Statistical methods in NLP
Introduction

Richard Johansson

January 19, 2016

today

- course matters
- analysing numerical data with Python
- basic notions of probability
- simulating random events in Python

overview

overview of the course

analysing numerical data in Python

basics of probability theory

randomness in Python

UNIVERSITY OF GOTHENBURG

why statistics in NLP and linguistics?

- in experimental evaluations:
  - a HMM tagger T_1 is tested on a sample of 1000 words and gets and accuracy rate of 0.92 (92%). **How precise** is this measurement?
  - a Brill tagger T_2 is tested on the same sample and gets an accuracy rate of 0.94. Is the Brill tagger **significantly** better than the HMM tagger?

why statistics in NLP and linguistics?

- in investigations of linguistic data:
  - what is the probability of object fronting in Dutch?
  - do speakers affected by Alzheimer's disease exhibit a **significantly** smaller vocabulary?

why statistics in NLP and linguistics?

- in language processing systems:
  - what is the probability of English *case* being translated to Swedish *kapsel*?
  - . . . of a noun if the previous word was a verb?
- data-driven NLP systems: we specify a general model and tune specific parameters by observing our data

course overview

- theoretical part:
  - probability theory
  - statistical inference
  - statistical methods in experiments
- applications in NLP:
  - classifiers, taggers, parsers, topic models
  - machine translation

course work

- lectures – all in **L308** (except tomorrow)
- lab sessions – in the computer lab **G212** (lab 4)
- always on Tuesdays at 10–12 (except March 15) or Thursdays at 13–15
- teachers:
  - **Richard**: gives most of the lectures and supervises the two introductory computer exercises
  - **Prasanth**: machine translation lecture and assignment
  - **Mehdi**: supervises and grades the other assignments

examination

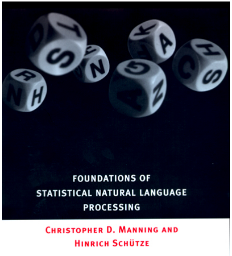
- 2 mandatory computer exercises
- 3 mandatory programming assignments
  - text categorization
  - evaluation
  - PoS tagger implementation
- 2 optional programming assignments for VG
  - topic modeling
  - machine translation

deadlines

- computer exercises: a few days
- programming assignments: 2 weeks after lab session
- VG assignments: March 28

literature

- Manning and Schütze: *Foundations of Statistical Natural Language Processing*



- `http://nlp.stanford.edu/fsnlp/`

literature linked from the web page

- Krenn and Samuelsson: The Linguist's Guide to Statistics – *Don't Panic!*
  - alternative to M&S for the theoretical part of the course
- lecture notes by Michael Collins
  - for the NLP part
  - more in-depth than M&S
- Mitchell: Naïve Bayes and Logistic Regression
- a few other research papers

overview

overview of the course

analysing numerical data in Python

basics of probability theory

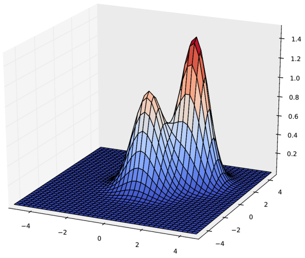
randomness in Python

summary statistics and plotting

- if we have some collection of data, it can be useful to summarize the data using plots and high-level measures
- a useful skill in general when carrying out experiments
- and we'll use it in the two computer exercises

some useful Python libraries we'll use in this course

- **SciPy**: a Python library for statistics and math in general
  - http://www.scipy.org/
- **NumPy**: efficient mathematical functions
  - http://www.numpy.org/
- **matplotlib**: using Python to draw diagrams
  - http://matplotlib.org/



an example dataset

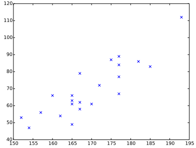
```
177 67 m
177 77 m
175 87 m
154 47 f
157 56 f
152 53 f
165 49 f
165 66 f
165 63 m
162 54 f
167 62 m
167 79 f
167 58 f
165 61 f
```

```
# read the data
data = []
with open('bodies.txt') as f:
    for l in f:
        h, w, s = l.split()
        data.append( (int(h), int(w), s) )

heights = [ h for h, _, _ in data ]
weights = [ w for _, w, _ in data ]

m_heights = [ h for h, _, s in data if s == 'm']
m_weights = [ w for _, w, s in data if s == 'm']
f_heights = [ h for h, _, s in data if s == 'f']
f_weights = [ w for _, w, s in data if s == 'f']
```

plotting the data

- import the plotting library

  `from matplotlib import pyplot as plt`



- plot a height/weight plot, each point as an 'x'

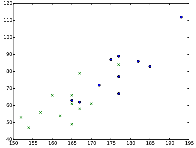
  `plt.plot(heights, weights, 'x')`

- plot height/weight plot by gender

  `plt.plot(m_heights, m_weights, 'o', f_heights, f_weights, 'x')`

- save the plot to a file

  `plt.savefig('myplot.pdf')`



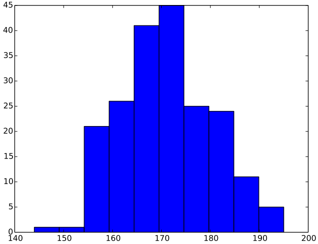
- alternatively, draw the plot on the screen

  `plt.show()`

plotting histograms

- a **histogram** is a diagram that shows how the data points are distributed
- the x axis shows "bins", e.g. 165–170 cm, and the y axis shows the number of data points in that bin
- here's how we draw a histogram with matplotlib:

```
plt.hist(heights, bins=10)
```



some basic data analysis

- **maximal** and **minimal** values:

```
max_f_height = max(f_heights)
print('Tallest female: {0} cm'.format(max_f_height))
```

- **sample mean** (average) and **median**:

```
mean_m_weight = scipy.mean(m_weights)
print('Mean male weight: {0} kgs'.format(mean_m_weight))

median_f_weight = scipy.median(f_weights)
print('Median female weight: {0} kgs'.format(median_f_weight))
```

measures of dispersion: variance and standard deviation

- recall that the mean $\bar{x}$ of a dataset x is defined

$$\bar{x} = \frac{1}{n}\sum_{i=1}^{n} x_i$$

- the **sample variance** $V(x)$ of a dataset x measures how much x is concentrated to the mean
  - it is the mean of the squares of the offsets from the mean

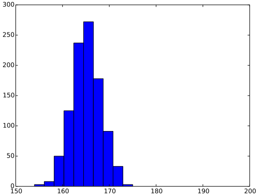
$$V(x) = \frac{1}{n}\sum_{i=1}^{n}(x_i - \bar{x})^2$$

- the **sample standard deviation** $\sigma(x)$ is the square root of the variance

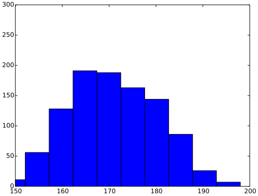
$$\sigma(x) = \sqrt{V(x)}$$

example

- low variance: data concentrated near the mean
  - in the extreme case: all values are identical
- high variance: data spread out



$\sigma = 2.9$



$\sigma = 9.1$

variance and standard deviation in SciPy

- variance:

```
var_height = scipy.var(heights)
```

- standard deviation:

```
std_m_weight = scipy.std(m_weights)
```

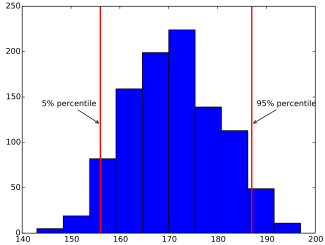
new feature in standard Python: the statistics library

- recently, some statistical functions were added to Python's standard library, for instance
  - `statistics.mean`
  - `statistics.median`
  - `statistics.stdev`
- see https://docs.python.org/3/library/statistics.html

percentiles

- how tall are the shortest 5% of the people in the dataset?
  - formally: what is the x such that 5% of the data is less than x?
- this number is called the **5% percentile**
- in Python:

```
p5 = numpy.percentile(heights, 5)
```



relations between two variables: correlation

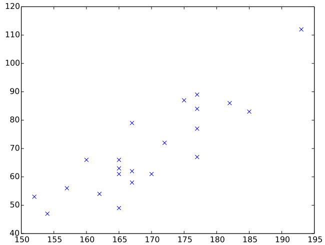
- the **correlation coefficient** or the **Pearson** r measures how close the data is to a linear relationship
- it is a number that ranges between -1 and +1
  - example [Wikipedia]:

- it is defined

$$r(x,y) = \frac{\sum_{i=1}^{n}(x_i - \bar{x})(y_i - \bar{y})}{\sigma(x)\sigma(y)}$$

correlation example

- for the height–weight data, $r = 0.87$



- with Python:

```
correlation = scipy.stats.pearsonr(heights, weights)[0]
```

overview

overview of the course

analysing numerical data in Python

basics of probability theory

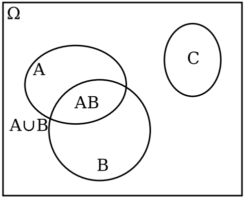
randomness in Python

what are probabilities?

- relative frequencies?
  - if we can repeat an experiment: **how often** does the event E occur?
  - when we roll a die, we may say that the probability of a "4" is 1/6 because we will get a "4" approximately 1/6 of the time
- degrees of belief?
  - what is the probability that Elvis Presley is alive?
  - with which probability could Germany have won WW2?

some formal notation: events

- the theory of probability is built on the theory of sets
  - so we can draw Venn diagrams to make the notions more intuitive

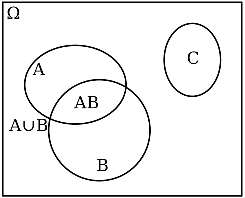


- Ω is the **sample set**: the set of all possible situations
- an **event** A is a subset of Ω
- the **union** event $A \cup B$ means that either A or B has happened
- the **joint** event AB (also written $A \cap B$) means that A and B have both happened
- two events A and C that can't happen at the same time (that is, the intersection is empty) are called **disjoint**

UNIVERSITY OF GOTHENBURG

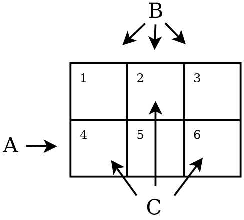
the mathematical definition: the Kolmogorov axioms

- the probability $P(A)$ is a number such that
  - $0 \leq P(A) \leq 1$ for every event A
  - $P(\Omega) = 1$
  - $P(A \cup B) = P(A) + P(B)$ if A and B are disjoint
- in the illustrations, $P(A)$ intuitively corresponds to the area covered by A in the Venn diagram



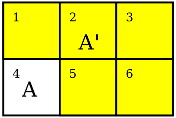
example: Dice rolling

- A = "rolling a 4"; $P(A) = ?$
- B = "rolling 3 or lower"; $P(B) = ?$
- C = "rolling an even number"; $P(C) = ?$
- $P(A \cup B) = P(A) + P(B)$?
- $P(A \cup C) = P(A) + P(C)$?
- P(rolling 1, 2, 3, 4, 5, or 6) $= ?$



some consequences

- $A' =$ "everything but A" $= \Omega \setminus A$



$$P(A') = 1 - P(A)$$

$$P(\emptyset) = 0$$

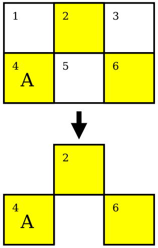
- $A =$ "rolling a 4"; $P(A) = ?$
- $A' =$ "not rolling 4"; $P(A') = ?$
- P(rolling neither 1, 2, 3, 4, 5, 6) $= ?$

joint and conditional probabilities

- the probability of both A and B happening is called the **joint probability**, written $P(AB)$ or $P(A, B)$
- definition: if $P(B) \neq 0$, then

$$P(A|B) = \frac{P(AB)}{P(B)}$$

  is referred to as the **conditional probability of A given B**
- intuitively in the Venn diagram: zoom in on B
  - "what is the probability of a 4 if we know it's an even number?"
- this is something we've already used in language models, taggers, etc



example: vowels in English

- $P(\text{vowel}) = 0.36$
- $P(\text{vowel} \mid \text{previous is vowel}) = 0.12$
- $P(\text{vowel} \mid \text{previous is not vowel}) = 0.50$

the multiplication rule and the chain rule

- if we rearrange the definition of the conditional probability, we get the **multiplication rule**

$$P(AB) = P(A|B) \cdot P(B)$$

- if we have more than two events, this rule can be generalized to the **chain rule**

$$P(ABC) = P(A|BC) \cdot P(B|C) \cdot P(C)$$

- this decomposition is used in taggers, language models, etc

independent events

- definition: two events A and B are **independent** if

$$P(AB) = P(A) \cdot P(B)$$

- this can be rewritten in a more intuitive way: "the probability of A does not depend on anything about B"

$$P(A|B) = P(A)$$

examples

- dice rolling:
  - A = "rolling a 4 the first time"
  - B = "rolling a 4 the second time"
- drawing cards:
  - A = "drawing the ace of spades as the first card"
  - B = "drawing the ace of spades as the second card"

recap: the Markov assumption in language models

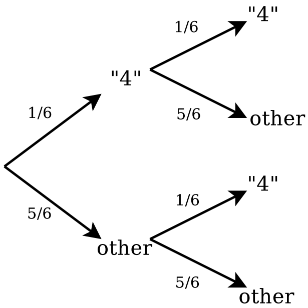
- unigram language model: we assume that the words occur independently

$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_3|w_2, w_1) \cdot P(w_2|w_1) \cdot P(w_1) \\ &= P(w_3) \cdot P(w_2) \cdot P(w_1) \end{aligned}$$

- bigram model: word is independent of everything but the previous word

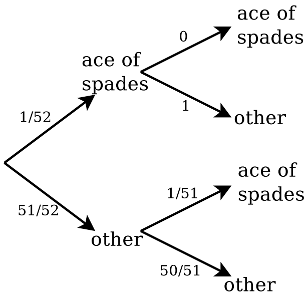
$$\begin{aligned} P(w_1, w_2, w_3) &= P(w_3|w_2, w_1) \cdot P(w_2|w_1) \cdot P(w_1) \\ &= P(w_3|w_2) \cdot P(w_2|w_1) \cdot P(w_1) \end{aligned}$$

drawing tree diagrams (1)



UNIVERSITY OF GOTHENBURG

drawing tree diagrams (2)



overview

overview of the course

analysing numerical data in Python

basics of probability theory

randomness in Python

a brief note on "random" numbers

- **pseudorandom numbers**: generating "random" numbers in a computer using a deterministic process
  - usually fast
  - we use a starting point called the **seed**
  - if we use the same seed, we'll get the same sequence
    - good for replicable experiments
  - might be a security risk in some situations
- hardware random numbers
  - sample noise from hardware devices
  - Linux: `/dev/random`

basic functions for random numbers: the `random` library

- reset the random number generator

  ```
  random.seed(0)
  ```

- generate a random floating-point number between 0 and 1

  ```
  random_float = random.random()
  ```

- generate a random integer between 1 and 6

  ```
  die_roll = random.randint(1, 6)
  ```

- shuffle the items of a list `lst`

  ```
  random.shuffle(lst)
  ```

- pick a random item from a list `lst`

  ```
  selection = random.choice(lst)
  ```

a note on the random number generators

- the two random number generating functions are examples of **random variables** with **uniform distributions**
  - this means that all outcomes are equally probable
  - if we generate a lot of random numbers, the histogram will be flat
- `random.randint(1, 6)` is a **discrete** uniform random variable
  - it generates 1, 2, 3, 4, 5, or 6 with equal probability $\frac{1}{6}$
- `random.random()` is a **continuous** uniform random variable
  - it generates any float between 0 and 1 with equal probability
- we'll come back to the notion of random variables and their distributions in the next lecture

simulating random events in Python

- random.random and random.randint can be used to simulate random events
- example of generating random words with different probabilities

```
import random

def random_word():
    r = random.random()
    if r < 0.4:
        return 'the'
    if r < 0.7:
        return 'and'
    if r < 0.9:
        return 'in'
    return 'is'

random_words = [ random_word() for _ in range(20) ]

print(random_words)
```

next lecture (Thursday)

- a few more notions from basic probability theory
- random variables and their distributions