TACC
TEXAS ADVANCED COMPUTING CENTER
WWW.TACC.UTEXAS.EDU

TEXAS
The University of Texas at Austin

Python for Machine Learning at TACC

Amit Gupta
Data Mining & Statistics
Texas Advanced Computing Center
University of Texas at Austin
April 16th, 2019

Slides

https://cutt.ly/2eJNUI

TACC AT A GLANCE

- **Personnel**
  - 150 Staff (~70 PhD)
- **Facilities**
  - 12 MW Data center capacity
  - Two office buildings, Three Datacenters, two visualization facilities, and a chilling plant.
- **Systems and Services**
  - A Billion compute hours per year
  - 5 Billion files, 50 Petabytes of Data, Hundreds of Public Datasets
- **Capacity & Services**
  - HPC, HTC, Visualization, Large scale data storage, Cloud computing
  - Consulting, Curation and analysis, Code optimization, Portals and Gateways, Web service APIs, Training and Outreach

Stampede
HPC Jobs
6400+ Nodes
10 PFlops
14+ PB Storage

Maverick
Vis & Analysis
Interactive
Access
132 K40 GPUs

Lonestar
HTC Jobs
1800+ Nodes
22000+ Cores
146 GB/node

Wrangler
Data Intensive
Computations
10 PB Storage
High IOPS

Stockyard
Shared Workspace
20 PB Storage
1 TB per user
Project Workspace

Corral
Data Collections
6 PB Storage
Databases
IRODS

Rodeo/
Chameleon
Cloud Services
User VMs

Vis Lab
mmersive Vis
Colaborative
ouch Screen
3D

Ranch
Tape Archive
160 PB Tape
1+ PB Access
Cache

High Performance Computing and Machine Learning

Stampede 2 – 18 Pflops for simulation and modeling
- Mixture of Xeon and Xeon Phi systems
- Larger per node memory footprint

Maverick2 – GPU Enabled system
- GTX - 23 Nodes, 4x Nvidia 1080 TI GPUs per node
- V100 - 4 Nodes, 2x Nvidia V100 GPUs per node
- P100 – 3 Nodes, 2x Nvidia P100 GPUs per node
- Machine Learning system optimized for GPU (Tensorflow, Caffe)

Wrangler – 96 nodes cluster
- Hadoop based for data intensive computing
- 10PB disk based Lustre file system
- Flash storage system


FRONTERA
TACC
NSF
TEXAS

FRONTERA

- Coming soon in 2019
- Aimed at the largest problems scientists and engineers currently face
- TACC will Support and operate this system for 5 years.
- Plan a potential phase 2 system, with 10x the capabilities, for the future challenges scientists will face.

FRONTERA - Hardware

- Primary compute system: DellEMC and Intel
  - 35-40 PetaFlops Peak Performance (Next Generation Xeon processors)
- Interconnect: Mellanox HDR and HDR-100 links.
  - Fat Tree topology, 200Gb/s links between switches.
- Storage: DataDirect Networks
  - 50+ PB disk, 3PB of Flash, 1.5TB/sec peak I/O rate.
- Single Precision Compute Subsystem: Nvidia
- Front end for data movers, workflow, API

Data Analysis Support at TACC

Agenda

- What is Python?
  - Overview
  - Uses
- Accessing Python on TACC Machines
  - Jupyter Notebook
- Python Language Basics
- Python Customization on TACC Systems
  - Installing Modules/Packages locally

Python

- Easy to learn & use
- Ubiquitous, Great for quick prototyping
- Supports various programming styles
  - Procedural, OO, Functional etc
- Dynamically typed
  - A variable can be assigned to any data type
- Strongly typed
  - Once assigned, it remains as that type
- Documentation
  - https://docs.python.org

Python (2)

- Modular Organization
  - Most of the utility comes from third party modules/libraries
  - Examples:
    - MatplotLib – Plotting and Visualization
    - Scikit-Learn – Machine Learning
- Error Handing
  - Try , Except
- Machine Learning
  - Data Cleaning and Preparation
- Versions
  - Python 2.7 - Legacy, More mature Libraries
    - https://pythonclock.org/ (Retires Jan 1st, 2020)
  - Python 3 - Future, New Language Features

Accessing Python

- Command Line (Interactive Prompt)
  - SSH into Wrangler/TACC Machine
    - Run ”ssh username@stampede2.tacc.utexas.edu ”
    - 2 factor authentication (TACC Token App)
  - Load the python module
    - Run “module spider python/2.7.13
    - Run “module load intel/17.0.4 python/2.7.13

```
login4(1001)$ module load intel/17.0.4 python/2.7.13

Lmod is automatically replacing "gcc/7.1.0" with "intel/17.0.4"


Activating Modules:
  1) libfabric/1.7.0

Due to MODULEPATH changes, the following have been reloaded:
  1) impi/17.0.3     2) python/2.7.13     3) python2/2.7.14

login4(1002)$
```

Accessing Python

- Command Line, Python Script
  - Write a script in a text file and run
  - “which python” gives path to interpreter
  - Be sure to give your script execute permissions

```
#!/opt/apps/intel15/python/\
2.7.13/bin/python

# Comments

#... your code here ...
# ...................
print "hello world"

-uu-:**-F1   test_python.py
```

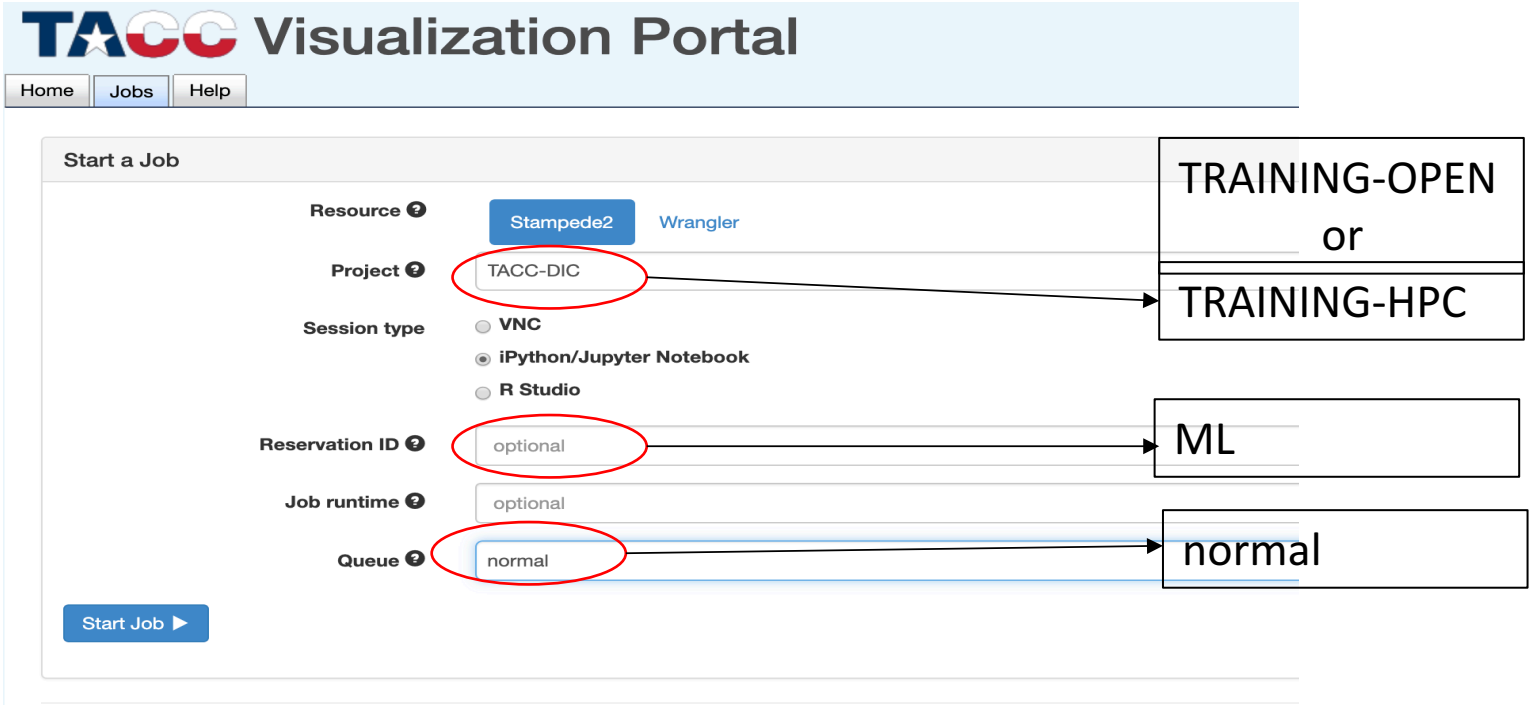
```
login1.wrangler(21)$
login1.wrangler(21)$ chmod +x test.py
login1.wrangler(22)$
login1.wrangler(22)$ python test.py
hello world
login1.wrangler(23)$
login1.wrangler(23)$
```

Development Environment

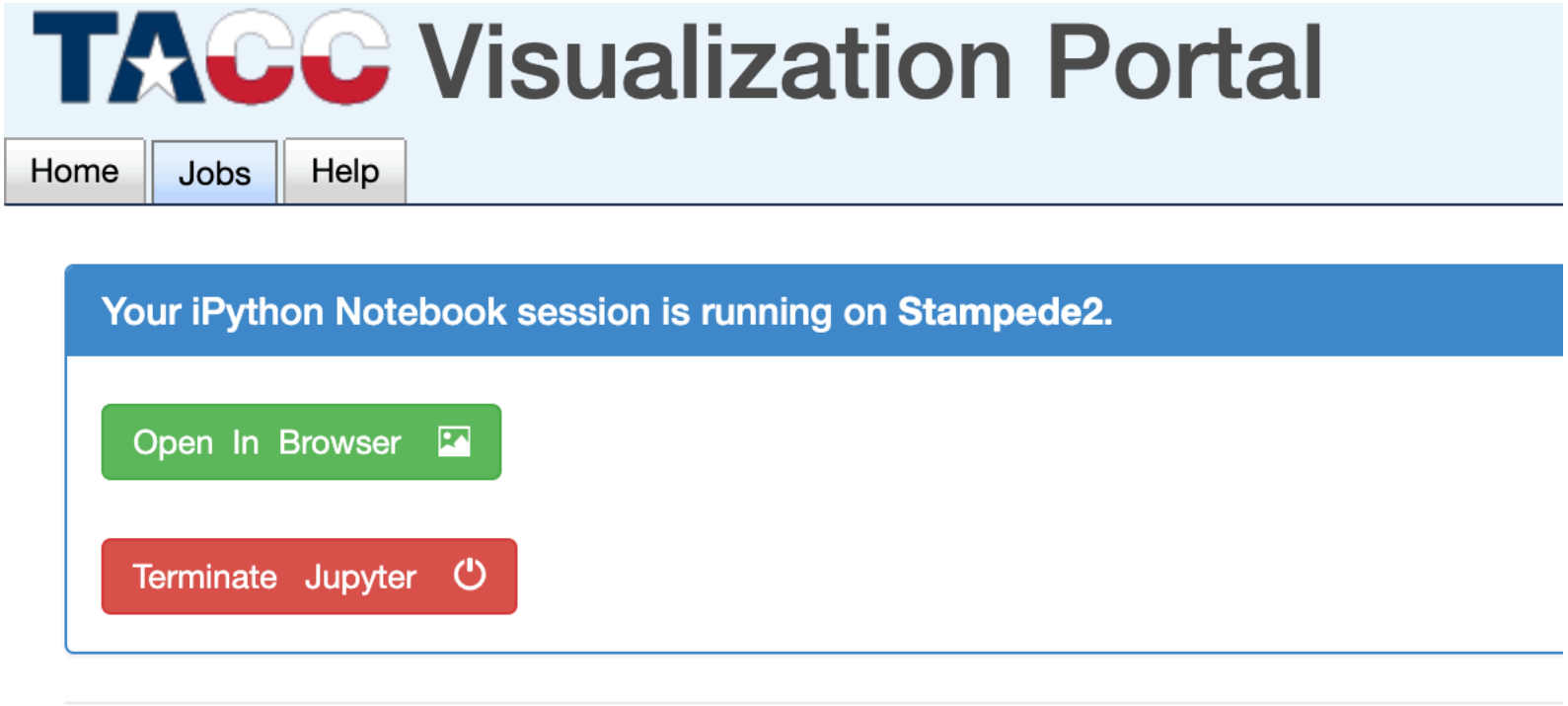
- Jupyter Notebook
  - http://jupyter.org/
  - Fork off the IPython
    - More fully featured Python shell
  - Requires a Python installation
  - Can be installed on your computer locally
    - *python -m pip install --upgrade pip*
    - *python -m pip install jupyter*
  - Provides an interactive interface in the browser
    - Can view output and plots inline
    - Makes Iterative/Interactive Development easy
- Available on TACC machines
  - Wrangler
  - Stampede2

Accessing Python (3)

- Jupyter Notebook via TACC VIS Portal
- Log in https://vis.tacc.utexas.edu/ with your TACC/XSEDE account

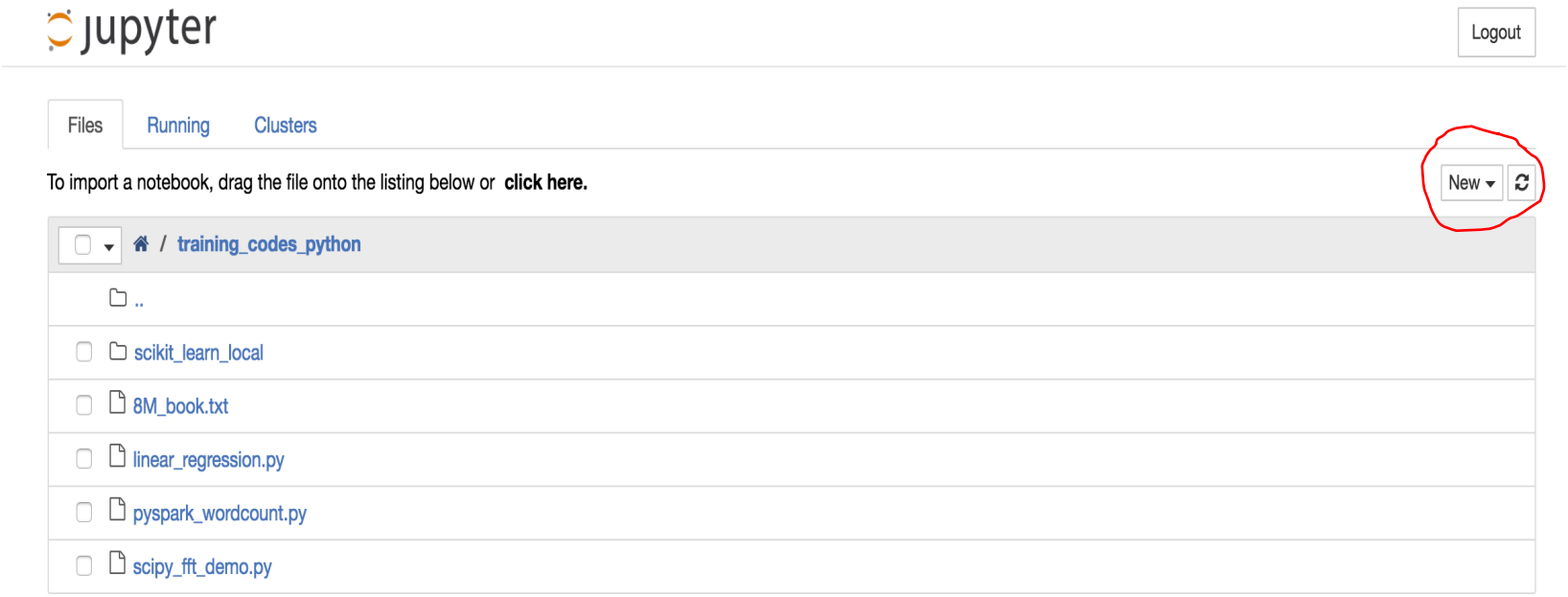


Accessing Python (4)

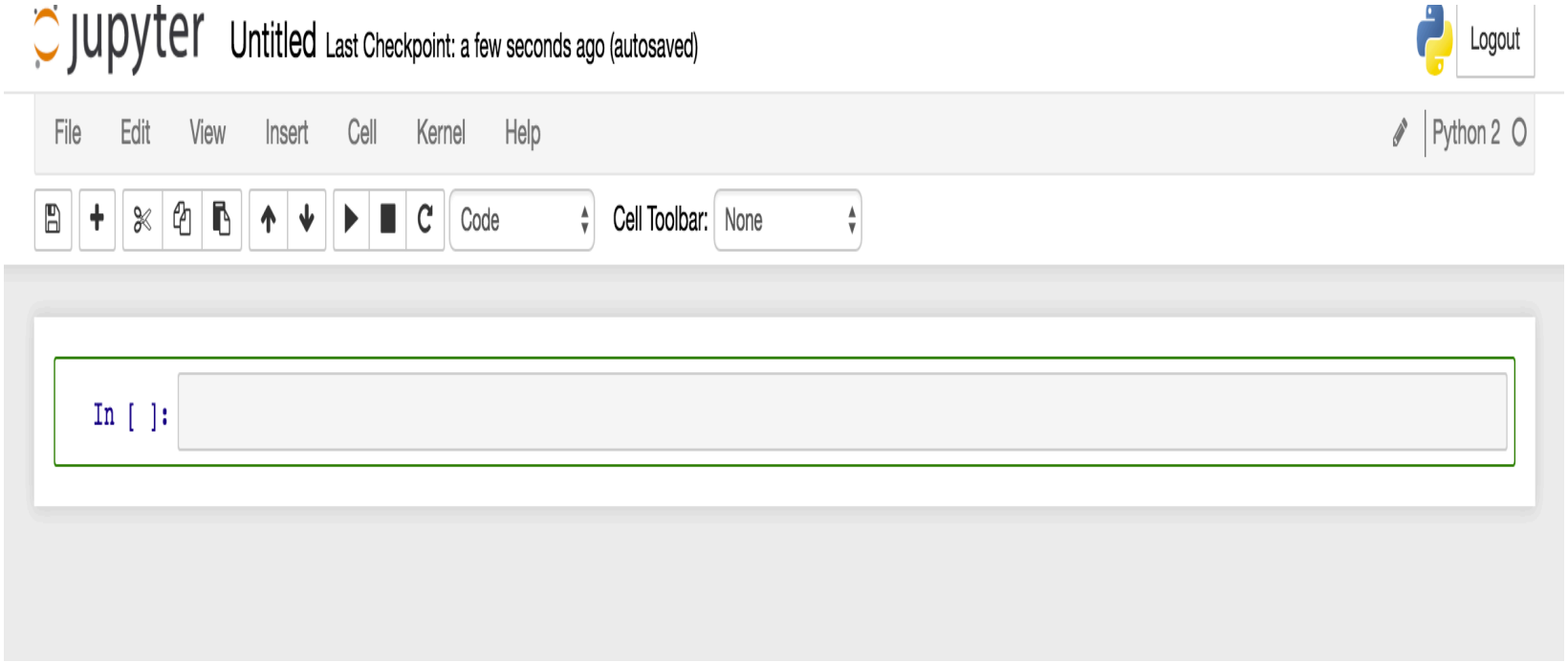


Accessing Python (5)

- ”New” -> Open IPython notebook



Ipython Notebook



Notebook Files

```
! cp -rf /work/02679/agupta/stampede2/python_for_ML_training $HOME/
```

- Or SSH into Stampede2 with your TACC credentials
- On command prompt, type:

  cp -rf /work/02679/agupta/stampede2/python_for_ML_training $HOME/

- Will create folder “python_for_ML_training” in your home folder

Python Program Organization

- Indentations matter
  - Code block boundaries (ex. if-else block, function block)
- Modularization
  - Enables code reuse, importing functionality from other code
  - Every file can be a module
    - Code in separate folder with empty file "_ _ init _ _.py"
  - Modules can be combined to make packages
- Packages => Module A => Module B => Module C
  - import Package.moduleA.moduleB.moduleC
  - "." dot operator

Python Basics – Built-in Datatypes

- Basic Types
  - int, float, long, complex, boolean
- Strings
- Collections
  - Dictionary
    - *Key:Value* pairs
  - List
    - Collection of objects (int, floats, any python object)
  - Tuple
    - Like lists, but unmutable
- Everything is an object
  - Will have its own methods to manipulate it

Python Basics – Example (1)

python_for_ML_training / introduction / introduction.ipynb

In [21]:
```
# Importing a module
import sys

# accessing a modules parameters
sys.path

# This is a single line comment
"""
This is a
multiline comment
"""
```

Out[21]:
```
'\nThis is a\nmultiline comment\n'
```

In [22]:
```
# Initialising a Variable
# Integer
x = 10
# String
y = " hello "
```

In [ ]:
```

```

Python Basics – Example (2)

introduction.ipynb

In [25]:
```
# Collections: List, Tuple, Dictionary
# List
example_list = [ "a", "b", "c" ]
# List : access element
example_list[2]
```

Out[25]:
```
'c'
```

In [26]:
```
# List : append element
example_list.append("d")
```

In [29]:
```
print example_list
```

```
['a', 'b', 'c', 'd']
```

In [30]:
```
print len(example_list)
```

```
4
```

In [31]:
```
# Initialize empty list
empty_list = list()
```

In [32]:
```
print empty_list
```

```
[]
```

Python Basics – Example (3)

introduction.ipynb

In [33]:
```
# Tuple
example_tuple = (1, 2, 3)
# Tuple : access element
example_tuple[1]
```

Out[33]: 2

In [34]:
```
# Tuple : no append method i.e immutable
# Get length of tuple
print len(example_tuple)
```

3

Python Basics – Control Flow

- **while:**

  while expression/boolean:

  statements ….

  ….

- **for:**

  for x in ‘some iterable collection’:

  statements …

- **break**
  - Terminates loop
- **continue**
  - Next iteration of loop
- **pass**

Python Basics – Control Flow (2)

if-else

```
If condition:
    statements….
else:
    statements….
```

functions

```
def function_name(arguments):
        statements …
```

try, except

```
try :
        statements….
except exception-name:
        statements….
```

Python Basics – Control Flow (1)

introduction.ipynb

In [38]:
```
# Example control flow
# if-elif-else : Number of elements even or odd or empty list
if len(example_list)==0:
    print "empty list"
elif len(example_list)%2==0:
    print "Even number of elements"
else:
    print "Odd number of elements"
```

```
Even number of elements
```

Python Basics – Control Flow (2)

introduction.ipynb

In [39]:
```
# For loop : Print even index elements of the list
for element in example_list:
    if example_list.index(element)%2==0:
        print element
    else:
        # pass does nothing
        pass
```

```
a
c
```

In [40]:
```
print example_list
```

```
['a', 'b', 'c', 'd']
```

Python Basics – Control Flow (3)

introduction.ipynb

```
In [44]: example_list.append("x")

In [45]: example_list
Out[45]: ['a', 'b', 'c', 'd', 'x']

In [46]: # While loop : Search for "x" in list
         index=0
         while index<len(example_list):
             if example_list[index] == "x":
                 print "Found element x"
                 break
             else:
                 index+=1
                 continue

         Found element x
```

Python Basics – Control Flow (4)

introduction.ipynb

In [47]:
```
# define function to search a list for an element
# and return its index if found and -1 if not found
def search_list_for_element(element, search_list):
    index=0
    while index < len(search_list):
        if search_list[index] == element:
            break
        else:
            index+=1
            continue
    #If index is less than length, element is found
    if index < len(search_list):
        return index
    else:
        return -1
```

Python Basics – Control Flow (5)

introduction.ipynb

```
In [48]: search_list_for_element("x", example_list)
Out[48]: 4

In [49]: example_list
Out[49]: ['a', 'b', 'c', 'd', 'x']
```

Lambda Functions

- Functions on the fly
  - Small one-liner functions
  - Can be assigned to a variable
  - Or used directly

```
In [50]: #Lambdas
         y = lambda x:x**3
         print y(3)

27

In [51]: z = (lambda x,y: x**y)
         print z(3,3)

27

In [52]: #anonymous
         print (lambda x,y: x**y)(3,3)

27
```

Exercise

- Write a function to compute a vector dot product of 2 lists
  - Define a function dot(x,y) that accepts 2 lists as arguments
    - Check if length of both lists are equal using len() function
    - If they are not equal print a message and return
    - Generate the range of indices to iterate over
    - Return the value of the dot product
  - Test it
    - Define 2 lists a=[1,2,3,4,5] and b=[6,7,8,9,10]
    - Compute Dot Product with your function
    - Print result

Exercise

introduction.ipynb

In [77]:
```
#define function to multiply (vector dot product) of 2 lists of integers
#input: list x and list y
#output: list result = x.y or -1 for error
def dot(x,y):
    #check if both lists are equal length
    if len(x) == len(y):
        # empty list
        result_list = list()
        #index
        index = 0
        #iterate through 2 lists
        while index < len(x):
            result_list.append(x[index]*y[index])
            index+=1
        return result_list
    else:
        print "Error: Lists of unqual length given"
        return -1
```

Python Basic File I/O

- Open a file
  - file_object = open(file_name [, access_mode][, buffering])
  - Access Mode : Read (r), Write (w), Append (a)
  - Buffering (Size of file buffer)
    - 0==Unbuffered, 1==Line Buffered, >1==Size of buffer
  - Returns a file object
    - Use its methods to manipulate file

Python Basic File I/O

- Read from a file
  - data_string = file_object.read(number_of_bytes_to_read)
  - Returns a string (even if file is binary)
  - Current position in the file
    - file_object.tell()
- Read a line from file
  - line_string = file.object.readline()
  - Reads till trailing newline (“\n”) character.
  - Retains it in the string
  - Returns blank string for EOF

Python Basic File I/O

- Write to a file
  - data_string = file_object.write(string_to_write)
  - Returns a string (even if file is binary)
  - Current position in the file
    - file_object.tell()
- Close a file
  - file_object.close()

Exercise

- Write a program to count the number of lines in a book (txt file provided: $HOME/python_for_ML_training/data/8M_book.txt).
  - Give a full path of the file to open()

Exercise

introduction.ipynb

Line Count

```
|: import os
home_path =  os.path.expanduser("~")
file_path = os.path.join(home_path, "python_for_ML_training/data/8M_book.txt")
```

```
|:
# define a function to count number of lines in a file
#input: open file object, with seek position 0
#output: number of lines in the file
def count_number_of_lines(f):
    #read a line
    line = f.readline()
    #initiate line count
    if line:
        line_count=1
    else:
        line_count=0
    #iterate through each line of file
    while line:
        line_count+=1
        line = f.readline()
    return line_count

# Open a file
file_object = open(file_path, "r")

print count_number_of_lines(file_object)

#file close
file_object.close()
```

```
146933
```

TACC

Installing python modules

- First on command line run
  - “module load python/2.7.13”
- Python Package Index (pip)
- Packages/Modules can be installed locally
  - No Root priviliges required
- When searching for an available package
  - pip search package_name

Installing python modules

- pip install --user package_name
  - Installs in the USER_BASE area
  - ~/.local/ folder
- To install in a folder you create

  pip install <package_name> --install-option="--prefix=<local install path>"
- For modules from third party distributors
  - python setup.py install --user
  - python setup.py install --home=<dir>

Installing python modules

- Install SciKit Learn (sklearn)
  - Machine Learning for Python
- Run PIP to install scikit-learn

pip install --user sklearn

```
: ! pip install --user sklearn

Requirement already satisfied: sklearn in /home1/02679/agupta/.local/lib/pyth
Requirement already satisfied: scikit-learn in /opt/apps/gcc7_1/python/2.7.13
rn)
You are using pip version 9.0.1, however version 19.0.3 is available.
You should consider upgrading via the 'pip install --upgrade pip' command.
```

Break

Resume at 2PM

Useful Packages for ML

- NumPy
- SciPy
- Pandas
- MatplotLib
- Scikit-Learn

Numpy

- Important package for Scientific Computing
- N-dimensional array object
- More efficient than python objects
- Tools for integrating low level code (C/C++/Fortran)
- Also has Mathematical Functions
- Documentation
  - https://docs.scipy.org/doc/numpy/

Numpy

- N-dimensional array
  - Fundamental part of Numpy Package
- Container for Homogenous data
- Contiguous in memory
  - Leads to faster operations
- Collection manipulated like scalars

Numpy

- Each Dimension is called an Axis
- Shape
  - Size of array along each dimension
  - Tuple of axis sizes
- Rank
  - Number of dimensions in array
  - Length of the Shape Tuple

1	16	35
10	45	33
9	8	23
64	90	87

- Shape = (4,3)
- Rank = 2

Numpy Array

- Data Types available
  - int8, uint8
  - int16, uint16
  - int32, uint32
  - int64, uint64
  - float16
  - float32
  - float64
  - float128
  - complex
  - boolean
  - ….

Numpy Array

python_for_ML_training / unsupervised / numpy_examples.ipynb

In [171]:
```
# np.arange(start, stop, step_size)
# Like range() for python lists/arrays
test_array = np.arange(1.0, 105.82, 2.3184)
print test_array
```

```
[   1.       3.3184    5.6368    7.9552   10.2736   12.592    14.9104
   17.2288   19.5472   21.8656   24.184    26.5024   28.8208   31.1392
   33.4576   35.776    38.0944   40.4128   42.7312   45.0496   47.368
   49.6864   52.0048   54.3232   56.6416   58.96     61.2784   63.5968
   65.9152   68.2336   70.552    72.8704   75.1888   77.5072   79.8256
   82.144    84.4624   86.7808   89.0992   91.4176   93.736    96.0544
   98.3728  100.6912  103.0096  105.328 ]
```

In [172]:
```
# np arrays can be reshaped using same number of elements
# Shape = (46,), Rank = 1
test_array.shape
```

Out[172]:
```
(46,)
```

In [173]:
```
print len(test_array.shape)
```

```
1
```

Numpy Array

numpy_examples.ipynb

```
In [174]: # np.array.reshape (23,2)  Returns a reshaped array
          print test_array.reshape(23,2)
```

```
[[   1.        3.3184]
 [   5.6368    7.9552]
 [  10.2736   12.592 ]
 [  14.9104   17.2288]
 [  19.5472   21.8656]
 [  24.184    26.5024]
 [  28.8208   31.1392]
 [  33.4576   35.776 ]
 [  38.0944   40.4128]
 [  42.7312   45.0496]
 [  47.368    49.6864]
 [  52.0048   54.3232]
 [  56.6416   58.96  ]
 [  61.2784   63.5968]
 [  65.9152   68.2336]
 [  70.552    72.8704]
 [  75.1888   77.5072]
 [  79.8256   82.144 ]
 [  84.4624   86.7808]
 [  89.0992   91.4176]
 [  93.736    96.0544]
 [  98.3728  100.6912]
 [ 103.0096  105.328 ]]
```

```
In [175]: # Shape = (46,), Rank = 1
          test_array.shape
```

```
Out[175]: (46,)
```

```
In [176]: print len(test_array.shape)
```

```
1
```

Numpy Array Slicing

numpy_examples.ipynb

test_array

1	2	3	4	5	6	7
3	6	9	12	15	18	21
5	10	15	20	25	30	35
7	14	21	28	35	42	49

Numpy Array Slicing

numpy_examples.ipynb

test_array[2]

	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	3	6	9	12	15	18	21
2	5	10	15	20	25	30	35
3	7	14	21	28	35	42	49

Numpy Array Slicing

numpy_examples.ipynb

test_array[1:3]

	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	3	6	9	12	15	18	21
2	5	10	15	20	25	30	35
3	7	14	21	28	35	42	49

Numpy Array Slicing

numpy_examples.ipynb

test_array[2,5]

	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	3	6	9	12	15	18	21
2	5	10	15	20	25	30	35
3	7	14	21	28	35	42	49

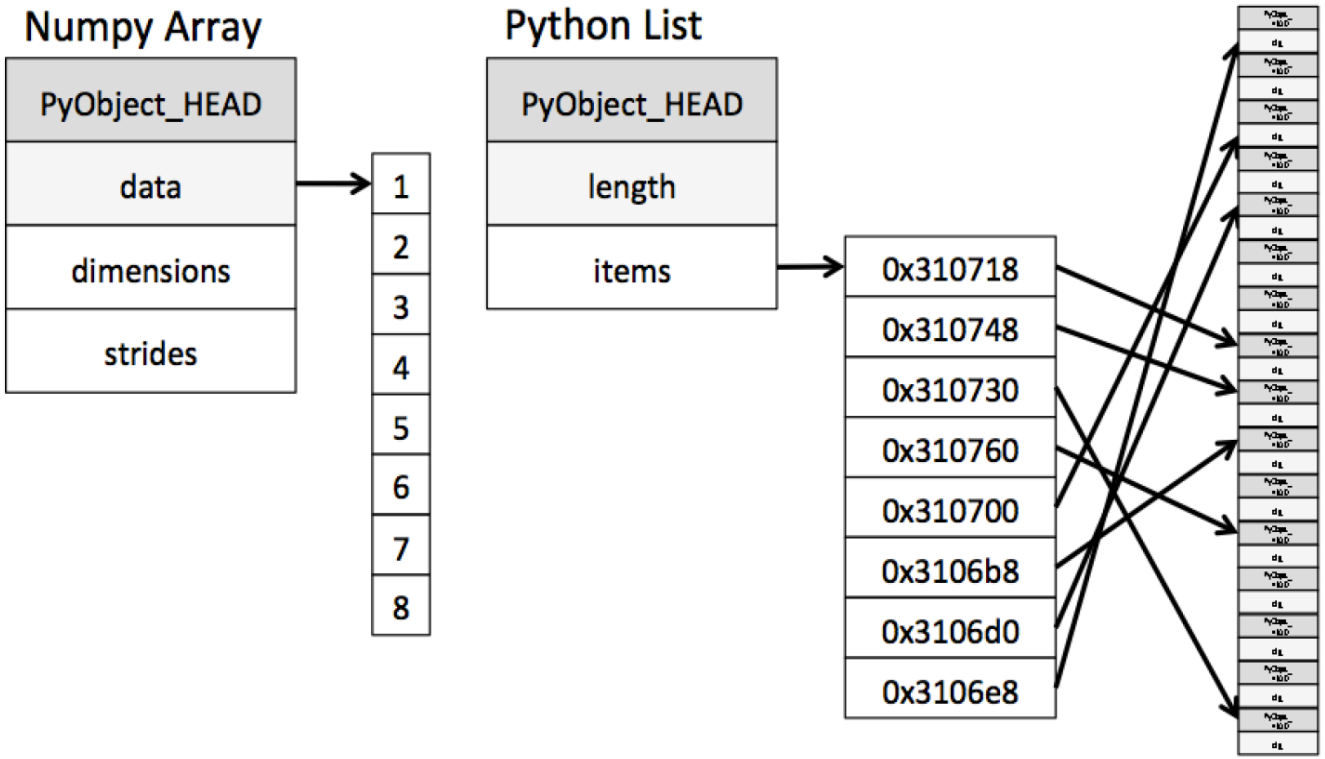
Numpy Array Slicing

numpy_examples.ipynb

test_array[1:3, 2:5]

	0	1	2	3	4	5	6
0	1	2	3	4	5	6	7
1	3	6	9	12	15	18	21
2	5	10	15	20	25	30	35
3	7	14	21	28	35	42	49

Numpy


Numpy Array
PyObject_HEAD
data
dimensions
strides
1
2
3
4
5
6
7
8
Python List
PyObject_HEAD
length
items
0x310718
0x310748
0x310730
0x310760
0x310700
0x3106b8
0x3106d0
0x3106e8

Numpy - ufunc

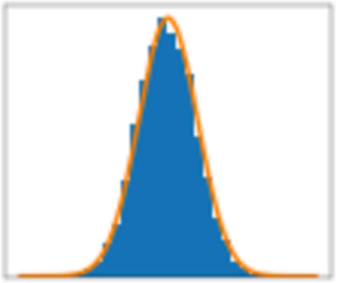
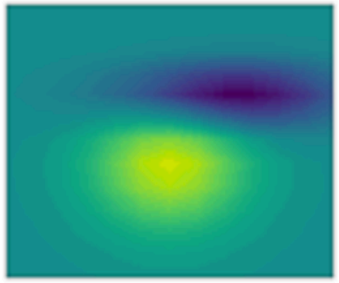
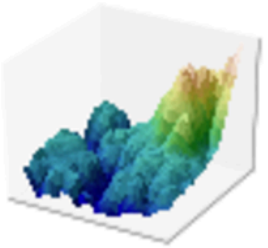
- Universal functions
  - Part of the np.”namespace”
  - np.sqrt(array)
- Operate on ndarrays element-by-element
- Builtin functions, implemented in C
  - Low level close to hardware
  - Efficient
  - Vectorized versions of their python counterparts

Numpy - ufunc

- np.arange(number)
- np.dot(array, array)
- np.sqrt(array)
- np.sort(array)
- np.mean(array)
- np.random.randn(x, y, z…)
- …

Matplotlib

- Mature plotting library for Python
- https://matplotlib.org/
- Many options plots, coloring and visualizations
- Demo codes

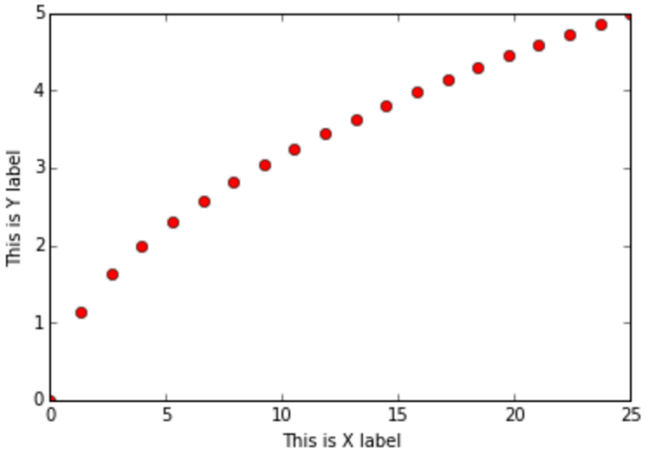





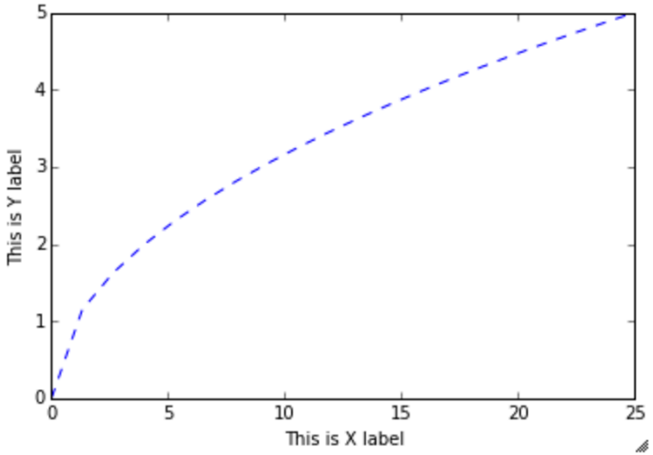
Matplotlib

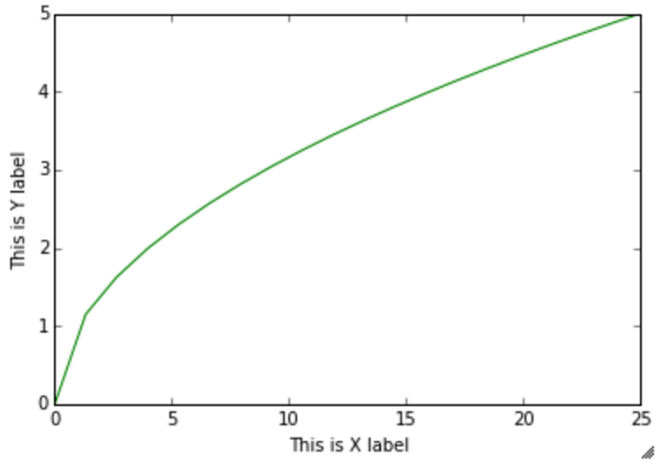
- In notebook
  - %matplotlib inline
- Scatter plot
- Line plot
- Histogram/Bar Plot
- 3D Plot

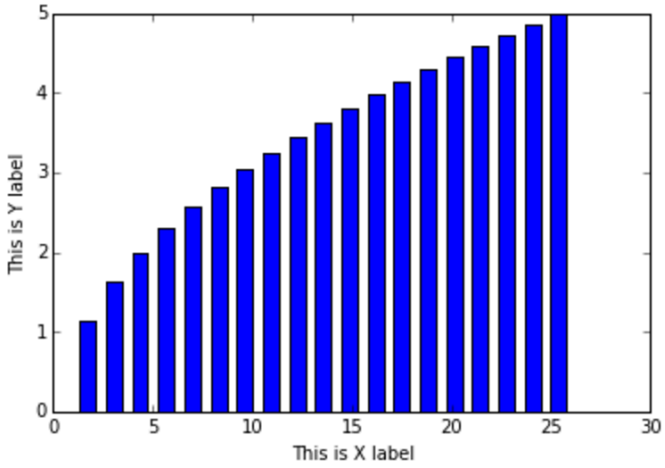
Matplotlib

matplotlib_examples.ipynb



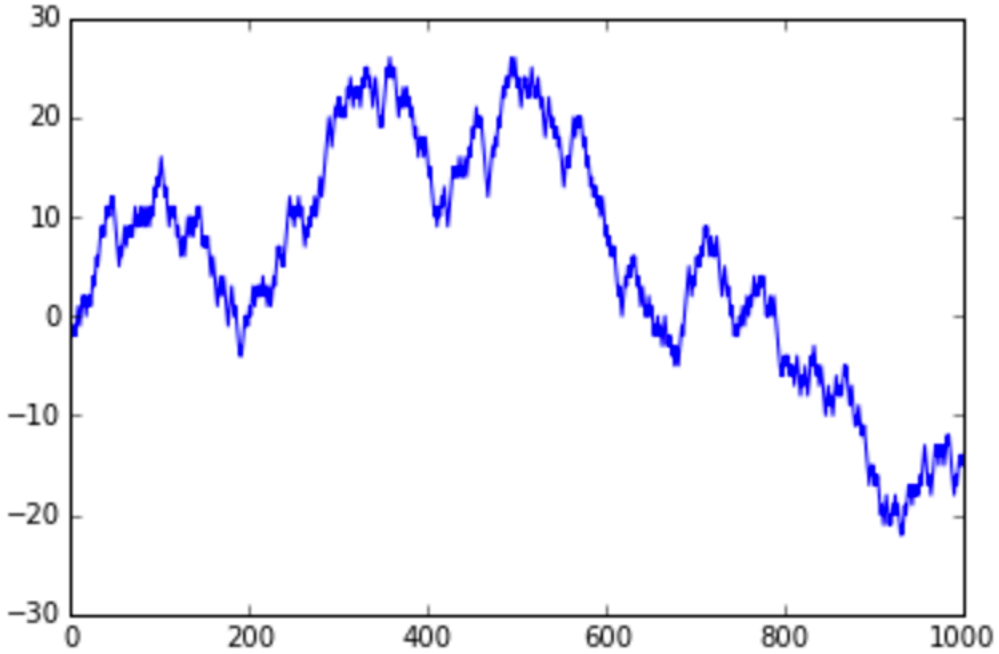






Example – Random Walking

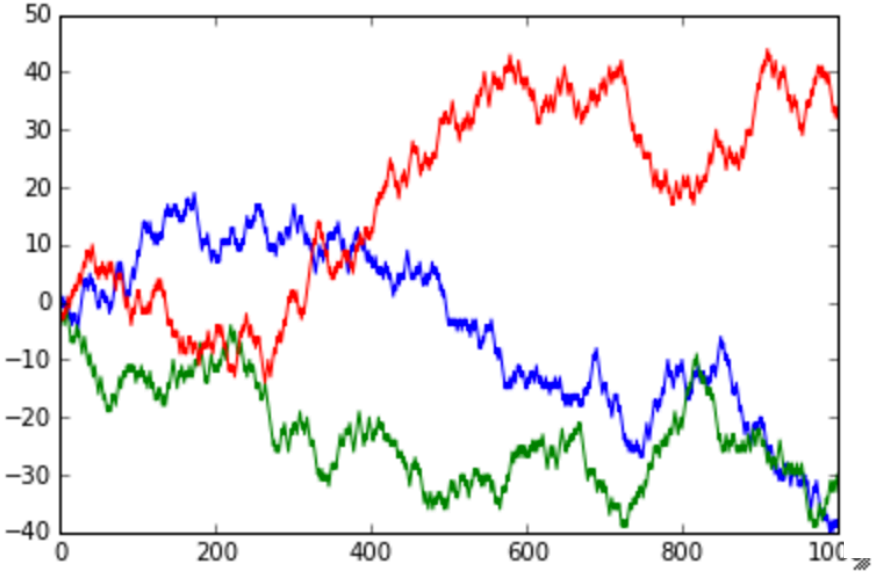
matplotlib_examples.ipynb

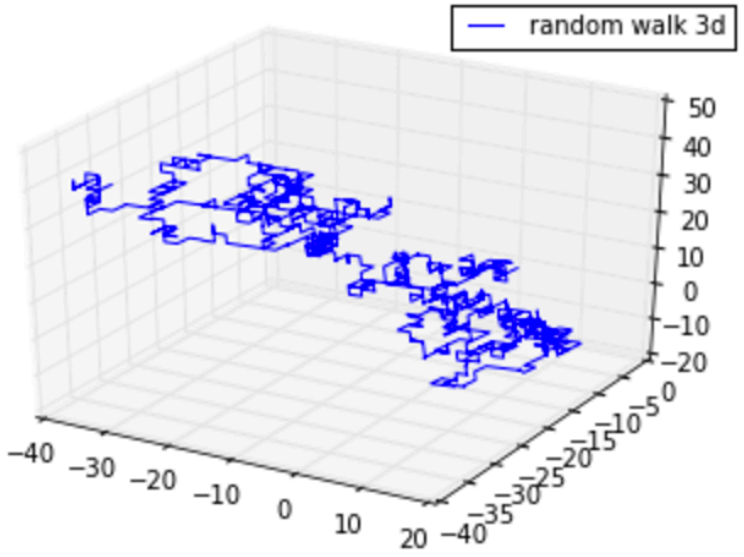



```
# Setup X and Y values
x = np.arange(steps)
y = np.array(walk)

# Plot the random walk
plt.plot(x,y)
```

Example – 3D Random Walking

matplotlib_examples.ipynb





SciPy

- Python package dedicated to scientific computing
- Module each for specialized areas
  - special functions, statistics, optimization etc.
- Builds upon NumPy
- To import from package
  - from scipy import sub-module-name

SciPy

- Special Functions
- Integration
- Optimization
- Interpolation
- Fourier Transforms
- Signal Processing
- Linear Algebra
- Statistics

SciPy

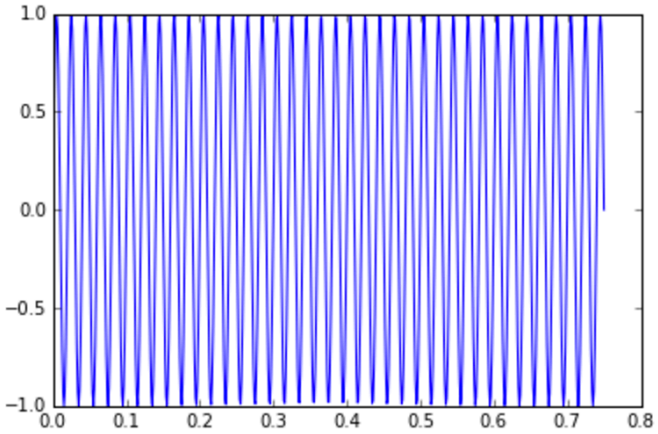
scipy_fft_demo.ipynb

In [69]:

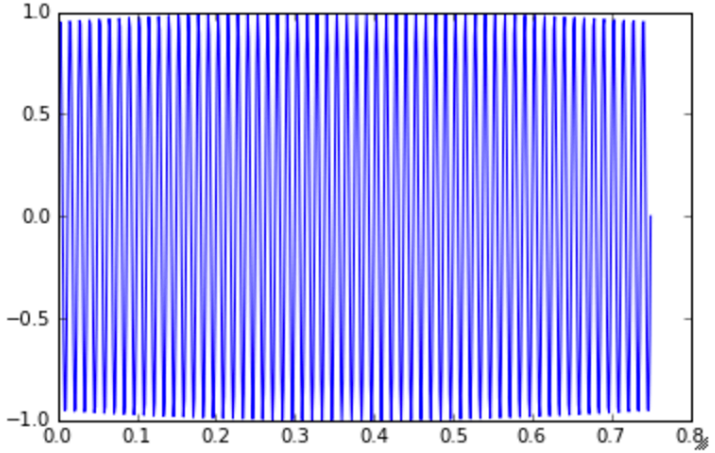
```
#for notebook inline plots
%matplotlib inline
#Import Numpy
import numpy as np
#Import SciPy FFT
from scipy.fftpack import fft
#Import Matplotlib
import matplotlib.pyplot as plt



# Number of sample points
N = 600

#sampling interval
T = 1.0 / 800.0

# Xaxis points
x = np.linspace(0.0, N*T, N)

#Yaxis points (Adding 2 sinusoidal signals)
y1= np.sin(50.0 * 2.0 * np.pi * x)
y2= np.sin(80.0*2.0*np.pi*x)
y =  y1 + 0.536 * y2
```

SciPy

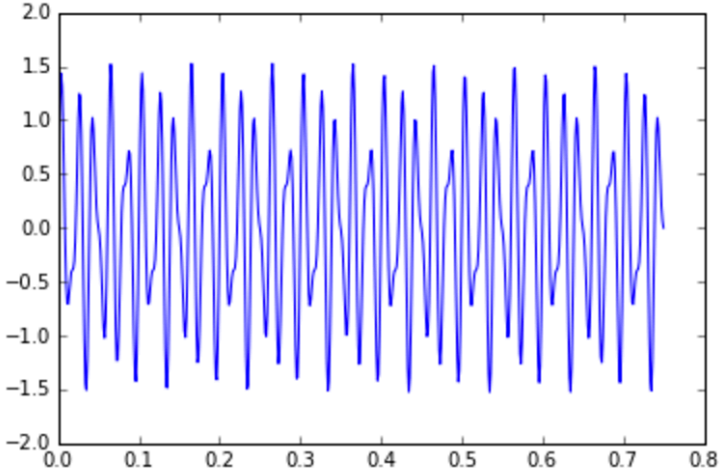
scipy_fft_demo.ipynb



+



=

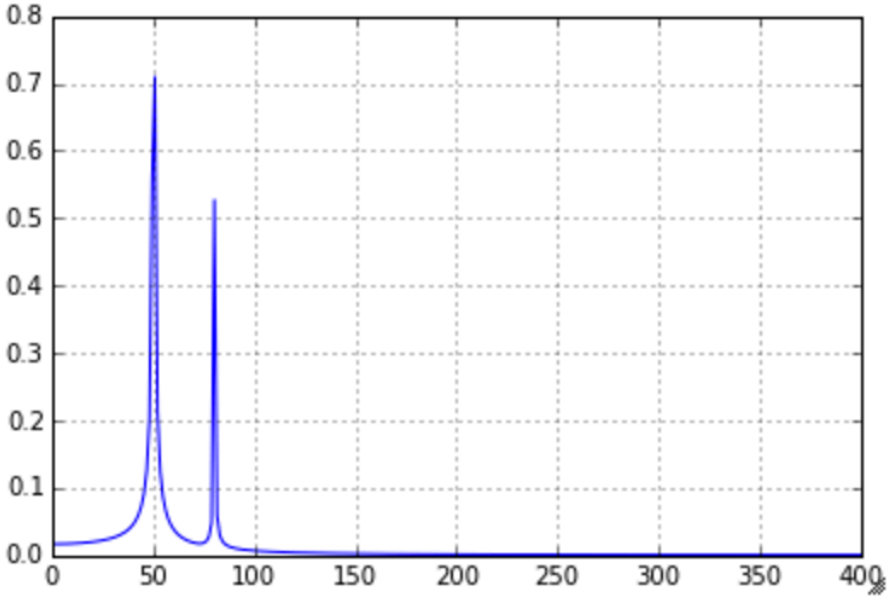


SciPy

scipy_fft_demo.ipynb

```
In [73]: # Y and X frequency domain
         yf = fft(y)
         xf = np.linspace(0.0, 1.0/(2.0*T), N/2)
```

```
In [74]: # plot frequecies
         plt.plot(xf, 2.0/N * np.abs(yf[0:N/2]))
         plt.grid()
         plt.show()
```



Pandas

- Python package that builds upon numpy arrays
- Useful in data analysis
  - Common manipulations of arrays
- Series
  - Useful for sequence data like time-series
- DataFrame
  - Useful for spreadsheet like data
  - Analogous to R Data Frames
- Documentation
  - http://pandas.pydata.org/

Pandas(2)

- Supports various file formats
  - CSV
  - HDF5
  - Excel Files
  - JSON
  - XML
  - PICKLE
- Graceful handling of missing values
- Has internal plotting functions
- Has many powerful functions
  - Database style joins

Series

pandas_examples.ipynb

In [63]:
```
# Using a dictionary to create a series with indexes from keys:values
test_dictionary = dict({"one":1, "two":2, "three":3, "four":4})
test_series = Series(test_dictionary)
#Keys in sorted order
print test_series
```

```
four      4
one       1
three     3
two       2
dtype: int64
```

Series

pandas_examples.ipynb

In [64]:
```
#data with null/NA values
test_index = ["one", "two", "three", "four", "five"]
test_series = Series(test_dictionary, index=test_index)
print test_series
```

```
one        1
two        2
three      3
four       4
five     NaN
dtype: float64
```

Series

pandas_examples.ipynb

```
In [66]: #detect NaN with isnull(), notnull() pandas methods
         pd.isnull(test_series)

Out[66]: one      False
         two      False
         three    False
         four     False
         five      True
         dtype: bool

In [67]: pd.notnull(test_series)

Out[67]: one       True
         two       True
         three     True
         four      True
         five     False
         dtype: bool
```

Series

pandas_examples.ipynb

```
In [68]: #print test_series for reference
         print test_series
```

```
one        1
two        2
three      3
four       4
five     NaN
dtype: float64
```

```
In [70]: #print test_series_2 for reference
         print test_series_2
```

```
five     5
one      1
three    3
two      2
dtype: int64
```

```
In [71]: #data alignment
         test_series + test_series_2
```

```
Out[71]: five     NaN
         four     NaN
         one        2
         three      6
         two        4
         dtype: float64
```

Data Frame

pandas_examples.ipynb

In [74]:
```
import numpy as np
import pandas as pd
from pandas import Series, DataFrame


# create dataframe data
data = {'state': ['Ohio', 'Ohio', 'Ohio', 'Nevada', 'Nevada'],
        'year': [2000, 2001, 2002, 2001, 2002],
        'pop': [1.5, 1.7, 3.6, 2.4, 2.9]}
```

Data Frame

pandas_examples.ipynb

```
In [81]: # Choosing multiple columns
         # Jupyters table display
         test_data_frame[['year', 'state']]
```

Out[81]:

	year	state
row one	2000	Ohio
row two	2001	Ohio
row three	2002	Ohio
row four	2001	Nevada
row five	2002	Nevada

Data Frame

pandas_examples.ipynb

In [101]:
```
# Assigning a column a scalar value
test_data_frame['debt'] = 25
test_data_frame
```

Out[101]:

	year	state	pop	debt
row one	2000	Ohio	1.5	25
row two	2001	Ohio	1.7	25
row three	2002	Ohio	3.6	25
row four	2001	Nevada	2.4	25
row five	2002	Nevada	2.9	25

Data Frame

pandas_examples.ipynb

In [102]:
```
# Assigning a column a vector value
test_data_frame['debt'] = np.arange(1,6,1)
test_data_frame
```

Out[102]:

	year	state	pop	debt
row one	2000	Ohio	1.5	1
row two	2001	Ohio	1.7	2
row three	2002	Ohio	3.6	3
row four	2001	Nevada	2.4	4
row five	2002	Nevada	2.9	5

Data Frame

pandas_examples.ipynb

```
In [103]: # Filling in holes in data
          test_data_frame['debt'] = np.NaN
          test_data_frame
```

Out[103]:

	year	state	pop	debt
row one	2000	Ohio	1.5	NaN
row two	2001	Ohio	1.7	NaN
row three	2002	Ohio	3.6	NaN
row four	2001	Nevada	2.4	NaN
row five	2002	Nevada	2.9	NaN

Data Frame

pandas_examples.ipynb

In [104]:
```
new_data = Series([1.2, 2.4, 5.7],index=['row two', 'row three', 'row five'])
test_data_frame['debt'] = new_data
test_data_frame
```

Out[104]:

	year	state	pop	debt
row one	2000	Ohio	1.5	NaN
row two	2001	Ohio	1.7	1.2
row three	2002	Ohio	3.6	2.4
row four	2001	Nevada	2.4	NaN
row five	2002	Nevada	2.9	5.7

Data Frame

pandas_examples.ipynb

In [107]:
```
# Transpose
test_data_frame.T
```

Out[107]:

	row one	row two	row three	row four	row five
year	2000	2001	2002	2001	2002
state	Ohio	Ohio	Ohio	Nevada	Nevada
pop	1.5	1.7	3.6	2.4	2.9
debt	NaN	1.2	2.4	NaN	5.7

Data Frame

pandas_examples.ipynb

```
In [108]: # Row shuffle
          test_data_frame.reindex(['row five', 'row four', 'row three', 'row two', 'row one'])
```

	year	state	pop	debt
row one	2000	Ohio	1.5	NaN
row two	2001	Ohio	1.7	1.2
row three	2002	Ohio	3.6	2.4
row four	2001	Nevada	2.4	NaN
row five	2002	Nevada	2.9	5.7

	year	state	pop	debt
row five	2002	Nevada	2.9	5.7
row four	2001	Nevada	2.4	NaN
row three	2002	Ohio	3.6	2.4
row two	2001	Ohio	1.7	1.2
row one	2000	Ohio	1.5	NaN

Data Frame

pandas_examples.ipynb

```
In [114]: # Column shuffle
          test_data_frame.reindex(columns=['pop', 'year', 'debt', 'state', 'new column'])
```

	year	state	pop	debt
row one	2000	Ohio	1.5	NaN
row two	2001	Ohio	1.7	1.2
row three	2002	Ohio	3.6	2.4
row four	2001	Nevada	2.4	NaN
row five	2002	Nevada	2.9	5.7

	pop	year	debt	state	new column
row one	1.5	2000	NaN	Ohio	NaN
row two	1.7	2001	1.2	Ohio	NaN
row three	3.6	2002	2.4	Ohio	NaN
row four	2.4	2001	NaN	Nevada	NaN
row five	2.9	2002	5.7	Nevada	NaN

Data Frame

pandas_examples.ipynb

	year	state	pop	debt
row one	2000	Ohio	1.5	NaN
row two	2001	Ohio	1.7	1.2
row three	2002	Ohio	3.6	2.4
row four	2001	Nevada	2.4	NaN
row five	2002	Nevada	2.9	5.7

In [115]:
```
# Summarization methods (excluse invalid N/A data)
# more robust than numpy methods
# sum of each column
test_data_frame.sum()
```

Out[115]:
```
year                          10006
state     OhioOhioOhioNevadaNevada
pop                            12.1
debt                            9.3
dtype: object
```

Data Frame Join

```
pd.merge(data_frame_1, data_frame_2,left_on='lkey', right_on='rkey')
```

	data1	lkey
0	0	b
1	1	b
2	2	a
3	3	c
4	4	a
5	5	a
6	6	b

⋈

	data2	rkey
0	0	a
1	1	b
2	2	d

	data1	lkey	data2	rkey
0	0	b	1	b
1	1	b	1	b
2	6	b	1	b
3	2	a	0	a
4	4	a	0	a
5	5	a	0	a

Scikit-Learn

- Machine Learning package
  - Good Documentation
  - Example Codes
- Common Machine Learning Methods
  - Regression
    - Predicting Value
  - Classification
    - Assigning predefined Class/Groups
  - Clustering
    - Learning Groups
  - Dimensionality Reduction
    - Simplifying (by reducing features)

Scikit-Learn

- Machine Learning library for Python community
- Built upon NumPy, SciPy and Matplotlib
- Comes with some standard datasets
- Supervised and Unsupervised Learning
- Documentation with numerous examples
  - http://scikit-learn.org

Dataset

- Boston Housing Dataset
  - Included in Sklearn
    - sklearn.datasets.load_boston()
  - 506 samples (rows)
  - 13 features
    - CRIME RATE
    - Pupil-Teacher Ratio
    - TAX
    - PRICE
    - AGE
    - …

Dataset

- CRIM - per capita crime rate by town
- ZN - proportion of residential land zoned for lots over 25,000 sq.ft.
- INDUS - proportion of non-retail business acres per town
- CHAS - Charles River dummy variable (= 1 if tract bounds river; 0 otherwise)
- NOX - nitric oxides concentration (parts per 10 million)
- RM - average number of rooms per dwelling
- AGE - proportion of owner-occupied units built prior to 1940
- DIS - weighted distances to five Boston employment centres
- RAD - index of accessibility to radial highways
- TAX - full-value property-tax rate per $10,000
- PTRATIO pupil-teacher ratio by town
- B - 1000(Bk - 0.63)^2 where Bk is the proportion of blacks by town
- LSTAT - % lower status of the population
- MEDV - Median value of owner-occupied homes in $1000's
- PRICE – Property Price

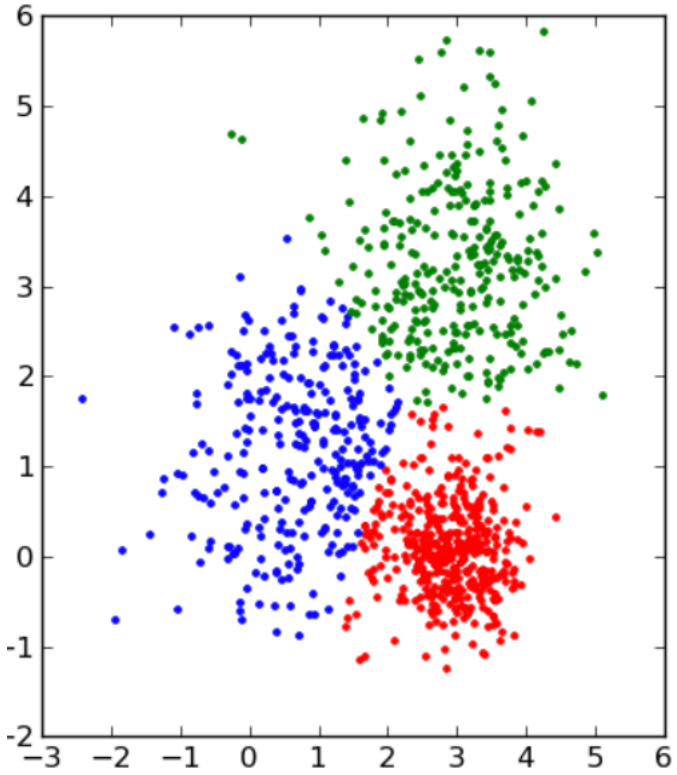
Dataset

```
bos.head()
```

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

Kmeans Clustering

- Unsupervised Learning Algorithm
- No Training/Testing data split



Kmeans Clustering

- Start:
  - Number of clusters to find
  - starting points as “center” for each cluser
- Calculate mean distance for each “center” point
- Divides into clusters based on nearest mean
- Centroid of each cluster becomes the new “center” point
- Repeat till convergence

Kmeans Clustering

```
from sklearn import cluster

kmeans = cluster.KMeans(n_clusters=3)
```

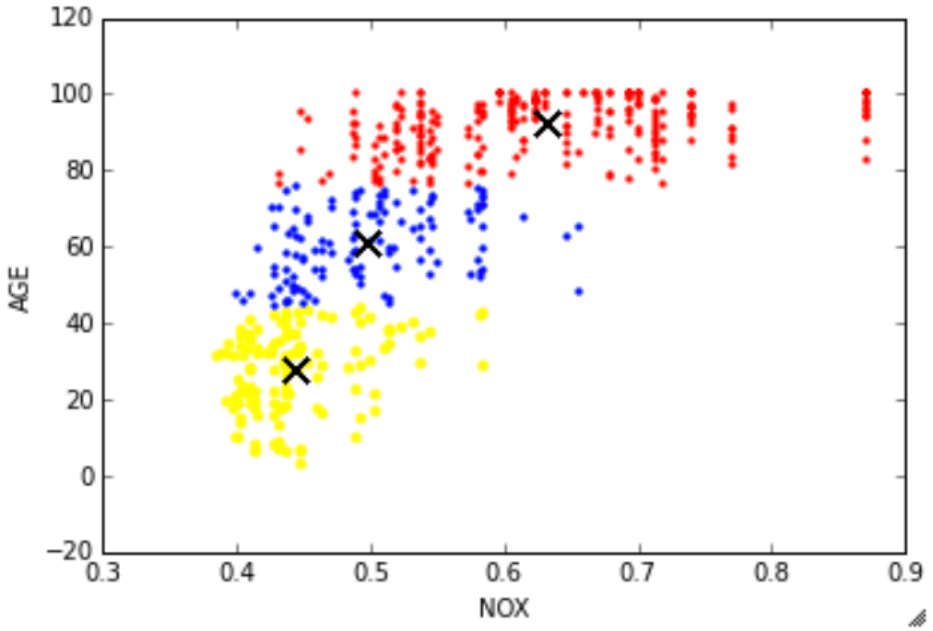
cluster.kmeans() expects arguments in dimension :
argument[number of samples*number of features]

```
#prepare the data vectors
data = np.concatenate((X.reshape(-1,1),Y.reshape(-1,1)), axis=1)
```

```
#run clustering
kmeans.fit(data)
```

```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
    n_clusters=3, n_init=10, n_jobs=1, precompute_distances='auto',
    random_state=None, tol=0.0001, verbose=0)
```

Kmeans Clustering



Kmeans Clustering

- Unlabeled data
- Some intuition of number of groups
  - Domain knowledge of the dataset
- No training needed
- Doesn’t work under certain conditions
  - Sparse clusters/ Overlapping clusters
  - Clusters of different sizes
  - Clusters in feature space (Dependent features)
  - Can get stuck in local minima
    - Kmeans++

Break

- Resume at 3PM

- Exercise:

Run kmeans on the IRIS Flower dataset:

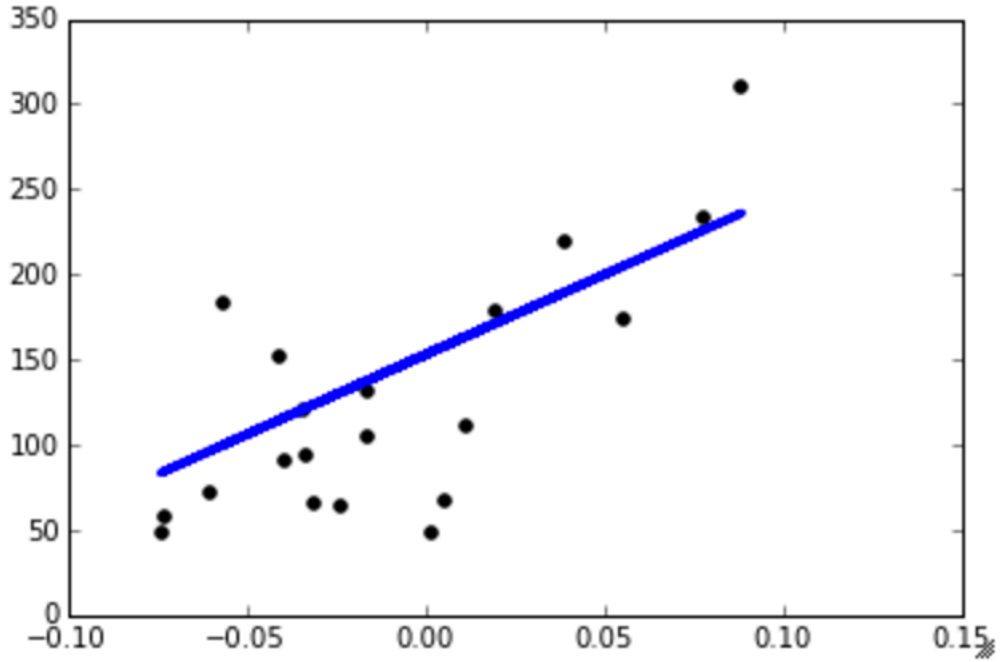
Code at:

https://scikit-learn.org/stable/auto_examples/cluster/plot_cluster_iris.html

Supervised Learning

- Linear Regression
- Support Vector Machines (SVM)
- Naïve Bayes Classifier

Linear Regression



Linear models: $y = X\beta + \epsilon$

- X: data
- y: target variable
- β: Coefficients
- ϵ: Observation noise

Linear Regression

python_for_ML_training / supervised / supervised.ipynb

```
# Independent Variables: All 13 features
X = bos.drop('PRICE', axis = 1)
# Target: House Price
Y = bos['PRICE']
```

```
#split data into train and test set
X_train, X_test, Y_train, Y_test = sklearn.model_selection.train_test_split(X, Y, test_size = 0.33, random_state = 5)

print(X_train.shape)
print(X_test.shape)
print(Y_train.shape)
print(Y_test.shape)
```

```
(339, 13)
(167, 13)
(339,)
(167,)
```

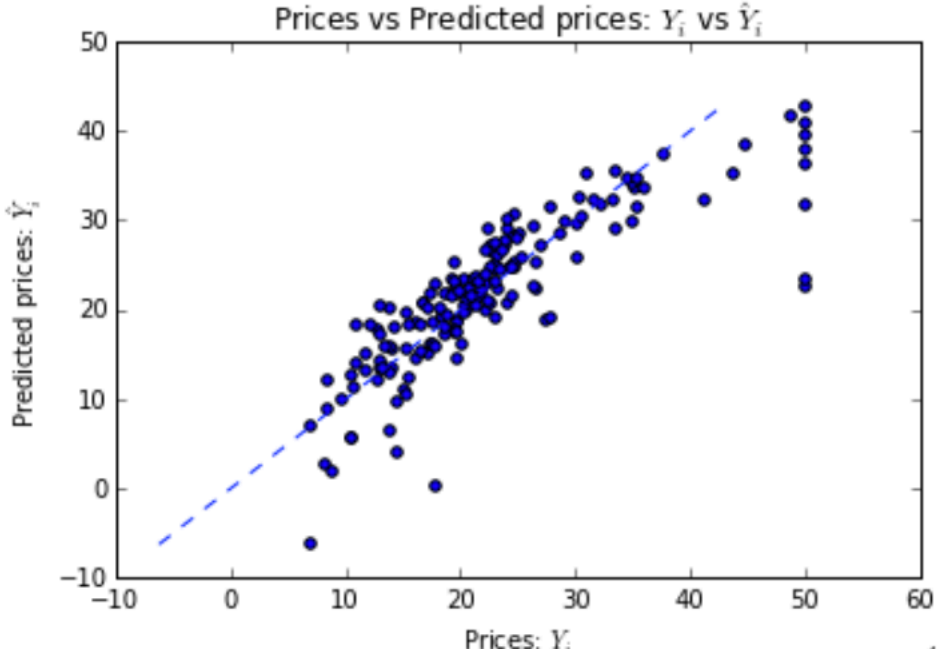
Linear Regression

supervised.ipynb

```
from sklearn.linear_model import LinearRegression


lm = LinearRegression()
#train model using training data
lm.fit(X_train, Y_train)

#make prediction
Y_pred = lm.predict(X_test)

plt.scatter(Y_test, Y_pred)
plt.plot([min(Y_pred),max(Y_pred)],[min(Y_pred),max(Y_pred)],'--')
plt.xlabel("Prices: $Y_i$")
plt.ylabel("Predicted prices: $\hat{Y}_i$")
plt.title("Prices vs Predicted prices: $Y_i$ vs $\hat{Y}_i$")
```

Linear Regression

supervised.ipynb




```
#calculate mean squared error of the model
mse = sklearn.metrics.mean_squared_error(Y_test, Y_pred)
print(mse)
```

```
28.5413672756
```

Linear Regression

supervised.ipynb

- Underlying relationship should be linear
- Only one Target variable
  - many covariates/independent variables ➔ One Target
- Assumes independent covariates
- Not suitable
  - Non Linear relationships
    - Oversimplification
    - Higher MSE
  - Many Targets
  - Dependency in Covariates

Support Vector Machine

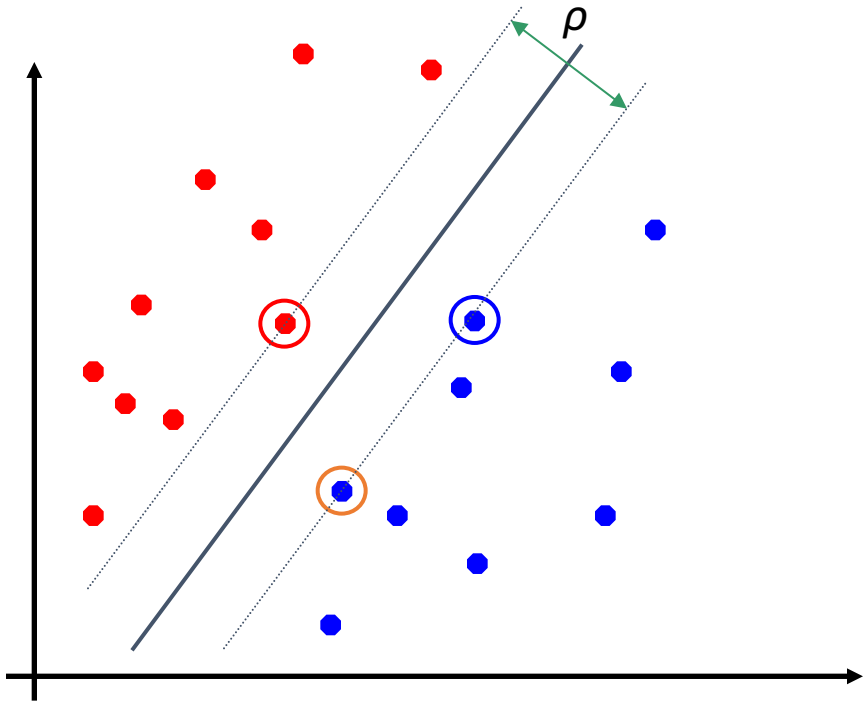
- Given a few sets of labeled training data
  supervised learning
- Generate an optimal hyperplane
- Categorizes new examples.

Binary Classification with Linear Separator

Red and blue dots are representations of objects from two classes in the training data

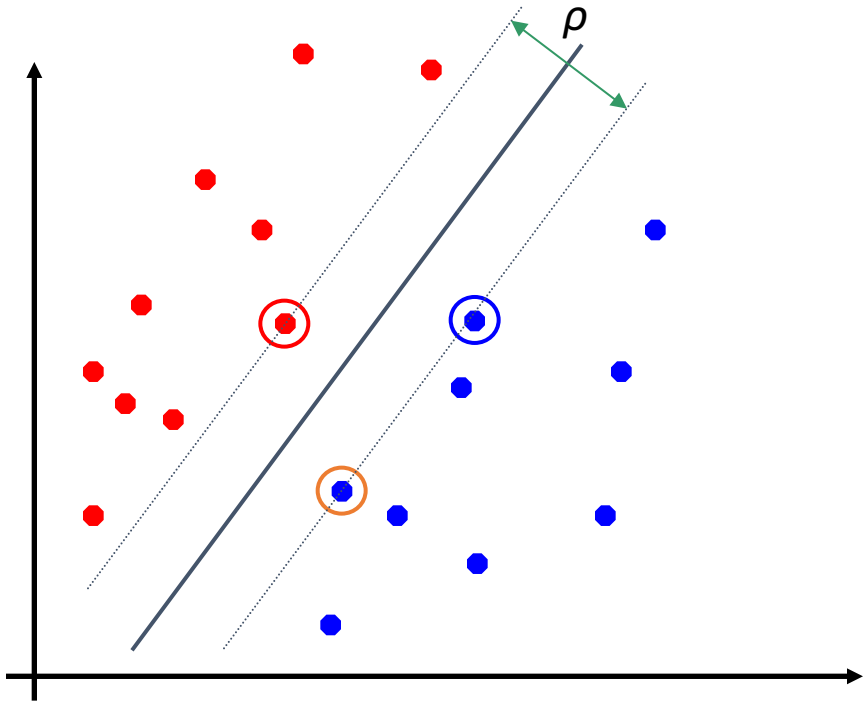
The line is a linear separator for the two classes

The closets objects to the hyperplane is the support vectors



Binary Classification with Linear Separator

- Find a line passing as far as possible fro both points
- The optimal separating hyperplane *maximizes* the margin of the training data.
- A line is bad if it passes too close to some points (noise sensitive)



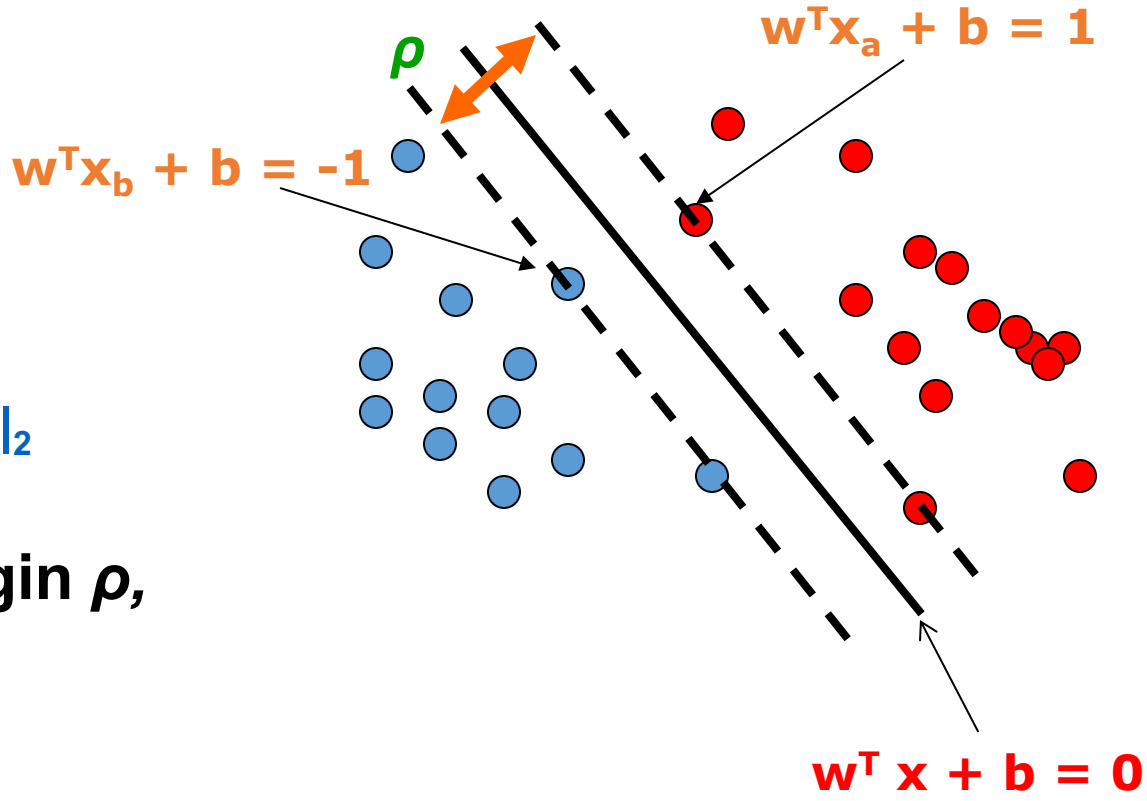
Linear Support Vector Machine

Hyperplane

$\mathbf{w}^T \mathbf{x} + b = 0$

$w^T(x_a - x_b) = 2$

$\rho = ||x_a - x_b||_2 = 2/||w||_2$



Maximize the margin *ρ*,

Minimize **||w||**

Dataset

- Wisconsin Breast Cancer Dataset
  - Available within sklearn.datasets
- 569 Samples
- 30 Features: From cancer diagnostic imaging
  - Area
  - Mean Radius
  - Perimeter
  - ….

Support Vector Machines

In [ ]:
```
# Support vector machine
# Linear Support Vector Machine
from sklearn.svm import SVC
from sklearn.svm import LinearSVC
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
```

In [ ]:
```
cancer = load_breast_cancer()
(X_cancer, y_cancer) = load_breast_cancer(return_X_y = True)
```

In [ ]:
```
X_train, X_test, y_train, y_test = train_test_split(X_cancer, y_cancer, random_state = 0)
```

In [ ]:
```
clf = LinearSVC().fit(X_train, y_train)
```

In [2]:
```
print('Breast cancer dataset')
print('Accuracy of Linear SVC classifier on training set: {:.2f}'
     .format(clf.score(X_train, y_train)))
print('Accuracy of Linear SVC classifier on test set: {:.2f}'
     .format(clf.score(X_test, y_test)))
```

```
Breast cancer dataset
Accuracy of Linear SVC classifier on training set: 0.93
Accuracy of Linear SVC classifier on test set: 0.94
```

In [ ]:

Naïve Bayes Classification

- Bayes Rule

$$P(c_i|x_0, \dots, x_n) = \frac{P(x_0, \dots, x_n|c_i) * P(c_i)}{P(x_0, \dots, x_n)}$$

$$P(c_i|x_0, \dots, x_n) \propto \frac{P(x_0, \dots, x_n|c_i) * P(c_i)}{Const}$$

$$P(c_i|x_0, \dots, x_n) \propto P(x_0, \dots, x_n|c_i) * P(c_i)$$

Naïve Bayes Classification

- Assuming all features are independent (“ Naïve ”)

$$P(x_0, \ldots, x_n | c_i) = \prod P(x_0 | c_i)$$

$$P(c_i | x_0, \ldots, x_n) \propto P(x_0, \ldots, x_n | c_i) * P(c_i)$$

$$P(c_i | x_0, \ldots, x_n) \propto P(c_i) * (\prod P(x_0 | c_i))$$

From Data — From Feature Distribution

- Therefore

$$y = argmax(P(c_i) * (\prod P(x_0 | c_i))$$

Naïve Bayes Classification

	CRIM	ZN	INDUS	CHAS	NOX	RM	AGE	DIS	RAD	TAX	PTRATIO	B	LSTAT	PRICE
0	0.00632	18	2.31	0	0.538	6.575	65.2	4.0900	1	296	15.3	396.90	4.98	24.0
1	0.02731	0	7.07	0	0.469	6.421	78.9	4.9671	2	242	17.8	396.90	9.14	21.6
2	0.02729	0	7.07	0	0.469	7.185	61.1	4.9671	2	242	17.8	392.83	4.03	34.7
3	0.03237	0	2.18	0	0.458	6.998	45.8	6.0622	3	222	18.7	394.63	2.94	33.4
4	0.06905	0	2.18	0	0.458	7.147	54.2	6.0622	3	222	18.7	396.90	5.33	36.2

```
features = bos.drop('CHAS',axis=1)
labels = bos['CHAS']
```

```
#use instead sklearn.model_selection.train_test_split
train, test, train_labels, test_labels = sklearn.model_selection.train_test_split(features,\
                                                                                  labels,\
                                                                                  test_size = 0.33,\
                                                                                  random_state = 5)
```

Naïve Bayes Classification

```
# Initialize our classifier
gnb = GaussianNB()

# Train our classifier
model = gnb.fit(train, train_labels)
```

```
# Make predictions
preds = gnb.predict(test)
print(preds)

# Evaluate accuracy
print(accuracy_score(test_labels, preds))
```

```
[ 1.  0.  0.  0.  1.  0.  1.  1.  1.  0.  1.  0.  1.  0.  0.  0.  0.  1.
  1.  0.  0.  1.  1.  1.  0.  0.  0.  1.  0.  0.  1.  0.  0.  0.  0.  1.
  1.  0.  1.  0.  0.  0.  1.  0.  0.  0.  0.  1.  1.  0.  0.  0.  0.  0.
  0.  0.  1.  0.  0.  0.  1.  0.  1.  1.  1.  0.  0.  0.  0.  0.  0.  1.
  0.  0.  0.  0.  0.  1.  1.  0.  0.  1.  1.  0.  0.  0.  1.  0.  0.  0.
  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.
  0.  0.  0.  1.  0.  0.  0.  0.  1.  0.  0.  0.  0.  0.  0.  0.  0.  0.
  0.  1.  1.  0.  1.  1.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.
  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  0.  1.  0.  0.  0.
  0.  0.  0.  0.  0.]
0.778443113772
```

Naïve Bayes Classification

- Assumes independent features
- Simplifies calculation
  - Fast computation
  - Even in higher dimensions
- Works well in most cases
- Assumptions on the distribution of feature vectors can be changed
  - For text data, Multinomial distribution
  - Most of the time, Normal (Gaussian) distribution

TACC Training Offerings

- TACC Institute Series
  - https://www.tacc.utexas.edu/education/institutes
  - Data Analysis
  - Visualization
  - HPC
  - Sys Admin
  - Life Sciences
- Data Institute
  - Deep Learning (Caffe, Tensorflow, MXNet, Horovod)
  - Machine Learning Methods
  - Hadoop & Spark (Scala)
  - Data Analytics with
    - R
    - Python

References

- Dive Into Python
  - http://www.diveintopython.net/
- Official Python Documentation
  - https://docs.python.org
- Numpy & SciPy
  - https://docs.scipy.org/doc/
  - www.numpy.org
- SciKit-Learn
  - http://scikit-learn.org/
- Always cross check Documentation and Tool/Library/Framework versions

Thanks

- Questions
- Contact: agupta@tacc.utexas.edu
- Issues with Python @ TACC Systems
  - Open a ticket a **http://consult.tacc.utexas.edu**
  - **Queue : DATA INTENSIVE COMPUTING**

Kmeans Clustering

sklearn_kmeans_example.ipynb

```
%matplotlib inline
# Code source: Gaël Varoquaux
# Modified for documentation by Jaques Grobler
# License: BSD 3 clause

import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D


from sklearn.cluster import KMeans
from sklearn import datasets
```

Kmeans Clustering

sklearn_kmeans_example.ipynb

In [145]:
```
np.random.seed(5)

centers = [[1, 1], [-1, -1], [1, -1]]
iris = datasets.load_iris()
```

In [146]:
```
X = iris.data
y = iris.target
```

In [147]:
```
# View the dataset headers
iris.feature_names
```

Out[147]:
```
['sepal length (cm)',
 'sepal width (cm)',
 'petal length (cm)',
 'petal width (cm)']
```

```
[ 5.1,  3.5,  1.4,  0.2],
[ 4.9,  3. ,  1.4,  0.2],
[ 4.7,  3.2,  1.3,  0.2],
[ 4.6,  3.1,  1.5,  0.2],
[ 5. ,  3.6,  1.4,  0.2],
[ 5.4,  3.9,  1.7,  0.4],
[ 4.6,  3.4,  1.4,  0.3],
[ 5. ,  3.4,  1.5,  0.2],
[ 4.4,  2.9,  1.4,  0.2],
[ 4.9,  3.1,  1.5,  0.1],
[ 5.4,  3.7,  1.5,  0.2],
```

Kmeans Clustering

sklearn_kmeans_example.ipynb

```
In [149]: iris.target

Out[149]: array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
                 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
                 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
                 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2])
```

Kmeans Clustering

sklearn_kmeans_example.ipynb

```
In [154]: #set number of clusters
          estimator = KMeans(n_clusters=3)
```

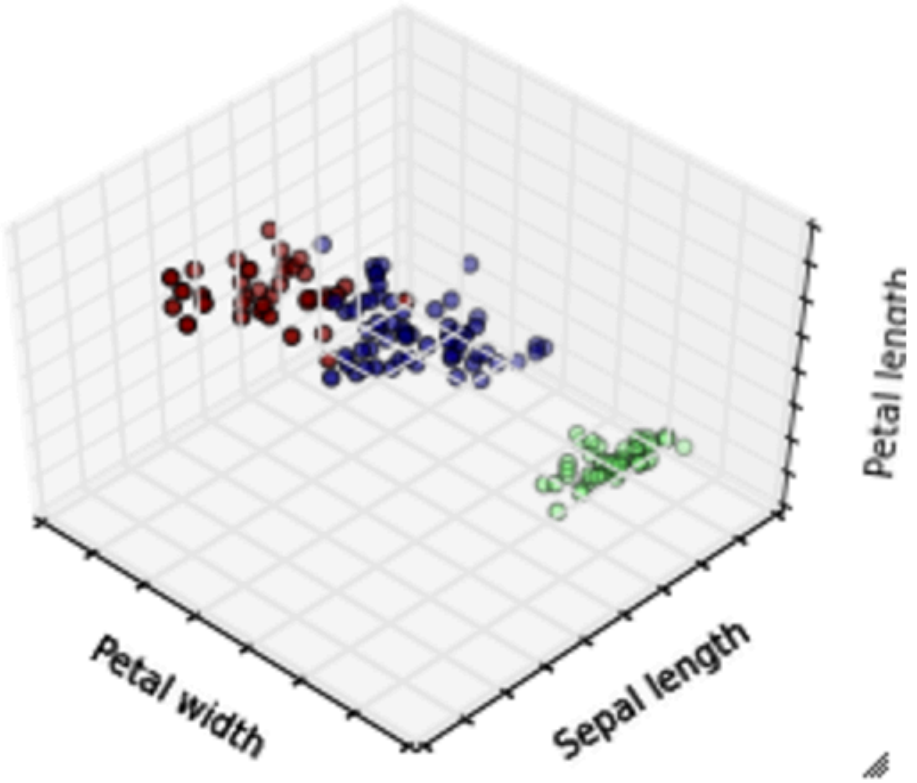
Kmeans Clustering

sklearn_kmeans_example.ipynb

In [151]:
```
fig = plt.figure(1, figsize=(4, 3))
plt.clf()
ax = Axes3D(fig, rect=[0, 0, .95, 1], elev=48, azim=134)
plt.cla()
estimator.fit(X)
labels = estimator.labels_
ax.scatter(X[:, 3], X[:, 0], X[:, 2], c=labels.astype(np.float))
ax.w_xaxis.set_ticklabels([])
ax.w_yaxis.set_ticklabels([])
ax.w_zaxis.set_ticklabels([])
ax.set_xlabel('Petal width')
ax.set_ylabel('Sepal length')
ax.set_zlabel('Petal length')
```

Kmeans Clustering

sklearn_kmeans_example.ipynb



Kmeans Clustering

sklearn_kmeans_example.ipynb