Web Mining and Recommender Systems

Tools and techniques for data processing and visualization
Some helpful ideas for Assignment 2...

1. How can we **crawl our own datasets** from the web?
2. How can we **process those datasets** into structured objects?
3. How can we **visualize and plot** data that we have collected?
4. What libraries can help us to **fit complex models** to those datasets?
Some helpful ideas for Assignment 2...

1. How can we **crawl our own datasets** from the web? → Python requests library + BeautifulSoup
2. How can we **process those datasets** into structured objects? → A few library functions to deal with time+date
3. How can we **visualize and plot** data that we have collected? → Matplotlib
4. What libraries can help us to **fit complex models** to those datasets? → Tensorflow
Web Mining and Recommender Systems

Collecting and parsing Web data with urllib and BeautifulSoup
Collecting our own datasets

Suppose that we wanted to collect data from a website, but didn't yet have CSV or JSON formatted data

- How could we collect new datasets in machine-readable format?
- What Python libraries could we use to collect data from webpages?
- Once we'd collected (e.g.) raw html data, how could we extract structured information from it?
Collecting our own datasets

E.g. suppose we wanted to collect reviews of "The Great Gatsby" from goodreads.com:
Collecting our own datasets

How could we extract fields including:

- The *ID* of the user,
- The *date* of the review
- The *star rating*
- The *text* of the review itself?
- The *shelves* the book belongs to

Oh Gatsby, you old sport, you poor semi-delusionally hopeful dreamer with 'some heightened sensitivity to the promises of life', focusing your whole self and soul on that elusive money-colored green light – a dream that shatters just when you are "this" close to it.

Jay Gatsby, who dreamed a dream with the passion and courage few possess – and the tragedy was that it was a wrong dream colliding with reality that was even more wrong – and deadly.

Just like the Great Houdini – the association the...
Our first step is to extract the html code of the webpage into a python string. This can be done using `urllib`:

```python
from urllib.request import urlopen
html = str(f.read())
```
This isn't very nice to look at, it can be easier to read in a browser or a text editor (which preserves formatting):
To extract review data, we'll need to look for the part of the html code which contains the reviews:

Here it is (over 1000 lines into the page!)
Reading the html data

To extract review data, we'll need to look for the part of the html code which contains the reviews:

• Note that each individual review starts with a block containing the text "<div id="review_..."

• We can collect all reviews by looking for instances of this text
To split the page into individual reviews, we can use the `string.split()` operator. Recall that we saw this earlier when reading csv files:

```python
In [5]: reviews = html.split('<div id="review_')[1:]
In [6]: len(reviews)
Out[6]: 30
In [7]: reviews[0]
```

**Note:** Ignore the first block, which contains everything before the first review.

**Note:** the page contains 30 reviews total.
Next we have to write a method to parse individual reviews (i.e., given the text of one review, extract formatted fields into a dictionary)

```python
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class="staticStars" title="">
    d['date'] = review.split('<a class="reviewDate">
    d['user'] = review.split('审"
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">
        for s in shelfBlock.split('shelf='):
            shelves.append(s.split('
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">
    d['reviewBlock'] = reviewBlock
    return d
```
Let's look at it line-by-line:

- We start by building an empty dictionary
- We'll use this to build a *structured* version of the review
Let's look at it line-by-line:

- The next line is more complex:

  ```
  d['stars'] = review.split('<span class="staticStars" title=""')[1].split('"')[0]
  ```

- We made this line by noticing that the stars appear in the html inside a span with class "staticStars":

  ```
  rated it
  <span class="staticStars" title="it was amazing">•<span size="15x15" class="staticStar p10">it was amazing p10</span>•</span><span size="15x15" class="staticStar p10"></span></span>
  ```

- Our "split" command then extracts everything inside the "title" quotes

Note: Two splits: everything after the first quote, and before the second quote
Let's look at it line-by-line:

- The following two lines operate in the same way:
  ```javascript
  d['date'] = review.split('<a class="reviewDate">')[1].split('>')[0].split('<')[0]
  d['user'] = review.split('<a title="'>')[1].split('"'>')[0]
  ```
  **Note:** Everything between the two brackets of this "<a" element

- Again we did this by noting that the "date" and "user" fields appear inside certain html elements:
  ```html
  <div class="left bodycol">
    <div class="reviewHeader uixtext stacked">
      <a class="reviewDate createdAt right" href="/review/show/101057684?book_show_action=true">May 02, 2010</a>
      <span itemprop="author" itemscope itemtype="http://schema.org/Person">
        <a title="Nataliya" class="user" itemprop="url" name="Nataliya" href="/user/show/3672777-nataliya">Nataliya</a>
      </span>
    </div>
  </div>
  ```
Let's look at it line-by-line:

- Next we extract the "shelves" the book belongs to
- This follows the same idea, but in a "for" loop since there can be many shelves per book:

```python
shelves = []
try:
    shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div>')[0]
    for s in shelfBlock.split('shelf=')[1:]:
        shelves.append(s.split('"')[0])
    d['shelves'] = shelves
except Exception as e:
    pass
```

- Here we use a try/except block since this text will be missing for users who didn't add the book to any shelves
Next let’s extract the review contents:

```python
def parseReview(review):
    d = {}
    d['stars'] = review.split('<span class="staticStars" title=""></span>')[1].split('"')[0]
    d['date'] = review.split('<a class="reviewDate"></a>')[1].split('"')[0]
    d['user'] = review.split('<a title=""></a>')[1].split('"')[0]
    shelves = []
    try:
        shelfBlock = review.split('<div class="uitext greyText bookshelves">')[1].split('</div>')[0]
        for s in shelfBlock.split('shelf="')[1:]:
            shelves.append(s.split('"')[-1])
        d['shelves'] = shelves
    except Exception as e:
        pass
    reviewBlock = review.split('<div class="reviewText stacked">')[1].split('</div>')[0]
    d['reviewBlock'] = reviewBlock
    return d
```
Now let’s look at the results:

- Looks okay, but the review block itself still contains embedded html (e.g. images etc.)
- How can we extract just the text part of the review?
Extracting the text contents from the html review block would be extremely difficult, as we'd essentially have to write a html parser to capture all of the edge cases.

Instead, we can use an existing library to parse the html contents: BeautifulSoup
BeautifulSoup will build an element tree from the html passed to it. For the moment, we'll just use it to extract the text from a html block.

```
from bs4 import BeautifulSoup

soup = BeautifulSoup(reviewDict[0]["reviewBlock"])

soup.text
```

"""Oh Gatsby, you old sport, you poor semi-delusionally hopeful dreamer with \"some heightened sensitivity to the promises of life\", focusing your whole self and soul on that elusive money-colored green light - a dream that shatters just when you are *this* close to it. Jay Gatsby, who dreamed a dream with the passion and courage few possess - and the tragedy was that it was a wrong dream colliding with reality that was even more wrong - and deadly. Just like the Great Houdini - the association the"""
In principle we could have used BeautifulSoup to extract *all* of the elements from the webpage.

However, for simple page structures, navigating the html elements is not (necessarily) easier than using primitive string operations.
1. What if we have a webpage that loads content **dynamically**?

(e.g. https://www.amazon.com/gp/profile/amzn1.account.AHQSDGUKX6BESSVAOWMIAJKBOZPA/ref=cm_cr_dp_d_gw_tr?ie=UTF8)

- The page (probably) uses javascript to generate requests for new content
- By monitoring network traffic, perhaps we can view and reproduce those requests
- This can be done (e.g.) by using the Developer Tools in chrome
Pages that load dynamically...

Scroll to bottom...
Pages that load dynamically...

Look at requests that get generated
Pages that load dynamically...

Let's try to reproduce this request
Pages that load dynamically...
2. What if we require passwords, captchas, or cookies?

- You'll probably need to load an actual browser.
- This can be done using a headless browser, i.e., a browser that is controlled via Python.
  - I usually use splinter (https://splinter.readthedocs.io/en/latest/)
- Note that once you've entered the password, solved the captcha, or obtained the cookies, you can normally continue crawling using the requests library.
Summary

• Introduced programmatic approaches to collect datasets from the web
• The urllib library can be used to request data from the web as if it is a file, whereas BeautifulSoup can be used to convert the data to structured objects
  • Parsing can also be achieved using primitive string processing routines

• Make sure to check the page's terms of service first!
Web Mining and Recommender Systems

Parsing time and date data
Dealing with time and date data can be difficult as string-formatted data doesn't admit easy comparison or feature representation:

- Which date occurs first, 4/7/2003 or 3/8/2003?
- How many days between 4/5/2003 - 7/15/2018?
- e.g. how many hours between 2/6/2013 23:02:38 - 2/7/2013 08:32:35?
Most of the data we've seen so far include plain-text time data, that we need to carefully manipulate:

```json
{
    'business_id': 'FYWN1wneVl8bWNQjJ2GNg',
    'attributes':
        {'BusinessAcceptsCreditCards': True,
         'AcceptsInsurance': True,
         'ByAppointmentOnly': True},
    'longitude': -111.9785992,
    'state': 'AZ',
    'address': '4855 E Warner Rd, Ste B9',
    'neighborhood': '',
    'city': 'Ahwatukee',
    'hours': {
        'Tuesday': '7:30-17:00',
        'Wednesday': '7:30-17:00',
        'Thursday': '7:30-17:00',
        'Friday': '7:30-17:00',
        'Monday': '7:30-17:00'},
    'postal_code': '85044',
    'review_count': 22,
    'stars': 4.0,
    'categories': ['Dentists', 'General Dentistry', 'Health & Medical', 'Oral Surgeons', 'Cosmetic Dentists', 'Orthodontists'],
    'is_open': 1,
    'name': 'Dental by Design',
    'latitude': 33.3306902
}
```
Here we'll cover a few functions:

- `Time.strptime`: convert a time string to a structured time object
- `Time.strftime`: convert a time object to a string
- `Time.mktime / calendar.timegm`: convert a time object to a number
- `Time.gmtime`: convert a number to a time object
Here we'll cover a few functions:

- **Time string**
  - `strptime` 
  - `strftime`

- **Structured time object**
  - `time.struct_time` with:
    - `tm_year=2019`
    - `tm_mon=5`
    - `tm_mday=28`
    - `tm_hour=21`
    - `tm_min=36`
    - `tm_sec=18`
    - `tm_wday=1`
    - `tm_yday=148`
    - `tm_isdst=-1`

- **Number**
  - `mktime` 
  - `timegm`
  - `gmtime`
  - `1464418800.0`
Internally, time is often represented as a number, which allows for easy manipulation and arithmetic.

- The value (Unix time) is the **number of seconds since Jan 1, 1970 in the UTC timezone**
- so I made this slide at 1532568962 = 2018-07-26 01:36:02 UTC (or 18:36:02 in my timezone)
- But real datasets generally have time as a "human readable" string
- Our goal here is to convert between these two formats
First, let's look at converting a string to a structured object (strptime)

```
21:36:18, 28/5/2019
```

```
time.struct_time(tm_year=2019, tm_mon=5, tm_mday=28, tm_hour=21, tm_min=36, tm_sec=18, tm_wday=1, tm_yday=148, tm_isdst=-1)
```
**Code: time.strptime()**

```python
In [1]:
import time
import calendar

String-formatted time data

In [2]:
timeString = "2018-07-26 01:36:02"

In [3]:
timeStruct = time.strptime(timeString, "%Y-%m-%d %H:%M:%S")

In [4]:
timeStruct

Out[4]:
time.struct_time(tm_year=2018, tm_mon=7, tm_mday=26, tm_hour=1, tm_min=36, tm_sec=2, tm_wday=3, tm_yday=207, tm_isdst=-1)

In [5]:
timeStruct.tm_wday

Note: this day is a Wednesday!

Out[5]:
3

In [6]:
help(time.strptime)

Help on built-in function strptime in module time:

strptime(...)  
strptime(string, format) -> struct_time  

Parse a string to a time tuple according to a format specification.

Note: different time formatting options in the help page
Strptime is convenient when we want to extract **features** from data

- E.g. does a date correspond to a weekday or a weekend?
- Converting month names or abbreviations (e.g. "Jan") to month numbers
- Dealing with mixed-format data by converting it to a common format
- But if we want to perform arithmetic on timestamps, converting to a number may be easier
For this we'll use `mktime` to convert our structured time object to a number:

```python
time.struct_time(tm_year=2019, tm_mon=5, tm_mday=28, tm_hour=21, tm_min=36, tm_sec=18, tm_wday=1, tm_yday=148, tm_isdst=-1)
```

1464418800.0
Code: `time.mktime()` and `calendar.timegm()`

- `time.mktime()` allows us to convert our structured time object to a number
- **NOTE:** `mktime` assumes the structure is a *local* time whereas `timegm` assumes the structure is a *UTC* time
- This allows for easy manipulation, arithmetic, and comparison (e.g. sorting) of time data
Finally, both of these operations can be reversed, should we wish to format time data as a string or structure.
These methods can be used to put adjusted times back into string format.
Web Mining and Recommender Systems

Introduction to Matplotlib
Matplotlib is a powerful library that can be used to generate both quick visualizations, as well as publication-quality graphics.

- We'll introduce some of its most basic functionality (via pyplot), such as bar and line plots.
- Examples (with code) of the types of plots that can be generated are available on [https://matplotlib.org/](https://matplotlib.org/)
First, let's quickly compile some statistics from (e.g.) Yelp's review data

```python
In [1]:
import json
import time
path = "datasets/yelp_data/review.json"
f = open(path, 'r')

In [2]:
dataset = []
for i in range(50000):
    dataset.append(json.loads(f.readline()))

In [3]:
datasetWithTimeValues = []

In [4]:
for d in dataset:
    d['date']
    d['timeStruct'] = time.strptime(d['date'], "%Y-%m-%d")
    d['timeInt'] = time.mktime(d['timeStruct'])
    datasetWithTimeValues.append(d)
```
Code: generating some simple statistics

```python
from collections import defaultdict

weekRatings = defaultdict(list)

for d in datasetWithTimeValues:
    day = d['timeStruct'].tm_wday
    weekRatings[day].append(d['stars'])

weekAverages = {}

for d in weekRatings:
    weekAverages[d] = sum(weekRatings[d]) * 1.0 / len(weekRatings[d])

weekAverages
```

Average ratings per day of week

```
{0: 3.7094594594594597, 1: 3.715375187253166, 2: 3.750551876379691, 3: 3.763655361751486, 4: 3.7551891653172382, 5: 3.7231843981953134, 6: 3.7072147651099713}
```
Code: drawing a simple plot

In [11]: `X = list(weekAverages.keys())` → [0,1,2,3,4,5,6]

In [12]: `Y = [weekAverages[x] for x in X]`

In [13]: `import matplotlib.pyplot as plt`

In [14]: `plt.plot(X, Y)`

Out[14]: `[<matplotlib.lines.Line2D at 0x7fc15a615a20>]`
Code: bar plots

In [15]: plt.bar(X, Y)
Out[15]: <Container object of 7 artists>

- Looks right, but need to zoom in more to see the detail
• Next let's add some details
Code: bar plots

In [17]:
   plt.ylim(3.6, 3.8)
   plt.xlabel("Weekday")
   plt.ylabel("Rating")
   plt.xticks([0,1,2,3,4,5,6],["S", "M", "T", "W", "T", "F", "S"])
   plt.title("Rating as a function of weekday")
   plt.bar(X, Y)

Out[17]: <Container object of 7 artists>
Example: sliding windows

Also useful to plot data:

BeerAdvocate, ratings over time

Scatterplot

BeerAdvocate, ratings over time

Sliding window (K=10000)

long-term trends

seasonal effects

Code on:

http://jmcauley.ucsd.edu/code/week10.py
Web Mining and Recommender Systems

Gradient descent in tensorflow
Tensorflow, though often associated with deep learning, is really just a library that simplifies gradient descent and optimization problems, like those we've already implemented.

Most critically, it computes gradients symbolically, so that you can just specify the objective, and Tensorflow can run gradient descent.

Here we'll reimplement some of our previous gradient descent code in tensorflow.
Reading the data is much the same as before (except that we first import the tensorflow library)

In [1]: import tensorflow as tf

In [2]: path = "datasets/PRSA_data_2010.1.1-2014.12.31.csv"
f = open(path, 'r')

In [3]: dataset = []
header = f.readline().strip().split(',
for line in f:
    line = line.split(','
    dataset.append(line)

In [4]: header.index('pm2.5')
Out[4]: 5

In [5]: dataset = [d for d in dataset if d[5] != 'NA']
Next we extract features from the data

```python
In [6]: def feature(datum):
    feat = [1, float(datum[7]), float(datum[8]), float(datum[10])]  # Temperature, pressure, and wind speed
    return feat

In [7]: X = [feature(d) for d in dataset]
y = [float(d[5]) for d in dataset]

In [8]: y = tf.constant(y, shape=[len(y), 1])

In [9]: K = len(X[0])
```

Note that we convert $y$ to a native tensorflow vector. In particular we convert it to a column vector. We have to be careful about getting our matrix dimensions correct or we may (accidentally) apply the wrong matrix operations.
Next we write down the objective – note that we use native tensorflow operations to do so

```python
In [10]: def MSE(X, y, theta):
    return tf.reduce_mean((tf.matmul(X, theta) - y)**2)
```

Next we setup the variables we want to optimize – note that we explicitly indicate that these are **variables** to be optimized (rather than constants)

```python
In [11]: theta = tf.Variable(tf.constant([0.0]*K, shape=[K,1]))
In [12]: optimizer = tf.train.AdamOptimizer(0.01)
In [13]: objective = MSE(X,y,theta)
```

Specify the objective we want to optimize – note that no computation is performed (yet) when we run this function
Boilerplate for initializing the optimizer...

In [14]: `train = optimizer.minimize(objective)` ← We want to **minimize** the objective

In [15]: `init = tf.global_variables_initializer()`

In [16]: `sess = tf.Session()
sess.run(init)`
Run 1,000 iterations of gradient descent:

```python
for iteration in range(1000):
    cvalues = sess.run([train, objective])
    print("objective = "+str(cvalues[1]))
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5107
objective = 7836.5103
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
objective = 7836.5093
```

Code: Gradient Descent in Tensorflow
Code: Gradient Descent in Tensorflow

Print out the results:

```
In [18]: with sess.as_default():
    print(MSE(X, y, theta).eval())
    print(theta.eval())

7836.5093
[[ 0.23223479]
 [-0.89481604]
 [ 0.11925128]
 [-0.4959688 ]]
Summary

Note that in contrast to our "manual" implementation of gradient descent, many of the most difficult issues were taken care of for us:

• No need to compute the gradients – tensorflow does this for us!
• Easy to experiment with different models
• Very fast to run 1,000 iterations, especially with GPU acceleration!
Other libraries

Tensorflow is just one example of a library that can be used for this type of optimization. Alternatives include:

- Theano - [http://deeplearning.net/software/theano/](http://deeplearning.net/software/theano/)
  - Keras - [https://keras.io/](https://keras.io/)
  - Torch - [http://torch.ch/](http://torch.ch/)
  - Etc.

Each has fairly similar functionality, but some differences in interface
Questions?