower   -0.7784268  0.8429834    0.8972570  1.0000000  0.8645377   -0.6891955
Weight       -0.8322442  0.8975273    0.9329944  0.8645377  1.0000000   -0.4168392
Acceleration  0.4233285 -0.5046834   -0.5438005 -0.6891955 -0.4168392    1.0000000
```

```python
R = np.corrcoef(X)
print(R)
```

```
[[ 1.         -0.80420282 -0.83174093  0.42028891]
 [-0.80420282  1.          0.93282415 -0.54368408]
 [-0.83174093  0.93282415  1.         -0.41745732]
 [ 0.42028891 -0.54368408 -0.41745732  1.        ]]
```