Mitschrift Information Retrieval

Vorwort:

Leider haben die meisten LVAs an der TU Wien ein riesiges Problem mit den Folien und den Vorträgen, worüber auch viele meiner Mitstudierenden seit Jahren klagen. Oft werden wichtige Informationen einfach ausgelassen und Zusammenhänge nicht erklärt. Das ist besonders nervig, wenn es nur Folien zu lernen gibt und keinerlei Videoaufzeichnungen oder Skripten. Wenn es mal Skripte gibt, dann sind sie sehr pseudointelligent geschrieben um zu beweisen wie elitär man nicht sei, was den Studierenden aber leider alles andere als hilft.

Diese Mitschrift hat daher das Ziel die Themen der LVA „Grundlagen des Information Retrieval“ in etwas einfacheren Worten zu erklären, ohne dabei wichtige Details auszulassen, und Zusammenhänge in Themen und Unterthemen aufzuzeigen. In dieser Mitschrift mache ich das konkret indem Unterkapitel nummeriert sind aber auch indem ich im Text auf andere Punkte verweise und Übergänge einbaue. Zumindest das mit der Nummerierung hätte man zwar auch gleich in den Folien machen können, aber was weiß ich schon - ich bin ja nur ein(e) StudentIn!

Da die Folien an einigen Stellen leider (ganz im Stil der TU Wien) unverständlich waren, (besonders ohne Vorlesungsaufzeichnung) habe ich auch viele Informationen aus anderen Quellen wie Youtube, Blogs, etc. bezogen. Aber keine Sorge, Telegram wurde nie als Quelle benutzt!

Achtung:

- das komplette Kapitel „Lecture 6 - Statistical Semantics and Explicit Semantics“ ist mit Vorsicht zu genießen, da die Folien hier besonders wirr und unverständlich waren!
- Mean-Average-Precision ebenso, da die verwendeten Beispiel nicht eindeutig den Ablauf darstellen (meiner Meinung nach), weil wie funktioniert das ganze wenn es 5 relevante Dokumente gibt aber nur 3 davon retrieved wurden? Hier am besten in der Vorlesung explizit nachfragen!

Mini-Appell an alle Personen die Folien oder Skripte für Studierende erstellen (liest vermutlich eh niemand aber why not): Informatik ist schon alleine schwer genug, daher wäre es für Studierende von Vorteil wenn Zusammenhänge klar aufgezeigt werden und es eine Art „roten Faden“ gibt - also „Was ist das Problem & wie löst man es?“ oder „Was ist der Jetzt-Stand & und wie verbessert man ihn?“, „Wie hängt das mit anderen Unterkapitel zusammen?“, „Was ist das Bigger Picture?“. Oft ist es aber so, dass einfach mal irgendwelche Algorithmen und Tools auf die Studierenden geworfen werden und die Frage nach dem „Wozu?“ und dem „Bigger Picture“ vernachlässigt wird. Kleine Änderungen würden die Situation schon deutlich verbessern wie: Änderung der Reihenfolge von Unterkapitel, Nummerierung von Unterkapiteln, eindeutigere Beispiele, zusätzliche trickreichere Beispiele, das „Bigger Picture“ erklären, einfachere Sprache wählen bei komplexen Themen, etc. Das Argument „ja es ist halt eine Universität“ wirkt immer mehr wie eine faule Ausrede.

1. Lecture 1 - Foundations of Information Retrieval

Beim Information Retrieval was wir in der Vorlesung behandeln geht es vor allem um Text als Eingabe bzw. Daten. Der Text hat dabei einige Eigenschaften & Herausforderungen.

- Text muss keine Bedeutung haben. So verstehen die wenigsten von uns vermutliche diesen Text: ฉันมรีถสแีดง
- Text beinhaltet aber viel Information, selbst wenn wir nicht wissen was er bedeutet. Zb sehen wir, dass dieser Text ฉันมรี ถสฟี ้า sehr viel mehr mit dem obigen Text gemeinsam hat als folgender Text: คณุ มรี ถไฟฟ้ า
- Text ist nicht zufällig: „*I drive a red car*“ ist sehr viel wahrscheinlicher als:
  - *„I drive a red horse“*
  - *„A red car I drive“*
  - *„Car red a drive I“*
- Bedeutung erfolgt über Benutzung:
  - Aus folgenden Phrasen lässt sich herleiten, dass *truck, car* und *bus* eine ähnliche Bedeutung haben
  - *„I drive a truck“*
  - *„I drive a car“*
  - *„I drive the bus“*

1.1. Begriffe

Die wichtigsten Werkzeuge für Information Retrieval sind Tokenizer, Stemmer, Lemmatizer, Index, Gewichtung und Ähnlichkeits-Funktionen.

1.1.1. Tokenizer & Tokens:

Bei der Tokenization wird Text in einzelne Tokens geteilt. Ein Token ist dabei eine Sequenz aus Characters. Außerdem werden bestimmte Characters oder Tokens verworfen, wie z.B Punkte, Apostroph und Stopwords wie „and“, „or“, „a“, etc. (je nach Einstellung des Tokenizers).

Beispiel:

Satz: „Friends, Romans and Countrymen“

Tokenization: | Friends | Romans | Countrymen |

1.1.1.1. Probleme bei Tokenization

Bei der Tokenization gibt es einige Herausforderungen. Wichtig zu merken ist dabei bei allen Herausforderungen, dass es nie die eine richtige Lösung gibt, sondern alles auf den Einzelfall ankommt und besonders auf die Sprache mit der man Arbeiten muss. So gibt es unterschiede zwischen Deutsch und Englisch oder Probleme die in der einen Sprache existieren, die es in der anderen aber nicht gibt.

Komposita:

Ein klassisches Beispiel ist das aneinanderketten von Nomen, welches in der Deutschen Sprache ein besonderes Merkmal ist und bei der Tokenization beachtet werden muss. Wenn man nur mit englischen Texten arbeitet, ist dies aber völlig egal. Bei Deutschen IR Systemen kann ein Compound Splitter Modul einen Performanceboost von 15% geben!

Apostroph:

Ein weiteres Problem stellen Apostroph dar. Was soll mit Apostroph passieren? Beispiel: „Finland's Capital". Soll es zu „Finland", „Finlands" oder „Finland's" werden?

Hyphens:

Worte die mit Bindestrich verbunden sind ebenfalls spannend. Bsp: „Hewlett-Packard". Sollen „Hewlett" und „Packard" eigene, getrennte Tokens sein? Was ist mit Wörtern bzw Phrasen wie „state-of-the-art"? Was ist mit unterschiedlichen Schreibweisen wie „lowercase" vs. „lower-case" vs. „lower case"? Wie sollen diese verarbeitet werden, sodass in einem Text zB. sowohl „lowercase", „lower-case" als auch „lower case" gefunden wird wenn man nur „lowercase" in die Suche eingibt? Schließlich will man ja alle passenden Ergebnisse finden!

Städte:

Soll „San Francisco" nur 1 Token sein oder 2? Wie entscheidet man das? Soll zB auch „San Marino" oder „San Diego" gefunden werden wenn man nach „San" sucht? All diese Entscheidungen spielen eine Rolle bei der Tokenization!

Nummern / Daten:

Weitere Herausforderungen stellen Daten (3/20/91 vs 20/3/91 vs Mar. 20, 1993), Keys, Telefonnummern und Fehlercodes dar. Ältere Information Retrievel Systeme verarbeiten Nummern garnicht. Dabei wäre es aber sehr hilfreich Fehlercodes auch zu indexieren, also zu verarbeiten, um Lösungen zu Problemen finden zu können.

Chinesisch / Japanisch:

In der Chinesischen und Japanischen Sprache gibt es keine Abstände zwischen Wörtern! Wie soll hier die Tokenization stattfinden? Hier ist es sogar so, dass nicht immer eine einzigartige Tokenization garantiert werden kann.

Arabisch / Hebräisch:

In diesen Sprachen wird von rechts nicht links geschrieben, aber Zahlen von links nach rechts!

1.1.2. Stop words

Stop words sind Wörter wie: the, and, a, to, be. Diese Wörter haben wenig semantischen Kontext und machen etwa 30% des Dokuments aus. Daher wurden sie früher bei der Tokenization aus dem Text entfernt.

Der Trend geht heutzutage aber da hin, Stop words nicht zu entfernen. Einerseits weil es im Gegensatz zu früher, heute gute Kompressionalgorithmen und gute Query Optimization gibt. Andererseits werden Stop words benötigt um Phrasen und Titel wie „King of Denmark“, „Let it be“, „To be or not to be“ oder Relationen zu erkennen wie „Flights to London“.

1.1.3. Type

Ein Type sind alle Tokens, welche die selbe Sequenz and Characters enthalten. So sind „Peter“, „Lisa, „U.S.A“ und auch „USA“ verschiedene Types. Jetzt ist es aber so, dass man vermutlich sowohl „U.S.A.“ als auch „USA“ in einem Dokument finden möchte, egal ob in der Query Punkte verwendet wurden oder nicht. Hier kommen die sogenannten „Terms“ ins Spiel!

1.1.4. Terms

Die Terms sind normalisierte Types, sodass „U.S.A.“ und „USA“ auf den selben Term gemappt werden. Diese Terms werden in das sogenannte Dictionary aufgenommen.

1.1.5. Normalization

Die Normalisierung wurde schon weiter oben kurz angesprochen. Damit in der Suche Ergebnisse gefunden werden, müssen die Tokens aus der Query die Tokens aus der Tokenlist des Dokuments matchen. Dh die Tokens im Index und in der Query müssen ggf. normalisiert werden. Zb damit eben „U.S.A“ und „USA“ gematched werden. Ansonsten würde „U.S.A“ nur gematched werden, wenn sowohl in der Query als auch im Dokument „U.S.A“ stünde. Andere Dokumente die nur „USA“ enhalten, würden dann nicht gematched werden!

Das Resultat dieser Normalisierung nennt sich wie gesagt „Term“. Dieser Term ist ein Eintrag in unserem IR Dictionary. Das Normalisieren geschieht zB über das Entfernen von Punkten oder Hyphens, oder das Umwandeln in Kleinbuchstaben, uvm.

Sehr wichtig bei der Normalisierung ist, dass sowohl die Query als auch der Index normalisiert werden muss! Ansonsten ist logischerweise kein Match möglich!

Beispiele:

U.S.A, USA -> USA

anti-discriminatory, antidiscriminatory -> antidiscriminatory

1.1.5.1.1. Herausforderungen

Wie auch bei der Tokenization gesprochen gibt es auch bei der Normalisierung Herausforderungen, allen voran bzgl. der Sprache.

- **Accents:** résumé vs resume
- **Umlaute:** Tübingen vs Tuebingen
- **Daten**

Hier ist es wichtig zu beachten, wie die User am wahrscheinlichsten ihre Queries schreiben. Es hat sich herausgestellt, dass sogar User die Accents oder Umlaute in ihrer (Mutter)sprache verwenden, diese aus Faulheit nicht unbedingt verwenden! So wird aus „Tübingen“ zb „Tuebingen“ oder sogar „Tubingen“.

1.1.5.2. Case Folding:

Case Folding bezeichnet die Strategie, alle Großbuchstaben in Kleinbuchstaben umzuwandeln. Das erlaubt es beispielsweise, dass auch Sätze die mit „Car“ (Achtung große Schreibweise wegen Satzanfang) beginnen, eine Query mit „car“ matchen. Außerdem ist es oft so, dass User zu faul sind Großbuchstaben zu setzen. Die Suchergebnisse wären ohne Casefolding also sehr schlecht.

1.1.6. Stemming und Lemmatization

Aufgrund von Grammatikregeln haben Wörter oft unterschiedliche Formen. Zb. „organize“, „organizes“, „organizing“. Das Grundlegende Wort ist aber das selbe, nämlich „organize“. Ein weiteres Beispiel wäre „am“, „are“, „is“. Das Grundlegende Word ist hier „be“.

Ähnlich ist es bei ableitungsverwandte Wörter. „Derivationally related words: Terms in different syntactic categories that have the same root form and are semantically related.“

Das soll nichts anderes heißen als, dass z.B. „democracy“ (Nomen), „democratic“ (Adjektiv), „democratize“ (Verb), „democratization“ (Nomen?) alle eine ähnliche semantische Bedeutung haben (anders als zB. „television“ und „telescope“).

Das Ziel von Stemming und Lemmatization ist es, diese unterschiedlichen Formen und eines Wortes auf ein Basiswort zu reduzieren, sodass sinnvolle Matches möglich sind. Wichtig anzumerken ist, dass beide Sprachen- und Anwendungsspezifisch sind!

Weitere Beispiele:

- am, are, is => be
- car, cars, car's, cars' => car
- the boy's cars are different colors => the boy car different color

1.1.6.1. Stemming

Stemming ist ein grober heuristischer Prozess bei dem die Enden von Wörtern einfach abgehackt werden in der Hoffnung, dass somit die Basisform erreicht wird. Es werden dabei oft Ableitungs-Affix entfernt.

Der bekannteste Algorithmus diesbzgl. für die englische Sprache ist der sogenannte „Porter's Algorithm". Für die deutsche Sprache gibt es den „German Snowball Stemmer". Es gibt aber noch weitere Stemmer wie zB. den Lovins Stemmer oder Paice/Hask Stemmer.

1.1.6.2. Lemmatization

Die Lemmatization ist im Gegensatz zum Stemming ein etwas anspruchsvollerer, komplexerer Ansatz, bei dem Vokabular und Morphologische Analyse verwendet wird um die Basisform zu erreichen. Dabei werden Flexionsenden (also -ing, -ed, -s, -ies, -er, -est) entfernt.

Näheres zu „Flexionsenden": https://www.theedadvocate.org/inflected-endings-everything-you-need-to-know/

1.2. Viele Features, und jetzt?

All diese Features [TODO: FEATURES KONKRET NENNEN] können verwendet werden, um schnell Dokumente zu finden, (semantische) Ähnlichkeiten zwischen Texten zu berechnen. Weiters können damit auch Dinge gelernt werden wie zB. Satzstrukturen oder generelles NLP (Natural Language Processing) gemacht werden.

1.3. Simple Search Engine

Eine Search Engine benötigt einen Index. Das Ziel vom Indexieren ist es Informationen so zu speichern, dass schnelle Abfragen möglich sind. Im folgenden wird eine sehr banale Implementierung für eine Search Engine erklärt, welche schnell auf Probleme stößt.

Hierzu wird eine Matrix aus den Dokumenten und den Termen erstellt. Wenn ein Term in einem Dokument vorkommt, so bekommt er an der entsprechenden Stelle in der Matrix eine 1, ansonsten eine 2. Mithilfe von bitweisen Verknüpfungen, lassen sich so dann alle Dokumente finden, welche ein besuchten Term enthalten.

Das ganze lässt sich aber besser mit einem Beispiel zeigen! Nehmen wir an wir wollen Theaterstücke finden können indem wir explizit angeben, welche Rollen darin vorkommen oder nicht vorkommen dürfen. In unserem konkreten Beispiel sollen alle Theaterstücke gefunden werden, in denen Brutus und Caesar vorkommen, aber nicht Calpurnia.

Nur um die Fachtermini zu verwenden: Unsere Dokumente sind also Theaterstücke und unsere Terme sind die Charaktere. Die Query ist unsere Forderung, welche Charaktere vorkommen oder nicht vorkommen sollen.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	1	1	0	0	0	1
Brutus	1	1	0	1	0	0
Caesar	1	1	0	1	1	1
Calpurnia	0	1	0	0	0	0
Cleopatra	1	0	0	0	0	0
mercy	1	0	1	1	1	1
worser	1	0	1	1	1	0

Brutus** AND **Caesar** BUT NOT **Calpurnia

1 if play contains word, 0 otherwise

Die Terme haben also einen Vektor mit binären Werten 0 oder 1, je nachdem ob der Term im Dokument vorhanden ist oder nicht. In anderen Worten, je nachdem ob ein Charakter im Theaterstück vorkommt oder nicht.

Somit ergeben sich folgende Vektoren für Brutus, Caesar und Calpurnia:

Brutus	110100
Caesar	110111
Calpurnia	010000

Um nun herauszufinden welche Dokumente die Query erfüllen, werden die Vektoren bitweise AND-verknüpft. Da wir aber in der Query ein „BUT NOT“ für Calpurnia haben, müssen wir diesen einen Vektor erst negieren. Somit bekommen wird folgende Vektoren heraus:

Brutus	110100
Caesar	110111
Calpurnia	100100

Nun können wir die bitweise AND-Verknüpfung durchführen:

- 110100 *AND*
- 110111 *AND*
- 101111 =
- **100100**

Die Theaterstücke die also unsere Query erfüllen sind „Antony and Cleaoprata“ und „Hamlet“.

1.3.1. Problem: Skallierbarkeit

Wie sicher leicht zu erkennen ist, skaliert diese Methode nicht besonders gut! Bei 1.000.000 Dokumenten mit jeweils 1000 Wörtern wird das offensichtlich zu einem Problem, denn selbst wenn es „nur“ 500.000 einzigartige Wörter gäbe, so hätte man eine 500K x 1M Matrix was immer noch viel zu riesig ist!

Deshalb gibt es die Invertierte Matrix bzw. den Invertierten Index!

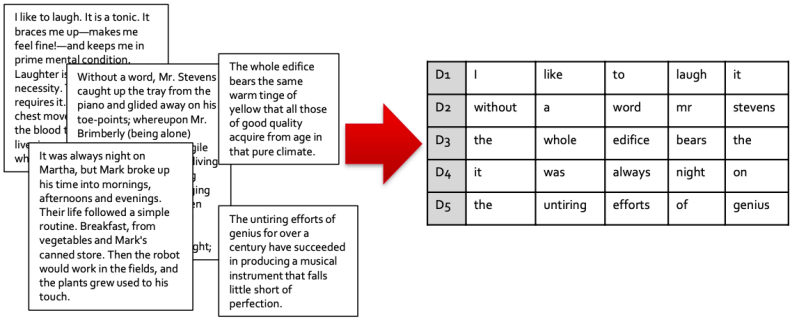
1.4. Inverted Index

Beim Information Retrieval geht es darum Dokumente aufgrund von bestimmten Wörtern zu finden, und das möglichst schnell. Da der Inverted Index genau das ermöglicht, ist er bei Information Retrieval System der Standard. Der Inverted Index ist so gut, weil er Wörter auf Dokumente mappt anstatt umgekehrt.

Das Gegenteil des Inverted Index wäre der sogenannte Direct Index (auch Forward Index genannt). Beim Forward Index wird ein Mapping von Dokumente auf Wörter erstellt. Wenn ich in einem Forward Index also suchen will welche Dokumente das Wort „Avocado“ beinhalten, müsste ich alle Dokumente durchgehen und prüfen ob das Wort Avocado darin vorkommt. Das wäre äußerst ineffizient!

Beim Inverted Index ist das Mapping wie gesagt umgekehrt, also Wörter werden auf Dokumente gemappt. Dadurch ist es sehr schnell möglich herauszufinden welches Wort in welchen Dokumenten enthalten ist!

Im folgenden sieht man die Konzepte eines Direct Index und eines Inverted Index gegenübergestellt.



Konzept des Direct Index

Inverted Index

- Default index structure in Information Retrieval
- Computationally very efficient. Scales well
- Words are sorted alphabetically to speed up access
- Frequency of a word in a document can also be stored

I	D1
like	D1
to	D1
laugh	D1
it	D1, D4
without	D2
a	D2
word	D2
mr	D2
stevens	D2
the	D3, D5
whole	D3
edifice	D3
bears	D3
was	D4
always	D4
night	D4
on	D4
untiring	D5
efforts	D5
of	D5
genius	D5

atics

Konzept des Inverted Index

For each term *t*, we must store a list of all documents that contain *t*.

- Identify each doc by a **docID**, a document serial number

Brutus	1	2	4	11	31	45	173	174
Caesar	1	2	4	5	6	16	57	132
Calpurnia	2	31	54	101				

Inverted Index mit Dictionary & Postings

1.4.1. Inverted Index - genauer + Beispiel

Gute Erklärung: https://www.youtube.com/watch?v=1g00SP56iSE

Den Invertierten Index wird am besten anhand eines Beispiels erklärt!

Dokument 1: Norway borders Sweden. Norway is to the west of Sweden.

Dokument 2: Magnus Carlsen is a chess player from Norway. He is the world chess champion.

Schritt 1: Tokenization (& Case folding, etc.)

1: [norway, borders, sweden, is, to, the, west, of]

2: [magnus, carlsen, is, a, chess, player, from, norway, he, the, world, champion]

Schritt 2: Liste aller Tokens (auch Duplikate) mit zugehöriger Dokument Id erstellen.

Schritt 3: Liste aus Schritt 2 alphabetisch sortieren.

Schritt 4: Invertierten Index mithilfe der Liste aus Schritt 3 erstellen.

Der Invertierte Index besteht aus 2 Teilen - dem sogenannten „Dictionary“ und den „Postings“. Das Dictionary ist eine Liste welche alle Terme aus allen Dokumenten enthält (keine Duplikate) und zu jedem Term eine Info wie oft er insgesamt über alle Dokumente vorkommt (=„document frequency“). Die Postings speichern, welches Wort in welchen Dokumenten enthalten ist.

Die Document Frequency stellt übrigens ein wichtiges Attribut dar, welches für einige Algorithmen bzw. Berechnungen verwendet wird, doch mehr dazu in einem späteren Kapitel.

Dictionary		Postings List
Term	**Doc. Freq.**	**Postings**
a	1	[2]
borders	1	[1]
carlsen	1	[2]
champion	1	[2]
chess	2	[2]
from	1	[2]
he	1	[2]
is	2	[1,2]
magnus	1	[2]
norway	2	[1, 2]
of	1	[1]
player	1	[2]
sweden	1	[1]
the	2	[1, 2]
top	1	[2]
west	1	[1]
world	1	[2]

Nun sind zB. Boolean Queries möglich! Die Document Frequency hilft hier sogar bei der Query Optimierung, denn die maximale Anzahl der Dokumente, welche eine AND Boolean-Query erfüllen, ist die Länge der kleinsten Postings List!

1.4.2. Inverted Index - Boolean Query Beispiel:

Dictionary **Postings List**

Term	Doc. Freq.	Postings
norway	2	[1, 2]
champion	1	[2]

Wir starten bei „champion“, weil die Document Frequency hier am niedrigsten ist. Wir sehen in der Postings List für „champion“, dass der Term als nächstes im Dokument 2 vorkommt.

Deshalb schieben wir den Index in den anderen Postings List (in unserem Bsp für „norway“) so lange nach rechts, bis wir die Nummer 2 treffen. Wenn die Nummer 2 in allen anderen Posting Lists getroffen wird, können wird Dokument 2 in unsere Ergebnisliste hinzufügen. Wenn die Nummer 2 in einer Postings List nicht vorkommt, können wir natürlich ausschließen, dass Dokument 2 die Query erfüllt!

Nun versuchen wir in der Postings List von „champion“ einen Index weiter zu gehen. Da wir aber sehen, dass die Liste zu ende ist, brauchen wir die anderen Postings Listen (in unserem Fall nur „norway“) nicht weiter zu analysieren. Wir wissen, dass nur in Dokument 2 die beiden Terme „norway“ und „champion“ vorkommen!

1.5. Phrase Queries

2. Lecture 3 - Scoring Documents

Es sind logischerweise nicht alle Dokumente einer Suche gleich relevant. Als User will man die relevantesten Dokumente als erstes sehen. In diesem Kapitel geht es darum, wie man die Relevanz eines Dokuments festlegen kann.

2.1. Retrieval Models

Retrieval Models sind mathematische Modelle bzgl. der Relevanz. So sind beim Boolean Model, welches wir bereits kennengelernt haben, alle Treffer gleich relevant. Beim Vector Space Model oder dem Probability Model ist es hingegen aber möglich die Relevanz eines jeden Dokumentes festzustellen.

2.2. Boolean Retrieval

Im ersten Kapitel haben wir Boolean Retrieval kennengelernt. Hierbei wird eine Boolean Query verwendet um Dokumente zu finden. Boolean Retrieval ist sehr präzise, denn entweder matched ein Dokument eine Query oder nicht. Daraus ergeben sich Vor- und Nachteile. Auch wenn Boolean Searches oft verwendet werden und definitiv ihren Nutzen haben, gibt es je nach Anwendungsfall aber Probleme!

Zum einen sind Boolean Queries nicht intuitiv! Es benötigt etwas Übung und man muss sich mit der Syntax auskennen um überhaupt Queries schreiben zu können. Wenn man sich nicht gut auskennt hat man das Problem, dass man entweder zu wenige oder zu viele Ergebnisse findet, was beides eher unbrauchbar ist!

Zum anderen gibt es keine Möglichkeit für ein Ranking - also festzustellen wie relevant Dokumente sind (das haben wir ja vorhin gerade schon gehört). Das erste Ergebnis könnte also auch völlig unbrauchbar sein. Als Kontrast: bei Google zB ist das erste Ergebnis das relevanteste! Bei einer Boolean Search sind außerdem alle Terme von gleicher Wichtigkeit, wodurch es nochmal schwieriger ist, die Relevanz festzustellen.

2.3. Ranked Retrieval

Ranked Retrieval soll das Problem mit der Relevanz lösen. Hierbei wird ein Algorithmus verwendet, welcher die Relevanz der Dokumente im Bezug zur aktuellen Query feststellt. Das Ergebnis ist eine Liste der Dokumente nach Relevanz sortiert. Das relevanteste Dokument ist logischerweise das erste.

Die Query ist außerdem keine Boolean Query sondern Freitext - also ein oder mehrere Wörter in ganz normaler menschlicher Sprache.

2.4. Term Frequency (TF) & TF-Weighting

https://www.youtube.com/watch?v=9UXM2NXVYY0

Damit die Relevanz eines Dokuments wird logischerweise unter anderem durch dessen Inhalt bewertet - also die Terme. Diese Terme können mit verschiedenen Gewichtsfunktionen gewichtet werden, um ihnen mehr oder weniger Relevanz zu geben, was sich dann letztendlich auf die Relevanz des Dokuments für die Query auswirkt. Eine dieser Gewichtsfunktionen ist das Term Frequency Weighting.

Die „raw term frequency“ zählt einfach wie oft ein bestimmter Term in einem bestimmten Dokument enthalten ist. Wenn der Term „squirrel“ in einem Dokument also 10 Mal vorkommt, dann ist die „raw TF“ für dieses eine Dokument 10.

Wieso sollte die raw TF gezählt werden? Die Idee ist dass, ein Dokument in dem der term t 10 Mal vorkommt, vermutlich relevanter als ein Dokument in dem der selbe Term t nur 1 Mal enthalten ist.

Allerdings ist es auch schlecht die absolute Anzahl einfach als Indiz für Relevanz zu nehmen, denn das Wort „the“ kommt beispielsweise sehr häufig vor. Soll heißen: Wenn ein Wort zu oft vorkommt ist es vermutlich eher unwichtig! Deshalb ist die Lösung nicht die „raw TF“, also keine lineares, sondern ein logarithmisches Model.

Wenn in einem Dokument ein Wort also 10-Mal vorkommt ist es zwar wichtiger als ein Dokument in dem es nur 1-Mal vorkommt, aber nicht 10-Mal so wichtig! Wenn in einem Dokument ein Wort 30 Mal vorkommt so ist es aber nicht 3-Mal relevanter als das Dokument in dem das wort „nur“ 10 Mal vorkommt, sondern maximal gleich relevant. Daher eben der Logarithmus!

$$TF\ \text{weight} = \log(1 + tf_{t,d})$$

Der Nachteil des TF-Weighing ist, dass seltene Terme keine große Bedeutung zugeschrieben wird. Das ist aber ein Problem, da seltene Terme in Wirklichkeit eine große Bedeutung haben können! Dieses Problem löst die Inverse Document Frequency (IDF).

2.5. Inverse Document Frequency (IDF)

Die Inverse Document Frequency ist ebenfalls eine Methode zur Gewichtung. Die Idee hierbei ist, dass seltene Terme vermutlich eine höhere Bedeutung haben. Zb „arachnocentric“ VS „increase“ oder „line“, etc. Ein Dokument welches also ein sehr seltenes Wort wie „arachnocentric“ enthält ist vermutlich sehr relevant für eine Query die „arachnocentric“ enthält. Daher wollen wir eine hohe Gewichtung für solch seltene Terme!

Für Terme die etwas öfter vorkommen wie „increase“ oder „line“ wollen wir zwar trotzdem eine Gewichtung, aber nicht so eine hohe wie bei „arachnocentric“. Die Inverse Document Frequency macht genau das!

Die Formel der IDF lautet:

$$idf_t = \log({}^{N}/_{df_t})$$

df_t… Anzahl an Dokumente welche den Term t enthalten

N…Anzahl aller Dokumente

Der Logarithmus wird als Dämpfung genutzt, da der „raw value“ zu stark wäre (wie bei TF)

2.5.1. IDF Beispiel

term	df_t	idf_t
calpurnia	1	log(1,000,000/1) = 6
animal	100	4
sunday	1,000	3
fly	10,000	2
under	100,000	1
the	1,000,000	0 (1,000,000/1,000,000)

idf = log (N/df)

2.6. TF-IDF Weighting

https://www.youtube.com/watch?v=4-P3ckZprBk

https://mtham8.github.io/tf-idf/

https://ishwortimilsina.com/vector-space-model/

https://www.learndatasci.com/glossary/tf-idf-term-frequency-inverse-document-frequency/

TF-IDF ist eine weitere Methode zu Gewichtung und setzt sich aus dem Produkt von TF und IDF zusammen. Wenn man also sowohl die TF-Weighting als auch das IDF-Weighting kennt, kann man das TF-IDF-Weighting berechnen!

$$tf\text{-}idf_{t,d} = \log(1 + tf_{t,d}) \times \log({}^{N}/_{df_t})$$

increases with the number of occurrences within a document

increases with the rarity of the term in the collection

Im unteren Beispiel hat das Dokument „Julius Caesar“ für die Gewichtung der Terme den Vektor <3.18, 6.1, 2.54, 1.54, 0, 0, 0>. Die Terme sind Antony, Brutus, Caesar, Calpurnia, Cleoprata, mercy, worser.

	Antony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
Antony	5.25	3.18	0	0	0	0.35
Brutus	1.21	6.1	0	1	0	0
Caesar	8.59	2.54	0	1.51	0.25	0
Calpurnia	0	1.54	0	0	0	0
Cleopatra	2.85	0	0	0	0	0
mercy	1.51	0	1.9	0.12	5.25	0.88
worser	1.37	0	0.11	4.15	0.25	1.95

Ergebnis nach der TF-IDF Gewichtung, aber ohne Length Normalization.

Das TF-IDF Weighting hat aber einen großes Problem - je länger ein Dokumente, desto mehr Terme enthält es und desto höher wird der Score des Dokuments! Das hat zur Folge, dass länger Dokumente eher einen höheren Score bekommen als kürzere, selbst wenn einige der kürzeren Dokumente tatsächlich relevanter wären! In einem IR-System in dem alle Dokumente etwa die gleiche Länge haben, ist das zwar kein Problem, aber bei allen anderen schon.

2.7. Vector Space Model (VSM) und Cosine Similarity

http://mlwiki.org/index.php/Cosine_Similarity

https://www.youtube.com/watch?v=o5nflzfX5tw

Bisher haben wir viele Gewichtungen kennengelernt - jetzt kommen wir endlich dazu, wie man die Relevanz eines Dokuments feststellen kann. Das Vector Space Model ist eine Möglichkeit die Relevanz eines Dokuments bzgl einer Query festzustellen. Die generelle Idee ist, dass die Dokumente und die Query als Vektoren in einem Vektorraum abgebildet werden. Wenn ein Dokument der Query ähnlich ist, so schauen ihre Vektoren in eine ähnliche Richtung. Die Formel dafür heißt auch „Cosine Similarity“.

Wie weiter oben besprochen sind Gewichtungen sinnvoll um festzustellen wie relevant die einzelnen Terme sind. Typischerweise wird für das VSM die TF-IDF Gewichtung verwendet. Daher werden die Vektoren mit TF-IDF gewichtet. Es existieren auch noch andere Gewichtungen, aber die schauen wir uns im Zuge der Lehrveranstaltung nicht an! Wenn man im VSM die TF-IDF Gewichtung verwendet so spricht man übrigens vom „TF-IDF VSM“.

Wir haben aber erst erfahren das TF-IDF ein großes Problem hat - längere Dokumente bekommen tendenziell einen höheren Score. Um diesen Effekt zu unterbinden bzw. zu minimieren benötigen wir eine Length Normalization. Diese Normalisierung ist in der Cosine Similarity enthalten (Division durch Länge der Vektoren).

Die Formel der Cosine Similarity, mit welcher der Winkel zwischen der Query q und einem Dokument d festgestellt wird lautet:

$$\cos(\vec{q},\vec{d}) = \frac{\vec{q}\cdot\vec{d}}{|\vec{q}||\vec{d}|} = \frac{\vec{q}}{|\vec{q}|}\cdot\frac{\vec{d}}{|\vec{d}|} = \frac{\sum_{i=1}^{|V|} q_i d_i}{\sqrt{\sum_{i=1}^{|V|} q_i^2}\sqrt{\sum_{i=1}^{|V|} d_i^2}}$$

Dot product

Unit vectors

q… Query mit Terme als Vektor

d… Dokument mit Terme als Vektor (typischerweise mit TF-IDF Gewichtet, siehe obiges oder unteres Beispiel)

2.7.1. Beispiel (ohne TF-IDF Weighting):

Query: can my cat eat chicken

Vektor: [1, 1, 1, 1, 1]

Dokument 1: A cat can eat chicken. Chicken is part of a cat's diet. <= relevant document

Vektor: [1, 0, 1, 1, 1]

Score: 4

Dokument 2: The cat ran past the chicken to try to eat my mouse. <= irrelevant document

Vektor: [0, 1, 1, 1, 1]

Score: 4

2.7.2. Beispiel (mit TF-IDF Weighting & Length Normalization):

Query: can my cat eat chicken

Vektor: [1, 1, 1, 1, 1]

Dokument 1: A cat can eat chicken. Chicken is part of a cat's diet.

Vektor: [0.2886751345948129, 0, 0.1716274399381882, 0.1716274399381882, 0.24271785323595957]

Score: 0.8746478677071488

Dokument 2: The cat ran past the chicken to try to eat my mouse.

Vektor: [0, 0.2886751345948129, 0.1716274399381882, 0.1716274399381882, 0.1716274399381882]

Score: 0.8035574544093774

2.8. Probabilistic Model

https://www.brainkart.com/article/Retrieval-Models_11608/

https://www.youtube.com/watch?v=auJh-SbYOlM

Beim VSM kann es zu False-Positives kommen. Deshalb gibt es das Probabilistic Model. Dieses Model verwendet Methoden aus der Statistik um Dokumente in „relevant“ und „non-relevant“ einzuordnen. Es ist ein Ranking basierend auf der **Wahrscheinlichkeit** dass ein Dokument **relevant** ist oder nicht. Also das Dokument das am wahrscheinlichsten als relevant eingestuft wird, ist auf Platz 1.

Es gibt daher 2 Sets - ein „relevant“ Set „ und ein „non-relevant“ Set. Die Aufgabe besteht nun darin die Wahrscheinlichkeit festzustellen, dass ein Dokument ins relevant Set passt und die Wahrscheinlichkeit festzustellen, dass ein Dokument ins non-relevant Set passt. Diese beiden Wahrscheinlichkeiten werden berechnet und miteinander verglichen, um dann letztendlich die Entscheidung zu treffen in welches der beiden Sets ein Dokument eingeordnet wird. Das wird für alle Dokumente gemacht.

Formal gesagt: Beim Probabilistischen Model versucht man herauszufinden wie Wahrscheinlich es ist, dass ein Dokument *d* für eine Query *q* relevant ist, basierend auf einer Datengrundlage *data*. Die Datengrundlage kann alles mögliche sein, dass dem IR-System zur verfügung steht, also Query *q*, Dokument *d*, User Context, User Profile, Logdaten, etc.

$p(d,q) = P(relevance(d, q) = 1 \mid data)$

Es gibt viele verschiedene Probabilistische Modelle. Sie Unterscheiden sich danach wie sie versuchen die Wahrscheinlichkeit der Relevanz eines Dokuments abzuschätzen. Im Zuge der Lehrveranstaltung sehen wir uns die beiden Probabilistischen Modelle BIM und BM25 grob an. Beide basieren auf dem Probability Ranking Principle (PRP).

2.8.1. Probability Ranking Principle (PRP)

Das PRP von Robertson aus 2009 grob übersetzt ins Deutsche: „Wenn man Dokumente absteigend nach ihrer Wahrscheinlichkeit an Relevanz sortiert, basierend auf einer Datengrundlage, dann ist es das Effektivste was man für das System mit den Daten machen kann.“

In anderen Worten: „Wenn ich eine Datenmenge habe aus der ich ableiten kann was User gerne machen, usw. Dann ist nach der Wahrscheinlichkeit zu ranken das beste was ich überhaupt machen kann.“

2.8.2. Die Mathematik dahinter (nicht aus Folien sondern von brainkart.com)

D… Dokument

R… Relevance

NR… Nonrelevance

Bedingte Wahrscheinlichkeit P(R|D) und P(NR|D)

$P(R|D) = P(D|R) \times P(R)/P(D)$

$P(NR|D) = P(D|NR) \times P(NR)/P(D)$

Ein Dokument wird als relevant klassifiziert, wenn P(R|D) > P(NR|D). Das ist Equivalent mit:

$P(D|R) \times P(R) > P(D|NR) \times P(NR)$

Der Likelidhoodquotient (engl. likelihood ratio) P(D|R)/P(D|NR) wird als Score verwendet um festzustellen ob ein Dokument zum relevanten oder irrelevanten Set gehört.

2.8.3. Binary Independence Model (BIM)

https://www.youtube.com/watch?v=auJh-SbYOlM

Im BIM haben Query und Dokumente einen Vektor welcher mit 1 und 0 gefüllt ist. Wenn ein Term aus dem Dokument in der Query vorkommt wird er im Vektor des Dokuments durch eine 1 repräsentiert, ansonsten durch eine 0 - deshalb spricht man auch von einem Binären Model.

Beispiel: x = [0, 1, 0, 0, 0, 1, …, 1]

Verschiedene Dokumente können daher zufällig den selben Vektor haben!

Query: „my cat ate my sandwich“

Document 1: „my cat ate my homework“ => [1, 1, 1, 1, 0]

Document 2: „my cat ate my grapes“ => [1, 1, 1, 1, 0]

Außerdem sind die Terme „Independent“. Soll heißen dass es zwischen einzelnen Terme keine Verbindung gibt. „Martin Luther King“ sind also 3 eigenständige Terme die nichts miteinander zu tun haben.

2.8.3.1. BIM - Relevanz einschätzen

Die Dokumente haben natürlich erstmal keine Infos über die Relevanz. Um diese festzustellen zu können ist es notwendig herauszufinden wie die Terme des Dokuments zur Relevanz beitragen. So ist es wichtig messbare Statistiken zu finden wie z.B. die Term Frequency, Document Frequency, Document Length, etc. Diese Statistiken können verwendet werden um abzuschätzen wie Wahrscheinlich es ist, dass ein Dokument relevant ist.

Die Dokumente können dann absteigend nach ihrer Wahrscheinlichkeit für Relevanz ausgegeben werden. Dabei muss angenommen werden, dass die Relevanz eines jeden Dokuments unabhängig von der Relevanz aller anderen Dokumenten ist. Diese Vermutung stimmt aber leider nicht - mehr dazu später!

2.8.3.2. Grenzen von BIM

BIM wurde ursprünglich dafür entwickelt einen überschaubaren Katalog an Dokumenten mit etwa gleicher Länge zu durchsuchen. Es wurde nach Titel oder Abstracts gesucht. In diesem Kontext funktioniert BIM auch wunderbar! Für eine moderne Volltextsuche ist BIM aber ungeeignet!

Für eine Volltextsuche gibt es daher als Probalistisches Model das BM25, welches seit 1994 eines der verbreitetsten und robustesten IR-Systeme ist.

2.8.4. (Okapi) BM25

http://ctp.di.fct.unl.pt/~jmag/ir/slides/a06%20Probabilistic%20retrieval%20models.pdf

https://www.elastic.co/de/blog/practical-bm25-part-3-considerations-for-picking-b-and-k1-in-elasticsearch

BM25 ist ebenfalls ein Probalistisches Model welches aber für Volltextsuche verwendet wird. Im Gegensatz zu BIM werden die Terme nicht in Vektoren binär repräsentiert, also „kommt vor“ / „kommt nicht vor“, sondern es wird mit der Term Frequency (TF), Inverse Document Frequency (IDF) und der Document Length (DL) gearbeitet. Das ist auch der größte Unterschied zu BIM **(Prüfungsrelevant)**!

BM25 ist also ein Model welches TF, IDF & DL berücksichtigt, was für eine moderne Volltextsuche sehr wichtig ist. Die TF wird nämlich verwendet um die Terme gewichten zu können (was auch ein Unterschied zu BIM ist).

2.8.4.1. BM25 - Formel

Die Formel für den Retrieval Status Value (RSV) eines Dokuments, also wie relevant das Dokument ist, lautet:

$$RSV_d = \sum_{t \in q} \log \left[\frac{N}{\mathrm{df}_t} \right] \cdot \frac{(k_1 + 1)\mathrm{tf}_{td}}{k_1((1-b) + b \times (L_d / L_{\mathrm{ave}})) + \mathrm{tf}_{td}}$$

In der Vorlesung gab es ein obligatorisches Rechenbeispiel, bei dem aber ganz im Stil der TU einige hilfreiche Rechenschritte ausgelassen wurden, um die Formel auch tatsächlich richtig verstehen zu können - einfach herrlich!

Stattdessen sehen wir uns lieber ein paar Parameter und ihre Bedeutung bzw. Auswirkung an.

$\boldsymbol{k_1}$ kontrolliert die **Skalierung der TF**. Wenn k_1 auf 0 gesetzt ist haben wir ein binäres Model. Wenn man sich die Formel von oben ansieht und k_1 auf 0 setzt ist der Bruch dann immer tf_{td}/tf_{td} was 1 ergibt (oder undefined bei 0/0, keine Ahnung was dann passiert). Somit hat man nur mehr $log(N/df_t)$ in der Summe.

b kontrolliert hingegen die **Normalisierung der Document Length**. Wenn *b* auf 0 gesetzt wird haben wir keine Length Normalization.

Typischerweise werden standardmäßig $\boldsymbol{k_1}$ zwischen **1.2** und 2, und ***b*** auf **0.75** gesetzt

2.9. Zusammenfassung (laut Folien)

Probabilisitc Models sind in der Theorie sauberer, aber in der Praxis nicht unbedingt, weil es einige Annahmen bzgl der Relevanz gibt.

Es gibt zwei Fundamentale Fragen die man sich stellen muss:

- Wie erscheinen Terme im Dokument?
- Wie sind diese relevant?

Binary Independence Model (BIM)

BM25

3. Lecture 4 - IR Evaluation

https://www.cl.cam.ac.uk/teaching/2006/InfoRtrv/lec3.2.pdf

Wenn wir eine Search Engine bzw. ein IR System haben, würden wir vielleicht gerne wissen wie gut es ist. Damit beschäftigt sich dieses Kapitel!

Evaluation ist wichtig, weil es ohne ihr keine Forschung gibt und auch keine Möglichkeit festzustellen wie gut ein IR System ist. Die IR Evaluation ist sogar ein eigenes Forschungsfeld! Einerseits weil es viele verschiedene IR Systeme gibt, die alle unterschiedliche Kriterien zur Evaluation benötigen und andererseits weil die Evaluation schwierig ist. Die Evaluation ist schwierig wegen der riesigen Menge an Dokumenten und wegen menschlicher Subjektivität, bzgl der Relevanz dieser Dokumente.

3.1. Arten der Evaluation

Es gibt verschiedene Arten der Evaluation. Diese sind:

- **Effizienz:** „Wie schnell bekomme ich Ergebnisse?“. Ist auch abhängig von Spezifikationen die für das IR-System gesetzt werden. Also soll es Ergebnisse in Real-Time liefern oder nicht, Wie groß ist der Index des Systems? Der Index hat auch Einfluss auf die Geschwindigkeit, etc. Die Fragen bzw. Ziele bzgl Effizienz sind allerdings recht einfach zu definieren!
- **Effektivität:** „Bekomme ich hoch-relevante Dokumente als Ergebnisse?“. Schwer zu bestimmen, denn woher soll ich wissen ob ein Dokument relevant für meine Query ist? Das festzustellen ist tatsächlich recht aufwendig! Doch wenn ich nicht weiß wie wichtig ein Dokument für eine bestimmte Query ist, kann ich auch nicht herausfinden ob mein IR-System qualitativ hochwertige Ergebnisse liefert!
- **User Studies:** beinhaltet viele Usability Issues. Ist recht schwierig durchzuführen und Ergebnisse zu bekommen, weshalb diese Tests nicht gerne gemacht werden.
- **Intrinsic:** ???
- **Extrinsic:** ???

3.2. Was wird gemessen?

Es gibt verschiedene Eigenschaften die gemessen werden können. Diese sind:

- **Coverage:** „Wie viele relevante Information deckt mein System ab?“
- **Time Lag:** Effizienz. „Wie schnell bekommt man Ergebnisse?“
- **Presentation:** (vermutlich Usability?? Es steht nichts genaues in den Folien)
- **Effort:** „on the part of the user to answer his/her information need“ (1:1 aus Folie kopiert)
- **Recall:** Wie viele der Dokumente aus unserer Datenbank die relevant sind wurden ausgegeben?“
- **Precision:** „Wie viele der ausgegebenen Dokumente sind relevant?“

3.3. Messung der Effizienz:

Die Messung der Effizienz ist ziemlich straight forward, da es ja nur um Geschwindigkeit und Platz geht. Gemessen wird:

- **Elapsed index time:** Measures the amount of time necessary to build a document index on a particular system.
- **Indexing processor time:** Measures the CPU seconds used in building a document index. This is similar to elapsed time, but does not count time waiting for I/O or speed gains from parallelism.
- **Query throughput:** Number of queries processed per second.
- **Query latency:** The amount of time a user must wait after issuing a query before receiving a response, measured in mil-liseconds. This can be measured using the mean, but is often more instructive when used with the median or a percentile bound.
- **Indexing temporary space:** Amount of temporary disk space used while creating an index.
- **Index size:** Amount of storage necessary to store the index files.

3.4. Messung der Effektivität (Retrieval Effectiveness Evaluation)

Die Messung der Effektivität ist sehr viel komplexer! Zur Erinnerung: Es geht darum zu bewerten ob ein IR-System auch tatsächlich für den User relevante Ergebnisse liefert. Wenn ein System bei einer Query 10 Ergebnisse liefert und nur 3 davon sind überhaupt für die Query relevant, so hat das System keine gute Effektivität!

Im Zuge der Vorlesung sehen wir verschiedene Möglichkeiten an die Effektivität zu messen. Diese Methoden sind unter anderem Precision & Recall, E-Meassure, F-Meassure, Precision-Recall-Curve, Mean Average Precision (MAP), Graded Relevance (CG, DCG, NCG).

3.4.1. Precision & Recall

Precision und Recall sind Metriken um die Effektivität eines IR Systems festzustellen. Wenn ein User nach etwas sucht könnte ein Dokument in 1 von 4 Kategorien eingeteilt werden:

- Relevant and retrieved (wünschenswert)
- Relevant and not retrieved (nicht wünschenswert)
- Non-relevant and retrieved (nicht wünschenswert)
- Non-relevant and not-retrieved (wünschenswert)

Die relevanten Dokumente sind jene, die dem User helfen seine Fragen zu beantworten. Im Idealfall werden alle relevanten Dokumente geliefert, das ist aber höchst unwahrscheinlich. In den meisten Fällen bekommt man sowohl relevante als auch irrelevante Dokumente als Ergebnis.

Precision ist das Verhältnis zwischen der Anzahl an relevanten Dokumenten aus unserem Ergebnis und der Gesamtanzahl an Dokumenten aus unserem Ergebnis.

Soll heißen: „Wie viele der ausgegebenen Dokumente sind relevant?“

Beispiel: Wir suchen nach „Can my cat eat grapes?“ und bekommen 10 Dokumente zurück. 6 davon sind relevant für unsere Query. Dh wir haben eine Precision von 0.6 (6 / 10).

Recall ist das Verhältnis zwischen der Anzahl an relevanten Dokument aus unserem Ergebnis und der Gesamtanzahl an relevanten Dokumenten aus der Datenbank.

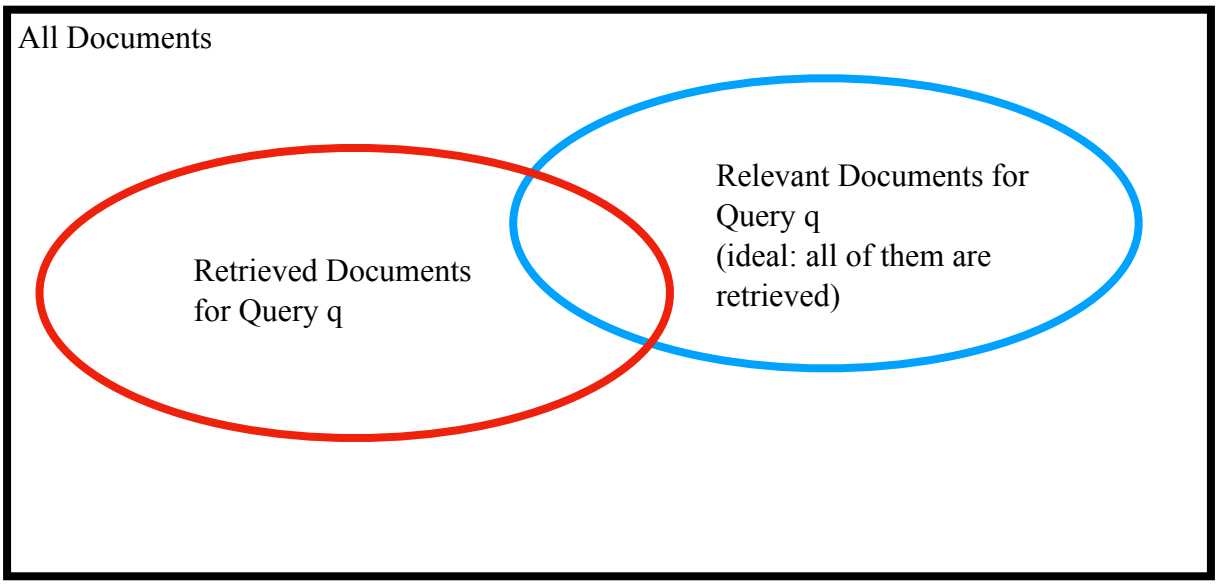
Soll heißen: „Wie viele der Dokumente aus unserer Datenbank die relevant sind wurden ausgegeben?“

Beispiel: Wir suchen nach „Can my cat eat grapes?“ und bekommen 10 Dokumente zurück. 6 davon sind relevant für unsere Query. Aus unserer Datenbank sind 100 von 1000 Dokumenten relevant für diese Query. Dh wir haben ein Recall von 0.06 (6 / 100).

3.4.1.1. Recall & Precision - Break-Even-Point

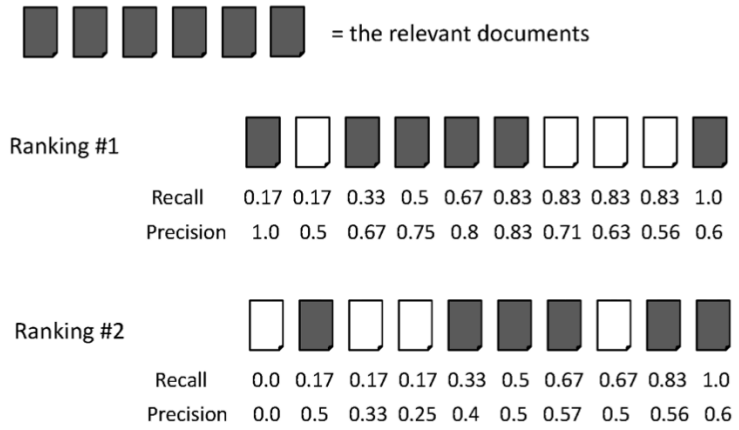
…

3.4.1.2. Erklärung mit Visualisierung:



Die obige Abbildung soll zeigen, dass nicht alle returnierten Dokumente auch tatsächlich relevant für eine Query q sind. Wie in der Abbildung zu sehen ist, sind einige der returnierten Dokumente irrelevant für den User. Man könnte die Abbildung als „Soll-Ist“ Vergleich interpretieren. Soll: Blau, Ist: Rot

3.4.1.3. Recall & Precision Beispiel aus Folien



Im Bild sind zwei Beispiele zur Berechnung von Precision und Recall zu sehen - Ranking #1 und Ranking #2. Das Beispiel ist recht vereinfacht, weil die Datenbank nicht besonders groß ist für die Query alle wichtigen Dokumente aus der Datenbank geliefert werden. Somit ist Recall immer 1.

Precision und Recall werden nach und nach mit jedem returnierten Dokument berechnet. Je nachdem ob es sich um ein relevantes oder irrelevantes Dokument handelt, steigt bzw. sinkt der Wert.

3.4.2. E-Measure & F-Measure

Precision und Recall sind zwei Metriken um die Retrieval Effectiveness zu messen. Aber was ist wenn man die Retrieval Effectiveness nur an bloß 1 Zahl festlegen möchte und nicht an 2 Dafür gibt es die F-Measure. Die F-Measure basiert übrigens auf die E-Measure von Van Rijsbergen. Allerdings sind einige der Meinung, dass es besser ist Precision und Recall separat anzugeben, anstatt F anzugeben, weil sonst die Kenntnis über Recall & Precision verloren geht.

$$E = 1 - \frac{1}{a\frac{1}{precision} + (1 - a)\frac{1}{recall}}$$

$$F = \frac{2 \times precision \times recall}{precision + recall}$$

3.4.3. Precision-Recall-Curve

https://scikit-learn.org/stable/auto_examples/model_selection/plot_precision_recall.html

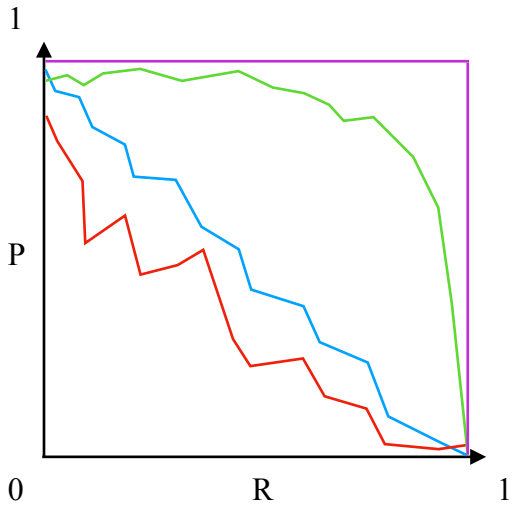
https://www.geeksforgeeks.org/precision-recall-curve-ml/

Die Precision Recall Curve zeigt den Tradeoff zwischen Precision und Recall für unterschiedliche Thresholds.

Um solch eine Kurve plotten zu können benötigen wir:

- Ein Set an Dokumenten
- Ein Set an Queries
- Für jede Query eine Entscheidung welche Dokumente relevant bzw. irrelevant sind.

Hohe Precision bedeutet wenig False Positives und eine hoher Recall bedeutet niedrige False Negatives. Der Idealfall ist also eine hohe Precision und ein hoher Recall, wie wir schon gehört haben. Eine größere Fläche der Kurve bedeutet also höhere Precision & Recall. Im unteren Beispiel ist also die grüne Kurve am besten und die rote am schlechtesten. Die violette Kurve stellt den Idealfall dar, welcher aber wahrscheinlich niemals erreicht werden wird.



3.4.3.1. Wie wird eine P-R-Curve erstellt?

https://demonstrations.wolfram.com/ThePrecisionRecallCurveInInformationRetrievalEvaluation/

https://medium.com/@douglaspsteen/precision-recall-curves-d32e5b290248

Wir wissen wie Recall und Precision berechnet wird (siehe „3.4.1.3 Recall & Precision Beispiel aus Folien“). Die Idee ist nun, dass für mehrere Queries eine P-R-Curve erstellt wird. Das Mittel aus all diesen Queries ist die P-R-Curve des IR Systems.

Die P-R-Curves unterschiedlicher IR Systeme können nun miteinander verglichen werden um festzustellen welches das beste ist.

3.4.3.2. Reality Check

3.4.3.3. When to stop retrieving?

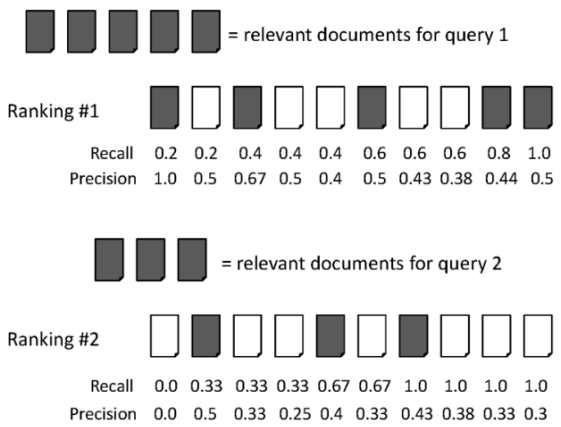
…

3.4.4. Mean Average Precision (MAP)

Wir kennen jetzt schon einige Metriken um IR-Systeme zu bewerten. Manchmal benötigen wir aber eine Metrik mit nur einer Zahl (im Vergleich zu Precision-Recall mit 2 Zahlen). Zum Beispiel wenn wir Systeme miteinander Vergleichen wollen oder ein Script haben, mit dem wir Parameter fein einstellen (tunen) wollen aufgrund von „Wie gut funktioniert unser IR-System?“. Das Script könnte zum Beispiel nicht den kompletten Precision-Recall Graph interpretieren, sondern braucht 1 konkrete Zahl! Genau für solche Fälle gibt es die Mean Average Precision (MAP).

Die MAP ist also eine weitere Metrik um die Precision eines IR-System zu bekommen und wird tatsächlich sehr häufig in Papers verwendet um IR-Systeme zu vergleichen. Anders als bei Precision & Recall, hat man bei MAP nur 1 Zahl anstatt 2.

Die MAP zu berechnen ist sehr einfach! Es muss lediglich der Mittelwert der Average Precision aller Test-Queries berechnet werden. Am besten veranschaulichen wir das an einem Beispiel:



Um die MAP zu bekommen müssen wir zunächst also die **Average Precision jeder Query** berechnen:

Average Precision Query 1: (1.0 + 0.67 + 0.5 + 0.44 + 0.5) / 5 = 0.62
Average Precision Query 2: (0.5 + 0.4 + 0.43) / 3 = 0.44

Jetzt können wir die MAP berechnen:

MAP: (0.62 + 0.44) / 2 = 0.53

3.4.5. Graded Relevance

Bisher haben wir die Dokumente aus den Suchergebnissen nur in „relevant“ und „nicht relevant“ geteilt (quasi binär). Nun ist es aber so, dass manche Dokumente zwar durchaus relevant sind, aber es andere Dokumente gibt die noch viel relevanter sind! Mit der bisherigen Bewertung ist es aber nicht möglich das auszudrücken! Dafür gibt es die Graded Relevance! Anstatt also einem Dokument aus den Suchergebnissen „relevant“ (1) oder „irrelevant“ (0) zuzuweisen, wird ein relativer Wert zugewiesen. Je höher der Wert, desto relevanter ist das Dokument für die Query.

Wichtig anzumerken ist, dass es hier nicht darum geht ein Scoring für Dokumente zu implementieren um dem User gute Antworten auf die Query zu geben (darum ging es ja im vorherigen Kapitel), sondern das Ziel ist es die Qualität eines IR System festzustellen, also wie akkurat bzw. hilfreich die Ergebnisse für eine Query sind!

Es gibt verschiedene Möglichkeiten die Graded Relevance zu implementieren. Im Zuge der Vorlesung sehen wir uns die Methoden CG, DCG und NDCG an.

3.4.5.1. Cumulative Gain (CG)

Das Cumulative Gain ist eine Implementierung der Graded Relevance. Anstatt also wie bisher einem Dokument aus den Suchergebnissen „relevant“ (1) oder „irrelevant“ (0) zuzuweisen, wird ein relativer Wert zugewiesen. Je höher der Wert, desto relevanter ist das Dokument für die Query.

Beispiel für eine Search Result List mit 10 Dokumenten: <5, 2, 4, 5, 5, 1, 0, 2, 4, 0>

Der Name „Cumulative Gain“ kommt daher, dass aus der Relevanz dieser Dokumente die Summe gebildet wird. Das Ergebnis ist das Cumulative Gain CG. Die Formel zur Berechnung des CG an einer Position *p* lautet:

$$\mathrm{CG}_p = \sum_{i=1}^{p} rel_i$$

In unserem Beispiel also: $CG_{10} = 5+2+4+5+5+1+0+2+4+0 = 28$

Der Rang des Dokuments, also die Position in der Search Result List, wird für die Berechnung der Qualität dabei nicht berücksichtigt! Wieso das schlecht ist wird im Anschluss beim Discounted Cumulative Gain (DCG) besprochen.

3.4.6. Discounted Cumulative Gain (DCG)

Wenn wir ein Ranked Retrieval Model verwenden, so werden die Dokumente nach Relevanz absteigend sortiert ausgegeben. Soll heißen, dass das erste Dokument in der Search Result List das wichtigste ist. Was ist aber wenn das bei unseren IR System aber nicht der Fall ist?

Wenn also zum Beispiel die Dokumente mit Graded Relevance in folgender Reihenfolge ausgespielt werden: <2, 5, 5, 4, 4, 3, 0, 5>

Das Dokument mit der Graded Relevance von 2 wird hier als erstes ausgespielt und an letzter Stell ist sogar ein Dokument mit der Graded Relevance von 5! Das ist nicht wünschenswert! Die Ideale Reihenfolge wäre also <5, 5, 5, 4, 4, 3, 2, 0>.

Beim CG wird auf die Richtigkeit der Reihenfolge nicht geachtet! Das Ergebnis ist am Ende unabhängig von der Reihenfolge der Dokumente. Die Idee von Discounted Cumulative Gain (CG) ist es daher, eine falsche Reihenfolge zu bestrafen.

Der Discounted Cumulative Gain hat 2 Annahmen (in Englisch Wiedergegeben, weil Prüfungsrelevant):

- Highly relevant documents are more useful than marginally relevant
- The lower the ranked position of a relevant document, the less useful it is for the user, since it is less likely to be examined

Wir wird eine falsche Reihenfolge nun konkret bestraft? Ganz einfach indem die Graded Relevance durch den Logarithmus der Position dividiert wird.

$$\mathrm{DCG_p} = \sum_{i=1}^{p} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{p} \frac{rel_i}{\log_2(i+1)}$$

Das hört sich vielleicht etwas komisch an aber ich versuche es zu erklären. Der Logarithmus liefert anfangs sehr geringe Zahlen aber wird nach und nach immer größer ($\log_2(2) = 1$, $\log_2(3) = 1.5$, …, $\log_2(5) = 2.3$, … $\log_2(10) = 3.3$). Somit ist eine Division durch diesen anfangs auch nicht, gegen Ende hin aber schon. Wenn die relevanten Dokumente also am Anfang stehen dann wird ihr Wert durch relativ kleine Zahlen dividiert und die Summe bleibt groß. Wenn die relevanten Dokumente aber hingegen am Ende stehen, dann wird ihre Wert durch eine relativ hohe Zahl dividiert. Das hat dann einen schlechten Einfluss auf die Summe!

3.4.6.1. DCG Beispiel

Wir bekommen mit einer Query folgende Search Result List: <D1, D2, D3, D4, D5, D6> mit den Graded Relevances <3, 2, 3, 0, 1, 2>

Daraus ergeben sich für unsere DCG-Formel folgende Werte (Rundungsfehler möglich):

i	rel_i	$log_2(1 + i)$	$rel_i / log_2(1 + i)$
1	3	1	3
2	2	1.58	1.26
3	3	2	1.5
4	0	2.32	0
5	1	2.58	0.38
6	2	2.80	0.71

Der DCG berechnet sich daher wie folgt: DCG = 3 + 1.26 + 1.5 + 0 + 0.38 + 0.71 = 6.85

3.4.7. Normalised Discounted Cumulative Gain (nDCG)

Leider können weder CG noch DCG benutzt werden um die Performance verschiedener Queries miteinander zu vergleichen, da die Search Result Lists nicht immer garantiert die selbe Länge haben! Deshalb muss der Cumulative Gain normalisiert werden. Daher der Name Normalised Discounted Cumulative Gain (nDCG).

Um den nDCG einer Query zu berechnen muss zuerst die Ideal DCG (IDCG) berechnet werden. Das ist nicht anders als der DCG bei dem die Dokumente nach der Graded Relevance sortiert sind. Danach wird der nDCG einer Query wird berechnet indem die unsortierte DCG durch die IDCG dividiert wird.

$$\mathrm{nDCG_p} = \frac{DCG_p}{IDCG_p}$$

Beispiel für den nDCG einer Query mit einer Search Result List der Länge 6:

$$\mathrm{nDCG_6} = \frac{DCG_6}{IDCG_6} = \frac{6.861}{8.740} = 0.785$$

Der nDCG eines IR-Systems ist übrigens der Mittelwert der nDCG aller Queries.

3.4.8. Pooling

Beim Pooling werden die Top *N* Retrieval Results von einem Set aus *n* Systemen in ein Pool geworfen. Menschen bewerten dann jedes Dokument aus diesem Pool. Dokumente die es nicht ins Pool geschafft haben können automatisch als irrelevant betrachtet werden. Das Pool ist in Wirklichkeit kleiner als $N * n$, da es einige Overlaps bei den Dokumenten gibt. Typischerweise handelt es sich um 1/3 der maximalen Größe $N * n$.

3.4.8.1. Voraussetzungen für Pooling

Damit Pooling sinnvoll ist müssen ein paar Voraussetzungen erfüllt sein. Es sollten verschiedene Arten an Systemen verwendet werden, auch manuelle. Die Pools sollten relativ tief sein, also *N* sollte gleich 100 oder mehr sein. Die Collections sollten aber auch nicht zu groß sein („groß“ ist aber relativ).

3.4.8.2. Vor- & Nachteile von Pooling

Der klare Vorteil von Pooling ist, dass weniger Dokumente bzgl. Relevanz bewertet werden müssen. Dadurch kann die Qualität des IR-Systems schneller bewertet werden. Es gibt allerdings auch ein paar Nachteile!

Zum einen kann nicht sichergestellt werden, dass alle Dokumente gefunden werden welche die Query erfüllen (die Recall Werte sind also nicht unbedingt richtig). Durchläufe die nicht am Pooling teilgenommen haben sind benachteiligt (engl. „runs that did not participate in the pooling may be disadvantaged“). Und „If only one run finds certain relevant documents, but ranked lower than the cutoff value (e.g. 100), it will not get credit for these.“

3.4.9. BPref

…

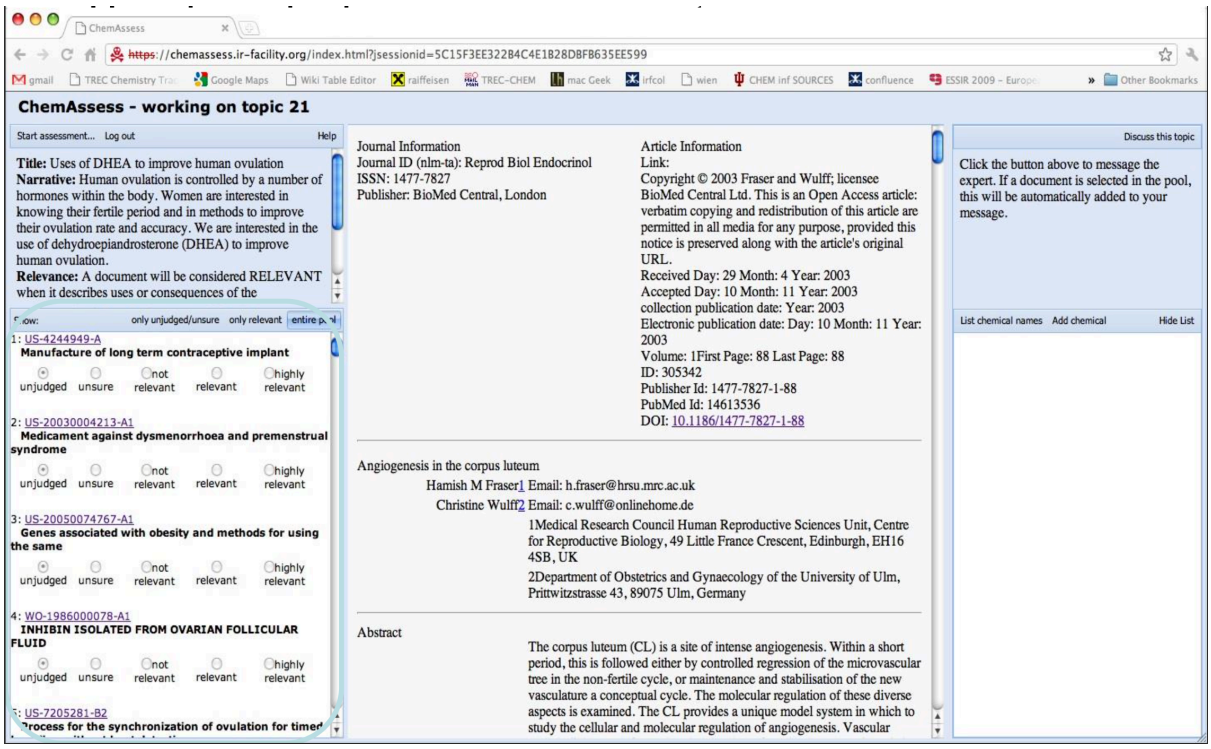
3.4.10. Test Collections

…

3.5. User based evaluation

In der User basierten Evaluation bewerten viele User das IR-System, wie der Name schon suggeriert. Diese Evaluation ist sehr subjektiv! Die User basierte Evaluation lässt sich in 2 Arten unterscheiden - „Relevant/non-relevant assessments“ und „User satisfaction evaluation“.

Beim „Relevance Assessment“ müssen User bewerten ob sie Dokumente für eine Query als wichtig oder unwichtig empfinden. Hierbei handelt es sich um ein Labor-Setting. Im Unterricht wurde erzählt dass an einem IR-System für die Medizin gearbeitet wurde. Einige Studierende der Medizinischen Universität Wien mussten die Relevanz sehr vieler Dokumente (hunderte) im Bezug auf verschiedene Queries bewerten. Das hilft dabei herauszufinden ob das IR-System die korrekten Ergebnisse liefert oder nicht.



Bei der „User satisfaction evaluation“ geht es um das System als Ganzes - also wie zufrieden die User sind. Hier spielt also UI und Usability eine große Rolle! Die Zufriedenheit kann aber auch negativ beeinträchtigt werden, wenn das IR-System wenig relevante Dokumente bei Suchen ausgibt!

3.6. Effektivität Lab VS User-based

Die Resultate sind Unterschiedlich. Manche Experimente zeigten eine Korrelation zwischen User Zufriedenheit und IR Effectiveness Measures.

3.7. Diskussion & Evaluation

Die Evaluierungen sind alle sehr Labor-spezifisch. Es ist nicht zu 100% klar wie gut sie in einem realen Situationen performen. Man kann das ganze mit einer Metapher vergleichen. Keine Evaluation eines Tennisschlägers wird aussagen können wie gut er während eines Spiels performed - denn die Performance ist zum größten Teil userabhängig! Allerdings werden viele Leute trotzdem basierend auf der Evaluation einen Tennisschläger kaufen.

TODO: Folien 90, 91, 92

3.8. Konklusion

IR Evaluation ist ein eigenes Forschungsfeld, da es sehr komplex ist. Ohne Evaluation kann es auch keine Forschung in diesem Bereich geben! Die meisten Evaluationsübungen sind sind Laborexperimente. Das bedeutet, dass die realen Bedürfnisse und Umstände der User bestmöglich berücksichtigt werden müssen, da es sonst unterschiede zwischen Labor und realer Anwendung gibt.

4. Lecture 5 - Web Search

In diesem Kapitel geht es um IR Systeme im Bezug auf das Web.

4.1. Was macht Web Search einzigartig?

Das Web ist riesig! Es gibt mehrere Milliarden Websites und sehr viel Inhalt. Pro Tag entstehen Terrabyte an Inhalten. Viele Menschen suchen Im Web nach Informationen aus verschiedensten Gründen, sie wollen Fakten finden, den besten Arzt in ihrer Umgebung, Online Shopping, etc. Daher wollen viele Webseiteninhaber bei einer entsprechenden Suche bei Google oder Bing (hahah Bing) logischerweise auf Platz 1 im Ranking erscheinen. Um diesen ersten Platz eher zu erreichen gibt es etwas das sich **Search Engine Optimization (SEO)** nennt, wie das genau funktioniert lernen wir später. Im Grunde geht es aber darum seine Webseite so zu optimieren, dass sie bei entsprechenden Suchen auf Google auf Platz 1 gelistet ist.

Es gibt aber auch Menschen, welche versuchen diesen ersten Platz unehrlich zu erreichen, um sich zu bereichern - immerhin klicken die meisten Menschen eher auf das erste Suchergebnis als auf das zweite oder dritte! Das nennt man dann **Spamdexing** (oder Search Engine Spamming) - also der Versuch das Ranking der Webseite zu manipulieren, selbst wenn sie keine Relevanz für die Suche der User hat.

Beispiele für Spamdexing sind:

- Keyword stuffing: viele Keywords bei der Seite verwenden die nichts mit der Suche zu tun haben - ähnlich wie hashtag spamming bei Social Media Postings
- Hidden text: weißer Text auf weißem Hintergrund - der Text ist für User also nicht sichtbar. Wieso sollte man das machen? Früher wurde für das Ranking der TF-IDF Score benötigt. Wenn also einzigartige Wörter ganz oft wiederholt wurden, dann bekam die Seite ein besseres Ranking. Zb wenn man 100 Mal „maui resort“ im Text versteckt. Diese Technik funktioniert heute nicht mehr.
- Link Spamming: die Link Analyse der Search Engines gezielt ausnutzen
  - Link Farms: Mehrere Webseiten die sich gegenseitig verlinken. Echte Beispiele:
    - http://www.doc-lopez.at/hno/
    - https://www.wien-kardiologie.at/expertennetzwerk.html
    - https://nerven-heilkunde.at/expertennetzwerk/
  - Spam Blogs: Von qualitativ schlechten Bloggs auf Webseite verlinken.
- Cloaking
- Scraper Site: hier wird der Content von anderen Websites gestohlen, und auf die eigene eingefügt.
- Weitere Beispiele: https://packted.com/what-do-you-mean-by-spamdexing/

Um dem Spamdexing entgegenzuwirken gibt es das sogenannte **Adversarial IR**. Es ist wichtig Spamdexing zu verhindern, weil jeder unverdiente Aufstieg im Ranking eine Verschlechterung der Precision des IR Systems bedeutet! Bei Adversarial IR geht es darum wie man mit **Daten arbeitet**, von denen man **weiß**, **dass** einige davon **manipuliert** sind. Adversarial IR beinhaltet die Lehre über Methoden mit denen solch eine Manipulation festgestellt (**detect**), isoliert (**isolate**) und besiegt (**defeat**) werden kann.

4.1.1. IR vs Web IR

- Web IR hat eine sehr viel größere Datenmenge
- Niemand überprüft den Inhalt
- Strukturiert (Links, Ankertexte, Layers)
- Dynamisch und ändert sich immer
- Menschen versuchen das Ranking zu manipulieren
- Viele Menschen tragen bei (Wie gemeint?? Tragen bei zu was?!!)

4.2. Crawling & Indexing the Web

https://www.seobythesea.com/2010/05/what-makes-a-good-seed-site-for-search-engine-web-crawls/

https://www.ionos.at/digitalguide/websites/web-entwicklung/was-ist-web-scraping/

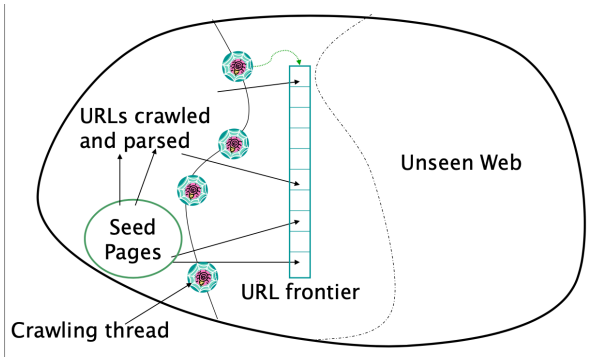
Wir haben schon gehört dass sich das Web ständig ändert. Ständig kommen neue Seite hinzu und alte Seiten verschwinden, in Webshops tauchen neue Produkte auf, etc. Darum müssen Search Engines ihre Indizes so aktuell wie möglich halten, um veralteten Content, gelöschte Seiten, usw zu verhindern und immer am aktuellsten Stand zu sein.

Crawler sind daher sehr wichtig für eine Search Engine, weil eben genau mit diesen Crawlern Änderungen auffallen und der aktuellste Inhalt und neue Seiten gefunden werden.

4.2.1. Wie ein Crawler Funktioniert

Der Startpunkt bei Crawler sind sogenannte „Seed Pages“. Auf diesen Seed Pages wird nach URLs gesucht auf die sie verlinken. Diese URLs landen dann in eine Queue. Auf den URLs in der Queue werden dann ebenfalls Crawler angewendet und wieder URLs extrahiert, die in dann wieder in der Queue landen.

Visualisierung eines Webcrawlers:



4.2.2. Web Crawler - Herausforderungen & Regeln

- Für ernsthaftes Web Crawling **benötigt** man **mehr als nur einen Rechner**! Die **Schritte** die oben erklärt wurden, werden daher auf mehrere Rechner **aufgeteilt**.
- Es gibt auch **bösartige Seiten**, wie **Spamseiten** oder sogenannte **Spider Traps** (oder Crawler Traps). Diese Spider Traps sind Strukturelle Probleme einer Webseite, bei denen die Crawler eine unendliche Anzahl an irrelevanten Links finden. Das kann übrigens auch das SEO negativ beeinflussen! Mehr dazu auf https://www.contentkingapp.com/academy/crawler-traps/
- Auch **nicht-bösartige Seiten** können **Herausforderungen** darstellen. So können die Latenz bzw. Bandbreite eines Servers, duplizierte Seiten oder Site Mirrors den Crawlingvorgang erschweren.
- Aus gründen der **Höflichkeit**, darf man eine Seite nicht zu oft crawlen und die Crawling Regeln die in der robots.txt gesetzt sind müssen beachtet werden!
- Web Crawler müssen über eine gewisse **Robustheit** verfügen um immun gegenüber bösartigem Verhalten wie Spider Traps zu sein.

4.2.3. Robots.txt

…

4.2.4. URL Frontier

…

4.3. Ranking the Web

…

4.3.1. PageRank

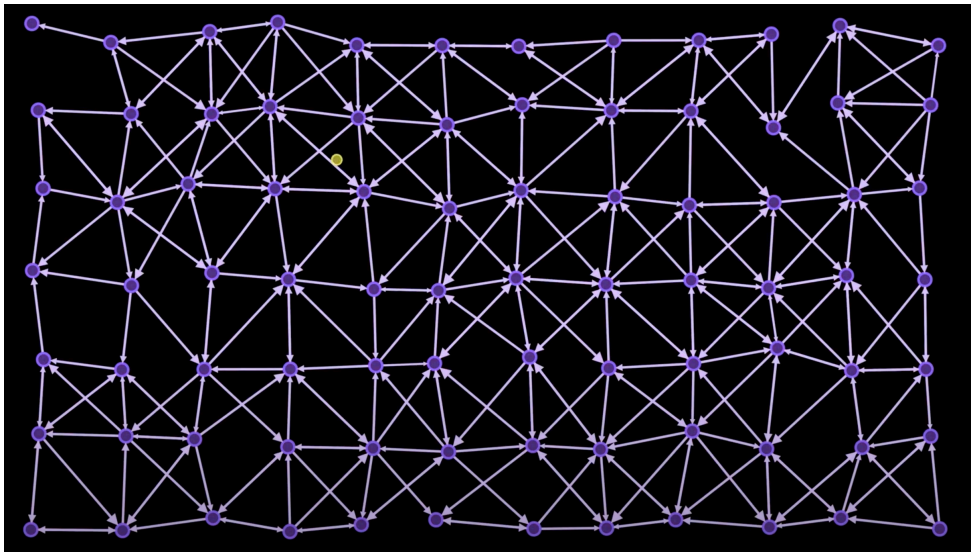
https://www.elegantthemes.com/blog/wordpress/what-is-pagerank-how-does-it-work-and-why-does-it-matter

https://www.youtube.com/watch?v=meonLcN7LD4

https://medium.com/analytics-vidhya/google-page-rank-and-markov-chains-d65717b98f9c

https://www.youtube.com/watch?v=JGQe4kiPnrU

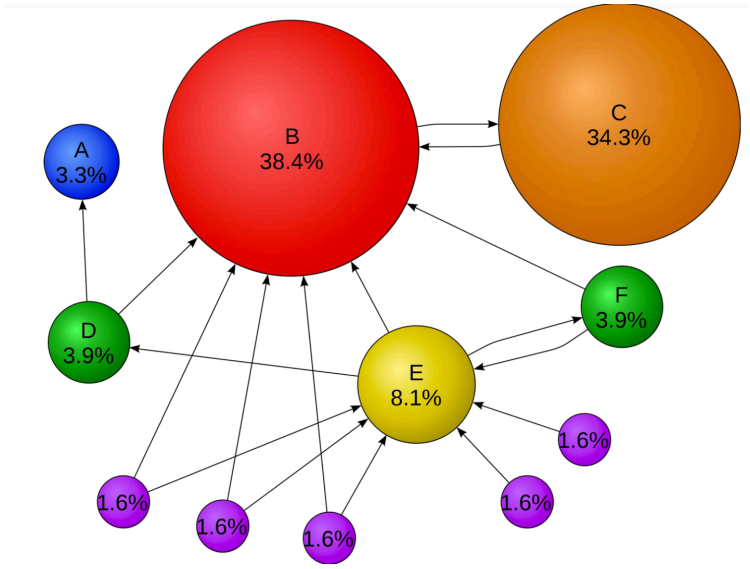
PageRank ist ein sehr wichtiger Algorithmus der von Google verwendet wird um das Ranking von Seiten festzustellen. Dazu stellen wir uns das Web als gerichteten Graphen vor. Dieser Graph heißt „Web Graph“ und jede Node stellt eine Seite dar und jede Kante einen Hyperlink zu einer Seite - ca so:



Grob gesagt, wird der PageRank festgestellt, indem man einen User simuliert der das Web surft. Dazu starten wir bei einer zufälligen Seite (Node) und folgen dann zufällig einen Hyperlink (Kante) dieser Seite und gelangen zu einer neuen Seite. Danach wird von dieser neuen Seite aus wieder einem Hyperlink gefolgt, auf eine weitere neue Seite, usw. Daraus lässt sich für jede Seite die Wahrscheinlichkeit berechnen, dass sie besucht wird. In den ersten paar Durchläufen ist diese Wahrscheinlichkeit noch eher ungenau aber je öfter man das wiederholt, desto genauer und verlässlicher werden die Berechnungen.

Auch wenn man das vielleicht nicht sofort glauben möchte aber wenn man das oft genug wiederholt ändern sich die Wahrscheinlichkeiten der Seiten nicht mehr - ca 50 Wiederholungen um genau zu sein. Das nennt sich dann „Stationary Distribution“ (oder auch „Steady State“) - es macht also keinen Sinn mehr die Simulation fortzusetzen, da sich die

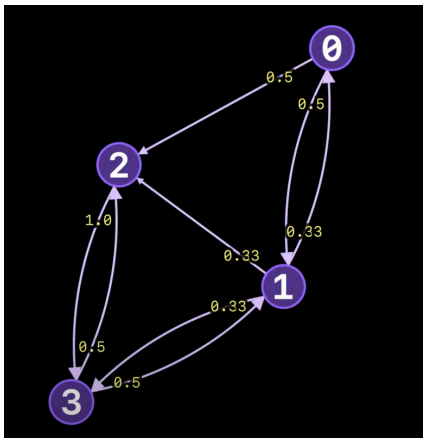
Wahrscheinlichkeiten nicht mehr ändern und man ist fertig. Die Wahrscheinlichkeiten der Knoten ergeben dann den PageRank - je höher die Wahrscheinlichkeit, desto höher ist auch logischerweise der PageRank. Folgendes Bild soll so einen Steady State verdeutlichen:



4.3.2. PageRank genauer - Markov Chain

Die Markov Chain (dt. Markov-Kette) ist ein mathematisches System bzw. ein stochastisches Model. In diesem Model gibt es eine Sequenz an Zuständen. Eine Zustandswechsel erfolgt aufgrund von bestimmten Wahrscheinlichkeitsregeln. Der Webgraph ist also eigentlich nichts anderes als eine Markov Chain!

Bemerkung: In der Zusammenfassung wird Knoten und Zustand ab jetzt als Synonym verwendet.



Beispiel einer Markov Chain

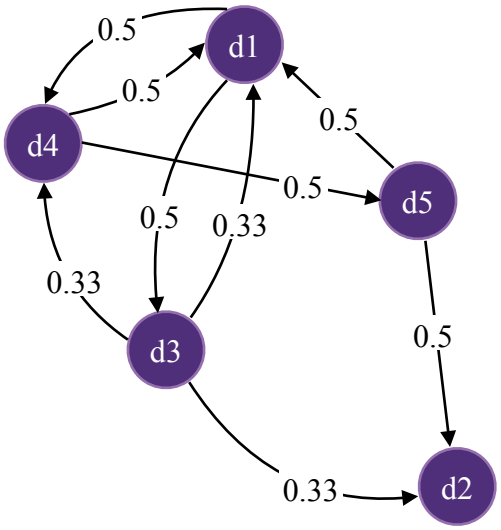
Die Wahrscheinlichkeiten der Zustandsänderungen sind typischerweise gleich verteilt - soll heißen, dass bei einem Knoten der 3 ausgehende Kanten hat, jede Kante mit der gleichen Wahrscheinlichkeit von 33% genommen wird. Zb wie bei Knoten 1. Es ist aber auch möglich, dass User mit einer viel höheren Wahrscheinlichkeit, von Seite 1 aus die Seite 2 besuchen statt die Seite 0. Dann haben die Kanten natürlich keine gleiche Wahrscheinlichkeit!

https://brilliant.org/wiki/markov-chains/

4.3.3. Transition Probability Matrix

Um den PageRank berechnen zu können müssen wir die Wahrscheinlichkeit eines jeden Zustandswechsels P_{ij} kennen. Also wie hoch ist zB. die Wahrscheinlichkeit auf Knoten d4 zu landen, wenn man bei Knoten d3 startet (= P_{34}), usw. Dazu wird einfach eine Tabelle bzw. eine Matrix erstellt, in der genau diese Wahrscheinlichkeiten verzeichnet sind.

Es folgt ein Beispiel für einen kleinen Webgraph und dessen Transition Probability Matrix P



P_{12}

P_{35}

	d1	d2	d3	d4	d5
d1	0.0	0.0	0.5	0.5	0.0
d2	0.0	0.0	0.0	0.0	0.0
d3	0.33	0.33	0.0	0.33	0.0
d4	0.5	0.0	0.0	0.0	0.5
d5	0.5	0.5	0.0	0.0	0.0

Transition Probability Matrix P mit den Wahrscheinlichkeiten P_{ij}.
Wie zu sehen ist ergibt die Summe in jeder Zeile 1 (außer bei sogenannten Dead Ends wie d2).

4.3.4. Dead Ends & Teleportation

Dieser Algorithmus der den Graphen einfach entlangläuft hat natürlich ein großes Problem - was passiert, wenn man auf eine Seite (Knoten) ohne ausgehende Links landet (ausgehende Kanten)? Wenn das der Fall ist, bleibt man stecken und der Algorithmus ist sinnlos.

Die Lösung dazu ist sowohl simpel als auch genial - immer wenn man ein Dead End erreicht, wählt man einfach zufällig einen anderen Knoten aus dem Graphen aus bei dem man weitermacht. Dieser Vorgang nennt sich auch „Teleportation“.

4.3.4.1. Teleportation genauer

Konkret wird die Teleportation wie folgt umgesetzt. Man wählt eine Teleportation Rate α zwischen 0 und 1. Alle Wahrscheinlichkeiten der Zustandswechsel werden um den Faktor (1-α) reduziert (Multiplikation). Da jeder Knoten einen zufälligen Zustandswechsel zu einem zufälligen Knoten bekommen soll (=Teleportationwahrscheinlichkeit), bekommt jeder Zustand nochmals die Wahrscheinlichkeit α/N (Addition), wobei N die Anzahl der Knoten des Graphen ist. Die Formel für die neue Wahrscheinlichkeit eines Zustandswechsels ist also:

$\mathbf{P_{ij} * (1-\alpha) + (\alpha/N)}$

Für Dead Ends wird irgend ein Zustand mit der Wahrscheinlichkeit von **1/N** gewählt.

Am besten veranschaulichen wir uns das Anhand eines Beispiels!

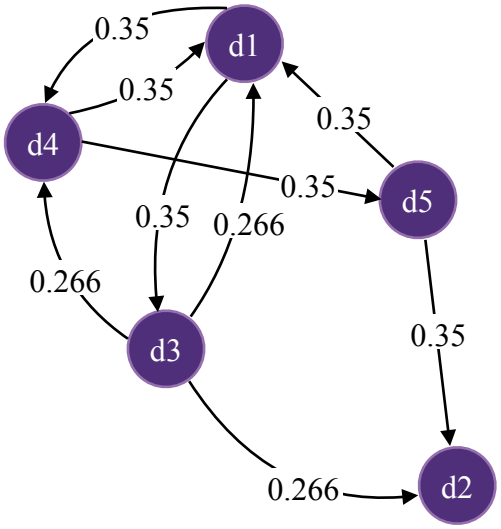
Wir haben den Graphen von vorher mit 5 Knoten. N ist also 5. Nun wählen wir eine Teleportation Rate α. Damit wir in unserem Beispiel einfacher Rechnen können und schönere Zahlen haben wählen wir 0.5 für α.

Somit ergibt sich der Faktor 0.5 (1-0.5) mit dem wir jeden Zustandswechsel multiplizieren und der zufälligen Zustandswechsel (=Teleportationswahrscheinlichkeit) von 0.1 (0.5/5) welcher hinzu addiert wird.

Somit reduzieren wir alle Wahrscheinlichkeiten der Zustandswechsel um den Faktor 0.5 (1-0.5). Der neue Zustandswechsel den jeder Knoten bekommt hat eine Wahrscheinlichkeit von 0.1 (0.5/5).

Also: $P_{ij} * 0.5 + 0.1$

Und für Dead Ends: 0.2 (also 1/5)



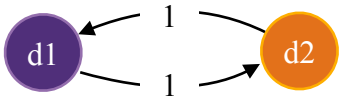
	d1	d2	d3	d4	d5
d1	0.1	0.1	0.35	0.35	0.1
d2	0.2	0.2	0.2	0.2	0.2
d3	0.266	0.266	0.1	0.266	0.1
d4	0.35	0.1	0.1	0.1	0.35
d5	0.35	0.35	0.1	0.1	0.1

Webgraph nach Teleportation Rate 0.5

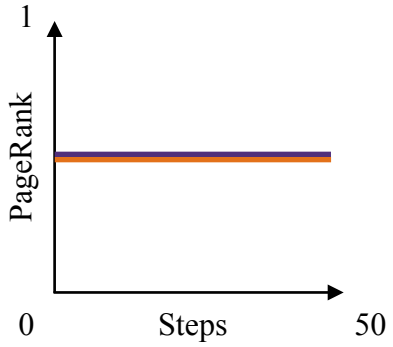
4.3.4.2. Ergodic Markov Chains

https://de.wikipedia.org/wiki/Irreduzible_Markow-Kette

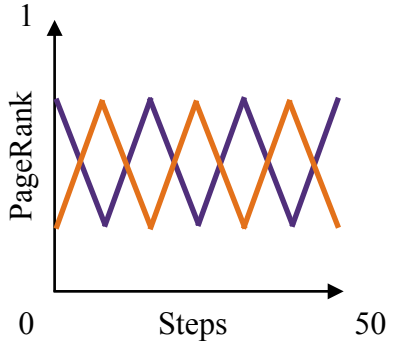
Die Dead Ends sind leider nicht das einzige Problem auf das wir stoßen können. Was ist zum Beispiel wenn wir Zyklen in unserer Markov-Kette haben, wie zum Beispiel hier:



Wenn wir eine gleichmäßige Startverteilung haben, ist das in diesem Fall kein Problem. Die Seiten haben einfach den selben PageRank. Der zugehörige Graph mit dem man den PageRank nach jeder Iteration ablesen kann würde also so aussehen:



Wenn wir bei der selben Markov-Kette aber eine ungleichmäßige Startverteilung von 80:20 haben, so oszilliert die Verteilung hier anstatt zu konvergieren - also anstatt einen Steady State zu erreichen. Der zugehörige Graph mit dem man den PageRank nach jeder Iteration ablesen kann würde so aussehen:



Deshalb ist es **wichtig**, dass die Markov-Kette **aperiodisch** ist - also keine Zyklen enthält!

But wait there is more! Leider ist das nicht das einzige Problem! Es kann zum Beispiel vorkommen, dass wir eine Markov-Kette haben, in der es unmöglich ist einen Pfad von einem Zustand zu einem anderen zu finden. Solche Markov-Ketten heißen **reduzierbar**. Ein sehr einfaches Beispiel für eine reduzierbare Markov-Kette ist folgendes:

Wenn es hingegen möglich ist, dass jeder Zustand von jedem Zustand aus erreichbar ist (nicht zwingend in einem Schritt, Umwege sind auch ok), dann ist die Kette **irreduzibel**. Damit eine Markov-Kette konvergiert und einen Steady State besitzt muss sie die Eigenschaften **aperiodisch** und **irreduzibel** besitzen! Solch eine Kette heißt auch „Ergodic Markov Chain“

Ok und was passiert wenn unser Webgraph diese Eigenschaften nicht besitzt? Die Antwort ist, dass unser Webgraph diese Eigenschaften garnicht besitzen muss, denn das **Teleportieren** stellt sicher, dass der Graph eine **Ergodic Markov Chain** ist!

4.3.5. Power Algorithmus

Mithilfe des Power Algorithmus wird der PageRank für jede Seite letztendlich berechnet. Dazu benötigen wir die **Transition Probability Matrix P** und eine **Startverteilung**.

Damit wird die Wahrscheinlichkeit eines jeden Zustandes **iterativ** ausgerechnet, **bis** ein **Steady Stare erreicht** ist - also bis sich die Wahrscheinlichkeiten nicht mehr ändern. Das sind dann die PageRanks der Seiten.

4.3.5.1. Die Formel

Die Wahrscheinlichkeit (P) eines Zustands (d) ist die Summe der Wahrscheinlichkeiten, dass dieser Zustand von einem anderen Zustand aus erreicht wird. Also die Wahrscheinlichkeit für Zustand d_1 in einem Webgraph mit 5 Seiten ist: die Summe aus „*P* dass d_1 von d_1 aus erreicht wird + *P* dass d_1 von d_2 aus erreicht wird + …. + *P* dass d_1 von d_5 aus erreicht wird, usw“.

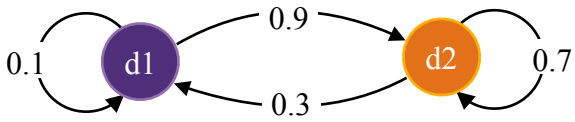
Das macht man in jedem Durchgang für jeden Zustand und wiederholt es so lange bis ein Steady State erreicht ist.

Die allgemeine Formel für die Wahrscheinlichkeit eines jeden Zustands bei jedem Durchlauf lautet also:

$$P_t(d_i) = P_{t-1}(d_1) * P_{1i} + P_{t-1}(d_2) * P_{2i} + P_{t-1}(d_3) * P_{3i} + \ldots + P_{t-1}(d_j) * P_{ji}$$

4.3.5.2. Beispiel

Am besten sehen wir uns den Power Algorithmus anhand eines Beispiels an. Gegeben ist folgender Webgraph welcher eine Ergodic Markov Chain ist.



Weiters gegeben ist die Startverteilung von $P_0(d_1) = 0$ und $P_0(d_2) = 1$. Soll heißen, dass User mit einer Wahrscheinlichkeit von 0% bei d_1 starten und mit einer Wahrscheinlichkeit von 100% bei d_2 starten. Es kann natürlich auch eine andere Startverteilung gegeben sein.

Die Teleportation wurde schon angewandt und aus dem Webgraph ergibt sich daher folgende Transition Probability Matrix P:

	d1	**d2**
d1	0.1	0.9
d2	0.3	0.7

0.3 * 0.1 + 0.7 * 0.3 **0.3 * 0.9 + 0.7 * 0.7**

0 * 0.1 + 1 * 0.3 **0 * 0.9 + 1 * 0.7**

Das sind die Gegebenheiten. Nun rechnen wir!

Iteration	**$P_{t-1}(d_1)$**	**$P_{t-1}(d_2)$**	**$P_t(d_1)$**	**$P_t(d_2)$**
0	0	1	0.3	0.7
1	0.3	0.7	0.24	0.76
2	0.24	0.76	0.252	0.748
3	0.252	0.748	0.2496	0.7504
…	…	…	…	…
34	0.25	0.75	0.25	0.75

Wir hören auf wenn die Wahrscheinlichkeiten aus der jetzigen Iteration gleich mit denen aus der vorherigen Iteration sind. In unserem Fall ist das bei Iteration #34 der Fall.

Steady State!

Das Ergebnis ist nun der PageRank Vektor π mit [0.25, 0.75]. Die Seite d_2 hat also ein höheres Ranking (0.75) als die Seite d_1 (0.25).

4.3.6. PageRank Zusammenfassung

Um also den PageRank zu bekommen sind folgende **Preprocessing** Schritte notwendig:

- Web als Markov Kette darstellen
- Transition Probability Matrix P erstellen
- Teleportation anwenden (Transition Probability Matrix P wird überarbeitet) → macht Webgraph zu einer Ergodic Markov Chain
- PageRank π für alle Seiten berechnen aufgrund von Matrix P, zB mittels Power-Algorithmus.

Außerdem sind folgende **Query Processing** Schritte notwendig:

- Seiten holen die Query matchen
- Liste der Seiten nach PageRank sortieren
- Sortiere Liste dem User geben

4.3.7. PageRank Probleme

4.4. Query Log Analysis

…

5. Lecture 6 - Statistical Semantics and Explicit Semantics

Es gibt Wörter die mehrere Bedeutungen haben (**Polysemy**). Zum Beispiel „mouse“ (Tier vs Computermaus), „charge“ (aufladen, jmd etwas verrechnen, etwas od. jmd. stürmen), „bus“ (Autobus, Bus auf einem Motherboard, etc.). Das VSM ist zum Beispiel nicht in der Lage zwischen diesen verschiedenen Bedeutungen zu unterscheiden!

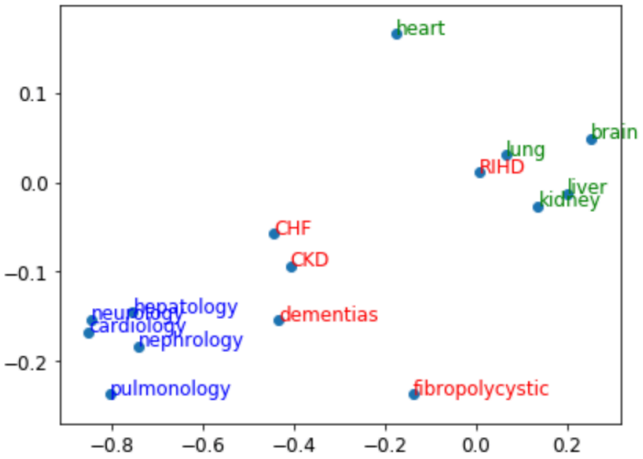
Dann gibt es auch Wörter die unterschiedlich sind, aber die selbe Bedeutung haben (**Synonymy**).Zum Beispiel „car“ und „automobile“ oder „aeroplane“, „aircraft“, „plane“ und „airliner“. Beim VSM wird auch keine Assoziation zwischen Wörtern gemacht, mit der eine Synonymität festgestellbar wäre.

Genau um diese Probleme zu Lösen gibt es die Statistical- und Explicit Semantics! Die Statistical und Explicit Semantics können in 4 Kategorien mit entsprechenden Maßnahmen unterteilt werden:

	Statistical	**Explicit**
Global	Latens Semantic Analysis (LSA)	Explicit Semantic Analysis (ESA)
Local	Word Embedding	Semantic Annotation

5.1. Word Embedding

Sehe wir uns zuerst die Maßnahme „Word Embedding“ an. Diese geht von folgender philosophischen Gedanken aus: „You shall know a word by the company it keeps“ bzw. „In most cases, the meaning of a word is its use“. Deshalb ist die Idee bzgl. der Umsetzung einen Vektorraum zu erstellen, in welchem verwandte Wörter auch nahe beieinander sind.



Word Embedding für Anatomie: https://www.researchgate.net/profile/Faiza-Khattak/publication/332543716/figure/fig4/AS:796161606705153@1566831127474/Pubmed-Word-embedding-visualization-Green-text-Anatomical-location-heart-lung-liver.ppm

5.1.1. Counting Co-Occurences (auch „One-Hot Encoding“ genannt)

https://www.youtube.com/watch?v=oUpuABKoElw

https://www.youtube.com/watch?v=5MaWmXwxFNQ

Das Word Embedding ist ein Machine Learning Prozess. Machine Learning kann Text nicht verarbeiten, Zahlen hingegeben schon! Deshalb müssen Terme durch Zahlen-Vektoren repräsentiert werden, so einen Vektor nennt man **„Embedding Vector“**. Eine Möglichkeit solch einen Vektor zu erstellen, ist es Wörter zu zählen.

Konkret heißt das, dass wir uns für jedes Wort verschiedene Sätze ansehen und überprüfen wie oft dieses Wort in Verbindung mit einem anderen Wort enthalten ist. So haben wir zum Beispiel folgende Sätze:

- „It is a **nice** day out“
- „Is it **nice** out?“
- „**Nice** Day“

Das Wort „day“ wird mit dem Wort „nice“ insgesamt also 2 mal in Verbindung gebracht! In echt ist die Menge an Texten natürlich viel höher! Da die meisten Wörter aber nicht im Kontext von anderen Wörtern vorkommen entsteht leider ein Vektor der licht (**sparse**) ist. Das bedeutet, dass der Vektor größtenteils aus Nullen besteht. Das ist eine enorme Platzverschwendung und ist daher nicht sehr effizient! Ein dichter (**dense**) Vektor wäre also effizienter und einfacher für Machine Learning und würde Synonyme im Vektorraum auch näher zusammen bringen. Wir wollen also einen Vektor der dicht (**dense**) ist!

Deshalb wird die Dimension der Vektoren verringern (zB auf 100). Ein Wort wird dann also durch einen Vektor mit 100 Dimensionen repreäsentiert. zB:

„tea“: <10, 3, 0, 2, 9, 4, … >,

„dog“ <1, 4, 0, 0, 5, 8, … >, usw.

5.1.2. word2vec

https://www.youtube.com/watch?v=5MaWmXwxFNQ

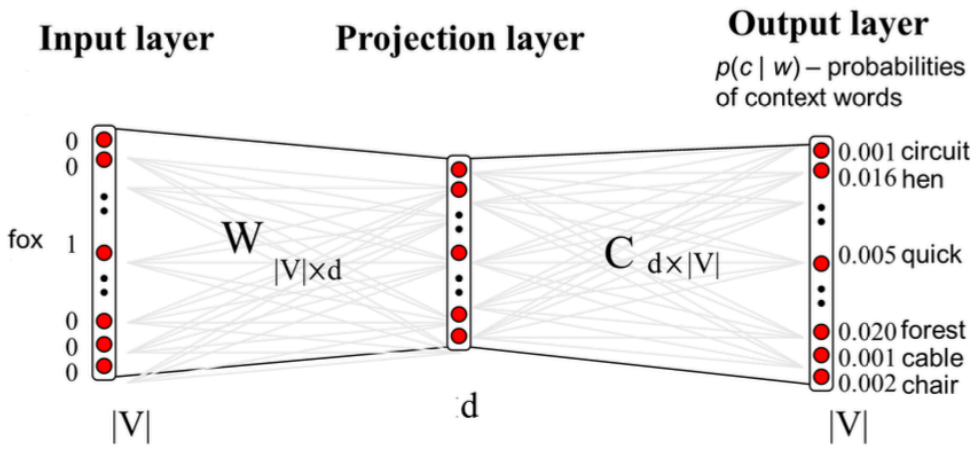
Word2vec ist Neural Network Model für Word Embedding. Der Prozess von word2vec sieht wie folgt aus:

Zuerst benötigt word2vec eine **große Menge** an **Text**. Dieser Text wird **verwendet** um das Neural Network **zu trainieren** und im Betrieb verlässliche Entscheidungen treffen zu können. Dazu erstellt word2vec aus dem Text ein **Vokabular**. Das Vokabular ist nichts anderes als ein Vektor V aus den Wörtern. **Um** den **Kontext** eines Wortes zu **verstehen zu können** werden sogenannte **Skip-Grams angewandt** - dazu kommen wir gleich! Das Neural Network wird also **mithilfe** dieser **Skip-Grams trainiert**. Am Ende werden die **„Embedding**

Vektors“ aus dem Neural Network **extrahiert** - also ein für jedes Wort ein Vektor aus Zahlen, welcher das entsprechende Wort repräsentiert.

Insgesamt kann der Prozess von „Input Text“ zu „Fertiges Resultat“ so visualisiert werden:

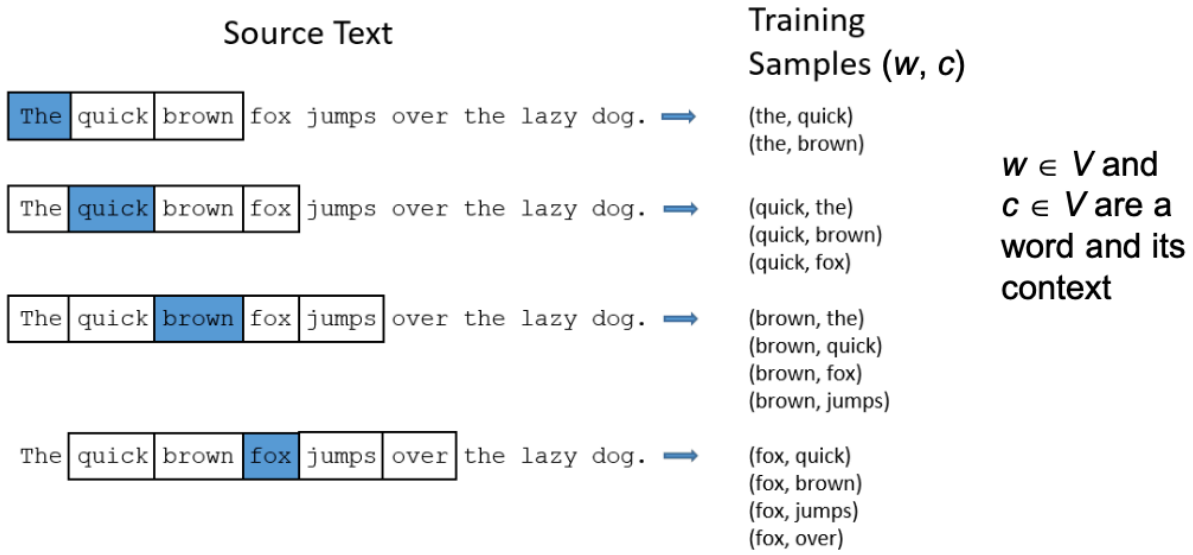
Das Neural Network kann so visualisiert werden:



5.1.2.1. Skip-Grams

Das Skip-Gram ist eine Möglichkeit herauszufinden, von welchen Wörtern ein bestimmtes Wort umgeben ist. Dazu wird ein Fenster mit einer fixen Große angegeben, zB der Größe 2. Das Skip Gram geht nun den Text Wort für Wort durch, und sieht sich an welche 2 Wörter link und welche 2 Wörter rechts vom aktuellen Wort stehen, und erstellt daraus Wort-Context Paare als Vektor (wenn das Fenster zB die Größe 3 hätte würden natürlich die nächsten bzw. vorherigen 3 Wörter zum aktuellen Wort angesehen werden).

Ein Beispiel für Skip-Grams mit der Fenstergröße 2 wäre folgendes:



Am Anfang können natürlich nicht die 2 Wörter links von „The“ angesehen werden, weil der Text gerade erst beginnt. Nach und nach entstehen immer mehr Word-Context Vektoren. Der Text ist dabei die Trainingsgrundlage des Neural Networks. Das heißt je mehr Text wir haben, desto besser funktioniert das System am Ende!

5.1.2.2. word2vec Output

…

5.1.2.3. Optimization

…

5.2. Latent Semantic Analysis (LSA)

https://blog.marketmuse.com/glossary/latent-semantic-analysis-definition/

Bei LSA (auch unter Latent Semantic Indexing - LSI bekannt) ist die Idee Terme einem Konzept zuzuweisen. Um das Konzept besser zu verstehen sehen wir uns folgendes Beispiel an:

term-concept matrix

	database concept	medical concept
data	1	
system	1	
retrieval	1	
lung		1
ear		1

document-concept matrix

	database concept	medical concept
doc1	1	
doc2	1	
doc3		1
doc4		1

Wir beginnen mit der Query „system“ und wollen alle Dokumente bekommen, die mit dem Wort zu tun haben. Wir suchen also zuerst in der Term-Concept Matrix nach unserem Term aus der Query „system“. Hier sehen wir, dass dieser Term etwas mit dem Concept „Database“ zu tun hat und nichts mit dem Konzept „Medical“. Daher gehen wir weiter zur Document-Concept Matrix und liefern die Dokumente die etwas mit dem Concept „Database“ zu tun haben - in unserem Fall doc1 und doc2.

LSA verwendet die sogenannte **„Singular Value Decomposition“** (SVD) um die Term-Document Matrix auf 3 Matrizen aufzuteilen - die **Term-Concept Matrix** U, die **Singular Value Matrix** S und die **Concept-Document Matrix** V.

Zur Erinnerung die Term-Document Matrix beschreibt wie häufig ein Term in einem Dokument vorkommt. Die Einträge in der Matrix können auch gewichtet sein, wodurch Kommazahlen entstehen können. Ein Beispiel für eine Term-Document Matrix:

	Anthony and Cleopatra	Julius Caesar	The Tempest	Hamlet	Othello	Macbeth
anthony	5.25	3.18	0.0	0.0	0.0	0.35
brutus	1.21	6.10	0.0	1.0	0.0	0.0
caesar	8.59	2.54	0.0	1.51	0.25	0.0
calpurnia	0.0	1.54	0.0	0.0	0.0	0.0
cleopatra	2.85	0.0	0.0	0.0	0.0	0.0
mercy	1.51	0.0	1.90	0.12	5.25	0.88
worser	1.37	0.0	0.11	4.15	0.25	1.95
. . .						

5.2.1. Singular Value Decomposition (SVD)

In den Folien wird noch sehr tiefgründig auf die SVD und die einzelnen Matrizen eingegangen aber aus Zeitgründen muss ich diese leider auslassen.

5.2.1.1. SVD - Summary

Die Term-Document Matrix *C* wird in die 3 Matrizen *U*, *V* und *S* aufgeteilt. Die Term Matrix U besteht aus aus einem Reihenvektor für jeden Term. Die Matrix V besteht aus einem Spalten Vektor für jeden Term und die Matrix ist eine Diagonale Matrix und dessen Werte repräsentieren die Wichtigkeit jeder Dimension.

C	d_1	d_2	d_3	d_4	d_5	d_6
ship	1	0	1	0	0	0
boat	0	1	0	0	0	0
ocean	1	1	0	0	0	0
wood	1	0	0	1	1	0
tree	0	0	0	1	0	1

$=$

U	1	2	3	4	5
ship	−0.44	−0.30	0.57	0.58	0.25
boat	−0.13	−0.33	−0.59	0.00	0.73
ocean	−0.48	−0.51	−0.37	0.00	−0.61
wood	−0.70	0.35	0.15	−0.58	0.16
tree	−0.26	0.65	−0.41	0.58	−0.09

$\times$

Σ	1	2	3	4	5
1	2.16	0.00	0.00	0.00	0.00
2	0.00	1.59	0.00	0.00	0.00
3	0.00	0.00	1.28	0.00	0.00
4	0.00	0.00	0.00	1.00	0.00
5	0.00	0.00	0.00	0.00	0.39

$\times$

V^T	d_1	d_2	d_3	d_4	d_5	d_6
1	−0.75	−0.28	−0.20	−0.45	−0.33	−0.12
2	−0.29	−0.53	−0.19	0.63	0.22	0.41
3	0.28	−0.75	0.45	−0.20	0.12	−0.33
4	0.00	0.00	0.58	0.00	−0.58	0.58
5	−0.53	0.29	0.63	0.19	0.41	−0.22

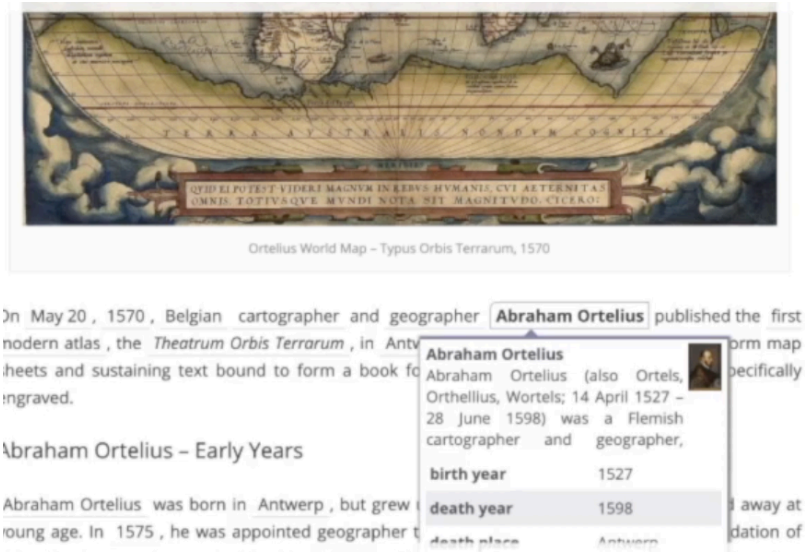
5.2.2. Ziele von LSI bzw. LSA

Dokumente die vom selben Thema handeln aber andere Worte **(Synonyme & semantisch verwandte Worte)** verwenden, würden im VSM nicht als ähnlich eingestuft werden, da die Vektoren der Dokumente aufgrund der anderen Wortwahl komplett unterschiedlich sind. Als User will man aber trotzdem Dokumente erhalten in denen es um das selbe Thema geht, auch wenn die Keywords der Query vielleicht nicht die Worte des Dokuments matcht. LSA löst diese Probleme **(Synonyme & semantic relatedness)**!

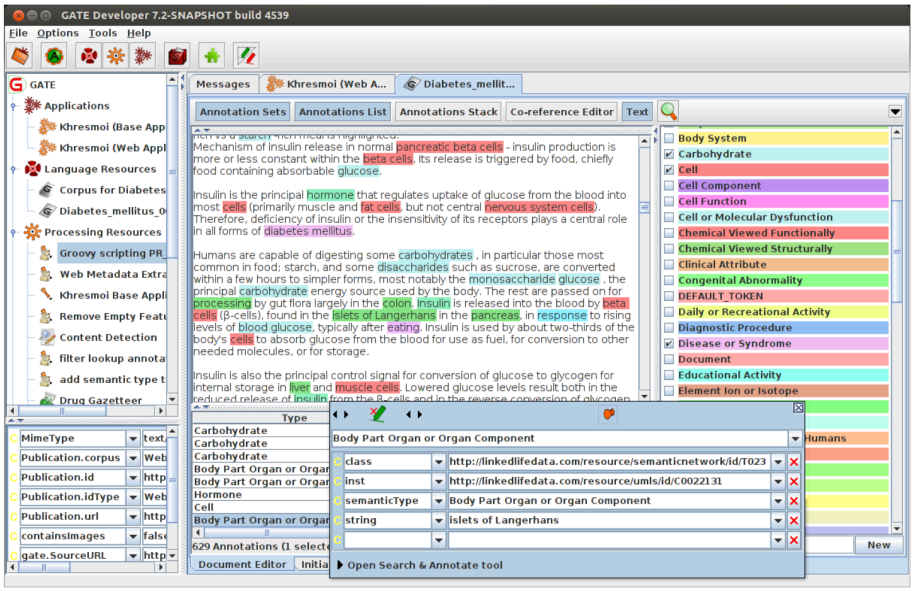
5.3. Semantic Annotation

Die Lösungen die wir bisher kennengelernt haben liefern Vektoren. Das Problem dabei ist, dass sie für das menschlichen Auge sehr obskur und nicht interpretierbar sind und noch dazu automatisch berechnet werden. Es wäre aber auch oft gut die Resultate schnell interpretieren zu können und genau darum geht es bei den Explicit Semantics! Durch Semantische Suche können Recall und Precision erhöht werden.

Es geht also darum Semantik einfach zu veranschaulichen, anstatt mit Vektoren. Bei der Semantic Annotation werden bestimmte Terme für den User optisch hervorgehoben. So ist es möglich in einem Wissenschaftlichen Blog, wichtige Personen oder Wörter optisch hervorzuheben, und bei einem Mouse-Over zusätzliche Informationen und Erklärungen oder Synonyme über das Wort anzuzeigen. In dem unteren Beispiel, wird für den den Kartographen Abraham Ortelius (in Echtzeit, also nicht Teil des HTMLs!!) eine Zusammenfassung seines Wikipedia Artikels generiert & angezeigt.



Es ist aber zum Beispiel auch möglich verschiedene Wörter verschiedenen Kategorien zuzuweisen, und diese Kategorien farblich zu encoden und entsprechend darzustellen, wie in diesem Beispiel:



https://www.youtube.com/watch?v=bYqbKrQ1N-8

5.3.1. Knowledge Bases

Um Semantische Suche und Semantische Annotation umsetzten zu können benötigt es sogenannte Knowledge Bases oder zu deutsch „Wissensdatenbanken“. Wichtig anzumerken ist, dass Datenbank und Knowledge Base sich stark unterscheiden und nichts miteinander zu tun haben! Sie unterscheiden sich schon im Aufbau und in der konkreten Verwendung.

Wir wissen ja wie eine Datenbank aufgebaut ist. Es werden Rohdaten in einer Tabelle als Strings, Zahlen oder Boolean gespeichert und es wird SQL verwendet um spezifische Daten auszulesen. Das Ergebnis ist ein Kumulat aus Rohdaten.

Die Idee einer Knowledge Base hingegen ist, dass konkrete Antworten zu konkreten Problemen gespeichert werden und die Suche intuitiv ist und keine eigene Sprache wie SQL benötigt wird. Es werden Objekte abgespeichert die mit Pointern mit anderen Objekten verbunden sind. Es ist sozusagen ein Netzwerk an Wissen - daher der Name.

Ein kommerzieller Anwendungsfall wäre folgender: Oft haben müssen Mitarbeiter Dinge über ihr Unternehmen herausfinden wie zB Arbeitsabläufe, Unternehmensstrukturen, Lagepläne, Guides, uvm. Knowledge Bases stellen für Mitarbeitern eine Art Bibliothek dar, mit der sie all diese konkrete Fragen beantwortet bekommen, anstatt Vorgesetzte zu fragen, die auch nicht immer eine Antwort auf alles haben. Das ist besonders bei großen Unternehmen sehr hilfreich! Die selbe Firme könnte auch eine andere Knowledge Base für Kunden zur Verfügung stellen, um generelle Fragen, Probleme, und Benutzung bzgl eines Produktes einfach, schnell und zuverflässig beantworten zu können.

Doch nun kommen wir zurück zu den Herausforderungen beim Information Retrieval! Anstatt Dokumente 1:1 anhand von konkreten Wörtern zu matchen, ist es mit einer Knowledge Base möglich die Suche basierend auf Konzepten durchzuführen. So werden Taxonomies, Theasuri oder Ontologies als Knowledge Bases verwendet. Für generelle Anwendungsfälle wie Sprachspezifische Probleme (Synonyme, Anglizismen, etc.) existieren schon Knowledge Bases. Andere wie Firmen-Spezifische Systeme, müssen aber selbst erstellt werden, da es natürlich kein öffentliches Interesse besteht.

5.4. Explicit Semantic Analysis

…