Statistical Language Modelling

COM6513 Natural Language Processing

Nikos Aletras

Computer Science Department

Week 4
Spring 2023

The University Of Sheffield.

In the previous lecture...

- Our first NLP problem: **Text classification**

In the previous lecture...

- Our first NLP problem: **Text classification**
- But we ignored **word order** (apart from short sequences, e.g. n-grams)!

In this lecture...

Our second NLP problem: Language Modelling

What is the probability of a given sequence of words in a particular language (e.g. English)?

In this lecture...

Our second NLP problem: Language Modelling

What is the probability of a given sequence of words in a particular language (e.g. English)?

Odd problem. Applications?

Applications of LMs

- Word likelihood for query completion in information retrieval ("Is Sheffield" $\rightarrow$ try it on your search engine)

Applications of LMs

- Word likelihood for query completion in information retrieval ("Is Sheffield" $\rightarrow$ try it on your search engine)
- Language detection ("Ciao Sheffield" is it Italian or English?)

Applications of LMs

- Word likelihood for query completion in information retrieval ("Is Sheffield" $\rightarrow$ try it on your search engine)
- Language detection ("Ciao Sheffield" is it Italian or English?)
- Grammatical error detection ("You're welcome" or "Your welcome"?)

Applications of LMs

- Word likelihood for query completion in information retrieval ("Is Sheffield" $\rightarrow$ try it on your search engine)
- Language detection ("Ciao Sheffield" is it Italian or English?)
- Grammatical error detection ("You're welcome" or "Your welcome"?)
- Speech recognition ("I was tired too." or "I was tired two."?)

Problem setup

Training data is a (often large) set of sentences $\mathbf{x}^m$ with words x_n:

$$D_{train} = \{\mathbf{x}^1, ..., \mathbf{x}^M\}$$
$$\mathbf{x} = [x_1, ...x_N]$$

for example:

$$\mathbf{x} = [\text{<s>}, \text{The}, \text{water}, \text{is}, \text{clear}, ., \text{</s>}]$$

<s>: denotes start of the sentence
</s>: denotes end of the sentence

Calculate sentence probabilities

We want to learn a model that returns the **probability of an unseen sentence x**:

$$P(\mathbf{x}) = P(x_1, ..., x_n), for\ \forall \mathbf{x} \in V^{maxN}$$

V is the vocabulary and V^{maxN} all possible sentences.

Calculate sentence probabilities

We want to learn a model that returns the **probability of an unseen sentence x**:

$$P(\mathbf{x}) = P(x_1, ..., x_n), for\ \forall \mathbf{x} \in V^{maxN}$$

V is the vocabulary and V^{maxN} all possible sentences.

How to compute probability?

Unigram language model

Multiply the probability of each word appearing in the sentence $\mathbf{x}$ computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n) = \prod_{n=1}^{N} \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

Unigram language model

Multiply the probability of each word appearing in the sentence **x** computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n) = \prod_{n=1}^{N} \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>

Unigram language model

Multiply the probability of each word appearing in the sentence **x** computed over the entire corpus:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n) = \prod_{n=1}^{N} \frac{c(x_n)}{\sum_{x \in V} c(x)}$$

<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>

$$P(\text{i love}) = P(\text{i})P(\text{love}) = \frac{2}{20} \cdot \frac{1}{20} = 0.005$$

What could go wrong?

$<$s$>$ i love playing basketball $<$/s$>$
$<$s$>$ arctic monkeys are from sheffield $<$/s$>$
$<$s$>$ i study in sheffield uni $<$/s$>$

- The most probable word is $<$s$>$ ($\frac{3}{20}$)

What could go wrong?

<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>

- The most probable word is <s> ($\frac{3}{20}$)
- The most probable single-word sentence is "<s>"

What could go wrong?

<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>

- The most probable word is <s> ($\frac{3}{20}$)
- The most probable single-word sentence is "<s>"
- The most probable two-word sentence is "<s> <s>"

What could go wrong?

<s> i love playing basketball </s>
<s> arctic monkeys are from sheffield </s>
<s> i study in sheffield uni </s>

- The most probable word is <s> ($\frac{3}{20}$)
- The most probable single-word sentence is "<s>"
- The most probable two-word sentence is "<s> <s>"
- The most probable *N*-word sentence is $N\times$ "<s>"

Maximum Likelihood Estimation

Instead of assuming **independence**:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n)$$

We assume that each word is **dependent** on all previous ones:

$$\begin{aligned} P(\mathbf{x}) &= P(x_1, ..., x_N) \\ &= P(x_1)P(x_2...x_N|x_1) \\ &= P(x_1)P(x_2|x_1)...P(x_N|x_1, ..., x_{N-1}) \\ &= \prod_{n=1}^{N} P(x_n|x_1, ...x_{n-1}) \quad \text{(chain rule)} \end{aligned}$$

Maximum Likelihood Estimation

Instead of assuming **independence**:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n)$$

We assume that each word is **dependent** on all previous ones:

$$\begin{aligned} P(\mathbf{x}) &= P(x_1, ..., x_N) \\ &= P(x_1)P(x_2 ... x_N | x_1) \\ &= P(x_1)P(x_2|x_1) ... P(x_N|x_1, ..., x_{N-1}) \\ &= \prod_{n=1}^{N} P(x_n|x_1, ... x_{n-1}) \quad \text{(chain rule)} \end{aligned}$$

What could go wrong?

Problems with MLE

Let's analyse this:

$$P(\mathbf{x}) = P(x_1)P(x_2|x_1)P(x_3|x_2,x_1)...P(x_N|x_1,...,x_{N-1})$$

$$P(x_n|x_{n-1...x_1}) = \frac{c(x_1...x_{n-1},x_n)}{c(x_1...x_{n-1})}$$

As we condition on more words, the counts become **sparser**

Bigram Language Models

Assume that the choice of a word **depends only on the one before it**:

$$P(\mathbf{x}) = \prod_{n=1}^{N} P(x_n | x_{n-1}) = \prod_{n=1}^{N} \frac{c(x_{n-1}, x_n)}{c(x_{n-1})}$$

k-th order **Markov assumption**:

$$P(x_n | x_{n-1}, ..., x_1) \approx P(x_n | x_{n-1}, ..., x_{n-k})$$

with k=1

Bigram LM: From counts to probabilities

Unigram counts:

arctic	**monkeys**	**are**	**my**	**favourite**	**band**
100	600	4000	3000	500	200

Bigram counts (rows: x_{i-1}, cols: x_i):

	arctic	**monkeys**	**are**	**my**	**favourite**	**band**
arctic	0	10	2	0	0	0
monkeys	0	0	250	1	5	0
are	3	45	0	600	25	1
my	0	2	0	1	300	5
favourite	0	1	0	0	0	50
band	0	0	3	10	0	0

Bigram LM: From counts to probabilities

From the bigram count matrix, compute probabilities by dividing each cell by the appropriate unigram count for its row.

Bigram probabilities (rows: x_{i-1}, cols: x_i):

	arctic	**monkeys**	**are**	**my**	**favourite**	**band**
arctic	0	0.1	0.02	0	0	0
monkeys	0	0	0.417	0.0017	0.008	0
are	0.0008	0.0113	0	0.15	0.0063	0.00003
my	0	0.0007	0	0.0003	0.1	0.0017
favourite	0	0.002	0	0	0	0.1
band	0	0	0.015	0.05	0	0

Example: Bigram language model

$$\mathbf{x} = [\text{arctic}, \text{monkeys}, \text{are}, \text{my}, \text{favourite}, \text{band}]$$

$$\begin{aligned}
P(\mathbf{x}) = {} & P(\text{monkeys}|\text{arctic})P(\text{are}|\text{monkeys})P(\text{my}|\text{are}) \\
& P(\text{favourite}|\text{my})P(\text{band}|\text{favourite}) \\
& = \frac{c(\text{arctic,monkeys})}{c(\text{arctic})} \ldots \frac{c(\text{favourite,band})}{c(\text{favourite})} \\
& = 0.1 \cdot 0.417 \cdot 0.15 \cdot 0.1 \cdot 0.1 \\
& = 0.00006255
\end{aligned}$$

Longer contexts (N-gram LMs)

$$P(x|context) = \frac{P(context, x)}{P(context)} = \frac{c(context, x)}{c(context)}$$

- In bigram LM *context* is x_{n-1}, trigram x_{n-2}, x_{n-1}, etc.
- The longer the context:
  - the more likely to capture long-range dependencies:
    "I saw a tiger that was really very..." fierce or talkative?
  - the sparser the counts (zero probabilities)
- 5-grams and training sets with billions of tokens are common.

Unknown Words

- If a word was never encountered in training, any sentence containing it will have probability 0
- It happens:
  - all corpora are finite
  - new words emerging

Unknown Words

- If a word was never encountered in training, any sentence containing it will have probability 0
- It happens:
  - all corpora are finite
  - new words emerging
- Common solutions:
  - Generate unknown words in the training data by replacing low-frequency words with a special UNKNOWN token
  - Use classes of unknown words, e.g. names and numbers

Implementation details

- Dealing with large datasets requires efficiency:
  - use log probabilities to avoid underflows (small numbers)
  - efficient data structures for sparse counts, e.g. lossy data structures Bloom filters)

Implementation details

- Dealing with large datasets requires efficiency:
  - use log probabilities to avoid underflows (small numbers)
  - efficient data structures for sparse counts, e.g. lossy data structures Bloom filters)

How do we train and evaluate our language models?

- We need train/dev/test data
- Evaluation approaches

Intrinsic Evaluation: Accuracy

- How well does our LM predict the next word?
- **I always order pizza with cheese and...**
  - mushrooms?
  - bread?
  - and?
- Accuracy: how often the LM predicts the correct word
- The higher the better

Intrinsic Evaluation: Perplexity

- **Perplexity**: the inverse probability of the test set $\mathbf{x} = [x_1, ..., x_N]$, normalised by the number of words N:

$$
\begin{aligned}
PP(\mathbf{x}) &= P(x_1, \ldots, x_N)^{1/N} \\
&= \sqrt[N]{\frac{1}{P(x_1, \ldots, x_N)}} \\
&= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(x_i | x_1, ... x_{i-1})}}
\end{aligned}
$$

Intrinsic Evaluation: Perplexity

- **Perplexity**: the inverse probability of the test set $\mathbf{x} = [x_1, ..., x_N]$, normalised by the number of words N:

$$\begin{aligned} PP(\mathbf{x}) &= P(x_1, \ldots, x_N)^{1/N} \\ &= \sqrt[N]{\frac{1}{P(x_1, \ldots, x_N)}} \\ &= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(x_i | x_1, ... x_{i-1})}} \end{aligned}$$

- Measures how well a probability distribution predicts a sample.
- The lower the better.

Why is a bigram language model likely to have lower perplexity than a unigram one?

Intrinsic Evaluation: Perplexity

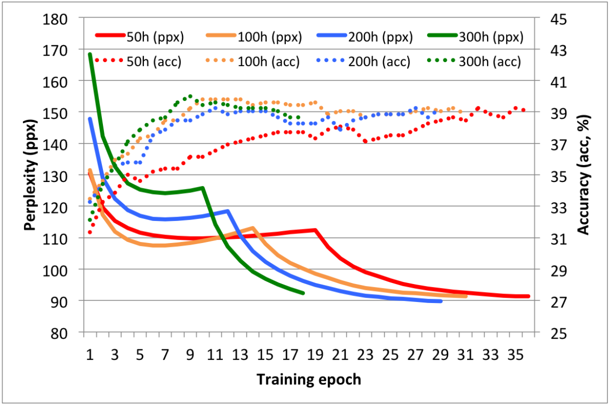
- **Perplexity**: the inverse probability of the test set $\mathbf{x} = [x_1, ..., x_N]$, normalised by the number of words N:

$$
\begin{aligned}
PP(\mathbf{x}) &= P(x_1, \ldots, x_N)^{1/N} \\
&= \sqrt[N]{\frac{1}{P(x_1, \ldots, x_N)}} \\
&= \sqrt[N]{\frac{1}{\prod_{i=1}^{N} P(x_i | x_1, ... x_{i-1})}}
\end{aligned}
$$

- Measures how well a probability distribution predicts a sample.
- The lower the better.

Why is a bigram language model likely to have lower perplexity than a unigram one? There is more context to predict the next word!

The problem with perplexity



- Doesn't always correlate with application performance
- Can't evaluate non probabilistic LMs

Extrinsic Evaluation

- Sentence completion
- Grammatical error correction: detecting “odd” sentences and propose alternatives
- Natural lanuage generation: prefer more “natural” sentences
- Speech recognition
- Machine translation

Smoothing

- What happens when words that are in our vocabulary appear with an unseen context in test data?

Smoothing

- What happens when words that are in our vocabulary appear with an unseen context in test data?
- They will be assigned with zero probability

Smoothing

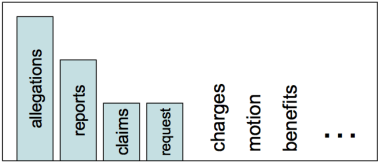
- What happens when words that are in our vocabulary appear with an unseen context in test data?
- They will be assigned with zero probability

Smoothing (or discounting) to the rescue: Steal from the rich and give to the poor!

Smoothing intuition

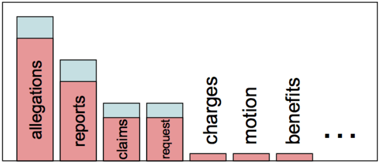
- We often want to make estimates from sparse statistics:

P(w | denied the)
3 allegations
2 reports
1 claims
1 request
7 total



- Smoothing flattens spiky distributions so they generalize better

P(w | denied the)
2.5 allegations
1.5 reports
0.5 claims
0.5 request
2 other
7 total



Taking from the frequent and giving to the rare (discounting)

Add-1 Smoothing

Add-1 (or Laplace) smoothing adds one to all bigram counts:

$$P_{add-1}(x_n|x_{n-1}) = \frac{c(x_{n-1}, x_n) + 1}{c(x_{n-1}) + |V|}$$

Pretend we have seen all bigrams at least once!

Add-k Smoothing

Add-1 puts too much probability mass to unseen bigrams, better to add-k, $k < 1$:

$$P_{add-k}(x_n|x_{n-1}) = \frac{counts(x_{n-1}, x_n) + k}{counts(x_{n-1}) + k|V|}$$

Add-k Smoothing

Add-1 puts too much probability mass to unseen bigrams, better to add-k, $k < 1$:

$$P_{add-k}(x_n|x_{n-1}) = \frac{counts(x_{n-1}, x_n) + k}{counts(x_{n-1}) + k|V|}$$

k is a hyperparameter: choose optimal value on the dev set!

Interpolation

Longer contexts are more informative:

dog bites … better than bites …

but only if they are frequent enough:

canid bites …better than bites …?

Interpolation

Longer contexts are more informative:

dog bites … better than bites …

but only if they are frequent enough:

canid bites …better than bites …?

Can we combine evidence from unigram, bigram and trigram probabilities?

Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n|x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n|x_{n-1}, x_{n-2}) \\ &+ \lambda_2 P(x_n|x_{n-1}) \\ &+ \lambda_1 P(x_n) \qquad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

Simple Linear Interpolation

For a trigram LM:

$$
\begin{aligned}
P_{SLI}(x_n|x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n|x_{n-1}, x_{n-2}) \\
&+ \lambda_2 P(x_n|x_{n-1}) \\
&+ \lambda_1 P(x_n) \qquad \lambda_i > 0, \sum \lambda_i = 1
\end{aligned}
$$

- Weighted average of unigram, bigram and trigram probabilities

Simple Linear Interpolation

For a trigram LM:

$$\begin{aligned} P_{SLI}(x_n|x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n|x_{n-1}, x_{n-2}) \\ &+ \lambda_2 P(x_n|x_{n-1}) \\ &+ \lambda_1 P(x_n) \qquad \lambda_i > 0, \sum \lambda_i = 1 \end{aligned}$$

- Weighted average of unigram, bigram and trigram probabilities
- How we choose the value of λs?

Simple Linear Interpolation

For a trigram LM:

$$
\begin{aligned}
P_{SLI}(x_n|x_{n-1}, x_{n-2}) &= \lambda_3 P(x_n|x_{n-1}, x_{n-2}) \\
&+ \lambda_2 P(x_n|x_{n-1}) \\
&+ \lambda_1 P(x_n) \qquad \lambda_i > 0, \sum \lambda_i = 1
\end{aligned}
$$

- Weighted average of unigram, bigram and trigram probabilities
- How we choose the value of λs? Parameter tuning on the dev set!

Backoff

Start with n-gram order of k but if the counts are 0 use $k-1$:

$$BO(x_n|x_{n-1}\dots x_{n-k}) = \begin{cases} P(x_n|x_{n-1}\dots x_{n-k}), & \text{if } c(x_n\dots x_{n-k}) > 0 \\ BO(x_n|x_{n-1}\dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

Backoff

Start with n-gram order of k but if the counts are 0 use $k-1$:

$$BO(x_n|x_{n-1}\dots x_{n-k}) =$$
$$\begin{cases} P(x_n|x_{n-1}\dots x_{n-k}), & \text{if } c(x_n\dots x_{n-k})>0 \\ BO(x_n|x_{n-1}\dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

Is this a probability distribution?

Backoff

NO! Must discount probabilities for contexts with counts $P^\star$ and distribute the mass to the shorter context ones:

$$P_{BO}(x_n|x_{n-1}\dots x_{n-k}) =$$
$$\begin{cases} P^\star(x_n|x_{n-1}\dots x_{n-k}), & \text{if } c(x_n\dots x_{n-k}) > 0 \\ \alpha^{x_{n-1}\dots x_{n-k}} P_{BO}(x_n|x_{n-1}\dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

$$\alpha^{x_{n-1}\dots x_{n-k}} = \frac{\beta^{x_{n-1}\dots x_{n-k}}}{\sum P_{BO}(x_n|x_{n-1}\dots x_{n-k+1})}$$

β, is the left-over probability mass for the (n-k)-gram

Absolute Discounting

Using 22M words for train and held-out

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Can you predict the heldout (test) set average count given the training?

Absolute Discounting

Using 22M words for train and held-out

Bigram count in training	Bigram count in heldout set
0	.0000270
1	0.448
2	1.25
3	2.24
4	3.23
5	4.21
6	5.23
7	6.21
8	7.21
9	8.26

Can you predict the heldout (test) set average count given the training?
Testing counts = training counts - 0.75 (absolute discount)

Absolute discounting

$$P_{AbsDiscount}(x_n|x_{n-1}) = \frac{c(x_n, x_{n-1}) - d}{c(x_{n-1})} + \lambda_{x_{n-1}} P(x_n)$$

- $d = 0.75$, λs tuned to ensure we have a valid probability distribution.
- Component of the **Kneser-Ney** discounting:
  - Intuition: a word can be very frequent, but if only follows very few contexts,

    e.g. **Francisco** is frequent but almost always follows **San**
  - The unigram probability in the context of the bigram should capture how likely x_n is to be a novel continuation.

Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.

Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.
- If scoring is enough, **stupid backoff** works adequately:

$$SBO(x_n|x_{n-1} \dots x_{n-k}) =$$
$$\begin{cases} P(x_n|x_{n-1} \dots x_{n-k}), & \text{if } c(x_n \dots x_{n-k}) > 0 \\ \lambda SBO(x_n|x_{n-1} \dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

- Empirically found that $\lambda = 0.4$ works well

Stupid Backoff

- Do we really need probabilities? Estimating the additional parameters takes time for large corpora.
- If scoring is enough, **stupid backoff** works adequately:

$$SBO(x_n|x_{n-1}\dots x_{n-k}) = \begin{cases} P(x_n|x_{n-1}\dots x_{n-k}), & \text{if } c(x_n\dots x_{n-k}) > 0 \\ \lambda SBO(x_n|x_{n-1}\dots x_{n-k+1}), & \text{otherwise} \end{cases}$$

- Empirically found that $\lambda = 0.4$ works well
- They called it stupid because they didn't expect it to work well!

Last words: More data defeats smarter models!

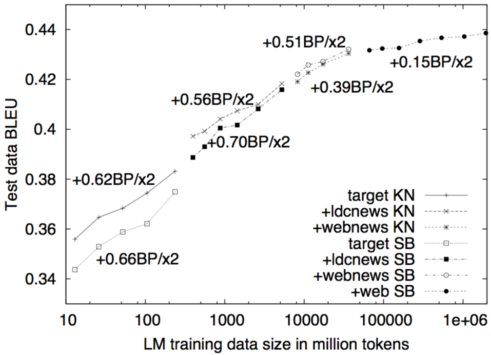


Figure 5: BLEU scores for varying amounts of data using Kneser-Ney (KN) and Stupid Backoff (SB).

From Large Language Models in Machine Translation

Bibliography

- Chapter 3 from Jurafsky & Martin
- Chapter 6 from Eisentein
- Michael Collins' notes on LMs

Coming up next...

- We have learned how to model word sequences using Markov models
- In the following lecture we will look at how to perform part-of-speech tagging using:
  - the **Hidden** Markov Model (HMM)
  - the **Conditional Random Fields** (CRFs), an extension of logistic regression for sequence modelling