

DATA MINING

政治大學金融所 112352014 劉冠銘

November 29, 2024

1 Data

這次kaggle有以下資料，在這裡我只會把我有使用到的資料以及欄位列出來：

- **data_identification.csv**: 此檔案包含推文的識別資訊，特別是用於區分訓練集與測試集的標籤。每條記錄由 *'tweet_id'* 和 *'identification'* 組成，其中 *'identification'* 標註為 *'train'* 或 *'test'*。
- **emotion.csv**: 此檔案記錄了推文的情緒標籤資訊，包含兩列：*'tweet_id'* 和 *'emotion'*，其中 *'emotion'* 表示該推文的情緒類別（共有8種）。
- **tweets_DM.json**: 這是一個包含推文資料的 JSON 檔案，每條記錄中包含推文的細節資訊，其中我們將會使用以下資料欄位，包括：
 - **tweet_id**: 推文的唯一標識符。
 - **hashtags**: 推文中包含的所有主題標籤（hashtags）。
 - **text**: 推文的文本內容。

1.1 Merge

在此部分，我們進行資料處理以生成訓練集與測試集，主要步驟如下：

- **資料讀取**：讀取辨識檔案 (*data_identification.csv*)、情緒標籤檔案 (*emotion.csv*) 和推文 JSON 資料 (*tweets_DM.json*)。
- **訓練集生成**：將推文資料與情緒標籤透過 *tweet_id* 進行內連結，生成訓練集 (*df_train*)。
- **測試集生成**：根據辨識檔案篩選出測試標籤的 *tweet_id*，從推文資料中提取對應推文生成測試集 (*df_test*)。
- **數據保存 (可選)**：可將訓練集與測試集保存為 *.pkl* 格式以便後續使用。

此步驟完成了推文與情緒的整合，並將資料分為 **training dataset** 與 **testing dataset**，為後面的模型訓練和測試提供基礎。並且兩資料集的尺寸分別為：

Train shape: (1455563, 4)

Test shape: (411972, 3)

```
1 df_train.head(5)
```

✓ 0.0s

	tweet_id	hashtags	text	emotion
0	0x376b20	[Snapchat]	People who post "add me on #Snapchat" must be ...	anticipation
1	0x2d5350	[freepress, TrumpLegacy, CNN]	@brianklaas As we see, Trump is dangerous to #...	sadness
2	0x1cd5b0	[]	Now ISSA is stalking Tasha 😬😬😬 <LH>	fear
3	0x1d755c	[authentic, LaughOutLoud]	@RISKshow @TheKevinAllison Thx for the BEST TL...	joy
4	0x2c91a8	[]	Still waiting on those supplies Liscus. <LH>	anticipation

Figure 1: Training Dataset Visualization

```
1 df_test.head(5)
```

✓ 0.0s ❷ Open 'df_test' in Data Wrangler

	tweet_id	hashtags	text
0	0x28b412	[bibleverse]	Confident of your obedience, I write to you, k...
1	0x2de201	[]	"Trust is not the same as faith. A friend is s...
2	0x218443	[materialism, money, possessions]	When do you have enough ? When are you satisfi...
3	0x2939d5	[GodsPlan, GodsWork]	God woke you up, now chase the day #GodsPlan #...
4	0x26289a	[]	In these tough times, who do YOU turn to as yo...

Figure 2: Testing Dataset Visualization

1.2 Feature Dustribution

我們可以先通過視覺化圖表分析推文的情緒分佈以及其文字長度特徵。

如圖 3 所示，推文情緒分佈呈現出一定的不平衡，其中 **joy** 是最常見的情緒，佔據了最多的推文數量，而其他情緒如 **anger** 和 **surprise** 的推文數量相對較少。從這邊可以看出來，在此資料數據中，正向情緒佔有更大的比例。

此外，圖 4 展示了不同情緒下推文的文字數量分佈。可以觀察到：

- **joy** 和 **anticipation** 等正向情緒的推文，整體文字數量分佈較廣泛，而且中位數相對較高，可能可以看出正向推文傾向於更詳細的表達。
- 負向情緒如 **fear** 和 **anger** 的推文文字數量則較為集中，則能看出這類推文更傾向於簡短的表達方式。
- 情緒 **surprise** 的推文具有明顯的文字數量範圍變化，可能因情緒表達的不確定性導致。

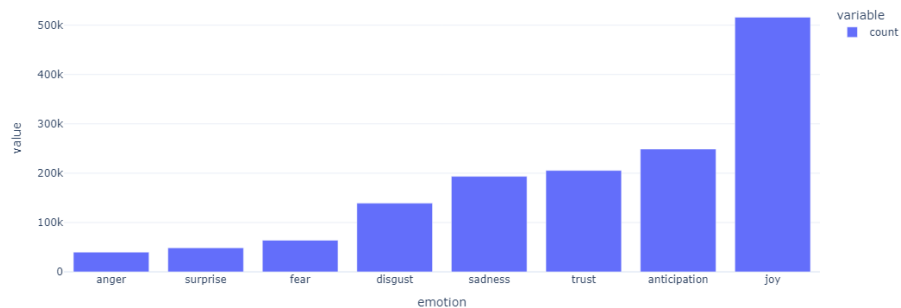


Figure 3: Emotion Distribution Across Tweets



Figure 4: Words Per Tweet Across Emotions

因此我們對於此資料集可以大致整理出以下幾點：

- 整體數據集中正向情緒（如 **joy** 和 **anticipation**）佔主導地位，可能需要在模型訓練中考慮情緒不平衡的影響。
- 不同情緒的推文文字數量存在顯著差異，這可能是模型預測中可利用的重要特徵。
- 負向情緒的推文更趨於簡短表達，這可能反映了用戶在負面情緒下的表達習慣。

2 Experiment

在這部分，我們針對推文情緒分類問題進行了四次實驗，探索不同模型架構與Feature engineering方法對模型性能的影響。每次實驗都會嘗試不同的操作。我們的實驗主要聚焦於以下幾點：

- 使用傳統的機器學習方法（如 Logistic Regression）進行基準測試。
- 引入預訓練語言模型（DistilBERT），並結合多種Feature engineering技術以提升分類效果。
- 針對Feature engineering方法（如 N-Gram 和情感詞匯分析）進行優化，並探索降維技術的影響。
- 結合深度學習與自定義特徵的混合模型，進一步提升模型表現。

第一次實驗使用 **Logistic Regression** 作為 baseline model，配合文本清理與 TF-IDF 特徵抽取。該模型提供了一個參考標準。第二次實驗則引入了 **DistilBERT** 預訓練模型，通過 Hugging Face 的 **Trainer** 進行微調，並生成提交文件。該實驗確實有打敗第一次使用傳統機器學習的 baseline model，明顯提升了模型性能。第三次實驗基於第二次實驗進行改進，結合 TF-IDF 提取的 N-Gram 特徵和情感詞彙計數（如正向/負向詞彙數量），並採用降維技術（SVD）將特徵維度進一步壓縮（因為用免費 colab，RAM一直爆掉...）。該模型使用了一個自定義的混合架構，將 DistilBERT 的嵌入層與後面新增加的特徵結合進行分類。第四次實驗在第三次實驗的基礎上進行優化，增加了降維的 TF-IDF 成分數量（從 50 提升至 80），以捕捉更多的語義信息。同時保持自定義混合模型結構。

其中我們在四次實驗中都會先把text數據進行清理:

```

1 def clean_text(text):
2     import re
3     text = re.sub(r"http\S+", "", text) # Remove URLs
4     text = re.sub(r"@w+", "", text)     # Remove mentions
5     text = re.sub(r"[^a-zA-Z\s]", "", text) # Remove special
6     characters
7     return text.lower().strip()

```

Listing 1: Text Cleaning Function

2.1 Baseline Model

第一次實驗旨在建立情緒分類的基線性能，我們採用了 **Logistic Regression** 作為分類器，並通過一系列的數據處理和Feature engineering完成模型訓練與測試。具體步驟如下：

- **數據清理**：如 Listing 1 所示。
- **特徵提取**：
 - 使用 **TF-IDF Vectorizer** 將清理後的文本轉換為數值特徵。TF-IDF 特徵捕捉了詞語在推文中以及整個數據集中出現的權重，範圍設置為最大 5000 個特徵。
- **數據分割**：

- 將數據集按照 80%-20% 的比例分割為訓練集和驗證集，用於模型的訓練和性能評估。

- **模型訓練與測試：**

- 使用 **Logistic Regression** 作為分類器，設置最大迭代次數為 2000，以確保收斂。

- **實驗繳交結果：**

- Kaggle 最後結果為 **0.39870**

這次實驗的主要目的是提供一個簡單而有效的起點，為後續實驗的改進提供對比依據。儘管 Logistic Regression 是一種簡單的分類器，但結合文本清理和 TF-IDF 特徵，模型仍能實現合理的性能，並為後續更複雜模型的提升空間提供參考。

2.2 Model 1: 引入預訓練模型

第二次實驗的目標是引入 Pre-Trained 好的深度學習模型 **DistilBERT**，以探索其在推文情緒分類任務中的應用效果。該實驗對數據進行了清理與標籤編碼，並使用 Hugging Face 提供的 **Trainer** 工具進行模型的微調。具體步驟如下：

- **數據清理：** 如 Listing 1 所示。
- **標籤編碼：**
 - 使用 **LabelEncoder** 將推文的情緒標籤（如 `joy`, `sadness`）轉換為整數類別，方便模型處理分類任務。
- **數據分割：**
 - 將數據按照 80%-20% 的比例劃分為訓練集和驗證集，保證模型的訓練和測試有統一的數據基礎。
- **模型初始化：**
 - 加載 Hugging Face 的 **DistilBERT** 預訓練模型和對應的標記器 (**Tokenizer**)。
 - 根據分類任務的需要，設置輸出類別數量 (`num_labels`) 為情緒標籤的數目。
- **數據集封裝：** 為了方便模型處理數據，我們自定義了一個 PyTorch **Dataset Class**，將推文文本和標籤封裝起來，並在其中調用 **DistilBERT** 的標記器進行標記化。這樣的設計能有效支持批量數據處理和進行 Feature engineering 的特徵的擴展。程式碼如下：

```

1 class EmotionDataset(Dataset):
2     def __init__(self, texts, labels=None, tokenizer=None,
3         max_length=64):
4         self.texts = texts
5         self.labels = labels
6         self.tokenizer = tokenizer
7         self.max_length = max_length
8
9     def __len__(self):
10         return len(self.texts)
11
12     def __getitem__(self, idx):
13         tokenized = self.tokenizer(self.texts[idx], truncation
14             =True, padding="max_length", max_length=self.
15             max_length, return_tensors="pt")
16         item = {key: val.squeeze(0) for key, val in tokenized.
17             items()}
18         if self.labels is not None:
19             item['labels'] = torch.tensor(self.labels[idx],
20                 dtype=torch.long)
21         return item

```

Listing 2: 自定義 PyTorch Dataset

使用自定義 Class 之原因：

- 能夠靈活處理多模態數據（如文本和其他特徵）或動態添加自定義字段。
- 支持 Hugging Face 模型所需的標記化過程，並且保證數據格式符合模型輸入要求。
- 與 PyTorch 的 `DataLoader` 配合，實現較高效的數據處理。

● 訓練與微調：

- 設置訓練參數，包括 learning rate、batch size、以及 Epoch。
- 使用 Hugging Face 的 `Trainer` 工具進行模型微調，並在每個訓練輪次結束後在驗證集上評估模型性能。
- 評估指標包括 加權 F1-score 和 準確率 (Accuracy)。

● 實驗繳交結果：

- Kaggle 最後結果為 **0.49449**

本次實驗顯示，通過引入預訓練的深度學習模型 **DistilBERT**，可以顯著提升情緒分類的性能。且 Hugging Face 提供的工具簡化了模型訓練和評估的流程，使得複雜的深度學習模型更易於應用。

2.3 Model 2: 混合特徵架構

第三次實驗的目標是結合深度學習模型與傳統的Feature engineering，構建一個混合特徵架構，探索各種 feature 對於推文情緒分類的影響。本次實驗基於第二次實驗進行衍伸，加入了一些其他經過Feature engineering的新特徵以及降維，具體步驟如下：

- **數據清理**：如 Listing 1 所示。
- **Feature engineering**：為了豐富模型的輸入，我們加入了以下幾類新特徵：

- **文本長度**：計算每條推文的單詞數，作為簡單的結構特徵。

```
1 # Calculate text length
2 df_train['text_length'] = df_train['text'].apply(lambda x:
3     len(x.split()))
4 df_test['text_length'] = df_test['text'].apply(lambda x:
5     len(x.split()))
```

Listing 3: 計算文本長度

- **情感詞彙計數**：基於 NLTK 的情感詞典，計算推文中正向的詞和負向的詞之數量。

```
1 # Load sentiment lexicon
2 from nltk.corpus import opinion_lexicon
3 from nltk.tokenize import word_tokenize
4 positive_words = set(opinion_lexicon.positive())
5 negative_words = set(opinion_lexicon.negative())
6
7 # Count sentiment words
8 def count_emotion_words(text):
9     words = word_tokenize(text)
10    pos_count = sum(1 for word in words if word in
11        positive_words)
12    neg_count = sum(1 for word in words if word in
13        negative_words)
14    return pos_count, neg_count
15
16 df_train[['pos_count', 'neg_count']] = df_train['text'].
17     apply(lambda x: pd.Series(count_emotion_words(x)))
18 df_test[['pos_count', 'neg_count']] = df_test['text'].
19     apply(lambda x: pd.Series(count_emotion_words(x)))
```

Listing 4: 計算情感詞彙數量

- **N-Gram 特徵**：使用 **TF-IDF** 抽取一元組到三元組的特徵，並限制最大特徵數量為 1000。

```
1 # Extract N-Gram features
2 from sklearn.feature_extraction.text import
3     TfidfVectorizer
4 tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, 3),
5     max_features=1000)
```

```

4 train_tfidf = tfidf_vectorizer.fit_transform(df_train['
  text'])
5 test_tfidf = tfidf_vectorizer.transform(df_test['text'])

```

Listing 5: 提取 N-Gram 特徵

- **降維**：透過 **Truncated SVD** 將 TF-IDF 特徵壓縮至 50 維，以減少維度並保留主要信息。

```

1 # Dimensionality reduction
2 from sklearn.decomposition import TruncatedSVD
3 svd = TruncatedSVD(n_components=50, random_state=42)
4 train_tfidf_reduced = svd.fit_transform(train_tfidf)
5 test_tfidf_reduced = svd.transform(test_tfidf)

```

Listing 6: 降維處理

- **自定義模型架構**：我們設計了一個混合模型架構，將深度學習模型 **DistilBERT** 與手工特徵結合：

- DistilBERT 模型用於提取文本的 embedding，並取其 [CLS] 標籤的輸出作為主要表示。
- 手工特徵（如文本長度、情感詞彙計數和降維後的 TF-IDF 特徵）經過線性層投影至 128 維嵌入。
- 最終將兩部分嵌入進行拼接，通過分類層輸出情緒標籤。

模型程式碼如下：

```

1 class CustomDistilBertForSequenceClassification(nn.Module):
2     def __init__(self, base_model, feature_dim, num_labels):
3         super(CustomDistilBertForSequenceClassification, self)
4             .__init__()
5         self.base_model = base_model
6         self.feature_proj = nn.Linear(feature_dim, 128) #
7             Project handcrafted features
8         self.classifier = nn.Linear(base_model.config.
9             hidden_size + 128, num_labels)
10        self.dropout = nn.Dropout(0.3)
11
12    def forward(self, input_ids, attention_mask, features,
13        labels=None):
14        base_output = self.base_model(input_ids=input_ids,
15            attention_mask=attention_mask)
16        pooled_output = base_output.last_hidden_state[:, 0, :]
17        # DistilBERT embedding
18        feature_proj = self.feature_proj(features).squeeze(1)
19        # Handcrafted feature projection
20        combined_output = torch.cat((pooled_output,
21            feature_proj), dim=1) # Concatenate embeddings
22        combined_output = self.dropout(combined_output)
23        logits = self.classifier(combined_output) #
24            Classification layer output
25
26        if labels is not None:

```



```

18         loss_fn = nn.CrossEntropyLoss()
19         loss = loss_fn(logits, labels)
20         return loss, logits
21     return logits

```

Listing 7: 自定義混合模型架構

- 訓練與評估：

- 使用 Hugging Face 的 **Trainer** 工具進行模型微調，設置混合精度 (FP16) 和 learning rate 調整策略。

- 實驗繳交結果：

- Kaggle 最後結果為 **0.50887**

本次實驗通過結合深度學習和經過 Feature engineering 的新特徵，在情緒分類中顯示了顯著的性能提升，特別是在多模態特徵的處理上展現了更强的泛化能力。

2.4 Model 3: 優化特徵處理

第四次實驗基於第三次實驗，並且在進行進一步優化，主要目的是探索增強特徵處理對模型性能的影響。本次實驗保留了深度學習嵌入與手工特徵的混合設計，並對特徵提取和降維進行調整，具體步驟如下：

- 數據清理：如 Listing 1 所示。

- **Feature engineering**：除了將降維後的特徵數量增加至 80 以外，其他都與第三次實驗之操作相同。

- **增強降維**：將 **Truncated SVD** 的降維後特徵數量從 50 增加到 80，以此保留更多的語義信息。程式碼如下：

```

1  # Dimensionality reduction with enhanced components
2  from sklearn.decomposition import TruncatedSVD
3  svd = TruncatedSVD(n_components=80, random_state=42)
4  train_tfidf_reduced = svd.fit_transform(train_tfidf)
5  test_tfidf_reduced = svd.transform(test_tfidf)

```

Listing 8: 增強降維處理

- **自定義模型架構**：模型架構延續了第三次實驗的設計，將 DistilBERT 的文本嵌入與經過增強處理的手工特徵結合。主要變化在於手工特徵中包含更多維度的 TF-IDF 表示，模型程式碼如下：

```

1  class CustomDistilBertForSequenceClassification(nn.Module):
2      def __init__(self, base_model, feature_dim, num_labels):
3          super(CustomDistilBertForSequenceClassification, self)
4              .__init__()
5          self.base_model = base_model

```

```

5         self.feature_proj = nn.Linear(feature_dim, 128) #
           Project handcrafted features
6         self.classifier = nn.Linear(base_model.config.
           hidden_size + 128, num_labels)
7         self.dropout = nn.Dropout(0.3)
8
9         def forward(self, input_ids, attention_mask, features,
           labels=None):
10            base_output = self.base_model(input_ids=input_ids,
           attention_mask=attention_mask)
11            pooled_output = base_output.last_hidden_state[:, 0, :]
           # DistilBERT embedding
12            feature_proj = self.feature_proj(features).squeeze(1)
           # Handcrafted feature projection
13            combined_output = torch.cat((pooled_output,
           feature_proj), dim=1) # Concatenate embeddings
14            combined_output = self.dropout(combined_output)
15            logits = self.classifier(combined_output) #
           Classification layer output
16
17            if labels is not None:
18                loss_fn = nn.CrossEntropyLoss()
19                loss = loss_fn(logits, labels)
20                return loss, logits
21            return logits

```

Listing 9: 自定義混合模型架構

- 訓練與評估：與第三次實驗相同。
- 實驗繳交結果：
 - Kaggle 最後結果為 **0.50889**

實驗結果與分析 第四次實驗通過增強降維處理顯著提高了模型的性能，最終實現了實驗中的最佳結果，F1-score 達到 **0.50889**。從這邊可以看出，將降維特徵數量從50提升至80，並不會對最後結果造成明顯的影響。

2.5 Extended Discussion

為了進一步提升模型的穩健性和準確性，我們嘗試對不同實驗的預測結果進行集成，採用 **多數決策策略** (Majority Voting)。具體內容如下：

方法描述 集成的目的是結合不同模型的優勢，平衡單一模型可能存在的偏差。實現過程中，將Model 1, Model 2 以及 Model 3 的預測結果進行多數決，取三者中的眾數作為最終的預測結果。以下為程式碼：

```

1 # Combine predictions
2 df = pd.concat([sub1[['id', 'emotion']], sub2['emotion_50'], sub3['
           emotion_80']], axis=1)
3
4 # Majority voting

```

```
5 df['emotion_res'] = df[['emotion', 'emotion_50', 'emotion_80']].  
    mode(axis=1)[0]
```

Listing 10: 多數決策策略

實驗結果 集成結果的性能與各單一模型的對比如下（如圖 5 所示）：

- Baseline model：**0.39870**
- Model 1（DistilBERT）：**0.49449**
- Model 2（混合特徵架構）：**0.50887**
- Model 3（增強降維模型）：**0.50889**
- 多數決策策略（集成結果）：**0.50774**







Submission and Description	Public Score ⓘ	Select
 submission_ensemble.csv Complete · 16h ago	0.50774	<input type="checkbox"/>
 submission_80.csv Complete · 17h ago	0.50889	<input checked="" type="checkbox"/>
 submission_50.csv Complete · 17h ago	0.50887	<input checked="" type="checkbox"/>
 submission.csv Complete · 2d ago	0.49449	<input type="checkbox"/>
 submission.csv Error · 2d ago · colab1		
 submission.csv Complete · 3d ago · XDD	0.39870	<input type="checkbox"/>

Figure 5: Comparison of Kaggle scores across models

可能原因分析 儘管集成方法通常能提升模型的穩健性，但本次多數決策策略未能超越單一性能最佳模型（第四次實驗），可能原因如下：

- 模型間偏差未充分互補：
 - **問題**：參與集成的模型對相同類別的預測偏差存在高度相似性，這可能在多數決中被放大。
 - **原因**：本研究中的三個模型均基於 **DistilBERT**，可能對某些相同樣本出現相同的錯誤。
- 特徵處理方法的相似性：
 - **問題**：參與集成的模型在特徵處理上過於相似，例如均使用 TF-IDF 作為主要特徵，導致模型的多樣性不足。
 - **原因**：特徵的相似性限制了各模型的互補能力，從而降低了多數決策策略的增益效果。

3 Conclusion

此報告針對推文情緒分類問題，通過一系列實驗探索了不同模型架構與Feature engineering方法對模型性能的影響，並嘗試了集成策略：

- **Baseline Model 建立了可靠的參考標準：** Logistic Regression 作為Baseline model，結合文本清理與 TF-IDF 特徵提取，取得 **0.39870** 的成績，為後續模型的性能提升提供了參考基準。
- **預訓練語言模型顯著提升了性能：** 第二次則實驗引入了 **DistilBERT** 預訓練模型，並使用 Hugging Face 提供的 **Trainer** 工具進行微調，最終在 kaggle 取得 **0.49449**，顯示了深度學習在文本語義理解上的優勢。
- **混合特徵架構展現了更高的分類能力：** 第三次實驗將 DistilBERT 的嵌入特徵與經過 Feature Engineering 處理的結構特徵（如文本長度、情感詞彙計數、降維後的 TF-IDF）結合，構建了自定義的混合模型架構，成績提升至 **0.50887**。
- **增強降維進一步優化模型表現：** 第四次實驗對降維特徵進行了優化，將 Truncated SVD 的特徵維度從 50 增加到 80，保留了更多信息，最終達到實驗中的最佳結果 **0.50889**。
- **集成策略未能超越單一最佳模型：** 採用多數決策策略（Majority Voting）集成三個模型的預測結果，成績比起最佳單一模型 Model 3，微降至 **0.50774**。分析表明，模型間的偏差未能充分互補、特徵處理的相似性和數據不平衡問題可能限制了集成策略的效果。