Python Data Products

Course 1: Basics

Lecture: Data filtering and cleaning

Learning objectives

In this lecture we will...

- Introduce basic protocols for filtering datasets
- Provide Python code to extract subsets of our data

Why preprocessing?

Although the datasets we're working with have already been "cleaned" to some extent, in general there may be many reasons we'd want to further clean or pre-process datasets, e.g.:

- Certain fields may be missing from some entries
- Certain entries may be garbled / poorly formatted
- Parts of a dataset may be "stale" or otherwise unusable
- Parts of the dataset may contain statistical outliers (which we may or may not want to keep)
- We may want to remove data pertaining to rare or inactive users
- May want to restrict our dataset to a certain demographic or region
- Etc.

Data preprocessing in Python

In this lecture we'll look at a few such examples in Python

In [1]:
```
import gzip
path = "/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
f = gzip.open(path, 'rt')
```

In [2]:
```
import csv
reader = csv.reader(f, delimiter = '\t')
```

In [3]:
```
header = next(reader)
```

In [4]:
```
dataset = []
for line in reader:
    d = dict(zip(header, line))
    for field in ['helpful_votes', 'star_rating', 'total_votes']:
        d[field] = int(d[field])
    for field in ['verified_purchase', 'vine']:
        if d[field] == 'Y':
            d[field] = True
        else:
            d[field] = False
    dataset.append(d)
```

Data preprocessing in Python

Recall that our data looks something like this:

```
In [5]: len(dataset)

Out[5]: 148310

In [6]: dataset[0]

Out[6]: {'customer_id': '24371595',
         'helpful_votes': 0,
         'marketplace': 'US',
         'product_category': 'Gift Card',
         'product_id': 'B004LLIL5A',
         'product_parent': '346014806',
         'product_title': 'Amazon eGift Card - Celebrate',
         'review_body': 'Great birthday gift for a young adult.',
         'review_date': '2015-08-31',
         'review_headline': 'Five Stars',
         'review_id': 'R27ZP1F1CD0C3Y',
         'star_rating': 5,
         'total_votes': 0,
         'verified_purchase': True,
         'vine': False}
```

Code: Filtering by date

First let's try to filter reviews by date. We'll see more on processing time/date data later, but for the moment we'll just filter based on the review's year. We first need to convert this to an integer:

Year portion of date

```
In [7]: for d in dataset:
            d['yearInt'] = int(d['review_date'][:4])
```

```
---------------------------------------------------------------------------
KeyError                                  Traceback (most recent call last)
<ipython-input-7-0343dcce14e0> in <module>()
      1 for d in dataset:
----> 2     d['yearInt'] = int(d['review_date'][:4])

KeyError: 'review_date'
```

That threw an error! Why?

- The 'review_date' field must be missing from some reviews

Code: Filtering by date

So, we'll first have to preprocess our dataset to extract only those entries containing a 'review_date' field:

```
In [8]: dataset = [d for d in dataset if 'review_date' in d]

In [9]: len(dataset)
Out[9]: 148309
```

- Looks like it was just one review!
- Now we can try again:

```
In [10]: for d in dataset:
             d['yearInt'] = int(d['review_date'][:4])
```

Code: Filtering by date

Finally let's filter out old reviews, e.g. those written before 2010:

```
In [11]: dataset = [d for d in dataset if d['yearInt'] > 2009]

In [12]: len(dataset)
Out[12]: 148095
```

- Note that we did this preprocessing (and will do most preprocessing) using a fairly simple list comprehension

Code: Filtering by review quality

Similarly, we might filter reviews based on their "helpfulness". Let's write another list comprehension to exclude reviews with low helpful rates:

```
In [13]: dataset = [d for d in dataset if d['total_votes'] < 3 or d['helpful_votes']/d['total_votes'] >= 0.5]

In [14]: len(dataset)
Out[14]: 147801
```

Keep reviews that haven't received many votes yet

Otherwise, delete any with less than 50% helpfulness

Code: Filtering by user activity

Next, let's filter our dataset to discard inactive users (in this case, users who have written only a single review in this category)

First we'll use the "defaultdict" class, as we did in the "simple statistics" lecture:

```
In [15]: from collections import defaultdict

In [16]: nReviewsPerUser = defaultdict(int)

In [17]: for d in dataset:
             nReviewsPerUser[d['customer_id']] += 1
```

Then we can filter to keep only users with 2 or more reviews:

```
In [18]: dataset = [d for d in dataset if nReviewsPerUser[d['customer_id']] >= 2]

In [19]: len(dataset)
Out[19]: 11172
```

Note: this cuts the dataset down **a lot**

Code: Filtering by review length

Finally, let's filter very short reviews, which may be uninformative

```
In [20]: dataset = [d for d in dataset if len(d['review_body'].split()) >= 10]

In [21]: len(dataset)
Out[21]: 7033
```

Data preprocessing

These are just a few of the possible ways that we could filter our dataset, for instance we could extend this by:

- Filtering products that have few reviews
- Filtering users who have only given '5-star' ratings
- Filtering reviews that aren't part of the "vine" program
- Filter users who haven't written a review for more than a year
- Etc.

But note that this type of filtering **quickly and drastically** decreases the amount of data we have to work with!

Summary of concepts

- Gave examples of protocols we may use to filter data
- Developed Python code to apply simple preprocessing schemes
- Saw the effect of preprocessing on real data

On your own...

- Try applying the same preprocessing protocols to the Yelp data (or a small subset of it) and see what effect it has on the amount of workable data