Klassifizierung von Dokumenten mit Clustering- und Machinelearning-Verfahren

Jan-Ole Esleben
esleben@cl.uni-heidelberg.de

Titus v. d. Malsburg
malsburg@cl.uni-heidelberg.de

10. Februar 2002

Motivation

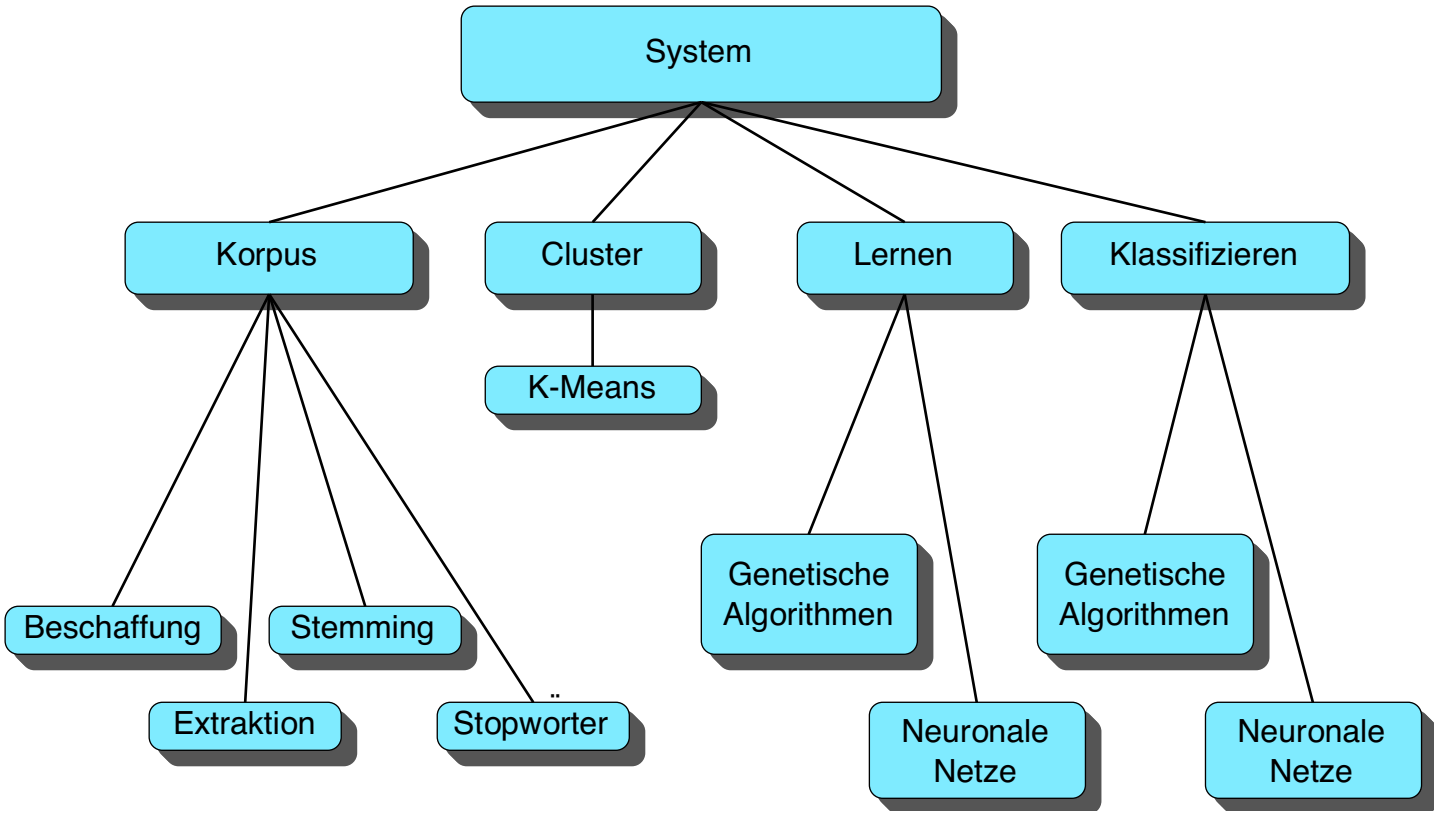
- Indexterme werden neben anderen Merkmalen zur Kassifikation von Dokumenten benutzt.
- Anzahl der Indexterme meistens gross.
- Lernalgorithmen (hier: Neuronale Netze, Genetische Algorithmen) skalieren schlecht.
- Clustering von Indextermen verringert die Anzahl der Attribute, die die Lerner verarbeiten müssen.

Kann Clustering der Indexterme Effizienz der Lerner steigern?

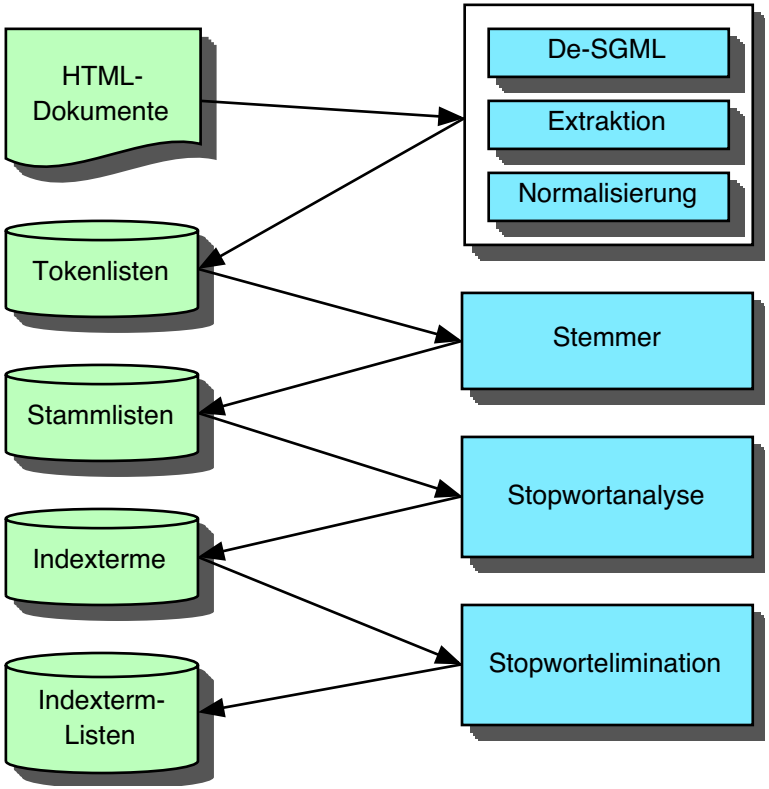
Übersicht

- Korpus: Online Ausaben von ”Die Zeit” des Jahres 2001 als HTML-Seiten
- Implementierung: Plattformunabhängig, in C, Python, Ruby, Java
- Kein monolithisches Werkzeug, sondern kleine Programme
- Datenaustausch über einfache Textdateien

Komponenten



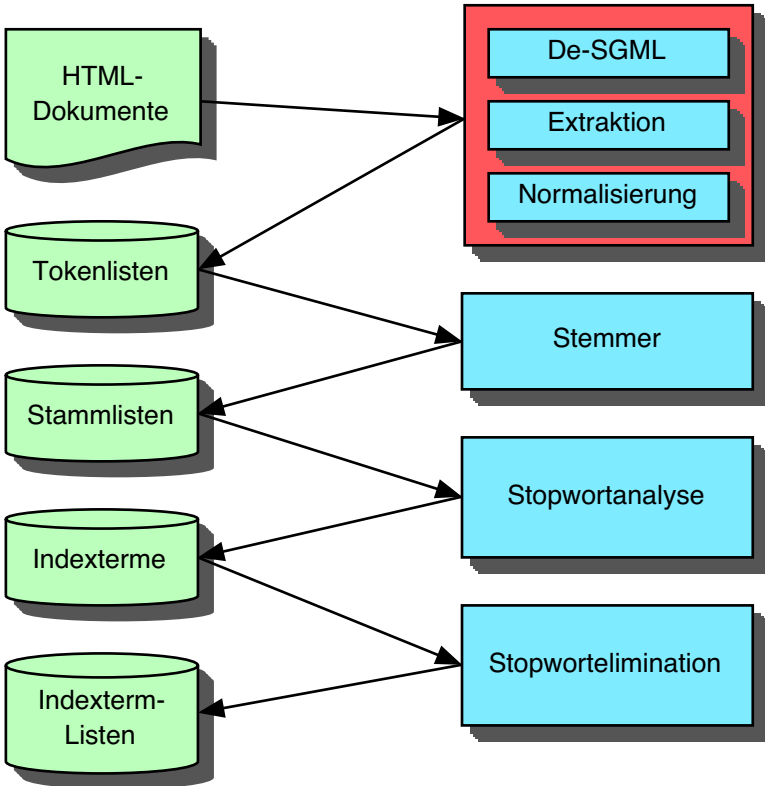
Korpus


HTML-
Dokumente
Tokenlisten
Stammlisten
Indexterme
Indexterm-
Listen
De-SGML
Extraktion
Normalisierung
Stemmer
Stopwortanalyse
Stopwortelimination

Orginaldaten

```
<img src="../../../bilder/a.gif" width="1" height="12"
alt="" border="0"><br>
<div class="margin">
<font face="Verdana,Geneva,Arial,Helvetica,
sans-serif" size="2">James Harris, General Manager des
Aladdin Hotel-Resorts, greift zum Handy: <I>żHi, </I>
Howard, du kannst uns jetzt das Gewitter schicken. Ich
bin mit meiner Gruppe am Hafen.ń Und schon geht’s los:
Der Himmel verdunke...
```

Aufbereitung



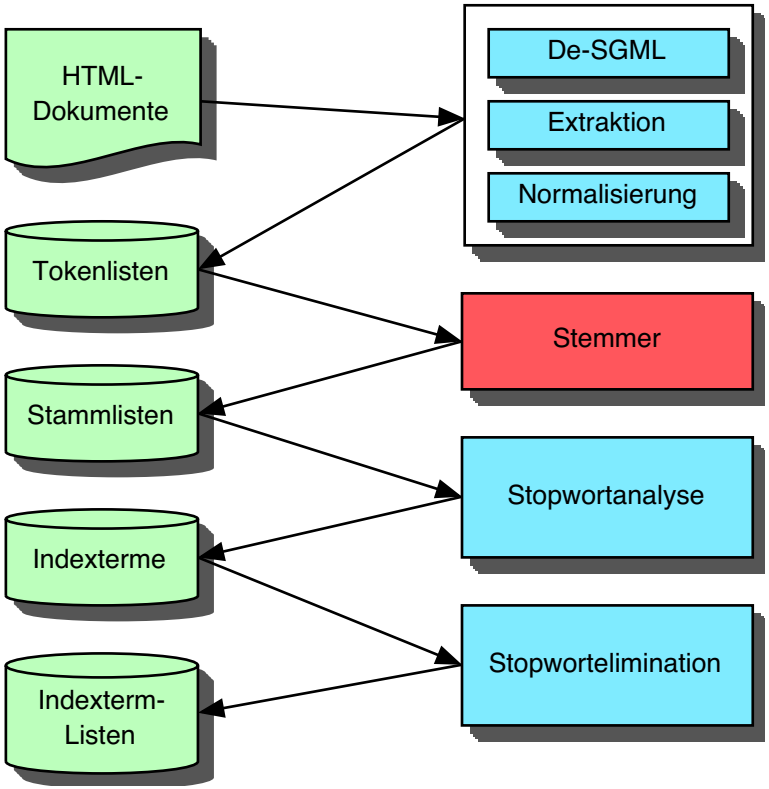
Aufbereitung

- De-SGML extrahiert Markup und ersetzt Entities
- Extraktion entfernt Navigation, Werbung, etc
- Normalisierung erzeugt Wortlisten
- Alle drei sind Lex-Programme

Tokenlisten

James	du	Und
Harris	kannst	schon
General	uns	gehts
Manager	jetzt	los
des	das	Der
Aladdin	Gewitter	Himmel
Hotel-Resorts	schicken	verdunkelt
greift	Ich	sich
zum	bin	Blitze
Handy	mit	zucken
Hi	meiner	
Howard	Hafen	

Stemming


HTML-
Dokumente
Tokenlisten
Stammlisten
Indexterme
Indexterm-
Listen
De-SGML
Extraktion
Normalisierung
Stemmer
Stopwortanalyse
Stopwortelimination

Stemming

- Der Porter-Algorithmus (kurzer Überblick)
- Implementierung

Der Porter-Algorithmus für das Deutsche

- weist einem Token Bereiche R1 und R2 zu,
- legt bestimmte Buchstabenfolgen fest, die -s und -st-Endungen erlauben,
- trennt danach in vier Durchläufen alle abgedeckten Endungen ab (im ersten Durchlauf beispielsweise -est, -en, -er und -s) – dabei müssen jeweils bestimmte Regeln beachtet werden (in welchem Bereich muß sich eine Endung befinden, um abgetrennt zu werden? Bleiben genügend Zeichen übrig?) –
- und führt abschließend einige Nachbearbeitungen durch

Implementierung

Um in der Vorbereitungsphase etwas optimalere Leistung zu bekommen, und weil es weniger Code ist, haben wir das mit `flex` implementiert.

Weil es ein so schönes Beispiel für die Vorteile von `flex` ist, folgt an dieser Stelle der Code für Phase 1.

```
%{
#include <string.h>
#define YYTRIM(n) yytext[strlen(yytext)-n] = '\0'
#define YYECHO printf("%s", yytext)
%}
VOWELS [äöüaeiou]
S_END [bdfghklmnrt]
%%
{VOWELS}[^{VOWELS}].*est$       YYTRIM(3); YYECHO;
{VOWELS}[^{VOWELS}].*(en)|(er)$ YYTRIM(2); YYECHO;
{VOWELS}[^{VOWELS}].*{S_END}s$  {
if(strlen(yytext) > 5) YYTRIM(1);
YYECHO;
}
{VOWELS}[^{VOWELS}].*e$ YYTRIM(1); YYECHO;
%
```

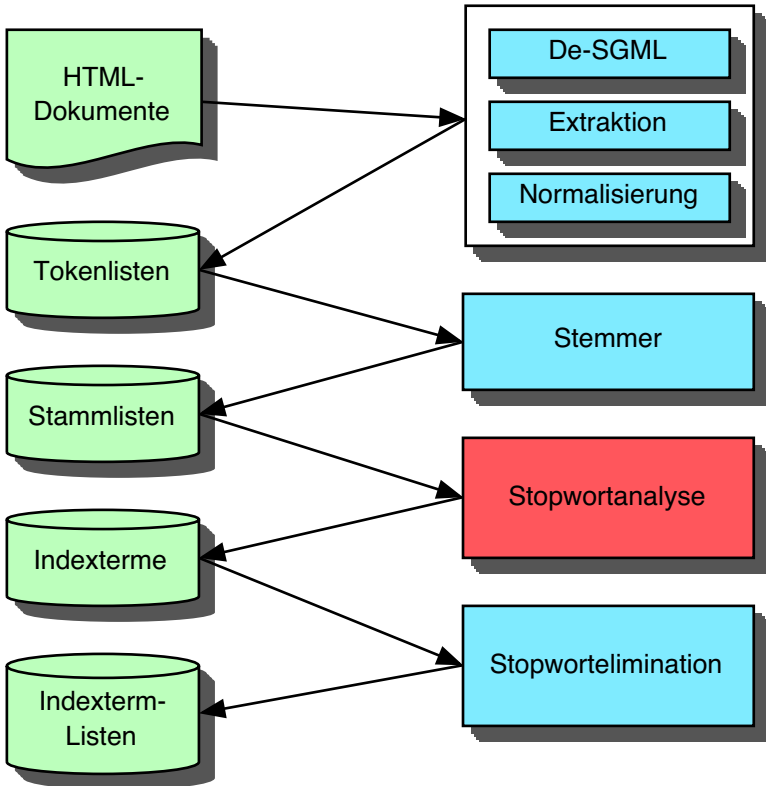
Stämme

James
Harris
General
Manag
des
Aladdin
Hotel-Resort
greift
zum
Handy
Hi
Howard
du

kann
uns
jetzt
das
Gewitt
schick
Ich
bin
mit
mein
Grupp
am
Haf

Und
schon
gehts
los
D
Himmel
verdunkelt
sich
Blitz
zuck
Donnergroll
Dann

Identifikation von Stopwörtern mit Hilfe der Informationstheorie



Identifikation von Stopwörtern mit Hilfe der Informationstheorie

- Der Korpus beinhaltet ca. 230.000 verschiedene Stämme.
- Die Anzahl der Indexterme sollte um Grössenordnungen kleiner sein, da die Lerner sonst praktisch nicht arbeiten.
- Viele Tokens haben für unseren Zweck keine Bedeutung.

Idee 1

In jedem Dokument wird der Autor angegeben, sodass in jedem Dokument ”von” vorkommt.

Annahme: ”von” kann nicht als ein Indiz für die Zugehörigkeit zu einer best. Dokumentenklasse dienen.

Wörter wie ”die”, ”und”, ”ein”, ”nicht”, ”an”, ... kommen sehr oft vor und haben für uns keinen Nutzen.

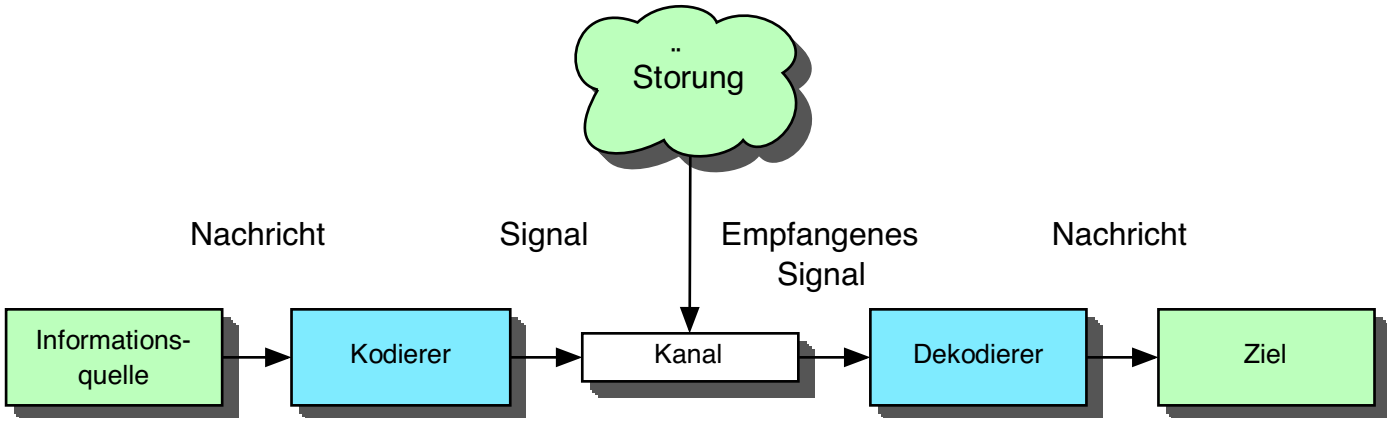
Annahme: Tokens, die in nur fast allen Dokumenten vorkommen, sind ebenfalls nicht geeignet.

Idee 2

- Wörter die nur einmal im Korpus vorkommen, sind gute Indikatoren, sie können nur leider fast nie verwendet werden.
- Der Korpus enthält über 100.000 Wörter, die nur einmal vorkommen!
- Beispiele: ”abba-Nachahmungstrauma”, ”abc-Abwehrfahig”, ”Abnickverein”, ”Raffgierkapitalismus”, ”Ökoromantik”, ”Zwitterwelt”, ”Zwiebelturmvilla”, ...

$\Rightarrow$ Aus Effizienzgründen will man diese Tokens alle loswerden.

Shannon’sche Informationstheorie: Model



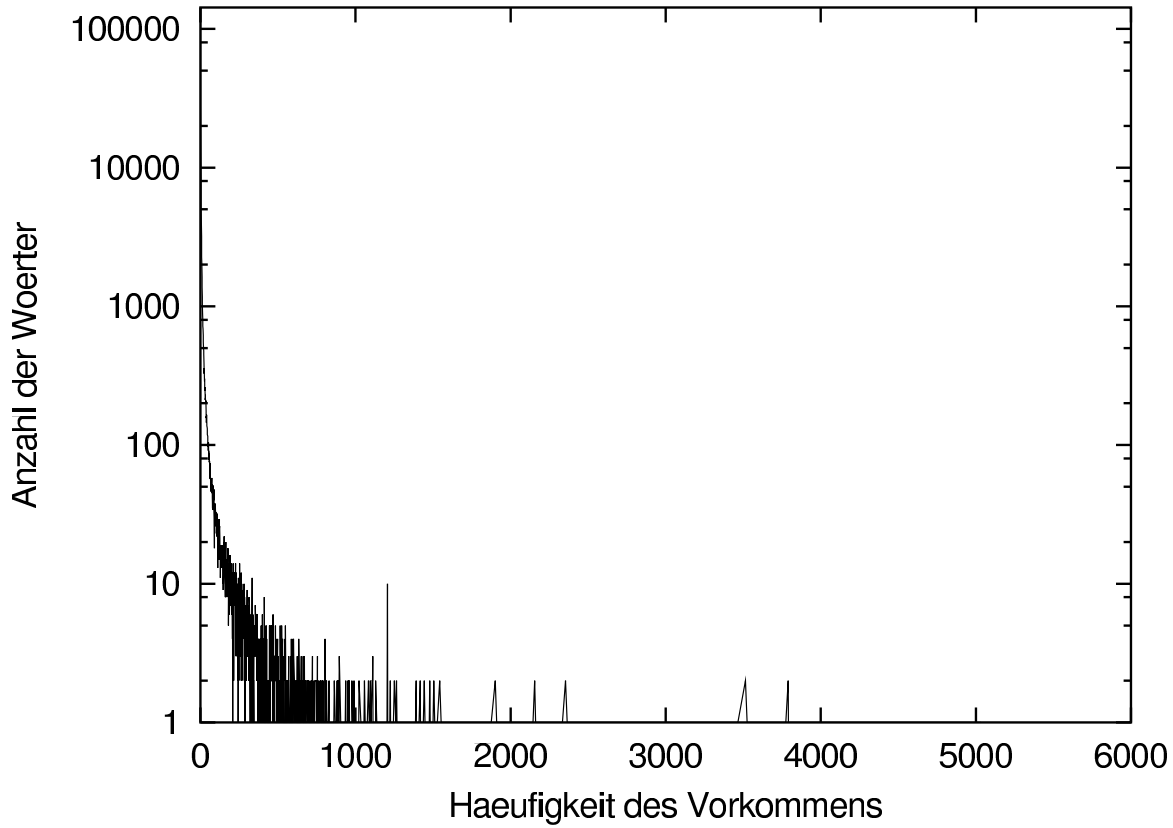
Information: Dokumentenklasse
Kodiert: Als ein Dokument

Informationsgehalt

- Haben alle Kodewörter ein gemeinsames Teilwort, so ist der Kode redundant.
- Der Informationsgehalt dieses Teilworts ist dann 0.
- Der Informationsgehalt ist gegeben durch

$$H(p_1, ..., p_n) = \sum_{i=1}^{n} p_i \log p_i$$

Schwellenwerte



Algorithmus

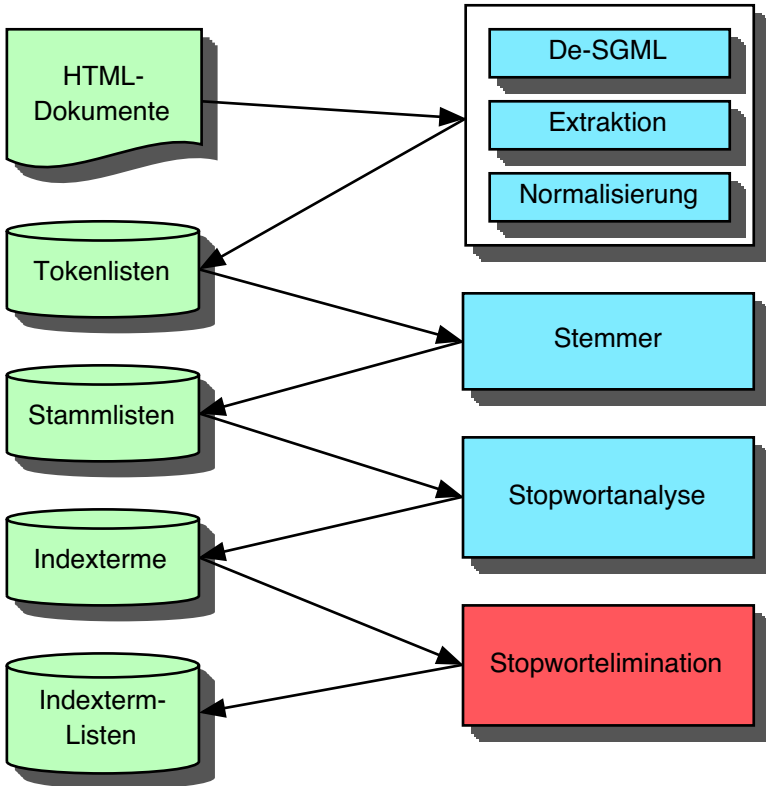
```
dict = {}
for d in documents:
  dhash = txt2dict(d)
  for w in dhash:
    dict[w] = dict[w] + 1

def txt2dict(d):
  dhash = {}
  for word in d:
    dhash[w] = 1
```

Eigenschaften

- Speicherverbrauch hauptsächlich von der Anzahl der versch. Wörter abhängig. Bei uns ca. 20 MB
- Komplexitätsklasse: $O(n)$
- Bei unserem Korpus 1-2 min

Elimination von Stopwörtern



Elimination von Stopwörtern

- Python-Programm liest Indexterme in Feld `iterms`

```
for d in documents:
  d1 = []
  for word in d:
    if word in iterms:
      d1.append(word)
  write(d1)
```

Indexterme

james
general
manag
greift
handy
du
schick
grupp
himmel
reg
aufs
aufs
beck

band
frisch
wind
kunstlich
erzeugt
Überrasch
gel
james
lachelt
innerhalb
nacht
las
gewalt

anstreng
bisher
herkomm
begriff
streck
jahrzehnt
sechs
kilomet
entstand
40
5000
bett
kunstlich

Ergebnisse 1

	Initial	Wortlisten	Stämme	Indexterme
Tokens	12.972.260	5.241.446	5.217.589	989.161
Types		313.085	231.160	3.056
KB	160.456	45.532	41.532	20.784

Ergebnisse 2

- Alle Verfahren sind $O(n)$.
- Der gesamte Korpus kann in wenigen Minuten verarbeitet werden.

KMeans

- Warum K-Means?
- Der K-Means-Algorithmus, wie wir ihn implementiert haben
- Ein Klassendiagramm
- Die verwendete Distanzfunktion

Warum K-Means?

- Die Means (Zentren) sind tatsächlich ein exaktes Mittel aller Vektoren eines Clusters, nicht nur einer dieser Vektoren

- k - Mindestens für den genetischen Algorithmus ist es von Vorteil, wenn man die Anzahl der Cluster festlegen kann. Zu wenige Cluster bringen keinen Lernerfolg, zu viele Cluster bringen den Algorithmus zum Stillstand. Ideal wäre bei knapp 4000 Termen ein k von etwa 100 bis 150.

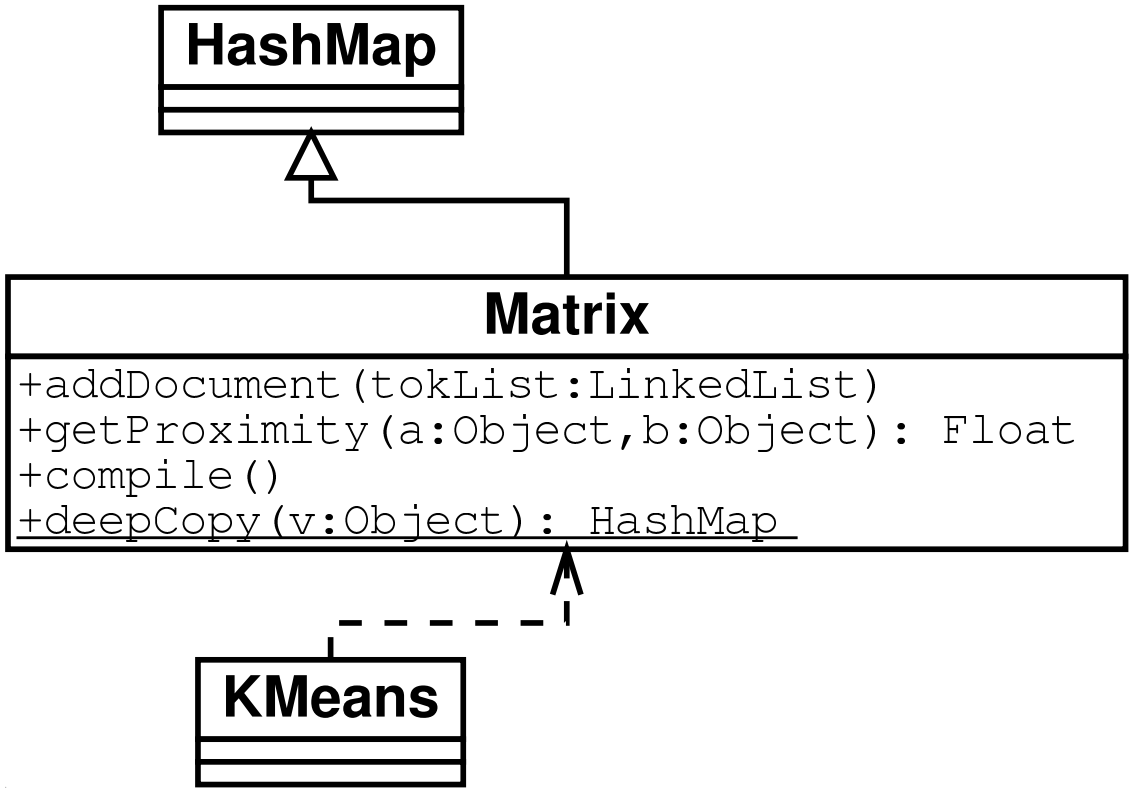
K-Means

Der Algorithmus

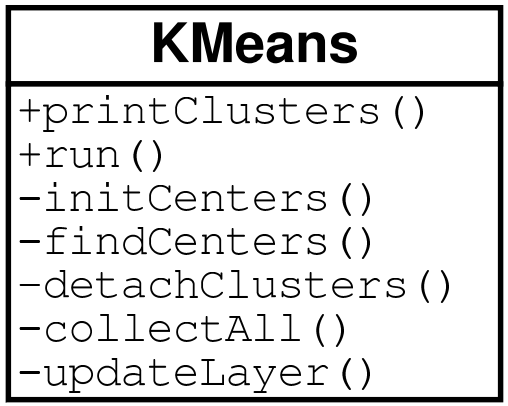
c: Cluster, z: Zentren, k: Anzahl der Cluster, m: Term-Vektor-Matrix

Initialisiere z mit k beliebigen Vektoren aus m
$c :=$ *Sammle um alle z die jeweils nächsten Vektoren aus m*
begin
  $old\_c := c$
  $c :=$ *Sammle um alle z die jeweils nächsten Vektoren aus m*
  Finde alle z für c
end while c != old_c

Klassendiagramm: Übersicht



K-Means


KMeans
+printClusters()
+run()
-initCenters()
-findCenters()
-detachClusters()
-collectAll()
-updateLayer()

Distanzfunktion (eigentlich eine Nähenfunktion)

$$\frac{2c}{a+b}$$

- c: Summe der gemeinsamen Häufigkeiten gemeinsamer Dokumente für beide Terme
- a: Summe der Häufigkeiten von Term eins in allen Dokumenten
- b: Summe der Häufigkeiten von Term zwei in allen Dokumenten
- Term 1: [1, .5, 0]
- Term 2: [1, 0, .2]

c = 1; a = 1.5; b = 1.2

Datenstrukturen in KMeans

Die verwendeten Datenstrukturen sind alle Hashes, das heißt:

- Der Zugriff auf Elemente geschieht in konstanter Zeit
- Der Speicherbedarf ist an der unteren Grenze des absolut Notwendigen

Anschließend an die Darstellung der Datenstrukturen folgt eine Darlegung der (praktischen) Gründe für die Entscheidung, Hashes zu verwenden.

Vektoren

`docID (Integer)` $\rightarrow$ `freq (Float)`

Vektoren mappen Dokumente zu Häufigkeiten. Dokumente werden über eine numerische ID vom Typ Integer indiziert, Häufigkeiten sind Fließkommazahlen, die die Anzahl der Vorkommen eines Tokens geteilt durch die Anzahl der interessanten Tokens eines Textes darstellen.

Beispiel:

```
{2 => 0.002345,
5 => 0.001987,
22 => 6.651562E-4,
...}
```

Matrix und Means

Matrix

`term (String)` → `vektor (HashMap)`

Die Matrix mappt einen Term auf einen Vektor.

Means (Zentren)

`centerID (Integer)` → `vektor (HashMap)`

Die Means mappen die numerischen IDs der Zentren auf Vektoren (die Zentren der Cluster).

Cluster

`term (String)` $\rightarrow$ `centerID (Integer)`

Die Cluster mappen Terme auf die ID des Zentrums, zu dem sie geclustert wurden.

Beispiel:

```
{"Pferd" => 0,
"Haus" => 1,
"Hund" => 0,
"Siedlung" => 1,
"Disneyland" => 2,
...}
```

Optimierung

- *Bottleneck*: Neuzuordnung der Cluster zu den Zentren (n Vektoren, m Dimensionen, z zugeordnete Vektoren):
  - pro Zuordnung eines Vektors zu einem Zentrum $(n - z) * m$ Vergleiche
  - n Schritte (pro Schritt z += 1)
  - Zeit-Korpus: (ca.) $n = 3000$, $m = 6000$
  - Pro Iteration: 27,009,000,000 (FlieSSkomma-)Vergleiche (27 Milliarden, wenn ich mich nicht irre)
- Mögl.: Reduktion der Dimensionen (LSI)
- Ansatz: Code-Optimierung

Optimierung: Hashes

- *Vektoren*: ursprgl. Arrays (Listen); Nachteile:
  - enorme Laufzeiten (weil die Listen nicht in konstanter Zeit indiziert werden und weil sie mit Nullen aufgefüllt werden müssen)
  - riesiger Speicherbedarf (nach 1500 Dokumenten über 512M)

- *Cluster*: ursprgl. Arrays (Listen); Nachteile:
  - Arrays müssen umgeordnet oder ab- und wiederaufgebaut werden
  - Arrays können nicht über die Terme indiziert werden (was beim Auffüllen der Cluster hilfreich ist)

Eigentlich. . .

Seien $X_1, ..., X_n$ die m-dimensionalen Termvektoren, dann ist das Ziel, Punkte $\{m_j \mid j = 1...k\}$ zu finden, sodass

$$\frac{1}{n}\sum_{i=1}^{n}(min_j d^2(X_i, m_j))$$

minimal wird.

Leider ist dieses Problem aus der Klasse der NP-vollständigen.

Deshalb...

k-means

- stellt eine einfach zu implementierende Approximation der Lösung dar,
- skaliert (relativ) gut,
- konvergiert schnell,
- konvergiert immer gegen ein lokales Minimum.

Leider..

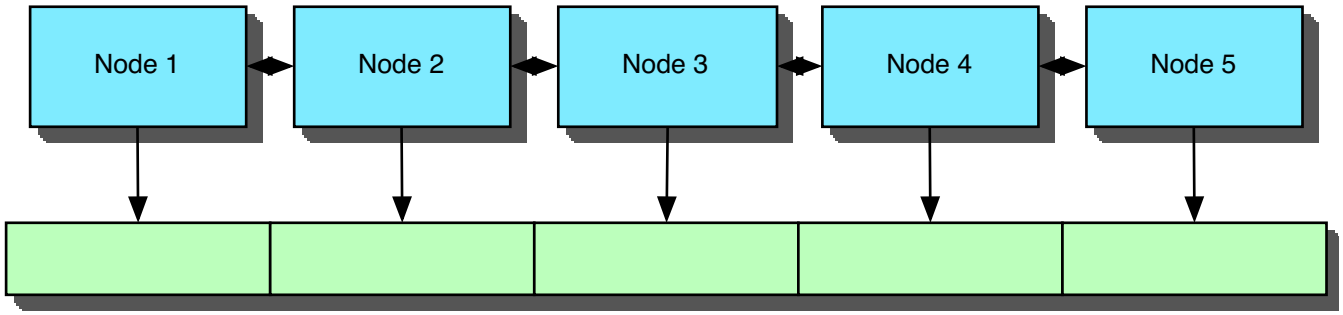
Die Komplexität von k-means ist $O(r * n * m * k)$, mit r = Anzahl der Iterationen, n = Anzahl der Dokumente, m = Anzahl der Terme, k = Anzahl der Cluster.

Für $r = 20$, $n = 5900$, $m = 3000$, $k = 20$ erhält man auf einer schnellen Workstation eine Laufzeit von mehreren Stunden.

k-means auf Parallelrechnern

- Inhärenter Datenparallelismus.
- Wenige Datenabhängigkeiten machen Parallelverarbeitung möglich.

Parallelcomputer-Setup



- Jeder Knoten operiert auf seine Teilmenge der Termvektoren.
- Wenig Kommunikation unter den Knoten.
- Skaliert gut und billig.

Genetische Algorithmen

- Überblick über genetische Algorithmen
- Einblick in die Funktionsweise
- Dokumentklassifikation mit GAs

Überblick

Hypothesen: GAs verwenden meist einfache Integers als Bitsets (bzw. Klassen wie die `std::bitset<int N>`-Klasse in C++), weil Operationen darauf sehr schnell sind

Lernverfahren: Gelernt wird nach dem Vorbild der natürlichen Evolution (Stichwort „Survival of the Fittest“)

Operationen: Teile der Population werden in jeder Iteration durch neue Hypothesen ersetzt, die durch genetische Operatoren (z. B. Mutation) gewonnen werden.

Hypothesen

- Bestehen aus Bitsets (Beispiel: 1001101)
- Bei binären Konzepten beschreibt das letzte Bit meist die Konzeptklasse (1 – true, 0 – false)
- Häufig repräsentiert der Bitstring eine Konjunktion von Konditionalen (z. B. Der Ball ist *rot* – 10 – und *rund* – 10 –, aber nicht *feuerfest* – 01 (und tatsächlich ein Ball – 1): 1010011)
- Disjunktionen sind schwierig darzustellen

Fitness-Funktion

- Beschreibt, ob ein Bitstring einen anderen, dessen Konzeptklasse bekannt ist, korrekt klassifiziert (bzw, ob ein Bitstring ein Konzept korrekt darstellt)

- Problemspezifisch: jedes Problem braucht seine eigene Fitnessfn. (es sei denn, es ist sehr simpel und kann etwas verallgemeinert werden)

Genetische Operatoren

- Wählen bestimmte Hypothesen aus und machen daraus neue Hypothesen (meist werden die „besten“ Hypothesen verwendet)

- Beispiele:

  Mutation: Kippt eine bestimmte Anzahl von Bits (häufig eins): 1010011 $\rightarrow$ 1011011

  Crossover: Nimmt Teile zweier Bitstrings und kombiniert sie neu: **1010011** und *1101011* $\rightarrow$ *11***1001**1 (yyxxxxz)

Dokumentklassifizierung

- Konjunktion je zweier Bits pro Cluster:

  10 Enthält einen Term aus dem Cluster
  01 Enthält keinen Term aus dem Cluster
  11 Kann einen Term aus dem Cluster enthalten

- Beispiel: 10 01 11 1 – muß einen Term aus Cluster eins enthalten, darf keinen Term aus Cluster 2 enthalten, Cluster 3 kann ignoriert werden

Literatur

1. Ricardo Baeza-Yates. *Modern Information Retrieval*. Addison-Wesley, New York, 1999.

2. Katharina Morik, Stefan Wrobel, Thorsten Joachims. *Maschinelles Lernen und Data Mining*. In: G. Görz, C. Rollinger, J. Schneeberger. *Einführung in die Künstliche Intelligenz*. 2000. Quelle: http://www.cl.uni-heidelberg.de/kurs/ss01/ml/

3. Tom M. Mitchell. *Machine Learning*. McGraw-Hill,

4. Robert B. Ash. *Information Theory*. New York, 1965. Singapur, 1997.