網路安全的資料科學_Project

系級：資管三乙　姓名：陳彥融　學號：110306018

一、 專案標題：

Spam mail偵測小助手—使用語言模型來分析郵件內容並判定是否是垃圾郵件。

二、 專案動機：

隨著網路的蓬勃發展，電子郵件如今已成為主要郵件傳輸方式，但其中卻有一種稱為垃圾郵件(spam mail/junk mail)的存在，垃圾郵件指不請自來，未經使用者許可就塞入信箱的濫發電子郵件，內容從較無害的廣告到有危險的詐欺、釣魚網站或惡意軟體等各式各樣，且數量與影響範圍龐大，根據統計，在2023年總計3470億封發送的電子郵件中垃圾郵件就佔了45.6%，因此偵測、過濾、阻擋這些垃圾郵件是很重要的，能降低受到攻擊的機率，以及避免除儲存空間的浪費。有鑑於郵件的內文通常是文字型為主，我便想知道如果使用語言模型去分析郵件內容，是否能精準的判斷出一封郵件是正常郵件還是垃圾郵件。

三、 專案目標：

概觀來說，此專案的目標是使用一個語言模型來分析電子郵件的文字內容，並據此判斷該郵件是正常的或是垃圾郵件，也就是做文字型資料的二元分類。模型部分會使用預訓練好的大型語言模型，並透過垃圾郵件相關的資料集來微調模型參數(fine-tune)，使其從具備各種知識的通才轉變為擅長判別垃圾郵件的專才，還須具備足夠高的分類準確度，並且能夠實際運用於現實生活中。

四、 下游任務：

主要是垃圾郵件分類，分析電子郵件所包含的所有文字內容，判斷此為正常郵件還是垃圾郵件，或可延伸到同為文字分類的應用，例如無用評論分辨或假訊息偵測等。

五、 資料集介紹

我使用的是Kaggle網站上找到的Ling-Spam Dataset，網址如下：

https://www.kaggle.com/datasets/mandygu/lingspam-dataset/data

這是一個包含2893筆電子郵件資料的資料集，其中有2412筆是正常郵件，481筆是垃圾郵件，由Androutsopoulos等人於2000年整理並建立，這些電子郵件都和與語言學相關，包含工作與研究等面向，整份資料集都是英文的純文字資料，非常適合用來做垃圾郵件分類。

這份資料集僅有三個變數，分別如下：

1. subject：電子郵件的主旨，資料型態為文字，含有缺失值。
2. message：電子郵件的內容，資料型態為文字，沒有缺失值，有些資料包含寄件人、收件人或內文長度，但有些僅有內文。
3. label：電子郵件是否是垃圾郵件，資料型態為整數，0代表正常郵件，1代表垃圾郵件，沒有缺失值。

因為這些變數的資料我認為都是有意義的，故微調模型時三個變數都有使用到。

六、 專案實作

完整程式碼連結：
https://colab.research.google.com/drive/170lLx0UekwMRoDdx_Ya6KhAJageRA52L?usp=sharing

實作流程：

1. 處理資料：首先匯入資料，接著做資料前處理，步驟如下：
   i. 將subject欄含有缺失值的資料全部去掉，剩下2831筆資料
   ii. 由於主旨與內容我都希望能保留，故將兩個欄位串接在一起成為新的欄位 “text”，若有寄件收件人或長度等資訊皆予以保留
   iii. 取text與label兩個欄位作為訓練資料
   iv. 將文字內容全數轉小寫，去除非單字與非空白的字符以及去除特殊標點符號，如下圖所示：

```
import re

def preprocess(text):
        text = text.lower()
        text = re.sub(r'[^\w\s]', '', text)
        text = text.replace("_", "")
        return text

data = raw_data.copy()
data["text"] = raw_data["text"].apply(preprocess)
```

另外，之所以沒有去除掉停用字，是因為我希望能盡可能的保留每一封郵件的完整內容，而且經過檢查，發現停用字的出現頻率沒有高得非常多，故保留之

v. 轉成Dataset格式，以便後續使用。

2. 模型選擇：我使用的模型是distilbert/distilbert-base-uncased-finetuned-sst-2-english模型(網址：https://huggingface.co/distilbert/distilbert-base-uncased-finetuned-sst-2-english)，是distilbert-base-uncased以SST-2這個文本情緒分析資料集微調過後得到的checkpoint，因此在分類任務上有很好的性能，此外，知識蒸餾技術讓此模型的參數量少於多數語言模型，較為方便做參數微調。

3. 分詞(tokenization)：載入模型的分詞器，並對資料做分詞(tokenize)，為避免出現長度過長的郵件內容，用truncation = True自動截斷超出最大序列長度的文字，如下圖所示：

```
[ ]  def tokenize(data):
         result = tokenizer(data["text"], truncation = True)
         return result

[ ]  # tokenize
     tokenized_dataset = dataset.map(tokenize, batched = True)
     tokenized_dataset
```

4. 切分資料：將所有資料分成訓練資料、驗證資料與測試資料三種，採用的比例為7:2:1，如圖所示，訓練資料用於微調參數，驗證資料用於檢驗模型當前的性能，測試資料用於評估最終微調完的模型之性能。

```
from datasets import DatasetDict

train_val_test = tokenized_dataset.train_test_split(test_size = 0.1, seed = 100)
train_val = train_val_test["train"].train_test_split(test_size = 0.2, seed = 100)

final_dataset = DatasetDict({"train": train_val["train"], "test": train_val_test["test"], "validation": train_val["test"]})
final_dataset
```

```
DatasetDict({
    train: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 2037
    })
    test: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 284
    })
    validation: Dataset({
        features: ['text', 'label', 'input_ids', 'attention_mask'],
        num_rows: 510
    })
})
```

5. 設定模型相關參數：此為微調模型所需要的參數，比較重要的有批次大小設32、為了充分微調故epoch數設為15，搭配監測驗證資料準確率的early-stop以防止overfitting、optimizer選用adamW、評估指標設定為accuracy，以及因為兩個類別資料量不均，所以有計算class weight讓模型更重視較少的垃圾郵件資料，如下圖所示：

```
neg, pos = np.bincount(data["label"])
total = neg + pos
print(f"Negative samples: {neg}")    # 正常郵件
print(f"Positive samples: {pos}")    # 垃圾郵件
print(f"Total samples: {total}")

weight_for_0 = (1 / neg) * (total / 2.0)
weight_for_1 = (1 / pos) * (total / 2.0)

class_weight = {0: weight_for_0, 1: weight_for_1}
```

6. 微調模型：最後載入預訓練模型與準備資料集後就可以開始微調模型，如下圖所示：

```
# compile
model.compile(optimizer = optimizer, metrics = ["accuracy"])

# fine-tune
training_proccess = model.fit(x = tf_train_set, validation_data = tf_val_set, epochs = num_epochs, callbacks = callback, class_weight = class_weight)
```

```
Epoch 1/15
63/63 [==============================] - 105s 1s/step - loss: 0.2977 - accuracy: 0.9142 - val_loss: 0.0609 - val_accuracy: 0.9843
Epoch 2/15
63/63 [==============================] - 74s 1s/step - loss: 0.0302 - accuracy: 0.9896 - val_loss: 0.0496 - val_accuracy: 0.9863
Epoch 3/15
63/63 [==============================] - 74s 1s/step - loss: 0.0074 - accuracy: 0.9985 - val_loss: 0.0401 - val_accuracy: 0.9882
Epoch 4/15
63/63 [==============================] - 74s 1s/step - loss: 0.0036 - accuracy: 0.9990 - val_loss: 0.0371 - val_accuracy: 0.9902
Epoch 5/15
63/63 [==============================] - 74s 1s/step - loss: 0.0015 - accuracy: 0.9995 - val_loss: 0.0356 - val_accuracy: 0.9902
Epoch 6/15
63/63 [==============================] - 74s 1s/step - loss: 7.8498e-04 - accuracy: 1.0000 - val_loss: 0.0366 - val_accuracy: 0.9902
Epoch 7/15
63/63 [==============================] - 74s 1s/step - loss: 5.8442e-04 - accuracy: 1.0000 - val_loss: 0.0375 - val_accuracy: 0.9902
Epoch 8/15
63/63 [==============================] - 74s 1s/step - loss: 4.4719e-04 - accuracy: 1.0000 - val_loss: 0.0383 - val_accuracy: 0.9902
```

可以看到因為early-stop的緣故微調過程停在第8個epoch，且微調不論是訓練資料還是驗證資料都有很高的accuracy。

七、 最終成果：

誠如上圖所顯示，最終模型的accuracy在訓練資料上達到100%，驗證資料上則是99%左右，可以說微調的成效非常好。

使用模型預測測試資料的類別，得到準確率約為98.9%，如下圖所示，可以看出此模型微調過後的確具備高水準的垃圾郵件分辨能力。

```
# 測試模型性能
test_loss, test_acc = model.evaluate(tf_test_set)
print("Test Accuracy:", test_acc)
print("Test Loss:", test_loss)
```

```
9/9 [==============================] - 3s 341ms/step - loss: 0.0236 - accuracy: 0.9894
Test Accuracy: 0.9894366264343262
Test Loss: 0.023632654920220375
```

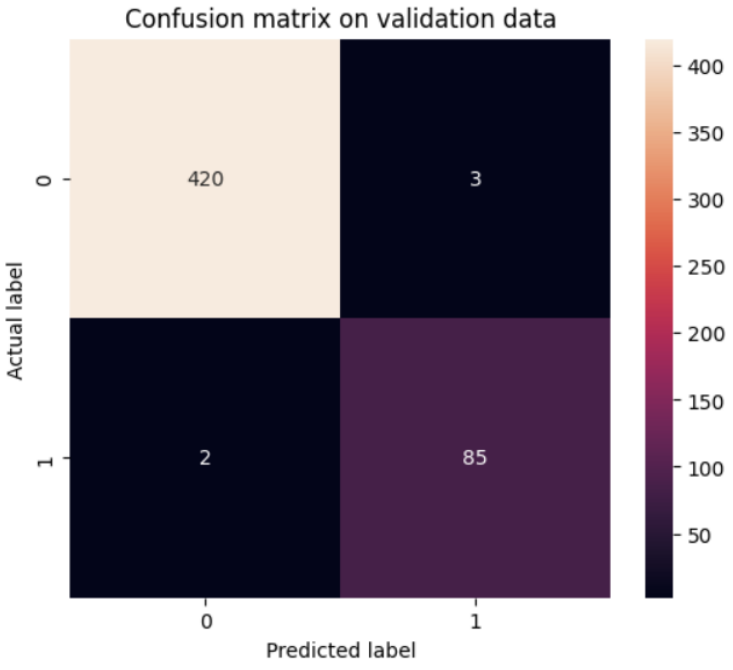
若使用沒有微調過的預訓練模型來分類測試資料，準確率就只有61.7%左右，如下圖所示：

```
# 使用未 fine-tune 過的預訓練模型來測試，查看 fine-tune 的成效
model_org = TFAutoModelForSequenceClassification.from_pretrained(checkpoint, num_labels = 2, id2label = id2label, label2id = label2id)
model.compile(optimizer = optimizer, metrics = ["accuracy"])

test_loss, test_acc = model.evaluate(tf_test_set)
print("Test Accuracy:", test_acc)
print("Test Loss:", test_loss)
```

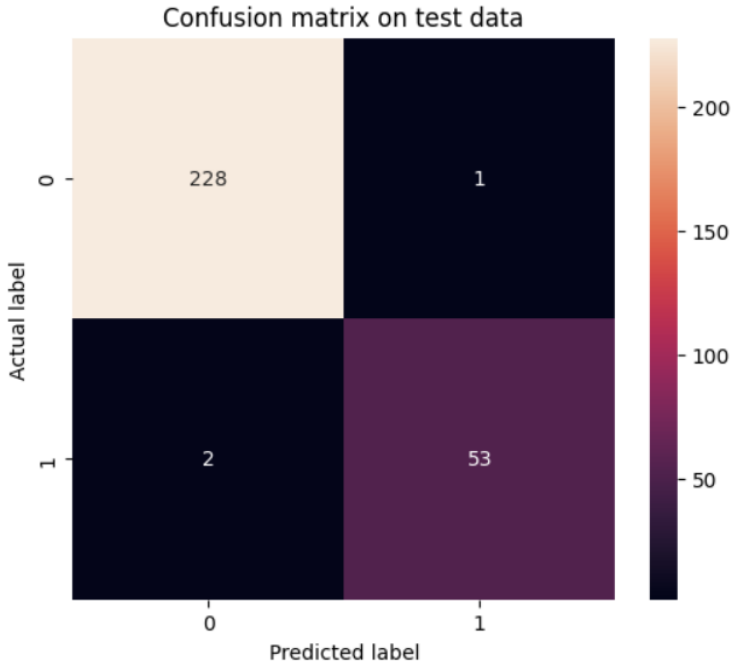
```
All PyTorch model weights were used when initializing TFDistilBertForSequenceClassification.

All the weights of TFDistilBertForSequenceClassification were initialized from the PyTorch model.
If your task is similar to the task the model of the checkpoint was trained on, you can already use TFDistilBertForSequenceClassification for pred
9/9 [==============================] - 17s 611ms/step - loss: 1.5733 - accuracy: 0.6162
Test Accuracy: 0.6161971688270569
Test Loss: 1.5733267068862915
```

兩相比較即可看出微調的成效顯著，也印證使用相關資料集微調預訓練模型能使其下游任務的表現有所進步。

另外，也可以使用混淆矩陣(confusion matrix)來察看分類結果，下圖為微調模型在驗證資料上的混淆矩陣：



下圖為微調模型在測試資料上的混淆矩陣：



兩個混淆矩陣都顯現出模型分類錯誤的次數相比於正確次數可謂極少，可見模型的高分類準確度。

小結：這次的微調可說是非常成功，使模型在垃圾郵件分類方面的準確率提升了超過30%，即使不是使用參數量龐大的高複雜度模型，微調後依然能夠有優異的表現，同時，參數量較少也能減少微調和使用上所需要的時間及運算資源。

八、 模型使用

模型微調好後，就可以拿來實際使用看看，首先我用測試資料的前五個電子郵件來讓模型分類，其結果如圖：

```
第1筆測試資料:
Token indices sequence length is longer than the specified maximum sequence length for this model (3359 > 512). Running this sequence throu
郵件文字: morphophonemics  slavic languages marta harasowska morphophonemic variability  productivity  and change the case of rusyn 1998  2
模型分類結果: [{'label': 'Negative', 'score': 0.9998393058776855}]
實際類別: NEGATIVE

第2筆測試資料:
郵件文字: sum  written signs in speech content  length  18171 i  m very grateful to those who responded  with such interesting references
模型分類結果: [{'label': 'Negative', 'score': 0.9997130036354065}]
實際類別: NEGATIVE

第3筆測試資料:
郵件文字: ijb web site correction the web address for the international journal of bilingualism  including content page of no  2 vol  1  is
模型分類結果: [{'label': 'Negative', 'score': 0.9982105493545532}]
實際類別: NEGATIVE

第4筆測試資料:
郵件文字: sum  clitic doubling in spanish to the linguist readers  some weeks ago  i posted a query about clitic doubling in spanish  i rec
模型分類結果: [{'label': 'Negative', 'score': 0.9998536109924316}]
實際類別: NEGATIVE

第5筆測試資料:
郵件文字: discontinuous constituency i am currently reviewing literature on the subject of discontinuous constituency and would like to get
模型分類結果: [{'label': 'Negative', 'score': 0.9997588992118835}]
實際類別: NEGATIVE
```

這五個電子郵件都是正常郵件，而從模型分類結果的 label 處可知模型也全部都預測為正常，分類皆正確，且 score 都非常高，代表模型對自己的預測很有信心。

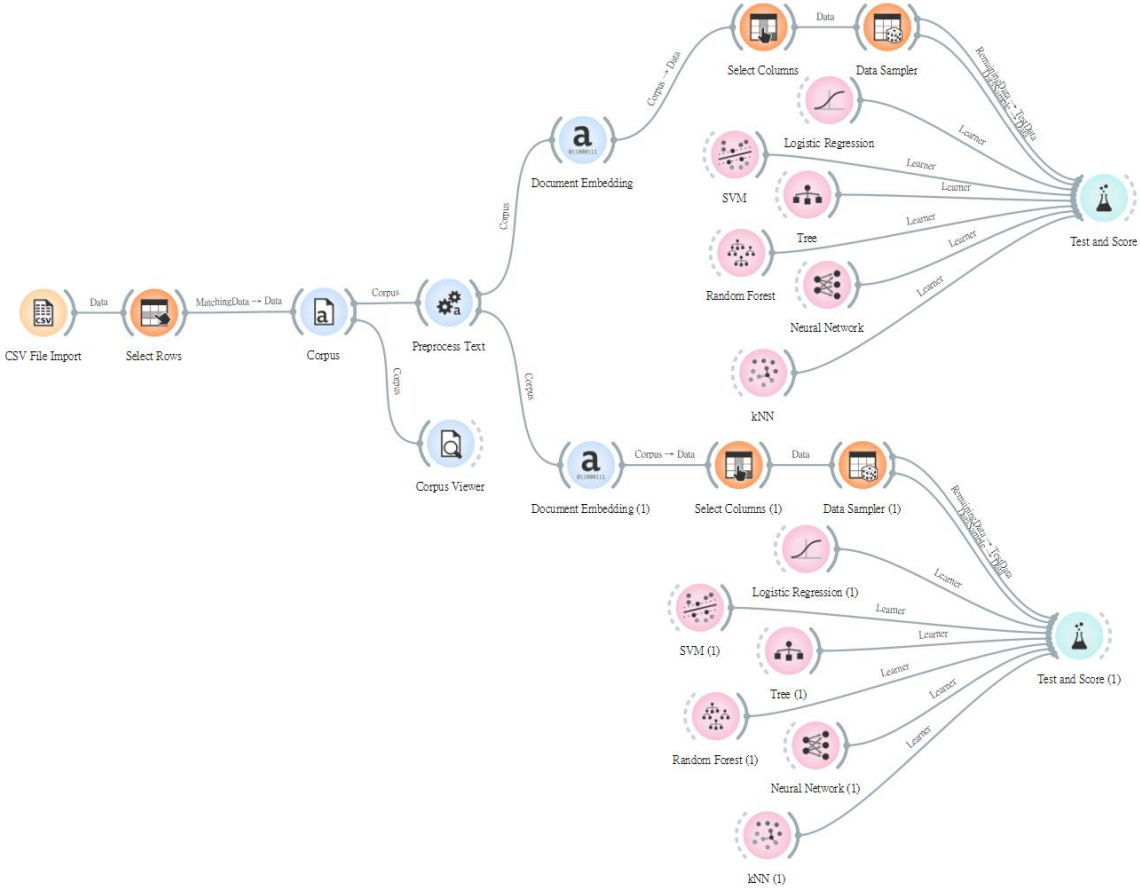
為了測試模型是否能正確地辨別出垃圾郵件，我使用一段常見的垃圾郵件內容文字讓它預測，結果如下：

```
# 以常見的 spam mail 內容來測試模型的分類能力，應該要分類為 Positive
text = "Hello, your new billing statement for April is now available online.
classifier("text")

[{'label': 'POSITIVE', 'score': 0.9930926561355591}]
```

如圖所示，模型依舊很有把握的分類正確，顯現出強大的分類能力。

九、 不同模型的比較

除了使用預訓練語言模型外，我也用同一份資料在 Orange 上用各種結構相對簡單的模型來試著做分類，workflow 圖如下：



我盡量讓整個流程都和微調預訓練模型時相同，包含文字前處理的方式、訓練測試資料集的比例及使用測試資料評估各模型。如圖所示，模型方面我選了五個常用的分類模型，神經網路的架構為三層 64 個神經元的隱藏層，learning rate 設 0.001，其他模型皆依照預設參數設定。Document Embedding 部分上方的是用 SBERT，下方使用 fastText。

使用 SBERT 所得到的模型評測如下圖：

Model	AUC	CA	F1	Prec	Recall	MCC
SVM	0.961	0.961	0.961	0.961	0.961	0.858
Logistic Regression	0.974	0.950	0.950	0.950	0.950	0.818
Neural Network	0.976	0.950	0.950	0.950	0.950	0.818
Random Forest	0.960	0.936	0.931	0.936	0.936	0.750
kNN	0.899	0.915	0.907	0.912	0.915	0.657
Tree	0.643	0.883	0.878	0.875	0.883	0.539

整體而言，所有模型的表現都還不錯，accuracy 最低也有 88%，表現最好的是 SVM，有 96%的 accuracy，但還是稍遜於微調模型的 98.9%，我認為這可能是因為面對這類文字資料時，還是需要預訓練模型這種較為複雜的架構才能很好的處理，抑或是其分詞器做出的 embeddings 較 SBERT 的 embeddings 更具代表性。

接著是使用 fastText 所得到的模型評測：

Model	AUC	CA	F1	Prec	Recall	MCC
Neural Network (1)	0.968	0.940	0.939	0.939	0.940	0.777
Random Forest (1)	0.945	0.922	0.915	0.921	0.922	0.688
Logistic Regression (1)	0.942	0.918	0.912	0.916	0.918	0.673
SVM (1)	0.957	0.908	0.911	0.916	0.908	0.688
kNN (1)	0.905	0.897	0.887	0.890	0.897	0.577
Tree (1)	0.714	0.869	0.861	0.858	0.869	0.474

整體而言所有模型的表現和使用 SBERT 時相比都有所退步，可看出 SBERT 的 embeddings 是要優於 fastText 的。即使是表現最佳的神經網路，和預訓練模型相比較同樣是略遜一籌，其原因如同上方所述可能出自於架構複雜度與 embeddings 品質。

小結：要實作垃圾郵件分類時，若是追求準確度愈高愈好，就應該使用預訓練模型搭配微調，不需要非常大量的資料或時間等資源，就可以做到很好的性能進步。

十、 參考資料：

https://huggingface.co/docs/transformers/tasks/sequence_classification
https://www.tensorflow.org/tutorials/structured_data/imbalanced_data