Web search and Text Analysis 概念对比

Stemming and Lemmatisation

- Both stemming and lemmatisation are mechanisms for transforming a token into a canonical (base, normalised) form. For example, turning the token walking into its base form walk.
- Both operate by applying a series of rewrite operations to remove or replace (parts of) affixes (primarily suffixes). (In English, anyway.)
- However, lemmatisation works in conjunction with a lexicon: a list of valid words in the language. The goal is to turn the input token into an element of this list (a valid word) using the rewrite rules. If the rewrite rules can't be used to transform the token into a valid word, then the token is left alone. (For example, the token lemming wouldn't be transformed into lemm because the latter isn't in the word list.)
- Stemming simply applies the rewrite rules, even if the output is a garbage token (like lemm).

Inflectional morphology and Derivational morphology

- Inflectional morphology is the systematic process (in many but not all languages) by which tokens are altered to conform to certain grammatical constraints: for example, if the English noun teacher is plural, then it must be represented as teachers. The idea is that these changes don't really alter the meaning of the term. Consequently, both stemming and lemmatisation attempt to remove this kind of morphology.
- Derivational morphology is the (semi-)systematic process by which we transform terms of one class into a different class (more on this next week). For example, if we would like to make the English verb teach into a noun (someone who performs the action of teaching), then it must be represented as teacher. This kind of morphology tends to produce terms that differ (perhaps subtly) in meaning, and the two separate forms are usually both listed in the lexicon. Consequently, lemmatisation doesn't usually remove derivational morphology in its normalisation process, but stemming usually does.

Term-document matrix and Inverted index

- Assuming Term-at-a-Time processing (not WAND) some kind of TF–IDF vector space model using cosine similarity, using accumulators to keep track of each document's score:
- For the term–document matrix, we will need to read the TDM value of every document for each query term. (Note that shortcuts — like skipping documents with a term weight of 0 — save little time here.)

- For the inverted index, once more we only need to examine each entry in each postings list once; except that, here we incrementing the accumulator weights rather than comparing documents identifiers.
- Algorithms such as WAND can further improve processing of the inverted index lists.

Query expansion and Relevance feedback

- Query expansion improves query recall by dealing with the vocabulary mismatch problem where terminology in the query and terminology in the document collection don't match (e.g. poison vs toxin).
- Relevance feedback is one way to expand/reformulate the query by incorporating feedback into the query process. There are many different ways to incorporate and retrieve relevance feedback (discussed below).
- However, there are other query expansions methods such as using a thesaurus or wordnet which do not require relevance feedback.

Synonyms and Hypernyms

- Two words are synonyms when they share (mostly) the same meaning, for example: snake and serpent are synonyms.
- One word is a hypernym of a second word when it is a more general instance ("higher up" in the hierarchy) of the latter, for example, reptile is the hypernym of snake (in its animal sense).

Hyponyms and Meronyms

- One word is a hyponym of a second word when it is a more specific instance ("lower down" in the hierarchy) of the latter, for example, snake is one hyponym of reptile. (The opposite of hypernymy.)
- One word is a meronym of a second word when it is a part of the whole defined by the latter, for example, scales (the skin structure) is a meronym of reptile.

Skip-gram model and CBOW model

- In short — the element in the condition of the posterior probability: skip– gram models analyse the probability of the context words given the target word; CBOW models analyse the probability of the target word given the context words.
- Another way of looking at this is how we lay out the term–term matrix (before, say, SVD): do we label the target words on the row, and contextual words on the columns, or vice versa? (Which one is which?)

N-gram language model and Feed-forward neural language model

- n-gram language models and FFNNs share the same setup of using a Markov chain.
- They factorise the probability of a sentence into the probability of each word given the n – 1 previous words. The models differ in how this word-based probability model (classifier) is formulated.
- n-gram models can be considered a ”feature-based” model where every ngram is a feature, with a corresponding weight (thus the ”weights” for a bigram models form a matrix, for a trigram model a 3d tensor etc.)
- FFNNLM use an embedding and un-embedding step, to limit the model size and force generalisation (e.g., to locate synonymous words near in vector space), along with more complex functions to couple context to the next word.
- • Another key difference between n-grams and FFLM is that n-grams work over highly sparse data (1-hot word vectors, sparse parameter matrices where more entries are 0), while FFLM work over dense representations

Recurrent neural network (RNN) and Feed-forward language model

- Recurrent means that model is structured such that it can be repeatedly applied for each item in a sequence. The recurrence in a RNN is over the hidden states, such that each as new input is fed into the model, this is used to formulate a new hidden state – as a non-linear transformation of the last hidden state, and the new input. In this way the approach can (in theory) represent long-distance phenomena in the sentence, over variable distances.
- A FFLM instead assumes a fixed sized context, which can be applied using a ”sliding window” over a sequence. The hidden state is a function of the n-1 inputs. There is no reuse of computation from previous applications to the sequence.
- RNNLM can capture long-distance dependencies, while FFLM cannot. For example, it can balance quotes and brackets over long distances.

Relation Extraction and Name Entity Recognition

- Relation Extraction attempts to find and list the relationships between important events or entities within a document.
- • Relations typically hold between entities (e.g., MP-for(Turnbull, Wentworth)), so in order to extract relations you first need to do NER to extract the entities from the text (e.g., Turnbull, the member of Wentworth, said ...; where the bolded items would be tagged using an NER system.)

Regular grammar and Regular language

- A language is a set of acceptable strings and a grammar is a generative description of a language. Regular language is a formal language that can be expressed using a regular expression. A language is regular if and only if it can be generated by a regular grammar.
- Regular grammar is a formal grammar defined by a set of productions rules in the form of A → xB, A → x and A → , where A and B are non-terminals, x is a terminal and is the empty string.

Probabilistic parsing and Chart parsing

- Parsing in general is the process of identifying the structure(s) of a sentence, according to a grammar of the language.
- In general, the search space is too large to do this efficiently, so we use a dynamic programming method to keep track of partial solutions. The data structure we use for this is a chart, where entries in the chart correspond to partial parses (licensed structures) for various spans (sequences of tokens) within the sentence. Probabilistic parsing adds real-valued weights (probability) to each production in the grammar, such that parse trees can be assigned a score, namely the product of the probabilities of the productions in the tree. This is important as it allows for discrimination between likely and unlikely parses, rather than just provide a list of all parses, as in standard chart parsing. This affects the algorithms as they need to track the maximum probability analysis for each span, rather than the set of grammar symbols. However the parsing algorithms are very closely related.

CYK parsing with grammar and HMM's Viterbi algorithm

- The first step is to write the probability assigned by the HMM to the tagged sentence.
- This can be drawn as a chain with the tag-tag transitions as the “spine”, with each observation as a leaf. The probabilities of initial / transitions / observations can be attached to each edge. Overall this gives a right-branching tree.
- If we treat this tree as a “parse tree”, then each clique from the tree (parent and direct children) can be treated as a CFG rule, parent → left-child rightchild. E.g., NNP → Donald VBZ.
- The score for this rule would be ANNP,VBZ × ONNP,Donald. Note that we could use different grammar symbols, such that each production maps to a single HMM component. E.g., split the above rule to NNP’ → NNP VBZ (ANNP,VBZ); and NNP → Donald (ONNP,Donald); where the “prime” version of the tag is a newly introduced grammar symbol.
- CYK parsing under this grammar will do the same as Viterbi in the HMM, but will waste effort assigning analysis to all word spans in the sentence, while Viterbi effectively only considers spans that start at the first word of the sentence.

Dependency parsing and CYK parsing

- The connections are a little tenuous, but let's see what we can come up with:
    - Both methods are attempting to determine the structure of a sentence; both methods are attempting to disambiguate amongst the (perhaps many) possible structures licensed by the grammar by using a probabilistic grammar to determine the most probable structure.
    - Both methods process the tokens in the sentence one–by–one, left–to– right.
- There are numerous differences (probably too many to enumerate here), for example:
    - Although POS tags are implicitly used in constructing the "oracle" (training), the depedency parser doesn't explicitly tag the sentence.
    - The transition–based dependency parser can potentially take into account other (non–local) relations in the sentence, whereas CYK's probabilities depend only on the (local) sub-tree.
    - CYK adds numerous fragments to the chart, which don't end up getting used in the final parse structure, whereas the transition–based dependency parser only adds edges that will be in the final structure.