Agenda

01. Pipeline

02. Tokenizer

03. Model

04. Datasets

05. Evaluate

06. Trainer

1 Pipeline

針對非ERP/MES 的應用系統，以與M365整合為前提

Part A: 維運與優化

- L5 1.0平台的惟運
- L5 2.0 上線-ChatGPT的應用、系統無須跳轉的執行Email/Teams (優化)、UI/UX 優化
- 指標市集系統 (新增模組)
- 專案管理系統 (新增模組)

Part B: 專案

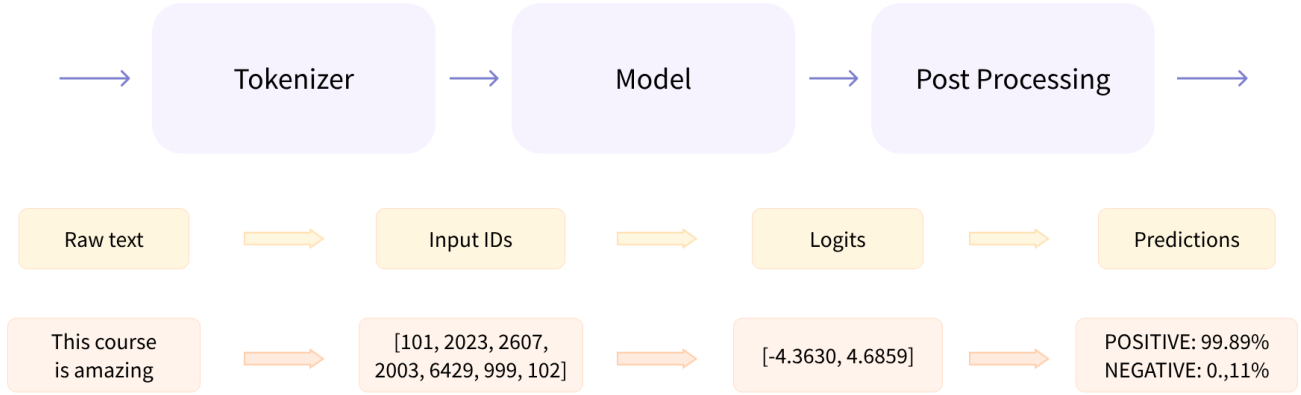
- 不銹鋼報價系統 (與業務預估、業務客訪整合)
- 線纜商機追蹤系統 (與專案追蹤、業務客訪整合)
- 出差 (國外) 報支系統

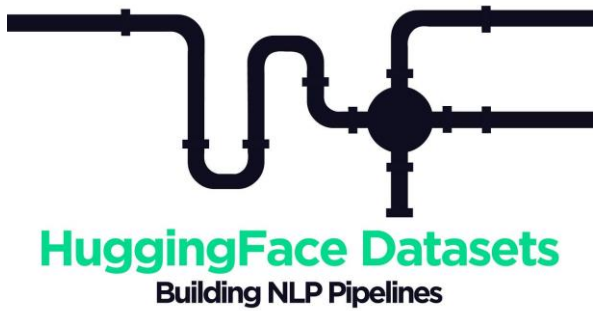
1 Pipeline

- 整合資料預處理, 模型載入, 推論過程的流水線

特色與價值:

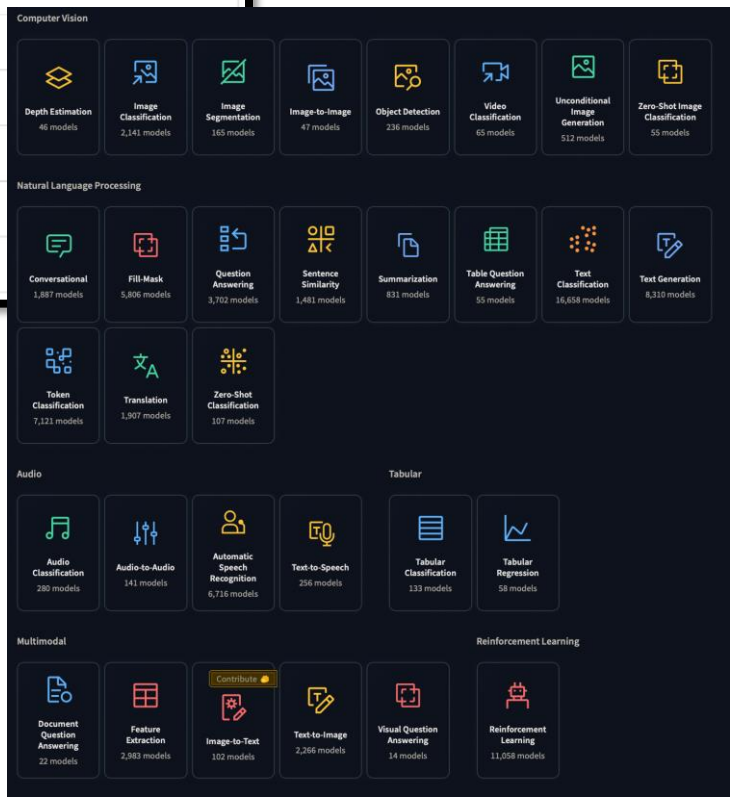
- **簡單易用**：Pipeline 提供了簡潔的API，使執行各種NLP任務變得非常容易，無需深入了解模型架構或複雜的預處理。
- **預訓練模型支援**：Pipeline 基於Hugging Face的Transformers庫，支援眾多預訓練的NLP模型，如BERT、GPT-2、RoBERTa、T5等。這些模型可以直接用於不同任務。
- **多任務支援**：Pipeline 允許您執行多個不同類型的NLP任務，如文本分類、命名實體識別、文本生成等，只需簡單的配置。
- **多語言支援**：Pipeline 支援多種語言，因此可以用於處理各種不同語言的文本。
- **管道化處理**：您可以將多個NLP任務連接成一個管道，使得數據可以流經不同任務進行處理，從而構建複雜的NLP工作流程。





1 Pipeline 支援的任務類型

名称	任务类型
text-classification（sentiment-analysis）	text
token-classification（ner）	text
question-answering	text
fill-mask	text
summarization	text
translation	text
text2text-generation	text
text-generation	text
conversational	text
table-question-answering	text
zero-shot-classification	text
automatic-speech-recognition	multimodal
feature-extraction	multimodal
audio-classification	audio
visual-question-answering	multimodal
document-question-answering	multimodal
zero-shot-image-classification	multimodal
zero-shot-audio-classification	multimodal
image-classification	image
zero-shot-object-detection	multimodal
video-classification	video



1 Pipeline 建立與使用

- 做甚麼(What) > 如何做(How)

特色與價值:

- **根據任務類型直接創建 Pipeline (隨機模型)：**
  1. pipe = pipeline(“text-classification”)
- **指定任務類型, 再指定模型, 再建立指定模型的Pipeline：**
  1. pipe = pipeline(“text-classification”, model=“uer/Roberta-base-finetuned-dianping-chinese”)
- **先下載模型，再建立 Pipeline：**
  1. model = AutoModelForSequenceClassification.from_pretrained(“uer/Roberta-base-finetuned-dianping-chinese”)
  2. tokenizer = AutoTokenizer.from_pretrained(“uer/Roberta-base-finetuned-dianping-chinese”)
  3. pipe = pipeline(“text-classification”, model=model, tokenizer=tokenizer)
- **使用GPU進行推理：**
  1. pipe = pipeline(“text-classification”, model=“uer/Roberta-base-finetuned-dianping-chinese”, device=0)

1 Pipeline 底層原理

- Tokenizer > Model > Post - Processing

轉換流程:

- **Step1 初始化 Tokenizer**
  tokenizer = AutoTokenizer.from_pretrained(“uer/roberta-base-finetuned-dianping-chinese”)
- **Step2 初始化 Model**
  model = AutoModelForSequenceClassification.from_pretrained(“uer/roberta-base-finetuned-dianping-chinese”)
- **Step3 數據預處理**
  Input_text = “我覺得不行”
  Inputs = tokenizer(input_text, return_tensors=“pt”)
- **Step4 模型預測**
  res = model(**inputs).logits
- **Step5 結果後處理**
  pred = torch.argmax(torch.softmax(logits, dim=-1)).item()
  result = model.config.id2label.get(pred)

Agenda

01. **Pipeline**

02. **Tokenizer**

03. **Model**

04. **Datasets**

05. **Evaluate**

06. **Trainer**

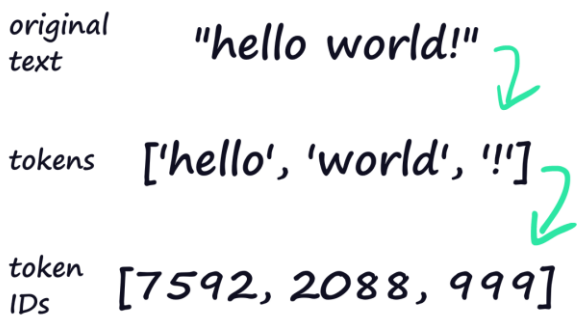
2 Tokenizer 簡介

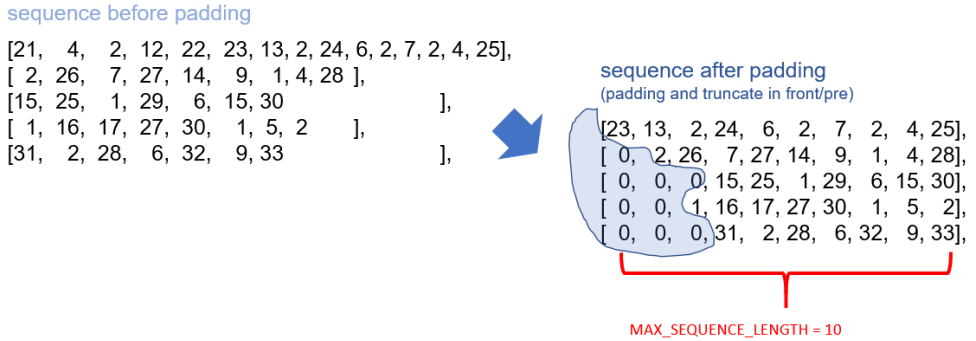
Tokenizer is all you need !

- 數據前處理工具 -> 分詞器

轉換流程:

- **Step1 分詞**
  使用分詞器對文本進行分詞
- **Step2 建構字典**
  根據分詞器的結果，創建字典映射表
- **Step3 數據轉換**
  根據建構好的字典，將處理後的 Token 進行映射，將文本序列轉換成數字序列
- **Step4 序列補齊或截斷**
  在批次輸入的序列串集合，確保每一個序列長度一致，針對過短的序列進行補齊，過長的序列進行截斷。



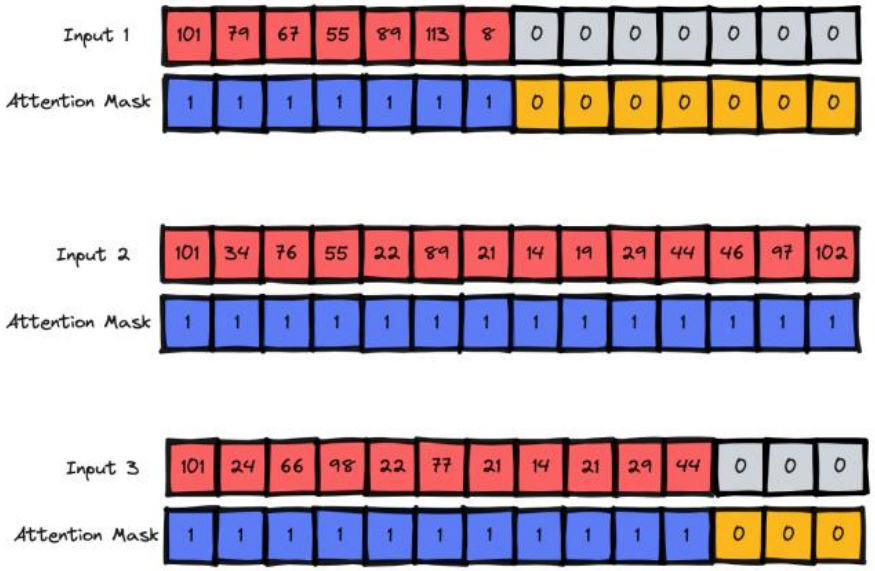


2 Tokenizer 基本用法

- 數據前處理工具 -> 分詞器

常用功能:

- 下載分詞模型(from pretrained/ save_pretrained)
- 句子分詞 (tokenize)
- 字典映射表 (vocab)
- 索引轉換 (convert_tokens_to_ids/ convert_ids_to_tokens)
- 填充截斷 (padding/ truncation)
- 其他輸入 (attention_mask/ token_type_ids)



2 Fast/Slow Tokenizer

特性	快速分詞器 (Fast Tokenizer)	慢速分詞器 (Slow Tokenizer)
實現語言	多以 Rust 語言實現	通常全用 Python 語言寫成
速度	因為使用 Rust，執行速度快	相較之下較慢，因為是 Python 實現
靈活性	較不靈活，自訂化能力有限	更靈活，可以自定義和調整
效能	高效能，適合大規模文字處理	效能較低，但適合研究和特殊需求
一致性	確保一致的分詞，重要於模型結果復現	可能因自訂化而導致分詞結果不一
適用場景	需要高速和高效能的場景	研究階段或需要特定分詞規則時適用

```python
import time
from transformers import AutoTokenizer

# Load the fast tokenizer
fast_tokenizer = AutoTokenizer.from_pretrained("uer/roberta-base-finetuned-chinanews-chinese")
# Convert the fast tokenizer into a slow one
slow_tokenizer = AutoTokenizer.from_pretrained("uer/roberta-base-finetuned-chinanews-chinese", use_fast=False)
# Define a sample text or a set of texts
text = "Hello, Hugging Face is amazing for NLP tasks!" * 100  # Replicate the text to make it larger
# Measure time for the fast tokenizer
start_time = time.time()
fast_tokenized_output = fast_tokenizer(text)
fast_time = time.time() - start_time

# Measure time for the slow tokenizer
start_time = time.time()
slow_tokenized_output = slow_tokenizer.encode_plus(text)
slow_time = time.time() - start_time

# Print the results
print(f"Time taken by Fast Tokenizer: {fast_time} seconds")
print(f"Time taken by Slow Tokenizer: {slow_time} seconds")
```

[4] ✓ 0.5s

```
Time taken by Fast Tokenizer: 0.0045223236083984375 seconds
Time taken by Slow Tokenizer: 0.02837061882019043 seconds
```

Agenda

01. Pipeline

02. Tokenizer

03. Model

04. Datasets

05. Evaluate

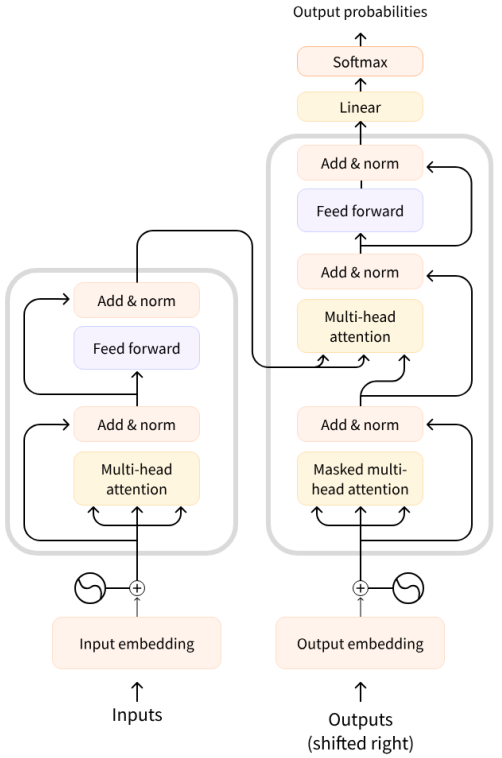
06. Trainer

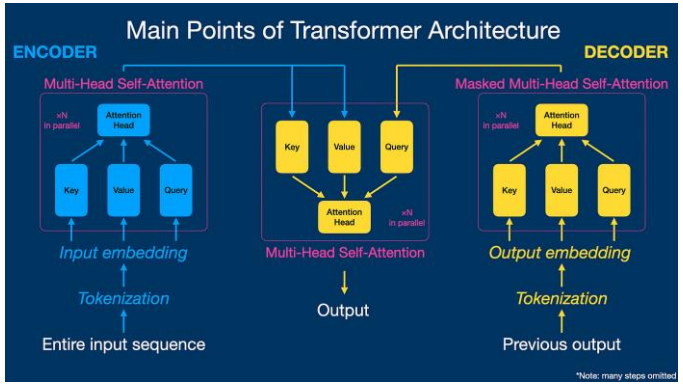
3 Model 簡介

- Transformer – base model

模型架構:

- Transformer
  - 基礎 Transformer 為編碼器(Encoder)、解碼器(Decoder) 模型
  - Encoder (編碼器) 負責利用向量化的 token 建構特徵
  - Decoder (解碼器) 結合特徵向量和輸入向量生成目標序列
  - 編碼器和解碼器，皆是由多個Transformer Block 堆疊而成
  - 一個 Transformer Block 單元，由注意力機制(Attention) 和前饋網路(FFN) 組合而成
- 注意力機制
  - 編碼器在計算當前詞的特徵表示時，可以通過注意力機制對上下文給予重要性的權重



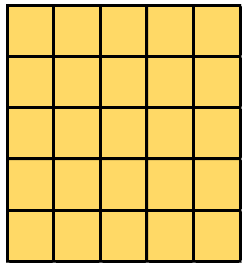


3 Model - 注意力矩陣

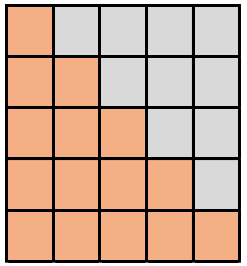
- Transformer component – attention matrix

模型編碼解碼類型:

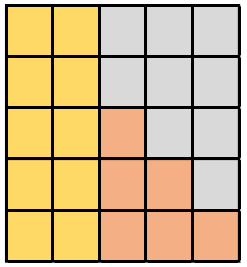
- 編碼模型
  - 自編碼模型, 雙向注意力機制, 特色會參考上下文
- 解碼模型
  - 自回歸模型, 單向注意力機制, 只會參考上文
- 編碼和解碼模型
  - 序列到序列, 使用 Encoder + Decoder



Encoder model



Decoder model



Encoder + decoder model

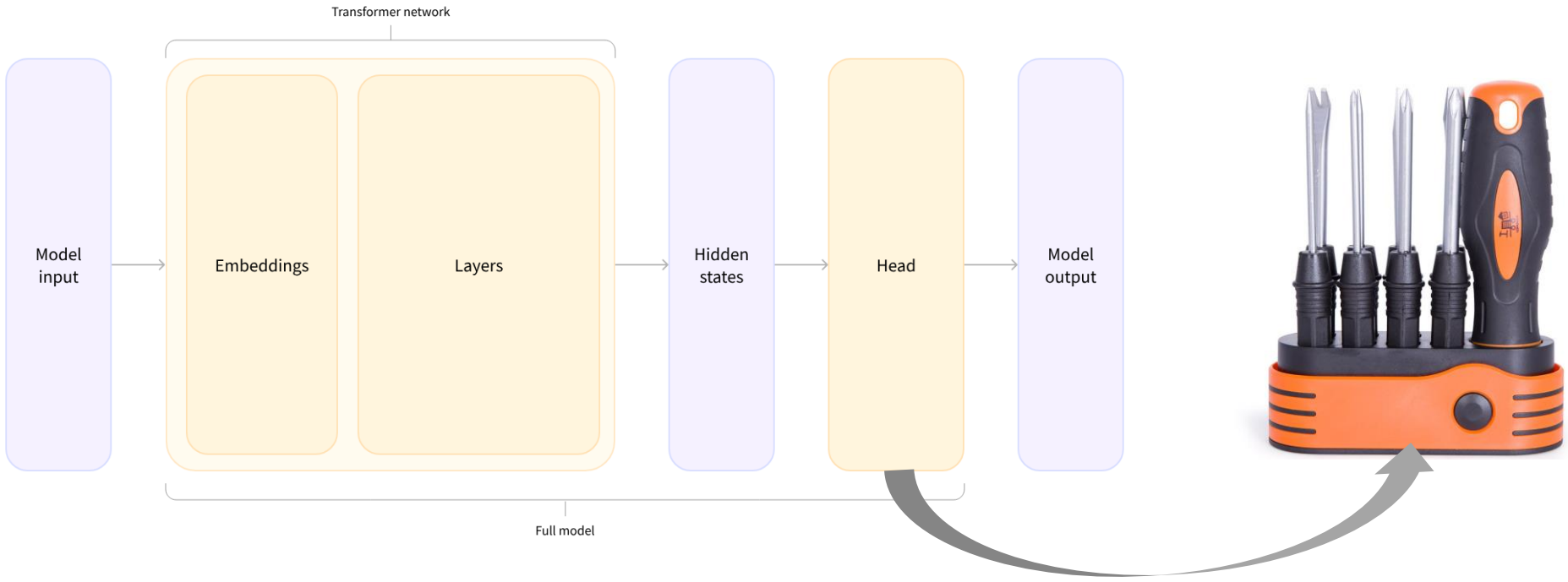
模型類型	常用預訓練模型	適用任務
編碼器模型, 自編碼	ALBERT, BERT, DistilBERT, RoBERTa	文本分類, 命名實體識別, 閱讀理解
解碼器模型, 自回歸	GPT, GPT2, Bloom, LLaMA	文本生成
編碼器/ 解碼器模型 seq 2 seq	BART, T5, Marian, mBART, GLM	文本摘要, 機器翻譯

3 Model – Head

- Transformer – base model

多任務微調:

- 尾椎應用
  - Model Head 是連接在模型的層，通常為 MLP model
  - Model Head 將模型的編碼表示結果進行非線性映射，應付不同任務類型



3 Model Head – Tasks introduction

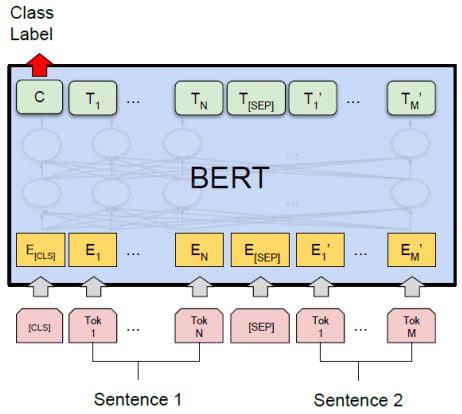
- Transformer – base model

模型架構:

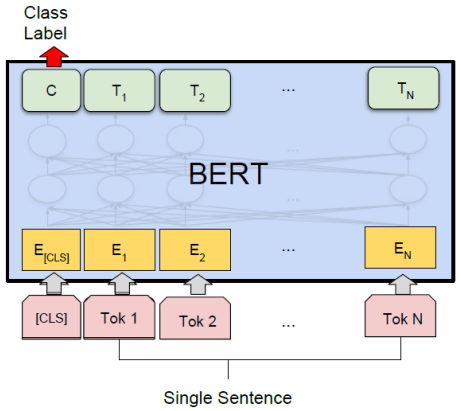
- Body Model

(main model, only return encoding result)

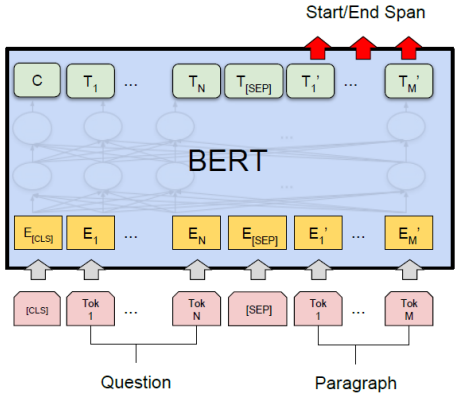
- **ForCausalLM
- **ForMaskedLM
- **ForSeq2SeqLM
- **ForMultipleChoice
- **ForQuestionAnswering
- **ForSequenceClassification
- **ForTokenClassificaation
- **...



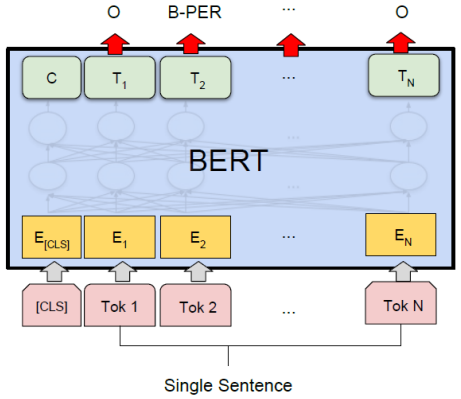
(a) Sentence Pair Classification Tasks: MNLI, QQP, QNLI, STS-B, MRPC, RTE, SWAG



(b) Single Sentence Classification Tasks: SST-2, CoLA



(c) Question Answering Tasks: SQuAD v1.1



(d) Single Sentence Tagging Tasks: CoNLL-2003 NER

3 Model 使用方式

- 選擇任務 -> 套用模型 -> 加載數據/處理-> 推論

流程:

- Model 下載與儲存
  - 雲端下載
  - 現有模型導入
  - 下載參數設定
- Model 套用
  - 外掛 model head
  - 不外掛 model head
- 實例應用
  - 任務類型
    - 文本分類
  - 使用模型
    - Hfl/rbt3
  - 數據集位置
    - Https://github.com/SophonPlus/ChineseNlpCorpus

```python
# 步驟 1: 導入庫
import torch
from transformers import AutoModelForSequenceClassification, AutoTokenizer

# 步驟 2: 加載分詞器和模型
model_name = "hfl/rbt3" # 或其他適合中文的模型
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

model.config.id2label[0] = "好評"
model.config.id2label[1] = "負評"

# 將模型設置為評估模式
model.eval()

# 步驟 3: 準備輸入數據
sen = "我覺得這家餐廳的飯菜味道不錯！"
inputs = tokenizer(sen, return_tensors="pt")

# 步驟 4: 模型推理
with torch.no_grad():  # 禁用梯度計算
    logits = model(**inputs).logits

# 步驟 5: 結果解釋
pred = torch.argmax(logits, dim=-1)
print(f"輸入: '{sen}'的預測結果是: {model.config.id2label[pred.item()]}")

# 使用 pipeline 進行簡化流程
from transformers import pipeline

# 創建文本分類管道
pipe = pipeline("text-classification", model=model, tokenizer=tokenizer)

# 使用管道進行預測
result = pipe(sen)
print(result)

```

Agenda

01. **Pipeline**

02. **Tokenizer**

03. **Model**

04. **Datasets**

05. **Evaluate**

06. **Trainer**

4 Datasets 介紹與使用

- 模型訓練資料準備 from HuggingFace Hub

功能清單:

- 下載線上數據 (load_dataset)
- 選擇特定集合 (load_dataset)
- 選擇特定類型, 訓練, 驗證, 測試 (load_dataset)
- 查看數據集 (index and slice)
- 數據切分 (train_test_split)
- 數據選取&過濾 (select and fliter)
- 數據映射 (map)
- 數據儲存和讀取 (save_to_disk / load_from_disk)

Docs:
https://huggingface.co/docs/datasets/index
Library:
https://huggingface.co/datasets

4 Datasets 下載數據方法

- 模型訓練資料準備 from HuggingFace Hub

種類:

- 本地端下載 to Datasets
  CSV, JSON
- 本地端文件夾批次下載 to Datasets
- 本地端格式轉換 to Datasets
  dict, dataframe, list
- Python 爬蟲腳本下載
  def_info(self)
  def_split_generators(self, dl_manager)
  def_generate_examples(self,filepath)

Agenda

01. **Pipeline**

02. **Tokenizer**

03. **Model**

04. **Datasets**

05. **Evaluate**

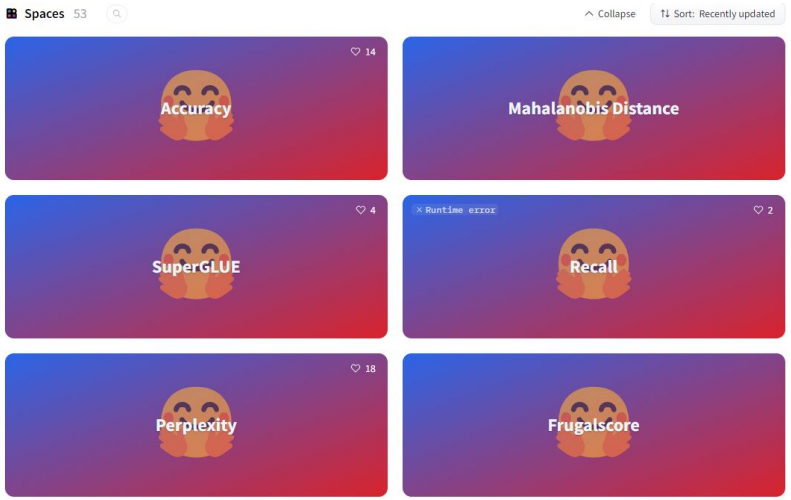
06. **Trainer**

4 Evaluate 介紹與使用

- 模型評估函式庫

方法清單:

- 查看支持的評估函數 (list_evaluation_modules)
- 下載評估函數(load)
- 查看評估函數說明(inputs_description)
- 評估指標計算(compute)
  - 全域計算(compute)
  - 迭代計算(add/ add_batch)
- 計算多個評估指標(combine)
- 評估結果可視化(radar_plot)



Evaluate

Docs:
https://huggingface.co/docs/evaluate/index
Library:
https://huggingface.co/evaluate-metric

Agenda

01. **Pipeline**

02. **Tokenizer**

03. **Model**

04. **Datasets**

05. **Evaluate**

06. **Trainer**

3 Trainer 介紹與使用

- 整合訓練器

功能:

- Trainer 是 transformers 庫中提供的訓練函數，封裝了完整訓練計劃，評估方法，並整合多種API，例如: DeepSpeed, Pytorch FSDP 等，搭配 TrainingArguments 對訓練過程中的各項參數進行配置，可以非常快速啟動單機或分布式運算

注意事項:

- 使用 Trainer 進行模型訓練對模型的輸入輸出有現制，模型必須返回tuple或是modelOutput的子類
- 如果輸入中提供 labels，模型會返回 loss 結果
- 如果輸入型態是 tuple，loss 結果為tuple中的第一個值

文件說明地址
https://huggingface.co/docs/transformers/main_classes/trainer

Agenda

Transformers NLP 實戰

01 介紹
02 命名實體識別
03 機器閱讀理解
04 選擇問答
05 文本相似度
06 檢索式問答
07 語言模型
08 遮罩語言模型
09 因果推論語言模型
10 文本摘要
11 生成式語言模型