FNLP - Week 2: Annotation, Evaluation, N-Gram Models & Smoothing

Antonio León Villares

Feburary 2022

Contents

1 Annotations

1.1 Considerations When Annotating

- **What factors should be considered when annotating a document?**
  - what is the **source**?
    - genre, size, licensing
  - how **complex** and **fine-grained** do we want to annotate?
  - what **guidelines** to follows when annotating?
  - what is the **expertise** of annotators? is **training** required?
  - which annotation **software** to use?
    - GUI vs CLI
- **What are gray areas in annotation?**
  - the existence of **ambiguities** in text, meaning that a word can be annotated with **different categories**
  - these are necessary to create a **flexible** language
  - for example, "walking" can act as a:
    - *noun*: "Walking was the remedy."
    - *adjective*: "My house is within walking distance"
    - *verb*: "They went walking together."

1.2 Ensuring Annotation Quality

- **What are annotation guidelines?**
  - set of rules which define how to act in these "gray areas"
  - ensure tagging is **consistent** and **reliable** (both for annotators and end-users)
  - for example:

> *"The temporal expressions yesterday, today and tomorrow should be tagged as nouns (NN) rather than as adverbs (RB). Note that you can (marginally) pluralize them and that they allow a possessive form, both of which true adverbs do not."*

- **Can gold labels still be tarnished despite annotation guidelines?**
  - yes:
    - context not considered
    - machine can make erronous pre-annotation, not noticed by human
    - forget guideline detail
    - guidelines don't cover all cases
- **What is inter-annotator agreement?**
  - a **measure** of the **reliability** of an annotation
  - **independent** human annotators work on the same sample, and **compare** annotations
  - typically use **raw agreement rate**
- **Why is IAA useful for guidelines?**
  - IAA $\implies$ upper bound for annotating system

  - if high disagreement, can be used to re-evaluate the **guidelines**

- **Why use Cohen's Kappa instead of Raw Agreement Rate?**
  - not all mistakes are the same
  - mistaking a word for a **noun** (more likely to be accidental) or an **interjection** (less likely to be accidental)
  - **Cohen's Kappa** checks that agreement is made beyond what is expected in a random situation (i.e 2 systems assigning nouns 90% of the time, CK expects random agreement of 81%)

- **What is crowdsourcing?**
  - let users make annotations for small sums of money
  - ensure users are **qualified** and **serious**
  - for example: each point annotated 5+ times; reject annotators with many wrong answers

2 Evaluation

2.1 The Need for Evaluation

- **Why is evaluation necessary?**
  - used to **test hypotheses**
  - used to **improve systems** (empirically)

- **Which hypothese are relevant in NLP?**
  - **Linguistic Objects**: "Is this written by Shakespeare or Marlowe?"
  - **System Output**: "How well is data predicted by the model? How accurate is this parser? How does this POS tagger compare to model X?"
  - **Humans**: "How reliable is annotator Y?"

- **How are models developed?**
  - train a model using a **training set**
  - fine tune a model using a **development set** (i.e model parameters, text features)
  - finally, evaluate a model using a **testing set**
  - **never should the model see data in the testing set**

- **What is k-fold cross validation?**
  - used to validate a model where little training/testing data is available
  - split training data into k folds (especially if we want to change model parameters; need to keep test data secret)
  - train k different models on $k-1$ of the folds, test on the remaining fold
  - accuracy: average accuracy over k folds

2.2 Measuring Model Performance

- **What is a confusion matrix?**
  - table used to gauge model's classification performance (with respect to **gold labels**)

		gold standard labels	
		gold positive	gold negative
system output labels	system positive	**true positive**	**false positive**
	system negative	**false negative**	**true negative**

- **What are TP, FP, TN, FN?**
  - **True Positives**: positive instances correctly classified
  - **False Positives**: negative instances incorrectly classified
  - **True Negatives**: negative instances correctly classified
  - **False Negatives**: positive instances incorrectly classified
- **What is accuracy?**
  - proportion of **correctly classified** instances, over all instances:

$$accuracy = \frac{TP + TN}{TP + FP + TN + FN}$$

- **Why is accuracy not the best measure?**
  - sensitive to **unbalanced classes**
  - for example, if 1% of population gets cancer, a classifier which **always** predicts "not cancer" would have 99% accuracy
- **What is precision?**
  - proportion of truly positive labels predicted, given all labels predicted as positive:

$$precision = \frac{TP}{TP + FP}$$

- **What is recall?**
  - proportion of truly positive labels predicted, given all truly positive labels:

$$recall = \frac{TP}{TP + FN}$$

- **What is the F-measure?**
  - a way of combining **precision** and **recall** into a **single** metric
  - the **F-Measure** is:

$$F_\beta = \frac{(\beta^2 + 1)precision \times recall}{\beta^2 precision + recall}$$

- **How does β affect the F-Measure?**
  - as $\beta \to 0$, $F_\beta = precision$, so precision becomes more important
  - as $\beta \to \infty$, $F_\beta = recall$, so recall becomes more important
- **What is the F_1-measure?**
  - in practice, use $\beta = 1$, to produce the F_1-measure:

$$F_1 = 2 \times \frac{precision \times recall}{precision + recall} = \frac{2}{\frac{1}{precision} + \frac{1}{recall}}$$

- this is the **harmonic mean** of **precision** and **recall**

- **Why is the harmonic mean good as a metric in F_1-measure?**
  - it ensures that **precision** and **recall** are not only **high**, but **similar**
  - it emphasises the lower of the 2 values
  - if they differ greatly, F_1 will be low (for example, if $precision = 1$ and $recall = 0$, $F_1 = 0$)

2.3 Evaluation With More Than 2 Labels

- **Can we evaluate classification for more than 2 labels?**
  - in practice, it is the most likely situation
  - confusion matrices can be extended

		gold labels urgent	normal	spam	
system output	urgent	8	10	1	**precision$_u$** $= \frac{8}{8+10+1}$
	normal	5	60	50	**precision$_n$** $= \frac{60}{5+60+50}$
	spam	3	30	200	**precision$_s$** $= \frac{200}{3+30+200}$
		recall$_u$ $= \frac{8}{8+5+3}$	**recall$_n$** $= \frac{60}{10+60+30}$	**recall$_s$** $= \frac{200}{1+50+200}$	

  - however, precision and recall need to be **modified** via **macroaveraging** or **microaveraging**

- **What is macroaveraging?**
  - compute performance for **each class**, and then compute the **average** over all classes

- **What is microaveraging?**
  - **combine** all decision into a single confusion matrix (whether a decision was correct or not, independent of the label), and compute the statistics from the matrix

Class 1: Urgent

	true urgent	true not
system urgent	8	11
system not	8	340

$$\text{precision} = \frac{8}{8+11} = .42$$

Class 2: Normal

	true normal	true not
system normal	60	55
system not	40	212

$$\text{precision} = \frac{60}{60+55} = .52$$

Class 3: Spam

	true spam	true not
system spam	200	33
system not	51	83

$$\text{precision} = \frac{200}{200+33} = .86$$

$$\text{macroaverage precision} = \frac{.42+.52+.86}{3} = \mathbf{.60}$$

Pooled

	true yes	true no
system yes	268	99
system no	99	635

$$\text{microaverage precision} = \frac{268}{268+99} = \mathbf{.73}$$

2.4 Significance Tests

- **How can we understand if a statistic is good?**
  - having 95% accuracy on its own is not **significant**
  - need to be able to **compare** or **bound** this performance
  - if we set **bounds**, these should depend on the task
- **How can we define an upper bound for performance?**
  - use a **Turing Test**: how much does a **human** agree with a **gold label**?
- **How can we define a lower bound for performance?**
  - use a **baseline** model, which is **simpler**
  - for example, a **majority baseline**: model picks **most frequent class** (i.e using a probability distribution matching what is observed)
- **How can we determine if the difference between models is significant?**
  - if 2 models differ in their accuracy by 10%, how can we know that this is not due to randomness in the testing data
  - we need to understand when **differences** are **significant**
  - this allows us to decided if a model is **superior** to the **baseline**
- **What is a parametric test?**
  - makes assumption about the **underlying distribution** of the data
  - typically assume **normality** (i.e t-test, ANOVE, z-test)
  - not typically applicable to NLP
- **What is non-parametric testing?**
  - testing based on **sampling**, making little assumptions about data
  - McNemar's Test, stochastic/permutation tests (i.e bootstrapping)
- **What is the effect size?**
  - used to **compare** 2 models, A, B when applied on data x:

$$\delta(x) = A(x) - B(x)$$

- **What is hypothesis testing?**
  - formalisation of 2 hypotheses, as to confirm whether a result is significant (i.e not due to randomness)
  - we define:
    - the **null hypothesis**, H_0: the result is not **significant** (i.e could be caused by randomness)
    - the **alternate hypothesis**, H_a: the result is **significant**
  - for example:

$$H_0 : \delta(x) \leq 0, \textit{ model A is not better than model B}$$

$$H_0 : \delta(x) > 0, \textit{ model A is better than model B}$$

- **What is a p-value?**
  - we want to know whether the value $\delta(x)$ is abnormal, or consistent with the fact that A is better than B
  - the **p-value** is the **probability** of, assuming the **null-hypothesis** (that is, that A no better than B), observing a δ greater than (or equal) to $\delta(x)$:

$$P(\delta(X) \geq \delta(x) \mid H_0)$$

- if the **p-value** is **small**, it indicates that the null hypothesis might not be valid, since the $\delta(x)$ is extremely **abnormal** under H_0
- if the **p-value** is **large**, it indicates that the observed evidence is in line with the null hypothesis

- **When is a result statistically significant?**
  - when the **p-value** is extremely small, we **reject** H_0
  - when we reject the **null hypothesis**, we say that the result we observed (i.e "A is better than B") is **statistically significant**
  - the **threshold** for a **p-value** is typically 0.05 or 0.01

- **What is the paired bootstrap test?**
  - **bootstrapping** generates artificial observations, based on observed data, by **sampling with replacement** from the original data
  - in a **paired bootstrap test**, we compare 2 classifiers on different data, and take **bootstrap samples** on this data

	1	2	3	4	5	6	7	8	9	10	A%	B%	$\delta()$
x	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.70	.50	.20
$x^{(1)}$	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.60	.60	.00
$x^{(2)}$	AB	AB	AB	AB	AB	AB	AB	AB	AB	AB	.60	.70	-.10
...													
$x^{(b)}$													

Figure 1: The first row represents observations in the original data. We generate new observations $x^{(1)}, x^{(2)}, \ldots$ by sampling with replacement from x. We can then compute the performance of classifiers, alognsides δ.

- **How is a p-value computed during bootstrapping?**
  - if we generate b bootstrap tests, then the **p-value** can be the proportion of those tests in which the bootstrap δ was greater than or equal to $\delta(x)$:

$$p = \frac{1}{b}\sum_{i=1}^{b} \mathcal{X}_{(\delta(x^{(i)})-\delta(x)\geq 0)}$$

  - the data doesn't have 0 means, then instead we consider how often δ was greater than $\delta(x)$ by at least $\delta(x)$:

$$p = \frac{1}{b}\sum_{i=1}^{b} \mathcal{X}_{(\delta(x^{(i)})\geq 2\delta(x))}$$

function BOOTSTRAP(test set x, num of samples b) **returns** p-*value*(x)

Calculate $\delta(x)$ # how much better does algorithm A do than B on x
$s = 0$
for $i =$ 1 **to** b **do**
  for $j = 1$ **to** n **do** # Draw a bootstrap sample $x^{(i)}$ of size n
    Select a member of x at random and add it to $x^{(i)}$
  Calculate $\delta(x^{(i)})$ # how much better does algorithm A do than B on $x^{(i)}$
  $s \leftarrow s + 1$ **if** $\delta(x^{(i)}) \geq 2\delta(x)$
p-value(x) $\approx \frac{s}{b}$ # on what % of the b samples did algorithm A beat expectations?
return p-value(x) # if very few did, our observed δ is probably not accidental

3 Language Models

3.1 The Purpose of Language Models

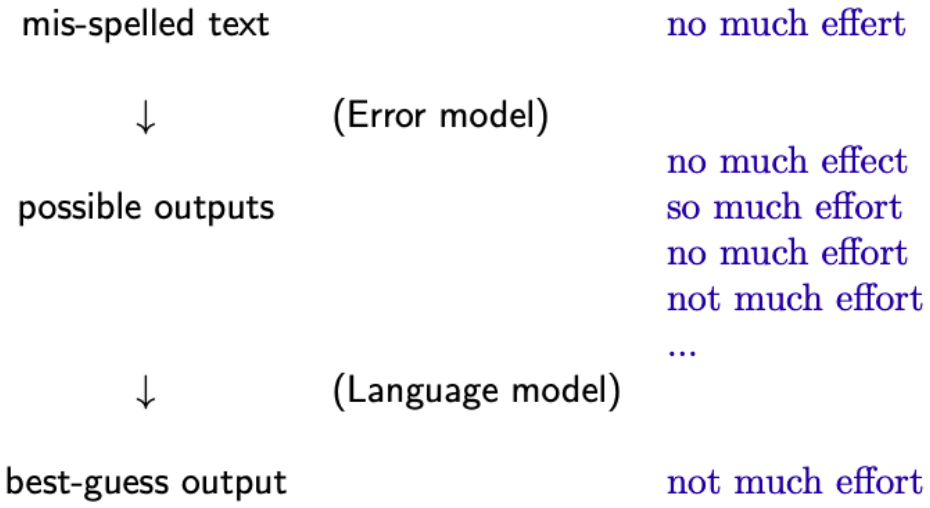
- **What is the probability of a sentence?**
  - the **likelihood** of a sentence being generated by a **natural language**
  - intuitively, we can see that sentences which make more "common sense" are more likely:

$$P(\textit{the cat slept peacefully}) > P(\textit{peacefully cat the slept})$$

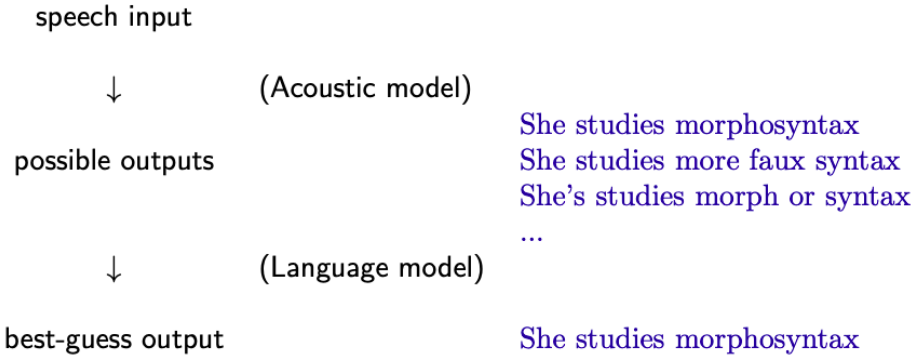
$$P(\textit{she studies morphosyntax}) > P(\textit{she studies more faux sin tax})$$

- **What is the purpose of a language model?**
  - **predicting/estimating** the approximate **probability** of a sentence
  - define a **probability distribution** over sequences of words
- **When can language models be used?**

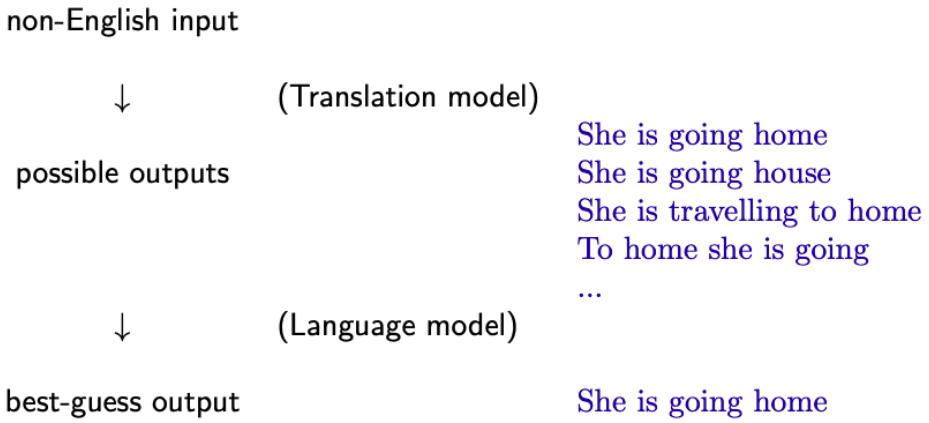
1. **Spelling Correction**: use an **error model** to determine spelling corrections for a phrase, and a **language model** to decide the most likely option:



2. **Automatic Speech Recognition**: use an **acoustic model** to turn sounds into possible sentences, and a **language model** to decide the most likely option:



3. **Machine Translation**: use a **translation model** to provide possible translations, and a **language model** to decide the most likely option:



4. **Prediction**: for example, when typing on your phone, gives you suggestions of misspelt words. In this case, the probability distribution can be over **characters**

- we use **separate** models because there is a lot of data to **judge** the feasability of a sentence, so we can train a LM well
- not enough data to train EM/AM/TM however, so makes sense to have these separate from LM

3.2 Language Models From MLE Estimation

- **How do probability theory and estimation theory differ?**
  - **probability theory** uses previous **evidence** to derive **true probabilities** of events
    - typical example of 4 red and 3 blue marbles; can easily derive the probability of picking a red marble
  - **estimation theory** uses data to **estimate** the **probability of events**
    - instead of knowing the number or colour of marbles, we know previous sampling events (i.e out of 10 samplings, 4 had red marbles, and 6 had blue marbles - compute the probability of seeing a red marble)
  - **language models** seek to **estimate** the probability of a **word sequence** $\underline{w}$ occurring
- **What is maximum likelihood estimation?**
  - estimates probabilities, such that they maximise the likelihood of observing the training data used:

$$P_{MLE}(S = \underline{w}) = \frac{C(\underline{w})}{N}$$

    where:
    - $\underline{w}$: sentence whose probability we estimate
    - $C(\underline{w})$: number of times $\underline{w}$ appeared in the training corpus
    - N: number of sentences in the training corpus
  - for example, if out of 1000 M&Ms we see there are 237 red ones, our MLE estimation would be $\frac{237}{1000} = 0.237$; this produces a distribution which best matches our observations during training
- **What is the main issue with MLE estimation?**
  - data is **sparse**, so there are many sentences in training with:

$$C(\underline{w}) = 0$$

    so their MLE estimate is:

$$P_{MLE}(S = \underline{w}) = 0$$

  - MLE thinks anything that hasn't occurred will never occur: not **enough observations** to estimate probabilities well simply by **counting** observed data

– this doesn't correspond with **reality**: unlikely, never-before written sentences can be perfectly valid:

"The Archaeopteryx soared jaggedly amidst foliage"

– in fact, many words occurring only once don't have the same probability (i.e cornflakes, mathematicians, pseudo-rapporteur, lobby-ridden, Lycketoft)

– many testing sentences will never occur during training

3.3 N-Gram Language Models

- **What are N-Gram Language Models?**
  - instead of estimating the probability of a **whole** sentence appearing, consider **subsentences**
- **How can we formalise an N-Gram Language Model?**
  - we want:

$$P(S = \underline{w}) = P(w_1, \ldots, w_n)$$

  - the chain rule for probability says that:

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

  - as of now, this is still subject to the **sparse data problem**. For example:

> *To compute the probability of:*
>
> $$\underline{w} = \text{"I spent three years before the mast."}$$
>
> *we would need to compute:*
>
> $$P(\textit{mast} \mid \textit{I spent three years before the})$$
>
> *and using an MLE estimation of this probability we get:*
>
> $$\frac{C(\textit{I spent three years before the mast})}{C(\textit{I spent three years before the})}$$
>
> *which are quite likely to get zero or very low counts.*

  - an N-gram model uses the **independence assumption** that the probability of a word occurring **only** depends on the $N-1$ previous words:

$$P(w_i \mid w_{i-N+1}, \ldots, w_{i-1})$$

> *For the previous example, if $N = 3$, then:*
>
> $$P(\textit{mast} \mid \textit{I spent three years before the}) \approx P(\textit{mast} \mid \textit{before the})$$

  - to compute the N-gram approximation, we can use MLE:

$$P(w_i \mid w_{i-N+1}, \ldots, w_{i-1}) = \frac{C(w_{i-N+1}, \ldots, w_{i-1}, w_i)}{C(w_{i-N+1}, \ldots, w_{i-1})}$$

– overall, our estimation for a phrase becomes:

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

- **What are common N-Gram Models?**

  1. a **unigram model** uses no past history:

     $$P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i) = \frac{C(w_i)}{V}$$

     where V is the number of words in the training corpus
  2. a **bigram model** uses the previous word:

     $$P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-1}) = \frac{C(w_{i-1}, w_i)}{C(w_{i-1})}$$

  3. a **trigram model** uses the 2 previous words:

     $$P(w_i \mid w_1, \ldots, w_{i-1}) \approx P(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)}{C(w_{i-2}, w_{i-1})}$$

- **Is the N-Gram assumption always good?**

  – no: many similar sentences with different probabilities will be assigned the same probability. The following all have approximate probability $P(mast \mid before\ the)$

  $$P(mast \mid I\ spent\ three\ years\ before\ the)$$
  $$P(mast \mid I\ revised\ all\ week\ before\ the)$$
  $$P(mast \mid I\ jumped\ around\ Mars\ before\ the)$$

  – moreover, sparsity can still be a problem (0 probabilities can still happen in short phrases)

  – nonetheless, it is a powerful model; as N increases we see great improvements. For example, we can generate Shakespeare using unigrams, bigrams, trigrams and tetragrams, with clear improvements:

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have –Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow. –What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done. –This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in; –It cannot be but so.

- **What is the tradeoff in N-Grams?**

- using **longer** histories captures a more coherent language (see above), but they are **sparse**
- using **shorter** histories reduces sparseness, but is a worse model

- **How do N-Grams account for the different distribution of words at the start, middle and end of sentences?**
  - different words are more likely to appear at the start, middle or end of sentences (for example, "hello")
  - N-grams introduce special tokens to signify:
    - the start of a sentence:

      `<s>`
    - the end of a sentence:

      `</s>`
  - these tokens are also necessary because:

> *" Without `<s>`, `</s>`, instead of the sentence probabilities of all sentences summing to one, the sentence probabilities for all sentences of a given length would sum to one. This model would define an infinite set of probability distributions, with one distribution per sentence length"*
> *More on this here.*

  - depending on N, we add more or less `<s>`:
    - **bigram**: `<s>`I like green eggs and ham.`</s>`
    - **trigram**: `<s><s>`I like green eggs and ham.`</s>`
  - alternative: model everything as a long sentence, and use punctuation as indicators

- **In practice, are probabilities computed using multiplication?**
  - since the probabilities are small, multiplying all of them would lead to infinitesimal probabilities
  - in practice, apply a **negative log** transformation, so we end up **summing** numbers between 0 and ∞

4 Evaluating Language Models

4.1 Considerations When Building an N-Gram Model

- **How is the development set used when developing an N-Gram Model?**
  - we can use it to construct several models with different N, and seeing which one performs best
  - we can construct the same model with different **smoothing**
  - if we used the **test set** to test the different models, we risk **overfitting**

- **How does the training data influence the model?**
  - a model trained on A will **learn** the idiosyncrasies of A
  - even if A, B are in the same **language**, the models produced will be radically different
    - for example, a model trained on WSJ vs Shakespeare:

1 gram	–To him swallowed confess hear both. Which. Of save on trail for are ay device and rote life have
	–Hill he late speaks; or! a more to leg less first you enter
2 gram	–Why dost stand forth thy canopy, forsooth; he is this palpable hit the King Henry. Live king. Follow.
	–What means, sir. I confess she? then all sorts, he is trim, captain.
3 gram	–Fly, and will rid me these news of price. Therefore the sadness of parting, as they say, 'tis done.
	–This shall forbid it should be branded, if renown made it empty.
4 gram	–King Henry. What! I will go seek the traitor Gloucester. Exeunt some of the watch. A great banquet serv'd in;
	–It cannot be but so.

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

  - it is important that **training** and **testing** data are chosen from the same **genre** and **dialect**:
    * to train an SMS text prediction model, use a corpus of SMS texts
    * to train a model for business meetings, use a corpus of business meeting transcriptions
    * African American communities tend to use different vocabulary to express themselves (i.e "finna")

- **What is the closed vocabulary assumption?**
  - we know all the words that can appear (during **training** and **testing**)
  - feasible in **speech recognition** and **machine translation** (they use a **fixed dictionary** of "allowed" lexicon)

- **What is an open vocabulary system?**
  - used to **model** the existence of **out of vocabulary** (OOV) or **unknown** words
  - includes the token:

    `<UNK>`

- **How are unknown word models trained?**

  1. **Convert to Closed Vocabulary**
     - define a **fixed vocabulary**
     - convert any **unknown** training word into `<UNK>`
     - estimate probabilities for `<UNK>` (as if it were a word)
  2. **Implicit Closed Vocabulary**
     - used if there is no pre-defined **closed vocabulary**
     - replace with `<UNK>` any training word appearing less than n times
     - train, using `<UNK>` as a word

4.2 Evaluation in NLP

- **What is extrinsic evaluation?**
  - measuring **performance** using an external application (i.e Google Translate)
  - most **reliable** (since used in a situation where it matters)
  - **time-consuming** (need to try many different models)
  - requires a measure of **performance** in the application
- **What is intrinsic evaluation?**
  - design a **measure** inherent to the task
  - **quicker** and **easier** during development (it applies directly to the model)
  - need to determine a **measure** which (hopefully) correlates with **extrinsic** measures
- **Is accuracy a good measure?**
  - **accuracy** of a model is an **intrinsic** measure:

$$\sum_{\underline{w} \in T} \frac{\underline{w}_n == \hat{\underline{w}}_n}{|T|}$$

    where T is the corpus of sentences of length n, and for each sentence $\underline{w} = w_1, w_2, \ldots, w_n$, the model predicts $\hat{\underline{w}}_n$ from $w_1, w_2, \ldots, w_{n-1}$.
  - this is not ideal, since even if $\underline{w}_n != \hat{\underline{w}}_n$, it might still be an appropriate/correct choice
  - we need **flexible** measures that account for the **variability** of language
    - "Pizza with cheese" and "Pizza with peppers" are both equally good constructions
- **Is a measure informative on its own?**
  - no: the value of a metric will depend on the **model** and the **corpus**
  - if a **corpus** is predictable, we expect metrics to be generally favourable, but that doesn't mean that a model is "good" (just that it is "easy" to learn)
  - need to compare metrics for **different models** on the **same corpus**
  - when comparing metrics, use the **testing/held-out data** (ensures an "even playing field)

4.3 Entropy

- **What is entropy?**
  - a measure of the **uncertainty** in a **probability distribution**, designed as an **intrinsic** measure
  - if X is a RV:

$$H(X) = -\sum_{x \in X} P(x) \log_2(P(x)) = E[-\log_2(P(X))]$$

  - practical experience suggests **entropy**-based metrics correlate with **extrinsic** evaluation.
- **What is an intuitive view of entropy?** (For a nice article on entropy and bits, see this Towards Data Science article.)
  - the **lower** the entropy, the **lower** the uncertainty of a model (it is more **confident** in its prediction)
  - can be phrased in terms of **bits** or **yes/no** questions:
    - how many questions would you need, in order to be certain about an outcome?
    - how many bits are necessary to encode an outcome? (for example, two outcomes can be encoded using a single bit - 0 or 1; 4 outcomes require 2 bits: 00,01,10,11).

Say we predict a pet, and there are 4 equally likely possibilities (dog, cat, parrot or goldfish). Then the entropy will be:

$$H(X) = -4 \times 0.25 \times \log_2(0.25) = -1 \times -2 = 2$$

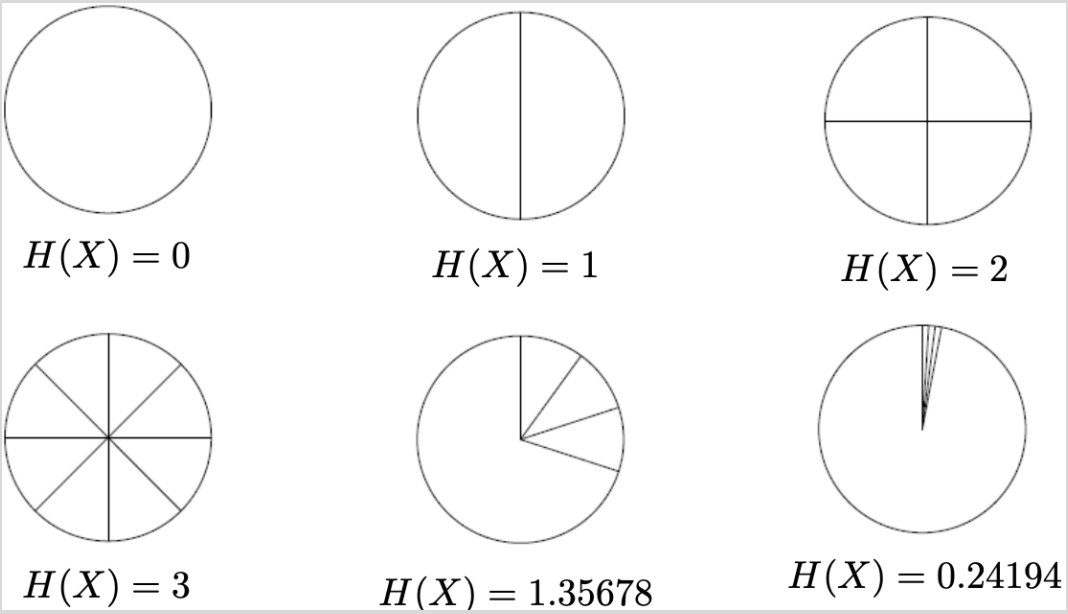
In other words, we need to ask 2 yes/no questions to determine which of the outcomes happened (for example: "Is it a mammal?" and "Does it fly?"). Alternatively, the outcome that happened is uniquely represented by 2 bits (say 10, a parrot).
If we again have 4 outcomes, but the person choosing is a dog lover and:

$$P(dog) = 0.97 \qquad P(\neg dog) = 0.01$$

then:

$$H(X) = -0.97 \times \log_2(0.97) - 3 \times 0.01 \times \log_2(0.01) = 0.24194$$

We need nearly 0 bits/questions: that is, we are very certain that the outcome is probably "dog". Nonetheless, we still capture the fact that at times we might need to ask a question, since there is uncertainty.



If we have 3 events, and 1 is more likely than the others, we could encode this using:

$$a = 0 \qquad b = 10 \qquad c = 11$$

In general, $H(X) \leq$ the average number of bits needed to encode X

4.4 Cross-Entropy

- **How does cross-entropy differ from entropy?**
  - in entropy, we are using **true probabilities**; LMs only **estimate** these probabilities

- **cross-entropy** measures how close the estimate $\hat{P}$ is to the true probability P:

$$H(P, \hat{P}) = -\sum_{x \in X} P(x) \log_2(\hat{P}(x))$$

- we have that:

$$H(P, \hat{P}) \geq H(X)$$

since the model **uncertainty** is at least as big as the **true uncertainty**

- **Cross-entropy still contains $P(x)$. How do we cope with this?**
  - use **per-word cross-entropy**, which can be approximated by:

$$H_M(w_1, \ldots, w_n) = -\frac{1}{n} \log_2(P_M(w_1, \ldots, w_n))$$

where M is our LM, P_M is the probability assigned by the model, and n is the number of words in each sentence.

  - **per-word** since we **normalise** using the length of the sentence
  - a lower H_M indicates a more confident model, which better predicts the next word

$$\begin{aligned}
& -\tfrac{1}{7}(\quad \lg_2(P(I)) + \lg_2(P(spent|I)) + lg_2(P(three|spent)) + \lg_2(P(years|three)) \\
& \qquad + \lg_2(P(before|years)) + \lg_2(P(the|before)) + \lg_2(P(mast|the)) \quad) \\
= \quad & -\tfrac{1}{7}(\quad -6.9381 - 11.0546 - 3.1699 - 4.2362 - 5.0 - 2.4426 - 8.4246 \quad) \\
= \quad & -\tfrac{1}{7}(\quad 41.2660 \quad) \\
\approx \quad & 6
\end{aligned}$$

Figure 2: The per-word cross-entropy for "I spent three years before the mast", using a bigram model. If we did this with a unigram model, the per-word cross-entropy would be about 11. Hence, a bigram has about 5 bits of uncertainty less.

- **How do language models contribute to infromation theory?**
  - better language models can lead to better **data compression**
  - using the above example:
    - a unigram uses $7 \times 11 = 77$ bits to encode a 7 word sentence
    - a bigram uses $6 \times 7 = 42$ bits to encode a 7 word sentence
    - on average, ASCII requires $24 \times 7 = 168$ bits to encode a 7 word sentence

4.5 Perplexity

- **What is perplexity?**
  - the preferred way of reporting LM performance:

$$PP_M(w_1, \ldots, w_n) = 2^{H_m(w_1, \ldots, w_n)} = (P_M(w_1, \ldots, w_n))^{-\frac{1}{n}} = \sqrt[n]{\frac{1}{P_M(w_1, \ldots, w_n)}}$$

  - for a bigram model:

$$PP_B(w_1, \ldots, w_n) = \sqrt[n]{\frac{1}{\prod_{i=1}^{n} P_m(w_i \mid w_{i-1})}}$$

- the **lower** the **perplexity**, the **higher** the confidence of the model (a corpus becomes more **predictable**)

- **How else can we think of perplexity?**
  - the weighted average **branching factor** of a language
  - the **branching factor** indicates the **number** of words which can follow a given word
  - the following are equivalent in terms of uncertainty:
    * 6 bits of cross-entropy
    * perplexity of model is 64
    * the uncertainty of a uniform distribution with 64 possible outcomes

If we have a language with words $0, 1, \ldots, 9$*, and* $P(n) = \frac{1}{10}$*, then the branching factor is 10, and indeed:*

$$PP(W) = P(w_1, \ldots, w_n)^{-\frac{1}{n}} = \left(\frac{1}{10^n}\right)^{-\frac{1}{n}} = 10$$

However, if we observe a string of 100 words, $W = 0000 \ldots 3001$*, with 91 0s, and 1 of each other digit, we will see we will have a much lower perplexity:*

$$PP(W) = \left(\frac{91^{91} \times 1^9}{100^{100}}\right)^{-\frac{1}{100}} \approx 1.65$$

since we have a much more predictable environment, so the model is more confident (even if the branching factor is still 10, the ***weighted*** *branching factor changes).*
This can be further seen if we compare the perplexity across unigrams, bigrams and trigrams, all trained on the same corpus. The branching factor will be the same, but the average branching factor (perplexity) differs:

	Unigram	**Bigram**	**Trigram**
Perplexity	962	170	109

5 Smoothing

5.1 Why Use Smoothing

- **How does sparsity affect evaluation?**
  - because of **sparsity**, many probabilities are 0: LM thinks a construction doesn't **exist**
  - **perplexity** then will be ∞ (it is infinitely surprised by a new phrase, it would **never** expect it)
  - this can happen even with **unigrams**

- **What is smoothing?**
  - MLE **maximises** the likelihood of **seen** events by **minimising** the likelihood of **unseen** events
  - **smoothing** "steals" probability mass from **seen events**, and "donates" it to **unseen events**
  - informally: **halluccinate counts for everything, seen and unseen**

5.2 Laplace Smoothing

- **What is Laplace Smoothing?**
  - we hallucinate that we have seen **everything** at least **once**
  - in a trigram model:

$$P_{+1}(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i)+\mathbf{1}}{C(w_{i-2}, w_{i-1})+\boldsymbol{V}}$$

We want:

$$\sum_{w_i \in V} \frac{C(w_{i-2}, w_{i-1}, w_i) + 1}{C(w_{i-2}, w_{i-1}) + x} = 1$$

Solve for x:

$$\begin{aligned}
\sum_{w_i \in V} (C(w_{i-2}, w_{i-1}, w_i) + 1) &= C(w_{i-2}, w_{i-1}) + x \\
\sum_{w_i \in V} C(w_{i-2}, w_{i-1}, w_i) + \sum_{w_i \in V} 1 &= C(w_{i-2}, w_{i-1}) + x \\
C(w_{i-2}, w_{i-1}) + v &= C(w_{i-2}, w_{i-1}) + x \\
v &= x
\end{aligned}$$

Figure 3: Since the **numerator** increases (everything is seen once), the denominator also increases (we have seen all of the vocabulary V at least once) - this ensures that the sum of probabilities is 1.

- **What are the issues with Laplace Smoothing?**
  - we assume that we know V in **advance**
    - if not, just use `<UNK>` when encountering an unknown word in testing)
  - it steals **too much probability mass**
  - this means that what MLE is good at (predicting frequent events) gets taken away by smoothing

	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

Figure 4: Consider the following counts from a corpus.

	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

Figure 5: We can compute probabilities for a bigram model, getting the following. For example, $P(want \mid i) = 0.33$.

	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

Figure 6: We can apply Laplace Smoothing, adding 1 to each count.

	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

Figure 7: We can then compute the bigram probabilities. We now see that $P(want \mid i) = 0.21$.

	i	want	to	eat	chinese	food	lunch	spend
i	3.8	527	0.64	6.4	0.64	0.64	0.64	1.9
want	1.2	0.39	238	0.78	2.7	2.7	2.3	0.78
to	1.9	0.63	3.1	430	1.9	0.63	4.4	133
eat	0.34	0.34	1	0.34	5.8	1	15	0.34
chinese	0.2	0.098	0.098	0.098	0.098	8.2	0.2	0.098
food	6.9	0.43	6.9	0.43	0.86	2.2	0.43	0.43
lunch	0.57	0.19	0.19	0.19	0.19	0.38	0.19	0.19
spend	0.32	0.16	0.32	0.16	0.16	0.16	0.16	0.16

Figure 8: We can reconstitute the adjusted counts for each bigram, to see how much mass was stolen. The formula for this is:

$$C^*(w_{n-1}, w_n) = \frac{(C(w_{n-1}, w_n) + 1) \times C(w_{n-1})}{C(w_{n-1}) + V}$$

We can see that "I want" went from appearing 827 times, to having an adjusted count of only 527.

5.3 Lidstone Smoothing

- **What is Lidstone Smoothing?**
  - instead of hallucinating everything happens **at least once**, we hallucinate it happening at least $\alpha < 1$ times:

$$P_{+\alpha}(w_i \mid w_{i-2}, w_{i-1}) = \frac{C(w_{i-2}, w_{i-1}, w_i) + \boldsymbol{\alpha}}{C(w_{i-2}, w_{i-1}) + \boldsymbol{\alpha V}}$$

  - also assumes that we know V
- **How is α chosen?**
  - use the **development set**:
    1. *training set*: train different models with varying α
    2. *development set*: pick model with **lowest cross-entropy**
    3. *test set*: report results of best model

5.4 Good-Turing

- **What is Good-Turing Smoothing?**
  - instead of changing the **denominator**, changes the **numerator** by creating **adjusted counts**
    - instead of hallucinating new counts, we keep the **total** observation the same, and just change (discount) the number of times we make a certain observation (i.e "I want" can be seen 3.7 times instead of 4 times)
    - the discounted probability mass can then be used to fill in problems caused by sparsity
- **What are the adjusted counts in Good-Turing?**
  - in MLE, if c is the number of times we see an n-gram (i.e "Yesterday I played football"), and n is the number of times we've seen the context (i.e "Yesterday I played"), the probability is:

$$P_{MLE} = \frac{c}{n}$$

  - in Good-Turing, we use an adjusted count c^*:

$$P_{GT} = \frac{c^*}{n}$$

  - define N_c to be the number of n-grams appearing c times (this will follow a **Zipf Curve**)

    - for example, N_0 can be the number of bigrams we've never seen, N_1 is the number of bigrams we've only seen once, N_2 are bigrams we've seen twice, etc ...
  - the new adjusted count c^* will be:
$$c^* = \frac{(c+1)N_{c+1}}{N_c}$$
    - for example, if "Yesterday I played football" appeared only 3 times, we adjust its count to:
$$c^* = \frac{4N_4}{N_3}$$
    - by Zipf, typcially $N_4 < N_3$, so $c^* < c$
  - the **probability** of n-grams seen c times changes. If N is the size of the corpus:
$$P_c = \frac{c}{N} \implies P_{c^*} = \frac{c^*}{N} = \frac{\frac{(c+1)N_{c+1}}{N_c}}{N}$$
  - so does the **total probability mass**:
$$P_c[total] = \frac{N_c}{N} \implies P_{c^*}[total] = \frac{c^*}{N} = \frac{(c+1)N_{c+1}}{N}$$
    - this means that those n-grams seen 0 times will have a total probability mass of $\frac{N_1}{N}$ - we can see that, considering all the discounts done to n-grams seen at least once, their sum is indeed $\frac{N_1}{N}$

- **How do we compute N_0, if we don't know how many n-grams will appear 0 times?**
  - we can't always know how many n-grams are missing
  - typically approximate using:
$$N_0 = V^2 - N$$
where V is the vocabulary size and N is the size of the corpus

- **How can we justify the adjusted counts?**
  - we consider the MLE probability of observing an n-gram c times, given that it has appeared $c-1$ times already
  - if something is previously unseen, the probability that it is seen next is:
$$P(unseen) = \frac{N_1}{N}$$
  - if we distribute it equally amongst all things we haven't seen before:
$$P_{GT}(unseen) = \frac{1}{N_0}\frac{N_1}{N} = \frac{\left(\frac{N_1}{N_0}\right)}{N}$$
so we must have that:
$$c^* = \frac{N_1}{N_0}$$
  - similarly, if we've already seen the n-gram, the probability of seeing it next is:
$$P(seen\ once) = \frac{2N_2}{N}$$
(since once we see it again
  - if we distribute it amongst all things seen once already:
$$P_{GT}(seenonce) = \frac{1}{N_1}\frac{N_2}{N}$$
so again:
$$c^* = \frac{N_2}{N_1}$$

- **What are the main issues of Good-Turing?**
  - again, we assume knowledge of vocabulary size (use `<UNK>`)
  - relies on $N_{c+1} < N_c$; good for low counts (Zipf curve); however, towards the end of the distribution, there are bumps (i.e certain n-grams can be seen 47 times, but none are seen 48 times, and again some are seen 49 times) - potentially fix by limiting GT to first counts/use linear regression to estimate values in gaps
  - high frequency counts discounted, when these are the most reliable counts