Python for Machine Learning

Devika Subramanian

Rice University

devika@rice.edu

August 2019

Overview

Introduction to Python

Basic types and control flow

Data structures: lists, tuples, strings and dictionaries

Building blocks: functions and modules

Python packages: numpy and pandas

The numpy module

The pandas module

Visualizations using matplotlib

Line plots

Histograms

Scatterplot matrix

Image plots

Contour, surface and 3d plots

1. Introduction to Python 3
    - Basic types, control flow
    - Data structures: lists, tuples, strings and dictionaries

Variables and Types

```
x = 3.1412 # a number
greeting = 'hello there' # a string
alist = [1,1,2,3,5,8,13] # a list
```

Variables and Types

```
x = 3.1412 # a number
greeting = 'hello there' # a string
alist = [1,1,2,3,5,8,13] # a list
```

Variables are dynamically typed. Basic types are integers, double, string, boolean.

```
x = 3.1412         # a number
x = 'hello there'  # a string
x = True           # a boolean
x = [1,4,9,16,25]  # a list
```

Arithmetic operations

```
2 + 3
9 - 2
2/3   # evaluates to 0.6666666666666666
```

Python overloads arithmetic operations on strings.

```
'hello ' + 'world' # string concatenation
'hi ' * 10         # repeated concatenation
```

Comparison and boolean operators

```
x = 1
y = 'hello'

x == 0                # equality check
y != 'hello there'    # inequality check
1 > 1                 # False
2 >= 2                # True
```

Comparison and boolean operators

```
x = 1
y = 'hello'

x == 0              # equality check
y != 'hello there'  # inequality check
1 > 1               # False
2 >= 2              # True
```

Python uses keywords and, or, not to represent Boolean operations.

```
x == 1 or y == 'hi'   # evaluates to True
x > 0 and y != 'hi'   # evaluates to True
not x == 0            # evaluates to True
```

Control flow: if, if/else

The `if` and `if/else` statements are the most basic way to control program flow. As boolean values, the numerical value 0 is False, all other numbers evaluate to True.

```
x = 1
if x:
   print('hello')
else:
   print('hiya')
```

Control flow: for

The Python for iterates over elements in a Python object.

```
alist = [1,2,3,4,5,6,7,8,9,10]
for number in alist:
    print(number, number**2)
```

Data structures: list

The Python `list` is a sequence of items that is indexed by position. Negative indexing is supported.

```
alist = [1,2,3,4,5,6,7,8,9,10]

print(alist[0])  # prints 1
print(alist[2])  # prints 3
print(alist[-1]) # prints 10
```

Operations on lists

To get a complete list of list operations, type `dir(alist)` at the Python interpreter.

```
alist.append(11)
print(alist[10])       # prints 11
blist = [12,13,14,15]
clist = alist + blist   # appends the two lists
clist.pop()             # removes the last element
print(blist[0:2])       # list slice  [12,13]
print(blist[2:] )       #  list slice  [14,15]
len(clist)
max(clist)
min(clist)
```

Iterators over lists

The for and in operators can be use to iterate over and find elements in a list.

```
alist = [2,4,'the quick brown fox', [6,8]]
for e in alist:
    print(e)

if 2 in alist:
   print('2 is in alist')
```

List comprehensions

```
alist = range(5)                    # list [0,1,2,3,4]
blist = [v*2 for v in alist]        # list [0,2,4,6,8]
clist = [v*2 for v in alist if v%2 == 0]
x_coord = [1,2,3]
y_coord = [4,5,6]
# nest list comprehensions
xy_grid = [[x,y] for x in x_coord for y in y_coord]
```

Data structures: Tuples

Tuples are similar to lists, but they are immutable.

```
tup = (2,4,6,8)
tup[2]                 # 6
tup[2:]             #  (6,8)
tup[2] = 7             # error!!
x1,x2,x3,x4 = tup    # unpacking tuples
```

Data structures: Strings

Strings are lists of characters. To see all the operations on strings type dir(str) at the Python interpreter.

```
astr = '12.4,2.3,1.7,6.2\n'
nums = [float(n) for n in astr.strip('\n').split(',')]
```

Data structures: dictionaries

Dictionaries are a set of (key,value) pairs. Entries are indexed by key.

```
adict = {'devika':['6100 Main St','348-5661'],
             'chris':['6100 Main St','348-5690']}
adict['devika']
adict['devika'][0]   # devika's address
adict['devika'][1]   # device's phone number
```

To see all the operations on dictionaries type dir(adict) at the Python interpreter.

Functions

Functions are used to abstract or generalize components of a program.

```
def polyval(p,x):
    vals = [p[i] * (x**i) for i in range(len(p))]
    return sum(vals)

print polyval([1,0,1],4)    # 1 + x**2 = 17 at x = 4
```

Note the indentation convention in a function. Python will complain if your indentation is incorrect.

Indenting functions

```
def fn_error1(x,y,z):
if x < y:
   return z
return 0

def fn_error2(x,y,z):
  if x < y:
  return z
  return 0

def fn_correct(x,y,z):
   if x < y:
     return z
   return 0
```

Modules

Python has many modules with useful functions which effectively extend the programming language. Scipy, numpy, pandas, math are modules that can be imported.

```
import math
print(math.pi)

import numpy as np
print(np.arange(0,10,1))
```

We will see more examples of use of modules in the class.

Numpy

Numpy adds support for large, multi-dimensional arrays and matrices, along with a large library of high-level mathematical functions to operate on these arrays.

- the numpy package uses the `ndarray` object which encapsulates n-dimensional arrays of homogeneous data.
- operations on `ndarray` objects execute in compiled code.
- all mathematical and scientific packages in Python use numpy arrays.

Numpy: array creation

```
import numpy as np
a = np.array([[1,2,3],[4,5,6]])
a = np.arange(0,10,1)
a = np.zeros((5,5))
a = np.ones((3,4))
a = np.diag([1,2,3,4])
a = np.random.random(size=(3,4))
```

Numpy: array attributes

Arrays are objects and have attributes and methods.

```
import numpy as np
a = np.arange(10).reshape((2,5))
a.ndim          # 2 :  number of dimensions
a.shape         # (2,5) : shape of the array
a.size          # 10 : number of elements
a.T             # transpose
a.dtype         # data type of array elements
```

There are more methods and attributes. Use `dir(a)` to explore these.

Numpy: array slicing and indexing

```
import numpy as np
a = np.arange(20).reshape((4,5))
a[0:4,3:5]        # all rows and the last two columns
a[[1,2],:]        # second and third row, all columns
a[[1,2],3:5]      # second and third row, last two columns
a[a%2==0]         # extracts even elements into a 1-d array
```

Numpy: array operations

Arrays are organized by axes, the first axis is row, then column, then the 3rd dimension,...

```
import numpy as np
a = np.random.random((2,5))
a.sum()                 # sum all elements in a
np.sum(a,axis=0)        # column sum
np.sum(a,axis=1)        # row sum
a = np.random.random((2,3,4))
np.sum(a,axis=2)        # returns a (2,3) array
```

Numpy: array operations

By default, array operations work in elementwise fashion.

```
import numpy as np
a1 = np.arange(10).reshape((2,5))
a2 = np.random.random((2,5))
a1+a2              # elementwise sum
a1==a2             # elementwise equality check
np.dot(a1,a2.T)    # matrix product
np.sin(a1)
np.sqrt(a2)
```

Numpy: further reference

Numpy supports vectorized operations which are key to implementing gradient descent efficiently. A comprehensive reference for numpy is at:
`http://docs.scipy.org/doc/numpy/reference/`

The pandas module

pandas is an elegant tool for manipulating tabular data structures. The basic data structure in pandas is the Data Frame which is similar to an R data frame.

```
import pandas as pd
from pandas import DataFrame
years = [1980,1990,2000,2010]
population = [14229191,16986510,20851820,25145561]
df = DataFrame({'year':years,'population':population})
```

Accessing and modifying a dataframe

```
df['year']       # get the year column
df.population    # get the population column
df.iloc[0]         # get the first row
df.iloc[0:2]       # get the first two rows
df['big_pop'] = (df.population > 20000000) # add column
```

Sorting and merging data in a dataframe

```
df=pd.DataFrame(np.random.randn(10,1),columns=['rand'])
df['OriginalOrder']=df.index.values
df=df.sort_values(by='rand',ascending=False) # sort by ran
df=df.sort_values(by='OriginalOrder') # restore df

keyvals=['a','b','c','d','e','f','g','h','i','j']
df1=DataFrame({'rfloat':np.random.randn(10),
                    'key':keyvals})
df2=DataFrame({'rint':np.random.randint(0,5,size=10),
                   'key':keyvals})
df_merge=pd.merge(df1,df2,on='key')
```

Computing aggregations

```
df_merge.groupby('rint').mean()
df_merge.groupby('rint').agg([np.sum,np.mean,np.std])
```

pandas has a lot of functionality not covered here. Please refer to the excellent tutorials at
http://pandas.pydata.org/pandas-docs/stable/tutorials.html.

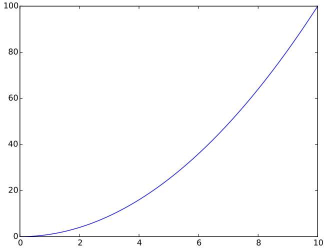
Matplotlib

matplotlib is a python plotting library which produces high quality figures in a variety of formats.

- matplotlib is the standard Python plotting library.
- We will use matplotlib.pyplot for our work here.
- It can create histograms, scatterplots, power spectra, bar charts, error plots, etc. with very few lines of code.

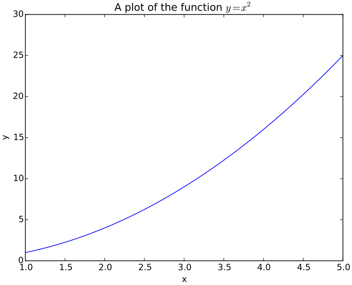
matplotlib: line plot

```
import numpy as np
import matplotlib.pyplot as plt
x = np.linspace(0,10,1000)
y = np.power(x,2)
plt.plot(x,y)
plt.show()
plt.savefig('line_plot.png')
```



matplotlib: line plot titles and labels

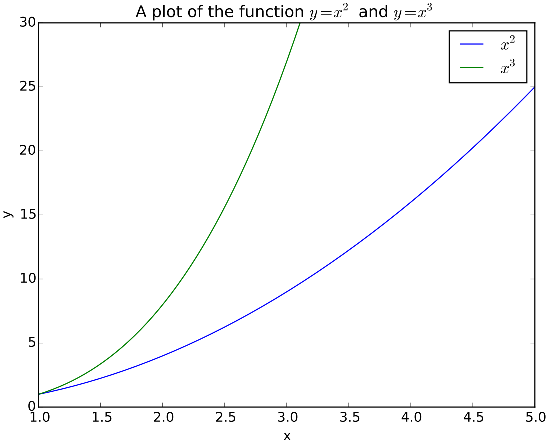
```
plt.xlim((1,5))
plt.ylim((0,30))
plt.xlabel('x')
plt.ylabel('y')
plt.title('A plot of the function $y=x^2$')
plt.savefig('line_plot2.png')
```



matplotlib: multiple lines and legends

```
x = np.linspace(0,10,1000)
y1 = np.power(x,2)
y2 = np.power(x,3)
plt.plot(x,y1,'b-',x,y2,'g-')
plt.xlim((1,5))
plt.ylim((0,30))
plt.xlabel('x')
plt.ylabel('y')
plt.title('A plot of the function $y=x^2$ and $y=x^3$')
plt.legend(('$x^2$','$x^3$'))
plt.savefig('line_plot3.png')
```

matplotlib: multiple lines and legends



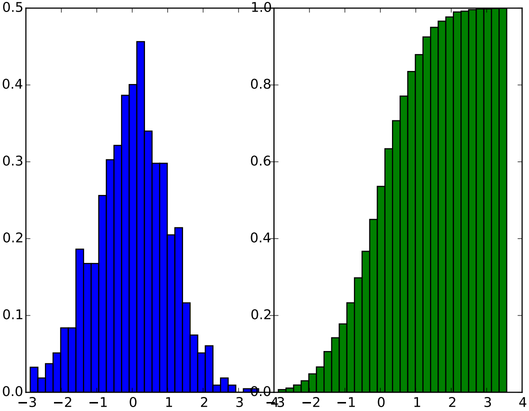
matplotlib: histograms

```
data = np.random.randn(1000)

plt.subplot(1,2,1)
plt.hist(data,bins=30,normed=True,facecolor='b')

plt.subplot(1,2,2)
plt.hist(data,bins=30,normed=True,color='g',
     cumulative=True)
plt.savefig('histogram.png')
```

matplotlib: histograms

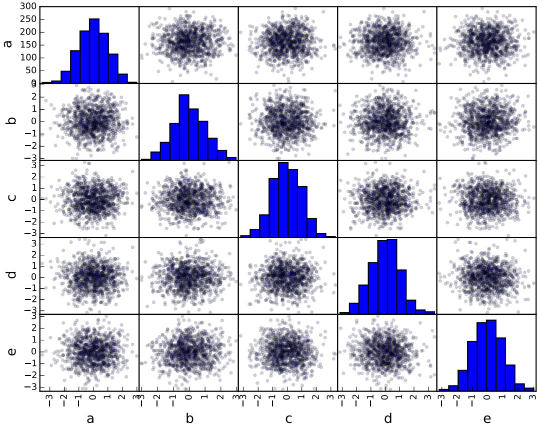


matplotlib: scatterplot matrix

matplotlib has a lot of functionality, but we have to go to a new module called pandas to draw a scatterplot of a data matrix.

```
from pandas.tools.plotting import scatter_matrix
from pandas import DataFrame
df = DataFrame(np.random.randn(1000,5),
        columns=['a','b','c','d','e'])
scatter_matrix(df,alpha=0.2,diagonal='hist')
plt.savefig('scatterplot.png')
```

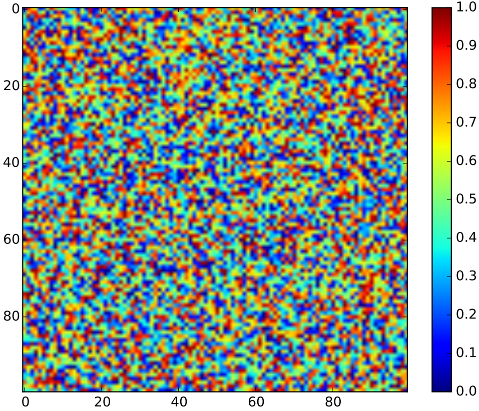
matplotlib: scatterplot matrix



matplotlib: displaying images

```
data = np.random.random((100,100))
plt.imshow(data)
plt.jet()
plt.colorbar()
plt.savefig('imageplot.png')
```

matplotlib: imageplot



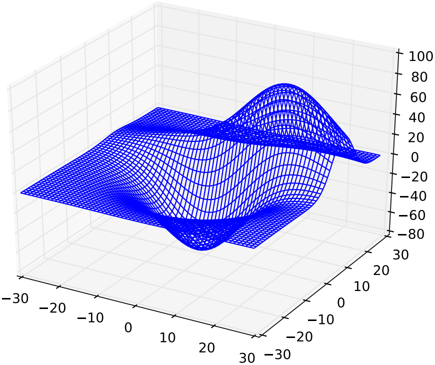
matplotlib: wire plot

matplotlib toolkits allow various 3-d plots: wire frame, contour and surface plots.

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt

ax = plt.subplot(111,projection='3d')
X,Y,Z = axes3d.get_test_data(0.1)
ax.plot_wireframe(X,Y,Z)
plt.savefig('wire.png')
```

matplotlib: wireplot

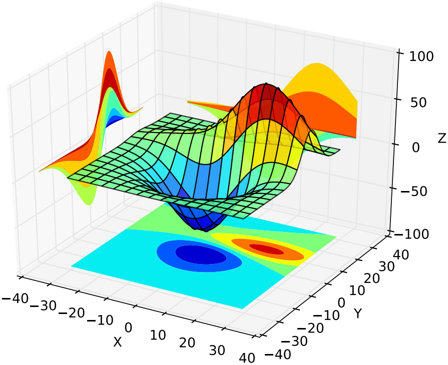


matplotlib: wireplot

matplotlib toolkits allow various 3-d plots: wire frame, contour and surface plots.

```
from mpl_toolkits.mplot3d import axes3d
import matplotlib.pyplot as plt
from matplotlib import cm
ax = plt.subplot(111,projection='3d')
X, Y, Z = axes3d.get_test_data(0.05)
ax.plot_surface(X, Y, Z, rstride=8, cstride=8, alpha=0.8,cmap=cm.jet)
cset = ax.contourf(X, Y, Z, zdir='z', offset=-100, cmap=cm.jet)
cset = ax.contourf(X, Y, Z, zdir='x', offset=-40, cmap=cm.jet)
cset = ax.contourf(X, Y, Z, zdir='y', offset=40, cmap=cm.jet)
ax.set_xlabel('X')
ax.set_xlim(-40, 40)
ax.set_ylabel('Y')
ax.set_ylim(-40, 40)
ax.set_zlabel('Z')
ax.set_zlim(-100, 100)
plt.savefig('contourplot.png')
```

matplotlib: contour plots and surface plots


100
50
0
−50
−100
Z
−40
−30
−20
−10
0
10
20
30
40
X
−40
−30
−20
−10
0
10
20
30
40
Y