

Data Science Ethics: Data Integration

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



OpenDS4All

*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-28-22>

Recall

- At the beginning of the semester, we spoke about why ethics are important in Data Science
- ... and we discussed the ethics in data acquisition, in particular Web scraping.

Ethical Principles around Data

Ownership

- The right to control your own data, possibly via surrogates

Consent

- You should understand how your data will be used, and explicitly approve its use

Privacy

- As your data is used, all reasonable efforts should be made to protect privacy.

Openness

- Data should be freely available and have no restrictions

Privacy is not simple...

- Recall that we gave the example of researchers who publicly released data about ~70,000 OKCupid users
 - Users had consented to their data to be used only by other logged in OKCupid users
 - The researchers had not attempted to anonymize the data
- Would anonymizing (de-identifying) the dataset been enough to obfuscate the identity of the OKCupid users?
 - Masking, generalizing, or deleting both direct and indirect identifiers

De-identification is not enough

Netflix Prize Competition

- In 2006, Netflix released a de-identified data set in an open competition for the best collaborative filtering algorithm to predict user ratings for films
 - Contained information <user, movie, date_of_grade, grade>
 - Users and movies were identified by numbers assigned for the contest
- In 2010, the competition was cancelled due to privacy concerns □ **Data re-identification**
 - Researchers at UT Austin were able to link users with film ratings on the IMDB's system, where the users were identified.

De-identification is not enough

Massachusetts re-identification incident

- In the mid 1990's Massachusetts Group Insurance Commissions (GIC) identified health records.
 - Identifiers such as names and SSN were removed, however zip code and sex were not.
- Latanya Sweeney combined the GIC data with the voter database of Cambridge, MA discovered the identity of then-Governor William Weld and located his health record.



Correlating data

- Even if data is de-identified, entries can be correlated (i.e. linked) with entries in other datasets to make informed guesses as to identity



Correlating data

- Even if data is de-identified, entries can be correlated (i.e. linked) with entries in other datasets to make informed guesses as to identity
- **Problem: “Sparsity” of data**
 - In Netflix data, no two profiles are more than 50% similar.
 - If a Netflix profile is more than 50% similar to a profile in IMDB, then there is a high probability that the two profiles are of the same person

87% of the U.S. population can be identified using a combination of their gender, birthdate and zip code.

A. Narayanan and V. Shmatikov, “Robust de-anonymization of large sparse datasets ...,”
Proc. 29th IEEE Symp. Security and Privacy, 2008.

<https://tinyurl.com/cis545-lecture-02-28-22>

Individual versus statistical information

- When do you feel safe releasing personal information, e.g. completing a survey about your tastes in movies?



Individual versus statistical information

- When do you feel safe releasing personal information, e.g. completing a survey about your tastes in movies?
 - My answers have no impact on the privatized released result?
 - With high probability, an attacker looking at the privatized released result cannot learn any new information about me?
 - **These are not achievable.**

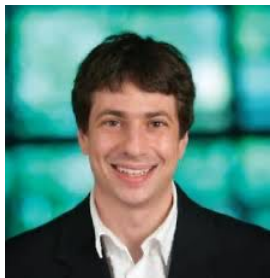
Differential Privacy

- **Differential privacy** aims to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records – it adds noise and provides guarantees against a “privacy budget”.

Harm to individual



Benefit to society



Differential Privacy



- **Differential privacy** aims to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records – it adds noise and provides guarantees against a “privacy budget”.
- “Algorithmic Foundations of Differential Privacy,” Foundations and Trends in Theoretical Computer Science (2014).
 - Penn CIS Professor Aaron Roth, and Turing Award winner Cynthia Dwork (Harvard University)

Differential Privacy

- **Differential privacy** aims to maximize the accuracy of queries from statistical databases while minimizing the chances of identifying its records – it adds noise and provides guarantees against a “privacy budget”.
- A [very accessible video](#) on the topic by Cynthia Dwork is linked to the course webpage.



<https://tinyurl.com/cis545-lecture-02-28-22>

Summary

- De-identification is not enough to ensure the privacy of individuals
- Only providing statistical summaries does not guarantee that no information will not be leaked about individuals

Harm to individual



Benefit to society

PageRank and Matrices in Spark

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



ODPi

OpenDS4All

*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-28-22>

From Data Representation and Summarization, to Modeling and Prediction

We've spent half the semester looking at how to get data into the right form for reasoning!

- Start with many sources, pull and wrangle, integrate and link, clean
- Dataframes + relations

We'll look more at how to get data into matrix form for machine learning analysis...

Recall Linear Algebra PageRank

- Create an $m \times m$ “weight transfer matrix” M to capture links:
 - $M(i, j) = 1 / n_j$ if page i is **pointed to** by page j
 and page j has n_j **outgoing links**
 $= 0$ otherwise
- Initialize all PageRanks to 1, multiply by M repeatedly until all values converge:

$$\begin{bmatrix} PageRank(p_1') \\ PageRank(p_2') \\ \dots \\ PageRank(p_m') \end{bmatrix} = M \begin{bmatrix} PageRank(p_1) \\ PageRank(p_2) \\ \dots \\ PageRank(p_m) \end{bmatrix}$$

Iterative computation:

(version w/o decay factor)

How Do We Make this Work in Spark?

- MLLib – Apache Spark's distributed ML libraries
 - Also include Matrix types, linear algebra, etc.
- Count $n = \# \text{ nodes}$
- Create a PageRank vector PR, with each node ID set to $1 / n$ for weight
- Create a weight transfer matrix M that's $n \times n$
- Do PageRank by iteration:
 - $PR = M \times PR$

<https://tinyurl.com/cis545-lecture-02-28-22>

Matrices in Spark

- Spark needs to **shard** matrices, just like tables
- Three basic ways of distributing:
 - Shard rows across machines, in no order (useful for machine learning training, we'll discuss soon)
 - Shard rows across machines, in order of index [IndexedRowMatrix]
 - Shard (row,col) -> value across machines [CoordinateMatrix]
- But all of these are terrible for matrix multiplication!
 - **BlockMatrix** splits the matrix into “tiles” so they can be coordinated
 - If we do a BlockMatrix multiply with another BlockMatrix, this will cause repartitioning so rows can be multiplied with columns

How Does this Look?

```
initial_graph = pd.DataFrame([{'from': 0, 'to': 1},\
                               {'from': 0, 'to': 2},\
                               {'from': 1, 'to': 2},\
                               {'from': 2, 'to': 0}])

sdf = spark.createDataFrame(initial_graph_dataframe)
sdf.createOrReplaceTempView( 'graph' )
transfer_weights = spark.sql( '''select graph.from as id,
                                1 / count("to") as weight
                                from graph group by from''' )

weighted_edges = sdf.join(transfer_weights, sdf[ 'from' ]
                          == transfer_weights.id)[['from', 'to', 'weight']]

# Build matrix M from the weighted edges
M = CoordinateMatrix(weighted_edges.rdd.map(
    lambda x: MatrixEntry(x[1], x[0], x[2]))).toBlockMatrix()
```

How Does this Look?

```
# Now figure out how many nodes, for initial PR
n = len(initial_graph[['from']].rename(columns={'from': 'id'}). \
      append(initial_graph[['to']].rename(columns={'to': 'id'})). \
      drop_duplicates() ['id'])

PR_sdf = sdf.select(sdf['from'], lit(1 / n).alias('weight'))
PR = CoordinateMatrix(PR_sdf.rdd.map(\
    lambda x: MatrixEntry(x[0], 0, x[1]))).toBlockMatrix()

# Run PageRank!
for i in range(15):
    PR = M.multiply(PR)
```

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771520> (12B)

With Spark DataFrames, we shard by:

- a. the index of a column
- b. the index of a row
- c. the values of a given join or grouping key
- d. the coordinates of a matrix

With Spark Matrices, we need to shard by:

- e. the values in a given column
- f. the coordinates or tiles in a matrix
- g. the join key
- h. the values in a given row

<https://tinyurl.com/cis545-lecture-02-28-22>

Matrices and Spark Recap

- Where possible, you should prefer local numpy matrices (which we'll discuss more) – they can use multiple CPU cores, vector instructions, etc.
- Spark MLLib matrices can be useful for very large data, but they are much slower
 - Operations such as multiply **only** don't work with all of the Matrix types... Only **BlockMatrix**

From DataFrames to Arrays for Machine Learning

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



OpenDS4All

*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-28-22>

DataFrames Are Human-Readable Data

- Our focus to this point has been on data that is human-understandable and machine-processable:
 - DataFrames, relations, JSON, etc.
- Machine learning algorithms generally require *numeric* data – we'll have to convert from DataFrames into this!


Matrices/2D Arrays for Machine Learning

Machine learning typically works best using *matrices*

Map from a row in the dataframe \square a row in a numeric matrix

customer	phone_color	purchase_amt
1	blue	\$30.90
2	black	\$55.22

Matrices have one type of data; change representations, e.g.:



1	2	31
2	1	56

Matrices in Python

Most commonly: **numpy arrays**, also MLLib **RowMatrix**

Recall that arrays have **all elements of the same type**, so must be floating-point- or integer-valued

```
# Basics of arrays: Numpy
import numpy as np

# We start with a simple array, initialized with random values

arr = np.ndarray((4,2))

# Show dimensions
arr.shape
```

Matrices in Python

Most commonly: **numpy arrays**, also MLLib **RowMatrix**

Recall that arrays have **all elements of the same type**, so must be floating-point- or integer-valued

```
# Basics of arrays: Numpy
import numpy as np
```

```
# We start with a simple array
# initialized with random values
```

```
arr = np.ndarray((4,2))
```

```
# Show dimensions
arr.shape
```

```
(4, 2)
```

```
[ ] arr
```

```
array([[1.58287250e-316, 1.58101007e-322],
       [2.95627109e-287, 1.54800841e+185],
       [4.55982032e-037, 6.07582975e-067],
       [9.74635144e-072, 5.83339361e-302]])
```

Matrices in Python

Most commonly: **numpy arrays**, also MLLib **RowMatrix**

Recall that arrays have **all elements of the same type**, so must be floating-point- or integer-valued

```
# Basics of arrays: Numpy
import numpy as np

# We start with a simple array
# initialized with random values

arr = np.ndarray((4,2))

# Show dimensions
arr.shape
```

```
(4, 2)

[ ] arr

array([[1.58287250e-316, 1.58101007e-322],
       [2.95627109e-287, 1.54800841e+185],
       [4.55982032e-037, 6.07582975e-067],
       [9.74635144e-072, 5.83339361e-302]])
```

```
arr = np.zeros((4,2))
arr = np.ones((4,2))
```

Matrices as Inputs to Machine Learning

- Each row represents an *instance* or *observation*
- Each column represents a *feature*

instance (row, sample, observation) {

x1	x2	x3	x4	x5	x6

extracted *feature*

Again, all data is integer or float!

- Requires us to think about *feature extraction*!

From DataFrames to Features:

Some Simple Ideas

Replace **string** or **categorical** data with integer codes

e.g., for US states: {0: 'Alabama', 1: 'Alaska', 2: 'Arizona', ..., 49: 'Wyoming'}

One-hot encoding: turn a set of potential values into a set of disjoint features
– in a bit vector

Alabama	Alaska	...	Wyoming
0	0	...	1
1	0	...	0

One-Hot Encoding in Pandas

```
addresses_df = \
pd.DataFrame([{'city': 'New York', \
'city': 'Los Angeles', 'state': 'CA'}, \
{'city': 'Chicago', 'state': 'IL'}, \
{'city': 'Houston', 'state': 'TX'}, \
{'city': 'Phoenix', 'state': 'AZ'}, \
{'city': 'Philadelphia', 'state': 'PA'}, \
{'city': 'San Antonio', 'state': 'TX'}])

pd.get_dummies(addresses_df['state'])

pd.get_dummies(addresses_df['state']).\
to_numpy()
```

	city	state
0	New York	NY
1	Los Angeles	CA
2	Chicago	IL
3	Houston	TX
4	Phoenix	AZ
5	Philadelphia	PA
6	San Antonio	TX

	AZ	CA	IL	NY	PA	TX
0	0	0	0	1	0	0
1	0	1	0	0	0	0
2	0	0	1	0	0	0
3	0	0	0	0	0	1
4	1	0	0	0	0	0
5	0	0	0	0	1	0
6	0	0	0	0	0	1

Sometimes We Want Only a Subset of Features

– *Slicing* a Matrix

NumPy lets you specify, for each dimension, a range of values to “slice” out...

Negative numbers mean “dimension size minus the number”

```
# In each dimension you can give a left:right  
# range. Ranges are left-inclusive and right exclusive  
arr[1:3,1]
```

```
X = input[:,0:-1]
```

Where this Is Frequently Useful in Machine Learning

We may have data in which the last column is the *label* for whether the data belongs to a *class*

```
array([[0, 0, 0, 1, 0, 0, 0],  
       [0, 1, 0, 0, 0, 0, 1],  
       [0, 0, 1, 0, 0, 0, 0],  
       [0, 0, 0, 0, 0, 1, 0],  
       [1, 0, 0, 0, 0, 0, 1],  
       [0, 0, 0, 0, 1, 0, 1],  
       [0, 0, 0, 0, 0, 1, 0]],  
      dtype=object)
```

We may split this into the matrix X (features) and vector y (class labels)

Where this Is Frequently Useful in Machine Learning

We may have data in which the last column indicates whether the data belongs to a certain class

```
array([[0, 0, 0, 1, 0, 0, False],  
       [0, 1, 0, 0, 0, 0, True],  
       [0, 0, 1, 0, 0, 0, False],  
       [0, 0, 0, 0, 0, 1, False],  
       [1, 0, 0, 0, 0, 0, True],  
       [0, 0, 0, 0, 1, 0, True],  
       [0, 0, 0, 0, 0, 1, False]],  
      dtype=object)
```

We may split this into two parts:
(features)
(class labels)

```
y (labels):  
[False True False False True True False]  
X (training data):  
[[0 0 0 1 0 0]  
 [0 1 0 0 0 0]  
 [0 0 1 0 0 0]  
 [0 0 0 0 0 1]  
 [1 0 0 0 0 0]  
 [0 0 0 0 1 0]  
 [0 0 0 0 0 1]]
```

```
# All rows, last column  
y = data[:, -1].apply(lambda x: x != 0)  
# All rows, all but last column  
X = data[:, 0:-1]
```

Sparse vs Dense Matrices

- A sparse matrix representation only tells you about the non-zero values.
- A dense matrix representation tells you about all values, zero or not.

Example of text data: Titles of Some Technical Memos

- c1: *Human machine interface for ABC computer applications*
- c2: *A survey of user opinion of computer system response time*
- c3: *The EPS user interface management system*
- c4: *System and human system engineering testing of EPS*
- c5: *Relation of user perceived response time to error measurement*

- m1: *The generation of random, binary, ordered trees*
- m2: *The intersection graph of paths in trees*
- m3: *Graph minors IV: Widths of trees and well-quasi-ordering*
- m4: *Graph minors: A survey*

A sparse matrix is usually implemented as a *dictionary* or equivalent!

Summary: The Basics of Data Representation in Arrays / Matrices

- Machine learning algorithms typically take an input 2D matrix
 - Rows = *instances* or samples
 - Columns = *features*
- We may need to *extract* or *encode* values from a DataFrame
 - Encodings for non-numeric data
 - One-hot encodings
- We can use *slicing* to restrict the set of instances or features we consider

Notation: Often, the matrix for machine learning is simply referred to as the **X** matrix!

We'll come back to these shortly – and use them often!!!

<https://tinyurl.com/cis545-lecture-02-28-22>

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771553> (12C)

We generally distinguish between a 2D array and a matrix in Numpy based

- a. on the operations we apply to the array
- b. the index coordinates
- c. whether there are an equal number of rows and columns
- d. whether we initialize using `np.array` or `np.matrix`

One-hot encoding refers to:

- e. a scheme in which each categorical attribute is represented by an int value in a column
- f. a scheme for encoding everything in 1's and 0's
- g. a scheme for encoding categorical attributes as strings
- h. a scheme in which each categorical attribute gets its own Boolean column

<https://tinyurl.com/cis545-lecture-02-28-22>

Operating on Matrices

Susan B. Davidson and Zachary G. Ives

University of Pennsylvania

CIS 545 – Big Data Analytics



ODPi

OpenDS4All

*Portions of this lecture have been contributed to the OpenDS4All project,
piloted by Penn, IBM, and the Linux Foundation*

<https://tinyurl.com/cis545-lecture-02-28-22>

Matrices Have “Bulk Operations” too

- Addition and multiplication with scalars
- Simple arithmetic over arrays / matrices (add, multiply)
- Linear algebra operations – multiply, transpose, determinant, norm
- “Slicing”
- Many of these are key building blocks for scientific computing
- Many can be parallelized via GPUs, vector instructions, etc. – which is why **numpy** can be fast!

<https://tinyurl.com/cis545-lecture-02-28-22>

An Example Application: Image Processing

- How does an image get represented?



```
face.shape
```

```
(1363, 2048, 3)
```

```
array([[ [ 45, 76, 35],  
        [ 64, 98, 47],  
        [ 63, 101, 44],  
        ...,  
        [ 39, 62, 20],  
        [ 50, 68, 28],  
        [ 45, 68, 22]],  
       [[ [ 56, 89, 44],  
        [ 67, 101, 50],  
        [ 69, 104, 48],  
        ...,  
        [ 53, 82, 38],  
        [ 51, 90, 33],  
        [ 61, 97, 33]])]
```

Array “Slicing” – Projection or Cropping

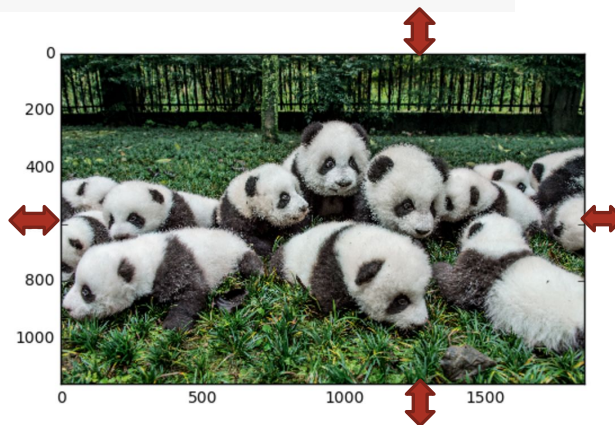
```
# Positive integer is relative to left (0)
# Negative integer is relative to right/bottom (end)

crop_face = face[100:-100, 100:-100]

# Each value is in fact an RGB triple in a "color space"
crop_face
```

```
array([[ [ 28,  67,  22],
        [ 39,  72,  29],
        [ 27,  59,  18],
        ...,
        [  0,   5,   1],
        [  0,   2,   0],
        [  3,   5,   2]],

       [[ [ 47,  79,  29],
        [ 40,  76,  28],
        [ 23,  64,  22],
        ...,
```



Beware: Numpy Slices Are NOT Copies

- By default, a “slice” points to the data in the original array!
- More efficient, but it means you need to be careful about changing the slice
 - i.e., if I change values in `crop_face`, I also change `face`
- If you're going to modify the values, call
`array.copy()`

<https://tinyurl.com/cis545-lecture-02-28-22>

Selecting Parts of an Array in Numpy

<https://tinyurl.com/cis545-lecture-02-28-22>

Row Selection in an Array

(And Populating a Random Array)

```
colors = np.array(['Green', 'Blue', 'Green', 'Yellow', 'Red'])
```

```
data = np.random.randn(5,3)
```

```
colors
```

```
array(['Green', 'Blue', 'Green', 'Yellow', 'Red'],  
      dtype='<U6')
```

```
data
```

```
array([[ 3.10669249, -0.29687844,  0.52947644],  
       [ 0.48135368,  0.76115606, -0.60468068],  
       [ 0.48787342,  0.35323244,  0.89169349],  
       [ 0.2318091 , -0.73546908, -0.2662448 ],  
       [ 0.09345196,  0.92481036,  0.53960019]])
```

```
colors
```

```
array(['Green', 'Blue', 'Green', 'Yellow', 'Red'],  
      dtype='<U6')
```

```
data
```

```
array([[ 3.10669249, -0.29687844,  0.52947644],  
       [ 0.48135368,  0.76115606, -0.60468068],  
       [ 0.48787342,  0.35323244,  0.89169349],  
       [ 0.2318091 , -0.73546908, -0.2662448 ],  
       [ 0.09345196,  0.92481036,  0.53960019]])
```

```
colors == 'Green'
```

```
array([ True, False,  True, False, False], dtype=bool)
```

```
data[colors=='Green']
```

```
array([[ 3.10669249, -0.29687844,  0.52947644],  
       [ 0.48787342,  0.35323244,  0.89169349]])
```

“Fancy Indexing” (Selecting by Index)

Choose a list of array components in a particular order...

```
arr = np.empty((8,4))  
  
for i in range(8):  
    arr[i] = i  
  
arr  
  
array([[ 0.,  0.,  0.,  0.],  
       [ 1.,  1.,  1.,  1.],  
       [ 2.,  2.,  2.,  2.],  
       [ 3.,  3.,  3.,  3.],  
       [ 4.,  4.,  4.,  4.],  
       [ 5.,  5.,  5.,  5.],  
       [ 6.,  6.,  6.,  6.],  
       [ 7.,  7.,  7.,  7.]])
```

```
arr[[7,2,4,1,4]]
```

```
array([[ 7.,  7.,  7.,  7.],  
       [ 2.,  2.,  2.,  2.],  
       [ 4.,  4.,  4.,  4.],  
       [ 1.,  1.,  1.,  1.],  
       [ 4.,  4.,  4.,  4.]])
```

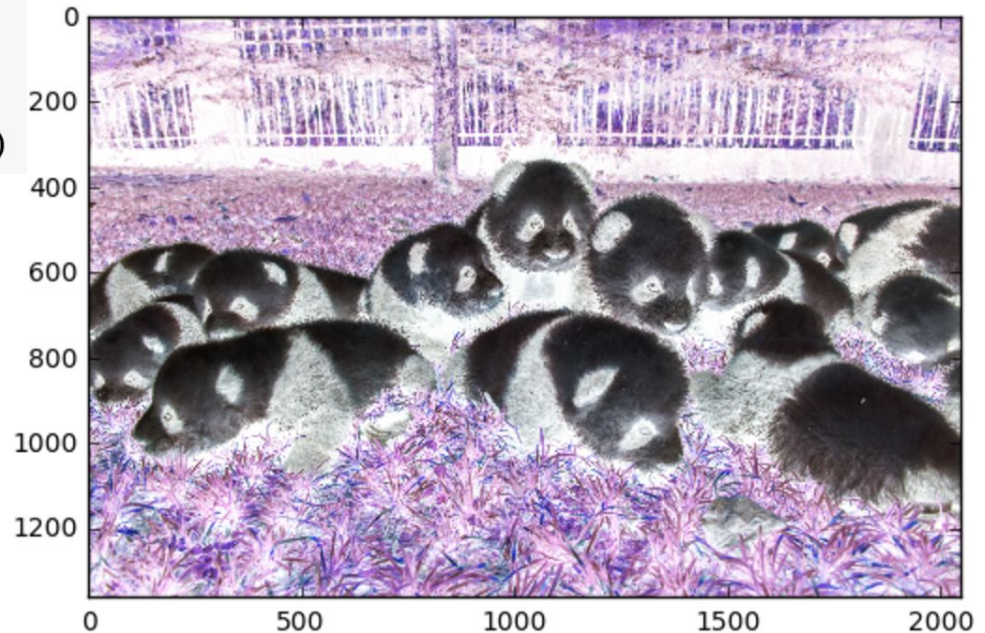
```
arr[[-1,-5,-2]]
```

```
array([[ 7.,  7.,  7.,  7.],  
       [ 3.,  3.,  3.,  3.],  
       [ 6.,  6.,  6.,  6.]])
```

Arithmetic on a Numpy Array

Exploits All RGB Values 0 \square 255

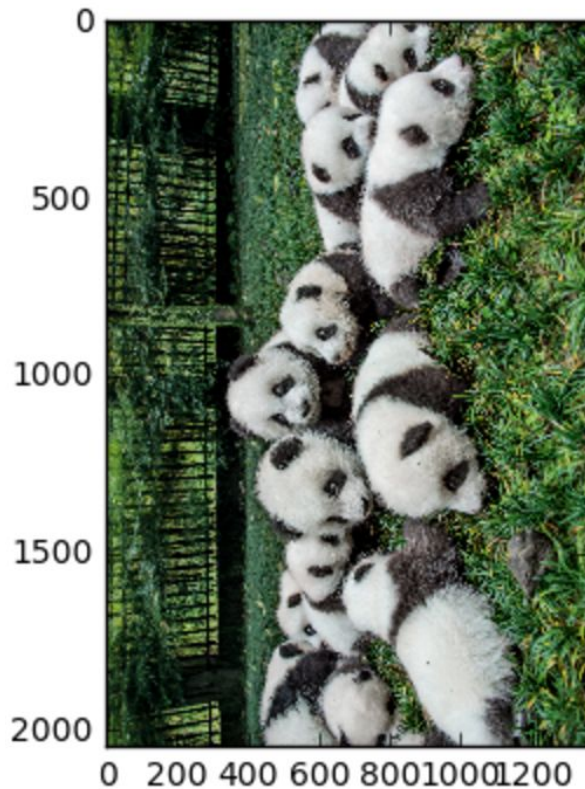
```
new_face = face.copy()  
inv_face = 255 - new_face  
plt.imshow(inv_face, cmap=plt.cm.gray)
```



<https://tinyurl.com/cis545-lecture-02-28-22>

Transposition – Swapping Dimensions

```
rotate_face = face.transpose(1,0,2)  
plt.imshow(rotate_face)
```



<https://tinyurl.com/cis545-lecture-02-28-22>

Common Operations:

Unary and Binary Functions on Elements

- abs
- sqrt
- square
- exp
- log
- ceil
- floor
- add
- subtract
- multiply
- power
- mod
- greater

• <https://tinyurl.com/cis545-lecture-02-28-22>

Linear Algebra

Numpy treats arrays as matrices and supports:

- `np.dot(A, B)` – matrix multiply, also in some versions of Python, `A @ B`
- Determinant (`det`), eigenvalues + eigenvectors (`eig`)
- Singular value decomposition (`svd`)
- And many more!

```
arr
```

```
array([[ 0.,  0.,  0.,  0.],  
       [ 1.,  1.,  1.,  1.],  
       [ 2.,  2.,  2.,  2.],  
       [ 3.,  3.,  3.,  3.],  
       [ 4.,  4.,  4.,  4.],  
       [ 5.,  5.,  5.,  5.],  
       [ 6.,  6.,  6.,  6.],  
       [ 7.,  7.,  7.,  7.]])
```

```
np.dot(arr, arr.T)
```

```
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],  
       [ 0.,  4.,  8., 12., 16., 20., 24., 28.],  
       [ 0.,  8., 16., 24., 32., 40., 48., 56.],  
       [ 0., 12., 24., 36., 48., 60., 72., 84.],  
       [ 0., 16., 32., 48., 64., 80., 96., 112.],  
       [ 0., 20., 40., 60., 80., 100., 120., 140.],  
       [ 0., 24., 48., 72., 96., 120., 144., 168.],  
       [ 0., 28., 56., 84., 112., 140., 168., 196.]])
```

Aggregation over Arrays

- Standard “averaging” measures – `arr.mean()`, `median()`, `mode()`
- Distributional info – `std()` (standard deviation) and `var()` (variance)
- `min()`, `max()`
- `argmin()`, `argmax()`

<https://tinyurl.com/cis545-lecture-02-28-22>

Randomly populating values

- Create array with random values from distributions:
 - `np.random.randn()` – normal distribution with mean 0, stdev 1
 - binomial
 - normal / Gaussian
 - beta
 - chisquare
 - gamma

<https://tinyurl.com/cis545-lecture-02-28-22>

Linear Algebra

Numpy treats arrays as matrices and supports:

- `np.dot(A, B)` – matrix multiply, also in some versions of Python, `A @ B`
- Determinant (`det`), eigenvalues + eigenvectors (`eig`)
- Singular value decomposition (`svd`)
- And many more!

```
arr
array([[ 0.,  0.,  0.,  0.],
       [ 1.,  1.,  1.,  1.],
       [ 2.,  2.,  2.,  2.],
       [ 3.,  3.,  3.,  3.],
       [ 4.,  4.,  4.,  4.],
       [ 5.,  5.,  5.,  5.],
       [ 6.,  6.,  6.,  6.],
       [ 7.,  7.,  7.,  7.]])
```

```
np.dot(arr, arr.T)
array([[ 0.,  0.,  0.,  0.,  0.,  0.,  0.,  0.],
       [ 0.,  4.,  8., 12., 16., 20., 24., 28.],
       [ 0.,  8., 16., 24., 32., 40., 48., 56.],
       [ 0., 12., 24., 36., 48., 60., 72., 84.],
       [ 0., 16., 32., 48., 64., 80., 96., 112.],
       [ 0., 20., 40., 60., 80., 100., 120., 140.],
       [ 0., 24., 48., 72., 96., 120., 144., 168.],
       [ 0., 28., 56., 84., 112., 140., 168., 196.]])
```

Brief Review

<https://canvas.upenn.edu/courses/1636888/quizzes/2771552> (12D)

Matrix bulk operations include:

- a. extract primes
- b. join, group-by, select
- c. multiply, add, transpose, determinant, slice
- d. crop, invert

"Fancy indexing" refers to

- e. creating a dataframe index
- f. extracting rows from an array given a list of indexing
- g. generating a B+ Tree index
- h. transposing rows and columns in an array

<https://tinyurl.com/cis545-lecture-02-28-22>

Recap and Take-aways: Arrays

- NumPy (and SciPy) provide powerful support for arrays and matrices
- Bulk operators across the various elements, different kinds of iteration, ...
- For some kinds of array computations, there is a natural mapping to Spark-style distributed computation
 - ... but for others we don't get much speedup
- Thus, we can do general-purpose matrix computation on a single machine, and specialized matrix computation in Spark
 - Sometimes it makes as much sense to encode the matrix data as a DataFrame as a Spark matrix...