HW2 Sentiment Analysis

1.資料前處理

1.1 讀取 csv 檔後取前 1 萬筆資料，僅保留 "Text"、"Score" 兩個欄位

```
import pandas as pd
import numpy as np
train = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_dataMining/dataMining/HW2/Reviews.csv
test = pd.read_csv('/content/drive/MyDrive/碩一上課堂/1131_dataMining/dataMining/HW2/test.csv')
train = train.iloc[:10000]

# 將 "Score" 欄位中的值大於等於 4 的轉換成 1，其餘轉成 0
train['Sentiment'] = train['Score'].apply(lambda x: 1 if x >= 4 else 0)
train['Score']=train['Sentiment']
# 只選擇 Sentiment 和 Text 欄位
train = train[['Sentiment', 'Text_Split']]
```

	Sentiment	Text_Split
0	1	[I, have, bought, several, of, the, Vitality, ...
1	0	[Product, arrived, labeled, as, Jumbo, Salted,...
2	1	[This, is, a, confection, that, has, been, aro...
3	0	[If, you, are, looking, for, the, secret, ingr...
4	1	[Great, taffy, at, a, great, price., There, wa...

1.2 去除停頓詞stop words

```
from sklearn.feature_extraction.text import CountVectorizer

# 建立 CountVectorizer 並去除英文停頓詞
vectorizer = CountVectorizer(stop_words='english')

# 將 "Text_Split" 欄位中的詞語列表轉換回字串，並用空格連接
train['Text_Joined'] = train['Text_Split'].apply(lambda x: ' '.join(x))

# 將 "Text_Joined" 欄位進行詞頻向量化，並轉換成詞頻矩陣
train_text_matrix = vectorizer.fit_transform(train['Text_Joined'])

# 查看結果，顯示前幾個向量化的特徵名稱
vectorizer.get_feature_names_out()[:10]
```

```
array(['00', '000', '0003', '000kwh', '002', '008', '0100', '0174', '02',
       '03'], dtype=object)
```

- 特徵名表示在文本資料中出現的單詞或詞組。`CountVectorizer` 在處理文本時，會將文本中的所有詞彙進行標準化（去除標點符號、停頓詞，轉為小寫等），並生成一個詞彙表。
- 詞彙表中的每個詞（如 `00`，`000`，`002`）對應著詞頻矩陣中的一個特徵（列），矩陣的行是每一個文檔，值則是該詞在這個文檔中的出現次數。

1.3 將文字轉換成向量，請實作 tf-idf 及 word2vec 並進行比較

tf-idf：

```
from sklearn.feature_extraction.text import TfidfVectorizer

# 使用 TfidfVectorizer 將文字轉換成 TF-IDF 向量
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(train['Text_Joined'])

# 查看 TF-IDF 特徵名稱與矩陣形狀
print("TF-IDF 特徵名稱:", tfidf_vectorizer.get_feature_names_out()[:10])
print("TF-IDF 矩陣形狀:", tfidf_matrix.shape)
```

```
TF-IDF 特徵名稱: ['00' '000' '0003' '000kwh' '002' '008' '0100' '0174' '02' '03']
TF-IDF 矩陣形狀: (10000, 18497)
```

- 這裡列出了前 10 個詞彙（特徵名稱），這些詞彙是來自文本資料中的獨特詞語。大部分特徵看起來都是數字或數字相關的詞，這意味著資料集中的文本可能包含大量與數字相關的內容（例如年份、型號、價格等）。
- 這表示生成的 TF-IDF 矩陣有 10,000 條文本（行數），每一條文本都有 18,497 個特徵（列數）。特徵是來自詞彙表中所有不重複的詞語。這個矩陣是稀疏矩陣，因為每篇文本通常只包含很少的特徵（詞語），因此大部分數值為零。

word2vec：

```
import gensim
from gensim.models import Word2Vec

# 將文本分詞，作為 Word2Vec 的輸入
sentences = [text.split() for text in train['Text_Joined']]

# 訓練 Word2Vec 模型
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=2, workers=4)

# 檢查詞向量
word_vectors = w2v_model.wv
print("單詞 'good' 的詞向量:", word_vectors['good'])

# 查看詞彙表中的單詞
print("詞彙表中前幾個單詞:", list(word_vectors.index_to_key)[:10])
```

```
單詞 'good' 的詞向量: [ 2.4034734  -0.8270319   0.1727147   0.3446103  -1.7308155  -0.82865894
  0.34852687  1.9742321  -0.8842776  -0.96510744  0.14217983 -0.77461904
  0.55122983  1.6944746   0.80295336 -0.5623864  -0.73749703 -0.98825336
 -0.05928293 -1.1157621   1.6548562   0.5519125   0.18382202 -1.2511077
  0.6433905   0.2991306  -0.5395504  -0.9274134   0.574444    0.7073296
 -1.4485229   0.7512876   1.0904139  -1.262725   -2.0988972   0.8171791
 -0.33381212  0.6371679  -1.930009    0.5408649   1.0071484  -1.3147972
  1.2180831  -0.35747477  0.79043454  0.02297055 -1.1849649  -1.0343721
  0.9464272  -0.36796412 -0.11647206 -1.5742607  -0.20654847  0.10169499
  0.44469744  0.48816502  0.2768431  -0.02774115  1.7046776  -1.0131913
  1.0318767   0.06271436  1.5615684  -0.68887967 -0.7026345   2.616153
  0.24217242  0.5240261  -2.7187324   0.93966645  0.35048658 -0.05421445
 -0.38595876 -0.29602447  0.94043404  0.47433406  0.801671    0.4561113
 -1.254309    0.83719665  0.765557    0.34376872 -1.7921617   0.3745821
  0.9282655  -0.36070132 -0.01664878  0.32435104  0.9191093   1.0208565
  0.30467892 -0.80205655 -0.9625194  -0.55568004  1.0864645   0.25395784
 -1.7357491  -0.25648335 -0.1950619  -0.25609306]
詞彙表中前幾個單詞: ['the', 'I', 'and', 'a', 'to', 'of', 'is', 'it', 'for', 'this']
```

- 這裡顯示的是 "good" 的詞向量，這是一個 100 維的數字陣列。這些數字代表了該單詞在文本中的語義關聯性。`Word2Vec` 通過考慮 "good" 在文本中與其他單詞的共現關係來生成這個向量。例如，向量中的某些維度可能表示情感、物理屬性或其他特徵。模型能夠通過這些向量捕捉單詞之間的語義關係，例如"good" 可能與 "great" 或 "excellent" 等詞向量相似。

- 以及顯示詞彙表中最常出現的前 10 個單詞，這些單詞都是文本中高頻詞。這些詞通常是英文中的常見詞，如 "the", "I", "and" 等。雖然這些詞出現頻率很高，但它們的語義信息較少，因為它們更多是功能詞，而非具有特定含義的實詞。

2. 使用 Random forest 進行分類；評估模型：進行 k-fold cross-validation 並計算 k=4 的 Accuracy

tf-idf（RF）：

```
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# 使用 TfidfVectorizer 將文字轉換成 TF-IDF 向量
tfidf_vectorizer = TfidfVectorizer(stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(train['Text_Joined'])

# 建立隨機森林模型
rf_tfidf = RandomForestClassifier(n_estimators=100, random_state=42)

# 使用 k=4 進行交叉驗證
cv_scores_tfidf = cross_val_score(rf_tfidf, tfidf_matrix, train['Sentiment'], cv=4, scoring='acc

# 輸出每次交叉驗證的準確率以及平均準確率
print("TF-IDF + 隨機森林 k=4 的準確率:", cv_scores_tfidf)
print("TF-IDF + 隨機森林 k=4 的平均準確率:", cv_scores_tfidf.mean())
```

```
TF-IDF + 隨機森林 k=4 的準確率: [0.7984 0.7984 0.7996 0.8052]
TF-IDF + 隨機森林 k=4 的平均準確率: 0.8004
```

在本地進行交叉驗證時，隨機森林模型與 TF-IDF 向量化的結合給出了穩定且不錯的結果：

- 每次交叉驗證的準確率分別為 `[0.7984, 0.7984, 0.7996, 0.8052]`，顯示模型在不同的訓練/測試分割中的表現相對穩定。
- 平均準確率為 `0.8004`，接近 80%，表示該模型在本地的表現是可靠的，能夠大致正確預測文本的情感。
- 在 Kaggle 上的準確率為 `0.62638`，這與本地交叉驗證的 80% 準確率相比，顯著降低，差異超過 17%。模型在 Kaggle 測試集上的表現明顯較差，可能存在過擬合本地數據的情況或是測試集的分布與訓練集存在較大差異。

word2vec（RF）：

```
import numpy as np
from gensim.models import Word2Vec
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import cross_val_score

# 定義一個函數來將每個文本的詞向量取平均值
def average_word_vectors(words, model, num_features):
    feature_vec = np.zeros((num_features,), dtype="float32")
    nwords = 0
    index2word_set = set(model.wv.index_to_key)  # 確保詞彙表存在
    for word in words:
        if word in index2word_set:  # 檢查該單詞是否在詞彙表中
            nwords += 1
            feature_vec = np.add(feature_vec, model.wv[word])
    if nwords > 0:  # 避免除以0
```

```
        feature_vec = np.divide(feature_vec, nwords)
    return feature_vec

# 確保 'Text_Split' 欄位是已經分詞的文本
# sentences 應該是分詞過的句子的列表
sentences = train['Text_Split'].tolist()

# 訓練 Word2Vec 模型
w2v_model = Word2Vec(sentences, vector_size=100, window=5, min_count=2, workers=4)

# 將每個文本轉換為詞向量的平均值
train_word2vec = np.array([average_word_vectors(text, w2v_model, 100) for text in train['Text_Sp

# 確保轉換後的詞向量是非空的
print(f"轉換後的詞向量形狀: {train_word2vec.shape}")

# 建立隨機森林模型
rf_w2v = RandomForestClassifier(n_estimators=100, random_state=42)

# 使用 k=4 進行交叉驗證
cv_scores_w2v = cross_val_score(rf_w2v, train_word2vec, train['Sentiment'], cv=4, scoring='accur

# 輸出每次交叉驗證的準確率以及平均準確率
print("Word2Vec + 隨機森林 k=4 的準確率:", cv_scores_w2v)
print("Word2Vec + 隨機森林 k=4 的平均準確率:", cv_scores_w2v.mean())
```

```
轉換後的詞向量形狀: (10000, 100)
Word2Vec + 隨機森林 k=4 的準確率: [0.7736 0.7672 0.7676 0.7564]
Word2Vec + 隨機森林 k=4 的平均準確率: 0.7662
```

在本地進行交叉驗證時，將 Word2Vec 與隨機森林模型結合的準確率為：

- 每次交叉驗證的準確率為 `[0.7736, 0.7672, 0.7676, 0.7564]`，這表示模型在不同的訓練/測試分割中有相對穩定的表現。
- 平均準確率為 `0.7662`，接近 77%，表示模型在本地的文本分類表現還不錯。

Kaggle 上的準確率為 `0.55979`，顯著低於本地交叉驗證的準確率（77%），表現差距超過 20%。這顯示模型在 Kaggle 的測試集上表現不佳。

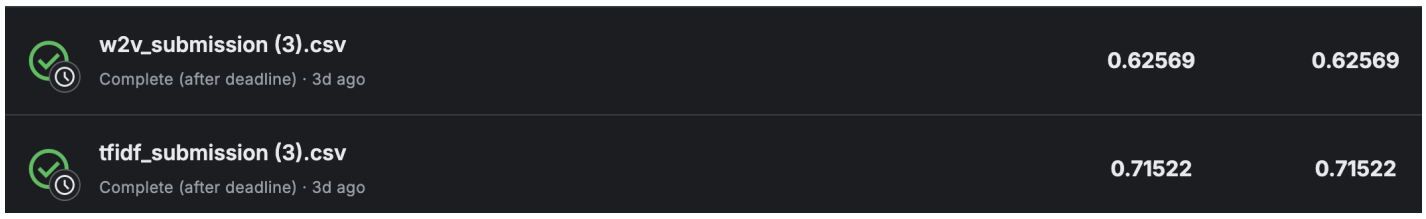
> 💡 由於在不改變資料量的情況下，隨機森林模型的表現未達到理想的效果，我決定嘗試使用 XGBoost 來提升預測能力。**這次的目標是檢驗隨機森林是否在泛化上存在問題，因此我不會調整資料量，而是改用 XGBoost，希望它能更有效地捕捉數據中的關鍵特徵**。

tf-idf、word2vec（XGboost）：

```
import xgboost as xgb
from sklearn.model_selection import cross_val_score

# 初始化 XGBoost 模型
xgb_tfidf = xgb.XGBClassifier(n_estimators=100, random_state=42)
```

```
xgb_w2v = xgb.XGBClassifier(n_estimators=100, random_state=42)

# 使用 TF-IDF 特徵進行 XGBoost 訓練
xgb_tfidf.fit(tfidf_matrix, train['Sentiment'])

# 使用 Word2Vec 特徵進行 XGBoost 訓練
xgb_w2v.fit(train_word2vec, train['Sentiment'])

# 使用 TF-IDF 特徵進行預測
test_predictions_tfidf = xgb_tfidf.predict(test_tfidf_matrix)

# 使用 Word2Vec 特徵進行預測
test_predictions_w2v = xgb_w2v.predict(test_word2vec)

# 進行 k-fold 交叉驗證 (k=4) 並計算準確率
cv_scores_tfidf = cross_val_score(xgb_tfidf, tfidf_matrix, train['Sentiment'], cv=4, scoring='ac
print("XGBoost (TF-IDF) k=4 的準確率:", cv_scores_tfidf)
print("XGBoost (TF-IDF) k=4 的平均準確率:", cv_scores_tfidf.mean())

cv_scores_w2v = cross_val_score(xgb_w2v, train_word2vec, train['Sentiment'], cv=4, scoring='accu
print("XGBoost (Word2Vec) k=4 的準確率:", cv_scores_w2v)
print("XGBoost (Word2Vec) k=4 的平均準確率:", cv_scores_w2v.mean())
```

```
XGBoost (TF-IDF) k=4 的準確率: [0.824  0.8228 0.8292 0.8288]
XGBoost (TF-IDF) k=4 的平均準確率: 0.8262
XGBoost (Word2Vec) k=4 的準確率: [0.7764 0.768  0.7716 0.7596]
XGBoost (Word2Vec) k=4 的平均準確率: 0.7688999999999999
```



TF-IDF 特徵的比較：

- **Random Forest**：
  - 本地交叉驗證平均準確率：`0.8004`
  - Kaggle 測試集準確率：`0.62638`
- **XGBoost**：
  - 本地交叉驗證平均準確率：`0.8262`
  - Kaggle 測試集準確率：`0.62569`

> 💡
> - 在使用 TF-IDF 特徵時，XGBoost 在本地交叉驗證的表現略優於 Random Forest（82.6% vs. 80%）。
> - 但在 Kaggle 測試集上的表現幾乎相同，XGBoost 和 Random Forest 都達到大約 62.5% 的準確率。
>
> 這說明無論是 XGBoost 還是 Random Forest，兩者在處理 TF-IDF 特徵時都面臨著類似的泛化挑戰，尤其是在 Kaggle 測試集上，模型的表現無法與本地交叉驗證匹配。

Word2Vec 特徵的比較：

- **Random Forest**：
  - 本地交叉驗證平均準確率：`0.7662`
  - Kaggle 測試集準確率：`0.55979`

- **XGBoost**：
  - 本地交叉驗證平均準確率：`0.7689`
  - Kaggle 測試集準確率：`0.71522`

> 💡
> - 在使用 Word2Vec 特徵時，XGBoost 和 Random Forest 在本地交叉驗證的表現相近（XGBoost：76.9%，Random Forest：76.6%）。
> - 然而，XGBoost 在 Kaggle 測試集上的表現（71.5%）明顯優於 Random Forest（55.9%），顯示出 XGBoost 能夠更好地泛化到測試數據集。

結論

1. TF-IDF 特徵結果：

- **Random Forest (TF-IDF)**：
  - **本地交叉驗證準確率**：大約 **80%**。
  - **Kaggle 準確率**：**0.62638**。
  - 隨機森林在本地的表現還不錯，但在 Kaggle 測試集上，表現稍顯不足，模型泛化能力可能受到資料分布的影響。
- **XGBoost (TF-IDF)**：
  - **本地交叉驗證準確率**：大約 **82.6%**。
  - **Kaggle 準確率**：**0.62569**。
  - 雖然 XGBoost 在本地的交叉驗證準確率高於 Random Forest，但在 Kaggle 測試集上的表現與 Random Forest 相當，甚至略有下降。

小結：使用 TF-IDF 特徵時，兩個模型在本地的表現都相對穩定，但 Kaggle 測試集上的表現差異不大，這可能表明 TF-IDF 特徵在測試數據上的泛化能力較弱。

2. Word2Vec 特徵結果：

- **Random Forest (Word2Vec)**：
  - **本地交叉驗證準確率**：大約 **76%**。
  - **Kaggle 準確率**：**0.55979**。
  - 隨機森林在使用 Word2Vec 特徵時，無論在本地還是 Kaggle 測試集上的表現都不如 TF-IDF，顯示出對於語義特徵的處理能力較弱。
- **XGBoost (Word2Vec)**：
  - **本地交叉驗證準確率**：大約 **76.9%**。
  - **Kaggle 準確率**：**0.71522**。
  - XGBoost 在使用 Word2Vec 特徵時，無論在本地還是 Kaggle 測試集上的表現都優於 Random Forest，並且 Kaggle 測試集上的表現顯著提升，說明 XGBoost 更能有效利用語義特徵進行預測。

小結：在使用 Word2Vec 特徵時，XGBoost 的表現明顯優於 Random Forest，尤其在 Kaggle 測試集上，準確率有明顯提升，說明 XGBoost 更適合處理語義嵌入特徵。

> 🔍 這次的結果顯示，隨著模型複雜度的提高，確實能夠更好地捕捉數據中的細節，從而提升預測效果。相比較簡單的 Random Forest，XGBoost 在處理像 Word2Vec 這類語義嵌入特徵時表現更為出色。不過，這次作業只使用了全部資料中的前一萬筆，因此可以推測，若增加資料量，模型的預測能力還有進一步提升的空間。