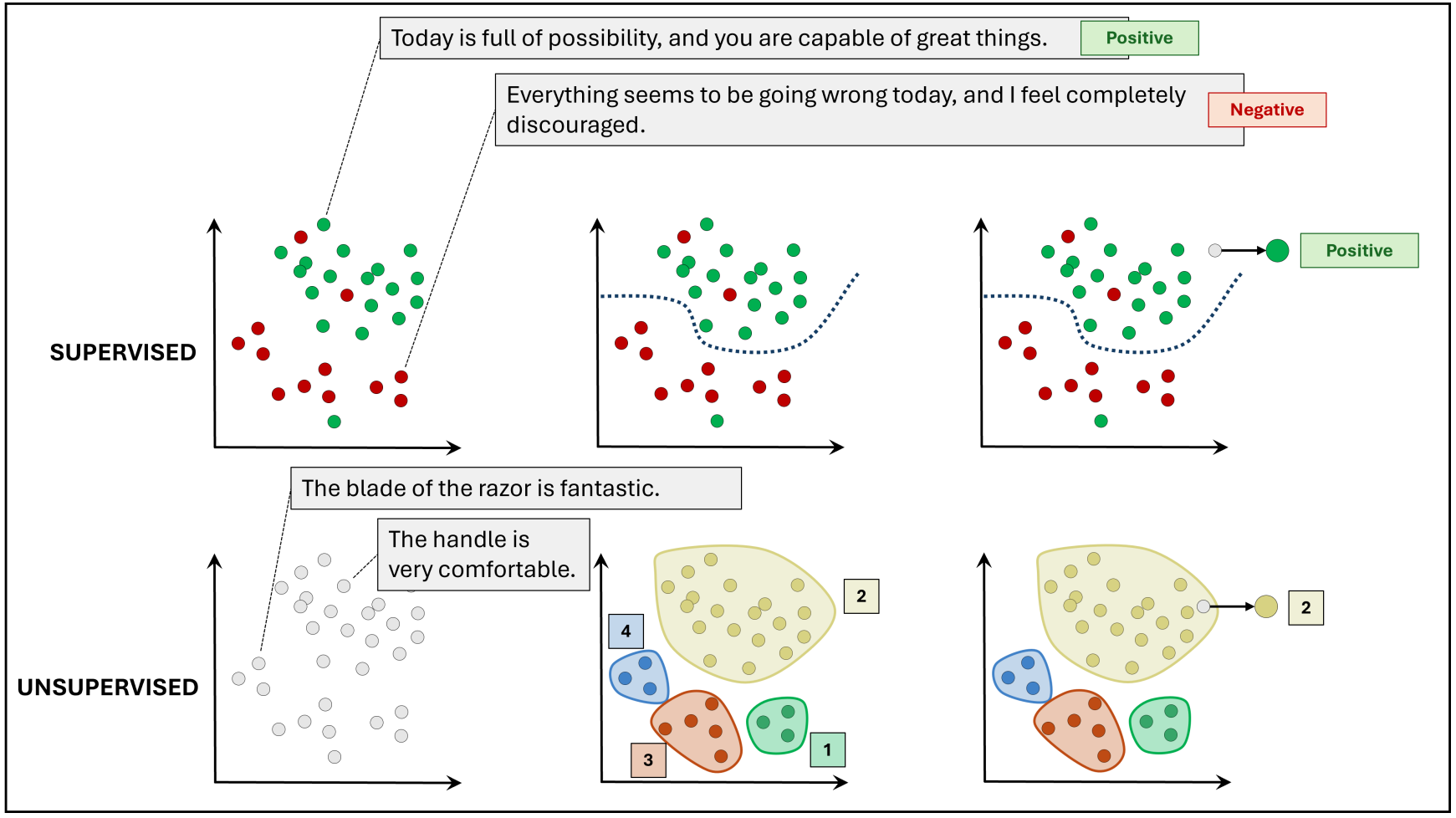
Text Modelling

Filippo Chiarello & Vito Giordano

Natural Language Processing Tasks

- **Supervised tasks**: A supervised NLP task is a task in which each text input is associated with a *ground truth*, that is, an expected output, reference label, or human annotation. The objective is to produce outputs consistent with the available ground truth for new inputs.
- **Unsupervised tasks**: An unsupervised NLP task is a task in which text data are available *without ground truth*, meaning that no expected output or reference annotation is provided for each instance. The objective is to identify patterns, structures, groups, or regularities in the text.

Natural Language Processing Tasks



Natural Language Processing Tasks

Supervised Tasks:

- Document Classification
- Sentence Classification
- Named Entity Recognition
- Relation Extraction
- Sentiment Analysis

Unsupervised Tasks:

- Topic Modelling
- Clustering
- Text Similarity

Dataset

First we have to select a dataset.

We selected a **dataset of tweets** from Kaggle that captures users' perceptions of ChatGPT.

📦 Source:
The ChatGPT Tweets Dataset

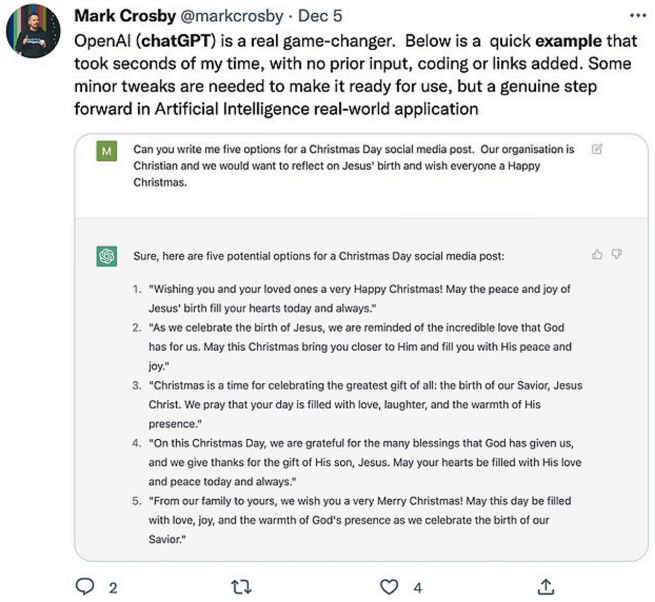
For our analysis, we selected a sample of **1,000 tweets**.

Dataset

This dataset contains the tweets which talks about how ChatGPT can be used for different tasks.

QUERY: keywords ***chatgpt*** or ***gpt***

The collection through Twitter API brought to the generation of a dataset containing **3 mln** tweets.



Dataset

We can see here the method applied for extracting the ChatGPT tasks.

Twitter Data Collection

Collect tweets talking about ChatGPT

Natural Language Processing

Use NLP to process the tweets collected

Task Extraction

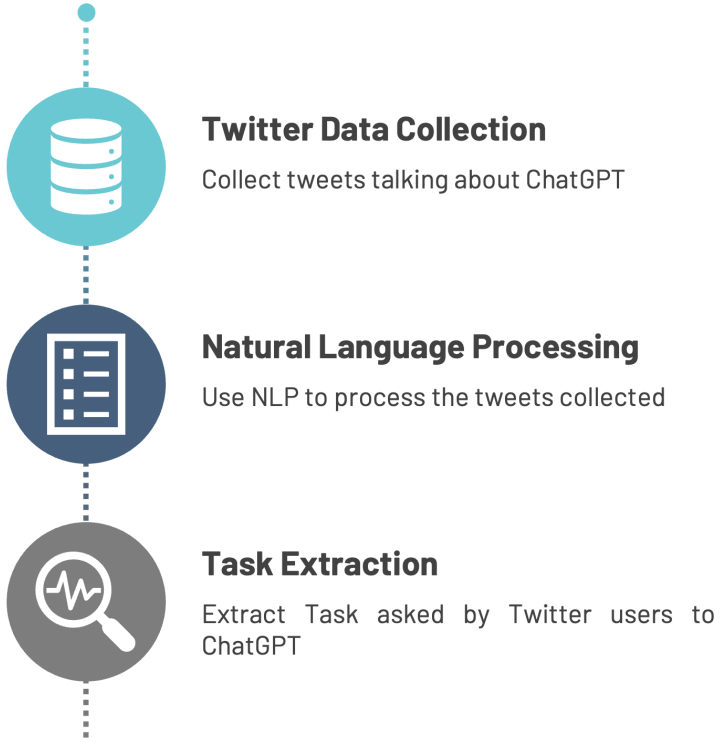
Extract Task asked by Twitter users to ChatGPT

Technology	Connection Element	Action Verb	Object
gpt, (gpt), #gpt, (#gpt), gpt3, (gpt3), #gpt3, (#gpt3), gpt-3, (gpt-3), gpt 3, (gpt 3), #gpt-3, (#gpt-3), chatgpt …	to, explains, can, could, about, for, may, might, knows how, knows how to, how to, can beused in, can be can support in, can support, can support at, how to help in, how to help for, how to help at, how to support in, how to support for	Optional Adverb + Verb + Optional word + Optional Article + Optional Possessive Adjective + Optional Adjective + Optional Compound	Direct Object (**excluding**: me, you, it, him, her, us, them, this, that, those, these)

I **asked chatgpt to write a code for website**, it's amazing. ChatGPT is the perfect tool to help you get creative ideas and automate conversations quickly and accurately. #ChatGPT #conversationalAI #chatbots

Dataset

And also here some results of the tasks extraction.



Top-20 tasks for number of tweets.

Task	# of tweets	Task	# of tweets
Write a poem	887	Gives answers	269
Write code	775	Write a script	234
Write an essay	617	Write tool	225
Write a song	509	Passed a wharton mba exam	220
Answer questions	477	Affect web3 space	212
Write story	463	Do work	196
Write article	332	Generate content	184
Write a tweet	326	Mean for the future	174
Generate code	301	Gave the correct answer	161
Generate text	296	Write emails	159

9,558 **tweets** have at least one task.

5,554 different **tasks** are collected.

Dataset

We now import the dataset.

```
#{r}
# Import packages
library(tidyverse)

# Import tweet
tweet_data <- read_csv("data/sample_tweet.csv")
```

```
## # A tibble: 4 × 16
##   tweet_id original_text       sentiment tasks users technologies
##      <dbl> <chr>               <chr>     <chr> <chr> <chr>
## 1  1.60e18 "[GPT-3] This post… neutral   ensu… <NA>  artificial …
## 2  1.60e18 "Today is the pre-… neutral   unde… <NA>  database
## 3  1.60e18 "@DeclanMccreary T… negative  inve… acad… model
## 4  1.60e18 "Screenshot of a p… neutral   wrot… <NA>  model
## # ℹ 10 more variables: organizations <chr>,
## #   job_competencies <chr>, job_profiles <chr>, nouns <chr>,
## #   verbs <chr>, adjectives <chr>, subjects <chr>,
## #   objects <chr>, predicates <chr>, topic <chr>
```

Dataset

Let's select only useful information.

```
#{r}
# Select only useful information
tweet_data <- tibble(tweet_data) |>
  select(tweet_id, original_text, tasks)
```

```
## # A tibble: 6 × 3
##   tweet_id original_text                                    tasks
##      <dbl> <chr>                                            <chr>
## 1  1.60e18 "[GPT-3] This post discusses the need for an au… ensu…
## 2  1.60e18 "Today is the pre-fine-tuning day! \r\n\r\nI'm … unde…
## 3  1.60e18 "@DeclanMccreary They don't program it to inven… inve…
## 4  1.60e18 "Screenshot of a project I'm working on using t… wrot…
## 5  1.60e18 "@Vkumzy @MsRabbitKick @rippleitinNZ It's OpenA… writ…
## 6  1.60e18 "@Andrew___Baker When I tried it, GPT-3 nailed … nail…
```

Text Classification

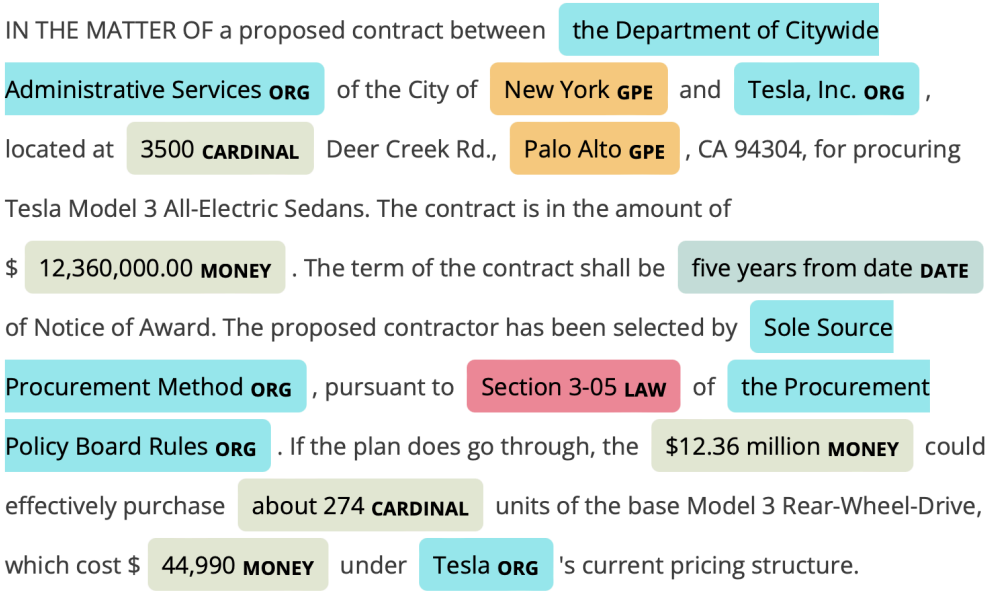
Text Classification - What is it?

Text classification is the task of assigning a text an appropriate category based on its content. Examples:

- Is this email spam?
- Sentiment analysis
- Tagging documents with categories
- Customer segmentation
- Detecting hate speech

What is Text Classification? Text Classification is the **process of assigning labels or metadata to words, phrases, sentences, documents or other text elements** to categorize or mark them based on specific attributes.

- **Manual Classification**: involves a **person assigning tags directly to content**. It is accurate and consistent but can be time-consuming and may vary based on the individual doing the classification.
- **Automatic Classification**: uses **algorithms to assign tags without human input**. It is fast and scalable, especially for large datasets, but may be less accurate with complex or ambiguous data.
- **Semi-automatic Classification**: a **system suggests tags, while a human reviews and adjusts them as necessary**. This approach accelerates the process while maintaining higher accuracy than fully automated methods, though it still involves some manual effort.



Types of text Classification: Documment & Sentence classification

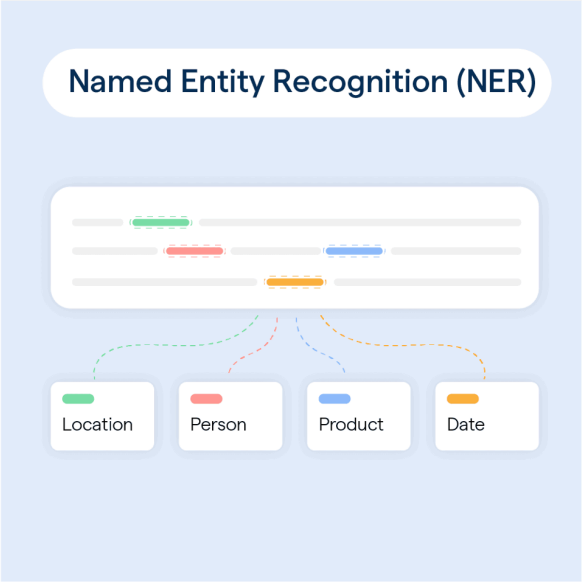
Documment & Sentence classification: This type of annotation involves **assigning one or more predefined categories to an entire text or document**. The goal is to classify the text based on its overall content, such as identifying sentiment (positive/negative). It focuses on the broader meaning of the text without analyzing specific parts.

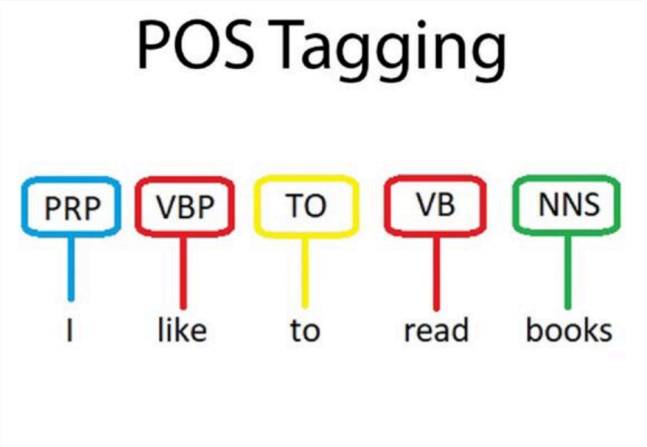
Training Examples	Labels
Simply loved it	Positive
Most disgusting food I have ever had	Negative
Stay away, very disgusting food!	Negative
Menu is absolutely perfect, loved it!	Positive
A really good value for money	Positive
This is a very good restaurant	Positive
Terrible experience!	Negative
This place has best food	Positive
This place has most pathetic serving food!	Negative

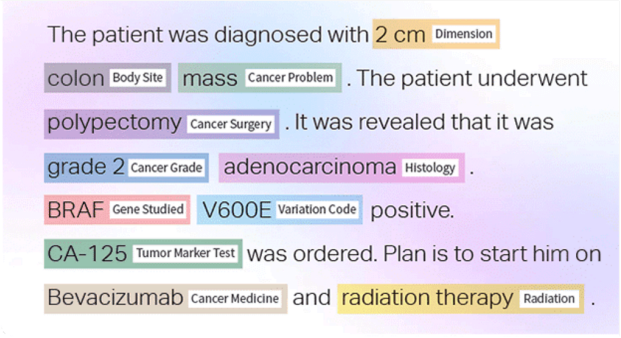
Labelled Training Dataset

Types of text Classification: Entity Recognition

Entity Recognition: This type of classification involves **identifying and classifying specific entities within the text**, such as names of people, organizations, locations, dates, etc. The goal is to label specific words or phrases with their corresponding entity types, requiring more fine-grained text analysis.

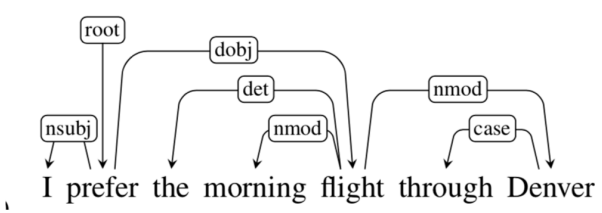


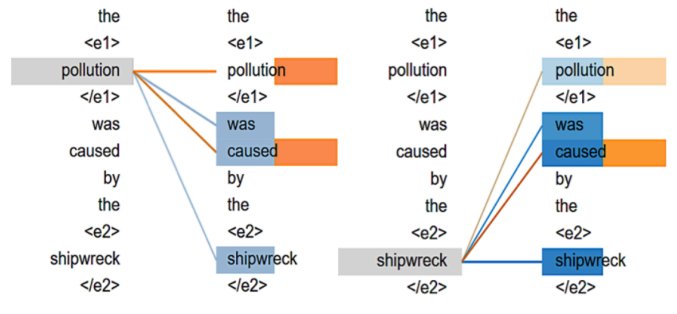


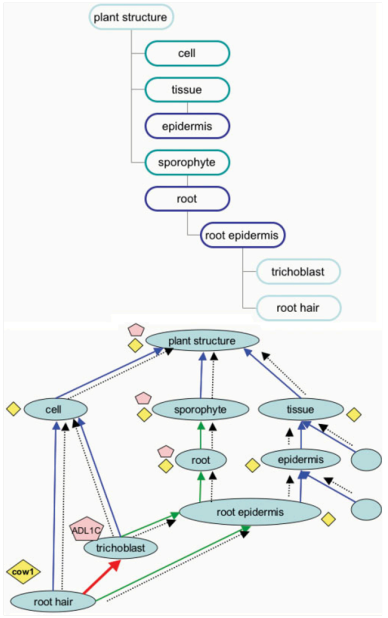


Types of text Classification: Relation Classification

Relation Classification: This is the most complex type of text Classification, where not only entities are identified, but also the relationships between them (e.g., a person working for an organization). We have to **understand the entities and how they are connected within the context of the sentence or text**, which requires deeper semantic understanding.

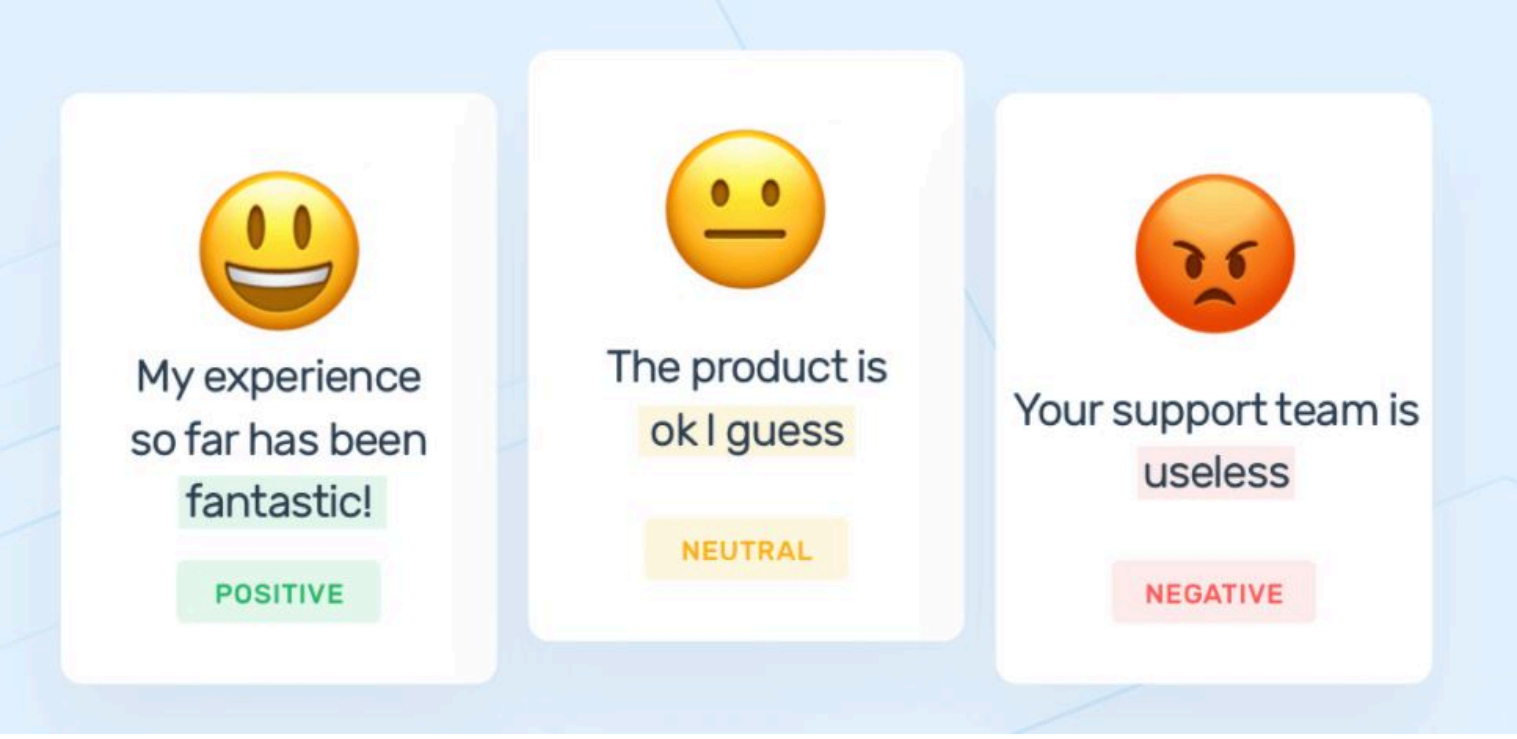






Sentiment Analysis

Sentiment Analysis is a classification task which aims to automatically detect whether the emotional tone of a text is positive, negative, or neutral. What is the sentiment expressed by the author in a document? It can be assessed through a plethora of different unsupervised techniques (both supervised and unsupervised).

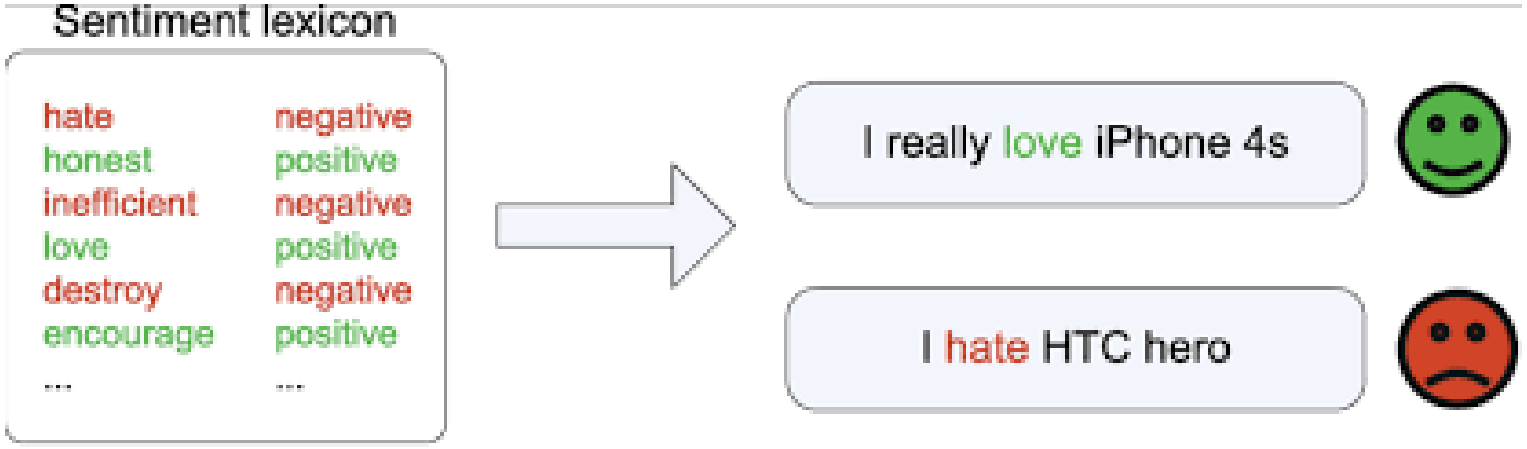


Sentiment Analysis - Applications

- Measuring consumer sentiment (e.g. reviews)
- Recommender systems
- Market research & competitive research
- Social media monitoring

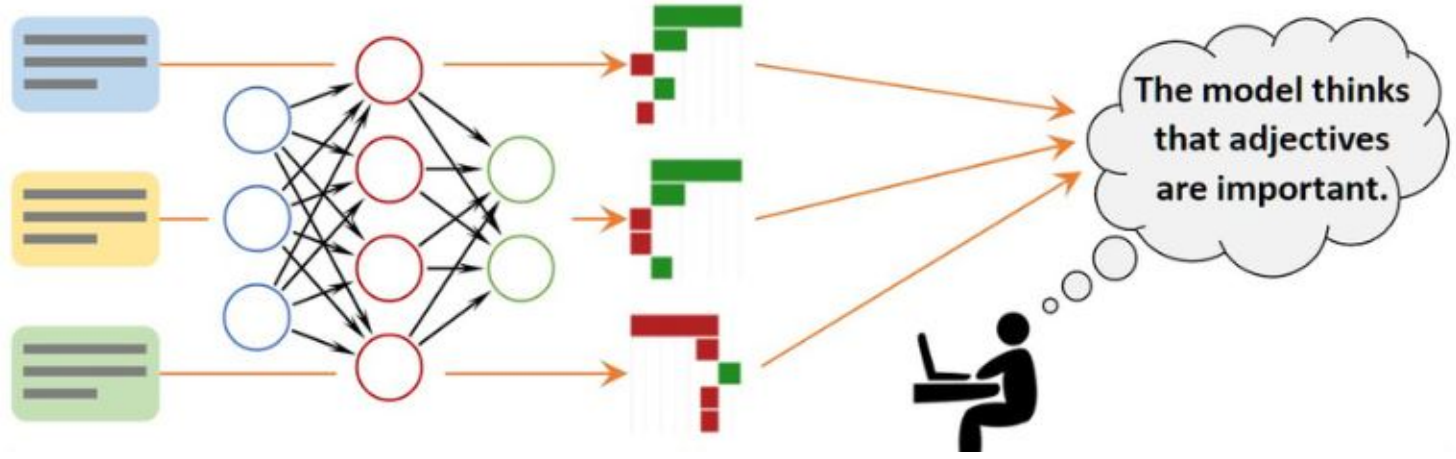
Sentiment Analysis - Approaches: Rule-Based

- Rely on manually created rules to analyze and categorize text sentiment.
- Dictionaries of words annotated with sentiments or grammatical rules to estimate the emotion.
- Simple to implement and understand (high interpretability), but require manual effort to create rules. Also cannot handle context-dependent meaning.



Sentiment Analysis - Approaches: Machine Learning

- utilize machine learning algorithms to automatically learn patterns from labeled training data to classify sentiment.
- High accuracy and adaptability, but require large amounts of labelled data, and models are usually black-boxes (less interpretable).



Sentiment Analysis - In practice

One way to analyze the sentiment of a text is to consider the text as a combination of its individual words and the sentiment content of the whole text as the sum of the sentiment content of the individual words. This isn’t the only way to approach sentiment analysis, but it is an often-used approach, and an approach that naturally takes advantage of the tidy tool ecosystem.

The tidytext package provides access to several sentiment lexicons, to evaluate the emotion of a text based on its unigrams. These lexicons contain many English words and the words are assigned scores for positive/negative sentiment.

Sentiment Analysis - Rule-based

With data in a tidy format, sentiment analysis can be done as an inner join. Let's use a dataset of Twitter.

```r
library(tidyverse)
library(tidytext)

# Twitter dataset
tweet_data |>
  head(10)
```

```
## # A tibble: 10 × 3
##   tweet_id original_text                                      tasks
##      <dbl> <chr>                                            <chr>
## 1  1.60e18 "[GPT-3] This post discusses the need for an au… ensu…
## 2  1.60e18 "Today is the pre-fine-tuning day! \r\n\r\nI'm … unde…
## 3  1.60e18 "@DeclanMccreary They don't program it to inven… inve…
## 4  1.60e18 "Screenshot of a project I'm working on using t… wrot…
## 5  1.60e18 "@Vkumzy @MsRabbitKick @rippleitinNZ It's OpenA… writ…
## 6  1.60e18 "@Andrew___Baker When I tried it, GPT-3 nailed … nail…
## # ℹ 4 more rows
```

Sentiment Analysis - Rule-based

Tidy the text of tweets with one word per row.

```
# Tokenization
tidy_twitter <- tweet_data |>
  select(-tasks) |>
  unnest_tokens(output = word, input = original_text, drop = F)
```

Sentiment Analysis - Rule-based

We use the NRC lexicon.

```
#install.packages("textdata")
library(tidytext)

nrc <- get_sentiments("nrc")

nrc
```

```
## # A tibble: 13,872 × 2
##   word      sentiment
##   <chr>     <chr>
## 1 abacus    trust
## 2 abandon   fear
## 3 abandon   negative
## 4 abandon   sadness
## 5 abandoned anger
## 6 abandoned fear
## # ℹ 13,866 more rows
```

Sentiment Analysis - Rule-based

We filter only the type of sentiment we are interested in.

```
nrc_sentiment <- nrc |>
  filter(sentiment == "positive" | sentiment == "negative")

nrc_sentiment
```

```
## # A tibble: 5,624 × 2
##   word        sentiment
##   <chr>       <chr>
## 1 abandon     negative
## 2 abandoned   negative
## 3 abandonment negative
## 4 abba        positive
## 5 abduction   negative
## 6 aberrant    negative
## # ℹ 5,618 more rows
```

Sentiment Analysis - Rule-based

Now we can perform the Sentiment Analysis through the inner join.

```r
# Perform Sentiment Analysi
tidy_twitter_sentiment <- tidy_twitter |>
  inner_join(nrc_sentiment)

# Example
tidy_twitter_sentiment |>
  head(5)
```

```
## # A tibble: 5 × 4
##   tweet_id original_text                          word  sentiment
##      <dbl> <chr>                                  <chr> <chr>
## 1  1.60e18 "[GPT-3] This post discusses the need… inte… positive
## 2  1.60e18 "[GPT-3] This post discusses the need… coop… positive
## 3  1.60e18 "[GPT-3] This post discusses the need… focus positive
## 4  1.60e18 "Today is the pre-fine-tuning day! \r… comp… positive
## 5  1.60e18 "@DeclanMccreary They don't program i… model positive
```

Sentiment Analysis - Rule-based

```
# Count nuymber of positive and negative tweets
tidy_twitter_sentiment |>
  select(tweet_id, sentiment) |>
  unique() |>
  count(sentiment, sort = TRUE)
```

```
## # A tibble: 2 × 2
##   sentiment     n
##   <chr>     <int>
## 1 positive    678
## 2 negative    362
```

Sentiment Analysis - Rule-based

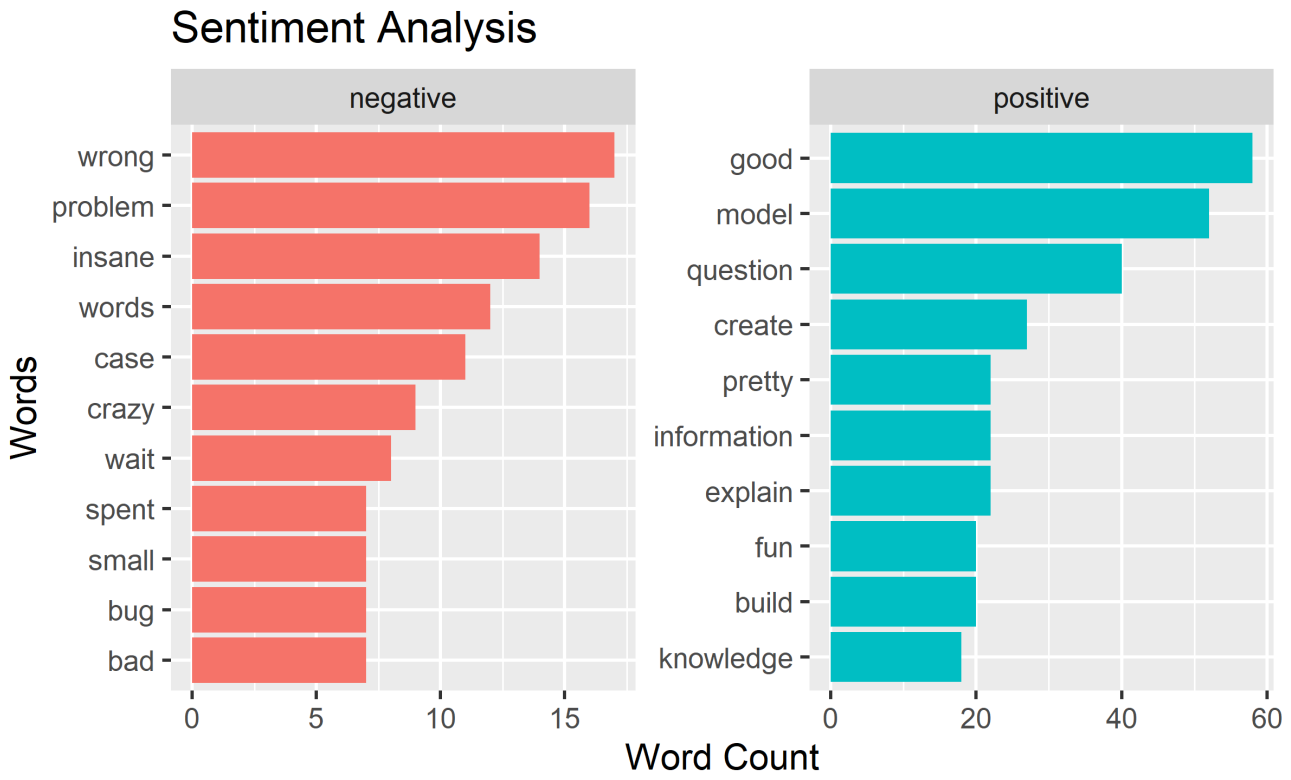
We can plot the top-10 negative and positive words.

```
# Top-10 negative and positive words
top_10_words <- tidy_twitter_sentiment |>
  count(word, sentiment) |>
  group_by(sentiment) |>
  top_n(10)

# Plot
top_10_words |>
  mutate(word = fct_reorder(word, .x = n)) |>
  ggplot(aes(x = word, y = n, fill = sentiment)) +
  geom_col(show.legend = F) +
  labs(title = "Sentiment Analysis",
       x = "Words",
       y = "Word Count") +
  coord_flip() +
  facet_wrap(~sentiment, scale = "free")
```

Sentiment Analysis - Rule-based

We can plot the top-10 negative and positive words.



Sentiment Analysis - Rule-based

We can calculate the **sentiment score** for each tweet using `sentimentr`. This package use also *amplifier* and *negation* lists.

```
# install.packages("sentimentr")
library(sentimentr)
library(magrittr)

# Get sentiment with NRC lexicon
twitter_polarity_score <- tweet_data %$%
  sentiment_by(original_text, by = tweet_id, polarity_dt = lexicon::hash_sentiment_nrc)
```

Sentiment Analysis - Rule-based

```r
# Most Negative tweets
twitter_polarity_score |>
  arrange(ave_sentiment) |>
  left_join(tweet_data) |>
  select(tweet_id, original_text, ave_sentiment) |>
  head(5) |>
  as_tibble()
```

```
## # A tibble: 5 × 3
##   tweet_id original_text                                ave_sentiment
##      <dbl> <chr>                                                <dbl>
## 1  1.60e18 "Seeing all the ChatGPT examples with m…            -0.970
## 2  1.60e18 "GPT definitely poses an existential th…            -0.636
## 3  1.60e18 "@LeftTheCoast The real death knell for…            -0.621
## 4  1.60e18 "@bentossell #ChatGPT also casts Hitler…            -0.614
## 5  1.60e18 "@Marc__Watkins @OpenAI @sharplm @Engli…            -0.568
```

Sentiment Analysis - Rule-based

```r
# Most Positive tweets
twitter_polarity_score |>
  arrange(desc(ave_sentiment)) |>
  left_join(tweet_data) |>
  select(tweet_id, original_text, ave_sentiment) |>
  head(5) |>
  as_tibble()
```

```
## # A tibble: 5 × 3
##   tweet_id original_text                                ave_sentiment
##      <dbl> <chr>                                                <dbl>
## 1  1.60e18 "have been using GPT-3 to adapt text fo…         1.71
## 2  1.60e18 "I asked chatgpt how to debug a distrib…         1.30
## 3  1.60e18 "ChatGPT might put Javascript and secur…         0.919
## 4  1.60e18 "Welp, working as a software developer …         0.879
## 5  1.60e18 "\"ChatGPT, write some haiku about the …         0.857
```

Sentiment analysis - Machine Learning

To prepare the data for Sentiment analysis, we first select only the tweet text from the dataset.

```
#{r}
# Select only the tweet text
sentiment_data <- tweet_data |>
  select(original_text) |>
  rename(text = original_text)
```

```
## # A tibble: 5 × 1
##   text
##   <chr>
## 1 "[GPT-3] This post discusses the need for an audit system for …
## 2 "Today is the pre-fine-tuning day! \r\n\r\nI'm going to start …
## 3 "@DeclanMccreary They don't program it to invent citations. Th…
## 4 "Screenshot of a project I'm working on using the new GPT-3 mo…
## 5 "@Vkumzy @MsRabbitKick @rippleitinNZ It's OpenAI's GPT-3 with …
```

Sentiment analysis - Machine Learning

Set the python environment.

```
#{r}
library(reticulate)

# Use the desired Python environment (adjust the path to your specific environment)
use_python("C:/Users/Vito Giordano/anaconda3/python.exe", required = TRUE)
py_config()
```

```
## python:         C:/Users/Vito Giordano/anaconda3/python.exe
## libpython:      C:/Users/Vito Giordano/anaconda3/python311.dll
## pythonhome:     C:/Users/Vito Giordano/anaconda3
## version:        3.11.4 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 13:38:37) [MSC v.1916 64 bit
## Architecture:   64bit
## numpy:          C:/Users/Vito Giordano/anaconda3/Lib/site-packages/numpy
## numpy_version:  1.24.3
##
## NOTE: Python version was forced by use_python() function
```

Sentiment analysis - Machine Learning

Set the python environment.

```
#{r}
# Import transformers library
transformers <- import("transformers")
torch <- import("torch")
```

Transformers is a library developed by Hugging Face that provides pre-trained models and tools for NLP.
PyTorch is a machine learning library developed by Facebook that provides tools for deep learning and tensor computation.

Sentiment analysis - Machine Learning

To perform Sentiment analysis, we use the **pipeline()** function from the Transformers library. The **pipeline()** function provides a simple **interface** for applying **pre-trained models from Hugging Face** to common NLP tasks. In our case, we use a model specifically trained on Twitter data:

```
#{python}
from transformers import pipeline, AutoTokenizer, AutoModelForSequenceClassification
import pandas as pd

# Select a pretrained model (from Huggingface)
model_name = "cardiffnlp/twitter-roberta-base-sentiment-latest"
```

Sentiment analysis - Machine Learning

To perform Sentiment analysis, we use the **pipeline()** function from the Transformers library. The **pipeline()** function provides a simple **interface** for applying **pre-trained models from Hugging Face** to common NLP tasks.

Then we apply the pipeline function:

```
#{python}
# Initialize the tokenizer
tokenizer = AutoTokenizer.from_pretrained(model_name)

# Initialize the model for the task
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# Load Sentiment analysis pipeline
sentiment_analyzer = pipeline("sentiment-analysis", model=model, tokenizer=tokenizer)
```

Sentiment analysis - Machine Learning

Now we calculate the sentiment of the tweets in the dataset.

```
#{python}
# Define a function that returns label and scores from the sentiment analyzer
def analyze_sentiment(text):
    result = sentiment_analyzer(text, top_k=3)

    # Create a dictionary to store scores
    scores = {entry['label'].lower(): entry['score'] for entry in result}

    # Get predicted sentiment (the one with highest score)
    top_sentiment = max(result, key=lambda x: x['score'])['label'].lower()

    return {
        "sentiment": top_sentiment,
        "positive_score": scores.get("positive", 0.0),
        "neutral_score": scores.get("neutral", 0.0),
        "negative_score": scores.get("negative", 0.0)
    }
```

Sentiment analysis - Machine Learning

Now we calculate the sentiment of the tweets in the dataset.

```
#{python}
# Convert the R datasets in python
sentiment_data = r.sentiment_data

# Apply the function to each tweet
sentiment_results = sentiment_data["text"].apply(analyze_sentiment)

# Convert the result to a dataframe
sentiment_df = sentiment_results.apply(pd.Series)

# Add the results to the sentences
sentiment_classification = pd.concat([sentiment_data, sentiment_df], axis=1)
```

Sentiment analysis - Machine Learning

We can see the results.

```
#{r}
# Convert python results in R
sentiment_results <- tibble(py$sentiment_classification)
```

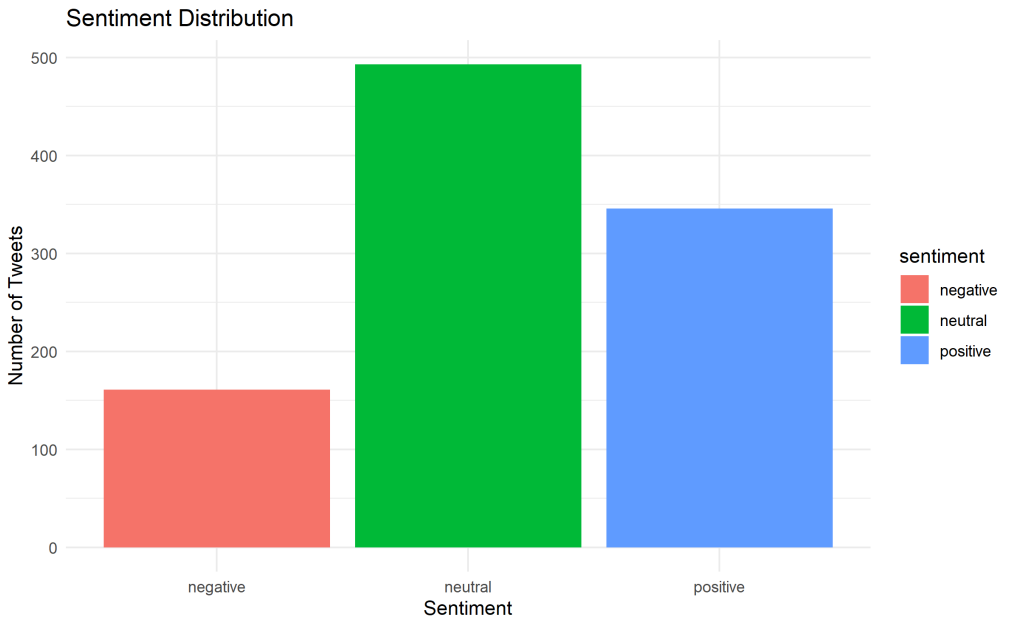
```
## # A tibble: 6 × 5
##   text      sentiment positive_score neutral_score negative_score
##   <chr>     <chr>              <dbl>         <dbl>          <dbl>
## 1 "[GPT-3]… neutral             0.09          0.87           0.03
## 2 "Today i… neutral             0.33          0.66           0.01
## 3 "@Declan… negative            0.01          0.21           0.78
## 4 "Screens… neutral             0.05          0.64           0.32
## 5 "@Vkumzy… neutral             0.09          0.88           0.03
## 6 "@Andrew… neutral             0.18          0.64           0.18
```

Sentiment analysis - Machine Learning

We can see here the number of tweets with the different sentiments.

```
#{r}
library(ggplot2)

# Plot sentiment distribution
ggplot(sentiment_results,
  aes(x = sentiment, fill = sentiment)) +
  geom_bar() +
  labs(title = "Sentiment Distribution",
       x = "Sentiment",
       y = "Number of Tweets") +
  theme_minimal()
```



Sentiment analysis - Machine Learning

We can see also the distribution of the sentiment scores.

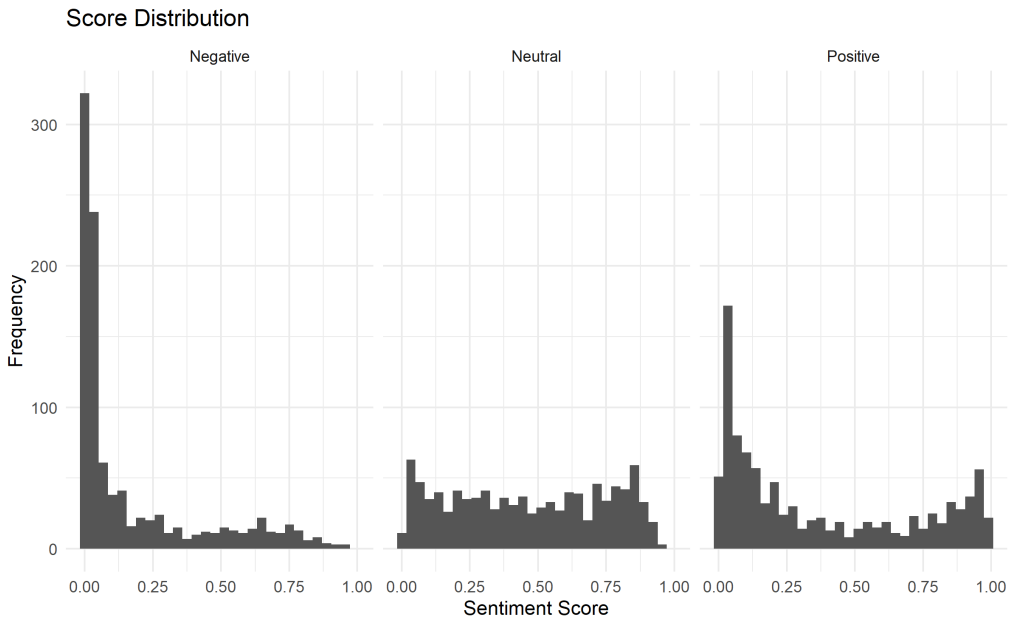
Before exploring the distribution, we have to preprocess the data.

```
#{r}
# Convert sentiment results data to long format
sentiment_long <- sentiment_results |>
  pivot_longer(cols = c(positive_score, neutral_score, negative_score),
               names_to = "score_type",
               values_to = "score") |>
  mutate(score_type = case_when(
    score_type == "positive_score" ~ "Positive",
    score_type == "neutral_score" ~ "Neutral",
    score_type == "negative_score" ~ "Negative"
  ))
```

Sentiment analysis - Machine Learning

Here the distribution of the sentiment scores.

```
#{r}
# Plot scores distribution
ggplot(sentiment_long, aes(x = score)) +
  geom_histogram() +
  facet_wrap(~ score_type) +
  labs(title = "Score Distribution",
       x = "Sentiment Score",
       y = "Frequency") +
  theme_minimal()
```

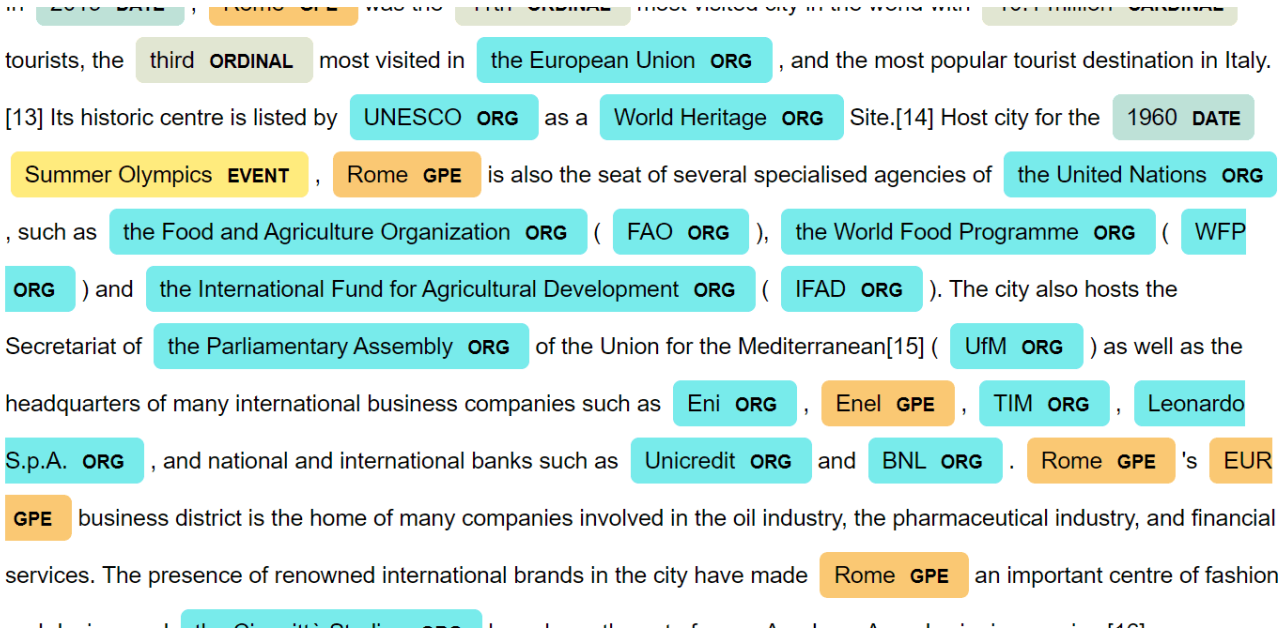


Named Entity Recognition

What Named Entity Recognition is?

Named Entity Recognition (NER) is a subfield of natural language processing (NLP) that involves **identifying** and **classifying** named entities in textual data.

Named entities are specific objects or concepts that have a name, such as people, organizations, locations, dates, times, and products.



NER Methods

There are three methods of NER:

- **Gazetteer-based NER** uses knowledge base resources (or gazette in jargon), consisting of lists of known instances, to map mentions of entities within texts to knowledge base resources (e.g. Wikipedia). It is a fast and accurate approach for NER. The gazette-based NER is a semi-supervised method since list creation usually requires manual effort and domain-specific knowledge;

- **Rule-based NER** uses regular expressions and morphosyntactic information to express knowledge based systems able to extract a certain type of entity. This system is unsupervised and top-down, and falls into the class of expert systems;

- **Machine Learning-based NER** uses manually annotated documents (training set) to train machine learning algorithms. These methods are the-state-of-the-art in many fields of Natural Language Processing, but they require a large amount of effort to collect and annotate large data-set;

- **Prompt-based NER** is a novel and flexible approach leveraging the capabilities of large pre-trained language models (e.g., GPT, BERT) to identify entities in text by providing specific prompts or instructions. It is a recent paradigm in NER that eliminates the need for extensive annotated datasets or rule-based systems by using natural language prompts to guide the model's understanding of entities.

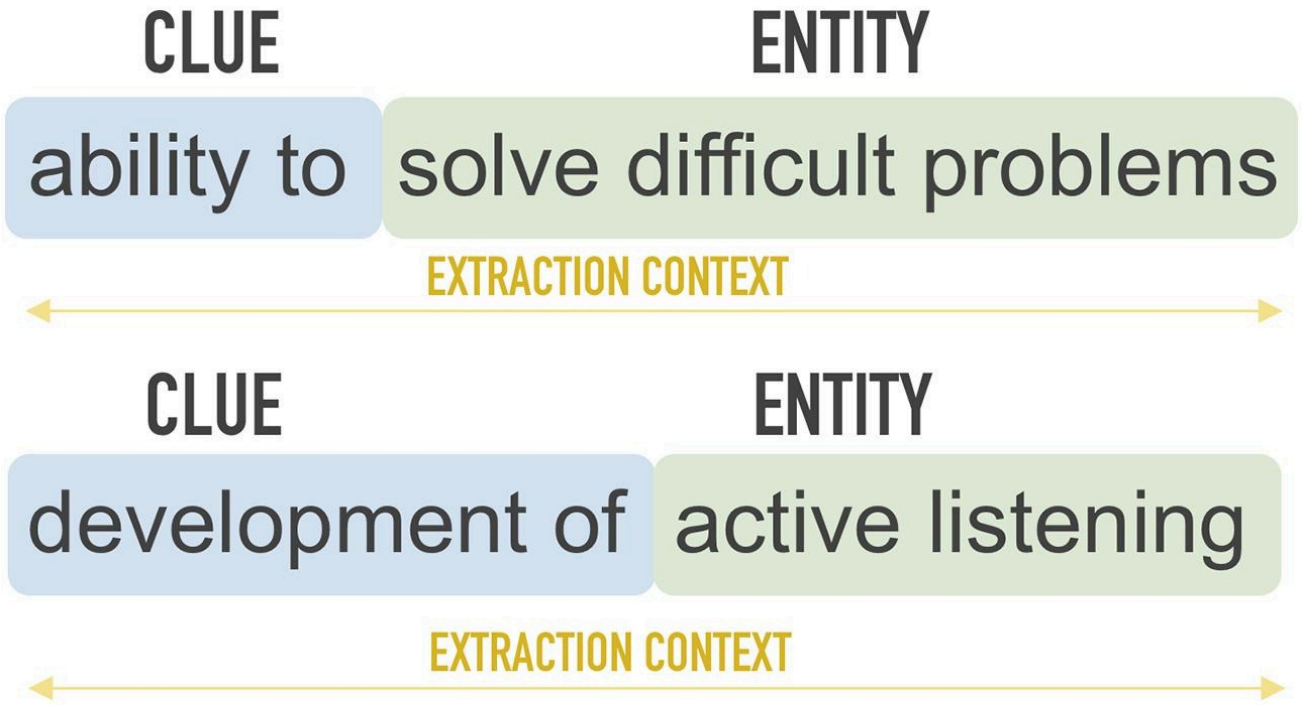
Gazetteer-based NER

Gazetteer-based NER uses knowledge base resources (or gazette in jargon), consisting of lists of known instances, to map mentions of entities within texts to knowledge base resources (e.g. Wikipedia). It is a fast and accurate approach for NER. The gazette-based NER is a semi-supervised method since list creation usually requires manual effort and domain-specific knowledge.

```
anchorman
anchorperson
andrologist
anesthesiologist
anesthesiologists
anesthetist
angels
angiologist
anglers
animators
anorectic
anorexic
antelope
anthropologist
anthropologists
antiquarian
ants
```

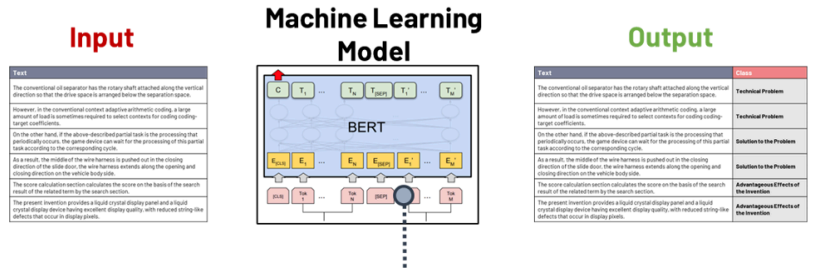
Rule-based NER

Rule-based NER uses regular expressions and morphosyntactic information to express knowledge based systems able to extract a certain type of entity. This system is unsupervised and top-down, and falls into the class of expert systems.



Machine Learning-based NER

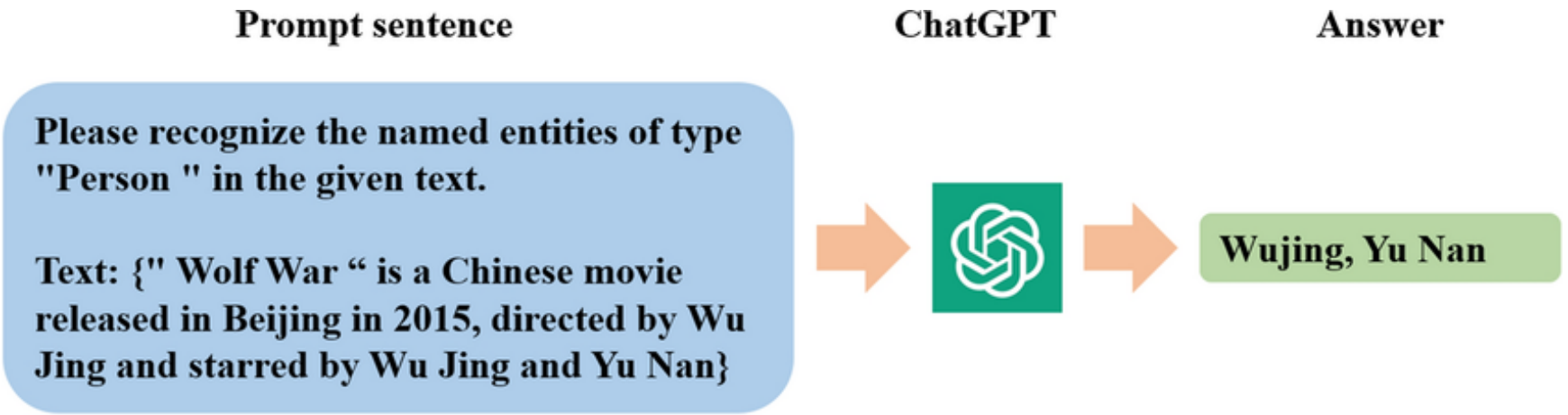
Machine Learning-based NER uses manually annotated documents (training set) to train machine learning algorithms. These methods are the-state-of-the-art in many fields of Natural Language Processing, but they require a large amount of effort to collect and annotate large data-set



Text	pen	dog	is	on	horizon	大	Class
The dog is on the horizon	0	1	1	1	1	0	Technical Problem
This pen is a pen	2	0	1	0	0	0	Solution
大	0	0	0	0	0	1	Other

Prompt-based NER

Prompt-based NER is a novel and flexible approach leveraging the capabilities of large pre-trained language models (e.g., GPT, BERT) to identify entities in text by providing specific prompts or instructions. It is a recent paradigm in NER that eliminates the need for extensive annotated datasets or rule-based systems by using natural language prompts to guide the model's understanding of entities.



Evaluate NER performance

To evaluate the performance of a Named Entity Recognition (NER) system, a common approach is to use Precision, Recall, and F1-score.

To know more about this measurements see this page: Precison and Recall Wikipedia

Precision

Precision measures the accuracy of the entities identified by the NER system. It is the ratio of correctly identified entities (true positives) to the total entities identified by the system (true positives + false positives). It is calculated as:

$$Precision = \frac{\text{True Positives}}{\text{True Positives} + \text{False Positives}}$$

Precision answers the question: *Of all the entities the system identified, how many are actually correct?*

Recall

Recall measures the system's ability to identify all relevant entities. It is the ratio of correctly identified entities (true positives) to the total number of actual entities (true positives + false negatives). It is calculated as:

$$Recall = \frac{\text{True Positives}}{\text{True Positives} + \text{False Negatives}}$$

Recall answers the question: *Of all the actual entities in the text, how many did the system correctly identify?*

F1-Score

The F1-score is the harmonic mean of precision and recall, balancing both measures. It is particularly useful when there is an uneven class distribution or when precision and recall need to be equally weighted. It is calculated as:

$$F1 = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}$$

F1-score answers the question: *What is the overall effectiveness of the system in identifying entities?*

Evaluate NER performance

This ergonomic office chair is a game-changer! It's comfortable, sturdy, and easy to adjust for perfect support during long work hours. Great value for the price!	This ergonomic office chair is a game-changer! It's comfortable, sturdy, and easy to adjust for perfect support during long work hours. Great value for the price!
I recently purchased this wireless Bluetooth speaker, and I couldn't be happier with my decision. The sound quality is crystal clear, with deep bass and balanced highs, perfect for any music genre. It's compact and lightweight, making it super easy to carry around, yet the battery life is impressive, lasting me an entire day of continuous use. The build quality feels premium, and it connects seamlessly to my devices. Definitely worth every penny!	I recently purchased this wireless Bluetooth speaker, and I couldn't be happier with my decision. The sound quality is crystal clear, with deep bass and balanced highs, perfect for any music genre. It's compact and lightweight, making it super easy to carry around, yet the battery life is impressive, lasting me an entire day of continuous use. The build quality feels premium, and it connects seamlessly to my devices. Definitely worth every penny!
This wireless keyboard is fantastic! It's lightweight, responsive, and pairs effortlessly with my devices. Perfect for both work and travel!	This wireless keyboard is fantastic! It's lightweight, responsive, and pairs effortlessly with my devices. Perfect for both work and travel!

	Positive	Negative
Positive	**True Positive**	**False Negative**
Negative	**False Positive**	**True Negative**

	Positive	Negative
Positive	**2**	**1**
Negative	**2**	**-**

Evaluate NER performance

	Positive	Negative
Positive	**True Positive**	**False Negative**
Negative	**False Positive**	**True Negative**

	Positive	Negative
Positive	**2**	**1**
Negative	**2**	**-**

$$Precision = \frac{\text{True Positives}}{\text{True Positives}+\text{False Positives}} = \frac{2}{2+2} = 0.50$$

$$Recall = \frac{\text{True Positives}}{\text{True Positives}+\text{False Negatives}} = \frac{2}{2+1} = 0.66$$

$$F1 = 2 \cdot \frac{\text{Precision}\cdot\text{Recall}}{\text{Precision}+\text{Recall}} = 2 \cdot \frac{0.50\cdot 0.66}{0.50+0.66} = 0.57$$

Evaluate NER performance

Remember that the performance of each Named Entity Recognition (NER) system depends on several factors:

- **Language**: NER models are typically language-specific. For example, a model developed for English cannot be directly applied to Italian.
- **Document Characteristics**: Each type of document has its own characteristics, such as structure, tone, and terminology. A NER system developed for scientific papers may not perform well on informal text like social media posts or casual emails.
- **Domain**: Many NER systems are designed for specific domains (e.g., medical, legal, or financial). The same entity type (e.g., "location") may be interpreted differently depending on the domain. For example, in a medical domain, "location" might refer to a body part, while in a geographic context, it refers to places.
- **Entity Definitions**: The definition of entities can vary. For instance, what constitutes an "organization" in one domain might be broader or narrower in another.

Open Source NER systems

In general, there are many open-source Named Entity Recognition (NER) systems already developed and widely used.

NER System	Description	Default Entity Types	Link
spaCy	A robust NLP library with built-in NER capabilities and pre-trained models.	PERSON, NORP, FAC, ORG, GPE, LOC, PRODUCT, EVENT, WORK_OF_ART, LAW, LANGUAGE, DATE, TIME, PERCENT, MONEY, QUANTITY, ORDINAL, CARDINAL	https://spacy.io/
NLTK	A foundational library for symbolic and statistical NLP with basic NER.	PERSON, ORGANIZATION, LOCATION, DATE, TIME, MONEY, PERCENT, FACILITY, GPE	https://www.nltk.org/
Stanford CoreNLP	An advanced NLP toolkit supporting dependency parsing, coreference, and NER.	PERSON, LOCATION, ORGANIZATION, MISC	https://stanfordnlp.github.io/CoreNLP/
Flair	A library offering state-of-the-art NER models with contextual embeddings.	PER, LOC, ORG, MISC	https://github.com/flairNLP/flair
AllenNLP	A platform for deep learning NLP models, including NER via datasets like CoNLL-2003.	PER, ORG, LOC, MISC	https://allennlp.org
Hugging Face Transformers	Provides pre-trained transformer models (e.g., BERT) fine-tunable for NER tasks.	PER, ORG, LOC, MISC (varies by model)	https://huggingface.co/transformers/
Apache OpenNLP	A machine learning-based NLP toolkit supporting custom and pre-trained models.	PERSON, LOCATION, ORGANIZATION	https://opennlp.apache.org/
Polyglot	A multilingual NLP library supporting NER for various languages.	PER, LOC, ORG, MISC	https://polyglot.readthedocs.io/

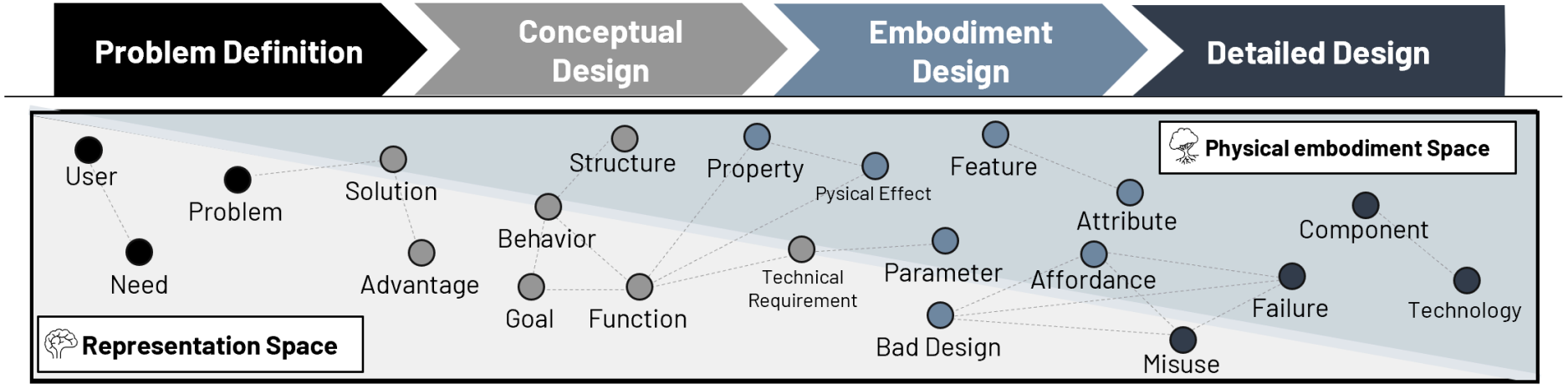
Which entities can be useful?

The problem definition phase concerns the **identification** of **users** and **needs** that the new technology satisfies.

Innovators sketch the broad **function** and **structure** of the new technologies and **identify potential solutions** to meet the user needs and solve the problems.

The **design is developed** and the functions, shapes, attributes, parameter values and properties are defined.

Each **component** of the solution **is detailed designed** and **tested** for ensuring that the technology meets the user needs.



Gazetteer-based NER

Gazetteer-based NER

The steps to create a gazetteer-based NER are:

1. **Identify the entities to be recognized**: The first step is to define the types of entities to be recognized, such as people, organizations, or locations.
2. **Create gazetteer lists**: Create gazetteer lists for each type of entity, which contain the names and synonyms of the entities you want to recognize.
3. **Apply gazetteer-based NER**: Once the textual data is preprocessed, you can apply gazetteer-based NER to recognize and extract the named entities. This involves matching the tokens in the text with the entries in the gazetteer lists. If a token matches an entry in a gazetteer list, it is classified as a named entity of the corresponding type.

Create gazetteer lists

The main sources to build a gazettee are:

- **Domain Database**: A domain database is a collection of entities related to a specific domain, such as geography, finance, or sports. Domain databases can be created *manually* or *automatically* by crawling relevant websites or databases. Once you have created a domain database, you can use it to create a gazetteer list of named entities in the corresponding domain.
- **Wikipedia**: Wikipedia is a vast online encyclopedia that contains a wealth of information about named entities in various domains. When using Wikipedia for creating a gazetteer-based NER, there are several ways to extract relevant information. Two common approaches are: (1) Using *hyperlinks*: Wikipedia articles often contain hyperlinks to other articles about related topics; (2) Using *"List of"* pages : Many Wikipedia pages are dedicated to providing lists of named entities related to specific domains.
- **ChtGPT**: ChatGPT is a large language model that can be used for various NLP tasks, including named entity recognition. You can use ChatGPT to generate named entities from a given text or domain. For example, you can provide ChatGPT with a list of keywords or phrases related to a specific domain, and it can generate a list of named entities based on the context and the language patterns in the given text.

Gazetteer-based NER: Applications in R

```r
library(tidyverse)
library(reticulate)
library(readxl)

# Import a list of papers
precision_agriculture_paper <- read_csv("data/precision_agriculture.csv")
```

Gazetteer-based NER: Applications in R

```r
# Glimpse on the table
glimpse(precision_agriculture_paper %>% select(1:20))
```

```
## Rows: 13,157
## Columns: 20
## $ Authors                     <chr> "Hale G.A.; Cox R.D.; Ritch…
## $ `Author full names`         <chr> "Hale, Gabriella A. (585752…
## $ `Author(s) ID`              <chr> "58575287900; 35323666500; …
## $ Title                       <chr> "Canopy detection beyond th…
## $ Year                        <dbl> 2023, 2023, 2023, 2023, 202…
## $ `Source title`              <chr> "Sensing and Bio-Sensing Re…
## $ Volume                      <chr> "42", "14", "13", "215", "2…
## $ Issue                       <chr> NA, NA, "1", NA, NA, NA, NA…
## $ `Art. No.`                  <chr> "100587", "100774", "5", "1…
## $ `Page start`                <chr> NA, NA, NA, NA, NA, NA, NA,…
## $ `Page end`                  <chr> NA, NA, NA, NA, NA, NA, NA,…
## $ `Page count`                <dbl> NA, NA, NA, NA, NA, NA, NA,…
## $ `Cited by`                  <dbl> 0, 0, 1, 0, 0, 0, 0, 0, 0, …
## $ DOI                         <chr> "10.1016/j.sbsr.2023.100587…
## $ Link                        <chr> "https://www.scopus.com/inw…
## $ Affiliations                <chr> "Fiber and Biopolymer Resea…
## $ `Authors with affiliations` <chr> "Hale G.A., Fiber and Biopo…
## $ Abstract                    <chr> "The development of Canopeo…
## $ `Author Keywords`           <chr> "GCF; Image analysis; Mobil…
## $ `Index Keywords`            <chr> NA, NA, "Anomaly detection;…
```

Gazetteer-based NER: Applications in R

```r
# Read a huge list of users
user <- read_lines("data/target_groups.txt")

user[1:10]
```

```
##  [1] "data scientist" "chimpanzee"     "fisherman"
##  [4] "marketer"       "agender"        "androgyny"
##  [7] "androsexual"    "androphilic"    "aromantic"
## [10] "asexual"
```

How the list of users is created

Gazetteer-based NER: Applications in R

Create a regex with all the users so that you can then extract them from the text

```
# Creo una regex
user_regex <- user %>%
  str_c("\\b", ., "\\b") %>%
  str_c(., collapse = "|")
```

Gazetteer-based NER: Applications in R

Or ...

```
from retrie.trie import Trie

# Create an element of trie regex
trie = Trie()

# Add the n-grams from R to the Trie
trie.add(*r.user)

# Generate the regex
trie_regex = trie.pattern()

# Add word boundaries around the generated pattern
user_regex = r"\b" + trie_regex + r"\b"
```

Gazetteer-based NER: Applications in R

Extract users from Title, Abstract and Keywords.

```
# Extract users
paper_ner <- precision_agriculture_paper %>%
  mutate(text = str_c( # merge title, abstract and keywords
    Title,
    Abstract,
    `Author Keywords`,
    `Index Keywords`
  )) %>%
  mutate(text = str_to_lower(text)) %>%  # testo tutto minuscolo
  mutate(user = str_extract_all(text, user_regex)) %>%
  unnest(user) %>%  # The extraction returns a vector, I put everything in tidy format
  filter(!is.na(user)) %>%  # remove articles where I found nothing
  select(-text) # remove the created column
```

Gazetteer-based NER: Applications in R

```r
paper_ner %>%
  select(EID, user)
```

```
## # A tibble: 12,737 × 2
##   EID                 user
##   <chr>               <chr>
## 1 2-s2.0-85175339657 trained
## 2 2-s2.0-85173184548 maps
## 3 2-s2.0-85175469914 experts
## 4 2-s2.0-85175469914 trained
## 5 2-s2.0-85172702458 nonhuman
## 6 2-s2.0-85173248612 african
## # ℹ 12,731 more rows
```

As you can see, sometimes the algorithm also catches entities that are not users. This is why an additional **Cleaning** step is necessary.

Gazetteer-based NER: Applications in R

To remove entities that are not users I manually analyzed each user and classified them.

1	user	user_label	label	label_spec_gen	remove
2	actor	actor	Professional Users	Specific	no
3	actors	actor	Professional Users	Specific	no
4	adolescent	young	Age User	Generic	no
5	adult	adult	Age User	Generic	no
6	adults	adult	Age User	Generic	no
7	adviser	advisor	Business Users	Specific	no
8	advisor	advisor	Business Users	Specific	no
9	aeronautics	aeronautic	Professional Users	Specific	no
10	african	african	Nationality Users	Specific	no
11	ancient	senior	Age User	Generic	no
12	animal	animal	Animal	Generic	yes
13	animals	animal	Animal	Generic	yes
14	antelope	antelope	Animal	Specific	yes
15	apprentice	apprentice	Professional Users	Generic	no
16	army	army	Other Users	Specific	no
17	artists	artist	Professional Users	Specific	no
18	as victim	victim	Hospital Users	Specific	no
19	astronautics	astronautic	Professional Users	Specific	no
20	baby	baby	Age User	Generic	no
21	badgers	badger	Animal	Specific	yes
22	bats	bat	Animal	Specific	yes
23	bear	bear	Animal	Specific	yes
24	bee	bee	Animal	Specific	yes
25	bicyclist	bicyclist	Professional Users	Specific	no
26	bird	bird	Animal	Specific	yes
27	birds	bird	Animal	Specific	yes
28	brokers	broker	Business Users	Specific	no
29	buffalo	buffalo	Animal	Specific	yes
30	canadian	canadian	Nationality Users	Specific	no

Gazetteer-based NER: Applications in R

At this point I used the table to remove entities from my extract table.

```
# Import table
user_class <- read_xlsx("data/User Classes.xlsx")

# Remove elements that are not "user"
paper_ner_clean <- paper_ner %>%
  left_join(user_class) %>%
  filter(remove == "no")
```

Gazetteer-based NER: Applications in R

Analyze the number of articles for each user to understand how much impact has been given by Precision Agriculture.

```r
# Total number of papers
n_of_papers <- precision_agriculture_paper %>%
  nrow()

# Count the number of papers that appears in different user categor
count_users_class <- paper_ner_clean %>%
  select(EID, label) %>%
  unique() %>%
  count(label) %>%
  mutate(perc = round(n/n_of_papers, 4)*100)
```

```
## # A tibble: 13 × 3
##   label                       n  perc
##   <chr>                   <int> <dbl>
## 1 Age User                   88  0.67
## 2 Agriculture Users         677  5.15
## 3 Business Users            384  2.92
## 4 Community Users          1658 12.6
## 5 Engineers and Designers   235  1.79
## 6 Family User               59  0.45
## # ℹ 7 more rows
```

Gazetteer-based NER: Applications in R

For each class of users I insert some **examples** to help read the table.

```
# Insert example
most_occ_users <- paper_ner_clean %>%
  select(EID, user_label, label) %>%
  unique() %>%
  count(label, user_label) %>%
  mutate(user_label = str_c(user_label, " (", n, ")")) %>%
  group_by(label) %>%
  arrange(desc(n)) %>%
  filter(row_number() < 11) %>%
  select(-n) %>%
  summarise(user = str_c(user_label, collapse = "; "))

# Join
user_count <- left_join(count_users_class, most_occ_users) %>%
  select(User = label,
         "No. of Papers" = n,
         "% of Papers" = n,
         "Top-10 Users (No. of Papers)" = user)
```

Show 7 entries Search:

	User	No. of Papers	% of Papers	Top-10 Users (No. of Papers)
1	Age User	88	88	young (41); adult (24); younger (14); senior (7); juvenile (3); baby (2); infant (1); kid (1)
2	Agriculture Users	677	677	farmer (348); agricultural engineer (196); agronomist (52); planter (27); biological engineer (21); farm manager (21); dairy farmer (8); agricultural scientist (4); earth scientist (3); winemaker (3)
3	Business Users	384	384	company (95); customer (94); manager (69); producer (63); consultant (28); leader (16); client (11); employee (10); entrepreneur (8); economist (7)
4	Community Users	1658	1658	society (712); individual (563); people (207); community (171); public (148); civilian (16); citizen (6); crew (2)
5	Engineers and Designers	235	235	mechanic (141); engineer (64); instrumentation engineer (14); electronic engineer (7); civil engineer (3); electrical engineer (3); automotive engineer (1); control engineers (1); designer (1); engineering manager (1)
6	Family User	59	59	family (44); mother (5); children (4); offspring (3); parental (3); dad (2); daughter (1); sister (1); son (1)
7	Gender Users	52	52	man (26); female (19); male (14); woman (3); lady (1)

Showing 1 to 7 of 13 entries Previous 1 2 Next

Rule-based NER

Rule-based NER

1. **Identify the entities to be recognized**: Define the categories of entities you aim to extract, such as names, dates, monetary amounts, or custom domain-specific entities.

2. **Define linguistic rules**: Create patterns and rules that identify these entities based on linguistic features, including part-of-speech (PoS) tags, syntactic structures, and dependency relations. These rules are often expressed using regular expressions, grammar rules, or rule templates.

3. **Apply preprocessing**: Tokenize the text and preprocess it with linguistic analysis tools such as PoS tagging, dependency parsing, and syntactic parsing.

4. **Apply rule-based NER**: Use the predefined rules to scan the processed text. Match tokens, phrases, or syntactic patterns that fit the rules. If a match is found, classify the matched sequence as a named entity of the specified type.

Rule-based NER

The rules can be based on:

- **Part-of-Speech Tags (PoS)**: Many entities are associated with specific PoS tags. For example: Proper nouns (NNP) often indicate person or organization names. Cardinal numbers (CD) may represent numerical entities like dates or monetary amounts.
- **Dependency Parsing**: Analyze syntactic dependencies in a sentence to identify relationships between words. For instance: A token with a dependency relation such as "nsubj" or "dobj" to a verb might indicate an entity role (e.g., "visited [Rome]" → Location).
- **Regular Expressions**: Define patterns for specific entity types. For example:
  - Dates: \b\d{1,2}[/-]\d{1,2}[/-]\d{2,4}\b
  - Email addresses: [a-zA-Z0-9._%+-]+@[a-zA-Z0-9.-]+.[a-zA-Z]{2,}
  - Monetary values: \$\d+(.\d{1,2})?

Rule-based NER: Applications in R

We aim to identify all references to users described as **engineers** in the abstracts of scientific publications.

```r
# Import a list of papers
precision_agriculture_paper <- read_csv("data/precision_agriculture.csv")
```

```
## # A tibble: 5 × 2
##   EID                Abstract
##   <chr>              <chr>
## 1 2-s2.0-85171140031 The development of Canopeo as a close-range…
## 2 2-s2.0-85169814363 In this work, the results of the research a…
## 3 2-s2.0-85160597659 Crop protection from diseases through appli…
## 4 2-s2.0-85175339657 The inner leaves of crop canopies are obscu…
## 5 2-s2.0-85173184548 Accurate monitoring of crop nitrogen (N) ac…
```

Rule-based NER: Applications in R

We want to extract every occurrence of the pattern **"ADJ + engineer"** from the abstracts. To do this, we use **UDPipe**, which allows us to annotate the text and identify the part of speech of each word.

```
library(udpipe)

# Remove empty abstract
precision_agriculture_paper_clean <- precision_agriculture_paper %>%
  mutate(doc_id = EID) |>
  filter(!is.na(Abstract))

# Download and load an English UDPipe model
# udpipe_download_model(language = "english-ewt")
ud_model <- udpipe_load_model("english-ewt-ud-2.5-191206.udpipe")
```

Rule-based NER: Applications in R

We want to extract every occurrence of the pattern **"ADJ + engineer"** from the abstracts. To do this, we use **UDPipe**, which allows us to annotate the text and identify the part of speech of each word.

```
# Annotate abstracts
paper_pos <- udpipe_annotate(
  object = ud_model,
  x = precision_agriculture_paper_clean$Abstract,
  doc_id = precision_agriculture_paper_clean$doc_id
) |>
  as_tibble()
```

Rule-based NER: Applications in R

We want to extract every occurrence of the pattern **"ADJ + engineer"** from the abstracts. To do this, we use **UDPipe**, which allows us to annotate the text and identify the part of speech of each word.

```
# 4. Extract ADJ + engineer pattern
adj_engineer_hits <- paper_pos |>
  arrange(doc_id, sentence_id, token_id) %>%
  group_by(doc_id, sentence_id) |>
  mutate(
    previous_token = dplyr::lag(token),
    previous_lemma = dplyr::lag(lemma),
    previous_upos = dplyr::lag(upos),
    pattern = str_c(previous_token, token, sep = " ")
  ) %>%
  ungroup() %>%
  filter(
    previous_upos == "ADJ",
    str_to_lower(lemma) %in% c("engineer", "engineering")
  )
```

Rule-based NER: Applications in R

We want to extract every occurrence of the pattern **"ADJ + engineer"** from the abstracts. To do this, we use **UDPipe**, which allows us to annotate the text and identify the part of speech of each word.

```r
# Example
adj_engineer_hits |>
  select(pattern) |>
  head(5)
```

```
## # A tibble: 5 × 1
##   pattern
##   <chr>
## 1 agricultural engineering
## 2 chemical engineering
## 3 agricultural engineering
## 4 mechanical engineering
## 5 agricultural engineering
```

Rule-based NER: Applications in Python

The best way to implement a NER system based on rules is SpaCy

Rule-based NER: Applications in Python

Extract the **tasks** performed by the operator from the following sentences:

eseguito verifica immagini e regolata tempo esposizione immagine da 8 a 7.500 per rilevazione marcatore non reale testato 258 pezzi scarti 0

Rule-based NER: Applications in Python

Extract the **tasks** performed by the operator from the following sentences:

eseguito verifica immagini e **regolata tempo esposizione** immagine da 8 a 7.500 per rilevazione marcatore non reale **testato 258 pezzi** scarti 0

Rule-based NER: Applications in Python

Set the python environment.

```
#{r}
library(reticulate)

# Use the desired Python environment (adjust the path to your specific environment)
use_python("C:/Users/Vito Giordano/anaconda3/python.exe", required = TRUE)
py_config()
```

```
## python:         C:/Users/Vito Giordano/anaconda3/python.exe
## libpython:      C:/Users/Vito Giordano/anaconda3/python311.dll
## pythonhome:     C:/Users/Vito Giordano/anaconda3
## version:        3.11.4 | packaged by Anaconda, Inc. | (main, Jul  5 2023, 13:38:37) [MSC v.1916 64 bit
## Architecture:   64bit
## numpy:          C:/Users/Vito Giordano/anaconda3/Lib/site-packages/numpy
## numpy_version:  1.24.3
##
## NOTE: Python version was forced by use_python() function
```

Rule-based NER: Applications in Python

Extract the **tasks** performed by the operator from the following sentences:

eseguito verifica immagini e regolata tempo esposizione immagine da 8 a 7.500 per rilevazione marcatore non reale testato 258 pezzi scarti 0

```
import spacy
from spacy.matcher import Matcher

# Load the spaCy Model to perform PoS tagging and dependency parsing
nlp = spacy.load("it_core_news_lg")

text = nlp("eseguito verifica immagini e regolata tempo esposizione immagine da 8 a 7.500 per rile

# menage float value (when the string is null)
txt = str(text)

# Pre-process the text (Tokenization, PoS tagging, Dependency Parsing)
doc = nlp(txt)
```

Rule-based NER: Applications in Python

```
# Define the mathcer
matcher = Matcher(nlp.vocab)

# Define the pattern for "VERB + OBJECT" structure
# Match any verb followed by a direct object (dobj)
pattern_action = [
  {"TEXT": {"REGEX":"(\.*ito$)|(\.*ato$)|(\.*atto$)|(\.*ita$)|(\.*ata$)|(\.*atta$)|(\.*iti$)|(\.*a
  {'POS': "DET", "OP": "?"},
  {'POS': {'IN':['ADJ', 'NOUN','VERB']}},
  {'POS': "DET", "OP": "?"},
  {"POS": "NOUN", "OP": "*"},
  {"POS": "ADJ", "OP": "*"}, {"LEMMA": "di", "OP": "*"}, {"POS":
    {'IN':['ADJ','NOUN','DET']}, "OP": "*"}] # Tak object

# Add the pattern to the matcher
matcher.add("action_object", [pattern_action], greedy = "LONGEST")
```

Rule-based NER: Applications in Python

```
# Use the matcher on the doc
matches = matcher(doc)

# Initialize action text
action = []

# Extract the span
for match_id, start, end in matches:
  span = doc[start:end]
  action.append(span.text)

print(action)
```

```
## ['regolata tempo esposizione immagine', 'eseguito verifica immagini', 'rilevazione marcatore']
```

Prompt-based NER

Prompt-based NER

The steps to perform Prompt-based NER are:

1. **Define the Entity Types**: Specify the categories of entities to extract (e.g., persons, organizations, dates, products). Ensure the prompt explicitly mentions these types;
2. **Create a Prompt**: Formulate a natural language instruction to guide the model. For example: "Identify all names of people and organizations in the following text:";
3. **Input Text**: Provide the input text that contains the entities to be extracted. For example: "Dr. Smith works at OpenAI in San Francisco and has an appointment on January 15th."

Prompt-based NER

Install and Load package. You need your own API key.

```
# Import packages
import os
import openai

# Install openai module to run the API
# pip install openai

# Open AI api
openai.api_key = "XXXXXXXXXXXXXXXX"
```

Prompt-based NER

Define a function able to ask to chatGPT. It takes as input the **prompt**.

```
# Get the response object eith ChatGPT API
def get_ChatGPT_response(prompt):
  completion = openai.chat.completions.create(
      model="gpt-3.5-turbo",
      max_tokens=20,
      temperature=0,
      messages=[{"role": "user", "content": prompt}]
      )
  return completion.choices[0].message.content
```

Prompt-based NER

Write the prompt and the text of the sentences where you want to extract the entity.

In this case the entity that we want to extract is the **user** of a technology. So the prompt is *Who is the users of this sentence (extract the user only when it is explcitly mentioned in the sentence)*

The sentence is: "*The system provides a method for monitoring and analyzing real-time health data of patients, enabling early detection of medical conditions and personalized treatment recommendations.*"

```
# Define Prompt
prompt_ner = "Who is the users of this sentence (extract the user only when it is explcitly mentioned in the sentence):"

# Define text
text = " The system provides a method for monitoring and analyzing real-time health data of patients, enabling early detection of medical condi

# Merge togheter the prompt
prompt_to_run = prompt_ner + text
```

Prompt-based NER

Ask to chatGPT.

```
# Ask chatgpt
ChatGPT_response = get_ChatGPT_response(prompt_to_run)

print(ChatGPT_response)
```

```
## patients
```