❖ Language Model:

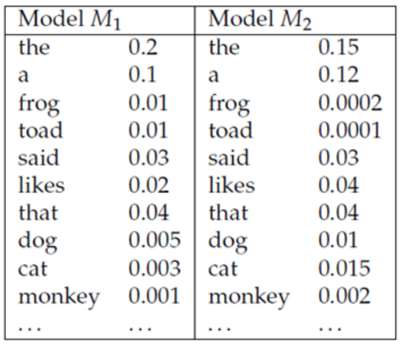
Language modeling (LM) is a fundamental task in natural language processing and is routinely employed in a wide range of applications, such as speech recognition, machine translation, etc.
It is the use of various statistical and probabilistic techniques to determine the probability of a given sequence of words occurring in a sentence. Language models analyze bodies of text data to provide a basis for their word predictions.
Traditionally, a language model is a probabilistic model which assigns a probability value to a sentence or a sequence of words. We refer to these as **generative language models**, which is actually a finite state machine. If each state in the FSM has a probability distribution on the production of different words, we call it a language model. The automaton itself has a probability distribution over the entire vocabulary of the model, summing to 1:

$$\sum_{s \,\epsilon\, \sum *} P(s) \ = 1$$

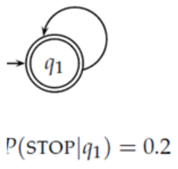
Some common statistical language modeling types are:

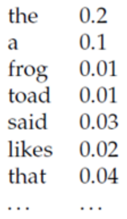
- **N-gram**. N-grams are a relatively simple approach to language models. They create a probability distribution for a sequence of n. The n can be any number, and defines the size of the "gram", or sequence of words being assigned a probability. For example, if n = 5, a gram might look like this: "can you please call me." The model then assigns probabilities using sequences of n size. Basically, n can be thought of as the amount of context the model is told to consider.

- **Unigram**. The unigram is the simplest type of language model. It doesn't look at any conditioning context in its calculations. It evaluates each word or term independently, so we only have one-state finite automata as units. It splits the probabilities of different terms in a context, e.g. from $P(t_1t_2t_3) = P(t_1)P(t_2 \mid t_1)P(t_3 \mid t_2t_1)$ to $P_{uni}(t_1t_2t_3) = P(t_1)P(t_2)P(t_3)$.
  Unigram models commonly handle language processing tasks such as information retrieval. The unigram is the foundation of a more specific model variant called the query likelihood model, which uses information retrieval to examine a pool of documents and match the most relevant one to a specific query.



Model M_1		Model M_2	
the	0.2	the	0.15
a	0.1	a	0.12
frog	0.01	frog	0.0002
toad	0.01	toad	0.0001
said	0.03	said	0.03
likes	0.02	likes	0.04
that	0.04	that	0.04
dog	0.005	dog	0.01
cat	0.003	cat	0.015
monkey	0.001	monkey	0.002
…	…	…	…





the	0.2
a	0.1
frog	0.01
toad	0.01
said	0.03
likes	0.02
that	0.04
…	…

Partial specification of two unigram language models.

Here we also need a probability of stopping after each word which is considered 0.2 in this example.

- **Bidirectional**. Unlike n-gram models, which analyze text in one direction (backwards), bidirectional models analyze text in both directions, backwards and forwards. These models can predict any word in a sentence or body of text by using every other word in the text. Examining text bidirectionally increases result accuracy. This type is often utilized in machine learning and speech generation applications.

- **Exponential**. Also known as maximum entropy models, this type is more complex than n-grams. Simply put, the model evaluates text using an equation that combines feature functions and n-grams.
  Basically, this type specifies features and parameters of the desired results, and unlike n-grams, leaves analysis parameters more ambiguous -- it doesn't specify individual gram sizes, for example.
  The model is based on the principle of entropy, which states that the probability distribution with the most entropy is the best choice. In other words, the model with the most chaos, and least room for assumptions, is the most accurate.

- **Continuous space**. This type of model represents words as a non-linear combination of weights in a neural network.
  This type becomes especially useful as data sets get increasingly large, because larger datasets often include more unique words. The presence of a lot of unique or rarely used words can cause problems for linear model like an n-gram. This is because the amount of possible word sequences increases, and the patterns that inform results become weaker.
  By weighting words in a non-linear, distributed way, this model can "learn" to approximate words and therefore not be misled by any unknown values. Its "understanding" of a given word is not as tightly tethered to the immediate surrounding words as it is in n-gram models.

Language modeling is crucial in modern natural language processing (NLP) applications. It is the reason that machines can understand qualitative information. Each language model type, in one way or another, turns qualitative information into quantitative information. This allows people to communicate with machines as they do with each other to a limited extent.

❖ **Query likelihood model:**

The query likelihood model is a language model used in information retrieval. A language model is constructed for each document in the collection. It is then possible to rank each document by the probability of specific documents given a query (rank documents by **P(d|q)**), where the probability of a document is interpreted as the likelihood that it is relevant to the query. The reason for this is the Bayes law.

$$P(d|q) = P(q|d)P(d)/P(q)$$

P(d|q) is the document's rank.
P(d) can be assumed similar for all documents, then there wouldn't be a need to handle it.
P(q) also only depends on the query.
so: $p(d|q) \approx p(q|d)$ and for calculating p(q|d) the multinomial naïve bayes model will be used.

For retrieval based on a language model (LM), we treat the generation of queries as a random process. The approach is to:

1) Infer a LM for each document.
2) Estimate **$P(q|M_{di})$**, the probability of generating the query according to each of these document models.
3) Rank the documents according to these probabilities.

In practice this language model is unknown, so it is usually approximated by considering each term (unigram) from the retrieved document together with its probability of appearance. So, **$P(t|M_d)$** is the probability of term **t** being generated by the language model **M_d** of document **d**. This probability is multiplied for all terms from query **q** to get a rank for document **d** in the interval [0,1]. The calculation is repeated for all documents to create a ranking of all documents in the document collection.

❖ Named-entity Recognition:

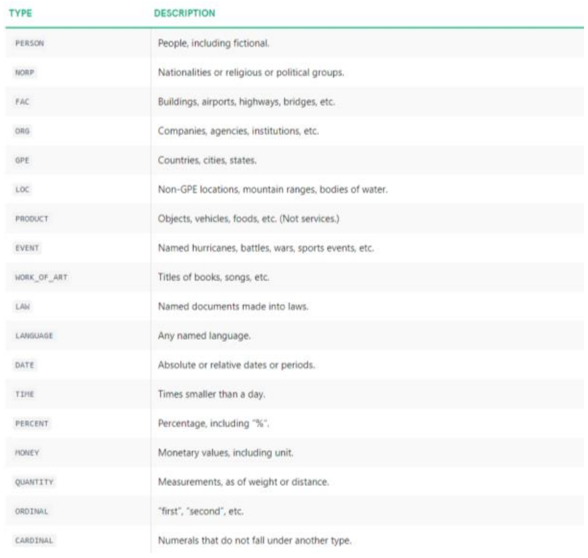
Named-entity recognition (NER) (also known as *(named) entity identification*, *entity chunking*, and *entity extraction)* is a subtask of information extraction that seeks to locate and classify named entities mentioned in unstructured text into pre-defined categories such as person names, organizations, locations, medical codes, time expressions, quantities, monetary values, percentages, etc.

Extracting the main entities in a text helps sort unstructured data and detect important information, which is crucial if you have to deal with large datasets.

Most research on NER/NEE systems has been structured as taking an unannotated block of text and producing an annotated block of text that highlights the names of entities.
Example: Jim bought 300 shares of Acme Corp. in 2006. => [Jim]$_{\text{Person}}$ bought 300 shares of [Acme Corp.]$_{\text{Organization}}$ in [2006]$_{\text{Time}}$.

Basically, NER is used for indexing these names and relating different entities. In searches, the answer is usually the name of an entity that can be easily searched by indexing the names.

- Ways of using NER:
  1. Using regular expressions
  2. Using classifiers:
     Generative: Naive Bayes
     Discriminative: MaxEnt algorithm models
  3. Sequence Modeling:
     Hidden Markov Models (HMM)
     Conditional Random Fields (CRF)
     Conditional Markov model (CMM) / Maximum-entropy Markov model (MEMM)

- Open-Source named entity recognition APIs for python:
  - **SpaCy**: a Python framework known for being fast and very easy to use. It has an excellent statistical system that you can use to build customized NER extractors.
    SpaCy's named entity recognition has been trained on the OntoNotes 5 corpus and it supports the following entity types:
    

TYPE	DESCRIPTION
PERSON	People, including fictional.
NORP	Nationalities or religious or political groups.
FAC	Buildings, airports, highways, bridges, etc.
ORG	Companies, agencies, institutions, etc.
GPE	Countries, cities, states.
LOC	Non-GPE locations, mountain ranges, bodies of water.
PRODUCT	Objects, vehicles, foods, etc. (Not services.)
EVENT	Named hurricanes, battles, wars, sports events, etc.
WORK_OF_ART	Titles of books, songs, etc.
LAW	Named documents made into laws.
LANGUAGE	Any named language.
DATE	Absolute or relative dates or periods.
TIME	Times smaller than a day.
PERCENT	Percentage, including "%".
MONEY	Monetary values, including unit.
QUANTITY	Measurements, as of weight or distance.
ORDINAL	"first", "second", etc.
CARDINAL	Numerals that do not fall under another type.

    More info: https://spacy.io/api/annotation#section-named-entities

  - **Natural Language Toolkit (NLTK)**: this suite of libraries for Python is widely used for NLP tasks. NLKT has its own classifier to recognize named entities called ne_chunk, but also provides a wrapper to use the Stanford NER tagger in Python.
    NLTK is not as easy as SpaCy to use and in the example used in https://towardsdatascience.com/named-entity-recognition-with-nltk-and-spacy-8c4a7d88e7da we see that Google is recognized as a person which is quite disappointing.
    More info: https://www.nltk.org/book/ch07.html