

✓ Hugging Face CloudPlatform Freshman TA Material

Temp self-HuggingFace Token:

hf_VDPJZxCYszInpdktlPnplEooEAzLcHKeX

下面是对 **Hugging Face** 的简要介绍，适合初学者快速理解它的核心用途和价值：

Hugging Face的datasets库允许用户导入自己数据集，并通过简单的接口进行处理。你可以将数据从各种格式（如CSV、JSON、文本文件等）加载到datasets库中，然后进行进一步的数据处理、预处理、转换和保存。

- 从CSV或JSON加载数据集
- 自定义加载脚本: (如果你的数据格式比较特殊，可以自定义加载脚本。Hugging Face允许你编写一个Python脚本，按照自己的方式读取数据并返回一个Dataset对象)
- 一旦数据加载进来，你就可以使用datasets库的函数进行数据的处理
- Package
 - datasets : Hugging Face 的数据集库，方便加载和处理数据
 - accelerate: 用于简化分布式训练和混合精度训练
 - transformers : Hugging Face Transformers 库本身


Step0: Introduction to Hugging Face

Hugging Face 是一家专注于自然语言处理（NLP）和人工智能的公司，他们开发了一个非常流行的开源平台，提供了：


- 🌟 **核心产品：Transformers 库**
- 这是一个包含 **数千个预训练模型** 的库，支持：
 - 文本分类（如情感分析）
 - 文本生成（如对话系统、写作助手）
 - 命名实体识别（NER）
 - 翻译、摘要、问答 等等
- 支持的模型包括：
 - BERT、GPT-2、GPT-3、RoBERTa、T5、DistilBERT、LLaMA 等等


- 📦 Hugging Face 提供的Components

组件	作用说明
🤖 Transformers	提供成千上万的预训练 NLP 模型，开箱即用
📚 Datasets	提供数百个高质量 NLP 数据集，支持快速加载

组件	作用说明
 Tokenizers	高效的文本编码工具，支持自定义 tokenizer
 Hugging Face Hub	模型和数据集的云平台（模型可以上传和下载）
 Accelerate, PEFT, Diffusers 等	支持模型加速、微调、图像生成等进阶任务

-  Hugging Face Hub（云平台）
 - 可以在 <https://huggingface.co>:
- 下载他人上传的模型和数据集
- 上传自己训练好的模型
- 在线测试模型（例如让一个 GPT 模型帮你写诗）

-  总结一句话:

 **Hugging Face** 是一个为 AI 开发者打造的生态系统，它让你可以快速使用最先进的自然语言处理模型和数据集。

✓ Step1: Install HuggingFace Package

```
1 !pip install transformers datasets accelerate torch # 或 tensorflow
```

```

⇒ Requirement already satisfied: transformers in /usr/local/lib/python3.11/dis
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: accelerate in /usr/local/lib/python3.11/dist-
Requirement already satisfied: torch in /usr/local/lib/python3.11/dist-packa
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/li
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.1
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/pyth
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: pyarrow>=8.0.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: dill<0.3.8,>=0.3.0 in /usr/local/lib/python3.
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: multiprocessing in /usr/local/lib/python3.11/dis
Requirement already satisfied: fsspec>=2021.11.1 in /usr/local/lib/python3.1
Requirement already satisfied: aiohttp in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: psutil in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: typing-extensions>=4.10.0 in /usr/local/lib/p
Requirement already satisfied: networkx in /usr/local/lib/python3.11/dist-pa
Requirement already satisfied: jinja2 in /usr/local/lib/python3.11/dist-pack
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.4.127 in /usr/loca
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.4.127 in /usr/lo
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.4.127 in /usr/loca
Requirement already satisfied: nvidia-cudnn-cu12==9.1.0.70 in /usr/local/lib

```

```

Requirement already satisfied: nvidia-cublas-cu12==12.4.5.8 in /usr/local/li
Requirement already satisfied: nvidia-cufft-cu12==11.2.1.3 in /usr/local/lib
Requirement already satisfied: nvidia-curand-cu12==10.3.5.147 in /usr/local/
Requirement already satisfied: nvidia-cusolver-cu12==11.6.1.9 in /usr/local/
Requirement already satisfied: nvidia-cusparselt-cu12==0.6.2 in /usr/local/l
Requirement already satisfied: nvidia-nccl-cu12==2.21.5 in /usr/local/lib/py
Requirement already satisfied: nvidia-nvtx-cu12==12.4.127 in /usr/local/lib/
Requirement already satisfied: nvidia-nvjitlink-cu12==12.4.127 in /usr/local
Requirement already satisfied: triton==3.2.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: sympy==1.13.1 in /usr/local/lib/python3.11/di
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/pyt
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/di
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.1
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11
Requirement already satisfied: yarll<2.0,>=1.17.0 in /usr/local/lib/python3.1
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dis
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.11/
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/pyth
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dis
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/d

```

✓ Step2: Loading Self-Custom Data

```

1 from datasets import Dataset
2
3 data = {
4     "text": [
5         "Absolutely loved it! The plot was tight and the characters were comp
6         "Terrible. Just terrible. I want my two hours back.",
7         "Not bad, but could have been so much better with a stronger script."
8         "The cinematography was breathtaking, but the story dragged on and lo
9         "A thrilling ride from start to finish. I was on the edge of my seat!
10        "I'm not sure why people liked this. I found it pretty boring.",
11        "The lead actor's performance saved the entire movie.",
12        "One of the worst films I've seen this year. Utterly forgettable.",
13        "An emotional rollercoaster. I cried and laughed – sometimes in the s
14        "If mediocrity had a name, it would be this film.",
15        "Witty dialogue, great pacing, and a satisfying ending. Highly recomm
16        "The trailers were misleading. This was nothing like I expected – and
17        "Visually stunning but lacked substance. All style, no soul.",
18        "A masterpiece! This deserves awards.",
19        "Predictable plot, flat characters, and cheesy lines. Pass.",
20        "Unexpectedly delightful. It far exceeded my expectations.",
21        "I fell asleep halfway through. That says everything.",
22        "Some interesting ideas, but poorly executed overall.",
23        "The sequel is a rare case of being better than the original.",
24        "So bad, it was actually kind of entertaining."

```

```

25 ],
26   "label": [
27       1, 0, 0, 0, 1,
28       0, 1, 0, 1, 0,
29       1, 0, 0, 1, 0,
30       1, 0, 0, 1, 0 # 1: 正面情绪, 0: 负面情绪
31   ]
32 }
33

```

✓ Step3: Load Pre-trained Model & Tokenizer

- Hugging Face Self-Tokens: hf_VDPJZxCYszInpdktlPnpIEooEAzLcHKeX
- 自定义化流程, 将整个登陆的点击过程自动化

```

1 from huggingface_hub import login
2 login(token="hf_VDPJZxCYszInpdktlPnpIEooEAzLcHKeX")

```

```
1 !pip install -U transformers
```

```

⇒ Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: huggingface-hub<1.0,>=0.30.0 in /usr/local/lib/
Requirement already satisfied: numpy>=1.17 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.11/d
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11
Requirement already satisfied: requests in /usr/local/lib/python3.11/dist-pac
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/pytho
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.1
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.11/dist-p
Requirement already satisfied: fsspec>=2023.5.0 in /usr/local/lib/python3.11/
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/p
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/pytl
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.1
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.1

```

- 你当前的问题是因为 train_dataset (和/或 eval_dataset) 没有经过 tokenizer 处理, 因此缺少 input_ids 字段。
- 你在训练模型时, 输入数据 (train_dataset) 没有提供模型所需的字段 input_ids 或 inputs_embeds, 导致模型不知道输入的是什么。
- 也就是说, Trainer 在尝试调用 model.forward(...) 时, 模型期望你提供 input_ids (或可选地是 inputs_embeds), 但 train_dataset 中并没有这些字段。

Tips:

- 在pre-trained模型中, 我们需要载入两个组件:

- **model:** {pre-trained model name}
- **tokenizer:** {pre-trained tokenize name}
 - 运用已经预训练好的 pre-trainer Tokenizer, 将新的数据进行 'Token IDs' encoder & decode

Example:

加载预训练的 Tokenizer

```
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
```

进行调用 tokenizer 进行 token化 的过程

```
def tokenize_function(example):
```

```
    return tokenizer(example["text"], truncation=True, padding="max_length")
```

经过处理后有用的数据


```
tokenized_dataset = raw_dataset.map(tokenize_function)
```

```
1 from transformers import AutoTokenizer, AutoModelForSequenceClassification
2 from transformers import BertTokenizer, BertForSequenceClassification, Traine
3
4 # 通用的, 并没有任何特定任务下特殊的 最后一层 "分类头" => 目前这个模型只是加载了通用的语言理
5 model_name = "bert-base-uncased" # 选择一个预训练的 BERT 模型
6 # tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
7
8
9 # 对于文本分类任务, 使用 AutoModelForSequenceClassification
10 # num_labels 对应你的分类类别数量 # 注意!!!
11 # 因为你加载的 bert-base-uncased 是一个通用预训练的语言模型, 它本身没有“针对具体分类任务”
12 # 而 BertForSequenceClassification 是为分类任务准备的, 它会在BERT模型的基础上添加一层
13 # ⚠️ 这部分分类层的参数在 bert-base-uncased 的模型权重中根本就没有, 所以只能自动“新建”并
14
15 #####
16
17 # 解决上面一开始垫底运用 "无分类头" 的通用模型时候, 临时随机初始化
18 # Solution: 你需要在你的具体分类任务上对这个模型进行微调, 例如情感分析:
19 # 两个组件的载入
20 model = BertForSequenceClassification.from_pretrained('bert-base-uncased', nu
21 tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
22 # model = AutoModelForSequenceClassification.from_pretrained(model_name, num_
23
24 raw_dataset = Dataset.from_dict(data)
25
26 def tokenize_function(example):
27     return tokenizer(example["text"], truncation=True, padding="max_length")
28
29 # 应用 tokenizer, 返回的是包含 input_ids 和 attention_mask 的新数据集
30 # 运用已经预训练好的 pre-trainer Tokenizer, 将新的数据进行 'Token IDs' encoder & de
31 tokenized_dataset = raw_dataset.map(tokenize_function)
```

```

32
33 split_dataset = tokenized_dataset.train_test_split(test_size=0.2)
34 train_dataset = split_dataset['train']
35 eval_dataset = split_dataset['test']

```

 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning: The secret `HF_TOKEN` does not exist in your Colab secrets. To authenticate with the Hugging Face Hub, create a token in your settings tab. You will be able to reuse this secret in all of your notebooks. Please note that authentication is recommended but still optional to access public datasets.

Some weights of BertForSequenceClassification were not initialized from the model checkpoint. You should probably TRAIN this model on a down-stream task to be able to use it.

Map: 100% 20/20 [00:00<00:00, 129.00 examples/s]

检查字段完整性:

对于 **BERT 模型**，每条数据必须包含以下字段（至少）：

- input_ids: Tokenizer 将文本转换成的 ID 序列
- attention_mask: 标记哪些 token 是有效的
- label: 对应分类任务的标签（数值型）

```


# 转化为 BERT 模型能够接收的数据类型结构：（交给Tokenizer来决定）
{
    'text': 'I love this movie!',
    'label': 1,
    'input_ids': [...],
    'attention_mask': [...]
}

```

```

1 print(tokenized_dataset[0])
2 print(tokenized_dataset[1])
3 print(tokenized_dataset[2])
4 print(tokenized_dataset[3])
5 print(tokenized_dataset[4])

```

 {'text': 'Absolutely loved it! The plot was tight and the characters were compelling.'}

{'text': 'Terrible. Just terrible. I want my two hours back.', 'label': 0, 'input_ids': [...], 'attention_mask': [...]}

{'text': 'Not bad, but could have been so much better with a stronger script.'}

{'text': 'The cinematography was breathtaking, but the story dragged on and lost interest.'}

{'text': 'A thrilling ride from start to finish. I was on the edge of my seat.'}

✓ 3.1: Small Dataset Training

```

1 # training_args = TrainingArguments(
2 #     output_dir="./results",
3 #     # evaluation_strategy="epoch",
4 #     per_device_train_batch_size=16,
5 #     per_device_eval_batch_size=16,
6 #     num_train_epochs=3,
7 #     weight_decay=0.01,
8 # )
9
10 # # 使用 Trainer 微调
11 # trainer = Trainer(
12 #     model=model,
13 #     args=training_args,
14 #     train_dataset=train_dataset,
15 #     eval_dataset=eval_dataset,
16 # )
17
18 # trainer.train()

```

✓ 3.2: Large Dataset Training

```

1 from transformers import TrainingArguments, Trainer
2
3 import numpy as np
4 from sklearn.metrics import accuracy_score, f1_score
5
6 def compute_metrics(eval_pred):
7     logits, labels = eval_pred
8     predictions = np.argmax(logits, axis=-1)
9     accuracy = accuracy_score(labels, predictions)
10    f1 = f1_score(labels, predictions)
11    return {"accuracy": accuracy, "f1_score": f1}
12
13 training_args = TrainingArguments(
14     output_dir="./results",           # 输出目录
15     eval_strategy="epoch",           # 每个 epoch 评估一次
16     per_device_train_batch_size=8,   # 训练批次大小
17     per_device_eval_batch_size=8,    # 评估批次大小
18     num_train_epochs=3,              # 训练 epoch 数量
19     weight_decay=0.01,               # 权重衰减
20     logging_dir='./logs',            # log 目录
21     logging_steps=100,
22     load_best_model_at_end=False,     # 训练结束后加载最佳模型
23     metric_for_best_model="accuracy", # 根据准确率选择最佳模型
24     push_to_hub=False,               # 不推送到 Hugging Face Hub
25 )
26
27 trainer = Trainer(
28     model=model,
29     args=training_args,
30     train_dataset=train_dataset,
31     eval_dataset=eval_dataset,

```

```

32     tokenizer=tokenizer, # 传入 tokenizer 会在保存模型时一并保存
33     compute_metrics=compute_metrics,
34 )
35
36 trainer.train()
37
38 # 评估模型
39 results = trainer.evaluate()
40 print(results)

```

→ /tmp/ipython-input-8-3410218278.py:27: FutureWarning: `tokenizer` is deprecated
 trainer = Trainer(

wandb: WARNING The `run_name` is currently set to the same value as `Training`

wandb: Currently logged in as: 2925795986 (2925795986-bnu-hkbu-united-interna

Tracking run with wandb version 0.21.0

Run data is saved locally in /content/wandb/run-20250714_102330-f2mn0bua

Syncing run [./results](#) to [Weights & Biases \(docs\)](#)

View project at <https://wandb.ai/2925795986-bnu-hkbu-united-international-college/huggingface>

View run at <https://wandb.ai/2925795986-bnu-hkbu-united-international-college/huggingface/runs/f2mn0bua>

 [6/6 04:42, Epoch 3/3]

Epoch	Training Loss	Validation Loss	Accuracy	F1 Score
1	No log	0.747113	0.500000	0.000000
2	No log	0.729823	0.500000	0.000000
3	No log	0.725395	0.500000	0.000000

 [1/1 : < :]

{'eval_loss': 0.7253952026367188, 'eval_accuracy': 0.5, 'eval_f1_score': 0.0,

✓ 3.3 Check Pred Result

```

1 # results = trainer.evaluate()
2 print(results)

```

→ {'eval_loss': 0.7253952026367188, 'eval_accuracy': 0.5, 'eval_f1_score': 0.0,

✓ 3.4 Temp-Local Model Saved

```

1 # 保存模型
2 model.save_pretrained("./my_finetuned_model")
3
4 # 保存 tokenizer (非常重要, 推理时要一致)
5 tokenizer.save_pretrained("./my_finetuned_model")

```

→ ('./my_finetuned_model/tokenizer_config.json',
 './my_finetuned_model/special_tokens_map.json',
 './my_finetuned_model/vocab.txt',
 './my_finetuned_model/added_tokens.json')

✓ 3.5 Temp-Local Model Load & Test

```

1 from transformers import BertTokenizer, BertForSequenceClassification
2
3 # 加载保存的模型和 tokenizer
4 model = BertForSequenceClassification.from_pretrained("./my_finetuned_model")
5 tokenizer = BertTokenizer.from_pretrained("./my_finetuned_model")
6
7 # 推理示例
8 text1 = "I absolutely hated the story!"
9 inputs1 = tokenizer(text1, return_tensors="pt", truncation=True, padding=True)
10 outputs1 = model(**inputs1)
11 logits1 = outputs1.logits
12
13 text2 = "I absolutely love the story!"
14 inputs2 = tokenizer(text2, return_tensors="pt", truncation=True, padding=True)
15 outputs2 = model(**inputs2)
16 logits2 = outputs2.logits
17
18 # 获取预测结果
19 import torch
20 predicted_class1 = torch.argmax(logits1, dim=1).item()
21 print(f"Predicted label:{text1}", predicted_class1, '\n')
22
23 predicted_class2 = torch.argmax(logits2, dim=1).item()
24 print(f"Predicted label:{text2}", predicted_class2)

```

➡ Predicted label:I absolutely hated the story! 0

Predicted label:I absolutely love the story! 1

✓ Step4: Upload Model to HuggingFace Cloud Platform

可选：上传到 Hugging Face Hub（可公开或私有）

```

1 from huggingface_hub import whoami
2 whoami()

```

➡

```

{'type': 'user',
 'id': '6874ca64686a78908c3f53f2',
 'name': 'Suleynan',
 'fullname': 'Minghao Lee',
 'isPro': False,
 'avatarUrl': 'https://cdn-avatars.huggingface.co/v1/production/uploads/no-auth/TfjQEXSu7EU-GgxXxYDp1.png',
 'orgs': [],
 'auth': {'type': 'access_token',
 'accessToken': {'displayName': 'colab-access-HuggingFace',
 'role': 'fineGrained',
 'createdAt': '2025-07-14T09:36:16.999Z',
 'fineGrained': {'canReadGatedRepos': False,
 'global': []},

```

```
'scoped': [{'entity': {'_id': '6874ca64686a78908c3f53f2',
  'type': 'user',
  'name': 'Suleynan'},
  'permissions': ['repo.content.read',
  'repo.write',
  'inference.serverless.write']}]}}]}
```

```
1 from huggingface_hub import whoami
2 print(whoami())
```

```
➦ {'type': 'user', 'id': '6874ca64686a78908c3f53f2', 'name': 'Suleynan', 'fullna
```

```
1 # model.push_to_hub("my-finetuned-bert")
2 from huggingface_hub import login
3 # login("hf_你的新token")
4 # login("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")
5
6 model.push_to_hub("Suleynan/my-finetuned-bert")
7 tokenizer.push_to_hub("Suleynan/my-finetuned-bert")
```

```
➦ Uploading...: 100% 438M/438M [00:08<00:00, 81.3MB/s]

README.md: 5.17k/? [00:00<00:00, 268kB/s]
CommitInfo(commit_url='https://huggingface.co/Suleynan/my-finetuned-
bert/commit/be8e7a0df86cd13cde1119175de2b205e63939ef', commit_message='Upload
tokenizer', commit_description='',
oid='be8e7a0df86cd13cde1119175de2b205e63939ef', pr_url=None,
repo_url=RepoUrl('https://huggingface.co/Suleynan/my-finetuned-bert').
```

```
1 # Use a pipeline as a high-level helper
2 from transformers import pipeline
3
4 pipe = pipeline("text-classification", model="Suleynan/my-finetuned-bert")
```

✓ Step5: Download Model from HuggingFace Cloud Platform

- HuggingFace Model 版本云端管理
 - 云端保存: 不用担心本地丢失、格式错乱、移动麻烦
 - 随时调用: 在不同项目、不同电脑、不同的同事可复用
 - 可共享: 可设为公开与他人共享
 - 支持继续训练: 可从中加载参数继续微调任务
 - **Spaces功能部署**: 可以一键在 Hugging Face Spaces 上部署成 Web 应用 (免费、零后
端部署的 Web 应用平台)
- HuggingFace Model Type:

- Public Model: 直接调用即可
- Private Model: 多加一项 "login"

```
# Work for Private Model Load
from huggingface_hub import login
login("Hugging_face_Access-Token") # 只需运行一次即可
```

```
# user_name: Suleynan
# model_name: xxx
model_name = 'user_name/model_name'
```

✓ 5.1: (Normal) Download from Colab Platform & Pred

```
1 # Load model directly
2 from transformers import AutoTokenizer, AutoModelForSequenceClassification
3 import torch
4
5 # 直接填写保存在 HuggingFace 的模型名字
6 model_name = "Suleynan/my-finetuned-bert"
7
8 # 直接调用存储在 HuggingFace 的云端模型
9 tokenizer = AutoTokenizer.from_pretrained(model_name)
10 model = AutoModelForSequenceClassification.from_pretrained(model_name)
11
12 text = "The movie was surprisingly good and emotional."
13 inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
14
15 # pred
16 with torch.no_grad():
17     outputs = model(**inputs)
18
19 # get type
20 logits = outputs.logits
21 predicted_class_id = torch.argmax(logits, dim=1).item()
22 print("Predicted label:", predicted_class_id) # 0 or 1
```

⇒ Predicted label: 0

✓ 5.2 (Faster) Fast Hugging Face pipeline

```
1 from transformers import pipeline
2
3 classifier = pipeline("text-classification", model="Suleynan/my-finetuned-ber
4
5 result = classifier("The acting was absolutely amazing.")
```

```
6 print(result)
7
```

⇒ Device set to use cpu
[{'label': 'LABEL_0', 'score': 0.6622660756111145}]

✓ 5.3 (Web-Deploy) 'Space' Web Function Implementation in HuggingFace Cloud Platform

- 1. enter the Huggingface 'Space' part
- 2. then click 'new space' to create a new space platform for your new Web
- 3. edit the 'file' part as fellow
 - just directly copy following content into certain code file 📌 📌 📌

5.3.1 Space [app.py] File Deploy

```
# 1: app.py
import gradio as gr
from transformers import AutoTokenizer, AutoModelForSequenceClassification
import torch

# 加载模型和 tokenizer (从 Hugging Face Hub)
model_name = "Suleynan/my-finetuned-bert"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(model_name)

# 文本分类函数
def classify_text(text):
    inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=True)
    with torch.no_grad():
        outputs = model(**inputs)
        probs = torch.nn.functional.softmax(outputs.logits, dim=-1)
    pred = torch.argmax(probs).item()
    confidence = probs[0][pred].item()

    if pred == 1:
        result = 'Positive'
    else:
        result = 'Negative'
    label = f"{result} | Label: {pred} | Confidence: {confidence:.2f}"

    return label
```

```
# 创建 Gradio 界面
iface = gr.Interface(
    fn=classify_text,
    inputs=gr.Textbox(lines=3, placeholder="Enter text to classify..."),
    outputs="text",
    title="Self-Design BERT Text Emotional Detection",
    description=f"使用模型 `{model_name}` 进行文本分类"
)

iface.launch()
```

5.3.2 Space [requirements.txt] File Deploy

```
# 2: requirements.txt
# additional add the following necessary package
transformers
torch
gradio
```

5.3.3 Space [README.md] File Deploy

- This is a file for Config, So it's a **must-file**

```
title: Simple EmoDetectionModel
emoji: 🗨️
colorFrom: yellow
colorTo: purple
sdk: gradio
sdk_version: 5.0.1
app_file: app.py
pinned: false
license: mit
short_description: 'First attempt to use Huggingface''s Space function to deploy
```

5.3.4 URL Path for Web-Deploy

Copy certain URL Path to get the Web servicer provided by 'HuggingFace' Platform

- https://huggingface.co/spaces/Suleynan/Freshman_EmoSentiment-app

✓ Step6: Cloud Model Remodeling & Fine-Tuning

✓ 6.1: Load Model & Fine-Tuning

```
1 # now we already have one model's cloud path:
2 # 模型路径: Suleynan/my-finetuned-bert
3
4 from transformers import AutoTokenizer, AutoModelForSequenceClassification
5
6 model_name = "Suleynan/my-finetuned-bert"
7
8 tokenizer = AutoTokenizer.from_pretrained(model_name)
9 model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

🔄 /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access private repositories.
warnings.warn()

✓ 6.2: Prepare New Training Data

✓ 6.2.1 Simple Customed New Data

```
1 # from datasets import Dataset
2
3 # new_data = {
4 #     "text": [
5 #         "I love this product. It's amazing!",
6 #         "Absolutely terrible experience.",
7 #         "This is decent but could be better.",
8 #         "Great job, well done!",
9 #         "I wouldn't recommend this to anyone."
10 #     ],
11 #     "label": [1, 0, 1, 1, 0] # 1 = Positive, 0 = Negative
12 # }
13
14 # train_dataset = Dataset.from_dict(new_data)
```

别忘了: 必须要将数据进行tokenize

```
1 !pip install faker
```

🔄 Requirement already satisfied: faker in /usr/local/lib/python3.11/dist-packages
Requirement already satisfied: tzdata in /usr/local/lib/python3.11/dist-packages

✓ 6.2.2 Complex Customed New Data

```
1 import random
2 import numpy as np
3 from faker import Faker
4
5 fake = Faker()
6 np.random.seed(42)
7 random.seed(42)
8
9 # 配置参数
10 NUM_SAMPLES = 1000
11 POSITIVE_RATIO = 0.6 # 60% 正样本
12 NEGATIVE_RATIO = 0.4 # 40% 负样本
13
14 # 语义构建组件
15 products = [
16     "wireless headphones", "smartphone case", "yoga mat", "coffee maker",
17     "fitness tracker", "blender", "desk lamp", "gaming mouse",
18     "water bottle", "laptop sleeve", "scented candle", "cookware set",
19     "gardening tools", "board game", "backpack", "skincare serum",
20     "protein powder", "novel", "throw pillow", "wall art"
21 ]
22
23 features = [
24     "battery life", "user interface", "durability", "noise cancellation",
25     "color accuracy", "setup process", "customer support", "mobile app",
26     "voice recognition", "ergonomic design", "water resistance", "charging s
27     "instruction manual", "warranty policy", "packaging", "software stabilit
28     "size options", "weight distribution", "material quality", "scent longev
29 ]
30
31 positive_modifiers = [
32     "exceptionally", "surprisingly", "astonishingly", "unbelievably",
33     "remarkably", "impressively", "outstandingly", "phenomenally"
34 ]
35
36 negative_modifiers = [
37     "disappointingly", "frustratingly", "unacceptably", "shockingly",
38     "dreadfully", "abysmally", "appallingly", "pitifully"
39 ]
40
41 intensifiers = [
42     "absolutely", "utterly", "completely", "totally", "truly",
43     "genuinely", "seriously", "extremely"
44 ]
45
46 contradictions = [
47     "despite initial concerns", "contrary to expectations", "although skepti
48     "even though I had doubts", "while it's not perfect", "admittedly there
49 ]
50
51 # 情感混合模板 - 修正了capitalize问题
```

```

52 mixed_templates = [
53     ("The {product} is {positive_modifier} {positive_adj} {contradiction}, b
54     ("{contradiction}, the {product} turned out to be {intensifier} {positiv
55     ("I {negative_verb} the {product}'s {feature} which is {negative_modifie
56     ("{contradiction}, I'd {recommendation} this {product} because of its {p
57 ]
58
59 # 情感词库 (扩展版)
60 positive_adjectives = ["superb", "exceptional", "revolutionary", "innovative
61                        "brilliant", "flawless", "premium", "outstanding", "re
62                        "impressive", "splendid", "first-rate", "top-notch", "
63
64 negative_adjectives = ["defective", "subpar", "atrocious", "unreliable", "sh
65                        "inferior", "faulty", "dismal", "appalling", "deplorab
66                        "mediocre", "unsatisfactory", "lacking", "disappointin
67
68 positive_verbs = ["admire", "applaud", "commend", "praise", "appreciate",
69                  "value", "treasure", "enjoy", "adore", "cherish"]
70
71 negative_verbs = ["detest", "despise", "lament", "deplore", "criticize",
72                  "disparage", "bewail", "censure", "condemn", "denounce"]
73
74 # 生成复杂情感语句的函数 - 修复了capitalize问题
75 def generate_complex_sentiment(is_positive):
76     product = random.choice(products)
77     feature = random.choice(features)
78     raw_contradiction = random.choice(contradictions)
79
80     # 随机决定是否将矛盾短语首字母大写
81     use_capitalized = random.choice([True, False])
82     contradiction = raw_contradiction.capitalize() if use_capitalized else r
83
84     if is_positive:
85         # 正面主导的复杂表达
86         template_choice = random.choices(
87             [t[0] for t in mixed_templates],
88             weights=[t[1] for t in mixed_templates]
89         )[0]
90
91         replacements = {
92             'product': product,
93             'feature': feature,
94             'contradiction': contradiction,
95             'positive_modifier': random.choice(positive_modifiers),
96             'negative_modifier': random.choice(negative_modifiers),
97             'positive_adj': random.choice(positive_adjectives),
98             'negative_adj': random.choice(negative_adjectives),
99             'intensifier': random.choice(intensifiers),
100             'negative_verb': random.choice(negative_verbs),
101             'recommendation': random.choice(['highly recommend', 'definitely
102         }
103     else:
104         # 负面主导的复杂表达
105         template_choice = random.choices(
106             [t[0] for t in mixed_templates],

```



```

107         weights=[1-t[1] for t in mixed_templates] # 反转权重
108     )[0]
109
110     replacements = {
111         'product': product,
112         'feature': feature,
113         'contradiction': contradiction,
114         'positive_modifier': random.choice(positive_modifiers),
115         'negative_modifier': random.choice(negative_modifiers),
116         'positive_adj': random.choice(positive_adjectives),
117         'negative_adj': random.choice(negative_adjectives),
118         'intensifier': random.choice(intensifiers),
119         'negative_verb': random.choice(negative_verbs),
120         'recommendation': random.choice(['cannot recommend', 'would caut
121     }
122
123     return template_choice.format(**replacements)
124
125 # 生成数据集
126 texts = []
127 labels = []
128
129 for _ in range(NUM_SAMPLES):
130     # 控制正负样本比例
131     is_positive = random.random() < POSITIVE_RATIO
132
133     # 随机选择生成模式: 简单表达 or 复杂表达
134     if random.random() < 0.7: # 70% 复杂表达
135         text = generate_complex_sentiment(is_positive)
136     else:
137         # 简单直接表达
138         product = random.choice(products)
139         if is_positive:
140             adj = random.choice(positive_adjectives)
141             text = f"This {product} is {random.choice(intensifiers)} {adj}!"
142         else:
143             adj = random.choice(negative_adjectives)
144             text = f"Avoid this {product} - {random.choice(negative_modifier
145
146     texts.append(text)
147     labels.append(1 if is_positive else 0)
148
149 # 添加10%的中性/模糊表达 (随机标记)
150 for _ in range(int(NUM_SAMPLES * 0.1)):
151     product = random.choice(products)
152     text = fake.sentence(ext_word_list=products+features).replace('.', '') +
153     texts.append(text)
154     labels.append(random.randint(0,1))
155
156 # 构建最终数据集
157 complex_dataset = {
158     "text": texts,
159     "label": labels
160 }
161

```

```

162
163 # 验证数据集
164 print(f"Generated {len(complex_dataset['text'])} samples")
165 print(f"Positive ratio: {sum(labels)/len(labels):.2f}")
166 print("\nSample data:")
167 for i in range(5):
168     print(f"{i+1}. [{labels[i]}] {texts[i]}")
169
170 # 保存数据集到CSV文件 (可选)
171 # import pandas as pd
172 # df = pd.DataFrame(complex_dataset)
173 # df.to_csv("complex_sentiment_dataset.csv", index=False)
174 # print("\nDataset saved to 'complex_sentiment_dataset.csv'")
175
176 new_data = complex_dataset

```

Generated 1100 samples
Positive ratio: 0.61

Sample data:

1. [0] I deplore the water bottle's mobile app which is appallingly defective
2. [1] Even though i had doubts, the desk lamp turned out to be completely fi
3. [1] The scented candle is impressively first-rate despite initial concerns
4. [0] I despise the novel's ergonomic design which is disappointingly unrelia
5. [0] contrary to expectations, the backpack turned out to be completely rem

6.2.3 New Data Tokenized

```

1 def tokenize_function(example):
2     return tokenizer(example["text"], truncation=True, padding="max_length")
3
4 from datasets import Dataset
5
6 # new_data = {
7 #     "text": [
8 #         "I love this product. It's amazing!",
9 #         "Absolutely terrible experience.",
10 #         "This is decent but could be better.",
11 #         "Great job, well done!",
12 #         "I wouldn't recommend this to anyone."
13 #     ],
14 #     "label": [1, 0, 1, 1, 0] # 1 = Positive, 0 = Negative
15 # }
16
17 train_dataset = Dataset.from_dict(new_data)
18
19 train_dataset = train_dataset.map(tokenize_function, batched=True)
20 train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask']

```

Map: 100%

1100/1100 [00:01<00:00, 895.02 examples/s]

✓ 6.3: Define Training params & Trainer

```
1 !pip install --upgrade numpy datasets transformers
```

```

⇒ Requirement already satisfied: numpy in /usr/local/lib/python3.11/dist-packages (1.26.4)
Requirement already satisfied: datasets in /usr/local/lib/python3.11/dist-packages (2.18.0)
Requirement already satisfied: transformers in /usr/local/lib/python3.11/dist-packages (4.41.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.11/dist-packages (3.13.1)
Requirement already satisfied: pyarrow>=15.0.0 in /usr/local/lib/python3.11/dist-packages (15.0.2)
Requirement already satisfied: dill<0.3.9,>=0.3.0 in /usr/local/lib/python3.11/dist-packages (0.3.8)
Requirement already satisfied: pandas in /usr/local/lib/python3.11/dist-packages (2.1.4)
Requirement already satisfied: requests>=2.32.2 in /usr/local/lib/python3.11/dist-packages (2.32.3)
Requirement already satisfied: tqdm>=4.66.3 in /usr/local/lib/python3.11/dist-packages (4.66.3)
Requirement already satisfied: xxhash in /usr/local/lib/python3.11/dist-packages (3.5.0)
Requirement already satisfied: multiprocessing in /usr/local/lib/python3.11/dist-packages (3.0.3)
Requirement already satisfied: fsspec<=2025.3.0,>=2023.1.0 in /usr/local/lib/python3.11/dist-packages (2024.10.0)
Requirement already satisfied: huggingface-hub>=0.24.0 in /usr/local/lib/python3.11/dist-packages (0.24.0)
Requirement already satisfied: packaging in /usr/local/lib/python3.11/dist-packages (24.1)
Requirement already satisfied: pyyaml>=5.1 in /usr/local/lib/python3.11/dist-packages (6.0.2)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.11/dist-packages (2024.7.24)
Requirement already satisfied: tokenizers<0.22,>=0.21 in /usr/local/lib/python3.11/dist-packages (0.21.0)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.11/dist-packages (0.5.1)
Requirement already satisfied: aiohttp!=4.0.0a0,!4.0.0a1 in /usr/local/lib/python3.11/dist-packages (3.10.10)
Requirement already satisfied: typing-extensions>=3.7.4.3 in /usr/local/lib/python3.11/dist-packages (4.11.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.2 in /usr/local/lib/python3.11/dist-packages (1.1.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.11/dist-packages (3.4.0)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.11/dist-packages (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.11/dist-packages (2.2.3)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.11/dist-packages (2025.7.11)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.11/dist-packages (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.11/dist-packages (2024.1)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.11/dist-packages (2024.1)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.11/dist-packages (2.4.4)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.11/dist-packages (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.11/dist-packages (25.3.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.11/dist-packages (1.5.0)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.11/dist-packages (6.1.0)
Requirement already satisfied: propcache>=0.2.0 in /usr/local/lib/python3.11/dist-packages (0.2.0)
Requirement already satisfied: yarl<2.0,>=1.17.0 in /usr/local/lib/python3.11/dist-packages (1.18.3)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.11/dist-packages (1.17.0)

```

```

1 from transformers import TrainingArguments, Trainer
2 import torch
3
4 # 确保数据集格式正确
5 train_dataset = train_dataset.map(
6     lambda example: tokenizer(example['text'], padding='max_length', truncation
7     batched=True
8 )
9
10 # 设置数据集格式为PyTorch张量
11 train_dataset.set_format(
12     type='torch',
13     columns=['input_ids', 'attention_mask', 'labels']
14 )

```

```
15
16 # 重命名标签列为'labels' (Trainer期望的默认列名)
17 # train_dataset = train_dataset.rename_column("labels", "labels")
18
19 # 创建训练参数
20 training_args = TrainingArguments(
21     output_dir="./results",
22     num_train_epochs=2,
23     per_device_train_batch_size=8,
24     logging_dir='./logs',
25     logging_steps=10,
26     save_strategy="epoch",
27     # 添加以下参数可预防类似问题
28     remove_unused_columns=True, # 自动移除未使用的列
29     dataloader_num_workers=4,   # 使用多进程加载数据
30 )
31
32 # 创建Trainer
33 trainer = Trainer(
34     model=model,
35     args=training_args,
36     train_dataset=train_dataset,
37     # 添加数据整理器
38     data_collator=lambda data: {
39         'input_ids': torch.stack([f['input_ids'] for f in data]),
40         'attention_mask': torch.stack([f['attention_mask'] for f in data]),
41         'labels': torch.stack([f['labels'] for f in data])
42     }
43 )
44
45 # 开始训练
46 trainer.train()
```



Map: 100%

1100/1100 [00:00<00:00, 1364.96 examples/s]

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: U:
warnings.warn(
```

 [276/276 03:38, Epoch 2/2]**Step Training Loss**

10	0.642000
20	0.599900
30	0.515300
40	0.545500
50	0.554200
60	0.507000
70	0.493600
80	0.454700
90	0.407300
100	0.419000
110	0.404300
120	0.376600
130	0.375100
140	0.507400
150	0.412700
160	0.435300
170	0.399500
180	0.371200
190	0.317100
200	0.348600
210	0.389100
220	0.430400
230	0.427100
240	0.440600
250	0.411800
260	0.540800
270	0.374700

```
/usr/local/lib/python3.11/dist-packages/torch/utils/data/dataloader.py:624: U:
```

```
1 # 登录 Hugging Face
2 from huggingface_hub import login
```

```

3 # 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
4 login("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") # 登录一次即可
5
6 # 推送模型（更新原来的）
7 model.push_to_hub("Suleynan/my-finetuned-bert")
8 tokenizer.push_to_hub("Suleynan/my-finetuned-bert")

```



README.md: 5.17k/? [00:00<00:00, 157kB/s]

Uploading...: 100%

438M/438M [00:09<00:00, 38.1MB/s]

```

CommitInfo(commit_url='https://huggingface.co/Suleynan/my-finetuned-
bert/commit/1fdee6f86fa9c5b5dd771c379eea8a8a232344bc', commit_message='Upload
tokenizer', commit_description='',
oid='1fdee6f86fa9c5b5dd771c379eea8a8a232344bc', pr_url=None,
repo_url=RepoUrl('https://huggingface.co/Suleynan/my-finetuned-bert').

```

✓ Step7: Optimizing Model Deployment Structure (Pipeline)

✓ 7.1 Check Events Prob

```

1 # now we already have one model's cloud path:
2 # 模型路径: Suleynan/my-finetuned-bert
3
4 from transformers import AutoTokenizer, AutoModelForSequenceClassification
5
6 model_name = "Suleynan/my-finetuned-bert"
7
8 tokenizer = AutoTokenizer.from_pretrained(model_name)
9 model = AutoModelForSequenceClassification.from_pretrained(model_name)

```



/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Use
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access private repos.
warnings.warn(

tokenizer_config.json: 1.27k/? [00:00<00:00, 26.0kB/s]

vocab.txt: 232k/? [00:00<00:00, 6.56MB/s]

tokenizer.json: 712k/? [00:00<00:00, 13.8MB/s]

special_tokens_map.json: 100%

695/695 [00:00<00:00, 15.3kB/s]

config.json: 100%

687/687 [00:00<00:00, 17.6kB/s]

model.safetensors: 100%

438M/438M [00:14<00:00, 57.2MB/s]

```

1 import torch
2 import torch.nn.functional as F

```

```
3 from transformers import AutoTokenizer, AutoModelForSequenceClassification
4
5 # 加载模型与 tokenizer (请根据你的模型路径调整)
6 model_name = "Suleynan/my-finetuned-bert"
7 tokenizer = AutoTokenizer.from_pretrained(model_name)
8 model = AutoModelForSequenceClassification.from_pretrained(model_name)
9
10 # 示例文本列表
11 texts = [
12     "I absolutely hated the story!",
13     "I absolutely love the story!",
14     "The movie was boring and way too long.",
15     "This product exceeded my expectations!",
16     "Terrible customer service experience.",
17     "What a fantastic performance by the lead actor!",
18     "I wouldn't recommend this to anyone.",
19     "Highly recommended, I loved every bit of it!",
20     "The food was cold and tasteless.",
21     "Amazing service and friendly staff!"
22 ]
23
24 # 批量处理文本
25 for text in texts:
26     # 编码文本
27     inputs = tokenizer(text, return_tensors="pt", truncation=True, padding=Tr
28     with torch.no_grad():
29         outputs = model(**inputs)
30
31     logits = outputs.logits
32     probs = F.softmax(logits, dim=1) # 计算概率
33     predicted_class = torch.argmax(probs, dim=1).item()
34     probs_list = probs.squeeze().tolist()
35
36     # 打印结果
37     print(f"Text: {text}")
38     print(f"Predicted Label: {predicted_class}")
39     for idx, prob in enumerate(probs_list):
40         print(f"  Label {idx}: {prob:.4f}")
41     print('-' * 60)
42
```

⇒ Text: I absolutely hated the story!

Predicted Label: 1

Label 0: 0.0020

Label 1: 0.9980

Text: I absolutely love the story!

Predicted Label: 1

Label 0: 0.0011

Label 1: 0.9989

Text: The movie was boring and way too long.

Predicted Label: 1

Label 0: 0.4291

Label 1: 0.5709

Text: This product exceeded my expectations!

Predicted Label: 1

Label 0: 0.0014

Label 1: 0.9986

Text: Terrible customer service experience.

Predicted Label: 0

Label 0: 0.5209

Label 1: 0.4791

Text: What a fantastic performance by the lead actor!

Predicted Label: 1

Label 0: 0.0011

Label 1: 0.9989

Text: I wouldn't recommend this to anyone.

Predicted Label: 0

Label 0: 0.9817

Label 1: 0.0183

Text: Highly recommended, I loved every bit of it!

Predicted Label: 1

Label 0: 0.0013

Label 1: 0.9987

Text: The food was cold and tasteless.

Predicted Label: 1

Label 0: 0.4179

Label 1: 0.5821

Text: Amazing service and friendly staff!

Predicted Label: 1

Label 0: 0.0010

Label 1: 0.9990

✓ 7.2 Pipeline Modular Packaging Process

```
# Model Pipeline Structure
my_finetune_pipeline/
├─ config.py           # 配置文件（模型名、路径、参数等）
├─ data_utils.py       # 数据加载与预处理模块
├─ model_utils.py      # 模型加载与封装模块
├─ train.py            # 训练模块
├─ evaluate.py         # 评估模块
├─ pipeline.py         # 管道总调度器
└─ run_pipeline.py     # 主运行入口（调用 pipeline）
```

7.2.1 cmd conduction


```
# (base) suleynan_suir@Suleynan-Auirs-Laptop # 当前的终端环境位于 Laptop 本地 (Laptop)

# 切换到 Desktop
cd ~/Desktop

# 创建文件夹 (创建后, 并没有进入当前文件夹)
mkdir my_opt_finetuned_pipeline

# 切换到 generated doc
cd my_opt_finetuned_pipeline

# 创建代码文件
touch config.py data_utils.py model_utils.py train.py evaluate.py pipeline.py run.py

# 运行代码命令行 (虚拟环境 / 本地环境)
# 本地环境 => 直接 python + file_name.py
# 虚拟环境: 找到 Anaconda文件夹下的目前 envs 子目录下的 python 文件
# 虚拟环境 => virtual_env_path/python file_name.py
/Users/suleynan_suir/Desktop/Anaconda/anaconda3/envs/torchCPU_env/bin/python run.py

# 重复上一条用户输入的命令
!!
```

✓ 7.2.2 IDE File

✓ (1) config.py [Config File]

- 定义需要 load 的云端模型
- 超参数设定
- 数据文件设定

```
1 # config.py
2 MODEL_NAME = "Suleynan/my-finetuned-bert"
3 NUM_LABELS = 2
4 BATCH_SIZE = 16
5 EPOCHS = 3
6 LR = 2e-5
7 MAX_LENGTH = 128
8 TRAIN_FILE = "train.csv"
9 TEST_FILE = "test.csv"
```

✓ (2) data_utils.py [Encode File]

- 定义func: 加载 + tokenize 向量化数据

```

1 # data_utils.py
2 from datasets import load_dataset
3 from transformers import AutoTokenizer
4 from config import MODEL_NAME, MAX_LENGTH
5
6 def load_and_tokenize_data(train_file, test_file):
7     tokenizer = AutoTokenizer.from_pretrained(MODEL_NAME)
8     dataset = load_dataset("csv", data_files={"train": train_file, "test": te
9
10     def tokenize_fn(example):
11         return tokenizer(example["text"], truncation=True, padding="max_lengt
12
13     dataset = dataset.map(tokenize_fn)
14     dataset.set_format(type='torch', columns=['input_ids', 'attention_mask'],
15     return dataset, tokenizer
16

```

✓ (3) model_utils.py [Load Model]

```

1 # model_utils.py
2 from transformers import AutoModelForSequenceClassification
3 from config import MODEL_NAME, NUM_LABELS
4
5 def load_model():
6     model = AutoModelForSequenceClassification.from_pretrained(MODEL_NAME, nu
7     return model
8

```

✓ (4) evaluate.py [Eval Model]

```

1 # evaluate.py
2
3 import torch
4 import torch.nn.functional as F
5 import wandb
6
7 def evaluate_model(trainer):
8     predictions = trainer.predict(trainer.eval_dataset)
9     logits = predictions.predictions
10    labels = predictions.label_ids
11
12    probs = F.softmax(torch.tensor(logits), dim=1).numpy()
13    confidences = probs.max(axis=1) # 每条样本最大概率
14
15    avg_confidence = confidences.mean()
16    wandb.log({"eval/avg_confidence": avg_confidence})
17
18    # 可选打印置信度分布
19    print(f"🇨🇳 Evaluation: Avg Confidence = {avg_confidence:.4f}")

```

```

20 print("Evaluation Results:", predictions.metrics)
21

```

✓ (5) losses.py [Loss Func]

```

1 # losses.py
2 import torch
3 import torch.nn as nn
4 import torch.nn.functional as F
5
6 class FocalLoss(nn.Module):
7     def __init__(self, gamma=2.0, alpha=None, reduction='mean', label_smoothing=0.0):
8         super(FocalLoss, self).__init__()
9         self.gamma = gamma
10        self.alpha = alpha
11        self.reduction = reduction
12        self.label_smoothing = label_smoothing
13
14    def forward(self, inputs, targets):
15        num_classes = inputs.size(1)
16        if self.label_smoothing > 0:
17            smooth_labels = torch.full_like(inputs, self.label_smoothing / (num_classes - 1))
18            smooth_labels.scatter_(1, targets.unsqueeze(1), 1.0 - self.label_smoothing)
19            targets = smooth_labels
20            log_probs = F.log_softmax(inputs, dim=1)
21            loss = -torch.sum(targets * log_probs, dim=1)
22        else:
23            log_probs = F.log_softmax(inputs, dim=1)
24            targets_one_hot = F.one_hot(targets, num_classes).float()
25            pt = torch.exp(torch.sum(log_probs * targets_one_hot, dim=1))
26            loss = -((1 - pt) ** self.gamma) * torch.sum(log_probs * targets_one_hot, dim=1)
27
28        return loss.mean() if self.reduction == 'mean' else loss.sum()
29

```

✓ (6) train.py [Train Model]

```

1 # train.py
2
3 from transformers import Trainer, TrainingArguments
4 import wandb
5 from losses import FocalLoss
6 import torch
7
8 class CustomTrainer(Trainer):
9     def __init__(self, *args, focal_gamma=2.0, label_smoothing=0.0, **kwargs):
10         super().__init__(*args, **kwargs)
11         self.loss_func = FocalLoss(gamma=focal_gamma, label_smoothing=label_smoothing)
12
13     def compute_loss(self, model, inputs, return_outputs=False, num_items_in_batch=None):
14         labels = inputs.get("labels")

```

```

15     outputs = model(**inputs)
16     logits = outputs.get("logits")
17     loss = self.loss_func(logits, labels)
18     return (loss, outputs) if return_outputs else loss
19
20 def train_model(model, tokenizer, dataset, epochs, batch_size, lr):
21     wandb.init(project="bert-finetune-pipeline", name="run-" + wandb.util.gen
22
23     training_args = TrainingArguments(
24         output_dir="./results",
25         num_train_epochs=epochs,
26         per_device_train_batch_size=batch_size,
27         per_device_eval_batch_size=batch_size,
28         eval_strategy="epoch",
29         save_strategy="epoch",
30         logging_dir="./logs",
31         logging_steps=10,
32         learning_rate=lr,
33         load_best_model_at_end=True,
34         report_to="wandb"
35     )
36
37     trainer = CustomTrainer(
38         model=model,
39         args=training_args,
40         train_dataset=dataset["train"],
41         eval_dataset=dataset["test"],
42         tokenizer=tokenizer,
43         focal_gamma=2.0,
44         label_smoothing=0.1,
45     )
46     trainer.train()
47
48
49
50     return trainer

```

▼ (7) upload_to_huggingface.py

```

1 import os
2 from transformers import AutoModelForSequenceClassification, AutoTokenizer
3 from huggingface_hub import login, create_repo, upload_folder
4 from huggingface_hub.utils import HfHubHTTPError
5
6 def upload_model_to_hub(
7     local_model_dir: str,
8     repo_id: str,
9     create_readme: bool = True,
10     hf_token: str = None
11 ):
12     """
13     将本地保存的 transformer 模型上传至 Hugging Face Hub。
14

```

```

15 参数:
16  - local_model_dir: str, 本地保存模型和 tokenizer 的目录
17  - repo_id: str, Hugging Face Hub 上的完整仓库名, 如 "username/my-model"
18  - create_readme: bool, 是否自动生成 README.md
19  - hf_token: str, 可选, Hugging Face Access Token (建议使用 login())
20
21 使用示例:
22  upload_model_to_hub("/path/to/model", "username/my-finetuned-bert")
23  """"
24
25  # ✅ 登录 (仅首次需要)
26  try:
27      login(token=hf_token) if hf_token else login()
28  except Exception as e:
29      print("❌ 登录失败, 请确认 token 是否有效。")
30      raise e
31
32  # ✅ 检查模型路径
33  if not os.path.exists(local_model_dir):
34      raise FileNotFoundError(f"模型目录不存在: {local_model_dir}")
35
36  print(f"📁 加载本地模型和 tokenizer: {local_model_dir}")
37  model = AutoModelForSequenceClassification.from_pretrained(local_model_dir)
38  tokenizer = AutoTokenizer.from_pretrained(local_model_dir, local_files_only=True)
39
40  # ✅ 上传模型和 tokenizer
41  print(f"🚀 正在上传到 Hugging Face Hub 仓库: {repo_id}")
42  try:
43      create_repo(repo_id, exist_ok=True)
44  except HfHubHTTPError as e:
45      print(f"❌ 创建仓库失败: {e}")
46      raise e
47
48  # ✅ 可选: 自动生成 README.md
49  if create_readme:
50      readme_path = os.path.join(local_model_dir, "README.md")
51      if not os.path.exists(readme_path):
52          with open(readme_path, "w") as f:
53              f.write(f"# {repo_id.split('/')[1]}\n\nThis is a fine-tuned")
54          print("📄 已自动创建 README.md")
55
56  # ✅ 上传整个模型文件夹 (包括 config、tokenizer、README 等)
57  upload_folder(folder_path=local_model_dir, repo_id=repo_id, commit_message="Upload model")
58
59  print(f"✅ 模型成功上传到: https://huggingface.co/{repo_id}")
60
61  # ✅ 如果你直接运行脚本, 这里设置模型路径和仓库名
62  if __name__ == "__main__":
63      model_path = "/Users/suleynan_suir/Desktop/my_opt_finetuned_pipeline/my_t"
64      huggingface_repo = "Suleynan/my-finetuned-bert"
65
66      upload_model_to_hub(local_model_dir=model_path, repo_id=huggingface_repo)
67

```

▼ (8) pipeline.py [Run Pipeline]

```

1 # pipeline.py
2
3 from data_utils import load_and_tokenize_data
4 from model_utils import load_model
5 from train import train_model
6 from evaluate import evaluate_model
7 from config import TRAIN_FILE, TEST_FILE
8 from upload_to_huggingface import upload_model_to_hub
9
10 def run_pipeline(epochs, batch_size, lr):
11     dataset, tokenizer = load_and_tokenize_data(TRAIN_FILE, TEST_FILE)
12     model = load_model()
13     import os
14     os.environ["WANDB_MODE"] = "disabled"
15
16     trainer = train_model(model, tokenizer, dataset, epochs, batch_size, lr)
17     evaluate_model(trainer)
18
19     # 保存
20     output_dir = "my_tunned_model"
21     trainer.save_model(output_dir)
22     tokenizer.save_pretrained(output_dir)
23
24     # ✅ 上传
25     upload_model_to_hub(local_model_dir=output_dir, repo_id="Suleynan/my-fine
26
27

```

▼ (9) run_pipeline.py [Total Run Pipeline]

```

1 import argparse
2 from pipeline import run_pipeline
3
4
5 import sys
6 import transformers
7
8 print("Python Executable:", sys.executable)
9 print("Transformers version:", transformers.__version__)
10
11 if __name__ == "__main__":
12     parser = argparse.ArgumentParser(description="Fine-tune BERT with pipelin
13     parser.add_argument("--epochs", type=int, default=3, help="Number of trai
14     parser.add_argument("--batch_size", type=int, default=16, help="Batch siz
15     parser.add_argument("--lr", type=float, default=2e-5, help="Learning rate
16     args = parser.parse_args()
17
18     run_pipeline(args.epochs, args.batch_size, args.lr)
19

```

✓ Step8: New Eval Model

```
1 from transformers import AutoTokenizer, AutoModelForSequenceClassification
2
3 model_name = "Suleynan/my-finetuned-bert"
4
5 tokenizer = AutoTokenizer.from_pretrained(model_name)
6 model = AutoModelForSequenceClassification.from_pretrained(model_name)
```

➡ /usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94: Using the secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab. You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access private repositories.

tokenizer_config.json: 1.46k/? [00:00<00:00, 29.6kB/s]

vocab.txt: 232k/? [00:00<00:00, 6.52MB/s]

tokenizer.json: 712k/? [00:00<00:00, 15.4MB/s]

special_tokens_map.json: 100% 695/695 [00:00<00:00, 28.6kB/s]

config.json: 100% 687/687 [00:00<00:00, 24.9kB/s]

model.safetensors: 100% 438M/438M [00:07<00:00, 104MB/s]

```
1 from transformers import pipeline
2
3 classifier = pipeline("text-classification", model="Suleynan/my-finetuned-bert")
4
5 result = classifier("I hate you")
6 print(result)
```

➡ Device set to use cuda:0
[{'label': 'LABEL_0', 'score': 0.830632746219635}]

✓ Step9: Display of finished products in Space

Want to try? Click 📌 📌 📌

https://huggingface.co/spaces/Suleynan/simple_emoDetectionModel

As a crazy fan of Taylor and currently fall in love with this TTDP, and this Self-Design BERT knows what I am thinking about this song!!! 🥰

