

2016 Machine Learning
Final Project Report
Transfer Learning on Stack Exchange Tags

★Team : NTU_b02901001_好多好多柳丁腳踏車

★Members :

b02901001 鍾勝隆, b02901014 王崇勳, b02901018 莊馥宇, b02901071 黃淞楓

★Work Division:

鍾勝隆 : model dicussing 、 sub sub coder 、 writing report

王崇勳 : model dicussing 、 sub coder 、 writing report

莊馥宇 : model dicussing 、 main coder 、 writing report

黃淞楓 : Github maintenance 、 running experiments 、 writing report

★Problem Description :

針對在物理論壇的提問文章，根據文章標題與內容，預測出其屬於的領域或相關主題 (tag)，最後再計算預測的準確性。

★Preprocess :

首先，要先對所有資料作前處理。在 content 的部分都會出現 HTML 的 tag，例如：<p>或是<\p>；另外，因為主題是**物理相關**文章，會出現以 **LaTeX** 格式編輯的數學公式，其中的數學符號會是大量且頻繁出現的資訊，在我們的模型之中，這些資訊並不會被使用到，反而還可能會誤認其為正確的 tag，故得先將他們過濾掉（之後的實驗主要是以其他主題的文章，這點的影響就比較小）。

1. Removing HTML Tags:

(1) Motivation

觀察最原始的文件檔案，可以看到 content 之中充滿了 HTML 的 tag，明顯對於我們建立模型與判斷標籤有不利的影響，必須將它們去除，以得到乾淨的內容。

(2) Implementation

使用 beautifulsoup4，一個專門在網路上爬蟲的套件，根據 html tag 將文章進行 DOM tree 結構的切割，將完整的內文取出，然後把所有的內文相接，刪除所有相關的 html tag，成功把 HTML tag 去除。

2. Removing Math Symbol and Notation:

(1) Motivation

經過上述 Beautifulsoup 處理過的標題與資料，仍然有許多對於後來模型判斷不利之處。例如在 title 與 content 之中，常常會遇到換行符號“\n”，且很多句子中都包含數學表示式，其中的標籤與數學符號都會對模型造成不利的影響，必須清除乾淨。

(2) Implementation

我們首先將原本以換行符號“\n”連結的句子切開，得到多個句子。所有在“\$”、“\$\$”、“`\\begin{equation}`”或“`\\begin{align}`”，與“\$”、“\$\$”、“`\\end{equation}`”或“`\\end{align}`”之間的内容都會被拔除，藉此將所有以這類方式表示的數學符號都刪掉。然而，經過這樣處理後的內文仍然會存在一些夾雜於文字間的數學符號，這部分我們並沒有多做處理。

我們將上述方法實作於 preprocess.py 的檔案中。對於整篇文章中的標題和內文都進行上述的步驟，完成前處理。

★Model :

Model 可以分成兩部分，生成 **bigram** 字典和產生 **feature** 兩部分。

※第一部分:

1. Bigram generation

(1.)Motivation

在 physics 的文章中，含有片語的 tag，在標準的格式上要用“-”連接，如 quantum-mechanics，因此我們嘗試先將文章中的片語取出，做為未來預測 tag 的依據。

(2.) Implementation

使用 sklearn 中的 countvectorizer 取文件檔案中的 bigram，透過設定 ngram_range=(2, 2)，建立出一個 bigram 字典，並取出出現次數前 600 個 bigram features，作為潛在的 tag。

2. Abbreviation replacement

(1.) Motivation

觀察模型的輸出可以發現，某些較長的詞彙有可能會以簡稱的方式出現，例如：
quantum field theory 變成 qft。然而正確的 tag 並不會以簡寫方式出現，因此我們必須將這些簡寫替換成原本的單字(片語)。

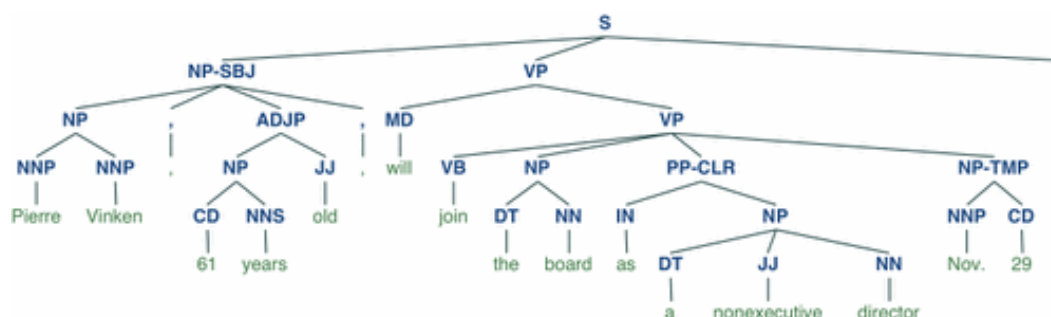
(2.) Implementation

我們之前所建出 bigram 字典中的詞彙，建立一個簡稱與 bigram 間互相對應的字典，法是對於每一個詞，我們都取組成這個詞的每一個單字的第一個字母，組合成一個簡寫，並建立一個 python 的 dictionary 紀錄"簡寫-詞"的對應關係。

最後們回頭掃過整份文件的 title 與 content，找尋屬於潛在詞彙(bigram)的字，透過一次詞句轉換兩個字之間加上“-”；同時在遇到簡稱時將其換掉。

※第二部分(擇一作爲一個 model):

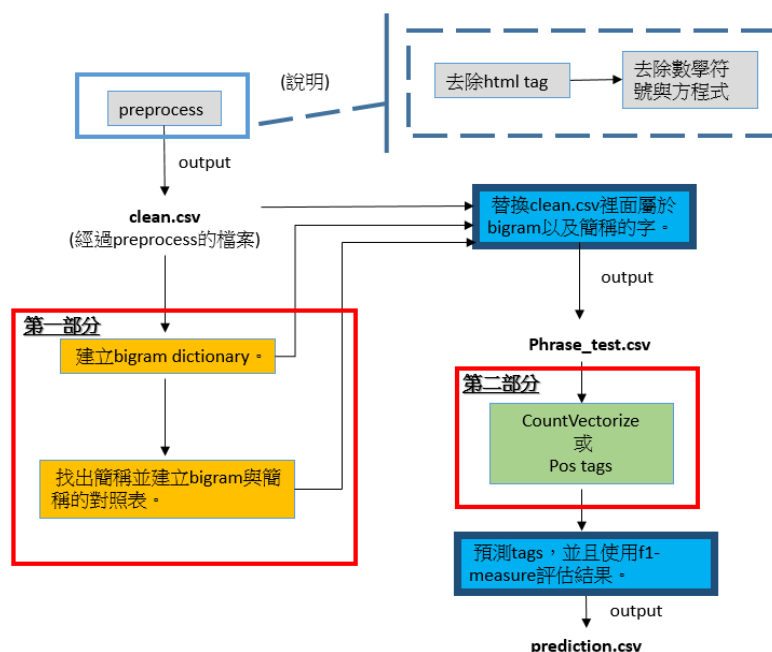
- 將 title 和 content 視爲一篇文章，並使用 sklearn 中的 countvectorizer 計算 **term frequency(tf)**以篩選重要的與 bigram 字句，進而將其選出當作是 tag。設定 min_df=100，代表出現次數少於 100 的字便不將其視爲 tag；而設定 max_df=0.3 代表只要出現次數超過整體字數的三成，便視爲不重要的字，也不會當成 tag。
- 將 title 和 content 視爲一篇文章，使用 sklearn 中的 tfidfvectorizer 計算 **term frequency(tf)** 以及 **inverse document frequency(idf)**以篩選重要的字句，進而將其選出當作是 tag。
- 由於要找出的 tag 幾乎大部分的 tag 都是名詞，因此在使用 bigram 之後，使用 nltk 中的 **Pos_tag** 濾出屬於名詞的字 ('NN'、'NNP'、'NNS'、'NNPS')，處理好後再套用到(a)中的 model。



(圖一) POS_tag 的分類與例子

- 將(b)中的模型加上 **POS_tag**，與(c)同理。

綜合 preprocess 以及 model 的部分，我們可以得到一個流程圖，如圖(二)。



(圖二) Model 流程圖

★Experiments and Discussion：

※Physics (Kaggle)

我們在 Kaggle 上的繳交紀錄主要可以分成四個階段，每個階段在模型中加入了一些新的功能，因此有較大的進步；其他模型上的微調則影響較小，我們在以下的表格中，皆只取每個階段中微調後的最好結果進行呈現。

1. 加入 Bigram

我們在一開始處理這個題目時，單純使用 sklearn 內建的 tfidf，計算每一個單詞的分數，並且取前三名作為每筆測試資料的輸出。然而，這個方法完全忽略了片語的存在，因此表現不佳。因此我們在之後使用 tfidf 同時計算 monogram 與 bigram，並且取前三名作為輸出，結果如下：

	Without bigram	With naive bigram
Kaggle public (best)	0.06455	0.08118

可以看到加入 bigram 後，表現有明顯提升，已經超過 weak baseline。

2. 在文件中連接片語

前一階段的方法雖然可以提升效果，但並不理想。不理想的原因主要在於 **tfidf** 計算 **monogram** 和 **bigram** 時會有一些冗餘的情形發生。舉例來說，在上面的方法我們會同時預測出 **quantum**、**quantum-mechanics**、**mechanics** 三個詞，但實際上只有片語，也就是 **bigram** 才是正確的 **tag**。因此我們在這個階段，先計算出前 600 名常出現的 **bigram**，在原始文件中連接後，再進行 **tfidf** 計算 **monogram**。結果如下：

	With naive bigram	Connect bigram in document
Kaggle public (best)	0.08118	0.09184

可以看到我們事先在文件中處理 **bigram** 的問題後，得到大約 1% 的效果提升。

3. 使用 POS tag 刪除非名詞

在前一階段中，我們模型預測出的 **tag** 包含了一些形容詞與動詞，但通常這些詞並不會是正確 **tag** 的一部分，因此必須將它們刪除。結果如下：

	Without POS tag	With POS tag
Kaggle public (best)	0.09184	0.11999

可以看到這個階段的進步超過 2%，進步幅度相當顯著。

4. 將 content 所有 bigram 加入 tag 中

在這個階段，我們認為所有在資料中出現的 **bigram** 都與資料的主題極度相關，因此我們將一筆資料中所有出現的 **bigram** 都加入預測 **tag** 中。結果如下：

	Without adding all bigram	Adding all bigram
Kaggle public (best)	0.11999	0.12978

結果進步約 1% 左右。

※Other topics

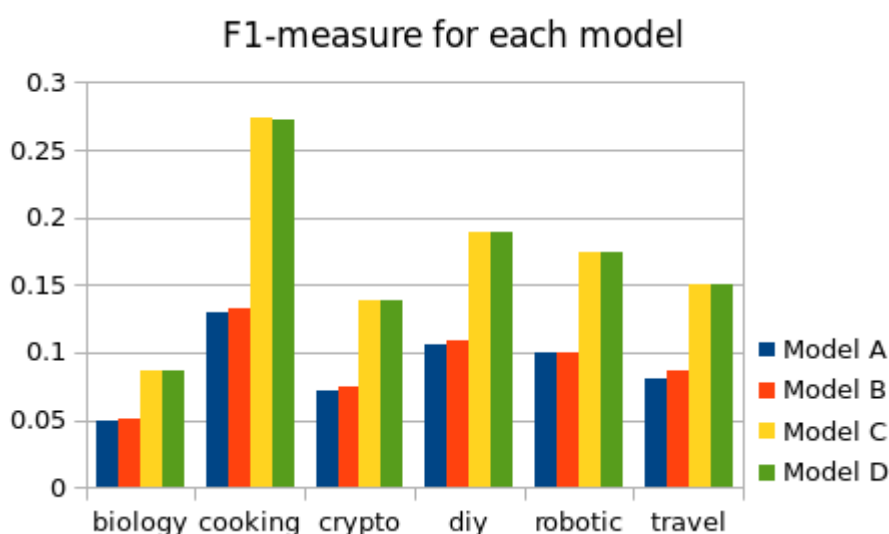
1. Model performance

要測試設計出的 **model**，我們嘗試用不同領域的文章當做輸入，預測出該領域中每篇文章對應的 **tag**，觀察使用不同 **model** 的表現下，預測的結果如何。我們使用的文章為 **biology.csv**, **cooking.csv**, **crypto.csv**, **diy.csv**, **robotic.csv**, **travel.csv**，這六個領域的文章集合。因為每個文章都有提供正確的 **tag**，方便計算其分數，以評估 **model** 的好壞。

每個 model 針對不同種類文章的 F1-measure 分數：
(分數計算方式為各個 id 的 F1 score 以其 tag 個數做加權平均)

category	Model A	Model B	Model C	Model D
biology	0.04901	0.05078	0.08600	0.08601
cooking	0.12958	0.13246	0.27373	0.27247
crypto	0.07165	0.07464	0.13869	0.13838
diy	0.10604	0.10868	0.18915	0.18901
robotics	0.09944	0.10019	0.17358	0.17402
travel	0.08002	0.08643	0.15031	0.15000

經過實驗可以發現 Model C 和 Model D 的效能表現沒有特別的差異，但 Model D 平均都比 Model C 的 accuracy 些微高一點，我們判斷可能是 tfidfvectorizer 中有考慮到 idf 這個參數，透過考慮到 idf 的變化，也把不是 tag 卻常常出現的字，從潛在詞彙中刪除，所以效能表現得比較好。可以考慮用 Ensemble 的方式進行投票，將兩者進行整合，達到更好的效果。



可以從直方圖中，清楚看到 Model C 和 Model D 都明顯優於 Model A 和 Model B，由此可知實際上的 tag 的確幾乎都是名詞的形式，而 POS tag 確實透過過濾掉不屬於名詞的詞彙提升不少準確率。

再來 Model B 都好於 Model A，顯示出在沒有 POS tag 的情況下，透過考慮到 idf 的變化，效能表現得比較好。但在 Model C 和 Model D 中，idf 的效果就不明顯，甚至有些文章種類中效果還比較差，可能的原因是 POS tag 留下的名詞就已經有 idf 期望的效果在，甚至能找出 idf 所沒能留下的 tag，因此 idf 才會失去效用。

【註】：前述的 model A 和 model B 為了不讓過多的單詞被當作 tag，我們只取出現頻率前

600 的作為可能的 tag，超過的一律刪掉。實際上我們也實作了另一種方法是每個 id 只取前 N 常出現的詞彙當作 tag，在下表中以 model A(N) 表示。

	model A	model A(3)	model A(4)	model A(5)
biology	0.04901	0.04468	0.04758	0.04894
cooking	0.12958	0.11078	0.12723	0.13545
crypto	0.07165	0.10716	0.10480	0.10111
diy	0.10604	0.07682	0.08861	0.09619
robotics	0.09944	0.11604	0.12312	0.12487
travel	0.08002	0.04292	0.05632	0.06605

由上表可看出，除了 robotics 之外，大部份的文章種類套用固定 tag 數量並不會比較好，可能的主要原因是因為常見詞彙的前三名文章的“類別 stop words”，也就是像 cooking 可能就會很常出現 cook 這個詞，但本身作為 tag 並沒有意義，而這些詞佔用了一個 tag 就表示減少其他真正可能 tag 被選上的機會，因此普遍成果沒有那麼好。因此我們仍採用是否屬於前 600 常見的詞的方式來選擇 tag。

2. Tag analysis

分析我們的模型跑出來的數量前二十名的 tag 與實際上數量前二十多的 tag 做比較，來判斷我們的模型是否能成功抓取到一些比較簡單的 tag。

(1.)Biology

Real tags(全部共 678 個 tags)

1. human-biology	6. cell-biology	11. physiology	16. entomology
2. genetics	7. bioinformatics	12. zoology	17. homework
3. evolution	8. dna	13. proteins	18. molecular-genetics
4. biochemistry	9. neuroscience	14. microbiology	19. immunology
5. molecular-biology	10. botany	15. species-identification	20. bacteriology

Our tags(全部共 8155 個 tags)

1. dna	6. gene	11. evolution	16. plants
2. cells	7. difference	12. plant	17. bacteria
3. cell	8. protein	13. body	18. genes
4. species	9. human-body	14. blood	19. amino-acids
5. human	10. animals	15. brain	20. water

(2.)Cooking

Real tags(全部共 736 個 tags)

1. baking	6. chicken	11. sauce	16. cheese
2. food-safety	7. storage-method	12. flavor	17. oven
3. substitutions	8. eggs	13. freezing	18. chocolate
4. equipment	9. meat	14. storage-lifetime	19. coffee
5. bread	10. cake	15. vegetables	20. oil

Our tags(全部共 5598 個 tags)

1. meat	6. oil	11.milk	16. pan
2. chicken	7. difference	12. substitute	17. rice
3. food	8. water	13. room-temperature	18. recipe-calls
4. bread	9. cake	14. chocolate	19. olive-oil
5. sauce	10. recipe	15. butter	20. coffee

(3.)Crypto

Real tags(全部共 392 個 tags):

1. encryption	6. cryptanalysis	11. protocol-design	16. symmetric
2. hash	7. elliptic-curves	12. random-number-generator	17. homomorphic- encryption
3. rsa	8. signature	13. diffie-hellman	18. provable-security
4. aes	9. block-cipher	14. keys	19. hmac
5. public-key	10. algorithm-design	15. authentication	20. reference-request

Our tags(全部共 4094 個 tags):

1. encryption	6. security	11.cipher	16. key-exchange
2. hash-function	7. secure	12. attack	17.algorithm
3. public-key	8. hash	13. asymmetric-encryption- schemes	18. hash-functions
4.rsa	9. key	14. message	19. cryptography
5. private-key	10. block-cipher	15.data	20. function

(4.)DIY

Real tags(全部共 734 個 tags)

1. electrical	6. bathroom	11. concrete	16.flooring
2. plumbing	7. repair	12. insulation	17. kitchens
3. wiring	8. water	13. walls	18. switch
4. lighting	9. wood	14. doors	19. shower
5. hvac	10. drywall	15. basement	20. heating

Our tags(全部共 7314 個 tags)

1. water	6. water-heat	11. power	16. circuit
2. wall	7. floor	12. light-switch	17. paint
3. house	8. wood	13. living-room	18. ceiling-fan
4. wire	9. ceiling	14. toilet	19. drywall
5. door	10. switch	15. light-fixture	20. furnace

(5.)Robotic

Real tags(全部共 231 個 tags):

1. quadcopter	6. sensors	11. slam	16. kinematics
2. mobile-robot	7. robotic-arm	12. ros	17. design
3. arduino	8. pid	13. raspberry-pi	18. kalman-filter
4. control	9. localization	14. irobot-create	19. computer-vision
5. motor	10. microcontroller	15. wheeled-robot	20. imu

Our tags(全部共 2259 個 tags):

1. robot	6. arduino	11. create	16. controller
2. motor	7. motors	12. pid-controller	17. inverse-kinematics
3. control	8. kalman-filter	13. system	18. camera
4. quadcopter	9. robotic-arm	14. position	19. robots
5. sensor	10. robotics	15. data	20. sensors

(6.)Travel

Real tags(全部共 1645 個 tags):

1. visas	6. customs-and-immigration	11. luggage	16. indian-citizens
2. air-travel	7. transit	12. tickets	17. europe
3. usa	8. trains	13. legal	18. india
4. schengen	9. passports	14. budget	19. online-resources
5. uk	10. public-transport	15. canada	20. germany

Our tags(全部共 8037 個 tags):

1. visas	6. schengen-visa	11. public-transport	16. days
2. airport	7. time	12. ticket	17. india
3. travel	8. transit-visa	13. europe	18. citizen
4. passport	9. country	14. train	19. flight
5. flight	10. tourist-visa	15. luggage	20. visa-application

從以上表格中可以發現，實際 **tags** 與我們自己預測出來的 **tags** 最大的差異是在數量，都有五倍以上的差距。其原因可能是：第一，我們並沒有處理名詞單複數的問題(有沒有加"s")，例如：預測 **robotics** 的 **tag** 時，前 20 名同時出現了 **sensor** 與 **sensors**。第二，我們辨識出的 **tag** 其實意思重複性很高，像是 **travel** 類別中，其實有很多 **visa** 主題的 **tag** 都應該直接歸類成 **visa** 而不是當作一個新的 **tag**。第三，有些 **tag** 在實際的 **tag** 裡面沒有出現過，但是可能因為在文章中常常出現在相鄰的位置，經過 **bigram** 的計算，最後有機會變成候選的 **tag**。例如：預測 **biolog** 的 **tag** 時，前 20 名出現了 **human-body**，但是直接觀察所有實際的 **tag** 發現 **human-body** 根本沒有出現過。

3. Abbreviation replacement experiment

透過實驗，可以看出雖然不是每個領域都有 **abbreviation** 需要去替換，但只要有 **abbreviation**，就能確實地使預測進步。

category	Model C with abbreviation replacement	Model C without abbreviation replacement
biology	0.08600	0.08600
cooking	0.27287	0.27287
crypto	0.14506	0.13851
diy	0.18924	0.18899
robotics	0.17399	0.17399
travel	0.15018	0.15003

★Conclusion :

可以由上述的實驗，看出我們 **model** 的效能，使用 **POS_tag**，明顯對整體預測 **tag** 的效能有所提升。而分析了 **tag** 之後，我們的方法在面對字意重複的字，並無法有效的融合成目標 **tag**，可能還需要再用 **clustering** 的方式將字詞融合，以及用 **Ensemble** 進行多個 **Model** 的整合。

★Reference :

1. Transfer Learning on Stack Exchange Tags
<https://www.kaggle.com/c/transfer-learning-on-stack-exchange-tags>
2. TfidfVectorizer and CountVectorizer of sklearn
http://scikit-learn.org/stable/modules/classes.html#module-sklearn.feature_extraction.text
3. POS of nltk language toolkit
<http://www.nltk.org/book/ch05.html>
4. BeautifulSoup4
<https://www.crummy.com/software/BeautifulSoup/bs4/doc/>
5. Mean F1-measure
<http://nlp.stanford.edu/IR-book/html/htmledition/evaluation-of-clustering-1.html>