Energy Informatics

System Design — Data Analysis

UNI FREIBURG

Albert-Ludwigs-Universität Freiburg

Peter Thiemann

15 Feb 2016

What will YOU use programming for?

UNI FREIBURG

Data Analysis

- Scrutinizing large data sets
  meter readings, usage statistics, connection data
- Coming up with hypotheses
- Verifying the hypotheses

Data Analysis

- Scrutinizing large data sets
  meter readings, usage statistics, connection data
- Coming up with hypotheses
- Verifying the hypotheses

Information Sources and Data Conditioning

- Reading data from files, CSV, XML, spreadsheet
- Cleaning up: detecting formating errors, removing implausible data, outliers, etc

Our goal

UNI FREIBURG

- Simple tools for simple data analysis
- Rehearse with small examples

Real world data

UNI FREIBURG

Problem

- Where to get it?
- Often sensitive personal information
- May be reconstruct identities from anonymized data
- Example: network logs of the university

Solution for the course

- Use publicly available data

First application
Text analysis

First application

Statistical analysis on public texts

- Obtain a public domain text
  - Gutenberg project
  - Wikipedia (very large)
  - public corpora (e.g., `https://en.wikipedia.org/wiki/Brown_Corpus`)
- Possible tasks
  - Which language?
  - Which genre?
  - Which author?

On the tasks

Which Language?

- Every language has a characteristic letter frequency
- `https://en.wikipedia.org/wiki/Letter_frequency`
- Also digrams and trigrams may be analyzed

On the tasks

Which Language?

- Every language has a characteristic letter frequency
- `https://en.wikipedia.org/wiki/Letter_frequency`
- Also digrams and trigrams may be analyzed

Which genre / author?

- Analyze usage patterns of common words
- `https://en.wikipedia.org/wiki/Most_common_words_in_English`

On letter frequency and cryptanalysis

Background: substitution cipher

- Plain text and cipher text (after encryption) are drawn from the same set of symbols
- A (monoalphabetic) **substitution cipher** is a one-to-one mapping between symbols
- Particularly simple example: Caesar's cipher, which rotates letters by 13 (how would you decrypt?)

Example: Caesar's cipher

Caesar's substitution

symbols	abcdefghijklmnopqrstuvwxyz
substitutes	nopqrstuvwxyzabcdefghijklm

Example: Caesar's cipher

Caesar's substitution

symbols	abcdefghijklmnopqrstuvwxyz
substitutes	nopqrstuvwxyzabcdefghijklm

Application

plain text	we had goldfish and they circled around
cipher text	jr unq tbyqsvfu naq gurl pvepyrq nebhaq

On letter frequency and cryptanalysis

Breaking a substitution cipher

- Assumptions:
  - language is known
  - cipher text is sufficiently long
- Analyze letter frequency
- Match with letter frequency table for the language
- Compute inverse substitution

Which substitution is the best match?

UNI FREIBURG

- To assess different substitutions, we need to compute the distance to the language's letter frequency.
- The standard distance function to minize computes the square root of the squares of the differences:

$$d(\bar{x}, \bar{y}) = \sqrt{\sum_i (x_i - y_i)^2}$$

Distance in Python

Code

```
def distance(xs, ys):
    s = 0
    for x, y in zip (xs, ys):
        s += (x - y) * (x - y)
    return math.sqrt(s)
```

Explanation

- `zip (xs, ys)` creates a list of pairs of corresponding entries of lists `xs` and `ys`
- `for x, y in sequence`
  loops over the entries in `sequence`, which must be pairs, and binds `x` and `y` to the first and second component of each pair, respectively

Useful Python I/O idioms

Python IO

Reading a file naively

```
# prepare to 'r'ead from file 'filename'
f = open('filename', 'r')
s = f.read()
# process s = content of file
f.close()
```

- Reads all of a file named "`filename`" into the string `s`
- Then work with `s`

Python IO

Reading a file naively

```
# prepare to 'r'ead from file 'filename'
f = open('filename', 'r')
s = f.read()
# process s = content of file
f.close()
```

- Reads all of a file named "`filename`" into the string `s`
- Then work with `s`
- Problems:
  - This will consume **a lot** of memory if the file is big
  - It's easy to forget to close the file
  - No error handling

More robust file handling

Reading a file (recommended)

```
with open ('filename', 'r') as f:
    for line in f:
        # process f line-by-line
        # line is a string
```

More robust file handling

Reading a file (recommended)

```
with open ('filename', 'r') as f:
    for line in f:
        # process f line-by-line
        # line is a string
```

Advantages

- No memory issues as file is read line-by-line
- Automatic close when leaving `with`
- (Hidden) error handling if there is a problem with the file

More robust file handling

Reading a file (recommended)

```
with open ('filename', 'r') as f:
    for line in f:
        # process f line-by-line
        # line is a string
```

Advantages

- No memory issues as file is read line-by-line
- Automatic close when leaving `with`
- (Hidden) error handling if there is a problem with the file

Disadvantage

Have to deal with file contents one line at a time

Example: the word count utility

UNI FREIBURG

```
# wc counts lines, words, and characters in a file
def exe(name):
    # initialization
    lcount = 0     # line count
    wcount = 0     # word count
    ccount = 0     # character count
    with open (name, 'r') as f:
        for line in f:
            # process one line
            lcount += 1
            ccount += len(line)
            for words in line.split():
                wcount += 1
    return (lcount, wcount, ccount)
```

End Part III