#### **Energy Informatics**

System Design — Data Analysis

Albert-Ludwigs-Universität Freiburg

#### Peter Thiemann

# What will YOU use programming for?

Thiemann Energy Informatics 15 Feb 2016 2 / 17

#### Data Analysis

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

Thiemann Energy Informatics 15 Feb 2016 3 / 17

#### 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

Thiemann Energy Informatics 15 Feb 2016 3 / 17

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

#### Real world data

#### **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

Thiemann Energy Informatics 15 Feb 2016 6 / 17

#### First application

#### Statistical analysis on public texts

- Obtain a public domain text
  - Gutenberg project
  - Wikipedia (very large)
  - public corpora (e.g., https://en.wikiped
    - 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?)

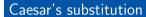
#### Example: Caesar's cipher



symbols	$\verb"abcdefghijklmnopqrstuvwxyz"$
substitutes	nopqrstuvwxyzabcdefghijklm

Thiemann Energy Informatics 15 Feb 2016 10 / 17

#### Example: Caesar's cipher



symbols	$\verb"abcdefghijklmnopqrstuvwxyz"$
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

Thiemann Energy Informatics 15 Feb 2016 11/17

#### Which substitution is the best match?

- 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

Thiemann Energy Informatics 15 Feb 2016 14 / 17

#### 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

#### Reading a file naively

```
# prepare to 'r'ead from file 'filename'
 = 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)

#### More robust file handling

#### Reading a file (recommended)

#### 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

Thiemann Energy Informatics 15 Feb 2016 16 / 17

#### More robust file handling

#### Reading a file (recommended)

#### 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

```
# 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)
```

Thiemann Energy Informatics 15 Feb 2016 17 / 17

### End Part III

Thiemann Energy Informatics 15 Feb 2016 18 / 17