Computational Syntax

Jeremy Barnes
HAP/LAP Master

10.01.2022

Quick introduction

About me

- Originally from Texas
- Undergrad in Linguistics: University of Houston
- Masters in Linguistics: Universitat Pompeu Fabra
- PhD Universitat Pompeu Fabra - Comp. Lingusitics $\rightarrow$ NLP
- Postdoc at University of Oslo - NLP

What about you?

Finite state NLP

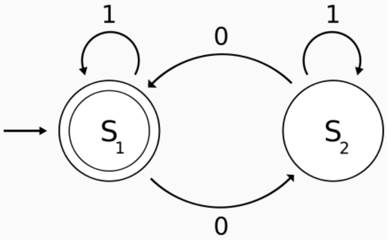
What are finite-state approaches to NLP?

- The application of a branch of mathematics
- ...the regular branch of automata theory
- ...to a branch of computational linguistics in which what is crucial is (or can be reduced to)
- ...properties of string sets and string relations.
- Problems which have a notion of bounded dependency.
- In summary: simplified, tracktable approaches to intractable problems

Applications

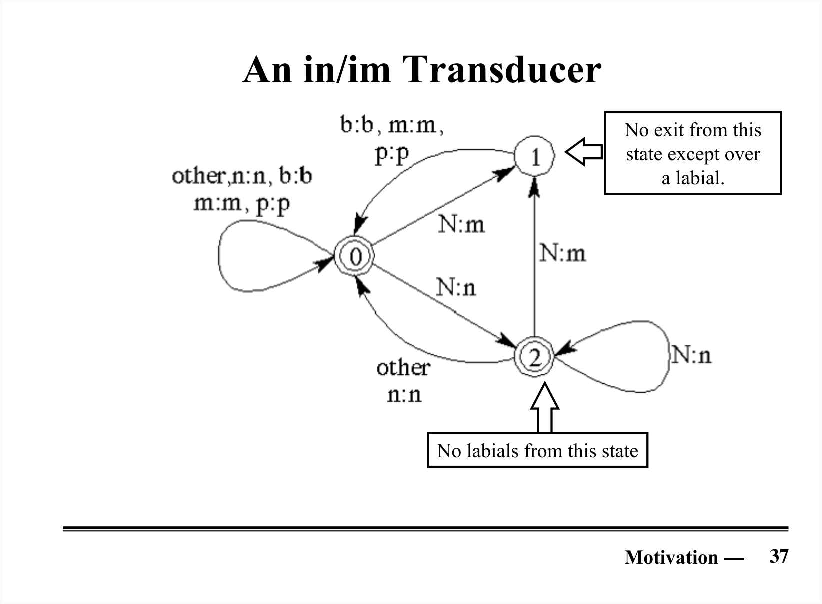
- Finite languages
  - dictionaries
  - compression
- Phenomena involving bounded dependencies
  - morphology
  - spelling
  - hyphenation
  - morphological analysis
  - phonology
- Approximations to phenomena involving mostly bounded dependencies
  - syntax

Basic idea



- At any given moment, an automaton is in one of a finite number of states
- A transition from one state to another is possible when the automaton contains a corresponding transition.
- The process can stop only when the automaton is in one of a subset of the states, called final.
- Transitions are labeled with symbols so that a sequence of transitions corresponds to a sequence of symbols.

Example: spelling of [IN] (intractable, impossible)



An in/im Transducer

$$\begin{tikzcd}
 & & 1 \arrow[dll, bend right, "{b:b,\ m:m,\ p:p}"'] \\
0 \arrow[loop left, "{other, n:n, b:b, m:m, p:p}"] \arrow[urr, "N:m"] \arrow[drr, "N:n"] & & \\
 & & 2 \arrow[ull, bend left, "{other, n:n}"] \arrow[uu, "N:m"'] \arrow[loop right, "N:n"]
\end{tikzcd}$$

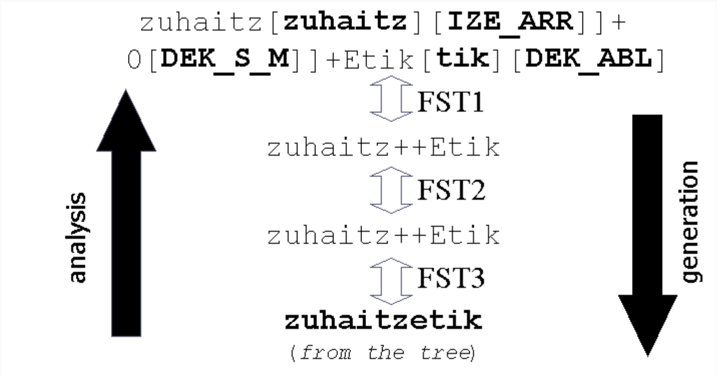
No exit from this state except over a labial.
No labials from this state

Motivation — 37

Example: lemmatization in Basque

- common application of finite state approaches in NLP
- benefit: very fast
- downside: requires time and knowledge to develop well

Alegria et al. (2002) Using Finite State Technology in Natural Language Processing of Basque



Language models

The rest of the class...

- we are going to dedicate to a specific NLP problem for which finite state methods are often used:
- Language modeling.

Language Modeling

What is language modeling?

Which of these two sentences is more likely?

- What do you feel like eating tonight?
- The unlikelihood of such an overestimated timeline is functionally null.

If we want to empirically estimate this, how would we go about it?

Formalization

- V = $\{$a, house, the, boy, ...$\}$
- Usually $|V|$ is very large ($>$20,000)
- We can generate many sentences with V:
    - $*$ The house *STOP*
    - $*$ The girl walks *STOP*
    - $*$ The woman went to sleep *STOP*
    - $*$ *STOP*
    - $*$ The the the to *STOP*
    - ...

Formalization

Assume that we have a corpus of sentences of a language

- Vocabulary: V = {a, house, the, boy, ...}
- $V^\dagger$: Set of strings from V
- We need a function that will give a probability for every sentence w:
  - $p(w) > 0, \forall w \in V^\dagger$
  - $\sum_{w \in V^\dagger} p(w) = 1$

Need for language models

- Often useful for real world problems
- Anytime we need to filter between several generated hypotheses
  - Speech recognition
  - Automatic translation
  - Spelling correction
  - Language identification
  - Language generation

MT example

Given a set of possible translations for “Jeg hadde det hyggelig”

1. “Yo lo tuve agradable”
2. “Tuve un tiempo agradable”
3. “Me lo pasé bien”

Even without knowing Norwegian, which one is the most plausible translation?

MT example

Which one is the most plausible translation?

- p("Yo lo tuve agradable") $= 0.00003$
- p("Tuve un tiempo agradable") $= 0.0001$
- p("Me lo pasé bien") $= 0.001$

...but again, we need a way to reliably estimate these probabilities.

First empirical solution

$$p(\text{“What do you feel like eating tonight?”})$$

$$= \frac{\text{count(“What do you feel like eating tonight?”)}}{\text{count(all sentences ever spoken)}}$$

Any problems here?

First approximation

Given a corpus C of N training sentences

- For every sentence $s = t_1, t_2, \ldots t_n$
- let's take $p(t_1, t_2, \ldots, t_n) = \frac{count(t_1, t_2, \ldots, t_n)}{N}$
- where $count(t_1, t_2, \ldots, t_n)$ is the number of times that $t_1, t_2, \ldots, t_n$ was seen in the corpus.

Any problems here?

First approximation

Problems:

- It's data hungry and suffers from high variance.
- Any grammatical sentence not in the training data will have prob $= 0$
- We need to introduce some useful bias to have a chance of making reliable estimates of probability.

Second approximation

Main idea: we can move another step down

- Instead of trying to estimate the probability of the full sentence as a single element...
- estimate the probability of subswequences of tokens...
- then combine these to estimate the probability of the full sentence.

A refresher on Probability Theory

The chain rule of probability

- $p(A, B) = p(A) \times p(B|A)$
- For a sequence of 4 words...
- $p(t_1, t_2, t_3) = p(t_1) \times p(t_2|t_1) \times p(t_3|t_1, t_2) \times p(t_4|t_1, t_2, t_3)$

Refactoring

- Each $t_i \in V$, assuming $t_0 =$* is a special start symbol
- $p(t_1, t_2, \ldots, t_n) =$
  $p(t_1|t_0) \times p(t_2|t_0, t_1) \times \ldots \times p(x_n|x_0, \ldots, x_{n-1}) =$

$$\prod_{i=1}^{n} p(x_i|x_0, \ldots, x_{i-1})$$

This equation, however, is exact.

Markov assumption

Introduce a first order Markov assumption

- Each word only depends on the previous $N-1$ words
- For a bigram model (N=2),
- $p(\text{feel}|\text{what do you}) \approx p(\text{feel}|\text{you})$
- This equation is not exact, but tractable.

$$\prod_{i=1}^{n} p(x_i|x_{i-1})$$

- This simplifies the number of parameters to $|V|^2$

N-gram models

Crucial simplifying assumption:

- condition on the past $n-1$ words only.
- This is the connection to finite state approaches
  - bounded dependency assumption
- An N-gram over a vocabulary containing V words defines a finite state machine with V^{N-1} states, and V^{N} edges.

Bigram language model example

Consider the sentence I live here STOP

- The probability of this sentence is:
- p(I live here STOP) =

  p(I | *) × p(live | I) × p(here | live) × p(STOP | here)

Can you think of a weakness of this model?

2nd order Markov Model (Trigram)

Each token is conditioned on the two previous tokens

- $p(x_i|x_0, \ldots, x_{i-1}) = p(x_i|x_{i-2}, x_{i-1})$

$$\prod_{i=1}^{n} p(x_i|x_{i-2}, x_{i-1})$$

- Assume there are two special beginning words $x_{-1}, x_0 =$*
- Number of parameters $|V|^3$
- This gives a more powerful model, able to handle longer relationships
- Compare $p(\text{she}|\text{what does})$

  bigram $\approx p(\text{she}|\text{does})$

  trigram $\approx p(\text{she}|\text{what does})$

Trigram language model

Trigram LM parameters

- There is a vocabulary V
- There is a parameter (probability) $q(w|u,v)$ for each possible trigram (u,v,w), where $w \in V \cup \{STOP\}$ and $u,v \in V \cup \{*\}$
- Given a sentence $s = t_1, t_2, \ldots, t_n$ where $t_i \in V$ for $i = 1 \ldots n_{-1}$ and $t_n =$STOP

  The probability of the sentence is

$$\prod_{i=1}^{n} p(t_i|t_{i-2}, t_{i-1})$$

Trigram language model example

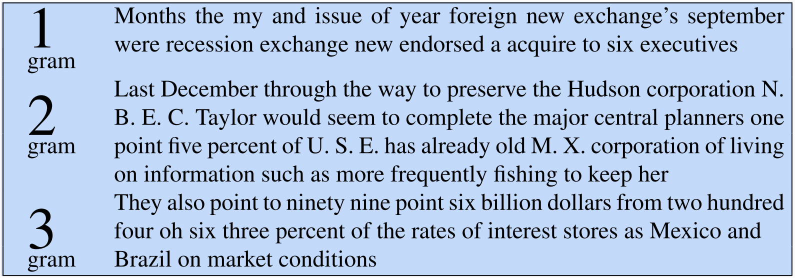
Let's return to the sentence I live here STOP

- The probability of this sentence with a trigram model is:
- p(I live here STOP) =

  p(I | * *) × p(live | * I) × p(here | I live) × p(STOP | live here)

N-gram language models: generation and examples

Text can be generated by sampling from the model until you reach the STOP symbol.



1 gram	Months the my and issue of year foreign new exchange's september were recession exchange new endorsed a acquire to six executives
2 gram	Last December through the way to preserve the Hudson corporation N. B. E. C. Taylor would seem to complete the major central planners one point five percent of U. S. E. has already old M. X. corporation of living on information such as more frequently fishing to keep her
3 gram	They also point to ninety nine point six billion dollars from two hundred four oh six three percent of the rates of interest stores as Mexico and Brazil on market conditions

Figure 3.5 Three sentences randomly generated from three n-gram models computed from 40 million words of the *Wall Street Journal*, lower-casing all characters and treating punctuation as words. Output was then hand-corrected for capitalization to improve readability.

Taken from "Speech and Language Processing (3rd ed. draft)" (Dan Jurafsky and James H. Martin)

`https://web.stanford.edu/~jurafsky/slp3/`

Parameter estimation

Estimation

- So far, we've been assuming that we already have a model with the probability estimates precalculated.
- In that case, all we had to do to calculate the probability was apply the model.
- But in reality, we want to estimate these probabilities ourselves.
- How can we go about this empirically?

Estimating the Trigram parameters

Take a corpus and use Maximum Likelihood Estimation (MLE)

- $q(w|u,v) = \frac{count(u,v,w)}{count(u,v)}$
- The probability of I live here would be:
- $q(here|I, live) = \frac{count(\text{I live here})}{count(\text{I live})}$ in the corpus

This is problematic, though. Why?

Maximum Likelihood Estimation

Problems:

- One of the main traits of human language is language-users can create and understand novel utterances (productivity in Hockett's design features).
- Previously unseen sequences will always appear.
- "Covid-19 cases surge"
- If any count $= 0$, this will give 0 probability for the entire sentence.

10 minute break

Group activity

- Make small groups (3-4)
- You will have 15-20 minutes to try to answer the following 4 questions.

Group activity

1. Can you think of any ways to avoid 0 probabilities for unknown n-grams?
2. Imagine you have bigram and a unigram model trained on a corpus. We know that the bigram model is more informative, but sparse. The unigram model is less informative, but more robust. Can you think of a way to combine the relative strengths of these models?

Group activity

3. Imagine you have a corpus where the bigram "San Francisco" is very common, but where neither "San" nor "Fracisco" occur by themselves. If you have a unigram model trained on this corpus, the probability of "Francisco" will be high, as count("Francisco") is high. But intuitively, this probability should not be high, as "Francisco" is only ever found in a single context, i.e. following "San". Is there any way we can incorporate this intuition when calculating the probabilities?
4. We were able to make our probabilty estimates more robust and tractable by moving from calculating the probability of a full sentence to multiplying the probability of n-grams of words. Can you think of a way to continue this trend and make an even more robust model? What possible problems would it have?

Group activity

What did you talk about in your groups?

Smoothing and interpolation strategies

Smoothing

- Intuition: give unseen events a small amount of probability.
- But to ensure that p(w) = 1, we have to 'steal' some probability from seen events

Smoothing

$P_{smooth}(w|u,v) = \frac{count(u,v,w)+\alpha}{\sum_z count(u,v,z)+|V|\times\alpha}$

Main ideas

- Imagine each word appears α more times
- Add a small amount to the count and to the denominator in order to make it a true probability
- Referred to as Laplace smoothing when $\alpha = 1$
- Jeffrey Perk's Law smoothing when $\alpha = 0.5$

Interpolation

Trade-off between models

- Trigram estimates: $q_{ML}(t_i|t_{i-2}, t_{i-1}) = \frac{count(t_{i-2},t_{i-1},t_i)}{count(t_{i-2},t_{i-1})}$
- Bigram estimates: $q_{ML}(t_i|t_{i-1}) = \frac{count(t_{i-1},t_i)}{count(t_{i-1})}$
- Unigram estimates: $q_{ML}(t_i) = \frac{count(t_i)}{count()}$

- Trigram estimates: better model, but unreliable for rare trigrams and sparse (lots of zeros)
- Unigram estimates: worse model, but more robust

Interpolation

Linear interpolation

- Trigram estimates: $q(t_i|t_{i-2}, t_{i-1}) =$ $\lambda_1 \times q_{LM}(t_i|t_{i-2}, t_{i-1}) + \lambda_2 \times q_{LM}(t_i|t_{i-1}) + \lambda_3 \times q_{LM}(t_i)$
- where $\lambda_1 + \lambda_2 + \lambda_3 = 1$ and $\lambda_i \geq 0$
- Higher λ_3 implies a more robust system, but uses less context
- Higher λ_1 implies more sparse data.

Interpolation

Simple example

- For example: $\lambda_1 = .5$, $\lambda_2 = .3$, $\lambda_3 = .2$
- p(here | I live) =

  $\lambda_1 \times$ p(here | I live) + $\lambda_2 \times$ p(here | live) + $\lambda_3 \times$ p(here)

- In order to estimate λ_i, we need to use a held out development set. See bibliography for further details.

State-of-the-art smoothing

Intuition to Kneser-Ney smoothing

- I can't see without my reading _______________.
- The word 'glasses' seems much more likely to follow here than, say, the word 'Francisco', so we'd like our unigram model to prefer 'glasses'. But in fact it's 'Francisco' that is more common, since San Francisco is a very frequent word. A standard unigram model will assign 'Francisco' a higher probability than 'glasses'.
- We hypothesize that words that have appeared in more contexts in the past are more likely to appear in some new context as well.

State-of-the-art smoothing

Continuation...

- Extending this line of reasoning, perhaps the unigram probability used should not be proportional to the number of occurrences of a word, but instead to the number of different words that it follows.
- To give an intuitive argument, imagine traversing the training data in order and building a bigram model on the preceding data to predict the current word. Then, whenever the current bigram does not occur i nthe preceding data, the unigram probability will be a large factor in the current bigram probability. If we assign a count to the corresponding unigram whenever such an even occurs, then the number of counts assigned to each unigram will simply be the number of different words that it follows.

Kneser-Ney smoothing

The Kneser-Ney intuition is to base our estimate of $P_{CONTINUATION}(w)$ on the number of different contexts word w has appeared in, that is, the number of bigram types it completes.

Formalizing continuation probability

$$P_{CONTINUATION}(w) = \frac{|\{v : C(vw) > 0\}|}{sum(w')|\{v : C(vw') > 0\}}$$

- The first term tells us how many contexts we've seen w in.
- The second term normalizes over the number of contexts.

Kneser-Ney smoothing

Full formula

$$P_{KN}(w_i|w_{i-1}) = \frac{max(C(w_{i-1}w_i)-d,0)}{C(w_{i-1})} + \lambda(w_i - 1) \times P_{CONTINUATION}(w_i)$$

- where d is a discounting factor
- we use max to avoid negative counts

λ

$$\lambda(w_{i-1}) = \frac{d}{sum(v)C(w_{i-1}v)} \; |w : C(w_{i-1}w) > 0|$$

- where the first term is to normalize the discount
- and the second term is the number of times we applied the normalized discount.

Kneser-Ney: main idea review

Count/Continuation_count:

$$C_{KN}(\cdot) = \begin{cases} count(\cdot), & \text{for the highest order} \\ continuation\_count(\cdot), & \text{for lower orders} \end{cases} \quad (1)$$

To review and go into details:

- Jurafsky and Martin. Speech and Language Processing.
- Chen and Goodman (1998) An empirical study of smoothing techniques for language modeling.

Logarithms in N-grams

Logarithms

If we have very long sentences...

- multiplying probabilities will produce very small numbers.
- In many cases, we will end up with 0 probabilities because of underflow.
- p(I live here STOP) $=$ p(I | *) $\times$ p(live | I) $\times$ p(here | live) $\times$ p(STOP | here)
- 0.0017 * 0.00008 * 0.00023 * 0.0001 $=$ 0.000000000000000312800000000000003

Logarithms

Solution: move to log probabilities

- Instead of $p(t_1, t_2, \ldots, t_n) = \prod_{i=1}^{n} p(x_i | x_{i-2}, x_{i-1})$
- we can use log probability
- logarithm of products becomes sum of logarithms
- $log\ p(t_1, t_2, \ldots, t_n) = \sum_{i=1}^{n} log\ p(x_i | x_{i-2}, x_{i-1})$
- summation faster and avoid underflow.

Logarithms

Going back to the earlier example

- log p(I live here STOP) = log p(I | *) + log p(live | I) + log p(here | live) + log p(STOP | here) =
- log(0.0017) + log(0.00008) + log(0.00023) + log(0.0001) =
- -6.377127 + -9.433483 + -8.377431 + -9.210340 = -33.398382

Other approaches to language modeling

Character-level language models

Use characters instead of words

- including whitespaces/punctuation/etc.
- very robust for unseen words (no need for smoothing)
- Often extends to higher order N-grams (5-10)
- Unfortunately, these models do not normally perform better than word-level models

For a nice tutorial on recurrent neural network character language models see: The Unreasonable Effectiveness of Recurrent Neural Networks (`http://karpathy.github.io/2015/05/21/rnn-effectiveness/`)

Neural language models

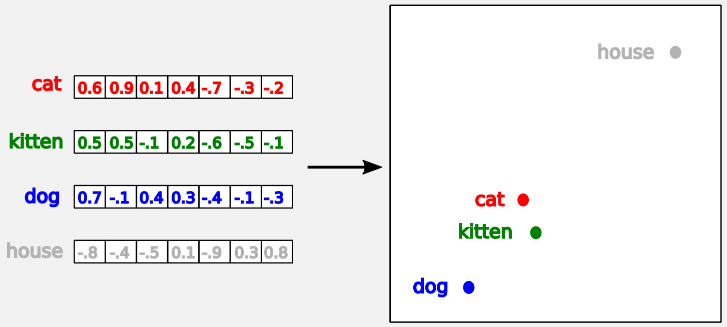
Problem with N-gram models

- In our N-gram models, each word is an atomic unit.
- No similarity between words that intuitively are very similar.
  - We would like a language model to assign similar probability to this sentence:
  - I was on the couch and my (dog|cat) jumped on me.
- But with N-grams, there is no way to capture this intuition.
- With Neural Networks, we can do just that.

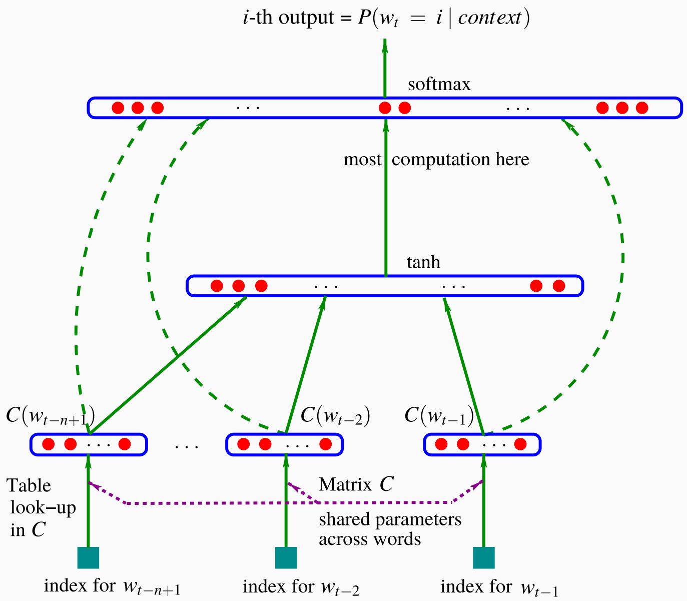
Word embeddings

Each word is represented as a D-dimensional vector

- similar words will have similar vectors
- Used as input to a neural network, similar words should also lead to similar outputs.



Original neural language model



LSTMS/Transformers/etc

Current approaches

- There have been many recent improvements
- Too many to name
- but many are available as online demos

Experimenting with available models

Let's see some of the modern language models in action.

- InferKit: `https://app.inferkit.com/demo`
- GPT 2: `https://huggingface.co/openai-gpt`
- GPT 2: `https://huggingface.co/gpt2`
- XLNet: `https://huggingface.co/xlnet-large-cased`

Choose 5 or six beginnings to have the model complete.

While you look at the models, try to fill in the following table based on your findings.

model	fluent	factual	non-redundant	size	speed
model 1					
model 2					

Evaluation

Evaluation

Ok, so now we have trained a model...

- but, how do we know if it is good?
- What is good enough?

Any suggestions?

Evaluation

Extrinsic

- Ideal setting
- Performance when used on a target task, e.g., machine translation, speech recognition. Uses the metric for the task itself.
- Can be difficult to tell how much better the language model itself is.
- Also costly, as you would need to rerun everything for each language model you train.

Evaluation

Intrinsic

- Faster, easier to iterate while training models.
- Not good unless test data is similar to training data.
- Measure likelihood on held-out (test) data.
- Perplexity $= 2^{-\sum_x p(x) log_2 p(x)}$
- Perplexity measure the surprisal, i.e., how surprising is it to find w_i in the context $w_1, \dots, w_{i-1}$.
- The lower the perplexity, the better (the higher the probability of the test set).

Review

Overview

- Language models are an approximation to finding the probability of a span of text.
- They have traditionally been a major component of NLP systems.
- Ngram models are fast, robust models for language modeling.
- But they require careful smoothing to be usable.

Bibliography

- Natural Language Processing (MIT Press). Jacob Eisenstein. `https://github.com/jacobeisenstein/gt-nlp-class/blob/master/notes/eisenstein-nlp-notes.pdf`
- Speech and Language Processing. Jurafsky and Martin. 3rd Draft. `https://web.stanford.edu/~jurafsky/slp3/`

References