Python Data Products
Course 1: Basics

Lecture: Matrix processing and numpy
Learning objectives

In this lecture we will...

- Briefly introduce the numpy library
- Perform simple matrix operations on datasets
Loading data

In[1]:
```python
import numpy
import json
```

In[2]:
```python
path = "datasets/yelp_data/review.json"
f = open(path)
```

In[3]:
```python
dataset = []
```

In[4]:
```python
while len(dataset) < 50000:
    dataset.append(json.loads(f.readline()))
```

In[5]:
```python
dataset[0]
```

Out[5]:
```json
{"business_id": "0W4lkclzzZThpx3V65bVgig",
 "cool": 0,
 "date": "2016-05-28",
 "funny": 0,
 "review_id": "v01_UHJMo_hPBq9bxWvwW4w",
 "stars": 5,
 "text": "Love the staff, love the meat, love the place. Prepare for a long line around lunch or dinner hours. \n\nThey ask you how you want you meat, lean or something maybe, I can't remember. Just say you don't want it too fatty. \n\nGet a half sour pickle and a hot pepper. Hand cut french fries too.",
 "useful": 0,
 "user_id": "bv2nC15Qv5vroFiqKGopiW"}
```
• First let's extract a few simple numerical features from the dataset:

```
In [6]: ratings = [d['stars'] for d in dataset]
In [7]: cool = [d['cool'] for d in dataset]
In [8]: funny = [d['funny'] for d in dataset]
In [9]: useful = [d['useful'] for d in dataset]
```

• These are just lists, but we can convert them to numpy arrays:

```
In [10]: ratings = numpy.array(ratings)
   cool = numpy.array(cool)
   funny = numpy.array(funny)
   useful = numpy.array(useful)

In [11]: ratings
```

```
Out[11]: array([5, 5, 5, ..., 5, 5, 5])
```
For the most part, numpy arrays can be treated much like regular python arrays, though they support a variety of additional operations, such as statistical operations:

```python
In [12]: numpy.mean(ratings)
Out[12]: 3.73154
```

```python
In [13]: numpy.var(ratings)
Out[13]: 1.9213092284000002
```
• We can also compose vectors to build ND-arrays, e.g.:

```python
In [14]: numpy.stack([[cool, funny, useful]])
Out[14]: array([[0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0],
             [0, 0, 0, ..., 0, 0, 0]])
```

• Once we have an array, we can perform other matrix operations like computing the transpose, e.g. to get a feature matrix ($X$) we might do the following:

```python
In [15]: features = numpy.stack([[cool, funny, useful]]).T
```
• Note that with an array, most operations (in particular multiplication) are overloaded to "elementwise" operations. For many linear algebra routines, it is more convenient to use the "matrix" type instead:

In [16]: features = numpy.matrix(features)

• This supports operations like standard matrix multiplication:

In [17]: features.T * features

Out[17]: matrix([[328903, 219632, 381580],
                 [219632, 213077, 288115],
                 [381580, 288115, 734845]])

• Or matrix inverse, etc.

In [18]: numpy.linalg.inv(features.T * features)

Out[18]: matrix([[ 1.2223023e-05, -8.55225321e-06, -2.99608975e-06],
                  [-8.55225321e-06,  1.59703874e-05,  1.82070967e-06],
                  [-2.99608975e-06, -1.82070967e-06,  3.63045498e-06]])
Finally, numpy overloads primitive operations on matrices, allowing matrices to be used within complex mathematical expressions, in order to perform transformations of our data:

```python
In [19]: 2*numpy.sin(features) + 3
Out[19]: matrix([[3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.],
              [3., 3., 3.]])
```

```python
In [20]: 2*numpy.sin(features) + 3 > 4
Out[20]: matrix([[False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False],
               [False, False, False]])
```
Other features...

- `ndarray.shape`: Get the shape of an array
- `reshape`: change the dimensions of an array/matrix
- `arange`: Create an array containing a range of numbers
- `numpy.random`: generate (arrays of) random numbers
- `sum, min, max, etc.`: reduction operations on matrices
- `eye`: identity matrix
- `trace, eig, etc.`: linear algebra operations
- See [https://docs.scipy.org/doc/numpy/user/quickstart.html](https://docs.scipy.org/doc/numpy/user/quickstart.html) for more
Summary of concepts

- Introduced the numpy library
- Demonstrated basic operations for data manipulation in numpy

On your own...

- Try reading the numerical features from the Amazon data into a numpy array, and compiling basic statistics about them (max, min, avg values, etc.)