WIKIPEDIA

Language model

A statistical **language model** is a probability distribution over sequences of words. Given such a sequence, say of length m, it assigns a probability $P(w_1, \ldots, w_m)$ to the whole sequence. Having a way to estimate the relative likelihood of different phrases is useful in many natural language processing applications, especially ones that generate text as an output. Language modeling is used in speech recognition, machine translation, part-of-speech tagging, parsing, handwriting recognition, information retrieval and other applications.

In speech recognition, the computer tries to match sounds with word sequences. The language model provides context to distinguish between word and phrases that sound similar. For example, in American English, the phrases "recognize speech" and "wreck a nice beach" are pronounced almost the same but mean very different things. These ambiguities are easier to resolve when evidence from the language model is incorporated with the pronunciation model and the acoustic model.

Language models are used in information retrieval in the query likelihood model. Here a separate language model is associated with each document in a collection. Documents are ranked based on the probability of the query Q in the document's language model $P(Q \mid M_d)$. Commonly, the unigram language model is used for this purpose—otherwise known as the bag of words model.

Data sparsity is a major problem in building language models. Most possible word sequences will not be observed in training. One solution is to make the assumption that the probability of a word only depends on the previous *n* words. This is known as an *n*-gram model or unigram model when *n* = 1.

Contents

Unigram models

A unigram model used in information retrieval can be treated as the combination of several one-state finite automata.[1] It splits the probabilities of different terms in a context, e.g. from $P(t_1t_2t_3) = P(t_1)P(t_2 \mid t_1)P(t_3 \mid t_1t_2)$ to $P_{\text{uni}}(t_1t_2t_3) = P(t_1)P(t_2)P(t_3)$.

In this model, the probability of each word only depends on that word's own probability in the document, so we only have one-state finite automata as units. The automaton itself has a probability distribution over the entire vocabulary of the model, summing to 1. The following is an illustration of a unigram model of a document.

Terms	Probability in doc
a	0.1
world	0.2
likes	0.05
we	0.05
share	0.3
...	...

$$\sum_{\text{term in doc}} P(\text{term}) = 1$$

The probability generated for a specific query is calculated as

$$P(\text{query}) = \prod_{\text{term in query}} P(\text{term})$$

For different documents, we can build their own unigram models, with different hitting probabilities of words in it. And we use probabilities from different documents to generate different hitting probabilities for a query. Then we can rank documents for a query according to the generating probabilities. Next is an example of two unigram models of two documents.

Terms	Probability in Doc1	Probability in Doc2
a	0.1	0.3
world	0.2	0.1
likes	0.05	0.03
we	0.05	0.02
share	0.3	0.2
...	...	...

In information retrieval contexts, unigram language models are often smoothed to avoid instances where P(term) = 0. A common approach is to generate a maximum-likelihood model for the entire collection and linearly interpolate the collection model with a maximum-likelihood model for each document to create a smoothed document model.[2]

n-gram models

In an *n*-gram model, the probability $P(w_1,\ldots,w_m)$ of observing the sentence $w_1,\ldots,w_m$ is approximated as

$$P(w_1,\ldots,w_m) = \prod_{i=1}^{m} P(w_i \mid w_1,\ldots,w_{i-1}) \approx \prod_{i=1}^{m} P(w_i \mid w_{i-(n-1)},\ldots,w_{i-1})$$

Here, it is assumed that the probability of observing the i^{th} word w_i in the context history of the preceding $i-1$ words can be approximated by the probability of observing it in the shortened context history of the preceding $n-1$ words (n^{th} order Markov property).

The conditional probability can be calculated from *n*-gram model frequency counts:

$$P(w_i \mid w_{i-(n-1)},\ldots,w_{i-1}) = \frac{\mathrm{count}(w_{i-(n-1)},\ldots,w_{i-1},w_i)}{\mathrm{count}(w_{i-(n-1)},\ldots,w_{i-1})}$$

The words **bigram** and **trigram** language model denote *n*-gram model language models with $n = 2$ and $n = 3$, respectively.[3]

Typically, however, the *n*-gram model probabilities are not derived directly from the frequency counts, because models derived this way have severe problems when confronted with any *n*-grams that have not explicitly been seen before. Instead, some form of *smoothing* is necessary, assigning some of the total probability mass to unseen words or *n*-grams. Various methods are used, from simple "add-one" smoothing (assign a count of 1 to unseen *n*-grams) to more sophisticated models, such as Good-Turing discounting or back-off models.

Example

In a bigram (n = 2) language model, the probability of the sentence *I saw the red house* is approximated as

$$\begin{aligned}&P(\text{I, saw, the, red, house})\\ \approx {}&P(\text{I}\mid\langle s\rangle)P(\text{saw}\mid\text{I})P(\text{the}\mid\text{saw})P(\text{red}\mid\text{the})P(\text{house}\mid\text{red})P(\langle/s\rangle\mid\text{house})\end{aligned}$$

whereas in a trigram (n = 3) language model, the approximation is

$$\begin{aligned}&P(\text{I, saw, the, red, house})\\ \approx {}&P(\text{I}\mid\langle s\rangle,\langle s\rangle)P(\text{saw}\mid\langle s\rangle,I)P(\text{the}\mid\text{I, saw})P(\text{red}\mid\text{saw, the})P(\text{house}\mid\text{the, red})P(\langle/s\rangle\mid\text{red, house})\end{aligned}$$

Note that the context of the first n – 1 n-grams is filled with start-of-sentence markers, typically denoted <s>.

Additionally, without an end-of-sentence marker, the probability of an ungrammatical sequence **I saw the* would always be higher than that of the longer sentence *I saw the red house.*

Exponential language models

Maximum entropy language models encode the relationship between a word and the n-gram history using feature functions. The equation is $P(w_1,\ldots,w_m)=\frac{1}{Z(w_1,\ldots,w_{m-1})}\exp(a^T f(w_1,\ldots,w_m))$ where $Z(w_1,\ldots,w_{m-1})$ is the partition function, a is the parameter vector, and $f(w_1,\ldots,w_m)$ is the feature function. In the simplest case, the feature function is just an indicator of the presence of a certain n-gram. It is helpful to use a prior on a or some form of regularization.

The log-bilinear model is another example of an exponential language model.

Neural language models

Neural language models (or *Continuous space language models*) use continuous representations or embeddings of words to make their predictions. These models make use of Neural networks.

Continuous space embeddings help to alleviate the curse of dimensionality in language modeling: as language models are trained on larger and larger texts, the number of unique words (the vocabulary) increases[a] and the number of possible sequences of words increases exponentially with the size of the vocabulary, causing a data sparsity problem because for each of the exponentially many sequences. Thus statistics are needed to properly estimate probabilities. Neural networks avoid this problem by representing words in a distributed way, as non-linear combinations of weights in a neural net.[4] An alternate description is that a neural net approximate the language function. The neural net architecture might be feed forward or recurrent, and while the former is simpler the later is more common.

Typically, neural net language models are constructed and trained as probabilistic classifiers that learn to predict a probability distribution

$$P(w_t|\text{context})\,\forall t\in V.$$

I.e., the network is trained to predict a probability distribution over the vocabulary, given some linguistic context. This is done using standard neural net training algorithms such as stochastic gradient descent with backpropagation.[4] The context might be a fixed-size window of previous words, so that the network predicts

$$P(w_t|w_{t-k},\ldots,w_{t-1})$$

from a feature vector representing the previous k words.[4] Another option is to use "future" words as well as "past" words as features, so that the estimated probability is[5]

P(w_{t}|w_{{t-k}},\dots ,w_{{t- .

A third option, that allows faster training, is to invert the previous problem and make a neural network learn the context, given a word. One then maximizes the log-probability[6]

\sum _{{-k\leq j-1,\,j\leq
k}}\log

This is called a skip-gram language model, and is the basis of the popular[7] word2vec program.

Instead of using neural net language models to produce actual probabilities, it is common to instead use the distributed representation encoded in the networks' "hidden" layers as representations of words; each word is then mapped onto an n-dimensional real vector called the word embedding, where n is the size of the layer just before the output layer. The representations in skip-gram models have the distinct characteristic that they model semantic relations between words as linear combinations, capturing a form of compositionality. For example, in some such models, if v is the function that maps a word w to its n-d vector representation, then

v({\mathrm {king}})-v({\mathrm

where ≈ is made precise by stipulating that its right-hand side must be the nearest neighbor of the value of the left-hand side.[5][6]

Other models

A positional language model[8] is one that describes the probability of given words occurring close to one another in a text, not necessarily immediately adjacent. Similarly, bag-of-concepts models[9] leverage on the semantics associated with multi-word expressions such as *buy_christmas_present*, even when they are used in information-rich sentences like "today I bought a lot of very nice Christmas presents".

See also

- Statistical model
- Factored language model
- Cache language model
- Katz's back-off model

Notes

a. See Heaps' law.

References

1. Christopher D. Manning, Prabhakar Raghavan, Hinrich Schütze: An Introduction to Information Retrieval, pages 237–240. Cambridge University Press, 2009
2. Buttcher, Clarke, and Cormack. Information Retrieval: Implementing and Evaluating Search Engines. pg. 289–291. MIT Press.
3. Craig Trim, *What is Language Modeling?* (http://trimc-nlp.blogspot.com/2013/04/language-modeling.html), April 26th, 2013.
4. Bengio, Yoshua (2008). "Neural net language models" (http://www.scholarpedia.org/article/Neural_net_language_models). *Scholarpedia*. **3**. p. 3881.
5. Mikolov, Tomas; Chen, Kai; Corrado, Greg; Dean, Jeffrey (2013). "Efficient estimation of word representations in vector space". arXiv:1301.3781 (https://arxiv.org/abs/1301.3781).
6. Mikolov, Tomas; Sutskever, Ilya; Chen, Kai; Corrado irst4=Greg S.; Dean, Jeff (2013). *Distributed Representations of Words and Phrases and their Compositionality* (http://papers.nips.cc/paper/5021-distributed-representations-of-words-and-phrases-and-their-compositionality.pdf) (PDF). Advances in Neural Information Processing Systems. pp. 3111–3119.
7. Harris, Derrick (16 August 2013). "We're on the cusp of deep learning for the masses. You can thank Google later" (https://gigaom.com/2013/08/16/were-on-the-cusp-of-deep-learning-for-the-masses-you-can-thank-google-later/). *Gigaom*.
8. Yuanhua Lv and ChengXiang Zhai, *Positional Language Models for Information Retrieval* (http://times.cs.uiuc.edu/czhai/pub/sigir09-PLM.pdf), in Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval (SIGIR), 2009.
9. E. Cambria and A. Hussain. Sentic Computing: Techniques, Tools, and Applications. Dordrecht, Netherlands: Springer, ISBN 978-94-007-5069-2 (2012)

Further reading

- J M Ponte and W B Croft (1998). "A Language Modeling Approach to Information Retrieval". *Research and Development in Information Retrieval*. pp. 275–281. CiteSeerX 10.1.1.117.4237.
- F Song and W B Croft (1999). "A General Language Model for Information Retrieval". *Research and Development in Information Retrieval*. pp. 279–280. CiteSeerX 10.1.1.21.6467.

- Chen, Stanley; Joshua Goodman (1998). *An Empirical Study of Smoothing Techniques for Language Modeling* (Technical report). Harvard University. CiteSeerX 10.1.1.131.5458.

External links

- Lecture notes on language models, parsing and machine translation with PCFG, CRF, MaxEnt, MEMM, EM, GLM, HMM by Michael Collins (Columbia University)
- CSLM – Free toolkit for feedforward neural language models
- DALM – Fast, Free software for language model queries
- IRSTLM – Free software for language modeling
- Kylm (Kyoto Language Modeling Toolkit) – Free language modeling toolkit in Java
- KenLM – Fast, Free software for language modeling
- LMSharp – Free language model toolkit for Kneser–Ney-smoothed *n*-gram models and recurrent neural network models
- MITLM – MIT Language Modeling toolkit. Free software
- NPLM – Free toolkit for feedforward neural language models
- OpenGrm NGram library – Free software for language modeling. Built on OpenFst.
- OxLM – Free toolkit for feedforward neural language models
- Positional Language Model
- RandLM – Free software for randomised language modeling
- RNNLM – Free recurrent neural network language model toolkit
- SRILM – Proprietary software for language modeling
- VariKN – Free software for creating, growing and pruning Kneser-Ney smoothed *n*-gram models.
- Language models trained on newswire data
- Web Page NGram Viewer