CM3005 Data Science

Course Notes

WEEK 3 COMPLETED
UPDATED APRIL 20, 2021 BY TONI HE

Week 1

1.0 Introduction to the course:

This module will introduce the ***theoretical background*** and ***practical aspect*** of data science.

Key concepts: python programming (NumPy, pandas, MatplotLib), Jupyter notebook, environment, data processing, data visualization, statistics, linear algebra, machine learning.

TOPIC 1: INTRODUCTION TO DATA SCIENCE

This topic will introduce the scope of data science and its impact ins academia and industry.

The main concept in data science will be discussed.

The topic will introduce the Python programming language as the language of choice for many data scientists.

The Jupiter environment for Python development will be introduced and a range of examples will be presented.

TOPIC 2: DATA PROCESSING

This topic will introduce a range of data processing techniques.

The tasks will be demonstrated by using the Python libraries NumPy and pandas.

The topic will present some statistical tasks involving measures of central tendency and measures of spread.

The main concepts of linear algebra will be presented and accompanied with solutions to some practical tasks.

TOPIC 3: INTRODUCTION TO DATA VISUALIZATION

This topic will introduce the main concepts of data visualization.

Different approaches to handling qualitative and quantitative data will be reviewed.

Different types of diagrams and their impact on the intended audience will be discussed and demonstrated.

Real-world examples will be illustrated by using the Python library MatplotLib.

TOPIC 4: STATISTICS

This topic will introduce theoretical foundations of statistics.

The main data types within the context of statistics will be discussed.

The topic will present the theory in which underpins the descriptive and inferential statistics.

The processing of different types of variables will be discussed and demonstrated.

TOPIC 5: MACHINE LEARNING (PART 1)

This topic will introduce the theoretical foundations of machine learning (ML).

The main Python libraries for ML will be discussed and demonstrated.

The concept of model validation and techniques for selecting the best model will be reviewed.

The processing of different types of data using feature engineering will be explored and demonstrated.

TOPIC 6: BASIC TEXT PROCESSING

This topic will introduce fundamental concepts and principles of processing unstructured data.

The main technique use for processing text will be discussed and demonstrated.

The concept of regular expressions and their application to text data will be reviewed.

The practical application of the above techniques to real-world data samples will be explored and demonstrated.

TOPIC 7: NATURAL LANGUAGE PROCESSING

This topic will introduce the fundamental concepts and principles of natural language processing (NLP).

Python libraries for processing natural language data will be discussed and demonstrated.

Techniques for representing word meanings will be reviewed and applied to various NLP tasks.

The practical application of the above techniques within NLP pipelines will be explored and demonstrated.

TOPIC 8: ADVANCED DATA VISUALIZATION

This topic will discuss some advanced visualization techniques.

The topic will discuss large and multidimensional datasets and the challenges accompanying visualization.

Real world examples of multidimensional datasets will be visualized by using:

- Heat Maps
- Parallel coordinates

TOPIC 9: MACHINE LEARNING (PART 2)

This topic will introduce the more advanced techniques in machine learning (ML).

The role of Bayes' theorem and its application to supervised learning will be demonstrated and discussed.

Practical examples of linear regression, support vector machines and decision trees will be reviewed.

Unsupervised learning using kMeans clustering will be explored and demonstrated.

TOPIC 10: CASE STUDIES

This topic will introduce several real-world examples of data science practice.

Topical issues will explore through interviews with practicing data scientists, such as:

- Evaluation techniques applied in industrial settings.
- The skills and competencies they look for in new hires.
- Bias, ethics, and diversity in data science.

1.101 Introduction to Topic 1: The scope of data science

DEFINING DATA SCIENCE:

Data science is a new and emerging subject, still evolving and re-defining itself.

Data science is fundamentally interdisciplinary.

It requires knowledge and skills in three major directions:

Mathematics: more specifically, statistics and linear algebra among others.

Computer Science: machine learning, big data, data visualization etc.

Domain-specific area: the specific area the data is related to.

MAIN PURPOSES OF DATA SCIENCE:

- Recognize sources of data
- Extract insights from data
- Present data in an informative and accessible way
- Enable informed decisions based on data
- Predict future outcomes based on existing data
- Build models of complex systems

WHO USES DATA SCIENCE?

It is widely used in business, science and engineering.

SOME NOTABLE EXAMPLES INCLUDE:

Marketing departments across the corporate world, ie: customer data and metrics.

Social media: user analytics for targeted advertisement.

News outlets: studying targeted audiences

Technology Companies: collect and analyze users' data

IMPACT OF DATA SCIENCE IN PEOPLE'S LIFE:

By using technology, we leave a so called digital footprint

In the age of significant computational and storage capabilities, computer systems have the ability to store and process huge amount of data.

This includes the data we generate as a result of of our activities.

In other words, it is possible to build systems which 'never forget'

In most cases, such systems benefit the customers, since the data allow companies to improve their products and services.

But in other cases, such data might affect people negatively.

A range of data-driven technologies have recently become ubiquitous.

They have the ability to store and process personal information.

SOME PROMINENT EXAMPLES INCLUDE:

- Internet of things
- Wearable devices
- Online services (banking, shopping, services provided by local and national governments)
- Various surveillance technologies.

The utilization of biometrics adds another level of concern, especially from a privacy perspective.

DATA SCIENCE AS A CAREER:

In many cases, data science professionals transition from other fields.

Those include computer science, information technologies, applied mathematics.

Recently, higher education institutions introduced degrees in data science.

SOME POPULAR DATA-DRIVEN CAREER PATHS INCLUDE:

Data Science consulting
Machine Learning Engineer
Big Data Engineer/Architect
Artificial Intelligence Architect

PROMINENT AREAS IN DATA SCIENCE:

Medicine, including drug discovery, medical imaging and diagnosis
Finance and various financial technologies
Social Media
Marketing
Robotics and automation

1.102 The scope of data science

Read Provost, F. and T. Fawcett *Data science for business: what you need to know about data mining and data-analytic thinking*. (Sebastopol, CA: O'Reilly Media, Inc., 2013) Chapter 1 Introduction: Data-Analytic Thinking.

1.103 Nature of data

DATUM NOUN [C] PLURAL: DATA

A journey from Latin to a modern-day buzzword.

Some use ‘data’ as a countable noun, but this is not yet adopted by most major dictionaries.

Can be defined as unit of information or abstraction of a real-world entity.

Used alongside, and sometime interchangeably, with other terms such as: variable, feature, attribute and especially ***information***.

Generally, in academic and scientific contexts the terms ***data*** and ***information*** have different meanings.

In most cases, entities subject to a data science study are described by a number of attributes [1] :

For example:

Entity: Book
- Attributes: Author, Title, Genre, Publisher, Year, Price

Entity: Vehicle
- Attributes: MPG, Cyl, Engine, HP, Weight, 0to60, Year, Origin.

[1] This is the case with structured data, as discussed later in this lecture.

DATA POINTS:

A data point can be considered as a single unit of observation.

The most immediate notion of a data point is a numerical value.

This can be integer, real or complex.

Or binary:

0/1 or True/False.

Computer systems are digital (binary) machines, work on any type of data (text, video, audio) is actually done done on digital (binary) representation of that data.

Programming languages and software systems add an abstraction layer, which allows users to process data in their original type and format.

This allows users to directly apply domain-specific functions on the data.

For example, in audio and image processing and editing.

DATASET

Most generally, a dataset is a collection of data points.

Usually two-dimensional, with rows representing entities (items) and columns representing attributes (features).

Entities, or items in most cases are part of a collection, such as books in a library or items in a shop.

Attributes, or features, describe the entities can be of different types.

ISSUES WITH DATA SETS:

Real world data is not always consistent

The most common issues include:
- missing attributes
- attributes of incorrect type of with incorrect values.

It requires pre-processing (cleaning).

In many cases, this is equivalent to transforming a table into first normal form in the context of a relational database.

STRUCTURE OF DATA

Datasets can be classified based on their structure:

Structured data
- Usually organized into tables
- Easy to search, process study

Unstructured data
- Cannot be organized into consistent structure
- Most real-world data are unstructured

DATA TYPES

The concept of data types is key in statistics and in computer science

The term has mostly overlapping meanings across both areas, although some nuances in the meanings exist.

DATA TYPES IN STATISTICS

Categorical: nominal, ordinal
Numerical: interval, ratio

DATA TYPES IN COMPUTER SCIENCE

Varying depending on the programming paradigm (concurrent, parallel, object oriented) and programming language, some languages share similar data types.

Specific language implementations, different interpreters, compilers and IDEs can offer some variations in data type range, precision and required memory.

General grouping of data:

Primitive (built in)
Composite (records, structures, classes)
Abstract (including data structures)

Ie: Boolean, integer, real (float), character, string, reference (pointer)

Datasets are stored in data structures before being processed:

ie: in Python: lists, dictionaries, arrays (NumPy), DataFrames (pandas), graphs (Networkx) etc.

1.104 Nature of data

Read EMC Education Services and EMC Education Services *Data science and big data analytics: discovering, analyzing, visualizing and presenting data*. (Indianapolis, IN: Wiley, 2015) Chapter 1 Introduction to Big Data Analytics and Chapter 2 Data Analytics Lifecycle.

Week 2

1.201 Python language and ecosystem

Python is an interpreted high-level general-purpose programming language. Python's design philosophy emphasizes code readability with its notable use of significant indentation.

Python is strongly & dynamically typed, designed to combine the computational capabilities of ANSI C and Shell Script. Python key concepts introduced: import, sum, max, lists, dictionaries.

Python library & associations:
- Pandas: data manipulation
- Matplotlib: plotting numerical data
- Scikit-learn: machine learning
- NumPy: n-dimensional array processing, data manipulation

1.203 Python tutorial

The following tutorial covers fundamentals of Python programming. In data visualisation, we will primarily work with higher level data science libraries, such as pandas. However, it is useful to have a basic grasp of Python, so if you are less confident with your Python programming skills, this is a useful resource to look over and keep at hand.

Python tutorial. Python Software Foundation.

1.204 Jupyter: Open platform for data science

Jupyter Notebook:

Open-source web-based application
Documents containing code, equations, visualizations and text
Code can be modified and executed on the fly
Goldsmith's Jupyter Notebook Server:
https://mscds.doc.gold.ac.uk/jupyter

JupyterLab is browser-based integrated development environment (IDE)

1.205 Jupyter Lab: Getting started

Click on the link below to read an overview of how to get started with JupyterLab:
- JupyterLab Getting started: Overview.

1.206 Jupyter user guide

Click the links to read the following Jupyter user guides:

Jupyter user guide: JupyterLab interface
Jupyter user guide: Text editor
Jupyter user guide: Notebooks
Jupyter user guide: Code consoles.

Week 3

2.001 Introduction to NumPy

In this lecture we'll discuss:

Data Types
Introducing the NumPy Library
Memory allocation
Initialize arrays
Random values
Array arithmetic

DATA TYPES

Data science deals with a wide range of data types. Examples include:

Boolean (logical) 0/1
Numerical
Textual
Images

Since computers are digital/binary machines, storing and processing any type of data is actually done on their corresponding representations.

There are a number of different binary representations.

Popular ones are:
- One's and two's complement
- IEEE standardized floating point number

Most programming languages, Python included support the following numerical data types:
- integer (positive/negative whole numbers)
- real (floating point, mantissa (significant digit and exponents))
- complex (two components, real and imaginary)

The NumPy library

NumPy is short for Numerical Python. It provides numerical functionality to process arrays (and primary data types). It is very popular for data science related tasks. It has a range of mathematical functions. It is used extensively along other libraries such as SciPy and pandas.

Note: primitive type variables and arrays correspond to scalars and vectors, as studied in linear algebra, a branch of mathematics with significant applications in data science.

THE MAIN FUNCTIONALITIES OF THE NUMPY LIBRARY:

Allocate memory for arrays (NumPy arrays) - Arrays may have various dimensions

Initialize arrays with arbitrary values, for example: zeroes or other constants, random values, external sources such as CVS, spreadsheets.

Extract statistical information from the data (aggregate)
Perform some linear algebra tasks.

MEMORY ALLOCATION:

ONE DIMENSIONAL ARRAY OR INTEGERS USING THE 'EMPTY' METHOD:

```
import numpy as np
# Number of elements
NumberOfElements = 50
# Allocate memory
MyArray = np.empty(shape=NumberOfElements, dtype=int)
```

This method does not initialize the elements of the array. The elements have random values found on the memory locations at the time of allocation.

TWO DIMENSIONAL ARRAY OR INTEGERS USING THE 'EMPTY' METHOD:

```
import numpy as np
# Define dimensions
NumberOfRows = int(10)
NumberOfColumns = int(50)
# Allocate memory
MyArray = np.empty(shape=(NumberOfRows, NumberOfColumns), dtype=int)
```

- The result is an array or 10×50 un-initialised elements.

INITIALISE ARRAYS

```
import numpy as np
# Initialise the random number generator
np.random.seed(0)
# Initialise array with 10 elements with random values between -5 and
5
MyArray = np.random.randint(-5, 5, 10, dtype=int)
```

- Array with 10 elements with values from the interval [-5 … 5]

One dimensional array or integers using a random number generator, we can specify the number of elements, in their range and type:

- Initialise all elements with a specific value and print the array
- Zeros:

Input:

```
# Number of elements
NumberOfElements = 10
# Allocate memory and intialise the elements with zeros
MyArray = np.zeros(shape=NumberOfElements, dtype=int)
# Print the array
print(MyArray)
```

Output:

```
[0 0 0 0 0 0 0 0 0 0]
```

- Initialise all elements with a specific value and print the array
- Ones:

Input:

```
# Number of elements
NumberOfElements = 10
# Allocate memory and intialise the elements with ones
MyArray = np.ones(shape=NumberOfElements, dtype=int)
# Print the array
print(MyArray)
```

Output:

```
[1 1 1 1 1 1 1 1 1 1]
```

- Initialise all elements with a specific value and print the array
- Arbitrary value:

Input:

```
# Number of elements
NumberOfElements = 10
# Allocate memory and intialise the elements with 7
MyArray = np.ones(shape=NumberOfElements, fill_value=7, dtype=int)
# Print the array
print(MyArray)
```

Output:

```
[7 7 7 7 7 7 7 7 7 7]
```

Random value

- Initialise array elements with random integer numbers

Input:

```
# Number of elements
NumberOfElements = 10
# Allocate memory and intialise the elements with random values
MyArray = np.random.randint(0, 100, NumberOfElements)
# Print the array
print(MyArray)
```

Output:

```
[82 79 45 57 3 43 98 23 14 15]
```

Random values

- Alternative approach
- Assigning values to each element individually

Note:

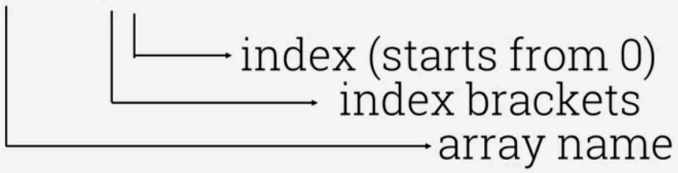
This approach gives better flexibility by individually assigning the values of the elements of the array.

```
# Number of elements
NumberOfElements = 10
# Initialise the random number generator
np.random.seed(0)
# Allocate memory the array
MyArray = np.empty(shape=NumberOfElements, dtype=int)
# Initialise and print the array
for i in range(0, 10):
  MyArray[i] = np.random.randint(-5, 5)
  print(i, ':', MyArray[i])
```

Random values

- Accessing individual elements

  MyArray [i]

  - Two-dimensional array
- Python allows two notations:
  - MyArray[i, j]
  - MyArray[i][j]

Note:

NumPy arrays find their origin in C/C++, as does Python itself.

16

NUMPY ARITHMETIC

Common tasks include: finding the minimum, maximum, sum, average

NEEDED WHEN:

calculating the range of the data
normalizing the data
summarizing and comparing datasets

NumPy arithmetic: finding the minimum and maximum

- Standard approach ('*the C way*')

```
# Assume the first element is the smallest
Minimum = MyArray[0]
# Compare with all other elements and update Minimum if needed
for i in range(1, 10):
    if Minimum > MyArray[i]:
        Minimum = MyArray[i]
# Display the result
print('Minimum:', Minimum)
```

Note:

In the age of Python there is a shorter approach.

20

NumPy arithmetic: finding the minimum and maximum

Input:

```
# Allocate memory the array and intialise the elements with random values
MyArray = np.random.randint(0, 100, NumberOfElements)
# Print the array
print(MyArray)
# Display the minimum and the maximum values
print('Minimum:', MyArray.min())
print('Maximum:', MyArray.max())
```

Output:

```
[10 2 59 7 16 46 89 23 12 94]
Minimum: 2
Maximum: 94
```

Array arithmetic

- Finding the sum per column and per row
- Parameter 'axis' specifies the 'direction' of the summation

```
# Summation by columns
SumByColumn = np.sum(MyArray, axis=0)
print('By column:', SumByColumn)
# Summation by rows
SumByRow = np.sum(MyArray, axis=1)
print('By row:', SumByRow)
```

```
0  9  9  2  1  6
7  6  3  2  2  3
3  2  7  9  2  4
1  8  6  5  8  8
8  6  8  8  3  2
By column: [19 31 33 26 16 23]
By row: [27 23 27 36 35]
```

- Per-column and per-row operations not limited to 'sum'
- Available for minimum, maximum and average

2.002 NumPy

Now read:
1. NumPy: Data types
2. NumPy: Array creation

2.101 Advanced data processing with NumPy

In this lecture we'll cover:

- Data preprocessing
- Advanced indexing
- Structured data
- Statistics with NumPy:
  - Measures of central tendency
  - Measures of spread

DATA PRE-PROCESSING

In many cases, acquired data lack the structure, consistency and format necessary for computer algorithms to perform their tasks successfully.

Data science-related tasks are no exceptions.

For example, machine learning, data visualization, computer vision and other data science related fields require the datasets to be consistent and well-structured.

Some of the most common issues with data are:

- Missing values
- Type inconsistency or type mismatch
- Duplicating values, including entire rows and columns

Resolving such problems is the first step in many data science tasks.

This step is known as 'pre-processing' - it corresponds to bringing a database to:
first normal form (1NF)

A common pre-processing task is to extract useful data from an existing dataset.

From a programming perspective, this amounts to generating a sub-array based on an original array.

This can be done with NumPy array by using advanced indexing.

ADVANCED INDEXING

Advanced indexing allows us to build new arrays based on existing arrays.

We can do this by specifying a set of elements which will be included in the new array.

This can be applied to arrays with different dimension.

- Extract a sub-array by specifying start and end index of the elements:

```
# Initialise array A with 10 values 0 to 9
A = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Extract sub-array B
B = A[2:5]
```

End index (exclusive)

Start index (inclusive)

Result:

[2 3 4]

- The resulting array does not need to be comprised of all elements between the start and end index. We can set a step (stride):

Input:

```
# Initialise array A with 10 values 0 to 9
A = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
# Extract sub-array B
B = A[2:8:2]
# Display the new array
print(B)
```

Output:

[2 4 6]

- By using the step, we can display all elements with even and odd indices:

Input:

```
A = np.array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
B = A[::2]
C = A[1::2]
```

Output:

[2 4 6 8]

[1 3 5 7]

- The same principles apply to two-dimensional arrays:

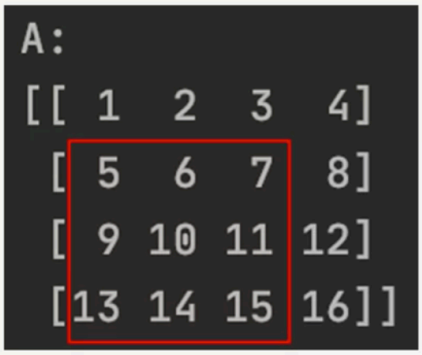
```
# Initialise 2D array of 3x3 elements
A = np.array([[1, 2, 3], [4, 5, 6], [7, 8, 9]])
# Display the array
print('A:')
print(A)
# Extract and display sub-arrays
B = A[2:3, 1:3]
C = A[1:3, 2:3]
print('B:')
print(B)
print('C:')
print(C)
```

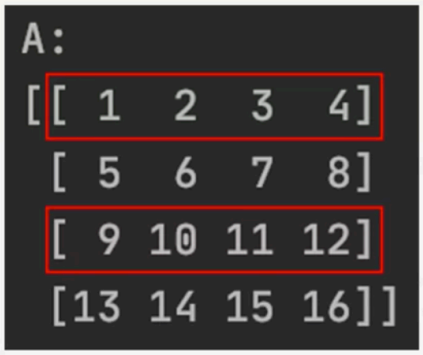
```
A:
[[1 2 3]
 [4 5 6]
 [7 8 9]]
B:
[[8 9]]
C:
[[6]
 [9]]
```

11

Advanced indexing: task

- Task: Write a Python script which initialises the arrays from the image and extracts the elements within the red rectangles.





Code solution on the next page.

In [1]:
```
import numpy as np
```

In [9]:
```
A = np.array([[1, 2, 3, 4], [5, 6, 7, 8], [9, 10, 11, 12], [13, 14, 15, 16]])


print('A:')
print(A)

B = A[1:4, 0:3]
print('B:')
print(B)

C = A[::2]
print('C:')
print(C)
```

```
A:
[[ 1  2  3  4]
 [ 5  6  7  8]
 [ 9 10 11 12]
 [13 14 15 16]]
B:
[[ 5  6  7]
 [ 9 10 11]
 [13 14 15]]
C:
[[ 1  2  3  4]
 [ 9 10 11 12]]
```

In most cases, data science deals with data organized in the form of a table, similarly to XML and spreadsheets.

It is not uncommon for these tables to contain rows and columns which prevent further processing.

This may be due to various reasons, including missing values, type mismatch and others.

Removing or ‘dropping’ problematic rows and columns is the first and in some cases the only step of the preprocessing task.

This can be achieved with other libraries such as pandas, which support alternative approaches with similar results.

Structured Data

Datasets can be split into two groups, based on the type of their data points:
‘

1. Datasets containing data points of the same type: this is usually numerical, either integer or real
2. Datasets containing data points of multiple data types: numerical, textual, boolean

NumPy library provides functionality to store and process arrays of **structured data**; that is, data consisting of multiple data types.

The structure needs to be consistent throughout the datasets.

NumPy array constructed in this way correspond to tables from a relational database.

Let's consider the following example: Using NumPy arrays organism the information about a number of books with the following information for each item: Title, Year, Price.

- The first step is to define the array and allocate the necessary memory[1]:

```
A = np.empty(shape=10, dtype=[('Title', 'U10'), ('Year', int), ('Price', float)])
```

- Structured data is usually stored as CSV, XML or a spreadsheet table.
- Other libraries, such as pandas, handle this task differently.

STATISTICS WITH NUMPY

Statistics is a major branch of mathematics. It studies the properties of numerical data; more specifically, data points organized in datasets.

Statistics can generalize the properties of datasets and draw conclusions based on the data.

This topic focuses on the statistical functions of NumPy.

Statistic with NumPy: measures of central tendency

Measures of central tendency evaluates central value, around which the data cluster around.

There are 3 popular measures of central tendency:

Mean (also known as the average)
Median (splits the dataset into two halves)
Mode (the data point which appears most frequently)

- The following script initialises a dataset with 20 data points and sort it in ascending order:

Input:

```
# Initialise dataset S
S = np.array([3, 8, 6, 3, 8, 5, 1, 5, 3, 6, 4, 5, 2, 4, 1, 5, 8, 4, 7, 5])
# Sort the array
NewS = np.sort(S)
# Display te new array
print(NewS)
```

Output:

```
[1 1 2 3 3 3 4 4 4 5 5 5 5 5 6 6 7 8 8 8]
```

- By observing the dataset, we can notice that 5 is the most common data points and it is also the mid-point of the dataset.

- The conclusions from previous slide were possibly due to the small size of the dataset (20 data points).
- In practice, this is rarely the case.
- The only feasible way to extract this information is by using the available functionality, as demonstrated in the following example:

```
Mean = np.mean(S)
Median = np.median(S)
Mode = st.mode(S)
print('Mean:', Mean)
print('Median:', Median)
print('Mode:', int(Mode[0]))
```

```
Mean: 4.65
Median: 5.0
Mode: 5
```

Note: sometimes we'd need to import spicy for more advanced functions:

"from scipy import stats as st"

Statistics of NumPy: measures of spread

Measures of spread evaluates how data are spread around certain central values.

Three common spreads are:

range, standard deviation, variance.

Since variance = sd^2 (we only need to calculate sd)

- Range is the difference between the maximum and the minimum value from the dataset:

```
print('Range:', np.max(S)-np.min(S))
```

- Standard deviation measures the average distance of the data points from the mean value:

```
print('Standard deviation', np.std(S))
```

- The result are the following values:
  - **Range: 7**
  - **Standard deviation: 2.080264406271472**

28

```
# Dimensions or the dataset
NumberOfRows = int(5)
NumberOfColumns = int(6)

# Allocate memory
MyArray = np.empty(shape=(NumberOfRows, NumberOfColumns), dtype=int)

# Initialise a dataset with random values
# Display the dataset as a table
for i in range(0, NumberOfRows):
    for j in range(0, NumberOfColumns):
        MyArray[i, j] = np.random.randint(0, 10)
        print(MyArray[i, j], ' ', end='')
    print(' ')
```

- Considering each column as an independent data series, we can extract statistical information per column:

```
# By columns mean, median, mode and standard deviation
MeanByColumn = np.mean(MyArray, axis=0)
MedianByColumn = np.median(MyArray, axis=0)
ModeByColumn = st.mode(MyArray, axis=0)
SD = np.std(MyArray, axis=0)

# Display the result
print('Mean by column:', MeanByColumn)
print('Median by column:', MedianByColumn)
print('Mode by column:', ModeByColumn)
print('Standard deviation by column:', SD)
```

- Output:

```
3 3 7 8 9 2
7 3 5 4 0 6
2 4 8 5 3 5
5 3 0 2 9 1
6 9 1 9 1 7

Mean by column: [4.6 4.4 4.2 5.6 4.4 4.2]

Median by column: [5. 3. 5. 5. 3. 5.]

Mode by column: ModeResult(mode=array([[2, 3, 0, 2, 9, 1]]), count=array([[1, 3, 1, 1, 2, 1]]))

Standard deviation by column: [1.8547237  2.33238076 3.18747549 2.57681975 3.87814389
2.31516738]
```

2.102 Advanced data processing with NumPy

Now read:
1. NumPy, Array Objects: Indexing
2. NumPy, Routines: Statistics

2.104 Linear algebra in practice

Introduction to linear algebra:

- Scalars, vectors and matrices
- Determinants, ranks and traces
- Matrix multiplication
- Inverse matrices
- Systems of linear equations

Linear algebra is a major branch of mathematics.

It has significant theoretical background and extensive applications in computer science and data science.

The lecture briefly introduces its main concepts. It demonstrates how to use the NumPy library to perform some key linear algebra tasks.

Some of the key concepts in linear algebra are (among others):

- linear combinations
- systems of linear equations
- linear transformations (also known as linear maps)

This lecture will focus on scalars, vectors and matrices and how to use them to solve linear equations.

SCALARS

In the mathematical sense, scalars are values which have only magnitude.

For example: the real numbers or any of their subsets, such as:
Natural, integer, rational

Complex numbers are usually treated as scalars, although some in context they can be considered two dimensional vectors.

Vectors

- In 2D and 3D spaces, a vector is a quantity represented by an arrow with both direction and magnitude (see Figure 1).
- In linear algebra, the concept of a vector is generalised for any dimension (n):
  - $v = (a_1, a_2, \dots a_n)$
- The algebraic notion of a vector corresponds to the **array data type**, as supported by a number of programming languages.

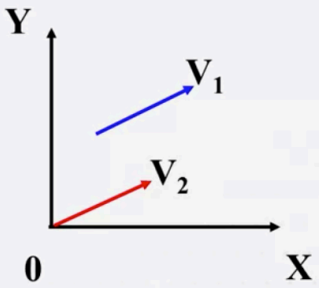


Figure 1: 2D vectors

Vectors: main operations - 1

1. **Scalar multiplication**:
   - Scalar: a
   - Vector: $v = (v_1, v_2, \dots , v_n)$
   - $av = a(v_1, v_2, \dots , v_n) = (av_1, av_2, \dots , av_n)$
2. **Vector addition**[1]:
   - Vectors: x, y, z (same dimension)
   - $x = (x_1, x_2, \dots , x_n)$, $y = (y_1, y_2, \dots , y_n)$
   - $z = x + y = (x_1+y_1, x_2+y_2, \dots , x_n+y_n)$

3. **Dot product:**
   - Vectors: x, y, z (same dimension)
   - $x = (x_1, x_2, \dots , x_n)$, $y = (y_1, y_2, \dots , y_n)$
   - $z = x \cdot y = x_1y_1 + x_2y_2 + x_ny_n$
   - The result of the dot product is a single value, a scalar

MATRICES

By definition a matrix is a rectangular table of components (scalars or numbers)

The scalars are arranged in rows and columns by using a two-index notation.

The following is a generalized matrix M with M x N components.

Similarly to vectors, matrices can be defined as two dimensional NumPy arrays.

MATRICES AND NUMPY: DETERMINANTS

Every square matrix has a determinant. The determinant is a single value, which has important mathematical meanings.

One of the key questions is whether the determinant has zero or non-zero value.

Calculating a determinant requires multiple additions and multiplications and in practice it is usually done by using a software tool.

Matrices and NumPy: matrix multiplication

Matrix multiplication is key operation in linear algebra. The result of the multiplication is also a matrix. The multiplication follows specific rules:

It is based on the dot product discussed on the previous slide.

It requires the number of columns from the first matrix (j) to be equal to the number of rows in the matrix(k):

- Dot product:
  - **Input:**

```
# Define three-dimensional vectors X and Y
X = np.array([1, 2, 3], dtype=int)
Y = np.array([5, 10, 15], dtype=int)
# Calculate the dot product D
D = np.dot(X, Y)
# Display dot product D on the screen
print('D:', D)
```

  - **Output:**

```
D: 70
```

Note:

Unlike vector addition, the result of the dot product is a single value.

- Scalar multiplication
  - **Input:**

```
# Define a scalar C
C = int(5)
# Define three-dimensional vector A
A = np.array([1, 2, 3], dtype=int)
# Multiply the scalar and the vector
B = C*A
```

Note:

There is no need to declare vector B before the

$$M = \begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix}$$

- Vector addition:
  - **Input:**

```
# Define three-dimensional vectors X and Y
X = np.array([3, 4, 5], dtype=int)
Y = np.array([5, 6, 7], dtype=int)
# Multiply the vectors
Z = np.add(X, Y)
# Display vector Z on the screen
print(Z)
```

  - **Output:**

```
[8, 10, 12]
```

Note:

There is no need to declare vector Z before the assignment.

- The following code initialises a 3 by 3 matrix (array) and calculates its determinant.

Input:

```
# Initialise a matrix M with 3 by 3 elements
M = np.array([[1, 2, 3], [2, 3, 4], [2, 5, 6]])
# Calculate the determinant of M
D = np.linalg.det(M)
# Display the matrix
print(M)
# Display the determinant
print('D:', D)
```

```
[[1 2 3]
 [2 3 4]
 [2 5 6]]
D: 2.0
```

- If one of the row (or column) is a **linear combination** of another row (or column), the value of the determinant is 0.

Input:

```
# Initialise a matrix M with 3 by 3 elements
M = np.array([[1, 2, 3], [2, 4, 6], [2, 5, 6]])
# Calculate the determinant of M
D = np.linalg.det(M)
# Display the matrix
print(M)
# Display the determinant
print('D:', D)
```

```
[[1 2 3]
 [2 4 6]
 [2 5 6]]
D: 0.0
```

- A square matrix[1] with non-zero determinant has **rank** equals to its dimensions.

Input:

```
# Initialise a matrix M with 3x3 elements
M = np.array([[1, 2, 3], [2, 3, 4], [2, 5, 6]])
# Calculate the rank of M
R = np.linalg.matrix_rank(M)
# Display the matrix
print(M)
# Display the rank
print('R:', R)
```

```
[[1 2 3]
 [2 3 4]
 [2 5 6]]
R: 3
```

- We can lower the rank of a matrix by defining a row as a linear combination of another row, as illustrated in the example here.

Input:

```
# Initialise a matrix M with 3x3 elements
M = np.array([[1, 2, 3], [2, 3, 4], [4, 6, 8]])
# Calculate the rank of M
R = np.linalg.matrix_rank(M)
# Display the matrix
print(M)
# Display the rank
print('R:', R)
```

```
[[1 2 3]
 [2 3 4]
 [4 6 8]]
R: 2
```

- The trace of a matrix is the sum of all elements on the main diagonal. The following example illustrates the calculation of trace.

Input:

```
# Initialise a matrix M with 3x3 elements
M = np.array([[1, 2, 3], [2, 3, 4], [4, 6, 8]])
# Calculate the trace of M
T = np.trace(M)
# Display the matrix
print(M)
# Display the trace
print('T:', T)
```

```
[[1 2 3]
 [2 3 4]
 [4 6 8]]
T: 12
```

- Multiplying matrices A and B

$$\begin{pmatrix} a_{11} & \cdots & a_{1n} \\ \vdots & \ddots & \vdots \\ a_{m1} & \cdots & a_{mn} \end{pmatrix} \times \begin{pmatrix} b_{11} & \cdots & b_{1k} \\ \vdots & \ddots & \vdots \\ b_{n1} & \cdots & b_{nk} \end{pmatrix} = \begin{pmatrix} c_{11} & \cdots & c_{1k} \\ \vdots & \ddots & \vdots \\ c_m 1 & \cdots & c_{mk} \end{pmatrix}$$

- $c_{11} = a_{11} \times b_{11} + a_{12} \times b_{21} + \ldots + a_{1n} \times b_{n1}$
- The NumPy library provides a matrix multiplication method.

- A square matrix with non-zero determinant has an inverse.
- In the case of real numbers, where the multiplicative inverse of r is 1/r (for example, the multiplicative inverse of 2 is 0.5)
- The inverse of a matrix M is usually denoted with M^{-1}
- $M \times M^{-1} = M^{-1} \times M = I$
- Where I is the identity matrix
- Multiplying any square matrix with I does not change the matrix

$$I = \begin{pmatrix} 1 & \cdots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \cdots & 1 \end{pmatrix}$$

- One of the most central topics in linear algebra are **systems of linear equations**[1] and the approaches to solving them.
- They have significant applications in computer science, engineering, finance technology and many other fields.
- The following is a general form of a system of linear equations:

$$\begin{cases} a_{11}x_1 + \cdots + a_{1n}x_n = c_1 \\ \cdots \\ a_{m1}x_1 + \cdots + a_{mn}x_n = c_m \end{cases}$$

- Let's consider this example: $\begin{cases} x + 2x + 3z = 1 \\ 2x + 3y + 4z = 4 \\ 2x + 5x + 6x = 7 \end{cases}$
- We can express the system of linear equations by using matrices:

$$\begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 2 & 5 & 6 \end{pmatrix} \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}$$

- And by using the following matrix notation:

$$MX = C, where: M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 2 & 5 & 6 \end{pmatrix}, X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, C = \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}$$

- Let's consider the last equation:

$$MX = C, where: M = \begin{pmatrix} 1 & 2 & 3 \\ 2 & 3 & 4 \\ 2 & 5 & 6 \end{pmatrix}, X = \begin{pmatrix} x \\ y \\ z \end{pmatrix}, C = \begin{pmatrix} 1 \\ 4 \\ 7 \end{pmatrix}$$

- And perform the following steps:

$$MX = C$$

$$M^{-1}MX = M^{-1}C$$

$$IX = M^{-1}C$$

$$X = M^{-1}C$$

- This code performs the calculations from the previous slide:

Input:

```
# Initialise a matrix M with 3x3 elements
M = np.array([[1, 2, 3], [2, 3, 4], [2, 5, 6]])
# Calculate the inverse of M
M1 = np.linalg.inv(M)
# Initialise matrix C
C = np.array([[1], [4], [7]])
# Calculate X = M1*C
X = np.matmul(M1, C)
# Display matrix X
print('X:', X)
```

```
X: [[ 1.5]
 [ 5. ]
 [-3.5]]
```


- Let's consider the result from the previous slide

$$X = \begin{pmatrix} x \\ y \\ z \end{pmatrix} = \begin{pmatrix} 1.5 \\ 5 \\ -3.5 \end{pmatrix}, x = 1.5, y = 5, z = -3.5$$

- We can substitute the values into the system of linear equation

$$\begin{cases} x + 2x + 3z = 1 \\ 2x + 3y + 4z = 4 \\ 2x + 5x + 6x = 7 \end{cases}$$

```
X: [[ 1.5]
 [ 5. ]
 [-3.5]]
```

- And confirm the result:

$$\begin{cases} 1.5 + 2 \times 5 - 3 \times 3.5 = 1 \\ 2 \times 1.5 + 3 \times 5 - 4 \times 3.5 = 4 \\ 2 \times 1.5 + 5 \times 5 - 6 \times 3.5 = 7 \end{cases}$$

2.105 Linear algebra in practice

Now read:

- Berberian, S.K. *Linear algebra*. (Mineola, NY: Dover Publications, 2014) Chapter 4: Matrices and Chapter 6: Determinants.

This book is in the Online Library in the ProQuest Collection. Find out how to search for the book.

- NumPy, Linear algebra (numpy.linalg)

2.107 Python data science handbook, Chapter 2

Now read VanderPlas, J. *Python data science handbook: essential tools for working with data*. (Sebastopol, CA: O'Reilly Media, Inc., 2016) Chapter 2: Introduction to NumPy.

Week 4

Week 5

Week 6

Week 7

Week 8

Week 9

Week 10

Week 11

Week 12

Week 13

Week 14

Week 15

Week 16

Week 17

Week 18

Week 19

Week 20

Week 21

Week 22

Learning Objectives

- describe the concepts taught in this course
- apply the concepts taught in this course to solve problems
- analyse algorithms and data structure in terms of the concepts taught in this course

EXAM: September 6, 2021