Python Data Products

Course 3: Making Meaningful Predictions from Data

Lecture: Setting up a codebase for evaluation and validation

Learning objectives

In this lecture we will...

- Setup a simple codebase to be used in future lectures for the purpose of evaluating regressors and classifiers, and for implementing training/validation/testing pipelines

Code example: sentiment analysis

In this lecture, we'll build a model that implements **sentiment analysis,** i.e., our goal is to predict star ratings based on the text in a review

We choose this problem mainly because it includes **complex, high-dimensional features,** such that model tuning and evaluation becomes important

Code example: sentiment analysis

Importing libraries and reading data:

In [1]:
```
import gzip
from collections import defaultdict
import string # Some string utilities
import random
from nltk.stem.porter import PorterStemmer # Stemming
import numpy
```

In [2]:
```
path = "/home/jmcauley/datasets/mooc/amazon/amazon_reviews_us_Gift_Card_v1_00.tsv.gz"
```

In [3]:
```
f = gzip.open(path, 'rt', encoding="utf8")
```

Code example: sentiment analysis

Importing libraries and reading data:

In [4]:
```
header = f.readline()
header = header.strip().split('\t')
```

In [5]:
```
dataset = []
```

In [6]:
```
for line in f:
    fields = line.strip().split('\t')
    d = dict(zip(header, fields))
    d['star_rating'] = int(d['star_rating'])
    d['helpful_votes'] = int(d['helpful_votes'])
    d['total_votes'] = int(d['total_votes'])
    dataset.append(d)
```

Code example: sentiment analysis

Our goal is going to be to build a classifier which estimates **sentiment** (e.g. a star-rating) based on the occurrence of words in a document:

i.e., Let's build a predictor of the form:

$$f(\text{text}) \rightarrow \text{rating}$$

using a model based on linear regression:

$$\text{rating} \simeq \alpha + \sum_{w \in \text{text}} \text{count}(w) \cdot \theta_w$$

Code example: sentiment analysis

Our first challenge is to build a (relatively) small dictionary of words to use in our model (since it would be impractical to include every word)

Code example: sentiment analysis

Counting unique words:

In [7]:
```
# How many unique words are there?
```

In [8]:
```
wordCount = defaultdict(int)
for d in dataset:
    for w in d['review_body'].split():
        wordCount[w] += 1

print(len(wordCount))
```

```
97289
```

Code example: sentiment analysis

Removing capitalization and punctuation

In [9]:
```
# What if we ignore capitalization and punctuation?
```

In [10]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in dataset:
    r = ''.join([c for c in d['review_body'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1

print(len(wordCount))
```

```
46283
```

Concept: Stemming

Stemming is a process that maps different instances of words to a unique word "stem":

E.g.

drinks → drink
drinking → drink
drinker → drink

argue → argu
arguing → argu
argues → argu
arguing → argu
argus → argu

Code example: sentiment analysis

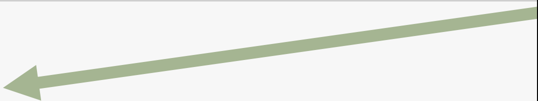
Stemming

```
In [11]: # What if we apply stemming?

In [12]: wordCount = defaultdict(int)
         punctuation = set(string.punctuation)
         stemmer = PorterStemmer()
         for d in dataset:
             r = ''.join([c for c in d['review_body'].lower() if not c in punctuation])
             for w in r.split():
                 w = stemmer.stem(w) # with stemming
                 wordCount[w] += 1

         print(len(wordCount))

         37480
```



We use a stemmer from the Python Natural Language Toolkit (NLTK) called the **Porter Stemmer**

Code example: sentiment analysis

Even after removing punctuation, capitalization, and stemming, we still have a dictionary that is **too large** to deal with practically

So, let's just take the subset of the **most popular** words to build our dictionary

Code example: sentiment analysis

Extracting the most popular words:

In [13]:
```
# Extract and build features from the most common words
```

In [14]:
```
wordCount = defaultdict(int)
punctuation = set(string.punctuation)

for d in dataset:
  r = ''.join([c for c in d['review_body'].lower() if not c in punctuation])
  for w in r.split():
    wordCount[w] += 1
```

Just removing capitalization and punctuation for the purposes of this example

In [15]:
```
counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

words = [x[1] for x in counts[:1000]]

wordId = dict(zip(words, range(len(words))))
wordSet = set(words)
```

Sort words by popularity and keep the **top 1000** most popular

Utility data structures to map each word to a unique ID

Code example: sentiment analysis

Extracting features from the most popular words

In [16]:
```
def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['review_body'].lower() if not c in punctuation])
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

Increment the counter of the corresponding word each time we see that word in the text

Append the offset feature to the end

Code example: sentiment analysis

Finally, having extract a (fixed-length) feature vector for each document (review), we can train our sentiment analysis model:

$$\text{rating} \simeq \alpha + \sum_{w \in \text{text}} \text{count}(w) \cdot \theta_w$$

Code example: sentiment analysis

For the moment, we fit our model much as we have done in previous lectures (though will change this in later lectures)

```
In [17]: random.shuffle(dataset)

In [18]: X = [feature(d) for d in dataset]

In [19]: y = [d['star_rating'] for d in dataset]

In [20]: theta,residuals,rank,s = numpy.linalg.lstsq(X, y)
```

Code example: sentiment analysis

Finally, we can examine which words have the most positive/negative sentiment by looking at their corresponding coefficients:

```
In [21]: wordWeights = list(zip(theta, words + ['offset']))
         wordWeights.sort()

In [22]: wordWeights[:10]

Out[22]: [(-1.2154565072717696, 'disappointing'),
          (-0.8574720172738775, 'disappointed'),
          (-0.7905359349220544, 'unable'),
          (-0.6808380275904105, 'waste'),
          (-0.6634111839366597, 'charged'),
          (-0.5391972452016565, 'supposed'),
          (-0.5292354754787302, 'unfortunately'),
          (-0.49739634446194575, 'australia'),
          (-0.4964445645712913, 'tried'),
          (-0.47776774788212384, 'wont')]

In [23]: wordWeights[-10:]

Out[23]: [(0.23601901891940102, 'whats'),
          (0.2383326014985222, 'problems'),
          (0.24436555664648224, 'particular'),
          (0.24700474913779302, 'worry'),
          (0.2536120024564045, 'exelente'),
          (0.2597913183314589, 'excelent'),
          (0.27148055221480827, 'excelente'),
```

Summary of concepts

- Developed a new codebase for a sentiment analysis problems
- Showed some of the challenges in modeling features from text

On your own...

- Modify the code to experiment with alternative feature representations (e.g. removing or keeping capitalization, punctuation, changing the dictionary size, etc.) to determine their impact on model accuracy