2022年4月9日 星期六

Introduction to AI HW2 report

109550155 端木竣偉

Part 0.Preprocessing

觀察dataset的一些評論，以及研究各種權威性文章提供的預處理方法後，筆者選用了以下幾種方法來預處理資料：

1.去除
：許多資料有這個奇怪的符號，據查詢是HTML的換行，因沒有意義，所以將其用replace()轉換成空白符號。

Ex:”I love it!
How about you?”–>”I love it! How about you?”

2.移除stopwords：太頻繁的停用詞沒有意義，使用助教提供的函式去除。

Ex:”It is a dog”–>”dog”

3.將大寫轉換成小寫：去除雜訊，用lower()處理。

Ex:”What??”–>”what??”

4.去除標點符號：使用not in string.punctuation與5.一同用for迴圈處理。

Ex:”Oh….”–>”Oh”

5.去除數字：資料中數字的意義太混亂，可能是正面意義的評分，也可能是負面的（例如：難看程度10/10），甚至有關於年代的，故使用not isdigit()去除。

Ex:”It’s the worst movie in 2020”–>”It’s the worst movie in”

6.Stemming:用SnowballStemmer去將單字還原成最原本的樣子，讓模型不需額外處理其他訊息。
EX:”loved”–>”love”

底下的英文例句是上述一起使用的結果：

“He respected Robert Tarjan
,and Tarjan algorithm is the best 77777.”
->”respect robert tarjan tarjan algorithm best”

Part1 Implement the bi-gram language model

Bi-gram的概念相當簡單，就是我們去遍歷資料庫，並計算一個詞出現在另一個詞後面的機率：
C是統計出現的次數

$$P(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)}{C(w_{n-1})}$$

我們只需要統計完之後，存為model[wi-1][wi]的形式，而feature[(wi-1,wi)]則是儲存bigram的頻率。

Part 2 perplexity computation

當我們建立完model後，就可以去計算一個句子出現的機率，實際上就是將各個bigram相乘，而公式如下：

$$2^{-l} \text{ where } l = \frac{1}{M}\sum_{i=1}^{m} \log p(s_i)$$

Log的base是2，取log是因為相乘數字會太小無法紀錄，因此進行這樣的轉換，當我們使用test data去測試model時，算出來的perplexity越低，代表模型認為test data中該句子出現的機率越高出現。

而我們在計算bigram的時候，可能會發生part2第三點提到的，分母為0的情況，即未曾出現在train data裡面的單字，因為我們訓練時不可能包含所有的可能。

要解決這個方式，就是給每種可能一個小小的機率，而在實踐方法中Add-1(Laplace) method，實際處理如下：

$$P_{\text{Laplace}}(w_n|w_{n-1}) = \frac{C(w_{n-1}w_n)+1}{\sum_w (C(w_{n-1}w)+1)} = \frac{C(w_{n-1}w_n)+1}{C(w_{n-1})+V} \qquad (3.23)$$

V是單字種類數量，在使用之後就不會發生上面提到的問題。

Part 3 Implement feature selection and conversion

這個步驟即是我們從所有的bigram裡面選取一些作為feature，但由於無法全部選擇，就在刪除了stopwords之後，將feature依照出現頻率排序，選擇最頻繁出現的作為feature。

Experiment

Preprocessing

feature_num固定在250，測試每一個preprocess方法(除了
，因為那原本就不是評論的一部分)移除之後的效果：

Without	Perplexity	F1 score	Precision	Recall
所有	3635.23368	0.6704	0.672	0.6709
無	18055.42932	0.6873	0.7017	0.6913
Remove stop words	2205.35291481	0.7019	0.7073	0.7033
去除標點符號	13499.0522446	0.6563	0.6798	0.6639
去除數字	18860.0874089	0.6889	0.7041	0.6931

Without	Perplexity	F1 score	Precision	Recall
Stemming	32678.8874653	0.6732	0.6933	0.6792
Remove stop words and 數字	2312.07711633	0.7007	0.7062	0.7022

通過上述數據能發現，移除remove stopwords和去除數字都使正確率上升，於是我後來又再測試了兩者都移除的結果，但反而讓正確率略低於只將remove stopwords的結果。而stopwords裡面包含了not，去除反而使正確率降低(not good變成good)或許這裡的test data中，數字大多數是有表達意義的，所以之後的測試會以移除
+標點符號+stemming來進行。

另外我們可以看到這裡的perplexity都是偏大的，我認為是因為在使用add-one的過程中，V的數量相相大，讓機率變低，使得perplexity變大。

但這樣的解釋會有一個問題，在沒有使用stopword的情況下，V應該會更大，但實際上perplexity卻顯著下降，我的判斷是因為stopwords移除的單字實際上不多(即V增加的數量不多)，但他們的出現的次數卻相當頻繁，機率會很高，根據perplexity的公式會使結果變小。而上面的表格，不做remove stopwords讓perplexity顯著降低，便印證這樣的假設。

還有一個現象也值得探討：根據定義，perplexity是判斷一個句子的通順程度，這裡的結果是test data裡面的句子在train data的架構下，句子的通順程度。而這跟情緒判斷沒有什麼關聯性，所以perplexity跟其他三項結果才無太大的關聯。

Feature_num

下面表格是我調整feature_num所得到的不同結果：

feature_num	Perplexity	F1 score	Precision	Recall
250	2312.07711633	0.7007	0.7062	0.7022
500	2312.07711633	0.7259	0.7278	0.7263
750	2312.07711633	0.75	0.7508	0.7502
1000	2312.07711633	0.7705	0.771	0.7706
1250	2312.07711633	0.7783	0.7784	0.7783

feature_num	Perplexity	F1 score	Precision	Recall
1500	2312.07711633	0.7922	0.7928	0.7923

經過測試，我們可以發現在bi-gram的model中，feature_num的多寡和正確率是成正相關的，當初我們去控制feature的數量純粹是因為我們無法紀錄所有的feature，雖然過多的feature可能會造成判斷困難，但250~1500這個範圍，提升feature_num顯然是有助於模型判斷。

Discussion

1.What are the reasons you think bi-gram cannot outperform DistilBert?

先將DistilBert的結果列在下表：

DistilBert	Perplexity	F1 score	Precision	Loss
No preprocess	X	0.931	0.931	0.2289
With preprocess	X	0.8781	0.8833	0.3283

這裡的第二個結果是包含所有的預處理的，我們可以看到正確率反而比未處理的低，我認為是因為預處理雖然是將資料的雜訊減少，但有時會讓能夠判斷的資訊一起被刪除。但即使較低，bi-gram最優的結果(feature_num=1500)仍只有0.7922，和Bert的正確率仍有落差。

Bert能夠基於上下文的關係去辨識句子，還能知道字詞跟答案的關係，又可以Fine-tune。而bi-gram的模型卻只考慮相鄰兩個詞的關係，所以正確率才低於BERT。

兩者在計算量上也有相當差距，即使DistilBert已經比Bert速度快上不少，但筆者用MAC筆電跑了一晚上的訓練才接近一半，而bi-gram頂多五分鐘之久，兩者計算量差距之大，這樣的差異也能側面顯示為何bi-gram無法outperform DistilBert了。

2.Can bi-gram consider long-term dependencies? Why or why not?

如上一題所提到的，bi-gram只考慮到前後兩個字之間的關係，前文的資訊並不會去影響判斷當前的資訊，而Bert正是因為能做到這點才得以勝過bi-gram。

3.Would the preprocessing methods improve the performance of the bi-gram model? Why or why not?

我認為是可以的。從experiment的結果能看出，適當的preprocess是能夠提升bi-gram的正確率和F1-score的，原因是preprocessing能夠過濾掉沒有意義的訊息，提高模型的判斷能力，但要挑選合適的preprocess，不然會過濾掉有用的資訊使得結果變壞。

4.If you convert all words that appeared less than 10 times as [UNK] (a special symbol for out-of-vocabulary words), would it in general increase or decrease the perplexity on the previously unseen data compared to an approach that converts only a fraction of the words that appeared just once as [UNK]? Why or why not?

我認為會decrease，相比於只將部分只出現一次的單字改成[UNK]，將所有出現頻率小於10次的字詞轉為[UNK]，低頻率的單字變得更少，計算時在句子中遇到沒看過的詞，給定的機率也會變大，算出來的機率變高，perplexity也會因此而降低。

Problem I met

1.觀念不熟與實作困難

雖然已經看完了李教授的講解但還是不懂實作細節，而Bi-gram雖然在其他課程已經寫過，但我是第一次試著用python寫，而且當初學到的是用來解密文，不清楚跟NLP有什麼關係經過上網查詢資料還有範例，成功理解並寫出了code，也學習到不少python特有的一些快速寫法。

2.perplexity大的誇張

一開始做完並跑了程式後，看到perplexity相當大，以為是自己寫錯了，相當緊張，經過半天的資料與王助教的幫助，才知道是正常現象，並且成功找出原因。

3.Bert運行時間相當長

一開始是用自己電腦跑run.sh，睡前啟動之後，隔天起床卻發現還在訓練且風扇聲很大，頓時認為應該尋找其他方法，詢問了其他同儕之後改用colab，選用GPU後約一小時完成，頓時理解到colab的強大之處。