
DataSpartan

Building a Semantic Search Engine with NLP

What is a semantic search engine? How does it work? How can I build one?

https://github.com / **rorisDS** / **workshop_semantic_search**

DataSpartan

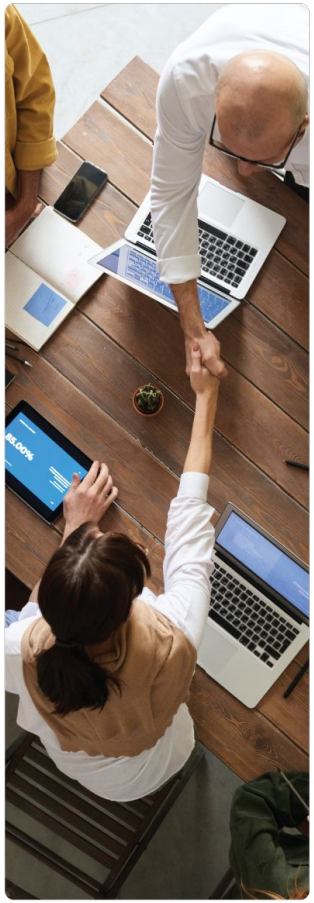
Experts in the fields of Artificial Intelligence (AI), Big Data, Quantitative Research, High-Performance Computing (HPC), Machine Learning (ML), and Risk Modeling.

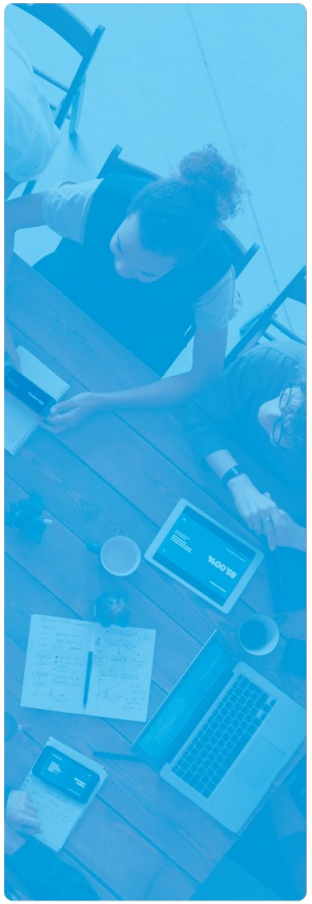
SMEs

ENTREPRENEURS

R&D Teams

We research, design, build, implement, and validate solutions to help companies reach their potential faster and more affordably than they could with their own resources alone.





Víctor Manuel Alonso Rorís

- Telecommunications Engineering (2003 - 2009)
- PhD in Telematics (2017)
  - *Semantic Technologies – Knowledge Representation*
  - *34 scientific publications (2011-2019)*

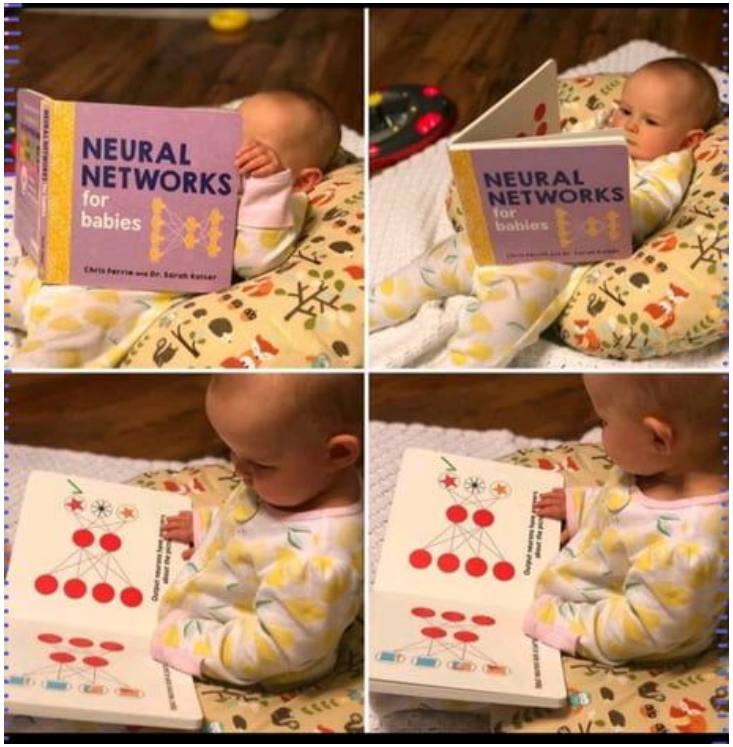
- Researcher at the University of Vigo (9 years)
- Data Scientist at DataSpartan (2018-present)
  - *Machine Learning Engineer – NLP specialist*

4 Workshop at UVigo's Foro de Emprego
  - *2022-2023: Building a Chatbot [link]*
  - *2024: LLMs [link]*
  - *2025: Building a Semantic Search Engine [link]*

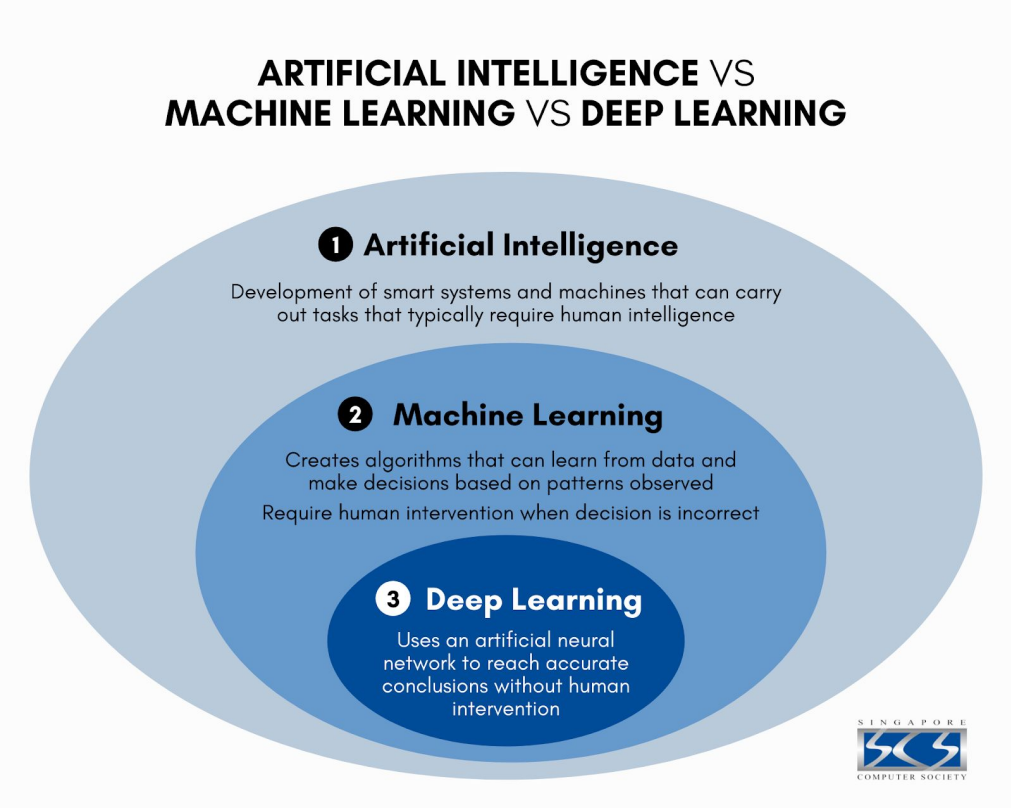
Contenidos

1. Basic Concepts of Machine Learning
2. What is NLP?
3. The Mathematics of Language
4. Building a Semantic Search Engine

Basic Concepts of Machine Learning



Definitions



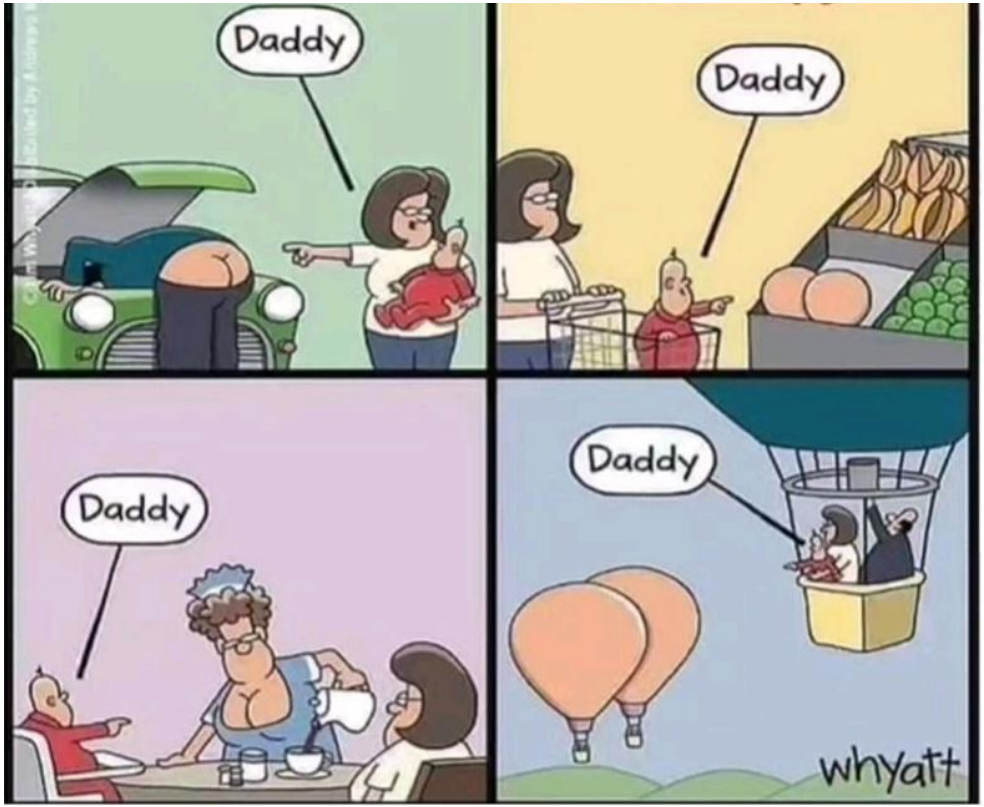
Supervised Learning

A type of ML that uses labeled data to train ML models.

Labeled data means that the expected output is known.

The algorithms measure their accuracy through prediction error, adjusting until the error is sufficiently minimized.

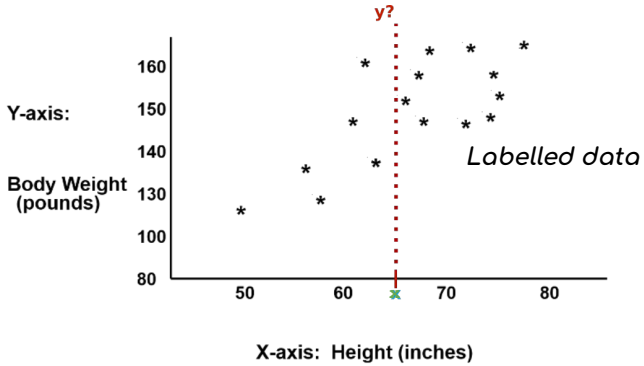
Supervised Learning



Simple Linear Regression

Predict a single output (***y***) using 1-dimensional input (***x***)

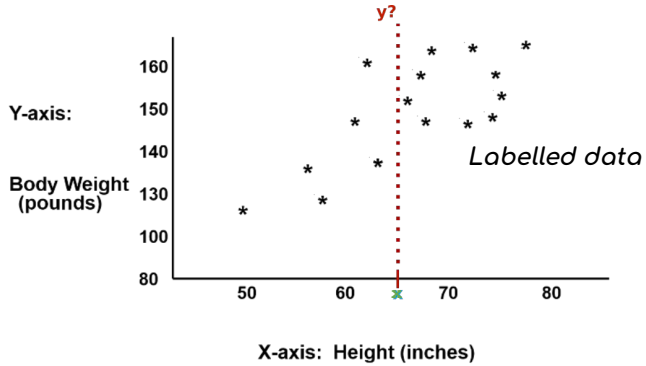
For example, predict body weight (***y***) from height (***x***)



Simple Linear Regression

Predict a single output (*y*) using 1-dimensional input (*x*)

For example, predict body weight (*y*) from height (*x*)

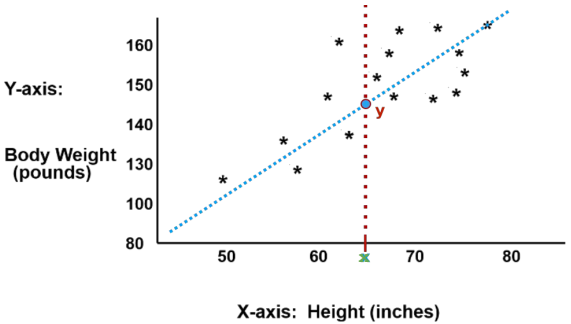


Linear Equation:

$$h(x) = m*x + b = y'$$

Where:

- *m* is the slope
- *b* is the y-intercept



Simple Linear Regression

1. Random value for ***m*** and ***b***
2. Introduce ***x*** into the function ⇒ predicts ***y'***
3. Calculate the error between ***y'*** and the expected ***y***
4. Adjusts ***m*** and ***b*** based on the error
   ⇒ Gradient descent
5. Repeats the process multiple times for each example in the training dataset

$h(x) = m*x + b = y'$

$h_0(x) = 2.3*x + 1 = y'$

$h_1(x) = 4.1*x + 5 = y'$

…

$h_M(x) = 3.032*x + 2.95 = y'$

Simple Linear Regression

1. Random value for ***m*** and ***b***
2. Introduce ***x*** into the function ⇒ predicts ***y'***
3. Calculate the error between ***y'*** and the expected ***y***
4. Adjusts ***m*** and ***b*** based on the error
   ⇒ Gradient descent
5. Repeats the process multiple times for each example in the training dataset

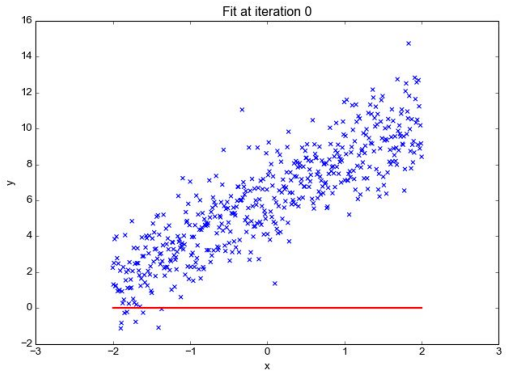
$h(x) = m*x + b = y'$

$h_0(x) = 2.3*x + 1 = y'$

$h_1(x) = 4.1*x + 5 = y'$

…

$h_M(x) = 3.032*x + 2.95 = y'$



Simple Linear Regression

1. Random value for ***m*** and ***b***
2. Introduce ***x*** into the function ⇒ predicts ***y'***
3. Calculate the error between ***y'*** and the expected ***y***
4. Adjusts ***m*** and ***b*** based on the error
   ⇒ Gradient descent
5. Repeats the process multiple times for each example in the training dataset

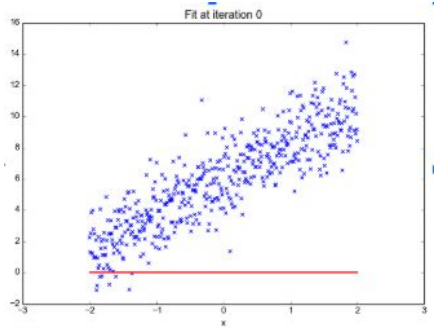
$h(x) = m*x + b = y'$

$h_0(x) = 2.3*x + 1 = y'$

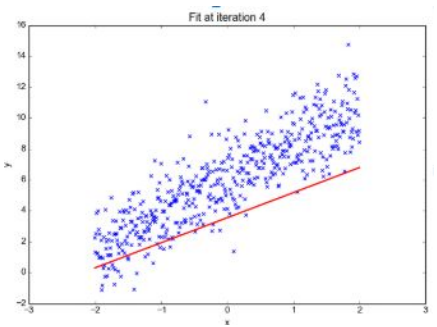
$h_1(x) = 4.1*x + 5 = y'$

…

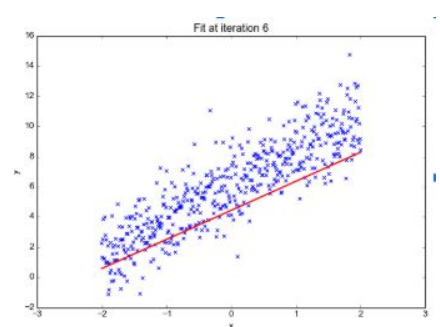
$h_M(x) = 3.032*x + 2.95 = y'$



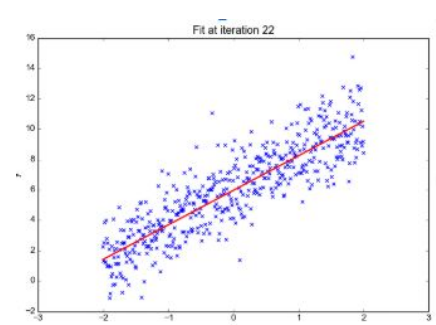
Iteration 0



Iteration 4



Iteration 6



Iteration 22

Multiple Linear Regression

n-dimensional input vector : $x = (x^1, x^2, x^3, \dots , x^N)$

$h(x) = \theta_0 * x_0 + \theta_1 * x_1 + \dots + \theta_N * x_N + b = y'$

Multiple Linear Regression

n-dimensional input vector : $x = (x^1, x^2, x^3, \dots , x^N)$

m-dimensional output : $y = (y^1, y^2, y^3, \dots , y^M)$

$$h_1(x) = \theta^1_0 * x_0 + \theta^1_1 * x_1 + \dots + \theta^1_N * x_N + b^1 = y_1'$$

$$h_2(x) = \theta^2_0 * x_0 + \theta^2_1 * x_1 + \dots + \theta^2_N * x_N + b^2 = y_2'$$

$$h_3(x) = \theta^3_0 * x_0 + \theta^3_1 * x_1 + \dots + \theta^3_N * x_N + b^3 = y_3'$$

$\dots$

$$h_M(x) = \theta^M_0 * x_0 + \theta^M_1 * x_1 + \dots + \theta^M_N * x_N + b^M = y_M'$$

Multiple Linear Regression

n-dimensional input vector : $x = (x^1, x^2, x^3, \dots , x^N)$

m-dimensional output : $y = (y^1, y^2, y^3, \dots , y^M)$

$$h_1(x) = \theta^1_0 * x_0 + \theta^1_1 * x_1 + \dots + \theta^1_N * x_N + b^1 = y_1'$$

$$h_2(x) = \theta^2_0 * x_0 + \theta^2_1 * x_1 + \dots + \theta^2_N * x_N + b^2 = y_2'$$

$$h_3(x) = \theta^3_0 * x_0 + \theta^3_1 * x_1 + \dots + \theta^3_N * x_N + b^3 = y_3'$$

$$\dots$$

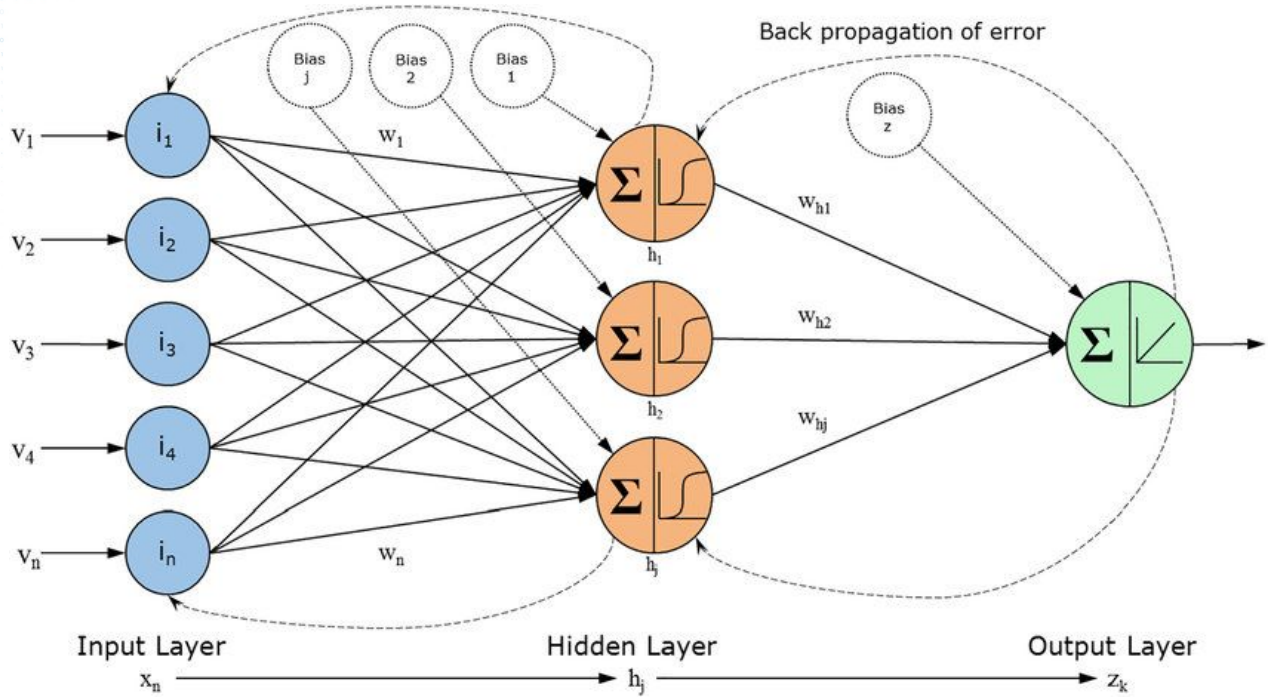
$$h_M(x) = \theta^M_0 * x_0 + \theta^M_1 * x_1 + \dots + \theta^M_N * x_N + b^M = y_M'$$

$$\begin{bmatrix} x_0 & x_1 & x_2 & \dots & x_N \end{bmatrix}$$

$$\begin{bmatrix} \theta^1_0 & \theta^1_1 & \dots & \theta^1_N \\ \theta^2_0 & \theta^2_1 & \dots & \theta^2_N \\ \theta^3_0 & \theta^3_1 & \dots & \theta^3_N \\ & \dots & & \\ \theta^M_0 & \theta^M_1 & \dots & \theta^M_N \end{bmatrix}^t + \begin{bmatrix} b^0 \\ b^1 \\ b^2 \\ \dots \\ b^M \end{bmatrix}$$

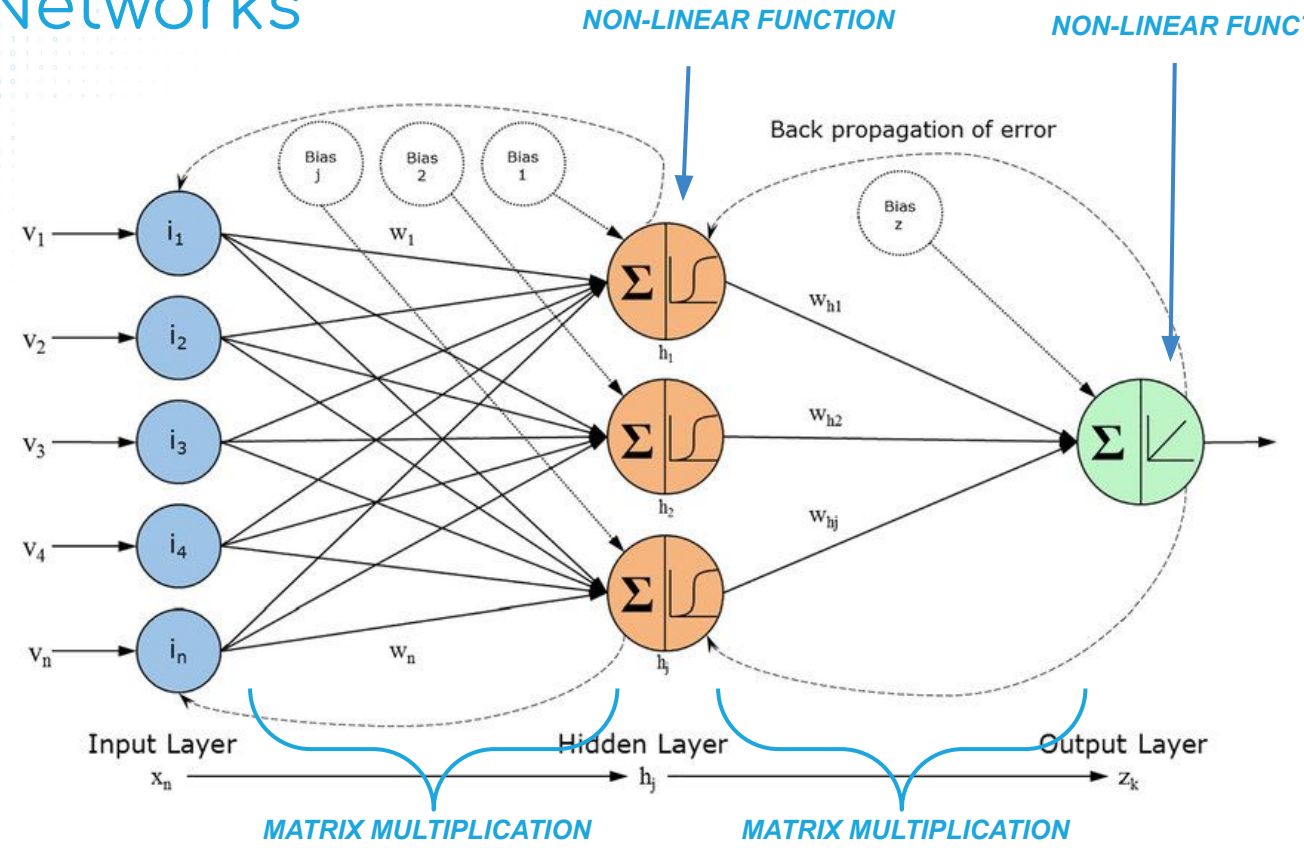
MATRIX MULTIPLICATIONS

Neural Networks



source

Neural Networks



Neural Networks - Let's Play

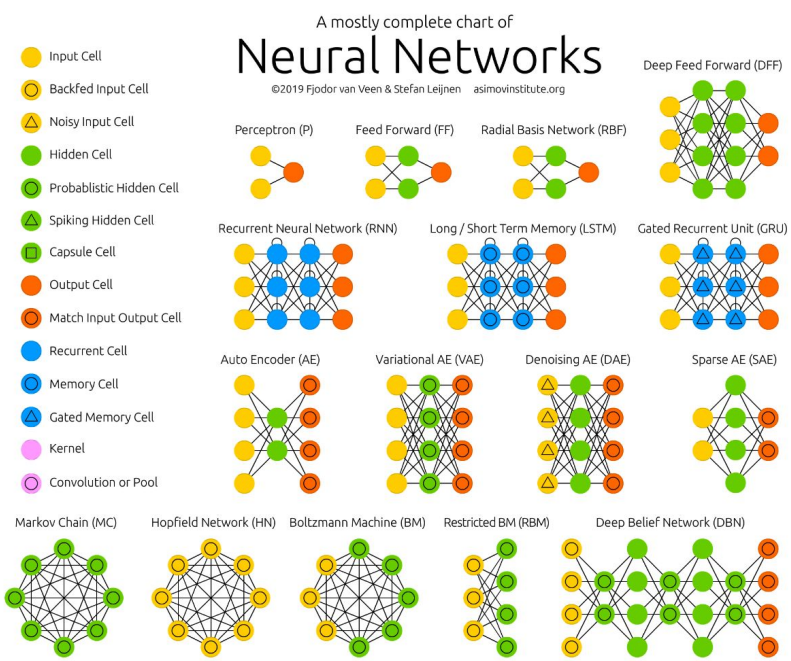
Let's play

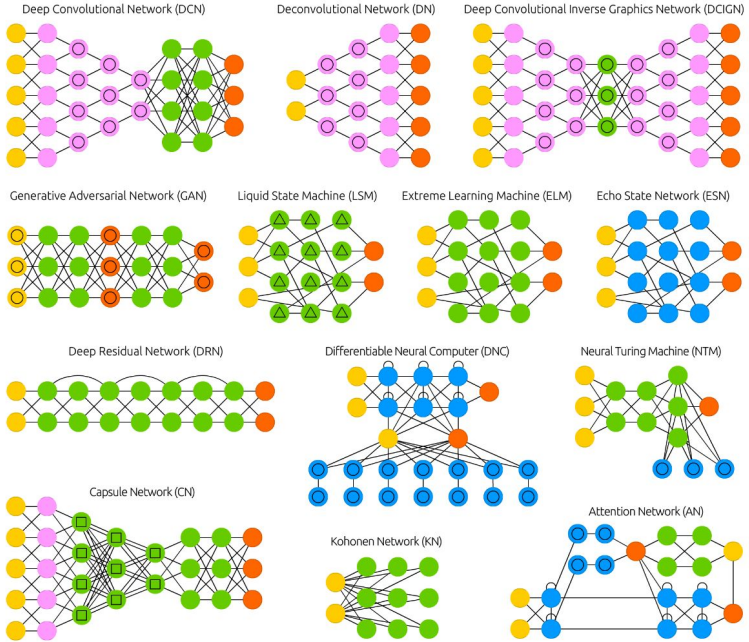
We define a simple model (*neural network*) using PyTorch

```python
import torch

class TinyModel(torch.nn.Module):

    def __init__(self):
        super(TinyModel, self).__init__()

        # Layer 1
        self.linear1 = torch.nn.Linear(100, 200)  # Matrix of size 100x200
        self.activation = torch.nn.ReLU()         # Non-linear function

        # Layer 2
        self.linear2 = torch.nn.Linear(200, 10)   # Matrix of size 200x10
        self.softmax = torch.nn.Softmax(dim=1)    # Non-linear function

    def forward(self, x):
        # Where x is a vector of size: 1x100
        x = self.linear1(x)      # Matrix multiplication: 1x100 * 100x200 = 1x200
        x = self.activation(x)  # f(1x200) => 1x200
        x = self.linear2(x)     # Matrix multiplication: 1x200 * 200x10 = 1x10
        x = self.softmax(x)    # f(1x10) => 1x10
        return x
```

Notebook link

Neural Network Architectures





source

Machine Learning Concepts

Mathematical operations ⇒ Pattern identification

No consciousness! Sorry, Sarah Connor!

Behavior modeled by training data

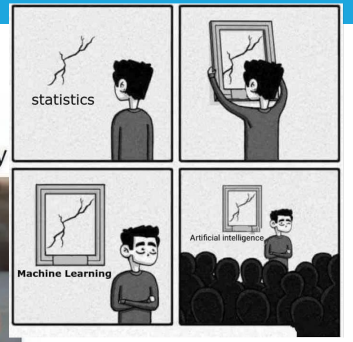
Racist or sexist models ⇒ Your data sucks!

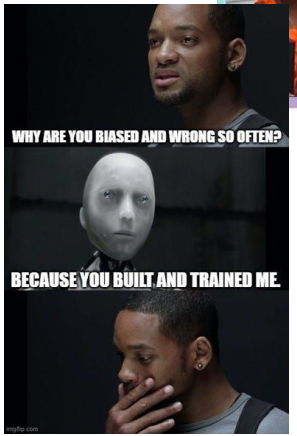
Libraries abstract the math: *Pytorch*, *Tensorflow*, *Keras*, Theano, …

People much smarter than me!

Want to learn more?

- *Coursera Course*
- *Google Course*
- *FastAI Course*



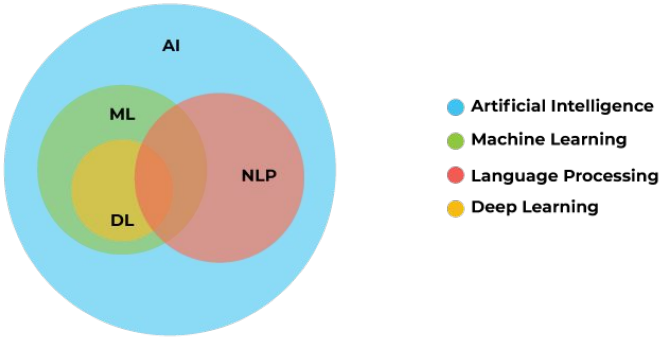


What is NLP?

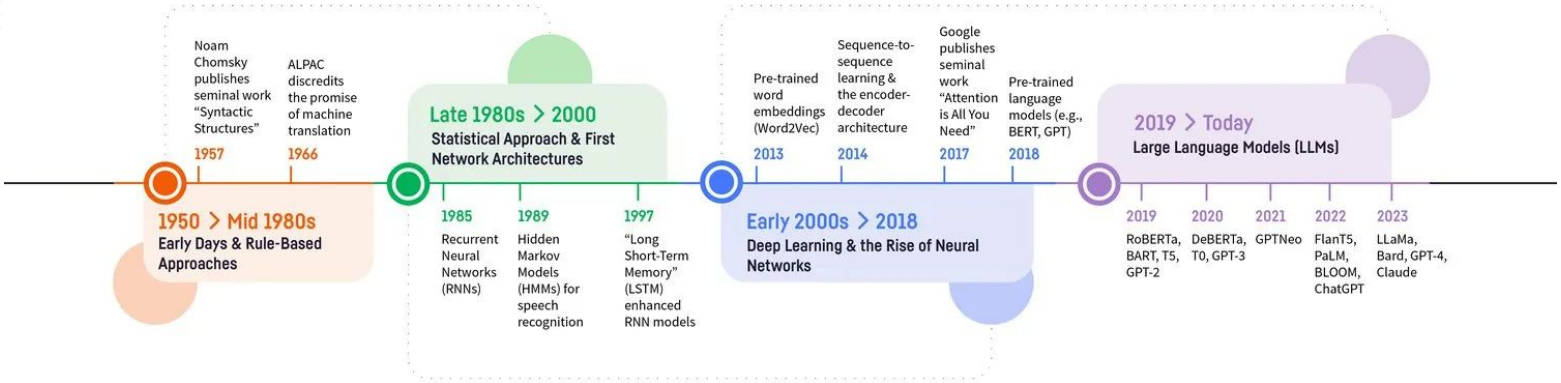
Natural Language Processing

Natural Language is the spoken or written language used by humans for general communication purposes ⇒ ***Ambiguous and unstructured***

Natural Language Processing (NLP) is the branch of AI responsible for enabling computers to understand texts in natural language.

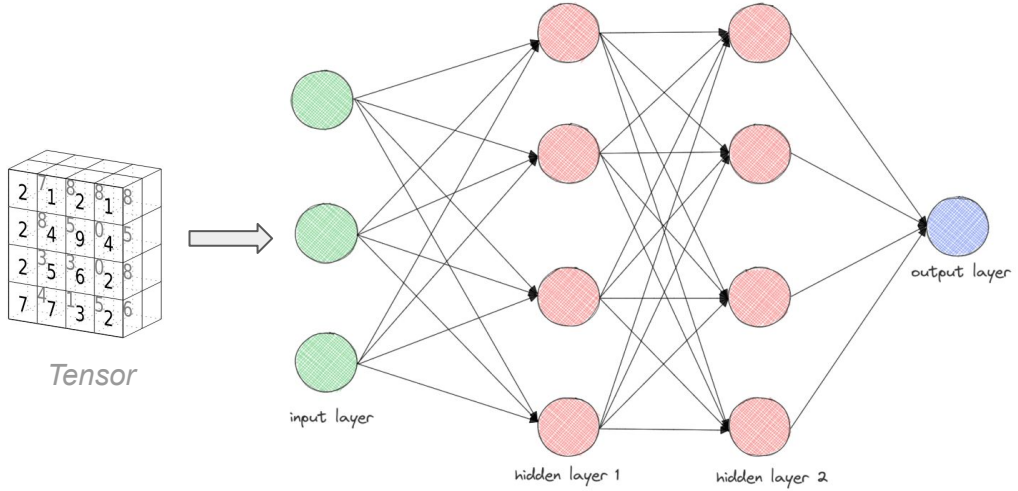


The history of NLP



Neural NLP

ML models use tensors as input ⇒ Numerical vectors

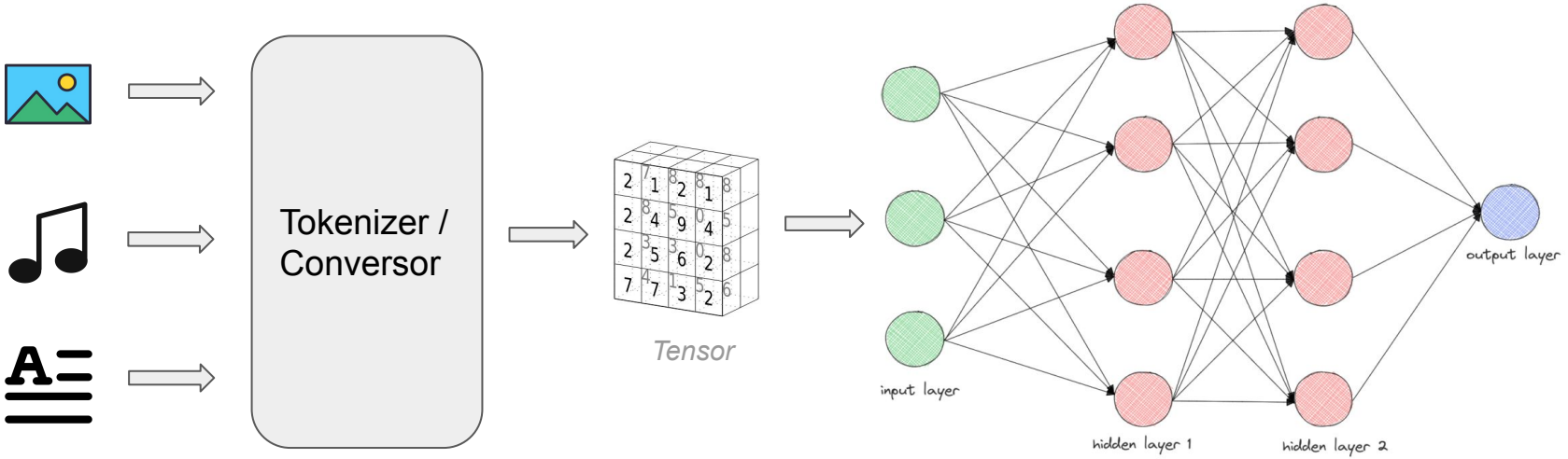


Neural NLP

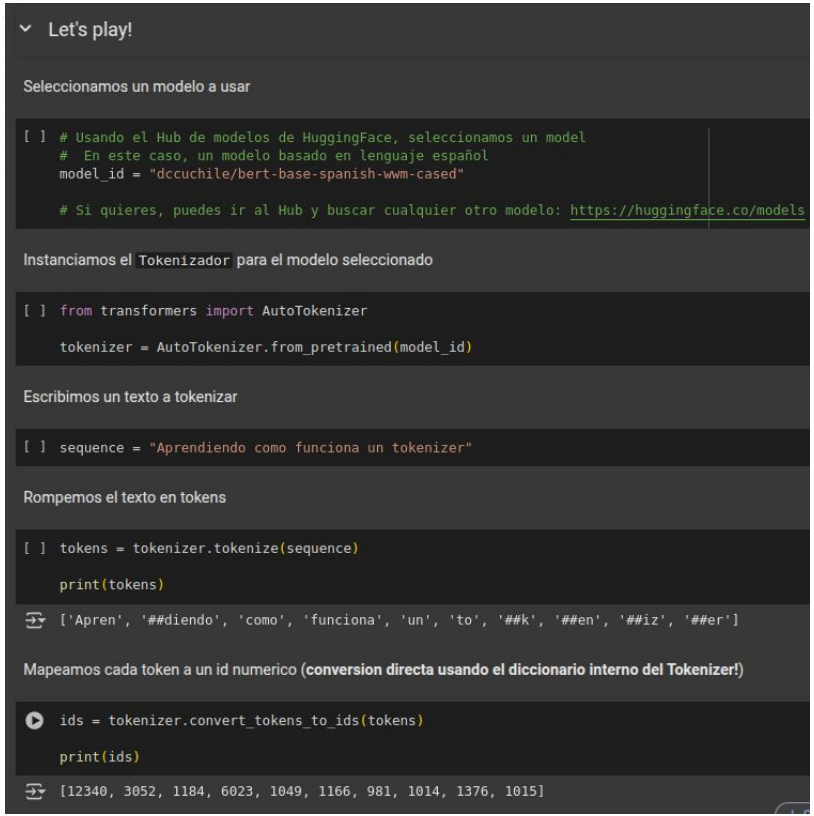
ML models use tensors as input ⇒ Numerical vectors

What about models that take images, audio, or **text** as input?

⇒ The input needs to be converted to a tensor (features) first.



Neural NLP - Let's Play



Notebook link

NLP Explosion

2017 - Google publishes "Attention is all you need"

⇒ *The Transformers ML architecture is introduced*

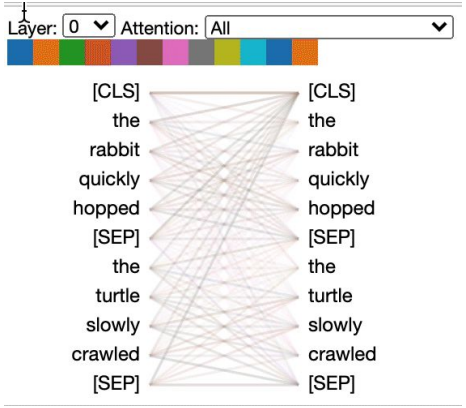
2018 - Pre-trained Language models ⇒ Fine-tuning

⇒ *ULMFit, Bert*

⇒ *Much smaller training datasets required*

2022 - OpenAi announces chatGPT

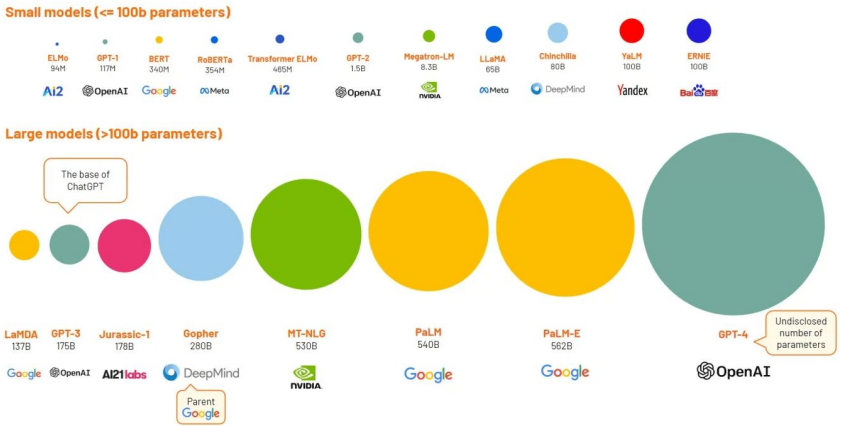
⇒ *Popularizes the term LLM (Large Language Model)*



LLMs Definition

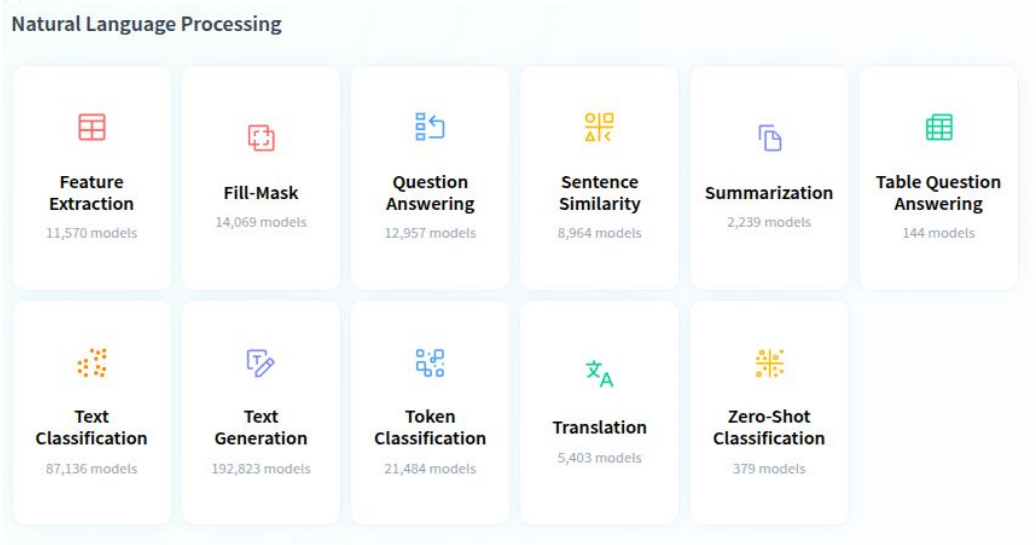
Large Language Model (LLM) is:

- An ML model for NLP based on the Transformers architecture
- Contains millions of parameters (known as: *m*, *θ*, *b*, weights, ...),
- Trained on a massive corpus of text
- Designed for general-purpose tasks but can be fine-tuned for specific NLP tasks or domains



source

NLP Tasks



https://huggingface.co/tasks

Tareas NLP - Let's Play

Let's play

A continuación, se presentan algunos ejemplos de código para realizar diversas tareas de Procesamiento del Lenguaje Natural (NLP) utilizando la biblioteca Transformers de Hugging Face. Para ello, se emplearán modelos disponibles en el Hugging Face Model Hub, explorando distintas formas de realizar predicciones mediante modelos, tokenizers y pipelines.

Traduccion

En este ejemplo, vamos a utilizar el directamente `modelo` y el `tokenizer` para generar la prediccion.

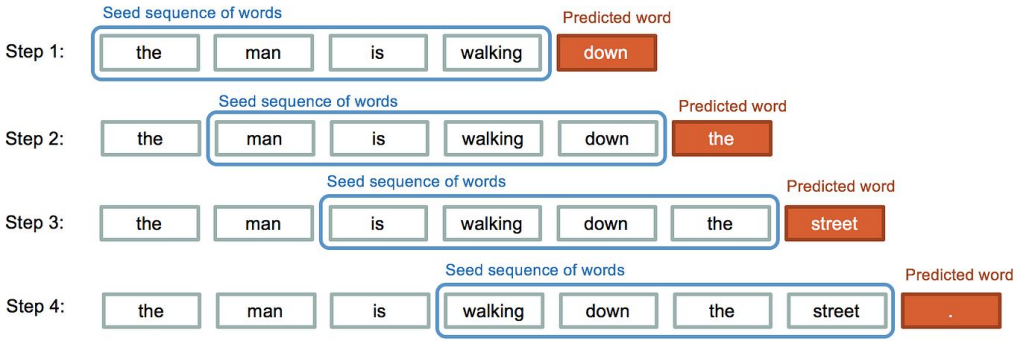
```
[57] from transformers import AutoModelForSeq2SeqLM
     from transformers import AutoTokenizer

     # Modelo disponible a través del hub de hugginface:
     #   https://huggingface.co/Helsinki-NLP/opus-mt-en-es
     model_key = "Helsinki-NLP/opus-mt-en-es"

     model = AutoModelForSeq2SeqLM.from_pretrained(model_key)
     tokenizer = AutoTokenizer.from_pretrained(model_key)
     model.eval()

     sentence = 'flowers are white'
     input_ids = tokenizer(sentence, return_tensors='pt')
     outputs = model.generate(**input_ids)
     # outputs = miak xochitl istak
     outputs = tokenizer.batch_decode(outputs, skip_special_tokens=True)[0]
     print(outputs)

     las flores son blancas
```

Notebook link

NLP Tasks: Text Generation

Generates new text from a given input

⇒ Predicts the next most likely word (or end of text)



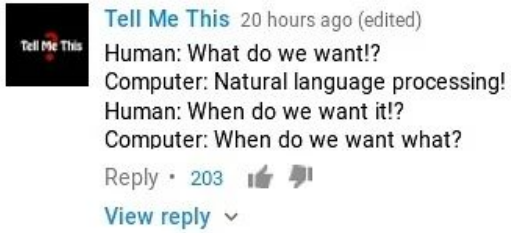
source

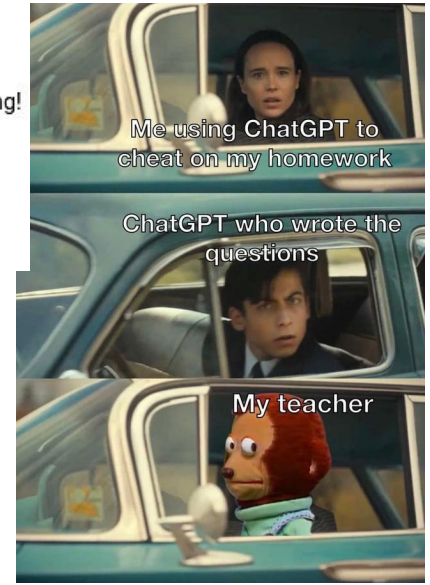
Training data is key to the model's performance

⇒ It doesn't mean it copies texts!

Should writers, screenwriters, or journalists be worried? Where does the training data come from? Is it lawful to use internet texts for training? Is it ethical to replace jobs with text-generation models?

NLP Concepts





NLP explosion due to the use of ML

⇒ Exploiting semantic relationships in language

LLM are massive models

⇒ They demand computational resources

Your laptop is not ready for it!

NLP Libraries: Transformers de HuggingFace, Spacy, NLTK, ...

LLM Libraries: HuggingFace, LangChain

People much smarter than me!

Want to learn more about NLP?

- *HuggingFace Course*
- *Stanford Course*
- *DeepLearning.ai Course*

Want to learn more about LLMs?

- LLM explained
- Build a chatGPT from scratch

The Mathematics of Language

Language Semantics

Semantics of any language

⇒ They follow statistical rules

Masking Technique

⇒ Introduce noise into sentences and train the model to correct it

⇒ Self-supervised Dataset

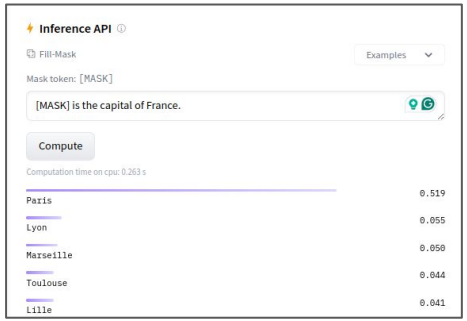
*The capital of France is **[MASK]***

***[MASK]**, the capital of France, is known as the city of lights*

*Madrid is to Spain what **[MASK]** is to France*

*During my visit to France, I have been in **[MASK]** visiting the Eiffel Tower*

*The great European capitals, Madrid, London, **[MASK]**, Rome, and Berlin, host …*



What if it were an extraterrestrial language, but you had millions of example texts?

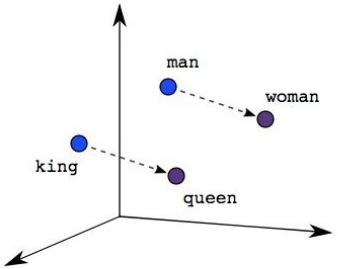
Embeddings

Numerical representation of words, phrases, text, documents, ... ⇒ vectors

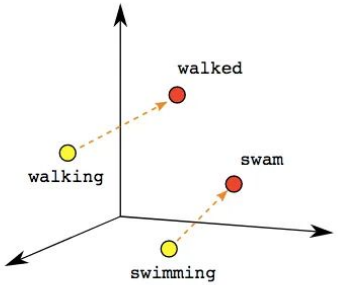
Word	Embedding			
abacus	0.19	-0.32	…	0.75
abandon	0.41	-1.27	…	-0.06
…			…	
zymosis	-0.76	-1.09		1.35

Do these embeddings have any meaning?

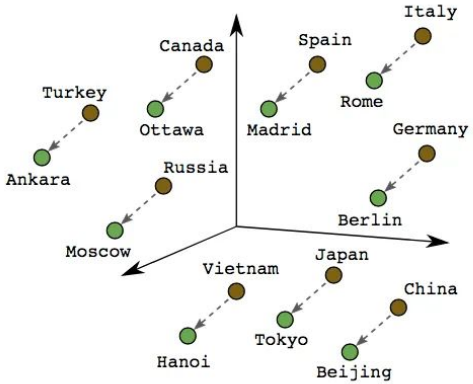
The Mathematics of Language



Male-Female

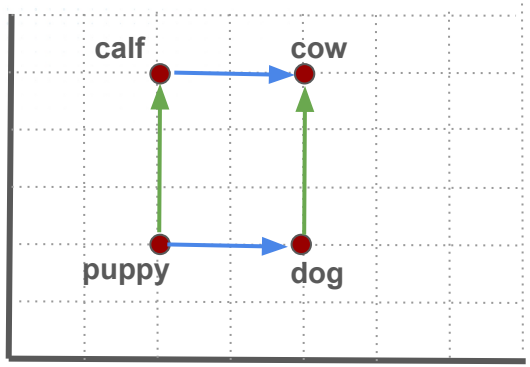


Verb Tense



Country-Capital

The Mathematics of Language

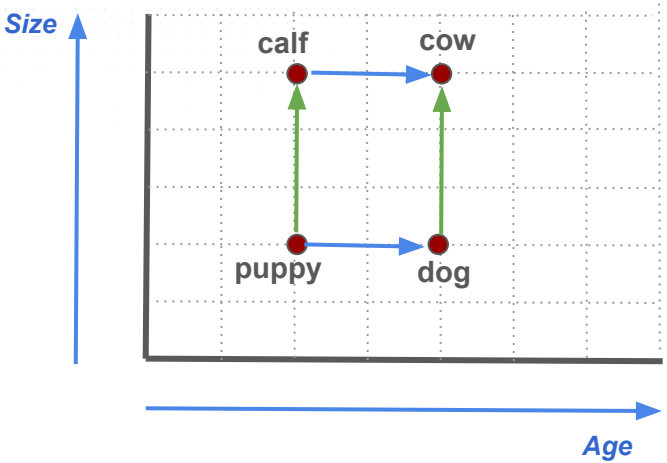


Analogy

*A **puppy** is to a **dog** as a **calf** is to a **cow***

*A **puppy** is to a **calf** as a **dog** is to a **cow***

The Mathematics of Language



Each axis represents a property

⇒ Embeddings (vector with multiple dimensions)

ML Models ⇒ automatic adjustment based on example texts

⇒ Property of each component is unknown

Dense Vectors

⇒ Information densely packed in each dimension

Where would you place a whale? And a piglet?

Sentence-Transformers - Let's play

Let's play

Comprobemos como poder obtener embeddings usando modelos de NLP.

```
[ ] # Modelo disponible a través del hub de hugginface:
    # https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2
    model_key = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"
```

```
[ ] # Frases a generar los embeddings
    sentences = ["Hola", "Buenos dias", "Adios"]
```

La obtención del embedding usando transformers es algo compleja, ya que implica varios cálculos sobre el vector de salida del modelo.

```
from transformers import AutoTokenizer, AutoModel
import torch

# Carga el modelo a partir del HuggingFace Hub
tokenizer = AutoTokenizer.from_pretrained(model_key)
model = AutoModel.from_pretrained(model_key)
model.eval()

# Tokeniza las frases
encoded_input = tokenizer(sentences, padding=True, truncation=True, return_tensors='pt')

# Calcula los embeddings de los token
model_output = model(**encoded_input)

#Mean Pooling - Take attention mask into account for correct averaging
def mean_pooling(model_output, attention_mask):
    token_embeddings = model_output[0] #First element of model_output contains all token embeddings
    input_mask_expanded = attention_mask.unsqueeze(-1).expand(token_embeddings.size()).float()
    return torch.sum(token_embeddings * input_mask_expanded, 1) / torch.clamp(input_mask_expanded.sum(1), min=1e-9)

# Agrupa los embeddings de los tokens para obtener el embedding de las frases
sentence_embeddings = mean_pooling(model_output, encoded_input['attention_mask'])

print("Embeddings de las frases:")
print(sentence_embeddings)

sentence_embeddings.shape
```

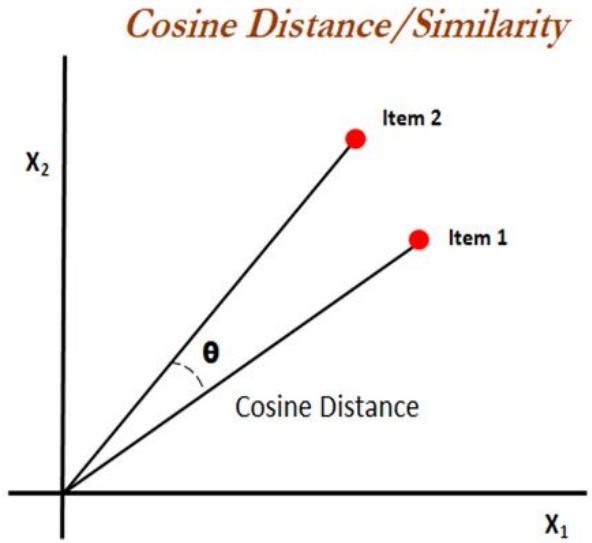
[Notebook link]

Embeddings Distance

Several techniques for calculating the distance or similarity between points in space

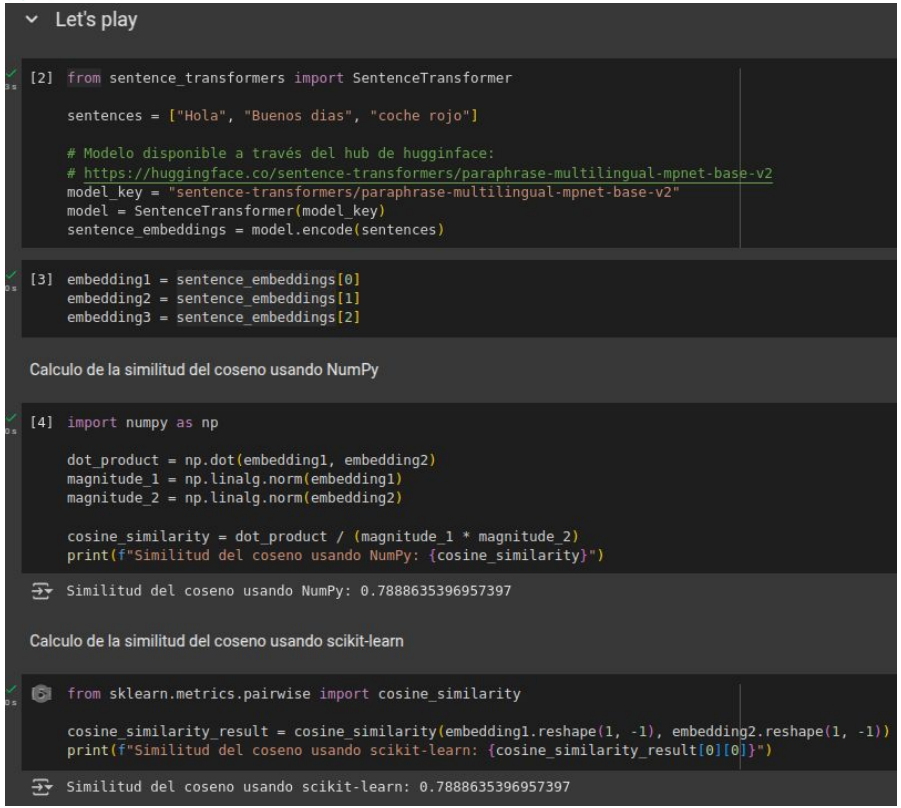
⇒ **Cosine similarity** standard in NLP

- **Cosine distance**: Measures the angular difference between two vectors, ignoring the magnitude.
- **Cosine Similarity**: A normalized variant that returns values between -1 and 1.



$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}},$$

Embeddings Distance - Let's play



Let's play

```
[2] from sentence_transformers import SentenceTransformer

    sentences = ["Hola", "Buenos dias", "coche rojo"]

    # Modelo disponible a través del hub de hugginface:
    # https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2
    model_key = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"
    model = SentenceTransformer(model_key)
    sentence_embeddings = model.encode(sentences)
```

```
[3] embedding1 = sentence_embeddings[0]
    embedding2 = sentence_embeddings[1]
    embedding3 = sentence_embeddings[2]
```

Calculo de la similitud del coseno usando NumPy

```
[4] import numpy as np

    dot_product = np.dot(embedding1, embedding2)
    magnitude_1 = np.linalg.norm(embedding1)
    magnitude_2 = np.linalg.norm(embedding2)

    cosine_similarity = dot_product / (magnitude_1 * magnitude_2)
    print(f"Similitud del coseno usando NumPy: {cosine_similarity}")
```

```
Similitud del coseno usando NumPy: 0.7888635396957397
```

Calculo de la similitud del coseno usando scikit-learn

```
from sklearn.metrics.pairwise import cosine_similarity

cosine_similarity_result = cosine_similarity(embedding1.reshape(1, -1), embedding2.reshape(1, -1))
print(f"Similitud del coseno usando scikit-learn: {cosine_similarity_result[0][0]}")
```

```
Similitud del coseno usando scikit-learn: 0.7888635396957397
```

[Notebook link]

NLP and Embeddings Concepts

NLP explosion due to the use of ML

⇒ Exploiting semantic relationships in language

Represent text as embeddings

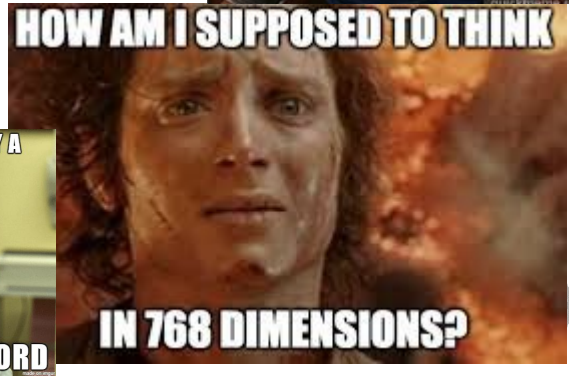
⇒ They encapsulate the semantic relationships of the text

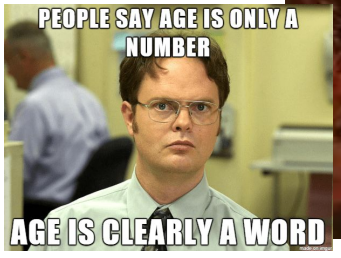
NLP Libraries: HuggingFace, Sentence-Transformers, ...

People much smarter than me!

Want to learn more?

- *Google Course*
- *DeepLearning.ai Course*



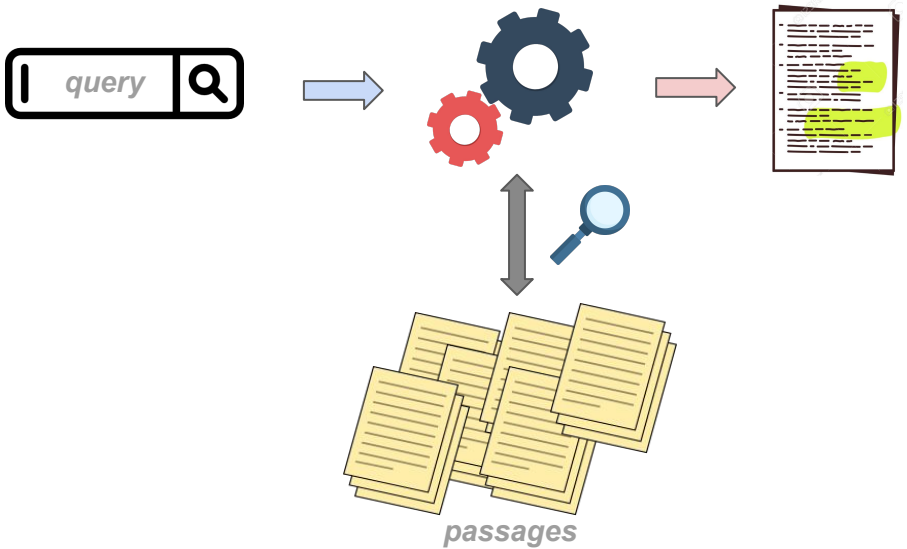


Building a Semantic Search Engine

Logic of a semantic search engine

Search engine:

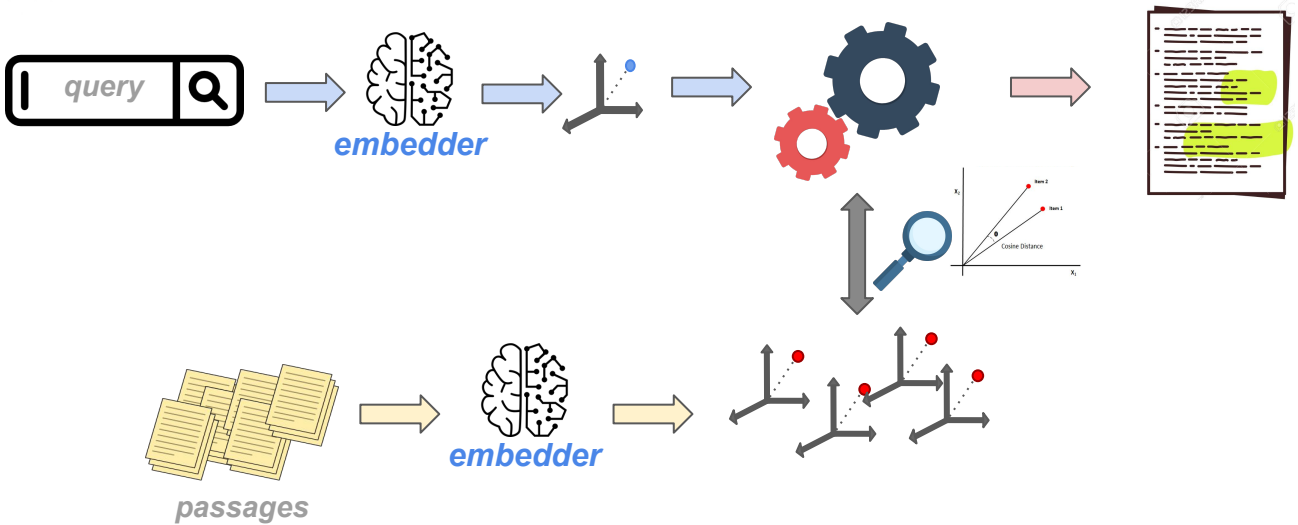
- Identify texts (**passages**) that are related to a **query** text



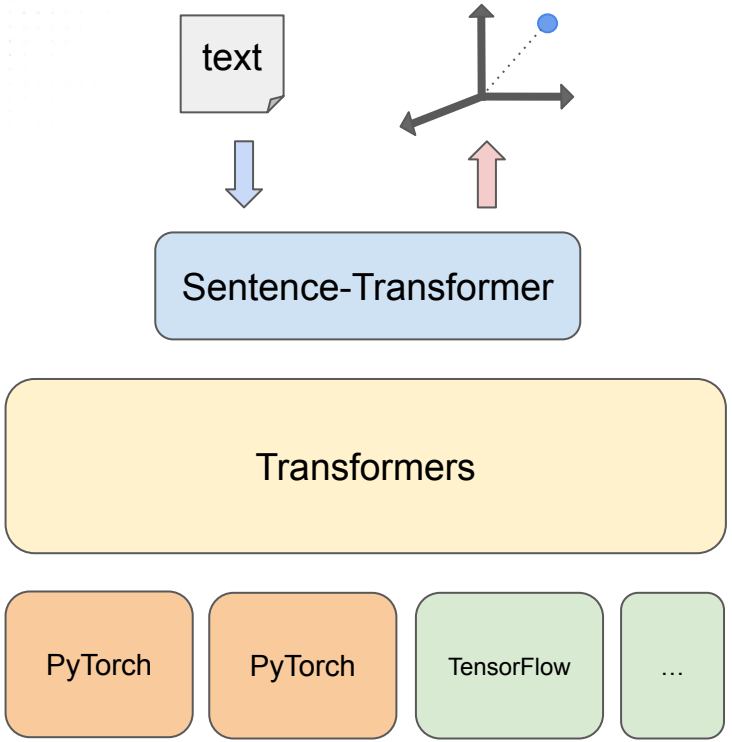
Logic of a semantic search engine

Semantic search engine:

- Identify embeddings similar/close to the query embedding
  - Convert the reference texts (**passages**) to embeddings
  - Convert the query text (**query**) to an embedding



Embedder technology stack



Embedding management abstraction

NLP model management abstraction

NLP Models

Architecture and value of the weights

Vector Databases

Semantic search engine:

⇒ Millions of passages ⇒ Millions of embeddings

⇒ An immense amount of calculations **(embedding distance)** per query

Vector Databases:

Vector indexing and similarity search ⇒ **Dense Search**

Examples: Faiss, Annoy, NMSLIB, Milvus, Pinecone, Zilliz, ...

Faiss *is a library for* ***efficient*** *similarity search and clustering of dense vectors. It contains algorithms that search in sets of* ***vectors of any size****, up to ones that possibly do not fit in RAM. It also contains supporting code for evaluation and parameter tuning. Faiss is written in C++ with complete wrappers for Python/numpy.*

Vector Databases - Let's play

Let's play

```
[26] from sentence_transformers import SentenceTransformer
     from typing import List

     class Embedder:

       def __init__(self, model_key: str = None):
         if model_key is None:
           # Modelo disponible a través del hub de hugginface:
           # https://huggingface.co/sentence-transformers/paraphrase-multilingual-mpnet-base-v2
           model_key = "sentence-transformers/paraphrase-multilingual-mpnet-base-v2"
         self.model = SentenceTransformer(model_key, trust_remote_code=True)

       def embed_passages(self, passages: List[str]):
         return self.model.encode(passages, convert_to_numpy=True)

       def embed_query(self, query: str):
         # Some models can have a different function to encode queries!
         return self.model.encode(query, convert_to_numpy=True)

     embedder = Embedder()
```

```
[27] # Indexar passages
     passages = [
         # Ruido
         "hola", "buenos dias", "coche rojo", "workshop", "la estufa esta encendida",
         "Pienso, luego existo",
         # Textos sobre comida
         "la comida del campus sabe a carton",
         "comer en un restaurante de estrella michelin es digno de estudio",
         "la comida de mi madre deberia estudiarse en la universidad"
     ]

     passage_embeddings = embedder.embed_passages(passages)
```

Construimos el indice

```
import faiss

# dimension de los embeddings
d = passage_embeddings[0].shape[0]
print(f"Dimension de los embeddings: {d}")

# Construimos un indice para embeddings de dimension d
index = faiss.IndexFlatL2(d)
```

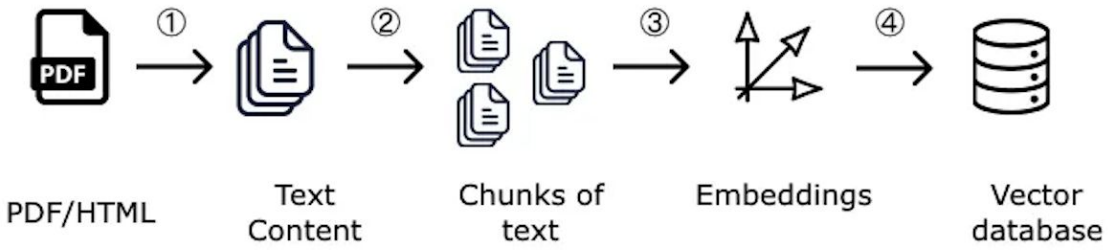
```
Dimension de los embeddings: 768
```

[Notebook link]

Business logic - Indexing

Ingestion phase (ETL):

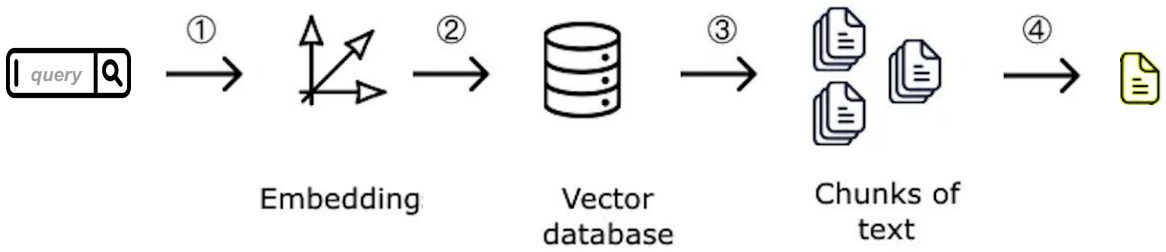
1) Documents to be indexed ⇒ extract text
2) **Chunking** ⇒ split large texts into fragments (**passages**)
3) Convert each **passage** to **embedding**
4) Index the **embeddings** in a vector database



Lógica de negocio - Search

Search phase:

1) Convert **query** to **embedding**
2) Search by **embedding** in the vector database
3) Identify the most similar **passages** to the **query**



Business logic - Let's play

```
[31] import requests
     import fitz  # PyMuPDF
     import os
     import faiss
     import numpy as np
     from sentence_transformers import SentenceTransformer

     def download_pdf(url, output_path: str):
         """Descarga un PDF de una url"""
         response = requests.get(url)
         with open(output_path, "wb") as f:
             f.write(response.content)
         print(f"PDF descargado: {output_path}")

     def pdf_to_text(pdf_path: str) -> str:
         """Convierte el contenido de un PDF a texto"""
         doc = fitz.open(pdf_path)
         text = "\n".join([page.get_text("text") for page in doc])
         return text

     def split_text(text: str, chunk_size: int=200) -> list:
         """Rompe el texto en trozos"""
         words = text.split()
         return [" ".join(words[i:i+chunk_size]) for i in range(0, len(words), chunk_size)]

     def create_embeddings(passages: list, model_name="sentence-transformers/all-MiniLM-L6-v2"):
         """Convierte textos a embeddings"""
         model = SentenceTransformer(model_name)
         embeddings = model.encode(passages, convert_to_numpy=True)
         return embeddings

     def store_embeddings(embeddings, index_path="faiss_index.bin"):
         """Indexa embeddings en Faiss"""
         dimension = embeddings.shape[1]
         index = faiss.IndexFlatL2(dimension)
         index.add(embeddings)
         faiss.write_index(index, index_path)
         print(f"Embeddings guardados en {index_path}")
```

```
# 0. Obtengo los documentos a procesar
pdf_url = "https://d1.awsstatic.com/aws-analytics-content/OReilly_book_Natural-Language-and-Search_web.pdf"  # URL de ejemplo
pdf_path = "documento.pdf"
download_pdf(pdf_url, pdf_path)

# 1. Extraccion de texto
text = pdf_to_text(pdf_path)

# 2. Division en fragmentos
passages = split_text(text)

# 3. Genera embeddings
embeddings = create_embeddings(passages)

# 4. Guarda en la base de datos de vectores
index_path = "faiss_index.bin"
store_embeddings(embeddings, index_path)
print("Indexación completada.")
```

Notebook link

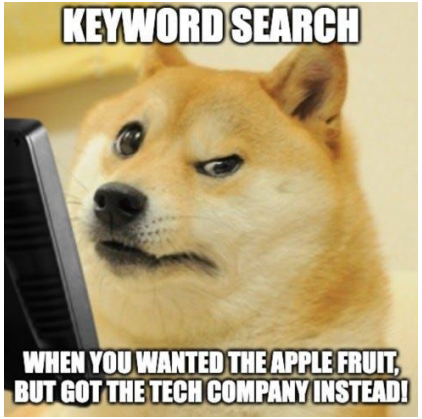
Advantages

Advantages over Traditional Search Engines:

- Understanding of Meaning and Context:
  - Does not search by keywords
  - Supports synonyms and semantic relationships *(e.g., search for "AI" and get docs about "ML")*

- Natural Language Search

- Scalability and Efficiency with Vector Databases



Applications

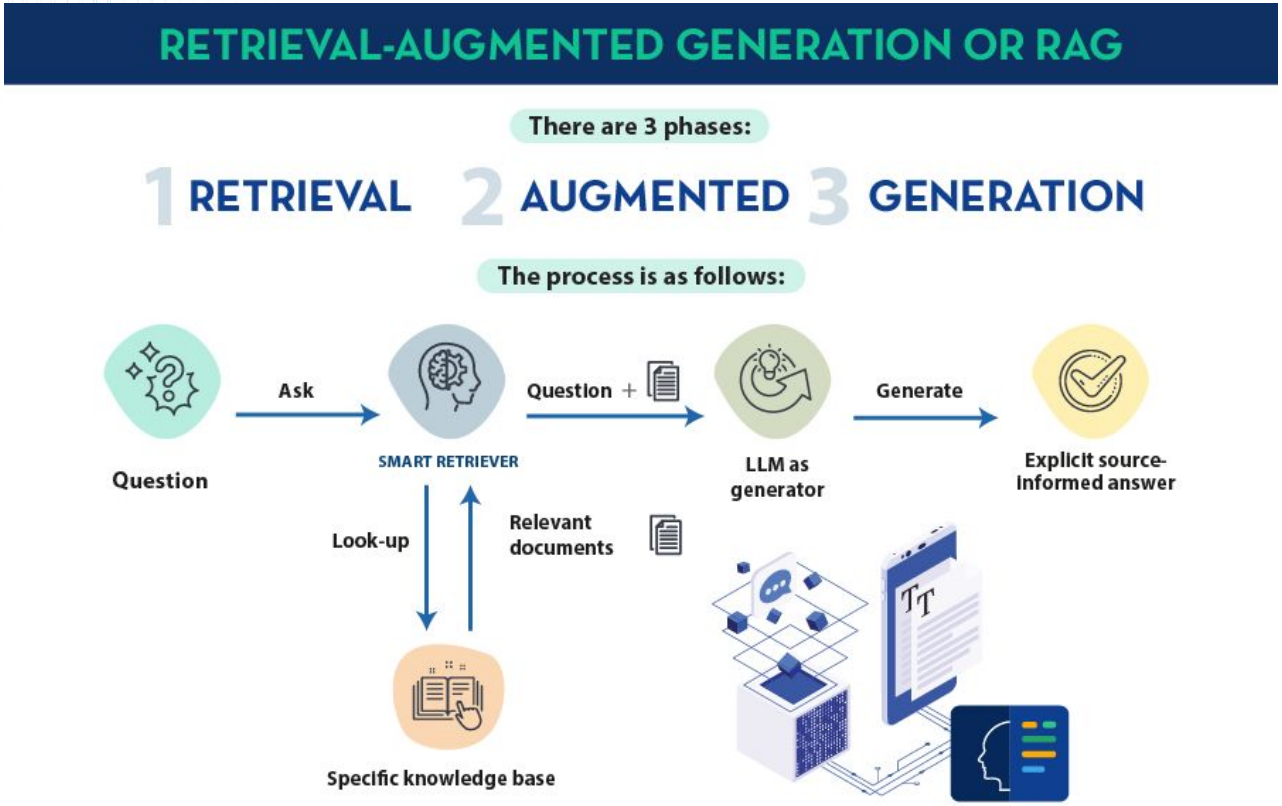
Applications across multiple industries, including:

- **Healthcare**: Finding medical studies and relevant articles for diagnoses.
- **Education**: Retrieving learning materials based on key concepts.
- **Legal**: Searching legal documents without requiring exact term matches.
- **Business**: Improving search capabilities in knowledge bases and customer support.

Integration with AI and Advanced Systems

⇒ RAG (Retrieval Augmented Generation)

RAG


RETRIEVAL-AUGMENTED GENERATION OR RAG
There are 3 phases:
1 RETRIEVAL
2 AUGMENTED
3 GENERATION
The process is as follows:
Question
Ask
SMART RETRIEVER
Question +
LLM as generator
Generate
Explicit source-informed answer
Look-up
Relevant documents
Specific knowledge base
Source: https://www.ml6.eu/blogpost/leveraging-llms-on-your-domain-specific-knowledge-base
datos.gob.es
reutiliza la información pública

RAG - Let's play

Let's play!

- OpenAI API KEY

Indica tu OpenAI API Key para poder usar la API de OpenAI (cómo obtener la API key: https://platform.openai.com/docs/quickstart)

```
[35] # Indica la API Key de OpenAI
     openai_api_key = "TU_API_KEY_DE_OPENAI"

     # --- Este proceso es para mi para no compartir mi token con todo el mundo mientras hago el workshop!!!
     from google.colab import userdata
     try:
       secret_openai_api_key = userdata.get('OPENAI_API_KEY')
     except userdata.SecretNotFoundError:
       secret_openai_api_key = None  # Assign None if secret is not found

     if secret_openai_api_key is not None:
       openai_api_key = secret_openai_api_key

     import os
     # LangChain toma el API KEY de las variables de entorno
     os.environ["OPENAI_API_KEY"] = openai_api_key
```

- Sin RAG

Lanzamos una consulta a OpenAI sin contexto

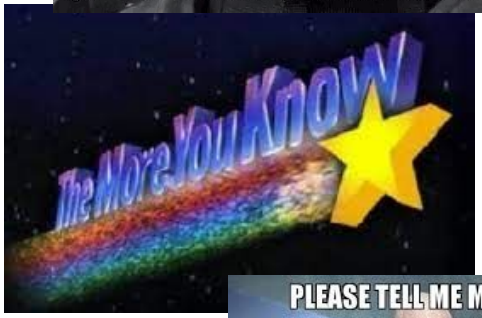
```
from langchain.chat_models import ChatOpenAI

chat = ChatOpenAI(
    model="gpt-3.5-turbo",
    temperature=0.0
)

# Prueba 1: Pregunta sobre un documento técnico
# query = "¿Cúal es la cuenta y password por defecto del router Livebox Fibra? Di 'No lo sé' si no sabes la respuesta"  # Si quieres forzar que no se invente la respuesta
query = "¿Cúal es la cuenta y password por defecto del router Livebox Fibra?"
response = chat.predict(query)
print("Pregunta:", query)
print("Respuesta del modelo:", response)
print("\n-----\n")
```

Notebook link

Know more

Want to learn more?

- *Embeddings, Models, Vector Databases and RAG*
- *LLM Embeddings Explained: A Visual and Intuitive Guide*
- *What are Vector Embeddings? An Intuitive Explanation*
- *Introduction to Embeddings & Vector Stores*
- *Embeddings & Vector Stores: A Beginner's Guide*
- *RAG with LangChain*
- *Vector Embeddings in RAG Applications*



DataSpartan

Rétanos hoy

informacion@dataspartan.com

www.es.dataspartan.com