What are n-grams and how are they used to build a data model

An n-gram is a sliding window of size n over a piece of text. Unigrams, n-grams of size 1 are simply all the words (and possibly symbols) in a piece of text.

N-grams can be used to build probabilistic language models, where the choice of corpus greatly influences the model. These statistical models can predict the likelihood of a given sequence of words in a language. The basic idea of such models is predicting the probability of a word occurring given the words that preceded it.

List a few applications where n-grams could be used

In the assignment, ngrams are used to predict the language of a sentence, but they have many more applications. Some other applications include:

- Testing the accuracy of voice-to-text applications by predicting the likelihood of the resulting sentence.
- They can be helpful for translation applications by identifying similar sequences of words in different languages
- They can be used for text generation by predicting the most likely next words given the current sentence, although these often do not work very well since they lack an understanding of semantics and context, thus often babbling on syntactically correct sentences that have no meaning.

How are probabilities calculated for unigrams and bigrams

Computing the probability of a unigram in a piece of text is relatively simple, as one can simply take the count of that word and divide by the total number of tokens in the text:
$P(w1) = C(w1) / N$

Calculating the probability of a sequence of words is a little more complicated. Essentially, we multiply the probabilities of each word in the sequence given the previous words:
$P(w1, w2, w3, w4) = P(w1)P(w2 \mid w1)P(w3 \mid w1, w2)P(w4 \mid w1, w2, w3)$

This formula can be made a little simpler by making a Markov assumption and looking only at the previous word:
$P(w1, w2, w3, w4) = P(w1)P(w2 \mid w1)P(w3 \mid w2)P(w4 \mid w3)$

On top of this, several smoothing techniques can be applied to fill in 0 values and give us more significant results, such as filling in 0s with a small probability mass, using LaPlace smoothing, or using Good-Turing smoothing

Importance of the source text in building a language model

In an n-gram language model, all of the knowledge possessed by the model comes directly from the source text, and thus it is important to pick an appropriate source text. The vocabulary of the language model will be the same as the source text, thus if the source text is small, the model may not recognize as many words in a new piece of text. The accuracy of the model also depends on the source text, thus if the source text has misspellings or grammar mistakes, these will also affect the model. The domain of the source text must also be taken into account, as it must be representative of the domain in which the model will be used (for example training a model on a medical corpus will not be very useful if we plan to use it in a financial context).

Importance of smoothing and a simple approach to smoothing

Smoothing is important because not every possible n-gram will be in the generated dictionaries of our model. This means that we may encounter some ngrams while processing new text that have probabilities of 0. If you recall the probability computation from earlier, the probabilities of these n-grams are multiplied, and thus having a probability of 0 will make the entire equation equal 0.

A simple approach to smoothing is to fill in the 0 values with some probability mass, so that they are a very small number, but not 0

How language models can be used for text generation and the limitation of this approach

Language models can be used for text generation by computing the most likely next word given the current sequence of words, and applying this recursively to generate a piece of text. The problem with this approach is that the generated text will likely have no coherent semantic meaning, since the n-gram model is not keeping any context on the larger topic of the generated text, it is simply probabilistically filling in the most likely next word. Such an approach will often result in syntactically correct but semantically incoherent pieces of text.

How language models can be evaluated

The more straightforward approach is to use human annotators to evaluate the quality of the language model. Another approach is to use an internal metric such as perplexity, which quantifies how well a language model predicts a set of test data. Perplexity (PP) can be computed as such:

$$PP(W) = P(w_1 w_2 \ldots w_N)^{-1/N}$$

The lower the perplexity, the better the model

Google's n-gram viewer

Google has a fun n-gram viewer tool accessible at https://books.google.com/ngrams/
It can be used to view the historical frequencies of certain n-grams. The data is pulled from Google books. The tool plots the probabilities of the n-gram occurring across all book that satisfy the given parameters.

Here is an example: