Document Analysis

Contents

Part I

Natural Language Processing

1 Introduction

Challenges

- Complicated structure in sentence
- Syntactic ambuigities ("*time flies like an arrow*", "*get the cat with the gloves*").
- Metaphores, humor, irony, ...
- Semantics can be very rich and dependent on context, not easy to distuingish
- Language requries knowledge about the world
- Hard to really formalise the notion of "meaning"

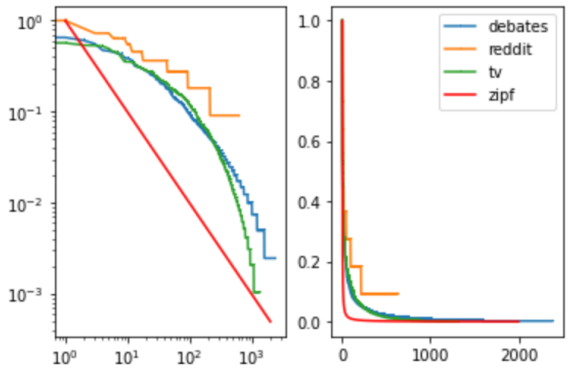
Basic approach Gather information on word based on its context. Given a large text corpus, we can assume this to be meaningful statistics.

Zipf's Law The number of elements with a given frequency follows a power law distribution. That is, there is a small number of elements which appear very often and the majority of elements appears rarely. In its most simple form, the probability of the n-th most common word $p(n)$ is

$$p(n) = 1/n$$

Elements of language We can have different points of view on language:

- Phonetics (sound)
- Grammar



  - Phonology
  - Morphology
  - Syntax
- Semantics (meaning)

Morphology How words are built up from smaller meaningful units, for instance *un-lady-like*, *dog-s*

- ***Inflection*** – variation in the form of a word (usually affix) that expresses a grammatical contrast
  - adds tense, number, person, mood, aspect, etc
  - e.g. *run→ run — running*
  - does not change word class
- ***Derivation*** – formation of a new word from another
  - e.g. nominalization (*computer→computerization*)
  - e.g. formation of adjectives (*computational*, *clueless*)
  - changes word class

Morphemes

- ***Root*** – equivalence class of a word when all affixes (inflectional and derivational) are removed; not further decomposable into meaningful elements. *The root and the suffixes are morphemes.*
- ***Stem*** – part of word that never changes when morphologically inflected, *i.e. without affixes describing tense, number, person, ...* — obtained by removing inflections
- ***Lemma*** – Base form of word
- From *produced*, lemma is *produce* but stem is *produc*

2 Tokenization

- ***Token*** A useful semantic unit of the input text, usually a single occurence of a word.
- ***Tokenization*** Split input (usually text) into sequence of tokens. Enables token/word-level analysis, first step for many other methods.
- ***Challenges***
  - Keep abbreviations, dates, numbers as single tokens
  - distuingish abbreviations from words
  - names and phrases (*queen of england*, *TU Wien*)
  - compound words
  - apostrophes, umlauts, etc and other linguistic characteristics
  - encoding issues like RTL/LTR
  - URLs, numbers, ...

Stop word removal

- *Stop words* – very frequent and semantically unimportant words
- Can obstruct/hinder analysis by skewing frequency distribution extremely (cf. Zipf's Law)
- Methods:
  - Use predefined dictionairy of stop words
  - Sort terms by their collection frequency, then remove top k terms (maybe in combination with dictionairy.)

Data cleaning *(in context of text data)* Methods such as

- *stop word removal* – remove frequent and semantically unimportant words
- *prune data* – remove unneeded, unnecessary, invalid data, metadata, formatting, capitalization, etc...
- *clean data* – fix formatting, encoding, ...
- *fix data* – fix missspelling, incorrect formatting

Maximum Matching algorithm Use a dictionary of known terms. Take the longest prefix of the input string that matches a dictionary item. Does not always make sense (*"Theta bled own there"*).

3 Stemming

Goal Map tokens to equivalence classes in order to treat different words with very similar semantics as equal (e.g. *runs* and *run*). In Stemming, do this by remove all inflectional affixes (tense, number, person, ...)

Porter Stemmer Iteratively removes suffixes of words. Applicability of rules is based on *measure* of a word w, which is the number m s.t. $w = C(VC)\{m\}V$ where C, V are arbitrary sequences of consontants, vowels, resp. — Indeed reduces the words to their *stems*.

***WordNet* MORPHY** (*Lemmatizer*)

- Has • a sophisticated set of rules about inflections • exception list
- Checks the result of transformation against an extensive dictionary
- Dont get stem but lemma! – Where PORTER gives *leav*, MORPHY gives LEAF

Note MORPHY reduces to *lemmas*, while PORTER reduces to *stems*.

- ***Over-stemming*** Two words are reduced to the same root when they should not be (e.g. *university, universal* are both reduced to *univers*)
- ***Under-stemming*** Should be reduced to the same root but are not.

4 POS-Tagging

Given some input text and some tags (usually word types such as *noun*, *verb*, etc.), want to assign tags to tokens. (*Sequence classification problem*).

Use Cases

- Parsing of grammatical structure, e.g. dependency parsing
- Spelling correction
- Sentiment analysis (e.g. *for the greater good* vs *good movie and great actors*)
- Named Entitiy Recognition (e.g. dont classify verbs as NEs)
- Disambiguation of word senses (e.g. *she saw a bear* vs *the tree will bear fruit*)

Def. Can divide words into two different classes

- ***Closed class*** — can enumerate all members, e.g. determiners, pronouns, prepositions, ...
- ***Open class*** — don't know all members, e.g. nouns, verbs, adjectives, ...

Note

- A single term (dictionary entry) can have different optimal POS-tags depending on its context.
- Tagging helps to resolve ambiguities that exist on term-level (.e.g *leaves* as NN or as VB)
- Tagging removes unnecessary distinctions e.g. all personal pronouns are PRP, determiners
- Naive method (assigning most frequent tag in training data to term) already has 90% accuracy.

Def.

- ***Informativeness*** — Assignment of tag adds information, reduces ambuigity
- ***Specifiability*** — Ease of mapping a term to a tag
- *Example*: Collapsing multiple related tags into one decreases informativeness, decreases specifiability.

Feature selection Can look at word-local features (term, pre-, suffixes, capitalization); but very often the tag of a word depends on its context in the sentence.

Main techniques

- *Probabilistic tagging* — Assign tag based on probability distributions of tags in manually tagged training data.
  - Need sufficient amount of training data – no meaningful result if no example for tag-word combination is in training data.
  - Tagging results rely on quality of training data.
- *Rule-based tagging* — use rules based on linguistic understanding
  - rules need to be created manually, need expert-knowledge and time
  - need a large number of rules to be effective
  - good to tailor solution to very specific problems

Backoff Tagger Combine different taggers (e.g. higher-order first, then lower)

4.1 Probabilistic tagging

n-gram language models Consider the definition of conditional probability

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

This gives rise to the *chain rule*

$$\Rightarrow P(A,B) = P(A|B) \cdot P(B)$$

or, more generally

$$P(w_1, ..., w_n) = \prod P(w_i \mid w_1, ..., w_{i-1})$$

The problem here is that we cannot realistically obtain all the components of the product because there are way too many possible sequences of words. Instead, we employ the *Markov assumption* that says that we can estimate the probabilities by only consiering only the k preceding terms

$$P(w_i \mid w_1 ..., w_{i-1}) \approx P(w_i \mid w_{i-k}, ..., w_{i-1})$$

For $k = 1$, this yields the *unigram model*, for $k = 2$ the *bigram model* (i.e. $P(w_i|w_{i-1})$).

n-gram modelling is insufficient because language has *long-distance depenencies*.

Unigram Tagger *(How probable is each tag for a given word?)* Assume that a unigram model generates the current tagging.

Assign a token w its most frequent tag, i.e.

$$t(w) := \text{argmax}_t P(t \mid w)$$

Use Bayes' formula to calculate probabilities, i.e. $P(A|B) = \frac{P(B|A)P(A)}{P(B)}$, omitting the quotient:

$$t(w) := P(t \mid w) = \text{argmax}_t P(t) \cdot P(w \mid t)$$

Considering the estimate the probabilities by their frequency in the training set, this can be simplified even further:

$$P(w|t) = \frac{count(w,t)}{count(t)}$$
$$P(t) = \frac{count(t)}{n}$$

where n is the number of words, equal to the number of assigned tags; when trying to find the argmax t, we dont even have to consider the denominator:

$$P(t|w) \approx P(w|t) \cdot P(t)$$
$$= \frac{count(w,t)}{n} \approx count(w,t)$$

n-gram tagger Use information about the previous n tokens in addition to information about current token. Can have *word-based* and *tag-based* (tags are more common, training data covers more ground).

Assume a bigram language model (generating the sequence of POS-tags).

Pick the tag t_i for word w that maximises

$$P(t_i \mid t_{i-1}) \cdot P(w \mid t_i)$$

For finding $P(t_i \mid t_{i-1})$, use the *Maximum Likelihood Estimate* (where c is the count of observations)

$$P(t_i|t_{i-1}) \approx \frac{count(t_{i-1} \text{ then } t_i)}{count(t_{i-1})}$$

$P(w|t_i)$ same as in unigram model

- (Use start and end symbols to be able to calculate probs for item first and last words)
- Problem: Large memory requirements to store n-grams (very sparse data structures)

Note These sequence-based taggers/models have the problem that if the tag for the preceding word cannot be determined, this influences the performance for the current word.

4.2 Rule-based tagging

Try to incorporate linguistic insight.

Feature-based tagger Only consider features of single word to be lagged. Look at word, prefixes, suffixes, capitalisation.

Brill tagger

1. Tag each word using a *baseline tagger* (e.g. unigram tagger, i.e. most common tag)
2. Apply patches that improve the result
   - e.g. *if one of the two preceding words is a determiner, change the tag from verb to noun*
   - Based on training data, compute the error between any two should-be/is-assigned: $(t_a, t_b, freq)$.
   - For each error triple, apply the patch that results in the greatest improvement, apply it.
   - Repeat until no further improvement is possible.

   The patches are learned from a training corpus.

5 Parsing

We try to determine the grammatical structure of a sentence.

- *Constituency parsing* yields parse tree according to a given formal grammar – useful when subtrees are interesting (e.g. spelling correction)
- *Dependency parsing* only yields pairwise relationships between words – useful when these are interesting, e.g. we want to identify modifier words (sentiment analysis, question answering)
- Top-down approach – Try to generate sentence
- Bottom-up approach – Try to contract tokens into nonterminals

Parsing problem Identify the parse tree (w.r.t some grammar) for a given sentence. *(There may be multiple parse trees, or none.)*

Parse tree depicts derivation of sentence beginning from start symbol. Obviously requires context-free grammar *(each set of children has exactly one parent)*

Motivation Grammar checking, Question answering, Machine translation, ...

Ambiguous grammar has more than one *leftmost* derivation parse tree.

5.1 Constituency parsing

Basic assumption Language is made up of *constituents*, i.e. basic, nested building blocks *(terminals and nonterminals of a formal grammar.)*

Leftmost derivation *(top-down)* Always expand the leftmost expandable nonterminal. Potentially lots of backtracking applied. Always yields unique parse tree (if grammar is unambiguous).

Shift-reduce parser *(bottom-up approach)* Push tokens onto stack until top of stack matches the *rhs* of a rule, then reduce (on stack). Accepts the word if start symbol alone is left on the stack.

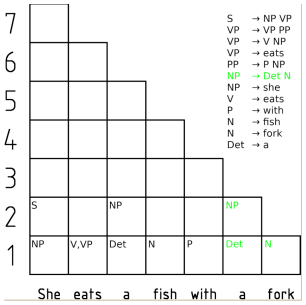
Chomsky normal form The *rhs* of a rule is of one of these shapes

- $A \to BC$
- $A \to a$
- $S \to \varepsilon$

TODO transform grammar into CNF

CYK Parser *(dynamic programming, bottom-up)* Assumes a grammar in CNF, i.e. *rhs* of rules is two nonterminals or a terminal. From the bottom up, in a dynamic programming fashion, remember if two subcells are the *rhs* of a production rule. The topmost cell produces the entire sentence. Note that this finds all possible parse trees. *(→ Assignment 5)*

- Potentially quadratic space complexity
- Finds all possible parse trees (and all possible parses for substructures) which might not always be needed.



5.2 Statistical Parsing

Motivation: Sentences can have many different parse trees but some are clearly more likely than others.

Basic idea

- Associate rules of a grammar with probabilities
- The probability of a parse tree is the product of all used productions/derivations.
- Probability of a sentence is the sum of all possible parse tree probablities.
- Probabilities can be learned from a training corpus.
- Can use probabilites in parsing algorithms such as the CYK parser to find the most likely parse tree.

Lexicalised Parsing Extend production rules to be specific to head word of sentenc (e.g. *VP(ate)* $\to$ *VP(ate) PP(with)*). Then, again, assign/learn probabilities. TODO Head of sentence

Shallow Parsing Simply break up text into (nested) chunks.

5.3 Dependency Parsing

Basic concept Dependencies (in the linguistic sense) between tokens (such as *determiner, subject, ...*), these form a tree. Again, want to find the most likely parse/dependency structure

Constraint-based Dependency Parsing Come up with rules describing what a dependency can possibly look like. Begin with a complete graph of pairwise dependencies, then iteratively eliminate dependencies according to rules.

Global linear models Assign a *local feature vector* $g(x, h, m)$ to a specific dependency (h, m) in a specific sentence x. Try to maximise the (weighted) sum of all depenencies in the sentence,

i.e. the best parse $y = F(x)$ is given by

$$F_y(x) = \text{argmax}_y \vec{w} \cdot \sum_{(h,m)\in y} g(x,h,m)$$

where $\vec{w}$ is a weight vector for components of g; for binary features it can be derived from a given set of training parses by $\frac{\#\text{ matching arcs for } i}{\#\text{ total arcs}}$. g_i are features/scorings for a given input/test sequence.

- Advantage: nicely generalisable, can e.g. incorporate semantics of word into feature vector.
- Possible features/scorings:
  - Dependency is between POS-tags that are likely related, e.g. noun and verb
  - Dependency is of short range

Perceptron algorithm As per Assignment 6, this is simply accepting the dependency structure with the higher score $F_y(x)$.

Part II

Similarity Measures

As there are many different expressions for the same semantic concept, we want to derive concepts of similarity between words. Further, this is useful in spelling correction. Focus on phonological and spelling similarity.

6 Word similarity

Phonological similarity How much do two words sound alike?

Soundex Reduce a given term/word into a phonetic representation based on rules.

Morphological similarity basically comes down to stemming.

Spelling similarity Mainly want to identify spelling errors

Levenshtein distance / edit distance (*dynamic programming*) We already know this.

- Advantages: simple, easy to implement, finds all optimal alignments, flexible by employing different selection mechanism in dynamic programming.
- Quadratic time and space complexity in its simple implementation
- Character-based, does not consider local or sentence-scope context.
- Extensions: Cost matrices, based for example on distance of letters on keyboard.

Semantic similarity

- *Synonym* – same meaning. A *synset* is the equivalence class of synonymity.
- *Antonym* – opposite meaning
- *hypernym* – more general meaning
- *hyponym* – more specific
- *meronym* – part of collective

WordNet provides a tree (forest) of semantic relationships between words. Similarity between v, w can then be defined e.g. TODO

7 Document similarity

Vector space model See IR lecture notes.

tf-idf weight

- *term frequency* $tf(t,d)$ — How often does term t appear in document d? Commonly dampened by logarithm.
- *document frequency* $df(t)$ — In how many documents does t appear?
- *inverse document frequency* $idf(t)$ — The basic idea: Rare terms are more informative than frequent terms and should thus receive a high score. Thus we do the inverse fraction and log-dampening:

$$idf(t) := \log \frac{n}{df(t)}$$

The tf-idf weight is given by the product of term frequency and inverse document frequency:

$$tf.idf(t,d) = \log(1 + tf(td)) \cdot \log\left(\frac{n}{df(t)}\right)$$

- term frequency increases with the number of occurences in the document
- inverse document frequency increases with the rarity of the term in the collection

The score for a multi-term query and a document is

$$score(q,d) := \sum_{t \in q \cap d} tf.idf(t,d)$$

Problems:

- Bag-of-words model, does not consider order
- Score is a sum, longer documents receive higher score

Vector space model Each document is a vector consisting of tf-idf weights of terms. *Distance measure*:

- euclidean distance is not suited because it considers the lengths (norms) of vectors, but we only really care about the distribution of terms
- hence use *cosine similarity*, which measures the angle between vectors. TODO calculation

Cosine similarity Consider the formula for the euclidean dot product:

$$a \cdot b = ||a||||b|| \cos\theta,$$

this yields for two vectors a and b

$$\text{sim(a,b)} = \cos\theta := \frac{a \cdot b}{||a||||b||}$$

where $a \cdot b = a^T b$ is the dot product, $||a|| = a \cdot a = a^T a$

Part III

Language Modelling

Basic pipeline

1. Corpus
2. high-dimensional vector space
3. lower-dimensional latent space (word embeddings)
4. word relationships

Word embedding "Embed" words in a vector space V s.t. similar word-vectors are similar w.r.t some measure in V. Motivation: Approaches like WordNet are static and thus insufficient. Possible information

- syntactic similarity (*run, ran*)
- semantic similarity (*large, big*)
- relatedness (*coffee, cup*)

Word context matrix (*co-occurence matrix*)

- Each row represents a word
- Each column represents some "context" (specific entity or other word)
- cell represents the strengh of association, for example *pointwise mutual information*

Basically the word-vectors put next to each other. This matrix is very sparse. We'd like to find a low-dimensional representation of our word vector (word embedding). We project into a lower-dimensional space, the so-called *latent space*.

Word2Vec *(neural network for learning word vectors, context-independent model)* Given a training corpus, every word in a fixed vocabulary is represented by a vector.

1. For each center word c and a context window of fixed size o...
2. use the word vector similarity of c and o to calculate $P(o|c)$ (or $P(c|o)$, see below)
3. Adjust word vector to maximise predictive accuracy

- *Continous Bag-of-Words* (CBOW) learns $P(c|o)$, i.e. focus word given the context
- *Skip-gram* learns $P(o|c)$, i.e. context given some focus word.

Limitations:

- Any given word is represented by a vector which has to be stored

- Context-independent, outputs always only one vector for a word: Polysemy (different meanings of same word) is not adressed at all
- Dependence of meaning on context is not considered (gives rise to *context-dependent models*)

BERT example for a context-dependent model, based on a neural network.

Given some word embeddings, how do we determine their similarity?

- similarity measures like cosine similarity
- visualise by dimension-reduction techniques like t-SNE, PCA, MDS, UMAP, ...
- *context-independent models* output a single word vector for a word
- *context-dependent models* generate multiple word embeddings for a word that capture different contexts.

Sentence embedding Basic approach with Bag-of-words assumption, sentence is represented by vector of tf-idf weights

- Motivation: Meaning of words is often defined by the entire sentence, this takes these larger structures into account
- Problems:
  - vectors are large and very sparse
  - cosine similiarity depends only on components, i.e. single words. Sentences might have a $\cos\theta$ of 0 but still be semantically similar.
- Alternatively, to represent a sentence, use the average of all its word embeddings. Disadvantage: Long sentences have less meaningful average.
- Deep learning approaches

Applications

- Assistants like Google, Siri, Alexa, ... for understanding an generating language, answering questions, ...
- Opinion mining
- Sentiment analysis
- Named Entity Recognition

Limitations Language models potentially contain biases (ethnic, gender) induced by the training data.

Part IV

Text Data Mining

8 Text classification

Applications

- categorise web pages
- spam vs. non-spam
- categorisation of correspondence
- categorisation of news articles
- classify to author
- classify reviews
- easy/hard to read, suitability for age groups
- writing style

Choice of classification algorithm is dependent in characteristics and amount of training data.

Feature selection

- tf-idf weights of terms
- phrases, characters, n-grams
- POS-tags
- metadata about the document
- non-text parts of document

As always, have a rule-based and a machine learning approach

Naive Bayes *(for text classification)* Assumes bag-of-words (independence between words). For a class c and words w_i, find the predicted class y by

$$y = \text{argmax}_c P(c) \prod_i P(w_i \mid c)$$

For individual probabilities use maximum likelihood estimate, i.e.

- $P(c) = \frac{\text{\# of training docs with class } c}{\text{\# all docs}}$
- $P(w_i|c) = \frac{count(w_i, c_j)}{\sum_w count(w, c_j)}$

Important: Use laplace smoothing here, i.e. with $|V|$ being vocabulary size:

$$P(w_i|c) = \frac{count(w_i, c_j) + 1}{\sum_w count(w, c_j) + |V|}$$

9 Text clustering

Want to put documents into groups that are similar within and dissimilar across groups. Usually unsupervised, i.e. no labels given in advance.

Feature selection Similar to classification.

10 Topic modelling

Possible improvements/preprocessing:

- Only keep nouns (per POS tag)

Latent Dirichlet Allocation (LDA) Try to find distributions that characterise topics (distributions over words) that, in some mixture, best generate/explain a given document. In other words, assume the document to come from a distribution that is a mixture of topic distributions.

- Need to give a fixed number of topics as parameter
- A single keyword can be part of multiple topics (with a specific probability each) because a topic is a distribution over words.
- Compared to basic text clustering, topic modelling yields probabilities for mapping document $\mapsto$ topic, clustering (most of the time) yields binary assignment. Clustering via tf-idf vectors only considers word frequencies whereas LDA considers correlations TODO
- Disadvantage: Does not consider word order
- Should do stop-world removal, lemmatization because these skew the distributions

Part V

Opinion Mining & Information Extraction

11 Information Extraction

To extract and structure relevant information from unstructured (text) data. In other words: the curation of structured representations that are easily accessible while maintaining semantic information of original text.

Here, we focus only on factual information (*What year was Francis Bacon born in?*) and not opinions.

11.1 Named Entity Extraction

Used for • summarizing text • answering questions • integrating into knowlege bases • associating information (e.g. opinions) to sentiments (e.g. of parts of printer in question)

Possible types of entities • location • time • person • organisation etc. which can stand in various relations to each other.

Supervised learning models Based on labelled training sequences (of tokens), train a classifier to predict labels (*Sequence Labelling Problem*)

- new data must fit training data
- time-consuming

To make this easier, use features that go beyond single tokens, e.g. context window of k words.

Sequence Labelling • reminiscent to POS-tagging • assuming that label is dependent on context. Typical models are • Markov models • Conditional Random Fields • Bidirectional LSTMs

Once we have identified the entities, we'd like to find relationships between them (e.g. triples of operators *is-a*, *daughter-of*, ...) *(Can save these triples in a knowledge base e.g. for question-answering; cf* RDF*-triples)*.

Relationship Extraction Try to find type of relationship between two entities. Possibilities

- Extract RDF triples from large corpora like Wikipedia
- Use (specialised) ontologies / knowledge bases (for ex. medical applications)

Methods to extract information:

- handwritten rules – e.g. *"Y such as X", "X, especially Y", "X, including Y" all express an is-a-relationship*. – there can be more specific relations that only make sense between certain types of entities (e.g. *cures(drug, disease)*) – pros: • precise • can be tailored to specific domains – cons: • low recall • high effort
- supervised,
- unsupervised machine learning.

semi-supervised learning: extract less common patterns based on training corpus?

11.2 Entity Relation Extraction

12 Opinion Mining

12.1 Opinion Mining at Document level

12.2 Attribute-based Opinion Analysis

12.3 Lexicon construction

13 Beyond Sentiment

Part VI

Question Answering & Text Summarization

14 Question Answering

14.1 Information-retrieval-based factoid question answering

14.2 Knowledge-based question answering

15 Text Summarization for Question Answering

15.1 Single-document

15.2 Multi-document