ML2016 Final Project

Transfer Learning on Stack Exchange Tags

0. Team name, Members and Work Division

- 隊名: NTU_r05922148_那大家先決定隊名吧
- 組員
  - 許正忞 (r05922148): 組長，會議記錄，Method 1 and Method 6
  - 陳力維 (b04901014): Method 2 and Method 6
  - 王冠驊 (r05942102): Method 3 and Method 5
  - 林家慶 (d05921018): Method 4, Method 5, and Method 6

1. Introduction

這個比賽提供了來自六個不同類別的 Stack Exchange 文章的標題、文本和標籤 (tags)。我們欲對未知 tags 的物理文章進行 tags 預測。

2. Preprocessing and Feature Engineering

2.1. Observe Real Tags

主題	文章數	相異 tag 數	相異二連字 tag 數	相異三連字 tag 數	每題平均 tag 數	每題最大 tag 數
biology	13196	678	189 (28%)	15	2.51	5
cooking	15404	736	177 (24%)	7	2.31	5
crypto	10432	392	163 (42%)	40	2.44	5
diy	25918	734	184 (25%)	18	2.28	5
robotics	2771	231	65 (28%)	3	2.35	5
travel	19279	1645	507 (31%)	75	3.39	5
physics	81926	?	?	?	?	?

在上表簡單統計當中，我們發現以下 3 點:

1. 每個主題只有數百個或一千多個相異的 tags
2. 在所有的 tags 中，2-grams 及 3-grams 占了不小的比例 (25%以上)
3. 每篇文章大多只有 2 個或 3 個 tags，且最多不會超過 5 個

一些其他的觀察:

4. 大多是名詞
5. 似乎不少為複數型態 (以 s 結尾)

我們可以針對這 5 點找出最適合每篇文章的 tags

2.2. Text Preprocessing

2.2.1. 移除 stopwords 及標點符號

我們以'nltk' package 內建的 english stopwords 濾掉文章中不重要的字詞，並以'string' package 的 punctuation 過濾標點符號，只留下有意義的字詞。

2.2.2. 移除 html 標記和數學式子

與 stopword 相同，html 標記和數學式子有極高機率不是一個 tag，尤其是網址類的 html 標記 (例如<a href=""http://physics.stackexchange.com/ questions/1/what-is-spin-as-it-relates-to-subatomic-particles"">)會大幅影響我們對於二連字或三連字的詞頻判斷。故我們在預處理時先把它們刪除。

2.2.3. 詞性標註 (POS Tagging)

經由觀察其他六類主題的真實 tags，我們判斷正確的 tags 大多都是名詞。我們使用 nltk.pos_tag，將每一類主題的整篇文章都各自掃過一遍。除了判斷每一個字在不同文章的不同地方分別屬於什麼詞性 (用在 **Method 3**) 之外，也做了簡單的加總: 計算每一個字在整個類別裡面被當作某一種詞性的次數。並且進一步定義一個字的 noun_ratio 為一個字被當作名詞 ('NN', 'NNS', 'NNP', or 'NNPS') 的比例 (用在 **Method 4, 5** and **6**)。

2.2.4. Tokenization

我們使用 sklearn.feature_extraction.text 的 TfidfVectorizer 來幫已經移除 stopwords、標點符號、html 標記和數學式子的文本進行斷詞。將字串轉化成字詞的 list。

2.2.5. Stemming and Lemmatization

我們使用 nltk.stem.wordnet 的 WordNetLemmatizer 以及 nltk.stem.porter 的 PorterStemmer (或是 nltk.stem.snowball 的其它 stemmers)，將每個字還原成它們的詞根 (stem)，在所有的 Methods 當中都有使用到。

2.3. Feature Engineering

我們根據不同的方法，有不同的 feature engineering。**Method 1** 是 doc2vec；**Method 2** 是將所有字詞轉化為 document appearance (一個字在每篇文章出現幾次) 的向量，將一個字視為一筆 sample；**Method 3** 則是將所有文章的所有字詞分別建立各自的 TF、IDF 等特徵 (因此同一個字可能會出現在多個 samples，因為一個字會再多篇文章出現多次)；**Method 5** 也是將一個字視為一筆 sample，建立多項統計特性。Feature engineering 的細節將在後面的章節做介紹。

3. Model Description

3.1. Method 1: doc2vec & Clustering

這個方法主要是希望透過 word embedding，將所有同一主題之內的所有文章利用 K-Means 分群，並對每一群建立一個名詞字典。最後計算出每篇文章與所屬分群的名詞字典當中最相似的字做為 tags 輸出。

3.1.1. Steps

1. 對 test.csv 中的文章進行 **2.2.** 節所描述的各項前處理。
2. 用處理過的文章建立 200 維的 doc2vec 模型。
3. 用 K-Means 把所有文章的 doc2vec 分為 20 群，對每一群建立一個名詞字典。
4. 根據每篇文章被分到哪一群，計算出這篇文章與這一群的名詞字典中最相似的 5 個詞，把它們作為 tag。

3.1.2. Performance: 0.02

3.1.3. Discussion

這個方法的效果不是很好，一方面可能因為 doc2vec 對文章的 feature 把握不是很準，另一方面可能也因為分群數量太少，這個問題將 **Method 6** 得到解決。另外，這個方法也沒有考慮到 2-grams 及 3-grams，而實際上很多真正的 tag 都是二連字和三連字，這個問題將 **Method 5** 得到解決。

3.2. Method 2: Find General Tagset by Neural Netowrks

我們想要利用類神經網路預測出所有在文章中的字，哪些在是這個類別的 tag 而那些不是。這個方法所建立的類神經網路的輸入是所有文章的所有字詞，輸出是 0 或 1，為二元分類模型。

3.2.1. Steps

1. 對所有主題的所有文章進行 2.2. 節所描述的各項前處理。
2. 接著要決定如何將一個 word embed 成向量。我們採取的作法是，看這個字在文章中出現幾次，如此產生一個維度為文章總數的向量，藉此來表示一個字。舉個例子，假如現在有三篇文章，分別為:
   - " What is spin as it relates to subatomic particles?"
   - " What is your simplest explanation of the string theory?"
   - " Lie theory, Representations and particle physics"

   依照上文，'theory'的向量表示法為 [0 1 1]，'string'的向量表示法則為 [0 1 0]。然而，因為文章數實在太多，無法直接將如此高維的向量輸入我們的類神經網路，故先將此向量表示法做 sparse SVD 得到一個 700 維的向量表示法，再以此當作輸入類神經網路的向量。
3. 建立類神經網路，模型架構如下圖:

   此模型前面的 Dense Layers 都是用 Relu 作為 Activation Function，最後一層的單神經元則是用 sigmoid 當作 Activation Function，Loss 採用 binary cross-entopy。
4. 訓練、驗證，與預測: 我們利用 biology 和 crypto 類別的資料來當作 training set，robotics 的類別作為 validation set，如此是考量到 robotics 與 physics 的主題較相近的關係。接著隨機初始化參數 train 500 次，並使用 Early Stopping Method，然

後將得出的模型去預測 robotics 類別的資料，取前五十個機率最高的字，將得到的 tag set 所有的字取其在答案中出現次數總和，最後保留其值最高的 model。最後我們將最好的模型去預測 physics 的 tag set，得到一個基本的 tag set，但因為這個方法沒辦法包含 2-gram，故我們又做了下一步的處理。

5. 我們將得到的 tag set 先刪掉一些在文章中出現次數過少的，在將這些單詞在經過前處理過的文本中，前後出現次數較多的字，將它們組成 2-grams 放入 tag set。

3.2.2. Performance: N/A (此方法僅做為輔助使用)

3.2.3. Discussion

1. 此方法只能作為輔助使用，但在主要方法加入此方法之後成效並不是很大，似乎沒有對分數有太大的影響。我們認為是由於要找出整個類別的 tag set 這個 task 太難了，往往預測出的 tag 太過於 general，對預測沒有太大的幫助。故先對文章進行分群之後，再利用此方法找出專屬於該群的 tag set 可能會比較有效。
2. 由於 SVD 要求要有一定的文章數和字數才能實作，一開始設的 700 維似乎太高。之後可以考慮調整模型的維度再試試看。

3.3. Method 3: Classifier

我們想從已經有 tags 的 6 個類別文章訓練一個類神經網路模型，利用這個模型，使其在 physics 類的文章同樣能準確預測出每篇文章的 tags。針對每一篇文章裡的每一個字，建立每一個字的特徵，此時同一個字可能會出現在多筆 samples，每出現一次就產生一筆。

3.3.1. Steps

1. 對所有主題的所有文章進行 2.2. 節所描述的各項前處理。
2. 建立每一個字的特徵，例如:
   - Term Frequency (TF)
     - 針對每一篇文章，計算某一個字在這篇文章出現的次數
   - Invese Document Frequency (IDF)
     - 若一個字詞只出現在某一篇文章中，顯然這個字詞對於此文章是十分獨特、具代表性的。為了突顯一個字詞在一篇文章中的獨特程度，我們考慮 IDF
   - 相對位置
     - 表示這個字詞出現在句子的哪個位置，計算方式為 (第幾個字/整篇文章的字數)，範圍在 0~1 之間。

- 是否出現在標題、是否出現在內文
  - 有出現則紀錄 1，沒出現則紀錄 0
- 詞性標註 (POS tagging)
  - 在 POS 這個特徵裡，我們本來想以 one-hot encoding 表示不同詞性，但是當時再存成 pickle 檔時，因為資料太大、存太久而做罷。最後我們改用正常編碼的方式給不同詞性不同的數字。

3. 建立類神經網路，在這裡我們使用最簡單的單層的 NN，activation function 為 sigmoid, loss function 使用 binary_crossentropy。
4. 此模型使用了其他六類主題所建立的特徵資料表作為訓練資料，得到模型之後，直接套用到 physics 類文章所建立的特徵資料表，用來預測每一篇 physics 文章的裡面的字是不是該篇文章的 tags。

3.3.2. Performance: 0.037

3.3.3. Discussion

在實做的時候，這個方法的效果不是很好，主要有兩個問題導致訓練成效有限

1. 資料量太大: 每篇文章中的一個字就會形成一筆資料，導致最終訓練時會有大約 460 幾萬筆資料，訓練時間太長。
2. 抽取的特徵稍嫌不足: 在這麼多字詞且特徵只有 6 個情況下，可能無法完整呈現每個字的差異。

經過組員討論之後，我們得到以下改進方法

1. 我們可以將 POS tag 當做篩選的門檻，這樣可以大幅降低資料量以及提高預測效果。
2. 我們可以再加入一些其他的特徵，像是字母是否為大寫、是否被雙引號" "包覆等等。

其中第二點所提到的一些特徵，將在 **Method 5** 當中以例外一種形式實做。

3.4. Method 4: Frequent Words and Phrases and Fine Tune

這個方法的想法較單純，以簡單統計的手法，找出同時出現在每篇文章的標題與內文的字，直接做為預測的 tags 輸出。另外也考慮了 phrases (僅考慮 2-grams 和 3-grams)。而 fine tune 的意思則是把原本輸出的答案再透過一些簡單的規則，合併較有可能是真正 tag 的連字，刪去較不可能是 tag 的字等等。

3.4.1. Steps

1. 對 test.csv 中的文章進行 2.2. 節所描述的各項前處理。

2. 所有的文章中，我們將相鄰的 2 個字組成一個 2-gram，將相鄰的 3 個字組成一個 3-gram。如果出現的頻率夠高，我們即把它視為此文章的 tag。
   - 對於 2-grams，剔除
     - 總長度< 5 個字母 (包含 hyphen: '-')者
     - 'A-A'樣式者，例如: 'time-time'、'charge-charge'
   - 對於 3-grams，剔除
     - 總長度< 7 個字母 (包含 hyphen: '-')者
     - 包含 2 個以上 stopwords 者，例如: 'up-to-date'
     - 'A-B-A'樣式者，例如: 'step-by-step'、'one-to-one'
3. 步驟 2 所預測出來的結果，包含了大約 17000 多個相異的 tags，遠大於其他六項主題數百至一千多的等級，顯然有太多的錯誤 tags。因此有必要針對此結果進行微調:
   - 若某一篇文章預測出來的 tags 的某兩個單詞或 2-grams，在其它文章的預測 tags 中曾以 2-grams 或 3-grams 的形式出現過，並且出現次數達到一定的標準 (一開始定為 10 次，就有不錯的效果，因此就固定下來)，則直接將這兩個字以它的 2-grams 取代。
     - 例如: "relativity special ..." ⇒ "special-relativity ..."
   - 若某一篇文章預測出來的 tags 包含兩個 2-grams，呈現'A-B'及'B-C'的形式，且'A-B-C'在其他的文章的預測 tags 中曾以 3-grams 形式出現至少一次，則直接將這兩個 2-grams 以合併後的 3-gram 取代。
     - 例如: "field-theory quantum-field ..." ⇒ "quantum-field-theory ..."
   - 剔除 noun_ratio < 0.5 的字。
   - 例如: 'force' vs. 'forces'，'electromagnetics' vs. 'electromagnetism'。這些字只有一個會是真正的 tag。針對這個問題，我們將每個字還原成它們的詞根 (stem)，將同一字根的字以下列兩個規則選出一個字來代表。
     - 若只是單複數的差異，統一選擇複數，例如: 'force' ➔ 'forces'。
     - 其餘則取擁有最高 noun_ratio 的字，例如: 'electromagnetics' ➔ 'electromagnetism'。

3.4.2. Performance

- 步驟 2 (fine tune 之前): 0.09897 (beyond weak baseline)
- 步驟 3 (fine tune 之後): 0.12561 (beyond strong baseline)

3.4.3. Discussion

1. 這個方法雖然簡單，但是效果還不錯。推測主要原因是 physics 類文章的 tags 真的常在標題與內文當中重複出現。
2. 若不考慮 phrases 則 Kaggle 分數只有 0.083 左右，考慮 phrases 之後進步超過 0.01，表示考慮相鄰的字組成 phrases 確實可以讓我們找到一些連字型的 tags。
3. Fine tune 當中使分數進步最高的是前兩項的 n-grams 合併處理，大約進步 0.015。這個步驟除了幫助我們找到更多連詞之外，還有另一個好處是可以提高 precision (因為這個階段大多數的 tags 估計都是錯誤的，把兩個錯的合併成一個錯的，可使 precision 的分母降低)。

3.5. Method 5: Estimate Tag Probability

針對每個主題，萃取每個字在所有文章當中的統計特性做為特徵，建立模型來判斷一個字「是否」為合法的 tag。

3.5.1. Steps

1. 對所有主題的所有文章進行 2.2. 節所描述的各項前處理。
2. 針對每一個主題，各自建立每個字在該主題內的各項統計特性做為特徵，建立二元分類模型來預測每個字是否為合法的 tag。我們建立的特徵如下表所示:

欄位	描述
word	所有出現在某一主題的字(有被算過 noun_ratio 者)
n_occurence	出現的總次數
n_title	出現在幾篇標題
n_question	出現在幾篇文章 (標題或內容)
ratio_capital	字首大寫的比例
ratio_upper	全部字母大寫的比例
ratio_before_ques	出現在問號之前的比例
ratio_noun	被當作名詞的比例
ratio_around_kw_tag	前後兩個字包含與'tag'同字根的字的比例
ratio_around_kw_understand	前後兩個字包含與'understand'同字根的字的比例
ratio_around_kw_study	前後兩個字包含與'study'同字根的字的比例

ratio_around_kw_introduction	前後兩個字包含與'introduction'同字根的字的比例
ratio_around_kw_explain	前後兩個字包含與'explain'同字根的字的比例
ratio_around_kw_principle	前後兩個字包含與'principle'同字根的字的比例
ratio_around_kw_interpret	前後兩個字包含與'interpret'同字根的字的比例
ratio_around_kw_difference	前後兩個字包含與'difference'同字根的字的比例
ratio_around_kw_book	前後兩個字包含與'book'同字根的字的比例
ratio_around_kw_knowledge	前後兩個字包含與'knowledge'同字根的字的比例
mean_occur_in_title	在標題出現的平均次數
mean_position_in_title	在標題出現的平均位置
ratio_in_title_first	出現在標題句首的比例
ratio_in_title_last	出現在標題句尾的比例
mean_occur_in_content	在內容出現的平均次數
mean_position_in_content	在內容出現的平均位置
ratio_in_content_first	出現在內容句首的比例
ratio_in_content_last	出現在內容句尾的比例
mean_occur_in_question	在一篇文章當中 (標題和內容) 出現的平均次數
is_end_with_s	是否以's'結尾
is_end_with_ing	是否以'ing'結尾
str_length	字串長度
is_tag	是否為 tag (只有 physics 以外的其他六項主題才有此欄，為模型的反應變數)

3. 我們以 robotics 所建立的資料表為 testing data，以 biology、cooking、diy、crypto 四類所建立的資料表合併為 training data (捨棄 travel，因為此主題的 tag 統計特性與其他五類差異較大)。在建模之前，我們對於「不是 tag」的字進行隨機取樣，使 training data 平衡 (balanced，tag 與非 tag 的 sample 數一樣多)。我們採用的模型為隨機森林 (random forest)，使用 R 語言的'randomForest' package，以預設的參數 (ntree = 500, mtry = floor(sqrt(ncol(x))) = 5，表示建立 500 顆決策

樹，每個節點隨機考慮 5 個變數)以及 training data 建立二元分類模型，並驗證模型在 testing data 上的預測正確率 (accuracy)。重複上述步驟 (平衡、建模、預測) 10 次，得到的平均正確率為 86.08%。結果顯示上述特徵對於判斷一個字是否為 tag 確實是有用的。我們再利用'randomForest' package 內建的 importance 函式，計算每個特徵對於正確率的重要性 (採用 MeanDecreaseAccuracy，其意義為: 打亂順序後會使得正確率下降多少，越多表示特徵越重要)，下表列出重要性前十名的特徵:

排名	特徵	重要性
1	ratio_noun	49.52
2	n_title	47.45
3	mean_occur_in_question	32.90
4	ratio_capital	32.61
5	ratio_in_title_last	28.82
6	mean_occur_in_content	28.10
7	ratio_before_ques	27.75
8	n_occurence	25.91
9	ratio_in_title_first	24.02
10	n_question	23.68

從上表可以看出，除了較顯而易見的「名詞比例」、「出現在標題的次數(類似 document frequency)」，以及「在一篇文章當中 (標題和內容) 出現的平均次數 (即平均 term frequency)」之外，還有一些有趣的特徵。例如「字首為大寫的比例」、「出現在標題句尾的比例」，以及「出現在問號前的比例」等等。另外，「出現在關鍵字附近」這樣的特徵比較沒有用，可能是因為我們僅考慮前後各兩個字做為鄰居來計算此特徵，因此大多數的字的這類特徵都很接近 0，使得此類特徵比較沒有鑑別度。之後可以考慮擴大定義鄰居的 window 大小，或著甚至把出現在同一句話裡面的其它字都視為鄰居。

4. 由於用 robotics 建立的 testing data 得到不錯的預測正確率，我們就直接將此模型套用到 physics 類文章所建立的資料表上，並將預測的結果形式從原本的二元輸出改為機率輸出 (R 語言語法：predict(..., type = "prob"))。預測出全部 16834 個字是合法 tag 的機率之後 (存成 physics_prob.csv)，就可以應用在 **Method 4** 的 fine tune 當中：將 tag 機率小於某個標準的 tag 拿掉。我們嘗試了幾種可能的標準 (0.5、0.7、0.75)，最後採用 Kaggle 分數較高的 0.75。

3.5.2. **Performance**: 0.13802 (into top-10%)

3.5.3. Discussion

1. 這個方法捨棄了每一個字在「每一篇」文章當中的特徵，取而代之的是每一個字在整個主題當中的一些統計特徵。雖然可能會失去一些資訊，但是優點是 sample 數量大幅降低，方便後續建模及預測。而且效果也還不錯。至少可以幫助我們砍掉更多不適當的 tags。
2. 此方法只試用了一種模型 (隨機森林)，這種 tree-based 的模型是很適合的，因為一個字是否為合法 tag 的條件通常具備以下形式:「如果以 s 結尾，而且字母數又大於 10，而且在標題出現的次數夠多，而且...」正好符合決策樹最終會帶給我們的結果。
3. 未來還可以將此方法延伸到 2-grams 甚至 3-grams。

3.6. Method 6: LSA K-Means clustering

Method 1 及 2 都有嘗試將文章分群，不過僅限於 20 群及 6 群，效果並不是很好。經過討論之後，我們將所有文章分成 500~1000 群。並將 Kaggle 上表現最佳的答案，藉由統計每一篇文章「所屬分群」的所有預測 tags (定義為 **cluster tags**)，找出出現頻率高的 tags，將它們加入此文章的 tags 當中。

3.6.1. Steps

1. 對 test.csv 中的文章進行 2.2. 節所描述的各項前處理。
2. 對 test.csv 每篇文章建立 LSA 特徵 (TF-IDF + SVD，取 200 維)。
3. 使用 K-Means 將所有文章分成 500 群，並且重覆 4 次(每次都設定 n_init=10，取 10 次不同初始化之下最好的分群結果)。因此最終每篇文章都會分別屬於 4 個不同的群。
4. 取 Kaggle 最高分的預測結果，統計每一篇文章「所屬分群」的 **cluster tag**，並以下列步驟決定如何從 **cluster tag** 抓出可能屬於某一篇文章的 tags
   - 首先，我們將 **cluster tag** 當中每個「單詞」出現總次數，乘上該字的是合法 tag 的機率 (來自 Method 5 的 physics_prob.csv)。若該單詞沒有出現在 physics_prob.csv，表示它完全沒有出現在任何一個標題，或是完全沒有出現在任何一個文本，此時直接把出現總次數歸零，捨棄不用。
   - 我們也提高二連字的比重 (直接把出現總次數兩倍)，提高二連字被選中的機率。
   - 接著我們將每篇文章的 **cluster tags** 依照調整過後的比重排序。

- 如果某一篇文章根據上面所有方法得到的預測 tags 為空白 (表示該文章沒有任何一個字滿足出現次數、noun_ratio、合法 tag 機率等條件)，則我們直接選擇前兩名 **cluster tags** 做為預測的答案。
- 其他情況則只選擇 **cluster tags** 當中的第一名加入。

下面舉一個在 biology.csv 上面觀察到的成功例子:

- Example in biology
  - **id**: 5
  - **title**: "Is exon order always preserved in splicing?"
  - **content**: "Are there any cases in which the splicing machinery constructs an mRNA in which the exons are not in the 5' -> 3' genomic order? I'm interested any such cases, whether they involve constitutive or alternative splicing."
  - **tags**: "splicing mrna spliceosome introns exons"
  - **Estimated tags** (原本預測出來的 tag): "order"
  - **Top-5 frequent cluster tags** (數字代表出現總次數): [('exons', 56), ('introns', 56), ('mrna', 14), ('splice', 13), ('sequences', 10)]

從這個例子可以看出，原本對此文章只預測出'order'一個 tag，經由分群之後，我們知道此文章所屬分群當中最常出現的 tags 有'exons'、'introns'、'mrna'等等，都是正確的答案。其中'introns'是完全沒有出現在此文章的標題或文本當中的字，也透過這個方法找到了。

3.6.2. **Performance**: 0.15278 (Kaggle best)

3.6.3. Discussion

1. 前面 Method 3, 4 and 5 都只能針對每一篇文章裡面有出現過的字來挑選可能的 tags，無法找到完全沒有出現在文章的 tags。此方法提供一種解決此問題的可能性。
2. K-Means 的分群結果不是固定的，是帶有隨機性的，因此我們可以利用這點，做多次分群，讓一篇文章屬於多種不同的分群結果。除了提高穩定度之外，也比較符合「一篇文章有多個 tags」這個事實。最理想的情況就是，一篇文章真正的 tags 為"A B C"，而第一次分群結果使得它被分到大多帶有"A"這個 tag 的群，第二次分群結果使得它被分到大多帶有"B"這個 tag 的群，第三次分群結果使得它被分到大多帶有"C"這個 tag 的群。不過，目前這種理想狀況還不是很明顯，也許多次分群要從建立特徵的時候就開始考慮，例如帶有"A"這個 tag 的文章比較容易因為每個字的 TF 特徵而被歸在同一群、帶有"B"這個 tag 的文章比較容易因為每個字的「出現在標題的總次數」特徵而被歸在同一群等等。