Text segmentation

Ambiguity

Word means two different things, confusing for algorithm.

Contextuality

Extract context from surroundings (“Are you coming? I have to study” where “I have to study” implies they are not coming).

Multilinguality

There are a lot of different languages

Combinatorial explosion

Rapid growth due to combinatorial consideration. For example, words can be part of multiple word classes, therefore a sentence might have two or more valid structures

Tokenization

Creating tokens out of full text

Word tokens

All word tokens, even duplicates

Word types

Unique words

Normalisation

“lowercasing”, harmonizing spelling variants (color/colour), suffix removal (wanted → want)

Stop words

Common words to filter out, don't add much to context. “The”, “a”, etc.

Morpheme

Smallest parts you can break up words into. One root morpheme and 0 or more affixes (draw, draw+s, draw+ing+s).

Lexeme

All words with the same base meaning (run, ran, running).

Lemma

The lexeme you'd find in a dictionary (run, in this case).

Part-of-speech

Word classes. Verb, adjective, noun, proper noun, etc.

Constituent

A part of a sentence that can be replaced with something else ("he read *the book*" → "he read *it*").

Syntactic head

Describes constituent ("warm water" is about water). Determines internal structure and external distribution.

Phrase structure tree

Describes sentence structure with a tree (sentence → subj-part + verb-part). Syntax trees.

Dependency tree

Shows what word is dependent on what by drawing arrows.

Treebank

Describing how words fit together. Examples: phrase structure tree, dependency tree.

Supervised machine learning

System has access to input and output.

Regression → predict numerical value given input (housing prices)

Classification → Predict which of k classes some input belongs to (parliament speeches)

Unsupervised machine learning

Has access to input but not output.

Clustering → N/A

Topic models → Put words in classes, see how many words of each class in text. "This text is 50% sports-related".

Text classification

Accuracy

Percentage of correctness.

Diagonal / whole

Precision

"If the system predicted c, how accurate is it?"

Exact match / column

Recall

"If the input is c, how often is the system correct?"

Exact match / row

Naive Bayes classifier

Bag of words model

For class c: score(c) = P(c) * P(every word in input | c)

Pick class with highest score.

Maximum likelihood estimation

Naive Bayes:

- P(c) = Probability of input being class c without looking at text
- P(w | c) = Probability of word w appearing in a document of class c.

Additive smoothing

Add k to every probability → fix issue with multiply by 0 for unknown words.

Evaluate a text classifier based on accuracy, precision, and recall

See above

Apply the classification rule of the Naive Bayes classifier to a text

Calculate score for each class, pick class with highest score.

score(class) = P(class) * P([every word] | class)

Learn the probabilities of a Naive Bayes classifier using maximum likelihood estimation and additive smoothing

P(c) = count(documents classified as c) / count(documents)

P(w | c) = count(w in documents classified as c) / count(all words in documents classified as c).

Language modelling

N-gram model

Generate sequence of words, looking N-1 words back.

next word = highest P([all words] | n-1 previous words)

Maximum Likelihood Estimation

P(w) = count(w) / count(all)
P(w | u) = count(uw) / count(u) → P("rights" | "your") = count("your rights") / count("your")

Additive smoothing

Add k to every probability → fix issue with multiply by 0 for unknown words.

Perplexity

2^entropy

Entropy

Probability high or low.

Count probabilities as negative log probabilities: surprisal.

Learn an n-gram model using additive smoothing

P(w) = count(w) + k / count(all) + (k * count(unique))

P(w | u) = count(uw) + k / count(u) + (k * count(unique))

Evaluate an n-gram model using entropy

-(1/count(all)) * log2(P(x1, …, xN)).

Part of speech tagging

Part of speech

A category of words that play similar roles within the syntactic structure of a sentence.

Part of speech tagging

Part of speech tagger = program that tags each word in sentence with its part of speech.

Can be approached using supervised learning (requires training data).

Ambiguity (words can have different tags) → combinatorial explosion

Accuracy

Diagonal / whole

Precision

Exact match / column

Recall

Exact match / row

Hidden Markov model (HMM)

Words have probabilities tied to each of its tags (jag → NN, jag → PN)

Tags have probabilities for its next tag (NN → VB)

HMM has two probabilities: transitional (tag2 given tag1) and output (word given tag).

Transitional first, then output at every junction.

P(VB | PN) → amount of PN followed by VB / all occurences of VB

P(jag | PN) → amount of jag when PN / all words that are PN

Multi-class perceptron

Feature window

HMM looks back once; might want to look further, or look forward. But dont want to see too much (efficiency).

Need a feature window. Feature window sees x in front and x in back of the current word.

Evaluate a part-of-speech tagger based on accuracy, precision, and recall

Compute the probability of a tagged sentence in a hidden Markov model

Probability of tagged sentence → product of transition and output (transition * output)

Syntactic Analysis (wildcard)

Phrase structure tree

Sentence divides into parts (S → Noun Phrase, Verb Phrase), which in turn divide into parts (NP → Pro → “I”, VP → Verb, NP).

Dependency tree

“This word depends on that word”. Verbs have subjects and objects, etc.

Probabilistic context-free grammar (PCFG)

Words within sentences form phrases:
“Kim read [a book]”, “Kim read [a very interesting book about grammar]”

Syntactic head → most important word in sentence.

Context free grammar → Phrases combine. How to combine? Context free grammar!
Example: Sentence → NP, VB (Basically BNF)

Probabilistic → Number of trees grows exponentially with length of sentence. Not all parse trees are relevant, only most probable.

PCFG → Every rule R has probability P(R), and sum of all P(R) with same left side is 1.

Tree probability = product of all P(R)

Transition-based dependency parser

Contains: buffer, stack, tree

Operations:

- Shift transition → Pop buffer, push to stack
- Left arc transition → Dependency from top of stack to second top, remove second top.
- Right arc transition → Dependency from second top of stack to top, remove top.

Terminate when buffer is empty and stack has 1 or less elements

Learn a probabilistic context-free grammar from a treebank

Estimate rule probabilities → count of specific rule / count of all rules with same left side.

Simulate a transition-based dependency parser

Stack → ← buffer

[] [I booked a flight from L.A.]

I booked a flight from L.A.

EVENTUALLY

[booked] []

```
          _____
___     |  __|  ___  ____
v   |    | v |v|   v|    v
I booked a flight from L.A.
```

Semantic Analysis (wildcard)

Word sense

Lexeme → set of words, same fundamental meaning (run, runs, ran → lexeme RUN)

Lemma → Lexeme you'd put in a lexicon

One lemma → multiple lexemes (word senses)

Homonymy

Same pronunciation/spelling, different meaning

Polysemy

Two senses of a lemma are semantically linked.

Synonymy

When two senses of two different lemmas are (nearly) identical.

If can substitute word A with word B without changing the meaning of the sentence, A & B are synonymous.

Antonymy

Opposite of synonyms. A & B are opposites.

Hyponymy

More specific (car → vehicle, mango → fruit).

Hyponym is the lower word in the word tree.

Hypernymy

Less specific (furniture → chair, fruit → mango).

Hypernym is the upper word in the word tree.

WordNet

Website. Three databases: nouns, verbs, adjectives + adverbs.

Each lemma has synset, a set of one or more senses.

Simplified Lesk algorithm

Given word in a context + number of senses for word.

Textual overlap of non-stopwords between context and sense → score of sense.

Word similarity

How similar is word A to word B? Synonym is boolean relation, want numeric representation.

Distributional hypothesis

Distance between two word senses by finding words with similar distributions in a corpus.

Represent words as vectors.

Co-occurrence matrix

		context words						
		crown	throne	reign	Sweden	match	goal	play
target words	queen	4	1	1	2	0	0	0
	king	3	2	1	3	1	0	0
	soccer	1	0	0	4	3	4	2
	hockey	0	1	0	1	2	1	1

Simulate the Simplified Lesk algorithm

Count non-stopword similarities between context and senses, take highest count.

Compute the path length-based similarity of two words

similarity = (word1, word2) → return 1 / (1 + pathlength(word1, word2));

pathlength = number of edges in shortest path between word1 and word2.

pathlength = Basically count the number of words you meet along the way minus the original word.

Derive a co-occurrence matrix from a document collection

Each cell → number of documents in which target word (row) co-occurs with context word (col).