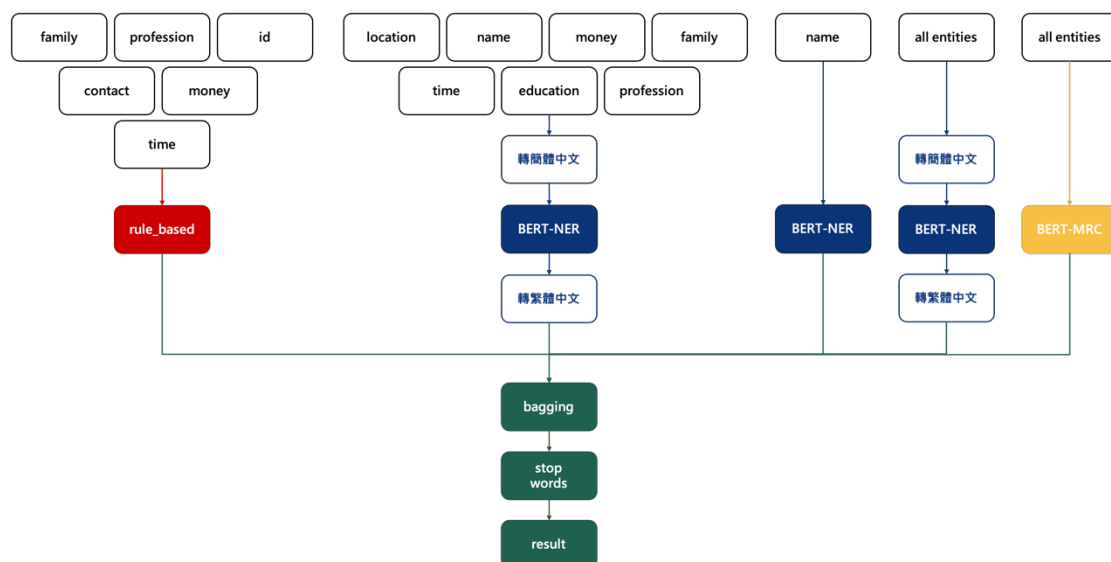


# 醫病訊息決策與對話語料分析競賽 - 秋季賽：醫病資料去識別化

隊名：HHH-LKT

## 一、系統架構



圖一、系統架構圖

### (一) Rule-based

將較明確的特徵利用 Rule-based 的方式辨認，若有規律的組合會利用 Regular Expression 的方法辨認，例如身份證為一個英文字母配上九個數字，錢為數字後加上「元」或「塊」等。若較不規律的組合則是會用比對字串的方式，再給予該特徵相對應的標記，例如爸爸、媽媽標記為家人，而老師則標記為職業別等。

### (二) Bert NER

將各個類別分別利用 bert-base-chinese 訓練，也就是一次只訓練一個類別。由於上傳的結果顯示簡體中文的預測表現比繁體中文的還要好，因此，我們先將所有資料轉成簡體中文，得到結果後，再轉為繁體中文。提及名字類別，因為該繁體中文特徵表現較佳，所以，將其納入 Bagging 特徵之一。最後，我們也將所有類別一起訓練，將其結果納入 Bagging 特徵。

### (三) Bert MRC

除了上述方法之外，本系統也嘗試利用閱讀理解方法(MRC)來達成此任務。首先，對一句話，建構數個不同的類別，接著，將每個類別與句子進行拼接，最後，得到多個不同 Bert 的輸入數據。其預測起始位置、結束位置以及哪個類型的機率，將最高的機率類型及位置之特徵輸出為結果。

#### (四) Bagging

本系統將上述所有結果，合併於一個檔案，其程序包含設定合併次序及清除相同特徵。首先，依照各個檔案的表現，由高到低設定合併的次序，較後面才合併的資料，會與前者比對，若有相同或重疊的特徵，將會移除。以此方法，將所有結果 Bagging 成總結果。

#### (五) Stop words

分析每份結果之詞頻，將不合理於某類別之詞類，進行字串比對後清除，輸出最終的預測結果。

## 二、程式碼

### (一) Rule based

功能	程式碼	註記
抓取特徵	<pre> regex = r'A\d{9,10} B\d{9,10} C\d{9,10} D\d{9,10} E\d{9,10} F\d{9,10} G\d{9,10}  H\d{9,10} I\d{9,10} J\d{9,10} K\d{9,10} L\d{9,10} M\d{9,10} N\d{9,10} O\ d{9,10} P\d{9,10} Q\d{9,10} R\d{9,10} S\d{9,10} T\d{9,10} U\d{9,10} V\d{ 9,10} W\d{9,10} X\d{9,10} Y\d{9,10} Z\d{9,10}' regex_phone = r'09\d{8} line LINE Line L I N E  EMAIL MAIL email mail FB 臉書  F B  Grindr Hornet Tinder tinder wootalk' regex_time = r'去年\d+月\d+號 去年\d+月 去年\d+號 去年 明年 正午 清晨  明天 昨天 今天 一個月 兩個月 三個月 一個禮拜 一個星期 兩個禮拜 兩個 星期 三個禮拜 三個星期 早上 中午 下午 晚上 一天 兩天 三天 四天 五天 六 天 \d+天 \d+月\d+號 \d+月\d+日 \d+個月 \d+年 聖誕節 清明節 情人節 感恩 節 端午節 [一二三四五六七八九十]月[一二三四五六七八九][日號]' regex_family = r'我姊姊 我姐姐 我妹妹 我老婆 我太太 我先生 我丈夫 我老 公 我爺爺 你姊姊 你姐姐 你妹妹 你老婆 你太太 你先生 你丈夫 你老公 你 奶奶 你爸爸 你叔叔 你嬸嬸 你舅舅 你阿姨 你姑姑 你大姑 你二姑 你外婆  你婆婆 你姊 你姐 你妹 你爺 你奶 你爸 你叔 你舅 你嬸 你姨 你姑 你婆 你 大女兒 你二女兒 你三女兒 你小女兒 你小女 你大兒子 你二兒子 你三兒子  你小兒子 你兒子 你大子 我爺爺 我奶奶 我爸爸 我叔叔 我嬸嬸 我舅舅 我 阿姨 我姑姑 我大姑 我二姑 我外婆 我婆婆 我爺 我奶 我爸 你姊 你姐 你妹  我叔 我舅 我嬸 我姨 我姑 我婆 我大女兒 我二女兒 我三女兒 我小女兒 我 小女 我女兒 我大兒子 我二兒子 我三兒子 我小兒子 我兒子 我大子 爺爺  奶奶 爸爸 爸 叔叔 阿姨 嬸嬸 嬸 姑姑 大姑 二姑 姑 外婆 姊姊 姐姐 妹妹 老 婆 太太 先生 丈夫 老公 婆 老大 老二 老三 大女兒 二女兒 三女兒 小女兒  小女 女兒 大兒子 二兒子 三兒子 小兒子 兒子 犬子' regex_profession = r'Google' regex_money = r'^\d+塊 \d+元'  df_for_test['ID']=df_for_test['words'].apply(lambda x:re.findall(regex, x)) df_for_test['contact']=df_for_test['words'].apply(lambda x:re.findall(regex_phone, x)) df_for_test['time']=df_for_test['words'].apply(lambda x:re.findall(regex_time, x)) df_for_test['profession']=df_for_test['words'].apply(lambda x:re.findall(regex_profession, x)) df_for_test['money']=df_for_test['words'].apply(lambda x:re.findall(regex_money, x)) df_for_test['family']=df_for_test['words'].apply(lambda x:re.findall(regex_family, x)) </pre>	利用 Regular Expression 的方式將身分證、電話、時間、家人、職業別、金錢方式進行標記。

## (二) Bert NER

## 1. 按照各類別做訓練

功能	程式碼	註記
資料 前處理	<pre>def split_conser_label(input_list):     # input word_list     convers = []     label = []     for i in input_list:         j = i.split('\n', 1)         if len(j[0]) &gt; 0:             convers.append(j[0])         if len(j[1].split('\n\n')[0]) &gt; 0:             label.append(j[1].split('\n\n')[0])     return convers, label  def label_to_dict(input_label_list):     # input label     dict_label = {}     for i in range(len(input_label_list)):         total_label = input_label_list[i].split('\n')         dict_label.update({i : dict()})         for j in range(len(total_label)-1):             each_row = total_label[j+1].split('\t')             duration = (int(each_row[2]) - int(each_row[1]))             for q in range(duration):                 # origin_label = each_row[4].lower()                 # chinese_label = my_map[origin_label]                 dict_label[i].update({int(each_row[1])+q : each_row[4]})     return dict_label  def fit_ner_form(input_con, input_label_dict):     #input convers dict_label     list_sentence_id = []     list_words = []     list_labels = []      for i in range(len(input_con)):         now = input_con[i]         for j in range(len(now)):             if j in input_label_dict[i].keys():                 list_sentence_id.append(i)                 list_words.append(now[j])                 list_labels.append(input_label_dict[i][j])             else:                 list_sentence_id.append(i)                 list_words.append(now[j])                 list_labels.append('O')     return list_sentence_id, list_words, list_labels</pre>	將對話及標記資料切割，建立類別索引，並將資料建構成 NER 的格式。
句子切割	<pre>def split_sentence(row):     global count_, word_len, list_warn     word_len+=1     if row.words in ['.', '?', '!', ',', ''] and word_len &gt; 20:         list_warn.append(word_len)         if word_len &gt; 128:             print(count_)             count_ +=1             word_len = 0             return (count_ -1)     else:         return count_</pre>	以標點符號分割句子，並設定句子長度 20 字以上。
轉簡體字	<pre>train_df['sentence_id'] = train_df.apply(split_sentence, 1) cc = OpenCC('t2s') train_df['words'] = train_df['words'].apply(lambda x: cc.convert(x))</pre>	利用 Opencc 將繁體中文轉為簡體中文。

## 建立模型

```
def train_and_predict(only_label, train_data):
    model = NERModel('bert', f'bert-base-chinese',
        labels=label_list, args={'train_batch_size':16,
        'overwrite_output_dir': True, 'output_dir':'output/ner/bert_sim/{only_label}',
        'reprocess_input_data': True, 'num_train_epochs': 15})
    model.train_model(train_data)
    g = open('test.txt', 'r')
    words = g.read()
    word_list = words.split('-----\n\n')
    def test_file_form(test_input):
        id_ = []
        string_ = []
        num = 0
        for i in test_input:
            if len(i) > 0:
                sequence = i.split("\n")[1]
                whole_split_sent = re.split('( ° | ? | ! | , )', sequence)
                for j in range(len(whole_split_sent)):
                    if whole_split_sent[j] not in [' ° ', '? ', '!', ',', ' ']:
                        if len(whole_split_sent[j]) > 0:
                            id_.append(i.split("\n")[0].split(' ')[1])
                            string_.append(' '.join(whole_split_sent[j]))
                            num += 1
            else:
                string_[num-1] += f '{whole_split_sent[j]}'

        return id_, string_
    id_, string_ = test_file_form(word_list)
    test_df = pd.DataFrame({'sentence_id':id_, 'words':string_})
    print(test_df.head())
    print(f'len test_df: {len(test_df)}')
    temp_id = []
    temp_sent = []
    trans_id = []
    trans_sent = []
    for i in range(len(test_df)):
        if len(temp_id) == 0 and len(temp_sent) == 0:
            if len(test_df.iloc[i]['words']) > 20:
                trans_id.append(test_df.iloc[i]['sentence_id'])
                trans_sent.append(test_df.iloc[i]['words'])
            else:
                temp_id.append(test_df.iloc[i]['sentence_id'])
                temp_sent.append(test_df.iloc[i]['words'])
        else:
            if temp_id[0] == test_df.iloc[i]['sentence_id']:
                now_sent = temp_sent[0] + ' ' + test_df.iloc[i]['words']
                if len(now_sent) > 20:
                    trans_id.append(test_df.iloc[i]['sentence_id'])
                    trans_sent.append(now_sent)
                    temp_id = []
                    temp_sent = []
                    now_sent = ""
                else:
                    temp_sent = [now_sent]
            else:
                trans_id.append(temp_id[0])
                trans_sent.append(temp_sent[0])
                temp_id = [test_df.iloc[i]['sentence_id']]
                temp_sent = [test_df.iloc[i]['words']]
    if len(temp_id) > 0 and len(temp_sent) > 0:
        trans_id.append(temp_id[0])
        trans_sent.append(temp_sent[0])
    test_df = pd.DataFrame({'sentence_id':trans_id, 'words':trans_sent})
    test_df['words'] = test_df['words'].apply(lambda x: cc.convert(x))
    print(test_df.head())
    print(f'len test_df: {len(test_df)}')
    predictions, raw_outputs = model.predict(test_df.words.values)
    test_df['predict'] = predictions
    test_df['raw_outputs'] = raw_outputs
    return test_df
```

利用 simple transformer 的 NER 訓練方式，以 bert-base-chinese 作為 pre-trained model，並產生預測資料。

<p><b>預測結果</b></p>	<pre> for i in whole_label:     train_data_each = train_df.copy()     train_data_each['labels'] = train_data_each['labels'].apply(trans_label, args=(i, 1)     if len(list(set(train_data_each['labels'].values))) &gt; 1:         df = train_and_predict(i, train_data_each)         print(df.head())         num = 0         sentence_id = []         words = []         predictions = []         for i in range(len(df)):             if i==0:                 sentence_id.append(df.iloc[i]['sentence_id'])                 words.append(df.iloc[i]['words'])                 predictions.append(eval(df.iloc[i]['predict']))             else:                 if df.iloc[i]['sentence_id'] == sentence_id[num]:                     words[num] += ' ' + df.iloc[i]['words']                     #print(words[num])                     #print(predictions[num])                     predictions[num].extend((eval(df.iloc[i]['predict'])))                 else:                     sentence_id.append(df.iloc[i]['sentence_id'])                     words.append(df.iloc[i]['words'])                     predictions.append(eval(df.iloc[i]['predict']))                     num+=1         df = pd.DataFrame({'sentence_id':sentence_id, 'words':words, 'predictions':predictions})         print(len(df))         art_id = []         order_ = []         word_ = []         label_ = []         for i in range(len(df)):             now = df.iloc[i]['predictions']             for j in range(len(now)):                 if 'O' not in now[j].values():                     art_id.append(i)                     order_.append(j)                     for k, v in now[j].items():                         word_.append(k)                         label_.append(v)                         #label_.append(v.split('-')[1])         print(len(pd.DataFrame({'art_id':art_id, 'order_':order_, 'word_':word_, 'label_':label_})))  total = len(art_id) i = 0 j = 0 article_id = [] start_position = [] end_position = [] entity_text = [] entity_type = [] while i&lt;total:     if i == 0:         article_id.append(art_id[i])         start_position.append(int(order_[i]))         end_position.append(int(order_[i])+1)         entity_text.append(word_[i])         entity_type.append(label_[i])         i+=1     else:         if article_id[j] == art_id[i] and end_position[j] == int(order_[i]) and entity_type[j] == label_[i]:             end_position[j] = int(order_[i])+1             entity_text[j] += word_[i]             i+=1 </pre>	<p>將訓練資料切割成各類別之特徵，進行預測後，轉換成比賽單位之格式，輸出檔案。</p>
--------------------	--	--

	<pre> else:     article_id.append(article_id[i])     start_position.append(int(order_[i]))     end_position.append(int(order_[i]+1))     entity_text.append(word_[i])     entity_type.append(label_[i])     i+=1     j+=1  df = pd.DataFrame({'article_id':article_id,                   'start_position':start_position, 'end_position':end_position,                   'entity_text':entity_text, 'entity_type':entity_type}) cc = OpenCC('s2t') df['entity_text'] = df['entity_text'].apply(lambda x: cc.convert(x)) df.to_csv('to_aidea/simplified_bert/bert_Simplified_{only_label}.tsv', sep = '\t', index=None) else:     pass </pre>	
--	--	--

## 2. 全部類別訓練

功能	程式碼	註記
資料 前處理	<pre> def split_conser_label(input_list):     # input word_list     convers = []     label = []     for i in input_list:         j = i.split("\n", 1)         if len(j[0]) &gt; 0:             convers.append(j[0])             if len(j[1].split("\n\n")[0]) &gt; 0:                 label.append(j[1].split("\n\n")[0])     return convers, label  def label_to_dict(input_label_list):     # input label     dict_label = {}     for i in range(len(input_label_list)):         total_label = input_label_list[i].split("\n")         dict_label.update({i : dict()})         for j in range(len(total_label)-1):             each_row = total_label[j+1].split("\t")             duration = (int(each_row[2]) - int(each_row[1]))             for q in range(duration):                 dict_label[i].update({int(each_row[1])+q : each_row[4]})     return dict_label  def fit_ner_form(input_con, input_label_dict):     list_sentence_id = []     list_words = []     list_labels = []     for i in range(len(input_con)):         now = input_con[i]         for j in range(len(now)):             if j in input_label_dict[i].keys():                 list_sentence_id.append(i)                 list_words.append(now[j])                 list_labels.append(input_label_dict[i][j])             else:                 list_sentence_id.append(i)                 list_words.append(now[j])                 list_labels.append('O')     return list_sentence_id, list_words, list_labels </pre>	將對話及標記資料切割，建立類別索引，並將資料建構成NER的格式。

句子切割	<pre>def split_sentence(row):     global count_, word_len, list_warn     word_len+=1     if row.words in ['。','?','!',' ',' ' ] and word_len &gt; 20:         list_warn.append(word_len)         if word_len &gt; 128:             print(count_)             count_ +=1             word_len = 0             return (count_ -1)         else:             return count_</pre>	以標點符號分割句子，並設定句子長度 20 字以上。
建立模型	<pre>model = NERModel('bert', 'bert-base-chinese',     labels=label_list, args={'train_batch_size':16, 'overwrite_output_dir':     True, 'output_dir':'output/ner/bert_Simplified', 'reprocess_input_data': True,     'num_train_epochs': 15})  model.train_model(train_df)</pre>	利用 simple transformer 的 NER 訓練方式，以 bert-base-chinese 作為 pre-trained model。
預測結果	<pre>def test_file_form(test_input):     # word_list     id_ = []     string_ = []     num = 0     for i in test_input:         #print(i)         if len(i) &gt; 0:             #print('test',i)             sequence = i.split('\n')[1]             #print(sequence)             whole_split_sent = re.split('(。 ? !   ,)', sequence)             for j in range(len(whole_split_sent)):                 if whole_split_sent[j] not in ['。','?','!',' ',' ']:                     if len(whole_split_sent[j])&gt;0:                         id_.append(i.split('\n')[0].split(' ')[1])                         string_.append(' '.join(whole_split_sent[j]))                         num+=1             else:                 string_[num-1] += f '{whole_split_sent[j]}'      return id_, string_  id_, string_ = test_file_form(word_list) test_df = pd.DataFrame({'sentence_id':id_, 'words':string_})  print(test_df.head()) print(f'len test_df: {len(test_df)}')  temp_id = [] temp_sent = [] trans_id = [] trans_sent = []  for i in range(len(test_df)):     if len(temp_id)==0 and len(temp_sent)==0:         if len(test_df.iloc[i]['words']) &gt; 20:             trans_id.append(test_df.iloc[i]['sentence_id'])             trans_sent.append(test_df.iloc[i]['words'])         else:             temp_id.append(test_df.iloc[i]['sentence_id'])             temp_sent.append(test_df.iloc[i]['words'])</pre>	將測試資料處理為與訓練資料相同格式後，預測出結果。

	<pre> else:     if temp_id[0] == test_df.iloc[i]['sentence_id']:         now_sent = temp_sent[0] + ' ' + test_df.iloc[i]['words']         if len(now_sent) &gt; 20:             trans_id.append(test_df.iloc[i]['sentence_id'])             trans_sent.append(now_sent)             temp_id = []             temp_sent = []             now_sent = ""         else:             temp_sent = [now_sent]     else:         trans_id.append(temp_id[0])         trans_sent.append(temp_sent[0])         temp_id = [test_df.iloc[i]['sentence_id']]         temp_sent = [test_df.iloc[i]['words']] if len(temp_id)&gt;0 and len(temp_sent)&gt;0:     trans_id.append(temp_id[0])     trans_sent.append(temp_sent[0]) test_df = pd.DataFrame({'sentence_id':trans_id, 'words':trans_sent}) test_df['words'] = test_df['words'].apply(lambda x: cc.convert(x)) print(test_df.head()) print(f'len test_df: {len(test_df)}') predictions, raw_outputs = model.predict(test_df.words.values) </pre>	
資料 後處理	<pre> while i&lt;total:     if i == 0:         article_id.append(art_id[i])         start_position.append(int(order_[i]))         end_position.append(int(order_[i])+1)         entity_text.append(word_[i])         entity_type.append(label_[i])         i+=1     else:         if article_id[j] == art_id[i] and end_position[j] == int(order_[i]) and entity_type[j] == label_[i]:             end_position[j] = int(order_[i])+1             entity_text[j] += word_[i]             i+=1         else:             article_id.append(art_id[i])             start_position.append(int(order_[i]))             end_position.append(int(order_[i])+1)             entity_text.append(word_[i])             entity_type.append(label_[i])             i+=1             j+=1 df = pd.DataFrame({'article_id':article_id, 'start_position':start_position, 'end_position':end_position, 'entity_text':entity_text, 'entity_type':entity_type}) cc = OpenCC('s2t') df['entity_text'] = df['entity_text'].apply(lambda x: cc.convert(x)) df.to_csv('to_aidea/simplified_bert/bert_Simplified_{only_label}.tsv', sep = '\t', index=None) </pre>	將結果轉換成與比賽 單位相符之格式。



### (三) Bert MRC

功能	程式碼	註記
建立模型	<pre> class BertLabeling(pl.LightningModule):     def __init__(         self,         args: argparse.Namespace):         super().__init__()         if isinstance(args, argparse.Namespace):             self.save_hyperparameters(args)             self.args = args         else:             TmpArgs = namedtuple("tmp_args",                                 field_names=list(args.keys()))             self.args = args = TmpArgs(**args)          self.bert_dir = args.bert_config_dir         self.data_dir = self.args.data_dir          bert_config =             BertQueryNerConfig.from_pretrained(args.bert_config_dir,             hidden_dropout_prob=args.bert_dropout,             attention_probs_dropout_prob=args.bert_dropout,             mrc_dropout=args.mrc_dropout)          self.model = BertQueryNER.from_pretrained(args.bert_config_dir,   config=bert_config)         logging.info(str(args.__dict__ if isinstance(args,             argparse.ArgumentParser) else args))          self.loss_type = args.loss_type         if self.loss_type == "bce":             self.bce_loss = BCEWithLogitsLoss(reduction="none")         else:             self.dice_loss = DiceLoss(with_logits=True,                                       smooth=args.dice_smooth)          weight_sum = args.weight_start + args.weight_end +         args.weight_span         self.weight_start = args.weight_start / weight_sum         self.weight_end = args.weight_end / weight_sum         self.weight_span = args.weight_span / weight_sum         self.flat_ner = args.flat         self.span_f1 = QuerySpanF1(flat=self.flat_ner)         self.chinese = args.chinese         self.optimizer = args.optimizer         self.span_loss_candidates = args.span_loss_candidates      @staticmethod     def add_model_specific_args(parent_parser):         parser = argparse.ArgumentParser(parents=[parent_parser],             add_help=False)         parser.add_argument("--mrc_dropout", type=float, default=0.1,             help="mrc dropout rate")         parser.add_argument("--bert_dropout", type=float, default=0.1,             help="bert dropout rate")         parser.add_argument("--weight_start", type=float, default=1.0)         parser.add_argument("--weight_end", type=float, default=1.0)         parser.add_argument("--weight_span", type=float, default=1.0)         parser.add_argument("--flat", action="store_true", help="is flat             ner")         parser.add_argument("--span_loss_candidates", choices=["all",             "pred_and_gold", "gold"],             default="all", help="Candidates used to compute span loss")         parser.add_argument("--chinese", action="store_true",             help="is chinese dataset")         parser.add_argument("--loss_type", choices=["bce", "dice"],             default="bce", help="loss type")         parser.add_argument("--optimizer", choices=["adamw", "sgd"],             default="adamw", help="loss type") </pre>	產生 MRC 模型，其包含資料讀取、建立 Loss function、參數設定等。

	<pre> parser.add_argument("--dice_smooth", type=float, default=1e-8,                     help="smooth value of dice loss") parser.add_argument("--final_div_factor", type=float, default=1e4,                     help="final div factor of linear decay                         scheduler")  return parser  def configure_optimizers(self):     no_decay = ["bias", "LayerNorm.weight"]     optimizer_grouped_parameters = [{"params": [p for n, p in self.model.named_parameters() if not any(nd in n for nd in no_decay)], "weight_decay": self.args.weight_decay,},     {"params": [p for n, p in self.model.named_parameters() if any(nd in n for nd in no_decay)], "weight_decay": 0.0, },]     if self.optimizer == "adamw":         optimizer = AdamW(optimizer_grouped_parameters,                            betas=(0.9, 0.98), # according to RoBERTa paper                            lr=self.args.lr, eps=self.args.adam_epsilon,)     else:         optimizer = SGD(optimizer_grouped_parameters, lr=self.args.lr, momentum=0.9)     num_gpus = len([x for x in str(self.args.gpus).split(",") if x.strip()])     t_total = (len(self.train_dataloader())// (self.args.accumulate_grad_batches * num_gpus) + 1) * self.args.max_epochs     scheduler=     torch.optim.lr_scheduler.ReduceLROnPlateau(optimizer, factor=0.8, mode='min', patience=1000, min_lr=1e-6)     return [optimizer], [{"scheduler": scheduler, "interval": "step"}]  def forward(self, input_ids, attention_mask, token_type_ids):     return self.model(input_ids, attention_mask=attention_mask, token_type_ids=token_type_ids)  def compute_loss(self, start_logits, end_logits, span_logits,                  start_labels, end_labels, match_labels,                  start_label_mask, end_label_mask):     batch_size, seq_len = start_logits.size()      start_float_label_mask = start_label_mask.view(-1).float()     end_float_label_mask = end_label_mask.view(-1).float()     match_label_row_mask = start_label_mask.bool().         unsqueeze(-1).expand(-1, -1, seq_len)     match_label_col_mask = end_label_mask.bool().         unsqueeze(-2).expand(-1, seq_len, -1)     match_label_mask = match_label_row_mask &amp;         match_label_col_mask     match_label_mask = torch.triu(match_label_mask, 0)     if self.span_loss_candidates == "all":         float_match_label_mask = match_label_mask.view(batch_size,         -1).float()      else:         start_preds = start_logits &gt; 0         end_preds = end_logits &gt; 0         if self.span_loss_candidates == "gold":             match_candidates = ((start_labels.unsqueeze(-1).expand (-1, -1, seq_len) &gt; 0) &amp; (end_labels.unsqueeze (-2).expand(-1, seq_len, -1) &gt; 0))         else:             match_candidates =             torch.logical_or((start_preds.unsqueeze(-1).expand(-1, -1, seq_len) &amp; end_preds.unsqueeze(-2).expand(-1, seq_len, - 1)), (start_labels.unsqueeze(-1).expand(-1, -1, seq_len)&amp; end_labels.unsqueeze(-2).expand(-1, seq_len, -1)))             match_label_mask = match_label_mask &amp; match_candidates             float_match_label_mask = match_label_mask.view(batch_size,         -1).float()      if self.loss_type == "bce":         start_loss = self.bce_loss(start_logits.view(-1), start_labels.view(-1).float()) </pre>	
--	--	--

	<pre> start_loss = (start_loss * start_float_label_mask).sum() /               start_float_label_mask.sum() end_loss = self.bce_loss(end_logits.view(-1),                         end_labels.view(-1).float()) end_loss = (end_loss * end_float_label_mask).sum() /            end_float_label_mask.sum() match_loss = self.bce_loss(span_logits.view(batch_size, -1),                           match_labels.view(batch_size, -1).float()) match_loss = match_loss * float_match_label_mask match_loss = match_loss.sum() /               (float_match_label_mask.sum() + 1e-10) else:     start_loss = self.dice_loss(start_logits, start_labels.float(),                               start_float_label_mask)     end_loss = self.dice_loss(end_logits, end_labels.float(),                               end_float_label_mask)     match_loss = self.dice_loss(span_logits, match_labels.float(),                               float_match_label_mask) return start_loss, end_loss, match_loss  def training_step(self, batch, batch_idx):     tf_board_logs = {         "lr": self.trainer.optimizers[0].param_groups[0]["lr"]     }     tokens, token_type_ids, start_labels, end_labels, start_label_mask,     end_label_mask, match_labels, sample_idx, label_idx = batch      attention_mask = (tokens != 0).long()     start_logits, end_logits, span_logits = self(tokens, attention_mask,   token_type_ids   )      start_loss, end_loss, match_loss =         self.compute_loss(start_logits=start_logits,                           end_logits=end_logits,                           span_logits=span_logits,                           start_labels=start_labels,                           end_labels=end_labels,                           match_labels=match_labels,                           start_label_mask=start_label_mask,                           end_label_mask=end_label_mask)     total_loss = self.weight_start * start_loss + self.weight_end *                 end_loss + self.weight_span * match_loss     tf_board_logs[f"train_loss"] = total_loss     tf_board_logs[f"start_loss"] = start_loss     tf_board_logs[f"end_loss"] = end_loss     tf_board_logs[f"match_loss"] = match_loss     return {'loss': total_loss, 'log': tf_board_logs}  def validation_step(self, batch, batch_idx):     output = {}     tokens, token_type_ids, start_labels, end_labels, start_label_mask,     end_label_mask, match_labels, sample_idx, label_idx = batch     attention_mask = (tokens != 0).long()     start_logits, end_logits, span_logits = self(tokens, attention_mask,   token_type_ids   )      start_loss, end_loss, match_loss =         self.compute_loss(start_logits=start_logits,                           end_logits=end_logits,                           span_logits=span_logits,                           start_labels=start_labels,                           end_labels=end_labels,                           match_labels=match_labels,                           start_label_mask=start_label_mask,                           end_label_mask=end_label_mask)      total_loss = self.weight_start * start_loss + self.weight_end *                 end_loss + self.weight_span * match_loss      output[f"val_loss"] = total_loss     output[f"start_loss"] = start_loss </pre>	
--	---	--

	<pre> output[f"end_loss"] = end_loss output[f"match_loss"] = match_loss start_preds, end_preds = start_logits &gt; 0, end_logits &gt; 0 span_f1_stats = self.span_f1(start_preds=start_preds, end_preds=end_preds, match_logits=span_logits, start_label_mask=start_label_mask, end_label_mask=end_label_mask, match_labels=match_labels) output["span_f1_stats"] = span_f1_stats return output  def validation_epoch_end(self, outputs):     avg_loss = torch.stack([x['val_loss'] for x in outputs]).mean()     tensorboard_logs = {'val_loss': avg_loss}     all_counts = torch.stack([x[f'span_f1_stats'] for x in                                 outputs]).sum(0)      span_tp, span_fp, span_fn = all_counts     span_recall = span_tp / (span_tp + span_fn + 1e-10)     span_precision = span_tp / (span_tp + span_fp + 1e-10)     span_f1 = span_precision * span_recall * 2 / (span_recall +   span_precision + 1e-10)      tensorboard_logs[f'span_precision'] = span_precision     tensorboard_logs[f'span_recall'] = span_recall     tensorboard_logs[f'span_f1'] = span_f1     return {'val_loss': avg_loss, 'log': tensorboard_logs}  def test_step(self, batch, batch_idx):     return self.validation_step(batch, batch_idx) def test_epoch_end(self, outputs) -&gt; Dict[str, Dict[str, Tensor]]:     return self.validation_epoch_end(outputs) def train_dataloader(self) -&gt; DataLoader:     return self.get_dataloader("train") def val_dataloader(self):     return self.get_dataloader("dev") def test_dataloader(self):     return self.get_dataloader("test") def get_dataloader(self, prefix="train", limit: int = None) -&gt; DataLoader:     json_path = os.path.join(self.data_dir, f'mrc-ner.{prefix}.json')     vocab_path = os.path.join(self.bert_dir, "vocab.txt")     dataset = MRCNERDataset(json_path=json_path,                            tokenizer=BertWordPieceTokenizer(vocab_path),                            max_length=self.args.max_length,                            is_chinese=self.chinese,                            pad_to_maxlen=False)     if limit is not None:         dataset = TruncateDataset(dataset, limit)     dataloader = DataLoader(         dataset=dataset,         batch_size=self.args.batch_size,         num_workers=self.args.workers,         shuffle=True if prefix == "train" else False,         collate_fn=collate_to_max_length)     return dataloader </pre>	
--	---	--

#### (四) Bagging

功能	程式碼	註記
過濾 相同特徵	<pre> deleteIndex = [] count = 0  for i in range(len(addedFile)):     for j in range(count, len(baseFile)):         if addedID[i] == baseID[j] and addedSP[i] == baseSP[j] and addedEP[i] == baseEP[j]:             deleteIndex.append(i)             count += 1             break;  filterFile = addedFile.drop(index = deleteIndex).copy() print(filterFile) </pre>	將文章編號、起始位置、結束位置、文字相同的特徵於疊加的檔案刪除。
過濾 重疊特徵	<pre> deleteIndex = [] count = 0 for i in filterFile.index:     for j in range(count, len(baseFile)):         if addedSP[i] &gt;= baseSP[j] and addedEP[i] &lt;= baseEP[j] and addedSP[i] &lt; baseEP[j]:             if addedTxt[i] in baseTxt[j]:                 deleteIndex.append(i)                 count += 1                 break  doubleFilterFile = filterFile.drop(index = deleteIndex).copy() print(doubleFilterFile) deleteIndex = [] count = 0 for i in doubleFilterFile.index:     for j in range(count, len(baseFile)):         if baseSP[j] &gt;= addedSP[i] and baseEP[j] &lt;= addedEP[i] and baseSP[j] &lt; addedEP[i]:             if baseTxt[j] in addedTxt[i]:                 deleteIndex.append(i)                 count += 1                 break  tripleFilterFile = doubleFilterFile.drop(index = deleteIndex).copy() print(tripleFilterFile) </pre>	將文章編號、起始位置、結束位置、文字重疊之特徵於疊加的檔案刪除。
合併檔案	<pre> mergeFile = pd.concat([baseFile, tripleFilterFile]) mergeFile = mergeFile.sort_values(['article_id', 'start_position']) mergeFile = mergeFile[['article_id', 'start_position', 'end_position', 'entity_text', 'entity_type']].reset_index(drop=True) print(mergeFile) </pre>	將原本的檔案與過濾特徵後之檔案合併。

#### (五) Stop words

功能	程式碼	註記
刪除 停止詞	<pre> def lenless1(row):     if row in ['禮拜', '一萬三左', '百塊', '眼前', 'HELL', 'UANG', 'ok', 'O K', 'Ok', '前二後二', '國外原', '0 點', '現在冬天', '2 次', '百七']:         return False     else:         return True mergeFile['split'] = mergeFile['entity_text'].apply(lenless1, 1) mergeFile = mergeFile[mergeFile['split']] mergeFile = mergeFile[['article_id', 'start_position', 'end_position', 'entity_text', 'entity_type']] </pre>	將不合理於正常字串的特徵，進行比對後移除。

### 三、結果分析

#### (一) 最佳結果

本系統利用上述方法，使其 F1-score 於 Public Leaderboard 達到 76.60%，Precision 及 Recall 分別為 71.63%、82.30%，而 F1-score 於 Private Leaderboard 達到 78.11%。

#### (二) 詞頻分析

	word	label_count	total_count	百分比
0	今天	171	450	38%
1	昨天	21	37	56%
2	明天	16	33	48%
3	後天	11	19	57%
4	前天	28	30	93%
5	早上	96	153	62%
6	中午	12	25	48%
7	晚上	27	105	25%
8	凌晨	1	1	100%

圖二、時間特徵詞頻分析結果

在實施本專案時，發現原始資料標記定義不一，以上述為例，在時間類別中，「今天」一詞於訓練資料出現了 450 次，有被標記的只有 171 筆，由於不清楚標記的定義為何，因此，我們嘗試各種方法，讓預測結果的表現更符合訓練資料的詞頻分布。

#### (三) 嘗試方法

##### 1. 更換特徵

本專案曾嘗試將類別標記加入 LMR，也就是除了判斷類型，也會判斷開頭字、中間字、結尾字，可惜其並未提升表現。此外，我們也曾利用全部類別進行訓練，發現 Recall 較 Precision 低，因此，將各個類別分別進行訓練，再將預測結果合併，以提升 Recall 的數值。

##### 2. 更換模型

除了使用 Bert、MRC 模型之外，我們曾嘗試使用 Crf，利用 Unigram、Bigram 方法擷取特徵，其方法之 Precision 達到 82%，其 Recall 卻只有 40-50%。此外，我們還嘗試使用 Bi-lstm 加上 Crf，其表現並未比 Bert 模型高。

### 3. 更換簡體中文

由於看到文獻結果分析，在 bert-base-chinese 的 Pre-trained 模型下，簡體中文的效能較繁體中文好，因此，我們將所有使用 Bert 所跑的模型，將其資料皆轉為簡體中文，輸出結果後，再轉回繁體中文，其 F1-score 提升了 2-3%。

### 4. 更換 Bagging 方式

在我們嘗試上述方法的過程中發現，若是增加特徵，通常會改善 Recall；但若是更換模型，通常會改善 Precision，因此，我們決定將所有預測出來的結果，利用簡單的演算法進行合併。我們曾嘗試將兩份資料格式改為 article id、position、text、type，例如今天會攤平成：[0, 0, 今天, 時間]、[0, 1, 今天, 時間]、[0, 2, 今天, 時間]，共三筆，接著比對兩份資料完全相同的特徵刪除於其中一份資料，再合併檔案。然而，該方法會有刪除不完全的問題等，因此，改為本專案提及上述之方法，完成 Bagging 程序。