ECS763U/P Natural Language Processing

Julian Hough

Week 2: Language Models

OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

Why Probability?

- Building a chatbot- what were the problems with ‘intents’?
- You can define an input->output mapping like a database look-up….
- But what if the user doesn’t say what you manually defined in the input examples?
- It ‘breaks’ with unseen input- we need something more **robust** which accepts things **close** to what is in its database, but not exact string matches.
- The system should be able to make a good guess at which intent the user’s contribution refers to. i.e. ‘A Hawaiwan with extra cheese please’ **probably means** *#UserRequestPizza*.

What is a Probability?

- The measure of the **likelihood** that an **event will occur** or that a given **proposition is the case**. A value between 0 (impossible) and 1 (certain). E.g.:
  - The probability that *the earth is flat:* ***close to 0***
  - The probability that *1 + 1 = 3:* ***0***
  - The probability that *the sun will set today:* ***close to 1***
  - The probability that *1 + 1 = 2:* ***1***
  - The probability that *6 is thrown in a fair die:* **1/6**

What is a Probability?

- Complex probabilities with more than one event/state of affairs in question can be cashed out in terms of *AND*s and *OR*s over single outcomes:
  - probability it will rain *and* be warm?
  - probability it will rain *or* snow?

What is a Probability?

- **AND:** The probability of two 6's being thrown in a fair die one after each other, i.e. the probability of one independent throw being a 6 <u>*and*</u> another independent throw being 6, just **multiply** the probabilities:

  p(6 thrown) x p(6 thrown) = 1/6 x 1/6 = 1/36

  (***conjunctive probability of independent events***)

- **OR:** The probability of either a 6 <u>*or*</u> a 3 being thrown in a fair die for a given throw, just **add** the probabilities :

  p(6 thrown) + p(3 thrown) = 1/6 + 1/6 = 1/3

  (***disjunctive probability of independent events***)

What is a Probability?

- Also what **GIVEN** we know that one event has happened, **THEN** want to know the probability another event has happened, i.e. **conditional probability.**
- Given I know I have thrown an *even numbered* die ({2,4,6}), then what's the probability of me having thrown a *6*?
  - Originally when there were 6 possible outcomes {1,2,3,4,5,6}, the probability was 1/6.
  - Now, given the new condition, this has changed, as there are only 3 possible outcomes {2,4,6} we need to be concerned with- so the likelihood changes to 1/3.
  - We narrow the denominator from all events in the event space to a subset of that space given the information we have.

What is a Probability?

- However, not everything is a die. For potentially **dependent events,** you can't just multiply for conjunction and add for disjunction. We need **general** rules.

- *A and B* can also be calculated in terms of *A given B* and *B given A*, so we have **the product rule**:

$$p(A \wedge B) = p(A \mid B) \times p(B) = p(B \mid A) \times p(A)$$

- For *A or B*, you have to factor out the probability of *A and B*, so in general we have **the sum rule**:

$$p(A \vee B) = p(A) + p(B) - p(A \wedge B)$$

What is a Probability?

- You can formulate conditional probability (i.e. A is the case, given B is the case) in terms of the probability of and A and B being the case over the probability B is the case:

$$p\left(A \mid B\right) = \frac{p(A \wedge B)}{P(B)}$$

- Using the product rule for the numerator, conditional probability can be formulated in terms of **Bayes rule**:

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$

- This is one of the **most important equations in probability theory** (and NLP)
- It allows estimation of *p(A|B)*, using *p(B|A)*, *p(A)* and *p(B)* without necessarily having full access to the full joint distribution *p(A,B)*, which is often very large.

What is a Probability?

- Bayes rule:

$$p(A \mid B) = \frac{p(B \mid A)p(A)}{p(B)}$$

What is a Probability?

- We can think of this in terms of **set theory** where a possible outcome can be seen as a **set**.
- The probability of an outcome, e.g. *throws 6*, is a function of the **cardinality** (size) of the set of all instances with that outcome and the number of *all* events in the **event space** *U*.
- This means any probability is between 0 and 1.

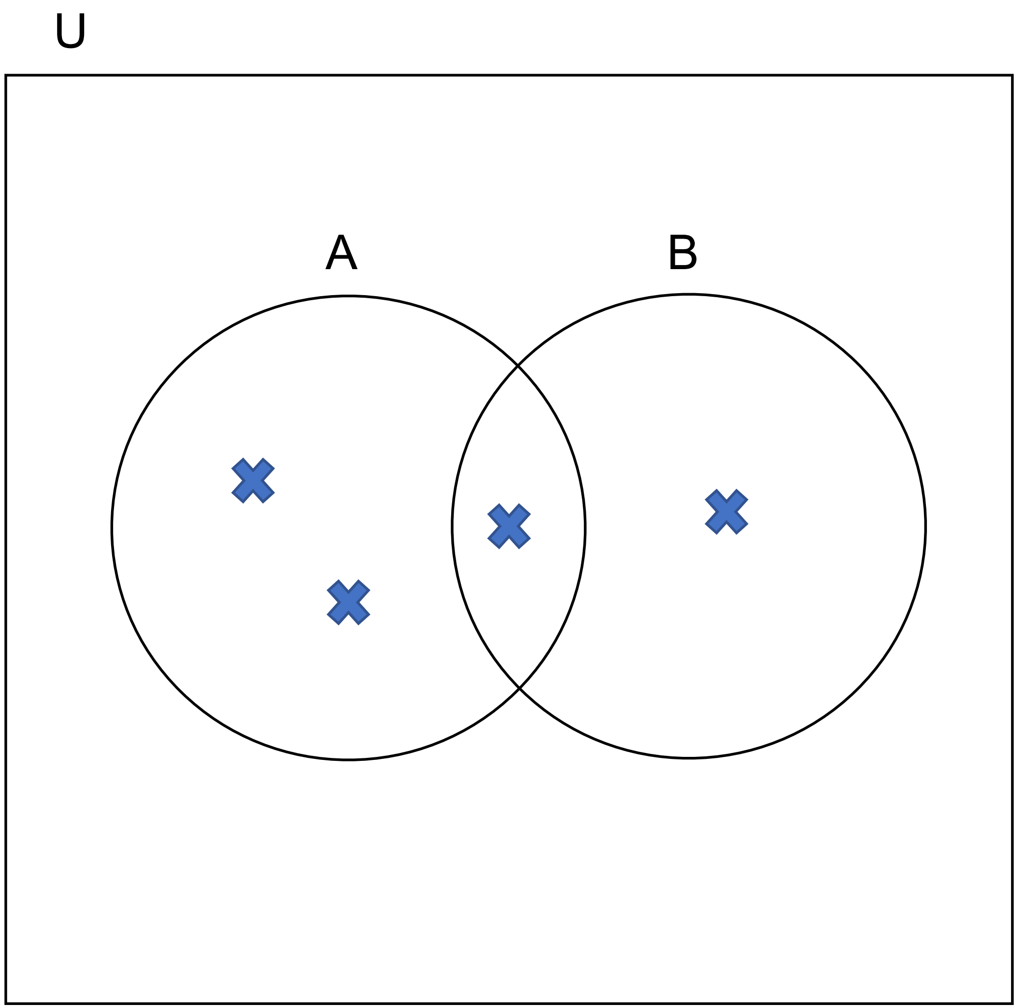
$$p(throws6) = \frac{|throws6|}{|U|} \qquad i.e. \quad for\ any\ event\ A, \quad p(\mathrm{A}) = \frac{|\mathrm{A}|}{|U|}$$

- We can use the analogues of conjunction (and) and disjunction (or) for set **intersection** and set **union**:

$$p(\mathrm{A} \wedge B) = \frac{|\mathrm{A} \cap B|}{|U|} \qquad p(A \vee B) = \frac{|A \cup B|}{|U|}$$

- And Bayes rule can be formulated as: $p(A|B) = \dfrac{|A \cap B|}{|B|}$

What is a Probability?



$$p(A) = \frac{|A|}{|U|} = \frac{3}{4}$$

$$p(B) = \frac{|B|}{|U|} = \frac{2}{4}$$

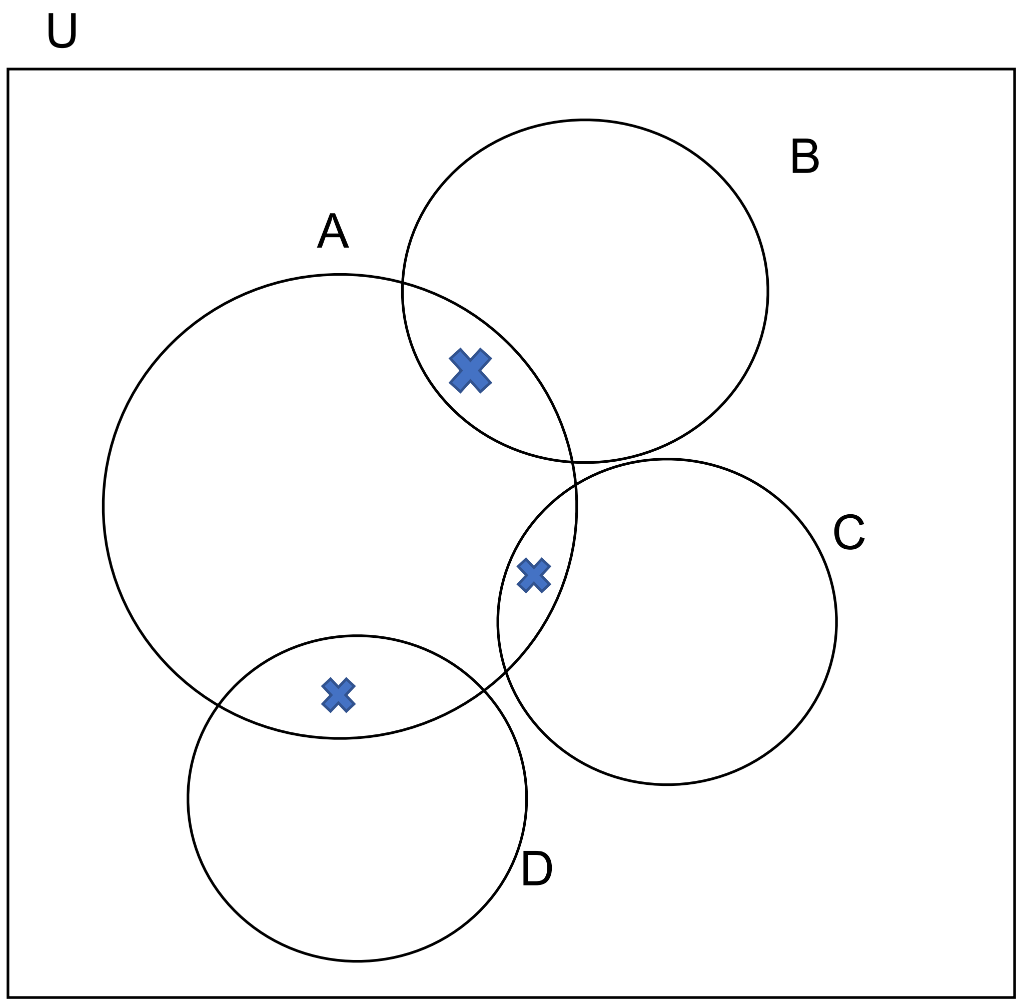
$$p(A \wedge B) = \frac{|A \cap B|}{|U|} = \frac{1}{4}$$

$$p(A \vee B) = \frac{|A \cup B|}{|U|} = \frac{4}{4} = 1$$

$$p\left(A \mid B\right) = \frac{|A \cap B|}{|B|} = \frac{1}{2}$$

$$p\left(B \mid A\right) = \frac{|A \cap B|}{|A|} = \frac{1}{3}$$

What is a Discrete Probability Distribution?

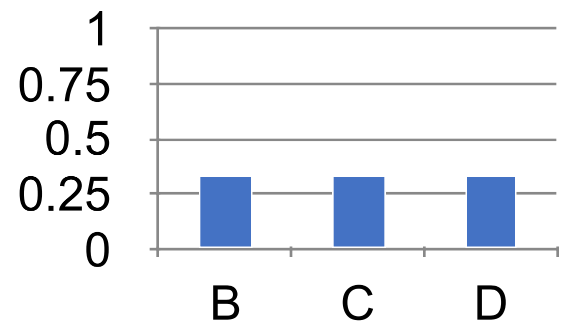


$$\boldsymbol{p}\left(\boldsymbol{X} = ? \mid \boldsymbol{A}\right)$$

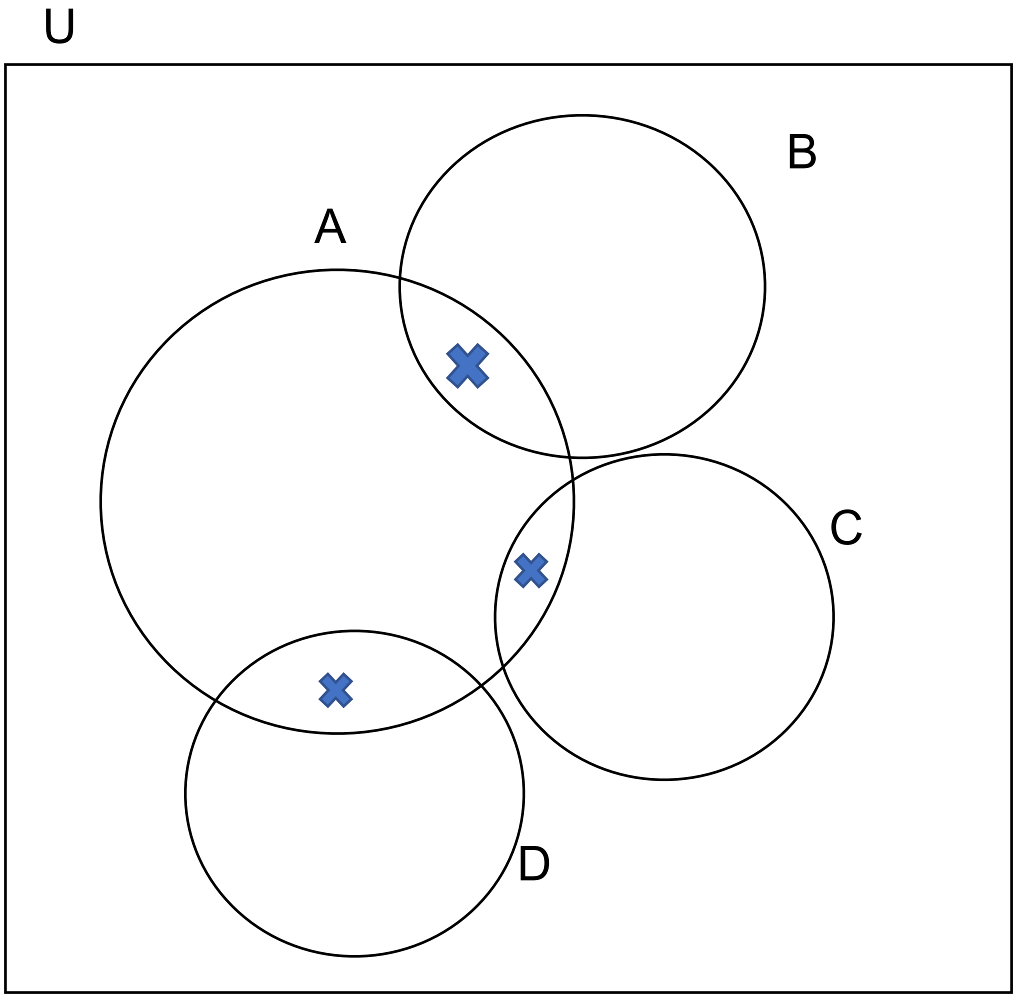
$$p\left(B \mid A\right) = \frac{\left|A \cap B\right|}{\left|A\right|} = \frac{1}{3}$$

$$p\left(C \mid A\right) = \frac{\left|A \cap C\right|}{\left|A\right|} = \frac{1}{3}$$

$$p\left(D \mid A\right) = \frac{\left|A \cap D\right|}{\left|A\right|} = \frac{1}{3}$$



What is a Discrete Probability Distribution?

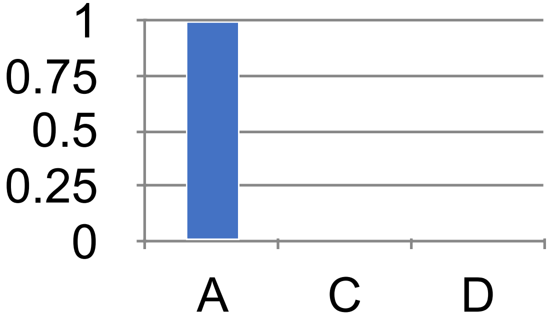


$$\boldsymbol{p}\left(\boldsymbol{X} = ?\mid B\right)$$

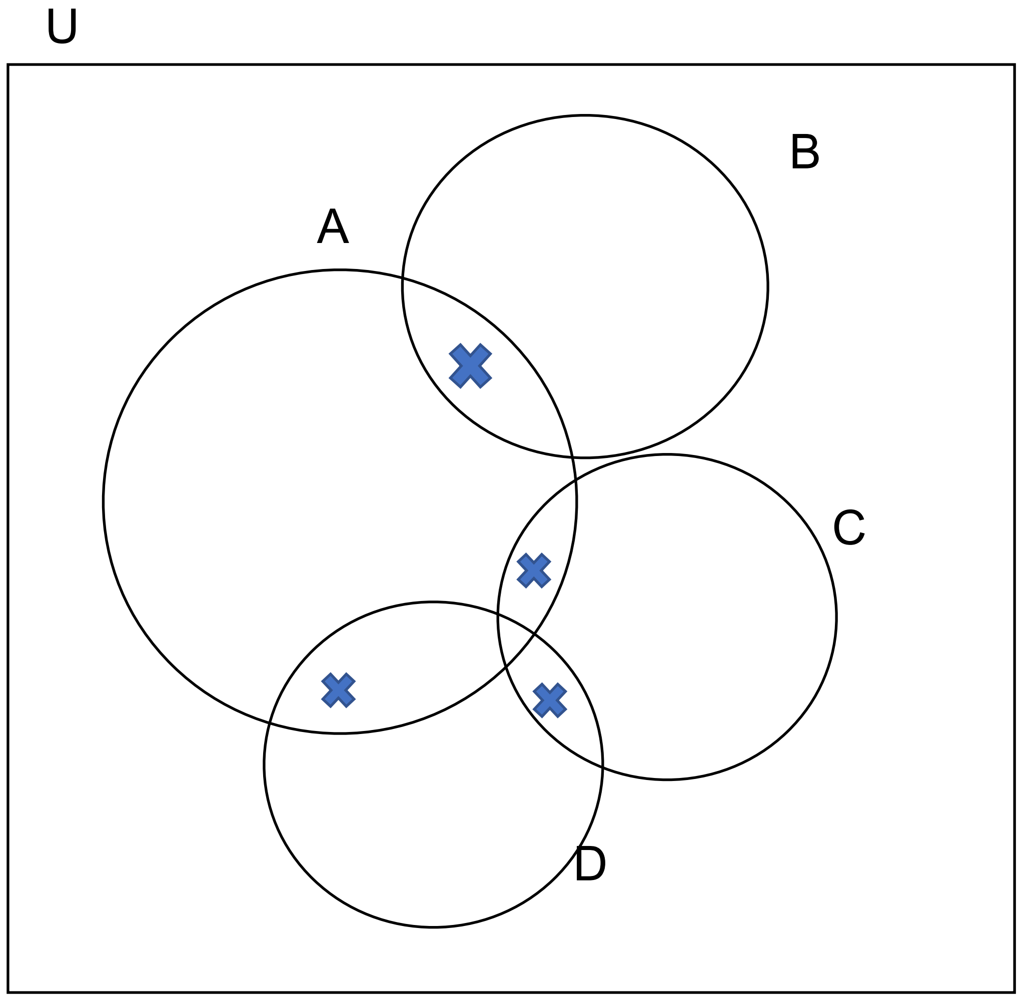
$$p(A \mid B) = \frac{|B \cap A|}{|B|} = \frac{1}{1} = 1$$

$$p(C \mid B) = \frac{|B \cap C|}{|B|} = \frac{0}{1} = 0$$

$$p(D \mid B) = \frac{|B \cap D|}{|B|} = \frac{0}{1} = 0$$



What is a Discrete Probability Distribution?

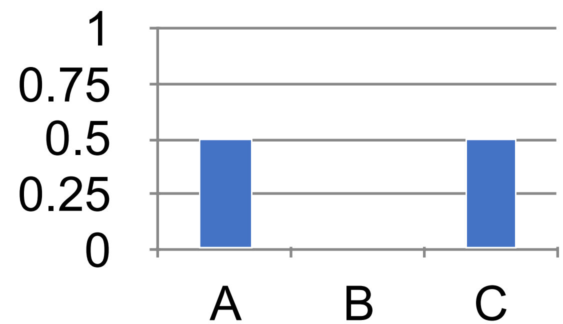


$$p\left(\boldsymbol{X} = ? \mid D\right)$$

$$p(A \mid D) = \frac{|D \cap A|}{|D|} = \frac{1}{2}$$

$$p(B \mid D) = \frac{|D \cap B|}{|D|} = \frac{0}{2} = 0$$

$$p(C \mid D) = \frac{|D \cap C|}{|D|} = \frac{1}{2}$$



OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

Sequence Modelling Tasks

- We considered a classification task last week (sentiment analysis) $d \rightarrow c$
- Many problems are about modelling (labelling, characterising, evaluating) **sequences**:
  - Part-of-speech tagging
  - Dialogue act tagging
  - Named entity recognition
  - Speech recognition
  - Spelling correction
  - Machine translation
  - ...

Sequence Likelihood Tasks

- Speech recognition

  `I saw a van`
  `eyes awe of an`

- Spelling correction

  `It's about fifteen minuets from my house`
  `It's about fifteen minutes from my house`

- Machine translation

  `vjetar će biti noćas jak:`
  `the wind tonight will be strong`
  `the wind tonight will be powerful`
  `the wind tonight will be a yak`

OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

Language Models

- To do effective sequence prediction we want to know the likelihood of different sequences (of words).
- Language models are designed to do this and are machines which play the Shannon Game (1951), reframing the challenge as:
  - How well can we **predict the next word given the history of previous words**?

I always order pizza with cheese and ____

The 33rd President of the US was ____

I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

What is a Language Model?

- Answering the following questions would be useful for assigning probabilities to sequences:
  - **What is the probability of observed sequence O?**
    $p(O) = p(o_1,o_2,o_3,\ldots o_n)$
  - **Given observed sequence O = $o_1\ldots o_{n-1}$, what is the probability of observing symbol o_n next?**
    $p(o_n|o_1,o_2,o_3,\ldots o_{n-1})$

- i.e. What is p("john likes mary") or p("john likes") or p("john likes"|"john")?
- A model which computes these is a **language model.**

What is a Language Model?

- A language model estimates the probability function *p*:

$$p(w_n \mid w_1, w_2, w_3 \dots w_{n-1})$$

Current word

History/context
(previous n-1 words)

- For each context it gives a discrete probability distribution over all words in the vocabulary.

- It assigns a probability value for a given word observed at position w_n given the context observed at $w_1 \dots w_{n-1}$

What is a Language Model?

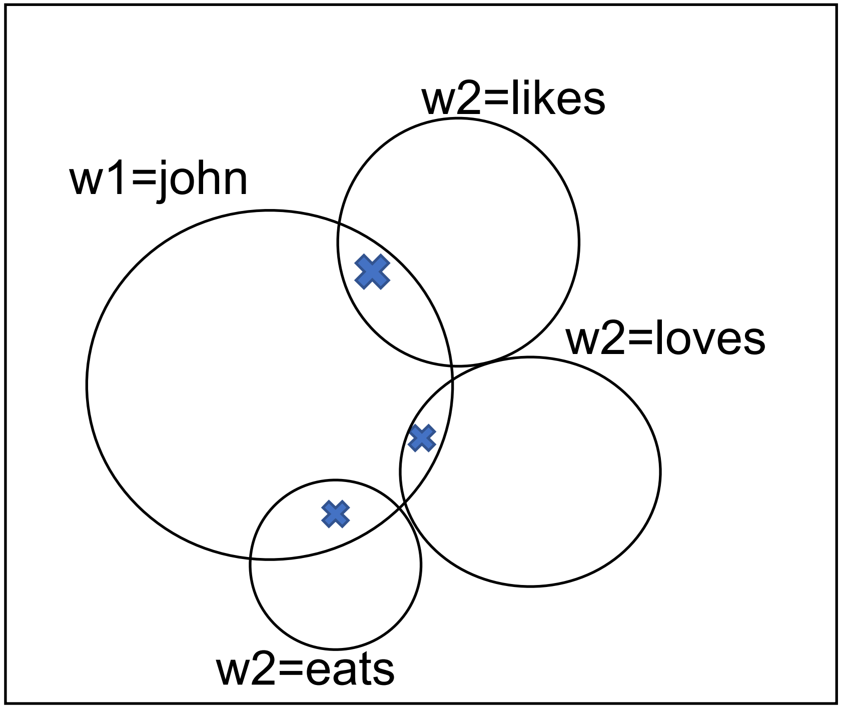
- The probability of the next word being a given value, (e.g. ‘loves’) independent of the previous words is the ***unigram*** probability. In event terms:

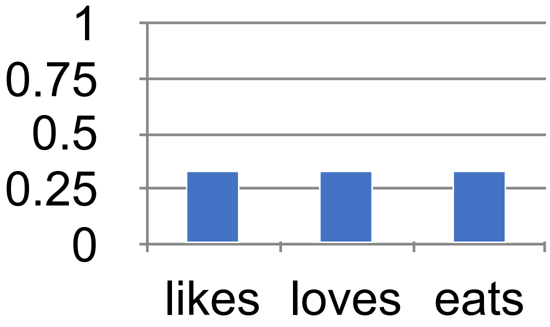
$$p(w_i = loves) = \frac{|w_i = loves|}{\sum_{x \in vocab} |w_i = x|}$$

- Using the probability of the next word given the previous one i.e. the conditional probability $p(w_i|w_{i-1})$ (e.g. for ‘john loves’) is the ***bigram*** probability. In event terms:

$$p(w_i = loves \,|\, w_{i-1} = john) = \frac{|w_{i-1} = john \cap w_i = loves|}{|w_{i-1} = john|}$$

What is a Language Model?





$$p(w2 = loves \mid w1 = john) = \frac{|w1 = john \cap w2 = loves|}{|w1 = john|} = \frac{1}{3}$$

$$p(w2 = likes \mid w1 = john) = \frac{|w1 = john \cap w2 = likes|}{|w1 = john|} = \frac{1}{3}$$

$$p(w2 = eats \mid w1 = john) = \frac{|w1 = john \cap w2 = eats|}{|w1 = john|} = \frac{1}{3}$$

The Chain Rule

- In ngram models, how do we assign probabilities to an entire sequence of words, or the probability of a word given the words so far?
- We can address both via the **chain rule**
- Recall the definition of conditional probabilities (through the **product rule**)

  Rewriting: $\mathbf{P(A,B) = P(A)P(B|A)}$

- More variables:

  $P(A,B,C,D) = P(A)P(B|A)P(C|A,B)P(D|A,B,C)$

- The Chain Rule in General

  $P(x_1,x_2,x_3,\ldots,x_n) = P(x_1)P(x_2|x_1)P(x_3|x_1,x_2)\ldots P(x_n|x_1,\ldots,x_{n-1})$

Using the chain rule

- How do we estimate probabilities? E.g. for the sentence 'Its water is so transparent'
- Count and divide:

$$p(its\ water\ is\ so\ transparent) = p(transparent \mid its\ water\ is\ so) = \frac{C(its\ water\ is\ so\ transparent)}{C(its\ water\ is\ so)}$$

- According to the chain rule:

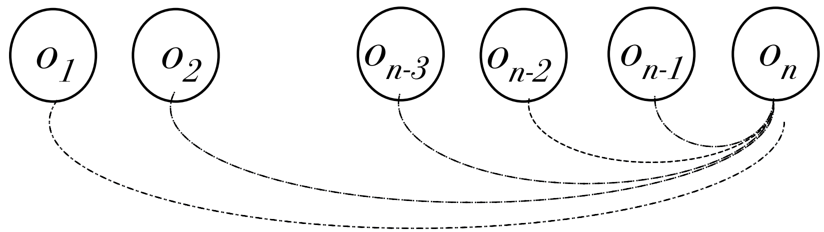
p("its water is so transparent") =

p(its) ×
p(water|its) ×
p(is|its water) ×
p(so|its water is) ×
p(transparent|its water is so)

- We'll never see enough data, so use the **Markov Assumption-** probability of next word only depends on a fixed number of words back, e.g. for a bigram, only depends on previous word:

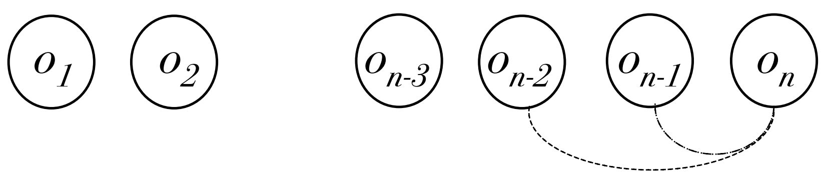
Markov Assumption

- Instead of:



- We approximate by:
  - "n-gram model of length k" (where k = n-1)

trigram model
(k=2):



- In general not sufficient – but often good approximation for high *k*.
  - Ignores long-distance dependencies:
    - "the computer I just put into the machine room on the fifth floor crashed"

Language Models

- This can go up to any arbitrary length (or '**order**'), e.g. unigram, bigram, trigram, 4-gram….7-gram… etc.
- In general **n-gram models** (Shannon ,1948).

- Unigram *its water is so transparent*

- Bigram *its water is so transparent*

- Trigram *its water is so transparent*

- 4-gram etc.

Language Models

- General method when processing sequences is to extract the relevant n-grams (word sequences) according of order *n.*

- In training count the frequency of the ngrams occurring in the training data and store the counts.

- In testing use those counts to get probabilities of sequences of unseen data.

- Deriving the probabilities can be done with a variety of methods!

Language Models

- After training a Maximum Likelihood Estimation (MLE) bigram model from counting function *C* from a corpus:

ngram

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

context
(previous n-1 words)

Language Models

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Example corpus

(note beginning (<s>) and end-of-sentence (</s>) markers):

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

- **<u>Exercise: what is the MLE estimate for:</u>**

p(I|<s>) =   p(Sam|<s>) =   p(am|I) =

p(</s>|Sam) =   p(Sam|am) =   p(do|I) =

Language Models

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Example corpus

(note beginning (<s>) and end-of-sentence (</s>) markers):

<s> I am Sam </s>

<s> Sam I am </s>

<s> I do not like green eggs and ham </s>

- **<u>Exercise: what is the MLE estimate for:</u>**

p(I|<s>) = 2/3 p(Sam|<s>) = 1/3 p(am|I) = 2/3

p(</s>|Sam) = 1/2 p(Sam|am) = 1/2 p(do|I) = 1/3

Language Models

- (Real corpus) Berkeley Restaurant Project sentences:

  - can you tell me about any good cantonese restaurants close by
  - mid priced thai food is what i’m looking for
  - tell me about chez panisse
  - can you give me a listing of the kinds of food that are available
  - i’m looking for a good place to eat breakfast
  - when is caffe venezia open during the day

Language Models

- Bigram counts from 9222 sentences

Word 2 (the bigram is Word 1,Word2)

Word 1
(context)

	i	want	to	eat	chinese	food	lunch	spend
i	5	827	0	9	0	0	0	2
want	2	0	608	1	6	6	5	1
to	2	0	4	686	2	0	6	211
eat	0	0	2	0	16	2	42	0
chinese	1	0	0	0	0	82	1	0
food	15	0	15	0	1	4	0	0
lunch	2	0	0	0	0	1	0	0
spend	1	0	1	0	0	0	0	0

Language Models

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Bigram MLE estimates:

- Normalize by unigram counts:

i	want	to	eat	chinese	food	lunch	spend
2533	927	2417	746	158	1093	341	278

- Result:

	i	want	to	eat	chinese	food	lunch	spend
i	0.002	0.33	0	0.0036	0	0	0	0.00079
want	0.0022	0	0.66	0.0011	0.0065	0.0065	0.0054	0.0011
to	0.00083	0	0.0017	0.28	0.00083	0	0.0025	0.087
eat	0	0	0.0027	0	0.021	0.0027	0.056	0
chinese	0.0063	0	0	0	0	0.52	0.0063	0
food	0.014	0	0.014	0	0.00092	0.0037	0	0
lunch	0.0059	0	0	0	0	0.0029	0	0
spend	0.0036	0	0.0036	0	0	0	0	0

Language Models

- Bigram MLE estimates (example knowledge of the model after counts):

  - p(english|want) = .0011
  - p(chinese|want) = .0065
  - p(to|want) = .66
  - p(eat|to) = .28
  - p(food|to) = 0
  - p(want|spend) = 0
  - p(i|<s>) = .25

Language Models

- Bigram MLE probability estimates of full sentences/ multiple contiguous ngrams- use multiplication of probabilities assuming **independence** of bigrams:

$$
\begin{aligned}
&p(\text{<s> I want english food </s>}) = \\
&\quad p(\text{I}|\text{<s>}) \\
&\quad \times\ p(\text{want}|\text{I}) \\
&\quad \times\ p(\text{english}|\text{want}) \\
&\quad \times\ p(\text{food}|\text{english}) \\
&\quad \times\ p(\text{</s>}|\text{food}) \\
&\qquad = .000031
\end{aligned}
$$

Language Models

- Practical reality: we do everything in **log** space
  - Avoids underflow
  - Adding is faster than multiplying.

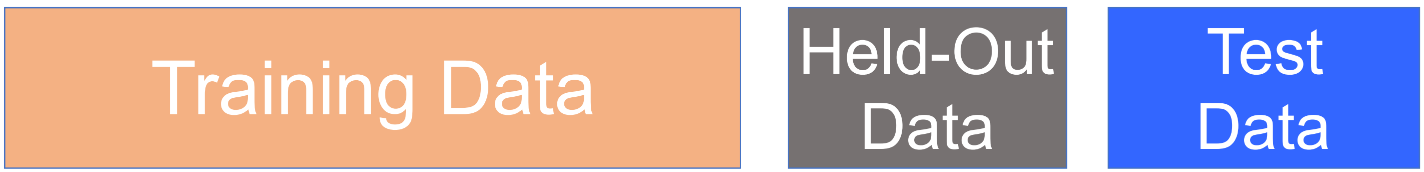
$$log(p(w_1) \times p(w_2) \times p(w_3)) = log(p(w_1)) + log(p(w_2)) + log(p(w_3))$$

OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

How do we evaluate a LM?

- Gather a corpus.
- Divide it into 3 standard sections:



- Gather all the counts/estimations from the training data
- Iteratively develop by assigning probability to the heldout (not the test!) data.
- Experiment with value of *n* and other parameters like *discounts* (more later).
- Get the **Perplexity** score on the Test data (measure of how confused the model is by the unseen corpus).

How do we evaluate a LM?

- Though, don’t forget the preprocessing first!
  - Tokenizing raw text.
  - Spelling normalization (including capitalization).
  - Removal of punctuation (though possibly not all!)
- What about words not in the training data but which appear at testing time (remember language is Zipfian!)
- These could give a zero and mess up the model/ perplexity!
- How do we estimate how many **unknown or ‘out of vocabulary’ (OOV)** words we’re likely to encounter at testing?

How do we evaluate a LM?

- Several approaches to unknown/OOV words:
  - 1. Set a **minimum document frequency** for words across the training data. Any words appearing less than that, replace with an unknown word token <unk/>
  - 2. Set some **heldout training data** aside, and any words appearing in that which are not in the training data, set as <unk/>
- At test time, replace all unknown words to the model with <unk/>.
- **Warning-** for a fair comparison of different models' perplexities, **the same vocab must be used!**

1. OOV words with minimum doc frequency

Training Data- pass 1

John likes Mary
John adores Mary
John adores Bill

Get counts only for the vocab selection.

Vocab counts: John: 3, likes:1, adores: 2, Mary: 2, Bill: 1

Min. Doc. freq = 2,
Vocab = {John, adores, Mary}

Training Data pass 2

John likes Mary
John adores Mary
John adores Bill

→

Replace OOV words with <unk/>, then get counts for the language model.

John **<unk/>** Mary
John adores Mary
John adores **<unk/>**

Counts: John: 3, adores: 2, Mary: 2, <unk/>: 2

Test Data

John despises Mary
Billl adores John

→

John **<unk/>** Mary
<unk/> adores John

2. OOV words from heldout training data

Training Data

John likes Mary
John adores Mary
John adores Bill

Do the counts for all words without replacement and define vocab as all words observed in this data.

Counts: John: 3, likes:1, adores: 2, Mary: 2, Bill: 1

Vocab = {John, likes, Mary, adores, Bill}

Held-Out Training Data

John hates Mary
Billl adores Mary

→

Replace OOV words with <unk/>, then keep adding to the model counts

John **<unk/>** Mary
Bill adores Mary

Counts: John: 4, likes:1, adores: 3, Mary: 4, Bill: 2, <unk/> : 1

Test Data

John despises Mary
Billl adores John

→

John **<unk/>** Mary
Bill adores John

Perplexity

- The Shannon Game:
  - How well can we predict the next word?

    I always order pizza with cheese and ____

    The 33rd President of the US was ____

    I saw a ____

mushrooms 0.1

pepperoni 0.1

anchovies 0.01

….

fried rice 0.0001

….

and 1e-100

  - Unigrams are terrible at this game. (Why?)

- A better model of a text
  - is one which assigns a **higher probability** to the word that actually occurs!

Perplexity

- The best language model is one that best predicts an unseen test data *W*, i.e. the one that gives the highest probability for those sentences.
- Perplexity is the inverse probability of the test set, normalised by the number of words:

$$
\begin{aligned}
\mathrm{PP}(W) &= P(w_1 w_2 \dots w_N)^{-\frac{1}{N}} \\
&= \sqrt[N]{\frac{1}{P(w_1 w_2 \dots w_N)}}
\end{aligned}
$$

Chain rule:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_1 \dots w_{i-1})}}$$

For bigrams:

$$\mathrm{PP}(W) = \sqrt[N]{\prod_{i=1}^{N} \frac{1}{P(w_i \mid w_{i-1})}}$$

Minimising perplexity is the same as maximising probability

Perplexity

- **Cross-entropy** is another metric used for evaluating the confusion of the language model on a test corpus.
- Practically, it is easy to calculate as just the negative sum of the log probabilities divided by the length of the corpus:

$$H(W) = -\frac{1}{N} \log P(w_1 w_2 \ldots w_N)$$

- Perplexity can be calculated from cross-entropy as it's 2 (or whatever log base you're using) to the power of the cross-entropy:

$$\text{Perplexity}(W) \;=\; 2^{H(W)}$$

So, minimising cross-entropy is also the same as maximising probability

Lower perplexity, better model

- Training 38 million words, test 1.5 million words, Wall St. Journal

N-gram Order	Unigram	Bigram	Trigram
Perplexity	962	170	109

Slide from Dan Jurafsky

OUTLINE

1) The need for a probabilistic approach to NLP
2) Probability introduction
3) Language Models: motivation
4) Language Models: ngram models
5) Language Models: evaluation
6) Smoothing

The danger of overfitting!

- N-grams only work well for word prediction if the test corpus looks like the training corpus
  - In real life, it often doesn’t
  - We need to train robust models that generalize!

- One kind of generalization: ngrams with 0 counts!
  - Things that don’t ever occur in the training set
  - But occur in the test set

Zeros!

- Training set:
  … denied the allegations
  … denied the reports
  … denied the claims
  … denied the request

- Test set
  - … denied the offer
  - … denied the loan

$$p(\text{offer} \mid \text{denied the}) = 0$$

- ngrams with zero probability
  - mean that we will assign 0 probability to the test set!
- And hence we cannot compute perplexity (can't divide by 0)!

What can we do about this?

- Three main approaches:
  - **Smoothing**
    - Hold back some probability mass for unseen events
  - **Backoff & Interpolation**
    - Estimate n-gram probability from (n-1)-gram probability
  - **Class-based models**
    - Group words together, estimate class n-gram probability

Smoothing

- When we have sparse statistics from the counts:

C(denied the, w)
3 allegations
2 reports
1 claims
1 request
7 total

- 'Steal'/spread around probability mass to generalize better. I.e. **Discount** some of the seen counts and add that discount to unseen counts:

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

Add-one smoothing

- Also called **Laplace smoothing**
- Pretend we saw each word one more time than we did
- Just add one to all the counts!

- MLE estimate: $$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Add-1 estimate:

$$p^{add-one}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + 1}{C(w_{i-1}) + V}$$

Vocab size

Add-one smoothing

- Add one to all counts (can be done during testing too). New counts will look like this:

	i	want	to	eat	chinese	food	lunch	spend
i	6	828	1	10	1	1	1	3
want	3	1	609	2	7	7	6	2
to	3	1	5	687	3	1	7	212
eat	1	1	3	1	17	3	43	1
chinese	2	1	1	1	1	83	2	1
food	16	1	16	1	2	5	1	1
lunch	3	1	1	1	1	2	1	1
spend	2	1	2	1	1	1	1	1

Add-one smoothing

- Results in a **discount** (reduction) of the seen counts, but adding to the unseen ones to give the smoothed probabilities.

	i	**want**	**to**	**eat**	**chinese**	**food**	**lunch**	**spend**
i	0.0015	0.21	0.00025	0.0025	0.00025	0.00025	0.00025	0.00075
want	0.0013	0.00042	0.26	0.00084	0.0029	0.0029	0.0025	0.00084
to	0.00078	0.00026	0.0013	0.18	0.00078	0.00026	0.0018	0.055
eat	0.00046	0.00046	0.0014	0.00046	0.0078	0.0014	0.02	0.00046
chinese	0.0012	0.00062	0.00062	0.00062	0.00062	0.052	0.0012	0.00062
food	0.0063	0.00039	0.0063	0.00039	0.00079	0.002	0.00039	0.00039
lunch	0.0017	0.00056	0.00056	0.00056	0.00056	0.0011	0.00056	0.00056
spend	0.0012	0.00058	0.0012	0.00058	0.00058	0.00058	0.00058	0.00058

Add-one smoothing

- Add-1 estimation is a blunt instrument
- So add-1 isn't used very much for language modelling:
  - We'll have a look at a couple of better methods!
- But add-1 is used to smooth other NLP models
  - For text classification.
  - In domains where the number of zeros isn't so huge.

Add-k smoothing (generalized additive smoothing)

- Also additive Laplace smoothing, though sometimes 'Lidstone' smoothing
- Pretend we saw each word a value *k* more than we did.
- MLE estimate:

$$p(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

- Add-k estimate:

$$p^{add-k}(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i) + k}{C(w_{i-1}) + kV}$$

- Add-1 smoothing a special case where *k*=1.

Backoff and Interpolation

- Sometimes it helps to use **less** context
  - Condition on less context for contexts you haven't learned much about
- **Backoff:**
  - use trigram if you have good evidence,
  - otherwise bigram, otherwise unigram
- **Interpolation (with lower orders):**
  - mix unigram, bigram, trigram
- Interpolation tends to work better in general.

Backoff and Interpolation

- Simple interpolation

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1 P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2 P(w_n|w_{n-1}) \\ &+\lambda_3 P(w_n)\end{aligned}$$

where:

$$\sum_i \lambda_i = 1$$

- Lambdas conditional on context:

$$\begin{aligned}\hat{P}(w_n|w_{n-2}w_{n-1}) &= \lambda_1(w_{n-2}^{n-1}) P(w_n|w_{n-2}w_{n-1}) \\ &+\lambda_2(w_{n-2}^{n-1}) P(w_n|w_{n-1}) \\ &+\lambda_3(w_{n-2}^{n-1}) P(w_n)\end{aligned}$$

Backoff and Interpolation

- Use a **held-out** corpus to get the right λs

Training Data | Held-Out Data | Test Data

- Choose λs to maximize the probability of held-out data:
  - Fix the N-gram probabilities (on the training data)
  - Then search for λs that give largest probability to held-out set

- Advanced interpolation + backoff technique- **Kneser-Ney smoothing.** Uses **absolute discounting** and the lower-order models. See Goodman (2001).

Summary

- Language models offer a way to assign a **probability** to a sentence or other sequence of words, and to **predict a word from preceding words.**
- n-gram models are **Markov models** that estimate words from a fixed window of previous words. n-gram probabilities can be estimated by counting in a corpus and normalizing (the maximum likelihood estimate).
- n-gram language models are evaluated extrinsically in some task, or intrinsically using **perplexity**.
- The perplexity of a **test set** according to a language model is the geometric mean of the inverse test set probability computed by the model.

Summary

- **Smoothing** algorithms provide a more sophisticated way to estimate the probability of n-grams. Commonly used smoothing algorithms for n-grams rely on lower-order n-gram counts through backoff or interpolation.
- Both **backoff** and **interpolation** require discounting to create a probability distribution.

- **Lab: Implement Add-one smoothing, generalised additive smoothing and Kneser-Ney smoothing.** The interpolated Kneser-Ney smoothing algorithm mixes a discounted probability with a lower-order continuation probability.

Reading

- Manning and Schuetze (1999) Chapters 2 and 6
- Jurafsky and Martin (3rd Ed) Chapter 3
- Goodman (2001)- “A bit of Progress in Language Modeling”