Information Retrieval and Extraction HW2

312657008_ 江祐姍

1. 資料前處理與資訊檢索方法

在資料前處理中，我進行了以下步驟：

文本清理：移除了超連結、數字、標點符號和多餘的空白，並將文本轉換為小寫，保證文本的一致性。停用詞去除：使用了 NLTK 提供的英語停用詞集，並將其從文本中過濾掉，避免這些常見的無實際意義的詞語影響模型效果。

```
def preprocess_text(text: str, stop_words: set) -> str:
    patterns_to_remove = [
        r'link',  # 重複超連結描述
        r'alternate',
        r'shorturl',
        r'profile',
        r'Submit',
        r'Overview',
        r'canonical',       # Canonical 錯誤標記
        r'https?://\S+|www\.\S+'  # 超連結
    ]
    for pattern in patterns_to_remove:
        text = re.sub(pattern, '', text)
    text = text.lower()
    text = re.sub(r'_{2,}', ' ', text)  # 去除連續兩個或以上的下劃線
    text = re.sub(r'-{2,}', ' ', text)  # 去除連續兩個或以上的中劃線
    text = re.sub(r'\d+', '', text)  # 去除數字
    text = re.sub(r'[^\w\s]', '', text)  # 去除標點
    text = re.sub(r'\s+', ' ', text)  # 去除多餘空白
    text = re.sub(r"\*", "", text)
    tokens = word_tokenize(text)
    tokens = [token for token in tokens if token not in stop_words]
    return ' '.join(tokens)

nltk.download('punkt')
nltk.download('stopwords')
stop_words = set(stopwords.words('english'))
```

比較兩種常見的資訊檢索方法：

(1) TF-IDF (Term Frequency - Inverse Document Frequency)：此方法用來表示文本中每個詞的重要性，能有效減少常見詞的權重，強化具有區分度的詞語。

```
        self.vectorizer = TfidfVectorizer(
            max_features=5000,
```

```
            ngram_range=(1, 2)
        )

        X = self.vectorizer.fit_transform(df['content'])
        y = df['rating']
```

(2) Doc2Vec 通過學習文檔的上下文信息，將文檔映射到一個向量空間中，使得相似的文檔能夠在向量空間中靠得更近。這種方法在處理長文本、文檔相似度計算以及信息檢索等任務中具有很大的應用潛力。

```
        self.doc2vec_model = Doc2Vec(
                vector_size=250,
                window=5,
                min_count=1,
                workers=multiprocessing.cpu_count(),
                epochs=30
        self.doc2vec_model.build_vocab(documents)
```

綜合比較:TF-IDF 在簡單的文本檢索任務中（如關鍵詞檢索、短文本分類）表現優異，尤其是在資料量較小的情況下，計算速度較快。Doc2Vec 更適用於語義相似度檢索和長文本的處理，因為它能捕捉整篇文檔的語義結構，能夠在大規模語料庫中發揮更大的作用。然而，Doc2Vec 需要更多的計算資源和時間來訓練模型。

2. 模型選擇

(1) XGBoost

經過多方嘗試得到最佳參數如下:

'objective': 'multi:softmax'：設定為多類別分類任務，輸出類別標籤。

'num_class': 3：指定目標變量有三個類別（根據需求修改為實際類別數）。

'max_depth': 6：樹的最大深度，控制模型的複雜度。較大的深度可能會導致過擬合。

'eta': 0.3：學習率，控制每次迭代對最終模型的貢獻程度，較低的學習率會使模型收斂更加平穩，但需要更多的迭代輪次。

'tree_method': 'gpu_hist'：這個設置告訴 XGBoost 使用 GPU 進行加速，若無法使用 GPU，則會使用 CPU 版本（hist）。

'predictor': 'gpu_predictor'：設定預測過程也使用 GPU 加速。

'eval_metric': 'mlogloss'：評估指標，這裡使用對數損失（log loss）來衡量多類別分類的性能

'nthread': 12：設定並行運行的執行緒數量，這有助於加速訓練過程

```
        params = {
            'objective': 'multi:softmax',
            'num_class': 3,
            'max_depth': 6,
            'eta': 0.3,
            'tree_method': 'gpu_hist' if self.use_gpu else 'hist',
            'predictor': 'gpu_predictor' if self.use_gpu else 'cpu_predictor',
            'eval_metric': 'mlogloss',
            'nthread': 12
}
```

```
        # 訓練模型
        self.logger.info(" 開始 XGBoost 訓練...")
        self.model = xgb.train(
            params,
            dtrain,
            num_boost_round=333,
            evals=[(dtrain, 'train')],
            early_stopping_rounds=10
        )
```

dtrain: 這是訓練資料，必須轉換成 xgboost.DMatrix 格式。這是 XGBoost 的資料格式，它會對資料進行內部優化以加速訓練過程。

num_boost_round: 訓練輪數，即迭代的次數。設定為 333 意味著模型會進行 333 輪訓練，這一過程中會逐漸更新樹的結構。

early_stopping_rounds: 若連續 10 輪訓練都未提高模型的效能，則提前停止訓練，防止過擬合。

(2) LSTM[預測結果不太好]

Embedding 層：將輸入的詞索引映射到密集的向量空間中，這樣可以將稀疏的詞彙轉換為較低維度的向量表示。

Bidirectional LSTM：LSTM 層包裝在雙向（Bidirectional）層中，這樣可以讓模型同時捕捉到序列中從前向後和從後向前的信息。這在處理序列數據時通常能提供更好的表現。

Dropout 層：Dropout 用來減少過擬合，隨機丟棄神經網絡中的一部分神經元。

Batch Normalization 層：這層可以加速訓練，並改善模型的穩定性。

Dense 層：全連接層，用來進行最終的分類。

```
    def create_model(self):

        input_layer = tf.keras.layers.Input(shape=(self.max_len,), dtype='int32', name='input_layer')

        # Embedding layer
        embedding = tf.keras.layers.Embedding(
            input_dim=self.max_words + 1,
            output_dim=64,
            input_length=self.max_len,
            mask_zero=True,
            name='embedding_layer'
        )(input_layer)


        spatial_dropout = tf.keras.layers.SpatialDropout1D(0.2)(embedding)

        lstm_1 = tf.keras.layers.Bidirectional(
            tf.keras.layers.LSTM(128, return_sequences=True)
        )(spatial_dropout)
        dropout_1 = tf.keras.layers.Dropout(0.2)(lstm_1)

        lstm_2 = tf.keras.layers.Bidirectional(
            tf.keras.layers.LSTM(64)
        )(dropout_1)
```

```
        dropout_2 = tf.keras.layers.Dropout(0.2)(lstm_2)


        dense_1 = tf.keras.layers.Dense(64, activation='relu')(dropout_2)
        batch_norm = tf.keras.layers.BatchNormalization()(dense_1)
        dropout_3 = tf.keras.layers.Dropout(0.2)(batch_norm)


        output_layer = tf.keras.layers.Dense(3, activation='softmax', name='output_layer')(dropout_3)


        model = tf.keras.Model(inputs=input_layer, outputs=output_layer)


        optimizer = tf.keras.optimizers.Adam(learning_rate=0.001)
        model.compile(
            optimizer=optimizer,
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )

      return model
```

(3) Groq LLM API[效果最差]

```
from groq import Groq
import os

import time


client = Groq(
    api_key="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"
)

max_length = 1000

for i in range(len(test)):
    article_id = test['id'].iloc[i]
    article = test['content'].iloc[i]

    if len(article) > max_length:
        article = article[:max_length]

        # 發送請求，並要求只返回 0、1、2 的數字
    chat_completion = client.chat.completions.create(
    messages=[{
                "role": "user",
                "content": f"classify it into a numerical format: 0 (False),
                1 (Partial True), 2 (True). Only return the number without any
                additional text.{article}"
            }],
            model="llama-3.1-8b-instant",
```

```
            stream=False,
        )


    result = chat_completion.choices[0].message.content.strip()
    print(result)

    time.sleep(1)
```

還有使用模型"llama3-8b-8192"，但效果都不佳且需要持續更換數個 API 才能完成

3. 探討

在使用線上 LLM 模型時，我發現模型幾乎從未分類到 0，預測結果大多集中在 1 和 2 這兩個類別，這與我最初觀察到的訓練集和驗證集資料分佈有顯著差異。在原始資料中，0 和 1 的數量相近，而 2 的樣本數量則較少。但在模型的預測中，這種情況並未反映出來。造成這一差異的可能原因，首先可能是前處理過程中資料處理不夠乾淨；另外，由於 token 長度的限制，我只選取了前一萬個字，這可能使模型未能充分捕捉到所有關鍵特徵。