

AI 文本偵測與微調完整對話紀錄

日期範圍：2025年12月21日 - 2025年12月25日

主題：AI生成文本偵測模型建構、Transformers微調、資料集整理、對抗性測試

目錄

1. 初始需求與架構設計
2. Transformers微調詳細指南
3. 資料集選用建議
4. 對抗性範例實作
5. SEO優化與內容生成

1. 初始需求與架構設計

用戶提問

我有一份作業，建立一個簡單的 AI vs Human 文章分類工具。

最低需求：

1. 使用者輸入一段文本 → 立即顯示判斷結果 (AI% / Human%)
2. 可採用 sklearn / transformers / 自建特徵法
3. 使用 Streamlit 作為 UI
4. 可視化統計量

回應概要

建議採用 TF-IDF + Logistic Regression (快速版) 或 Transformers 微調 (高準確版) 的雙軌方案。

整體架構設計

- 模型部分 (offline訓練)：
 - 收集 AI 文章 (ChatGPT生成) + 真人文章，標記 label (AI=1, Human=0)
 - TfidfVectorizer 轉向量，ngram_range=(1,2)，max_features=5000
 - LogisticRegression 或 LinearSVC 訓練分類器
 - joblib 儲存 vectorizer 和 model
- 線上推論 (Streamlit app)：
 - 啟動時載入已訓練模型
 - 使用者貼文字按按鈕
 - 即時輸出 AI% / Human%
 - 視覺化：機率條圖 + 文字統計量 (長度、詞彙數、不重複詞數)

模型訓練範例

train_ai_human_classifier.py

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import classification_report
import joblib
```

**載入資料：csv 欄位 text, label (AI=1,
Human=0)**

```
df = pd.read_csv("ai_human_corpus.csv")

X_train, X_test, y_train, y_test = train_test_split(
    df["text"], df["label"], test_size=0.2, random_state=42, stratify=df["label"]
)

vectorizer = TfidfVectorizer(
    max_features=5000,
    ngram_range=(1, 2),
    stop_words=None
)
X_train_vec = vectorizer.fit_transform(X_train)
X_test_vec = vectorizer.transform(X_test)

clf = LogisticRegression(max_iter=1000)
clf.fit(X_train_vec, y_train)

y_pred = clf.predict(X_test_vec)
print(classification_report(y_test, y_pred))

joblib.dump(vectorizer, "tfidf_vectorizer.pkl")
joblib.dump(clf, "ai_human_clf.pkl")
```

Streamlit App 基本框架

app.py

```
import streamlit as st
import joblib
import numpy as np
import matplotlib.pyplot as plt
```

載入模型

```
@st.cache_resource
def load_model():
    vectorizer = joblib.load("tfidf_vectorizer.pkl")
    clf = joblib.load("ai_human_clf.pkl")
    return vectorizer, clf

vectorizer, clf = load_model()

st.title("AI vs Human 文章分類器")
st.write("貼上一段文字，系統會預測是 AI 還是 Human 撰寫，並顯示 AI% / Human% 。")

user_text = st.text_area("輸入待判斷文本：", height=200)

if st.button("開始判斷"):
    if user_text.strip() == "":
        st.warning("請先輸入文本")
    else:
        X_vec = vectorizer.transform([user_text])
        proba = clf.predict_proba(X_vec)[0]
        ai_prob = float(proba[1])
        human_prob = float(proba[0])

        st.subheader("分類結果")
        st.metric("AI 機率", f"{ai_prob*100:.2f}%")
        st.metric("Human 機率", f"{human_prob*100:.2f}%")

# 可視化：條形圖
st.subheader("機率分佈視覺化")
fig, ax = plt.subplots()
ax.bar(["Human", "AI"], [human_prob, ai_prob], color=["#4CAF50", "#FF5722"])
ax.set_ylim(0, 1)
ax.set_ylabel("Probability")
for i, v in enumerate([human_prob, ai_prob]):
    ax.text(i, v + 0.02, f"{v*100:.1f}%", ha="center")
st.pyplot(fig)

# 文本統計量
st.subheader("文本統計量")
char_len = len(user_text)
tokens = user_text.split()
token_len = len(tokens)
```

```
vocab_size = len(set(tokens))

col1, col2, col3 = st.columns(3)
col1.metric("字元數", char_len)
col2.metric("詞數（以空白分詞）", token_len)
col3.metric("不重複詞彙數", vocab_size)
```

作業報告可寫的重點

- **模型方法**：TF-IDF 將文本轉成詞語權重向量，Logistic Regression 做二元分類並輸出機率
- **特徵設計想法**：使用 unigram + bigram，max_features 控制維度，避免過度稀疏與過擬合
- **視覺化設計**：條形圖展示 AI/Human 機率，文本長度、詞彙數等簡單統計探索 AI vs Human 風格差異

2. Transformers 微調詳細指南

用戶提問

如何用 transformers 微調模型判別 AI 生成文本

回應概要

Hugging Face Transformers 可輕鬆微調 BERT-like 模型進行二元分類 (AI=1, Human=0) ，流程包含資料準備、模型載入、微調訓練與儲存。

資料準備步驟

```
import pandas as pd
from datasets import Dataset
from sklearn.model_selection import train_test_split

df = pd.read_csv("ai_human_dataset.csv") # text, label
train_df, test_df = train_test_split(df, test_size=0.2, stratify=df['label'])

train_dataset = Dataset.from_pandas(train_df)
test_dataset = Dataset.from_pandas(test_df)
```

資料來源建議：

- **Human 文本** - 從新聞、書籍、論壇下載真實文章，label=0
- **AI 文本** - 用 ChatGPT/GPT-4 等生成類似主題文章，label=1
- **規模建議** - 至少各 1000 筆，最好平衡 (1:1)，總計 2000+ 筆

完整微調程式碼

```
from transformers import (
    AutoTokenizer, AutoModelForSequenceClassification,
    TrainingArguments, Trainer
)
from datasets import Dataset
```

```
import torch
import numpy as np
from sklearn.metrics import accuracy_score, f1_score
```

1. 載入 tokenizer 與模型 (用 distilbert 輕量快速)

```
model_name = "distilbert-base-uncased" # 或 "bert-base-chinese" (中文)
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForSequenceClassification.from_pretrained(
    model_name, num_labels=2 # 二元分類
)
```

2. Tokenization 函數

```
def tokenize_function(examples):
    return tokenizer(examples['text'], truncation=True, padding='max_length',
                    max_length=512)

train_dataset = train_dataset.map(tokenize_function, batched=True)
test_dataset = test_dataset.map(tokenize_function, batched=True)
```

設定 format 紿 Trainer

```
train_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
test_dataset.set_format(type='torch', columns=['input_ids', 'attention_mask', 'label'])
```

3. 計算指標函數

```
def compute_metrics(eval_pred):
    logits, labels = eval_pred
    predictions = np.argmax(logits, axis=-1)
    return {
        'accuracy': accuracy_score(labels, predictions),
        'f1': f1_score(labels, predictions, average='binary')
    }
```

4. 訓練設定

```
training_args = TrainingArguments(
    output_dir='./ai-human-detector',
    num_train_epochs=3, # 依資料量調整 2-5
    per_device_train_batch_size=8, # 依 GPU 記憶體調整
    per_device_eval_batch_size=8,
    warmup_steps=500,
```

```
weight_decay=0.01,  
logging_dir='./logs',  
evaluation_strategy="epoch",  
save_strategy="epoch",  
load_best_model_at_end=True,  
metric_for_best_model='f1',  
)
```

5. Trainer 啟動訓練

```
trainer = Trainer(  
    model=model,  
    args=training_args,  
    train_dataset=train_dataset,  
    eval_dataset=test_dataset,  
    compute_metrics=compute_metrics,  
    tokenizer=tokenizer,  
)  
  
trainer.train()
```

6. 儲存模型 (給 Streamlit 用)

```
trainers.save_model("./ai_human_model")  
tokenizers.save_pretrained("./ai_human_model")
```

Streamlit 推論整合

```
from transformers import pipeline  
  
@st.cache_resource  
def load_transformer_model():  
    classifier = pipeline(  
        "text-classification",  
        model="./ai_human_model",  
        return_all_scores=True  
)  
    return classifier  
  
classifier = load_transformer_model()
```

推論部分

```
if st.button("開始判斷"):  
    result = classifier(user_text)[0]  
    ai_prob = next(score['score'] for score in result if score['label'] == 'LABEL_1')  
    human_prob = next(score['score'] for score in result if score['label'] == 'LABEL_0')  
    # 顯示 AI% / Human% (同前)
```

關鍵參數調整建議表

參數	建議值	說明
max_length	256-512	AI 文本常較規律，短文足夠
num_train_epochs	3-5	觀察 eval_f1 收斂
learning_rate	2e-5 ~ 5e-5	Transformers 標準
batch_size	8-16	Colab T4 GPU 安全值
模型選擇	distilbert / bert-base	輕量 vs 高準確

預期效能與改進

- 準確率 - 用 2000 筆資料可達 85-95% F1-score
- 改進方向：
 - 加 perplexity、burstiness 等手作特徵
 - 用 RoBERTa 或 DeBERTa (更高準確但較慢)
 - 多模型 ensemble

執行前安裝依賴： pip install transformers datasets torch accelerate evaluate

3. 資料集選用建議

用戶提問

使用哪些資料集可以用於 AI 生成文本偵測微調

回應概要

Hugging Face 和 Kaggle 上有多個公開資料集，大多已預處理成 text + label (0=Human, 1=AI) 格式，適合 Transformers Trainer 直接使用。

推薦 Hugging Face 資料集

資料集名稱	來源	特色	載入方式
artem9k/ai-text-detection-pile	HF	GPT2/3/ChatGPT 生成 + Human，長文優化	load_dataset("artem9k/ai-text-detection-pile")
Ateeqq/AI-and-Human-Generated-Text	HF	平衡資料，適合偵測任務	load_dataset("Ateeqq/AI-and-Human-Generated-Text")
gsingh1-py/train	HF	NYT 真文 + 多 LLM 生成 (Gemma/Mistral/GPT-4o)	load_dataset("gsingh1-py/train")

推薦 Kaggle 資料集

Kaggle LLM Detect AI 比賽資料 (10k essays)

```
from datasets import load_dataset
dataset = load_dataset("csv", data_files="train_essays.csv")
```

label 欄位: generated

其他 Kaggle 選項

- sunilthite/llm-detect-ai-generated-text-dataset
- aknjit/human-vs-ai-text-classification-dataset (5k samples)

快速使用範例

```
from datasets import load_dataset
```

方法1：直接載入 HF 資料集

```
dataset = load_dataset("artem9k/ai-text-detection-pile", split="train")
dataset = dataset.train_test_split(test_size=0.2) # 自動分割
```

方法2：Kaggle CSV 轉換

```
dataset = load_dataset("csv", data_files=
{"train": "train.csv", "test": "test.csv"})
```

重新標籤（若 label 不是 0/1）

```
dataset = dataset.map(lambda x: {"label": int(x["generated"])})
```

接著用前面的 tokenize_function + Trainer...

```
train_dataset = dataset["train"].map(tokenize_function, batched=True)
```

資料集比較表

資料集	總筆數	AI 來源	語言	優點	下載連結
ai-text-detection-pile	數萬	GPT2/3/J/C hatGPT	英文	長文、多模型	HF
Kaggle LLM Detect	14k	多 LLM	英文	比賽驗證、標準 benchmark	Kaggle
AI-and-Human-Generated	數千	通用 AI	英文	平衡、易用	HF
AH&AITD	11k	多領域	多語	領域多樣	Figshare

中文資料建議

中文資料較少，可用以下策略：

1. 合成資料 - 用 ChatGPT/Claude 生成中文文章 + 爬 PTT/新聞當 Human 文本
2. 翻譯資料 - 取英文資料集，用 googletrans 翻譯成中文微調
3. 混合 - bert-base-chinese + 英文資料集（跨語言能力強）

合成中文範例（作業加分）

```
human_texts = ["從 PTT 爬的文章...", "新聞內容..."] # label=0
ai_texts = ["請用 GPT 生成類似主題的中文文章"] # label=1
df = pd.DataFrame({
    "text": human_texts + ai_texts,
    "label": [0]*len(human_texts) + [1]*len(ai_texts)
})
```

立即可跑完整程式

```
dataset = load_dataset("artem9k/ai-text-detection-pile", split="train_5000")
dataset = dataset.train_test_split(test_size=0.2, seed=42)
```

直接接前面微調程式碼，30 分鐘內出模型！

優先推薦 - 從 artem9k/ai-text-detection-pile 開始，免下載、即用、品質高。

4. 對抗性範例實作

用戶提問

如何實作對抗性範例來測試偵測模型健壯性

回應概要

對抗性範例（Adversarial Examples）是專門設計來欺騙 AI 文本偵測模型的輸入，常見方法包括改寫（paraphrasing）、同義詞替換、back-translation 等，可有效測試模型健壯性。

常見對抗攻擊類型表

攻擊類型	方法	難度	效果
同義詞替換	單詞 → 同義詞	低	破壞詞頻特徵
改寫（Paraphrasing）	用 LLM 重述	中	最常見、語義保留
Back-translation	英→德→英	中	語法變化
GCG Attack	優化 token 序列	高	白箱最強
添加噪音	插入/刪除字元	低	測試魯棒性

同義詞替換 + Paraphrasing 攻擊

```
import nlpaug.augmenter.word as naw  
from transformers import pipeline
```

載入你的偵測模型

```
classifier = pipeline("text-classification", model="./ai_human_model")
```

同義詞替換攻擊器

```
aug_synonym = naw.SynonymAug(aug_src='wordnet')  
  
def create_adversarial_examples(texts, attack_type="synonym"):  
    adv_examples = []  
    for text in texts:  
        if attack_type == "synonym":  
            adv_text = aug_synonym.augment(text)[0]  
        elif attack_type == "paraphrase":  
            para_pipe = pipeline("text2text-generation",  
                model="t5-small",  
                tokenizer="t5-small")  
            adv_text = para_pipe(f"paraphrase: {text}")[0]['generated_text']  
        adv_examples.append(adv_text)  
    return adv_examples
```

測試

```
original_texts = ["這是 ChatGPT 生成的文章範例。"] * 10  
adv_synonyms = create_adversarial_examples(original_texts, "synonym")
```

評估攻擊成功率

```
orig_preds = [pred['score'] for pred in classifier(original_texts)]  
adv_preds = [pred['score'] for pred in classifier(adv_synonyms)]  
  
success_rate = sum(1 for o, a in zip(orig_preds, adv_preds)  
                   if o > 0.8 and a < 0.5) / len(orig_preds)  
print(f"攻擊成功率: {success_rate:.2%}")
```

Back-translation 攻擊（最簡單有效）

```
from transformers import MarianMTModel, MarianTokenizer
```

英→德→英

```
de_en_model = MarianMTModel.from_pretrained('Helsinki-NLP/opus-mt-en-de')
de_en_tokenizer = MarianTokenizer.from_pretrained('Helsinki-NLP/opus-mt-en-de')
en_de_model = MarianMTModel.from_pretrained('Helsinki-NLP/opus-mt-de-en')
en_de_tokenizer = MarianTokenizer.from_pretrained('Helsinki-NLP/opus-mt-de-en')

def back_translate(text):
    # 英→德
    inputs = en_de_tokenizer([text], return_tensors="pt", padding=True)
    translated_de = en_de_model.generate(**inputs)
    de_text = en_de_tokenizer.decode(translated_de[0], skip_special_tokens=True)

    # 德→英
    inputs = de_en_tokenizer([de_text], return_tensors="pt", padding=True)
    translated_en = de_en_model.generate(**inputs)
    return de_en_tokenizer.decode(translated_en[0], skip_special_tokens=True)
```

生成對抗樣本

```
adv_backtrans = [back_translate(text) for text in original_texts]

LLM 改寫攻擊（最強黑箱攻擊）

def llm_paraphrase(text):
    prompt = f"""
    重寫以下文章，使其聽起來更像人類寫的，不要改變主要意思：
    "{text}"
    """

    response = openai.ChatCompletion.create(
        model="gpt-3.5-turbo",
        messages=[{"role": "user", "content": prompt}]
    )
    return response.choices[0].message.content
```

批次改寫

```
adv_llm = [llm_paraphrase(text) for text in original_texts]

完整測試腳本 + 可視化

import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns

def robustness_test(model, test_texts):
    results = []
```

```

attack_methods = ["original", "synonym", "backtrans", "paraphrase"]

for method in attack_methods:
    if method == "original":
        texts = test_texts
    else:
        texts = create_adversarial_examples(test_texts, method)

    ai_probs = [pred['score'] for pred in model(texts) if pred['label'] == 'LABEL_1']
    results.append({
        'method': method,
        'mean_ai_prob': sum(ai_probs)/len(ai_probs),
        'std_ai_prob': np.std(ai_probs)
    })

df_results = pd.DataFrame(results)
return df_results

```

執行測試

```

test_results = robustness_test(classifier, original_texts)
st.dataframe(test_results) # Streamlit 顯示

```

繪圖

```

fig, ax = plt.subplots()
sns.barplot(data=test_results, x='method', y='mean_ai_prob', ax=ax)
ax.set_title("模型對抗性攻擊健壯性測試")
ax.set_ylabel("平均 AI 機率")
st.pyplot(fig)

```

改進模型健壯性策略

1. 對抗訓練 - 用生成的對抗樣本重新訓練

在訓練資料中加入 30% 對抗樣本

```

train_dataset = concatenate_datasets([train_dataset, adv_dataset])

```

2. 多模型集成 - TF-IDF + Transformer ensemble
3. 特徵層面 - 加 perplexity、burstiness 等語法統計

預期結果表

攻擊方法	一般模型下降	對抗訓練後
同義詞替換	20-30%	10-15%
Paraphrasing	40-60%	20-30%
Back-translation	30-50%	15-25%

作業建議 - 實作「改寫攻擊」 + 「同義詞替換」，用圖表展示成功率，寫一頁「模型健壯性分析報告」。這是業界標準測試方法。

5. SEO 優化與內容生成

用戶提問

為產品頁面撰寫SEO友善的描述和關鍵字建議

回應概要

產品頁 SEO 的核心是圍繞 1 個主關鍵字 + 2–4 個相關長尾關鍵字，並在標題、前兩句、小標題、圖片 alt 與 meta description 中自然出現，同時描述要寫給「人」看的。

SEO 產品頁通用模板

假想產品：環保防滑瑜伽墊（Eco-friendly Yoga Mat）

產品標題（含主關鍵字）

環保防滑瑜伽墊 | 加厚支撐、無毒材質，適合居家與健身房練習

主關鍵字示例：環保瑜伽墊、防滑瑜伽墊

Meta description（150–160 字內）

環保防滑瑜伽墊，採用無毒材質與加厚設計，提供穩定支撐與優異止滑效果，適合瑜伽、皮拉提斯與居家運動。現在下單享免運與限時折扣。

- 自然帶入主關鍵字 + 賣點 + 行動呼籲

SEO 友善產品描述正文

產品特色

- 採用環保無毒材質，低氣味、對肌膚友善，適合長時間練習使用
- 表面特殊防滑紋理，即使流汗也能提供穩定抓地力，減少滑倒風險
- 6mm 加厚設計，兼顧支撐與緩衝，保護膝蓋與關節
- 輕量材質搭配束帶，捲起即可帶出門，方便攜帶上課或旅行

為什麼適合你

- 新手入門需要穩定、安全的瑜伽墊，幫助建立正確姿勢與習慣
- 有固定運動習慣的你，會需要一張耐用、好清潔、不易變形的瑜伽墊
- 如果你在意環保與材質來源，這款瑜伽墊能兼顧舒適體驗與永續選擇

規格與細節

- 尺寸：183 x 61 cm
- 厚度：6 mm
- 材質：環保 TPE / 天然橡膠
- 重量：約 900 g
- 清潔方式：可用濕布擦拭，置於陰涼處自然風乾

使用情境建議

- 適合瑜伽、皮拉提斯、伸展訓練與居家地板運動
- 可搭配瑜伽磚、瑜伽繩等輔具，打造完整練習空間
- 放在臥室或客廳，不使用時捲起收納不佔空間

結構同時兼顧「敘事+賣點+規格」，也自然重複關鍵字與相關語意字詞，有助 SEO。

關鍵字建議拆解

以「環保瑜伽墊」為例：

主關鍵字（1 個）：

- 環保瑜伽墊 / 防滑瑜伽墊

次關鍵字（2-4 個）：

- 加厚瑜伽墊、無毒瑜伽墊、初學者瑜伽墊、瑜伽墊推薦

長尾關鍵字（更貼近搜尋意圖）：

- 環保防滑瑜伽墊 推薦
- 居家運動用 瑜伽墊
- 無毒瑜伽墊 適合新手

放置位置建議：

- 標題 - 主關鍵字 + 1 個賣點（如「環保防滑瑜伽墊 | 加厚支撐、無毒材質」）
- 內文前兩句 - 再自然提一次主關鍵字
- 小標題 - 如「為什麼選擇這款環保瑜伽墊」
- 圖片 alt - alt="環保防滑瑜伽墊 加厚無毒材質"

SEO 友善描述的重點

1. 圍繞 1 個主關鍵字 + 2–4 個相關長尾關鍵字，不要堆砌
 2. 主關鍵字在關鍵位置出現：
 - 產品標題 (H1)
 - 前 1–2 句描述
 - 1–2 個小標題 (H2 / H3)
 - 圖片 alt 文字與 meta description
 3. 描述要寫給「人」看的：清楚、可掃讀、強調好處 (benefits)，再補充規格與細節；避免只是貼廠商制式說明
-

總結與學習要點

核心技術流程

1. 快速原型 - TF-IDF + Logistic Regression，2–3 小時內產出可用模型
2. 深度優化 - Transformers 微調，搭配現成資料集，3–7 天內達到生產級準確率
3. 健壯性測試 - 對抗性攻擊測試，確保模型不易被規避
4. 內容應用 - SEO 優化與風格判別，可應用於內容營運與廣告審核

資料集與模型選擇矩陣

時間壓力	準確率要求	推薦方案
急 (1-2天)	中等 (75-85%)	TF-IDF + Logistic Regression
中等 (3-7天)	高 (85-95%)	Transformers + HF 資料集
充足 (2週+)	很高 (95%+)	對抗訓練 + Ensemble + 客製資料

作業組合建議

- 基礎版 - sklearn 版本 + Streamlit UI + 簡報 1 頁
- 進階版 - 加上 Transformers 微調 + 對抗性測試 + 健壯性分析報告
- 完整版 - 上述全部 + SEO 應用案例展示 + GitHub repo + 部署到 Hugging Face Spaces

後續延伸方向

1. 跨語言支援 - 用 mBERT 或 XLM-RoBERTa 支援多語言偵測
 2. 即時API服務 - 用 FastAPI + Docker 部署模型服務
 3. 企業應用 - 整合到內容審核、廣告管理系統
 4. 學術深化 - 研究 AI 文本特徵 (perplexity、burstiness) 、對抗方法的理論基礎
-

對話完整紀錄至此結束

紀錄日期：2025年12月26日

涵蓋主題：AI文本偵測、Transformers微調、資料集整理、對抗性測試、SEO優化

