2. Machine Learning with Python

SNU
BIOINFORMATICS
INSTITUTE

Table of Contents

What is Anaconda?

- For machine learning, a lot of external libraries are needed.
  - It's too bothersome to install them one by one.
  - As libraries are stacked, there exist the version dependencies among libraries.
  - Integrated ecosystem => Anaconda.
- Anaconda
  - Can install useful packages for data science all at once.
  - Can work as package manager.

Install Anaconda

- How to install?
  - Get script from https://www.continuum.io/downloads
  - bash Anaconda_xxxx.sh
    - Follow the instructions.
    - At last, when prompt asks you to prepend anaconda path at '.barshrc', please type 'yes'.
  - source ~/.bashrc

What is IPython

- IPython is a powerful interactive Python interpreter. In particular, it offers the following functionalities.
  - **Tab-completion**. Autocomplete the name of an object after you've started typing it. To explore the structure of an object, and in particular to get a list of its attributes, simply type object name.
  - **Inline help**. Within IPython, you can get help about a particular module or function by typing help(module name). You can also get detailed information about a particular object by typing object name?.
  - **IPython magic functions**. Magics are predefined functions, which are called by prepending their name with "%". For example,
    - %history prints out all commands typed into the session
    - %paste pastes and run what is copied to your clipboard
    - %run script.py allows you to run the script script.py within Ipython
    - %save saves a set of lines to a file.
    - You can find a list of magics here: http://ipython.org/ipython-doc/stable/interactive/magics.html.

What is IPython (cont.)

- **The IPython Notebook**. An interactive computational environment in which you can combine code execution, rich text, math formulas, plots and more.
  - You use it through your web browser and generate IPython notebooks files in the .ipynb format.
  - They can readily be shared between IPython notebook users, or converted to a variety of formats (including HTML).
  - To set up an IPython notebook, start here: http://ipython.org/ipython-doc/stable/notebook/notebook.html.
  - You will find more information about the IPython Notebook here: http://ipython.org/notebook.html

What is NumPy?

- Numpy is the core library for scientific computing in Python. It provides a high-performance multidimensional array object, and tools for working with these arrays.
  - Array
  - Array indexing
  - Datatypes
  - Array math

NumPy - Array

- A numpy array is a grid of values, all of the same type, and is indexed by a tuple of nonnegative integers.
- The number of dimensions is the rank of the array; the shape of an array is a tuple of integers giving the size of the array along each dimension.
- We can initialize numpy arrays from nested Python lists, and access elements using square brackets.

NumPy - Array

```
import numpy as np

a = np.array([1, 2, 3])   # Create a rank 1 array
print(type(a))            # Prints "<class 'numpy.ndarray'>"
print(a.shape)            # Prints "(3,)"
print(a[0], a[1], a[2])   # Prints "1 2 3"
a[0] = 5                  # Change an element of the array
print(a)                  # Prints "[5, 2, 3]"

b = np.array([[1,2,3],[4,5,6]])    # Create a rank 2 array
print(b.shape)                     # Prints "(2, 3)"
print(b[0, 0], b[0, 1], b[1, 0])   # Prints "1 2 4"
```

SNU BIOINFORMATICS INSTITUTE

NumPy - Array

```
import numpy as np

a = np.zeros((2,2))   # Create an array of all zeros
print(a)              # Prints "[[ 0.  0.]
                      #          [ 0.  0.]]"

b = np.ones((1,2))    # Create an array of all ones
print(b)              # Prints "[[ 1.  1.]]"

c = np.full((2,2), 7)  # Create a constant array
print(c)               # Prints "[[ 7.  7.]
                       #          [ 7.  7.]]"

d = np.eye(2)         # Create a 2x2 identity matrix
print(d)              # Prints "[[ 1.  0.]
                      #          [ 0.  1.]]"

e = np.random.random((2,2))  # Create an array filled with random values
print(e)                     # Might print "[[ 0.91940167  0.08143941]
                             #               [ 0.68744134  0.87236687]]"
```

NumPy - Array

```python
import numpy as np

x = np.arange(6)          # Create an array filled with values of range [0, 6)
print(x)                  # Prints "[0 1 2 3 4 5]"


y = x.reshape((2,3))      # Create an array with same values but different shape
print(y)                  # Prints "[[0 1 2]
                          #          [3 4 5]]"

z = x.reshape((3,2))      # Create an array with same values but different shape
print(y)                  # Prints "[[0 1]
                          #          [2 3]
                          #          [4 5]]"
```

SNU BIOINFORMATICS INSTITUTE

NumPy – Array indexing

- **Slicing:** Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array.

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Use slicing to pull out the subarray consisting of the first 2 rows
# and columns 1 and 2; b is the following array of shape (2, 2):
# [[2 3]
#  [6 7]]
b = a[:2, 1:3]

# A slice of an array is a view into the same data, so modifying it
# will modify the original array.
print(a[0, 1])   # Prints "2"
b[0, 0] = 77     # b[0, 0] is the same piece of data as a[0, 1]
print(a[0, 1])   # Prints "77"
```

SNU BIOINFORMATICS INSTITUTE

NumPy – Array indexing

- **Slicing:** Similar to Python lists, numpy arrays can be sliced. Since arrays may be multidimensional, you must specify a slice for each dimension of the array.

```
import numpy as np

# Create the following rank 2 array with shape (3, 4)
# [[ 1  2  3  4]
#  [ 5  6  7  8]
#  [ 9 10 11 12]]
a = np.array([[1,2,3,4], [5,6,7,8], [9,10,11,12]])

# Two ways of accessing the data in the middle row of the array.
# Mixing integer indexing with slices yields an array of lower rank,
# while using only slices yields an array of the same rank as the
# original array:
row_r1 = a[1, :]    # Rank 1 view of the second row of a
row_r2 = a[1:2, :]  # Rank 2 view of the second row of a
print(row_r1, row_r1.shape)  # Prints "[5 6 7 8] (4,)"
print(row_r2, row_r2.shape)  # Prints "[[5 6 7 8]] (1, 4)"

# We can make the same distinction when accessing columns of an array:
col_r1 = a[:, 1]
col_r2 = a[:, 1:2]
print(col_r1, col_r1.shape)  # Prints "[ 2  6 10] (3,)"
print(col_r2, col_r2.shape)  # Prints "[[ 2]
                             #          [ 6]
                             #          [10]] (3, 1)"
```

NumPy – Array indexing

- **Integer array indexing:** When you index into numpy arrays using slicing, the resulting array view will always be a subarray of the original array. In contrast, integer array indexing allows you to construct arbitrary arrays using the data from another array.

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

# An example of integer array indexing.
# The returned array will have shape (3,) and
print(a[[0, 1, 2], [0, 1, 0]])  # Prints "[1 4 5]"

# The above example of integer array indexing is equivalent to this:
print(np.array([a[0, 0], a[1, 1], a[2, 0]]))  # Prints "[1 4 5]"

# When using integer array indexing, you can reuse the same
# element from the source array:
print(a[[0, 0], [1, 1]])  # Prints "[2 2]"

# Equivalent to the previous integer array indexing example
print(np.array([a[0, 1], a[0, 1]]))  # Prints "[2 2]"
```

NumPy – Array indexing

- **Boolean array indexing:** Boolean array indexing lets you pick out arbitrary elements of an array. Frequently this type of indexing is used to select the elements of an array that satisfy some condition.

```
import numpy as np

a = np.array([[1,2], [3, 4], [5, 6]])

bool_idx = (a > 2)   # Find the elements of a that are bigger than 2;
                     # this returns a numpy array of Booleans of the same
                     # shape as a, where each slot of bool_idx tells
                     # whether that element of a is > 2.

print(bool_idx)      # Prints "[[False False]
                     #          [ True  True]
                     #          [ True  True]]"

# We use boolean array indexing to construct a rank 1 array
# consisting of the elements of a corresponding to the True values
# of bool_idx
print(a[bool_idx])  # Prints "[3 4 5 6]"

# We can do all of the above in a single concise statement:
print(a[a > 2])     # Prints "[3 4 5 6]"
```

NumPy – Datatypes

- Every numpy array is a grid of elements of the same type.
- Numpy provides a large set of numeric datatypes that you can use to construct arrays.
- Numpy tries to guess a datatype when you create an array, but functions that construct arrays usually also include an optional argument to explicitly specify the datatype.

```
import numpy as np

x = np.array([1, 2])   # Let numpy choose the datatype
print(x.dtype)         # Prints "int64"

x = np.array([1.0, 2.0])   # Let numpy choose the datatype
print(x.dtype)             # Prints "float64"

x = np.array([1, 2], dtype=np.int64)   # Force a particular datatype
print(x.dtype)                         # Prints "int64"
```

NumPy – Array math

```
import numpy as np

x = np.array([[1,2],[3,4]], dtype=np.float64)
y = np.array([[5,6],[7,8]], dtype=np.float64)

# Elementwise sum; both produce the array
# [[ 6.0  8.0]
#  [10.0 12.0]]
print(x + y)
print(np.add(x, y))

# Elementwise difference; both produce the array
# [[-4.0 -4.0]
#  [-4.0 -4.0]]
print(x - y)
print(np.subtract(x, y))
```

NumPy – Array math

```
# Elementwise product; both produce the array
# [[ 5.0 12.0]
#  [21.0 32.0]]
print(x * y)
print(np.multiply(x, y))

# Elementwise division; both produce the array
# [[ 0.2         0.33333333]
#  [ 0.42857143  0.5       ]]
print(x / y)
print(np.divide(x, y))

# Elementwise square root; produces the array
# [[ 1.          1.41421356]
#  [ 1.73205081  2.        ]]
print(np.sqrt(x))
```

NumPy – Array math

```
import numpy as np

x = np.array([[1,2],[3,4]])
y = np.array([[5,6],[7,8]])

v = np.array([9,10])
w = np.array([11, 12])

# Inner product of vectors; both produce 219
print(v.dot(w))
print(np.dot(v, w))

# Matrix / vector product; both produce the rank 1 array [29 67]
print(x.dot(v))
print(np.dot(x, v))

# Matrix / matrix product; both produce the rank 2 array
# [[19 22]
#  [43 50]]
print(x.dot(y))
print(np.dot(x, y))
```

SNU BIOINFORMATICS INSTITUTE

NumPy – Array math

```
import numpy as np

x = np.array([[1,2],[3,4]])

print(np.sum(x))  # Compute sum of all elements; prints "10"
print(np.sum(x, axis=0))  # Compute sum of each column; prints "[4 6]"
print(np.sum(x, axis=1))  # Compute sum of each row; prints "[3 7]"
```

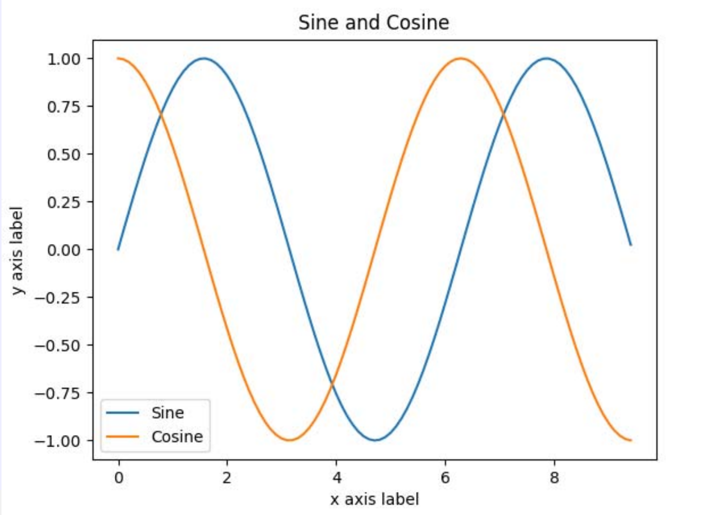
```
import numpy as np

x = np.array([[1,2], [3,4]])
print(x)    # Prints "[[1 2]
            #          [3 4]]"
print(x.T)  # Prints "[[1 3]
            #          [2 4]]"

# Note that taking the transpose of a rank 1 array does nothing:
v = np.array([1,2,3])
print(v)    # Prints "[1 2 3]"
print(v.T)  # Prints "[1 2 3]"
```

SNU BIOINFORMATICS INSTITUTE

Matplotlib

- A plotting library of Python.

```
import numpy as np
import matplotlib.pyplot as plt

# Compute the x and y coordinates for points on sine and cosine curves
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Plot the points using matplotlib
plt.plot(x, y_sin)
plt.plot(x, y_cos)
plt.xlabel('x axis label')
plt.ylabel('y axis label')
plt.title('Sine and Cosine')
plt.legend(['Sine', 'Cosine'])
plt.show()
```



SciPy, Scikit-learn

- Numpy provides a high-performance multidimensional array and basic tools to compute with and manipulate these arrays. SciPy is built on this, and provides large number of functions that operate on Numpy arrays.
- Scikit-learn is a simple and efficient machine learning library which is built on NumPy, SciPy and matplotlib. Scikit-learn provides general algorithms and helper functions.

NumPy exercise

1. Create a 5 x 5 arrays of zeros.
2. Create a 5 x 5 arrays (X) which looks like

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 11 12 13 14]
 [15 16 17 18 19]
 [20 21 22 23 24]]
```

3. Select first row of X.
4. Select last column of X.
5. Change all values of X which is greater than 10 to -1.

```
[[ 0  1  2  3  4]
 [ 5  6  7  8  9]
 [10 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]
 [-1 -1 -1 -1 -1]]
```

Hint: change values of some indexes

```
a = np.array([1,2,3])        # [1 2 3]
a[1:3] = 9
print(a)                     # [1 9 9]
```

SNU BIOINFORMATICS INSTITUTE

Linear Models

- Scikit-learn offers various linear models.
  - 'linear_model' class.
  - For linear model $\hat{y}(x, w) = w_0 + w_1 x_1 + \cdots + w_p x_p$
    - $w = (w_1, \dots, w_p)$ is called as 'coef_'
    - w_0 is called as 'intercept_'

SNU BIOINFORMATICS INSTITUTE

Least Squares Estimation (Linear Regression)

- Linear regression model.
  - $\underset{\theta}{argmin} \frac{1}{2} \sum_{n=1}^{N} \{\hat{y}(x_n, w) - t_n\}^2$

Methods

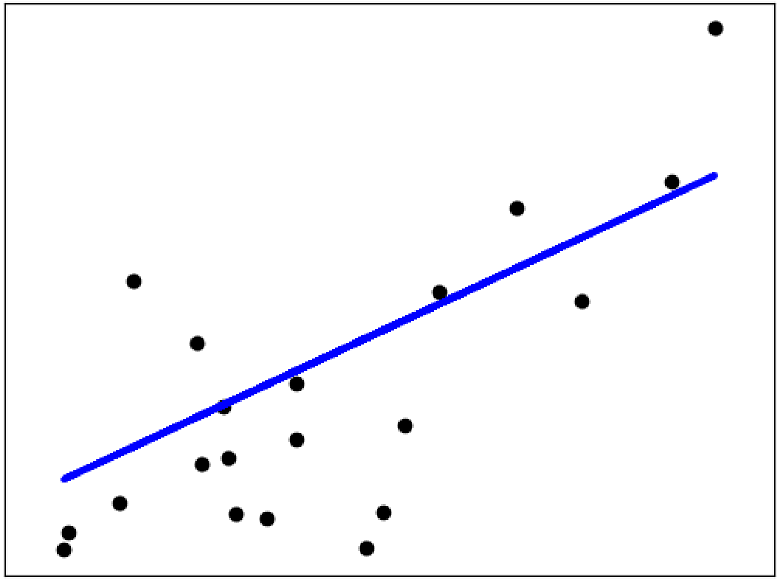
decision_function (*args, **kwargs)	DEPRECATED: and will be removed in 0.19.
fit (X, y[, sample_weight])	Fit linear model.
get_params ([deep])	Get parameters for this estimator.
predict (X)	Predict using the linear model
score (X, y[, sample_weight])	Returns the coefficient of determination R^2 of the prediction.
set_params (**params)	Set the parameters of this estimator.

SNU BIOINFORMATICS INSTITUTE

Least Squares Estimation (Linear Regression)

```
### LR
import numpy as np
from sklearn import datasets, linear_model
import matplotlib.pyplot as plt

# Load the diabetes dataset
diabetes = datasets.load_diabetes()

# diabetes has two attributes: data, target
print(diabetes.data.shape)
print(diabetes.target.shape)

# diabetes consists of 442 samples
#with 10 attributes and 1 real target value.

# Use only one feature
diabetes_X = diabetes.data[:, 2:3]

# Split the data into training/testing sets
diabetes_X_train = diabetes_X[:-20]
diabetes_X_test = diabetes_X[-20:]

# Split the targets into training/testing sets
diabetes_y_train = diabetes.target[:-20]
diabetes_y_test = diabetes.target[-20:]

# Create linear regression object
regr = linear_model.LinearRegression()

# Train the model using the training sets
regr.fit(diabetes_X_train, diabetes_y_train)
```

```
# The coefficients
print('Coefficients: \n', regr.coef_)
# The mean squared error
print("Mean squared error: %.2f"
      % np.mean((regr.predict(diabetes_X_test)
        - diabetes_y_test) ** 2))


# Plot outputs
plt.scatter(diabetes_X_test, diabetes_y_test,  color='black')
plt.plot(diabetes_X_test, regr.predict(diabetes_X_test),
    color='blue', linewidth=3)

plt.xticks(())
plt.yticks(())

plt.show()
```



Maximum Likelihood

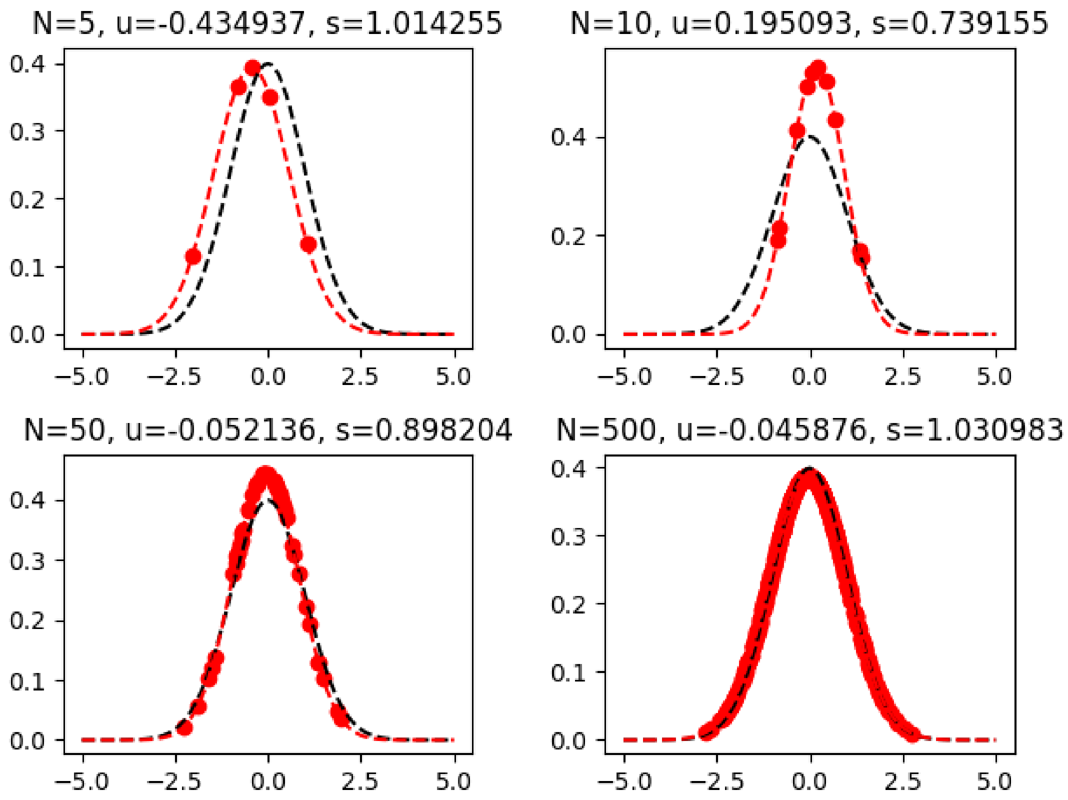
- Estimate normal distribution.
  - Randomly generate points according to normal distribution.
  - By maximum likelihood estimation, estimate original distribution.

Maximum Likelihood

```
######## ML
import numpy as np
import matplotlib.pyplot as plt
from scipy.stats import norm


fig = plt.figure()
for i, N in enumerate([5, 10, 50, 500]):
  # estimate normal distribution
  data = np.random.normal(loc=0, scale=1, size=N) # ~N(0, 1)

  # From pre-calculation,
  u = np.sum(data) / N
  s = np.sqrt(np.sum( (data-u)**2 ) / N)

  # dist
  orig = norm(loc=0, scale=1)
  est = norm(loc=u, scale=s)

  # plot
  subplot = fig.add_subplot(2,2,i+1)
  subplot.set_title("N=%d, u=%f, s=%f" % (N, u, s))

  linex = np.arange(-5,5.1,0.1)
  subplot.plot(linex, orig.pdf(linex), color='black', linestyle='--')
  subplot.plot(linex, est.pdf(linex), color='red', linestyle='--')
  subplot.scatter(data, est.pdf(data), color='red')


plt.tight_layout()
fig.show()
```

INSTITUTE

Maximum Likelihood



SNU BIOINFORMATICS INSTITUTE

Linear Regression Exercise

1. Use boston house-prices dataset.
    1. sklearn.datasets.load_boston()
    2. There are 506 samples with 13 features.
2. Use last feature as ‘x’ value.
3. Among 506 samples, use first 70% of them as training data while using 30% of them as test data.
4. Plot original data points and regression line.

references

- http://cazencott.info/dotclear/public/lectures/ma2823_2015/scikit-learn.pdf
- http://cazencott.info/dotclear/public/lectures/ma2823_2015/scikit-learn-2.pdf
- http://cs231n.github.io/python-numpy-tutorial/