Report

INTRODUCTION

題目，Transfer Learning on Stack Exchange Tags

組名，NTU_哈根大隻

組員，林俊佑、胡凱文、張家瑋、孫盟強

此篇題目想要藉由多個不同種類的論壇，去預測新種類的論壇文章的 TAG，因此目標想要得到一個有效的 Transfer Learning Model。在此篇題目我們利用 Word-to-Vector 以及 TF-IDF，等 NLP 處理方式，去得到較高的預測分數。並嘗試建立 Transfer Learning Model，不過效果不如預期。接下來將會分別介紹並分析此兩種方法。

PREPROCESSING

在兩種方法上，一開始就需先處理文章雜亂不堪的問題，因此我們藉由正則表示法，有效處理，HTML Tag、Code、http 連結，等不這麼直接影響文意的字符。

- HTML Tag： 、 、<lo>….，直接消除。
- Code：<code> … </code>，連同內文一併消除。
- Http 連結：http://....，整串消除。
- Formulation：$...$、&…、\...，等以數學表達的式子，直接消除。
- Number：0000，等純數字，直接消除。

接著進行大小寫轉換，並且抓出 Bigram/Trigram 等字的可能性，因為 Bigram /Trigram 在此題目是非常重要的，藉由其他種類的文章，我們發現大約有 2/3 的 Tag 是屬於 Bigram/Trigram。並且我們在預測 Tag 上有無放 Bigram/Trigram，可以增加快 50%。

在 Bigram/Trigram，我們目前是手刻來進行抓取 Bigram/Trigram，以下是步驟。

1. 進行字與字的串聯，若遇到 Stop word 則取消，並重新開始串聯。
2. 計算串聯字的出現次數。
3. 計算串聯字同時出現的最大機率，例如 Stop Word 此串聯後出現次數為 5，而 Word 出現次數為 10；Stop 為 5，則同時出現機率為 100%。
4. 把串聯字出現次數與機率相乘，排序後抓出前 50~150 的字。(稍後會分析)

schr-dinger-equation	double-slit-experiment	cross-sectional-area	biot-savart-law
stress-energy-tensor	van-der-waals	pauli-exclusion-principle	quantum-field-theory
newtons-third-law	klein-gordon-equation	black-holes	magnetic-fields
electric-fields	kinetic-energy	quantum-mechanics	angular-momentum
black-holes	big-bang	time-dilation	hilbert-space
special-relativity	event-horizon	general-relativity	string-theory

圖(一)，為抓出的串聯字。

MODEL DESCRIPTION

(1) Word-to-Vector with TF-IDF

此方法目標想利用 TF-IDF 抓出價值比重較高的字，想法為 Tag 會廣泛地出現在各篇章中。因此藉由 TF-IDF，應可有效抓出文章的重要單字，並以此基礎去推定其為 TAG。另外使用 Word-to-Vector，是想補足 TF-IDF 的不足，因為對於單一文章並非所有 TAG 都會出現〈此可以藉由其他種類的 Training Data 發現〉。所以藉由 Word-to-Vector 的特性，把文章轉化為一組 Vector，並以此找出相近字，此時可以得到沒有出現在文章內的相近字，進而增強猜到 Tag 的機率。整體方法如下。

1. 訓練 Word-to-Vector Model。(訓練資料，同時用有轉成 Bigram 和無)
2. 計算字在每篇文章的 TF-IDF，並排序。(以下文章是指包含 Title、content)
3. 計算此文章的 Vector，並找出相近字。(相近字只針對 Bigram/Trigram)
4. 挑出前 5-10 高的 TF-IDF 字，並加入找到的 Bigram/Trigram 當作此文章的 Candidate Tags。(因為我們認為 Bigram/Trigram 較重要)
5. 把每個 Candidate Tag，找出其成為複數的可能性。(單純看整個論壇的文章其單複數，出現的比例次數)
6. 最後刪除出現次數太少的 Tag。

以上為，整個 Method 所採取的方式，可以看見此方式中每個步驟可以互相獨立拆開，因此稍後會分析，每一項步驟以及參數選擇對於效能之影響。

(2) Zero shot learning

此方法目標想利用 Neural Network 以及 Word-to-Vector，來達成 Transfer learning，想法為在 Training 的時候，Input 為每一篇 Document 的 Embedding Vector，Output 是這篇 Document 的 Tag 的 Embedding Vector。整體上就是一個 Regression 的問題。我們的期望是，Neural Network 在 Training 完成之後，能夠自動學習到 Document 至 Tag 的映射關係。在使用上，我們用的是 Gensim 的 Word-to-Vector 來實作 Doc Vector 以及 Tag Vector。以下為方法細節。

- **獲得 Document Vector 及 Tag Vector**
  - 首先，先清除掉每一篇文章 Title 和 Content 中的特殊字元，再將全部轉成小寫英文，接著將 Title 以及 Content 接在一起形成當作一篇 Doc。再來，利用 Gensim 的 Word-to-Vector 將每一篇 Doc 轉成 100 維的 Embedding，此時，丟到 Word-to-Vector 裡面 Train 的 Document 包含所有 topic。接下來，在 Document 上，一篇 Document 扣掉 stop words 後所有 Word 的 Vector 都加起來之後平均，產生出 Doc Vector，在 tag 上也是一樣，將所有 Tag 的 Vector 加起來平均，產生 tag vector。
- **Training**

- 在建置 Neural Network 這個部分，我們試過很多種的 Hidden Layer 數以及其 Node 數，還有不同的 Activation Function，最後唯一能 Fit Training Set 的只有使用 RELU 當作 Activation Function，最後，我們的 NN 架構為三層 Hidden Layer、Node 數分別為 5000、2500、1000，Training Set 為扣掉 Validation Set 後剩下的所有文章。

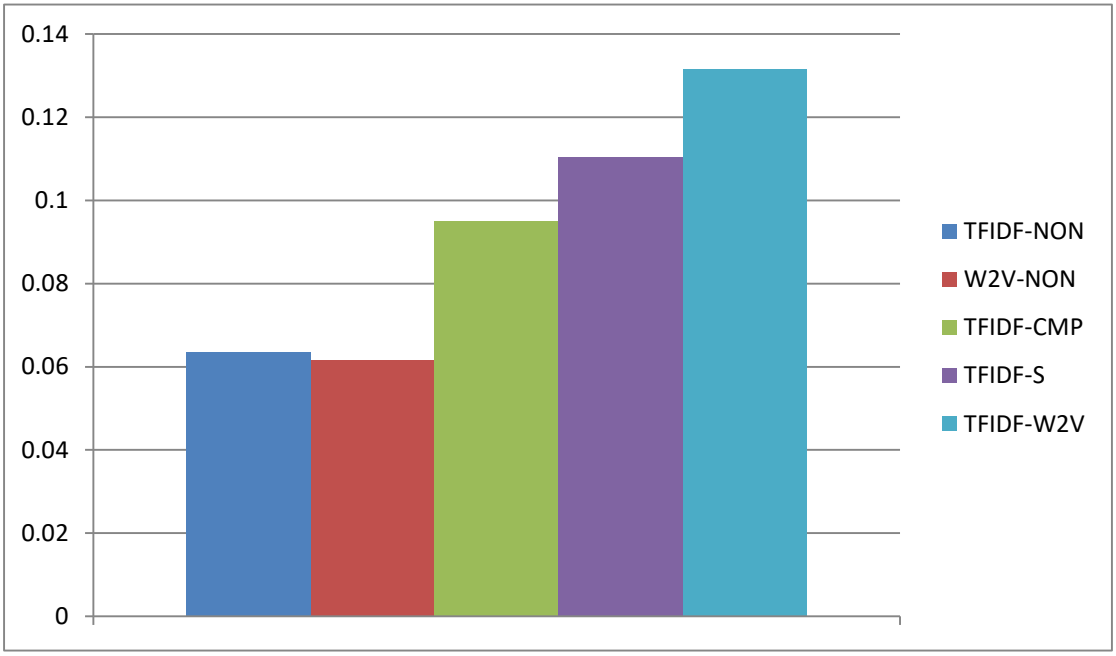
- **Testing**
  - 將 Testing Set 的 Document 丟到上一步 Train 好的 NN 來獲得 Output Layer 那一百維的 Vector，利用這個 Vector 從第一步 Word-to-Vector 的 Model 中找到最相似的五個字來做為這整篇 Testing Document 的 Tag。

EXPERIMENTS AND DISCUSSION

(1) Word-to-Vector with TF-IDF

接下來將會分別討論，不同處理方式以及不同參數對於預測的 Performance 所造成的影響。

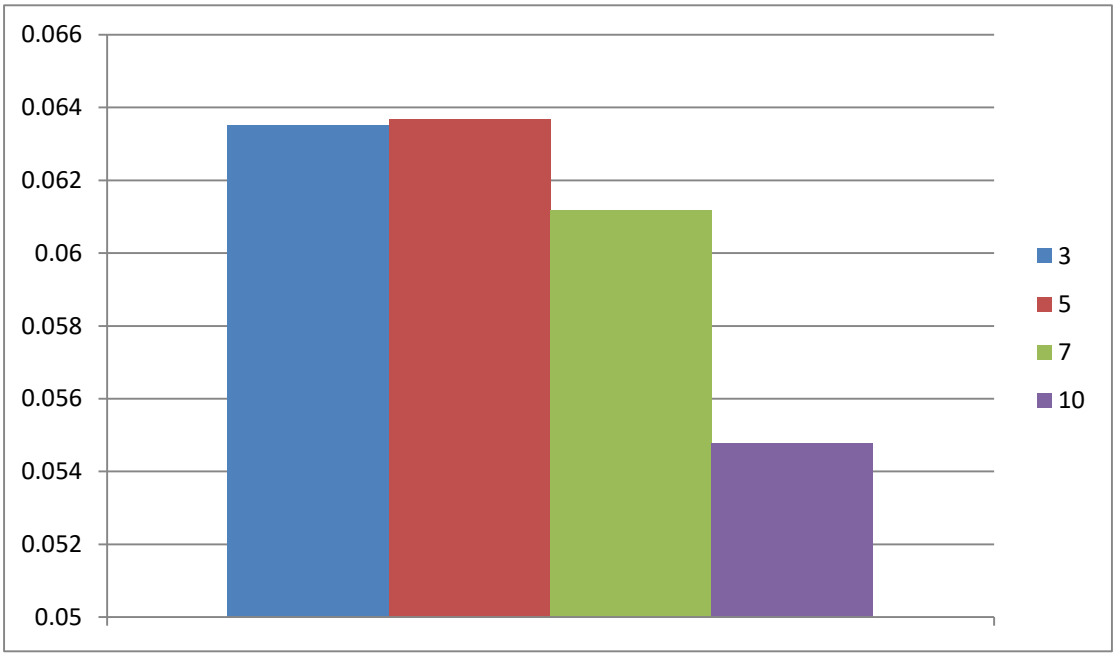
- **各步驟之影響**



TFIDF-NON 代表，只單純使用 TFIDF 並且沒做任何處理；W2V 也是。
TFIDF-CMP 代表，多加入 Bigram/Trigram。
TFIDF-S 代表，多考慮複數的情形。
TFIDF-W2V 代表，多 W2V 找出有可能的字，並去除低頻率的 Candidate Tags。
從上表可以看出，每多一個步驟就有大幅度的提升，尤其多額外 Bigram/Trigram 時，上升幅度更可以達到 50%，因此找到 Bigram 此件事情是非常重要的。所以在結合 W2V 時，我們只針對 Bigram 的相近字。另外複數也是躍進的重點。最後的重點就是去除低頻率的 Candidate Tags，也就是犧牲猜對的次數，以避免過

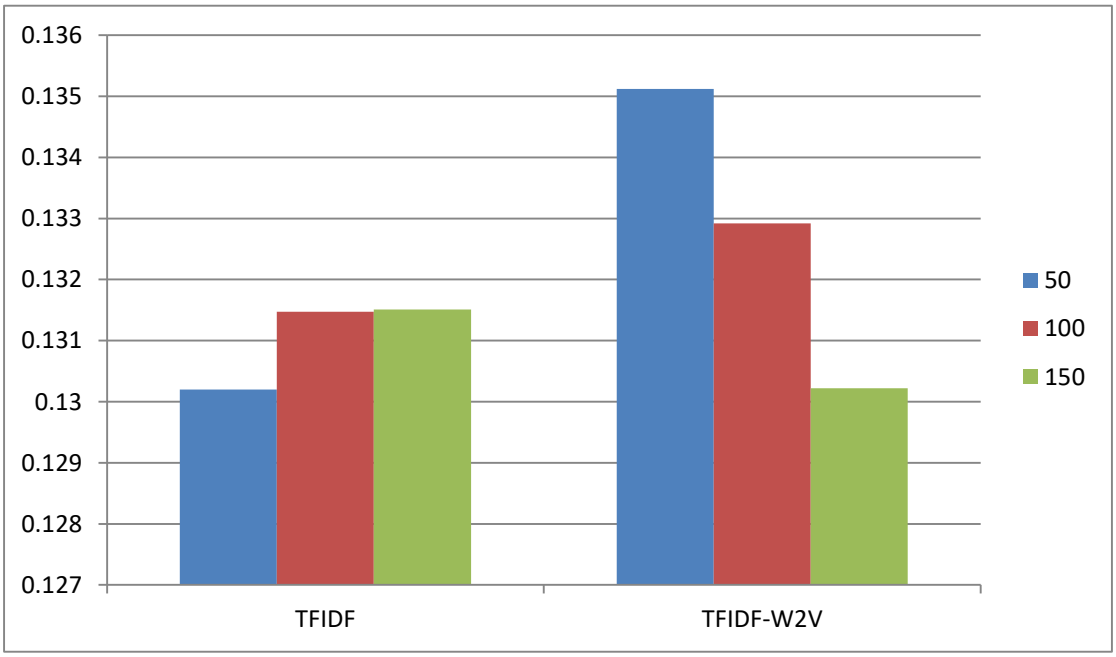
多的猜錯，因此提升精確率，最終提升整體預測結果，稍後會進行討論。

- **TF-IDF，取前幾高的影響**



此實驗，只單純使用 TF-IDF，並且並無任何 Bigram、複數、刪除，等動作，可以發現當 TF-IDF 對每一篇文章取太多的 Candidate Tags，預測結果會愈來愈下降。因為愈多的 Candidate Tags，即便有機會增加猜對次數，但是因應先前所述，Tag 通常為文章中的關鍵字，也就是 TF-IDF 分數較高，所以 TF-IDF 排名愈高，Tag 機會也就愈高。所以取太多 Candidate Tags，就會更大比例的增加猜錯次數，造成精確度下降。

- **取 Bigram/Trigram 的字數**



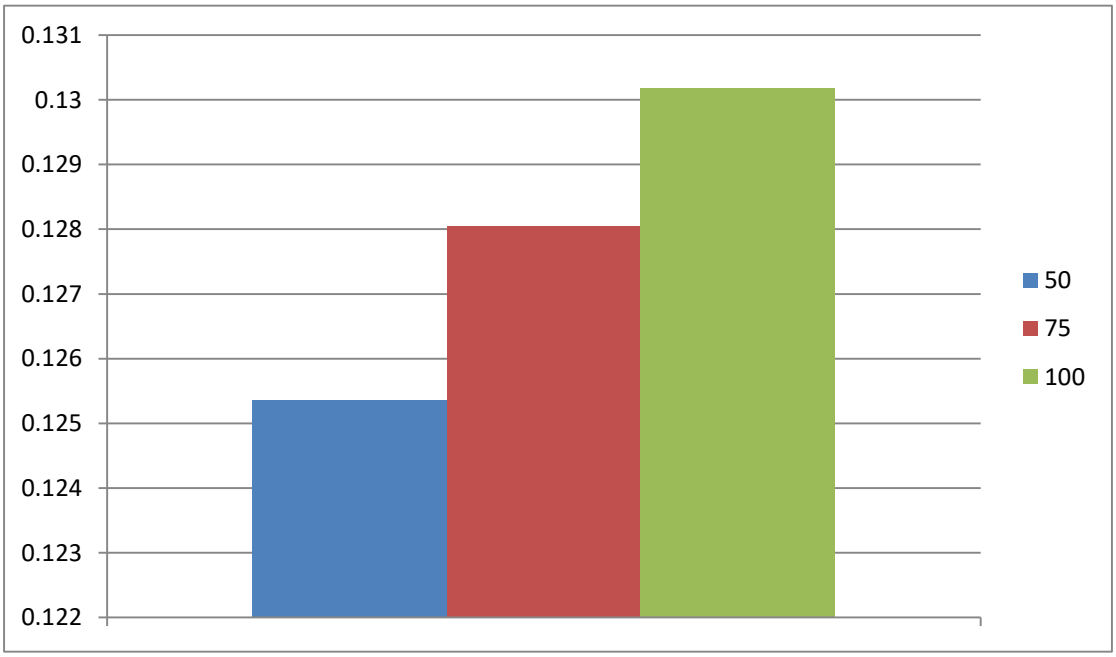
此實驗想要探討，取不同的 Bigram/Trigram 的字數，因為我們找 Bigram/Trigram 方式是，算出此串聯字成為 Bigram/Trigram 的分數，並取前幾高的作為 Compound Words，所以取不同的參數就會有不一樣的 Compound Words List。在這邊可以預想的到，取愈多代表猜對次數愈高，但猜錯次數也會上升。我們實驗對象有兩個類別，一個是使用 TF-IDF，另一個使用 TF-IDF 結合 W2V 找出相近字的能力。我們可以發現，兩者的 Pattern 其實是不一樣的。

我們先討論單純使用 TF-IDF，可以發現，隨著取的字數上升，會造成分數隨之上升。但是這代表說，我們的串聯字排序結果，並非愈前面就會是愈有機會成為 Bigram Tag 嗎？其實不然，經過我們檢查之後，我們發現串聯字排序愈前面，就愈有機會成為 Bigram Tag，但是，因為我們計算方式，除了出現次數，也一併考慮同時出現機率。這代表說，其實排序在後面的正確 Bigram Tag，也就是沒取到的 Bigram Tag，其出現次數可能高過於前面選擇的 Bigram Tag，因此沒答對率大於答錯率，造成選愈多的 Bigram Tag，分數隨之上升。

接著討論使用 TFIDF-W2V，此時分數卻是隨之下降。因為 W2V 會找出相近字，所以剛剛所說的次數推論，就符合此目前的 Pattern。因為，排序跟出現次數並沒有相對關係，這在使用 TF-IDF 就會造成問題〈因為文章有此字才會選其為 Candidate Tags〉，但是經過 W2V 找的相近字，出現次數就與找到的 TAG，並無太大關聯，所以造成此 Pattern 結果。

另外，藉由此圖，我們也可以發現，在使用 TF-IDF with W2V，在某些情況下，會高於純使用 TF-IDF，因此 W2V 來找出相近字，是有其用處的。

- **刪除低頻率 Tags 之影響**



此實驗想要探討刪除不同的頻率 Tags 之影響，也藉是當我們找出每一個文章的 Candidate Tags 之後，會進一步找出每一個 Tag 的在 Candidate Tags 之出現頻率，

當低於 50、75、100 次，會造成預測分數之變化，很清楚可以看見，當刪除愈多的低頻率 Candidate Tags，分數就會持續上升。此結果也很容易明白，因為愈低頻率的 Candidate Tags 代表是真正 Tag 的機率會下降，而且若他不幸為真正的 Tag，但因為他的頻率較低，因此有此 Tag 的文章也較少，所以造成答對率上升幅度比答錯率幅度來的小，所以精確率會下降，預測分數也下降。

(2) Zero Shot Learning

我們把 robotics.csv 當作 Validation Set，從下圖可以看到，雖然我們可以在 Training Set 上 Fit 很好，但在 Validation Set 上的 Loss 卻沒有下降，而且還有上升的趨勢。

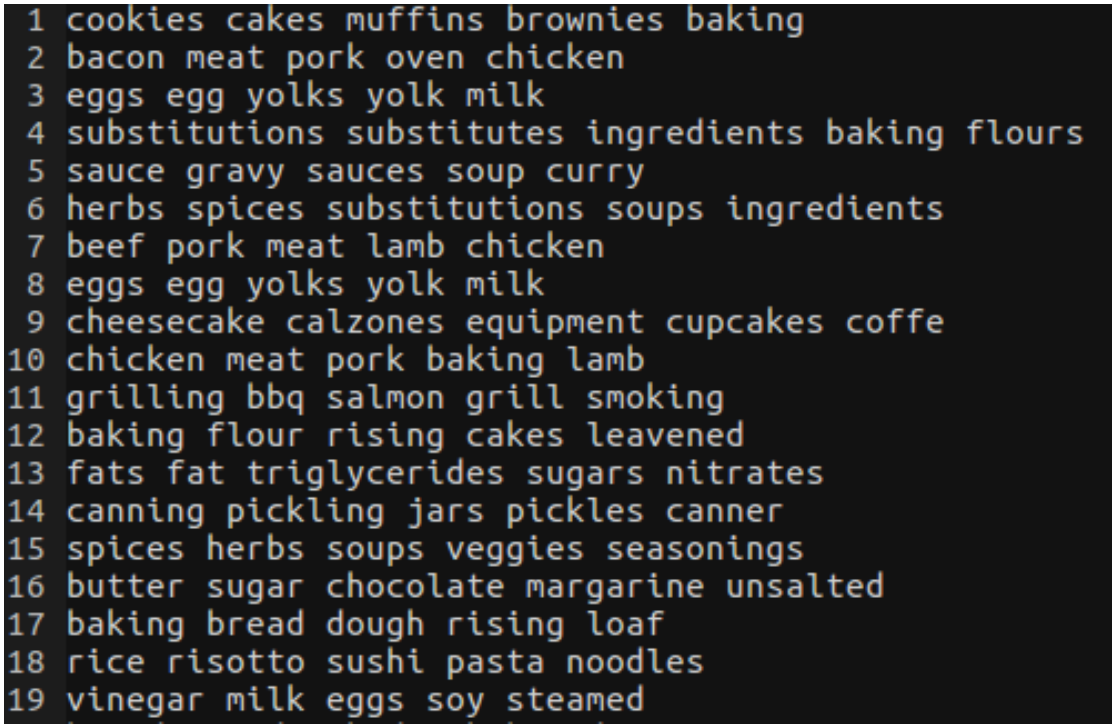
```
84229/84229 [==============================] - 7s - loss: 1.0113 - val_loss: 1.5409
Epoch 2/10
84229/84229 [==============================] - 6s - loss: 0.9148 - val_loss: 1.5381
Epoch 3/10
84229/84229 [==============================] - 6s - loss: 0.8690 - val_loss: 1.5536
Epoch 4/10
84229/84229 [==============================] - 6s - loss: 0.8267 - val_loss: 1.5759
Epoch 5/10
84229/84229 [==============================] - 6s - loss: 0.7794 - val_loss: 1.6221
Epoch 6/10
84229/84229 [==============================] - 6s - loss: 0.7241 - val_loss: 1.6375
Epoch 7/10
84229/84229 [==============================] - 6s - loss: 0.6586 - val_loss: 1.6606
Epoch 8/10
84229/84229 [==============================] - 6s - loss: 0.5899 - val_loss: 1.6534
Epoch 9/10
84229/84229 [==============================] - 6s - loss: 0.5197 - val_loss: 1.6765
Epoch 10/10
84229/84229 [==============================] - 6s - loss: 0.4579 - val_loss: 1.7138
```

圖(二) Loss Rate 之變化

從下圖可以看到，我們確實可以從 Training Set 的 Document Vector 通過 NN 獲得的 Tag Vector 還原出接近答案的 Tag。下圖為在 Training Set 上的結果，圖三為 Ground truth ，圖四為 NN 的 Output。

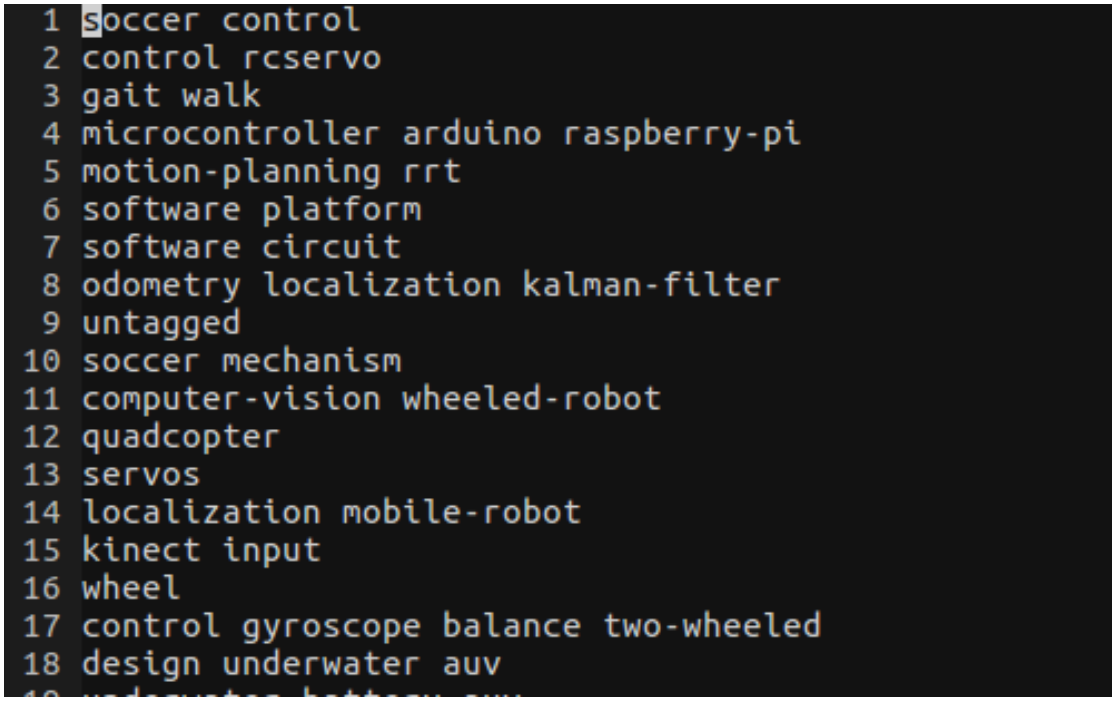
```
 1 baking cookies texture
 2 oven cooking-time bacon
 3 eggs
 4 substitutions please-remove-this-tag baking-soda baking-powder
 5 sauce pasta tomatoes italian-cuisine
 6 substitutions herbs parsley
 7 food-safety beef cooking-time
 8 eggs basics poaching
 9 ice-cream
10 baking chicken cooking-time
11 grilling salmon cedar-plank
12 baking flour measurements sifting
13 storage-method storage-lifetime fats
14 canning pressure-canner food-preservation
15 spices resources basics learning
16 food-safety storage-method storage-lifetime butter
17 baking bread dough
18 rice italian-cuisine risotto
19 eggs food-science vinegar poaching
```

圖(三) Training Set Truth Tags

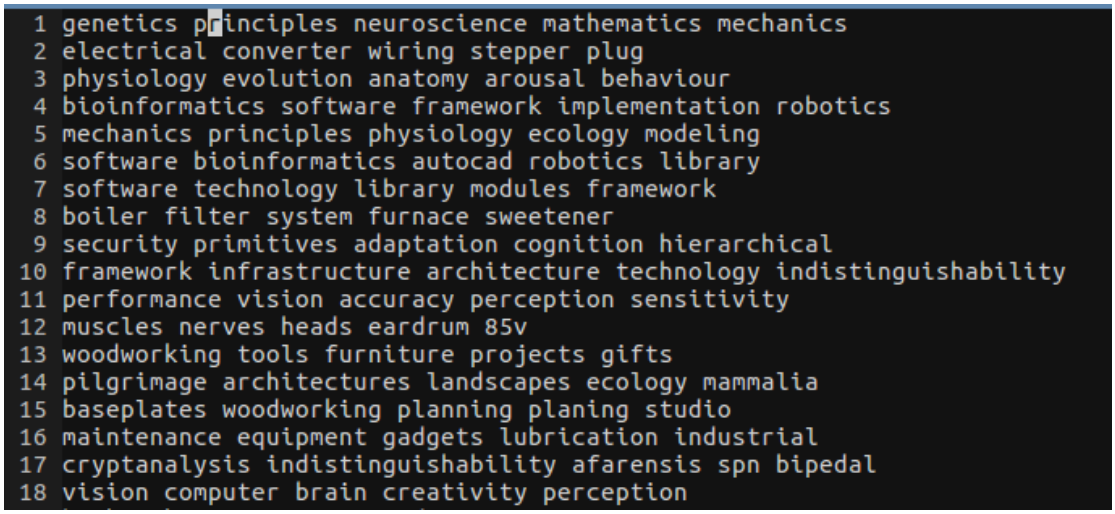


圖(四) Training Set Output Tags

但我們在 Validation Set 上的表現卻是非常差的，下面兩張圖為 Validation Set 的結果。



圖(五) Validation Set Truth Tags



圖(六) Validation Set Output Tags

可以發現我們 Inference 出來的 Tag 與答案差距非常遠。

- 失敗原因

從上面實驗結果可以發現，我們在沒有 Train 過的 Set 上 Inference 出來的 Tag 很常是之前 Training Set 上面的關鍵字，我們推測失敗的原因為 NN 對於 Training Set Overfitting 了。對此，我們有試過尋找只在 Validation Set 上出現過的字才可以加入 Tag，但加入此限制後，Inference 出來的 Tag 卻非常非常少，所以表示整個 Model 在 Train 的時候並沒有像我們想像的學習到 General 的 Doc 至 Tag 的映射關係，但雖然我們在 Validation Set 上表現的不好，我們卻可以在 Training Set 上表現得很好，這意味著我們的 DocVec 以及 TagVec 的取得方法是可行的，我們可能不能期望 NN 自動學習 General 的映射，而是要設計另一種 Model 來消除 Domain 之前的差異，若是利用老師上課教的 Domain-Adversarial Training 應該會表現得很好。