# Introduction to NLP (CS7.401)

Spring 2022, IIIT Hyderabad
07 Jan, Friday (Lecture 2)

Taught by Prof. Manish Shrivastava

## Tokenisation

Words are the smallest meaningful units of a language. They are the primary component of utterances.
Thus, we want to deal with words when processing. To this end, we *tokenise* the input – separate it into its constituent tokens. However, it is not always clear where divides should exist: should "This is my brother's cat" be tokenised as `["This", "is", "my", "brother's", "cat"]` or as `["This", "is", "my", "brother", "'", "s", "cat"]`?

Punctuation presents another challenge; for instance, disambiguation between full stops ending a sentence and a full stop in an abbreviation is nontrivial. Thus sentence segmentation is a closely related task.

The term *token* refers to a single surface form word. *cat* and *cats* are both tokens, and by convention, any punctuation mark is counted as a token. Therefore there are 8 tokens in the sentence *My cat is afraid of other cats.*

Contractions form another edge case – should they be split or retained? Although it is conventional *not* to split them, this may cause problems in processing.

The *type* of a token is a vocabulary word, or a dictionary entry. For example, *cat* and *cats* have the same type, *cat*; but *go* and *went* do not.
The set of all types is the vocabulary of a language. A high token-to-type ratio indicates that a single word can have many forms, or that it is morphologically rich.

Therefore there are a number of significant challenges that come up in tokenisation – hyphenation, numbers, dates, units, abbreviations, URLs, special tokens and punctuations.

## Applications

### Text Classification

Text classification is a problem in NLP involving finding the category to which a document must belong. Its input consists of a document $d$ and a set $C$ of classes $\{c_1, c_2, ...\}$, and its output is a single class $c \in C$.

One method is by hand-coded rules, based on combinations of words or other features. The accuracy of this method can be high, but building and maintaining the rules is expensive. Moreover, the recall is poor.
On the other hand, ML systems tend to be high-recall and low-precision. There are a number of supervised ML methods that can be applied to classification; one such is Naïve Bayes classification.

Naïve Bayes is a simple classification method, based on Bayes' Rule. It assumes that a document can be represented in a *bag of words* form – a set of the words occurring in it, along with their frequencies. Thus it abstracts away from the order of the words in the document.

Mathematically, for a document $d$ and a class $c$, we have

$$P(c \mid d) = \frac{P(d \mid c)P(c)}{P(d)},$$

by Bayes' Rule. Note that we need the prior probability $P(c)$ of the class occurring; for example, when classifying mails as spam or ham, the usual spam/ham frequency can be taken. Further, we usually ignore $P(d)$, so now we have

$$\begin{aligned} c_{\text{MAP}} &= \operatorname*{argmax}_{c \in C} P(c \mid d) \\ &= \operatorname*{argmax}_{c \in C} \frac{P(d \mid c)P(c)}{P(d)} \\ &= \operatorname*{argmax}_{c \in C} P(d \mid c)P(c). \end{aligned}$$

In this expression, $P(d \mid c)$ is called the *likelihood*, and $P(c)$ the *prior*. MAP stands for "maximum a posteriori".

This can be further developed to

$$c_{\text{MAP}} = \operatorname*{argmax}_{c \in C} P(x_1, ..., x_n \mid c)P(c),$$

where $x_i$ are the feature vectors of the words in document $d$. Assuming that these events are independent, we can reduce the expression to

$$P(c) \cdot \prod_i P(x_i \mid c).$$

## $n$-Grams

The basic idea underlying $n$-grams is to examine short sequences of words and find out how likely the sequences are. This relies on the "Markov assumption" – a word is affected only by its prior local context, or the last few words.

For example, if we know only the four sentences *The boy ate a chocolate*, *The girl bought a chocolate*, *The girl then ate a chocolate*, and *The boy bought a horse*. Suppose we want to find out how likely a sentence like *The boy bought a chocolate* is.
$n$-grams can be unigrams, bigrams, trigrams, or any other number. We attempt to predict the $n^{\text{th}}$ word, given the previous $(n-1)$ words. Mathematically,

$$\begin{aligned}\text{Unigrams} : p(w) &= \frac{c(w)}{N}\\ \text{Bigrams} : P(w_i \mid w_{i-1}) &= \frac{c(w_i, w_{i-1})}{c(w_{i-1})}\\ \text{Trigrams} : P(w_i \mid w_{i-1}, w_{i-2}) &= \frac{c(w_i, w_{i-1}, w_{i-2})}{c(w_{i-1}, w_{i-2})}\end{aligned}$$

Larger $n$-grams give more discrimination. Consider, for example, the sequences *large green [?]* and *swallowed the large green [?]* – naturally a larger $n$-gram makes a better prediction here.
However, larger $n$-grams also tend to become unreliable, as the instances in training data will be fewer. Furthermore, the number of bins (possible distinct $n$-grams) rises exponentially in $n$ for a given vocabulary size.

A major problem with this form of estimation is *sparseness* – if one of the sequences does not occur in the dataset, or occurs a small number of times, the entire probability falls to zero or a very small number. Moreover, we have not defined the model's behaviour when presented with unknown words.