2020計算機程式 期末專題發表報名

真的? 假的?!

Twitter災難性貼文之真實性判斷

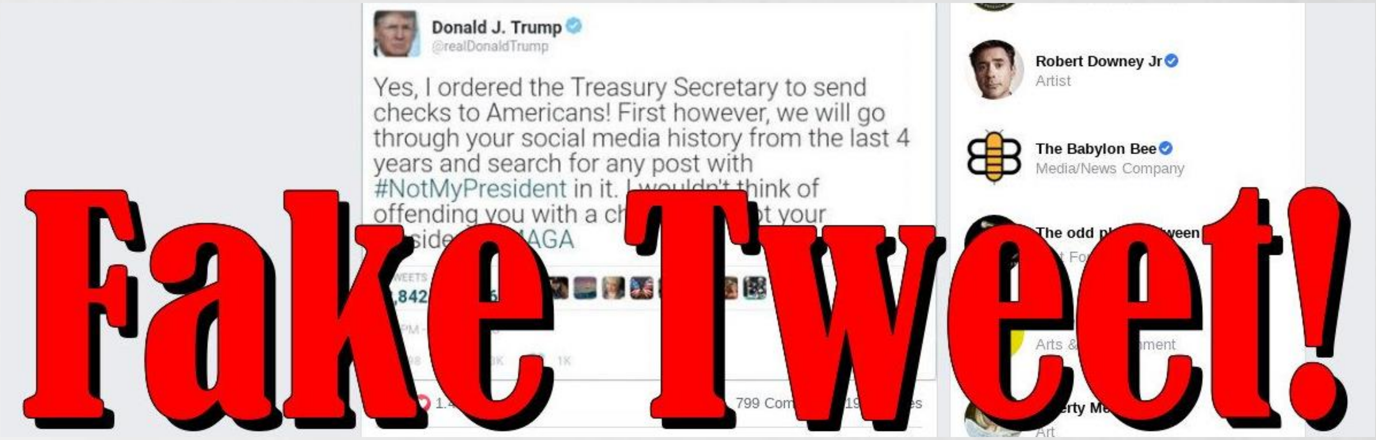
許智超 李郁萱 楊宗泰

目錄

CONTENT

01 選題動機

選題動機
Reasons for Choosing this Topic



01 假貼文猖獗
輕則造成財物損失，重可造成人員傷亡

02 NLP 分類問題應用廣泛，如: 電影評論

03 打擊假貼文，為眾多國家與社交媒體平台公司近年首要之務，為 NLP 重要的發展領域

02 專案介紹

題目介紹
Introduction

題目啟發

- Kaggle Ongoing Competition
- 任務: 對災難性推特貼文，判斷真實性
- 評分: 放上Kaggle競賽平台，由平台公正資料測試預測準確度

我們的專案

1. 下載Kaggle資料集
2. 訓練機器學習和深度學習模型，解決NLP任務
3. **LineBot聊天機器人**:
    - 使用者介面
    - 即時查詢留言，快速判斷真假

03 資料庫與資料介紹

資料庫與資料集介紹

Data Description

- Id：每個貼文-Tweet的獨特編號
- Text: 貼文內文
- Location: 貼文發送位置
- Keyword: 貼文中特定關鍵詞
- Target: 貼文是否為真實災難
若真則值為1；否則值為0
只有Train Data有

Train Data

```
In [5]: df_train.sample(n=10, random_state=1)
Out[5]:
```

	id	keyword	location	text	target
3228	4632	emergency%20services	Sydney, New South Wales	Goulburn man Henry Van Bilsen missing: Emergen...	1
3706	5271	fear	NaN	The things we fear most in organizations--fluc...	0
6957	9982	tsunami	Land Of The Kings	@tsunami_esh ?? hey Esh	0
2887	4149	drown	NaN	@POTUS you until you drown by water entering t...	0
7464	10680	wounds	cody, austin follows ?*?	Crawling in my skin\nThese wounds they will no...	1
2539	3643	desolation	Istanbul	#np agalloch - the desolation song	0
6837	9794	trapped	NaN	Hollywood Movie About Trapped Miners Released ...	1
7386	10570	windstorm	Houston	New roof and hardy up..Windstorm inspection to...	0
1506	2174	catastrophic	Inexpressible Island	The Catastrophic Effects of Hiroshima and Naga...	1
1875	2694	crush	Everywhere	tiffanyfrizzell has a crush: http://t.co/RaF73...	0

Test Data

```
df_test.sample(n=10, random_state=1)
```

	id	keyword	location	text
1787	6035	heat%20wave	Brooklyn, NY	I added some dumb ideas to beat the #summer he...
666	2168	catastrophic	NaN	If a â£1 rise in wages is going to have such a...
93	317	annihilated	NaN	How do I step outside for 5 seconds and get an...
2924	9682	tornado	Canada	Tornado warnings issued for southern Alberta h...
1735	5857	hailstorm	Calgary, Alberta, Canada	Get out of the hailstorm and come down to @The...
1296	4267	drowning	real world	and no one knows that i'm drowning and i know ...
2467	8248	rioting	GO Bucks!	we shootn each other over video games...........
1587	5364	fire%20truck	Lampe, MO	What a night! Kampers go on vacation to end ou...
1336	4414	electrocute	NaN	Photo: weallheartonedirection: I wouldn‰Ûªt le...
2132	7133	military	Worldwide	Obama warns there will be another war without ...

04 實作規劃

實作規劃
Project Planning

DL Model

BERT

XLNet

正則表達式

LineBot

05 語料預處理

預處理: 正則表達式

- ◆移除顏文字和一些怪符號
- ◆移除 hashtag 的符號'#'
- ◆移除人名標記
- ◆移除網址
- ◆全部小寫化
- ◆移除無關用語: wa, ha, ve

```python
def clean(text):
    # 移除顏文字和一些怪符號
    reg = re.compile('\\.+?(?=\B|$)')
    text = text.apply(lambda r: re.sub(reg, string = r, repl = ''))
    reg = re.compile('\x89Û_')
    text = text.apply(lambda r: re.sub(reg, string = r, repl = ' '))
    reg = re.compile('\&amp')
    text = text.apply(lambda r: re.sub(reg, string = r, repl = '&'))
    reg = re.compile('\\n')
    text = text.apply(lambda r: re.sub(reg, string = r, repl = ' '))
    # 移除 hashtag 的符號'#'
    text = text.apply(lambda r: r.replace('#', ''))
    # 移除人名標記
    reg = re.compile('@\w+')#\w 匹配字母或数字、英文字母或汉字
    text = text.apply(lambda r: re.sub(reg, string = r, repl = '@'))
    # 移除網址
    reg = re.compile('https?\S+(?=\s|$)')
    text = text.apply(lambda r: re.sub(reg, string = r, repl = ' '))
    # 全部小寫化
    text = text.apply(lambda r: r.lower())
    # 移除無關用語
    text = text.apply(lambda r: r.replace('wa', ' '))
    text = text.apply(lambda r: r.replace('ha', ' '))
    text = text.apply(lambda r: r.replace('ve', ' '))
    return text
```

資料清理結果

```
train['cleaned'] = clean(train['text'])
test['cleaned'] = clean(test['text'])
```

```
train.head()
```

	id	keyword	location	text	target	cleaned
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	our deeds are the reason of this earthquake ma...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	all residents asked to 'shelter in place' are ...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13,000 people receive wildfires evacuation ord...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	just got sent this photo from ruby alaska as s...

資料處理(訓練資料)

```
Max_Len = 84
#df_train = df_train[~(df_train.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
Sample_Fraction = 0.5
df_train = df_train.sample(frac = Sample_Fraction, random_state = 9527)
```

```
df_train = df_train.reset_index()
df_train = df_train.loc[:,['cleaned', 'target']]
```

```
df_train.columns = ['cleaned', 'target']
```

```
len(df_train)
```

```
3806
```

```
df_train.head()
```

	cleaned	target
0	@ @ no one is rioting burning down buildings o...	1
1	stu put beetroot in his cake and even lost to ...	0
2	hot funtenna: hijacking computers to send dat...	1
3	1 of those days when ya don't realize till alr...	1
4	fresh out da shower lookss ?? (still loving th...	0

資料處理(測試資料)

```
df_test = pd.read_csv("test_cleaned.csv")
```

```
s = []
for i in range(len(df_test)):
    s.append(len(df_test['cleaned'][i]))
```

```
print(max(s))
print(min(s))
print(sum(s)/len(s))
```

```
147
5
84.4134232301563
```

```
Max_Len = 84
#df_test = df_test[~(df_test.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
df_test = df_test.reset_index()
df_test = df_test.loc[:,['cleaned']]
df_test.columns = ['cleaned']
```

```
len(df_test)
```

```
3263
```

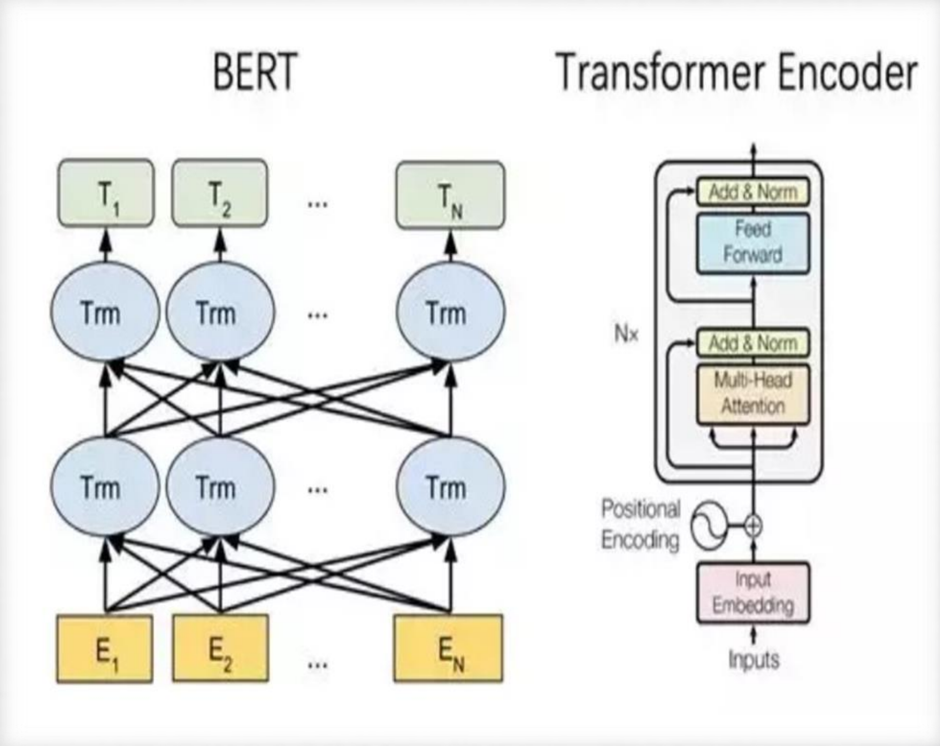
```
df_test.head()
```

	cleaned
0	just happened a terrible car crash
1	heard about earthquake is different cities, st...
2	there is a forest fire at spot pond, geese are...
3	apocalypse lighting spokane wildfires
4	typhoon soudelor kills 28 in china and taiwan

06 BERT 簡介

模型介紹：Bert

- **B**idirectional **E**ncoder **R**epresentations from **T**ransformer
- 架構：Transformer中的Encoder
- 傳統語言模型的變形



「如果有一個能直接處理各式 NLP 任務的通用架構該有多好？」

兩階段遷移學習！

- 預先訓練出對自然語言有一定理解的語言模型
- 用該模型做特徵擷取，或Fine-tune不同的下游監督式任務
- 訓練好預訓練模型，就可用遷移學習做Fine-tune，大大減低花費成本！

預訓練任務

- 1. 「**克漏字填空**」：學會處理每個詞在不同語境下該有的向量表示
  - Masked Language Model, MLM
  - 潮水「？」了，就知道誰沒穿褲子。
- 2. 「**下句預測**」：學會處理兩個句子之間的關係
  - Next Sentence Prediction, NSP
  - 醒醒吧 + 你沒有妹妹 → OK？
  - 用處：問答系統QA、自然語言推論NLI

兩階段遷移學習！

- 但……光是

  「預先訓練好對自然語言有一定理解的語言模型」

  這個步驟本身就非常燒錢阿！
  - BERT-BASE：$500
  - BERT-LARGE：$7000

- 好家在……

不論是Tensorflow，還是PyTorch

都已經有現成的BERT可用了！

總而言之，BERT 是一個強大的模型！！！

對下游的 NLP 任務很有幫助！

07 BERT 實作

匯入套件與檢視資料

```
import torch
from transformers import BertTokenizer
from IPython.display import clear_output
```

```
import pandas as pd
```

```
df_train = pd.read_csv("train_cleaned.csv")
```

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

	id	keyword	location	text	target	cleaned
0	1	NaN	NaN	Our Deeds are the Reason of this #earthquake M...	1	our deeds are the reason of this earthquake ma...
1	4	NaN	NaN	Forest fire near La Ronge Sask. Canada	1	forest fire near la ronge sask canada
2	5	NaN	NaN	All residents asked to 'shelter in place' are ...	1	all residents asked to 'shelter in place' are ...
3	6	NaN	NaN	13,000 people receive #wildfires evacuation or...	1	13,000 people receive wildfires evacuation ord...
4	7	NaN	NaN	Just got sent this photo from Ruby #Alaska as ...	1	just got sent this photo from ruby alaska as s...

資料處理(訓練資料)

```
Max_Len = 84
#df_train = df_train[~(df_train.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
Sample_Fraction = 0.5
df_train = df_train.sample(frac = Sample_Fraction, random_state = 9527)
```

```
df_train = df_train.reset_index()
df_train = df_train.loc[:,['cleaned', 'target']]
```

```
df_train.columns = ['cleaned', 'target']
```

```
len(df_train)
```

```
3806
```

```
df_train.head()
```

	cleaned	target
0	@ @ no one is rioting burning down buildings o...	1
1	stu put beetroot in his cake and even lost to ...	0
2	hot funtenna: hijacking computers to send dat...	1
3	1 of those days when ya don't realize till alr...	1
4	fresh out da shower lookss ?? (still loving th...	0

資料處理(測試資料)

```
df_test = pd.read_csv("test_cleaned.csv")
```

```
s = []
for i in range(len(df_test)):
    s.append(len(df_test['cleaned'][i]))
```

```
print(max(s))
print(min(s))
print(sum(s)/len(s))
```

```
147
5
84.4134232301563
```

```
Max_Len = 84
#df_test = df_test[~(df_test.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
df_test = df_test.reset_index()
df_test = df_test.loc[:,['cleaned']]
df_test.columns = ['cleaned']
```

```
len(df_test)
```

```
3263
```

```
df_test.head()
```

	cleaned
0	just happened a terrible car crash
1	heard about earthquake is different cities, st...
2	there is a forest fire at spot pond, geese are...
3	apocalypse lighting spokane wildfires
4	typhoon soudelor kills 28 in china and taiwan

實作一個與BERT相容的Dataset

```python
"""
此 Dataset 每次將 tsv 裡的一條text轉換成與BERT相容的格式，並回傳2個tensors：
- tokens_tensor：text經過tokenize後的索引序列，包含 [CLS] 與 [SEP]
- label_tensor：將分類標籤轉換成類別索引的tensor, 如果是測試集則回傳None
"""
class FakeNewsDataset(Dataset):
    # 把資料讀進來和初始化參數
    def __init__(self, mode, tokenizer):
        assert mode in ["train", "test"]
        self.mode = mode
        self.df = pd.read_csv(mode + ".tsv", sep = "\t").fillna("")
        self.len = len(self.df)
        self.tokenizer = tokenizer # 我們使用BERT tokenizer
        if mode == 'train':
            self.target = self.df.target

    @pysnooper.snoop() #用來監控logging訊息，可以觀看每個步驟喔！
```

```python
# 初始化一個專門讀取訓練樣本的Dataset，使用英文BERT斷詞
trainset = FakeNewsDataset("train", tokenizer = tokenizer)
```

```python
# 定義回傳一筆訓練/測試數據的函式
# 用來生成tokens(詞粒)的張量和label(標籤)的張量
def __getitem__(self, idx):
    if self.mode == "test":
        text = self.df.iloc[idx, :].values
        label_tensor = None
    else:
        text, target = self.df.iloc[idx, :].values
        label_tensor = torch.tensor(self.df.iloc[idx, -1])

    # 建立第一個句子的BERT tokens並加入分隔符號[SEP]
    # "CLS" token
    word_pieces = ["[CLS]"]
    # tokenize過程
    tokens = self.tokenizer.tokenize(text)
    # 加上分隔用的"SEP" token
    word_pieces += tokens + ["[SEP]"]
    length = len(word_pieces)

    # 將整個 token 序列轉換成索引序列
    ids = self.tokenizer.convert_tokens_to_ids(word_pieces)
    tokens_tensor = torch.tensor(ids)

    return (tokens_tensor, label_tensor)

def __len__(self):
    return self.len
```

BERT的tokenizer

```
In [22]: PRETRAINED_MODEL_NAME = "bert-base-cased" #使用bert-base，有區分大小寫的英文BERT模型
         tokenizer = BertTokenizer.from_pretrained(PRETRAINED_MODEL_NAME) #BERT斷詞用的tokenizer
         clear_output()
```

```
內文: #NoChillLukeHammings
IM SCREAMING
分類: 1

tokens_tensor: tensor([  101,   108,  1302,  1658,  6690,  2162, 16140,  3048,  2312,  5031,
         1116,   146,  2107,  9314, 16941, 10964, 15740,   102])

label_tensor: 1

還原tokens_tensor
[CLS]#No##C##hill##L##uke##H##am##ming##sI##MSC##RE##AM##ING[SEP]
```

一次傳一小batch的dataloader

```python
"""
這個函式的輸入"samples"是一個list，裡頭每個element都是
剛剛定義的"FakeNewsDataset"回傳的一個樣本
每個樣本都包含2個tensors：
- tokens_tensor
- label_tensor
它會對tokens_tensor作zero padding，並產生前面說明過的masks_tensors
"""
def create_mini_batch(samples):
    tokens_tensors = [s[0] for s in samples]

    # 測試集有label
    if samples[0][1] is not None:
        label_ids = torch.stack([s[1] for s in samples])
    else:
        label_ids = None

    # zero pad到同一序列長度
    tokens_tensors = pad_sequence(tokens_tensors, batch_first = True)

    # Attention Masks(注意力遮罩)
    # 將tokens_tensors裡頭不為zero padding的位置設為1
    # 讓BERT只關注這些位置
    masks_tensors = torch.zeros(tokens_tensors.shape, dtype = torch.long)
    masks_tensors = masks_tensors.masked_fill(tokens_tensors != 0, 1)

    return tokens_tensors, masks_tensors, label_ids
```

```python
# 初始化一個每次回傳33個訓練樣本的DataLoader
# 利用"collate_fn"將樣本的list合併成一個mini-batch
BATCH_SIZE = 33
trainloader = DataLoader(trainset, batch_size = BATCH_SIZE, collate_fn = create_mini_batch)
```

拿出一個batch看看

```
tokens_tensors.shape = torch.Size([33, 30])
tensor([[  101,   137,   137,  1185,  1141,  1110, 14807,  1158,  4968,  1205,
          2275,  1137, 25338, 13460,   102,     0,     0,     0,     0,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
        [  101,   188,  7926,  1508, 17775,  8005,  3329,  1107,  1117, 10851,
          1105,  1256,  1575,  1106,   170,  9052,  1116, 18498,   102,     0,
             0,     0,     0,     0,     0,     0,     0,     0,     0,     0],
        [  101,  2633,  4106,  5208,  1605,   131, 20844, 19617,  1158,  7565,
          1106,  3952,  2233,  1112,  1839,  5685,   164,  1602,  6131,  1410,
           166,  7001,   102,     0,     0,     0,     0,     0,     0,     0],
```

```
masks_tensors.shape = torch.Size([33, 30])
tensor([[1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0,
         0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0,
         0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0,
         0, 0, 0, 0, 0, 0],
        [1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
         0, 0, 0, 0, 0, 0],
```

```
label_ids.shape = torch.Size([33])
tensor([1, 0, 0, 1, 0, 1, 1, 0, 0, 1, 0, 1, 0, 0, 0, 0, 0, 0, 1, 1, 0, 0, 0, 0,
        0, 1, 1, 0, 1, 1, 1, 1, 1])
```

訂定測試準確率的函式

```python
# 寫一個簡單函式測試現在model在訓練集上的分類準確率
# 針對特定 DataLoader的
def get_predictions(model, dataloader, compute_acc = False):
    predictions = None
    correct = 0
    total = 0

    with torch.no_grad():
        # 遍巡整個資料集
        for data in dataloader:
            # 將所有tensors移到GPU上(如果有)
            if next(model.parameters()).is_cuda:
                data = [t.to("cuda:0") for t in data if t is not None]

            tokens_tensors, masks_tensors = data[:2]
            outputs = model(input_ids = tokens_tensors,
                            attention_mask = masks_tensors)

            logits = outputs[0]
            _, pred = torch.max(logits.data, 1)

            # 用來計算訓練集的分類準確率
            if compute_acc:
                labels = data[2]
                total += labels.size(0)
                correct += (pred == labels).sum().item()

             # 將當前batch的結果記錄下來
            if predictions is None:
                predictions = pred
            else:
                predictions = torch.cat((predictions, pred))

    if compute_acc:
        acc = correct / total
        return predictions, acc
    return predictions
```

參數數量

我們加上去的線性分類器
如滄海一粟般渺小

```
def get_learnable_params(module):
    return [p for p in module.parameters() if p.requires_grad]

model_params = get_learnable_params(model)
clf_params = get_learnable_params(model.classifier)

print(f"""
整個分類模型的參數量：{sum(p.numel() for p in model_params)}
線性分類器的參數量：{sum(p.numel() for p in clf_params)}
""")
```

```
整個分類模型的參數量：108311810
線性分類器的參數量：1538
```

訓練

```
%%time

model.train()

optimizer = torch.optim.Adam(model.parameters(), lr = 1e-5)

EPOCHS = 7
for epoch in range(EPOCHS):

    running_loss = 0.0

    for data in trainloader:

        tokens_tensors, masks_tensors, labels = [t.to(device) for t in data]

        optimizer.zero_grad()

        outputs = model(input_ids = tokens_tensors,
                        attention_mask = masks_tensors,
                        labels = labels)

        loss = outputs[0]

        loss.backward()
        optimizer.step()

        running_loss += loss.item()
    _, acc = get_predictions(model, trainloader, compute_acc = True)

    print("[epoch %d] loss: %.3f, acc: %.3f" % (epoch + 1, running_loss, acc))
```

```
[epoch 1] loss: 60.887, acc: 0.825
```

```
[epoch 2] loss: 45.499, acc: 0.880
```

```
[epoch 3] loss: 34.858, acc: 0.924
```

```
[epoch 4] loss: 27.710, acc: 0.942
```

```
[epoch 7] loss: 18.212, acc: 0.964
Wall time: 7min 40s
```

測試、部分預測結果

```
%%time
testset = FakeNewsDataset("test", tokenizer = tokenizer)
testloader = DataLoader(testset, batch_size = BATCH_SIZE, collate_fn = create_mini_batch)

predictions = get_predictions(model, testloader)

df = pd.DataFrame({"predicted": predictions.tolist()})
df_pred = pd.concat([testset.df, df.loc[:, 'predicted']], axis = 1)
df_pred.head()
```

```
13:36:27.529406 line        28         tokens_tensor = torch.tensor(ids)
New var:....... tokens_tensor = tensor([  101,   164,   112,  1331, 10008,  7867... 6063,
102])
13:36:27.529406 line        30         return (tokens_tensor, label_tensor)
13:36:27.530403 return      30         return (tokens_tensor, label_tensor)
Return value:.. (tensor([  101,   164,   112,  1331, 10008,  786... 2772,  1306,   112,
Elapsed time: 00:00:00.003967
```

```
Wall time: 14.9 s
```

	cleaned	predicted
0	just happened a terrible car crash	1
1	heard about earthquake is different cities, st...	1
2	there is a forest fire at spot pond, geese are...	1
3	apocalypse lighting spokane wildfires	1
4	typhoon soudelor kills 28 in china and taiwan	1

08 XLNet 簡介

自回歸(AR)語言模型

- 自回歸(Auto Regressive, AR)：利用前向(上文)或後向(下文)的情景信息來預測下一個詞，是單方向的。
- 缺點：只能利用其中一個方向來判斷，然而預測一個詞通常是需要上下文一起判斷的。
- GPT、ELMO皆屬之
  - 雖然ELMO同時包含了前後兩個方向，但這兩方向是獨立計算的。因此仍然無法很好地判斷上下文。

自編碼(AE)語言模型

- Auto Encoder(AE)
- BERT的Masked Language Model (MLM)
便是基於「去噪自編碼器(DAE)」
  - 是傳統AE的其中一種改良版。
  - 透過在輸入層加入隨機噪聲，來緩解過擬合現象。
- 優點：可以做到同時看上下文
- 缺點：
    - 1. 克漏字的預訓練階段，使用了[MASK]遮罩(噪聲)，但是真正在Fine-tune的時候沒有這種東西。會造成資訊不對稱(Input Noise)。
    - 2. BERT假設被預測的[MASK] token獨立於未屏蔽的其他token，然而真正的自然語言中並非如此。

XLNet！

- 基於自回歸(AR)，採用了新方法實現雙向編碼。
  - 因為是基於AR方法，所以沒有BERT的痛點！
- XLNet的創新點：

  1. 置換語言建模(Permutation Language Modeling, PLM)
  - 實現了傳統AR模型做不到的雙向學習，可以捕捉上下文了！

  2. 雙流自注意機制(Two-Stream Self-Attention)
  - 包含了Query stream(只在預訓練階段用到)及Content stream
  - 分別取代了[MASK]遮罩的兩種功能：告訴模型預測單詞位置及前後文關係。

  3. 借鑑Transformer-XL，實現大型文本的學習！

作法、差異簡介

與BERT相比~

- 預訓練階段：還是用Transformer：
  Seq2Seq模型 + 自注意力(self-attention)機制

- 採取了一些改進方法，例如置換語言建模(PLM)等等。

09 XLNet 實作

匯入套件與資料

```
import numpy as np
import pandas as pd
import tensorflow as tf
import tensorflow.keras as K
import seaborn as sns
import transformers
import nltk
import re
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, roc_auc_score, roc_curve

plt.style.use('seaborn')
```

```
train = pd.read_csv('train.csv')
test = pd.read_csv('test.csv')
```

```
print(tf.__version__)
```

```
2.2.0
```

資料處理(訓練資料)

```
Max_Len = 84
#df_train = df_train[~(df_train.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
Sample_Fraction = 0.5
df_train = df_train.sample(frac = Sample_Fraction, random_state = 9527)
```

```
df_train = df_train.reset_index()
df_train = df_train.loc[:,['cleaned', 'target']]
```

```
df_train.columns = ['cleaned', 'target']
```

```
len(df_train)
```

```
3806
```

```
df_train.head()
```

	cleaned	target
0	@ @ no one is rioting burning down buildings o...	1
1	stu put beetroot in his cake and even lost to ...	0
2	hot funtenna: hijacking computers to send dat...	1
3	1 of those days when ya don't realize till alr...	1
4	fresh out da shower lookss ?? (still loving th...	0

資料處理(測試資料)

```
df_test = pd.read_csv("test_cleaned.csv")
```

```
s = []
for i in range(len(df_test)):
    s.append(len(df_test['cleaned'][i]))
```

```
print(max(s))
print(min(s))
print(sum(s)/len(s))
```

```
147
5
84.4134232301563
```

```
Max_Len = 84
#df_test = df_test[~(df_test.text.apply(lambda x : len(x)) > Max_Len)]
df_train['cleaned'] = df_train['cleaned'].apply(lambda x: x[:Max_Len] if len(x) > Max_Len else x)
```

```
df_test = df_test.reset_index()
df_test = df_test.loc[:,['cleaned']]
df_test.columns = ['cleaned']
```

```
len(df_test)
```

```
3263
```

```
df_test.head()
```

	cleaned
0	just happened a terrible car crash
1	heard about earthquake is different cities, st...
2	there is a forest fire at spot pond, geese are...
3	apocalypse lighting spokane wildfires
4	typhoon soudelor kills 28 in china and taiwan

XLNET的tokenizer

```
from transformers import TFXLNetModel, XLNetTokenizer
```

```
# This is the identifier of the model,
# which is required for downloading the weights
# and initialize the architecture
model = 'xlnet-large-cased'
tokenizer = XLNetTokenizer.from_pretrained(model)
```

建造模型

```python
def modeling(name):
    """
    創造模型，是由XLNet主區塊加上一個分類的頭組成
    """
    # 定義輸入，為120維的token之id，也就是我們取120字
    inputs = K.Input(shape = (120, ), name = 'inputs', dtype = 'int32')

    # 匯入XLNet模型
    xlnet = TFXLNetModel.from_pretrained(name)
    # xlnet.trainable = False，凍結效果並不好
    xlnet_encodings = xlnet(inputs)[0]
    print(xlnet_encodings)

    # 分類頭(Classification head)
    # CLS部分(就是所謂Collect last step from Last hidden State)
    doc_encodings = tf.squeeze(xlnet_encodings[:, -1:, :], axis = 1)
    # 採用dropout以利於正則化(regularization)，防止過擬合(overfitting)
    doc_encodings = K.layers.Dropout(0.1)(doc_encodings)
    # 定義最終的輸出層
    outputs = K.layers.Dense(1, activation = 'sigmoid', name = 'outputs')(doc_encodings)

    #組裝模型
    model = K.Model(inputs = [inputs], outputs = [outputs])

    model.compile(optimizer = K.optimizers.Adam(lr = 2e-5), loss = 'binary_crossentropy',
                  metrics = ['accuracy', K.metrics.Precision(), K.metrics.Recall()])
    # Precision是精確率、PPV = TP / (TP + FP)
    # Recall是召回率、靈敏度 = TP / (TP + FN)

    return model
```

```python
xlnet = modeling(model)
```

模型摘要

```
xlnet.summary()
```

```
Model: "model"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
inputs (InputLayer)          [(None, 120)]             0
_________________________________________________________________
tfxl_net_model (TFXLNetModel ((None, 120, 1024),)      360268800
_________________________________________________________________
tf_op_layer_strided_slice (T [(None, 1, 1024)]         0
_________________________________________________________________
tf_op_layer_Squeeze (TensorF [(None, 1024)]            0
_________________________________________________________________
dropout_73 (Dropout)         (None, 1024)              0
_________________________________________________________________
outputs (Dense)              (None, 1)                 1025
=================================================================
Total params: 360,269,825
Trainable params: 360,269,825
Non-trainable params: 0
```

分割資料集、產生tensor

```python
# 針對訓練集，再將其分割一部分(此設15%)，作為測試集
# 可以想像是做模擬考，而真正的test.csv則是正式的大考
x_train, x_test, y_train, y_test = train_test_split(train['cleaned'], train['target'], test_size = 0.15, random_state = 150)
```

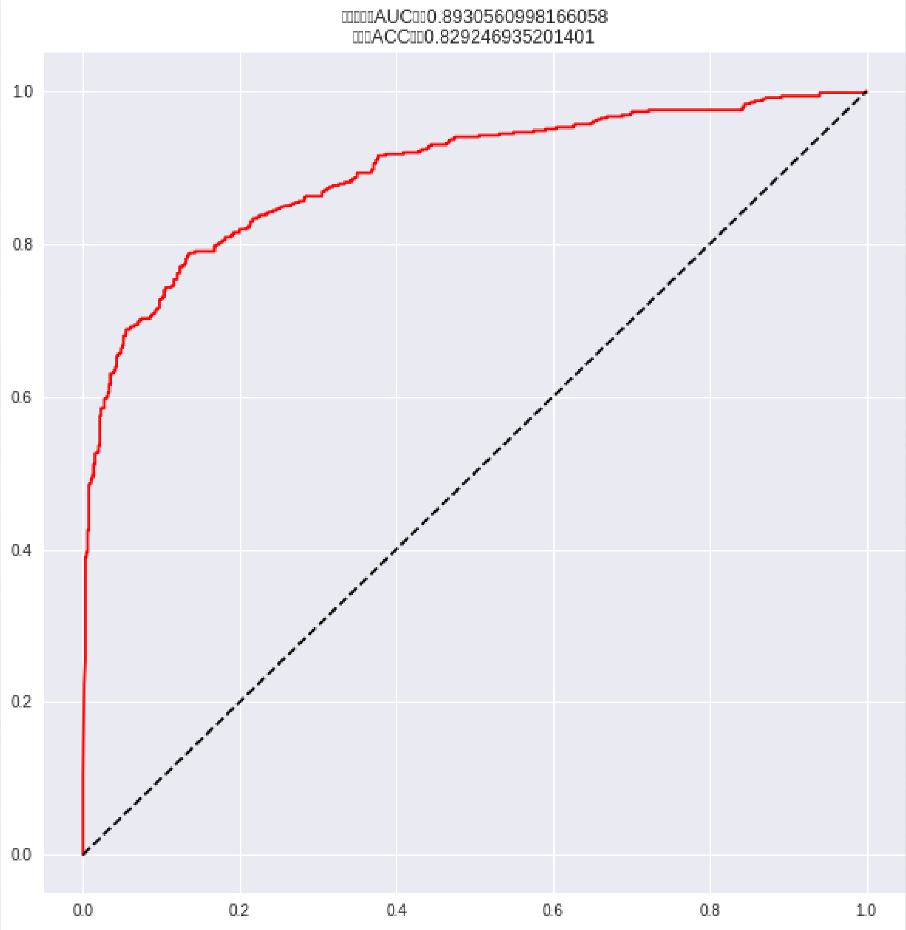
```python
# 使用給定的tokenizer來從文字內容中，生成由token之id組成的120維tensor
def get_inputs(texts, tokenizer, max_len = 120):
    inputs = [tokenizer.encode_plus(t, max_length = max_len, pad_to_max_length = True, add_special_token = True) for t in texts]
    inp_tokens = np.array([i['input_ids'] for i in inputs])
    indexes = np.array([i['attention_mask'] for i in inputs])
    segments = np.array([i['token_type_ids'] for i in inputs])
    return inp_tokens, indexes, segments
```

```python
# 生成輸入的張量(tensor)
inp_tokens, indexes, segments = get_inputs(x_train, tokenizer)
```

ROC圖與AUC

```python
# 以下為畫ROC圖的函數，使用準確率與AUC畫
def plot_ROC(prediction, label):
    acc = accuracy_score(label, np.array(prediction.flatten() >= 0.5, dtype = 'int'))
    FP, TP, thresholds = roc_curve(label, prediction)
    # 看ROC圖曲線下面積，介於0~1之間
    # 值越高表示準確率越高，0.5以下表示模型沒用
    auc = roc_auc_score(label, prediction)
    #以下畫圖
    figure, ax = plt.subplots(1, figsize = (10, 10))
    ax.plot(FP, TP, color = 'red')
    ax.plot([0,1], [0,1], color = 'black', linestyle = '--')
    ax.set_title(f"曲線下面積AUC為：{auc}\n準確率ACC為：{acc}")
    return figure
```

```python
ax = plot_ROC(prediction, y_test)
```



定義回調函數(Callback fn.)

3個回調函數，用逗號隔開

```
callbacks = [
    # 監控測試集的準確率，若該準確率連續三回合沒提升，就停止訓練。過程中只保存在測試集中表現最好的模型(權重)
    K.callbacks.EarlyStopping(monitor = 'val_accuracy', patience = 3, min_delta = 0.02, restore_best_weights = True),
    # 學習率動態調整，使用剛剛的自訂函數slowly調整，不印出該動作訊息
    K.callbacks.LearningRateScheduler(slowly, verbose = 0),
    # 如果連續2個epoch模型性能沒提升，就嘗試減少學習率lr，每次減少lr*1e-6，不印出該動作訊息
    # 學習率減少後，不進行cooldown,。學習率下限為0
    K.callbacks.ReduceLROnPlateau(monitor = 'val_accuracy', factor = 1e-6, patience = 2,
                                  verbose = 0, mode = 'auto', min_delta = 0.001, cooldown = 0, min_lr = 1e-6)
]
```

開始訓練！

```
history = xlnet.fit(x = inp_tokens, y = y_train, epochs = 5, batch_size = 5, validation_split = 0.15, callbacks = callbacks)
```

```
1100/1100 [==============================] - 1345s 1s/step - loss: 0.8045 - accuracy: 0.7035 - precision: 0.6662 - recall: 0.6243
Epoch 2/5
1100/1100 [==============================] - 1338s 1s/step - loss: 0.5048 - accuracy: 0.7945 - precision: 0.7875 - recall: 0.7163
Epoch 3/5
1100/1100 [==============================] - 1335s 1s/step - loss: 0.4422 - accuracy: 0.8282 - precision: 0.8321 - recall: 0.7531
Epoch 4/5
1100/1100 [==============================] - 1335s 1s/step - loss: 0.3605 - accuracy: 0.8627 - precision: 0.8722 - recall: 0.7982
Epoch 5/5
1100/1100 [==============================] - 1337s 1s/step - loss: 0.2720 - accuracy: 0.9045 - precision: 0.9138 - recall: 0.8594
```

```
val_loss: 0.7007 - val_accuracy: 0.7837 - val_precision: 0.9437 - val_recall: 0.5253 - lr: 2.1000e-05

val_loss: 0.5555 - val_accuracy: 0.8239 - val_precision: 0.8211 - val_recall: 0.7518 - lr: 2.2000e-05

val_loss: 0.5322 - val_accuracy: 0.7992 - val_precision: 0.8767 - val_recall: 0.6169 - lr: 2.3000e-05

val_loss: 0.4581 - val_accuracy: 0.8229 - val_precision: 0.8240 - val_recall: 0.7446 - lr: 2.4000e-05

val_loss: 0.8110 - val_accuracy: 0.7971 - val_precision: 0.7781 - val_recall: 0.7349 - lr: 2.0000e-05
```

問題1：梯度消失！

```
Epoch 1/5
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0'
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0'
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0'
WARNING:tensorflow:Gradients do not exist for variables ['tfxl_net_model/transformer/mask_emb:0'
```

深層網路的訓練常常面臨
梯度消失或梯度爆炸的阻礙，
尤其是像這樣的大型網路

雖然有跳出警告，但仍能跑出結果

問題2：資源耗盡！

ResourceExhaustedError (see above for traceback):
OOM when allocating tensor with shape[2304,384]

記憶體被撐爆了！

解決方法：降低batch_size(20→5)
雖然慢了點，但至少可以動了

BERT vs XLNet

(文獻來源)
Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Ruslan Salakhutdinov and Quoc V. Le., "XLNet: Generalized Autoregressive Pretraining for Language Understanding"(2019)

- 文本分類任務

Model	IMDB	Yelp-2	Yelp-5	DBpedia	AG	Amazon-2	Amazon-5
CNN [15]	-	2.90	32.39	0.84	6.57	3.79	36.24
DPCNN [15]	-	2.64	30.58	0.88	6.87	3.32	34.81
Mixed VAT [31, 23]	4.32	-	-	0.70	4.95	-	-
ULMFiT [14]	4.6	2.16	29.98	0.80	5.01	-	-
BERT [35]	4.51	1.89	29.32	0.64	-	2.63	34.17
XLNet	**3.20**	**1.37**	**27.05**	**0.60**	**4.45**	**2.11**	**31.67**

Table 4: Comparison with state-of-the-art error rates on the test sets of several text classification datasets. All BERT and XLNet results are obtained with a 24-layer architecture with similar model sizes (aka BERT-Large).

- 閱讀理解任務: XLNet表現較好

SQuAD2.0	EM	F1	SQuAD1.1	EM	F1
Dev set results (single model)					
BERT [10]	78.98	81.77	BERT† [10]	84.1	90.9
RoBERTa [21]	86.5	89.4	RoBERTa [21]	88.9	94.6
XLNet	**87.9**	**90.6**	XLNet	**89.7**	**95.1**

模型比較

AutoRegressive(AR)自迴歸:下一個字的出現依賴於上文
AutoEncoding(AE)自編碼: <Mask> 替代與還原，充分利用上下文資訊

Model	BERT	XLNet
模型特點	AE 性質捕捉bidirectional context 處理各式 NLP 任務的通用架構: Pretrain + Fine tune	也是通用架構 結合 AE 和 AR ，發明出 PLM 訓練的資料庫要比BERT大得多
優點	考慮上下文資訊	以 PLM 解決BERT資訊不對稱問題
缺點	<Mask> 相互獨立 預訓練-微調差異 → 資訊不對稱	訓練速度慢
參數數量 運算成本	一億零八百萬個參數 108 s/epoch	三億多個參數 1345s/epoch
Kaggle Score (準確率)	**0.83231 (50% Sample)**	0.81492

模型表現分數

訓練內容

1. BERT: 100% Sample,
   10% Sample, 50% Sample
2. XLNet: 一般, 凍結(Frozen)

目前排名

最佳排名: 297 / 1811

No 1: BERT 50% Sample

297 NLP_nccu 0.83231 5 7d

Your Best Entry ↑

Your submission scored 0.80572, which is not an improvement of your best score. Keep trying!

df_pred_0511.csv 0.83231

12 days ago by JudyLee

df_pred_智超0511 (BERT sample 50%訓練)

No 2: XLNet

submission_XLNet0530_colabbs5.csv 0.81492

3 days ago by JudyLee

submission_XLNet0530_colab不凍結bs=5

10 使用者介面

Line Bot 原理

1. 用戶發送訊息至LINE 官方帳號
2. LINE Platform將一個webhook事件轉送至bot server的webhook URL，也就是deploy於Heroku平台的server端
3. Bot server將依據webhook event，透過LINE Platform 回應用戶

Line Bot 優點

為什麼要使用Line Bot？

- 易於使用，可以自動回復，台灣2018年數據統計，達2100萬活躍用戶，高達九成使用率
- 不需要安裝其他應用程式
- 即時服務
- 可專注於後端處理
- 可處理群體回復
- 開發、維護成本低

Line Bot Demo

輸入範例參考 :

（目前僅提供英文災難貼文的辨識服務）

災難性貼文

- Damage to school bus on 80 in multi car crash #BREAKING

非災難性貼文

- It’s scary to see my mom
  when I was playing videos games loooool

11 完整程式碼

Click Me! ↓

https://drive.google.com/drive/folders/10rcM9h9QnMJoL04jj8ty0MSAt0b-_1Xz?usp=sharing

Thanks For Your Time