AI CUP 2019 – 新聞立場檢索

隊伍名稱:

NTU_b06901087_87

隊員:

B06901087 翁瑋襄

B06901063 黃士豪

B06901020 張恆瑞

B07901069 劉奇聖

Introduction & Motivation

作為機器學習課程的期末專題，我們有三個不同的題目可以選，因此我們打算三個題目同時進行，最終取一個 performance 最好的作為最終主題。雖然預想是如此，但是因為在 Image Dehazing 與 RSNA Pneumonia Detection Challenge 兩個 task 上我們嘗試了許多種方法都距離 simple baseline 仍有一小段差距，我們只好選擇 Intent Retrieval from Online News 進行更深入的研究。

有鑑於爭議性新聞一直是閱聽人關注與討論的焦點，新聞媒體常需報導不同的立場。若能從大量的新聞文件裡，快速搜尋各種爭議性議題中具特定立場的新聞，不但有助於人們理解不同立場對這些議題的認知與價值觀，對制定決策的過程而言也相當有參考價值。這個競賽是 AIdea 人工智慧共創平台所主辦，目的是要競賽隊伍開發出一搜尋引擎，找出「與爭議性議題相關」且「符合特定立場」的新聞。在本組報告中，我們將應用資訊索引技術排序新聞與爭議性議題的相關度。

Data Preprocessing/Feature Engineering

在此部分，我們只稍微進行了三項簡單的處理：

1. 文章（語句）斷詞
2. 建立 stop word 詞典
3. 建立適合此分類任務的分詞詞典

接下來將進行這三項處理詳細的介紹。

✧ 文章（語句）斷詞

之前在作業六已經有做過語句的分類任務，在當中有用到 jieba 這個模組來進行斷詞的操作，於是我們在這個期末專題中也同樣使用這個模組來進行斷詞。我們認為要進行文章主題分類最重要的是找到一篇文章中「關鍵詞」，找到關鍵詞後我們才能更簡單的抓到一篇文章的重點。下圖便是我們實作的部分 code：

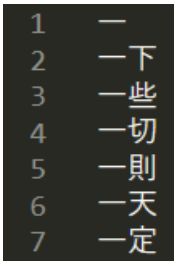
```
jieba.enable_parallel(4)
jieba.load_userdict('./auxiliary_data/dict.txt.big')
jieba.load_userdict('./auxiliary_data/my_dict.txt')
jieba.analyse.set_stop_words('./auxiliary_data/stop_words.txt')
analyzer = ChineseAnalyzer()
```

✧ 建立 stop words 詞典

因為中文中有很多冗言贅字，有些詞幾乎在每篇文章中都會出現，因此對於判斷新聞立場並沒有任何幫助，這些詞我們稱他為「stop words」。為了要

在斷詞時便提早去除這些 stop word，我們在 jieba 中加入了我們上網查找到的 stop word 辭典，並將其放入 stop_words.txt 這個檔案中。

下圖是 stop words 的一些範例：



✧ 建立適合此分類任務的分詞詞典(my_dict.txt)

因為在資料提供的某些爭議性議題中，會出現一些平常不會作為斷詞依據的詞語，例如「通姦除罪化」這個詞是作為一個議題的主題，但在斷詞時會被切成「通姦」、「除罪」、「化」三個無相關的詞語，因此我們要在 my_dict.txt 中加入「通姦除罪化」這個詞才能正確的斷句。

下圖是我們自行增加的一些詞語的範例：

完成這些預處理後，我們就可以開始進行我們的資訊索引。

Methods

我們共實作了兩種方法來進行新聞分類：tf-idf 與 Whoosh。其中 tf-idf 是一種評估一字詞對於一個文章重要程度的統計方法，此方法幫助我們突破了 simple baseline，但此統計的方法在 strong baseline 釋出後遭遇瓶頸，無法突破 0.23 以上，因此我們開始尋覓其他的方法來增進正確率。我們找到了 Whoosh，jieba 的官方 github 上推薦的搜尋引擎。Whoosh 是由 Side Effects Software 所開發，原先應用於在線協助系統的純 python 全文搜尋模組，我們將利用 Whoosh 進行文章搜尋的功能，得到與爭議性議題立場相符的新聞文章。

1. tf-idf

tf-idf 全名 term frequency–inverse document frequency，是計算特定詞語在文章中出現的頻率，再計算特定詞語出現在所有文章中所佔的篇數，其

中出現的機率越低代表該詞語對文章語意有重大影響，接著將這兩項指標相乘，得出某詞語在一文章中的重要性。在此，我們使用 scikit-learn 中的 TfidfVectorizer 進行實作，實作方法是將所有新聞斷詞後，利用該套件把每一篇新聞變成 tf-idf bag of word vectors 的形式，接著對標題作斷詞，分析該標題中的每一個詞，若詞出現在 tf-idf bag of word 中的 feature names 裡，則將該 feature name 對應的 tf-idf 欄位的分數相加，此種算法稱為 matching score。如下圖所示：

```
def cut_news():
    # load news and cut them, if there exists precut news, directly access it
    if not os.path.exists('./processed_data'):
        os.mkdir('./processed_data')
    with shelve.open('./processed_data/jieba_cut', 'c') as db:
        try:
            news = db['cut_news']
            print('Directly access precut news(jieba_cut).')
        except KeyError:
            news = utility.io.load_news()
            news = utility.cut_sentences.jieba_cut(news)
            db['cut_news'] = news

    return news

def main():
    # load news and cut them
    news = cut_news()

    # load queries and cut them
    queries = utility.io.load_queries()
    queries = utility.cut_sentences.jieba_cut(queries)

    # compute tfidf and score matrix
    matrix, feature_names = utility.embedding.tfidf(news, queries)
    score_matrix = utility.similarity.tfidf_score(matrix, feature_names, queries)

    # retrieval and output result
    if not os.path.exists('./result/'):
        os.mkdir('./result/')
    result = utility.io.retrieval(score_matrix, rank = 300)
    utility.io.output_result(result, './result/tfidf_result.csv')
```

其中所使用的 function 如下：

```
def tfidf_score(matrix, feature_names, queries):
    '''
    For each query, directly add those feature names exists in both the
    query and the tfidf bag of word vector that each news corresponding to.

    # Arguments:
        matrix(scipy.sparse.csr.csr_matrix)
        feature_names(list of str)
        queries(list of str): precut queries
    '''
    score_matrix = list()
    for query in queries:
        query = query.split()
        idx = [i for i, name in enumerate(feature_names) if name in query]
        score_array = np.sum(matrix.T[idx], axis = 0).A1 # flatten the "numpy.matrix"
        score_matrix.append(score_array)

    return score_matrix
```

最後使用 priority queue(heapq)，取出前 300 個分數最高的新聞：

```
def tfidf(news, queries):
    '''
    Use sklearn.feature_extraction.text.TfidfVectorizer to convert
    documents or sentences into tfidf bag of word vectors.

    # Arguments:
        news(list of str): precut news
        queries(list of str): precut queries

    # Returns:
        matrix(scipy.sparse.csr.csr_matrix)
        feature_names(list of str)
    '''
    print('compute tfidf...')
    vectorizer = TfidfVectorizer()
    vectorizer.fit(news + queries)
    matrix = vectorizer.transform(news)
    feature_names = vectorizer.get_feature_names()

    return matrix, feature_names
```

```
queries_1 = load_queries()
queries_2, news_index, relevance = load_training_data()
result = list()
td_sz = len(relevance)
for idx,score_array in enumerate(score_matrix):
    ques = queries_1[idx]
    popout = []
    popout_0 = []
    popout_1 = []
    popout_2 = []
    popout_3 = []
    for j in range(td_sz):
        if queries_2[j] == ques and relevance[j] == 0 and (news_index[j] not in popout):
            popout_0.append(news_index[j])
        elif queries_2[j] == ques and relevance[j] == 1 and (news_index[j] not in popout):
            popout_1.append(news_index[j])
        elif queries_2[j] == ques and relevance[j] == 2 and (news_index[j] not in popout):
            popout_2.append(news_index[j])
        elif queries_2[j] == ques and relevance[j] == 3 and (news_index[j] not in popout):
            popout_3.append(news_index[j])
        popout.append(news_index[j])
    pairs = [(-score, i) for i, score in enumerate(score_array)]
    heapq.heapify(pairs)
    # single_result = [heapq.heappop(pairs)[1] for _ in range(rank)]
    single_result = []
    cnt = 0
    while cnt<rank-len(popout_1)-len(popout_2)-len(popout_3):
        element = heapq.heappop(pairs)[1]
        if (element not in popout_0) and (element not in popout_1) and (element not in popout_2) and (element not in popout_3):
            single_result.append(element)
            cnt+=1
    result.append(popout_3+popout_2+popout_1+single_result)
result = np.array(result)[:300]

return result
```

2. Whoosh

在使用 Whoosh 之前，我們要先利用 jieba 建立一個 Chinese Analyzer：

```
jieba.enable_parallel(4)
jieba.load_userdict('./auxiliary_data/dict.txt.big')
jieba.load_userdict('./auxiliary_data/my_dict.txt')
jieba.analyse.set_stop_words('./auxiliary_data/stop_words.txt')
analyzer = ChineseAnalyzer()
```

接著使用 Whoosh 創建一個 Schema object：

```
schema = Schema(title = TEXT(stored = True),
                content = TEXT(stored = True, analyzer = analyzer))
```

利用此 Schema 創建一個 index directory，並將所有的新聞加入這個 directory 裡面：

```
if not os.path.exists('./indexdir/'):
    os.mkdir('./indexdir/')
    ix = create_in('./indexdir/', schema)
    # add documents
    news = utility.io.load_news()
    writer = ix.writer()
    print('Add documents...')
    for i, x in enumerate(news):
        if i % 1000 == 0:
            print('\t%d documents have been added.' % i)
        writer.add_document(title = 'news_%06d'%(i+1), content = x)
    writer.commit()
else:
    print('Directly open previous indexed directory...')
    ix = open_dir('./indexdir')
```

接著要挑出對應每個議題的關鍵字，利用 Whoosh 的邏輯式串連起來再進行索引搜尋。因為會有立場判斷的問題，因此我們會在某些判斷式上加上帶有立場的詞彙，藉此區別出新聞的立場。例如當我們處理「支持博弈特區在台灣合法化」這個議題時，我們會先取出「博弈」、「特區」和「合法化」三個詞彙作為最主要的搜尋目標，只要帶有這三個詞彙的文章基本上可以斷定是跟此議

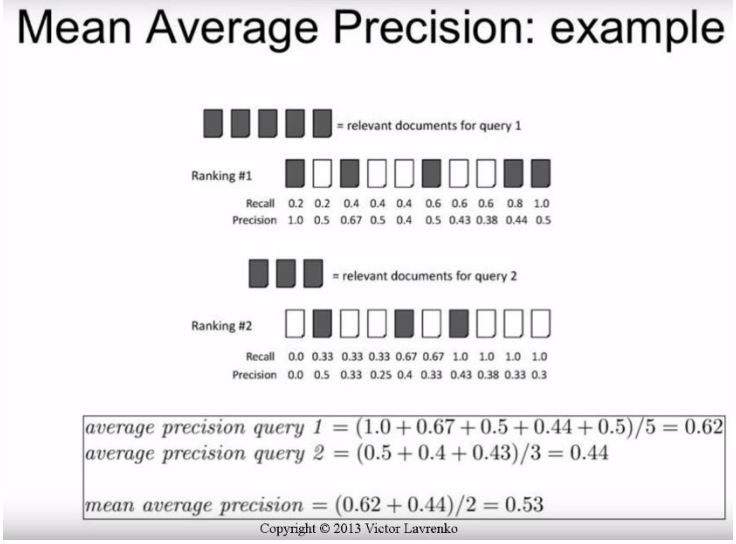
題有相當大的關聯性。這時我們需要考慮的立場問題就利用 NOT 語句來排除反面的論述。例如因為當時反對博弈的主要勢力是反賭博合法化聯盟，只要有出現這個聯盟的新聞基本上就不太會支持與此議題相同的立場，因此我們予以刪除。最終便可以得出下面的式子：

```
AND('博弈', '特區', '合法', NOT(OR('反賭博合法化聯盟', '反賭聯盟', '反彈'))),
```

因為有時會有搜尋出的新聞量太多或太少的問題，因此有些 case 我們要不斷反覆調整關鍵字內容才能得出較好的結論。如「不支持使用加密貨幣」這項議題因為領域性較嚴重，搜尋時一直沒辦法得到超過 300 篇的新聞數量，因此我們將判斷式加長了許多次才成功得到足夠數量的搜尋結果。如下圖所示：

```
AND(OR('關閉','禁止','禁令','警告', '停止', '威脅', '反對', '暫停', '中止', '終止'),\
    OR('交易所','虛擬貨幣','加密貨幣', '數位貨幣', AND('加密', '幣'), AND('比特', '幣'))),
```

單單藉由 Whoosh 並不能一舉通過 strong baseline，因為仍有一些新聞是包有正反論述，甚至是多重否定語句，當碰到這些特殊情況時我們所利用的簡單判斷式便不能準確地取出相同立場、主題的新聞。我們原先想利用 training 的方式得出每種新聞中的立場，但無奈主辦方給予的 labeled data 實在太少，我們 train 出的 model 只能判斷出主題的異同卻無法判斷立場的正反。這時我們突然想到在助教的投影片中有提到主辦方計分方式的介紹，如下圖所示：



這種計分方式代表如果我們將前 300 則新聞越前面的資料正確率提升，那麼連帶著整體判斷出的分數也會提高。秉持著這種想法，我們將主辦方給予的少數 label 加入我們判斷前 300 則新聞的排序，如下圖所示：

```
with ix.searcher() as searcher:
    queries_1 = utility.io.load_queries()
    queries_2, news_index, relevance = utility.io.load_training_data()
    td_sz = len(relevance)
    L = list()
    for idx,keyword in enumerate(QUERIES):
        ques = queries_1[idx]
        popout = []
        popout_0 = []
        popout_1 = []
        popout_2 = []
        popout_3 = []
        for j in range(td_sz):
            if queries_2[j] == ques and relevance[j] == 3 and (news_index[j] not in popout):
                popout_3.append(news_index[j])
            elif queries_2[j] == ques and relevance[j] == 2 and (news_index[j] not in popout):
                popout_2.append(news_index[j])
            elif queries_2[j] == ques and relevance[j] == 1 and (news_index[j] not in popout):
                popout_1.append(news_index[j])
            elif queries_2[j] == ques and relevance[j] == 0 and (news_index[j] not in popout):
                popout_0.append(news_index[j])
            popout.append(news_index[j])
        if len(popout)>0:
            print(popout[0])
        print("result of ", keyword)
        q = keyword
        # q = parser.parse(keyword)
        results = searcher.search(q, limit = 400)
        print(len(results))
        print(len(results.top_n))
        res = []
        cnt = 0
        i = 0
        while cnt<300-len(popout_3)-len(popout_2)-len(popout_1) and i<len(results.top_n):
            element = results[i]['title']
            i+=1
            if (element not in popout_1) and (element not in popout_2) and (element not in popout_3) and (element not in popout_0):
                res.append(element)
                cnt+=1
        # print(results[0])
        print('='*40)
        ans = popout_3+popout_2+popout_1+res+popout_0
        L.append(ans[:300])
```

事實證明我們的策略是正確的，在將官方 labeled 好的 data 前推/去除後，大幅度的增加了正確率，一下子來到全排行榜的第二名。

Experiment and Discussion

下表為 tf-idf 與 whoosh 的實驗結果:

	jieba+TFIDF	jieba+whoosh
沒加官方 label	0.1911667	0.2856494
加上官方 label	0.2305621	0.4255962

除了前段敘述過的 tf-idf 以及 whoosh，我們還嘗試了使用不同的方法，列舉如下：

1. Word2Vec + Word Mover’ s Distance
   我們試著將每篇文章段詞後，把字詞轉為向量表示法，利用 word mover’ s distance 來找出文章間的關係。但是整個資料庫有十萬筆新聞，要全部匹配是非常困難的事情，因此最後沒有採用。
2. BERT + Cosine Similarity
   利用 BERT (Bidirectional Encoder Representation from Transformers)，把每篇文章轉為一個個向量，並且使用 cosine similarity 來算和新聞主題間

的關聯性。然而不知為何，這個方法就是無法判別新聞立場(score: 0.0470163)。

3. ERNIE
   因為 ERNIE 是專門設計給中文的 NLP 用，所以我們想用 ERNIE (Enhanced Representation from kNowledge IntEgration)，但是因為 ERNIE 是用 paddlepaddle 套件，讓我們有點難懂，因此最後就不使用這個方法了。

探討上面的結果:

1. 有把官方 label 好的資料放入結果中能讓準確率突飛猛進。
2. 用 TFIDF 搜尋文章的時候，把標題裡的每一個詞都當做同等重要的計算標題和文章的相似度；用 whoosh 搜索文章的時候，則是自己定義要搜尋的關鍵詞，因此 whoosh 搜尋的結果可能因此比 TFIDF 更加準確。
3. 我們起初想要設計一個 binary classifier 判斷 query 跟新聞的立場，但是多方嘗試後沒有得到一個足夠好的結果，最後透過設定判斷立場的關鍵字，能對正負立場的判斷有較好的結果。
4. 我們發現當還沒有加上官方 label 資料的準確度分數越高的情況下，可能因為篩選出來的資料跟主題比較相近的緣故，使得加上官方 label 資料後分數進步較多。

Conclusion

我們用 jieba 對文章做斷詞作為前處理，之後嘗試了各種不同的方法，包含了機器學習的方法：BERT、ERNIE、gensim Word2Vec、RNN 等，以及非機器學習的方法：Tf-Idf、Whoosh 等，最終表現最好的方法是 Whoosh 加上官方訓練資料。機器學習的方法全部失敗，Tf-Idf 可以通過 simple baseline。

在這次的 final project 中我們獲益良多，不只嘗試了各種機器學習和非機器學習的方法去判斷新聞立場與議題關聯性，更學會了許多資料探勘的套件使用方法及原理。雖然最終大部分的方法，尤其是機器學習的部分都失敗了，但我們仍在此次的專題中體驗了腦力激盪、搜尋資料和將想法付諸實踐的過程。

Reference

1. AIdea: 新聞立場檢索技術獎金賽
https://aidea-web.tw/topic/b6abbf14-2d60-456c-8cbe-34fdfcd58967
2. stop_words.txt

https://dannypheobe.blogspot.com/2016/07/python.html?fbclid=IwAR2IOos2Oofdr3pe1U0PjKSV9sgSWNNJW-zQkUQwWMJySklUaW8ilEbs8cs

3. jieba

https://github.com/fxsjy/jieba

4. Whoosh

https://whoosh.readthedocs.io/en/latest/

5. TFIDF (scikit-learn)

https://scikit-learn.org/stable/modules/feature_extraction.html

6. TFIDF (gensim)

https://radimrehurek.com/gensim/models/tfidfmodel.html

7. W2V+WMV (gensim)

https://radimrehurek.com/gensim/similarities/docsim.html

8. BERT

https://github.com/hanxiao/bert-as-service

9. ERNIE

https://github.com/PaddlePaddle/LARK/tree/develop/ERNIE