Introduction to Artificial Intelligence

Homework 2 report

109550127 宋哲頤

1. preprocessing method

input: Basically ' family little boy (Jake) thinks ' zombie closet &

parents fighting time.<br / ><br / >This movie slower soap opera ... suddenly , Jake decides become Rambo kill zombie.<br / ><br / >OK , first ' going make film must Decide thriller drama ! drama movie watchable. Parents divorcing & arguing like real life. Jake closet totally ruins film ! expected see BOOGEYMAN similar movie , instead watched drama meaningless thriller spots.<br / ><br / >3 10 well playing parents & descent dialogs. shots Jake : ignore .

```
#拔HTML
preprocessed_text = re.sub("<[^>]+>", "", preprocessed_text).strip()
```

Output: Basically ' family little boy (Jake) thinks ' zombie closet & parents fighting time.This movie slower soap opera ... suddenly , Jake decides become Rambo kill zombie.OK , first ' going make film must Decide thriller drama ! drama movie watchable. Parents divorcing & arguing like real life. Jake closet totally ruins film ! expected see BOOGEYMAN similar movie , instead watched drama meaningless thriller spots.3 10 well playing parents & descent dialogs. shots Jake : ignore .

```
#拔標點符號
preprocessed_text = re.sub(r'[^\w\s]','',preprocessed_text)
```

Output: Basically family little boy Jake thinks zombie closet amp parents fighting timeThis movie slower soap opera suddenly Jake decides become Rambo kill zombieOK first going make film must Decide thriller drama drama movie watchable Parents divorcing amp arguing like real life Jake closet totally ruins film expected see BOOGEYMAN similar movie instead watched drama meaningless thriller spots3 10 well playing parents amp descent dialogs shots Jake ignore

```
#lemmatize
lemmatizer = nltk.stem.wordnet.WordNetLemmatizer()
word_tokenizer = nltk.tokenize.regexp.WordPunctTokenizer()
preprocessed_text = word_tokenizer.tokenize(preprocessed_text)
def lemmatize(word):
    lemma = lemmatizer.lemmatize(word,'v')
    if lemma == word:
        lemma = lemmatizer.lemmatize(word,'n')
    return lemma
preprocessed_text=[lemmatize(token) for token in preprocessed_text]
preprocessed_text = ' '.join(preprocessed_text)
```

轉換詞性，如:divorcing -> divorce

Output: Basically family little boy Jake think zombie closet amp parent fight timeThis movie slower soap opera suddenly Jake decide become Rambo kill zombieOK first go make film must Decide thriller drama drama movie watchable Parents divorce amp argue like real life Jake closet totally ruin film expect see BOOGEYMAN similar movie instead watch drama meaningless thriller spots3 10 well play parent amp descent dialog shot Jake ignore

2. different numbers of feature_num

我用了 feature_num:500,1000,2000 隨著 feature 的數量越多，準確率也越高。因為當 bi-gram 的 feature tuple(s1,s2)越多，能夠合理判斷評論為 positive 或 negative 的 feature 也越多，但同時跑的時間也會多很多。

Feature500

Preprocess0:

F1 score: 0.7057, Precision: 0.7088, Recall: 0.7065

Preprocess1:

F1 score: 0.695, Precision: 0.7054, Recall: 0.6978

Feature1000

Preprocess0:

F1 score: 0.7373, Precision: 0.7401, Recall: 0.7379

Preprocess1:

F1 score: 0.7291, Precision: 0.7376, Recall: 0.731

3. perplexity、F1-score、precision、recall of different methods

Prepocess0:

F1 score: 0.7057, Precision: 0.7088, Recall: 0.7065

Perplexity: 116.34664594156884

Prepocess1:

F1 score: 0.695, Precision: 0.7054, Recall: 0.6978

Perplexity: 231.6915678041184

在我處理完標點符號、HTML、詞性變換後的句子進行 ngram，按照數據顯示幾乎一樣，我認為去除掉標點符號和 HTML 標籤是對 ngram 的判斷影響不大的，而 lemmatize 的詞性轉換或許會讓整個動詞變得比較難以判別，但在經過 Prepocess 過後有一個很大優點，就是因為不重要的標點符號和 HTML 標籤讓整體的 ngram 執行速度快了快兩倍，而準確率卻差不多，顯然有做 preprocess 還是對 NLP 的演算法的執行有一定的幫助。

Preplexity 在做完 preprocess 後反而變高，我認為是我處理 divide 0 的情況造成的，詳細我在 5.(1)敘述。當我遇到 key error 就是 train 完後的 model 在 test 中沒有對應的 bigram，而我

4. difference between the bi-gram model and DistilBert

   a. DistilBert 延續 Bert 的優點，使用大模型的學到的知識訓練小模型，從而讓小模型具有大模型的泛化能力。移除了 BERT 模型的 token 類型 embedding 和 NSP（下一句預測任務），保留了 BERT 的其他機制，然後把 BERT 的層數減少為原來的 1/2。而他的大量參數也造就高度的準確度，相對 bi-gram 只將固定頻率的 feature 來判斷機率再預測，雖然較快但準確度也沒比較高。

   b.不行。因為 bigram 的做法為兩兩一個做預測，沒辦法像 BERT 一樣 long-term dependencies

   c.我認為 preprocess 對準確率沒有太大的提升，但對速度卻有顯著的提升，因為句子在 preprocess 之後長度縮短了，減少了每個迴圈需要跑的 case，而或許我 lemmatize 過後詞性的改變造成 ngram 更難判斷，而 bert 裡他是偏重看整體句子的做法，在 preprocess 後容易使準確率降低

   d.我認為會 increase perplexity 因為把頻率從 10 次變 1 次後，[UNK]的數目會大幅增加會造成難以預測下一個字出現的機率，最後只能 smooth，如此會使混淆度變大。

5. Describe problems you meet and how you solve them

(1) perplexity encounter zero division error

當我遇到 key error 就是 train 完後的 model 在 test 中沒有對應的 bigram，我的作法是假如一段句子[CLS] A B C，假定一個數 s=0.01，當成是跑到 A，而 A 不在我 train 好的 self.model.keys()裡，我就會讓 tmp+=log(s)，又如果 A 在但 self.model[A]的 keys()沒有 B，那我就找 A 的前面[CLS]存不存在，存在我就找 self.model[[CLS]].keys()，有的話就加上 self.model[[CLS]][B]沒有的話就 tmp+=s。我認為是我處理 divide 0 的特判太多造成經 preprocess 後的句子沒有容易出現的逗號或句號來猜後面的字串，這也造成混淆度增加。

(2) feature collection

我在 parl 蒐集 train.csv 裡全部的 feature 並尋找每個 feature 的出現頻率時，我將全部所有組合存下來並且再從 data 的自兩個兩個掃過去看有沒在在前面蒐集的 feature 資料庫裏面，但這樣居然要跑快 8 個小時，後來仔細想想只要記錄一個句子掃過去的兩兩線性組合就好，果然時間從 8 小時變 20 分鐘完成，效率差非常多。