Natural Language Processing

Uses for NLP:

1. Machine-Human communication (commands, instructions)
2. To learn from humans (documents, Wikipedia, Reddit)
3. Help humans:
    1. Automatic Translation/Transcription
    2. Character Recognition
    3. Text Generation/Summarization
    4. Question answering

Understandably it is one of the oldest AI goals. Early on there were developed several techniques:

- Grammar
- Special (ATN) grammars
- Parsing, Semantics, Pragmatics
- Statistical methods
- Deep Learning (Present)
  Very messy and difficult (because of ambiguity) field.
  There can be also many exceptions in language because of the different registers (formal, informal...) that humans use and some words that can have different meanings or pronunciations but are written the same (Homonym and Homograph).

Recent advances

- Web-Scale information retrieval
- Watson IBM (Question Answering)
- Siri/Cortana/Alexa
- Automatic Machine Translation
- Text-to-speech
- Large Language Models (LLMs) (e.g. ChatGPT)

NLP and Probability

First of all we write the probability that an event A happens as $P(A)$.

There is a threshold for the more probable answer (IBM Watson). The candidates (words/letters) are chose based on **probability**.

We are asking for the probability of a sentences so how can be calculate it?

- prob(sentence) = $P(\text{sentence})$= product of probabilities of its words?
- Words in a sentences are not independent, there needs to be a correlation obviously.

> A: you are in the UAB school of Engineering
> B1: you dice is 6
> B2: the AI degree is your course
> The sentences A and B2 are much more correlated with each other that the sentences A and B1. If you read A, then you could imagine that it is much more likely that the following sentences is B2 (*the AI degree is your course*)

We can see that there are more general rules also make sense expressing them in this probabilistic manner: Because of how we use grammar, if you have a verb in a sentence there is a high probability that there is a noun too (an actions and who does that action).

We can have many candidates that have very low probability but not zero, perhaps they are:

- exceptions
- a non-frequent word
- a new word

For example after the sentences *"today for dinner I'm gonna have"*, we can have many many candidates with a low probability that still make sense ("*pancakes, bananas, fruit, cake*").

If we want to make a larger text that has coherence, we need to take into account the words that came before the one that we want to predict:

> "*In the fridge I have chicken with rice, apples and tequila, therefore today for dinner I'm gonna have* [`word to predict`]"

It would make sense that the following word is *chicken* because we don't usually dinner apples or tequila. We need to take into account the context of a prediction to make it as accurate/coherent as possible. Words in a sentences or a text are not independent of each other, there needs to be some type of correlation.
This also applies to text that doesn't have so much context:

> "I got into an accident with my [`to predict`]"

Thus, it makes sense to think about text prediction as a chain of probabilities and words that depend on each other (dependent probabilities).

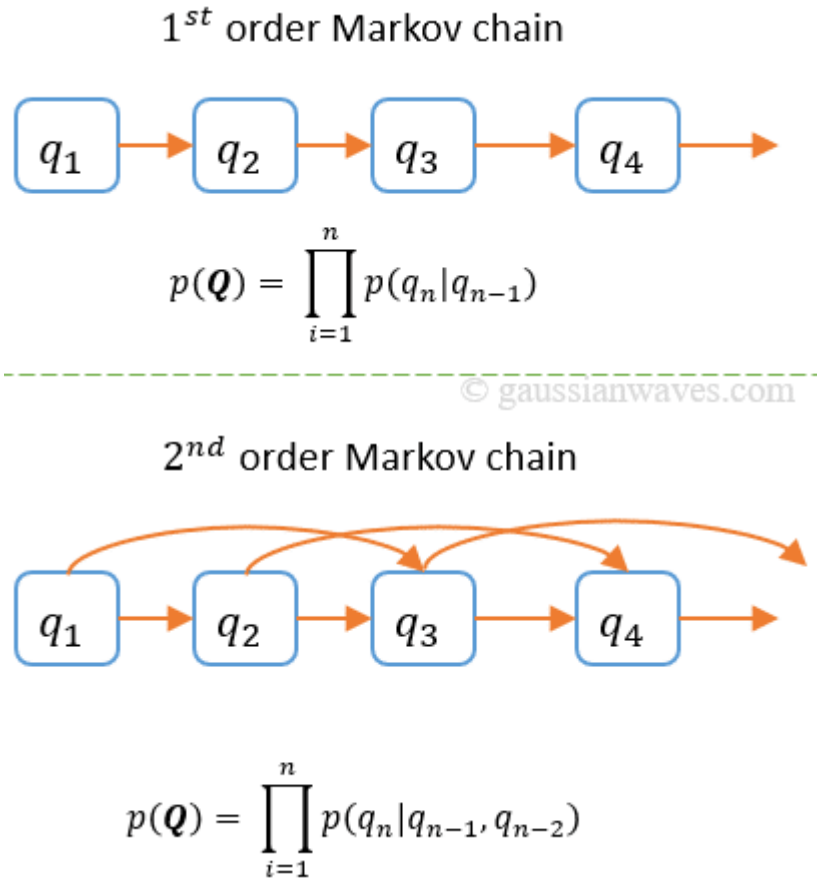
Markov Chain/Process

Assumption made in the creation of Markov chains:

> the current state depends on only a finite fixed number of previous states

This is also the core assumption made in the creation of n-gram models.

"chains" of probabilities $p(q_i)$:



First Order:

- depend on the previous state

Second Order:

- depend on the two previous states

So we can think of sentences as Markov chains of words. **Thus, the probability that a sentences appears in a text or is predicted is the product of the conditioned probabilities of its words**:

$$P(\text{sentences} = \text{product of conditioned probabilities of its words})$$

Text corpus

Definition of **Corpus**:

> Dataset of texts. They are usually digitalized very large datasets.

Example:
If we take our corpus as: *Dábale arroz a la zorra el abad*
It is composed of 25+6 characters (25 letters and 6 spaces). We can form a table to represent the frequencies of each letter:

character	frequency
d	2

character	frequency
a	7
b	2
l	3
e	2
r	4
o	2
z	2
'space'	6

n-gram models

n-gram simply means a sequence of n stuff. In our case it could be characters so a list of 1-grams can be `['a', 'b', 'z', 'i']` .
A unigram is a 1-gram, a bigram is a 2-gram, a trigram is a 3-gram and so for...

Taking our example of corpus if we group the letters in trigrams we have the following (uncompleted) list:

```
['aaa', 'aab', 'abb', 'bbb', 'abc', 'zor', 'led', 'dab', ....]
```

We can create n-grams from words also:

- unigrams: `[hola, que, tal, estoy, cansado, bulding, coffee]`
- bigram: `[hi again, im tired, drink coffee, mug building]`
- trigram: `[hi again again, im super tired, mug building helicopter]`
- pentagram?: ``[hi again and again, im super duper tired, mug building helicopter ruler]

A model of the probability distribution of n-letter sequences is called an n-gram model. An n-gram model is defined as a Markov chain of order $n-1$, the probability of character c_i depends only on the immediately preceding $n-1$ characters.

The probability of a character w given a context of history h, is expressed as such:

$$P(w|h)$$

- w is the possible prediction (candidate)
- h are the previous states or History

So the probability that we get the character `c` following the two characters `ab` is:

$$P(\mathrm{c}|\mathrm{ab})$$

This particular example can be implemented with a trigram model, because we take a possible candidate and the two preceding states.

This context/history is usually a sequence of n-grams, that we are going to express using the notation $c_{1:N}$, where N is the number of n-grams in the sequence. Thus, if we what to know the probability that the characters c_i follow the history $c_{1:i-1}$, we would write:

$$P(c_i|c_{1:i-1})$$

You can see pretty clearly how $c_{1:i-1}$ is the history because is a sequence of characters that starts at position 1 and ends at $i-1$.

The, for example in a trigram model:

$$P(c_i|c_{1:i-1}) = P(c_i|c_{i-2:i-1})$$

Because from the complete history $c_{1:i-1}$ we only take the two previous characters to our candidate $c_{i-2:i-1}$ (our candidate is c_i). Moreover, we can define the probability of a sequence of characters $P(c_{1:N})$ under a trigram model by first factoring with the chain rule (multiplication of probabilities) and using the Markov assumption:

$$P(c_{1:N}) = \prod_{i=1}^{N} P(c_i|c_{1:i-1}) = \prod_{i=1}^{N} P(c_i|c_{i-2:i-1})$$

Note that we end up with the product of the probabilities of chunks of the sentences. The chunks are defined as a character c_i and the two preceding ones $c_{i-2:i-1}$, because we are using second order Markov chain.

Probability models can be create using n-grams:

- bigram model: using 2 conditional probabilities including the prediction, so only takes into account the preceding state of the prediction: "*the following character of* `a` *is* `b`*, because given* `a`*, we have that* `b` *is our prediction with greater probability.* "
- trigram model: using 3 conditional probabilities including the prediction, so only takes into account the 2 preceding states: "*the next character of* `ab` *is* `c`"

To summarize, for example a **trigram model** can allow us to compute the conditioned probability of each character to any previous two.

Example of Trigram character model

Using the corpus from before:

> *Dábale arroz a la zorra el abad*

We can create a trigram model that uses the characters from the corpus.

The probabilities that the letter `d` follows any possible combination of two more letters (in combination forming a trigram) can be expressed as the following matrix:

d|dd	d|ad	d|bd	d|ld	d|ed	d|rd	d|od	d|zd	d|_d
d|da	d|aa	d|ba	d|la	d|ea	d|ra	d|oa	d|za	d|_a
d|db	d|ab	d|bb	d|lb	d|eb	d|rb	d|ob	d|zb	d|_b
d|dl	d|al	d|bl	d|ll	d|el	d|rl	d|ol	d|zl	d|_l
d|de	d|ae	d|be	d|le	d|ee	d|re	d|oe	d|ze	d|_e
d|dr	d|ar	d|br	d|lr	d|er	d|rr	d|or	d|zr	d|_r
d|do	d|ao	d|bo	d|lo	d|eo	d|ro	d|oo	d|zo	d|_o
d|dz	d|az	d|bz	d|lz	d|ez	d|rz	d|oz	d|zz	d|_z
d|d_	d|a_	d|b_	d|l_	d|e_	d|r_	d|o_	d|z_	d|__

The dimensions of these type of matrices for bigram models are $n \times x$ because you have to compute the probability of any character with any other characters. In the case of trigram you have to compute the probability of $P(x|yz)$, given all x, y, z, thus the matrix will have dimensions $n \times x \times x$, therefore it can be though as a tridimensional matrix.

To create a trigram model from the characters that compose the corpus (can also be done from the words), we compute a matrix of all possible characters that can follow the current state.

A trigram models predicts a new word/character using the previous two. In the example above the probability $P(\mathrm{d}|\mathrm{ba})$, is the probability that given `ba`, the character that follows is `d`. Altogether to form the word `bad` for example.

In other words, in the case of a trigram model, the probability of the trigram $P(\mathrm{d}|\mathrm{ba})$ is the probability of finding the letter `d` in position i, given that bigram `bad` appears in position $i-1$ and $i-2$, for each of the words. Using the notation that we defined before we have that `ba` $= c_{i-1:i-2}$ and `d` $= c_i$

In the case of a bigram model, the probability of the bigram `p|q` is the probability of finding the letter `q` in position i, given that the letter `p` appears in position $i-1$.

Smoothing n-gram models

Consider that we are talking about an n-gram model for words (e.g. bigram = `of the`).

There are many words that have a strong frequency in the corpus (e.g. in English `the`, or in Spanish `de`), so the probability that these words are predicted is quite high.
However, there are other n-grams that they directly do not appear on the corpus (e.g. `the elephant`) or they have very low frequency. These low-frequency n-gram are subject to random noise, that we have to control somehow.

There is also the possibility that there are n-grams that don't exist in our corpus but exist in the language. If the model assigns a probability of 0 to these n-gram the whole sentence will also have probability 0, we can solve this issue using smoothing.
We don't want some of the n-grams to have zero probability (the ones that are not in the corpus) while others have a small positive probability (the flow-frequency words), we want to apply smoothing to all the similar n-grams.

Bayes' Theorem

We were talking about the "conditioned probabilities of words", this can be expressed by the Bayes' Theorem because we consider words/characters in a text to be dependent events in relation to each other.

Independent Events

Independent events like two dice rolls have combined probability of:

$$P(A|B) = P(A)P(B)$$

Where A and B are the independent events. For example, A = dice with 6, B = dice with 6, probability of having two 6s in a row is $P(A|B) = \frac{1}{6}\frac{1}{6} = \frac{1}{12}$
The probabilities of the events are just multiplicated.

Dependent Events (Bayes' Theorem)

> Thomas Bayes (1702-1761)

Conditional probabilities for two dependent events, the theorem is as follows.
Given two dependent probabilistic events A and B, the conditional probability $P(A|B)$ is:

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

Where $P(B) = 0$, thus we need smoothing to do not have zero probability events (in our case words/characters). See the example 13.5.1 in the book **(3th edition, page 496)**.

We are going to use this rule to express the probability of characters/words in text. This is done by Markov chains as we have seen before.

Use on n-gram models

Often we computing Bayes' rule we are interested in finding the maximum value possible of a probability of a variable. Thus, we employ the argmax_V function to represent the maximum value of V that gives out the highest possible probability $P(V)$. applying Bayes' rule:

$$\frac{P(B|V)P(V)}{P(B)}$$

Since for all the possible values of V, the denominator remains constant (equal for all V), we ignore it:

$$P(B|V)P(V)$$

n-gram characters models

Application: language identification

Problem definition: using a corpus formed of trigrams of characters identify from what language a text is. We build a trigram model of each candidate language, the probability that we get the character c_i following the history $c_{i-2:i-1}$ is:

$$P(c_i|c_{i-2:i-1}, l)$$

where l is a variable that we use to specify the language. thus, for each l the model is built by counting trigrams in a corpus of that language.

What we want to know is $P(\text{Text}|\text{Language})$, we give a text and language as input

Probability that a sentence is from a language? `my tailor is rich`:

1. Multiply the probability of its characters $\rightarrow$ they are not independent
   so we multiply the conditioned probabilities using Bayes' Rule
2. Assume the probability of a character depends on the previous two (trigram model):

$$P(c_i|c_{i-2:i-1}, l)$$

3. Compute them in a language corpus
4. Our predicted language L as to come from the largest $P(l)$ that we can compute:

$$\begin{aligned} L &= \text{argmax}_l\, P(l|c_{1:N}) \\ &= \text{argmax}_l\, P(l)P(c_{1:N}|l) \quad \Leftarrow \text{by Bayes' Theorem} \\ &= \text{argmax}_l\, P(l) \prod_{i=1}^{N} P(c_i|c_{i-2:i-1}, l) \end{aligned}$$

   Note that we do not know $P(l|c_{1:N})$ (probability that we get a language l out of a history $c_{1:N}$), but we know $P(c_{1:N}|l)$, the probability that a sequence $c_{1:N}$ is in language l.
   We try this over many languages.
5. The language that you want is the one with the highest probability. The function argmax_l is there to express that we want L to the highest language that exist in all the languages l.
6. Useful to automatically identify languages in translators.

n-gram word models

Upon this point we mostly talked about n-gram characters models, now we are going to look into n -gram word models that use sequences of words (i.e. the n-grams are words).

The main difference is that the vocabulary in the corpus is much much much larger. Word models have at least tens of thousands of symbols, sometimes millions.

Because we cannot be sure that the corpus that we define the model contains all the words that we need, the out of vocabulary words become a problem.

Moreover, a "word" needs to be well defined. For example in Catalan we can have `trobeu-les` or in English `we'd` that are contractions that simply language by not having to write the pronouns in their entirety. Also, other languages do not separate words by spaces. This was not a problem in n-characters models because the space was a characters included in the model. But it is clearly a problem with n-gram word models.

To summarize:

- Much larger "vocabulary" of units
- Out of vocabulary becomes a problem
- "word" needs to be defined precisely

trigram word model example

With n-gram models we can also do text classification (categorization). For this example of text classification we are going to do a **spam detector** that classifies between two categories spam and ham (ham = not-spam = ¬spam), here are some text examples:

```
Spam: Wholesale Fashion Watches -57% today. Designer watches for cheap ...
Spam: You can buy ViagraFr$1.85 All Medications at unbeatable prices! ...
Spam: WE CAN TREAT ANYTHING YOU SUFFER FROM JUST TRUST US ...
Spam: Sta.rt earn*ing the salary yo,u d-eserve by o’btaining the prope,r crede’ntials!
Ham: The practical significance of hypertree width in identifying more ...
Ham: Abstract: We will motivate the problem of social identity clustering: ...
Ham: Good to see you my friend. Hey Peter, It was good to hear from you. ...
Ham: PDS implies convexity of the resulting optimization problem (Kernel Ridge ...
```

We want to implement a spam detection an specific language through the trigram models that gives us the probability that a message (mess) can be in either class

$$P(\text{class}|\text{mess})$$

Thus, we have to make two models, $P(\text{spam}|\text{mess})$ and $P(\text{ham}|\text{mess})$; we compute the probability and see what class has the higher one.

For the vocabulary we are going to use a long list of words and we must considers words that are unknown.

As for the n-gram character models we get that the probability of a sentence is the multiplicated probability of its elements using the Bayes' rule. We use a trigram model and therefore a second order Markov chain. The final classification is going to be:

$$\begin{aligned} C &= \text{argmax}_{\text{class}}\, P(\text{class}|\text{mess}) \\ &= \text{argmax}_{\text{class}}\, P(\text{mess}|\text{class})P(\text{class}) \end{aligned}$$

Each of these models corresponds to an array of probabilities.

Word Embeddings

These part can be found in the 4th edition book. All the other parts are in the 3th edition.

We can encode (transform) words into vectors of many dimensions via a Encoder-Decoder neural network. These NN is going to give semantic meaning to these vector. For example words that are similar (`computer, supercomputer, electronics` or `dog, cat, pet`) are going to be very close together in this vector space. Similar words are mapped into close regions of that multidimensional space.

Moreover, there is operations (like summation) that we can perform to get vectors in the word embedding that have semantic meaning. For example is we sum the words `king - man + woman` we are going to have the word `queen`.

Also we can see what relationship have the words with each other:

A	B	C	D=C+(B-A)	Relatrionship
Athens	Greece	Oslo	Norway	Capital
Astana	Kazahastan	Harare	Zimbabwe	Capital
Angola	kwanza	Iran	Rial	Currency
Copper	Cu	Gold	Au	At. Symbol
Microsoft	Windows	Google	Android	Op. System
New York	New York Times	Baltimore	Baltimore Sun	Newspaper
Berlusconi	Silvio	Obama	Barak	First name
Switzerland	Swiss	Cambodia	Cambodian	Nationality
Einstein	Scientist	Picasso	painter	Occupation

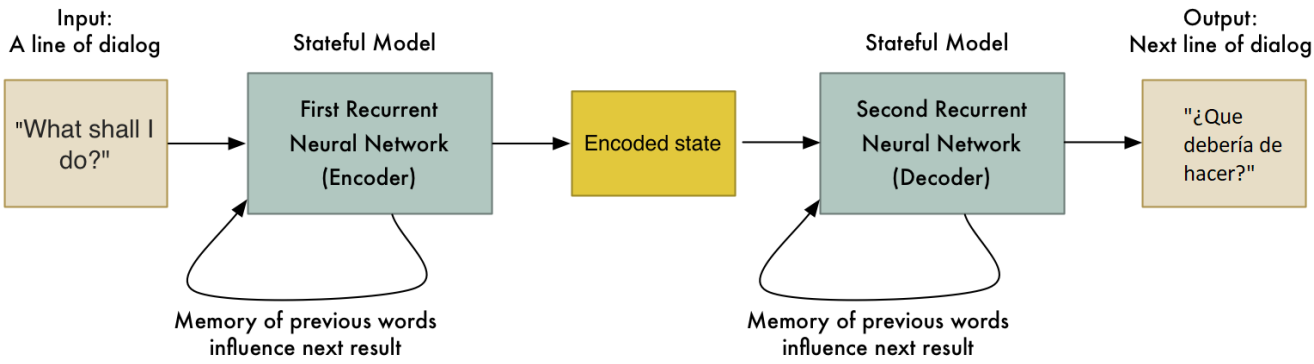
A word embedding model can sometimes answer the question "A is to B as C is to [what]?" with vector arithmetic: given the word embedding vectors for the words A, B, and C, compute the vector D=C+(B−A) and look up the word that is closest to D. (The answers in column D were computed automatically by the model. The descriptions in the "Relationship" column were added by hand.) Adapted from Mikolov et al. (2013, 2014). 36

Explore Word Embeddings in this website

Automatic Translation

On recent years n-gram models applied to language translation for example were outclassed by a type of neural networks called Recurrent Neural Networks (RNN). These NNs form part of the deep learning paradigm, they are very large and have to be trained in a large amount of data.

They work through embedding the input sentence into a vector space (like with word embeddings) and then use an attention mechanism to see it is more importante in order to translate.



The main characteristic of the RNNs is that they have a memory of the previous states and that the data input is sequential. We can see how this is perfect in order to work with language.

Despite having many similarities with n-gram models do not confuse them, because RNNs to do work by using Markov chains.

They need to be trained on a large parallel corpus, that is one large dataset of already translated sentences:

1	Buttons	Botones
2	On/Off button	Botón de encendido/apagado
3	When you're not using iPhone, you can lock it to turn off the display and save the battery.	Cuando no utilice el iPhone, puede bloquearlo para apagar la pantalla y ahorrar batería.
4	Lock iPhone: Press the On/Off button.	Bloquear el iPhone: Pulse el botón de encendido/apagado.
5	When iPhone is locked, nothing happens if you touch the screen.	Cuando el iPhone está bloqueado, no ocurre nada si toca la pantalla.
6	iPhone can still receive calls, text messages, and other updates.	El iPhone puede seguir recibiendo llamadas, mensajes de texto y otras actualizaciones.
7	You can also:	También puede:
8	Listen to music	Escuchar música
9	Adjust the volume using the buttons on the side of iPhone (or on the iPhone earphones) while you're on a phone call or listening to music	Ajustar el volumen con los botones del lateral del iPhone (o de los auriculares del iPhone) mientras habla por teléfono o escucha música
	Use the center button on iPhone earphones to answer or end a call, or to control audio	Usar el botón central de los auriculares del iPhone para responder o finalizar una llamada, o para controlar la

LLMs (ChatGPT)

The Large Language Models widely used today are based on the Transformers architecture that published by Google Brain in 2017 (Attention is all you need)

This architecture is solely based on Attention mechanisms. These mechanisms provide the model a metric/measure of what is important in the text (Attention), this gives the model memory of what is important. When reading you remember the last 3-5 words clearly, but you still remember the important words/concepts that you already read before.

Moreover, this type of models work by having and Encoder and a Decoder:

- Encoder: takes input sentence and maps it into a higher dimensional space (converts the text to a vector/token)
- Decoder: this vector is processed and the decoder turns the output vector into an output sequence (text)

Text predictor functionality:

- predict the next word (token)
- associate each candidate with a probability
- selecting the most probable does not generate good text
- random selection among the most probable
- assessed by a parameter called Temperature

This temperature parameter can think of as the creativity of the model. A low Temperature gives gives a more direct, less creative answer; whereas a high Temperature given a less expected answer that ends up being more creative.