Probabilistic Language Models

Andrea Esuli

Probabilistic Language Model

A probabilistic/statistical language model is a *probability distribution P* over sequences of terms.

A language model can assign a probability to a document

$$P(d) = P(w_1 w_2 w_3)$$

$$P(\textit{Divina commedia}) = P(\textit{'Nel mezzo del cammin…'})$$

Probabilistic Language Model

Applications:

Machine Translation:

$$P(\text{high winds tonite}) > P(\text{large winds tonite})$$

Spell Correction:

“The office is about fifteen minuets from my house"

$$P(\text{about fifteen minutes from}) > P(\text{about fifteen minuets from})$$

Speech Recognition:

$$P(\text{I saw a van}) >> P(\text{eyes awe of an})$$

Summarization, question-answering, etc.

Probabilistic Language Model

A probabilistic/statistical language model is a *probability distribution* P over sequences of terms.

Given a document d that is composed of a sequence of words $w_1 w_2 w_3$, we can use the chain rule to decompose the probability of the document in terms of the probabilities of its words:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$P(\text{Divina commedia}) = P(\text{'Nel mezzo del cammin...'}) = P(\text{'Nel'})P(\text{'mezzo'} \mid \text{'Nel'})P(\text{'del'} \mid \text{'Nel mezzo'})P(\text{'cammin'} \mid \text{'Nel mezzo del'})...$$

Probabilistic Language Model

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$P(\text{Divina commedia}) = P(\text{'Nel mezzo del cammin...'}) =$$
$$P(\text{'Nel'})P(\text{'mezzo'} \mid \text{'Nel'})P(\text{'del'} \mid \text{'Nel mezzo'})P(\text{'cammin'} \mid \text{'Nel mezzo del'})...$$

The formula above makes **no assumptions** and can **exactly model** any language, yet it is **impractical** because it requires to learn the probability of **any sequence in the language**.

Depending on the **assumptions** we make on the probability distribution, we can create statistical model of different complexity.

Unigrams model

A **unigram** model assumes a *statistical independence* between words, i.e., the probability of d is the product of the probabilities of its words:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$=_{unigram} P(w_1)P(w_2)P(w_3) = \prod_i P(w_i)$$

$$P(\text{Divina commedia}) = P(\text{'Nel mezzo del cammin}\ldots\text{'})$$

$$=_{unigram} P(\text{'Nel'})P(\text{'mezzo'})P(\text{'del'})P(\text{'cammin'})\ldots$$

Unigrams model

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$=_{unigram} P(w_1)P(w_2)P(w_3) = \prod_i P(w_i)$$

$$P(\text{Divina commedia}) = P(\text{'Nel mezzo del cammin}\ldots\text{'})$$

$$=_{unigram} P(\text{'Nel'})P(\text{'mezzo'})P(\text{'del'})P(\text{'cammin'})\ldots$$

Unigram model assumption of statistical independence also loses information about word order.

How do we determine the probabilities?

Counting

How do we determine the probabilities?

Counting from a collection of documents (training set) for which we will derive the parameters of our Language Model.

$$P(w) = \#(occurrences\ of\ w\ in\ D)/\#(occurrences\ in\ D)$$

$$P('che') = \#(occurrences\ of\ 'che'\ in\ 'Divina\ commedia')/\#(occurrences\ in\ 'Divina\ commedia') = 3738/106297 = 0.035$$

The ratio of counts of occurrences of a word by the total number of occurrences of word produces a **maximum likelihood estimation**, i.e., the one that maximises the probability of train data in the model.

Unigrams model

Given a training collection D we can use to fit our model, and then compute probability of any text:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$=_{unigram} P(w_1)P(w_2)P(w_3) = \prod_i P(w_i)$$

$$= \prod_i \#(occurrences\ of\ w_i\ in\ D)/\#(occurrences\ in\ D)$$

What if **a word does not appear in D**? $P(w)=0/\#(occurrences\ in\ D) = 0$

If even a single word is missing from the language model we get $P(d)=0$

Smoothing

What if a word does not appear in D? $P(w)=0$

If even a single word is missing from the language model we get $P(d)=0$

Words that are never observed in the collection used to infer LM statistics can be given a **non-zero probability**,

> so that a single word does not crash $P(d)$ to zero,

yet **very small**,

> so text with unknown words are given a lower probability than those with all-known words.

Smoothing

Smoothing methods add some mass probability to unknown events, taking it from known events.

Add one (Laplace smoothing): adding one occurrence to every event, including the unknown one.

$$P_{add\text{-}1}(w) = \frac{\#(occurrences\ of\ w\ in\ D)+1}{\#(occurrences\ in\ D)+V}$$

where V = number of observed words + 1

Adding 1 may alter too much the probabilities of known events, specially when using small collections.

Smoothing

Adding 1 may be alter too much the probabilities for known events, and can give relatively too much importance to unknown events.

We can choose to add a smaller value k<1, e.g., k=0.01

$$P_{add\text{-}k}(w) = \frac{\#(occurrences\ of\ w\ in\ D)+k}{\#(occurrences\ in\ D)+kV}$$

Smoothing adds robustness with respect to unknown and rare event, but it is always preferable, when possible, to add more text to collection to get stronger statistics.

More info on smoothing.

Log space computation

Instead of multiplying probabilities, we can take logarithms of probabilities and sum them (log-space).

$$log(P_1 P_2 P_3 P_4) = log(P_1) + log(P_2) + log(P_3) + log(P_4)$$

Advantages:

- avoid numbers smaller than machine precision (underflow error)
- it is a linear model
- we can easily go back to probabilities at the end of computation
- marginal efficiency gain: sum is faster than multiplication (IF individual logs are precomputed)

From unigrams to n-grams

With unigram we made an assumption of total independence between words, which is too harsh and leads to poor models.

Yet, we know that considering all the dependencies between words is too complex and statistically fragile.

Can we find a balance between these two extremes?

Markov property

In a stochastic process the **next event may depend on an arbitrarily long sequence of past events**, e.g.: sampling without replacement.

In a stochastic process that satisfies the Markov property **the next event depends only on the present event**, and it this thus **independent for any other past event**.

N-order Markov property states that event at time T depends only from the past N events.

We can define a simple language model, yet making it more powerful than simple unigrams, by assuming that a Markov assumption of some order holds for our language.

Bigrams model

A bigram model assumes a *statistical dependence* of a word from the preceding one:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$= P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2) = \prod_i P(w_i \mid w_{i-1})$$

$$P(\text{Divina commedia}) = P(\text{'Nel mezzo del cammin...'}) =_{bigram}$$
$$P(\text{'Nel'})P(\text{'mezzo'} \mid \text{'Nel'})P(\text{'del'} \mid \text{'mezzo'})P(\text{'cammin'} \mid \text{'del'})...$$

In this way, some information about **word order** is captured.

This simple addition is already able to capture a good amount of **syntactic** regularities.

Smoothing

How to handle bigrams that are never observed in the collection?

- Add-1/Add-k

$$P_{add\text{-}k}(w_i \mid w_{i-1}) = \frac{\#(occurrences\ of\ "w_{i-1}w_i"\ in\ D)+k}{\#(occurrences\ of\ w_{i-1}\ in\ D)+kV}$$

- Bayesian prior smoothing (back-off to unigrams):

$$P_{add\text{-}k}(w_i \mid w_{i-1}) = \frac{\#(occurrences\ of\ "w_{i-1}w_i"\ in\ D)+P(w_i)}{\#(occurrences\ of\ w_{i-1}\ in\ D)+1}$$

N-grams models

A n-gram model assumes a *statistical dependence* of a word from the *n-1* preceding ones:

$$P(d) = P(w_1 w_2 w_3) = P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_1 w_2)$$

$$=_{n\text{-}gram} P(w_1)P(w_2 \mid w_1)P(w_3 \mid w_2) = \Pi_i P(w_i \mid w_{i-1})$$

This simple addition is already able to capture a good amount of language regularities.

In general, the longer the **n-gram** we adopt for the model:

- the more semantic is captured;
- the less statistical significant is the model (memorization/generalization).

Evaluation of language models

Language models usually get evaluated **extrinsically**, i.e., as the benefit they contribute to another problem, e.g., text classification.

Extrinsic evaluation is costly, because it requires running multiple additional experiments.

Extrinsic evaluation may have different outcomes with respect to the specific tested problem, yet this may be desirable because we want to find the best model for a specific task.

Intrinsic evaluation of language models is based on some metric that directly measures how a language model is able to represent effectively the information represented in language.

Evaluation: Perplexity

Once a probabilistic model is fit on training data, we can use test data **from the same source** to evaluate how good it is at predicting probabilities.

Perplexity measures how much *surprised* is a model to observe some text.

Given a test text W its (normalized) perplexity is: $PP(W) = \sqrt[N]{\frac{1}{P(w_1 w_2 ... w_N)}}$

A high perplexity means that the text has got a low probability, a low perplexity means that the text has got a high probability, which is what we aim for, given that test text is from the same source of training data.

Evaluation: Perplexity

The numeric value of the perplexity measure can be interpreted as the **average branching factor** in predicting the next word.

E.g., a perplexity value of 178 means that when the model predicts the next word, it is like choosing between 178 words with uniform probability (note that the actual vocabulary is usually much larger).

N-grams models have lower perplexity than unigrams models, but when the N values gets to high it get worse again due to overfitting on the training data.

Perplexity can be also interpreted in terms of cross-entropy.