NLP 2017 Project 2 Report

Team name: Hotel? Trivago!
Members:
B03902101 楊力權 b03902101@ntu.edu.tw
R05922038 黃郁庭 r05922038@ntu.edu.tw
B03902086 李鈺昇 b03902086@ntu.edu.tw
（可以分享到作業觀摩區）

Method 1

一、簡介

由於近年來使用 Recurrent Neural Network (RNN) 來處理語言相關的問題非常流行，因此我們嘗試的第一個方式就是使用 RNN。最基本的流程大致如下：

1. 將 train.csv 與 test.csv 分別作斷詞
2. training 的詳細過程（由於本來 data 量就很少，因此沒有切出 validation set）：
    a. 將所有句子的詞轉成向量（word embedding）
    b. 分批把三維的向量（#batch, maximum sentence length, embedding size）餵給 LSTM，再過 fully connected layer，產出 4 個實數
    c. 出來的這些值經過某個 activation function map 到 (0, 1)
    d. 如此一來，每一句都會被轉換成四個 (0, 1) 的分數，分別對應到 4 個分類，就取這四個中最大的當作預測的類別
3. testing 的流程與 training 基本上差不多

二、嘗試與分析

最原始的參數與設定：

- 句子的 word vectors 全部靠左對齊，往右補 zero vectors 直到與最長的句子長度一樣
- optimizer: Adam
- 對於一行 data，把兩個 clauses 接在一起視為一個句子
- 使用 jieba 斷詞
- LSTM 總共跑 100 epochs
- dropout = recurrent_dropout = 0.2
- activation function: sigmoid f(s) = (1 + e^(-s)) ^ (-1)
- 一層 LSTM
- 使用中研院平衡語料庫 (ASBC) pretrain 好的 word2vec model

以下是在其他參數/設定完全不變的情況下，嘗試過但變差的改變：

- 使用 Bidirectional LSTM
- sigmoid 換成 softmax
- 使用 inverse frequency 作為 class weight
- 多加一層 LSTM
- 使用 keras 的 Embedding layer

進步過程：（以下表格在其他參數/設定完全不變的情況下，從上一列做的更改）

更改	備註	Kaggle score	
		public	private
-	最原始的版本	0.644	0.636
word vectors 靠右對齊，往左補 0	應該是 LSTM 的 memory 機制導致最後都是 0 的話效果比較不好（根據 public）	0.664	0.630
把 optimizer 換成 RMSprop	epoch-accuracy 曲線看起來比較不穩定，但是 accuracy 上升比較快	0.664	0.610
把兩個 clauses 分別餵給兩個 LSTM，兩個結果 concatenate 之後再給 fully connected layer	雖然 public 分數沒有變好，但是相信這樣做是比較好的，而 private 也的確有上升	0.648	0.646
將每個 clause 都先左右反轉	想法部分來自 Sequence to Sequence Learning with Neural Networks 這篇論文	0.678	0.670
改為用 CKIP 斷詞；在 training accuracy 到達 0.999 時 early stop（停在 57 epochs）	0.999 是經過前幾次實驗的結果人工挑的數字	0.682	0.660
dropout = recurrent_dropout = 0.3	實驗結果是 0.3, 0.4 差不多一樣好	0.694	0.688
先把所有 data 裡面出現最多的前 M 個詞過濾掉，然後從每個類別分別挑出出現最多的前 N 個關鍵詞（共 4N 個），去掉重複的之後剩下 K 個，姑且叫作 f[1], f[2] …, f[K]。 之後對每個句子，多建立一個 K+1 維向量 v[1..K+1]：對句子裡面的詞 w，如果 w = f[i]，就讓 v[i] += 1；否則讓 v[K+1] += 1。 model 則改為把兩個 LSTM 輸出的結果跟 K+1 維的 v 向量concatenate 在一起（共三個向量），再給 fully connected layer	M 設為 30，N 設為 10，這樣的 K 剛好是 30。30 跟 10 都是經過肉眼觀察之後挑的值（看起來比較符合各個分類的 keywords） 30 個關鍵詞： 時、來、得、又、再、將、沒有、開始、卻、並、當、但是、自己、這、後、由於、因為、到、能、所以、雖然、與、然後、更、之後、一個、您、或、最、！	0.700	0.674
把 M 從 30 改成 22（過濾掉少一點 IDF 高的詞，保留多一點各類別高 TF 的詞）	所有上傳中 private 最高，不過因為 public 不高而沒選	0.682	0.698
對上傳過的前 23 高分的結果做 uniform ensemble（對每筆 testing data 做投票，23 個結果票票等值）	23 是實驗過的數字中 public score 最高的	0.702	0.680
對上傳過的前 10 高分的結果做 weighted ensemble（對每筆 testing data 做投票，23 個結果用 public score 做為票數的權重）	會做 weighted 版本是因為覺得 public score 比較好的應該會做出比較好的預測	0.696	0.684

最後我們選擇的是表格中的最後兩筆，最終 private score 為 0.684，而 private 比 public 進步一名。

Method 2

一、簡介

對每個句子來說，它含的字詞就像是它的feature，因為這些feature不論是對句子承接關係或是語意轉折，都有很大的影響，簡單來說就是有些字詞在某種類的句中常出現，卻不常在其他種類的句子中。
且因為這次的題目，是把給定的句子分類成Temporal, Contingency, Comparison, Expansion四種關係，所以如果在會影響句子關係的關鍵字很多且影響力不小的情況之下，把train.csv中的句子，兩句(2 clauses)當成一句，train一個classifier model來分類test.csv中的句子是一個很直觀的想法。
以下是詳細的步驟與分析：

二、步驟

1. 用jieba斷詞，把每個句子，斷成詞建成詞典
2. 留下部分的詞當作feature
   斷完的相異詞太多，且有些詞只出現過極少次，因此要限制能成為feature的詞。
   此步驟有很多種方式也有不同的結果：
   - 出現次數前面的詞
   - 出現次數前面的連接詞與一般詞
   - 出現次數前面的連接詞、副詞、介詞與一般詞
3. 把每個句子轉換成一個多維的feature陣列
4. train一個4 classes的classifier
5. 將test data斷詞並做步驟3.
6. 把每一句的feature丟進classifier model得到每一句的類別

三、分析

這個方法要考慮的點就是feature要取多少以及怎麼取，所以我們分析了幾種方式：

1. 出現次數前1500的詞(1500數字是由切train data得到準確率高的數字)
   - 兩個句子之間的關係，會跟某些出現的詞有關係，因此把出現率高的詞當作feature訓練classifier model，可能可以看出詞對兩句關係之間的影響。
   - kaggle public:0.628 private:0.594
2. 出現次數前100的連接詞與900的一般詞
   - 兩個句子會有關係，連接詞(如：但是、卻、可是、所以...等)有很大很根本的影響力，但用「方法1」 可能只取到一點點的連接詞，所以希望能保障一定數量連接詞，觀察對句子關係的影響。
   - 用jieba的詞性斷詞，把前100個連接詞混合900個一般詞當作feature訓練。
   - kaggle public:0.634 private:0.654
3. 出現次數前120的連接詞與1080的一般詞
   - 因為「方法2」的效果不錯，因此我們試著加入更多連接詞
   - kaggle public:0.650 private:0.664
4. 出現次數前120的連接詞、副詞、介詞與1080的一般詞
   - 雖然大部分的關鍵詞都在連接詞中，但有一些詞(如仍然、仍舊...等)屬於副詞與介詞，而他們對句意有很直接影響，因此把這兩種詞納入考量。
   - kaggle public:0.628 private:0.664
   - 或許是副詞與介詞中大部分的詞都與兩句承接無關，因此得到比只取連接詞的結果還差。

四、結論

因此對這個方法的分析是，混合少量連接詞與一些一般詞為feature，能讓我們訓練得到最好的classifier model。

Method 3

一、方法

1. 資料前處理

a.斷詞：利用jieba將train.csv與test.csv做斷詞。
b.利用詞性篩選比較重要的詞：使用coreNLP標註詞性，把標為與Temporal, Contingency, Comparison, Expansion比較有關連的詞性的詞彙增加詞頻、達到放大重要feature的效果，例如：在原本詞彙後面在串接一次。這些比較有關連的詞性包括：NT (temporal noun) 、NN (Common nouns)、VV (verbs)、VA (Predicative adjective)、JJ ([Noun-modifier other than nouns)等。然而此種做法可能會破壞句子結構，進而影響performance的表現，從後面的結果與分析中顯示此種做法並不如預期的有效。

2. 模型

由於目前在自然語言領域中，能達到一定預測精確度的模型一般都是以RNN為主，而我們好奇同為深度學習的另一知名架構—convolutional neural networks (CNN) 是否能達到不輸RNN的效果，因此除了前面提到的RNN架構，我們也嘗試了CNN模型。我們嘗試的CNN架構是根據Yoon Kim於2014年發表的論文 (https://arxiv.org/abs/1408.5882) 而改為較簡單的模型，我們主要的架構如下：

- word embedding：有嘗試過使用gensim套件的word2vec()訓練產生每個詞彙所對應到的vector，或在CNN模型中自動訓練，但效果差不多。
- 1st dropout：減少overfitting的發生。
- 1-D Convolution layer + 1-D Max Pooling layer + Flatten：另外也有嘗試2-D版本。
- 2nd dropout
- Fully Connected Feedforward network：中間有一層hidden layer，最後一層為size等於4的output layer(經過softmax後)，值最大的index即為所預測對應到的relation。

二、結果與分析

參數設定	備註/分析	Kaggle score 結果	
		public	private
1. ***增加重要詞性的字頻*** 2. 1-D Convolution layer：有三種大小的filter (2,3,5)，每種有10個，embedding 大小為100，hidden layer大小為50，batch size=64，epoch數為30，兩次dropout皆為0.5。	由此可知，雖然我們直覺一些詞性對結果會有影響，但可能因為加上這些feature的方式不適當，而破壞句子結構，進而降低performance。	0.600	0.582
與上述方法相同，除了沒有改變詞彙頻率。		0.624	0.618
1-D Convolution layer：有三種大小的filter (2,3,5)，每種有10個，***embedding 大小為180，hidden layer大小為150***，batch size=32，epoch數為20，兩次dropout分別為0.8、0.25。	我們有切validation set調過參數，發現embedding 大小略大於hidden layer大小效果較好。而根據結果得知，embedding 大小越大，效果越好。	0.638	0.634
與上述方法相同，除了embedding 大小為100，hidden layer大小為50。		0.630	0.594
1-D Convolution layer：有***五種大小的filter (2,3,5,8,12)***，每種有10個，embedding 大小為180，hidden layer大小為150，batch size=32，epoch數為20，兩次dropout分別為0.8、0.25。	因為判斷relation的重要關鍵字常常會橫跨兩個句子，因此增加filter大小比較可能使performance提升。	0.642	0.610
與上述方法相同，除了filter 只有三種(2,3,5)。		0.638	0.634
2-D Convolution layer：有五種大小的filter ((2,2),(2,3),(2,5),(2,8),(2,12))，每種有30個，embedding 大小為180，hidden layer大小為150，batch size=32，epoch數為10，兩次dropout分別為0.8、0.25。	2d的convolution layer將每個instane拆成兩維(兩個clauses)處理，由結果得知比1d效果較好，很有可能是因為兩個clauses之間有某種特別的對應關係，若只用一維的filter較難處理這種比較複雜的問題。	0.656	0.676
1-D Convolution layer：有五種大小的filter (2,3,5,8,12)，每種有30個，embedding 大小為180，hidden layer大小為150，batch size=32，epoch數為10，兩次dropout分別為0.8、0.25。		0.636	0.618