The Noisy-Channel Model

One way to predict the most likely sequence of words given the acoustic features (like the frequency and amplitude of an audio signal) is by using the noisy-channel model. In this method, a true word is believed to be masked by some noise (like substitutions/insertions/deletions of the letters) when passed through a channel. So by replicating this channel using a probabilistic model, we can essentially find the true word that when passed through this channel gives a word closest to the misspelled word. This model is based on Bayes Rule as follows:

\[ w^* = \underset{w \in V}{\text{argmax}} P(w|a) \]

This means that, out of all the words \( w \) in the vocabulary \( V \) we would like to find the best word \( w^* \) that maximises the probability of the word given the observation \( a \) of the misspelled word. Further,

\[ w^* = \underset{w \in V}{\text{argmax}} \frac{P(w, a)}{P(a)} \]
\[ = \underset{w \in V}{\text{argmax}} \frac{P(a|w)P(w)}{P(a)} \]

Here, for a given misspelling \( a \), the maximising function for any two words \( w_1 \) and \( w_2 \) of the vocabulary would have \( P(a) \) as a constant. Hence, it can be ignored.

\[ w^* \propto \underset{w \in V}{\text{argmax}} P(a|w)P(w) \]

The above equation has two factors: the language model \( (P(w)) \) and the acoustic model \( (P(a|w)) \). The **language model tells us how probable is it for a word \( w \) to actually be a word (by itself or in the context).** And given this word \( w \), the acoustic model tells us the likelihood of generating the exact error as the misspelled word \( a \).

Machine Translation

Similarly, given a French sentence \( f \) to a machine translation system, it’s equivalent English sentence \( e \) using the noisy channel model can be given by

\[ e^* = \underset{e \in V}{\text{argmax}} P(e|f) \]
\[ = \underset{e \in V}{\text{argmax}} \frac{P(e, f)}{P(f)} \]
\[ = \underset{e \in V}{\text{argmax}} \frac{P(f|e)P(e)}{P(f)} \]
\[ \propto \underset{e \in V}{\text{argmax}} P(f|e)P(e) \]
Also, spelling correction, handwriting recognition and OCR are the other processes that can be built using the noisy-channel model.

### Probabilistic Language Models

**Goal:**
Given a training data of numerous sentences $x$ we would like to assign the probabilities $P(x)$ to each of the sentences.

**Reason:**
These probabilities $P(x)$ would make the model to output the most plausible sentence on the test data.

**Explanation:**
Though the sentences "I saw a van" and "eyes awe of an" are phonetically similar, the model should be trained to assign a higher probability to the first sentence than to the second as it is much less likely to have a sequence of words as in "eyes awe of an".

$$P(I\ saw\ a\ van) \gg P(eyes\ awe\ of\ an)$$

**Method 1: $P(Sentence)$ - Empirical Distribution Over Training Sets**
One way of estimating $P(x)$ would be by considering the empirical distribution over training sentences. That is, estimating the probability of a sentence in terms of the number of times the sentence appears as it is in the corpora.

$$P(\text{Today is September}) = \frac{\text{Count}(\text{Today is September})}{\text{Total no. of sentences}}$$

But, this could cause a problem during the test time. Though the training and the test data are drawn from the same distribution they might not actually be similar. Hence, the no. of times the sentence "Today is September" is present in the training data cannot, actually, lead to the count of exactly the same sentence being present in the test data.

**Method 2: $P(\text{Words})$ - Generalization**
Hence, by intuition, it makes more sense to predict the probability of words rather than the probability of the sentence itself. Also, decomposing the sentences into words gives us the flexibility of

- recombining them in new ways (conditional independence)
- smoothing which allows us to handle the unseen (UNKNOWN or UNK) words in a better way

Thus, we now have two ways of computing the probability:

- **Joint Probability (Probability of a sentence or a sequence of words)**
  $$P(w) = P(w_1, w_2, w_3, ... w_n)$$

- **Conditional Probability (Probability of an upcoming word given the previous words)**
  $$P(w) = P(w_n | w_1, w_2, w_3, ... w_{n-1})$$

A model that computes either of the above is called a **language model**.
N Gram Model Decomposition

The joint probability of any sentence, using the Chain Rule, can be expressed as a product of the probabilities of an $n^{th}$ word given the previous $n-1$ words.

$$P(w_1w_2w_3...w_n) = \prod_i P(w_i|w_1, w_2, ...w_{i-1})$$

For example,

$$P("I am a student") = P(I) \times P(am|I) \times P(a|I am) \times P(student|I am a)$$

But, as it’s impossible to calculate the total number of times a sentence is present in English language we consider a simplifying assumption called the Markov assumption. According to this assumption, a word can be considered to be dependent only on a few previous words.

Unigram

In unigram language model, the probability of a sequence of words is estimated as the product of probabilities of only the individual words.

$$P(w_1w_2w_3...w_n) \approx \prod_i P(w_i)$$

Disadvantage: $P(\text{the the the}) \gg P(\text{I like ice cream})$

Bigram

Bigram model considers every word to be dependent only on its immediate previous word.

$$P(w_1w_2w_3...w_n) \approx \prod_i P(w_i|w_{i-1})$$

For example,

$$P(\text{please close the door}) = P(\text{please}|\text{START}) \times P(\text{close}|\text{please}) \times P(\text{the}|\text{close}) \times P(\text{door}|\text{the}) \times P(\text{STOP}|\text{door})$$

N-Gram Model Parameters

Let $p$ be the probability of seeing a word $w$ after $w_{-1}$ in the entire text. We wish to maximize the log likelihood for finding out the MLE $\hat{P}$.

Now assume that the data has $k$ instances of $w$ followed by $w_{-1}$, and a total of $n$ instances of $w_{-1}$.

$$P(\text{data}) = p.(1-p)p....(1-p)...$$

$$= p^k(1-p)^{n-k}$$
This is precisely the likelihood. We can take the logarithm of both sides and find $\hat{P}$ by finding the $p$ which maximizes this log likelihood.

$$\hat{P} = \arg\max_{0 \leq p \leq 1} \log(p^k(1-p)^{n-k})$$

$$= \arg\max_{0 \leq p \leq 1} k \log(p) + (n-k) \log(1-p)$$

Now differentiating wrt. $p$ and setting to 0, we find $\hat{P}$,

$$\frac{k}{\hat{P}} = \frac{n-k}{1-\hat{P}}$$

or,

$$\hat{P} = \frac{k}{n} \frac{ct(w_{-1},w)}{\sum_{w'} ct(w_{-1},w')}$$

The computational complexity of the model would be $O(\text{No. of tokens})$.

**Problems With Unsmoothed N-Gram Language Models**

- As language can have long-distance dependencies it might not be a good idea to calculate the probability based on just a few preceding words (This problem can be rectified, in most of the cases, by using 4-gram or 5-gram).
- The unseen words in the training data will be assigned a probability of 0 during the test time.
- Sparsity:
  A vast majority of words are uncommon. Hence, longer n-grams involving these uncommon words are much rarer.

**Measuring Model Quality**

Any two models $A$ and $B$ can be compared using the evaluation metrics. The evaluation metrics can be calculated either through extrinsic evaluation or intrinsic evaluation.

**Extrinsic Evaluation:**
Both the models are put into a task (like spelling corrector or speech recognizer or machine translation system) and the accuracies calculated for models $A$ and $B$ (by looking at the number of misspelled words corrected properly or that of the words translated correctly) are compared. While this is a credible way of evaluation it might take a long time for the task to be completed.

**Intrinsic Evaluation:**
In cases where the test data resembles the training data, a model can be evaluated by calculating it’s perplexity. **Perplexity** is the probability of the test set, normalized by the number of words.

$$\text{perp}(W) = P(w_1w_2w_3...w_n)^{\frac{1}{n}}$$

Or,

$$\text{perp}(X, \theta) = 2^{H(X|\theta)}$$
where $H(X|\theta)$ is called **Entropy** (per word test log likelihood) and is given by

$$H(X|\theta) = -\frac{1}{|X|} \sum_{x \in X} \log_2 P(x|\theta)$$

The lower the perplexity the better the model. And the best entropy a model can have is zero.

**Word Error Rate:**
It is one of the extrinsic evaluation metrics given by

$$WER = \frac{Insertions + Deletions + Substitutions}{True \ sentence \ size}$$

**Zipf’s Law**

It states that given the frequency $f$ of a word and it’s rank $r$ in the list of words ordered $f \propto \frac{1}{r}$ or $f \times r = constant$

---

**Smoothing**

Maximum likelihood estimates suffer from sparsity. Such a language model based on a N-gram scheme is restricted by outputting non zero probabilities only for the instances seen yet in the training data. This problem can be alleviated somewhat by using **smoothing**.

We may do this using the general method (procedurally) which basically involves taking the empirical counts and modifying them in various ways to improve estimates or the same may be done by using a formal statistical interpretation.

The former may be accomplished by stealing counts from words which have non zero counts and distributing them to the words with no count. Stealing a fixed ratio from the higher counts penalizes them more. This is because we are more confident about their occurrence and the amount of counts stolen from them is still relatively high. More formal mathematical approaches based on statistical interpretations are discussed below.
Add one smoothing (Laplace smoothing/Dirichlet prior)

This smoothing is called a Dirichlet prior because we assume that we have seen all the words once before actually seeing the data. For a unigram model add 1 or $\delta$ to all counts, and normalize accordingly.

$$P_{\text{add-}\delta}(x) = \frac{c(x) + \delta}{\sum_{x'} c(x') + \delta}$$

For a bigram model counts shaped like unigram are added.

$$P_{\text{dir}}(w|w_{-1}) = \frac{c(w|w_{-1}) + k\hat{P}(w)}{\sum_{w'} c(w'|w_{-1}) + k}$$

Shortcomings of the add one smoothing are that in a usual text the number of unseen bigrams is way higher than the bigrams in the training text, and add one smoothing shifts a lot of the mass of the probability distribution to the portion of unseen words. This makes us believe that the probability of a unseen bigram is too high, whereas languages are very repetitive.

**Held out reweighing**

As seen in class, the empirical results suggest that the count of bigram words seen in a corpora of text tend to decrease by a constant over the next corpus of same size on an average. Also, the add anything method vastly underestimates the expected ratios of the occurrences of words with count 2 to that of words with count 1.

**Linear Interpolation**

In the Trigram model sparsity is an even bigger problem as $\hat{P}(w|w_{-1}, w_{-2})$ is supported by even fewer counts. A simple workaround for this problem is to construct a mixture model of the less complex models.

$$P(w|w_{-1}, w_{-2}) = \lambda \hat{P}(w|w_{-1}, w_{-2}) + \lambda' \hat{P}(w|w_{-1}) + \lambda'' \hat{P}(w)$$

The disadvantage of this model is that by using $\lambda$ we are penalizing all the trigrams equally. This is not ideal as we should not penalize the trigrams with high counts as we are certain of their occurrence. Similar explanation follows for $\lambda'$ and $\lambda''$.

**Absolute Discounting**

$$P_{\text{ad}}(w|w') = \frac{\max(c(w', w) - d, 0)}{\sum_w c(w', w)} + \alpha(w')P(w)$$

$$\alpha(w') = \frac{d \times \{w : c(w', w) > 0\}}{\sum_w c(w', w)}$$

**Problems with Absolute discounting**

$$\hat{P}(w) = \sum_{w'} \hat{P}(w|w')P(w')$$
But because of the discounting, the property does not hold.

\[ \hat{P}(w) \neq \sum_{w'} \hat{P}_{ad}(w|w')P(w') \]

This discounting basically falls back to the maximum likelihood unigram model if the count is low.

Note: Do not confuse this inequality with the total law of probability. Absolute discounting does maintain the law of total probability in that if you marginalize over \( w' \) the resulting unigram distribution \( \hat{P}_{ad}(w) \) will sum to one. \( \sum_w \hat{P}_{ad}(w) = 1 \). The problem is that it is not "calibrated" i.e. it is not the same as the MLE estimate \( \hat{P}(w) \).

**Kneser Ney Smoothing**

In this smoothing procedure, we modify the lower order second term. For example, if we use the absolute discounting in the bigram model, the model would only consider the unigram probability. This discounting will not only take into account the current context (unigram probability when smoothing bigram model), but also the number of contexts that the word appears in.

The example used in class comparing New York and Crayon can be further explained. Let us assume a corpora of text which is abundant of the word pair ‘New York’. Now the unigram probability of York will be high.

\[ \hat{P}(York) > \hat{P}(Crayon) \]

Using absolute discounting in such a case where the bigram model is weak and the lower model overtakes will cause undesired results.

Example,

He scribbled on it with _____.

Though crayon should fill in this blank, but because New York is a common term, absolute discounting might output York instead of crayon. Kneser Ney modification will take care of this problem and use the Diversity of History instead. i.e,

\[ \hat{P}_{kn}(York) < \hat{P}_{kn}(Crayon) \]

\[ P_{kn}(w|w') = \frac{\max(c(w', w) - d, 0)}{\sum_w c(w', w)} + \alpha(w')P_{kn}(w) \]

**Idea:**

This probability measure should be constrained to satisfy the MLE distribution, i.e., the lower order term is fixed so that the problem is calibrated.

\[ \hat{P}(w) = \sum_{w'} \hat{P}_{kn}(w|w')P(w') \]

or in other words,

\[ c(w) = \sum_{w'} \hat{P}_{kn}(w|w')c(w') \]

**Notation:**

\[ N_{1+}(w', .) := \{|w : c(w', w) > 0| \} \]

\[ N_{1+}(., w) := \{|w' : c(w', w) > 0| \} \]
\[ N_{1+}(\cdot, \cdot) := \sum_{w'} N_{1+}(w', \cdot) \]

**Claim:**

\[ P_{kn}(w) = \frac{N_{1+}(\cdot, w)}{N_{1+}(\cdot, \cdot)} \]

**Derivation:**

\[
c(w) = \sum_{w'} c(w') \left( \frac{\max(c(w', w) - d, 0)}{\sum_w c(w', w)} + \frac{d}{\sum_w c(w', w)} N_{1+}(w', \cdot) P_{kn}(w) \right) \\
= \sum_{w'} c(w') \frac{\max(c(w', w) - d, 0)}{c(w')} + \sum_{w'} c(w') \frac{d}{c(w')} N_{1+}(w', \cdot) P_{kn}(w) \\
= c(w) - N_{1+}(\cdot, w) d + d \times P_{kn}(w) \times \sum_{w'} N_{1+}(w', \cdot) \\
= c(w) - N_{1+}(\cdot, w) d + d P_{kn}(w) N_{1+}(\cdot, \cdot)
\]

Solving for \( P_{kn}(w) \) gives,

\[ P_{kn}(w) = \frac{N_{1+}(\cdot, w)}{N_{1+}(\cdot, \cdot)} \]

The important thing to note here is that this \( P_{kn} \) is based on the 'diversity' of the history, i.e, it is based on the count of unique bigrams instead of the total number.