

01-Lecture-Slides.pdf	2
02-Lecture-Slides.pdf	24
03-Lecture-Slides.pdf	51
04-Lecture-Slides.pdf	81
05-Lecture-Slides	112
06-Lecture-Slides	136
07-Lecture-Slides.pdf	166
08-Lecture-Slides.pdf	200
09-Lecture-Slides.pdf	226
10-Lecture-Slides.pdf	263

Applications of Big Data

Lecture 1 — Introduction to Big Data

Nick Tothill

2019 July 22

Unit Introduction

Basic information

- ▶ 301110 Big Data (UG)
 - ▶ Level 3 Undergraduate Unit
 - ▶ 10 Credit Points
 - ▶ Courses: BDS
- ▶ Assumed Knowledge
 - ▶ No prerequisite unit
 - ▶ Basic programming skills required
 - ▶ We will use Python in this unit
 - ▶ Prior experience with any programming language is useful
 - ▶ Numeric skills
 - ▶ Working knowledge of elementary probability and statistics

Teaching staff

Unit Coordinator: Nick Tothill

Phone: x2956

Email: n.tothill@westernsydney.edu.au (most reliable contact method)

Office: Y.2.09, Kingswood campus (Penrith)

Consultation hours: 1300-1400 every Monday, ER.1.06

What will you learn?

- ▶ Principles and technologies of retrieving, processing and managing massive real-world data sets
- ▶ The basic techniques required by any discipline for making sense out of the growing amount of data
 - ▶ Big Data is not an isolated field
 - ▶ Its principles and techniques can be applied to many different areas in computing, social science, business, science etc.
- ▶ Knowledge and key set of skills to be competitive in the growing job market
 - ▶ Provides a good starting point but not everything

How does the unit run?

- ▶ 4 Hours per week
 - ▶ two hours lecture (not compulsory but strongly recommended)
 - ▶ Two hour practical (mandatory)
- ▶ About 5–8 hours self-study
- ▶ Lecture sessions
 - ▶ Scheduled for 13 weeks
 - ▶ Special for week 14, the final exam
- ▶ Practical sessions
 - ▶ Scheduled for 12 weeks
 - ▶ Special for week 14 (Final Exam)
- ▶ Check Student Learning Guide for further details

Assessment

- ▶ Quizzes: 30%
 - ▶ 5 quizzes altogether
 - ▶ All should be attempted
 - ▶ At least one *must* be attempted
- ▶ Assignment: 30% (Programming and Analysis)
 - ▶ Assignment questions will be released at least 3 weeks before deadline
 - ▶ This *must* be attempted
- ▶ Final Examination: 40%, 90 mins (Open Book)
 - ▶ Exam during week 14 practical class
 - ▶ This *must* be attempted

No threshold requirement. You must obtain at least 50% in total to pass the unit

Textbook and references

- ▶ Textbook:
 - ▶ No prescribed textbook
 - ▶ Big Data is still an emerging field
 - ▶ Too many topics to be included in a single book
- ▶ Main reference for Python and data analysis

*Wes McKinney, “Python for Data Analysis”, ISBN 1449319793,
O'Reilly Media, 2012.*

- ▶ Other references
 - ▶ Heaps of reference materials covering different aspects of the subject: books, technical papers, web links

See learning guide for details

Syllabus

- ▶ Fundamentals of Big Data (Week 1)
- ▶ Python Basics (Weeks 2 – 4)
 - ▶ Data Types, Operations, Control Flows
 - ▶ Advanced Data Types, Functions, File I/O
- ▶ Databases
 - ▶ Relational Databases and SQL (Week 5)
 - ▶ NoSQL Databases (Week 6)
- ▶ Data Format (Week 7)
 - ▶ CSV, XLS, XML, JSON

Syllabus

- ▶ Data Acquisition and Wrangling (Week 8)
 - ▶ Web APIs
 - ▶ Python Pandas Library
- ▶ MapReduce (Weeks 10–11)
 - ▶ Fundamentals of Data Parallelism
 - ▶ Examples of MapReduce/Hadoop
- ▶ Predictive Analytics (Week 13)
 - ▶ Machine Learning, Classification

Note: the syllabus may subject to further modification based on student progress

Introduction to Big Data

What is Big Data?



Top recurring themes in people's definitions (wordle)

Three Vs of Big Data

- ▶ Volume
 - ▶ Size of data
 - ▶ Gigabyte -> Terabyte -> Petabyte -> Exabyte -> Zettabyte
-> Yottabyte
- ▶ Velocity
 - ▶ Speed of data processing relative to size and growing demand for interaction
 - ▶ Latency: cache > memory >> SSD > HDD > LAN > WAN
>> other
- ▶ Variety
 - ▶ Diversity of data sources, formats, types

The Beer and Nappies Story



Source: <http://asillywhim.blogspot.com.au/2013/09/huggies-chuggies.html>

Recommendation System

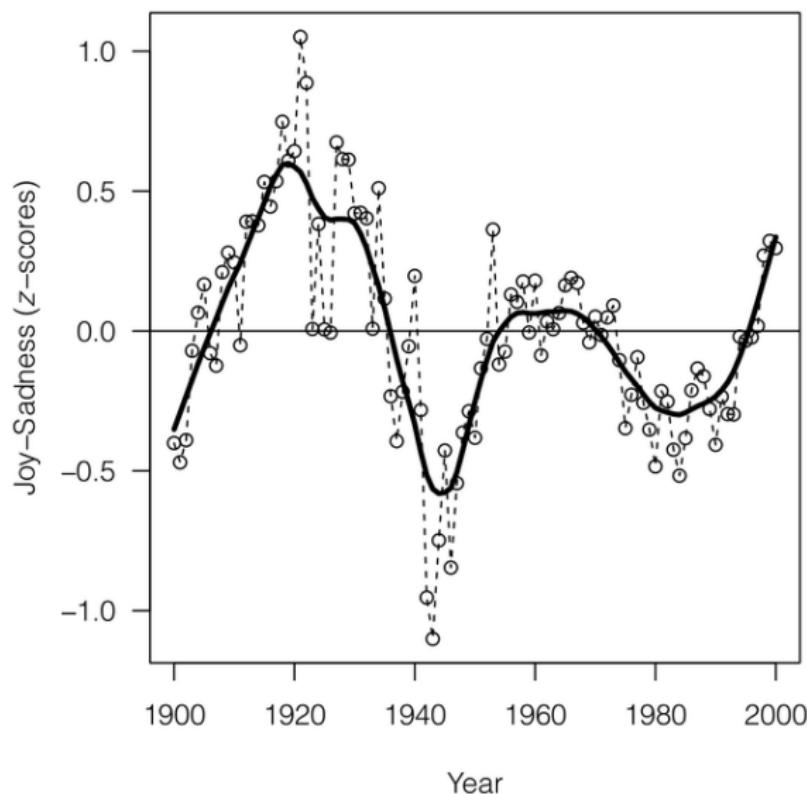
- ▶ “Beer and nappies” is a specific example of a recommendation system
- ▶ Recommendation System: an information system aiming to predict the ratings user give to items
 - ▶ Collaborative filtering: based on user preference data
 - ▶ More efficient and effective; most widely used
 - ▶ Can help identify similar items and similar users
 - ▶ Content based: based on content analysis
 - ▶ Harder to implement, need good prediction techniques
- ▶ Examples
 - ▶ Music recommendation: Last.fm, Pandora Radio
 - ▶ Movie recommendation: Netflix, Rotten Tomatoes
 - ▶ Product recommendation: Amazon, Ebay

Text Mining for Mood Analysis

Acerbi A, Lampis V, Garnett P, Bentley RA (2013) The Expression of Emotions in 20th Century Books. PLoS ONE 8(3): e59030. doi:10.1371/journal.pone.0059030

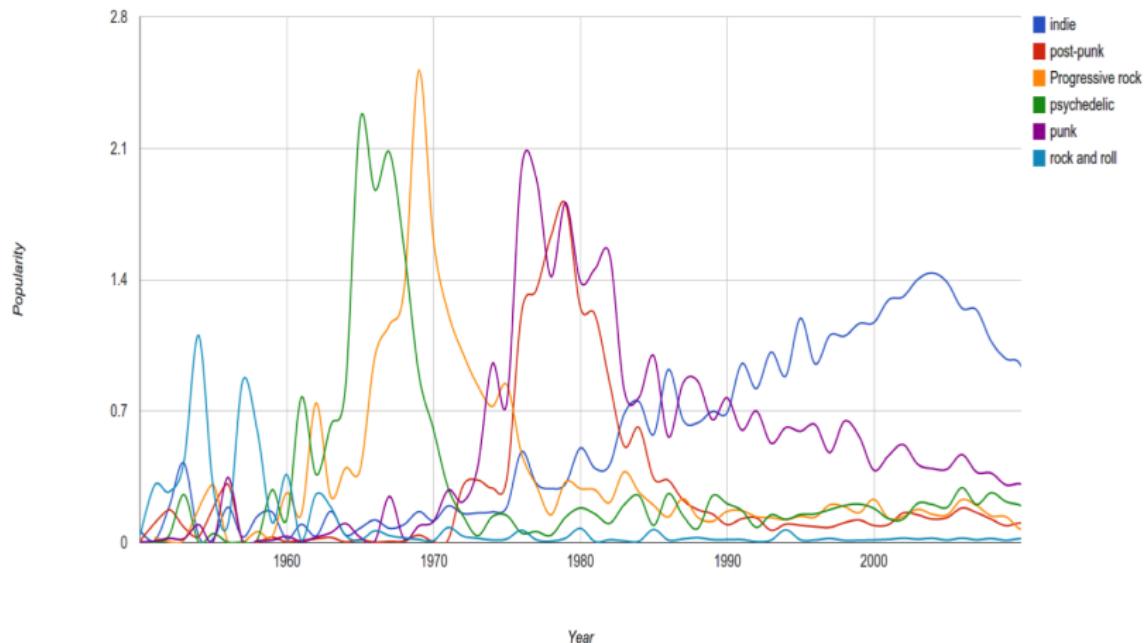
1. Convert all digitised books in 20th century into collection of words (<http://books.google.com/ngrams>)
2. Label each word with a mood score. E.g.: happy ≤ 0.7 , sad ≤ -0.7 , good ≤ 0.8 , terrible ≤ -0.9 , ... (<http://wordnet.princeton.edu/>)
3. Calculate the book score by accumulating the mood scores for all words occurring in the book.
4. Add the scores for all books published in the same year.

Text Mining for Mood Analysis



What are the strengths and weaknesses of this approach?

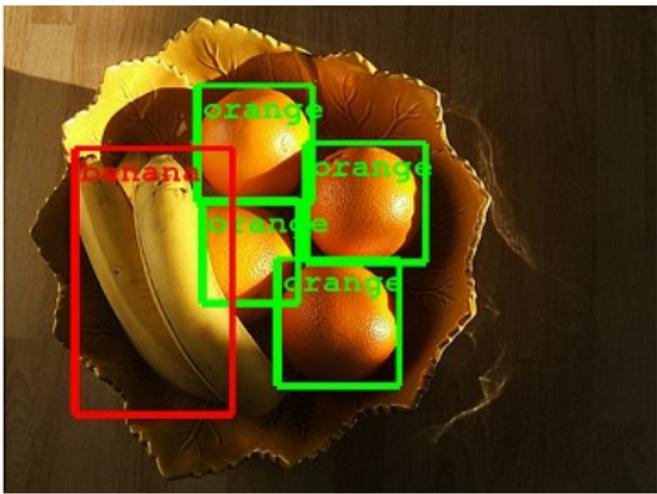
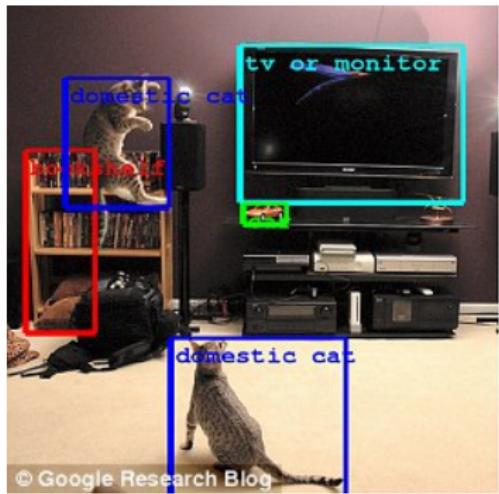
Popularity of Music Genres



<https://musicmachinery.com/tag/lastfm/>

Large Scale Visual Recognition

Given sufficient amount of images and object tags, computers can be trained to automatically detect different objects in the image



ImageNet LSVRC: <http://www.image-net.org/challenges/LSVRC/>

Data Science

- ▶ The science of dealing with big data
 - ▶ Statistics with programming?
 - ▶ Programming with statistics?
- ▶ Using big data to do science (widely defined)
 - ▶ Traditional science: theory + experiments -> data or observations -> deductions
 - ▶ Data science: extract knowledge from data carry out 'experiments' on data
- ▶ What makes data science useful?
 - ▶ ↓ Cost of getting data: cheap equipment, crowd sourcing
 - ▶ ↑ Cost of analysing data: the new bottleneck
- ▶ Data scientist
 - ▶ Well-known: Emerging trend in business analysis
 - ▶ Less-known: Use in academia, government...
 - ▶ 'Sexiest Job Of The 21st Century' according to HBR(!)

Skills for a Data Scientist

- ▶ Analytic Skills
 - ▶ Understand the problem including the requirements of customers/stakeholders
 - ▶ Develop solution for the problem to be solved
- ▶ Communication Skills
 - ▶ Need to talk to customers, colleagues...
- ▶ Computing Skills
 - ▶ Programming
 - ▶ Algorithms
 - ▶ Databases
 - ▶ Cloud computing
- ▶ Numerical Skills
 - ▶ Statistics and Machine Learning
 - ▶ Numerical Optimisation

Coming up next

- ▶ Basic Python Programming
- ▶ Relational Databases and SQL
- ▶ NoSQL Databases

Applications of Big Data

Lecture 2 — Python Basics I

Nick Tothill

2019 July 29

Introduction to Python Programming

Why Python Programming

- ▶ Why Programming?
 - ▶ Programming is a tool to realise your data analysis ideas
 - ▶ Data Science relies on programming heavily (why?)
- ▶ Why Python Programming?
 - ▶ Interpreted Programming Language
 - ▶ Can run interactively (natively interactive including terminal and IPython)
 - ▶ Other fancy stuff: Jupyter Notebook, python markdown, ...
 - ▶ Easy and flexible syntax
 - ▶ *Powerful* third-party package support
 - ▶ Convenient interface with other languages such as C/C++

The classic Hello, World! program

Python version

```
print "Hello, world!"
```

C++ version

```
#include <iostream>
int main(){
    std::cout<<"Hello, World!"<<std::endl;
    return 0;
}
```

Java version

```
public class HelloWorldApp {
    public static void main(String [] args)
    {
        System.out.println("Hello, World!");
    }
}
```

Basics of Python Programming

Output and input

The print command

print command is used in Python to display messages

- ▶ `print "Hello, World"` ✓
- ▶ `print 'Hello, World'` ✓
- ▶ `print Hello, World` ✗ Must enclose message with quotation marks
- ▶ `print "Hello, World'` ✗ Quotation marks do not match
- ▶ `Print "Hello, World"` ✗ print should be lower case

Display on Multiple Lines

```
print "Hello, World"  
print "I love programming"  
print "I love Python"
```

```
## Hello, World  
## I love programming  
## I love Python
```

or

```
print "Hello, World\nI love programming\nI love Python"
```

```
## Hello, World  
## I love programming  
## I love Python
```

'\n' to start a new line

Display Special Characters

```
print '\"Hello, World\"'  
print '\"\'Hello, World\'\"'
```

```
## "Hello, World"  
## 'Hello, World'
```

Display Variable Values

```
s = "Hello"  
t = "World"  
print s, t
```

```
## Hello World
```

Keyboard input method

Read a string

```
str = raw_input("Enter a string: ")
```

Read an integer

```
x = input("Enter an integer: ")
# read a string and convert it to int
x = raw_input("Enter an integer: ")
x = int(x)
```

Read a fractional number

```
x = input("Enter a number: ")
# read a string and convert it to float
x = raw_input("Enter a number: ")
x = float(x)
```

Variables, data types and operators

Variables

- ▶ A variable is a name for data stored 'in' the program
- ▶ A variable is automatically created by assignment
 - ▶ No need to define variables (unlike C++ or Java)

variable = expression

- ▶ Variable naming rules
 - ▶ **Cannot** use keywords (if, else, while, for, ...)
 - ▶ Must be one word and cannot contain spaces
 - ▶ First character must be a (upper or lower-case) letter or '_'
 - ▶ Cannot contain any other character than letters, numbers and '_'
 - ▶ Case sensitive: **student** and **Student** are distinct variables

Data types

- ▶ A variable can be used to hold different types of data
- ▶ Common data types used in Python
 - ▶ Whole Numbers (Integer): -5, 1, 3
 - ▶ Fractional Numbers (Float): 1.5, -0.8
 - ▶ Strings: "hello", "Room", "PYTHON"
 - ▶ Ordered Data — call by index
 - ▶ Lists: ['Australia', 'China', 'USA']
 - ▶ List can contain mixed types: ['Australia', 'China', 2, 5.8]
 - ▶ Tuples: ('Australia', 'China', 'USA')
 - ▶ Tuples are immutable (can't change once created)
 - ▶ Dictionaries: {'name': 'Jackson', 'Title': 'Dr', 'Age': 30}
 - ▶ Basically a list of key-value pairs
 - ▶ Unordered Data — call by key

Complex Data types, e.g. lists of lists

Examples of variable assignment

- ▶ `room=234` ✓ Assign an integer 234 to the variable named room.
- ▶ `room='234'` ✓ Assign a string '234' to the variable named room.
- ▶ `room=[234,123]` ✓ Assign a list to the variable named room; the list contains two integers: 234 and 123.
- ▶ `234=room` ✗ Variable can only be placed on the left side of = operator.
- ▶ `2room=234` ✗ Illegal variable name

Operators

- ▶ Arithmetic Operators (+,-, *, /, %, **)
- ▶ Order: $** > * / \% > + -$, use () to change order — or always use ()!
- ▶ Integer division vs float division

```
a=5/2 # a=2,  
b=5.0/2 # b=2.5
```

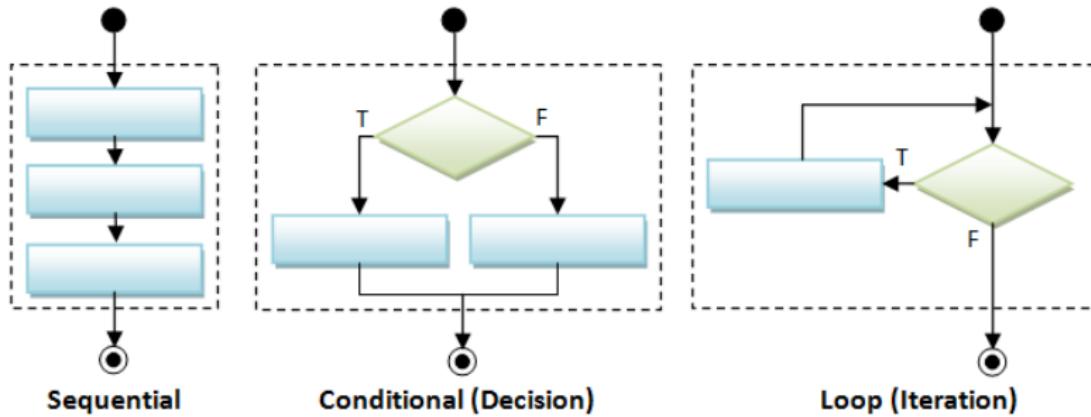
- ▶ Relational Operators (>, >=, <, <=, ==, !=)
- ▶ Used for variable comparison, e.g. numbers and strings
- ▶ == (equality) vs = (assignment)

```
if a==b:  
    print "a equals b"
```

- ▶ Logical Operators (and, or, not)
- ▶ Order: not > and > or — use ()
- ▶ Arithmetical > Relational > Logical — use ()

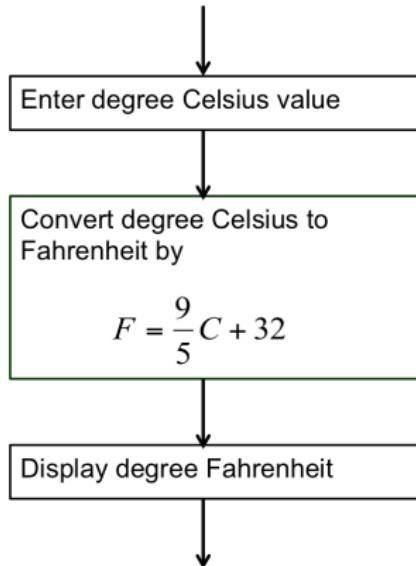
Program structures

Control structures



Sequential

Example: Celsius to Fahrenheit converter



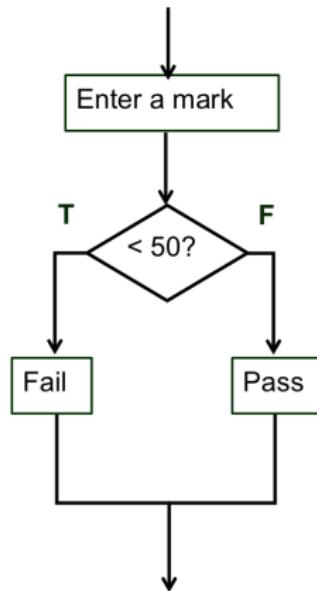
```
C = input("Enter a Celsius value: ")
```

```
F = 9.0/5*C+32
```

```
print C,"Celsius =",F,  
"Fahrenheit"
```

Conditional

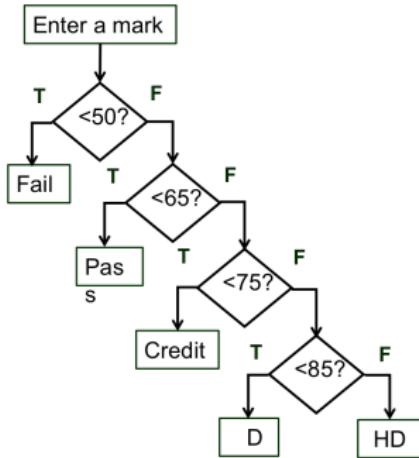
Example: grade calculator I



```
mark = input("Enter  
your mark: ")
```

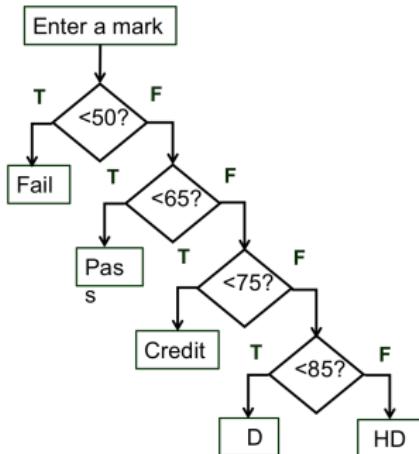
```
if mark<50:  
    print "Fail"  
else:  
    print "Pass"
```

Example: grade calculator II



```
mark = input("Enter your mark: ")
if mark<50:
    print "Fail"
else:
    if mark<65:
        print "Pass"
    else:
        if mark<75:
            print "Credit"
        else:
            if mark<85:
                print "D"
            else:
                print "HD"
```

Example: grade calculator II (using elif statement)



```
mark = input("Enter your mark: ")
if mark<50:
    print "Fail"
elif mark<65:
    print "Pass"
elif mark<75:
    print "Credit"
elif mark<85:
    print "D"
else:
    print "HD"
```

More about conditional structure

- More complex conditions can be described by using logical operators (and, or, not)

```
if age>65 and income<10000:  
    print "qualify for pensioner discount"
```

- **Correct indentation is important** in if and *all* other python code blocks!

✗

```
if mark<50:  
print "Fail"
```

✗

```
if mark<50:  
    print "Fail"  
else:  
    print "Pass"
```

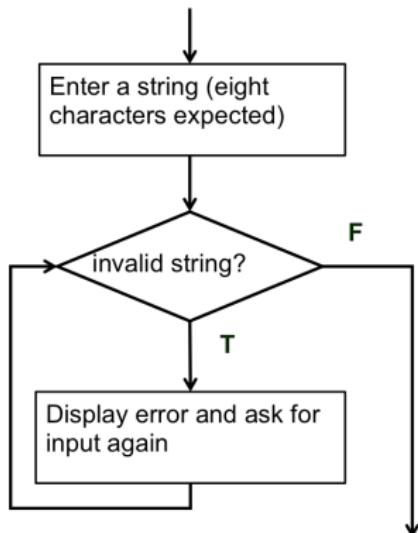
✓

```
if mark<50:  
    print "Fail"  
else:  
    print "Pass"
```

Don't miss out the ':' at the end of if statement

Iterative (loops)

Example: input validation



```
str = raw_input("Enter a 8-character string")
```

```
# len() returns the size of string,  
# list, tuple, ...
```

```
while len(str)!=8:  
    print "Input error"  
    str = raw_input("Enter a  
8-character string")
```

Example: range-based loop

Loops can be used to go through all items in a list

```
# print all items in the list          # sum over all items in the list

xlist = [1, 3, 5, 7, 9]                xlist = [1, 3, 5, 7, 9]
for x in xlist:                         sum = 0
    print x,                            for x in xlist:
print                                         sum = sum + x
                                            print "sum =", sum

## 1 3 5 7 9                           ## sum = 25
```

How about searching for a number in the list?

Or finding the maximum/minimum value?

Specifying range

```
range(stop)  
range(start, stop[, step])
```

- ▶ return a list of numbers
(integers only!)
- ▶ start: begin value
(inclusive) of the list, 0 if omitted
- ▶ stop: end value
(exclusive) of the list
- ▶ step (optional): interval between two values, 1 by default
- ▶ equivalent to Matlab ':' operator (= start:step:stop)

Examples:

```
# x=[0,1,2,...,9]  
x = range(10)  
x = range(0,10)  
# x=[1,3,5,7,9]  
x = range(1,10,2)  
# x=[10,9,...1]  
x = range(10, 0, -1)  
# how about [10,9,...,0]?
```

Get help

1. Use `help()` function: `help(cumsum)`
2. Use `?cmd` or `cmd?: ?sum` (type in `q` to quit help page)
3. Use `cmd` without brackets: `range` (with very limited text)
4. Use internet: most comprehensive way of getting help

Coming up next

Coming up next

- ▶ Python Basics II
- ▶ Relational Databases and SQL
- ▶ NoSQL Databases

Applications to Big Data

Lecture 03 – Python Basics II

Nick Tothill

Strings

String operations

```
# Basic manipulations
# create a string
s = "Hello world"

# Length of string
len(s) # = 11

# Type conversion
# string to number
s = "123.0"
s = float(s) # s = 123.0
s = "123"
s = int(s) # s = 123
# number to string
s = str(123) # s = "123"
```

```
#Type checks
s.isalpha() # letters
s.isdigit() # digits
# letters + digits
s.isalnum()
s.isspace() # spaces
s.isupper() # upper-case
s.islower() # lower-case
#Case conversion
s = "hEllo"
s = s.upper() # "HELLO"
s = s.lower() # "hello"
s = s.capitalize() # "Hello"
```

String operations

```
# String concatenation
t = "Hello"
# s = "Hello world"
s = t + " world"
```

```
#Create substring
s = "Hello world"
# s = "ell"
s = s[1:4]
# s = "world"
s = s[6:]
s = s[-5:]
```

```
# String repetition
# s = "Hello Hello Hello "
s = "Hello "*3
```

s[0] s[1]	
H	e
I	I
o	w
	o
	r
	l
	d
0	1
1	2
2	3
3	4
4	5
5	6
6	7
7	8
8	9
9	10
-11	-10
-10	-9
-9	-8
-8	-7
-7	-6
-6	-5
-5	-4
-4	-3
-3	-2
-2	-1

How about `s[:5]`? or `s[:-6]`?

String operations

```
# Split and join
s = "Python is fun"
# toks[0] = "Python"
# toks[1] = "is"
# toks[2] = "fun"
toks = s.split()
# s = "Python,is,fun"
s = ",".join(tok)

# Remove spaces
s = " Python is fun "
# s = "Python is fun"
s = s.strip()
```

```
# Find and replace
# "Hello Hello Hello "
s = "Hello " * 3
# pos = 0
pos = s.find("Hello")
# pos = -1, not found
pos = s.find("hello")
# pos = 6, skip first one
pos = s.find("Hello", 2)
# s = "world world world"
s = s.replace("Hello", "world")
# s = "world Hello Hello"
s = s.replace("Hello", "world", 1)
```

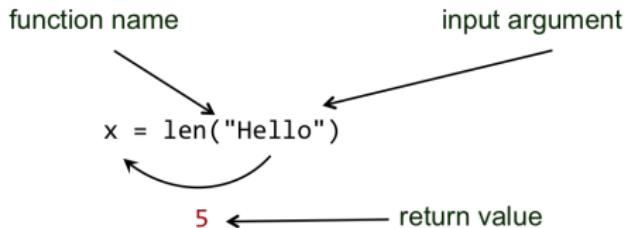
Functions

Functions

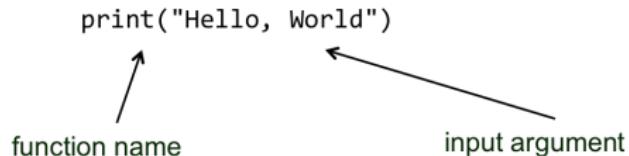
- ▶ Function is a stored procedure to performs some task
- ▶ How to use functions?
 - ▶ Function definition: define the function
 - ▶ Function call: use the defined function elsewhere
- ▶ Two types of functions in Python
 - ▶ Built-in (system-defined) functions
 - ▶ `print()`, `input()`, `raw_input()`, `float()`, `int()`, `len()`...
 - ▶ Must treat built-in function names as reserved words
 - ▶ User-defined Functions

Elements in functions

Function with return values



Function with no return values



Define and use a function

Function definition

```
def add(a, b):  
    c = a + b  
    return c
```

Use a function

```
a = input("Enter the first number: ")  
b = input("Enter the second number: ")  
c = add(a, b)    # Function call  
print "sum =", c
```

- ▶ keyword **def** indicates start of function definition
- ▶ indentation is used to indicate content of function
- ▶ **return** statement returns value to caller
 - ▶ multiple values can be returned by tuple e.g. **return (a,b)**

Why use functions?

```
# x = [0,1,2,3,4,5,6,7,8,9]
xlist = range(10)
# the following code prints the sum of xlist
sum = 0
for x in xlist:
    sum = sum + x
print "sum =", sum
# x = [1,3,5,7,9]
xlist = range(1, 10, 2)
# the following code prints the sum of xlist
sum = 0
for x in xlist:
    sum = sum + x
print "sum =", sum

## sum = 45
## sum = 25
```

Code reuse

```
# define the sum function
def sumFunc(xlist):
    sum = 0
    for x in xlist:
        sum = sum + x
    print "sum =", sum

# x = [0,1,2,3,4,5,6,7,8,9]
xlist = range(10)
# the following code prints the sum of xlist
sumFunc(xlist)
# x = [1,3,5,7,9]
xlist = range(1, 10, 2)
# the following code prints the sum of xlist
sumFunc(xlist)

## sum = 45
## sum = 25
```

Summary

- ▶ A function can be defined once and used everywhere throughout the program
 - ▶ Enhance code reusability and maintainability
 - ▶ Avoid changing one place without updating other parts
 - ▶ Improve readability and create better structured program
- ▶ Function design considerations
 - ▶ What does the function do?
 - ▶ What input does the function take? (input arguments)
 - ▶ What result should the function return?
 - ▶ No return value: e.g. print the result inside function
 - ▶ Return value required: e.g. return the calculation result to the caller of the function

More on data types

List and tuples

List

- A list is a collection of values

```
food = ["chicken", "beef", "egg", "milk"]
```

- '[' and ']' are used to define the list
- items are separated by ','s — A list item can be any object — even another list

```
xlist = ["chicken", 2.5, ["egg", "milk"], 50]  
        ↑   ↑   ↑   ↑  
xlist[0] xlist[1] xlist[2] xlist[3]  
           ↖   ↗  
           xlist[2][0] xlist[2][1]
```

Lists behaves like arrays in C++ and Java and follow similar indexing rules.

List operations

Create a list

```
x = ['hello', 'world']    # list of two words
x = []      # an empty list
x = range(5)    # x=[0,1,2,3,4]
# x = [0,1,4,9,16]
x = [i**2 for i in range(5)]
```

Search a list

```
x = ['hello', 'world']
# return the position of "world" in the list
pos = x.index("world")      # pos = 1
# raises a ValueError if item not found
pos = x.index("work")       # pos undefined
```

Modify a list

Initial: `x = [0,1,2]`

1. add elements to the end

```
x = x + [3]      # x = [0,1,2,3]  
x.append(3)      # x = [0,1,2,3]  
x = x + [3,4,5]  # x = [0,1,2,3,4,5]  
x.extend([3,4,5]) # x = [0,1,2,3,4,5]
```

2. insert element in the middle

```
x.insert(1,5)  # x = [0,5,1,2]
```

3. add elements in the front

```
x.insert(0,3)    # x = [3,0,1,2]  
x = [3] + x     # x = [3,0,1,2]  
x = [3,4] + x   # x = [3,4,0,1,2]
```

Iteration and List Comprehension

List items can be iterated in a loop

```
for x in range(5):  
    print x
```

```
## 0  
## 1  
## 2  
## 3  
## 4
```

This is the same as

```
for x in [0,1,2,3,4]:  
    print x
```

```
## 0  
## 1  
## 2  
## 3  
## 4
```

Return the list index in a loop

```
Z = ["Hello", "world", "Python"]
for i,x in enumerate(Z):
    print i, x

## 0 Hello
## 1 world
## 2 Python
```

List comprehension offers easy and natural ways to construct lists

```
squares = [x**2 for x in range(5)]
print squares
evens = [ x for x in range(10) if x % 2==0]
print evens

## [0, 1, 4, 9, 16]
## [0, 2, 4, 6, 8]
```

Tuples

- ▶ Tuples are sequences that behave like lists
 - ▶ Unlike lists, tuples are immutable and can't be changed
 - ▶ Tuples are defined by '(' and ')', lists use '[' ']'
`x = ('Jack', 'Smith', 'Lecturer', 'B', 1)`
Note items can have different data types
- ▶ Retrieve items of a tuple
 - ▶ `x[0], x[1], ...`: first, second,... items of a tuple
 - ▶ Tuple Expansion:
`(fName, lName, title, level, step) = x`
equivalent to 5 separate assignments so that `fName = 'Jack'`, `lName = 'Smith'`, and so on.

Mutable vs Immutable Types

Mutable Types (List)

```
# create a list
x = ["Hello", "World"]
# change the second item
to "Python"
x[1] = "Python" ✓
# add an item to list
x.append("Python") ✓
# insert an item before
the second item in list
x.insert("Python",1) ✓
# remove an item
x.remove("Python") ✓
```

Immutable Types (String, Tuple)

```
x = ("Hello", "World")
# Error, not allowed to
change the content
x[1] = "Python" ✗
x = x + "Python" ✗
# OK, create a new tuple
x = ("Hello", "Python") ✓

x = "Hello, World"
# Error, not allowed
x[5] = ' '
# OK, create a string
x = x + " Python" ✓
```

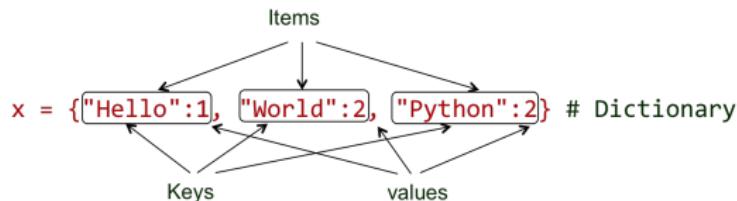
Dictionary

Collection is a bunch of values in a single variable

- ▶ **List, Tuple:** collection of single values in order

```
x = ["Hello", "World", "Python"]      # List
```

- ▶ **Dictionary:** order-less collection of key-value pairs



Keys must be unique, case sensitive if keys are strings

Create a dictionary

```
x = {"Hello":1, "World":2, "Python":2}  
# the following defines the same dictionary  
x = dict()  
x["Hello"] = 1  
x["World"] = 2  
x["Python"] = 2
```

Modify a dictionary

```
# update the value for an existing key  
x["Hello"] = 2  
# add a new key-value pair  
x["Programming"] = 5  
# delete a key-value pair  
del x["World"]
```

Dictionary example: counting words

```
wordList = ["hello", "hello", "world", "python", "PYTHON", "Hello"]
# define a list of words
wordDict = dict() # create an empty dictionary
for word in wordList:
    word = word.lower() # convert to lower case
    # add a new word to dictionary
    if word not in wordDict:
        wordDict[word] = 1
    else: # increment count for old word
        wordDict[word] = wordDict[word] + 1
print wordDict

## {'python': 2, 'world': 1, 'hello': 3}
```

Files

Open files

Files can be opened with the `open()` function

```
fid = open("mytext.txt")
```

`open()` returns a file identifier (stored in `fid`) – a handle for further file operations

It returns error if file does not exist

File reading

```
# read all lines into lines variable  
lines = fid.read()  
# read the next line into line variable  
line = fid.readline()
```

An open file must be closed by `fid.close()`

File read example

```
fid = open("mytext.txt")
# print each line of mydata.txt in a loop
for line in fid:
    print line
fid.close()

## First line
##
## Second line
##
## Three lines in total

fid = open("mytext.txt")
# print each line of mydata.txt in a loop
for line in fid:
    print line.strip()
fid.close()

## First line
## Second line
## Three lines in total
```

Unpleasant extra line break

Write to files

Opening files for writing

```
fid = open("mydata.txt", "w")
```

- ▶ creates a new file if mydata.txt does not exist
- ▶ *overwrites* the old file if mydata.txt already exists
- ▶ use "a" instead of "w" to append to mydata.txt instead of overwriting

Write to files

```
fid.write(line)
```

- ▶ need to pay attention to "newlines"
- ▶ print() prints a new line automatically, write() does not
- ▶ may have to use fid.write(line+'\n') in most cases

Coming up next

Coming up next

- ▶ Relational Databases and SQL
- ▶ NoSQL Databases
- ▶ Data Format

Applications of Big Data

Lecture 04 — Relational Databases & SQL

Nick Tothill

Overview of Relational Database

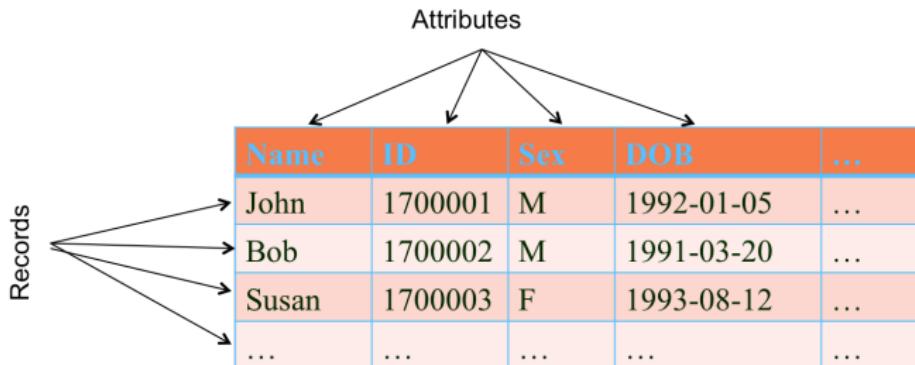
Relational Databases

- ▶ Database: collection of data and rules to organise the data
- ▶ Relational Database
 - ▶ based on the relational model of data
 - ▶ organise data into one or more tables
- ▶ RDBMS (Relational Database Management System)
 - ▶ Oracle, MS SQL Server, MySQL, SQLite, ...
- ▶ All RDBMS implements a version of the Structured Query Language (SQL)
- ▶ We will use SQLite for this unit
 - Simple standalone application, no installation needed

Tables in Relational Databases

- ▶ Table: a single database file that represents an entity
 - Important building block of a relational database
- ▶ Schema: logical definition of the table
 - name of table
 - attribute names and data types
- ▶ Components of a table
 - Records: rows of the table
 - Attributes: columns of the table
 - Fields/Elements: data items contained in the table
- ▶ Limitations of table
 - Requires structured and clean data

Table Example



Schema: Table PersonallInfo
(Name: Text, ID: Integer, Sex: Text, DOB: Date)

SQL Basics

Why SQL?

- ▶ Standard language for database access and manipulation
- ▶ Widely implemented in RDBMS and supported by various programming languages for embedded applications
- ▶ *Scale up* to bigger systems
- ▶ Hard to *Scale out* to parallel systems
- ▶ Implications for data warehouse infrastructure and modern big data platforms (e.g. Hadoop)

Examples of software to allow massively-parallel SQL:

- ▶ Apache Pig (Pig Latin)
- ▶ Apache Hive (HiveQL)
- ▶ Cloudera Impala

Example Database

records

Sid	Sname	Sex	DOB	Uid	Mark	Uname	Utype
1839 5061	Adam Ross	M	1988-06-20	300580	88	Programming Fundamentals	Program ming
1839 5061	Adam Ross	M	1988-06-20	300581	79	Programming Techniques	Program ming
1839 5061	Adam Ross	M	1988-06-20	300585	74	Systems Analysis and Design	Analysis
1839 5061	Adam Ross	M	1988-06-20	300144	59	Object Oriented Analysis	Analysis
1839 5061	Adam Ross	M	1988-06-20	301046	92	Big Data	Other
...							

View Table

Show records in the table

```
SELECT * FROM record; --> all records
```

```
SELECT * FROM record LIMIT 10; --> the first 10 records
```

Note:

- ▶ SQL commands, table and attribute names are case **insensitive**.
- ▶ Attribute values are **case sensitive**.

Column selection

```
SELECT Sid, Sname FROM record;
```

```
SELECT Sname, Mark FROM record;
```

Record Selection Examples

Names of all female students

```
SELECT Sname FROM record WHERE Sex='F';  
SELECT DISTINCT Sname FROM record WHERE Sex='F';
```

Marks for programming units

```
SELECT Mark FROM record WHERE Utype='Programming';
```

Record Selection Examples

Records for all Smiths

```
SELECT * FROM record WHERE Sname like '%Smith';
```

Students who failed programming units

```
SELECT Sname, Uname, Mark FROM record WHERE  
Utype='Programming' AND Mark<50;
```

Aggregate Examples

Total records

```
SELECT COUNT(*) FROM record;
```

Total students

```
SELECT COUNT(*) FROM  
(SELECT DISTINCT Sid FROM record);
```

Aggregate Examples

Lowest, average and highest marks

```
SELECT MIN(Mark), AVG(Mark), MAX(Mark) from record  
where Uid=300581;
```

Top results in Programming Techniques

```
SELECT Sname, Mark FROM record WHERE Uid=300581  
ORDER BY Mark DESC LIMIT 5;
```

Group By Examples

Group by students

Number of units taken by each student

```
SELECT Sname, COUNT(*) FROM record GROUP BY Sid;
```

Average mark for each student

```
SELECT Sname, AVG(Mark) FROM record GROUP BY Sid;
```

Group By Examples

Results by sex

```
SELECT AVG(Mark) FROM record WHERE Utype="Analysis" GROUP BY Sex;
```

=

```
SELECT AVG(Mark) FROM record WHERE Utype="Analysis" and Sex="F";
```

+

```
SELECT AVG(Mark) FROM record WHERE Utype="Analysis" and Sex="M";
```

Data Analysis with SQL and Python

Performance by Sex Groups

Problem: test whether female students under perform male students in programming units using SQL queries.

- ▶ Average marks of female and male students in programming

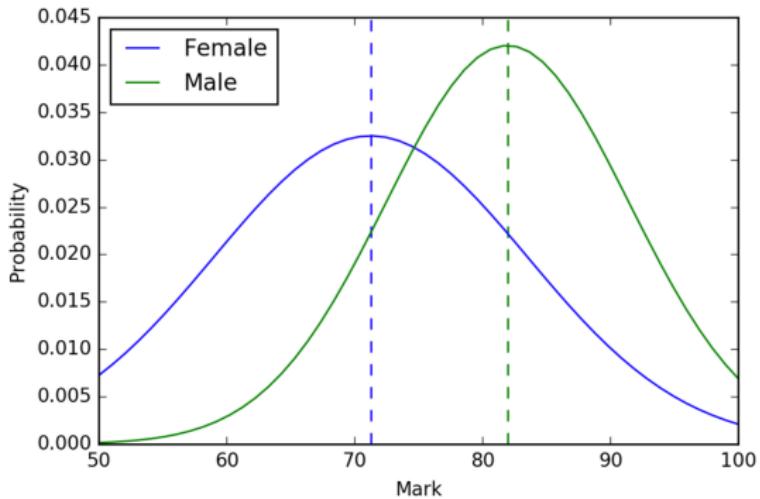
```
SELECT Sex, AVG(Mark) FROM record  
where Utype="Programming" GROUP BY Sex;
```

- ▶ Standard deviation of marks in programming

$$std(X) = \sqrt{avg(X^2) - avg^2(x)}$$

```
SELECT Sex, AVG(Mark*Mark) FROM record  
where Utype="Programming" GROUP BY Sex;
```

Sex	Avg(Mark)	Avg(Mark ²)	Std(Mark)
F	71.25	5227.05	$\sqrt{5227.05 - 71.25^2} = 12.27$
M	81.95	6805.85	$\sqrt{6805.85 - 81.95^2} = 9.49$



Alternatively, we can check the number of students falling in each grade level.

Number of fails

```
SELECT COUNT(*) FROM record  
where Utype="Programming" and  
Mark<50 GROUP BY Sex;
```

Number of passes

```
SELECT COUNT(*) FROM record  
where Utype="Programming" and  
(Mark>=50 and Mark<65)  
GROUP BY Sex;
```

Similar for D and HD

	F	M
F	1	0
P	4	1
C	7	4
D	5	5
HD	3	10

Correlation of Similar Units

Problem: test whether students get higher marks in one programming unit also obtain higher marks in the other.

- ▶ Retrieve all marks in Programming Techniques (300581)

```
.mode csv --> set to csv mode, only need to do it once  
.output 300581.csv --> write results to csv file  
SELECT Mark FROM record where Uid=300581;
```

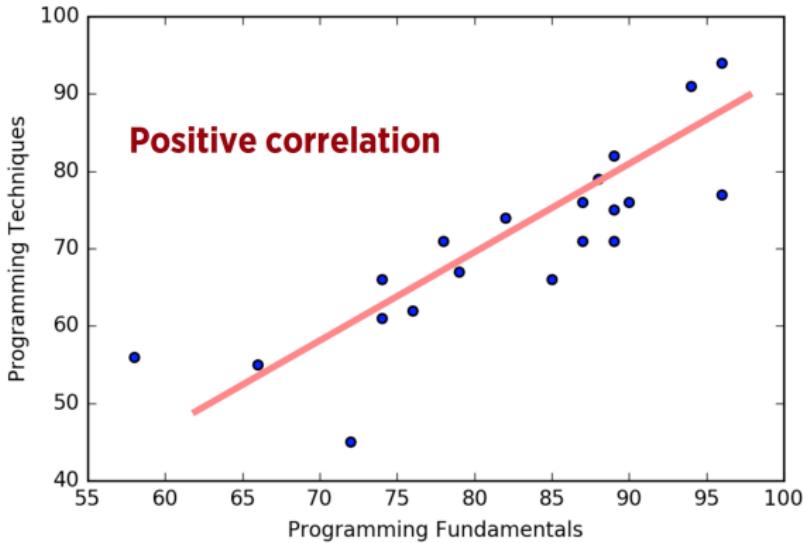
- ▶ Retrieve all marks in Programming Fundamentals (300580)

```
.output 300580.csv --> write results to csv file  
SELECT Mark FROM record where Uid=300580;  
.output stdout --> reset display to screen
```

Python data analysis in a Jupyter Notebook (with Anaconda)

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

# import marks for programming techniques
pt = pd.DataFrame.from_csv("300581.csv", header=None, index_col=None)
# import marks for programming fundamentals
pf = pd.DataFrame.from_csv("300580.csv", header=None, index_col=None)
# create a scatter plot
plt.scatter(pt, pf)
plt.show()
```



Performance Over Age

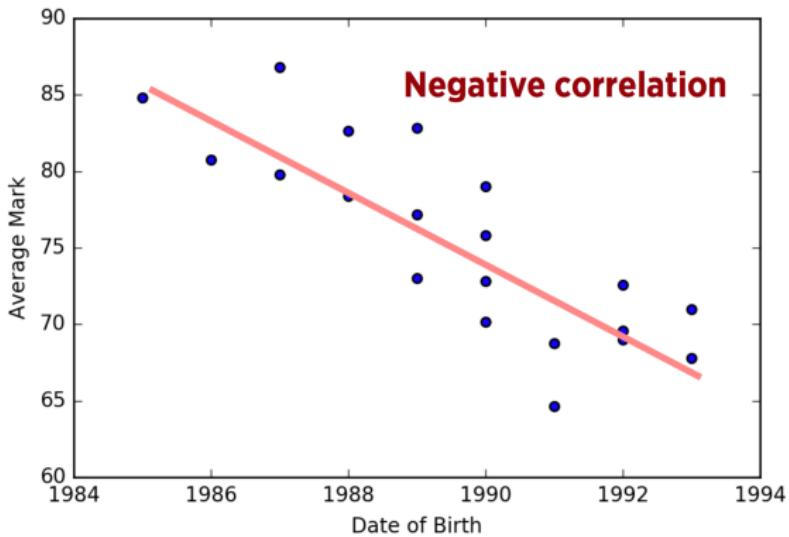
Problem: test if mature students achieve better marks on average than younger students.

Retrieve average marks for each student alongside their age — Age can be calculated by subtracting DOB from current date

```
.mode csv    --> set to csv mode, only have to do once  
.output marks.csv --> write results to csv file  
SELECT DATE("now")-DOB, AVG(Mark) FROM record GROUP BY Sid;  
.output stdout --> reset display to screen
```

Python data analysis in a Jupyter notebook

```
# No need to import the libraries if already done so
import pandas as pd
import matplotlib.pyplot as plt
# import data file, the return value is a DataFrame with
# two columns for ages and marks respectively
res = pd.DataFrame.from_csv("marks.csv",header=None, index_col=0)
# create a scatter plot between DOB and mark
plt.scatter(2019-res[0], res[1])
plt.show()
```



More about Correlation Analysis

- ▶ Correlation captures the relations between two variables
- ▶ x and y are positively correlated if increased x is found along with increased y
 - ▶ Marks of two programming (or analysis) units
 - ▶ Age versus Mark
 - ▶ Education Level versus Salary
- ▶ x and y are negatively correlated (or anti-correlated) if decreased x is found alongside increased y
 - ▶ DOB versus Mark
 - ▶ Education Level versus Crime Rate
- ▶ *Correlation does not necessarily indicate causality*
 - ▶ Ice-cream Sales versus Drowning Rate

Access SQL database through python

Python has many packages that enable one to access SQL database directly. The simplest one is through IPython extension.

One has to install IPython-sql package: install it through specified repository or through pip. We introduce pip installation which is fully automated. Steps:

- ▶ install pip package in canopy using package manager
- ▶ open an canopy terminal in **Tools** menu
- ▶ input `pip install ipython-sql` to install the package

N.B. `ipython-sql` is not a standard canopy package, so it will *not* appear in installed package list in canopy package manager. But you can use it freely once you install it correctly. One can always use `pip list` to check installed packages including non-standard ones like `ipython-sql`.

Connect to SQL database

```
# Use %load_ext to load sql extension
%load_ext sql
%sql sqlite:///testdb.sqlite # Connect to testdb.sqlite

# Perform SQL queries
pt = %sql SELECT Mark FROM record WHERE Uid = 300581;
pf = %sql SELECT Mark FROM record WHERE Uid = 300580;

# Now we can do something such as correlation analysis
# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model
regr.fit(pf, pt)

plt.scatter(pf, pt)
plt.ylabel("Programming Techniques")
plt.xlabel("Programming Fundamentals")
plt.plot(pf, regr.predict(pf), color='pink', linewidth=3)
plt.show()
```

Caveat: **ONLY works in IPython terminal and IPython notebook!** Not working in batch mode. It is often used as a data exploration tool instead.

Conclusions

In this lecture, we covered

- ▶ Database Fundamentals
- ▶ Structured Query Language (SQL)
- ▶ Simple Data Analysis with SQL and Python
 - ▶ Distribution Comparison
 - ▶ Correlation Analysis

Coming up next

Coming up next

- ▶ NoSQL Databases
- ▶ Data Format
- ▶ Web Scraping
- ▶ APIs
- ▶ Pandas Library

Applications of Big Data

Lecture 05 — NoSQL Databases

Nick Tothill

Introduction to NoSQL Databases

Pros and Cons of Relational Databases

Pros:

- ▶ Standard SQL for data access and manipulation
- ▶ Proper schema design ensures low redundancy
- ▶ Transaction mechanism ensures strong consistency
- ▶ Backed up by mature and rigid theory

Cons:

- ▶ Inflexible for modelling complex relations
- ▶ Poor scalability for big data
- ▶ High latency for concurrent data accesses
- ▶ Reliability and availability issues

We need different types of database systems for different application needs.

NoSQL Databases

- ▶ Relational Databases are similar
 - ▶ All of them follow the relational data model
 - ▶ And same SQL standard
- ▶ NoSQL (Not Only SQL) is a generic term for any DBMS different from the traditional relational model
 - ▶ Do not rely on tabular relations
 - ▶ Light transaction semantics
- ▶ There are many different types of NoSQL databases
 - ▶ They follow different data storage models
 - ▶ Different languages and tools for database access

SQL vs NoSQL Databases

	SQL	NoSQL
Data Model	Relational Model	Non-relational, distributed
Data Storage	Tables	Key-value pairs, documents, graphs
Flexibility	Fixed schema, data must follow the same pattern	Dynamic schema, can hold different types of data
Scalability	Vertically Scalable	Horizontally Scalable
Concurrency and Consistency	Low concurrency and strong consistency	High concurrency and weak consistency
Data Access and Manipulation	SQL	Depends on the particular DBMS used
Examples	MySQL, Oracle, MS SQL Server, Sqlite, Postgres SQL, ...	MongoDB, CouchDB, Cassandra, BigTable, Redis, Hbase, Neo4j, ..

Vertical vs Horizontal Scalability

Vertical (SQL)



Horizontal (NoSQL)



- Impractical for Big Data
- Low reliability
- Interruption of service

- Feasibility
- Fault tolerant
- Availability

Types of NoSQL Databases

- ▶ Key-value store
 - ▶ Each record is a key-value pair
 - ▶ Examples: MemcacheDB, Redis, Riak
- ▶ Document store
 - ▶ Each record is a list of key-value pairs
 - ▶ Examples: MongoDB, CouchDB
- ▶ Wide-columns
 - ▶ Store columns of data together
 - ▶ Examples: BigTable, Cassandra, HBase
- ▶ Graph database
 - ▶ Examples: Neo4J, HyperGraphDB

MongoDB

MongoDB

- ▶ MongoDB is a document based NoSQL database
 - ▶ Document (record): list of key-value pairs
 - ▶ Collection (table): list of documents
- ▶ Why MongoDB?
 - ▶ Convenience: easy to learn and get started
 - ▶ Most similar to SQL database in the NoSQL family
 - ▶ Agility: iterative development cycle
 - ▶ Easy to accommodate for changes in development
 - ▶ Flexible data model: adding/changing fields made easier
 - ▶ Highly scalable: auto sharding

Case Study: Product Catalogue

Problem: product catalogue must have the capacity to store many different types of objects with different attributes.

Relational Database Solutions

- ▶ Separate tables for each different product category
 - ▶ Inflexible: new tables for new categories
- ▶ One table for all products with all attributes taken from different product categories
 - ▶ Poor Scalability: new columns for new products
- ▶ One table for the common attributes, a separate table for other attributes of each separate product category
 - ▶ Need to join the tables for full product information

Source:

<http://docs.mongodb.org/ecosystem/use-cases/product-catalog/>

Case Study: Product Catalogue

Problem: product catalogue must have the capacity to store many different types of objects with different attributes.

MongoDB Solution

- ▶ One collection for all products, each document describes one product and can take different keys (attributes)

Books: Title, Author, ISBN, Publisher, Shipping info ...

Songs: Title, Artist, Album, Genre, Label, Producer, ...

Movies: Title, Cast, Director, Genre, Classification, ...

...

Source:

<http://docs.mongodb.org/ecosystem/use-cases/product-catalog/>

JavaScript Object Notation (JSON)

- ▶ MongoDB uses Binary JSON to store documents
- ▶ JSON: a data exchange format widely used in web and database applications
 - ▶ Easy for human to read and machine to parse the format
 - ▶ Similar to a dictionary or list of key/value pairs

Examples

1. {“Name”:“Jack”, “Sex”:M, “Age”: 54}
2. {“Name”:“Jill”, “Sex”:F, “Age”: 51, “Address”: {“Post”: “Locked Bag 1797”, “Perm”: “2 Second Ave”}}
3. {“Name”:“Bill”, “Age”: 75, “Children”: [“Jack”, “Jill”, “John”]}

JavaScript Object Notation (JSON)

- ▶ MongoDB uses Binary JSON to store documents
- ▶ JSON: a data exchange format widely used in web and database applications
 - ▶ Easy for human to read and machine to parse the format
 - ▶ Similar to a dictionary or list of key/value pairs

Examples

```
{  
  "Name": "Jill",  
  "Sex": "F",  
  "Age": 51,  
  "Address":  
  {  
    "Post": "Locked Bag 1797",  
    "Residential": "2 Second Ave"  
  }  
}
```

MongoDB Command: insert

insert command can be used to create a new document (record) in the current collection

```
db.CollectionName.insert(doc)
```

Example

```
doc = {"Type": "Book", "Name": "The old men and the sea", "Author": "Hemingway", Year: 1952}  
db.product.insert(doc)
```

Or

```
db.product.insert( {"Type": "Book", "Name": "The old men and the sea", "Author": "Hemingway", Year: 1952})
```

A unique objectID is automatically created by MongoDB for each newly created document

MongoDB Command: update

update command can be used to overwrite or modify an existing document in the current collection

`db.CollectionName.update(query, operation)`

Example 1 (Overwrite the document)

```
db.product.update({"Author":"Hemingway"}, {"Author":"Ernest Hemingway"})
```

Example 2 (Modify the document)

```
db.product.update({"Author":"Hemingway"},  
{$set:{'Author':'Ernest Hemingway'}})  
db.product.update({"Author":"Hemingway"}, {$set:{'ISBN':  
"0684801221"}})
```

MongoDB command: find

find is the most important command for data access

- ▶ Syntax: `db.Collection.find(condition, projection)`
- ▶ find to MongoDB is select to SQL

SQL Select

```
SELECT Title, Author, Publisher, Year FROM product WHERE  
Year<"2000" and Type="Book";
```

MongoDB find

```
db.product.find({"Year":{$lt:"2000"}, "Type":"Book"}, {"Title":1,  
"Author":1, "Publisher":1, "Year":1});
```

More about query conditions

SQL to MongoDB Cheatsheet

SQL ←WHERE	MongoDB ←find({})
Year=2000	"Year": 2000
Year!=2000	"Year":{\$ne: 2000}
Year>2000	"Year":{\$gt: 2000}
Year>=2000	"Year":{\$gte: 2000}
Year<2000	"Year":{\$lt: 2000}
Year<=2000	"Year":{\$lte: 2000}
Title LIKE "Programming"	"Title": /Programming/
Title LIKE "Programming" AND Year>=2000	"Title": /Programming/, "Year": {\$gte:2000}
Shipping IS NULL	"Shipping": {\$exists: false}

More commands: count/limit/sort

Count the number of songs in the collection

```
db.product.count({"Type": "Song"})
```

Show 3 songs in the collection (arbitrary order)

```
db.product.find({"Type": "Song"}).limit(3)
```

Display 3 oldest songs in the collection

```
db.product.find({"Type": "Song"}).sort(  
{"ReleaseDate": 1}) → ORDER BY ReleaseDate  
).limit(3)
```

MongoDB command: aggregate

aggregate is similar to (but a lot more sophisticated than) group by in SQL

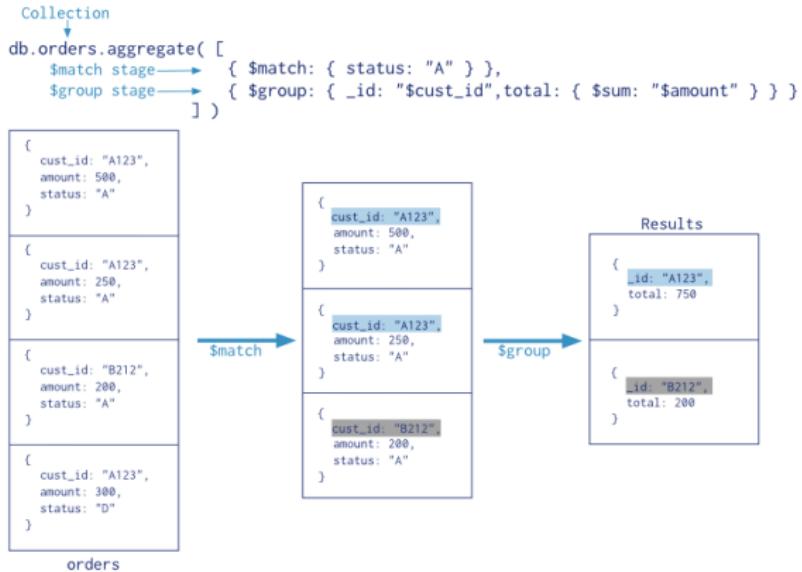
- ▶ Syntax: `db.CollectionName.aggregate(pipeline)`
- ▶ aggregate can be efficiently performed by map-reduce operation

Example: calculate the total price for each type of products

```
db.product.aggregate([ { $group:{_id: '$Type', total: {$sum:'$Price'}} } ])
```

```
SELECT Type as _id, SUM(Price) as total FROM product  
GROUP BY Type;
```

Mongodb aggregate pipeline



Source:

<https://docs.mongodb.com/manual/core/aggregation-pipeline/>

More aggregate examples

Calculate the total number for each type of products

```
db.product.aggregate([
  { $group:{_id: '$Type', total: {$sum:1}} }
])
```

Calculate the total price of all products

```
db.product.aggregate([
  { $group:{_id: null, total: {$sum:'$Price'}} }
])
```

Calculate the average price of books published before 2000

```
db.product.aggregate([
  { $match:{Type:'Book', Year:{$lt:2000}} }
  { $group:{_id: null, total: {$avg:'$Price'}} }
])
```

Conclusions

In this lecture, we covered

- ▶ NoSQL Introduction
 - ▶ Pros and cons of SQL
 - ▶ NoSQL vs SQL
 - ▶ Types of NoSQL Databases
- ▶ MongoDB
 - ▶ Case Study
 - ▶ Data Manipulation: insert, update
 - ▶ Data Queries: find, count, aggregate

Coming up next

Coming up next

- ▶ Web Service and APIs
- ▶ Map Reduce
 - ▶ Concept and Examples
 - ▶ Implementation: Hadoop
- ▶ Predictive Modelling
 - ▶ Data Pre-processing
 - ▶ Regression
 - ▶ Model Selection

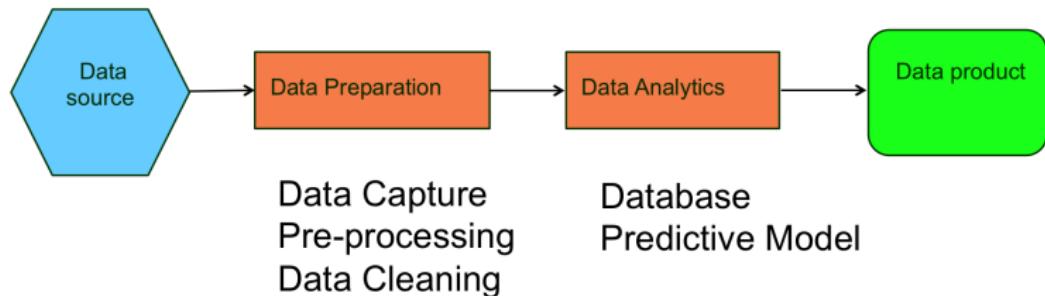
Applications of Big Data

Lecture 06 — Web Services and APIs

Nick Tothill

Introduction to Web Services and JSON

The challenges of Big Data



1. Data scientists need clean data for data analytics
2. Data are usually presented in raw and noisy form

Pros and Cons of Databases

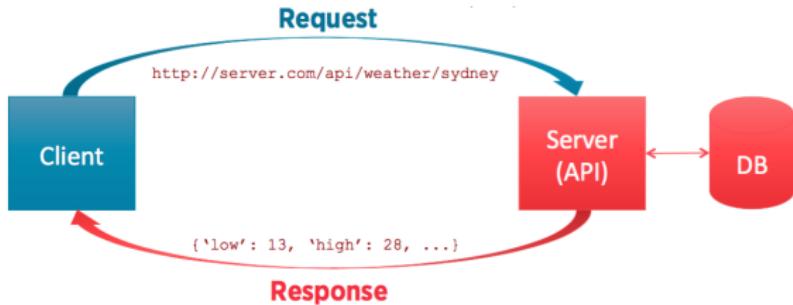
- ▶ Databases are good for accessing structured and clean data
- ▶ Real-world data are unstructured and noisy
 - ▶ E.g.: tweets, blogs, wiki, discussion forums
 - ▶ Html web pages in general
- ▶ We need to develop techniques and tools for data capture
 - ▶ Data wrangling: process raw data to present them in a form amenable to further analytics steps
 - ▶ Web scraping/crawling: a wrangling technique for directly grabbing and cleaning the data from the web
- ▶ Web services can help!

JavaScript Object Notation (JSON)

Most non-trivial web applications use web services

Services publish the “rules” (API) that applications must follow to use the service in their own programs

- ▶ Google Map APIs (<https://developers.google.com/map>)
- ▶ Twitter APIs (<https://dev.twitter.com>)
- ▶ Yahoo Query Language (<https://developer.yahoo.com/yql>)
- ▶ Usually implemented on top of HTTP/HTTPS
 - ▶ User submits a tailored HTTP(S) request to web server
 - ▶ Service responds with application-specific data
- ▶ Server response can be in different formats
 - ▶ XML, JSON, CSV



JavaScript Object Notation (JSON)

- ▶ JSON: standard data exchange format for web applications
 - ▶ Inspired by the object and array format used in JavaScript
 - ▶ Basically an embedded list of key value pairs

source: <http://www.json.org>

Example

```
{  
  "name": "Jack Smith",  
  "phone": "47320614",  
  "email": "j.smith@yahoo.com.au"  
}
```

Define an object

JavaScript Object Notation (JSON)

- ▶ JSON: standard data exchange format for web applications
 - ▶ Inspired by the object and array format used in JavaScript
 - ▶ Basically an embedded list of key value pairs

Example

```
{  
  "name": "Jack Smith",  
  "phone": "47320614",  
  "email": ["j.smith@westernsydney.edu.au",  
            "j.smith@yahoo.com.au"]  
}
```

A value can be
a list of items

JavaScript Object Notation (JSON)

- ▶ JSON: standard data exchange format for web applications
 - ▶ Inspired by the object and array format used in JavaScript
 - ▶ Basically an embedded list of key value pairs

Example

```
{  
  "name": "Jack Smith",  
  "phone": {  
    "type": "office",  
    "number": "47320614"  
  },  
  "email": ["j.smith@westernsydney.edu.au",  
            "j.smith@yahoo.com.au"]  
}
```

A value can be
another list of
key value pairs

JSON in Python

- ▶ JSON uses the same notation as Python dictionaries and lists

```
mydata =  
{  
    "name": "Jack Smith",           mydata["name"]  
    "phone": {  
        "type": "office",          mydata["phone"][0]  
        "number": "47320614"       mydata["phone"][1]  
    }  
    "email": ["j.smith@westernsydney.edu.au",  
              "j.smith@yahoo.com.au"]   mydata["email"][0]  
}                                     # ...
```

More JSON in Python

Iteration

```
for key in mydata:  
    print(mydata[key])
```

Condition

```
if "phone" in mydata:  
    # do something  
if "position" in mydata:  
    # do something  
if "number" in mydata:  
    # do something
```

More JSON in Python

Serialise and Deserialise

```
import json

jsondata = json.loads(strdata) # str to json
strdata = json.dumps(jsondata) # json to str

with open('file.json', 'r') as jsonfile:
    jsondata = json.load(jsonfile) # file to json

with open('file.json', 'w') as jsonfile:
    json.dump(jsondata, jsonfile) # json to file
```

Case Studies

Google Geocoding APIs

- ▶ Geocoding Service
 - ▶ Convert addresses into geographic coordinates (latitude/longitude) that can be positioned on the map
- ▶ Google Geocoding API
 - ▶ Allows access to the geocoding web services via HTTP request
 - ▶ API Request Format
 - ▶ JSON: [https://maps.googleapis.com/maps/api/geocode/json?
address=Parramatta](https://maps.googleapis.com/maps/api/geocode/json?address=Parramatta)
 - ▶ XML: substitute json with xml above
- ▶ Authentication
 - ▶ Use of API key is recommended but not enforced
 - ▶ Fair use policy: usage limits apply
 - ▶ Better to create an account and use an API key

JSON Output Format

```
{  
  "results": [{  
    "address_components": [  
      {"long_name": "Parramatta",  
       "short_name": "Parramatta",  
       "types": ["locality", "political"]}  
    ], {  
      "long_name": "New South Wales",  
      "short_name": "NSW",  
      "types": ["administrative_area_level_1", "political"]}  
    ], {  
      "long_name": "Australia",  
      "short_name": "AU",  
      "types": ["country", "political"]}  
  ],  
  "formatted_address": "Parramatta NSW, Australia",  
  "geometry": {  
    "bounds": {  
      "northeast": {  
        "lat": -33.8031802,  
        "lng": 151.0286023  
      },  
      "southwest": {  
        "lat": -33.8270708,  
        "lng": 150.9879955  
      }  
    },  
    "location": {  
      "lat": -33.8151285,  
      "lng": 151.0031549  
    },  
    "location_type": "APPROXIMATE",  
    "viewport": {  
      "northeast": {  
        "lat": -33.8031802,  
        "lng": 151.0286023  
      },  
      "southwest": {  
        "lat": -33.8270708,  
        "lng": 150.9879955  
      }  
    },  
    "place_id": "ChIJT69n4RijEmsRAMYyFmh9AQU",  
    "types": ["locality", "political"]  
  ],  
  "status": "OK"  
}
```

Geocoding API Code Example

```
import urllib
import json

list_of_suburbs = ['parramatta', 'campbelltown', 'bankstown',
'bondi junction', 'hornsby', 'emu plains']
service_url = 'http://maps.googleapis.com/maps/api/geocode/json'
for suburb in list_of_suburbs:
    uh = urllib.urlopen(service_url+suburb)
    data = uh.read()
    info = parseData(data)
    print info['country'], ',', info['locality'], ',', \
          info['latitude'], ',', info['longitude']
```

Geocoding API Code Example (Ctd.)

```
def parseData(data):
    info = {}
    # convert raw data to json format
    jsondata = json.loads(data)
    result = jsondata['results'][0]
    info['latitude'] = result['geometry']['location']['lat']
    info['longitude'] = result['geometry']['location']['lng']
    addr_components = result['address_components']
    for addr in addr_components:
        # locality, administrative_area_level1, country
        addrtype = addr['types'][0]
        addrval = addr['long_name']
        info[addrtype] = addrval
    return info
```

Geocoding API Code Example (Ctd.)

How about save the results to file? ➔ **Solution: modify the for loop and use JSON dump funtion**

```
info_suburbs = []
for suburb in list_of_suburbs:
    uh = urllib.urlopen(service_url+suburb)
    data = uh.read()
    info = parseData(data)
    print info['country'], ',', info['locality'], ',', \
          info['latitude'], ',', info['longitude']
    info_suburbs.append(info)
# save results to file
fp = open('suburbs.json', 'w')
json.dump(info_suburbs, fp)
fp.close()
```

Twitter APIs

- ▶ API objects and keys
 - ▶ **Tweets:** contributors, created_at/delete, lang, text, id, user, place, entities, retweet_count
 - ▶ **Users:** created_at, id, name, location, lang, entities, followers_count, friends_count
 - ▶ **Entities:** url, media, hashtags, user_mentions, ...
 - ▶ **Places:** bounding_box, country, full_name, ...
- ▶ More information can be found at
 - ▶ <https://dev.twitter.com/overview/api>

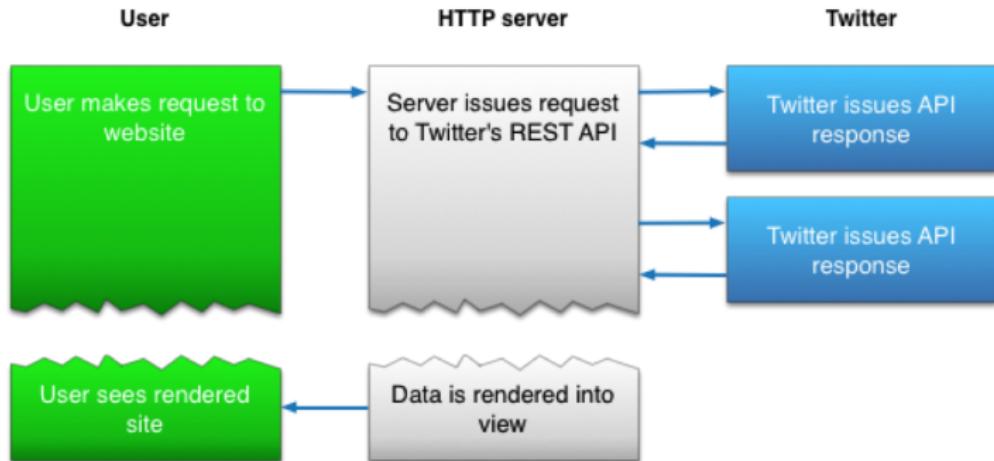
Twitter Authentication Model

- ▶ Twitter uses OAuth to provide authorized access to its API
 - ▶ Oauth: an open protocol to allow secure authorisation for web and mobile applications
 - ▶ Secure: using keys and secrets to access shared resources
 - ▶ Standard: third-party libraries and example code available for OAuth implementation
- ▶ Twitter Authentication Model
 - ▶ Application-User Authentication: API key and secret (per user based), Access key and secret (per app based)
 - ▶ Application-Only Authentication: Access key and secret
 - ▶ HTTP authentication is deprecated since API v1.1

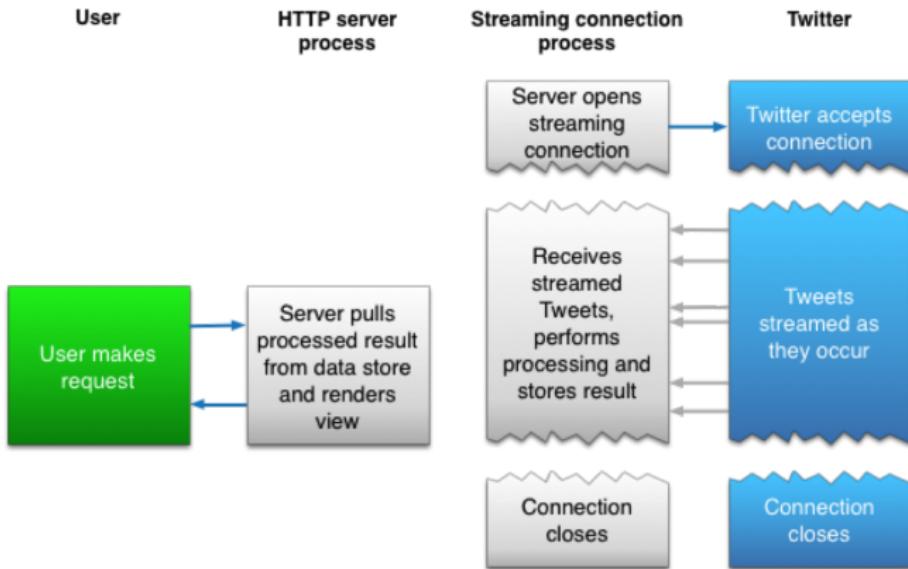
Twitter APIs

- ▶ REST APIs
 - ▶ Provides programming access to standard Twitter functions
 - ▶ e.g. read and write Twitter data, author a new Tweet, search authors and tweets, read author profile, ...
- ▶ Streaming APIs
 - ▶ Provides low latency access to Tweet data stream
 - ▶ Public Streams, User streams, Site streams
 - ▶ Requires persistent HTTP connection open
- ▶ When to use which APIs?
 - ▶ REST: standard Twitter functions, focused applications
 - ▶ Streaming: data mining, follow specific users

Twitter REST API



Twitter Streaming API



REST API Example

- ▶ Search tweets by keyword
- ▶ We use tweepy, a Python wrapper for Twitter APIs
- ▶ The following code sets up OAuth authentication

```
import tweepy

access_token = "Your access token goes here"
access_secret = "Your access token secret goes here"
consumer_key = "Your consumer key goes here"
consumer_secret = "Your consumer secret goes here"

auth = tweepy.OAuthHandler(consumer_key, consumer_secret)
auth.set_access_token(access_token, access_secret)
```

REST API Example (Ctd.)

The following code performs keyword search for tweets using REST API and the OAuth object created before

```
api = tweepy.API(auth)
results = api.search(q="Python")
# results contain a list of tweets returned by keyword search
for tweet in results:
    # print tweet text if it is in English
    if tweet.lang=='en':
        print tweet.text
```

Streaming API Example

```
# create a customised listener class
class listener(tweepy.streaming.StreamListener):
    # defines the behaviour when a new tweet is captured
    def on_data(self, data):
        print data.strip() # print the tweet
        return True
    # defines the behaviour when an error is occurred
    def on_error(self, status):
        print status

# create a new stream using the OAuth object and an
# instance of customised listener
tstream = tweepy.streaming.Stream(auth, listener())
tstream.sample() # sampling Twitter streams
```

Streaming API Example

Print to file instead of display on screen?  **Solution: create a constructor and modify the on_data function of the listener class**

```
class listener(tweepy.streaming.StreamListener):
    # define the constructor, invoked once when the class instance :
        def __init__(self):
            self.fid = open("samples.json", 'w')
    # defines the behaviour when a new tweet is captured
        def on_data(self, data):
            self.fid.write(data.strip()+'\n')
            return True
```

Streaming API Example

Choose the Tweets to be saved to file

```
class listener(tweepy.streaming.StreamListener):
    def __init__(self):
        self.fid = open("samples.json", 'w')
    # defines the behaviour when a new tweet is received
    def on_data(self, data):
        # decode the raw data into json format
        decoded = json.loads(data.strip())
        # save the tweet if it contains "created_at" key
        if "created_at" in decoded:
            self.fid.write(data.strip()+'\n')
    return True
```

About the code

All python code is available on *vuws* in `lec6.py`. You may need to tweak them to work for your application though.

Coming up next

Coming up next

- ▶ Map Reduce
 - ▶ Concept and Examples
 - ▶ Implementation: Hadoop
- ▶ Predictive Modelling
 - ▶ Data Pre-processing
 - ▶ Regression
 - ▶ Model Selection

Applications of Big Data

Lecture 7 - Map Reduce Part I

Nick Tothill

Spring 2019

Parallel Computing

Scalability and Parallelism

- ▶ Scaling Up (Vertical)
 - ▶ The older model
 - ▶ Upgrade existing hardware
 - ▶ faster CPU, larger memory and disks
- ▶ Scaling Out (Horizontal)
 - ▶ The newer model
 - ▶ Add new computers to the existing cluster
 - 1000 cheaper computers may work better than 1 powerful one

Similarities to the evolution of storage: High-spec storage; RAID; JBOD

Parallel Computing

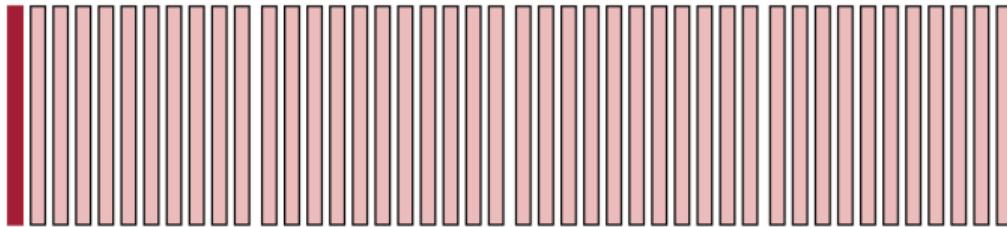
- ▶ Large problem can be divided into smaller ones
- ▶ Computations can be carried out simultaneously for each smaller task
- ▶ Some parallel computing done in your CPU (SIMD)
- ▶ Parallel Computing depends on:
 1. your algorithm
 2. your code
 - ▶ Is the code parallel?
 - ▶ Is it parallelisable? (Language. . .)
 - ▶ Can you rewrite it?

Scalability and Algorithm

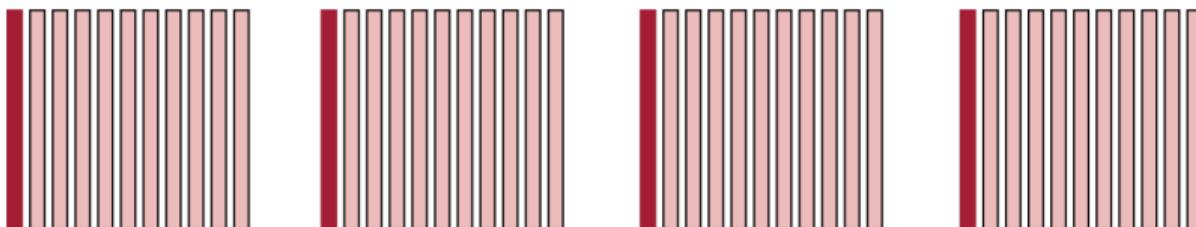
- ▶ Complexity of Algorithms
 - ▶ $O(m^N)$: exponential time
 - ▶ Infeasible: exponential increase with N (number of items)
 - ▶ $O(N^m)$: polynomial time
 - ▶ Feasible: still value of m could make a difference
 - ▶ Not good enough for big data applications
 - ▶ $O(N)$: linear time (polynomial with $m = 1$)
 - ▶ Streaming operations: one pass with input data
- ▶ Algorithmic constraints for big data
 - ▶ linear or sub-linear time complexity algorithms
 - ▶ Utilise data parallelism
 - ▶ $O(N) \rightarrow \frac{O(N)}{k}$ for k computers

Parallel Processing ($t = 1$)

Single Processor

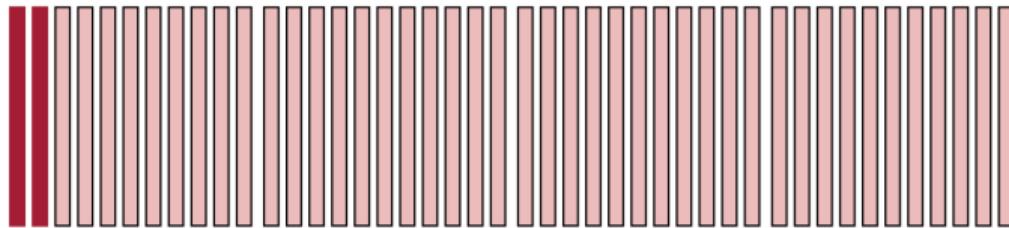


Multiple Processors (k=4)

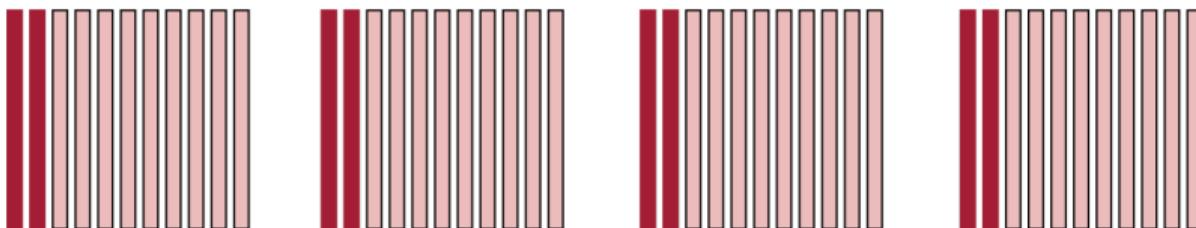


Parallel Processing ($t = 2$)

Single Processor

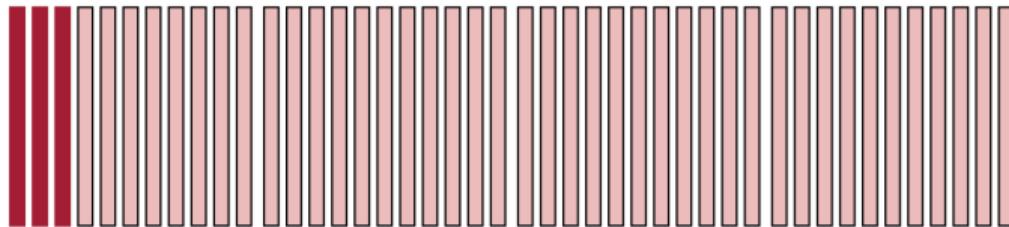


Multiple Processors (k=4)

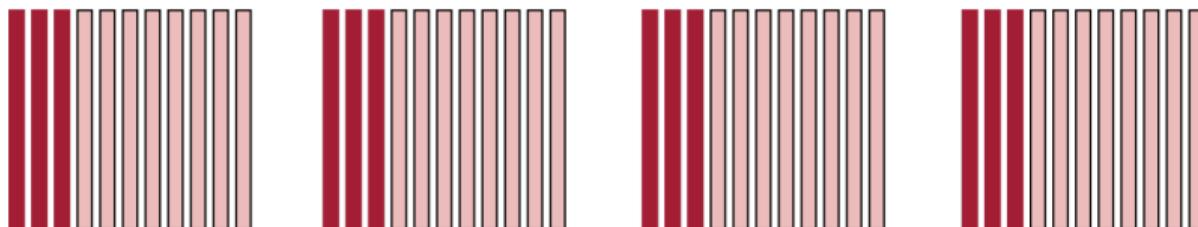


Parallel Processing ($t = 3$)

Single Processor

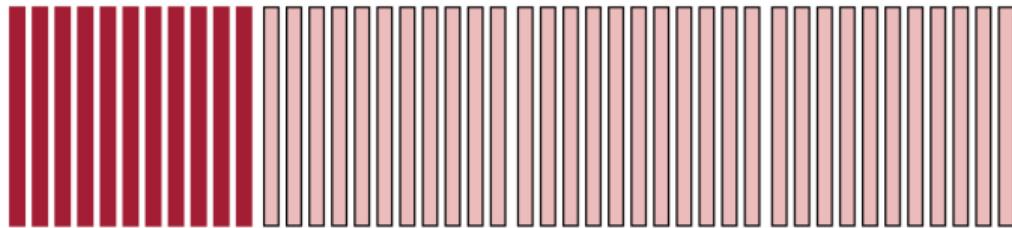


Multiple Processors (k=4)

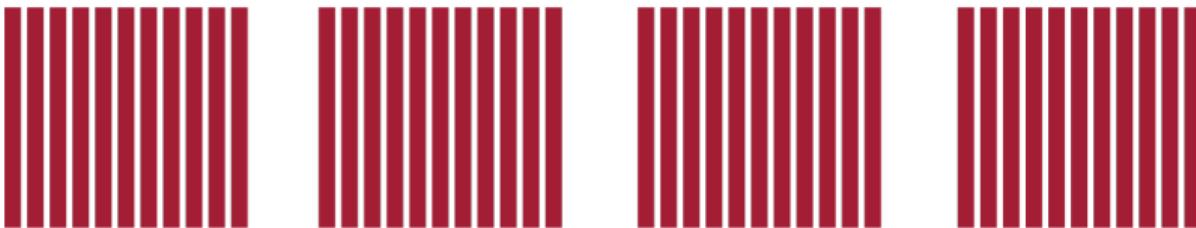


Parallel Processing ($t = N/K$)

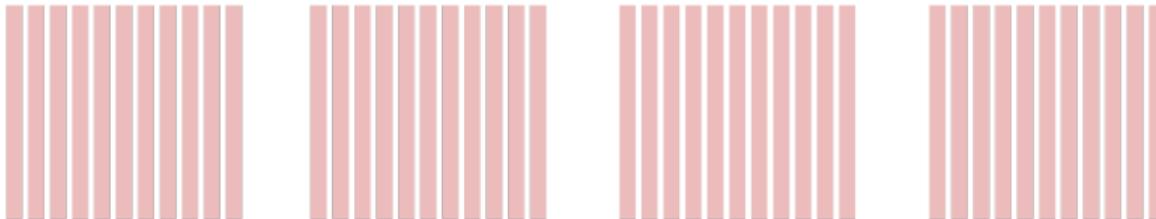
Single Processor



Multiple Processors (k=4)



Parallel Processing Examples 1: Word Count



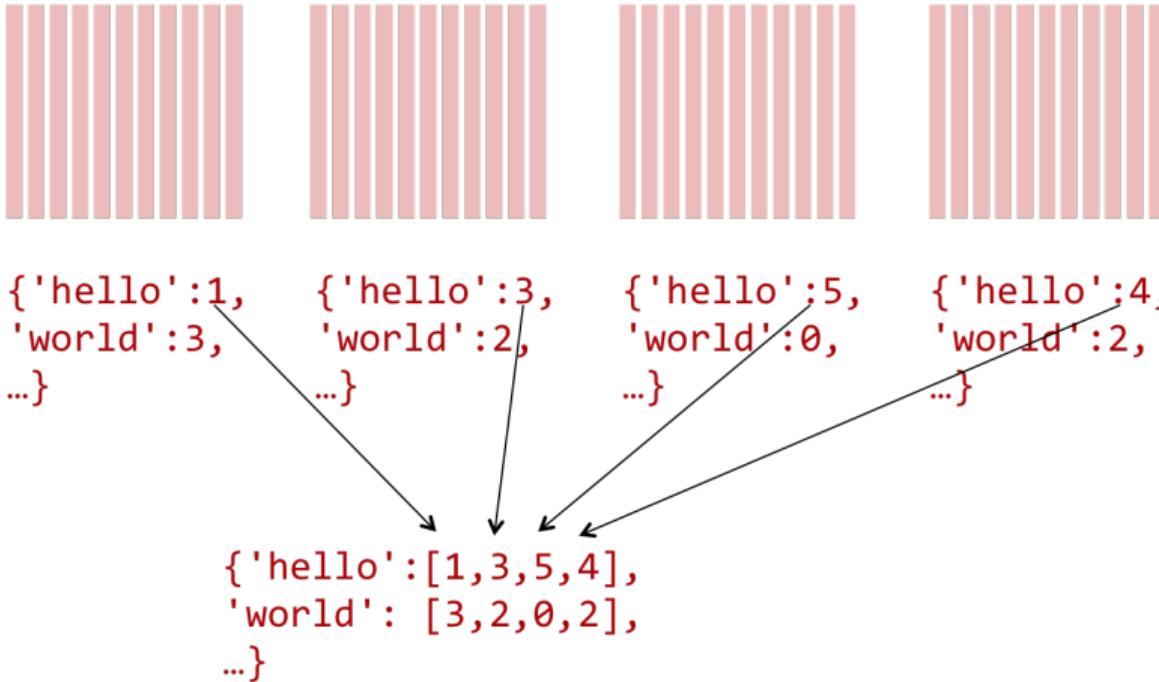
```
{'hello':1,  
'world':3,  
...}
```

```
{'hello':3,  
'world':2,  
...}
```

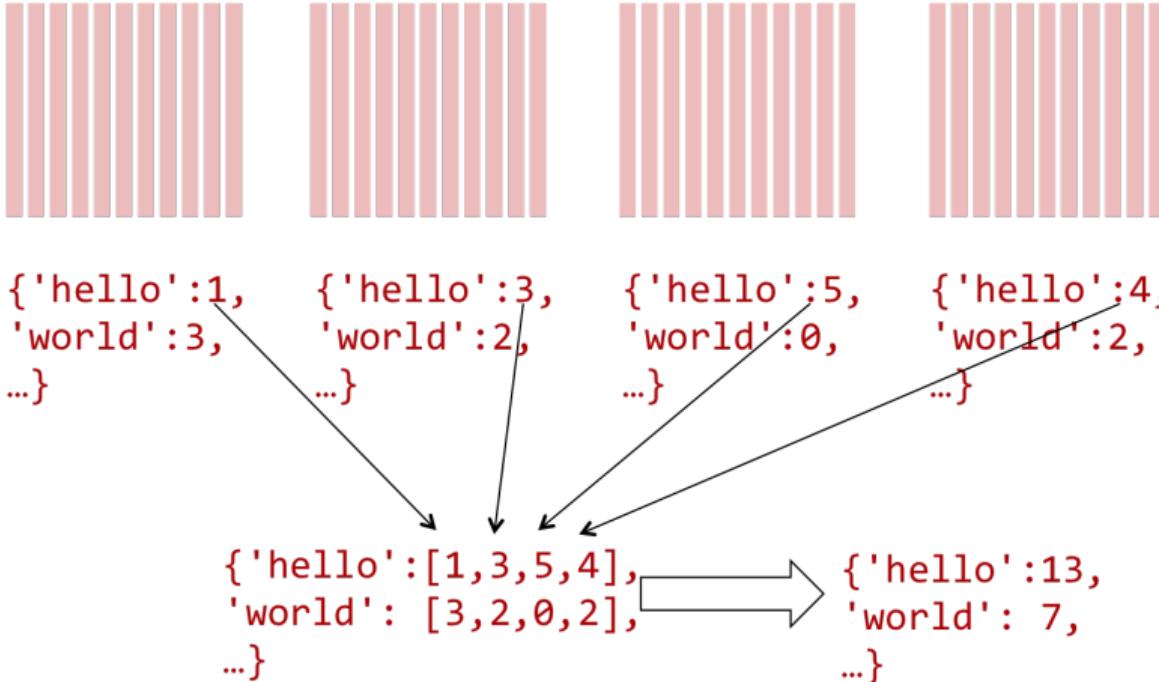
```
{'hello':5,  
'world':0,  
...}
```

```
{'hello':4,  
'world':2,  
...}
```

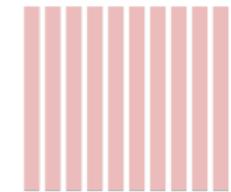
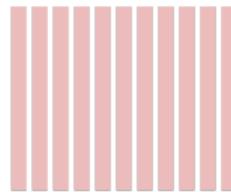
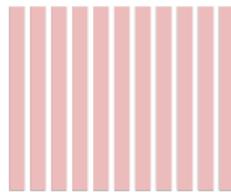
Parallel Processing Examples 1: Word Count



Parallel Processing Examples 1: Word Count



Parallel Processing Examples 2: Association Mining



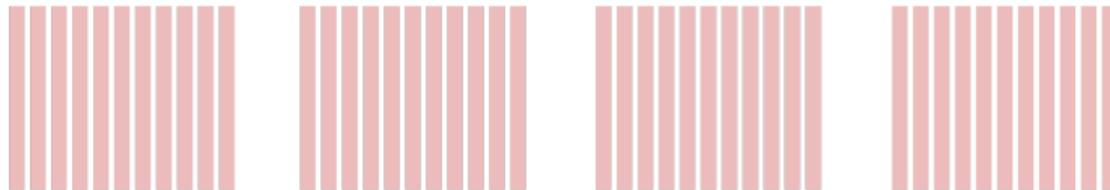
$\{(p_1, p_2): 9,$
 $(p_3, p_4): 5,$
 $(p_1, p_3): 0,$
 $\dots\}$

$\{(p_1, p_2): 12,$
 $(p_3, p_4): 8,$
 $(p_1, p_3): 1,$
 $\dots\}$

$\{(p_1, p_2): 8,$
 $(p_3, p_4): 10,$
 $(p_1, p_3): 0,$
 $\dots\}$

$\{(p_1, p_2): 11,$
 $(p_3, p_4): 7,$
 $(p_1, p_3): 1,$
 $\dots\}$

Parallel Processing Examples 2: Association Mining



```

{(p1, p2):9,   {(p1, p2):12,   {(p1, p2):8,   {(p1, p2):11,
(p3, p4): 5,   (p3, p4): 8,   (p3, p4): 10,  (p3, p4): 7,
(p1, p3): 0,   (p1, p3): 1,    (p1, p3): 0,   (p1, p3): 1,
...}           ...}           ...}           ...}

```

```

{(p1, p2): [9,12,8,11],           {(p1, p2): 40,
(p3, p4): [5,8,10,7],           (p3, p4): 30,
(p1, p3): [0,1,0,1],           (p1, p3): 2,
...}                                ...}

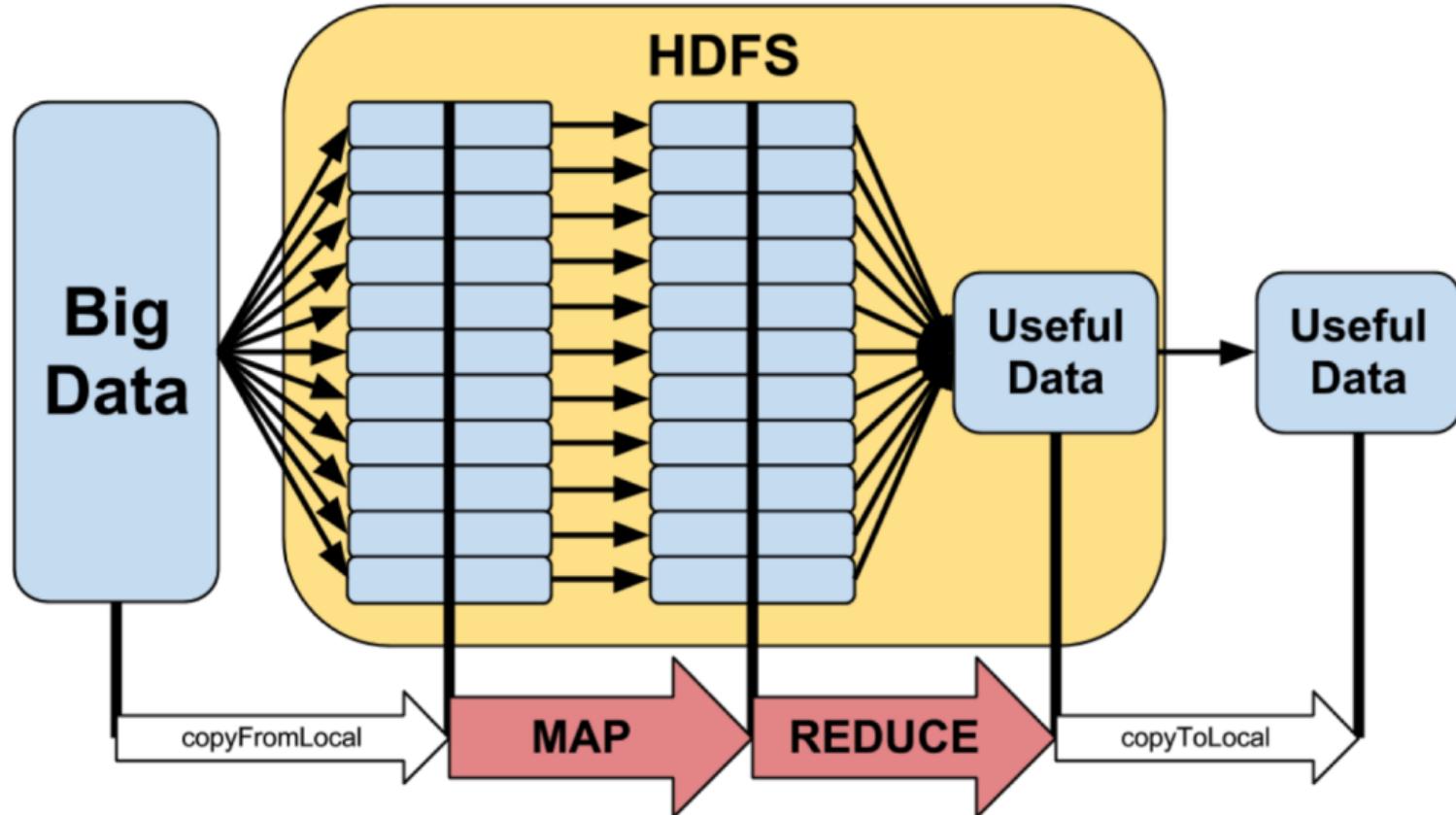
```

MapReduce

Overview of Framework - The MapReduce Model

- ▶ MapReduce is a high-level programming model for large-scale parallel data processing
- ▶ It provides a conceptual framework and has different implementations
 - ▶ Original implementation by Google in 2004
 - ▶ Free implementation: Apache Hadoop
- ▶ The MapReduce Model
 - ▶ Input & Output: a set of records and key/value pairs
 - ▶ Two important user-defined functions
 - ▶ Mapper: input->intermediate set of pairs
 - ▶ Reducer: intermediate set of pairs -> output

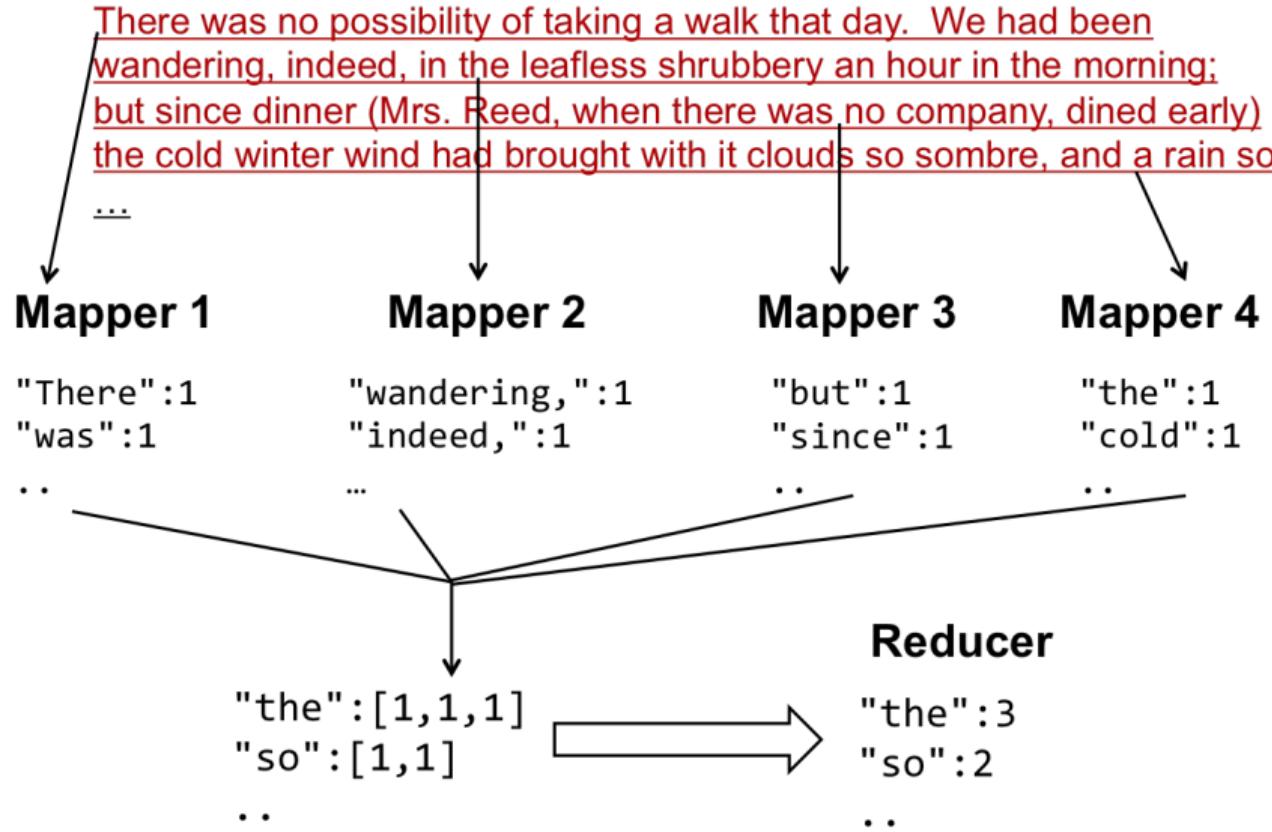
Illustration of MapReduce Framework



Mapper and Reducer

- ▶ Mapper
 - ▶ Input: item1, item2, ...
 - ▶ Output: (key1, value1)
 - ▶ Processes input items and produces set of intermediate pairs
 - ▶ System applies the mapper function in parallel to all input pairs
- ▶ Reducer
 - ▶ Input: (key1,[value1, value2, .]), (key2, [value1, value2, .]), ...
 - ▶ Output: (key1, out_value1), (key2, out_value2), ...
 - ▶ System groups all pairs with the same intermediate key and passes to the reducer function
 - ▶ Reducer function produces a set of merged output values (one for each key)

The Word Count Example



Hadoop Streaming for MapReduce

Apache Hadoop Overview

Apache Hadoop is an open-source MapReduce framework

- ▶ Hadoop Common: libraries and utilities that support various Hadoop modules
 - Hadoop Streaming: create and run MapReduce jobs with any programming language
- ▶ Hadoop Distributed File System (HDFS): a distributed file system with high-throughput access to large data
- ▶ Hadoop YARN: a framework for job scheduling and cluster resource management
- ▶ Hadoop MapReduce: an implementation of the MapReduce model for parallel processing of large data sets

The Hadoop Eco-System

- ▶ A family of Hadoop based softwares for big data applications

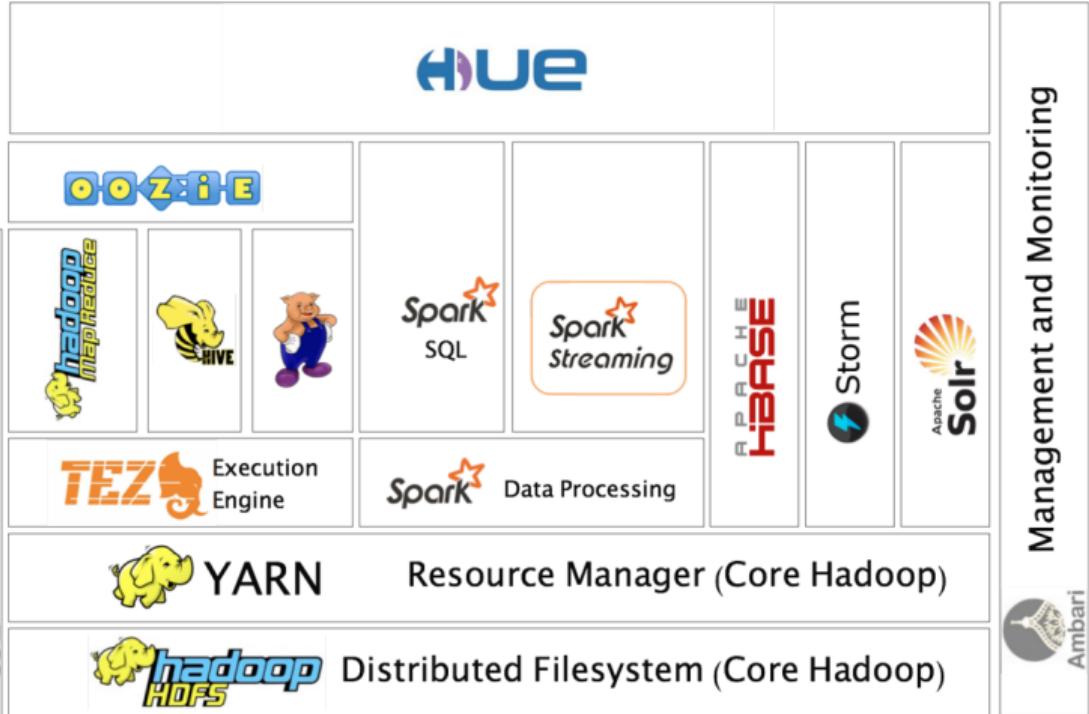
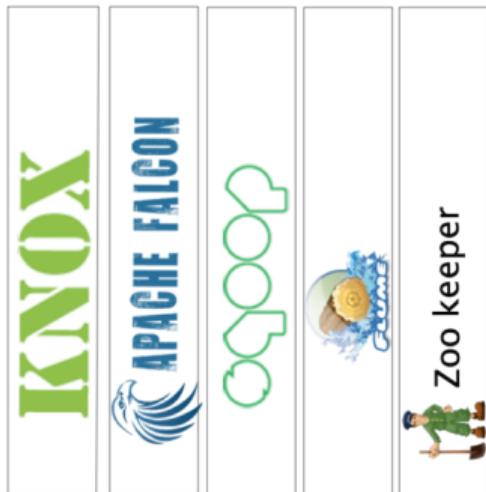
Other MapReduce frameworks based on Hadoop

- ▶ CDH, MapR, Hortonworks, ...

Hadoop Eco-system



Hadoop Ecosystem



Management and Monitoring

Hadoop Distributed File System (HDFS)

HDFS: a distributed file system for Hadoop software

- ▶ High reliability and low cost (run on commodity hardware)
- ▶ Provides high-throughput access to big data

HDFS has a master/slave architecture

- ▶ NameNode: a master server to manage the filesystem and regulates access to files by clients
- ▶ DataNodes: actual nodes where data are stored and managed
- ▶ HDFS can be configured on a single node or a cluster
 - ▶ An HDFS cluster usually consists of a single NameNode and multiple DataNodes
 - ▶ Data are replicated in copies to avoid single point of failure
 - ▶ Large files are broken into smaller blocks to save on DataNodes

Some HDFS Commands

```
hdfs dfs -put /localfs/source/path /hdfs/destination/path  
hdfs dfs -get /hdfs/source/path /localfs/destination/path  
hdfs dfs -ls /hdfs/path  
hdfs dfs -cat /hdfs/path  
hdfs dfs -mkdir /hdfs/path  
hdfs dfs -rm [rm options] /hdfs/path
```

or replace `hdfs dfs` by `hadoop fs` for the same operation. These are used to put/get/list/show/create dir/remove dir/files in HDFS, very similar to their Linux counterparts.

See <https://dzone.com/articles/top-10-hadoop-shell-commands> for 10 commonly used Hadoop commands to manage HDFS.

WSU SCEM Big Data Facility

- ▶ Hardware: PowerEdge R730 Server + Dell Storage MD1400 Rackmount + NVIDIA Quadro K6000 12GB
- ▶ Cluster structure: 6 nodes (head + 5 slaves) each with 1x8TB disc
- ▶ Apps: Hadoop 2.7.3, YARN, Spark, MySQL, MongoDB and many more
- ▶ Entry point: `hadoop-01.scem.westernsydney.edu.au`. *You need to create an account to access it.*

All the rest of the unit relies on this big data facility.

Connect to SCEM Big Data Facility

- ▶ Linux like systems: ssh
- ▶ Windows systems: putty + Xming

All testing data are hosted in

hdfs://hadoop-01-149-21-172.scem.uws.edu.au:9000/users/ugbigdata/

Hadoop Streaming

What is Hadoop Streaming

- ▶ Hadoop streaming is a utility that comes with the Hadoop distribution.
- ▶ The utility allows you to create and run Map/Reduce jobs with *any executable or script* (including python of course!) as the mapper and/or the reducer.

How Streaming Works

- ▶ Mapper task takes a line from files or specified file in input directory and feed it through *stdin* into a mapper executable. Many mappers will be executed in parallel;
- ▶ Mappers generate key-value pair to *stdout* in parallel;
- ▶ Reducer task launch reducer executable and feed lines from *stdout* (emitted by mappers) and feed into reducer through *stdin* to produce results, i.e. key-value pairs.

Syntax

```
hadoop jar hadoop-streaming-2.7.3.jar \
    -input myInputDirs \
    -output myOutputDir \
    -mapper mapper_executable \
    -reducer reducer_executable
```

In the above example, both the mapper and the reducer are executables that read the input from stdin (**line by line**) and emit the output to stdout. The utility will create a Map/Reduce job, submit the job to an appropriate cluster, and monitor the progress of the job until it completes.

By default, the prefix of a line up to the first tab character is the key and the rest of the line (excluding the tab character) will be the value. If there is no tab character in the line, then entire line is considered as key and the value is null. However, this can be customized.

Source: <http://hadoop.apache.org/docs/r2.7.3/hadoop-streaming/HadoopStreaming.html>

Example: Counting Words in Movie Reviews

Some movie review examples

- ▶ 165_7.txt: One of my favorite scenes is at the beginning when guests on a private yacht decide to take an impromptu swim - in their underwear! Rather risqué for 1931!
- ▶ 111_10.txt: Before Dogma 95: when Lars used movies as art, not just a story. A beautiful painting about love and death. This is one of my favorite movies of all time. The color... The music... Just perfect.

MapReduce by Hadoop Streaming for this simple task

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-input /users/ugbigdata/examples/smallmoviedata/ \  
-mapper /usr/bin/cat \  
-reducer /usr/bin/wc \  
-output /users/ugbigdata/examples/alloutput/movieoutput/
```

Run the job:

```
packageJobJar: [/tmp/hadoop-unjar5010589065423160824/] [] /tmp/streamjob2813071620167732194.jar tmpDir=null
09:45:23 INFO client.RMProxy: Connecting to ResourceManager at hadoop-01.scem.westernsydney.edu.au/137.154.151.183:8032
09:45:23 INFO client.RMProxy: Connecting to ResourceManager at hadoop-01.scem.westernsydney.edu.au/137.154.151.183:8032
09:45:23 INFO mapred.FileInputFormat: Total input paths to process : 9
09:45:23 INFO mapreduce.JobSubmitter: number of splits:9
09:45:23 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1501473534110_0008
09:45:24 INFO impl.YarnClientImpl: Submitted application application_1501473534110_0008
09:45:24 INFO mapreduce.Job: The url to track the job: http://hadoop-01.scem.westernsydney.edu.au:8088/proxy/application_1501473534110_0008
09:45:24 INFO mapreduce.Job: Running job: job_1501473534110_0008
09:45:29 INFO mapreduce.Job: Job job_1501473534110_0008 running in uber mode : false
09:45:29 INFO mapreduce.Job: map 0% reduce 0%
09:45:33 INFO mapreduce.Job: map 11% reduce 0%
09:45:34 INFO mapreduce.Job: map 100% reduce 0%
09:45:37 INFO mapreduce.Job: map 100% reduce 100%
Some output omitted.
```

Check the output:

```
-bash-4.2$ hdfs dfs -cat /users/ugbigdata/examples/alloutput/movieoutput/part-00000
9      1949    11271
```

We got **1949** words in those reviews (only a tiny subset of all the reviews).

Checking

Check the result by using mimicking the Hadoop streaming locally:

```
-bash-4.2$ ls *.txt | xargs -I % bash -c "cat % <(echo)" | sort | wc
```

The above command line is actually mimicking what Hadoop Streaming is doing at the backend.

The result is:

```
9      1949    11262
```

Found **1949** words too! The sort command in the pipe reveals key-value pairs generated by mappers are **sorted by keys** by Hadoop streaming utility.

Monitoring jobs

Web browser based job monitoring interface is provided.

Go to `http://hadoop-01.scem.westernsydney.edu.au:8088` to check how jobs are progressing.

MapReduce Job job_1501473534110_0019 - Mozilla Firefox

about:sessionrestore x RUNNING Applications x MapReduce Job job_1501473534110_0019 +

hadoop-01.scem.westernsydney.edu.au:8088/proxy/application_1501473534110_0019/mapreduce/job/jo | C Search

Logged in as: drwho

hadoop

MapReduce Job job_1501473534110_0019

job Overview

Job Name: streamjob4265117812317449139.jar
State: RUNNING
Uberized: false
Started: Thu Aug 03 10:24:03 AEST 2017
Elapsed: 8mins, 54sec

ApplicationMaster

Attempt Number	Start Time	Node	Logs
1	Thu Aug 03 10:24:00 AEST 2017	hadoop-03-149-21-172.scem.uws.edu.au:8042	logs

Task Type

	Progress	Total	Pending	Running	Complete
Map	<div style="width: 100%;">1000</div>	459	37	498	
Reduce	<div style="width: 1%;">1</div>	0	1	0	

Attempt Type

	New	Running	Failed	Killed	Successful
Maps	459	37	0	0	504
Reduces	0	1	0	0	0

ssh -Y hadoop-01 IPython: yguo@... yguo@skywalk...

17/08/03 10:27:25 INFO mapreduce.Job: map 18% reduce 6%
17/08/03 10:28:01 INFO mapreduce.Job: map 19% reduce 6%
17/08/03 10:28:06 INFO mapreduce.Job: map 20% reduce 6%
17/08/03 10:28:11 INFO mapreduce.Job: map 21% reduce 6%
17/08/03 10:28:17 INFO mapreduce.Job: map 21% reduce 7%
17/08/03 10:28:21 INFO mapreduce.Job: map 22% reduce 7%
17/08/03 10:28:27 INFO mapreduce.Job: map 23% reduce 7%
17/08/03 10:28:31 INFO mapreduce.Job: map 24% reduce 7%
17/08/03 10:28:36 INFO mapreduce.Job: map 24% reduce 8%
17/08/03 10:28:37 INFO mapreduce.Job: map 25% reduce 8%
17/08/03 10:28:43 INFO mapreduce.Job: map 26% reduce 8%
17/08/03 10:28:45 INFO mapreduce.Job: map 27% reduce 8%
17/08/03 10:28:46 INFO mapreduce.Job: map 27% reduce 9%
17/08/03 10:28:51 INFO mapreduce.Job: map 28% reduce 9%
17/08/03 10:29:03 INFO mapreduce.Job: map 29% reduce 9%
17/08/03 10:29:08 INFO mapreduce.Job: map 30% reduce 9%
17/08/03 10:29:12 INFO mapreduce.Job: map 30% reduce 10%
17/08/03 10:29:15 INFO mapreduce.Job: map 31% reduce 10%
17/08/03 10:29:17 INFO mapreduce.Job: map 32% reduce 10%
17/08/03 10:29:25 INFO mapreduce.Job: map 33% reduce 10%
17/08/03 10:29:27 INFO mapreduce.Job: map 33% reduce 11%
17/08/03 10:29:29 INFO mapreduce.Job: map 34% reduce 11%
17/08/03 10:30:13 INFO mapreduce.Job: map 35% reduce 11%
17/08/03 10:30:22 INFO mapreduce.Job: map 36% reduce 12%
17/08/03 10:30:38 INFO mapreduce.Job: map 37% reduce 12%
17/08/03 10:30:54 INFO mapreduce.Job: map 38% reduce 12%
17/08/03 10:30:57 INFO mapreduce.Job: map 39% reduce 12%
17/08/03 10:31:04 INFO mapreduce.Job: map 40% reduce 13%
17/08/03 10:31:17 INFO mapreduce.Job: map 41% reduce 13%
17/08/03 10:31:21 INFO mapreduce.Job: map 41% reduce 14%
17/08/03 10:31:25 INFO mapreduce.Job: map 42% reduce 14%
17/08/03 10:31:31 INFO mapreduce.Job: map 43% reduce 14%
17/08/03 10:31:41 INFO mapreduce.Job: map 44% reduce 14%
17/08/03 10:31:49 INFO mapreduce.Job: map 45% reduce 15%
17/08/03 10:32:02 INFO mapreduce.Job: map 46% reduce 15%
17/08/03 10:32:10 INFO mapreduce.Job: map 47% reduce 15%
17/08/03 10:32:14 INFO mapreduce.Job: map 48% reduce 15%
17/08/03 10:32:15 INFO mapreduce.Job: map 48% reduce 16%
17/08/03 10:32:31 INFO mapreduce.Job: map 49% reduce 16%
17/08/03 10:32:58 INFO mapreduce.Job: map 50% reduce 16%
17/08/03 10:32:57 INFO mapreduce.Job: map 51% reduce 16%
17/08/03 10:33:02 INFO mapreduce.Job: map 51% reduce 17%
17/08/03 10:33:06 INFO mapreduce.Job: map 52% reduce 17%
17/08/03 10:33:15 INFO mapreduce.Job: map 53% reduce 17%
17/08/03 10:33:22 INFO mapreduce.Job: map 53% reduce 18%
17/08/03 10:33:25 INFO mapreduce.Job: map 54% reduce 18%

Following topics

Future topics

- ▶ Map Reduce
 - ▶ More Examples
- ▶ Predictive Modelling
 - ▶ Data Pre-processing
 - ▶ Regression
 - ▶ Model Selection

Applications of Big Data

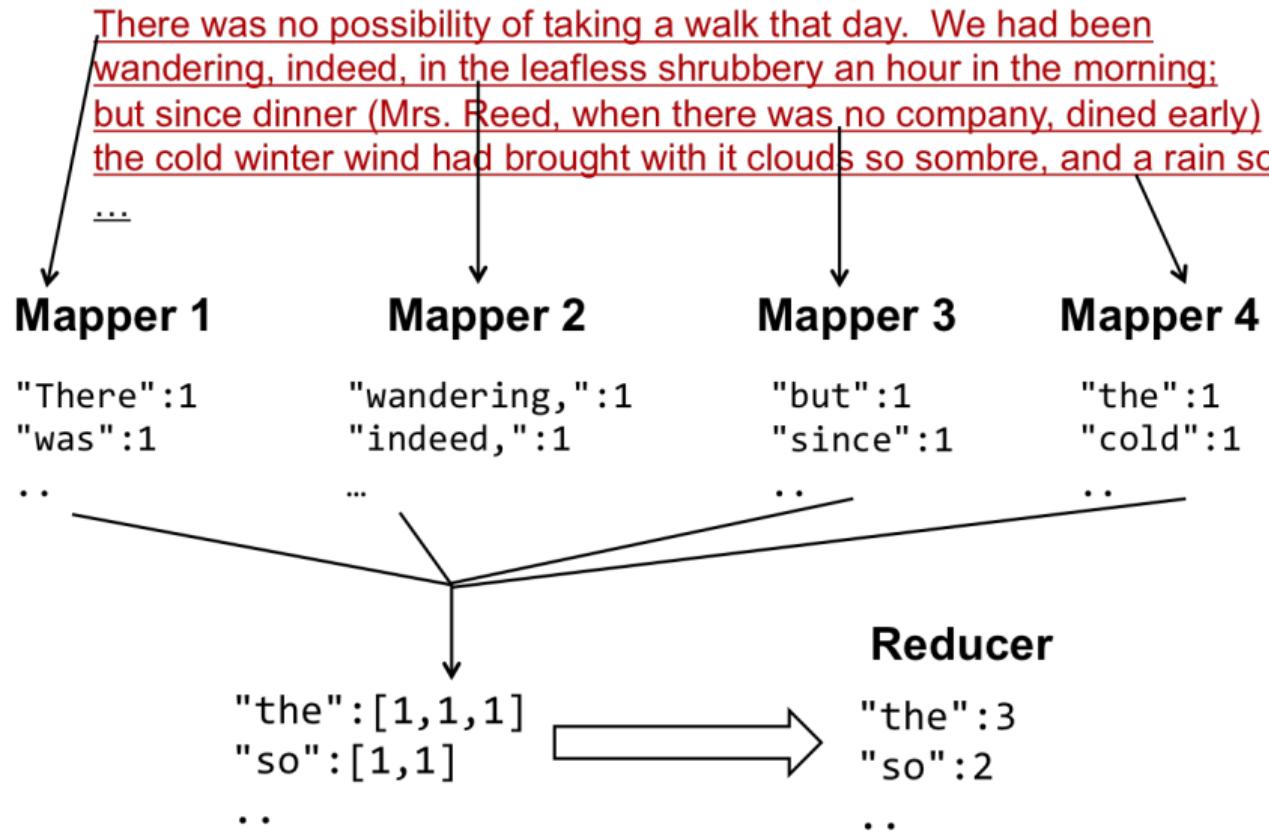
Lecture 8 - Map Reduce Part II

Nick Tothill

Spring 2019

MapReduce Model

Recap: The Word Count Example



Recap: Hadoop Streaming for Words Counting

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-input /users/ugbigdata/examples/smallmoviedata/ \  
-mapper /usr/bin/cat \  
-reducer /usr/bin/wc \  
-output /users/ugbigdata/examples/alloutput/movieoutput/  
  
-bash-4.2$ hdfs dfs -cat \  
/users/ugbigdata/examples/alloutput/movieoutput/part-00000  
9      1949    11271
```

Got the counts right but *not exactly* what we want: word1: count, word2: count,...

Hadoop Streaming with Python for Word-Counting

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-mapper hadoopmapper_wcnt.py \  
-reducer hadoopproducer_wcnt.py \  
-input /users/ugbigdata/examples/moviedata/ \  
-output /users/ugbigdata/examples/alloutput/TFmovieoutput \  
-file ~/bigdata/hadoopmapper_wcnt.py \  
-file ~/bigdata/hadoopproducer_wcnt.py
```

Mapper Code: hadoopmapper_wc.py

```
#! /usr/bin/env python
import sys
import re
word_count = 0
WORD_RE = re.compile(r"\w'"]+")
res = dict()
for line in sys.stdin:
    for word in WORD_RE.findall(line):
        if len(word) > 0:
            word_count += 1
            if word.lower() in res.keys():
                res[word.lower()] += 1
            else:
                res[word.lower()] = 1
for key in res.keys():
    print("\t".join([key,str(res[key])]))
```

☞ Things to note:

1. The first line:

```
#! /usr/bin/env python  
is not normal comment  
line!
```

2. Reading from *stdin* distributed by *Hadoop Streaming utility*:

```
for line in sys.stdin:  
    ...
```

3. One mapper gets **one line** from *stdin*!

Reducer Code: hadoopproducer_wcnt.py

```
#! /usr/bin/env python
import sys
result = dict()
for line in sys.stdin:
    line = line.strip()
    word, count = line.split("\t")
    count = int(count)
    if word not in result.keys():
        result[word] = count
    else:
        result[word] += count

for key in result.keys():
    print("\t".join([key, str(result[key])]))
```

Things to note:

1. The first line:

```
#! /usr/bin/env python
```

is **not normal comment line!**

2. Reading from *stdin* produced by *mappers*:

```
for line in sys.stdin:
```

...

3. One reducer gets **multiple lines** from *stdin*!

Workflow

1. Hadoop streaming utilities take one file out of `/users/ugbigdata/examples/moviedata/` directory and feed one line to a mapper;
2. Step1 is parallelised across available work nodes depending on split size (more in `mapred-default.xml`);
3. Hadoop streaming utilities launch reducer(s) and feed it/them with collected output from mappers *after sorting keys*;
4. Hadoop streaming utilitie collect reducer(s)'s results and save them into specified output folder.

Data flow through `stdin` and `stdout` pipes.



Action!

Omit many lines...

```
15:26:40 INFO mapreduce.Job: map 0% reduce 0%
15:26:44 INFO mapreduce.Job: map 33% reduce 0%
15:26:45 INFO mapreduce.Job: map 56% reduce 0%
15:26:49 INFO mapreduce.Job: map 100% reduce 0%
15:26:50 INFO mapreduce.Job: map 100% reduce 100%
15:26:50 INFO mapreduce.Job: Job job_1501473534110_0009 completed successfully
15:26:50 INFO mapreduce.Job: Counters: 50
```

Omit many lines...

```
File Output Format Counters
      Bytes Written=7265
INFO streaming.StreamJob: Output directory: /users/ugbigdata/examples/alloutput/TFmovieoutput
```

Check out results in /users/ugbigdata/examples/alloutput/TFmovieoutput folder:

```
hdfs dfs -ls /users/ugbigdata/examples/alloutput/TFmovieoutput
```

Found 2 items

```
/users/ugbigdata/examples/alloutput/TFmovieoutput/_SUCCESS  
/users/ugbigdata/examples/alloutput/TFmovieoutput/part-00000
```

Display what is in part-00000:

```
hdfs dfs -cat /users/ugbigdata/examples/alloutput/TFmovieoutput/part-00000
```

It produces a long list, part of it looks like

as	13	Exactly what we want!
enable	1	This is actually called <i>term frequency (TF)</i> in text mining, which accounts for
goodness	1	how many times a term (word) is used in a collection of documents.
believable	1	
film	20	

Some Handy Tricks in Hadoop Streaming with Python

Argument Passing

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-files path/to/mapper.py, path/to/reducer.py \  
-mapper 'python mapper.py arg1 arg2' \  
-reducer 'python reducer.py arg1 arg2' \  
-input HDFS_source_path \  
-output HDFS_destination_path
```

-files option (a generic option) supplies files used in Hadoop Streaming. *Generic options have to come at the start of the Hadoop streaming command.*

Source: <http://hadoop.apache.org/docs/r2.7.3/hadoop-streaming/HadoopStreaming.html>

Python Source

Arguments parsing

Add the following few lines in the mapper and reducer source:

```
import sys  
arg1 = sys.argv[1]  
arg2 = sys.argv[2]  
for line in sys.stdin:  
    ...
```

Example: word counting (mr_one.py)

```
#!/usr/bin/env python
import sys
import re
class CountMapReduce():
    def __init__(self):
        self.current_item = None
        self.count_sum = 0
    def __output(self):
        if self.current_item:
            print self.current_item, '\t', self.count_sum
    def map(self, line):
        WORD_RE = re.compile(r"\w+")
        res = dict()
        for word in WORD_RE.findall(line):
            if len(word) > 0:
                if word.lower() in res.keys():
                    res[word.lower()] += 1
                else:
                    res[word.lower()] = 1
        for key in res.keys():
            print("\t".join([key, str(res[key])]))
    def reduce(self, item, count):
        if self.current_item != item:
            self.__output()
            self.current_item = item
            self.count_sum = 0
        self.count_sum += int(count)
    def reduce_end(self):
        #Print the last item count sum.
        self.__output()

if __name__ == '__main__':
    mr_flag = sys.argv[1]
    mapreduce = CountMapReduce()
    if mr_flag == 'map':
        for line in sys.stdin:
            line = line.strip()
            mapreduce.map(line)
    elif mr_flag == 'reduce':
        for line in sys.stdin:
            line = line.strip()
            item, count = line.split('\t', 1)
            mapreduce.reduce(item, count)
        mapreduce.reduce_end()
```

Hadoop Streaming Commands for Using mr_one.py

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-files ~/bigdata/mr_one.py \  
-mapper 'python mr_one.py map' \  
-reducer 'python mr_one.py reduce' \  
-input /users/ugbigdata/examples/moviedata/ \  
-output /users/ugbigdata/examples/alloutput/mrnewcnt \  
-numReduceTasks 5
```

mr_one.py is made available to the utility by -files option. There is one extra in above: -numReduceTasks, which is specified as 5, i.e. 5 reducers will be launched.

Passing Files

Read file

```
export HLIB=/local/apps/hadoop/share/hadoop/tools/lib  
hadoop jar ${HLIB}/hadoop-streaming-2.7.3.jar \  
-files path/to/mapper.py, path/to/reducer.py, path/to/userfile \  
-other similar options omitted
```

The `userfile` will be available to mappers and reducers just as a local file. Access it as

```
fid = open('userfile', 'r')  
lines = fid.readlines()  
. . .
```

Get File Names Passed to Mappers

```
import os  
file_name = os.getenv('map_input_file')
```

More MapReduce Examples

Inverse Word Count

- ▶ Certain words occur frequently in text
 - e.g. "the", "a", "an", "I", "you", "and", "but", "it", ...
- ▶ Inverse Document Count (idc) is a useful metric to measure the commonality of a word
 - ▶ defined as the number of documents a word appears in
 - ▶ the larger the idc value, the more common the word, the less meaningful it is
 - ▶ How meaningful are words like "the", "a", "an" in text?
 - ▶ How about "excruciating", "ecstatic"?
- ▶ Word/term frequency (tf) and inverse document frequency (idf) are useful for text analysis

Inverse Word Count

This film has a special place in my heart, as when I caught it the first time, I was teaching adult literacy.

doc1

If you can enjoy a story of real people and real love - this is a winner

doc2

My only complaint was that in the beginning it was kind of slow and it took awhile to get to the basis of things. Other than that it was great.

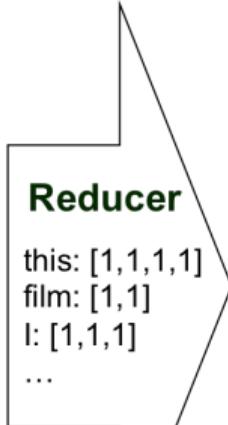
doc3



This: 1
film: 1
I: 1
...

If: 1
this: 1
real: 1
...

My: 1
was: 1
that: 1
...



this: 4
film: 2
I: 3
...

o o o

Inverse Word Count implementation

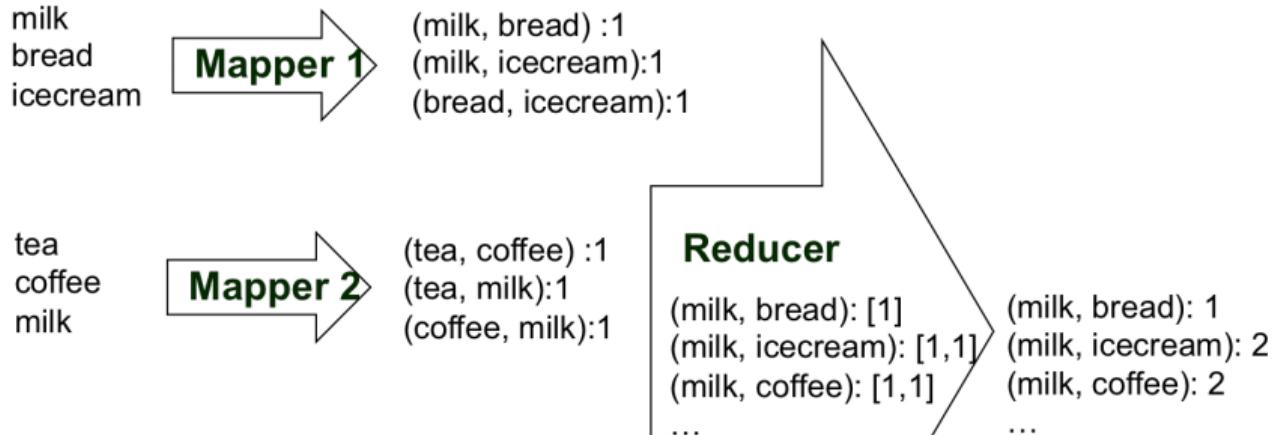
Use `mr_one.py` skeleton:

```
import re

def map(self, line):
    allWords = {}
    wordsList = re.split('\s|.|;|!|\?|"|--|\(|\)', line)
    for tok in wordsList:
        if len(tok)>0 and tok not in allWords:
            allWords[tok] = 1
    for tok in allWords:
        print("\t".join([tok,"1"]))
```

The same reducer as in `mr_one.py` is applicable.

Co-Occurring Products



o o o

Co-Occurring Products Implementation

Use mr_one.py skeleton:

```
def map(self, line):
    products = line.strip().split(',')
    # generate all product pairs and send to reducer
    for i in range(len(products)-1):
        for j in range(i+1,len(products)):
            # ('milk', 'bread')=('bread', 'milk'), so?
            if products[i]<products[j]:
                print products[i],",",products[j],"\t1"
            else:
                print products[i],",",products[j],"\t1"
```

The same reducer as in mr_one.py is still applicable!

Calculate simple statistics - mean of multivariate observations

Data with D variables are in this format:

observation1:observation2:...:observation121

observation122:observation123:...

...

In each observation, variables are organised as:

var1,var2,var3,...,varD

Pseudo code for mapper and reducer:

Mapper:

```
split to K observations by ":"  
for each observation:  
    split by ","  
    accumulate summation  
print K, '\t', mean of K obs
```

Reducer:

```
m = 0; N = 0  
for each key-value pair:  
    split to K and mean of K obs  
    m = m + K * mean  
    N = N + K  
print m/N
```

Calculate simple statistics - COVAR of multivariate observations

- ▶ COVAR: covariance matrix of multivariate observations, written as C

$$C = \sum_{i=1}^N \tilde{\mathbf{x}}_i \tilde{\mathbf{x}}_i^\top$$

where

$$\tilde{\mathbf{x}}_i = \mathbf{x}_i - \mathbf{m}$$

and \mathbf{m} is the mean of all observations.

- ▶ Assume mean has been calculated by above procedure.

Following topics

Future topics

Predictive Modelling

- ▶ Data Pre-processing
- ▶ Regression
- ▶ Model Selection
- ▶ Deep Learning

Applications of Big Data

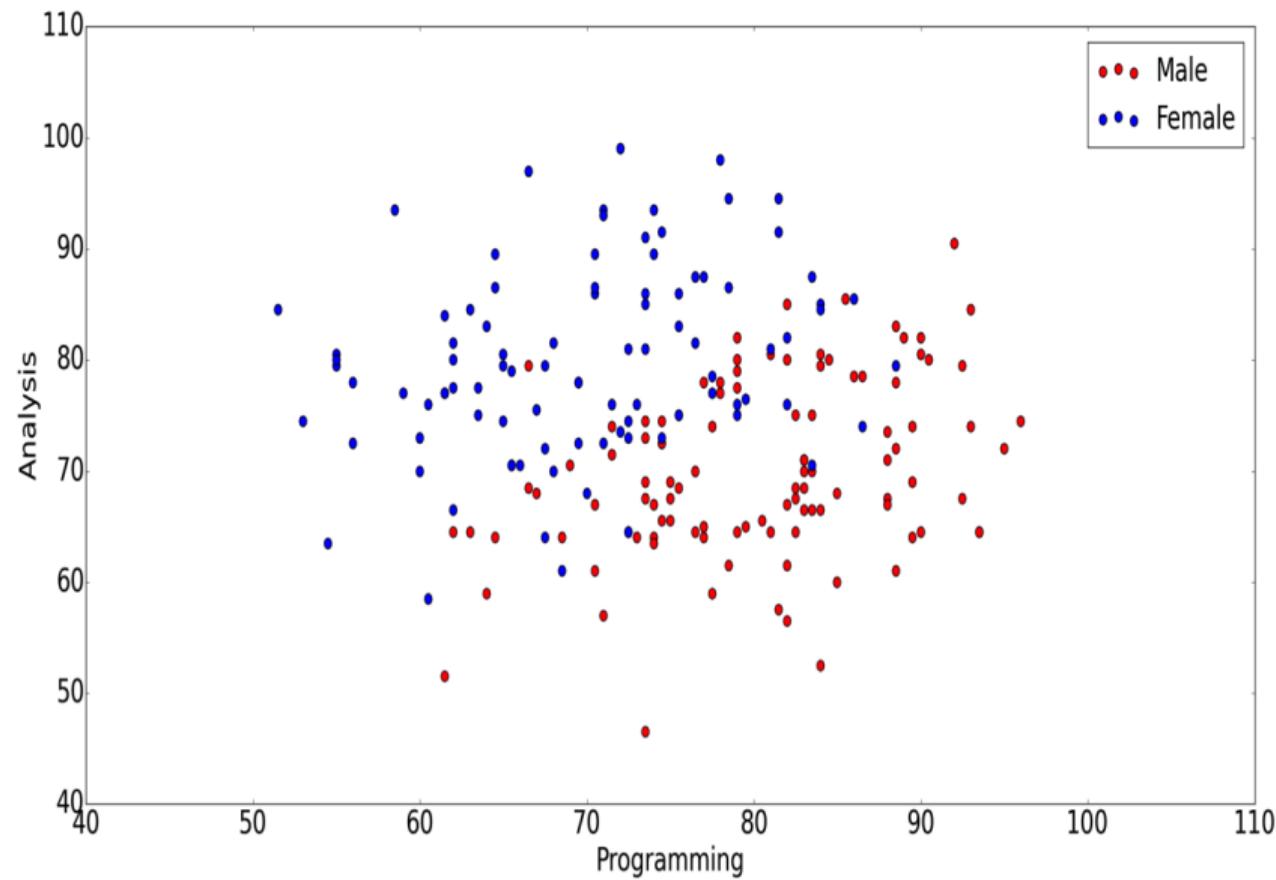
Lecture 9 — Big Data Analytics (Part I)

Nick Tothill

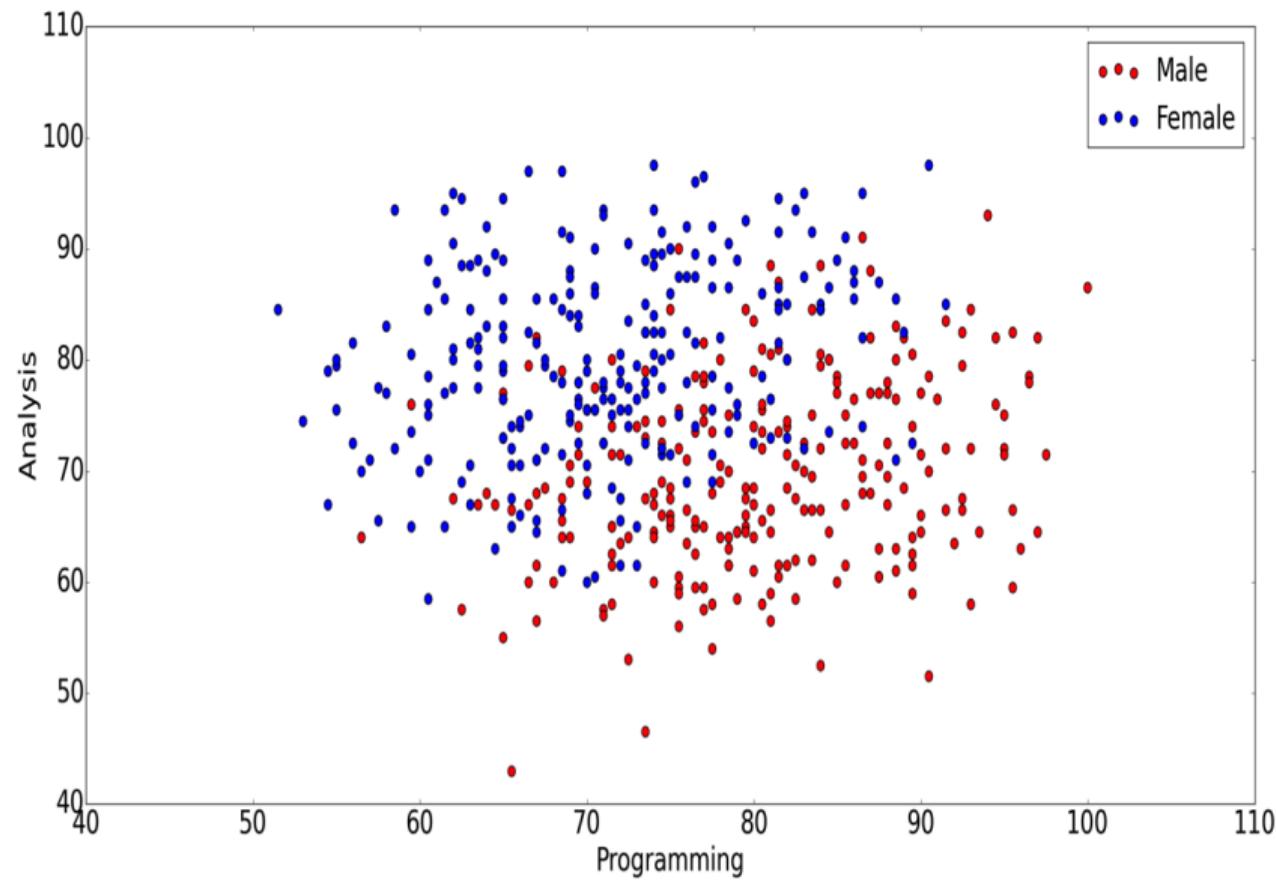
Spring 2019

Introduction to Data Analysis

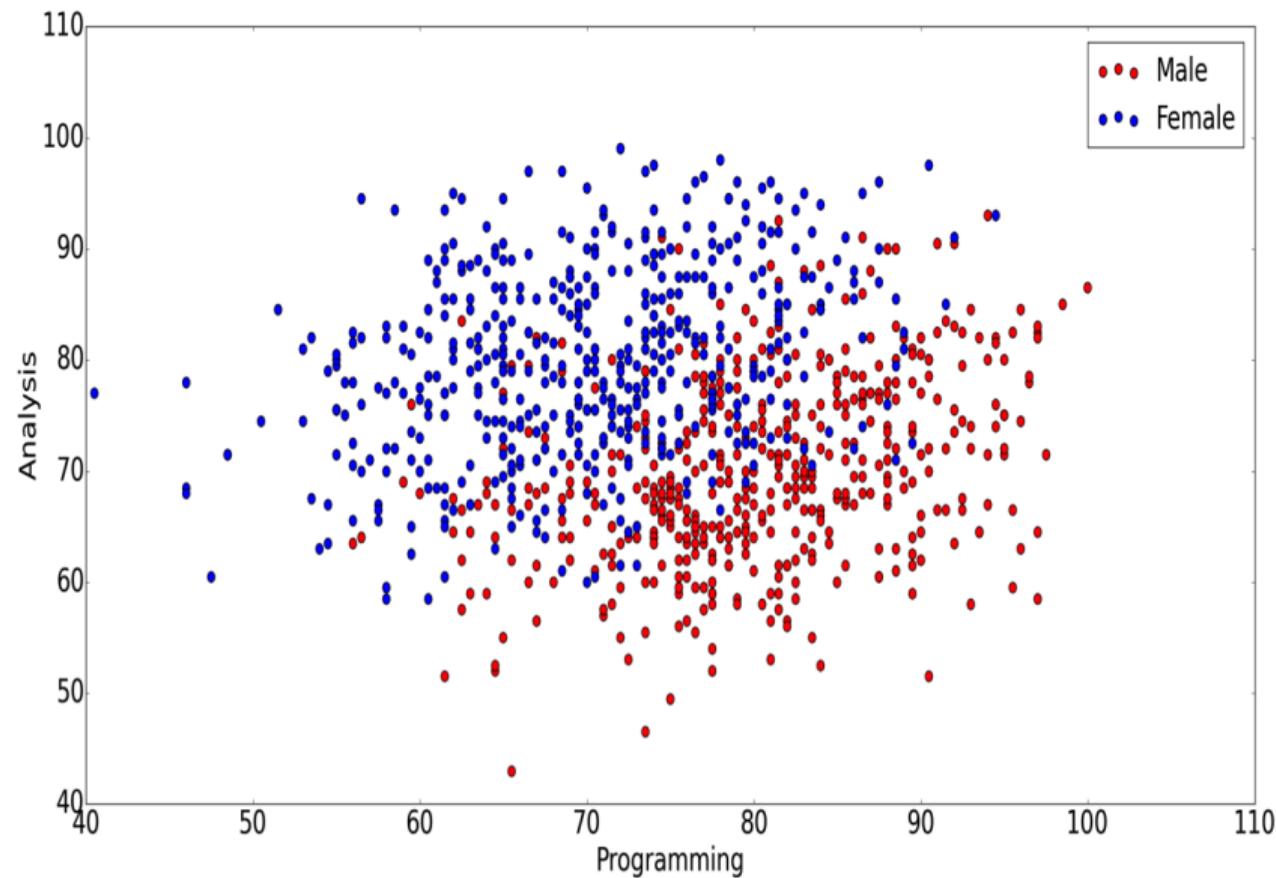
Student Record Example (200 Records)



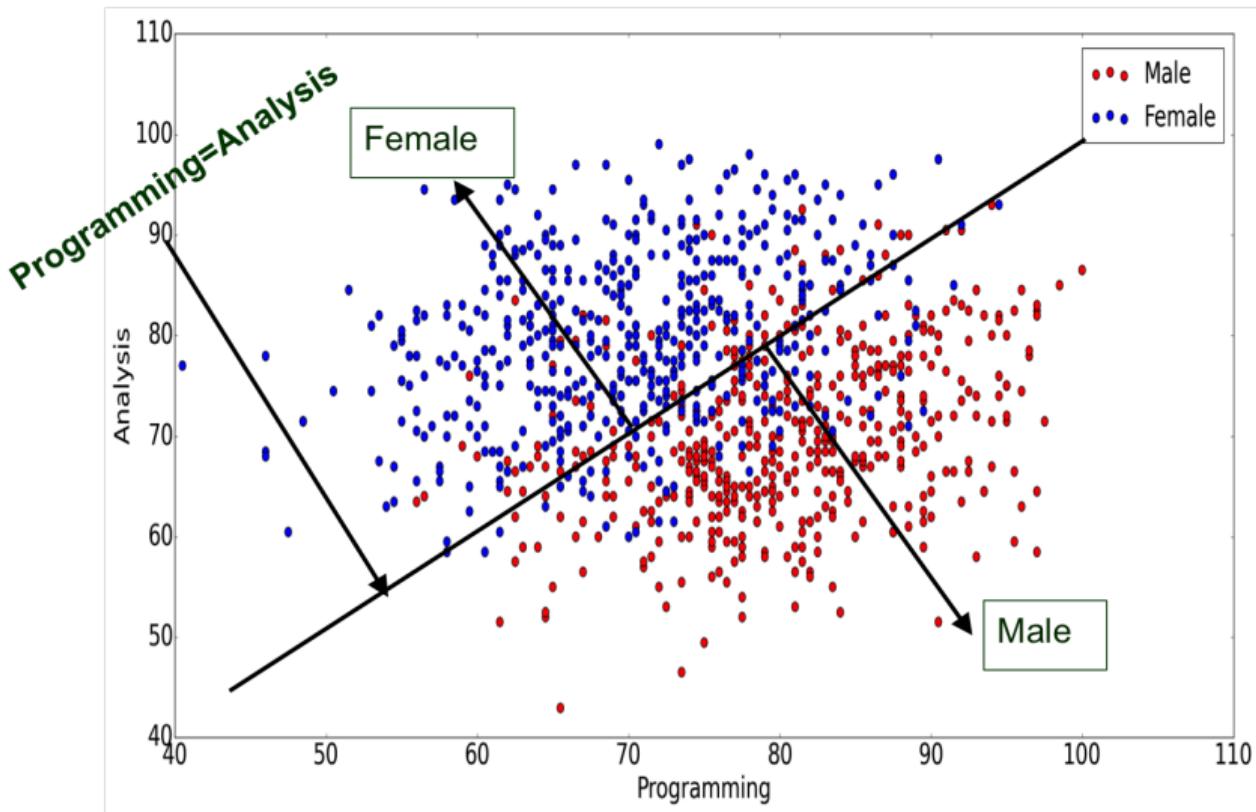
Student Record Example (500 Records)



Student Record Example (1000 Records)



Student Record Example (1000 Records)



Fundamentals of Data Analysis

Predictive Modelling

Data has properties (obviously). They are called differently according to the context:

For knowns (observables):

- ▶ features, attributes, covariates, predictors

For unknowns (interesting, unobservable but available during modeling)

- ▶ targets/labels/dependents/response...

Predictive modeling involves searching for structure in the dataset

- ▶ Use multiple attributes/features/properties in conjunction to make some *prediction* about other properties
- ▶ we want these predictions to be statistically helpful — not perfect

Predictive Modelling

Diagram illustrating Predictive Modelling:

The process involves **Features (attributes)**, **Examples (instances)**, and **Target**.

The **Features (attributes)** are represented by columns: **Programming** and **Analysis**. The **Target** is represented by the **Sex** column.

The **Examples (instances)** are shown as rows of data points. Each row contains values for Programming and Analysis, followed by the corresponding Sex value.

Features (attributes)		Target
Programming	Analysis	Sex
61.5	89	F
72	78	F
86.5	69.5	M
80	74	M
66.5	73.5	F
91	76.5	M
...



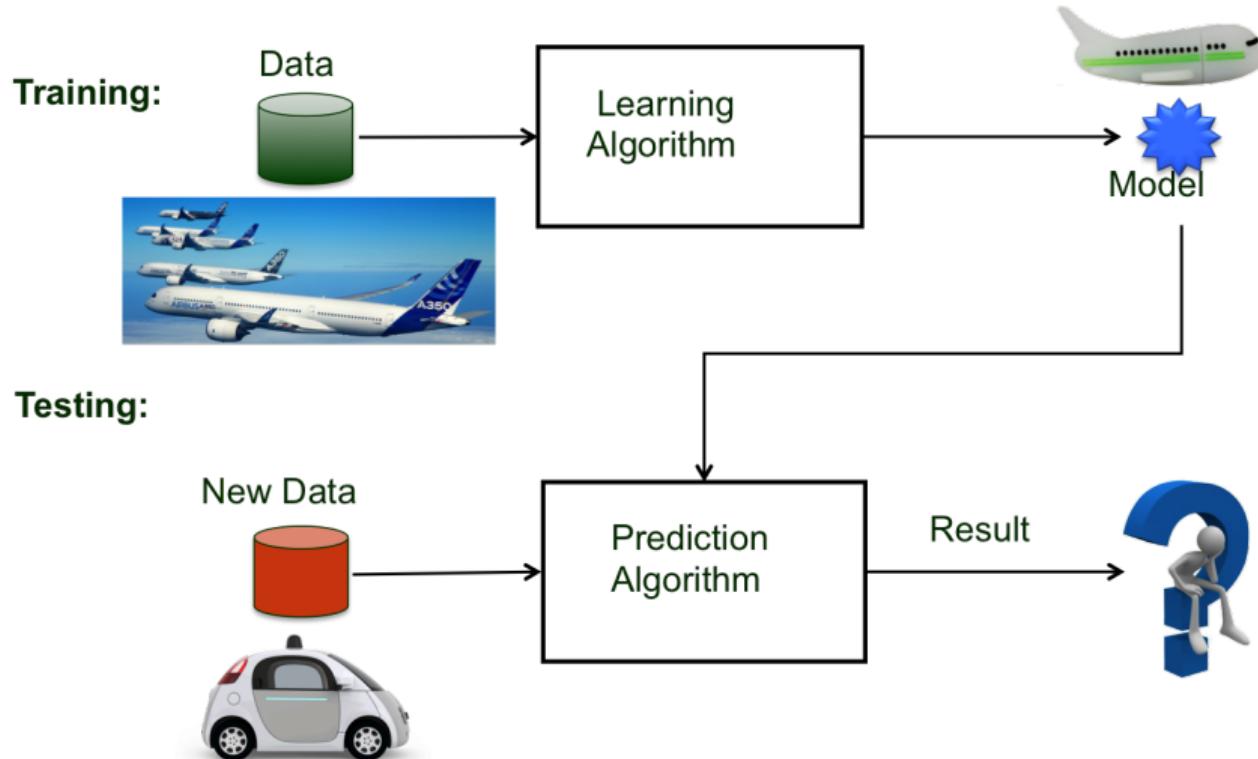
Predictive Modelling

- ▶ A key technique to extract knowledge from data
- ▶ Two Stage Process
 - ▶ Stage 1 – Training
 - ▶ Supervised machine learning: training examples with known target values
 - ▶ give examples of the data that would be used to predict the target and the ‘true’ value of the target attribute
 - ▶ Stage 2 – Testing
 - ▶ Use the trained model for prediction on testing data

Questions:

- ▶ Once trained, does the model do a good job?
- ▶ A good predictive model will achieve high accuracy or low error on unseen data
- ▶ Error seen on training set does not count?!
- ▶ Zero training error can easily be achieved by cheating (any ideas why/how?)
- ▶ Consider how the error may correlate with model complexity/ degree of freedom

Predictive Modelling



Classification vs Regression

Classification

- ▶ Target values: categorical labels
- ▶ Binary (two classes) vs multiclass (>2) classification
- ▶ Examples
 - ▶ Cancer diagnosis (target: positive/negative)
 - ▶ Face detection (target: face/non-face)
 - ▶ Digit recognition (target: digits 0 ~ 9)

Regression

- ▶ Target values: non-categorical output
- ▶ Examples: weather forecast, stock price prediction, traffic condition prediction, ...

Actually, classification can be regarded as a special case of regression if we remove *non-categorical* condition from regression.

Learning Algorithms for Classification

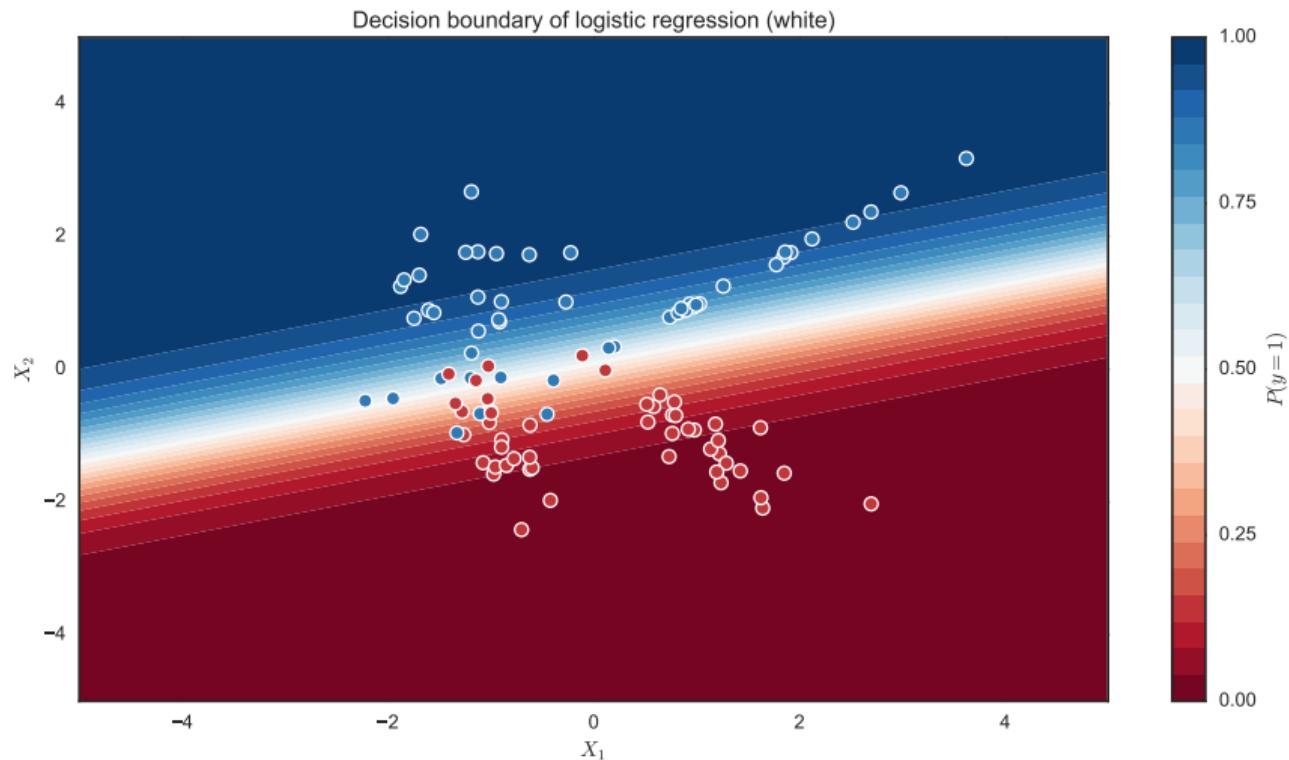
Learning Algorithms are central to classification

- ▶ How to model the feature for classification purposes

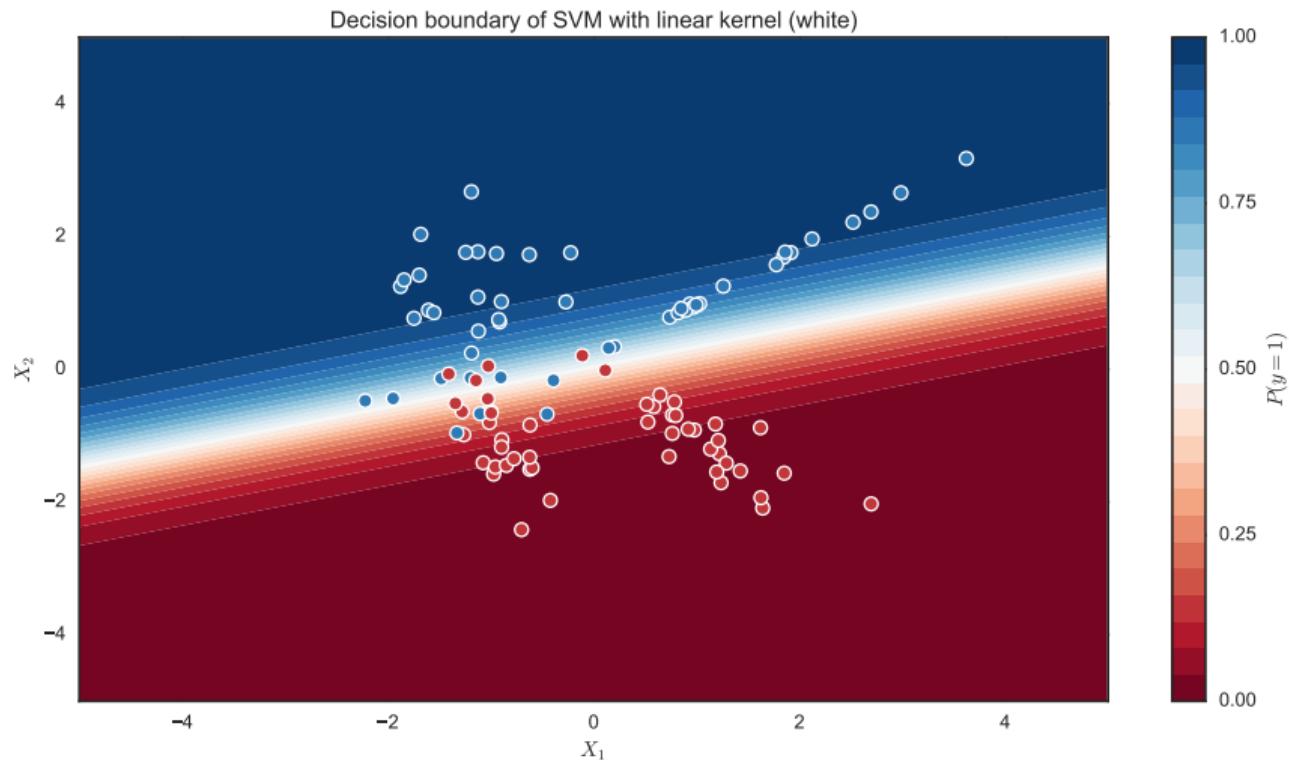
Types of Learning Algorithms

- ▶ Linear: The decision boundaries are linear (straight lines). Typical examples are: linear discriminant analysis, Support Vector Machine (SVM) with linear kernel, Logistic Regression (LR), ...
- ▶ Nonlinear: The decision boundaries are nonlinear (curves)
 - ▶ Parametric: SVM with nonlinear kernels, neural networks, ...
 - ▶ Non-Parametric: K-Nearest Neighbour (K-NN), ...
 - ▶ Ensemble: bagging, boosting, random forest, ...

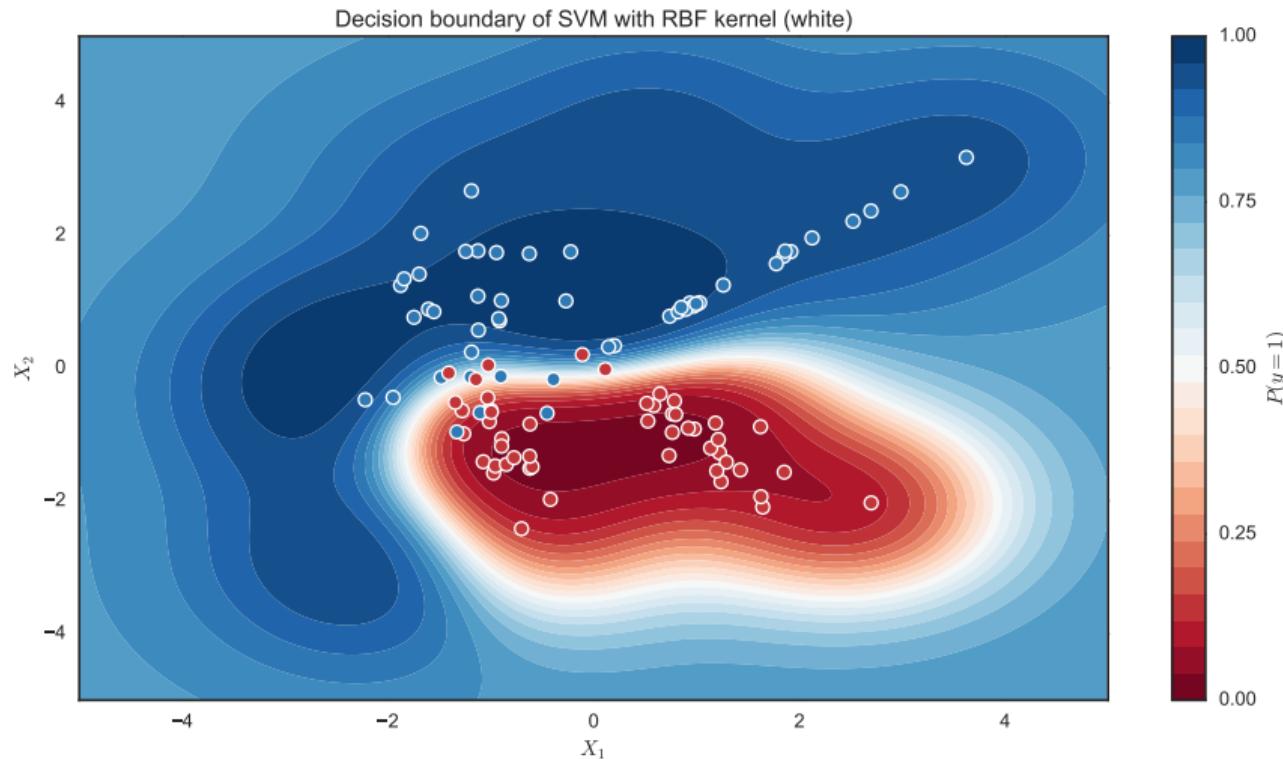
Decision boundary produced by logistic regression



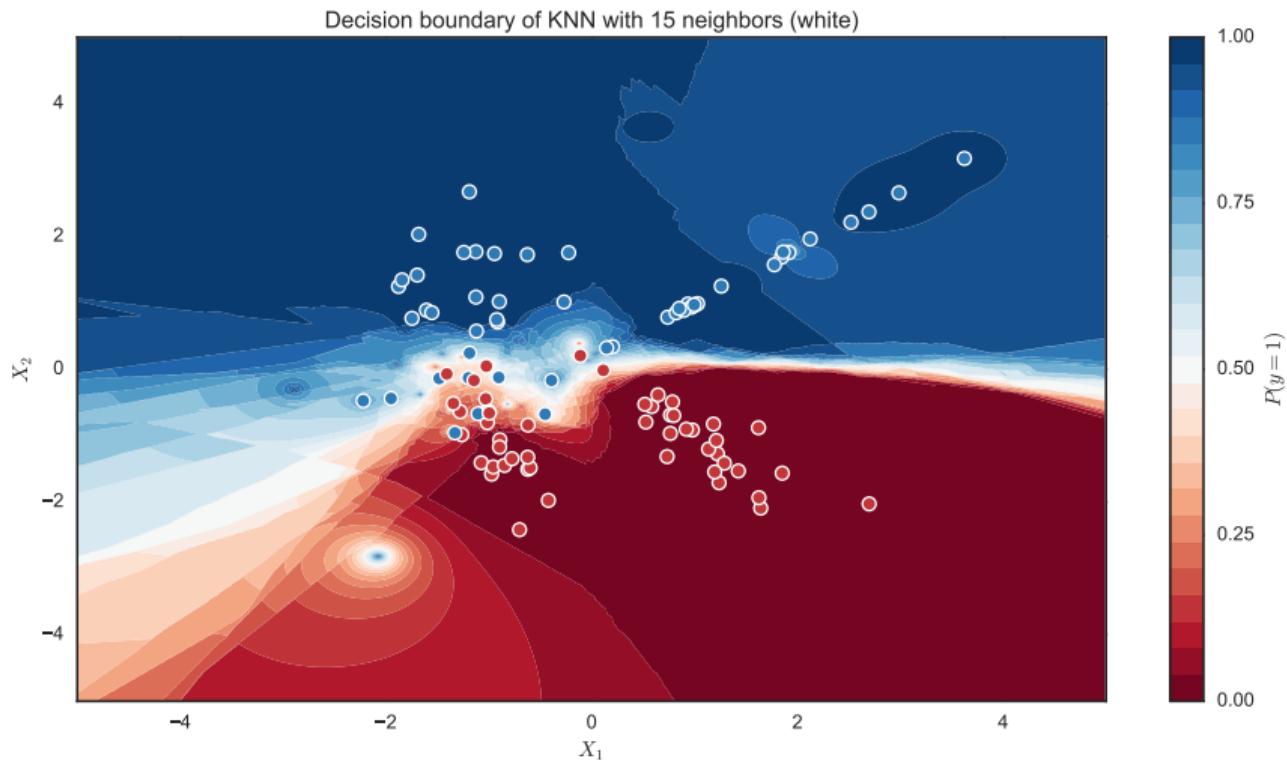
Decision boundary produced by support vector machine with linear kernel



Decision boundary produced by support vector machine with RBF kernel



Decision boundary produced by KNN with 15 neighbors



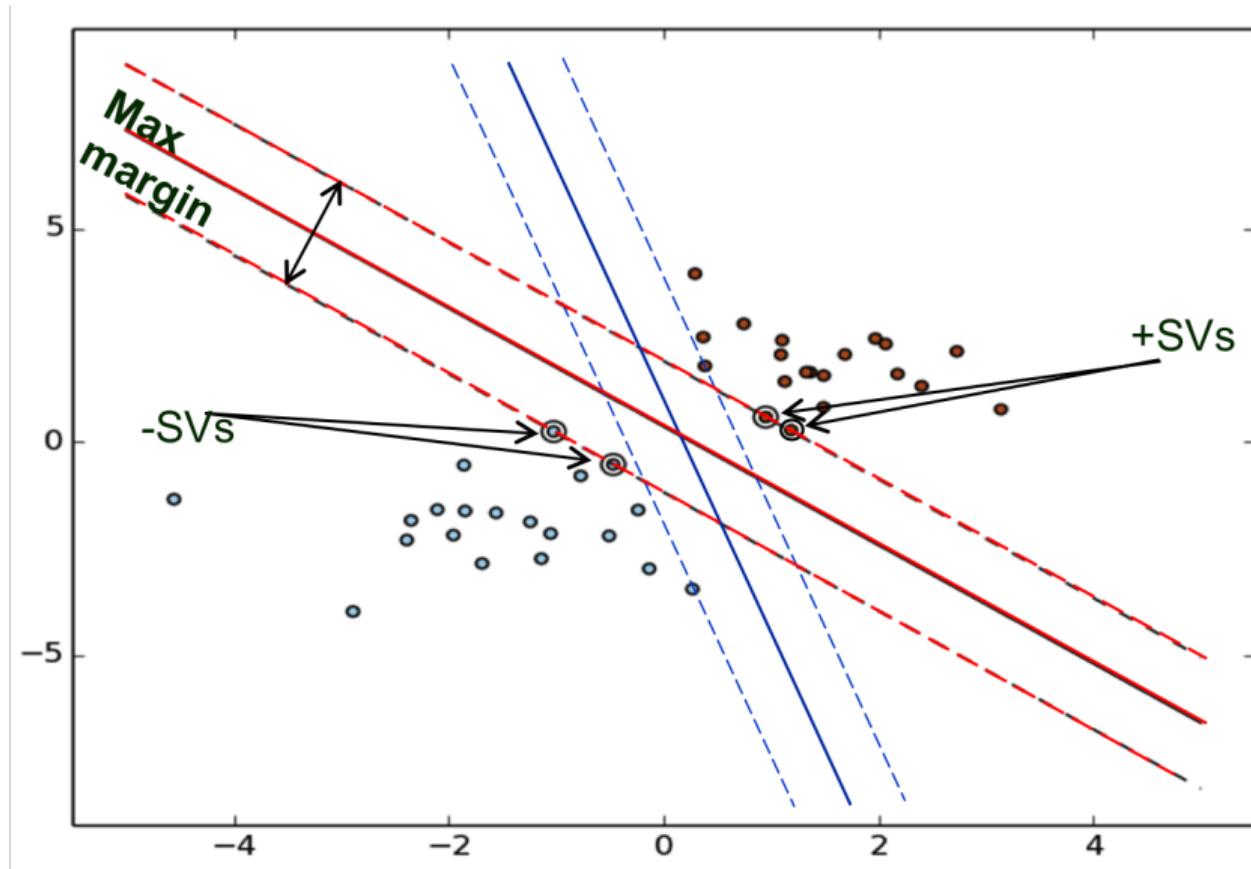
Learning Algorithms for Classification

Classifier (Classification/Predictive Model): Output of the learning algorithm that can be used for prediction

Forms of Classifiers

- ▶ Rule-based
 - ▶ Simple logic-based rules: e.g. decision trees
 - ▶ Prototype-based: K-NN uses labels of nearest points to vote on the target label value
- ▶ Function-based:
 - ▶ Linear function: hyperplanes that split the feature space into different regions
 - ▶ Non-linear function: complex surface instead of hyperplane that separates different classes.

Linear Support Vector Machine (SVM)



Linear Support Vector Machine (SVM)

- ▶ Linear SVM produces a classifier with linear decision boundary and maximum margin
 - ▶ Larger margin makes it harder to cross the boundary and hence produces a more robust classifier
 - ▶ Larger margin improves the generalisation performance
- ▶ The final classifier only depends on the support vectors
 - ▶ Support Vectors (SV): points on or within the margin on either side of the decision boundary
 - ▶ Other points do not affect the classifier trained
- ▶ Linear SVM is error-tolerant to training data -> soft margin vs hard margin: the tolerance to errors
- ▶ Multiple classes extension: one-vs-one or one-vs-all

Using SVM in python

```
import numpy as np
from sklearn import svm, datasets

# import some data to play with
iris = datasets.load_iris()
trainidx = np.random.choice(iris['data'].shape[0], int(iris['data'].shape[0]*0.7))
testidx = list(set(range(iris['data'].shape[0])) - set(trainidx))
trainX = iris.data[trainidx, :2] # Training set: Take the first two features
trainy = iris.target[trainidx] # Training set: trainy contains the class labels
testX = iris.data[testidx,:2] # Test set: features
testy = iris.target[testidx] # Test set: class labels

# we create an instance of SVM and fit the data.
C = 1.0 # SVM regularization parameter
svc = svm.SVC(kernel='linear', C=C) # Define SVM model
svc.fit(trainX, trainy) # Fit SVM model to the training data
# lin_svc = svm.LinearSVC(C=C).fit(trainX, trainy) # LinearSVC is a more scalable version

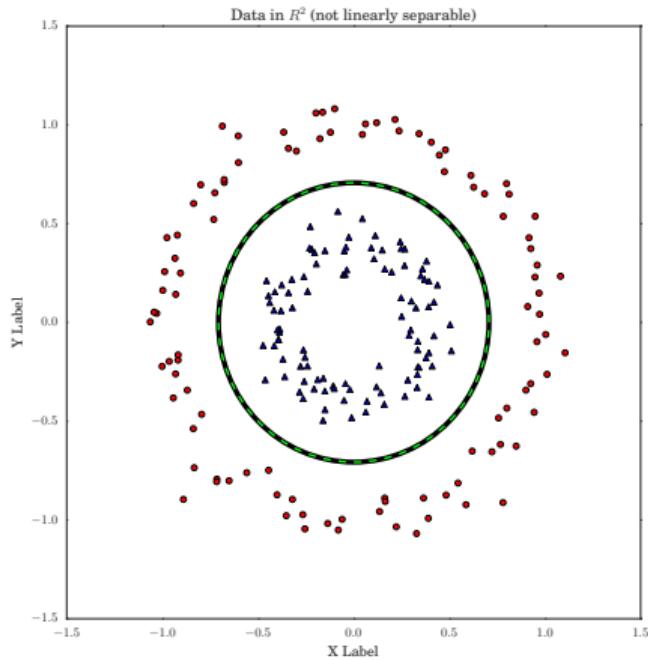
Z = svc.predict(testX) # Make prediction using SVM model for test data
# Collect prediction accuracy
print "Prediction accuracy:", sum(Z==testy)/float(len(Z))*100, '%'
```

Prediction accuracy: 80.2816901408 %

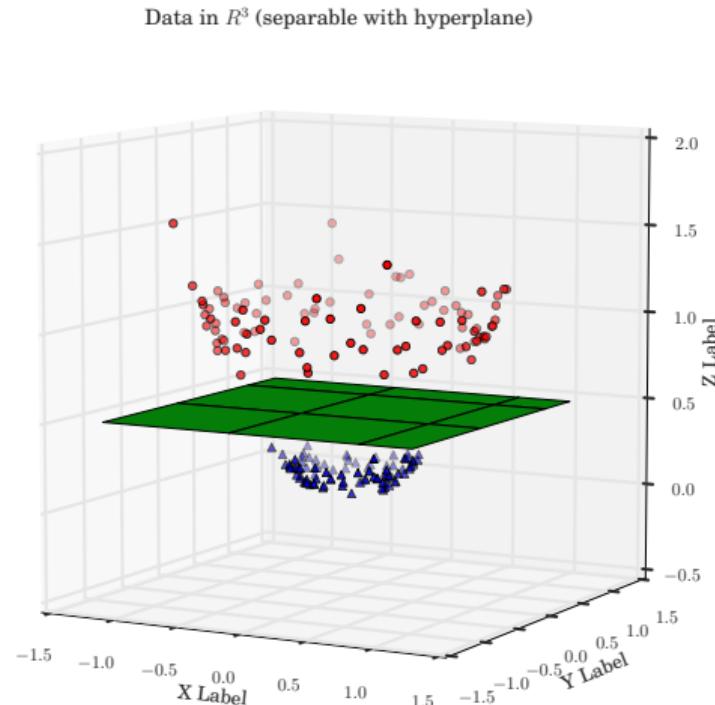
Non-linear (Kernel) SVM

- ▶ Linear SVM can not handle non-linear data sets
- ▶ Possible to map non-linear data to a new space
 - ▶ Data become linearly separable in the new space
- ▶ Ideas behind kernel SVM
 - ▶ Map non-linear data to the kernel space with an implicit feature mapping induced by the kernel function
 - ▶ Train a Linear SVM in the kernel space
- ▶ Different kernel functions lead to different classifiers
- ▶ Common kernel functions
 - ▶ Linear kernel: equivalent to Linear SVM
 - ▶ RBF/Gaussian kernel (most popular for nonlinear classification)
 - ▶ Polynomial kernel

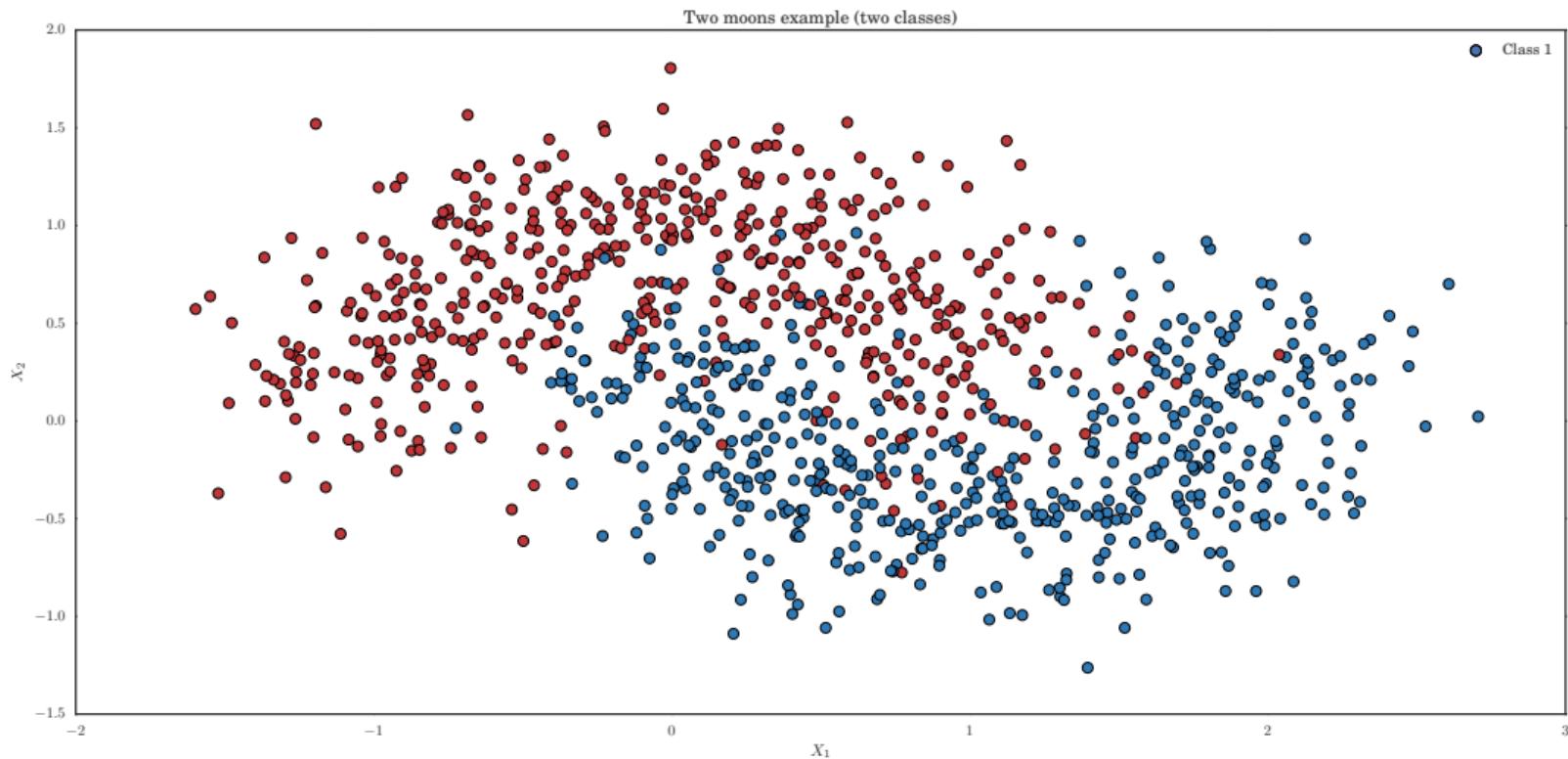
Non-linear (Kernel) SVM - “kernel trick”



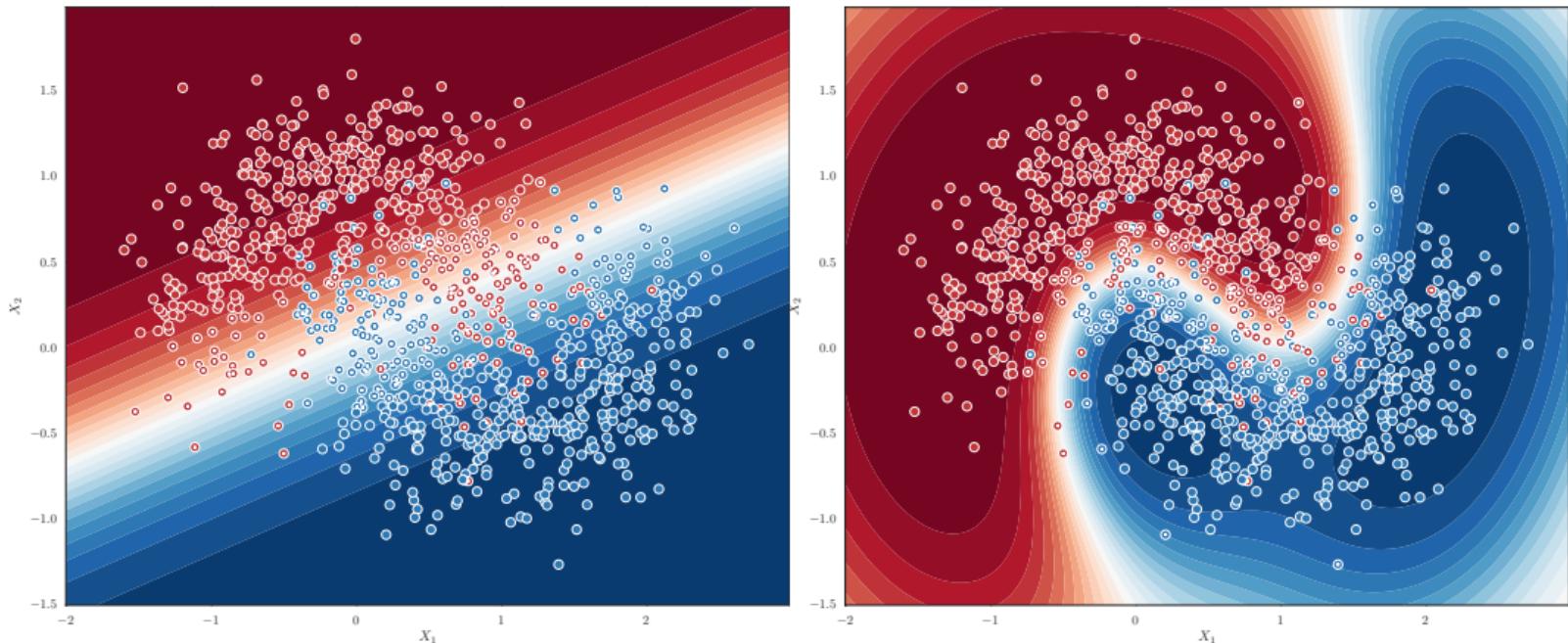
$$\begin{aligned}x &\mapsto y \\ \iff \\ \phi(x) &= y\end{aligned}$$



Two moons example



Two moons example

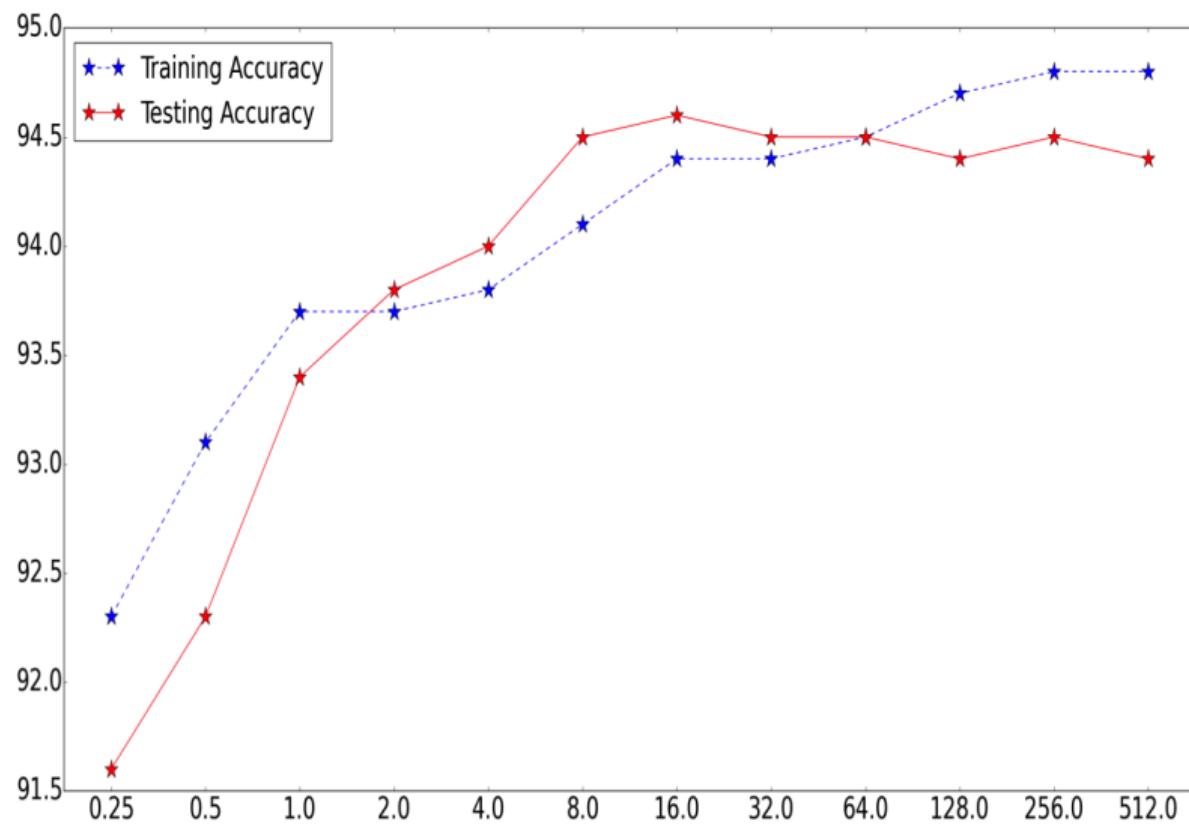


Linear SVM (left): Accuracy = 86.0%, Kernel SVM (right): Accuracy = 93.4%

Parameter Selection

- ▶ Predictive performance of classification model depends on the parameters used in training
- ▶ Regularisation Parameter (C) for SVM classifier
 - ▶ Trade-off between classifier margin and training errors
 - ▶ $C \nearrow$, Margin \searrow , Training Error \searrow
- ▶ Traps for parameter selection
 - ▶ Wrong: use training set performance as guide
 - ▶ Why not training performance?
 - ▶ Overfitting: classifier fits the training data too well (including errors in the data) and gets excessively complex, lacking generality and leading to poor testing performance
 - ▶ use *testing set* performance as guide
 - ▶ testing set must be completely different to training set — no overlap

Training vs Testing Accuracy



Cross Validation

- ▶ Cross Validation is an effective technique for empirical selection of classifier parameters
- ▶ K-fold Cross Validation
 1. Divide the training data into K-folds Stratified Sampling: each fold keeps the same proportion of data from different classes as the original training set
 2. Select 1 fold for testing (validation set) and train a classification model using data from remaining K-1 folds
 3. Apply trained model on validation set and calculate the accuracy
 4. Repeat above two steps K times and get the average accuracy
- ▶ Bottom line: Never use labels of testing data in classifier training and parameter selection

What is the right practice?

Scenario: you are given a training dataset of 500 labelled examples to learn a model for prediction on a large testing dataset of 5 million examples. The testing labels are provided for you to test your algorithm.

- ▶ Train a classifier on all training and testing data
- ▶ Train a classifier on training data plus 500 examples randomly selected from the testing dataset
- ▶ Use testing data for parameter selection and train a classifier on training data only
- ▶ Use the testing data to identify the best 5 features and train a classifier on training data only using the 5 features chosen

None of the above is CORRECT!

Feature Pre-processing

The Census Data Example

	age	workclass	fnlwgt	years-of-edu	marital-status	occupation	relationship	race	sex	capital-gain	capital-loss	hours	salary
0	39	State-gov	77516	13	Never-married	Adm-clerical	Not-in-family	White	Male	2174	0	40	<=50K
1	50	Self-emp-not-inc	83311	13	Married-civ-spouse	Exec-managerial	Husband	White	Male	0	0	13	<=50K
2	38	Private	215646	9	Divorced	Handlers-cleaners	Not-in-family	White	Male	0	0	40	<=50K
3	53	Private	234721	7	Married-civ-spouse	Handlers-cleaners	Husband	Black	Male	0	0	40	<=50K
4	28	Private	338409	13	Married-civ-spouse	Prof-specialty	Wife	Black	Female	0	0	40	<=50K
5	37	Private	284582	14	Married-civ-spouse	Exec-managerial	Wife	White	Female	0	0	40	<=50K

fnlwgt: final weight (CPS weighting: current population weighting)

Challenges of the Census Data

Mix of numerical features and nominal features

- ▶ Numerical features
 - ▶ Columns taking numerical values
 - ▶ E.g.: age, fnlwgt, years-of-edu, ..., hours
- ▶ Nominal features
 - ▶ Categorical values (1 out of many)
 - ▶ E.g.: workclass, marital-status, sex, race, ...
 - ▶ Need to convert nominal features to numerical ones

Different scales of Numerical features

- ▶ < 100: age, hours, years-of-edu
- ▶ > 1000: fnlwgt, capital-gain, capital-loss
- ▶ Need to pre-process features to the same scale

Nominal-to-Numeric Conversion

Substitution scheme

- ▶ Each unique feature value converted to one number
 - ▶ Sex: Male → 0 | Female → 1
 - ▶ Relationship: Husband → 0 | Wife → 1 | Not-in-family → 2
 - ▶ Race: White → 0 | Black → 1 | Asian → 2 | Aboriginal → 3

One-Of-K scheme

- ▶ Each unique feature value convert to a K-vector
 - ▶ Sex: Male → [0,1] | Female → [1,0]
 - ▶ Race: White → [0,0,0,1] | Black → [0,0,1,0] | Asian → [0,1,0,0]...
- ▶ Length of vector K is determined by the number of unique feature values

Comparison of Conversion Schemes

Efficiency (Substitution>One-Of-K)

- ▶ Substitution scheme converts one nominal column to one numerical column
- ▶ One-Of-K scheme greatly increases the number of features due to one-to-K mapping

Effectiveness (One-Of-K>Substitution)

- ▶ Problem with multiple values for substitution scheme
 - ▶ Race: White → 0 | Black → 1 | Asian → 2 | Aboriginal → 3
 - ▶ $\text{dist}(\text{White}, \text{Black})=1$, $\text{dist}(\text{White}, \text{Aboriginal})=3?$
- ▶ One-Of-K scheme does not have the problem
 - ▶ Distance between any two values equals 1
- ▶ One-Of-K achieves better results in general

Pre-Processing for Numerical Features

Numerical features need to be transformed to the same scale for training and prediction

- ▶ Method 1: standardisation
 - ▶ Each feature is linearly scaled to have zero mean and unit standard deviation

$$x_i = (x_i - \mu)/\sigma$$

- ▶ Method 2: min-max scaling
 - ▶ Each feature is linearly scaled to have a maximum value of 1 and minimum value of 0

$$x_i = (x_i - \min(x_1, \dots, x_n)) / (\max(x_1, \dots, x_n) - \min(x_1, \dots, x_n))$$

- ▶ Testing data need to be transformed in the same way

Following topics

Future topics

Data Analysis for Big Data (part II)

- ▶ Tree based classifiers
- ▶ Regression

Data Analysis for Big Data (part III)

- ▶ Spark

Applications of Big Data

Lecture 10 - Big Data Analytics (Part II)

Nick Tothill

Spring 2019

Recap: Classification

Predictive Modelling

Diagram illustrating Predictive Modelling:

The process involves **Features (attributes)**, **Examples (instances)**, and **Target**.

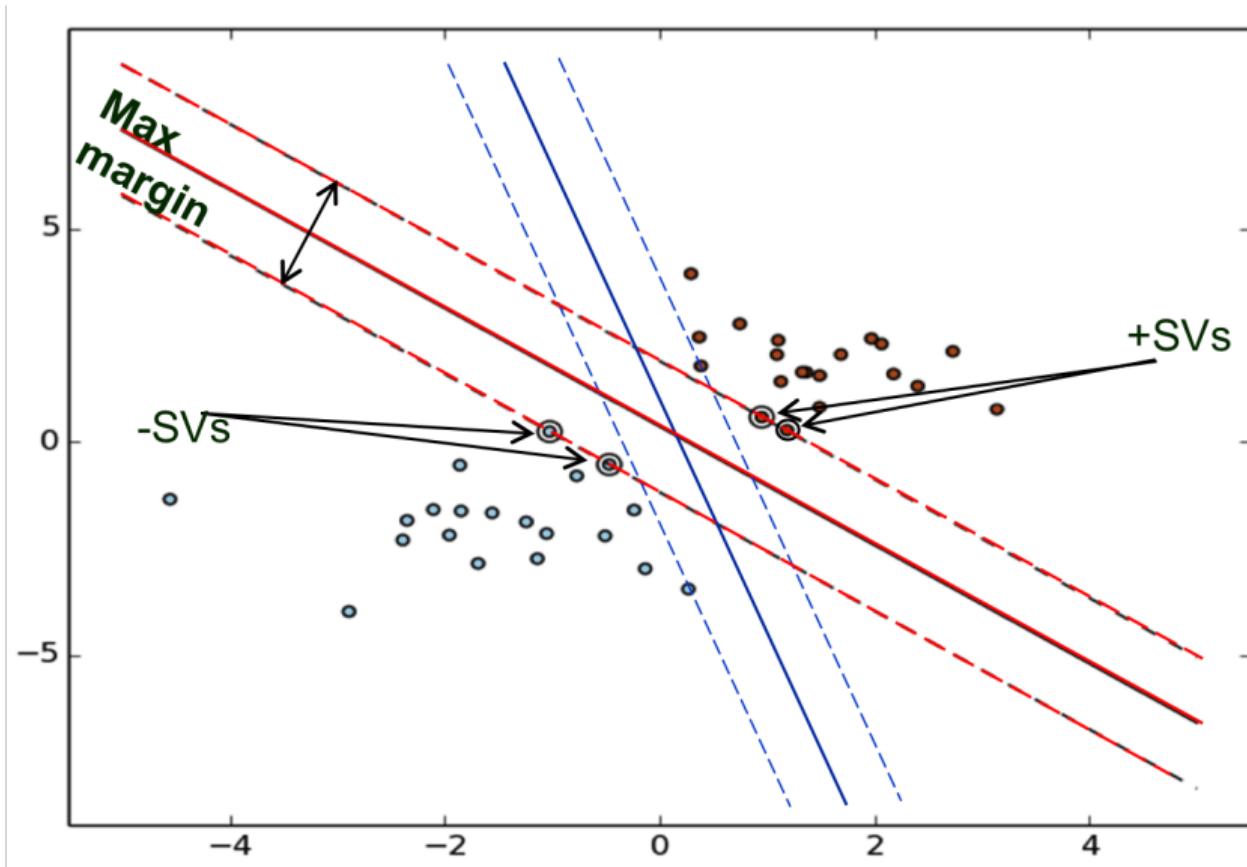
The **Features (attributes)** are represented by columns: **Programming** and **Analysis**. The **Target** is represented by the column **Sex**.

The **Examples (instances)** are shown as rows of data points. Each row contains values for Programming and Analysis, followed by the corresponding Sex value.

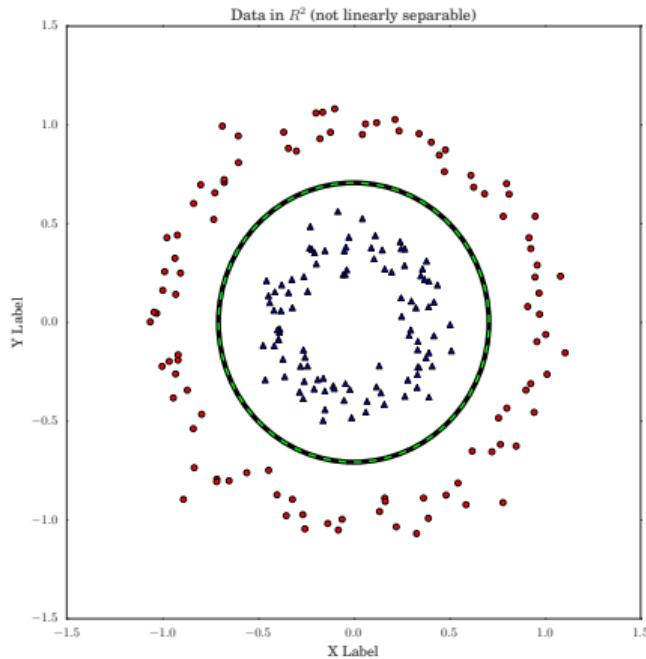
	Features (attributes)	Target	
	Programming	Analysis	Sex
1	61.5	89	F
2	72	78	F
3	86.5	69.5	M
4	80	74	M
5	66.5	73.5	F
6	91	76.5	M
...



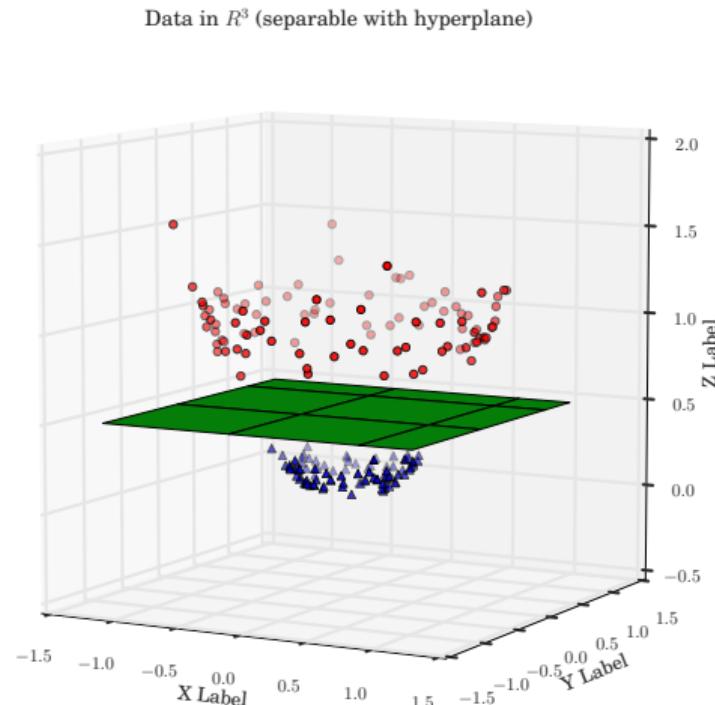
Support Vector Machine (SVM)



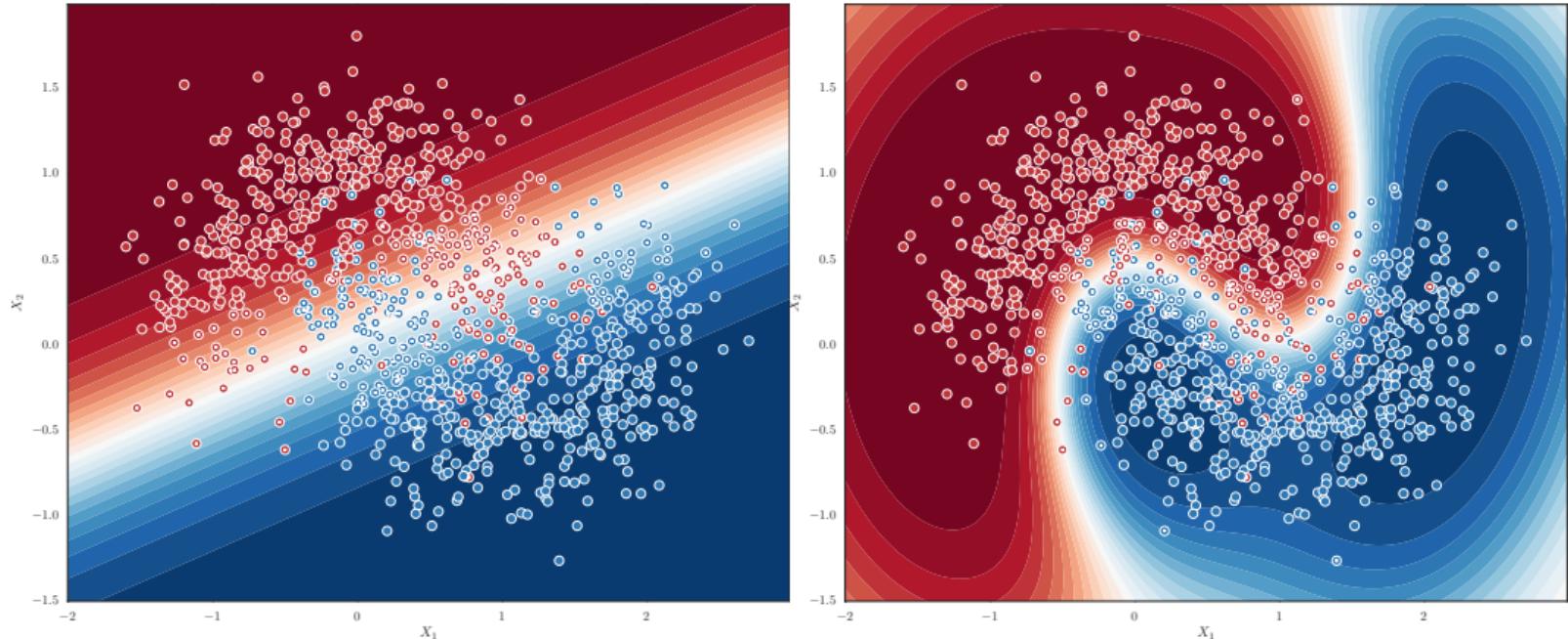
From linear to nonlinear SVM using “kernel trick”



$$\begin{aligned}x &\mapsto y \\ \iff & \\ \phi(x) &= y\end{aligned}$$



Two-moons dataset



Linear SVM (left): Accuracy = 86.0%, Kernel SVM (right): Accuracy = 93.4%

Linear vs Kernel SVM

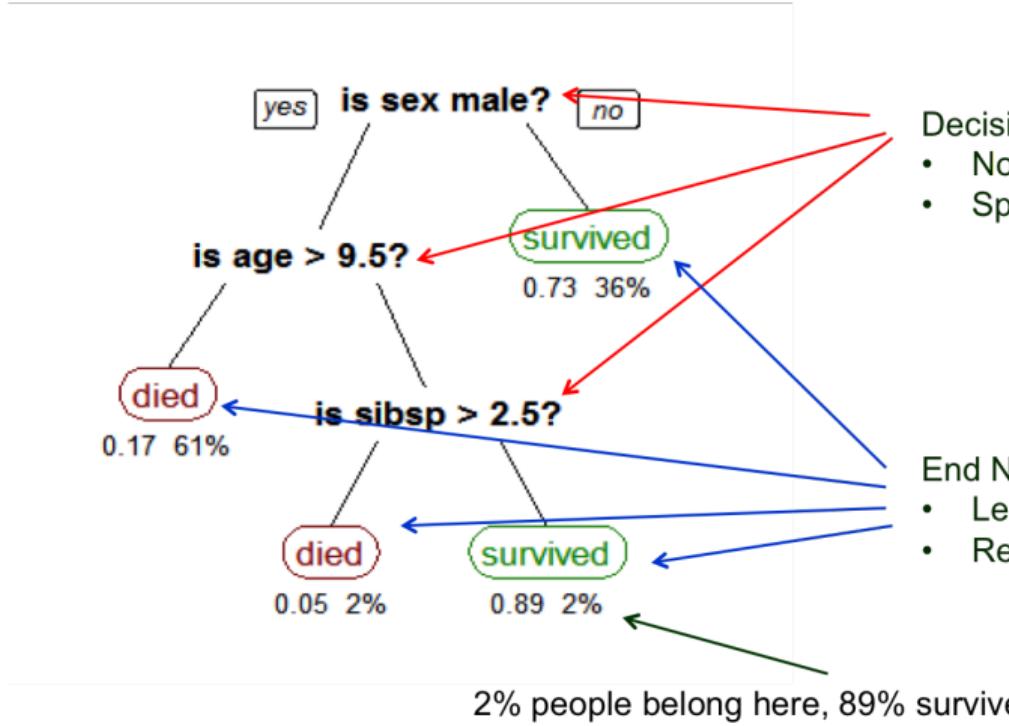
- ▶ Speed
 - ▶ Training: Linear SVM is much faster
Kernel SVM cannot handle large-scale datasets
 - ▶ Prediction: Linear SVM is faster
Only matters for real-time applications
- ▶ Performance
 - ▶ Kernel SVM is better than linear SVM in general Usually depends on data size, number of features, and importantly data distributions
- ▶ Practical tips
 - ▶ Always try linear SVM first
 - ▶ Approximation methods exist for faster kernel SVM training

Tree-Based Classifiers

Decision Tree Classifier

- ▶ Decision tree: a decision support tool with a tree-like structure
- ▶ Decision tree uses logical statements for decision making
 - ▶ Each decision rule only involves a single variable
 - ▶ Final rule is a conjunction of individual rules
 - E.g.: pension<100 and donation>50,000 \Rightarrow good citizen
 - ▶ Can visualise with a tree with multiple levels
- ▶ Decision tree can be used as a classification model
 - ▶ Equivalent to recursively partition the feature space into different regions with axis-parallel splits (we can only split the data by a boundary in one feature)
 - ▶ Regions labelled with the majority class
 - ▶ Training Algorithms: ID3, C4.5/5.0, CART

Decision Tree Example



Decision Node

- Non-leaf node
- Split point

End Node

- Leaf of tree
- Result obtained

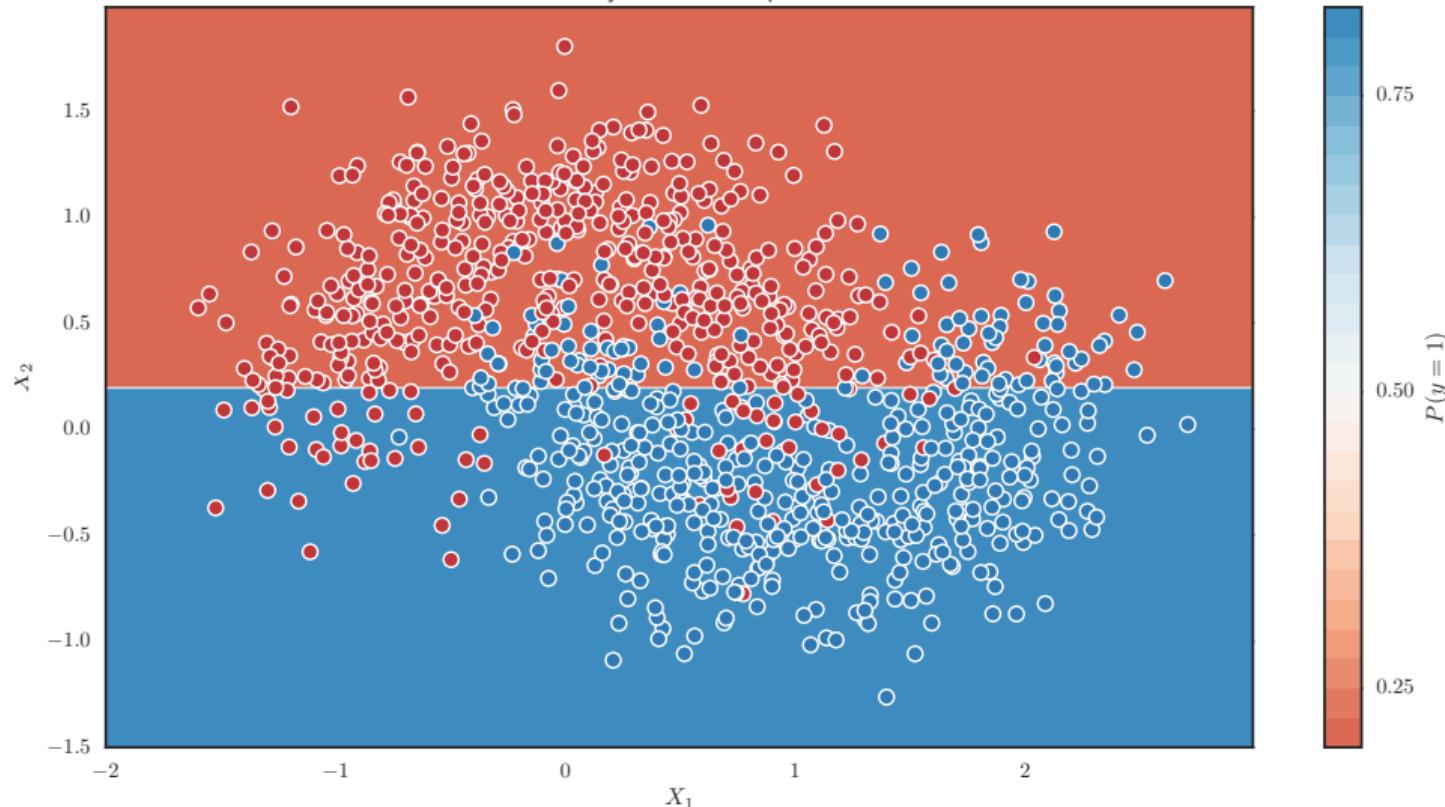
Python code for decision tree using sklearn package

```
import numpy as np
from sklearn.tree import DecisionTreeClassifier
from sklearn.cross_validation import train_test_split
# Generate two moons data
X, y = make_moons(n_samples = 1000, noise=0.3, random_state=0)
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

# Define decision tree classifier with max_depth 4.
clf=DecisionTreeClassifier(max_depth=d)
clf.fit(X_train,y_train) # Train it on training set
score = clf.score(X_test, y_test) # Test on test set
```

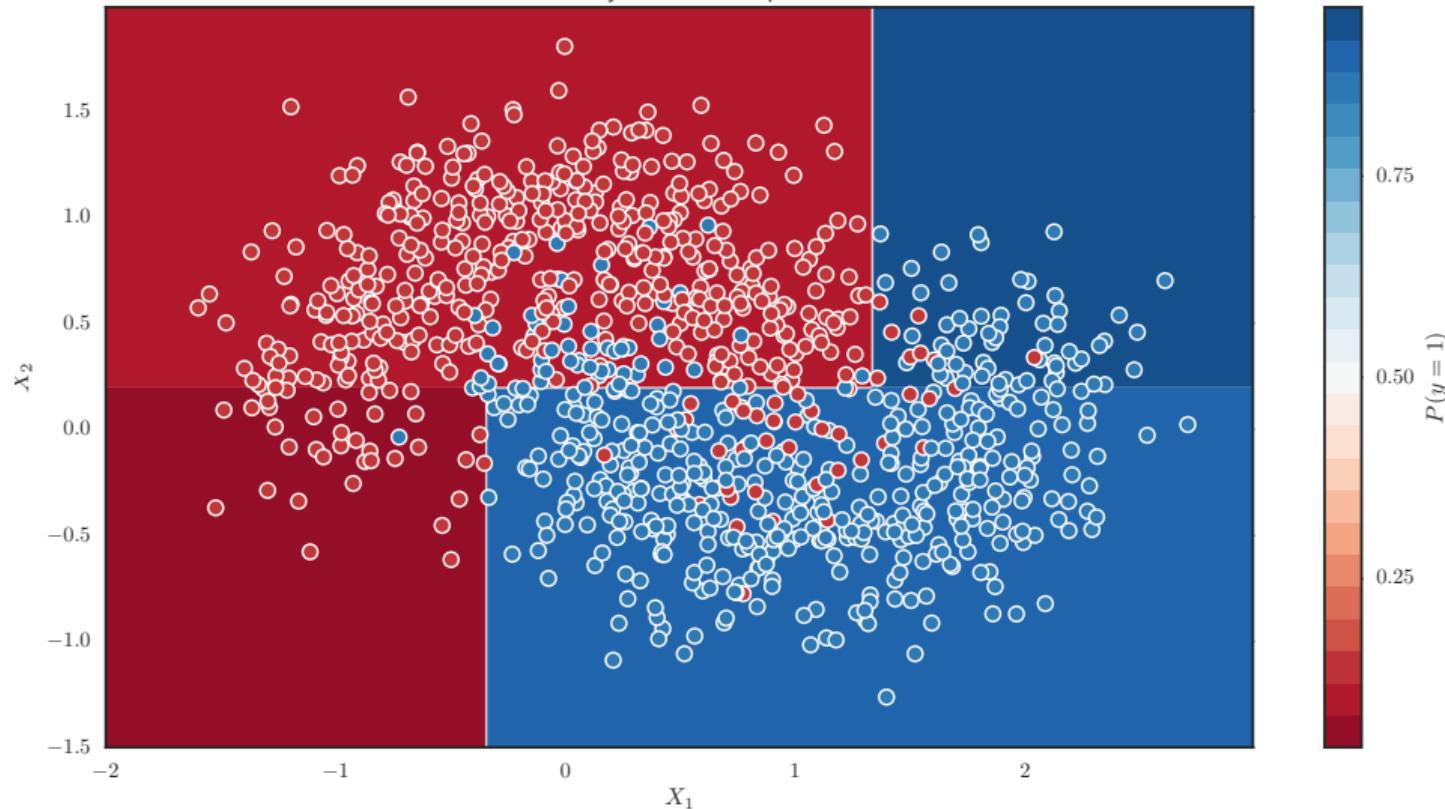
Decision Tree on Two Moons (maximum tree depth = 1)

Accuracy: 0.81 with depth=1

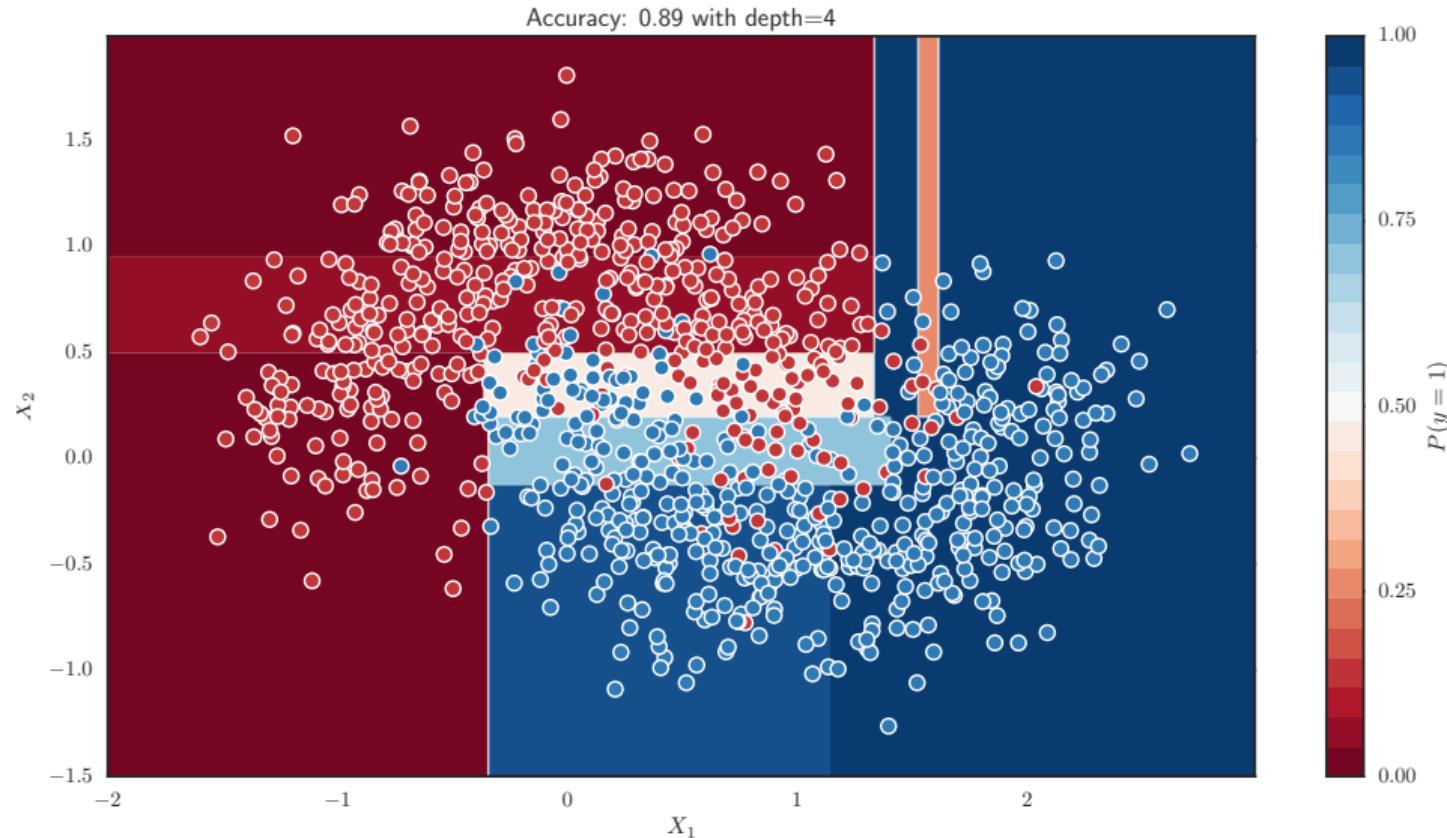


Decision Tree on Two Moons (maximum tree depth = 2)

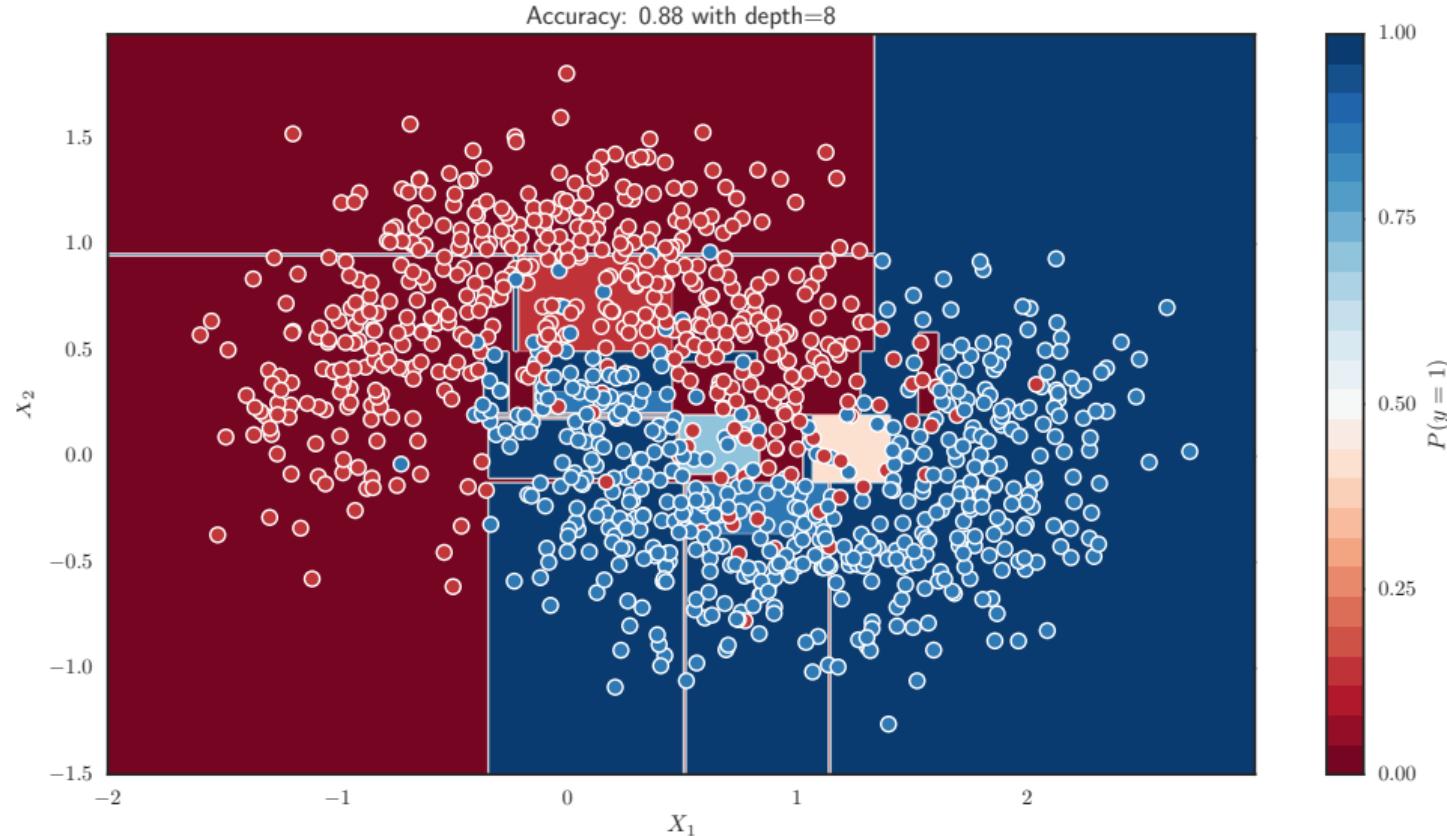
Accuracy: 0.89 with depth=2



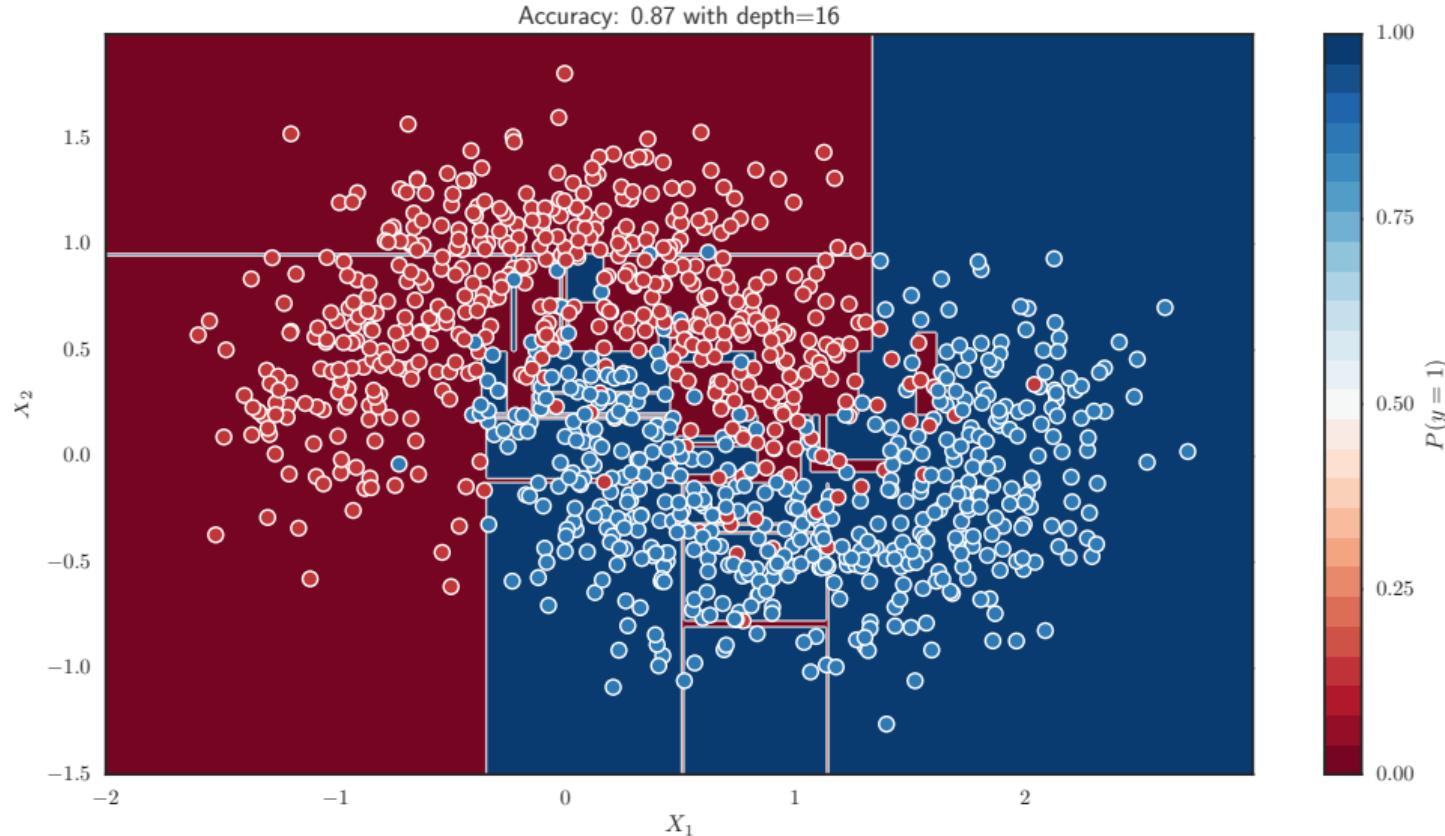
Decision Tree on Two Moons (maximum tree depth = 4)



Decision Tree on Two Moons (maximum tree depth = 8)



Decision Tree on Two Moons (maximum tree depth = 16)



Notes on decision trees

- ▶ Increasing the depth of the tree (`max_depth`) usually improves performance of decision tree classifier
 - ▶ The deeper the tree, the more sophisticated the decision boundary, the more powerful the model
 - ▶ Complexity could be a double-edged sword
 - it can lead to overfitting: good training but poor testing performance
- ▶ Decision tree generates spiky decision boundaries
 - ▶ Due to the use of axis-parallel splits
 - ▶ Can use pruning or multiple trees (random forest)
- ▶ Random factors in decision tree learning
 - ▶ Results are slightly different for multiple runs
 - ▶ Does this matter to you?

Ensemble Model and Random Forest

Ensemble: collection of individual classifiers

- ▶ Reduce biases in individual models
- ▶ Types of ensemble classification models: bagging, boosting, random forest

Random Forest: decision tree ensemble

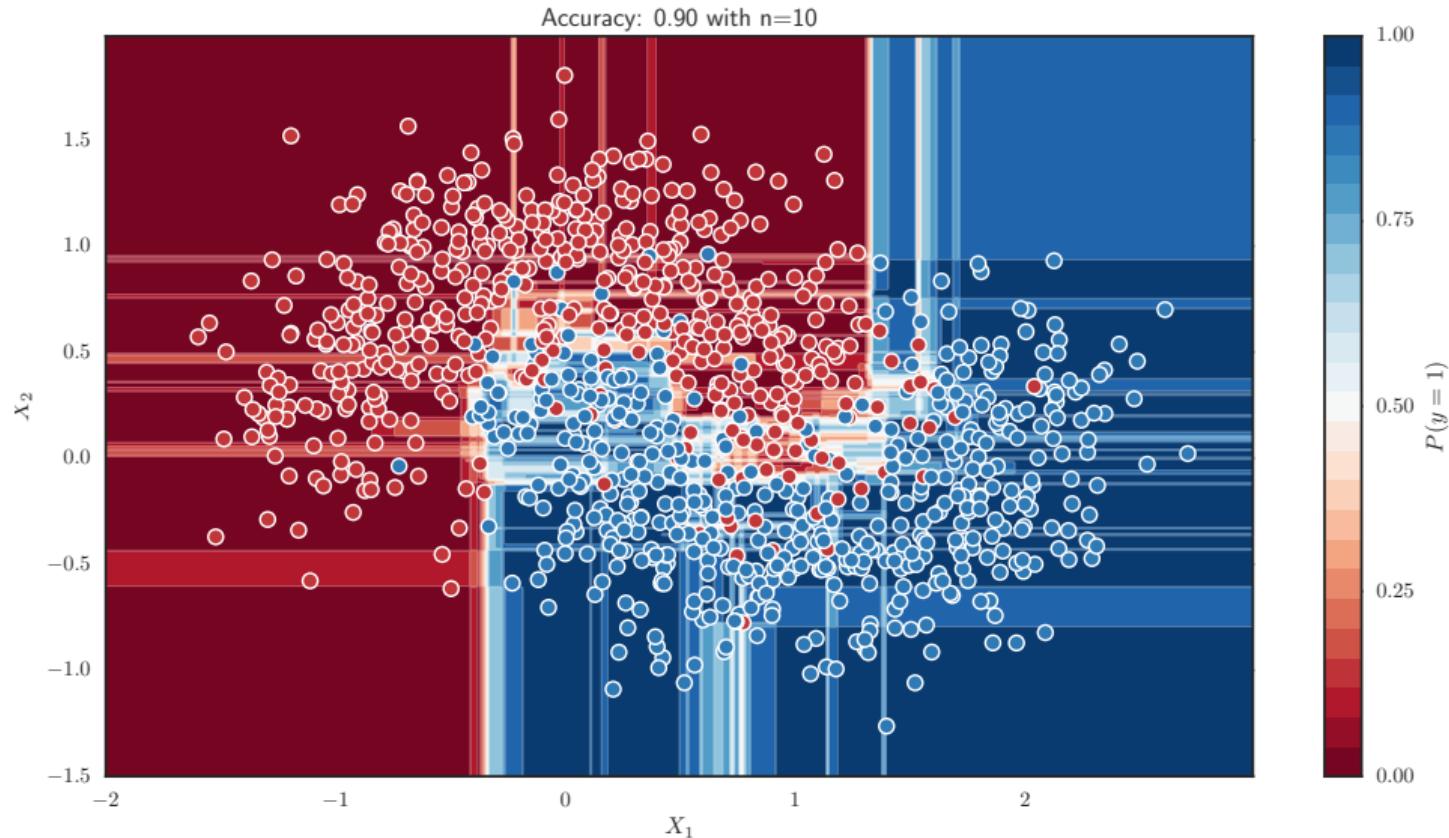
- ▶ Train multiple decision trees on different random sub-samples and aggregate the results for prediction
- ▶ Bootstrapping: sampling with replacement for obtaining training sub-sample for each tree
- ▶ Training and prediction can be parallelised
- ▶ Produce smoother decision boundary and achieve improvement in predictive performance

Python code for random forest using sklearn package

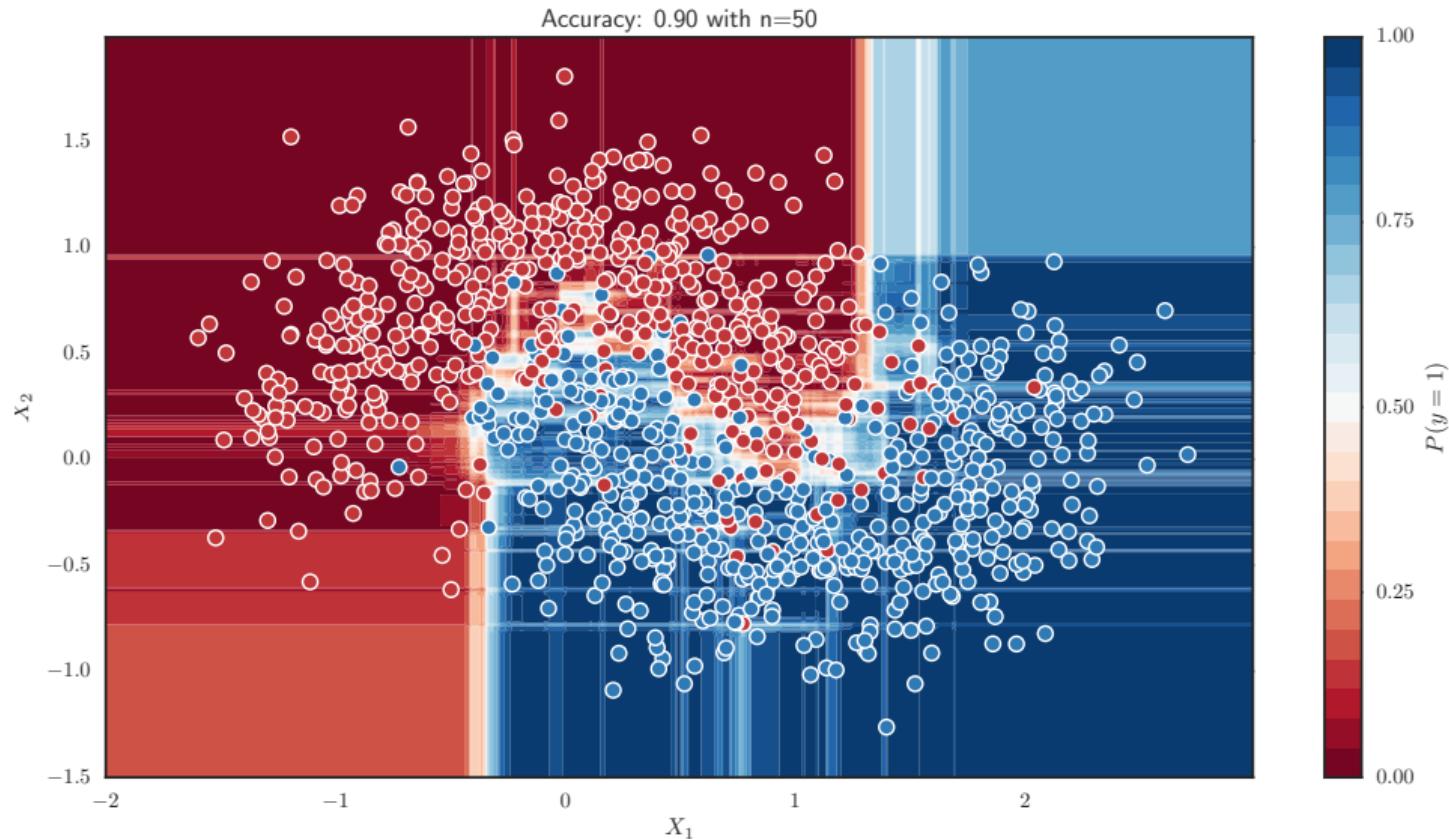
```
import numpy as np
from sklearn.ensemble import RandomForestClassifier
from sklearn.cross_validation import train_test_split
# Generate two moons data
X, y = make_moons(n_samples = 1000, noise=0.3, random_state=0)
# Split data into training and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=.4)

# Define decision tree classifier with n_estimators 10 and max_depth 16.
clf=RandomForestClassifier(n_estimators=10,max_depth=16)
clf.fit(X_train,y_train) # Train it on training set
score = clf.score(X_test, y_test) # Test on test set
```

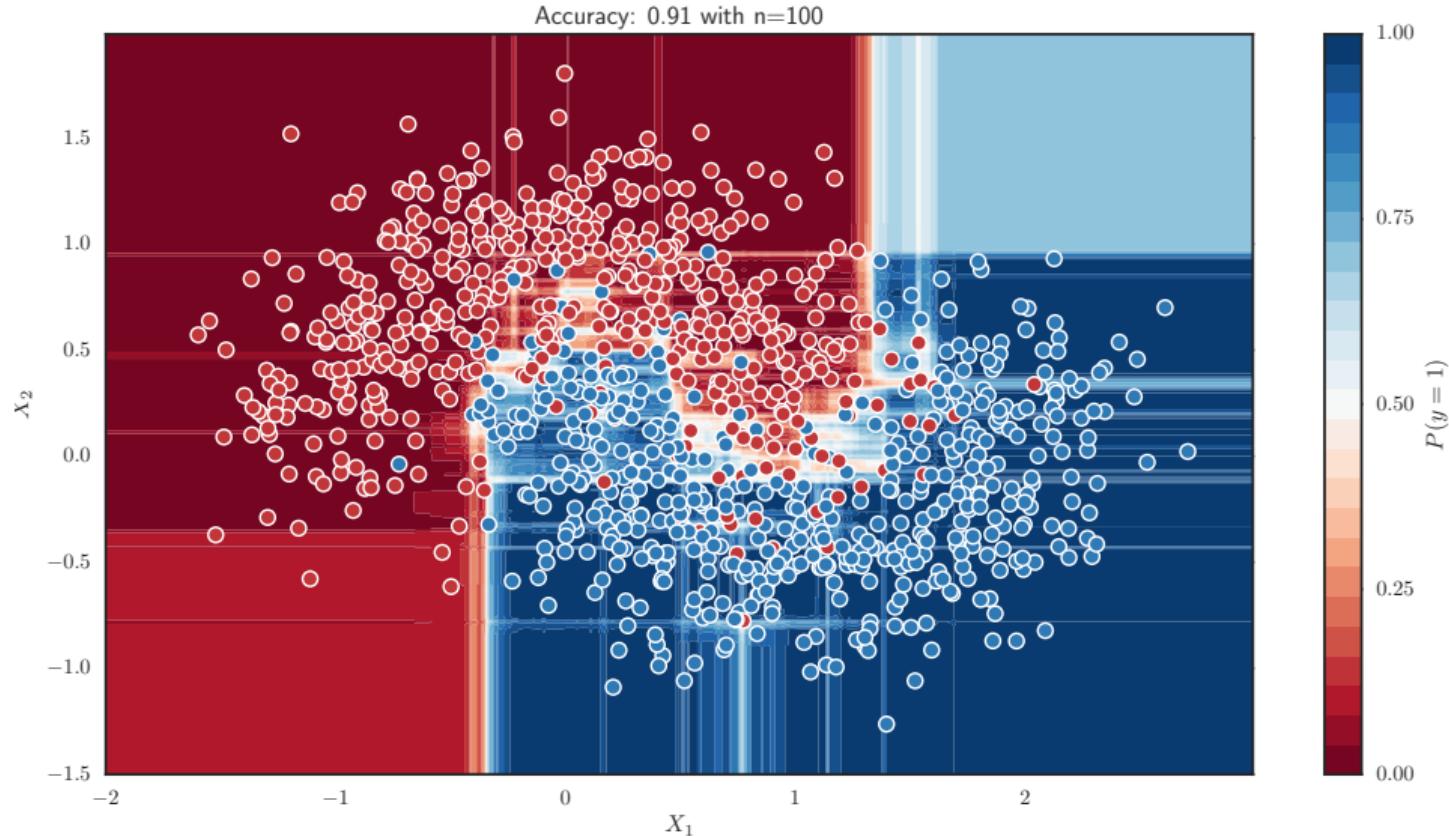
Random forest on Two Moons (N=10, D=16)



Random forest on Two Moons (N=50, D=16)



Random forest on Two Moons (N=100, D=16)



Observations and discussions

- ▶ Increasing the number of the trees (N) usually improves performance of random forest classifier
 - ▶ Performance is not overly sensitive to choice of N
 - ▶ Sufficiently large N will ensure good performance
 - ▶ Robust against overfitting even with very large value of N
 - ▶ Trade-off between speed and performance with N
- ▶ Random forest generates smoother decision boundaries
 - ▶ Aggregation of multiple trees averages out spikes and bumps
- ▶ Performance and speed
 - ▶ Much more efficient than SVM in training
 - ▶ Inherently nonlinear classification model
 - ▶ Better performance than linear SVM

Classification Model Comparison

Speed for big data (fastest to slowest)

- ▶ In general: Decision Tree, Random Forest, Linear SVM, SVM
- ▶ For some applications, Linear SVM may be faster

Predictive Performance (best to worst)

- ▶ Hard to say, very much depends on the dataset
- ▶ In general: SVM>Random Forest>Linear SVM

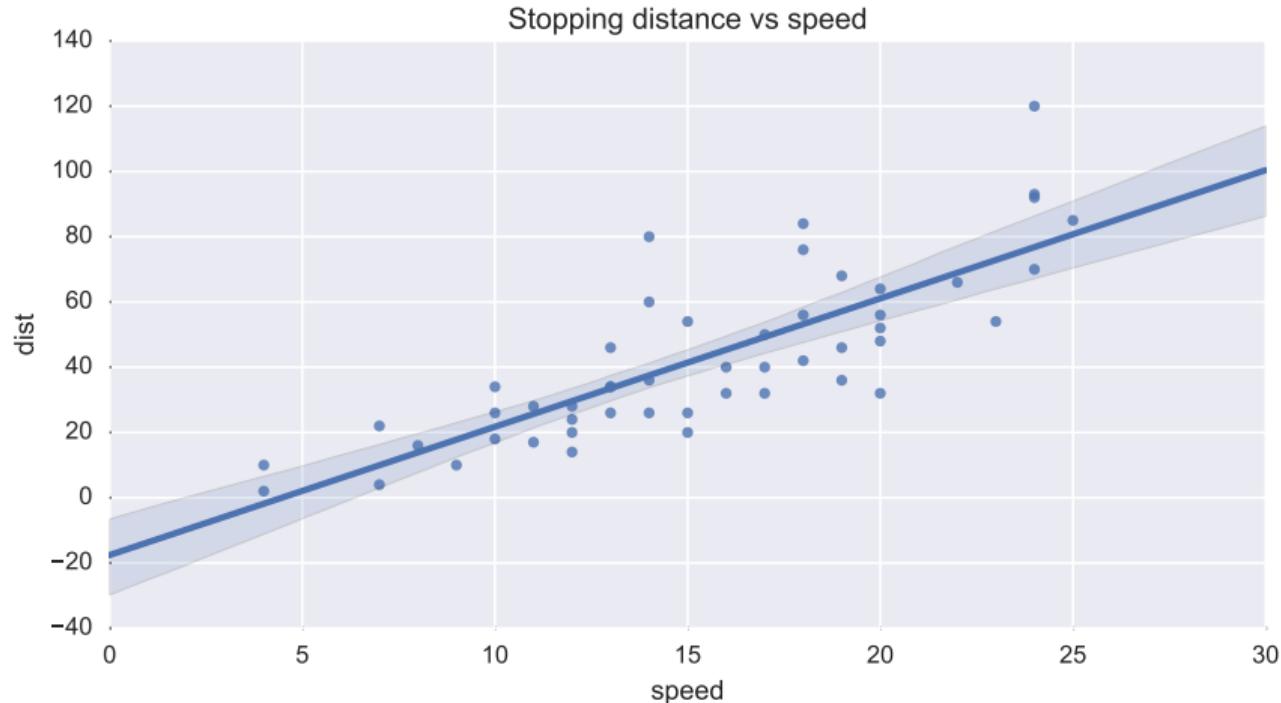
Classification Model Comparison

Best Scenarios

- ▶ SVM:
 - ▶ nonlinear data
 - ▶ high priority on performance
 - ▶ powerful computing facilities are available
- ▶ Linear SVM:
 - ▶ high priority on speed
 - ▶ big data
 - ▶ high-dimensional problems (large number of features)
- ▶ Random Forest:
 - ▶ nonlinear data
 - ▶ many nominal features
 - ▶ need interpretable models

Regression

Regression Problems



Regression vs Classification

Regression

- ▶ Target values: non-categorical output
- ▶ Examples
 - ▶ temperature forecast (target: temperature)
 - ▶ gene expression (target: gene expression level)
 - ▶ Cancer diagnosis (target: probability of getting cancer)

Classification

- ▶ Target values: categorical labels (binary (two classes) and multiclass (>2) classification)
- ▶ Examples
 - ▶ Face detection (target: face/non-face)
 - ▶ Digit recognition (target: digits 0 ~ 9)
 - ▶ Cancer diagnosis (target: positive/negative)

Actually, classification can be regarded as a special case of regression if we remove the *non-categorical* condition from regression.

Regression Models

Linear regressors

The relationship between predictors/covariates/explanatory variables (Y) and response/dependent variable/target \mathbf{X} is linear

$$Y \approx \mathbf{X}^\top \mathbf{w}$$

\mathbf{w} is the vector of parameters of the model (e.g. gradient, intercept)

- ▶ Linear regression
- ▶ Ridge regression
- ▶ Sparse regression
- ▶ ...

Regression Models

Nonlinear regressors

The relationship between predictors/covariates/explanatory variables (Y) and response/dependent variable/target \mathbf{X} is *nonlinear*

$$Y \approx f(\mathbf{X}, \mathbf{w})$$

where f is a nonlinear function, e.g. $f(x) = \exp(x)$.

- ▶ Generalised linear regression
- ▶ Support vector regression
- ▶ Gaussian processes regression
- ▶ ...

Linear Regression by Ordinary Least Squares

```
from sklearn import linear_model

cars = pd.read_csv('cars.csv')
y = np.array(cars['dist'].astype('float'))
x = np.array(cars['speed'].astype('float'))
x_train, x_test, y_train, y_test = train_test_split(x,y,
test_size=0.4,random_state=42)

reg = linear_model.LinearRegression(fit_intercept=True)
reg.fit(x_train.reshape(-1,1),y_train)
y_pred = reg.predict(x_test.reshape(-1,1))
```

Linear Regression by Ordinary Least Squares



Sparse Linear Regression

Important observation

There are many predictors but only a few of them are involved in the prediction for the response.

Example

Diabetes data: Ten baseline variables, *age*, *sex*, *body mass index*, *average blood pressure*, and *six blood serum measurements* were obtained for each of $n = 442$ diabetes patients, as well as the response of interest, a quantitative measure of disease progression one year after baseline.

Question: the disease progression is only determined by a couple of things.
What are they?

Sparse Linear Regression

- ▶ This is really a *variable selection* problem, i.e. select a *the best* combination of variables that can explain the response.
- ▶ A *variable selection* problem is a combinatorial optimisation problem, which is **NP** hard, with $O(2^N)$ complexity.
- ▶ Solutions: approximation, i.e. solve the *variable selection* problem approximately instead of exactly.
 - ▶ Greedy selection methods: forward/backward selection, all subset selection, BP, OMP,
...
 - ▶ Sparse linear regression:
 - ▶ LASSO, $\min \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_1$
 - ▶ Elastic net, $\min \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda_1 \|\beta\|_1 + \lambda_2 \|\beta\|_2$
 - ▶ Many more: $\min \|\mathbf{y} - \mathbf{X}\beta\|_2^2 + \lambda \|\beta\|_p$, where $\|\mathbf{v}\|_p$ is the ℓ_p norm (sparse encourage norm) of \mathbf{v} .

LASSO

Least Absolute Shrinkage and Selection Operator

General regression:

If the model looks like:

$$y_{model} = b_1x_1 + b_2x_2 + b_3x_3 + \dots + b_px_p$$

we need to find \mathbf{b} , i.e. $(b_1, b_2, b_3, \dots, b_p)$

Sparse regression:

zero out the bs that have little effect, i.e. $\mathbf{b} \approx (0, 0, 0, \dots, b_i, \dots, b_m, \dots)$

LASSO:

minimize $(y_{data} - y_{model})^2$ subject to $\sum \text{abs}(b_k) \leq t$

Sparse Linear Regression

```
from sklearn import datasets
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target
X_train, X_test, y_train, y_test = train_test_split(X,y,
    test_size=0.4,random_state=42)
reg = linear_model.Lasso(alpha = 1.5,normalize=True)
reg.fit(X_train,y_train)
print 'Selected: '+str([varnames[i] for i in np.where(reg.coef_!=0)[0]])
```

Selected: ['body mass index', 's5']

Sparse Linear Regression

- ▶ The problem for sparse linear regression is the selection of regularisation parameter λ 's.
- ▶ Solution: path algorithms, i.e. efficient algorithms to find sparse linear regression solutions for an entire range of λ 's.
 - ▶ path algorithms for parse regression models: LASSO path, elastic net path, and so on
 - ▶ Least Angle Regression (LARS)

Sparse Linear Regression

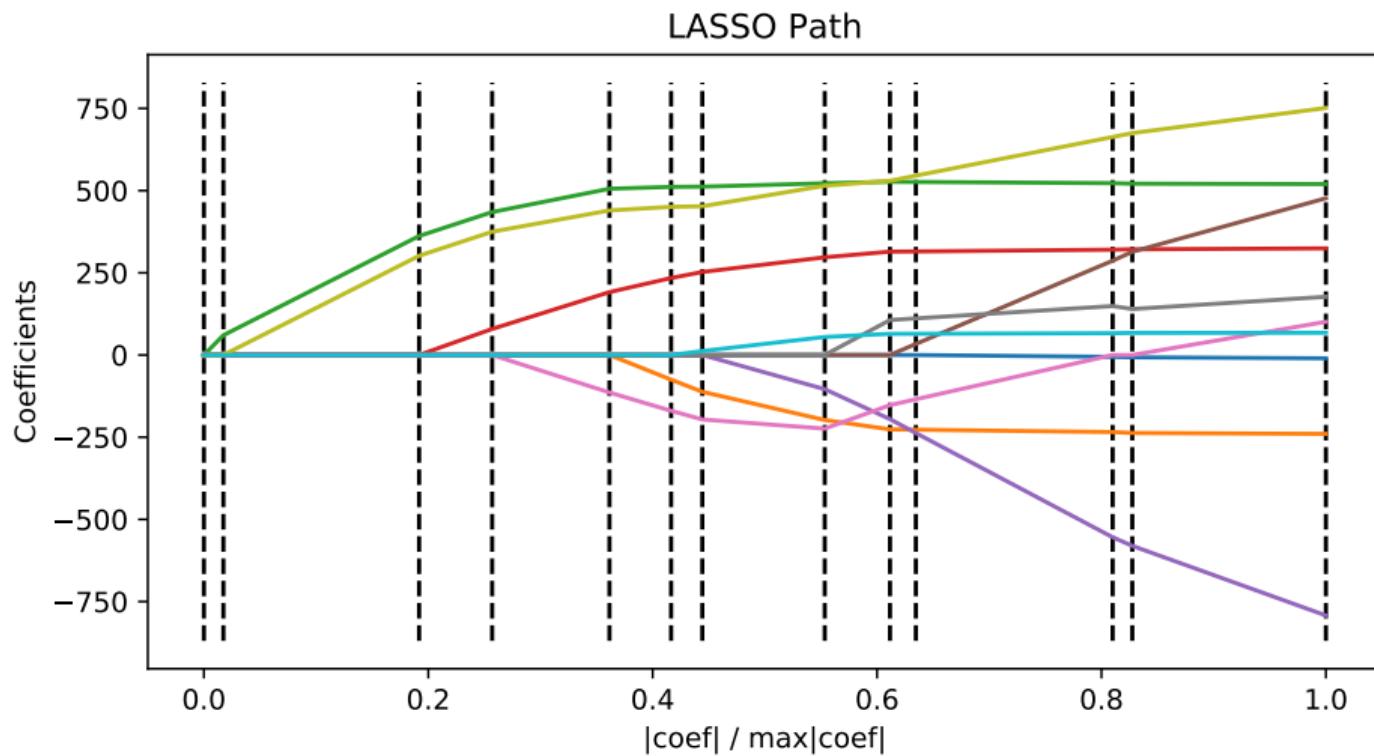
```
import numpy as np
import matplotlib.pyplot as plt
from sklearn import linear_model
from sklearn import datasets
diabetes = datasets.load_diabetes()
X = diabetes.data
y = diabetes.target

alphas, _, coefs = linear_model.lars_path(X, y, method='lasso', verbose=True)

xx = np.sum(np.abs(coefs.T), axis=1)
xx /= xx[-1]

plt.plot(xx, coefs.T)
ymin, ymax = plt.ylim()
plt.vlines(xx, ymin, ymax, linestyle='dashed')
plt.xlabel('|coef| / max|coef|');plt.ylabel('Coefficients')
plt.title('LASSO Path'); plt.axis('tight')
plt.show()
```

Sparse Linear Regression



Following topics

Future topics

Data Analysis for Big Data (part III)

- ▶ Spark & pyspark
- ▶ Deep Learning