NLP4Web Summary

Moritz Gerhardt

Table of Contents

1 Basics

The web itself is an **application area** of NLP.

Web Applications of NLP

- Search Engines
- Spelling Correction
- Machine Translation
- Speech Recognition
- Plagiarism Detection
- Summarization
- Diachronic Analysis (Use of terms over time)
- Text Generators (LLMs)

The web also lends itself as a **resource** for improving NLP.

Web as a resource

- Web as a corpus
- Analysis of web content, especially knowledge repositories
  - Wikipedia
  - Wiktionary
  - etc.
- Recognising synonyms, paraphrases etc.

Some typical challenges for NLP are:

Challenges for NLP

- Removal of noise (duplicates, typos, etc.)
- Quality Assessment of content
- Integration of heterogeneous and scattered content (Content from different sources, regarding the same topic)
- Error handeling (spelling, grammar, syntax etc.)
- ”Clean” data (Errors, emoticons, abbreviations,etc.)

1.1 Analytic Linguistics in NLP

Just how natural language is composed of different linguistic elements which differ from one linguistic viewpoint to the others.

Linguistic Analysis Levels

- Phonetics and Phonology
- Segmentation
- Morphology
- Syntax
- Semantics
- Pragmatics and Discourse

1.1.1 Phonetics and Phonology

Phonetics and Phonology are Aspects that are concerned with the speech sounds of the language. Hereby Phonetics are concerned with the **physical** Aspects, such as articulatory (production of sound, articulation), acoustic (transmission of sound, amplitude, frequency etc.) and auditory (sound perception/reception). Phonology is concerned with the **functional** aspects of speech, such as how different sounds carry different meanings and how these sounds can change the meaning.

One especially important aspect of this are **homophones**, words which are pronounced the same but have different meanings. (For example, *night* and *knight*)

1.1.2 Segmentation

Segmentation is used to split up an input stream into an ordered sequence of units called **tokens**. The process is called **Tokenization**, which can be done by a **tokenizer** system. Tokens can correspond to a word form, sub-word units or punctuation. They do NOT have to be only one word, for example with names: "New York" can be one token. They may be subject to subsequent morphological analysis.

Example: John likes Mary and Mary likes John. → {"John", "likes", "Mary", "and", "Mary", "likes", "John", "."}

Tokenization however is not as simple as it seems, as you can't just split them up at specific characters. You also have to take into account the different meanings of the characters, as well as where a token starts and ends.

Tokenization Ambiguities

- Period
  - Periods do not only indicate the end of a sentence
  - Can also be:
    - Part of an abbreviation: R.E.M. (Rapid Eye Movement)
    - Numbers: 3.14159
    - References to websites and similar: www.wikipedia.com
    - etc.
  - In these cases they can not be split up at the period, as its one token.
  - Can get even more complicated with combined meanings: When an abbreviation is the last word in a sentence the period is part of the abbreviation but at the same time also the sentence ender and therfore its own token.
- Whitespace
  - Do not only seperate words
  - Can also be used as part of a name ("New York") or number (245 623)
- Comma
  - Do not only seperate sentence parts
  - Can also be used as a part of a number (3,14159)
- Single quotes
  - Not only used to signal quotations
  - Can also be used to signal contractions (don't, you're) or elisions (Moritz' summary)
  - In some languages even part of a word
- Dash -
  - Not only used to connect words
  - Can signal a range (pages 23-45)
  - In some languages signals close connection between words

Some words can even be split up into two tokens, which means there is no character seperating them. (Stau**becken** and **Staub**ecken)

1.1.3 Morphology

Morphology studies word formation and word forms. Hereby a word is defined as made up by **morphemes**. A morpheme is the **smallest unit that still carries meaning**. Hereby morphemes are split up into two categories: **bound** and **free**.

Bases and Affixes: Bound and Free Morphemes

- Free morpheme:
  - A morpheme which is a word →Can stand by itself
  - Example: *Housekeeper* →Can be split up into *House* and *keeper*. Both can stand by themselves and are therefore free morphemes
- Bound morpheme:
  - A morpheme which is not a word →Can not stand by itself
  - Needs to be used in combination with a free morpheme
  - Often called **Affixes**
  - Example: Cats →Can be split up into *Cat* and *s*. *Cat* can stand by itself, therefore it's a free morpheme, while *s* cannot, which makes it a bound morpheme.
- A minimal free morpheme is called a **stem**. They carry the main meaning of a word.
- Affixes can be added to a stem to alter the meaning

Types of Affixes

- Suffixes
  - Added to the end of a stem
  - Cat + s, nice + ly
- Prefixes
  - Added to the beginning of a stem
  - un + true, im + possible
- Infixes
  - Added to the middle of a stem
  - fan + bloody + tastic
  - Not used often in formal language
- Circumfixes
  - Added around a stem
  - ge + sagt + t

Multiple affixes can be added to stem.

For analysis it is important to find morphological related words. This can be done through **morphological normalization**. Hereby the goal is to find a representation for related words.

Methods for Morphological Normalization

- Stemming
  - Algorithmic approach to strip off endings of words.
  - Will not necessarily produce a real word form
  - Does not distinguish between inflection and derivation
  - Example:
    - sitting →sitt
    - anarchism, anarchy, anarchistic →anarchi
  - Stemming is ruled-based, which means that it will be very quick, however it might also yield arbitrary distinctions
  - As it is rule-based, it is also prone to errors, like under- and over-stemming →removing too little or too much
  -
    - Under-Stemming: adhere →adher, adhesion →adhes
    - Over-Stemming: appendicitis →append, append →append
  - Another error that can happen is when homographs (Words which are spelled the same but have different meanings) are used.
    - saw (past. see) vs. saw (Tool)
    - Will have the same representation but are totally different
    - Stemming cannot solve these problems
- Lemmatization
  - Gets the base form of the word form
  - This needs access to lexical data and part of speech tagging
  - Due to access to lexical data it can also handle irregular forms
  - Not as fast as stemming but more accurate
  - Example:
    - left (verb) →leave
    - left (noun) →left
    - indices (noun) →index
    - saw (verb) →see

1.1.4 Syntax

Syntax refers to the order in which words appear in or the way in which words are arranged. There is a theoretical infinite number of ways in which words can be arranged together, we can discern the meaning anyway.

Parts of Speech

- N: Noun →chair, bird, etc.
- V: Verb →walk, run, etc.
- ADJ: Adjective →red, fast, etc.
- ADV: Adverb →fortunately, slowly, etc.
- P: Preposition →in, of, etc.
- Pro: Pronoun →he, I, etc.
- Det: Determiner →the, a, etc.
- Intj: Interjection →oh, wow, etc.

We can assign parts of speech to each word in a corpus (POS tagging). Using these we can use Lemmatization to get the base form of a word.

There are some problems with POS tagging though. As words can have different meanings and can represent different parts we can't simply assign each word a part of speech.

These issues can be improved by using different models. The baseline is simply using the most frequent part of speech for a word. Other models are based on probabilistic tagging or rule-based tagging. The most accurate approach to this day are neural approaches, which are about 98% accurate.

Another topic within syntax is parsing. A sentence can have multiple meanings, as the strucuture itself is ambigous. For example: ”I saw the man with a telescope” →I saw the man that had a telescope OR I saw the man through a telescope.

1.1.5 Semantics

Semantics study the meaning of words, phrases, sentences etc. Further lexical semantics study the meaning of words.

In semantics lexical ambiguity can cause different interpretations. For example: I hit the ball with the bat →I hit the ball with the bat (animal) OR I hit the ball with the bat (sports instrument)

1.1.6 Pragmatics and Discourse

Pragmatics are concerned with the purpose of utterances. For example: Simply putting an emphasis on a word can change the meaning of a sentence. I never said she stole my money

- **I** never said she stole my money →Someone else said she stole my money
- I **never** said she stole my money →I didn't say that
- I never **said** she stole my money →I might have implied it but never said it
- I never said **she** stole my money →I was not reffering to her
- I never said she **stole** my money →I was not reffering to stealing
- I never said she stole **my** money →It was not my money
- I never said she stole my **money** →She didn't steal money

So the intended meaning of an utterance can be different than the lexical meaning of the utterance.

2 Text Classification

Text classification is an important part of NLP4Web. It has many applications and it's especially important to classify text to improve the date of the models.

Examples of Text Classification

- Spam Detection
- Sentiment Analysis
- Topic Labeling
- Age/Gender Identification
- Authorship Identification
- Language Identification

Some approaches to text classification are:

Approaches to Text Classification

- Rule-Based
  - Rules are "handcrafted" and human comprehensible
  - As such precision can be high
  - However it might also be very expensive to build and maintain as it needs to cover a lot of cases and handle changing languages
- Supervised Learning
  - Processes by which tagged/labeled data is picked and fed into an algorithm that then builds a model capaable of approximating text
  - Can be human comprehensible - but doesn't need to be
  - Easier to meaintain and fairly accurate
  - Needs a lot of data
- Unsupervised Learning (Deep Learning)

2.1 Naive Bayes Classifier

Naive Bayes is an algorithm for text classification. It is based on conditional Probabilites and Bayes Rule.

Conditional Probabilites

The probability that something will happen, given that else has already happened

- Assume **Outcome O** and **Evidence E**
- Then
  - $P(O, E)$ describes the probability of both O and E occuring

    $\rightarrow \boxed{P(O, E) = P(O) \times P(E|O)}$
  - $P(O)$ describes the probability of O occuring

    $\rightarrow \boxed{P(O) = \frac{P(O, E)}{P(E|O)}}$
  - $P(E|O)$ describes the probability of E given O

    $\rightarrow \boxed{P(E|O) = \frac{P(E, O)}{P(O)}}$

Bayes Rule

Bayes Rule conceptually describes a way to go from $P(E|O)$ to $P(O|E)$
This can be done with the formula:

$$\boxed{P(O|E) = \frac{P(E|O) \times P(O)}{P(E)}}$$

Unfortunately, Bayes rule only works for one piece of evidence at a time. For efficient text classification this is not enough, which is what **Naive Bayes** is for, as it can predict the outcome given multiple pieces of evidence.

Naive Bayes Classifier

Naive Bayes treats each piece of evidence as independent, which makes it easier to calculate (hence why *naive*)

$$\boxed{P(O|E1, \ldots, E_n) = \frac{P(E_1|O) \times P(E_2|O) \times \ldots \times P(E_n|O) \times P(O)}{P(E_1, E_2, \ldots, E_n)}}$$

Easier to remember:

$$\boxed{P(\text{outcome}|\text{evidence}) = \frac{P(\text{Likelihood of Evidence}) \times \text{Prior probability of outcome}}{P(\text{Evidence})}}$$

Multinomial Naive Bayes Model

$$\boxed{\begin{gathered} P(c|d) \propto P(c) \prod_{1 \le k \le n_d} P(t_k|c) \\ \text{with Posterior Probability } P(c|d) \text{ and Prior Probability } P(c) \end{gathered}}$$

The probability of document **d** belonging to class **c** is proportional to the product of the probabilities of terms **t** belonging to **c** and to the class prior **P(c)**

$$\boxed{c_{\text{map}} = \arg\max_{c \in C} \hat{P}(c|d) = \arg\max_{c \in C} \hat{P}(c) \prod_{1 \le k \le n_d} \hat{P}(t_k|c)}$$

The best class **c** for a document **d** is found by selecting the class for which the maximum a posteriori (map) probability is maximal.

So overall Naive Bayes provides us with a method to determine the most probable hypothesis given the data. This is based on probability calculation, therefore no training of a model is necessary.

However, Naive Bayes fails if the independence assumption is violated too much. Meaning that identical or overlapping features pose a problem, which necessitates proper feature selection.

2.2 Hidden Markov Models (HMM)

As many problems in NLP do have overlapping or identical data, Naive Bayes cannot always be relied upon.

2.2.1 Sequence Labeling through Classification

Many NLP problems can be viewed as sequence labeling. Hereby each token in a sequence is assigned a label. These labels are not independent of each other but depend on the other tokens in the sequence, especially their neighbors. One example of this is part of speech tagging.

This can be done using different techniques, the simplest being classification. Hereby tokens are identified by the token itself (using lexical knowledge) but also by their neighbors. This ”rolling window” will then go through the whole sequence classifying each token.

This does not give us a good solution, as the classification of the surrounding tokens are more important than the token itself for classification. To improve on this, we can now go through the sequence again using the same approach but this time using the classifiers itself as the inputs. This can be done forwards or backwards, which might result in different classifications.

This yields some problems though. As forwards and backwards can yield different results, it's hard to tell which classifications are correct or wrong and which should be adopted. This uncertainty cannot be solved in this method.

2.2.2 Hidden Markov Model Concept

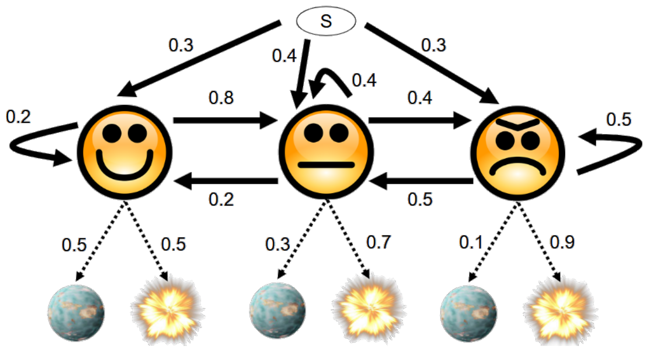
A Hidden Markow Model (HMM) is a statistical model of hidden, stochastic state transitions with observable, stochastic output.

Hidden Markov Model Key Characteristics

- Fixed set of states: At each time the model is in exactly one of these states
- State transition probabilities: The probability of transitioning from one state to another. The starting state can be fixed or random.
- Fixed set of possible outputs
- Distribution of probabilities for each possible output (Emission probabilities)

The HMMs purpose is to answer the question: **For an observed output sequence, what is the (hidden) state sequence that has the highest probability to produce this output?**

One example of such a model is the following. Here the states are representative of the moods of Darth Vader (wearing a mask, therefore hidden) and the emissions are whether or not he blows up a planet on this day. S represents the starting mood. (Not Blown up (NB), Blown up (BU), Happy (H), Angry (A), Neutral (N))



Now, we can observe whether or not he blows up a planet on a given day. But what is the most probable sequence of moods for a given sequence of emission?

Assume the following sequence of emissions: Not Blown Up, Blown up, Blown up. What is the probability that this sequence coincides with the following sequence of moods: Angry, Neutral, Angry?

For this we need to consider the transition and emission probabilities on each day.

For the first day this means, that we need the transition probability from starting point to angry (0.3) and the emission probability of Not Blown up for Angry (0.1). The joint probability on the first day is then given by $\boxed{P(\text{NB}, \text{A}) = 0.3 \cdot 0.1 = 0.03}$

For the second day we need the transition probability from angry to neautral (0.5) and the emission probability of Blown up for Neutral (0.7). The joint probability on the second day is then given by $\boxed{P(\text{BU}, \text{N}) = 0.5 \cdot 0.7 = 0.35}$
Additionally this gives us the probability for the sequence $\boxed{P(\{\text{A} \to \text{N}\}|\{\text{NB}, \text{BU}\}) = 0.03 \cdot 0.35 = 0.0105}$

For the third day we need the transition probability from neutral to angry (0.4) and the emission probability of Blown up for Angry (0.9). The probability for the joint probability on the third day is then given by $\boxed{P(\text{BU}, \text{A}) = 0.4 \cdot 0.9 = 0.36}$.
This also gives us the probability for the sequence $\boxed{P(\{\text{A} \to \text{N} \to \text{A}\}|\{\text{NB}, \text{BU}, \text{BU}\}) = 0.03 \cdot 0.35 \cdot 0.36 = 0.00378}$

2.2.3 Hidden Markov Model in NLP

We can apply this model to POS tagging. The sequence of qords are the observable emissions, while the POS tags are the hidden states. We also require the transition probabilities between the states, the initial probabilities and the emission probabilities.

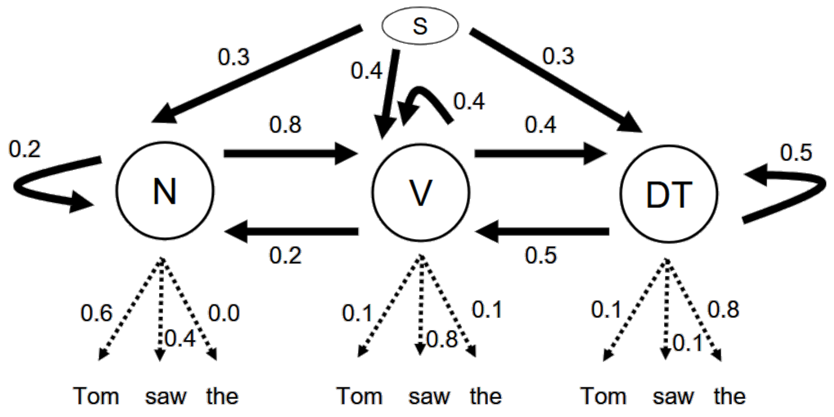
These probability are usually estimated by counting on a tagged training corpus. Hereby the following probabilities are given:

Transition probabilities

$$P(t_i|t_{i-1}) = \frac{C(t_{i-1}, t_i)}{C(t_{i-1})}$$
$$= \frac{\text{How often the first tag is followed by the second tag}}{\text{How often the first tag occured in the labeled corpus}}$$

Emission probabilities

$$P(w_i|t_i) = \frac{C(t_i, w_i)}{C(t_i)}$$
$$= \frac{\text{How often the word was associated with the tag in the labeled corpus}}{\text{How often the first tag occured in the labeled corpus}}$$



Example HMM for Sequence Tagging

$$\hat{t}_1^n = \arg\max_{t_1^n} P(t_1^n|w_1^n) \approx \arg\max_{t_1^n} \prod_{i=1}^{n} \underbrace{P(w_i|t_i)}_{\text{Emission}} \underbrace{P(t_i|t_{i-1})}_{\text{Transition}}$$

$\hat{t}_1^n$ is the most likely state sequence given an output sequence

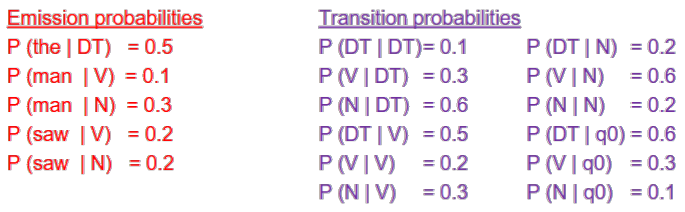
How do we calculate this? One solution would be brute force search by enumerating all possible sequences of states. This gives us a complexity of $O(s^m)$ where s is the number of states and m the length of the sequence.

A better solution would be the Viterbi algorithm, which uses a Dynamic Programming approach. This gives us a complexity of $O(ms^2)$.

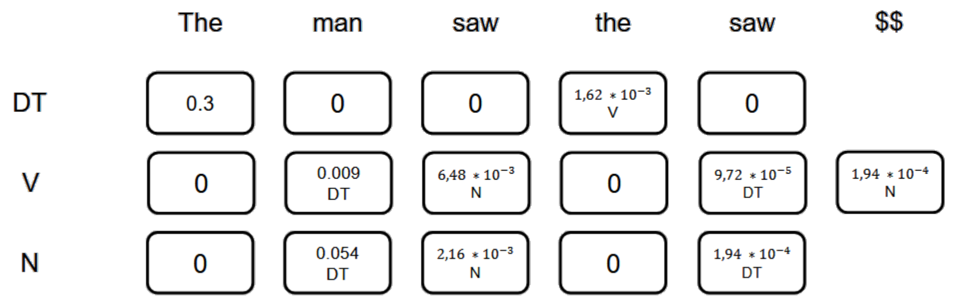
2.3 Viterbi Algorithm

Viterbis algorithm works by calculating the joint probability for each tag for each word. It uses the principles of HMM to get the necessary probabilities.

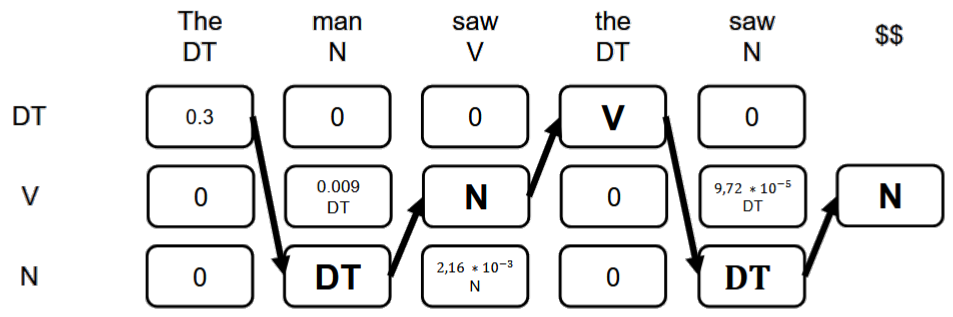
After it has done that for each word it iterates backwards through the input again and finds the most probable tag for the word.



Emission probabilities	Transition probabilities	
P (the | DT) = 0.5	P (DT | DT)= 0.1	P (DT | N) = 0.2
P (man | V) = 0.1	P (V | DT) = 0.3	P (V | N) = 0.6
P (man | N) = 0.3	P (N | DT) = 0.6	P (N | N) = 0.2
P (saw | V) = 0.2	P (DT | V) = 0.5	P (DT | q0) = 0.6
P (saw | N) = 0.2	P (V | V) = 0.2	P (V | q0) = 0.3
	P (N | V) = 0.3	P (N | q0) = 0.1



Probabilities of each tag for each token



Applied Sequence of tags

3 Information Retrieval 1

Information Retrieval concerns itself with how we can get relevant information from a corpus of data.

Relevant Terms for Information Retrieval

- Document: Can be anything (web page, word file, text file, article, etc.)
- Collection: A set of documents (Assumed static for now, makes it easier to use)
- Query: Piece of information that is searched for
- Relevance: Whether a document satisfies the information need of the user.
  - Relevance often measured by how often a term appears in a document.
  - This needs to be in conjunction with the length of the document, as the longer a document is the more likely it is to contain the relevant query more often.
  - However, this counting approach gets inefficient for queries.

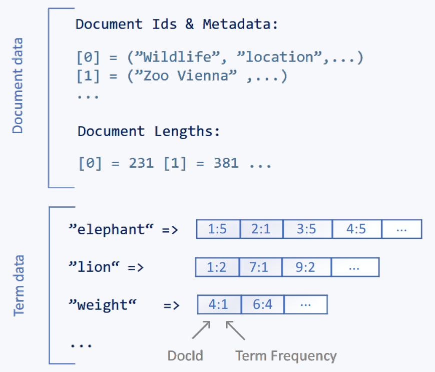
3.1 Inverted Index

The Inverted Index is a method of efficiently retrieving documents from large collections. Hereby the inverted index stores all statistics that the scoring model needs per term, such as:

- Document frequency: How many documents contain the term
- Term frequency per document: How often the term appears in the document
- Document length: Length of the document
- Average document length: Average length of all documents

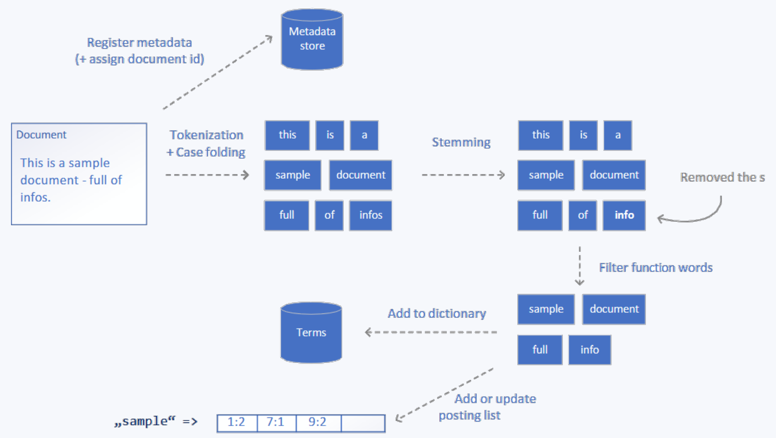
These statistics are saved in a format that is accessible by a given term. Also saves metadata of the documents (Name, location, etc.)

Inverted Index Structure



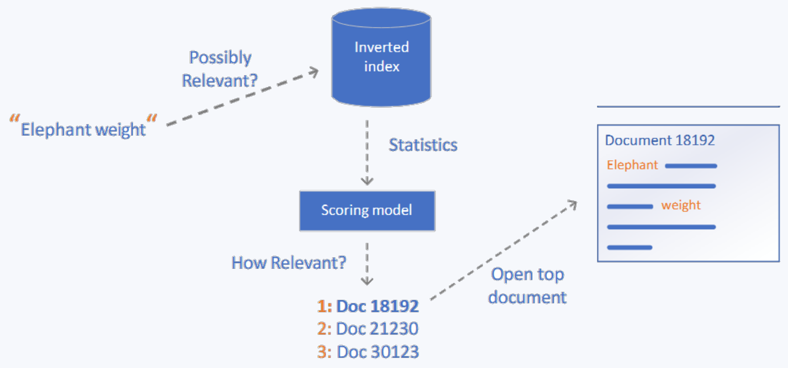
- Every document has internal document ID
- Term dictionary is saved as search friendly data structure (dictionary (HashMap))
- Term frequencies are stored in a "posting list" of doc ids and frequency pairs
  → not good for random lookups

Inverted Index Creation



- Linguistic models are language dependent
- A query text and a document text undergo the same process

Inverted Index Querying



- No need to read full documents
- Only operates on frequency numbers of potentially relevant documents
- Sort documents based on relavance

3.2 Search & Relevance

Types of queries

A query can take many forms, for example:

- Exact matching: Document needs to contain the query exactly
- Boolean queries: Uses logical operators (AND, OR, NOT) to combine queries
- Expanded queries: Incorporates synonyms and other relevant words into the query
- etc.

Queries do not have to be in text form, they can also be sound/phonetic (e.g. Music recognition, speech recognition), images (e.g. Image search), etc. We do focus on text queries for now.

Every search is based on how relevant the documents are to the query. Hereby we use a scoring model that takes inputs from the document and the query and outputs a relevance score (floating point value).

A simple search algorithm utilizing the inverted index is:

Search Algorithm

```
Function search(query, k):
    float [] scores = $\emptyset$;
    ForEach query term q in query do
        Fetch term data tf for q;
        ForEach pair(d, $tf_{t,d}$ in term data) do
            If d not in scores then
                scores[d] = 0;
            scores[d] += score(d, $\mathrm{tf}_{t,d}$);
    return k biggest entries of scores;
```

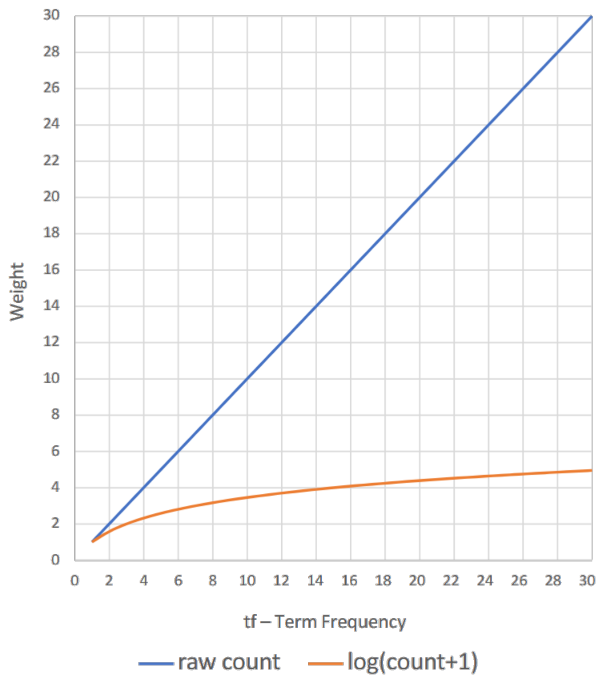
Relevance Scoring

- Relevance scoring is usually based on word occurences
- Simply counting how often a term appears in a document is not enough as it does not take into account the length of the document
- Therefore, we need to adjust the scoring accordingly.
- Relevance is often assumed to be binary. This means that a document is either relevant or not, no partials.
- Relevancy is too complex, therefore we need oversimplifications to create and evaluate mathematical models.

3.3 Term Frequency & Document Frequency Models

3.3.1 Term Frequency

Term frequency is the number of times a term appears in a document. Often denoted as $tf_{t,d}$, how often term t appears in document d. However, often relative frequencies make more sense to use than absolute frequencies (See above). Retrieval experiments show that logarithm of term frequency works well for this.



3.3.2 Document Frequency

The document frequency is the number of documents that contain a term. Often denoted as df_t. Here rare terms are more informative than frequent terms (like function words: the, or, a, etc.) Therefore for our frequency we want a high weight for rare terms.

IDF - Inverse Document Frequency

The common way to define the inverse document frequency of a term:

$$idf(t) = \log(\frac{|D|}{df_t})$$

with total number of documents $|D|$ and number of documents with $tf_t > 0$ df_t

It describes how common the term is in the corpus. The higher the *idf* the more common the term is.

TF-IDF

The TF-IDF is a combination of term frequency and inverse document frequency:

$$TF\_IDF(q,d) = \sum_{t \in T_d \cap T_q} \underbrace{\log(1 + tf_{t,d})}_{*1} \cdot \underbrace{\log(\frac{|D|}{df_t})}_{*2}$$

with query q, document d, term frequency $tf_{t,d}$, total number of documents $|D|$, and number of documents with $tf_t > 0$ df_t
$*1$: increases with the occurences of the term in the document
$*2$: increases with the rarity of the term in the collection

A rare word in the collection appearing a lot in one document creates a high score, while common words are downgraded.
TF-IDF is not only useful in document retrieval. The weights can also be used as a base for other retrieval models and also for generic word weighting mechanism for NLP

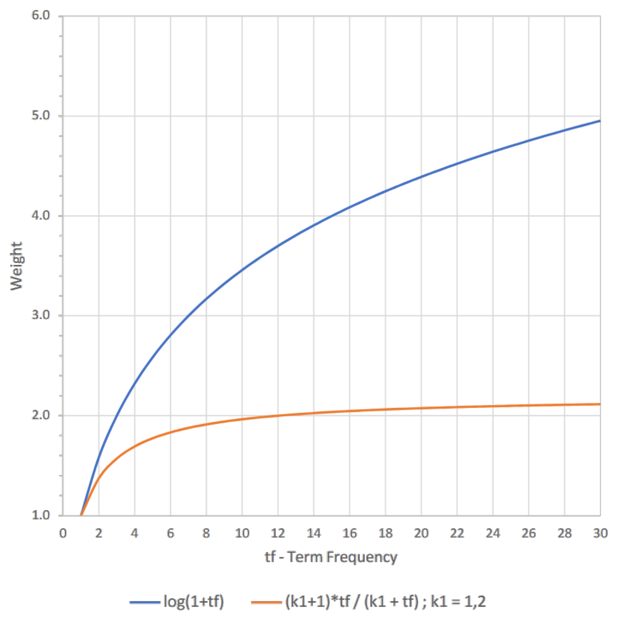
BM25

BM25 is an extension of TF-IDF.

$$BM25(q,d) = \sum_{t \in T_d \cap T_q} \frac{tf_{t,d}}{k_1((1-b) + b\frac{dl_d}{\text{avgdl}}) + tf_{t,f}} \cdot \log \frac{|D| - df_t + 0.5}{df_t + 0.5}$$

with query q, document d, term frequency $tf_{t,d}$, document length dl_d, average document length avgdl, total number of documents $|D|$, number of documents with $tf_t > 0$ df_t and hyperparameters k_1 and b

The hyperparameters k_1 and b are set by the user. Hereby k_1 controls the term frequency scaling ($k_1 = 0$ is a binary model, large k_1 is raw term frequency). b controls document length normalization ($b = 0$ is no normalization; $b = 1$ is relative frequency (full scale by document length)). Common ranges: $0.5 < b < 0.8$ and $1.2 < k_1 < 2$.



TF-IDF alwasy increases, while BM25 asymptotes at $k_1 + 1$

3.4 Evaluation

The evaluation of systems is done by observing concrete evidence for a hypothesis. It is compared to other systems on performance.

Performance can be different things depending on context:

- Result quality: Are the results correct / as expected?
- Efficieny: How fast is the system?
- Fairness, diversity, content quality, source credibility, effort, etc.
- Retrieval of Context of a Larger Goal:
  - How big is the scope of the system?
  - How well does it integrate with the web?

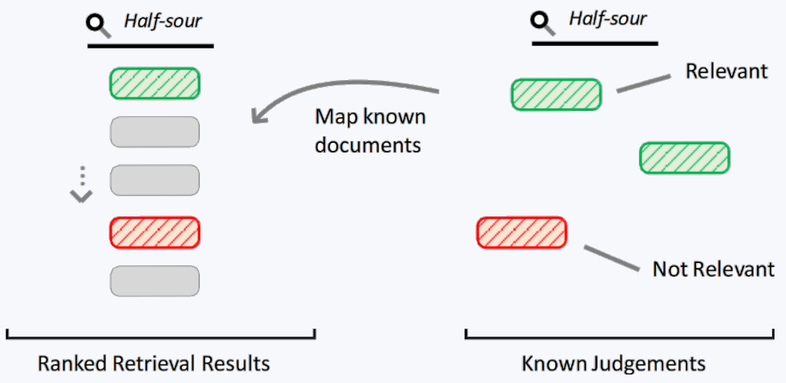
Information Retrieval Systems are hard to evaluate, due to their ambiguity (what is relavance, context, etc.) and collection size (The more the size of the collection differs from one system to another, the more the evalutation setting differs from each other).

In general there are two types of evaluation:

- **Intrinsic:** Fixed set, same collection, query and labels
- **Extrinsic:** Observe behaviour of users

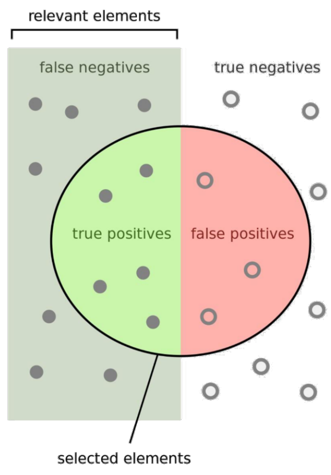
Extrinsic Evaluation Setup

- Considers the quality of systems that produce a ranked list of documents.
- Compared to a (predefined, desired) pool of judgements (not necessarily the whole list)
- Missing judgements are often considered as non-relevant

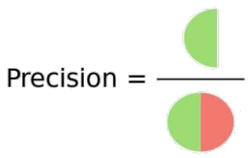


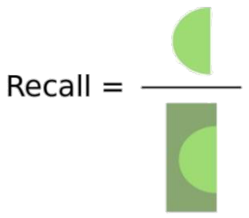
Using this scheme we can evaluate and compare systems on how well they rank documents.

3.4.1 Precision & Recall



Precision and Recall are both metrics which can be used to compare a systems performance on.





Hereby **Precision** is the ratio of relevant documents found to the total number of documents found.

Recall is the ratio of relevant documents found to the total number of relevant documents.

Building a model that increases recall or precision is fairly easy.

If a system needs less certainty to return a document it'll return more documents total. This means that it'll be more likely to return more relevant documents, but also more irrelevant documents. Since Recall is not dependent on the irrelevant documents, it'll increase Recall (This also means that a system that returns all documents will have 100% Recall). The oposite is true for precision, if you increase the certainty needed to return a document it'll increase the precision, but it'll also miss more relevant documents.

Due to this we often measure performance as the combined measure **F-Score**. It's mostly used as F_1, which describes the harmonic mean of P and R:

$$\boxed{F_1 = 2 \times \frac{P \times R}{P + R}}$$

So in general the forms look like this:

	Relevant	**Non-Relevant**
Retrieved	TP (True Positive)	FP (False Positive)
Not Retrieved	FN (False Negative)	TN (True Negative)

Precision

$$P = \frac{TP}{TP + FP}$$

Recall

$$R = \frac{TP}{TP + FN}$$

F-Score

$$F_1 = \frac{2 \times P \times R}{P + R}$$

3.5 MRR & MAP

The MRR (Mean Reciprocal Rank) and MAP (Mean Average Precision) are metrics that can be used to evaluate the performance of a system. Using these we can devise a way to rank and compare the retrieval of documents of systems.

Usually these are measured over all retrieved documents but only over the top @k retrieved documents. For MAP / Recall higher numbers like @100 or @1000 are used, while MRR / Precision are more likely to be smaller like @5 or @10.

3.5.1 MRR

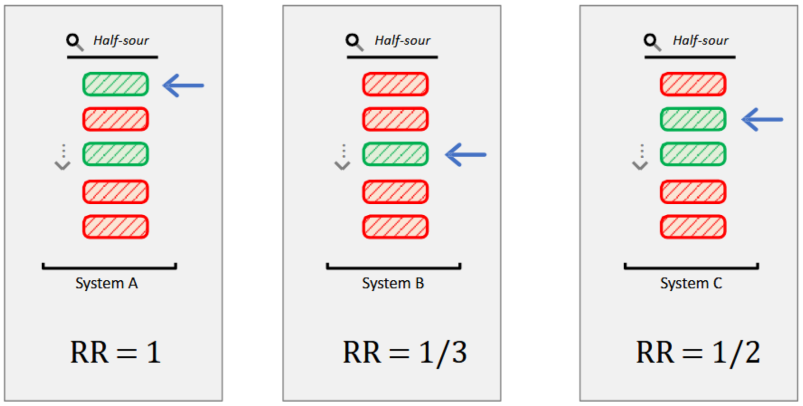
MRR: Mean Reciprocal Rank

The MRR basically looks at ranking from the perspective of a single query. It can be seen as the solution for a user that only cares about the first relevant document and doesn't care about any others.

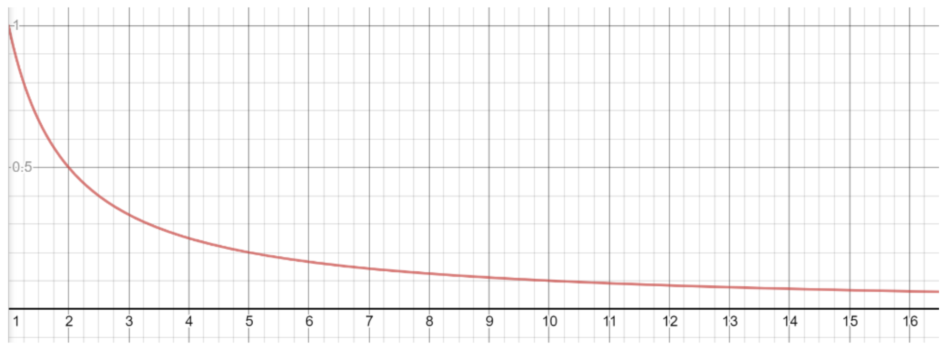
$$\text{MRR}(Q) = \underbrace{\frac{1}{|Q|}}_{\text{MQ}} \cdot \sum_{q \in Q} \underbrace{\frac{1}{\text{FirstRank}(q)}}_{\text{RR}}$$

with Query Set Q and FirstRank(q) returning the first relevant document for the query where MQ is the Mean over all queries and RR is the Reciprocal Rank

Hereby a higher result indicates a better model.
The MRR is applicable with sparse judgements or assuming users are satisfied with one relevant document.



Reciprocal Rank Example



* x is plotted continuously, but in MRR x is discrete with the position in step size of 1

Reciprocal Rank Progression

3.5.2 MAP

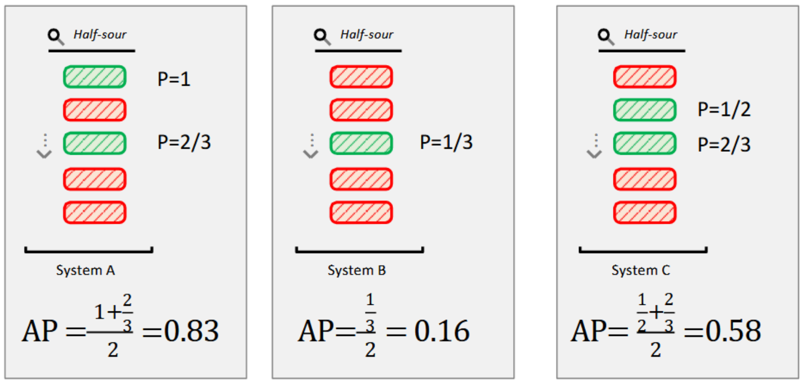
MAP: Mean Average Precision

The MAP responds to the area under the Precision-Recall curve.

$$\text{MAP}(Q) = \underbrace{\frac{1}{|Q|}}_{\text{MQ}} \cdot \sum_{q \in Q} \frac{\overbrace{\sum_{i=1}^{k} P(q)_{@i} \cdot \text{rel}(q)}^{\text{PRD}}}{\underbrace{|\text{rel}(q)|}_{\text{AP}}}$$

with Query Set Q, Precision of query after the first i documents,
relevance of doc at position i rel(q), number of relevant documents $|\text{rel}(q)|$
where MQ is the mean over all queries,
PRD is the Precision per relevant doc and AP is the Average Precision

k describes the first k results of a system and can be chosen accoding to the needs. The number of relevant docs is also defined beforehand by the annotators.
This metric also considers the relevance and order of the documents in the ranking.



Average Precision Example

3.6 nDCG

The normalized Discounted Cumulative Gain (nDCG) is another metric for ranking list evaluation. Hereby it differs from MRR and MAP because it uses graded relevance. MRR and MAP used Binary, meaning that in the model a document could either be relavant or not relevant.

nDCG works with graded relevance, meaning that a relevant document can be less or more relevant than another relevant document.

Common Graded Relevance Labels

[3] Perfectly relevant: Dedicated to the query, worthy of being a top result
[2] Highly relevant: Provides substantial information on the query
[1] Relevant: Provides some information on the query, may be minimal
[0] Irrelevant: Provides no information on the query
Relevance can also be defined otherwise. Another common approach is to use a floating point value instead of a label.

nDCG: normalized Discounted Cumulative Gain

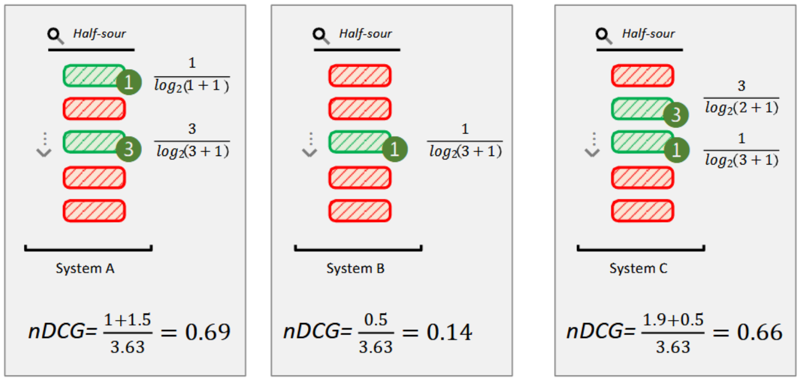
$$\mathrm{DCG}(D) = \sum_{d \in D, i=1} \frac{\mathrm{rel}(d)}{\log_2(i+1)}$$

with single document results list D and relevance grade for query-doc pair $\mathrm{rel}(d)$

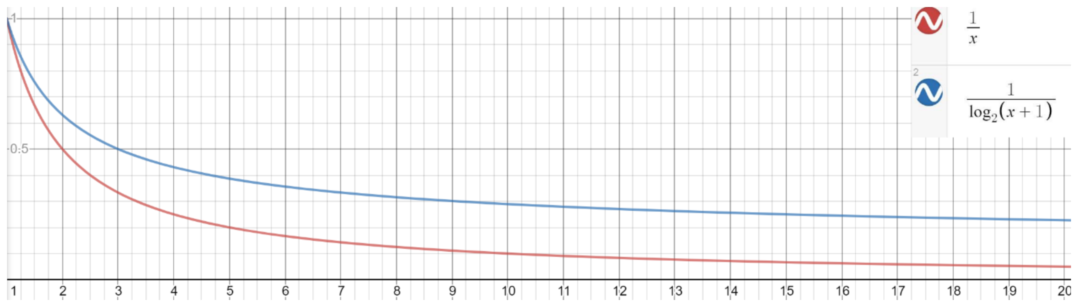
$$\mathrm{nDCG}(Q) = \frac{1}{|Q|} \cdot \sum_{q \in Q} \frac{\overbrace{\mathrm{DCG}(q)}^{\text{Actual Result}}}{\underbrace{\mathrm{DCG}(\mathrm{sorted}(q))}_{\text{Best Possible Result}}}$$

with Query Set Q and sorted list of relevance grades for a query $\mathrm{sorted}(\mathrm{rel}(q))$

$$\text{Ideal DCG} = \frac{3}{log_2(1+1)} + \frac{1}{log_2(2+1)} = 3.63$$



DCG Example



DCG vs. RR