Approach

Question 1: The approach for this question was to first create the matrix and then calculate the m rows and n columns. After calculating these values then create the iteration for rows and columns. Then we need to consider the logic to traverse to the index in the matrix that needs to be updated. The index which needs to be updated are:

i=0 and j=0

i=2 and j=2

i=3 and j=4

when the iterator reaches these matrices then change the values if the indices according to the requirements. This will be wrapped as a python function.

Pseudocode

Question 2.

The approach for this question is to create a function which takes in as input a file path and filename of the resultant csv file. The function shall have two parts. First part would be to generate all the files under the given path along with its attributes into a list. This lost shall include (filename, filetype, file size (MB/KB/etc.), Creation Date, Last Modified Date etc. Then I would create a new file, open a csv file into a write mode and write the data from the above array per line into the opened csv.

Pseudocode

Input: path to the directory, name of the file which would form the output.

Output: The resultant csv which shall have line by line all the files in the folder

start

Function\_to \_create metadata()

create array as null

iterate on the path to directory and extract all the contents

check if the file is a directory or not

if not directory print and store in the array

initiate variables

create a new file,

open csv to write mode,

because we're creating a file object we open with newline=''

iterate on the array created above

put the metedata per line

end

Question 3

The solution is quite straightforward. Steps are as follows:

Unpack the archive, and then run the Netlogo binary executable from a terminal in the installation directory. I would first download the latest NetLogo files and have extracted them.

Then I would I place the files in the /opt/netlogo-5.3.0/ directory. I then proceeded to create a symbolic link to the NetLogo executable from the /usr/bin directory.

I would do so as follows:

wget http://ccl.northwestern.edu/netlogo/6.0.2/NetLogo-6.0.2-64.tgz

tar -xzf NetLogo-6.0.2-64.tgz

I do believe I would need java 7 installed as well if the machine does not have java installed.

The script would be as follows, after I download the package and unzip the files. I would check if all pre-requisite commands are available and then install netlogo

#!/bin/bash

set -eu -o pipefail # fail on error , debug all lines

sudo -n true

test $? -eq 0 || exit 1 "you should have sudo priveledge to run this script"

echo installing the must-have pre-requisites

while read -r p ; do sudo apt-get install -y $p ; done < <(cat << "EOF"

perl

zip unzip

exuberant-ctags

mutt

libxml-atom-perl

postgresql-9.6

libdbd-pgsql

curl

wget

libwww-curl-perl

EOF

)

echo installing the nice-to-have pre-requisites

echo you have 5 seconds to proceed ...

echo or

echo hit Ctrl+C to quit

echo -e "\n"

sleep 6

sudo yum install netlogo

Question 4

Natural Language Generation, as defined by[Artificial Intelligence: Natural Language Processing Fundamentals](http://www.techprenuer.com/entrepreneur/artificial-intelligence-natural-language-processing-fundamentals/), is the “process of producing meaningful phrases and sentences in the form of natural language.” In its essence, it automatically generates narratives that describe, summarize or explain input structured data in a human-like manner at the speed of thousands of pages per second.

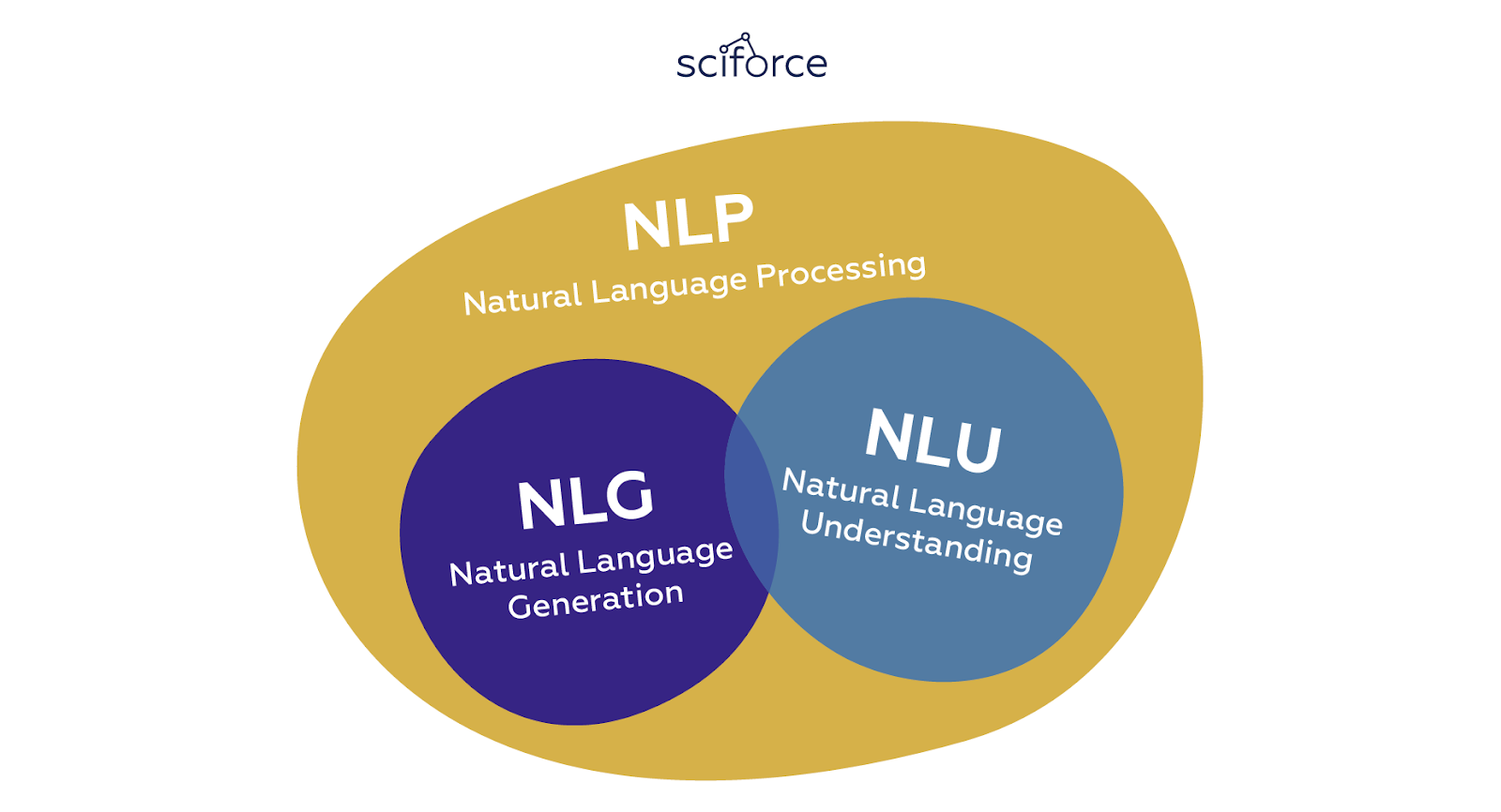
However, while NLG software can write, it is not able to read. The part of NLP that reads human language and turns its unstructured data into structured data understandable to computers is called Natural Language Understanding.

In general terms, NLG (Natural Language Generation) and NLU (Natural Language Understanding) are subsections of a more general NLP domain that encompasses all software which interprets or produces human language, in either spoken or written form:

NLU takes up the understanding of the data based on grammar, the context in which it was said and decide on intent and entities.

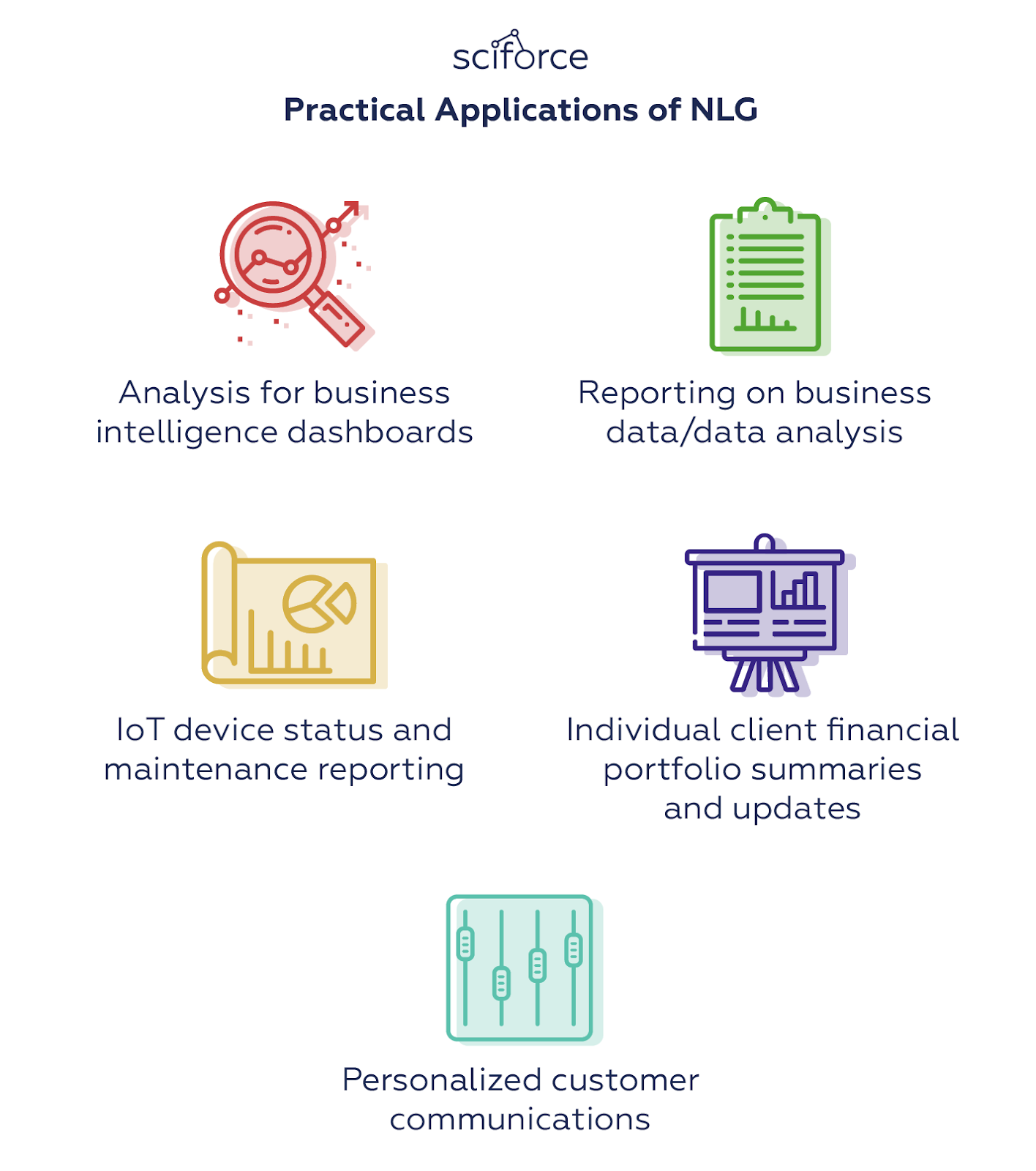
NLP converts a text into structured data.

NLG generates a text based on structured data.



Major applications of NLG

NLG makes data universally understandable making the writing of data-driven financial reports, product descriptions, meeting memos, and more much easier and faster. Ideally, it can take the burden of summarizing the data from analysts to automatically write reports that would be tailored to the audience. The main practical present-day applications of NLG are, therefore, connected with writing analysis or communicating necessary information to customers:



Practical Applications of NLG

At the same time, NLG has more theoretical applications that make it a valuable tool not only in Computer Science and Engineering, but also in Cognitive Science and Psycholinguistics. These include:



The purpose behind such a domain is to identify how to make computers understand and make sense out of commonly spoken human language apart from the usual and rationally defined set of instructions.

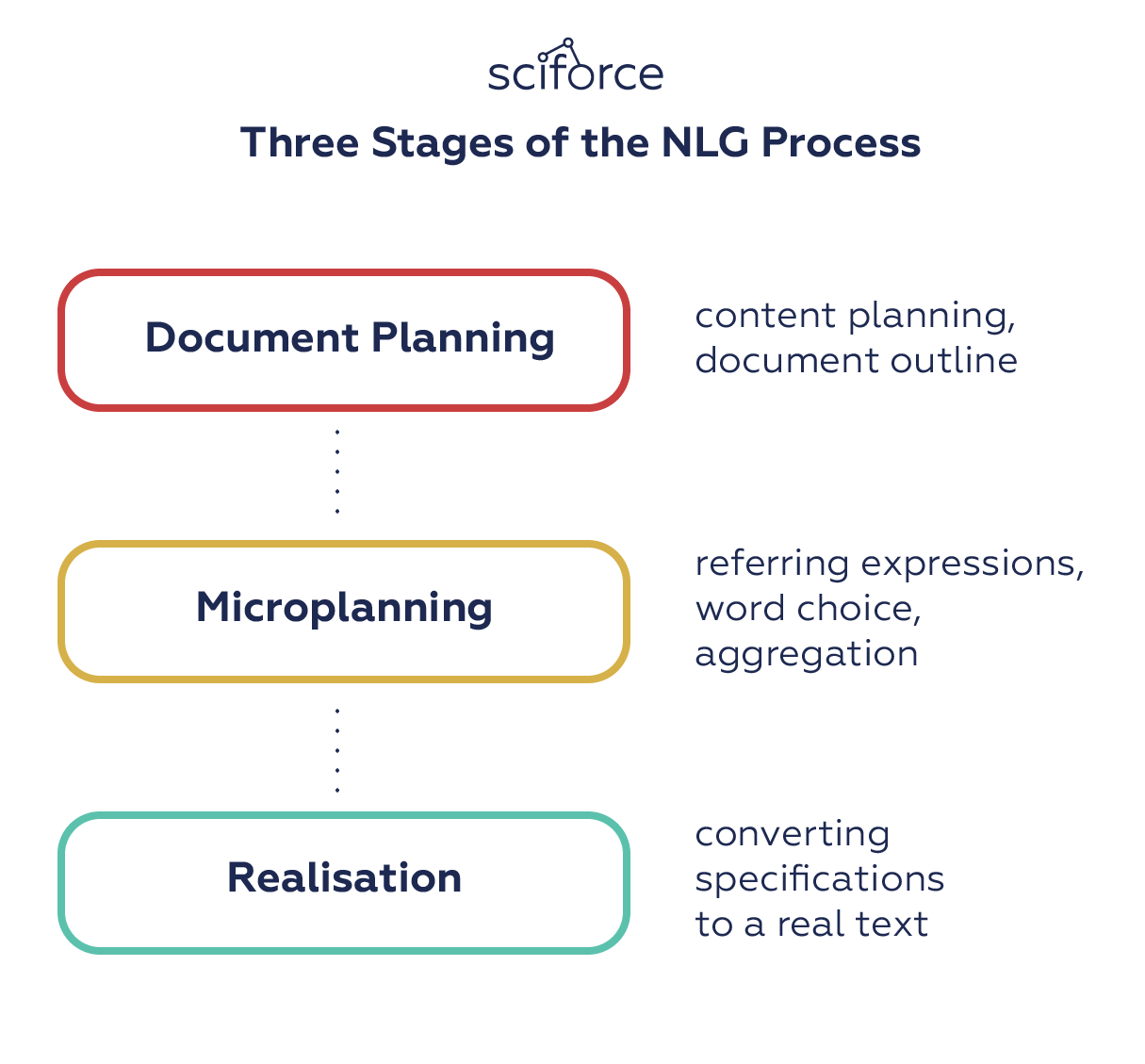
Being a part of AI, NLP relies heavily upon Machine Learning and the process of NLP goes something like this:

● Capturing the human input which ranges from either text input or voice input

● Conversion of voice data to text

● Processing the text using the grammar, parsing techniques, semantics and similar methods to identify a meaning of data

● Relaying the processed output to the human by either displaying it on screen or playing it via audio



Three Stages of the NLG Process

This pipeline shows the milestones of natural language generation, however, specific steps and approaches, as well as the models used, can vary significantly with the technology development.

There are two major approaches to language generation: using templates and dynamic creation of documents. While only the latter is considered to be “real” NLG, there was a long and multistage way from basic straightforward templates to the state-of-the-art and each new approach expanded functionality and added linguistic capacities:

Simple Gap-Filling Approach

One of the oldest approaches is a simple fill-in-the-gap template system. In texts that have a predefined structure and need just a small amount of data to be filled in, this approach can automatically fill in such gaps with data retrieved from a spreadsheet row, database table entry, etc. In principle, you can vary certain aspects of the text: for example, you can decide whether to spell numbers or leave them as is, this approach is quite limited in its use and is not considered to be “real” NLG.

Word-Level Grammatical Functions

A logical development of template-based systems was adding word-level grammatical functions to deal with morphology, morphophonology, and orthography as well as to handle possible exceptions. These functions made it easier to generate grammatically correct texts and to write complex template systems.

Dynamic Sentence Generation

Finally taking a step from template-based approaches to dynamic NLG, this approach dynamically creates sentences from representations of the meaning to be conveyed by the sentence and/or its desired linguistic structure. Dynamic creation means that the system can do sensible things in unusual cases, without needing the developer to explicitly write code for every boundary case. It also allows the system to linguistically “optimise” sentences in a number of ways, including reference, aggregation, ordering, and connectives.

Dynamic Document Creation

While dynamic sentence generation works at a certain “micro-level”, the “macro-writing” task produces a document which is relevant and useful to its readers, and also well-structured as a narrative. How it is done depends on the goal of the text. For example, a piece of persuasive writing may be based on models of argumentation and behavior change to mimic human rhetoric; and a text that summarizes data for business intelligence may be based on an analysis of key factors that influence the decision.

NLG Models

Even after NLG shifted from templates to dynamic generation of sentences, it took the technology years of experimenting to achieve satisfactory results. As a part of NLP and, more generally, AI, natural language generation relies on a number of algorithms that address certain problems of creating human-like texts:

Markov chain

The Markov chain was one of the first algorithms used for language generation. This model predicts the next word in the sentence by using the current word and considering the relationship between each unique word to calculate the probability of the next word. In fact, you have seen them a lot in earlier versions of the smartphone keyboard where they were used to generate suggestions for the next word in the sentence.

Recurrent neural network (RNN)

Neural networks are models that try to mimic the operation of the human brain. RNNs pass each item of the sequence through a feedforward network and use the output of the model as input to the next item in the sequence, allowing the information in the previous step to be stored. In each iteration, the model stores the previous words encountered in its memory and calculates the probability of the next word. For each word in the dictionary, the model assigns a probability based on the previous word, selects the word with the highest probability and stores it in memory. RNN’s “memory” makes this model ideal for language generation because it can remember the background of the conversation at any time. However, as the length of the sequence increases, RNNs cannot store words that were encountered remotely in the sentence and makes predictions based on only the most recent word. Due to this limitation, RNNs are unable to produce coherent long sentences.

LSTM

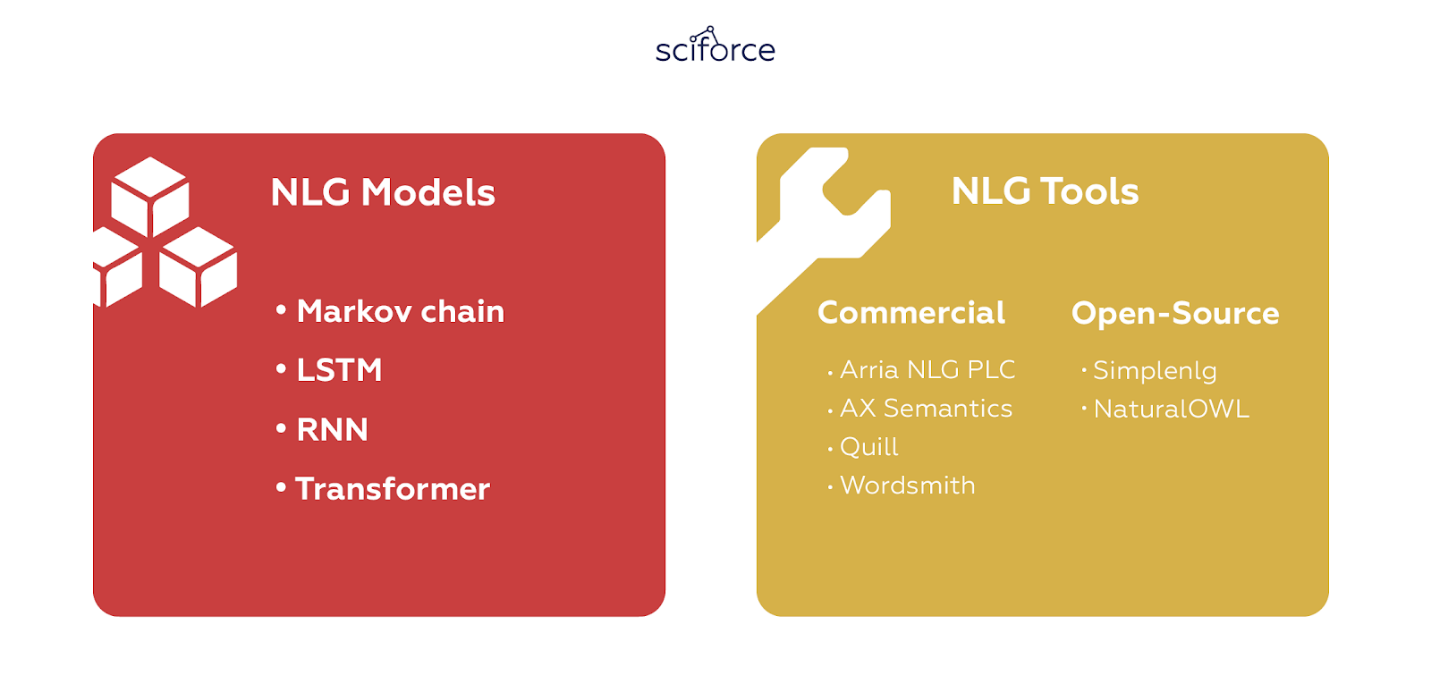
To address the problem of long-range dependencies, a variant of RNN called Long short-term memory (LSTM) was introduced. Though similar to RNN, LSTM models include a four-layer neural network. The LSTM consists of four parts: the unit, the input door, the output door and the forgotten door. These allow the RNN to remember or forget words at any time interval by adjusting the information flow of the unit. When a period is encountered, the Forgotten Gate recognizes that the context of the sentence may change and can ignore the current unit state information. This allows the network to selectively track only relevant information while also minimizing the disappearing gradient problem, which allows the model to remember information over a longer period of time.

Still, the capacity of the LSTM memory is limited to a few hundred words due to their inherently complex sequential paths from the previous unit to the current unit. The same complexity results in high computational requirements that make LSTM difficult to train or parallelize.

Transformer

A relatively new model was first introduced in the 2017 Google paper [“Attention is all you need”](https://papers.nips.cc/paper/7181-attention-is-all-you-need.pdf), which proposes a new method called “self-attention mechanism.” The Transformer consists of a stack of encoders for processing inputs of any length and another set of decoders to output the generated sentences. In contrast to LSTM, the Transformer performs only a small, constant number of steps, while applying a self-attention mechanism that directly simulates the relationship between all words in a sentence. Unlike previous models, the Transformer uses the representation of all words in context without having to compress all the information into a single fixed-length representation that allows the system to handle longer sentences without the skyrocketing of computational requirements.

One of the most famous examples of the Transformer for language generation is OpenAI, their GPT-2 language model. The model learns to predict the next word in a sentence by focusing on words that were previously seen in the model and related to predicting the next word. A more recent upgrade by Google, the Transformers two-way encoder representation (BERT) provides the most advanced results for various NLP tasks.



NLG Tools

You can see that natural language generation is a complicated task that needs to take into account multiple aspects of language, including its structure, grammar, word usage and perception. Luckily, you probably won’t build the whole NLG system from scratch as the market offers multiple ready-to-use tools, both commercial and open-source.

Python libraries for Natural Language Processing

Natural Language Processing is considered one of the many critical aspects of making intelligent systems. By training your solution with data gathered from the real-world, you can make it faster and more relevant to users, generating crucial insight about your customer base.

In this article, we will be taking a look at how Python offers some of the most useful and powerful libraries for leveraging the power of Natural Language Processing into your project and where exactly do they fit in.

● Spacy

Open-source library for industrial-strength Natural Language Processing in Python.

Often recognized as a professional-grade [Python library](https://towardsdatascience.com/best-python-libraries-for-machine-learning-and-deep-learning-b0bd40c7e8c) for advanced Natural Language Processing, [spaCy](https://spacy.io/) excels at working with incredibly large-scale information extraction tasks.

Built using Python and Cython, spaCy combines the best of both languages, the convenience from Python and the speed from Cython to deliver one of the best-in-class NLP experiences. spaCy is a no-frills library that is designed to get things done by minimizing time spent on mundane tasks. Key takeaways from spaCy are:

● Part-of-Speech Tagging

● Tokenization

● Dependency Parsing

● Sentence Segmentation

● Entity & Sentence Recognition

● Seamless integration with Deep Learning

● Methods for cleaning and normalising text

Resources -

[spaCy Documentation](https://spacy.io/docs/) — Official documentation and quickstart guide.

[Intro to NLP with SpaCy](https://nicschrading.com/project/Intro-to-NLP-with-spaCy/) — Short tutorial showcasing spaCy’s functionality.

[spaCy 101: Everything you need to know · spaCy Usage Documentation](https://spacy.io/usage/spacy-101/" \t "_blank)

[The most important concepts, explained in simple terms Whether you’re new to spaCy, or just want to brush up on some…](https://spacy.io/usage/spacy-101/" \t "_blank)

[spacy.io](https://spacy.io/usage/spacy-101/" \t "_blank)

● CoreNLP

[Stanford CoreNLP](https://stanfordnlp.github.io/CoreNLP/) is a suite of tools built for implementing a Natural Language Processing into your project. Originally written in Java, CoreNLP works with other languages such as Python, JavaScript and many more.

It offers several linguistic features such as indicating sentiment, extracting relations between entities, mark up sentence structures, provide the base form of words and more.

In fancier terms, these features translate to Lemmatization, Part-of-Speech and Morphological Tagging, Named Entity Recognition, Tokenization and Sentence Split with more following.

It’s a perfect choice if you’re looking for an NLP toolkit that is modern and robust with a wide variety of grammatical analysis tools that are regularly updated and offer high-quality analytics.

The flexible nature of CoreNLP allows it to integrate well with other languages, making it a smooth extensible and functional NLP option for your needs.

Resources-

[CoreNLP Documentation](http://stanfordnlp.github.io/CoreNLP/index.html) — Official documentation and resource compilation.

[List of Python wrappers for CoreNLP](http://stanfordnlp.github.io/CoreNLP/other-languages.html#python) — Kept up-to-date by Stanford NLP.

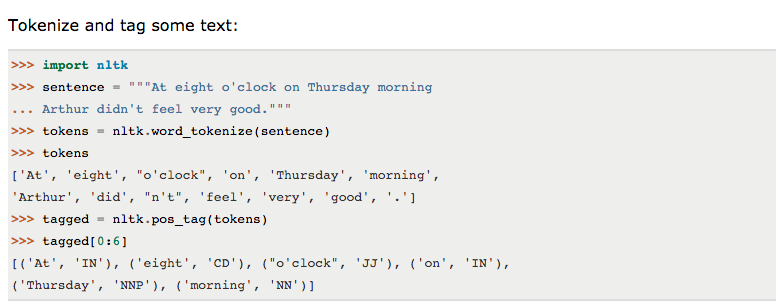
● NLTK — The most widely-mentioned NLP library

Short for [Natural Language ToolKit](http://www.nltk.org/), NLTK is the leading and one of the best Natural Language Processing libraries for Python. It has over 100 corpora and related lexical resources, such as WordNet, Web Text Corpus, NPS Chat, SemCor, FrameNet and many more.

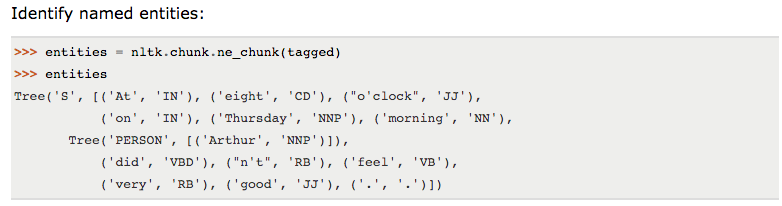
NLTK’s goal is to make learning and working with computational linguistics easier by offering features such as classification, stemming, tagging, parsing, classification semantic reasoning and wrappers for NLP libraries.

It’s a free and open-source library that is available on Windows, Mac OS and Linux with plenty of tutorials to make your entry into the world of NLP smooth.

Things you can do with NLTK —



NLTK — Tokenize and tag some text([source](http://www.nltk.org/))



NLTK — Identify Named Entities([source](http://www.nltk.org/))

Resources —

[NLTK Book](http://www.nltk.org/book/) — Complete course on Natural Language Processing in Python with NLTK.

[Dive into NLTK](http://textminingonline.com/dive-into-nltk-part-i-getting-started-with-nltk) — Detailed 8-part tutorial on using NLTK for text processing.

● Gensim

[Gensim](https://radimrehurek.com/gensim/) is a library for Topic Modelling, Similarity Retrieval and Natural Language Processing written in Python.

Developed by Radim Řehůřek in 2009, Gensim aims to excel at two things, one being the processing of natural language and the other being information retrieval. It works on vast collections of data from specific categories and provides remarkably precise insight from it.

Gensim offers memory-independent implementation capabilities for several popular algorithms such as Latent Dirichlet Allocation (LDA), Random Projections (RP), Latent Semantic Analysis (LSA/LSI/LVD) and Hierarchical Dirichlet Process (HDP), to name a few.

Gensim comes with smooth API support making integration with other programming languages a breeze while the extensive documentation and tutorials guide you in your programming journey. A pre-requisite to using Gensim is the NumPy and Scipy packages for getting it to work.

Resources-

[gensim Documentation](https://radimrehurek.com/gensim/) — Official documentation and tutorials. The tutorials page is very helpful.

● PyNLPI

Extensive functionality regarding FoLiA XML and many other common NLP format (CQL, Giza, Moses, ARPA, Timbl, etc.).

Pronounced as “[Pineapple](https://github.com/proycon/pynlpl)”, it is an open-source Natural Language Processing library for Python. PyNLPl is ideal for a variety of tasks ranging from building simplistic models and extraction of n-grams and frequency lists, with support for complex data types and algorithms.

It also comes equipped with support for standard NLP file formats such as Moses, Timbl, Giza and a handful more. PyNLPl’s library consists of several packages that go into the specifics of what it has to offer.

Mentioned below are a few of these packages:

● pynlpl.datatypes adds additional data types

● pynlpl.formats.giza for reading data from GIZA++ word alignment

● pynlpl.formats.tmbl allows reading Timbl data

● pynlpl.textprocessors as a simple tokeniser and n-gram extraction

● pynlpl.formats.cgn for parsing CGN part-of-speech tags

● Pattern

Web (data) mining / crawling and common NLP tasks.

Pattern is primarily a web mining module for Python with the inclusion of tools for a host of purposes such as Data Mining, Natural Language Processing, Machine Learning and Network Analysis.

Focussing on the NLP aspect of Pattern, it is well equipped with the essential features of any NLP such as Part-of-Speech Taggers, n-grams, Sentiment Analysis, WordNet, Text Classification and Tokenization to name a few.

It comes fully documented with more than 350 unit tests and above 50 examples to get you started right away with it. Support for web APIs enables effortless integration with other programming languages for extending Pattern’s functionality.

Google Trend — Pattern Interest Over Time



Google Trends — Pattern([source](https://trends.google.com/trends/explore?date=today%205-y&geo=US&q=TextBlob))

● Polyglot

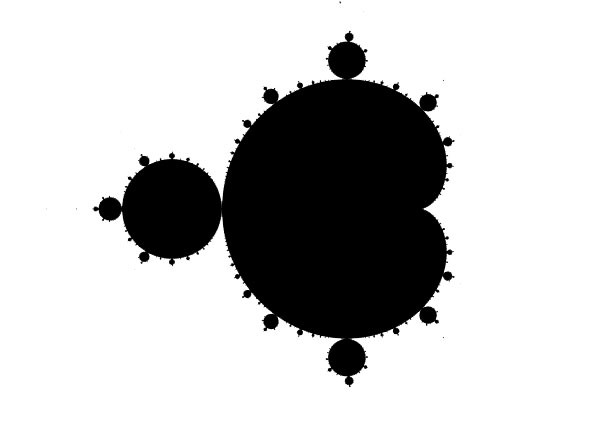
Multilingualism and transliteration capabilities.

Developed by Rami Al-Rfou, Polyglot is a Python Natural Language Processing library that is perfect for applications that must deal with an extensive collection of languages.

With support for several of the traditional Natural Language Processing features such as tokenization, language detection, part-of-speech tagging and the rest, each of its NLP features is further boosted by support for multiple languages.

The support for multiple languages making it a viable option where localization plays a crucial role. It comes with detailed documentation, simplifying the entry process for anyone.

● TextBlob



Textblob ([source](https://textblob.readthedocs.io/en/dev/))

[Textblob](https://textblob.readthedocs.io/en/dev/) is another open-source Python library for processing text-based data and offers smooth integration with other programming languages via its APIs. Textblob can be your savior if you’re in a hurry to perform standard NLP operations. It provides several of the features standard to any Natural Language Processing library such as:

● Part-of-Speech Tagging

● Sentiment Analysis

● Classification

● Tokenization

● n-grams

● Word Inflection

● WordNet Integration

● Language translation and detection powered by Google Translate

● Word and phrase frequencies

● Parsing

● Spelling correction

● Add new models or languages through extensions