Machine Learning Final Project Report

Transfer Learning on Stack Exchange Tags

隊名： NTU_r04942070_進擊的邊緣人

隊長： R04942070 吳冠融

隊員： R03945014 黃柏承
（學號排序） R04942077 許晉維
R05922120 蔡佳睿

Work Division

吳冠融	資料清洗 訓練 Phrases 並調整參數 訓練 Word2Vec 並調整參數 訓練 TF-IDF 並調整參數 撰寫報告
許晉維	測試預測可能成為tag的單字 撰寫報告
黃柏承	測試預測可能成為tag的單字 撰寫報告
蔡佳睿	實作與測試model3 實作experiment3的方法 撰寫報告

Data Preprocessing / Feature Engineering

- 資料清洗步驟：
  - Step 1: 去除 html tags （BeautifulSoup）
  - Step 2: 去除文中的 url （自己寫regular expression）
  - Step 3: 將文章切成句子 （nltk的english tokenizer）
  - Step 4: 去除數字及符號 （自己寫regular expression）
    - 會保留用 - 連接的英文單字，例如：double-helix
    - 會保留「英文開頭、中間有數字」的字，例如：co2
  - Step 5: 大寫轉小寫 （string.lower()）
  - Step 6: 將句子拆成單字 （string.split()）
  - Step 7: 將可能是phrase的字用 - 連接起來 （gensim Phrases）
    - 嘗試過bigram、trigram
  - Step 8: 去除 stopwords （http://www.ranks.nl/stopwords）
- 特徵抽取：
  - word2vec
    - dataset 用 training data + testing data 的 title + content
    - 資料清洗用上述的Step 1~7
    - 維度嘗試過200, 500, 1000
  - TF-IDF
    - 資料清洗用上述的Step 1~8
    - dataset我們嘗試過很多種，目的各不相同：

1. 把單一 domain 的每一道題目的 title 當成一個 document 來算 TF-IDF。目的是根據TF-IDF的大小，直接從title中猜tag，或把這些數值餵給模型，用來預測tag。
2. 把單一 domain 的每一道題目的 title + content 當成一個 document 來算 TF-IDF。目的同上。
3. 把一整個 domain 的 title + content 當成一個 document,（所以總共有7個document）算TF-IDF。目的是找出不同domain中的文字頻率的差異。

Model Description

Model 1: Only TF-IDF

- 主要想法
  - TF-IDF 本來就可以根據 document 之間的詞頻分析，找出 document 中的 keywords
  - 我們的假設是：「在單一篇 stack exchange 的問題中，TF-IDF較高的單字，很可能也是tags」
  - 問題是「TF-IDF 要多高才應該當成 tag？」，我們採用的方法是，根據training data中每篇文章的 tag 的平均數量，來決定最後要取多少個tags。我們統計出，在training data中，一篇文章的tag數量，平均是2.59個，所以我們在test的時候，就每篇文章取大概2~4個tag
- 使用的 Feature
  - 把每一道題目的 title 當成一個 document來算 TF-IDF
  - 把每一道題目的 title + content 當成一個 document 來算 TF-IDF

Model 2: 用word2vec + DNN / random forest 預測一個詞是不是tag

- 說明
  - 在這個model中，我們把「標注一篇文章的tag」拆成兩個步驟：
    - 1. 先找出該領域中有哪些tag
    - 2. 根據一篇文章跟每一個tag的相關程度，預測該篇文章是否應該標注某一個tag
  - 找出一個領域中有哪些tag的方法是：
    - 1. 先用word2vec取得每個字的word vector（這相當於去了解一個詞的性質和意思）
    - 2. 根據training data，我們可以知道一個詞是或不是tag。把是tag的字label 為 1，不是的為 0
    - 3. 把 word vector 當作input，is_tag 當作 output，train DNN或random forest。

        - 4. 用train好的DNN或random forest預測testing data中的字是不是tag
    - 這邊會遇到一個問題：
        - 某個字在某個領域中不是tag，但在另一個領域可能是tag。例如”brain”有出現在cooking領域的文章中，但在cooking中不是tag，而在biology中是。如果我們因為某個字在training data有出現但不是tag，就判斷這個字不可能是tag，應該會有問題。
        - 我們最後的解決方法是：我們只看較常出現的一些字。例如 “describe” 這個字常常出現，而且在training data中都不曾是tag，那我們就推測它在任何領域都不會是tag。
    - Feature 細節
        - 用training data + testing data 的 title + content，經過前述的step 1~7的清洗後，用來train word2vec，得到500維的word vectors

Model 3:

- 主要想法
    - 將所有領域的document標題及內容以vector表示，並train word2vec將所有word也以vector表示，再train model以所有領域的vector預測所有的word，達到以文章預測tags的目的。
- Content的部分
    - 首先將所有各別類別的文章 (title + content) (包含physics) 斷詞，並濾掉stopwords後，做Tfidfvectorizer產生document-term matrix，其中matrix的各個元素為各個term於文章中的tf-idf，接著再用sklearn的TruncatedSVD將matrix做Latent semantic analysis (LSA)，進行semantic analysis降維至1000維的vector space。
- Word2vec的部分
    - 將所有類別的document斷詞，但不remove stopwords。原因是希望保留document中sentence的原始架構，則在train word2vec的時候可以更加精確地找到words之間的相關性。這部分，我們是使用gensim來train word2vec，預設使用skip-gram來implement，而vector則為100維。
- Prediction的部分
    - training data為6個類別（biology、cooking、crypto、diy、robotics、travel）的documents (title + content)。此6個類別中，每個document 皆有對應的tags，將tags的words以train好的word2vec model轉換成vectors，並將其取平均作為document的tags vector。
- DNN的架構設定
    - 定義一個function，其input為第一步驟產生的document vector，而output為tags vector，使用fully connected 的neuron network來train此model，neuron network的structure如下圖，而Loss function為mean-square-error。

```
model = Sequential()
model.add(Dense(500, input_dim=1000, init='normal'))
model.add(BatchNormalization())
model.add(Activation('relu'))
model.add(Dropout(0.2))

model.add(Dense(100, init='normal'))
model.add(BatchNormalization())
model.add(Activation('tanh'))
```

- 在predict test document的時候，輸入physics類別的document vector並predict physics的tags vector，再以此vector使用word2vec model的most_similar() function，即可找出與此vector最相似的word，而最相似的前5個words則為預測出的tags。

Experiments and Discussion

Experiment 1:

- 說明
  - 我們做的第一個實驗，是希望在只使用TF-IDF的情況下，嘗試預測tag，也就是前述的Model 1的方法。
  - 在這個主軸之下，我們嘗試了以下變化：
    - 使用不同的stopword list
    - 使用phrases bigram
    - 使用phrases trigram
    - 抽出的keywords中，只取出是名詞的字
    - 排除掉在training data中常見但不是tag的字
    - 將名詞轉換為複數型態
- 實驗結果
  - 以下列出比較重要的實驗結果

細節	分數
不使用phrases	0.05429
使用phrases bigram	0.09730
使用phrases trigram	0.09877
不使用phrases keyword過濾出名詞	0.08603
使用phrases bigram keyword過濾出名詞	0.10681
使用phrases trigram	0.10803

keyword過濾出名詞	
使用phrases trigram keyword過濾出名詞 排除掉在training data中常見但不是tag的字	0.11159
使用phrases trigram keyword過濾出名詞 排除掉在training data中常見但不是tag的字 將名詞轉換為複數型態	0.11811

- 最後一個實驗結果就是我們這次最好的成績
- 我們發現將名詞轉換為複數型態也有效果，原因可能是蠻多tag是以複數型態作為tag的，但在文章中卻是以單數型態出現。透過這個步驟把一些原本以單數型態出現在文章中的tag，轉換成正確的形式，等於同時減少了錯誤率又增加了正確率，所以分數提高蠻多。

Experiment 2:

判斷成為關鍵字重要的決定因素是，這個單字是否夠專業，也就是說這個單字是否只有高可能性被某個領域所用，如果這項推斷是合理的，那就有可能可以用他來預測一個字的專業程度，並拿來當作是否能夠成為tag的指標。

- 直接從physics的標題及文章中預測tag
  - 目的
    - 希望可以從在不同domain中找出一些可能成為tag的關聯性，並用來當作預測是否成為tag的方法。
  - 想法
    - 當一個單字被轉成word vector後，是否有其中有feature是用來判斷這個單字的跟該領域的關聯，如果此項假設為真的話，那就有可能以這個feature去預測出這個單字跟該領域的相關性，甚至，藉以判斷出此單字在領域的tag的指標。
  - 結果
    - 在使用physics領域來預測的時候，準確率並沒有顯著進步。後來在實驗過程中，可以發現不同domain各個tag雖然是不同領域，但不同領域卻可能擁有相同的tag。因此我們決定把所有domain的tag都當作training set，對於prediction的準確度相信對結果會有顯著的進步，所以先調整我們的dataset型態。

Experiment 3:

model3的部分，DNN在training的時候已經執行了10幾個epoch，其loss卻沒有明顯下降，而以train 20個epoch的model做predict，其結果卻只是一些肉眼看就知道不是tags的words，下圖為physics的第一篇document所predict出來tags，所以最後沒有採用此方法。

```
possible thrower time horizontal created
```

- 改善方法

  此方法在implement的時候沒有將document的words 做phrase轉換，而且只使用document的title和content做word2vec training，所以在做tags vector轉換的時候，會發生tags的word不存在於word2vec model裡的情況，這邊只是將tags進行拆解，若tags為phrase的話，則拆成個別的words再進行word2vec轉換，未來可以加上phrase的轉換，可能會使得model將容易train的起來，但也可能會使得tags不存在word2vec的情況更加嚴重，這部分還需要再思考怎麼進行。

Experiment 4:

- 想法
  - 為了能從其他documents取出合適的tags，以解決若tags不存在原本document的狀況，我們將每篇document找出其最相似的document，並將其找出的tags加入於原本的tags。
- 方法
  - 運用到上面提到的對每篇documents先做tf-idf 後再做LSA 轉換到semantic vector space，再以cosine similarity做相似度計算找出最相似的document，並將document以找出的tags加入原本documents的tags，在對所有tags以tf-idf做排序取最大的前5個terms作為最終的tags。
- 結果
  - 此方法於test data (physics)的prediction上，在kaggle 的public set表現比沒加此方法（單純以原document的title中找tags）還要來得差一些。