CS229
Python & Numpy

Jingbo Yang, Andrey Kurenkov

How is python related to with others?

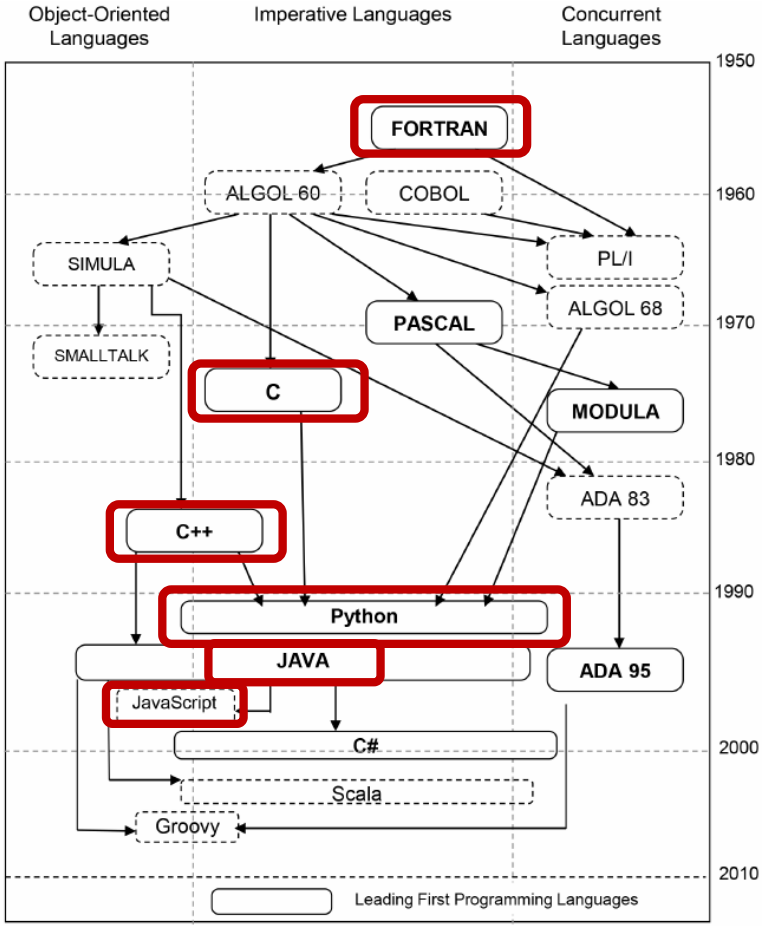
Python 2.0 released in 2000

(Python 2.7 “end-of-life” in 2020)

Python 3.0 released in 2008

(Python 3.6 for CS 229)

Can run interpreted, like MATLAB



https://www.researchgate.net/figure/Genealogy-of-Programming-Languages-doi101371-journalpone0088941g001_fig1_260447599

Before you start

Use Anaconda

Create an environment (full Conda)

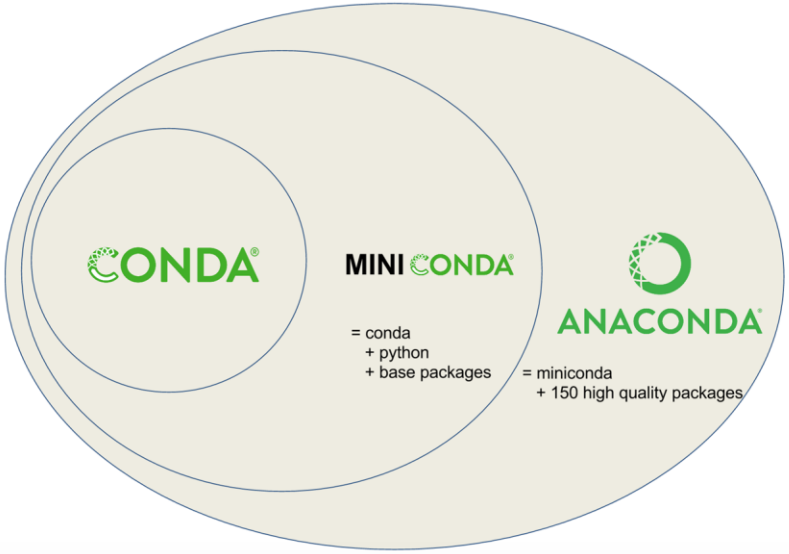
```
conda create -n cs229
```

Create an environment (Miniconda)

```
conda env create -f environment.yml
```

Activate an environment

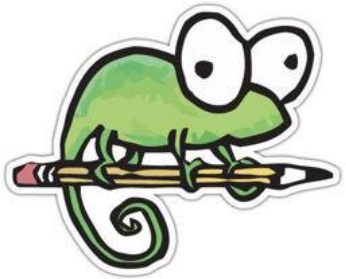
```
conda activate cs229
```



Notepad is not your friend …

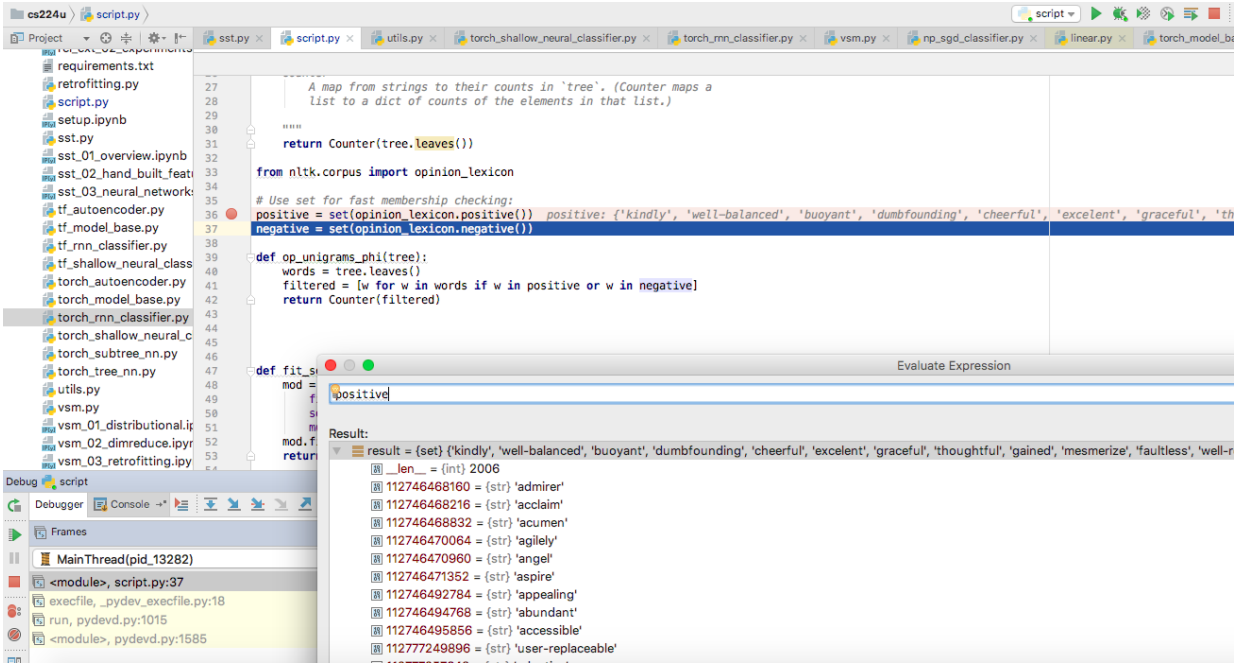
Get a text editor/IDE

- PyCharm (IDE)
- Visual Studio Code (IDE??)
- Sublime Text (IDE??)
- Notepad ++/gedit
- Vim (for Linux)



To make you more prepared

PyCharm magic:



FYI, professional version free for students: https://www.jetbrains.com/student/

Basic Python

Where does my program start?

It just works

```
def do_something(number):
    for i in number:
        print(f'Hello {i}')

do_something(5)
```

A function

Properly

```
def do_something(number):
    for i in number:
        print(f'Hello {i}')

if __name__ == '__main__':
    do_something(5)
```

What is a class?

Initialize the class to get an **instance** using some parameters

Instance variable

Does something with the **instance**

```
class Vehicle:

    def __init__(self, make, name, year,
                 is_electric=False, price=100):
        self.name = name
        self.make = make
        self.year = year
        self.is_electric = is_electric
        self.price = price

        self.odometer = 0

    def drive(self, distance):
        self.odometer += distance

    def compute_price(self):
        if self.is_electric:
            price = self.price / (self.odometer * 0.8)
        else:
            price = self.price / self.odometer
        return price
```

To use a class

Instantiate a class,

get an **instance**

Call an instance method

```
if __name__ == '__main__':

    family_car = Vehicle('Honda', 'Accord', '2019',
                         price=10000)

    print(family_car.compute_price())
    family_car.drive(100)
    print(family_car.compute_price())
```

HW1 with random classifier

Data Structures

Basic data structures

List

```
example_list = [1, 2, '3', 'four’]
```

Set (unordered, unique)

```
example_set = set([1, 2, '3', 'four’])
```

Dictionary (mapping)

```
example_dictionary =
        {
                '1': 'one',
                '2': 'two',
                '3': 'three'
        }
```

More on List

2D list

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

List comprehension

```
initialize_a_list = [i for i in range(9)]
initialize_a_list = [i ** 2 for i in range(9)]
initialize_2d_list = [[i + j for i in range(5)] for j in range(9)]
```

Insert/Pop

```
my_list.insert(0, ‘stuff)
print(my_list.pop(0))
```

More on List

Sort a list

```
random_list = [3,12,5,6]
sorted_list = sorted(random_list)

random_list = [(3, ‘A’),(12, ’D’),(5, ‘M’),(6, ‘B’)]
sorted_list = sorted(random_list, key=lambda x: x[1])
```

More on Dict/Set

Comprehension

```
my_dict = {i: i ** 2 for i in range(10)}
my_set = {i ** 2 for i in range(10)}
```

Get dictionary keys

```
my_dict.keys()
```

Numpy & Scipy

What is Numpy? What is Scipy?

Numpy – package for vector and matrix manipulation

Scipy – package for scientific and technical computing

How do “those guys” make things run faster?

- Read on AVX instruction set (SIMD) and structure of x86 and RISC
- Read on OpenMP and CUDA for multiprocessing
- Read on assembly-level optimization, memory stride, caching, etc.
- Or even about memory management, virtualization
- More bare metal ⟶ FPGA, TPU

Some numpy usage

Popular usage, read before use!

Python Command	Description
scipy.linalg.inv	Inverse of matrix (numpy as equivalent)
scipy.linalg.eig	Get eigen value (Read documentation on eigh and numpy equivalent)
scipy.spatial.distance	Compute pairwise distance
np.matmul	Matrix multiply
np.zeros	Create a matrix filled with zeros (Read on np.ones)
np.arange	Start, stop, step size (Read on np.linspace)
np.identity	Create an identity matrix
np.vstack	Vertically stack 2 arrays (Read on np.hstack)

Your friend for debugging

Python Command	Description
array.shape	Get shape of numpy array
array.dtype	Check data type of array (for precision, for weird behavior)
type(stuff)	Get type of a variable
import pdb; pdb.set_trace()	Set a breakpoint (https://docs.python.org/3/library/pdb.html)
print(f'My name is {name}')	Easy way to construct a message

So many things to remember

Why can't I just write loops?

Remember all the fancy low-level stuff?

How do "those guys" make things run faster?

- Read on AVX instruction set (SIMD) and structure of x86 and RISC
- Read on OpenMP and CUDA for multiprocessing
- Read on assembly-level optimization, memory stride, caching, etc.
- Or even about memory management, virtualization
- More bare metal ⟶ FPGA, TPU

Power of vectorization

```
a = [i for i in range(10000)];
b = [i for i in range(10000)];

tic = time.clock()
dot = 0.0;
for i in range(len(a)):
    dot += a[i] * b[i]
toc = time.clock()

print("dot_product = "+ str(dot));
print("Computation time = " + str(1000*(toc - tic )) + "ms")

n_tic = time.clock()
n_dot_product = np.array(a).dot(np.array(b))
n_toc = time.clock()

print("\nn_dot_product = "+str(n_dot_product))
print("Computation time = "+str(1000*(n_toc - n_tic ))+"ms")
```

```
andrey@regal-algorithm:~/Documents$ python test.py
dot_product = 3.33283335e+11
Computation time = 13.743ms

n_dot_product = 333283335000
Computation time = 5.588ms
```

Plotting

Matplotlib is your friend

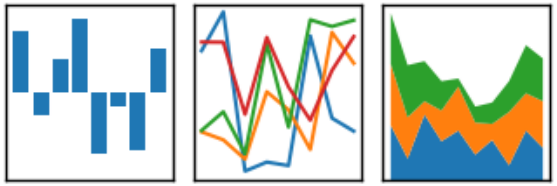
Scatter plot
Line plot
    Duo y-axis
    Log-log
Bar plot (Histogram)
3D plot

Jupyter Notebook is another friend

And if you want to get fancy:

pandas

$y_{it} = \beta' x_{it} + \mu_i + \epsilon_{it}$



Example plots

https://matplotlib.org/3.1.1/gallery/index.html

Import

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

Create data

```
# Data for plotting
t = np.arange(0.0, 2.0, 0.01)
s = 1 + np.sin(2 * np.pi * t)
```

Plotting

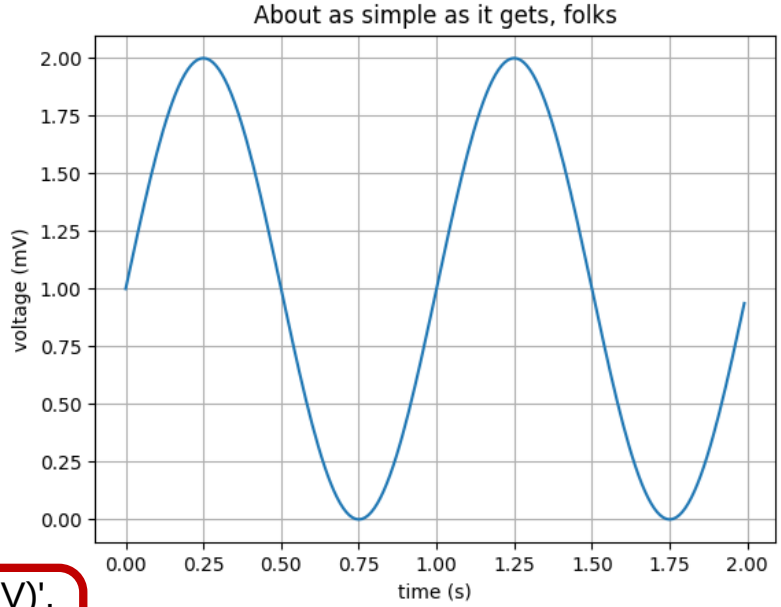
```
fig, ax = plt.subplots()
ax.plot(t, s)
```

Format plot

```
ax.set(xlabel='time (s)', ylabel='voltage (mV)',
       title='About as simple as it gets, folks')
ax.grid()
```

Save/show

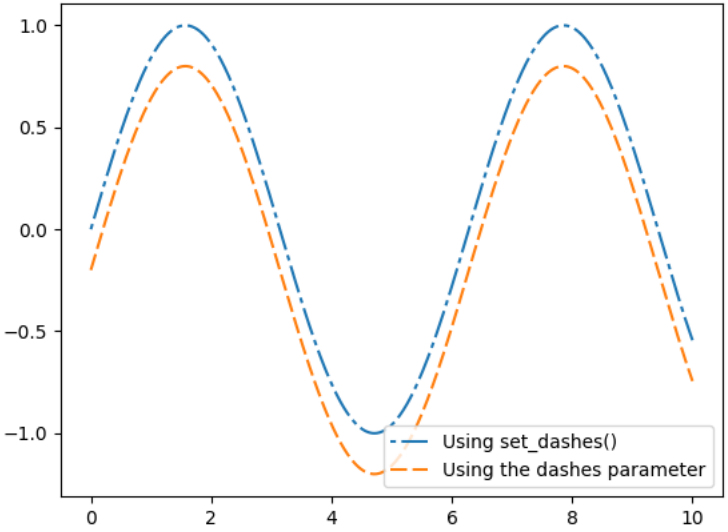
```
fig.savefig("test.png")
plt.show()
```



Plot with dash lines and legend

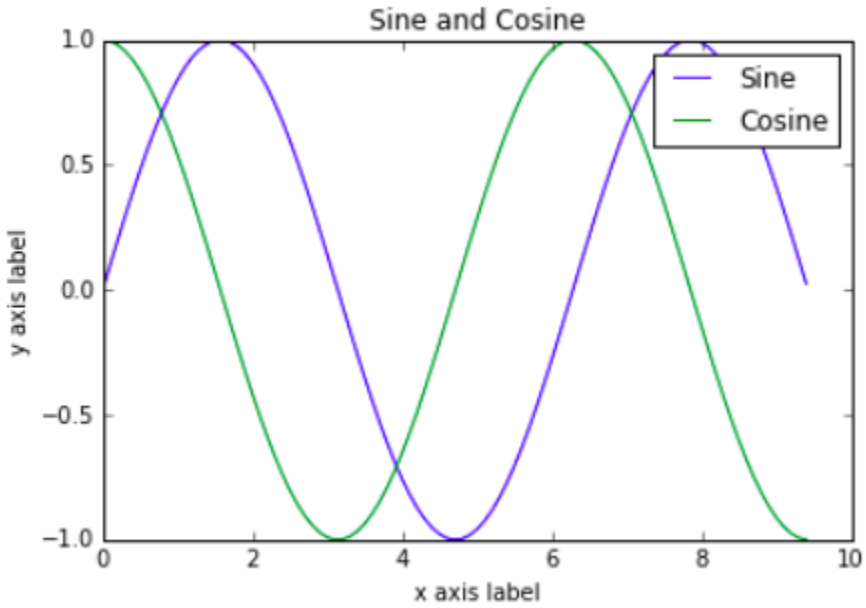
https://matplotlib.org/3.1.1/gallery/index.html

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

x = np.linspace(0, 10, 500)
y = np.sin(x)

fig, ax = plt.subplots()

line1, = ax.plot(x, y, label='Using set_dashes()')
line1.set_dashes([2, 2, 10, 2])  # 2pt line, 2pt break, 10
break

line2, = ax.plot(x, y - 0.2, dashes=[6, 2],
          label='Using the dashes parameter')

ax.legend()
plt.show()
```



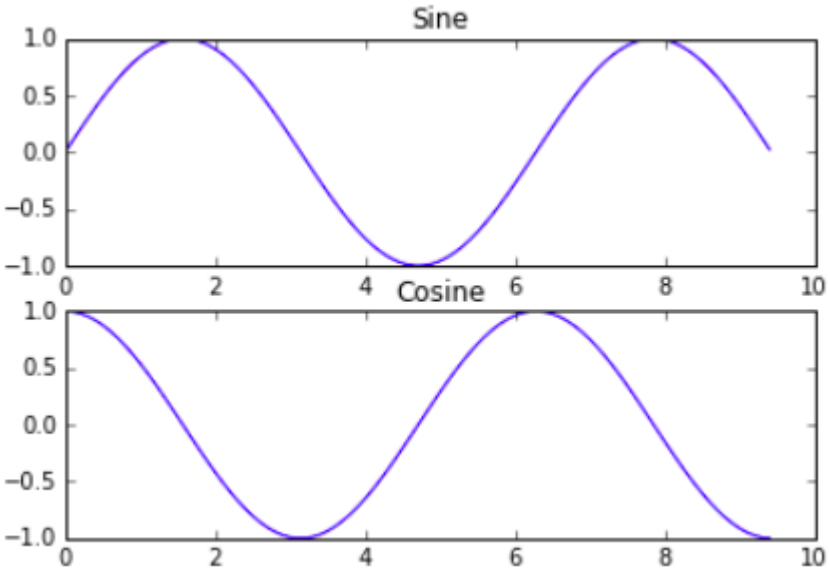
Another way for legend

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

# Compute the x and y coordinates for po
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()
```



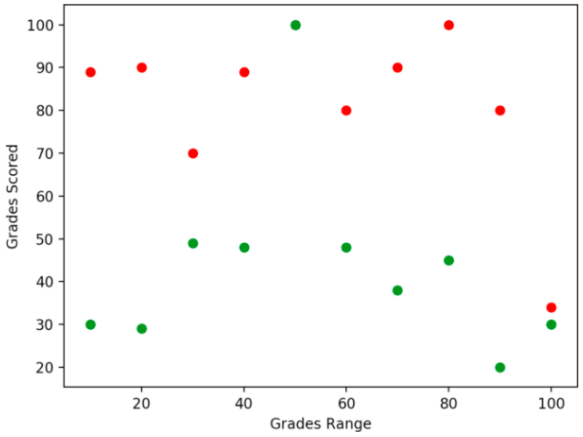
Using subplot

```
x = np.arange(0, 3 * np.pi, 0.1)
y_sin = np.sin(x)
y_cos = np.cos(x)

# Set up a subplot grid that has height 2 and width 1,
# and set the first such subplot as active.
plt.subplot(2, 1, 1)

# Make the first plot
plt.plot(x, y_sin)
plt.title('Sine')

# Set the second subplot as active, and make the second plot.
plt.subplot(2, 1, 2)
plt.plot(x, y_cos)
plt.title('Cosine')

# Show the figure.
plt.show()
```



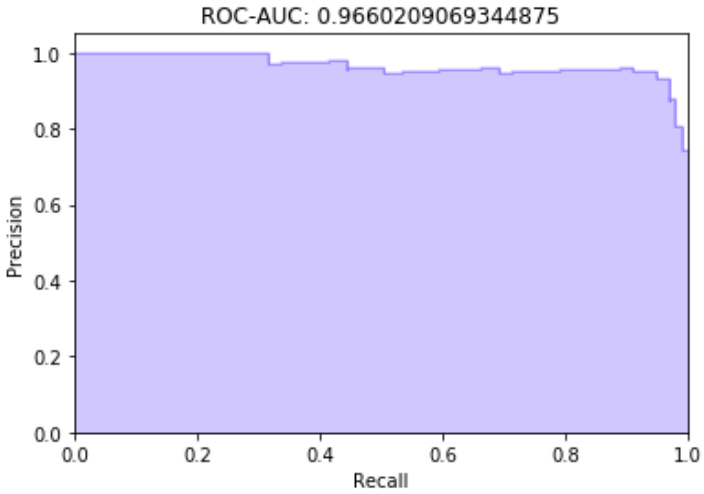
Scatter plot

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

girls_grades = [89, 90, 70, 89, 100, 80, 90, 100, 80, 34]
boys_grades = [30, 29, 49, 48, 100, 48, 38, 45, 20, 30]
grades_range = [10, 20, 30, 40, 50, 60, 70, 80, 90, 100]
plt.scatter(grades_range, girls_grades, color='r')
plt.scatter(grades_range, boys_grades, color='g')
plt.xlabel('Grades Range')
plt.ylabel('Grades Scored')
plt.show()
```



Plot area under curve

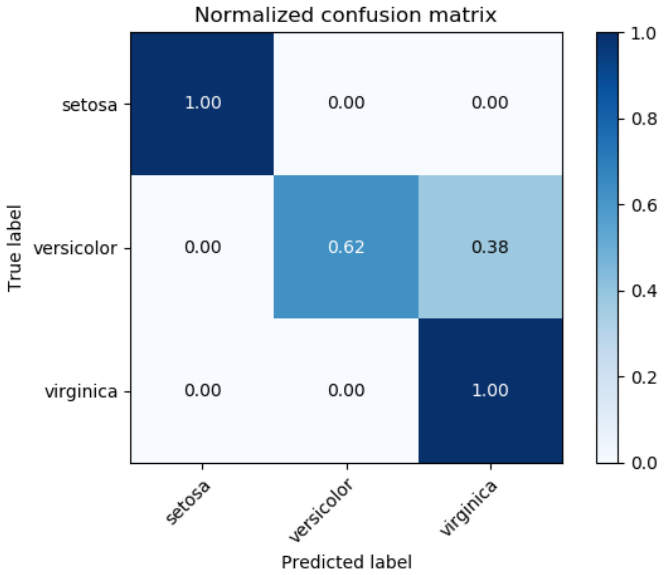
```
def prec_rec_curve(model, X, Y_true, title="", verbose=False):
    probas_pred = model.predict_proba(X)[:, 1]
    pos_label = 1.0
    precision, recall, thresholds = precision_recall_curve(Y_true,
                                                            probas_pred,
                                                            pos_label=pos_label)
    step_kwargs = ({'step': 'post'}
               if 'step' in signature(plt.fill_between).parameters
               else {})
    plt.step(recall, precision, color='b', alpha=0.2,
         where='post')
    plt.fill_between(recall, precision, alpha=0.2, color='b', **step_kwargs)

    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.ylim([0.0, 1.05])
    plt.xlim([0.0, 1.0])
    plt.title(title+ "ROC-AUC: {}".format(auc(recall, precision)))
    plt.show()
```



Confusion matrix

https://scikit-learn.org/stable/auto_examples/model_selection/plot_confusion_matrix.html

```
fig, ax = plt.subplots()
im = ax.imshow(cm, interpolation='nearest', cmap=cmap)
ax.figure.colorbar(im, ax=ax)
# We want to show all ticks...
ax.set(xticks=np.arange(cm.shape[1]),
       yticks=np.arange(cm.shape[0]),
       xticklabels=classes, yticklabels=classes,
       title=title, ylabel='True label', xlabel='Predicted label')

# Rotate the tick labels and set their alignment.
plt.setp(ax.get_xticklabels(), rotation=45, ha="right", rotation_mo

# Loop over data dimensions and create text annotations.
fmt = '.2f' if normalize else 'd'
thresh = cm.max() / 2.
for i in range(cm.shape[0]):
    for j in range(cm.shape[1]):
        ax.text(j, i, format(cm[i, j], fmt),
                ha="center", va="center",
                color="white" if cm[i, j] > thresh else "black")
fig.tight_layout()
```



Numpy & Scipy

What is Numpy? What is Scipy?

Numpy – package for vector and matrix manipulation

Scipy – package for scientific and technical computing

How do “those guys” make things run faster?

- Read on AVX instruction set (SIMD) and structure of x86 and RISC
- Read on OpenMP and CUDA for multiprocessing
- Read on assembly-level optimization, memory stride, caching, etc.
- Or even about memory management, virtualization
- More bare metal → FPGA, TPU

Power of vectorization

```
a = [i for i in range(10000)];
b = [i for i in range(10000)];

tic = time.clock()
dot = 0.0;
for i in range(len(a)):
    dot += a[i] * b[i]
toc = time.clock()

print("dot_product = "+ str(dot));
print("Computation time = " + str(1000*(toc - tic )) + "ms")

n_tic = time.clock()
n_dot_product = np.array(a).dot(np.array(b))
n_toc = time.clock()

print("\nn_dot_product = "+str(n_dot_product))
print("Computation time = "+str(1000*(n_toc - n_tic ))+"ms")
```

```
andrey@regal-algorithm:~/Documents$ python test.py
dot_product = 3.33283335e+11
Computation time = 13.743ms

n_dot_product = 333283335000
Computation time = 5.588ms
```

Popular usage, read before use!

Python Command	Description
scipy.linalg.inv	Inverse of matrix (numpy as equivalent)
scipy.linalg.eig	Get eigen value (Read documentation on eigh and numpy equivalent)
scipy.spatial.distance	Compute pairwise distance
np.matmul	Matrix multiply
np.zeros	Create a matrix filled with zeros (Read on np.ones)
np.arange	Start, stop, step size (Read on np.linspace)
np.identity	Create an identity matrix
np.vstack	Vertically stack 2 arrays (Read on np.hstack)

Your friend for debugging

Python Command	Description
array.shape	Get shape of numpy array
array.dtype	Check data type of array (for precision, for weird behavior)
type(stuff)	Get type of a variable
import pdb; pdb.set_trace()	Set a breakpoint (https://docs.python.org/3/library/pdb.html)
print(f'My name is {name}')	Easy way to construct a message

Links

CS 231N Python Tutorial

Good luck on your HW/Project!

Questions?