INTRODUCTION TO DATA SCIENCE

JOHN P DICKERSON

Lecture #4 – 9/5/2019

CMSC320
Tuesdays & Thursdays
5:00pm – 6:15pm

COMPUTER SCIENCE
UNIVERSITY OF MARYLAND

QUIZ #1 RECAP

TBD

ANNOUNCEMENTS

Register on Piazza: piazza.com/umd/fall2019/cmsc320

- XXX have registered already?!♡
- XXX have not registered yet?! 💔

We will release the first mini-project soon.

- Please make sure Jupyter installed correctly!
- (See any of us if it didn't.)

NEXT FEW CLASSES

1. **NumPy: Python Library for Manipulating nD Arrays**

   Multidimensional Arrays, and a variety of operations including Linear Algebra

2. **Pandas: Python Library for Manipulating Tabular Data**

   Series, Tables (also called **DataFrames**)
   Many operations to manipulate and combine tables/series

3. **Relational Databases**

   Tables/Relations, and SQL (similar to Pandas operations)

4. **Apache Spark**

   Sets of objects or key-value pairs
   MapReduce and SQL-like operations

NEXT FEW CLASSES

1. **NumPy: Python Library for Manipulating nD Arrays**

   Multidimensional Arrays, and a variety of operations including Linear Algebra

2. **Pandas: Python Library for Manipulating Tabular Data**

   Series, Tables (also called **DataFrames**)
   Many operations to manipulate and combine tables/series

3. **Relational Databases**

   Tables/Relations, and SQL (similar to Pandas operations)

4. **Apache Spark**

   Sets of objects or key-value pairs
   MapReduce and SQL-like operations

NUMERIC & SCIENTIFIC APPLICATIONS

Number of third-party packages available for numerical and scientific computing

These include:

- NumPy/SciPy – numerical and scientific function libraries.
- numba – Python compiler that support JIT compilation.
- ALGLIB – numerical analysis library.
- pandas – high-performance data structures and data analysis tools.
- pyGSL – Python interface for GNU Scientific Library.
- ScientificPython – collection of scientific computing modules.

NUMPY AND FRIENDS

By far, the most commonly used packages are those in the NumPy stack. These packages include:

- NumPy: similar functionality as Matlab
- SciPy: integrates many other packages like NumPy
- Matplotlib & Seaborn – plotting libraries
- iPython via Jupyter – interactive computing
- Pandas – data analysis library
- SymPy – symbolic computation library

THE NUMPY STACK

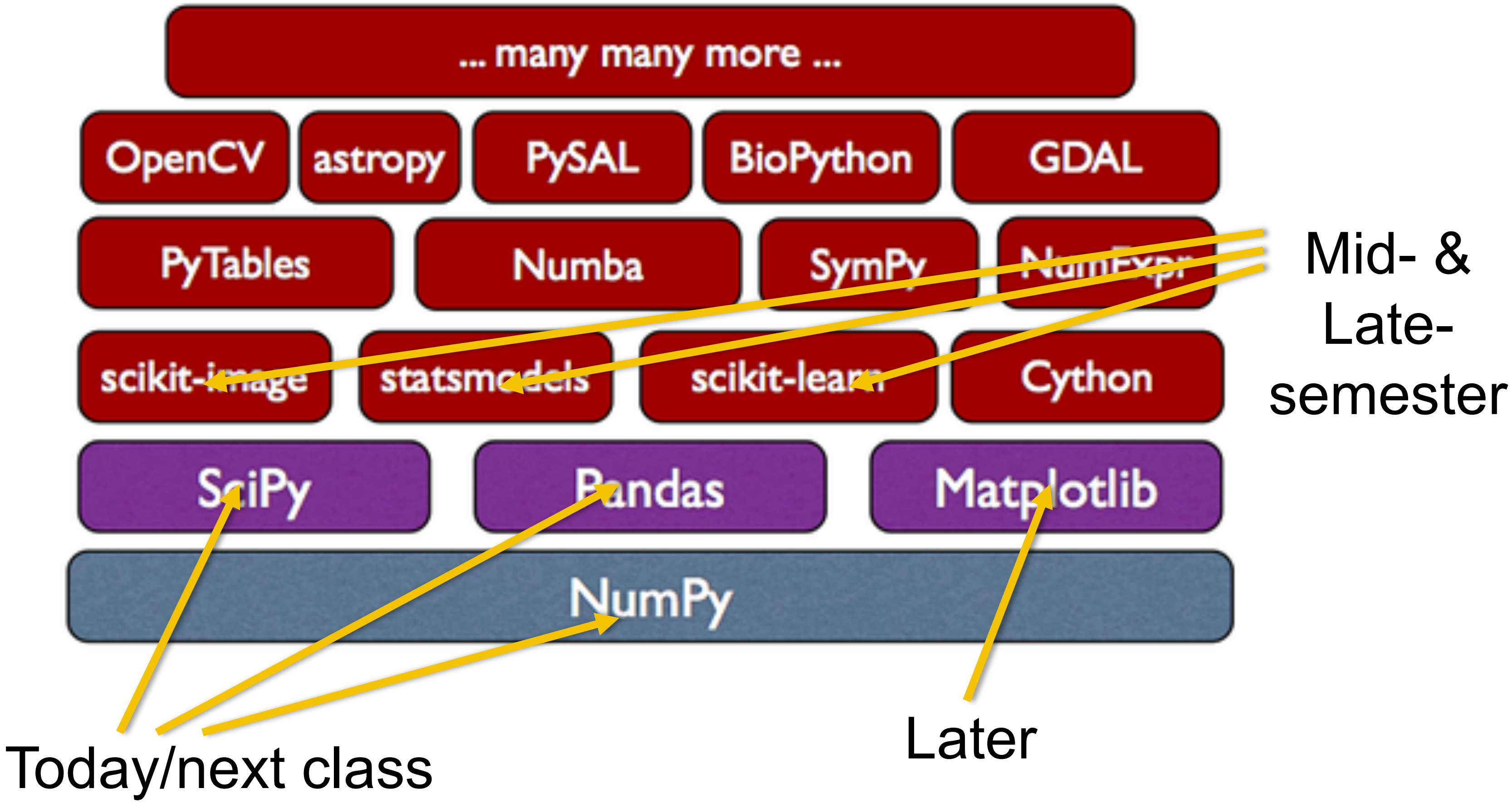


Image from Continuum Analytics

NUMPY

Among other things, NumPy contains:

- A powerful *n*-dimensional array object.
- Sophisticated (broadcasting/universal) functions.
- Tools for integrating C/C++ and Fortran code.
- Useful linear algebra, Fourier transform, and random number capabilities, etc.

Besides its obvious scientific uses, NumPy can also be used as an efficient multi-dimensional container of generic data.

NumPy

[FSU]

NUMPY

`ndarray` object: an *n*-dimensional array of homogeneous data types, with many operations being performed in compiled code for performance

Several important differences between NumPy arrays and the standard Python sequences:

- NumPy arrays have a fixed size. Modifying the size means creating a new array.
- NumPy arrays must be of the same data type, but this can include Python objects – may not get performance benefits
- More efficient mathematical operations than built-in sequence types.

[FSU]

NUMPY DATATYPES

Wider variety of data types than are built-in to the Python language by default.

Defined by the `numpy.dtype` class and include:

- intc (same as a C integer) and intp (used for indexing)
- int8, int16, int32, int64
- uint8, uint16, uint32, uint64
- float16, float32, float64
- complex64, complex128
- bool_, int_, float_, complex_ are shorthand for defaults.

These can be used as functions to cast literals or sequence types, as well as arguments to NumPy functions that accept the `dtype` keyword argument.

NUMPY DATATYPES

```
>>> import numpy as np
>>> x = np.float32(1.0)
>>> x
1.0
>>> y = np.int_([1,2,4])
>>> y
array([1, 2, 4])
>>> z = np.arange(3, dtype=np.uint8)
>>> z
array([0, 1, 2], dtype=uint8)
>>> z.dtype
dtype('uint8')
```

NUMPY ARRAYS

There are a couple of mechanisms for creating arrays in NumPy:

- Conversion from other Python structures (e.g., lists, tuples)
  - Any sequence-like data can be mapped to a ndarray
- Built-in NumPy array creation (e.g., `arange`, `ones`, `zeros`, etc.)
  - Create arrays with all zeros, all ones, increasing numbers from 0 to 1 etc.
- Reading arrays from disk, either from standard or custom formats (e.g., reading in from a CSV file)

[FSU]

NUMPY ARRAYS

In general, any numerical data that is stored in an array-like container can be converted to an `ndarray` through use of the `array()` function. The most obvious examples are sequence types like lists and tuples.

```
>>> x = np.array([2,3,1,0])

>>> x = np.array([2, 3, 1, 0])

>>> x = np.array([[1,2.0],[0,0],(1+1j,3.)])

>>> x = np.array([[ 1.+0.j, 2.+0.j], [ 0.+0.j, 0.+0.j],
[ 1.+1.j, 3.+0.j]])
```

[FSU]

NUMPY ARRAYS

Creating arrays from scratch in NumPy:

- `zeros(shape)` – creates an array filled with 0 values with the specified shape. The default `dtype` is `float64`.

```
>>> np.zeros((2, 3))
array([[ 0., 0., 0.], [ 0., 0., 0.]])
```

- `ones(shape)` – creates an array filled with 1 values.
- `arange()` – like Python's built-in `range`

```
>>> np.arange(10)
array([0, 1, 2, 3, 4, 5, 6, 7, 8, 9])
>>> np.arange(2, 10, dtype=np.float)
array([ 2., 3., 4., 5., 6., 7., 8., 9.])
>>> np.arange(2, 3, 0.2)
array([ 2. , 2.2, 2.4, 2.6, 2.8])
```

[FSU]

NUMPY ARRAYS

`linspace()`– creates arrays with a specified number of elements, and spaced equally between the specified beginning and end values.

```
>>> np.linspace(1., 4., 6)
array([ 1. , 1.6, 2.2, 2.8, 3.4, 4. ])
```

`random.random(shape)` – creates arrays with random floats over the interval [0,1).

```
>>> np.random.random((2,3))
array([[ 0.75688597, 0.41759916, 0.35007419],
       [ 0.77164187, 0.05869089, 0.98792864]])
```

[FSU]

NUMPY ARRAYS

Printing an array can be done with the print

- statement (Python 2)
- function (Python 3)

```
>>> import numpy as np
>>> a = np.arange(3)
>>> print(a)
[0 1 2]
>>> a
array([0, 1, 2])
>>> b = np.arange(9).reshape(3,3)
>>> print(b)
[[0 1 2]
 [3 4 5]
 [6 7 8]]
>>> c =
np.arange(8).reshape(2,2,2)
>>> print(c)
[[[0 1]
  [2 3]]

 [[4 5]
  [6 7]]]
```

[FSU]

INDEXING

Single-dimension indexing is accomplished as usual.

```
>>> x = np.arange(10)
>>> x[2]
2
>>> x[-2]
8
```

Multi-dimensional arrays support multi-dimensional indexing.

```
>>> x.shape = (2,5) # now x is 2-dimensional
>>> x[1,3]
8
>>> x[1,-1]
9
```

INDEXING

Using fewer dimensions to index will result in a subarray:

```
>>> x = np.arange(10)
>>> x.shape = (2,5)
>>> x[0]
array([0, 1, 2, 3, 4])
```

This means that `x[i, j] == x[i][j]` but the second method is less efficient.

INDEXING

Slicing is possible just as it is for typical Python sequences:

```
>>> x = np.arange(10)
>>> x[2:5]
array([2, 3, 4])
>>> x[:-7]
array([0, 1, 2])
>>> x[1:7:2]
array([1, 3, 5])
>>> y = np.arange(35).reshape(5,7)
>>> y[1:5:2,::3]
array([[ 7, 10, 13], [21, 24, 27]])
```

ARRAY OPERATIONS

Basic operations apply element-wise. The result is a new array with the resultant elements.

```
>>> a = np.arange(5)
>>> b = np.arange(5)
>>> a+b
array([0, 2, 4, 6, 8])
>>> a-b
array([0, 0, 0, 0, 0])
>>> a**2
array([ 0,  1,  4,  9, 16])
>>> a>3
array([False, False, False, False,  True], dtype=bool)
>>> 10*np.sin(a)
array([ 0., 8.41470985, 9.09297427, 1.41120008, -
7.56802495])
>>> a*b
array([ 0,  1,  4,  9, 16])
```

ARRAY OPERATIONS

Since multiplication is done element-wise, you need to specifically perform a dot product to perform matrix multiplication.

```
>>> a = np.zeros(4).reshape(2,2)
>>> a
array([[ 0.,  0.],
       [ 0.,  0.]])
>>> a[0,0] = 1
>>> a[1,1] = 1
>>> b = np.arange(4).reshape(2,2)
>>> b
array([[0, 1],
       [2, 3]])
>>> a*b
array([[ 0.,  0.],
       [ 0.,  3.]])
>>> np.dot(a,b)
array([[ 0.,  1.],
       [ 2.,  3.]])
```

ARRAY OPERATIONS

There are also some built-in methods of `ndarray` objects.

Universal functions which may also be applied include `exp`, `sqrt`, `add`, `sin`, `cos`, etc.

```
>>> a = np.random.random((2,3))
>>> a
array([[ 0.68166391, 0.98943098,
0.69361582],
       [ 0.78888081, 0.62197125,
0.40517936]])
>>> a.sum()
4.1807421388722164
>>> a.min()
0.4051793610379143
>>> a.max(axis=0)
array([ 0.78888081, 0.98943098,
0.69361582])
>>> a.min(axis=1)
array([ 0.68166391, 0.40517936])
```

ARRAY OPERATIONS

An array shape can be manipulated by a number of methods.

`resize(size)` will modify an array in place.

`reshape(size)` will return a copy of the array with a new shape.

```
>>> a =
np.floor(10*np.random.random((3,4)))
>>> print(a)
[[ 9.  8.  7.  9.]
 [ 7.  5.  9.  7.]
 [ 8.  2.  7.  5.]]
>>> a.shape
(3, 4)
>>> a.ravel()
array([ 9.,  8.,  7.,  9.,  7.,  5.,  9.,
7.,  8.,  2.,  7.,  5.])
>>> a.shape = (6,2)
>>> print(a)
[[ 9.  8.]
 [ 7.  9.]
 [ 7.  5.]
 [ 9.  7.]
 [ 8.  2.]
 [ 7.  5.]]
>>> a.transpose()
array([[ 9.,  7.,  7.,  9.,  8.,  7.],
       [ 8.,  9.,  5.,  7.,  2.,  5.]])
```

LINEAR ALGEBRA

One of the most common reasons for using the NumPy package is its linear algebra module.

It's like Matlab, but free!

```
>>> from numpy import *
>>> from numpy.linalg import *
>>> a = array([[1.0, 2.0],
               [3.0, 4.0]])
>>> print(a)
[[ 1. 2.]
 [ 3. 4.]]
>>> a.transpose()
array([[ 1., 3.],
       [ 2., 4.]])
>>> inv(a) # inverse
array([[-2. , 1. ],
       [ 1.5, -0.5]])
```

LINEAR ALGEBRA

```
>>> u = eye(2) # unit 2x2 matrix; "eye" represents "I"
>>> u
array([[ 1., 0.],
       [ 0., 1.]])
>>> j = array([[0.0, -1.0], [1.0, 0.0]])
>>> dot(j, j) # matrix product
array([[-1., 0.],
       [ 0., -1.]])
>>> trace(u) # trace (sum of elements on diagonal)
2.0
>>> y = array([[5.], [7.]])
>>> solve(a, y) # solve linear matrix equation
array([[-3.],
       [ 4.]])
>>> eig(j) # get eigenvalues/eigenvectors of matrix
(array([ 0.+1.j, 0.-1.j]),
 array([[ 0.70710678+0.j, 0.70710678+0.j],
        [ 0.00000000-0.70710678j,
0.00000000+0.70710678j]]))
```

SCIPY?

In its own words:

> SciPy is a collection of mathematical algorithms and convenience functions built on the NumPy extension of Python. It adds significant power to the interactive Python session by providing the user with high-level commands and classes for manipulating and visualizing data.

Basically, SciPy contains various tools and functions for solving common problems in scientific computing.

SCIPY

SciPy gives you access to a ton of specialized mathematical functionality.

- **Just know it exists.** **We won't use it much in this class.**

Some functionality:

- Special mathematical functions (scipy.special) -- elliptic, bessel, etc.
- Integration (scipy.integrate)
- Optimization (scipy.optimize)
- Interpolation (scipy.interpolate)
- Fourier Transforms (scipy.fftpack)
- Signal Processing (scipy.signal)
- Linear Algebra (scipy.linalg)
- Compressed Sparse Graph Routines (scipy.sparse.csgraph)
- Spatial data structures and algorithms (scipy.spatial)
- Statistics (scipy.stats)
- Multidimensional image processing (scipy.ndimage)
- Data IO (scipy.io) – overlaps with pandas, covers some other formats

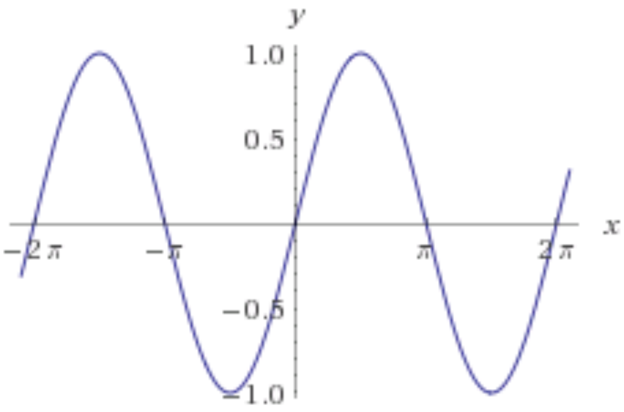
ONE SCIPY EXAMPLE

We can't possibly tour all of the SciPy library and, even if we did, it might be a little boring.

- Often, you'll be able to find higher-level modules that will work around your need to directly call low-level SciPy functions

Say you want to compute an integral:

$$\int_a^b \sin x \, dx$$



SCIPY.INTEGRATE

We have a function object – `np.sin` defines the sin function for us.

We can compute the definite integral from $x = 0$ to $x = \pi$ using the quad function.

```
>>> res = scipy.integrate.quad(np.sin, 0, np.pi)
>>> print(res)
(2.0, 2.220446049250313e-14) # 2 with a very small error margin!
>>> res = scipy.integrate.quad(np.sin, -np.inf, +np.inf)
>>> print(res)
(0.0, 0.0) # Integral does not converge
```

SCIPY.INTEGRATE

Let's say that we don't have a function object, we only have some (*x*,*y*) samples that "define" our function.

We can estimate the integral using the trapezoidal rule.

```
>>> sample_x = np.linspace(0, np.pi, 1000)
>>> sample_y = np.sin(sample_x) # Creating 1,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
1.99999835177

>>> sample_x = np.linspace(0, np.pi, 1000000)
>>> sample_y = np.sin(sample_x) # Creating 1,000,000 samples
>>> result = scipy.integrate.trapz(sample_y, sample_x)
>>> print(result)
2.0
```

WRAP UP: FIRST PART

Shift thinking from imperative coding to operations on datasets

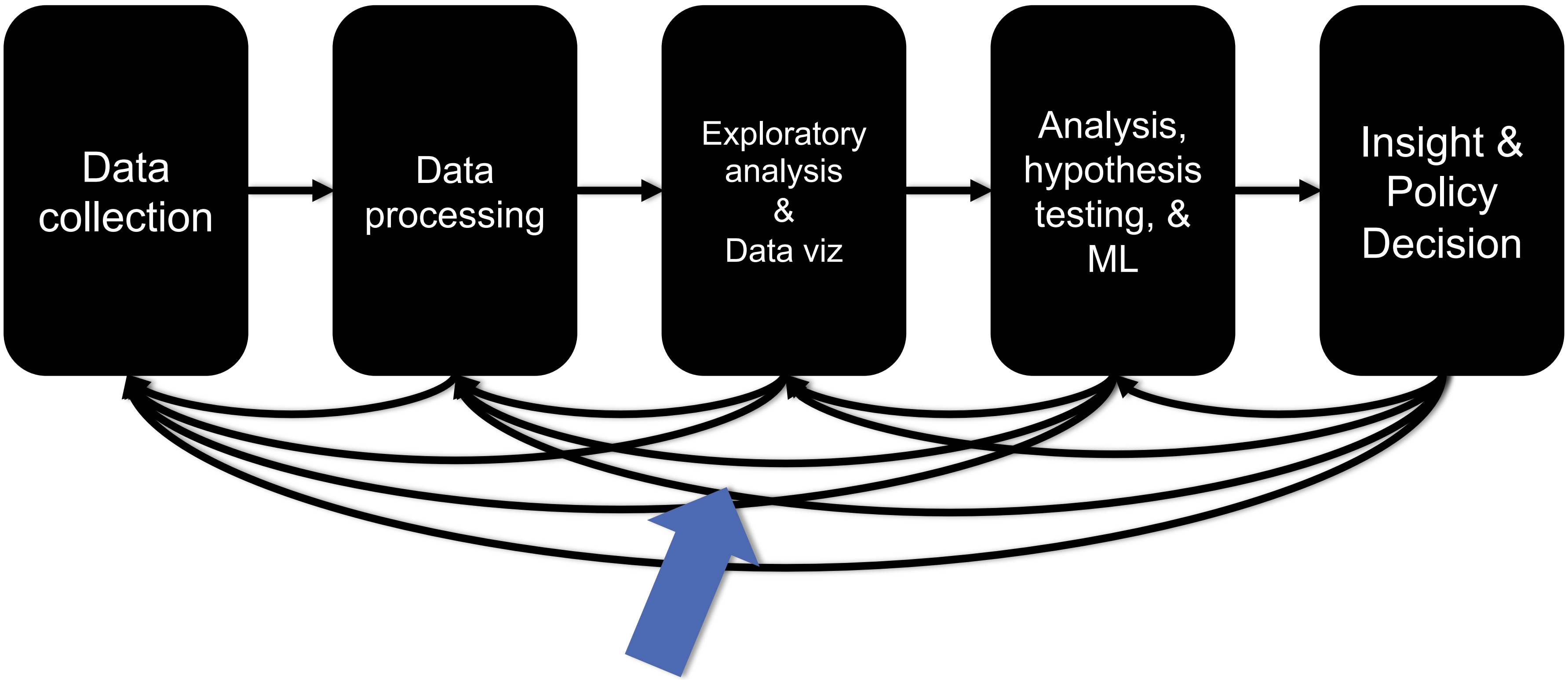
Numpy: A low-level abstraction that gives us really fast multi-dimensional arrays

Next class:

Pandas: Higher-level tabular abstraction and operations to manipulate and combine tables

Reading Homework focuses on Pandas and SQL: Aim to release by tomorrow

REMAINDER OF TODAY'S LECTURE



Quick best practices
for managing this monstrosity.

REPRODUCIBILITY

Extremely important aspect of data analysis

- "Starting from the same raw data, can we reproduce your analysis and obtain the same results?"

Using libraries helps:

- Since you don't reimplement everything, reduce programmer error
- Large user bases serve as "watchdog" for quality and correctness

Standard practices help:

- Version control: git, git, git, …, git, svn, cvs, hg, Dropbox
- Unit testing: unittest (Python), RUnit (R), testthat
- Share and publish: github, gitlab

Many slides in this lecture adapted from Hector Corrado Bravo

REPRODUCIBILITY

Open data:

"**Open data** is the idea that some data should be freely available to everyone to use and republish as they wish, without restrictions from copyright, patents or other mechanisms of control"

Open Data movement website

- http://www.opendatafoundation.org/

PRACTICAL TIPS

Many tasks can be organized in modular manner:

- Data acquisition:
  - Get data, put it in usable format (many 'join' operations), clean it up
- Algorithm/tool development:
  - If new analysis tools are required
- Computational analysis:
  - Use tools to analyze data
- Communication of results:
  - Prepare summaries of experimental results, plots, publication, upload processed data to repositories

Usually a single language or tool does not handle all of these equally well – **choose the best tool for the job!**

PRACTICAL TIPS

Modularity requires organization and careful thought

In Data Science, we wear two hats:

- Algorithm/tool developer
- **Experimentalist**: we don’t get trained to think this way enough!

It helps two consciously separate these two jobs

THINK LIKE AN EXPERIMENTALIST

Plan your experiment

Gather your raw data

Gather your tools

Execute experiment

Analyze

Communicate

THINK LIKE AN EXPERIMENTALIST

Let this guide your organization. One potential structure for organizing a project:

```
project/
| data/
| | processing_scripts
| | raw/
| | proc/
| tools/
| | src/
| | bin/
| exps
| | pipeline_scripts
| | results/
| | analysis_scripts
| | figures/
```

THINK LIKE AN EXPERIMENTALIST

Keep a lab notebook!

Literate programming tools are making this easier for computational projects:

- http://en.wikipedia.org/wiki/Literate_programming (Lec #2!)
- https://ipython.org/
- http://rmarkdown.rstudio.com/
- http://jupyter.org/

THINK LIKE AN EXPERIMENTALIST

Separate experiment from analysis from communication

- Store results of computations, write separate scripts to analyze results and make plots/tables

Aim for reproducibility

- There are serious consequences for not being careful
  - Publication retraction
  - Worse: http://videolectures.net/cancerbioinformatics2010_baggerly_irrh/
- Lots of tools available to help, use them! Be proactive: learn about them on your own!

BIAS, ETHICS, & RESPONSIBILITY

DATA SCIENCE LIFECYCLE: AN ALTERNATE VIEW

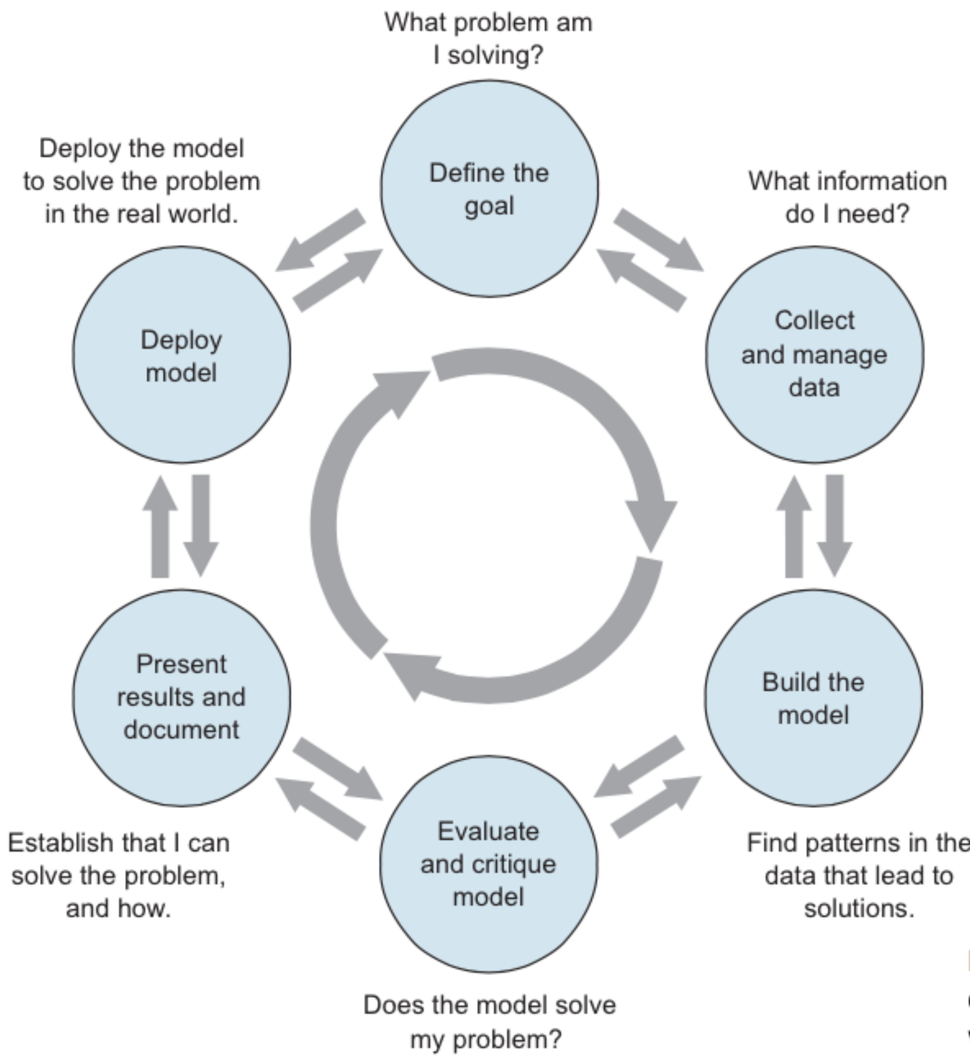


Figure 1.1 The lifecycle of a data science project: loops within loops

EXAMPLES OF BIAS

Genetic testing

- Genetic tests for heart disorder and race-biased risk (NYTimes)
- Race-bias in ancestry reports

Search results / feed optimization

- Google
- Facebook

COMBATING BIAS

Fairness through blindness:

- Don't let an algorithm look at protected attributes

Examples currently in use ??????????

- Race
- Gender
- Sexuality
- Disability
- Religion

Problems with this approach ?????????

COMBATING BIAS

Demographic parity:

- A decision must be independent of the protected attribute
- E.g., a loan application's acceptance rate is independent of an applicant's race (but can be depenedent on non-protected features like salary)

Formally: binary decision variable C, protected attribute A

- $P\{ C = 1 \mid A = 0 \} = P\{ C = 1 \mid A = 1 \}$

Membership in a protected class should have no correlation with the final decision.

- Problems ????????

Example from Moritz Hardt's blog

COMBATING BIAS

What if the decision isn't the thing that matters?

"Consider, for example, a luxury hotel chain that renders a promotion to a subset of wealthy whites (who are likely to visit the hotel) and a subset of less affluent blacks (who are unlikely to visit the hotel). The situation is obviously quite icky, but demographic parity is completely fine with it so long as the same fraction of people in each group see the promotion."

Demographic parity allows classifiers that select qualified candidates in the "majority" demographic and unqualified candidate in the "minority" demographic, within a protected attribute, so long as the expected percentages work out.

More: http://blog.mrtz.org/2016/09/06/approaching-fairness.html

Example from Moritz Hardt's blog

FATML

This stuff is really tricky (and really important).

- It's also not solved, even remotely, yet!
- CMSC498/499 ♡

New community: Fairness, Accountability, and Transparency in Machine Learning (aka **FATML**)

"… policymakers, regulators, and advocates have expressed fears about the potentially discriminatory impact of machine learning, with many calling for further technical research into the dangers of inadvertently encoding bias into automated decisions."

F IS FOR FAIRNESS

In large data sets, there is always proportionally less data available about minorities.

Statistical patterns that hold for the majority may be invalid for a given minority group.

Fairness can be viewed as a measure of diversity in the combinatorial space of sensitive attributes, as opposed to the geometric space of features.

Thanks to: Faez Ahmed

A IS FOR ACCOUNTABILITY

Accountability of a mechanism implies an obligation to report, explain, or justify algorithmic decision-making as well as mitigate any negative social impacts or potential harms.

- Current accountability tools were developed to oversee human decision makers
- They often fail when applied to algorithms and mechanisms instead

Example, no established methods exist to judge the intent of a piece of software. Because automated decision systems can return potentially incorrect, unjustified or unfair results, additional approaches are needed to make such systems accountable and governable.

Thanks to: Faez Ahmed

T IS FOR TRANSPARENCY

Automated ML-based algorithms make many important decisions in life.

- Decision-making process is opaque, hard to audit

A transparent mechanism should be:

- understandable;
- more meaningful;
- more accessible; and
- more measurable.

Thanks to: Faez Ahmed

DATA COLLECTION

What data should (not) be collected

Who owns the data

Whose data can (not) be shared

What technology for collecting, storing, managing data

Whose data can (not) be traded

What data can (not) be merged

What to do with prejudicial data

Thanks to: Kaiser Fung

DATA MODELING

Data is biased (known/unknown)

- Invalid assumptions
- Confirmation bias

Publication bias

- WSDM 2017: https://arxiv.org/abs/1702.00502

Badly handling missing values

Thanks to: Kaiser Fung

DEPLOYMENT

Spurious correlation / over-generalization

Using “black-box” methods that cannot be explained

Using heuristics that are not well understood

Releasing untested code

Extrapolating

Not measuring lifecycle performance (concept drift in ML)

We will go over ways to counter this in the ML/stats/hypothesis testing portion of the course

Thanks to: Kaiser Fung

GUIDING PRINCIPLES

Start with clear user need and public benefit

Use data and tools which have minimum intrusion necessary

Create robust data science models

Be alert to public perceptions

Be as open and accountable as possible

Keep data secure

GOV.UK

Thanks to: UK cabinet office

SOME REFERENCES

Presentation on ethics and data analysis, Kaiser Fung @ Columbia Univ. http://andrewgelman.com/wp-content/uploads/2016/04/fung_ethics_v3.pdf

O'Neil, Weapons of math destruction. https://www.amazon.com/Weapons-Math-Destruction-Increases-Inequality/dp/0553418815

UK Cabinet Office, Data Science Ethical Framework. https://www.gov.uk/government/publications/data-science-ethical-framework

Derman, Modelers' Hippocratic Oath. http://www.iijournals.com/doi/pdfplus/10.3905/jod.2012.20.1.035

Nick D's MIT Tech Review Article. https://www.technologyreview.com/s/602933/how-to-hold-algorithms-accountable/

NEXT CLASS:

PANDAS, TIDY DATA & (MAYBE) START ON RELATIONAL MODEL OF DATA