pusilkom ui

quality results

Introduction to Python and its Scientific Library for Data Science/Mining

Instructor:
Heru Praptono

Agenda

- Overview on Python and Numpy
  - Why Numpy
  - Some basic popular math programming with Numpy
  - Numpy vs Scipy
- Introduction to Scikit-Learn
  - Some basic well-packaged algorithm in Scikit-Learn
- Some useful IDE’s

SO..

- A very important property of this universe….
- Universe contains **randomness**. From randomness, there is uncertainty. We need to know for at least a phenomena on what’s going on between variables and their interaction.
- Many of us have limited information on it.
- Fortunately, at least there is data emission as the consequences of their existence. → resulting a bunch of data. **Big data**. So good news: we can possibly develop approximation models in order to enable inferences.

Data Mining, in simple practical perspective

- **Data mining** is an **applied area**, considering the perspective of theory of probability, estimation/approximation theory, computation and statistics, and other math related area.
- We have **a bunch of data**. We then first “see” the data, and think what sort of operations/tasks to be implemented to the data, so that we **gather information from data**.
- In general, the task is about to get the **knowledge representations**, that are constructed by **learning from data.**
- The **central concepts/tasks** can be either supervised, or unsupervised

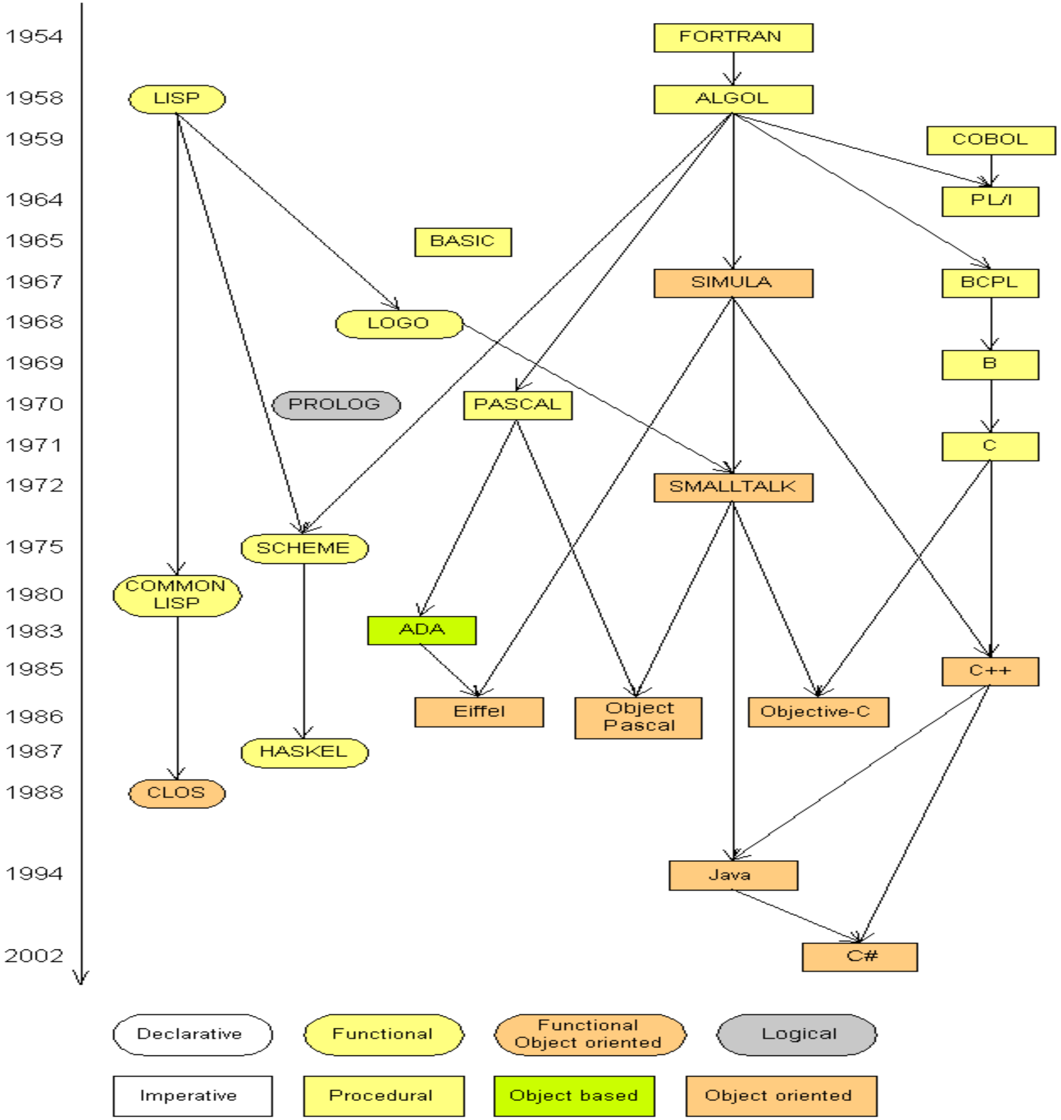
What you need to know afterwards

- We help build your insight on “learn from data” algorithms for data mining from and in big data.
- You will need to familiarise yourself with math and stats, and then need to be able to implement them in Python in efficient ways.
- Of course you have had well-packaged ML tool, but at least you will have understood how it works, and how is it likely to be improved, considering the data you have, and the goal you expect from the data

Overview on Python and Numpy

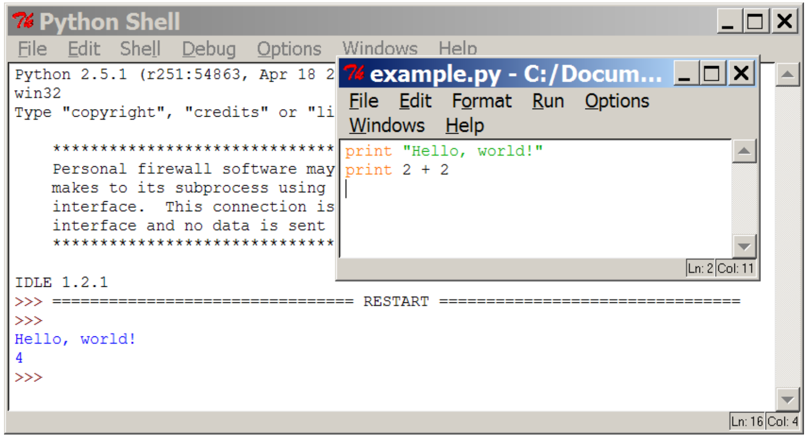
Languages

- Some influential ones:
  - FORTRAN
    - science / engineering
  - COBOL
    - business data
  - LISP
    - logic and AI
  - BASIC
    - a simple language



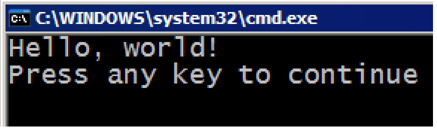
Programming Basics

- **code** or **source code**: The sequence of instructions in a program.
- **syntax**: The set of legal structures and commands that can be used in a particular programming language.
- **output**: The messages printed to the user by a program.
- **console**: The text box onto which output is printed.
- Some source code editors pop up the console as an external window, and others contain their own console window.



```
print "Hello, world!"
print 2 + 2
```

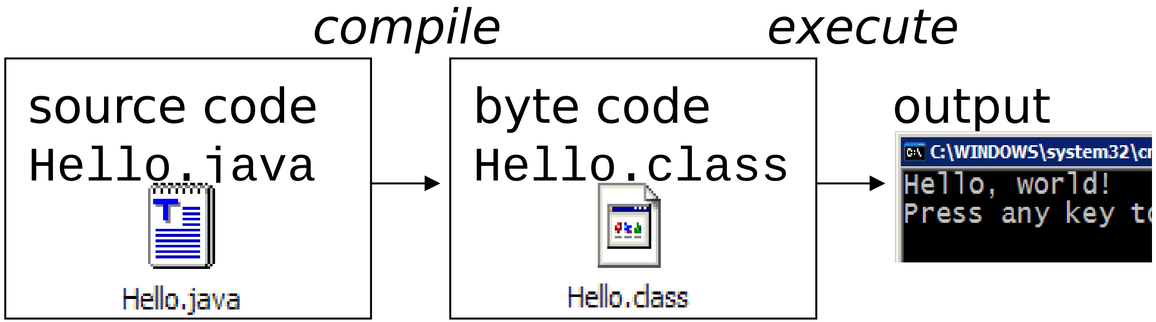
```
Python 2.5.1 (r251:54863, Apr 18 2
win32
Type "copyright", "credits" or "li

    ****************************
    Personal firewall software may
    makes to its subprocess using
    interface.  This connection is
    interface and no data is sent
    ****************************

IDLE 1.2.1
>>> ================================ RESTART ================================
>>>
Hello, world!
4
>>>
```



```
Hello, world!
Press any key to continue
```

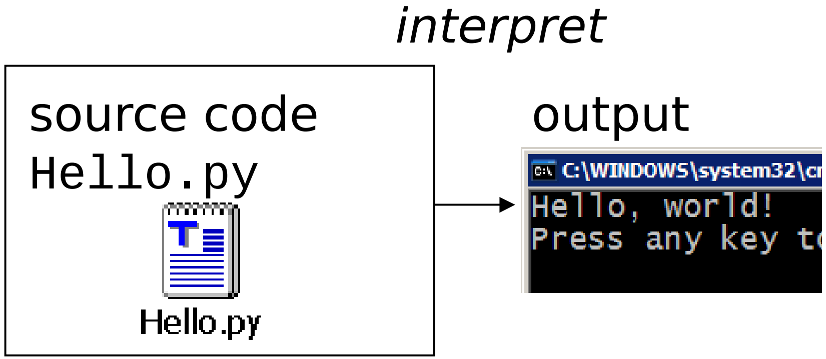
Compiling and Interpreting

- Many languages require you to *compile* (translate) your program into a form that the machine understands.



```
Hello, world!
Press any key to
```

- Python is instead directly *interpreted* into machine instructions.



```
Hello, world!
Press any key to
```

Python, in relation with data science

- Advantages:
  - Very rich scientific computing libraries (a bit less than Matlab, though)
  - Well thought out language, allowing to write very readable and well structured code: we “code what we think”.
  - Many libraries for other tasks than scientific computing (web server management, serial port access, etc.)
  - Free and open-source software, widely spread, with a vibrant community.
- Drawbacks:
  - less pleasant development environment than, for example, Matlab. (More geek-oriented)
  - Not all the algorithms that can be found in more specialized software or toolboxes

Expressions

- **expression**: A data value or set of operations to compute a value.

  Examples:

  ```
  1 + 4 * 3
  42
  ```

- Arithmetic operators we will use:

`+ - * /`	addition, subtraction/negation, multiplication, division
`%`	modulus, a.k.a. remainder
`**`	exponentiation

- **precedence**: Order in which operations are computed.

  `* / % **` have a higher precedence than `+ -`

  `1 + 3 * 4` is `13`

- Parentheses can be used to force a certain order of evaluation.

  `(1 + 3) * 4` is `16`

Integer Division

- When we divide integers with / , the quotient is also an integer.

```
     3                     52
4 ) 14                27 ) 1425
    12                     135
     2                       75
                             54
                             21
```

- More examples:
  35 / 5 is 7
  84 / 10 is 8
  156 / 100 is 1

- The % operator computes the remainder from a division of integers.

```
     3                     43
4 ) 14                 5 ) 218
    12                     20
     2                      18
                            15

                             3
```

Real Numbers

- Python can also manipulate real numbers.
  - Examples: `6.022  -15.9997 42.0   2.143e17`

- The operators `+ - * / % ** ( )` all work for real numbers.
  - The `/` produces an exact answer: `15.0 / 2.0` is **`7.5`**
  - The same rules of precedence also apply to real numbers: Evaluate `( )` before `* / %` before `+ -`

- When integers and reals are mixed, the result is a real number.
  - Example: `1 / 2.0` is `0.5`

- The conversion occurs on a per-operator basis.

```
7 / 3 * 1.2 + 3 / 2
  2   * 1.2 + 3 / 2
    2.4     + 3 / 2
    2.4     +   1

         3.4
```

Math Commands

- Python has useful commands for performing calculations.
-
- To use many of these commands, you must write the following at the top of your Python program:

```
from math import *
```

Command name	Description
abs(***value***)	absolute value
ceil(***value***)	rounds up
cos(***value***)	cosine, in radians
floor(***value***)	rounds down
log(***value***)	logarithm, base *e*
log10(***value***)	logarithm, base 10
max(***value1***, ***value2***)	larger of two values
min(***value1***, ***value2***)	smaller of two values
round(***value***)	nearest whole number
sin(***value***)	sine, in radians
sqrt(***value***)	square root

Constant	Description
e	2.7182818...
pi	3.1415926...

Variables

- **variable**: A named piece of memory that can store a value.
  - Usage:
    - Compute an expression's result,
    - store that result into a variable,
    - and use that variable later in the program.
-
- **assignment statement**: Stores a value into a variable.
  - Syntax:

    name* = *value

- Examples:

```
x = 5
gpa = 3.14
```

- A variable that has been given a value can be used in expressions.

  `x + 4` is 9

- **Exercise:** Evaluate the quadratic equation for a given *a*, *b*, and *c*.

Print

- `print` : Produces text output on the console.
- Syntax:

  `print("`***Message***`")`

  `print(`***Expression)***
- Prints the given text message or expression value on the console, and moves the cursor down to the next line.

  `print(`***Item1*, *Item2*, …, *ItemN)***
- Prints several messages and/or expressions on the same line.
- Examples:

```
print("Hello, world!")
age = 45
print("You have", 65 - age, "years until retirement")
```

- `Output:`

```
Hello, world!
You have 20 years until retirement
```

Input

- `input` : Reads a number from user input.
- You can assign (store) the result of `input` into a variable.
- Example:

  -
    ```
    age = input("How old are you? ")
    print("Your age is", age)
    print("You have", 65 - age, "years until retirement")
    ```
  - Output:

    ```
    How old are you? 53
    Your age is 53
    You have 12 years until retirement
    ```

- **Exercise:** Write a Python program that prompts the user for his/her amount of money, then reports how many Nintendo Wiis the person can afford, and how much more money he/she will need to afford an additional Wii.

Repetition and Selection

- **`for` loop**: Repeats a set of statements over a group of values.
- Syntax:

```
for variableName in groupOfValues:
    statements
```

- We indent the statements to be repeated with tabs or spaces.
- ***variableName*** gives a name to each value, so you can refer to it in the ***statements***.
- ***groupOfValues*** can be a range of integers, specified with the `range` function.

- Example:

```
for x in range(1, 6):
    print(x, "squared is", x * x)
```

Output:

```
1 squared is 1
2 squared is 4
3 squared is 9
4 squared is 16
5 squared is 25
```

Range

- The `range` function specifies a range of integers:

  `range(`***start***`,` ***stop***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive)

- It can also accept a third value specifying the change between values.

  `range(`***start***`,` ***stop***`,` ***step***`)` - the integers between ***start*** (inclusive) and ***stop*** (exclusive) by ***step***

- Example:

```
for x in range(5, 0, -1):
    print(x)
print("Blastoff!")
```

Output:

```
5
4
3
2
1
Blastoff!
```

- **Exercise:** How would we print the "99 Bottles of Beer" song?

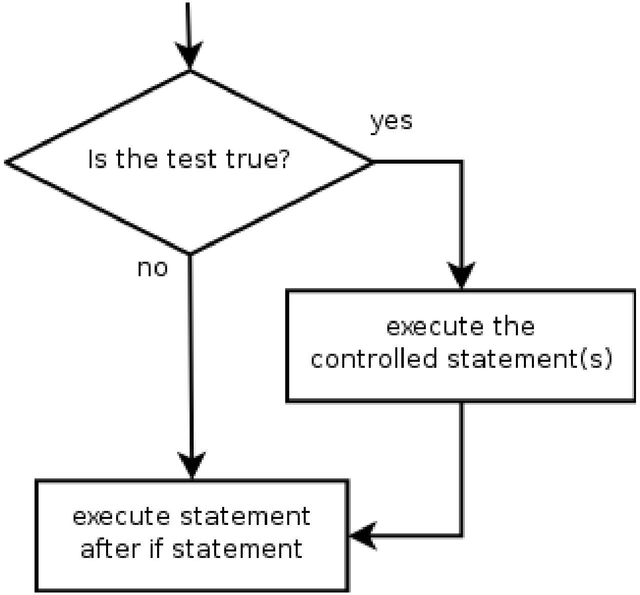
if

- **`if` statement**: Executes a group of statements only if a certain condition is true. Otherwise, the statements are skipped.
- Syntax:

```
if condition:
    statements
```

- Example:

```
gpa = 3.4
if gpa > 2.0:
   print("Your application is accepted.")
```


Is the test true?
yes
no
execute the controlled statement(s)
execute statement after if statement

if/else

- **`if/else` statement**: Executes one block of statements if a certain condition is True, and a second block of statements if it is False.
- Syntax:

```
if condition:
    statements
else:
    statements
```

- Example:

```
gpa = 1.4
if gpa > 2.0:
    print("Welcome to Mars University!")
else:
    print("Your application is denied.")
```

- Multiple conditions can be chained with `elif` ("else if"):

```
if condition:
    statements
elif condition:
    statements
else:
    statements
```

while

`while` loop: Executes a group of statements as long as a condition is True.

- – good for *indefinite loops* (repeat an unknown number of times)

- Syntax:

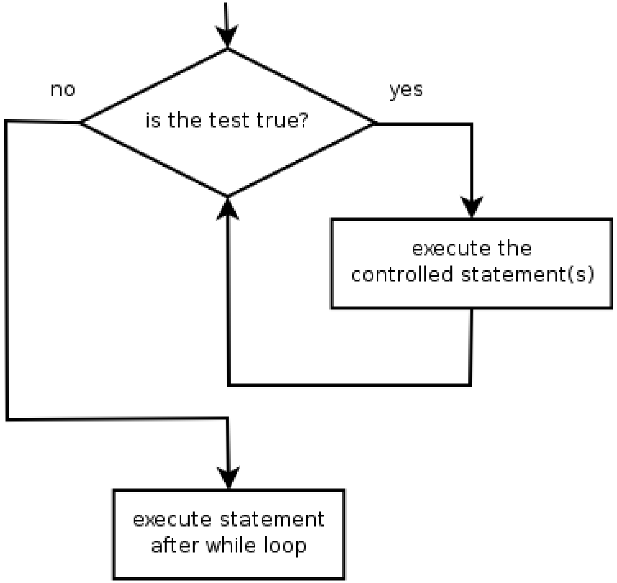
```
while condition:
    statements
```

- Example:

```
number = 1
while number < 200:
    print(number,)
    number = number * 2
```

- Output:

```
1 2 4 8 16 32 64 128
```



no
yes
is the test true?
execute the controlled statement(s)
execute statement after while loop

Logic

- Many logical expressions use *relational operators*:

Operator	Meaning	Example	Result
`==`	equals	`1 + 1 == 2`	`True`
`!=`	does not equal	`3.2 != 2.5`	`True`
`<`	less than	`10 < 5`	`False`
`>`	greater than	`10 > 5`	`True`
`<=`	less than or equal to	`126 <= 100`	`False`
`>=`	greater than or equal to	`5.0 >= 5.0`	`True`

- Logical expressions can be combined with *logical operators*:

Operator	Example	Result
`and`	`9 != 6 and 2 < 3`	`True`
`or`	`2 == 3 or -1 < 5`	`True`
`not`	`not 7 > 0`	`False`

Find by yourself..

- List
- Dictionary
- Classes
- ..and other useful Python features

How are these things related each other, in relation with our syllabus (learn from data)?

Well packaged ML

scikit learn ✓

SciPy

✓ NumPy

Stat & math computing

Pandas

matplotlib

Seaborn

Data Framing & Visualisation

Enable Distributed Computing

PySpark ✓

hadoop ✓

python™

Numpy

- a Python extension module that provides efficient operation on arrays of homogeneous data
- allows python to serve as a high-level language for manipulating numerical data, much like IDL, MATLAB, or Yorick
- enabling Linear Algebra (numpy.linalg)
- Vectorisation (e.g. change primitive foor loop into inner product)

Numpy - ndarray

- NumPy's main object is the homogeneous multidimensional array called `ndarray`.
  - This is a table of elements (usually numbers), all of the same type, indexed by a tuple of positive integers. Typical examples of multidimensional arrays include vectors, matrices, images and spreadsheets.
  - Dimensions usually called axes, number of axes is the rank

[7, 5, -1]

An array of rank 1 i.e. It has 1 axis of length 3

[[1.5, 0.2, -3.7] ,
[0.1, 1.7, 2.9]]

An array of rank 2 i.e. It has 2 axes, the first length 3, the second of length 3 (a matrix with 2 rows and 3 columns

Numpy – array creation and use

```
>>> a = numpy.arange(4.0)
>>> b = a * 23.4
>>> c = b/(a+1)
>>> c += 10
>>> print c
[ 10.   21.7  25.6  27.55]

>>> arr = numpy.arange(100, 200)
>>> select = [5, 25, 50, 75, -5]
>>> print(arr[select])  # can use integer lists as indices
[105, 125, 150, 175, 195]

>>> arr = numpy.arange(10, 20 )
>>> div_by_3 = arr%3 == 0  # comparison produces boolean array
>>> print(div_by_3)
[ False False  True False False  True False False  True False]
>>> print(arr[div_by_3])  # can use boolean lists as indices
[12 15 18]

>>> arr = numpy.arange(10, 20) . reshape((2,5))
[[10 11 12 13 14]
 [15 16 17 18 19]]
```

Numpy – array creation and use

```
>>> arr.sum()
145
>>> arr.mean()
14.5
>>> arr.std()
2.8722813232690143
>>> arr.max()
19
>>> arr.min()
10
>>> div_by_3.all()
False
>>> div_by_3.any()
True
>>> div_by_3.sum()
3
>>> div_by_3.nonzero()
(array([2, 5, 8]),)
```

Numpy – array - Sorting

```
>>> arr = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> arr.sort()  # acts on array itself
>>> print(arr)
[ 1.2  1.8  2.3  4.5  5.5  6.7]

>>> x = numpy.array([4.5, 2.3, 6.7, 1.2, 1.8, 5.5])
>>> numpy.sort(x)
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])

>>> print(x)
[ 4.5  2.3  6.7  1.2  1.8  5.5]

>>> s = x.argsort()
>>> s
array([3, 4, 1, 0, 5, 2])
>>> x[s]
array([ 1.2,  1.8,  2.3,  4.5,  5.5,  6.7])
>>> y[s]
array([ 6.2,  7.8,  2.3,  1.5,  8.5,  4.7])
```

Numpy – Array Operations

```
>>> a = array([[1.0, 2.0], [4.0, 3.0]])
>>> print a
[[ 1. 2.]
 [ 3. 4.]]

>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])

>>> inv(a)
array([[-2. , 1. ],
       [ 1.5, -0.5]])

>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"

>>> u
array([[ 1., 0.],
       [ 0., 1.]])

>>> j = array([[0.0, -1.0], [1.0, 0.0]])

>>> dot (j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
```

Numpy - Statistics

In addition to the mean, var, and std functions, NumPy supplies several other methods for returning statistical features of arrays. The median can be found:

```
>>> a = np.array([1, 4, 3, 8, 9, 2, 3], float)
>>> np.median(a)
3.0
```

The correlation coefficient for multiple variables observed at multiple instances can be found for arrays of the form [[x1, x2, …], [y1, y2, …], [z1, z2, …], …] where x, y, z are different observables and the numbers indicate the observation times:

```
>>> a = np.array([[1, 2, 1, 3], [5, 3, 1, 8]], float)
>>> c = np.corrcoef(a)
>>> c
array([[ 1.        ,  0.72870505],
       [ 0.72870505,  1.        ]])
```

Here the return array c[i,j] gives the correlation coefficient for the ith and jth observables. Similarly, the covariance for data can be found::

```
>>> np.cov(a)
array([[ 0.91666667,  2.08333333],
       [ 2.08333333,  8.91666667]])
```

Using arrays wisely

- Optimised algorithms - i.e. fast!
- Python loops (i.e. for i in a:…) are much slower
- Prefer array operations over loops, especially when speed important
- Also produces shorter code, often more readable

Numpy – arrays, matrices

For **two dimensional** arrays NumPy defined a special matrix class in module matrix. Objects are created either with matrix() or mat() or converted from an array with method asmatrix().

```
>>> import numpy
>>> m = numpy.mat([[1,2],[3,4]])
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.mat(a)
or
>>> a = numpy.array([[1,2],[3,4]])
>>> m = numpy.asmatrix(a)
```

Note that the statement m = mat(a) creates a copy of array 'a'.
Changing values in 'a' will not affect 'm'.
On the other hand, method m = asmatrix(a) returns a new reference to the same data.
Changing values in 'a' will affect matrix 'm'.

Numpy - matrices

```
>>> a = array([[1,2],[3,4]])
>>> m = mat(a) # convert 2-d array to matrix
>>> m = matrix([[1, 2], [3, 4]])
>>> a[0]                # result is 1-dimensional
array([1, 2])
>>> m[0]                # result is 2-dimensional
matrix([[1, 2]])
>>> a*a                 # element-by-element multiplication
array([[ 1, 4], [ 9, 16]])
>>> m*m                 # (algebraic) matrix multiplication
matrix([[ 7, 10], [15, 22]])
>>> a**3                # element-wise power
array([[ 1, 8], [27, 64]])
>>> m**3                # matrix multiplication m*m*m
matrix([[ 37, 54], [ 81, 118]])
>>> m.T                 # transpose of the matrix
matrix([[1, 3], [2, 4]])
>>> m.H                 # conjugate transpose (differs from .T for complex matrices)
matrix([[1, 3], [2, 4]])
>>> m.I                 # inverse matrix
matrix([[-2. , 1. ], [ 1.5, -0.5]])
```

Numpy - matrices

- Operator *, dot(), and multiply():
  - For array, '*' **means element-wise multiplication**, and the dot() function is used for matrix multiplication.
  - For matrix, '*'**means matrix multiplication**, and the multiply() function is used for element-wise multiplication.
- Handling of vectors (rank-1 arrays)
  - For array, the vector shapes 1xN, Nx1, and N are all different things. Operations like A[:,1] return a rank-1 array of shape N, not a rank-2 of shape Nx1. Transpose on a rank-1 array does nothing.
  - For matrix, rank-1 arrays are always upgraded to 1xN or Nx1 matrices (row or column vectors). A[:,1] returns a rank-2 matrix of shape Nx1.
- Handling of higher-rank arrays (rank > 2)
  - array objects can have rank > 2.
  - matrix objects always have exactly rank 2.
- Convenience attributes
  - array has a .T attribute, which returns the transpose of the data.
  - matrix also has .H, .I, and .A attributes, which return the conjugate transpose, inverse, and asarray() of the matrix, respectively.
- Convenience constructor
  - The array constructor takes (nested) Python sequences as initializers. As in array([[1,2,3],[4,5,6]]).
  - The matrix constructor additionally takes a convenient string initializer. As in matrix("[1 2 3; 4 5 6]")

Numpy – array mathematics

```
>>> a = np.array([1,2,3], float)
>>> b = np.array([5,2,6], float)
>>> a + b
array([6., 4., 9.])
>>> a – b
array([-4., 0., -3.])
>>> a * b
array([5., 4., 18.])
>>> b / a
array([5., 1., 2.])
>>> a % b
array([1., 0., 3.])
>>> b**a
array([5., 4., 216.])
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)

>>> a * a
array([[  1.,   4.],
       [  9.,  16.],
       [ 25.,  36.]])
>>> b * b
array([ 1.,  9.])
>>> a * b
array([[ -1.,   6.],
       [ -3.,  12.],
       [ -5.,  18.]])
>>>
```

```
>>> a = np.array([[1, 2], [3, 4], [5, 6]], float)
>>> b = np.array([-1, 3], float)
>>> a
array([[ 1.,  2.],
       [ 3.,  4.],
       [ 5.,  6.]])
>>> b
array([-1.,  3.])
>>> a + b
array([[ 0.,  5.],
       [ 2.,  7.],
       [ 4.,  9.]])
```

Numpy – array mathematics

```
>>> A = np.array([[n+m*10 for n in range(5)] for m in range(5)])
>>> v1 = arange(0, 5)
>>> A
array([[ 0, 1, 2, 3, 4],
[10, 11, 12, 13, 14],
[20, 21, 22, 23, 24],
[30, 31, 32, 33, 34],
[40, 41, 42, 43, 44]])
>>> v1
array([0, 1, 2, 3, 4])
>>> np.dot(A,A)
array([[ 300,  310,  320,  330,  340],
       [1300, 1360, 1420, 1480, 1540],
       [2300, 2410, 2520, 2630, 2740],
       [3300, 3460, 3620, 3780, 3940],
       [4300, 4510, 4720, 4930, 5140]])
>>>
>>> np.dot(A,v1)
array([ 30, 130, 230, 330, 430])
>>> np.dot(v1,v1)
30
>>>
```

Numpy – array mathematics

Alternatively, we can cast the array objects to the type matrix. This changes the behavior of the standard arithmetic operators +, -, * to use matrix algebra.

```
>>> M = np.matrix(A)
>>> v = np.matrix(v1).T
>>> v
matrix([[0],
        [1],
        [2],
        [3],
        [4]])
>>> M*v
matrix([[ 30],
        [130],
        [230],
        [330],
        [430]])
>>> v.T * v   # inner product
matrix([[30]])
# standard matrix algebra applies
>>> v + M*v
matrix([[ 30],
        [131],
        [232],
        [333],
        [434]])
```

In relation with SciPy

- The NumPy (Numeric Python) package provides basic routines for manipulating large arrays and matrices of numeric data. The SciPy (Scientific Python) package extends the functionality of NumPy with a substantial collection of useful algorithms, like minimization, Fourier transformation, regression, and other applied mathematical techniques
- Explore SciPy on your own!

How are these things related each other, in relation with our syllabus (learn from data)?

Well packaged ML

scikit learn ✓

SciPy

✓ NumPy

Stat & math computing

Pandas

matplotlib

Seaborn

Data Framing & Visualisation

Enable Distributed Computing

PySpark ✓

hadoop ✓

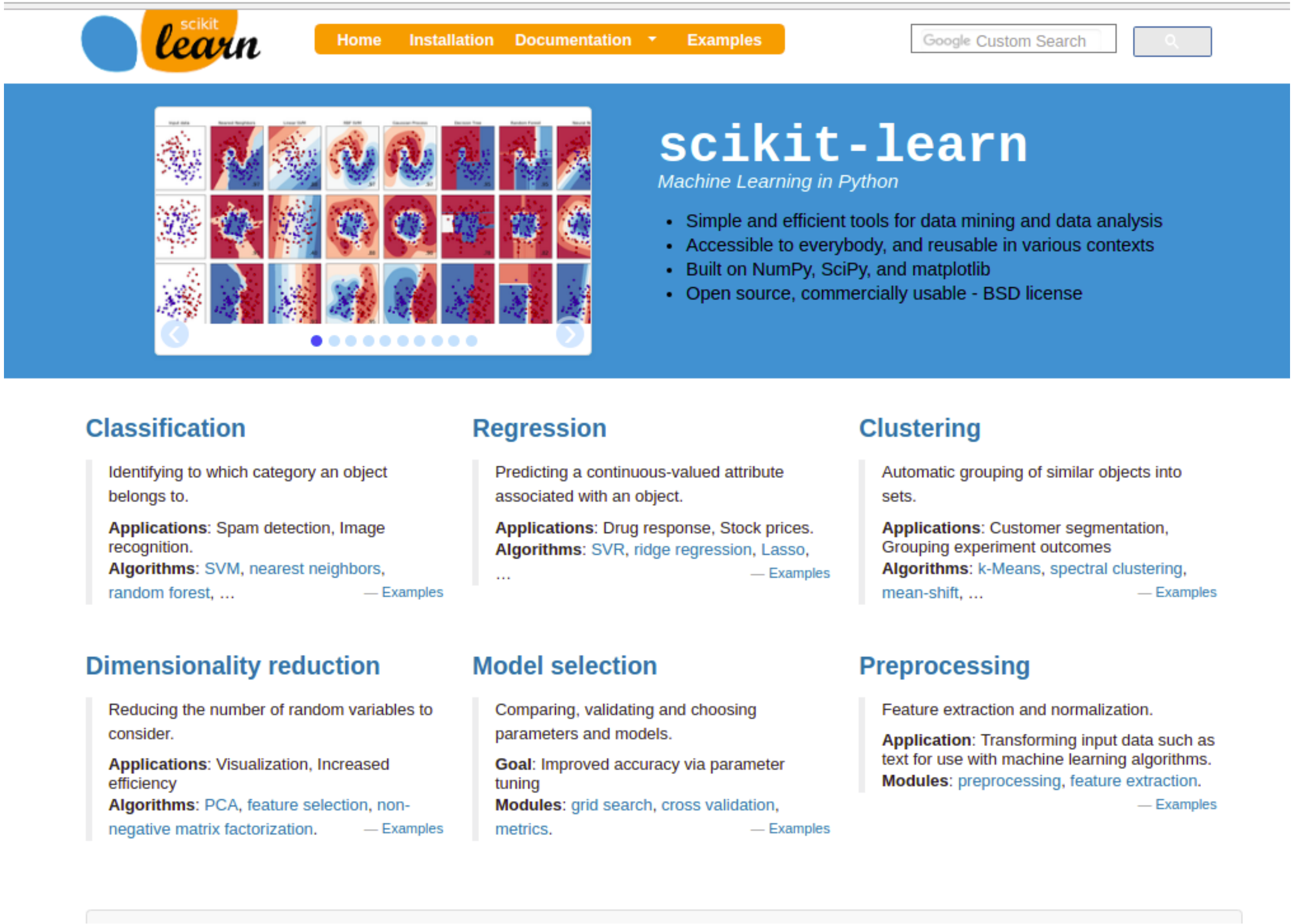
python™

Overview Scikit-Learn

Motivation

- Most of our activity in data science for data mining, consists of either finding pattern in data, or function fitting.
- We may develop from scratch, but….
- There is some general well-packaged available, for those who want just to apply to the data.

In Scikit-Learn



scikit-learn

Machine Learning in Python

- Simple and efficient tools for data mining and data analysis
- Accessible to everybody, and reusable in various contexts
- Built on NumPy, SciPy, and matplotlib
- Open source, commercially usable - BSD license

Classification

Identifying to which category an object belongs to.

Applications: Spam detection, Image recognition.
Algorithms: SVM, nearest neighbors, random forest, ...

— Examples

Regression

Predicting a continuous-valued attribute associated with an object.

Applications: Drug response, Stock prices.
Algorithms: SVR, ridge regression, Lasso, ...

— Examples

Clustering

Automatic grouping of similar objects into sets.

Applications: Customer segmentation, Grouping experiment outcomes
Algorithms: k-Means, spectral clustering, mean-shift, ...

— Examples

Dimensionality reduction

Reducing the number of random variables to consider.

Applications: Visualization, Increased efficiency
Algorithms: PCA, feature selection, non-negative matrix factorization.

— Examples

Model selection

Comparing, validating and choosing parameters and models.

Goal: Improved accuracy via parameter tuning
Modules: grid search, cross validation, metrics.

— Examples

Preprocessing

Feature extraction and normalization.

Application: Transforming input data such as text for use with machine learning algorithms.
Modules: preprocessing, feature extraction.

— Examples

So, what’s on Scikit Learn

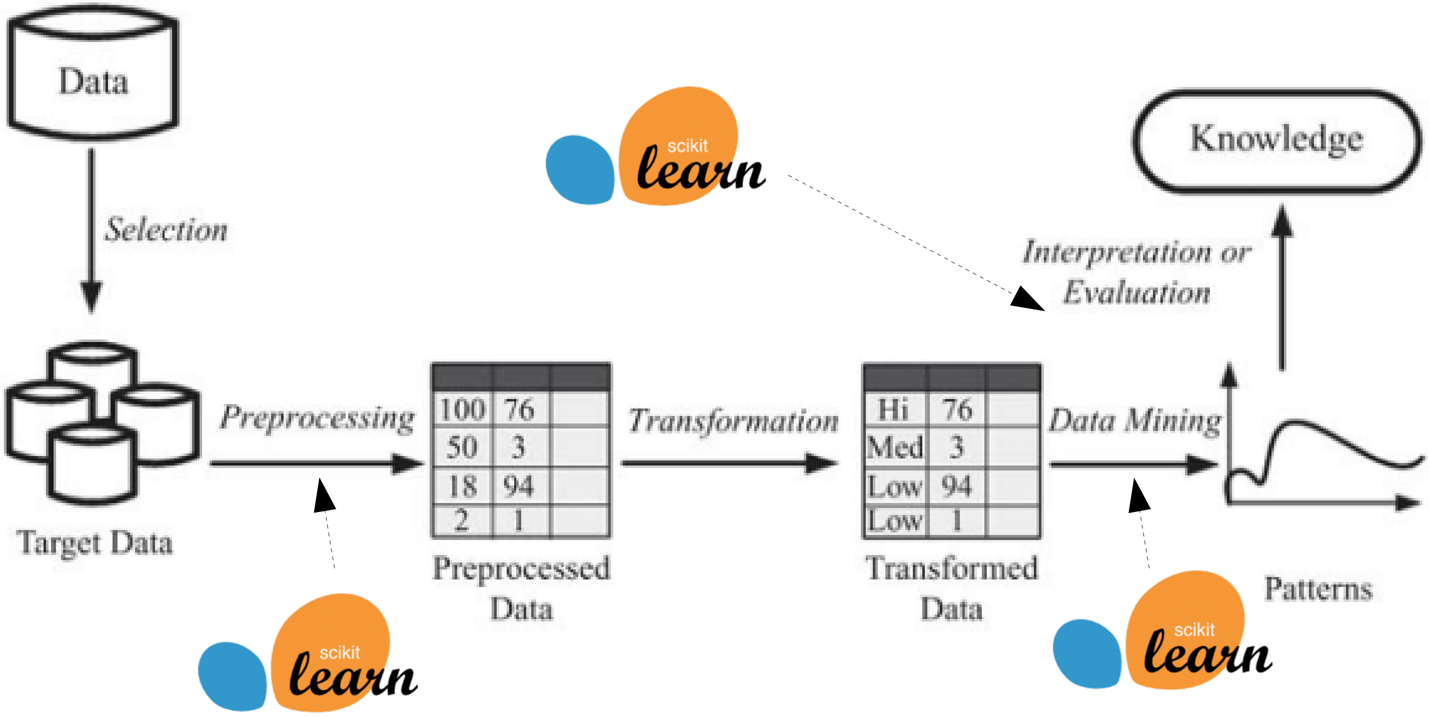
- In general, a learning problem considers a set of n samples of data and then tries to predict properties of unknown data.
- If each sample is more than a single number and, for instance, a multi-dimensional entry (aka multivariate data), it is said to have several attributes or features. **X =** $(x_{1,} x_{2,} \ldots x_D)$

Learning Problems

- supervised learning, in which the data comes with additional attributes that we want to predict (Click here to go to the scikit-learn supervised learning page).This problem can be either:
  - classification: samples belong to two or more classes and we want to learn from already **labeled** data how to predict the class of unlabeled data.
  - regression: if the desired output consists of one or more **continuous** variables
- unsupervised learning, in which the training data consists of a set of input vectors x **without** any corresponding **target** values.
  - E.g. discover groups of similar examples within the data, where it is called clustering, or to determine the distribution of data within the input space, known as density estimation
  - or to project the data from a high-dimensional space down to two or three dimensions for the purpose of visualization (e.g. PCA, Autoencoder)

Insight from Previous Session

KD, on Rahmad Mahendra's slide



Supporting tools: Supervised p(y|**x, param**)

- Regression
  - General Linear Models (..and it’s variation)
    - OLS, Bayesian Regression, Ridge Regression
  - Linear/Quadratic Discriminant Analysis
  - Gaussian Process (GP)
- Classification
  - Naive Bayes
  - Decision Tree
  - Neural Network
  - Gaussian Process (GP)
  - ...ensemble methods, multiclass/multilabel scenario
  - And the others..

Supporting tools: UN-Supervised p(**x**|**param**)

- Sklearn.cluster
  - KMeans
  - DBSCAN
  - MeanShift
  - SpectralClustering
  - ...and so on..

Preprocessing

- Sklearn.preprocessing
  - provides several common utility functions and transformer classes to change raw feature vectors into a representation that is more suitable for the downstream estimators
  - e.g scale, normalize, min_max_scale, maxabs_scale, binarize, etc.

Tools/IDE

Tools/IDE

- Terminal
- Notepad
- Jupyter Notebook
- PyCharm
- PyDev for Eclipse

See you at hands on session

Combination from some resources:
Marty Stepp (CS Washington)
Scikit-Learn