💬

Introduction to Natural Language Processing

Table of contents

📌 **Reminders**

- If possible, please turn on your video camera
- We'll do live Q&A at the end
- We're keeping the programs light today, so there's more time for FAQ
- Please add your questions in the Zoom chat during the all-hands

▼ About me

- Industrial Engineer 👨‍💻
- PhD in experimental physics 👨‍🔬
- Data Scientist since 2015 👨🏼
- Leading NLP projects in different industries and companies
- Surfing the ML and data wave 🌊
- Always learning. Probably I should donate to wikipedia, stackoverflow, and get a chatGPT pro license 🤷

https://www.linkedin.com/in/juan-huguet-garcia-phd-60903076/

Materials

Github repository:

https://github.com/juanhuguet/intro_to_nlp

Recommended book (I have no commission !) :

Natural Language Processing with Transformers, Revised Edition

Since their introduction in 2017, transformers have quickly become the dominant architecture for achieving state-of-the-art results on a variety of natural language processing tasks. If you're a data scientist or … - Selection from Natural Language Processing with Transformers, Revised Edition [Book]

https://www.oreilly.com/library/view/natural-language-processing/9781098136789/

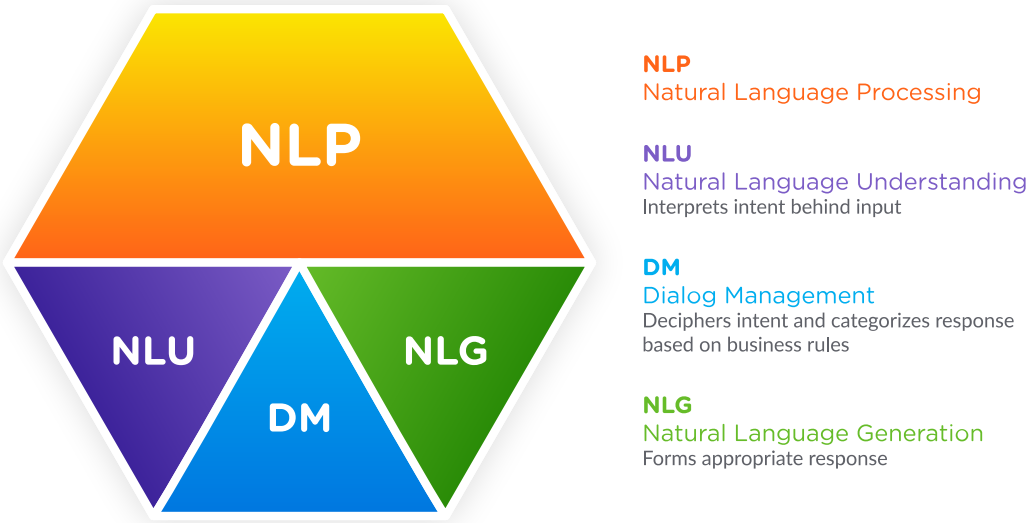
Introduction

▼ Introduction

Welcome to the Natural Language Processing workshop! This course is designed to introduce you to the exciting world of natural language processing and how it can be used in different applications. From text classification to question answering, this course will provide you with the necessary tools and knowledge to work with and manipulate natural language data. Join us on this journey as we explore the fascinating intersection of linguistics, computer science, and artificial intelligence.

👉 Natural Language Processing, or NLP, is the process of extracting the meaning, or intent, behind human language. In the field of Conversational artificial intelligence (AI), NLP allows machines and applications to understand the intent of human language inputs, and then generate appropriate responses, resulting in a natural conversation flow.

👉 Most commonly, NLP is used as an umbrella term to include Natural Language Understanding (NLU), Natural Language Generation (NLG), and Dialog Management.



- Natural Language Understanding
- Dialog Management:
- Natural Language Generation

(NLU):

- Syntax and Morphology
- Semantics
- Discourse
- Pragmatics
- Named Entity Recognition
- Sentiment Analysis

- Speech Recognition

(NLG):

- Machine Translation
- AI-powered transcription tools

source

▼ A gentle reminder on machine learning

> 🤯 A computer program is said to **learn** from experience *E* with respect to some class of tasks *T* and performance measure *P*, if its performance at tasks in *T*, as measured by *P*, improves with experience *E*. - T. Mitchel

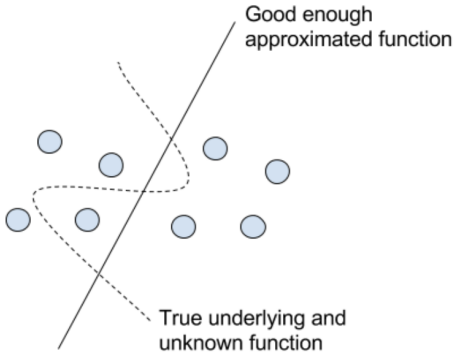
> 🧐 *The field of machine learning is concerned with the question of how to construct computer programs that automatically improve with experience - T. Mitchel*

Imagine a computer program that is designed to identify whether an image contains a cat or not. The program can learn from experience if it is given a large dataset of images with known labels (i.e., whether they contain a cat or not) and uses this dataset to improve its performance at the task of identifying cats in new images.

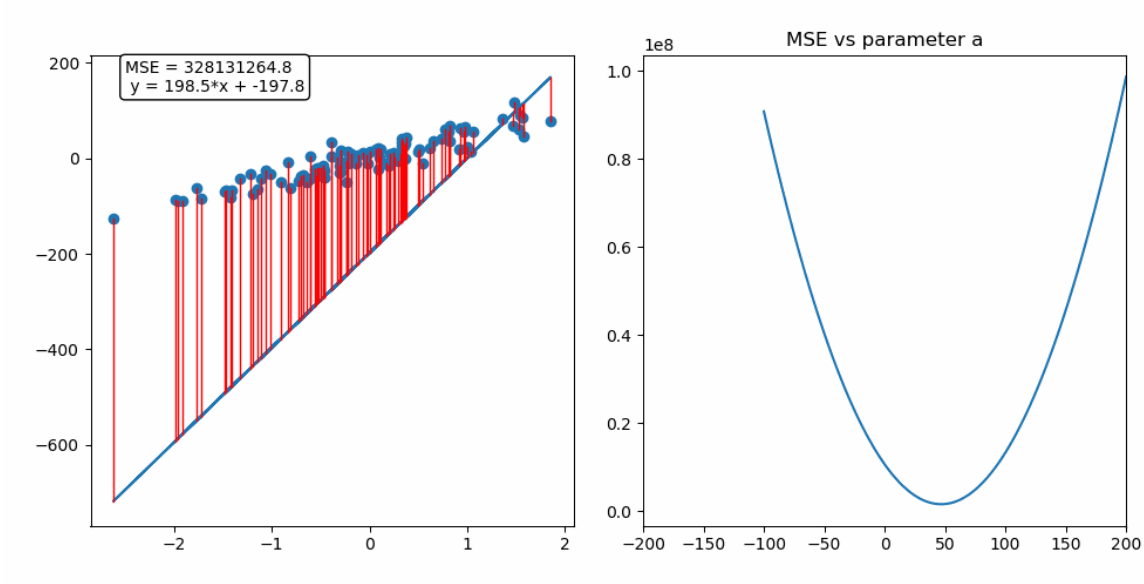
For example, the program may initially misidentify many images that contain cats as not containing cats, but as it gains more experience with the dataset, it learns to better distinguish between cat and non-cat images. The performance measure, P, could be the percentage of images correctly identified as containing a cat, and the class of tasks, T, would be image classification. The experience, E, would be the dataset of images with known labels that the program uses to learn.

Now, as we already know, there are several machine learning algorithms capable of learning from experience: from linear regression to decision trees ensembles and the underlying mechanism is always the minimisation of a cost function that tells us how wrong we. So “learning” is now “mathematical optimization”, and maths are, basically, done with numbers.

> 👉 We can understand a ML algorithm as **an algorithm that changes its internal state** to find the best **mapping** between an output variable and its input features.



$$\hat{y} = wx + b$$



⚠️ Machine Learning algorithms are by design to be used with numerical data representations and work by reducing a function that measures the error of the predictions

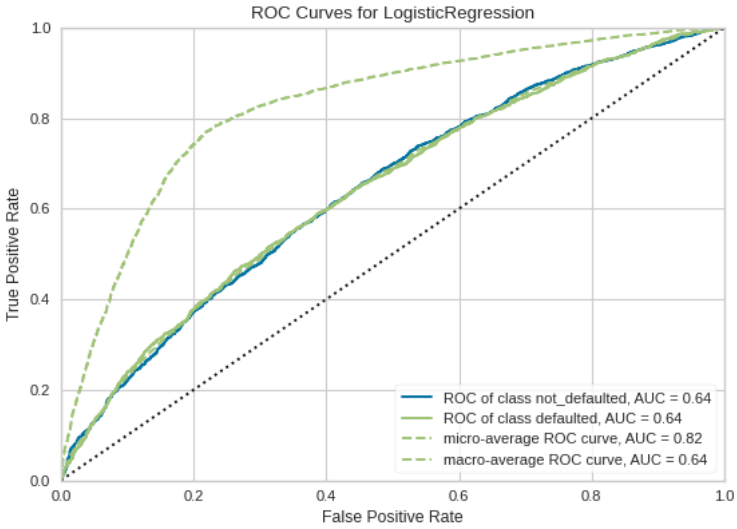
```
from yellowbrick.classifier.rocauc import roc_auc
from yellowbrick.datasets import load_credit

from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split

#Load the classification dataset
X, y = load_credit()

#Create the train and test data
X_train, X_test, y_train, y_test = train_test_split(X,y)

# Instantiate the visualizer with the classification model
model = LogisticRegression()
roc_auc(model,
        X_train,
        y_train,
        X_test=X_test,
        y_test=y_test,
        classes=['not_defaulted', 'defaulted']
        )
```



source

Notebook:

intro_to_nlp/01-basic-ml-classification-tabular-data.ipynb at 92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2 · juanhuguet/intro_to_nlp

Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.

https://github.com/juanhuguet/intro_to_nlp/blob/92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2/notebooks/01-basic-ml-classification-tabular-data.ipynb

▼ How do we make machines read text

We have seen that machine learning models are devised to be used with numerical data. We need then to find a numerical representation for textual data. Let's make a simple example for a sentiment classification.

▼ Simple sentiment classification

Let's say we have the next restaurant review:

> "Overall the restaurant is very good as the food was tasty and good even though the service was bad."

We could use a simple approach, which is, count how many positive or negative words we have and calculate the overall sentiment.

1. First, you need to choose a sentiment dictionary that maps words to sentiment scores. A popular choice is the AFINN lexicon, which assigns a score to words between -5 (very negative) and 5 (very positive).
2. Next, you need to count the occurrence of each word in your text. You can do this by tokenizing the text into words and then counting the frequency of each word.
3. Finally, you can calculate the sentiment score of the text by summing the sentiment scores of each word, weighted by its frequency.

Word	Word Score	Num Occur	Total
good	3	2	6
tasty	1	1	1
bad	-3	1	-3
		Total	4

4. You can then classify the text as positive, negative, or neutral based on the sentiment score. For example, you could say that a score greater than 0 is positive, less than 0 is negative, and 0 is neutral.

> 👉 Even though we have not applied Machine Learning using this approach, we see we can actually text to word frequencies based on their appearance within the documents. This is an example of feature extraction process

Notebook:

intro_to_nlp/02-basic-sentiment-classification-dictionary.ipynb at 92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2 · juanhuguet/intro_to_nlp
Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.

https://github.com/juanhuguet/intro_to_nlp/blob/92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2/notebooks/02-basic-sentiment-classification-dictionary.ipynb

▼ **Feature extraction for ML classification**

A count vectorizer is a feature extraction technique in natural language processing (NLP) that converts a collection of text documents into a matrix of word counts. In other words, it transforms text data into a numerical representation that machine learning models can understand and process.

Here's how it works:

1. The count vectorizer takes a collection of text documents and tokenizes them into individual words.
2. It then counts how many times each word appears in each document.
3. Finally, it creates a matrix where each row represents a document and each column represents a word, and the values in the matrix are the word counts for each document.

For example, imagine you have a collection of three text documents:

- review 1: "The food was good"
- review 2: "The service was bad"
- review 3: "The service and the food were good"

The count vectorizer would convert these documents into a matrix like this:

	food	good	service	bad
review 1	1	1	0	0
review 2	0	0	1	1
review 3	1	1	1	0

Each row in the matrix represents a document, and each column represents a word. The values in the matrix are the word counts for each document.

Now that we have a numerical representation, if we have also the label “positive” or “negative” of each of the examples, we could train a classifier like a logistic regression without explicitly having to categorise the positive or negative intent of each word as it would automatically infer such relationships. This means, we would have transformed the sentiment classification into a supervised machine learning classification problem

We have seen that `CountVectorizer` is simple, efficient, and effective. It can be used as input to many machine learning algorithms, such as Naive Bayes and logistic regression, to build models for text classification, sentiment analysis, and more.

One of the main limitations of the count vectorizer is that **it treats all words as equally important**, regardless of their actual importance in distinguishing between different documents. For example, common words like "the", "a", and "an" may appear frequently in all documents, but they don't provide much information about the content of the document. Similarly, **rare words may be important for distinguishing between documents, but they may not appear frequently enough to be useful**.

To address these limitations, **a more advanced feature extraction technique called TF-IDF (term frequency-inverse document frequency**) was developed. TF-IDF takes into account **not only the frequency of a word in a document, but also the frequency of the word across all documents in the corpus**. This means that words that are common across all documents (like "the" and "a") are given a lower weight, while words that are rare but important for distinguishing between documents are given a higher weight.

Notebook

intro_to_nlp/03-basic-sentiment-classification-supervised-classification.ipynb at 92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2 · juanhugu
Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.

https://github.com/juanhuguet/intro_to_nlp/blob/92e9d259b0a2d445d4b6a32fcccf77da6cb7edf2/notebooks/03-basic-sentiment-classification-supervised-c
nb

The NLP revolution

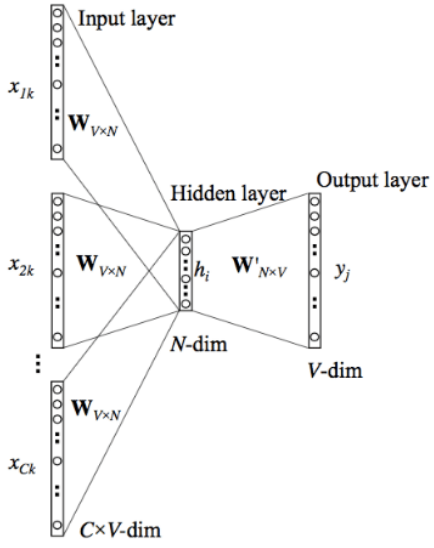
The main takeaway of the section above is we need to transform text into a mathematical representation, a.k.a vectors, to be able to feed the documents to machine learning algorithms so they can learn to map from the features to the labels.

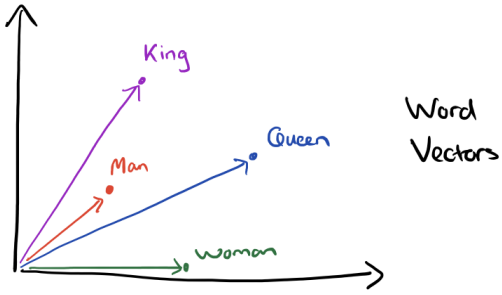
▼ Word Embeddings

Word embeddings are a type of vector representation for words in natural language processing (NLP) that have revolutionized the field by enabling more accurate and efficient text analysis.

As we have seen before, early embedding techniques like CountVectorizer or TF-ID have limitations in capturing the complex relationships between words and their meanings. They are just a sparse word representation of the universe of words in the documents. To address this issue, more sophisticated techniques like Global Vectors for Word Representation (**GloVe**) and **Word2vec** were developed.

These techniques use deep learning networks to transform sparse Bag of Word representation of text to dense vector representations by taking into account the context the word is appearing





🚀 Thanks to deep neural networks, **word embeddings** have evolved from sparse simple representations to **dense representations** that take into account the context the word normally appears. Also, these techniques show consistency in mathematical operations over the representations of the words.

Notebook:

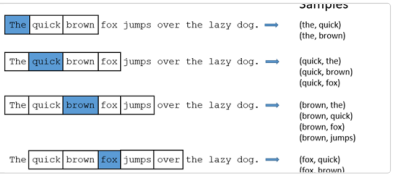
intro_to_nlp/04-basic-sentiment-classification-supervised-classification-embeddings.ipynb at 8e4208df7caef7680ab8660c1ce032b96448965d ·
Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.

https://github.com/juanhuguet/intro_to_nlp/blob/8e4208df7caef7680ab8660c1ce032b96448965d/notebooks/04-basic-sentiment-classification-supervised-classi

More details about word2vec from scratch

A simple Word2vec tutorial
In this tutorial we are going to explain, one of the emerging and prominent word embedding technique called Word2Vec proposed by Mikolov…

https://medium.com/@zafaralibagh6/a-simple-word2vec-tutorial-61e64e38a6a1



▼ Transformers, fine-tuning, and transfer learning

We have seen the power and effectiveness of dense word embeddings. However, they have a weakness: the word representations are contextless and do no capture long-term dependencies between words due to the limited span of the sliding window.

To illustrate this example, let's imagine the next sentences:

> An apple a day keeps the doctor away!

> My doctor buys Apple stocks every day

We, humans, can quickly differentiate between both "apples" as we are context aware. The first sentence refers to a fruit whereas the second refers to a popular company.

▼ Transformers

Google researchers introduced the Transformer architecture in 2017 [1], which outperformed RNNs in translation quality and cost. This, along with ULMFiT's transfer learning approach [2], led to the development of GPT [3] and BERT [4]. These transformer models combined the Transformer architecture with unsupervised learning and surpassed benchmarks in NLP by a significant margin.

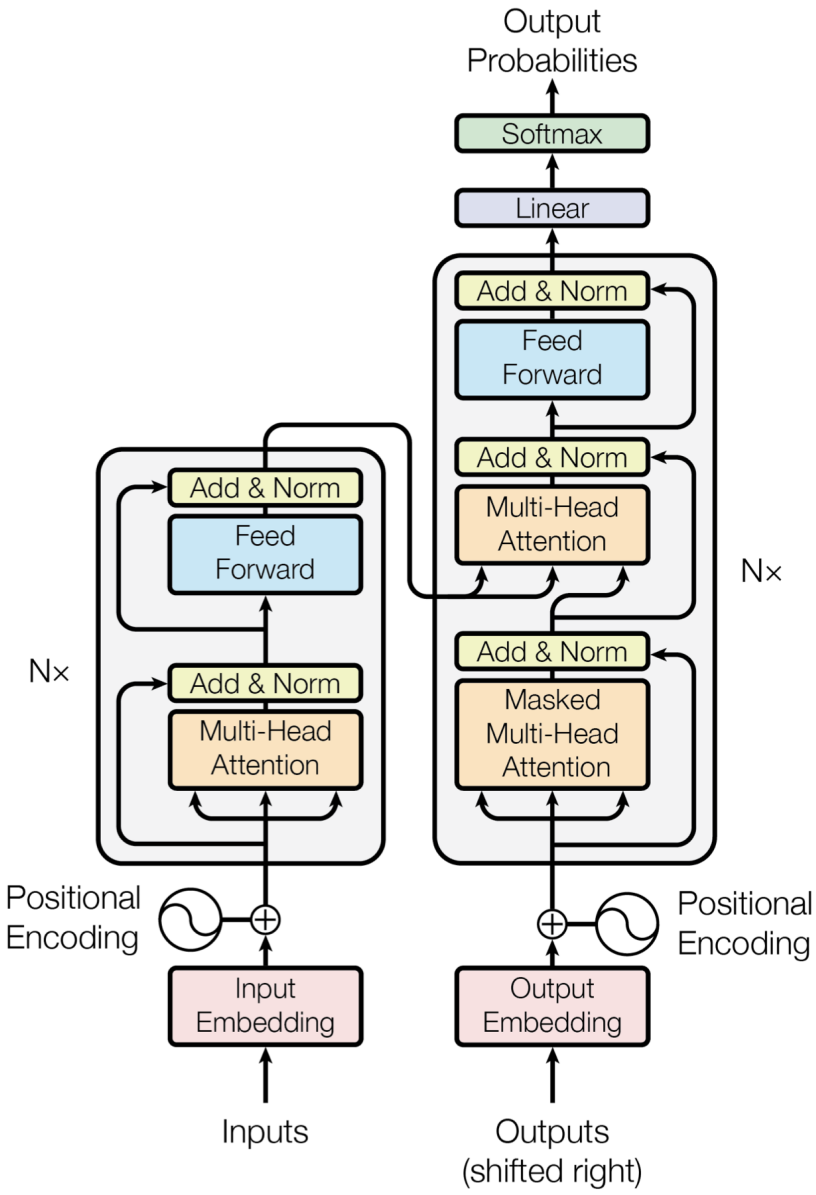
Transformers have solved several problems for natural language processing (NLP) applications, including:

1. **Capturing long-term dependencies**: Unlike earlier NLP models, such as recurrent neural networks (RNNs) and convolutional neural networks (CNNs), transformers are able to capture long-term dependencies in text. This makes

it possible to understand the context of a sequence of words, even if the dependent words are far apart.

2. **Efficient processing of long sequences**: Transformers are also able to process sequences of words more efficiently than RNNs or CNNs. This is because they use self-attention mechanisms, which allow them to focus on important parts of the input sequence and ignore irrelevant information. This has made it possible to train much larger models on much larger datasets than was previously possible.
3. **Pre-training and fine-tuning:** Pre-training and fine-tuning using large transformer models has become the basis for many state-of-the-art NLP models. By pre-training a large transformer model on massive amounts of text data, and then fine-tuning it on smaller datasets for specific tasks, it is possible to achieve highly effective transfer learning. This approach has been used to achieve state-of-the-art results on a wide range of NLP tasks, including language modeling, sentiment analysis, machine translation, and more.

▼ The anatomy of a transformer



The Transformer architecture consists of an encoder and a decoder, both of which use self-attention to process sequences of input tokens.

▼ Encoder

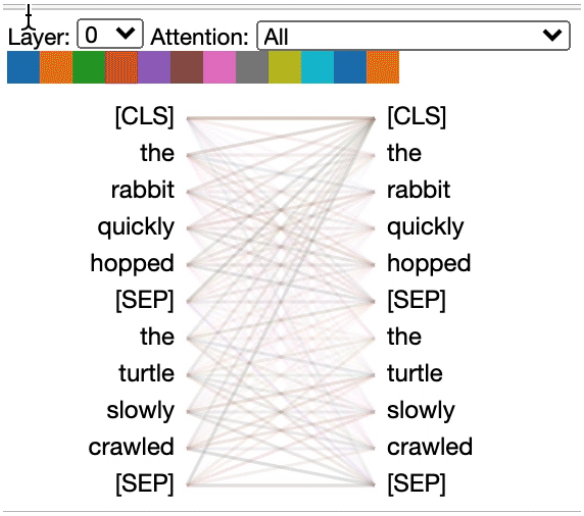
The encoder takes in a sequence of input tokens and generates a sequence of hidden representations for each token. Each hidden representation is generated by attending to all the other tokens in the sequence using self-attention. This means that the encoder assigns weights to each token in the sequence based on its relevance to the other tokens, allowing it to capture long-range dependencies and context.

▼ **Decoder**

The decoder takes in the sequence of hidden representations generated by the encoder, along with a target sequence, and generates a sequence of output tokens. Like the encoder, the decoder uses self-attention to attend to the input sequence, but it also uses a second type of attention called "encoder-decoder attention" to attend to the hidden representations generated by the encoder. This allows the decoder to generate output tokens that are conditioned on the input sequence.

▼ **Self-attention**

Self-attention is a mechanism that allows the model to attend to different parts of the input sequence while generating the hidden representations or output tokens. It does this by computing a weighted sum of all the tokens in the sequence, where the weights are determined by a learned function that calculates the relevance of each token to the current token being processed. By using self-attention, transformers can capture long-range dependencies and context more effectively than previous NLP models.

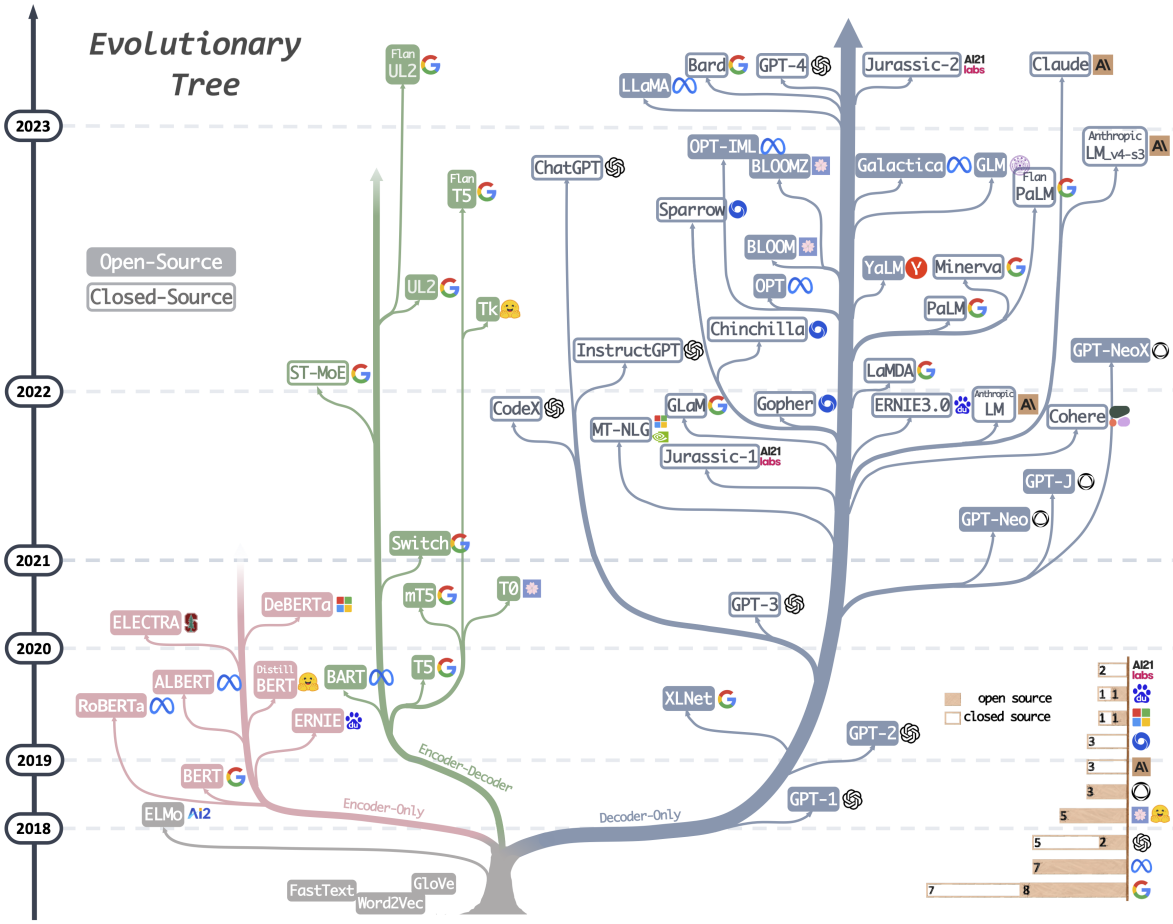


Check them out interactively!

Google Colaboratory

https://colab.research.google.com/drive/1hXIQ77A4TYS4y3UthWF-Ci7V7vVUoxmQ?usp=sharing

▼ The transformers family



source: here

Tipe	Definition	Family	Tasks
encoder-only	input sequence of text into a rich numerical representation	BERT and variants	text classification, named entity recognition
Decoder-only	tThese models will auto-complete the sequence by iteratively predicting the most probable next word	GPT and variants	text generation, question answering, summarization
encoder-decoder	used for modeling complex mappings from one sequence of text to another	BART, T5	machine translation and summarization tasks

Biblio recommended: Tunstall, Lewis; Werra, Leandro von; Wolf, Thomas. Natural Language Processing with Transformers, Revised Edition (p. 117). O'Reilly Media. Kindle Edition.

Technology landscape

▼ Model hubs: Hugging Face transformers library

The Transformers library is a popular open-source library developed by Hugging Face that provides a high-level API for natural language processing (NLP) tasks. It is built on top of PyTorch and TensorFlow, and it implements state-of-the-art models like BERT, GPT-2, and RoBERTa.

The library provides pre-trained models that can be fine-tuned on specific NLP tasks with relatively small amounts of task-specific data. The library also provides utilities for data preprocessing, model evaluation, and inference.

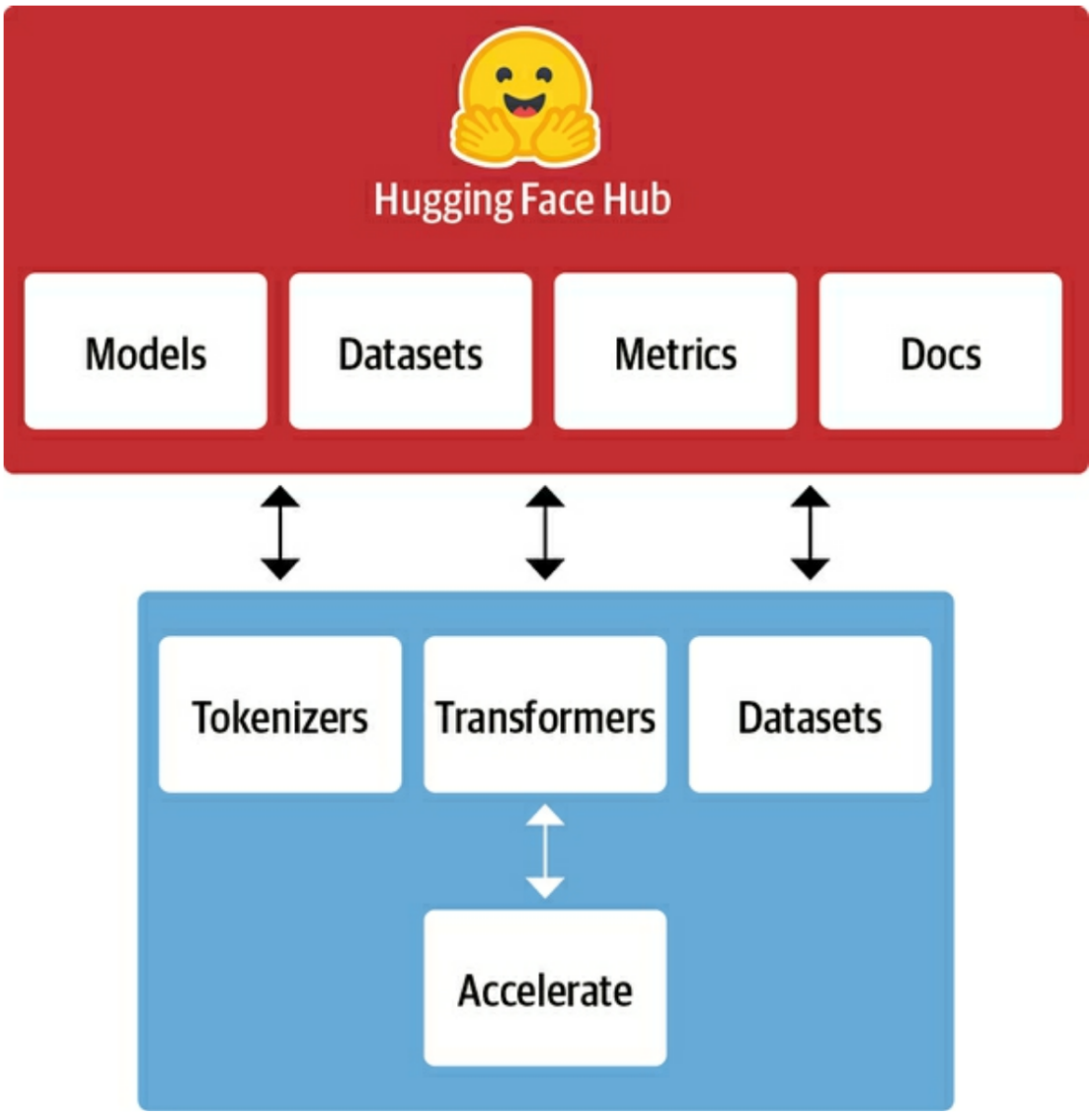
The Transformers Hub is a platform built on top of the Transformers library that allows users to share and discover pre-trained models. It provides a central repository of models trained on various tasks and languages, which can be easily downloaded and used for downstream tasks. Users can also upload their own models to the Hub, making it easy to share and collaborate on NLP projects.

🤗 Transformers

We're on a journey to advance and democratize artificial intelligence through open source and open science.

🤗 https://huggingface.co/docs/transformers/index

Transformers
Documentation



💡 Whenever you face a new NLP task/challenge, review what has been already done by the community, choose the best base model and adapt it to your needs. Most of the times you will find an out-of-the-box solution that performs well enough or that needs little fine tuning

▼ NLP powered industrial applications

spacy	Python library for NLP tasks like NER and text classification, optimized for performance and capable of handling large-scale text processing
langchain	building, training, and deploying custom language models, with a user-friendly interface and support for various NLP tasks
haystack	end-to-end framework for building scalable and efficient question-answering systems

Hands-on workshop

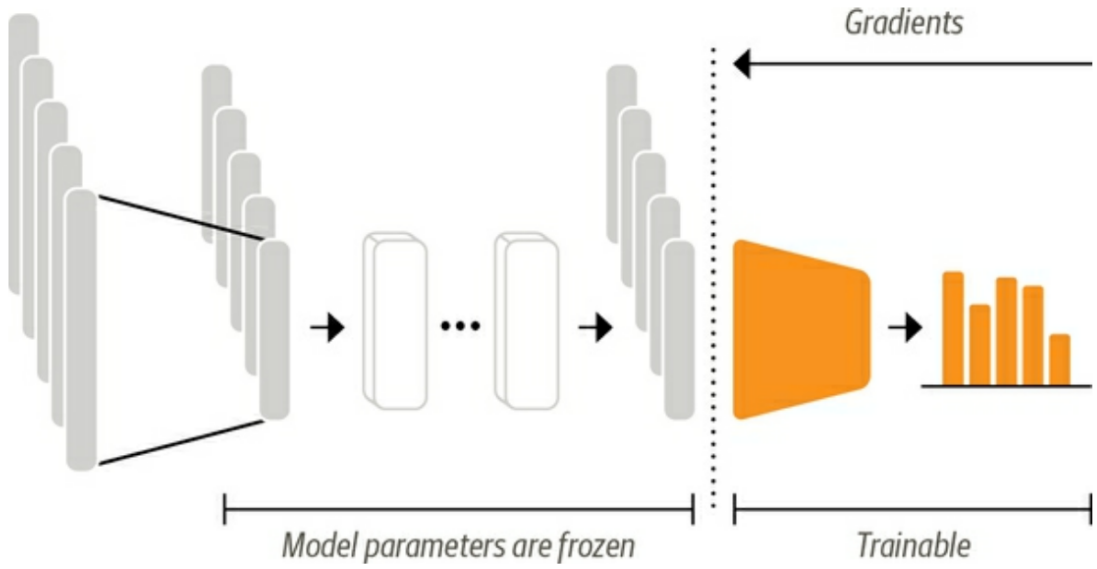
▼ Transformers in practice

Nowadays, thanks to the transformers revolution, most NLP tasks can be solved by choosing a pre-trained model and use fine-tuning and transfer learning for our particular use case.

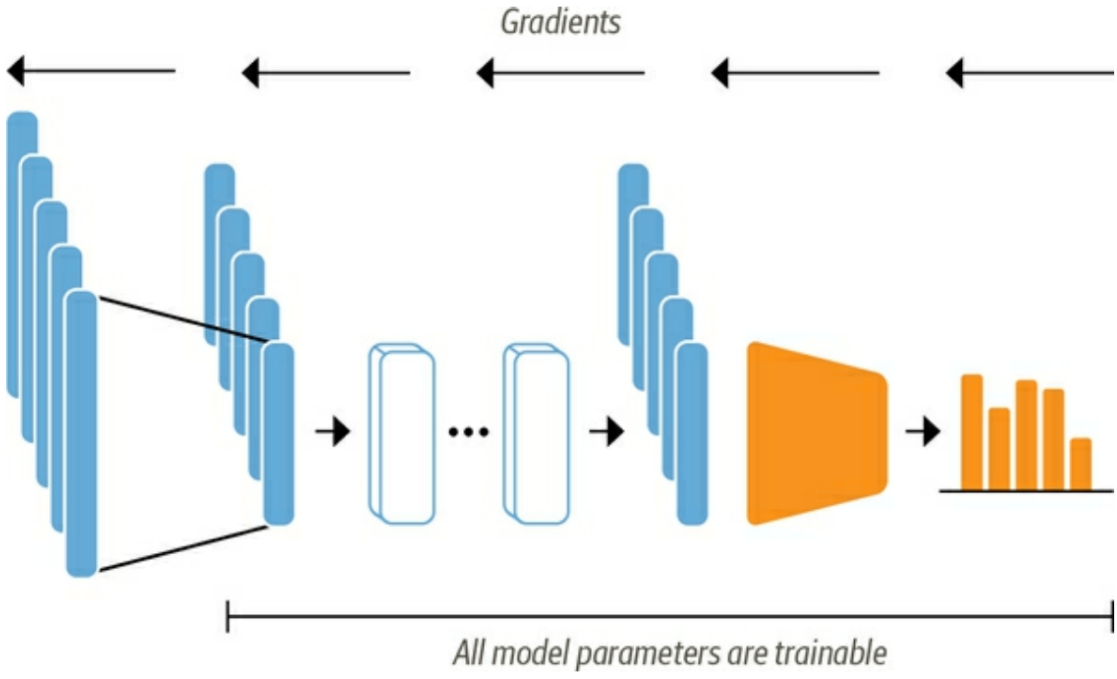
> 💡 The key difference between fine-tuning and transfer learning is that fine-tuning involves adjusting the pre-trained model's parameters to fit a specific task, while transfer learning involves applying the pre-trained model to a new task without further training.

Architecturally, this involves splitting the model into of a body and a head, where the head is a task-specific network and the body is normally the pre-trained full encoder or decoder layers of the model.

transfer learning



Fine tuning



▼ hands-on text classification with transformers

intro_to_nlp/05-transformers-text-classification.ipynb at 26fb0883c92e7c92ba91c26cd1ba3137ae200d74 · juanhuguet/intro_to_nlp
Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.
https://github.com/juanhuguet/intro_to_nlp/blob/26fb0883c92e7c92ba91c26cd1ba3137ae200d74/notebooks/05-transformers-text-classification.ipynb

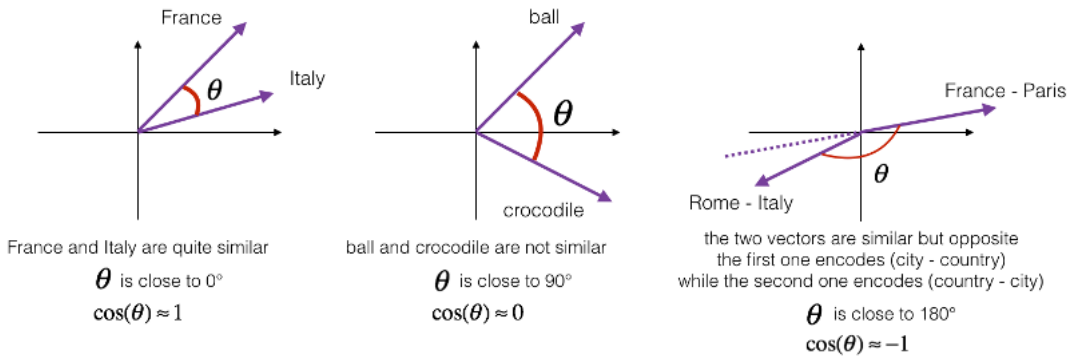
▼ Text clustering with sentence embeddings

Cosine similarity is a way to measure the similarity between two pieces of text. It works by comparing the angle between the two text embeddings rather than the distance between them.

Think of it this way: imagine two arrows pointing in different directions. If the arrows are pointing in the same direction, the angle between them is small, and we would say they are similar. If the arrows are pointing in opposite directions, the angle between them is large, and we would say they are dissimilar.

Similarly, with cosine similarity, if the angle between the text embeddings is small, we say the two pieces of text are similar. If the angle is large, we say they are dissimilar.

Cosine similarity is often used in applications like search engines or recommender systems, where we want to find items that are similar to a given query. By calculating the cosine similarity between the query and other items in a database, we can rank the items by similarity and return the most relevant results.



intro_to_nlp/06-text-clustering-with-embeddings.ipynb at d304c3830d2ac31e84274b41b3841844ce8a965e · juanhuguet/intro_to_nlp
Contribute to juanhuguet/intro_to_nlp development by creating an account on GitHub.
https://github.com/juanhuguet/intro_to_nlp/blob/d304c3830d2ac31e84274b41b3841844ce8a965e/notebooks/06-text-clustering-with-embeddings.ipynb

Q&A

▼ 📮 Q&A Notes

Recording link