自然語言處理與文件探勘

議論探勘
(ARGUMENT MINING)

110820031 翁廷豪
110820048 許家睿

議 論 探 勘

目錄

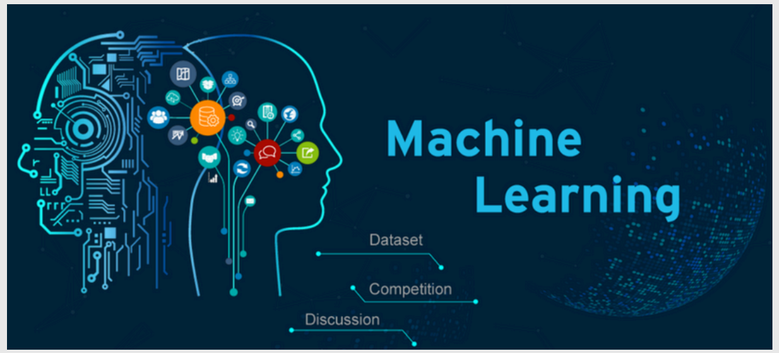
前言

議論探勘（ARGUMENT MINING）

給定一個主張（claim）與一個前提（premise），由模型判斷兩者之間的議論關係，例如該項前提為支持或反駁該主張。

換言之，這是一個典型的分類任務，判斷一組主張與前提的關係。

資料集介紹



資料來源:
2022自然語言理解的解釋性資訊標記競賽

```
['8', '"It can go both ways . We all doubt . It is what you do with it that matters ."', '"True ."', 'AGREE']
['8', '"It can go both ways . We all doubt . It is what you do with it that matters ."', '"True ."', 'AGREE']
['8', '"It can go both ways . We all doubt . It is what you do with it that matters ."', '"True ."', 'AGREE']
['9', '"once again , you seem to support the killing of certain people ... based on what ?"', '"based on the idea that people a
re dispensible , particularly if they obstruct your well-being . a woman would abort her baby because being a mother contradict
s her idea of well-being . in the same way we send soldiers to kill the enemy if they are deemed contrary to the well-being of
our country can you be against abortion and pro-war ?"', 'AGREE']
['9', '"once again , you seem to support the killing of certain people ... based on what ?"', '"based on the idea that people a
re dispensible , particularly if they obstruct your well-being . a woman would abort her baby because being a mother contradict
s her idea of well-being . in the same way we send soldiers to kill the enemy if they are deemed contrary to the well-being of
our country can you be against abortion and pro-war ?"', 'AGREE']
行數: 38326
```

資料集前處理

```python
import csv
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import numpy as np

# 讀取數據並去重
def load_data(file_path):
    data = []
    with open(file_path, 'r', encoding='latin-1') as csvfile:
        csv_reader = csv.reader(csvfile)
        next(csv_reader)  # 跳過標題行
        for row in csv_reader:
            if row[3] in ['AGREE', 'DISAGREE'] and not any(cell.strip() == '' for cell in row[:4]):
                data.append(tuple(row[:4]))
    unique_data = list(set(data))
    return np.array(unique_data)

# 載入資料
train_data = load_data('train_data.csv')

# 取得每筆資料的 q 和 r 欄位
q_data = [row[1] for row in train_data]
r_data = [row[2] for row in train_data]

# 將 q 和 r 合併成一個新的列表
combined_data = [q + ' ' + r for q, r in zip(q_data, r_data)]

# 取得目標值 s
y = [row[3] for row in train_data]
encoder = LabelEncoder()
y = encoder.fit_transform(y)
```

模型訓練

SVM

RNNs

BERT

Random Forest

GRU

CNNs

議論探勘

RECURRENT NEURAL NETWORKS (RNNS)

循環神經網絡是一種適合處理序列數據的神經網絡。

RNN通過其內部循環結構，能夠記憶序列中的信息，適用於自然語言處理和時間序列分析等任務。

優點：

- 能夠處理序列數據，捕捉數據中的時間依賴性
- 適合處理自然語言處理相關任務

缺點：

- 訓練過程中容易出現梯度消失問題
- 訓練時間較長，計算資源消耗大

議論探勘

RNNS

->模型構建

```python
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, LSTM, Dense, Dropout, Bidirectional

# 構建RNN模型
model = Sequential()
model.add(Embedding(input_dim=5000, output_dim=128, input_length=100))
model.add(Bidirectional(LSTM(64)))
model.add(Dropout(0.5))
model.add(Dense(1, activation='sigmoid'))

# 編譯模型
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])
```

->訓練模型

```python
# 訓練模型
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
767/767 [==============================] - 108s 135ms/step - loss: 0.3528 - accuracy: 0.8635 - val_loss: 0.2445 - val_accuracy:
0.9132
Epoch 2/5
767/767 [==============================] - 109s 142ms/step - loss: 0.1301 - accuracy: 0.9591 - val_loss: 0.1382 - val_accuracy:
0.9576
Epoch 3/5
767/767 [==============================] - 106s 138ms/step - loss: 0.0487 - accuracy: 0.9865 - val_loss: 0.1099 - val_accuracy:
0.9731
Epoch 4/5
767/767 [==============================] - 100s 130ms/step - loss: 0.0280 - accuracy: 0.9931 - val_loss: 0.1150 - val_accuracy:
0.9742
Epoch 5/5
767/767 [==============================] - 101s 132ms/step - loss: 0.0175 - accuracy: 0.9957 - val_loss: 0.0941 - val_accuracy:
0.9822
```

RNNS

->訓練結果

Accuracy : 0.75
F1 score : 0.745
Precision : 0.735
Recall : 0.745

```
232/232 [==============================] - 6s 27ms/step
231/231 [==============================] - 8s 34ms/step
Public Test Data:
Accuracy: 0.7546839196657231
              precision    recall  f1-score   support

           0       0.28      0.23      0.25      1322
           1       0.84      0.87      0.85      6097

    accuracy                           0.75      7419
   macro avg       0.56      0.55      0.55      7419
weighted avg       0.74      0.75      0.75      7419


Private Test Data:
Accuracy: 0.7547731888964117
              precision    recall  f1-score   support

           0       0.30      0.23      0.26      1371
           1       0.83      0.87      0.85      6014

    accuracy                           0.75      7385
   macro avg       0.56      0.55      0.56      7385
weighted avg       0.73      0.75      0.74      7385
```

議論探勘

GATED RECURRENT UNIT (GRU)

GRU是一種改進的循環神經網絡（RNN），旨在解決RNN中的梯度消失問題。
GRU通過引入更新門和重置門來控制信息流，以提高模型的記憶能力和訓練效率。

優點：

- 與LSTM相比，結構簡單，訓練速度較快
- 能夠有效處理長序列數據，捕捉長距依賴

缺點：

- 訓練過程中依然可能出現梯度消失問題
- 對於非常長的序列數據，表現可能不如Transformer

議論探勘

GRU

->模型構建

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, GRU, Dense, Dropout

# 參數設定
embedding_dim = 64
gru_units = 64

# 建立GRU模型
model = Sequential([
    Embedding(len(word_index) + 1, embedding_dim, input_length=max_length),
    GRU(gru_units, return_sequences=True),
    Dropout(0.2),
    GRU(gru_units),
    Dropout(0.2),
    Dense(64, activation='relu'),
    Dense(1, activation='sigmoid')
])

# 編譯模型
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 顯示模型摘要
model.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 embedding_1 (Embedding)     (None, 128, 64)           2017472

 gru_2 (GRU)                 (None, 128, 64)           24960

 dropout_2 (Dropout)         (None, 128, 64)           0

 gru_3 (GRU)                 (None, 64)                24960

 dropout_3 (Dropout)         (None, 64)                0

 dense_2 (Dense)             (None, 64)                4160

 dense_3 (Dense)             (None, 1)                 65

=================================================================
Total params: 2071617 (7.90 MB)
Trainable params: 2071617 (7.90 MB)
Non-trainable params: 0 (0.00 Byte)
```

->訓練模型

```python
# 訓練模型
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
767/767 [==============================] - 153s 192ms/step - loss: 0.4359 - accuracy: 0.8358 - val_loss: 0.3897 - val_accuracy:
0.8487
Epoch 2/5
767/767 [==============================] - 144s 188ms/step - loss: 0.3374 - accuracy: 0.8765 - val_loss: 0.3311 - val_accuracy:
0.8748
Epoch 3/5
767/767 [==============================] - 139s 182ms/step - loss: 0.2251 - accuracy: 0.9162 - val_loss: 0.1660 - val_accuracy:
0.9408
Epoch 4/5
767/767 [==============================] - 140s 182ms/step - loss: 0.0749 - accuracy: 0.9769 - val_loss: 0.1048 - val_accuracy:
0.9721
Epoch 5/5
767/767 [==============================] - 141s 184ms/step - loss: 0.0316 - accuracy: 0.9913 - val_loss: 0.0758 - val_accuracy:
0.9804
```

GRU

->訓練結果

Accuracy : 0.75
F1 score : 0.735
Precision : 0.715
Recall : 0.755

```
232/232 [==============================] - 12s 50ms/step
231/231 [==============================] - 10s 43ms/step
Public Test Data:
Accuracy: 0.7623668958080604
              precision    recall  f1-score   support

           0       0.24      0.15      0.19      1322
           1       0.83      0.89      0.86      6097

    accuracy                           0.76      7419
   macro avg       0.53      0.52      0.52      7419
weighted avg       0.72      0.76      0.74      7419


Private Test Data:
Accuracy: 0.7473256601218686
              precision    recall  f1-score   support

           0       0.26      0.19      0.22      1371
           1       0.83      0.87      0.85      6014

    accuracy                           0.75      7385
   macro avg       0.54      0.53      0.54      7385
weighted avg       0.72      0.75      0.73      7385
```

議論探勘

CONVOLUTIONAL NEURAL NETWORKS (CNNS)

CNN是一種專門處理圖像數據的深度學習模型，它通過卷積層提取圖像中的局部特徵，在多種計算機視覺任務中表現優異。

優點：

- 能夠有效提取局部特徵
- 訓練速度快，計算資源需求相對較低

缺點：

- 對於序列數據和長距依賴關係捕捉效果不佳
- 在處理文本數據時，需進行適當的特徵工程

議論探勘

CNNS

->模型構建

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout

# 參數設定
embedding_dim = 64
num_filters = 64
kernel_size = 5
dropout_rate = 0.5

# 建立CNN模型
model = Sequential([
    Embedding(len(word_index) + 1, embedding_dim, input_length=max_length),
    Conv1D(filters=num_filters, kernel_size=kernel_size, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(dropout_rate),
    Dense(1, activation='sigmoid')
])

# 編譯模型
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 顯示模型摘要
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 128, 64)           2017472

conv1d (Conv1D)              (None, 124, 64)           20544

global_max_pooling1d (Glob   (None, 64)                0
alMaxPooling1D)

dense (Dense)                (None, 64)                4160

dropout (Dropout)            (None, 64)                0

dense_1 (Dense)              (None, 1)                 65

=================================================================
Total params: 2042241 (7.79 MB)
Trainable params: 2042241 (7.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

->訓練模型

```python
# 訓練模型
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
767/767 [==============================] - 46s 59ms/step - loss: 0.3421 - accuracy: 0.8682 - val_loss: 0.1430 - val_accuracy:
0.9560
Epoch 2/5
767/767 [==============================] - 47s 62ms/step - loss: 0.0626 - accuracy: 0.9834 - val_loss: 0.0627 - val_accuracy:
0.9814
Epoch 3/5
767/767 [==============================] - 45s 58ms/step - loss: 0.0204 - accuracy: 0.9965 - val_loss: 0.0630 - val_accuracy:
0.9852
Epoch 4/5
767/767 [==============================] - 45s 59ms/step - loss: 0.0149 - accuracy: 0.9978 - val_loss: 0.0663 - val_accuracy:
0.9868
Epoch 5/5
767/767 [==============================] - 46s 60ms/step - loss: 0.0128 - accuracy: 0.9980 - val_loss: 0.0690 - val_accuracy:
0.9876
```

CNNS

->訓練結果

Accuracy : 0.786
F1 score : 0.76
Precision : 0.745
Recall : 0.785

```
232/232 [==============================] - 2s 8ms/step
231/231 [==============================] - 2s 7ms/step
Public Test Data:
Accuracy: 0.7839331446286562
              precision    recall  f1-score   support

           0       0.31      0.17      0.22      1322
           1       0.84      0.92      0.87      6097

    accuracy                           0.78      7419
   macro avg       0.57      0.55      0.55      7419
weighted avg       0.74      0.78      0.76      7419


Private Test Data:
Accuracy: 0.7880839539607312
              precision    recall  f1-score   support

           0       0.36      0.18      0.24      1371
           1       0.83      0.93      0.88      6014

    accuracy                           0.79      7385
   macro avg       0.59      0.55      0.56      7385
weighted avg       0.74      0.79      0.76      7385
```

SVM

支持向量機是一種二類分類模型，它通過在高維空間中尋找一個最佳超平面來將不同類別的數據點分開。

優點：

- 在小型數據集上表現良好
- 訓練速度快，對記憶體需求低
- 對於高維數據的效果好

缺點：

- 對於大型數據集訓練時間較長
- 不適合處理大量特徵的情境

議 論 探 勘

SVM

->模型構建

```python
import tensorflow as tf
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense, Dropout

# 參數設定
embedding_dim = 64
num_filters = 64
kernel_size = 5
dropout_rate = 0.5

# 建立CNN模型
model = Sequential([
    Embedding(len(word_index) + 1, embedding_dim, input_length=max_length),
    Conv1D(filters=num_filters, kernel_size=kernel_size, activation='relu'),
    GlobalMaxPooling1D(),
    Dense(64, activation='relu'),
    Dropout(dropout_rate),
    Dense(1, activation='sigmoid')
])

# 編譯模型
model.compile(loss='binary_crossentropy', optimizer='adam', metrics=['accuracy'])

# 顯示模型摘要
model.summary()
```

```
Layer (type)                 Output Shape              Param #
=================================================================
embedding (Embedding)        (None, 128, 64)           2017472

conv1d (Conv1D)              (None, 124, 64)           20544

global_max_pooling1d (Glob   (None, 64)                0
alMaxPooling1D)

dense (Dense)                (None, 64)                4160

dropout (Dropout)            (None, 64)                0

dense_1 (Dense)              (None, 1)                 65

=================================================================
Total params: 2042241 (7.79 MB)
Trainable params: 2042241 (7.79 MB)
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
```

->訓練模型

```python
# 訓練模型
history = model.fit(X_train, y_train, epochs=5, batch_size=32, validation_split=0.2)
```

```
Epoch 1/5
767/767 [==============================] - 46s 59ms/step - loss: 0.3421 - accuracy: 0.8682 - val_loss: 0.1430 - val_accuracy:
0.9560
Epoch 2/5
767/767 [==============================] - 47s 62ms/step - loss: 0.0626 - accuracy: 0.9834 - val_loss: 0.0627 - val_accuracy:
0.9814
Epoch 3/5
767/767 [==============================] - 45s 58ms/step - loss: 0.0204 - accuracy: 0.9965 - val_loss: 0.0630 - val_accuracy:
0.9852
Epoch 4/5
767/767 [==============================] - 45s 59ms/step - loss: 0.0149 - accuracy: 0.9978 - val_loss: 0.0663 - val_accuracy:
0.9868
Epoch 5/5
767/767 [==============================] - 46s 60ms/step - loss: 0.0128 - accuracy: 0.9980 - val_loss: 0.0690 - val_accuracy:
0.9876
```

SVM

->訓練結果

核函數 : 線性
隨機狀態 : 42

Accuracy : 0.7866
F1 score : 0.76
Precision : 0.745
Recall : 0.785

Public Test Data:
Accuracy: 0.7804286291953093

	precision	recall	f1-score	support
AGREE	0.28	0.15	0.19	1322
DISAGREE	0.83	0.92	0.87	6097
accuracy			0.78	7419
macro avg	0.56	0.53	0.53	7419
weighted avg	0.73	0.78	0.75	7419

Private Test Data:
Accuracy: 0.7928232904536222

	precision	recall	f1-score	support
AGREE	0.40	0.23	0.29	1371
DISAGREE	0.84	0.92	0.88	6014
accuracy			0.79	7385
macro avg	0.62	0.57	0.58	7385
weighted avg	0.76	0.79	0.77	7385

RANDOM FOREST

隨機森林是一種基於決策樹的集成學習方法。它通過構建多棵決策樹並在分類時輸出最多數樹的分類結果，以提高模型的穩定性和準確性。

優點：

- 能夠處理高維數據和大量特徵
- 對於過擬合有良好的抗性
- 適合處理非線性數據

缺點：

- 訓練時間和資源消耗較高
- 解釋性較差，模型較為黑盒

RF

->特徵提取/切分資料集

```
###Step 2: 特徵提取
```

```
from sklearn.feature_extraction.text import TfidfVectorizer

# 使用 TF-IDF 向量化器將文本資料轉換成數值特徵向量
vectorizer = TfidfVectorizer(max_features=5000)
X = vectorizer.fit_transform(combined_data)
```

```
###Step 3: 切分訓練集和測試集
```

```
from sklearn.model_selection import train_test_split

# 切分訓練集和測試集
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
```

->訓練模型

```
from sklearn.ensemble import RandomForestClassifier

# 使用隨機森林模型進行訓練
rf_model = RandomForestClassifier(n_estimators=100, random_state=42)
rf_model.fit(X_train, y_train)
```

RANDOM FOREST

->訓練結果

Accuracy : 0.82
F1 score : 0.75
Precision : 0.765
Recall : 0.82

Public Test Data:
Accuracy: 0.8222132362852136

	precision	recall	f1-score	support
0	0.53	0.02	0.04	1322
1	0.82	1.00	0.90	6097
accuracy			0.82	7419
macro avg	0.68	0.51	0.47	7419
weighted avg	0.77	0.82	0.75	7419

Private Test Data:
Accuracy: 0.8192281651997292

	precision	recall	f1-score	support
0	0.69	0.05	0.09	1371
1	0.82	1.00	0.90	6014
accuracy			0.82	7385
macro avg	0.76	0.52	0.49	7385
weighted avg	0.80	0.82	0.75	7385

TRANSFORMER-BASED MODELS (BERT)

一種基於Transformer的預訓練語言模型。它通過在大規模語料上進行雙向預訓練，能夠捕捉句子中的上下文信息，並在多種自然語言處理任務中表現出色。

優點：

- 能夠處理長距依賴關係，效果顯著
- 在多種自然語言處理任務中表現優異

缺點：

- 訓練成本高，計算資源需求大
- 模型複雜度高，需大量數據進行預訓練

BERT ->模型構建

```
!pip install transformers

from transformers import BertTokenizer, TFBertForSequenceClassification
import tensorflow as tf

# 下載BERT tokenizer和模型
tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
model = TFBertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=2)

# 將文本數據轉換為BERT輸入格式
def encode_data(text_list, tokenizer, max_length=128):
    input_ids = []
    attention_masks = []

    for text in text_list:
        encoded_dict = tokenizer.encode_plus(
            text,
            add_special_tokens=True,
            max_length=max_length,
            pad_to_max_length=True,
            return_attention_mask=True,
            return_tensors='tf'
        )
        input_ids.append(encoded_dict['input_ids'])
        attention_masks.append(encoded_dict['attention_mask'])

    input_ids = tf.concat(input_ids, axis=0)
    attention_masks = tf.concat(attention_masks, axis=0)

    return input_ids, attention_masks

# 編碼訓練和測試數據
X_train_ids, X_train_masks = encode_data(X_train, tokenizer)
X_test_ids, X_test_masks = encode_data(X_test, tokenizer)
```

->訓練模型

```
# 定義模型編譯和訓練
optimizer = tf.keras.optimizers.Adam(learning_rate=2e-5, epsilon=1e-8)
loss = tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True)
metrics = ['accuracy']

model.compile(optimizer=optimizer, loss=loss, metrics=metrics)

# 訓練模型
model.fit(
    [X_train_ids, X_train_masks],
    y_train,
    batch_size=16,
    epochs=3,
    validation_split=0.2
)
```

BERT

->訓練結果

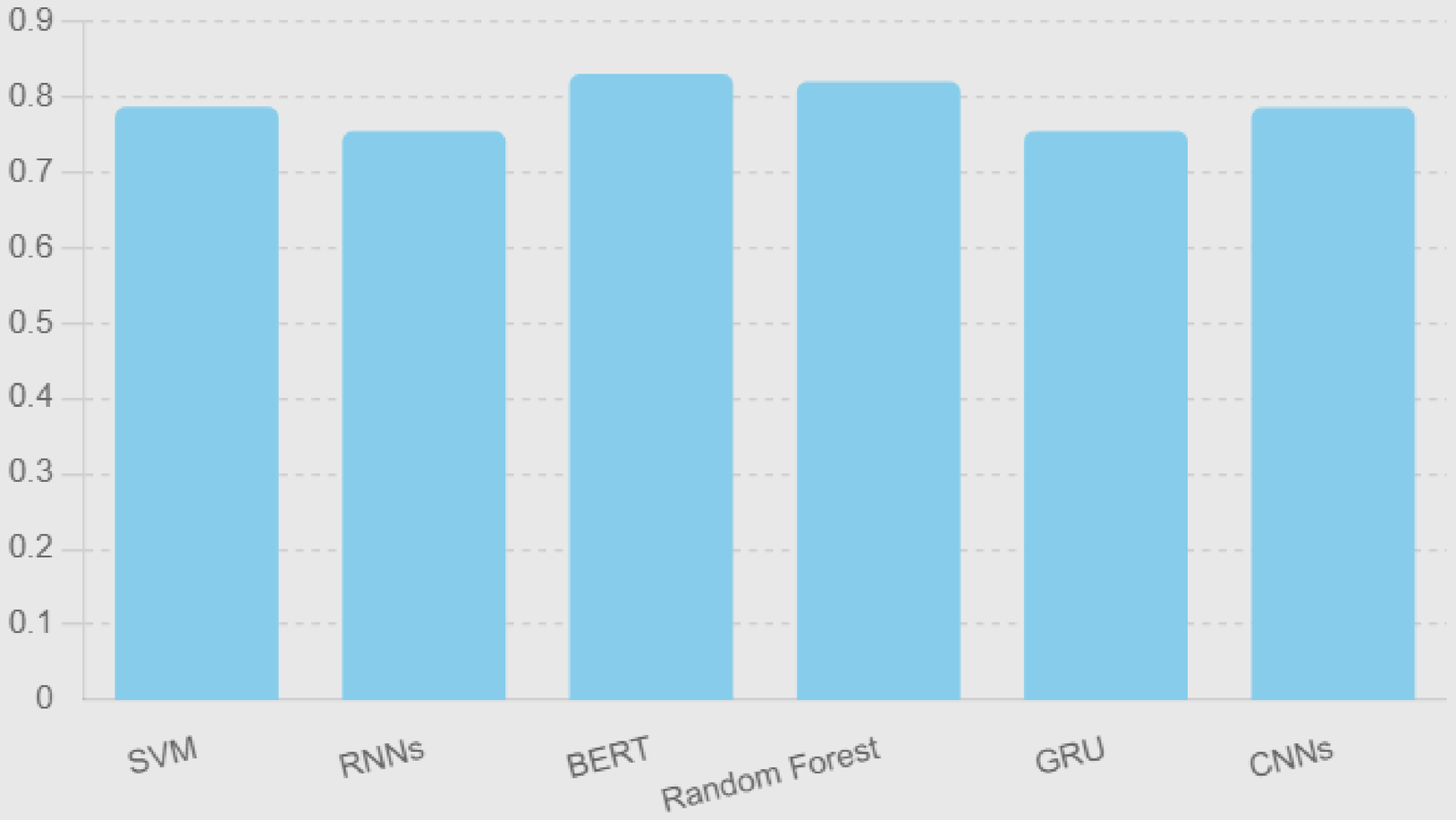
Accuracy : 0.83
F1 score : 0.81
Precision : 0.8
Recall : 0.83

```
32/32 [==============================] - 31s 972ms/step
32/32 [==============================] - 31s 973ms/step
Public Test Data:
Accuracy: 0.8214285714285714
              precision    recall  f1-score   support

           0       0.50      0.28      0.36       180
           1       0.86      0.94      0.90       828

    accuracy                           0.82      1008
   macro avg       0.68      0.61      0.63      1008
weighted avg       0.79      0.82      0.80      1008


Private Test Data:
Accuracy: 0.8392857142857143
              precision    recall  f1-score   support

           0       0.61      0.36      0.45       187
           1       0.87      0.95      0.91       821

    accuracy                           0.84      1008
   macro avg       0.74      0.65      0.68      1008
weighted avg       0.82      0.84      0.82      1008
```

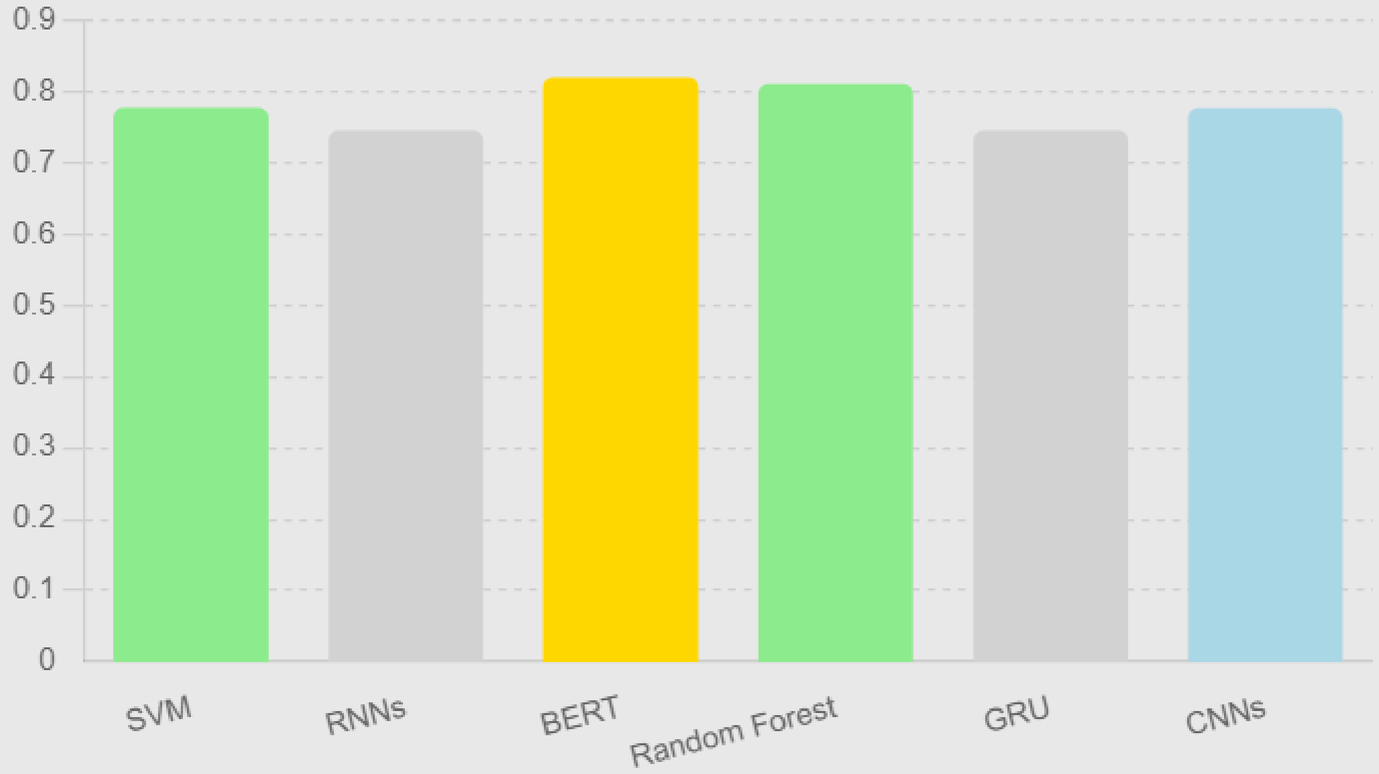
議 論 探 勘

模型訓練結果

Average Accuracy



結論



- **金色（Best Performance）**：BERT模型在性能上表現最佳，建議在未來的相關任務中優先使用。
- **淺綠色（Good for Training Time & Interpretability）**：SVM和Random Forest模型在訓練時間和模型解釋性方面具有優勢。
- **淺藍色（Good for Limited Resources）**：CNN模型適合於資源有限且需要快速實施的應用場景。
- **灰色（Not Recommended）**：RNNs和GRU模型在本次實驗中表現不佳，不推薦使用。

報告結束

感謝聆聽

110820031 翁廷豪
110820048 許家睿