N-Grams

Daniël de Kok

October 23, 2017

Practical matters

Today

- Practical matters
- Course overview
- Language guessing using n-grams
- Maven

Course information

- See http://www.sfs.uni-tuebingen.de/~ddekok/dsa3/
- Schedule, complementary literature, assignments, slides

Who's who?

- Lecturer: Daniël de Kok
- Tutors:
  - Peter Schoener
  - Teslin Roys

Prerequisites

- Data structures and Algorithms for CL I
- Data structures and Algorithms for CL II

Coursework

- Reading material for most lectures
- Weekly assignments, of which 3 are graded ($3 \times 20\%$).
- Written exam (40%).

Literature

- *Algorithms*, Sedgewick & Wayne, 4th Edition, Addison-Wesley Professional
- *Speech and Language Processing*, Jurafsky & Martin, 2nd Edition, Prentice Hall
- *Dependency Parsing*, Kübler, McDonald & Nivre, Morgan & Claypool

Literature (availability)

- Sedgewick & Wayne is readable from the university network through Safari books: proquest.tech.safaribooksonline.de/9780132762571
- Jurafsky & Martin are currently working on the 3rd edition of their book. Draft chapters are available at: web.stanford.edu/~jurafsky/slp3/

Registration

- Send an e-mail to: dsa3+registration@danieldk.eu
- Before October 26 (Thursday)
- Include:
  - Your name
  - Your 'Matrikelnummer'

Practical

- Office hour: Tuesday, 10:00-11:00, please make an appointment!
- Please ask questions about the material presented in the lectures during the lectures — Everyone benefits
- We will discuss each assignment that is not graded during the next lab.

Presence

- A presence sheet is circulated **purely** for statistics.
- Experience: those who do not attend lectures or do not make the assignments usually fail the course.
- Do not expect us to answer your questions if you were not at the lectures.

Honesty statement

- Feel free to cooperate on assignments that are not graded.
- Assignments that are graded must be your own work. Do **not**:
  - Copy a program (in whole or in part).
  - Give your solution to a classmate (in whole or in part).
  - Get so much help that you cannot honestly call it your own work.
  - Receive or use outside help.
- Sign your work with the honesty statement (provided on the website).
- Above all: You are here for yourself, practice makes perfection.

Course sneak preview

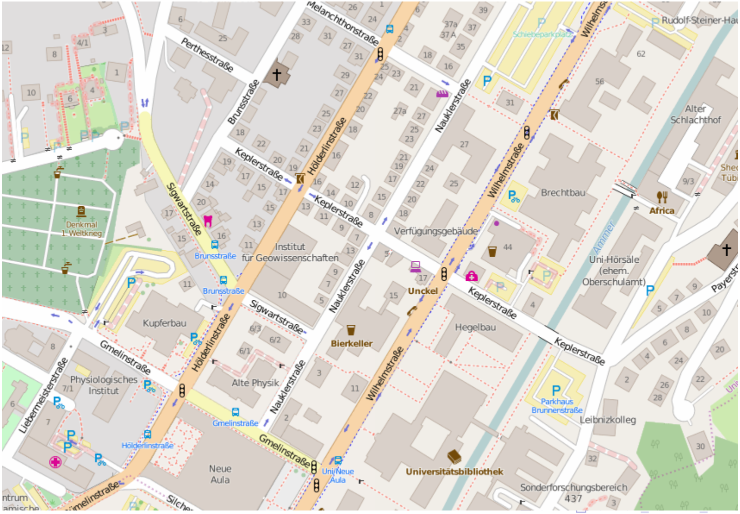
Sorting

Bundesarchiv, Bild 183-22350-0001
Foto: Junge, Peter Heinz | 20. November 1953

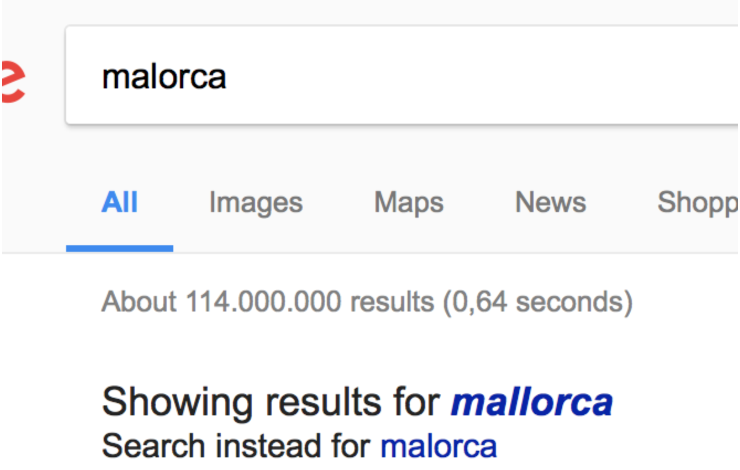
Undirected graph[1]

[1] Source: `https://www.flickr.com/photos/dylan20/6172752831/in/album-72157627605561687/`

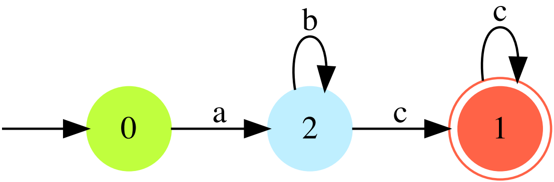
Directed graph



String distance

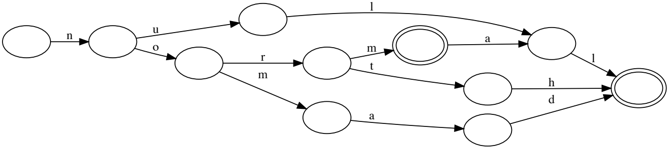


Finite state automata

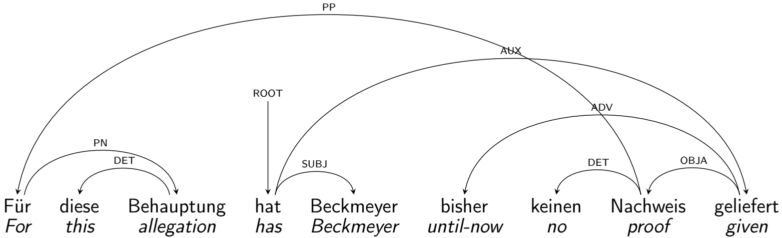


Finite state automata (2)

$\{nomad, norm, normal, north, null\}$



Parsing


PP
AUX
ROOT
ADV
PN
DET
SUBJ
DET
OBJA
Für For
diese this
Behauptung allegation
hat has
Beckmeyer Beckmeyer
bisher until-now
keinen no
Nachweis proof
geliefert given

Language identification

Guess the language!

Language 1

A **lambda-kalkulus** (vagy λ-kalkulus) egy formális rendszer, amit eredetileg matematikai függvények tulajdonságainak (definiálhatóság, rekurzió, egyenlőség) vizsgálatára vezettek be. Az elmélet kidolgozói Alonzo Church és Stephen Cole Kleene voltak az 1930-as években. Church, 1936-ban, a λ-kalkulus segítségével bizonyította, hogy nem létezik algoritmus a híres Entscheidungsproblem (döntési probléma) megoldására. A λ-kalkulus (akárcsak a Turing-gép) lehetővé teszi, hogy pontosan (formálisan) definiáljuk, mit is értünk kiszámítható függvény alatt.

A λ-kalkulust nyugodtan nevezhetjük a legegyszerűbb általános célú programozási nyelvnek. Csak egyfajta értéket ismer: a függvényt (absztrakciót), és csak egyfajta művelet van benne: a függvény alkalmazás (változó-behelyettesítés). Ezen látszólagos egyszerűsége ellenére minden algoritmus, ami Turing-gépen megvalósítható, az megvalósítható tisztán a λ-kalkulusban is. Ez az azonosság a λ-kalkulus és a Turing-gép kifejező ereje (expressive power) között adja egyébként a Church–Turing-tézis alapját.

Míg korábban a λ-kalkulus elsősorban a kiszámíthatóságelmélet (Theory of Computation) miatt volt érdekes, napjainkban ez már kevésbé hangsúlyos, és sokkal inkább a funkcionális programozási nyelvek elméleti és gyakorlati megalapozásában játszott jelentős, mondhatni központi szerepe került előtérbe.

Language 2

De **lambdacalculus**, soms ook als **λ-calculus** geschreven, is een formeel systeem dat in de wiskunde en theoretische informatica wordt gebruikt om het definiëren en uitvoeren van berekenbare functies te onderzoeken. Hij werd in 1936 door Alonzo Church en Stephen Kleene geïntroduceerd als onderdeel van hun onderzoek naar de grondbeginselen van de wiskunde, maar wordt tegenwoordig vooral gebruikt bij het onderzoeken van berekenbaarheid. De lambdacalculus kan worden gezien als een soort minimale programmeertaal die in staat is elk algoritme te beschrijven. De lambdacalculus is Turing-volledig en vormt de basis van het paradigma voor functionele programmeertalen.

De rest van dit artikel gaat over de oorspronkelijke, ongetypeerde lambdacalculus. De meeste toepassingen gebruiken varianten daarvan met een type-aanduiding.

Language 3

Ля́мбда-исчисле́ние (*λ-исчисление*) — формальная система, разработанная американским математиком Алонзо Чёрчем, для формализации и анализа понятия вычислимости.

λ-исчисление может рассматриваться как семейство прототипных языков программирования. Их основная особенность состоит в том, что они являются языками *высших порядков*. Тем самым обеспечивается систематический подход к исследованию операторов, *аргументами* которых могут быть другие операторы, а значением также может быть оператор. Языки в этом семействе являются функциональными, поскольку они основаны на представлении о функции или операторе, включая функциональную аппликацию и функциональную абстракцию. λ-исчисление реализовано Джоном Маккарти в языке Лисп. Вначале реализация идеи λ-исчисления была весьма громоздкой. Но по мере развития Лисп-технологии (прошедшей этап аппаратной реализации в виде Лисп-машины) идеи получили ясную и четкую реализацию.

Language 4

関数を表現する式に文字ラムダ（λ）を使うという慣習からその名がある。アロンゾ・チャーチとスティーヴン・コール・クリーネによって1930年代に考案された。1936年にチャーチはラムダ計算を用いて一階述語論理の決定可能性問題を（否定的に）解いた。ラムダ計算は「計算可能な関数」とはなにかを定義するために用いられることもある。計算の意味論や型理論など、計算機科学のいろいろなところで使われており、特にLISP、ML、Haskellといった関数型プログラミング言語の理論的基盤として、その誕生に大きな役割を果たした。

ラムダ計算は1つの変換規則（変数置換）と1つの関数定義規則のみを持つ、最小の（ユニバーサルな）プログラミング言語であるということもできる。ここでいう「ユニバーサルな」とは、全ての計算可能な関数が表現でき正しく評価されるという意味である。これは、ラムダ計算がチューリングマシンと等価な数理モデルであることを意味している。チューリングマシンがハードウェア的なモデル化であるのに対し、ラムダ計算はよりソフトウェア的なアプローチをとっている。

この記事ではチャーチが提唱した元来のいわゆる「型無しラムダ計算」について述べている。その後これを元にして「型付きラムダ計算」という体系も提唱されている。

Language 5

λ演算（英语：lambda calculus，λ-calculus）是一套用于研究函数定义、函数应用和递归的形式系统。它由阿隆佐・邱奇和他的学生斯蒂芬・科尔・克莱尼在20世纪30年代引入。邱奇运用λ演算在1936年给出判定性问题（Entscheidungsproblem）的一个否定的答案。这种演算可以用来清晰地定义什么是一个可计算函数。关于两个lambda演算表达式是否等价的命题无法通过一个“通用的算法”来解决，这是不可判定性能够证明的头一个问题，甚至还在停机问题之先。Lambda演算对函数式编程语言有巨大的影响，比如Lisp语言、ML语言和Haskell语言。

Lambda演算可以被称为最小的通用程序设计语言。它包括一条变换规则（变量替换）和一条函数定义方式，Lambda演算之通用在于，任何一个可计算函数都能用这种形式来表达和求值。因而，它是等价于图灵机的。尽管如此，Lambda演算强调的是变换规则的运用，而非实现它们的具体机器。可以认为这是一种更接近软件而非硬件的方式。

本文讨论的是邱奇的“无类型lambda演算”，此后，已经研究出来了一些有类型lambda演算。

How can we guess a language?

Any ideas?

Introduction

- We can make a computer guess the language:
  - Using simple n-gram statistics
  - Using a small amount of training data
  - With high accuracy
- Here we will discuss the method of *Cavnar and Trenkle, 1994*

Necessary information

We can usually identify a language using only a very short fragment. E.g.:

- German: *plötzlichen Ausbruch des Vulkans Ontake in Japan*
- English: *cross-country navigational exercise and made a banking*
- Spanish: *provenientes del idioma japonés que describen una*

Language by n-grams

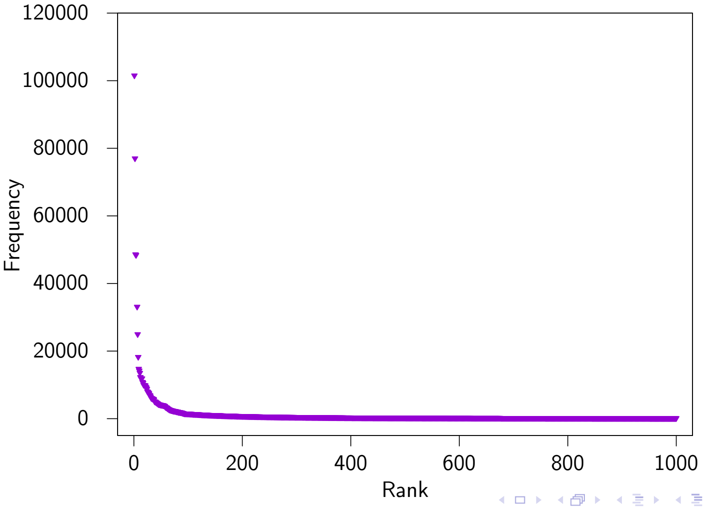
Some examples of n-grams that frequently occur these languages:

- German: *ung*, *chen*, *der*, *die*, *ö*
- English: *th*, *y_*, *ed_*, *wh*
- Spanish: *la*, *que*, *ió*, *los_*

How much information needed?

- If we were to build a model of a couple of languages, how much information do we need per language to classify most texts correctly?
- To find an answer to this question, we look at Zipf's law: *The frequency of a word is inversely proportional to its frequency-based rank*
- That is, the most frequent word will occur approximately twice as many times as the second most frequent word, thrice as many times as the third most frequent word, etc.

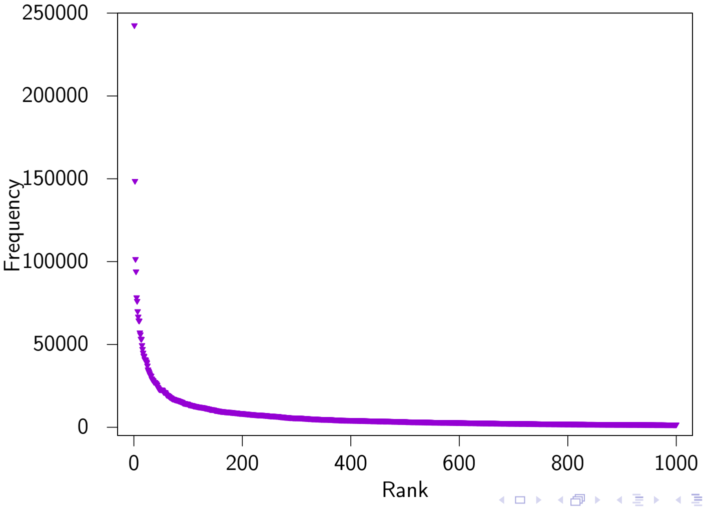
Distribution of tokens in TüBa-D/Z



Distribution of n-grams

- Not surprisingly, it turns out that the distribution of character n-grams is also zipfian.

Distribution of character trigrams in TüBa-D/Z



Lessons learned

- A small number of n-grams pop up 'all over the place';
- consequently, only a small number of n-grams are effective indicators;
- documents from a language should have similar n-gram frequency distributions.

Language profile

Cavnar and Trenkle create a *profile* of a language using a small amount of text in the following manner:

- Count each *1..5*-gram in the text
- Sort the n-grams by frequecy (most frequent first)
- Retain the 300 most frequent n-grams

Note: Cavnar and Trenkle discard all characters that are not letters or quotes.

Example: *bananas*

Example: *bananas*

n-gram	freq	n-gram	freq
a	3	ba	1
an	2	ban	1
ana	2	bana	1
n	2	banan	1
na	2	nan	1
anan	1	nana	1
anana	1	nanas	1
anas	1	nas	1
as	1	s	1
b	1		

Example: ranks *bananas*

n-gram	rank	n-gram	rank
a	1	ba	11
an	2	ban	12
ana	3	bana	13
n	4	banan	14
na	5	nan	15
anan	6	nana	16
anana	7	nanas	17
anas	8	nas	18
as	9	s	19
b	10		

In-class assignment

Make a language profile for the string:

`ananas`

Language identification

- Generate a profile for each language, based on a longer text.
- When classifying the language of a document:
  - Create a profile of the document.
  - Compare the profile of the document with the profile of each language.
  - Choose the language with the most similar profile.
- How do we compare two profiles?

Comparing profiles

Given the document profile p and language profile q, the distance function is defined as:

$$d(p,q) = \sum_{\text{ngram} \in p} |rank(\text{ngram}, p) - rank(\text{ngram}, q)|$$

The *rank* function gives the rank of an n-gram in the profile, or the size of the profile if it does not have the n-gram.

Example (Cavnar and Trenkle)

p	q	$\|\Delta\|$
th	th	0
ing	er	3
on	on	0
er	le	2
and	ing	1
ed	and	max
...	...	...

Basic algorithm

$L \leftarrow \{\langle \text{language}, \text{profile} \rangle\}$
$p \leftarrow$ document profile
$guess \leftarrow \epsilon$
$guess\_dist \leftarrow \infty$
for all $(l, q) \in L$ **do**
    dist $\leftarrow d(p, q)$
    if dist $<$ guess_dist **then**
        guess_dist $\leftarrow$ dist
        guess $\leftarrow$ l
    end if
end for

Results (Cavnar and Trenkle)

Article Length	≤ 300	≤ 300	≤ 300	≤ 300	> 300	> 300	> 300	> 300
Profile Length	**100**	**200**	**300**	**400**	**100**	**200**	**300**	**400**
Newsgroup								
australia	100.0	100.0	100.0	100.0	100.0	100.0	100.0	100.0
brazil	70.0	80.0	90.0	90.0	91.3	91.3	95.6	95.7
britain	96.9	100.0	100.0	100.0	100.0	100.0	100.0	100.0
canada	100.0	100.0	100.0	100.0	100.0	*99.6	100.0	100.0
celtic	100.0	100.0	100.0	100.0	99.7	100.0	100.0	100.0
france	90.0	95.0	100.0	*95.0	99.6	99.6	*99.2	99.6
germany	100.0	100.0	100.0	100.0	98.9	100.0	100.0	100.0
italy	88.2	100.0	100.0	100.0	91.6	99.3	99.6	100.0
latinamerica	91.3	95.7	*91.3	95.7	97.5	100.0	*99.5	*99.0
mexico	90.6	100.0	100.0	100.0	94.8	99.1	100.0	*99.5
netherlands	92.3	96.2	96.2	96.2	96.2	99.0	100.0	100.0
poland	93.3	93.3	100.0	100.0	100.0	100.0	100.0	100.0
portugal	100.0	100.0	100.0	100.0	86.8	97.6	100.0	100.0
span	81.5	96.3	100.0	100.0	90.7	98.9	98.9	99.45
Overall	**92.9**	**97.6**	**98.6**	**98.3**	**97.2**	**99.5**	**99.8**	**99.8**

Note: Asterisks indicate combinations of test variables that did worse than similar combinations using shorter profiles.

Task difficulty

The classification problem can be made more complicated by:

- Adding more languages
- Adding languages that are very similar
- Adding dialects
- Identification of very short fragments

Obviously, there are newer methods that use more sophisticated statistical modeling and/or machine learning to identify languages.

Real-world usage

Some real-world usage examples:

- Spamassassin uses the guessed language as a feature in spam identification.
- Web browsers language identification to offer you to translate a page when it is not in your native language.
- Google Translate uses language identification to determine the source language of a text to be translated.

Apache Maven

Maven introduction

Apache Maven is a tool for building and managing Java projects. Advantages of Maven are:

- Declarative: you do not have to specify the steps to build a project.
- Dependency management: you specify the dependencies of your project and Maven will automatically download them and make them available in the classpath.
- IDE agnostic: all major IDEs (including IntelliJ, NetBeans, and Eclipse) have plugins for Maven, meaning that you can open a Maven project in any IDE.
- Plugins: the functionality of Maven can easily be extended using plugins.

Maven project layout

- **pom.xml** Maven project description
- **src/main** Main project sources
  - **java** Main Java sources
  - **resources** Resources
- **src/test** Test sources
  - **java** Java sources for tests
  - **resources** Resources for tests

pom.xml

Example of a “Project Object Model”

Dependencies

- Many Java libraries are in the Maven Central Repository.
- Usually, you will find a fragmant on the website of a project. In the form of a snippet that goes into *pom.xml*. For instance:

```
<dependency>
  <groupId>com.google.guava</groupId>
  <artifactId>guava</artifactId>
  <version>18.0</version>
</dependency>
```

- You can also search the Central Repository using search.maven.org

Basic Maven commands:

```
# Clean up a project (remove compiled Java code)
mvn clean
# Compile a project
mvn compile
# Run unit tests
mvn test
```

Maven

More about Maven tomorrow!