Machine Learning Final Report

2016 Fall -- 2017.01.20

Topic:

Transfer Learning on Stack Exchange Tags

Team name:

NTU_b02901053_Ayou7995 needs girlfriends plzContact（+886）918836002

members:

電機四 b02901053 許義宏
電機四 b02504086 陳品融
電機四 b02901011 趙祐毅

Work division:

許義宏(隊長)：Part-Of-Speech Tagging and Select on title
陳品融 ：Preprocessing & Clustering
趙祐毅：Tf-idf & Conclusion

1.Preprocessing/Feature Engineering：

Preprocessing:

我們對corpus data的前處理共有以下五步驟：

1.1. 使用regular expression：

先把html的tag、latex $或begin/end中夾的數學式子，以及非英文字母的character都刪除，同時也會把以@或&符號開頭的word去掉。

1.2. 濾掉stop words：

stop words set爲助教提供的網站上的stop words與sklearn內建的stop words作聯集。同時在這一步我們會只留下長度介於2到20之間的字，因爲單一character的字通常沒有意義，至於要刪除大於20個character的字則是因爲第一步在使用re作處理時，有可能把原本分開的字黏成一串很長且無意義的字，因此如此的處理是有其必要的。

1.3. Bigram及trigram：

助教有提示，tag可能以phrase的形式出現，為了驗證，我們針對六大主題的答案進行統計，分析其總共有幾個不同的 tag，分別是 monogram, bigram, 或 trigram 的形式出現，統計如下表一。

我們可以從表一看到，在全部的 Tag 裡，幾乎都有超過 100 個 bi-gram，也有數十的 tri-gram，比例來講是高的，因此把 corpus 內的字用 □-□ 連起來形成 phrase 對於這個 task 可說是一個必要的步驟。我們建立 phrase 的方法是使用 Gensim 這個 library 裡的 Phrases。其原理相當簡單，就是把 corpus 丟進去這個 object，而如果某兩個字在 corpus 中常常一起出現的話，最後就會被連成 phrase。至於 trigram 的道理也是相同，只要做完 bigram 後，再做一次 bigram，即可產生出三個 phrase 的字。

Corpus	Number of Items	# of tags	# of mono	# of bi-gram	# of tri-gram
Biology	13196	678	474	189	15
Cooking	15404	736	551	177	7
Crypto	10432	392	187	163	40
Diy	25918	734	532	184	18
Robotics	2771	231	163	65	3
Travel	19279	1645	1053	507	75

（表一）

而設定 Phrases 的參數 min_count 以及 threshold是重要的。所謂的min_count是這個bigram出現的次數一定要大於此值，否則便會被忽略。而threshold則是決定要形成bigram的難易程度，越大代表越少，公式如下

$$\frac{(cnt(a,b)-min\ count)\times N}{cnt(a)\times cnt(b)} > threashold$$

其中a，b是兩個字，N是vocabulary size。由上面的公式我們可看出，這兩個參數會互相影響，因此權衡兩個參數值使得產生出來的phrases比較可能是tag，是一大課題。我們考量的因素如下：盡可能讓越多的字形成phrase，而同時希望不要把一些少見奇怪的字也連起來，因此我們最後設定的參數爲min_count = 50，threshold = 2。

此外，這個部分的trade-off是究竟要先濾 stop words 再作 bigram 還是先作 bigram 再過濾 stop words。如果先濾 stop words 的話，問題是：原本在corpus分開的兩個字，會因爲中間的stop words被濾掉而變成相鄰在一起，接著作bigram便有可能讓這兩個本來不相鄰的字形成phrase，不是我們理想上的結果；然而如果先作bigram再濾stop words的話，bigram就會有一堆無關緊要的字被連起來導致後續處理上的困難。因此在實際試驗的結果下，我們選擇採用前者的作法。

1.4. 建立縮寫與phrases之間的mapping：

由於在觀察其他corpus predict出來的tags時，我們時常發現有些只有2~3個字母，且看起來不像一般英文單字的字卻不斷被預測成tag，讓我們覺得相當奇怪。後來回去原本的corpus檢查，才赫然發覺那些字竟是某些專有名詞的縮寫，例如：cns與central-nervous-system。因此，爲了讓這些縮寫能轉回其原來的形式，使後續的處理與預測能夠更準確，我們必須建立一個mapping的機制。方法很簡單，只要找出所有的bigram與trigram，把他們各部分的字首取出來作爲key，value便是對應到的phrase。建好這樣的map後，我們接著就去把corpus整個跑過一遍，如此一來就能把一部分的縮寫轉成phrase的形式。

1.5. 濾掉非英文單字：

我們使用PyEnchant的dictionary，把除了phrases之外，不是英文單字的字都去除。最後，把長度介於3~30之間的字詞留下，目的是爲了再次確認經過上述一連串process的步驟後，所有的word都不會是太短或太長的怪字。

2. Model Description：

2.1. TF-IDF（term frequency–inverse document frequency)

2.1.1. 動機與想法

在做完資料的前處理之後，我們直覺想到，要使用 TF-IDF 可以找到代表字。

2.1.2. Model 基本形式

我們使用 python library 'sklearn' 的 'TfidfVectorizer'，基本的定義如下：

```
TfidfVectorizer(max_df=0.5, min_df=1, analyzer='word',
token_pattern=r'\b(\w\w+\S\w\w+\S\w\w+)|\w\w+\b',
use_idf=False,stop_words='english')
```

其中，因為在做前處理的時候，已經把一些片語連成 bigram 或是 trigram，如果用原本預設的 token_pattern，這些片語會被去掉，變成單詞。因此我們自行設定 token_pattern，才能保留片語中間的 '-'。

在設定好 vector 的特性之後，我們會先對整個 corpus 做 fit，corpus 裡面包含所有標題和內文的字，fit 的意思是讓這個 corpus 裡面出現過的字，存成一個字典，接下來分別對 title 和 content 做 transform 的時候，可以得到相同長度的 vector，接下來對 title 和 content 的特徵做權重，再取特徵高的前幾名，取出它對應的字當作 tag。

2.1.3. Model 調整與成果估計

Model 有一些參數可以調整，例如：有沒有使用 idf，以及有沒有使用 sublinear_tf (把 tf 的值換成 1 + log(tf))，後續會針對著些可以調整的參數進行實驗。

另外，要輸出幾個 Tag 也是一件可以調整的事情，我們分析目前有答案的類別，觀察每一個類別有幾個答案（如下表）。可以發現每個類別裡面，最高有 5 個 Tags，最少有 1 個 tag。平均介在 2.28 到 3.39 之間。這些數據都能當作我們之後設計實驗的參考。我們也去嘗試把一些出現次數太少的 Tag 濾掉。

Corpus	Number of Items	Min. tags	Max. tags	Ave.tags
Biology	13196	1	5	2.51
Cooking	15404	1	5	2.31
Crypto	10432	1	5	2.44
Diy	25918	1	5	2.28
Robotics	2771	1	5	2.35

Travel	19279	1	5	3.39

這些不同實驗的將在 Experiment 部分進行詳細說明，並討論結果。

2.2. Part-Of-Speech Tagging and Select on title

2.2.1. 動機與想法:

文章的標題，通常含有大量的關鍵字，以利搜尋與迅速瞭解文章大意，因此我們推論能透過title用較少的失誤率拿到tags。此外，名詞（或是專有名詞）通常會具有比較高的代表性，因此tags也有比較高的可能性是以名詞出現，我們利用這個性質來當做篩選字詞的一個方式。

簡而言之，這個model的主要方法是將原資料進行前處理後，把 title 裡的字進行名詞的篩選，將非名詞的字詞拿掉後，生成初步的tag list，接着透過實驗與猜測，我們利用一些優化的方法來提高正確率，來生成這個model的答案。

2.2.2. 想法驗證：

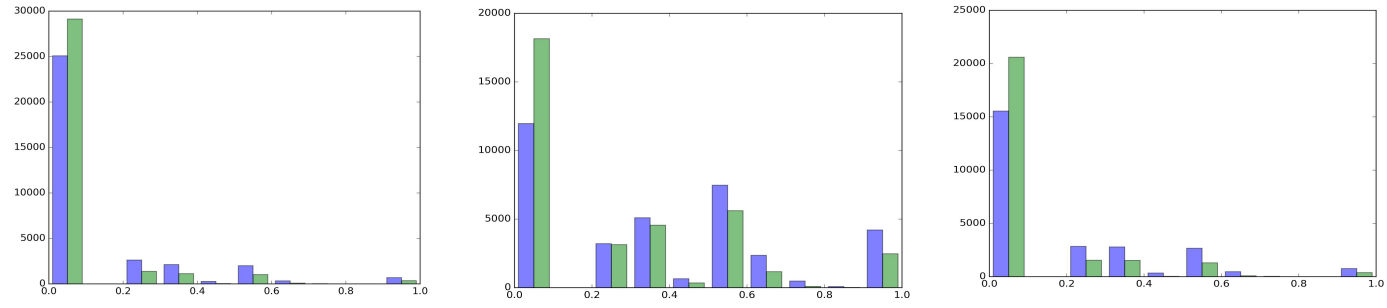
a. 統計title的代表性：

在Kaggle上的討論區，有人經由統計得到tags能在對應的title/content中被找到的比例（https://goo.gl/YLJUDc），例如:若tag 爲“baking cookies texture”，title爲”How can I get chewy chocolate chip cookies？”，則在這個id，tag能在title裡被找到的比例是0.33（即“cookies”的出現）。受到這項統計的啓發，我們對已做完資料前處理的corpus（章節1的前處理）上做了類似的統計，結果顯示爲Figure1，橫軸爲比例，我們以0.1爲級距，1代表全部的tag都能在title或是content裡找到；縱軸則是累計的item數。我們能夠得到兩個結論：

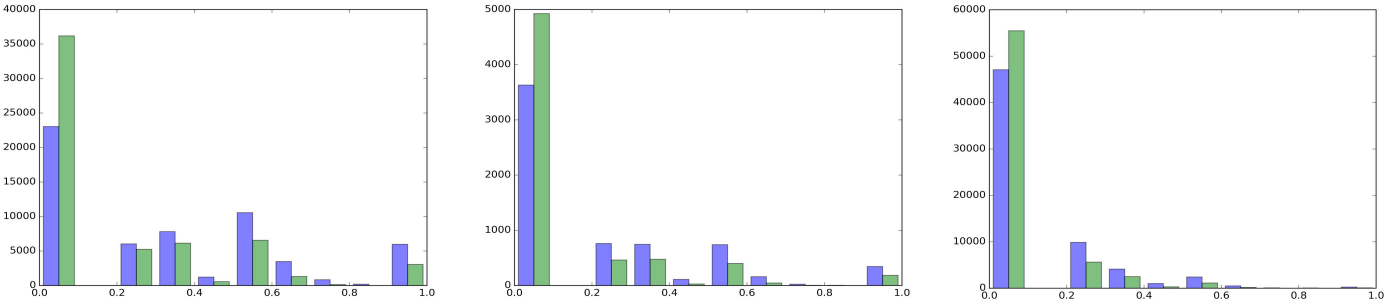
1. 多數的tags是沒有辦法直接在title或是content裡找到的，我想這也是爲什麼多數kaggle上的答案，正確的比例都不太高，多半是受到這樣的限制。
2. 比較了“tag能在title中找到的比例”，與“tag能在title+content裡找到的比例”，我們發現兩者的落差不會太過於巨大，也就是我們針對title過濾tag，我們大概就能拿到大約50%的“可獲得tag”。

再來我們能透過前述的統計知道，content所含有的總資訊量是比較高的，然而卻必須考慮到content本來就有比較多的字，因此我們進行一項統計，想比較title與content平均一個字所含有的資訊量有多少，結果顯示爲Figure2，橫軸爲id，縱軸爲title(或content)中含有的tag數除上title（或content)的總字數，例如：若tag 爲”baking cookies texture”，title爲”how can I get chewy chocolate chip cookies？”則此值爲$\frac{1}{8}$。可以從圖上發現，在title裡，平均每個字所含有的資訊量比較高。

綜合以上兩點的結論，我們驗證選取title來做過濾，是有足夠高的代表性。

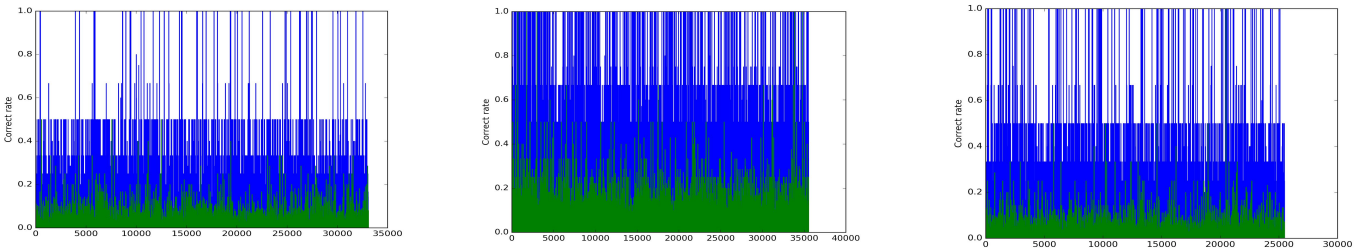


Biology Cooking Crypto

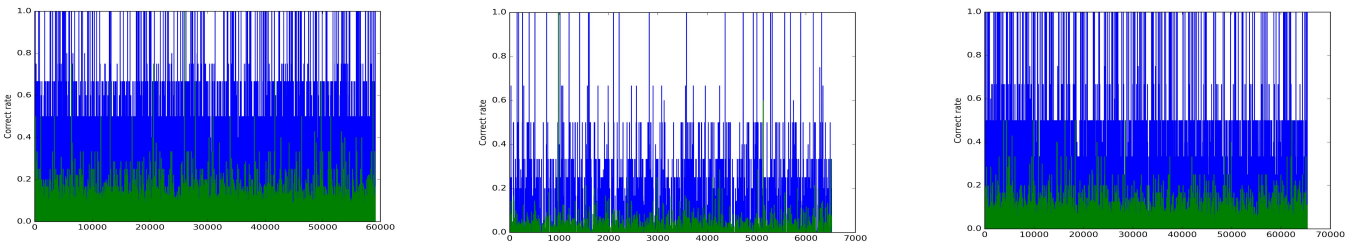


DIY Robotics Travel

Figure 1 .Tag 出現比例分佈
綠色爲在title裡，藍色爲在title+content裡



Biology Cooking Crypto



DIY Robotics Travel

Figure 2. Percentage of correct tag vs words in test/content
藍色的爲title，綠色的爲content

2.2.3. Model的介紹與優化簡述：

最一開始的model是將原資料進行前處理後，把 title 裡的字進行詞性的篩選，將非名詞的字拿掉後，作爲輸出。建立在這個架構下，我們又做了以下幾點的優化。

a. 刪除重複的字
b. 取bigram的詞形
c. 過濾少出現的tags
d. 將空的輸出值填入高機率出現的tag

詳細的介紹將在第三部分與實驗結果一同詳述。

2.3. Clustering

我們另外嘗試了clustering的技巧。方法是把data通過TfidfVectorizer算tf的features，接著用lsa降維並透過kmeans去做分群，最後便會從每篇文章所在的群中選前幾名作爲其tags。而最後實驗的結果是此方法對於我們的kaggle score無法有所提升。以下爲我們將此方法用在其他corpus上所做的分析。

此實驗中，predict tags爲每個cluster取其分數最高的字
A: 所有有重複的true tags中，有多少比例被我們的predict tags所包含
B: 所有不重複的true tags中，有多少比例被我們的predict tags所包含
C: 所有不重複的predict tags中，有多少比例爲true tags
D: predict tags中不重複的總個數

corpus	A	B	C	D
biology	0.13034	0.04130	0.26923	104
cooking	0.35184	0.07065	0.58427	89
crypto	0.19942	0.03061	0.15	80
diy	0.36057	0.06948	0.57955	88
robotics	0.22868	0.09524	0.25	88
travel	0.10082	0.00851	0.16867	83

使用corpus進行cluster

title	A	B	C	D
biology	0.13749	0.04425	0.3	100
cooking	0.35386	0.07065	0.57778	90
crypto	0.31714	0.03827	0.16484	91
diy	0.38264	0.07357	0.6	90

robotics	0.25890	0.11255	0.24762	105
travel	0.16921	0.00973	0.17582	91

使用title進行cluster

由上表的B可發現，我們predict tags所包含不重複的true tags比例實在相當低，所以驗證了爲何clustering的效果不顯著。此外，我們從這個表格也可發現，若只拿title與拿整個corpus去做cluster，其得到的結果是非常相近的，因此這再次說明了只取title是有足夠的代表性。

3. Experiments and Discussion

3.1. Validation

因為一天上傳次數只有 5 次，因此我們嘗試發展一套 Validation 的機制。有答案的 Data 有 biology, cooking, crypto, diy, robotics, travel 六個類別。由於不知道哪個算出來的分數會跟 physics 最接近，我們選擇算出六個類別的分數，並根據每個類別用有的條目數量，將其做 weighted 平均。

Corpus	Number of Items	Percentage
Biology	13196	15.2%
Cooking	15404	17.7%
Crypto	10432	12.0%
Diy	25918	29.8%
Robotics	2771	3.1%
Travel	19279	22.2%

以下圖表畫出實驗幾種方式與 physics 結果的曲線：

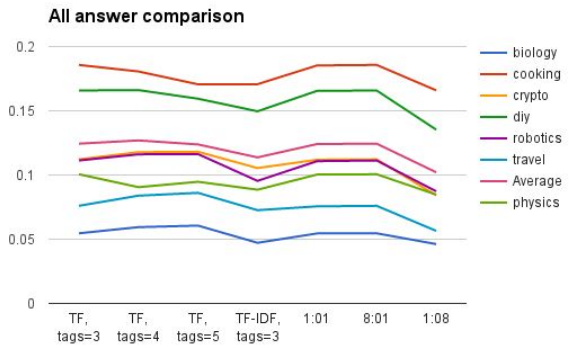


Figure 3. 6類別, 平均, 物理做7次實驗的分數比較
（X 軸代表 7 次實驗，Y 軸代表分數）

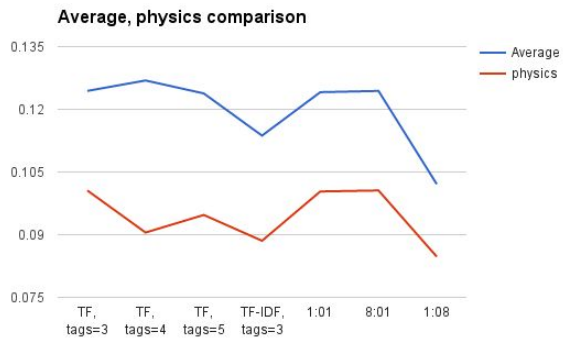


Figure 4. 單獨出平均, 物理兩條線的比較
（X 軸代表 7 次實驗，Y 軸代表分數）

從Figure3可以看到，六個有答案的類別，結果相差非常大，其中，cooking 的結果通常最好，biology 的結果通常最差。

在Figure4我們把 Average 的結果和 Physics 的結果單獨出來比較，可以看到，在某些情形會有類似的趨勢，但在第二欄做 TF, 取四個 tags 的結果，Average 上升，Physics 卻下降。

這樣的結果，表示這種 Validation 的方式不夠準確，但由於沒有想到其他更好的 validation 方法，因此我們會先用這個validation的方法做初步測試，若進步的幅度很顯著，我們會上傳嘗試。

3.2. TF-IDF experiment

針對 TF-IDF 的實驗，我們設計四種不同的方法嘗試。以下詳細介紹：

3.2.1. 給 Title, content 不同 weighting

我們分別對 title 和 content 做 transform，取出他們的 feature，接著去測試不一樣權重得出的結果。由上面的分析可以發現，其他項目的 Tags 數量，平均介在 2.28 到 3.39 之間。我們先用取 3 個 Tags 來做第一個實驗。

這裡我們設定只做 TF，並且都取出 3 個 tag。

以下是實驗結果：

特徵權重 Title : content	分數
8:1	0.10058
1:1	0.10034
1:8	0.08474

從表中可以看到，Title 權重增加的時候，會有比較好的結果，我們猜測是因為 Title 裡面含有 Tag 的機率是比較高的，在 Content 裡面有些字雖然出現次數多，但可能不是 Tag，造成混淆的結果。透過這個實驗，選訂權重比為 8:1，繼續往下做實驗。

3.3.2. 比較 tag 的數量

上面的實驗讓我們知道 Title : content 權重 8:1 的時候較好，但我們想試試看取不同數量的 Tags 會不會有幫助。

實驗設定：只做 TF，Title : content 權重 8:1，取不同數量的 Tags

取 Tags 的個數	分數
3	0.10058
4	0.09051
5	0.09472

實驗結果可以發現，取 4 個 tags 的時候分數會急劇下滑，取 5 個 tags 的時候分數會再上升一些，可以推測，在取第四個 tag 的時候，命中機率很低，取的五個 tag 的時候，會有比較高的機率命中。

3.2.3. 比較不同 vector 參數的設計

實驗設定：只取 3 個 tags，並且 Title : content 權重為 8:1。

vector 設計	分數
use_idf=False, sublinear = True	0.10058
use_idf=False, sublinear = False	0.10058
use_idf=True, sublinear = False	0.08854
use_idf=True, sublinear = True	0.08854

從上表可以看到，有沒有加上 sublinear 這個參數，結果都一模一樣。
加上 IDF 的效果並不好，分數還往下掉，我們推測，可能有些真實的 Tags 出現的頻率偏高，卻因為 IDF 而被濾掉。

3.2.4. 濾掉比較少出現的 tag

做完上述的實驗，我們在想，是不是可以做一些後處理，把一些有蠻大機會不是 tags 的濾掉，於是我們想到：Tag 有一定的代表性，如果在全部答案裡面出現次數太少，有比較高的機率不是 Tag，於是我們把輸出的答案讀進來，測試所有字出現的次數，並訂一個 Threshold，如果出現次數比 Threshold 低，就把它濾掉。
由於要濾掉字，3 個答案太少，濾到剩 1~2 個也不太好，3.3.2 的實驗可以看到，五個 tag 的結果算還可以，比四個 tags 的結果好，如果在濾掉一些不好的 tag，很有機會讓分數往上走。以下是實驗結果：

濾掉 Tag 的 Threshold	# of tags	Mono tags	bi tags	tri tags	分數
原始	17264	15608	1656	0	0.09472
10	5337	3849	1488	0	0.10039
30	2409	1907	502	0	0.10352
100	765	679	86	0	0.09907
300	199	185	14	0	0.07928

從實驗結果可以觀察到，在 Threshold 定為 30 的時候，我們可以得到比原本好約 0.01 的分數，並且 Tag 的數目也從原本的 17,264 濾為 2,409，讓剩下的這些 Tag 的命中率更高。

3.3. Part-Of-Speech Tagging and Select on title Experiment

如2.2.3所述，我們有四次的優化嘗試，以及在資料前處理的變化。

以下詳細介紹概念與實驗結果（實驗成果皆爲實際上傳kaggle的分數）：

3.3.1. 刪除重複的字

我們利用原架構進行實驗，會發現title裡常常會有重複出現的字，因此我們將輸出裡重複的字給刪除，予以優化

實驗結果：

刪除前	刪除後
0.09882	**0.10076**

3.3.2. 取bigram的詞形

我們發現在nltk的pos tag library裡，所有的bigram都會被轉換成名詞，然而使用bigram的意義通常是取得一些複合名詞，或是專有名詞，因此我們針對bigram，去測驗"-"後面的字是不是名詞，如此能刪掉一些不精確的bigram；此外像"-"前的字我們也會檢測他會不會是形容詞或是名詞，這樣的可以過濾掉像是："make-sense"類的bigram。

實驗結果：

只過濾Monogram 非名詞	新增檢測"-"後面的字是不是名詞	新增檢測"-"前是不是形容詞或是名詞
0.10076	0.10625	**0.10719**

以上的測試是先採用只有bigram的corpus，若將corpus加上**trigram與mapping**則能提升到**0.11728**

3.3.3. 過濾少出現的tags

在每個corpus裡，其實tag是有一般性的，也就是說tags通常是會廣泛的出現在不同的id裡，因此我們將我們的輸出進行統計，把未超過threshold的tag從裏面刪除，以除去一些不太普便存在的tag

針對threshold的調整，我們透過以下的實驗數據，決定把threshold設在15

實驗結果：

Threshold	10	14	15	16	18	20	30
Score	0.11981	0.12030	**0.12049**	0.12025	0.12003	0.11973	0.11673

我們發現在過濾的程度太過誇張時（例如30）得到的效果反而會比原本還差，但若在良好的設定threshold的情況下，善用tag的普遍性能得到很好的提升

3.3.4. 將空的輸出值填入高機率出現的 tag

在經由以上的優化後，我們會發現，有一些id的output會是空的，由於空的代表完全沒有進行預測，爲了避免這種情況，我們從3.2.3的統計中，找出前三名的tag填入空的output裡。

統計出來的前三名爲：“ quantum-mechanics”“energy” “mass”，利用相同的理由，考慮tag在corpus的一般性，我們嘗試在了幾種填空的組合，實驗結果如下表：

Type：	都沒有填入	三個皆填入	只填入 quantum-mechanics	只填入 energy	只填入 mass
Score：	0.12049	0.12119	**0.12141**	0.12057	0.12053

我們發現，在四個實驗數值的比較裡，只填入quantum-mechanics的效果最顯著，而填入energy或mass則只上升一些，這樣的結果符合預期，因爲quantum-mechanics是在原先tag裡是最多的，再來依序是energy與mass，因此上升的幅度會隨之下降。

有這樣的經驗，我們又做了別的嘗試，我們將所有的答案都initial先填入一個答案（quantum-mechanics 或 energy 或 mass)，試看看這些答案在全部的corpus的表現如何，實驗結果如下：

Type：	都沒有加入	全加入 quantum-mechanics	全加入energy	全加入mass
Score：	0.12049	**0.13635**	0.10462	0.10078

意外的結果是全加入quantum-mechanics的結果獲得顯著的提升，顯示quantum-mechanics在tag的分佈是十分廣泛的，而其他全加的結果下降也是可以預期，因爲把原本回答到正確答案的比例拉低。

綜合以上的優化與嘗試實驗，我們的最佳上傳結果即是0.13635採用全加入quantum-mechanics的結果，只使用純粹的優化，則是以0.12141爲最高，透過一系列對於tag與corpus的分析，獲得一個還“不錯”的答案。

4. Discussion:

我們去估計，前處理完之後，把所有文章的 Title 都混在一起，做成一個 title pool，去看所有的 tag 有出現在 title pool 的比率。

Corpus	% of all tags in title pool	% of all different tags in title pool
Biology	58.1%	47.9%
Cooking	85.0%	70.2%
Crypto	54.4%	26.0%
Diy	88.1%	72.1%
Robotics	56.1%	45.8%
Travel	46.7%	20.4%

上述估計發現，如果用整個 title pool 去估計，幾乎可以涵蓋一半以上的 tag，代表用 title 去估計的方法還有發展空間，只是可能不能限定只在自己的 title去找。

Conclusion

本次題目的目的，是要從Title 和 Content 之中獲取 Tags。在第一部分，我們分析其他六個類別的答案，從前處理下手，把一些干擾的字（如 html 的字）去掉，加上 Phrase (bi-gram, tri-gram)，並且用字典濾掉非英文字，去除資料的干擾因子。

第二部分，我們設計三種取 Tag的架構：TF-IDF, Part-Of-Speech Tagging and Select on title（只從Title 裡面找並分析詞性），和Cluster。分別描述這三種架構的發想，也做一些分析以驗證想法。

第三部分，我們先說明了嘗試製作Validation的過程，並且分別描述TF-IDF, POS tagging 實驗的設計及結果，其中，使用 POS tagging 並搭配濾掉少出現的Tag，並全部加上常出現的Tag (quantum-mechanics)，在Kaggle 上可以得到0.13635的分數，是我們目前實驗的最高分。

但是，這樣的結果距離準確預測仍相當遙遠，未來我們也許可以嘗試融合上述三種 Model，做出更好的預測。

References

1. sklearn documentation: TfidfVectorizer
2. matplotlib documentation: pyplot
3. python documentation: re
4. python documentation: subprocess
5. nltk documentation
6. gensim documentation: Phrases
7. python documentation: collections
8. Kaggle kernel: Transfer Learning on Stack Exchange Tags