DS4NER Kit
Manual

Author：周建龍

NCU WIDM Lab

EMail：formatc.chou@gmail.com

Version：v1.03 2018/07/09

一、 系統介紹

命名實體辨識 (Named Entity Recognition, NER) 是知識工程相關研究與應用的最重要基礎之一，但現有的公開 NER 套件大多僅支援傳統的 NER 類型 (人名、地名、組織與其他)，且為使模組效果較佳，其訓練資料多使用品質較佳的新聞文章。

因此，若欲使用公開 NER 套件於非新聞文章的資料來源中擷取命名實體，其效果並不理想，其主要原因為訓練資料與測試資料的不一致 (cross domain)。更嚴重的情況為：欲擷取的命名實體不被支援，例如知名的 Stanford NER 不支援日文亦無法擷取非傳統的命名實體，例如：電影名、歌名、食物名、興趣點(point of interest, POI)…等。

基於以上原因，我們推出 DS4NER Tool Kit (以下簡稱 NER Kit)，此為一基於 Distant Supervision 所撰寫的 NER 客製工具，提供自資料的收集、前處理、訓練模型至測試效能的整套開源工具，讓所有的使用者均可針對自己的需求客製其 NER 模組，可解決公開 NER 套件的擷取效果低落問題，亦不必擔心沒有任何現成的公開套件支援欲擷取的命名實體的窘狀。

不但如此，DS4NER 亦包含 Self-Testing 以及 Tri-Training 的實作[1]，前者可減少 Distant Supervision 過程中的標記錯誤資料量，以提升資料品質，進而得到更好的 NER 辨識效能；後者利用已訓練好的三個模型標記新資料中的答案，當模型效能不太差時，相較於 Distant Supervision，可利用較少的新資料量以及較低的系統要求達到差不多的效能提升結果。

假設使用者已經讀取 DS4NER 的網頁說明，本文件將補充說明相關細節以及使用者必須知道的最重要事項，其他個別模組的使用說明，可於遇到問題時查詢本手冊即可。

- 系統要求與環境設定

本套件利用 Java 語言撰寫，可於安裝 Java Runtime (JRE)之 Windows 系統，或安裝 Oracle JRE 或 OpenJDK 之 64 位元 Linux 作業系統下運行。執行方式可透過 Java IDE 環境執行(例如 Eclipse)，亦可匯出 Jar 檔於 Console 下執行。

注意：因 Java 9 變動幅度較大，目前 NER Kit 並不支援 Java 9，請安裝 JRE 8 或 JDK 8 方可正常使用本套件。

- Global\Setting.java 此程序會在整個系統任一程序執行前先呼叫，其主要任務為：
  1. 讀取 Config.ini 並更新所有相對應的變數值
  2. 定義 NER Kit 版本以及最後更新日期
  3. strType 的值(IDE or Console)決定 DS4NER 的執行環境，若要執行於 Eclipse 之下，請修改為 IDE；若要匯出 jar 檔時，記得將 strType 的值改為 Console
- 參數格式說明

於 Console 下執行程序時，若僅加上-h 參數時(例：Java -cp NER.jar Training.CRF -h)，

或輸入參數數量、格式有誤，則會輸出如下圖的 Command Line Help，表示此程序需要加上四個參數 (順序不拘)。

當參數名稱為-str 開頭，表示此參數是一個字串，通常用來表示某檔案名稱，例如 Input.txt，而且此檔案必須放在<WorkFolder\Training>之下；同理，Output.txt 會輸出至<WorkFolder\Training>此目錄之下。另一個參數名稱為-b 開頭的參數，表示參數的輸入為 boolean (True or False)。

```
usage: Argument examples
 -bFilterNegExamples <arg>   True or False
 -Config <arg>
 -h,--help                   Shows argument examples
 -strInput <arg>             <WorkFolder\Training\>strInput.txt
 -strOutput <arg>            <WorkFolder\Training\>strOutput.txt
 -strSeeds <arg>             <WorkFolder\Training\>strSeeds.txt
```

Command Line Parameters Example

DS4NER 包含八個模組：Crawler、Pre-Processing、Prepare Data、Trainng、Testing、Baseline、Self-Testing 與 Tri-Training，其中第二章將說明前六個基本模組，第三章將說明最後兩個進階模組。

二、 基本模組說明

使用 Distant supervision 建立基本客製模組，依使用順序為 Crawler、Pre-Processing、Prepare Data、Training、Testing 五個模組。

2.1 Crawler (網路爬蟲)

Crawler 可以幫我們自特定的資料來源收集資料，在此提供針對 Google 搜尋結果的 Snippets 摘要內容的爬蟲。

1. GoogleSnippetCrawler

- 目的：依據 Seeds_Crawler.txt 內容為關鍵字，拜訪 Google 的搜尋結果，並收集包含關鍵字的片段 (Snippet)
- 參數

  -strSeeds　　<Corpus\Training\>Seeds.txt

  -strOutput_Dir　　<Corpus\Training\>strOutput_Dir (directory)

- 注意：在輸出目錄下的 result 目錄中，可找到許多透過 Crawler 所收集的、並以以 json 格式儲存的搜尋結果片段。

```
java -cp NER.jar Crawler.GoogleSnippetCrawler -strSeeds Seeds_Crawler.txt -strOutput_Dir Corpus_Crawler
```

2.2 PreProcessing (前處理)

透過 Crawler 所收集的資料中，包含很長的句子，以及類似意義的符號 (例如半形與全形的逗號)，因此需經過適當的前處理將長句子切割成較短的句子，將類似意義的符號統一，若有連續多個空白，則以一個空白代表之。

1. Segmentation

- 目的：移除替換語料庫中的符號，例如半形逗號一律替換成全形逗號，以及依據標點符號作句子的分割，依據輸入參數-strType 得知處理對象後，自動給予下列兩個參數的建議值，以及輸入/輸出檔案的路徑。參數：

  -strInput <Corpus\Training or Testing\>strInput.txt
  -strOutput_S <WorkFolder\Training or Testing\>strOutput_S.txt
  -strType <Training, Seeds, or Testing >
  -bSplitBySpace <True or False >
  -bFilterNoneECJK <True or False >
  -bFilterEllipsis <True or False >
- 注意：此程序參考依據 Config.ini 的 Sentence_Length_Min、Sentence_Length_Max 決定太短/太長的句子長度。輸出參數的「_S」表示此檔案是一個經前處理處理後的檔案。
- bSplitBySpace 決定是否針對空白做句子切割(限制前/後字元必須皆非英文與數字)；bFilterNoneECJK 可過濾包含過多非英文、非中文、非日文、非韓文字元的子句，定義為$\frac{|non\ ECJK\ character|}{|sentence|} \geq threshold$，可於 config.ini 的 NoneCJK_Threshold 修改 threshold；bFilterEllipsis 可過濾開始或結束字元為「...」「…」「⋯」「⋯⋯」的子句

```
java -cp NER.jar PreProcessing.Segmentation -strInput Corpus.txt -strOutput_S Corpus_S.txt -strType Training -bFilterNoneECJK false -bFilterEllipsis false
```

不同的 Type 其相關參數

strType	bSplit	bFilter	bDeDuplicate	bSortByLength
Training	True	True	True	True
Seeds	False	False	True	True
Testing	True	False	False	False

bSplit：是否切割字句
bFilter：是否過濾過常或過短的字句
bDeDuplicate：是否移除重複的子句
bSortByLength：是否依據 Token Length 由長到短排序

2.3 Prepare Data (資料準備)

本模組主要包含 AutoLabeling、MineDict、GenFeature、RemoveLabeling 四個程序，前三個程序執行完畢後，即可準備好訓練模型的所有相關要素，RemoveLabeling 程序可用來移除可能為誤標的結果。

1. AutoLabeling

- 目的：利用已知的 Entities (Seeds)，對訓練資料作標記，作為後續建立模型的訓練資料，並可依據 bFilterNegExamples 參數，決定是否移除 Negative Examples (即不含任何 Entity 的句子)；bPreProcessing 參數可決定是否要先做前處理。

  標記完成後，可於<WorkFolder\Training\>下產生 Labeling Report 可提供使用者分析是否有標記錯誤的 Noise (例如標記次數異常高)，若有 Noise 可手動自 Seed 中移除後再執行一次程序，或是執行 RemoveLabeling 程序將已標記的 Entities 移除。

  有時候 Sentence 與較長的 Seed 可能無法 100% Match，導致標記的 entities 數量極少，導致最後得到的訓練資料也很少。此問題可利用 LCS 演算法解決，但其需花費大量時間(時間複雜度：$O(n^2)$)，故本程序亦可透過 LSH 機制大量減少需執行 LCS 的 Seeds 數量，即可將總花費時間減少到可接受的程度。
- 參數：

  -strInput_S　　<WorkFolder\Training\>strInput_S.txt

-strSeeds <WorkFolder\Training\>strSeeds.txt
-strOutput_L <WorkFolder\Training\>strOutput_L.txt
-bFilterNegExamples <True or False>
-bPreProcessing <True or False>

- 注意：本程序依據 Config.ini 的 DataPreparation 下的 Medium_Entity_Length_Min 與 Long_Entity_Length_Min 定義 short、medium 與 long seed 的長度。

  本程序亦參考 Config.ini 的 DataPreparation 下 Labeling_Strategy_Short 等三個參數決定標記策略，其中 Igone 表示忽略不標記；Exact 表示 Exact Matching；Guillemet 表示 entity 必須包含在特定 Symbol 裡面才會標記，例如《飄》、「希望」。

  若欲啟動 LSH 功能，請於 Labeling_Strategy_Short 等三個參數中加上 LSH 即可，例如 Exact+LSH 或 LCS_LSH。strCondition 則限定長 Seeds 使用 LCS 加上 LSH 做標記時的額外條件，可用來提升標記品質，降低 Noise。此參數為 None 表示不限制 Seed 必須包含 LCS Result 的任何 Token；同理，此參數為 FirstLast，表示 LCS Result 必須同時包含 Seed 的第一個 Token 與最後一個 Token。輸出參數的「_L」表示此檔案是一個已標記後的檔案。

```
java -cp NER.jar PrepareData.AutoLabeling -strInput_S Input_S.txt -strSeeds Seeds.txt -strOutput_L Output_L.txt -bFilterNegExamples True -bPreProcessing False
```

2. MineDict

- 目的：分析已標記的訓練資料產生四個 Dictionary (CommonBefore、EntityPrefix、EntitySuffix、CommonAfter)並放置於<WorkFolder>\Dictionary 下
- 參數：

  -strInput_L <WorkFolder\Training\>strInput_L.txt
  -strMethod <Supp or Conf or HMCS>
  -fThreshold <Cumulative Score Threshold, 0.0f-1.0f>
- 注意：此程序的 fThreshold 參數介於 0~1 之間，作為過濾 Dictionary Terms 的門檻值，此值越低則 Dictionary Terms 數量越少

```
java -cp NER.jar PrepareData.MineDict -strInput_L Input_L.txt -strMethod Supp -fThreshold 0.5f
```

3. GenFeature

- 目的：將訓練資料參考 Dictionary 轉換成特徵矩陣的格式，並包含 BIESO 標籤
- 參數：

  -strInput_L <WorkFolder\Training\>strInput_L.txt
  -strOutput_F <WorkFolder\Training\>strOutput_F.txt
  -strType <Training, Testing, Baseline, or Extractor>
  -bPreProcessing <True or False>
- 注意：輸出參數的「_F」表示此檔案是一個已轉成 Features Matrix 格式的檔案

```
java -cp NER.jar PrepareData.GenFeature -strInput_L Input_L.txt -strOutput_F Output_F.txt –strType Training -bPreProcessing False
```

4. RemoveLabeling

- 目的：在已標記的訓練資料中，移除 strRemove.txt 中所有 entities 標記，亦即解除標記，若-bFilterNegExamples 為 True，且 sentence 解除 Entity 標記後沒有包含任何 entity，則此 sentence 將被移除。
- 參數：

  -strInput_L <WorkFolder\Training\>strInput_L.txt
  -strRemove <WorkFolder\Training\>strRemove.txt
  -strOutput_L <WorkFolder\Training\>strOutput_L.txt
  -bFilterNegExamples <True or False>

```
java -cp NER.jar PrepareData.RemoveLabeling -strInput_L Input_L.txt -strRemove Remove.txt -strOutput_L Output_L.txt -bFilterNegExamples True
```

2.4 Training (訓練模型)

本模組負責呼叫外部的序列標記模組，當 NER Tool 欲新增支援的序列標記模型時，可參考 Config.ini 與 Training.CRF.java 即可新增支援的模型。

1. CRF

- 目的：呼叫外部 Sequence Labeling Package 訓練 NER 模型
- 參數：

  -strInput_F　　<WorkFolder\Training\>strInput_F.txt

  -strModel　　<WorkFolder\Training\>strModel
- 注意： DS4NER 依據 config.ini 的[CRF++] section，將[variable]部份替換為正確路徑與檔名，以利呼叫 CRF++。

```
java -cp NER.jar Training.TrainModel -strInput_F Input_F.txt -strModel Model
```

```
[CRF++]
CRFLocation = crfpp ; the location of crf_learn.exe/crf_tset.exe and libcrfpp.dll
Training = [CRFLocation]crf_learn -f 5 -p 4 crfpp[separator]template_default.txt [TrainingData] [Model]
Training_Separator = " "
Testing = [CRFLocation]crf_test [-n num] -m [Model] [TestingData] -o [TestingResult]
Testing_Separator = "\t"
```

2.5 Testing (效能評估)

此模組包含三部份：Evaluation、Extracctor 與 Baseline 模組，前者針對已標記答案的測試文件，評估模型效能；後者針對未標記答案的文件擷取命名實體。Baseline 使用來評估此份測試文件的效能基準。

其中的 Evaluation 模組，亦帶有 Error Analysis 的功能，使用者可依據自訂的邏輯(例如新聞類別：政治、娛樂、運動...，或 entity 長度)，透過自己撰寫的程式或人工將測試資料分成數個部份，將測試資料拆開。

這樣子 DS4NER 即可個別算出效能，供使用者知道整體系統的改善方向，例如下表可知整體效能 F-Measure 為 0.8985，但是 sports 此類別的文章效能僅有 0.8174，若可針對此類別的文章改善效能，整體效能可望進一步提升。

Name	Corrections	Extractions	Entities	Precision	Recall	F-Measure
Politics	4,765	4,922	5,413	0.9681	0.8803	0.9221
Finance	2,228	2,453	2,454	0.9083	0.9079	0.9081
Sports	4,549	4,762	6,368	0.9553	0.7144	0.8174
Summary	11,542	12,137	14,235	0.9510	0.8515	0.8985

1. Evaluation

- 目的：針對訓練好的模型做效能評估，但 Entity 只要錯一個 Token 就算全錯，例如欲辨識王建民此人名，必須三個 Token 全對才有分數，此為 Exact Evaluation；若 Entity 即使錯一個 Token 還是有部分給分則為 Partial Evaluation，例如欲辨識「蔣渭水紀念館」此一地名，假設僅辨識出「紀念館」，還是有 3/6=1/2 的分數
- 參數：

  -strModel <WorkFolder\Training\>strModel
  -strMethod <Exact or Partial>
  -strOutput_Dir <WorkFolder\>strOutput (directory)

```
java -cp NER.jar Testing.Evaluation -strModel Model -strMethod Exact –strOutput_Dir Eva_Exact
```

2. Extractor

- 目的：利用已訓練的 NER Model 於未標記答案的文件辨識 Named Entity
- 參數：

  -strModel <NER Model>
  -strInput <Input File>
  -strOutput <Output File>
- 注意：辨識成功的 Named Entity 將使用 config.ini 裡面的 Entity_Start 與 Entity_End 將 Entity Tag 起來。

```
java -cp NER.jar Testing.Extractor -strModel Model -strInput Input.txt -strOutput Output.txt
```

3. SplitByLength

- 目的：可將 Testing Date 依據 Entity 長度，切成 n+1 個檔案，分別做測試，以利檢視不同長度的 Entity 其辨識效能是否較差，可用來分析系統效能與 Entity 長度的相關性。
- 參數：

  -strInput_S <WorkFolder\\Testing\\>strInput_S.txt
  -iSplited <1-n>

- 注意：假設分割成 n 部分，編號 00 的檔案僅包含沒有任何 entity 的句子（即 negative examples）；01 僅含 entity 長度為 1 的句子；最後一個檔案，包含 Entity 長度大於等於 n 的 Entities

```
java -cp NER.jar Testing.SplitByLength -strInput_S Input_S.txt -iSplited 10
```

4. SplitByLanguage

- 目的：可將 Testing Date 語言(英文、中文、日文、韓文、數字、不含任何 Entity 與其他)，切成 7 個檔案，分別做測試，以利檢視不同語言的 Entity 其辨識效能是否較差，可用來分析系統效能與語言的相關性。
- 參數：

  -strInput_S　　<WorkFolder\\Testing\\>strInput_S.txt

```
java -cp NER.jar Testing.SplitByLanguage -strInput_S Input_S.txt
```

5. SplitByVocabulary

- 目的：可將 Testing Date 依據 Entity 是否位於 Seeds，分成 InV (in vocabulary)、OOV (out-of-vocabulary)與 NoEntity 三個檔案，分別做測試，以利檢視 Entity 是否已知與辨識效能之間的關連性。
- 參數：

  -strInput_S　　<WorkFolder\\Testing\\>strInput_S.txt

  -strSeeds_S　　<WorkFolder\\Training\\>strSeeds_S.txt

```
java -cp NER.jar Testing.SplitByVocabulary -strInput_S Input_S.txt -strSeeds_S Seeds_S.txt
```

6. Baseline (效能基準)

Baseline 可用來來評估此份測試文件的效能基準，即使用所有的已知 Seeds 標記已知答案的測試文件後(事先將答案移除)，再與標準答案比較即可得到一個效能基準，若 NER 模型的效能低於此效能基準，可視為此 NER 模型沒有意義。

(a). RemoveTags

- 目的：將已標記答案的測試文件中的已標記答案移除
- 參數：

  -strInput_S <WorkFolder\Testing\>strInput_S.txt
  -strOutput_S <WorkFolder\Training\>strOutput_S.txt

```
java -cp NER.jar Baseline.RemoveTags -strInput_S LabeledCorpus_S.txt -strOutput_S Baseline_S.txt
```

(b). Evaluation

- 目的：針對 Baseline 做效能評估，亦可指定使用 Exact Match 或 Partial Match 來計算效能
- 參數：

  -strTesting_S <WorkFolder\Testing\>strTesting_S.txt
  -strBaseline_L <WorkFolder\Training\>strBaseline_L.txt
  -strMethod <Exact or Partial>

```
java -cp NER.jar Baseline.Evaluation -strTesting_S LabeledCorpus_S.txt -strBaseline_L Baseline_L.txt
-strMethod Exact
```

(c). Baseline Script

Script 說明：要計算 Baseline 時，需執行一系列分屬不同模組的相關程序

```
java -cp NER.jarPreProcessing.Segmentation -strInput LabeledCorpus.txt -strOutput_S LabeledCorpus_S.txt
-strType Testing -bFilterNoneECJK false -bFilterEllipsis false
java -cp NER.jar Baseline.RemoveTags -strInput_S LabeledCorpus_S.txt -strOutput_S Baseline_S.txt
java -cp NER.jar PrepareData.AutoLabeling -strInput_S Baseline_S.txt -strSeeds_S Seeds_S.txt
-strOutput_L Baseline_L.txt -bFilterNegExamples False -bPreProcessing False
java -cp NER.jar Baseline.Evaluation -strTesting_S LabeledCorpus_S.txt -strBaseline_L Baseline_L.txt
-strMethod Exact
java -cp NER.jar Baseline.Evaluation -strTesting_S LabeledCorpus_S.txt -strBaseline_L Baseline_L.txt
-strMethod Partial
```

2.6 Other

此模組包含兩支程式，可用來檢查資料中是否有巢狀標記或標記不完整的情況，以及轉換成 Feature Matrix 格式後是否有無法對齊的問題。

1. VerifyTags

- 目的：用來檢查檔案內容是否有不合理的巢狀標記或標記不完整的情況，若有則回報該句子編號，若問題不嚴重，可採用人工處理，若類似的問題發生次數很多，亦可透過自行撰寫程式修正之。檢查完畢後，輸出 Entity List 檔，供使用者分析之用。
  依據 strType 的值，資料來源分別是<WorkFolder\Training>或<WorkFolder\Testing>
- 參數：
  -strInput <WorkFolder\Training or Testing\>strInput.txt
  -strType <Training or Testing>

```
java -cp NER.jar Other.VerifyTags -strInput Input.txt -strType Testing
```

2. VerifyTokens

- 目的:用來檢查兩個已轉換成 feature matrix 格式的檔案,每行的 token 是否都一致，若有則回報該句子編號
  依據 strType 的值，資料來源分別是<WorkFolder\Training>或<WorkFolder\Testing>

- 參數：

-strInput_1	<WorkFolder\Training or Testing\>strInput_1.txt
-strInput_2	<WorkFolder\Training or Testing\>strInput_2.txt
-strType_1	<Training or Testing>
-strType_2	<Training or Testing>

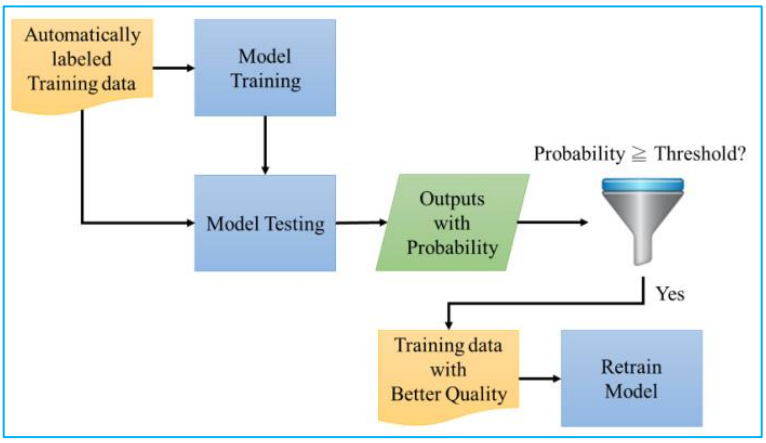
```
java -cp NER.jar Other.VerifyTokens -strInput_1 Input_1.txt -strInput_2 Input_2.txt -strType_1 Testing
-strType_2 Testing
```

三、 Self-Testing 與 Tri-Training

Self-Testing 可減少 Distant Supervision 過程中的標記錯誤資料量，以提升資料品質；Tri-Training 利用已訓練好的三個模型標記新資料中的答案，可利用較少的新資料量達到差不多的效能提升結果。

3.1 Self-Testing

已訓練完成的模型，對自己(訓練資料)作 Testing 並輸出 top 1 probability，然後移除低於門檻值的資料後，即可得到品質較高的資料。接下即可使用資料量較少但品質較高的資料得到一個效能較好的模型。



Self-Testing 示意圖

1. RemoveFeatureTag

- 目的：移除 Feature Matrix 中的 BIESO Column
- 參數：

-strInput_F	<WorkFolder\Training\>strInput_F.txt
-strOutput_NoTag_F	<WorkFolder\Training\>strOutput_NoTag_F.txt

```
java -cp NER.jar SelfTesting.RemoveFeatureTag -strInput_F Corpus_F.txt -strOutput_NoTag_F
Corpus_NoTag_F.txt
```

2. TestingWithProb

- 目的：對訓練資料作測試，並輸出 Top 1 probability
- 參數：

-strModel	<WorkFolder\Training\>strModel
-strInput_NoTag_F	<WorkFolder\Training\>strInput_NoTag_F.txt
-strOutput_NoTag_TR	<WorkFolder\Training\>strOutput_NoTag_TR.txt

- 注意：輸出參數的「_TR」表示 Testing Result

```
java -cp NER.jar SelfTesting.TestingWithProb -strModel Corpus.model -strInput_NoTag_F
Corpus_NoTag_F.txt -strOutput_NoTag_TR Corpus_NoTag_TR.txt
```

3. Filtering

- 目的：移除 Top 1 Probability 低於 Threshold 的 Sentence。輸出資料量少但品質較高的兩個 Outputs，其中一個是 Features Matrix 格式，可供 Training.TrainModel 使用以訓練 Model。Threshold 越高，移除的資料越多，剩餘的資料越少
- 參數：

-strInput_L	<WorkFolder\Training\>strInput.txt
-strInput_NoTag_TR	<WorkFolder\Training\>strInput_NoTag_TR.txt
-strOutput_L	<WorkFolder\Training\>strOutput_L.txt
-strOutput_F	<WorkFolder\Training\>strOutput_F.txt
-fThreshold	<0 – 1 float threshold>

```
java -cp NER.jar SelfTesting.Filtering -strInput_L Corpus_L.txt -strInput_NoTag_TR Corpus_NoTag_TR.txt
-strOutput_L Corpus_08_L.txt -strOutput_F Corpus_08_F.txt -fThreshold 0.8f
```

4. Self-Testing Script

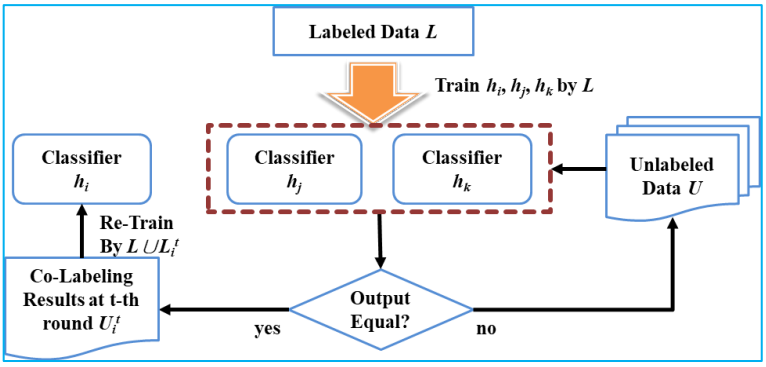
Script 說明：要執行 Self-Testing 時，需執行一系列分屬不同模組的相關程序

```
REM Self-Testing
java -cp NER.jar SelfTesting.RemoveFeatureTag -strInput_F Corpus_F.txt -strOutput_NoTag_F
Corpus_NoTag_F.txt
java -cp NER.jar SelfTesting.TestingWithProb -strModel Corpus.model -strInput_NoTag_F
Corpus_NoTag_F.txt -strOutput_NoTag_TR Corpus_NoTag_TR.txt

REM Remove Probability < 0.8 training data, then re-train model and evaluation
java -cp NER.jar SelfTesting.Filtering -strInput_L Corpus_L.txt -strInput_NoTag_TR Corpus_NoTag_TR.txt
-strOutput_L Corpus_08_L.txt -strOutput_F Corpus_08_F.txt -fThreshold 0.8f
java -cp NER.jar Training.CRF -strInput_F Corpus_08_F.txt -strModel Corpus_08.model
```

3.2 Tri-Trainng

將已標記資料 L 透過 Bootstrap 後分成三部份訓練 h_i, h_j, h_k，任選兩個分類器 h_j 與 h_k 對未標記資料 U 選取的資料標記答案，若兩個分類器的答案一致，則將此 example (x,y) 放入 U_i^t 作為 h_i 於 t-th round 的新增的資料。最後對 U_i^t 作 subsample 後，取 $L \cup U_i^t$ 的聯集對 h_i 重新訓練，重複此過程，每一回合對 h_i, h_j, h_k 三個分類器皆自 U 中選取新的資料，並重新訓練以提升效能。



Tri-training 架構圖

1. Initialization

- 目的：首先在<WorkFolder>下自動建立 TriTraining 目錄，再於此目錄下再建立 ErrorRate、Model、Trainingng 三個目錄。並準備六個後續使用的檔案(L.txt、L_F.txt、L_NoTag.txt、L_NoTag_F.txt、U.txt、U_F.txt)，除此之外，亦透過不同的 sample 方法對三個分類器準備好初始資料，並呼叫 Sequence Labeling Package 訓練模型。
- 參數：

  -strInput_L　　<WorkFolder\Training\>strInput_L.txt
  -strInput_L_F　　<WorkFolder\Training\>strInput_L_F.txt
  -strInput_U　　<WorkFolder\Training\>strInput_U.txt
  -strSample　　<Bootstrap or Random70 or DivideEqual>
- 注意：

  請使用者利用之前的相關程序(RemoveTag … etc.)，準備好 L.txt、L_F.txt 以及 U.txt 這三個檔案，並存放於<WorkFolder\Trfaining>目錄之下，這三個檔案最好是經過同樣的前處理程序處理過。

  -strSample 意指如何自 L 中準備訓練資料 S_i、S_j、S_k，然後用來訓練 h_i、h_j、h_k

  1. Bootstrap：自 L 取$|L|$次並重複拿，$|S_i| = |L|$
  2. Random70：亂數取 70%並可重複拿，$|S_i| = |L|$*0.7
  3. DivideEqual：不可重複，$|S_i| = |L| * 1/3$

```
java -cp NER.jar TriTraining.Initialization -strInput_L Corpus_08_L.txt -strInput_L_F Corpus_08_F.txt
-strInput_U U_S.txt -strSample Bootstrap
```

2. TrainModel

- 目的：TriTraining 的核心，每一次疊代都會先經過計算 ErrorRate、計算可新增資料上限、準備新資料、重新訓練此四個步驟。若-strEvaluation 參數不為 None，模型效能會輸出於 TriTraining\Evaluation_Exact 與 TriTraining\Evaluation_Partial 下。TriTraining 的細節、虛擬碼請參考論文[1]。
- 參數：

  -iIter 1
  -strEvaluation <Exact, Partial, All, or None>
  -bRemoveFiles <True or False>
- 注意：
  1. -Iter：1 表示開始執行 Tri-Training 時，以 Iteration=1 開始執行
  2. -strEvaluation：TriTraining 執行完畢後，是否直接計算效能
  3. -bRemoveFiles：TriTraining 執行完畢後，是否移除每一回合的相關檔案 (例如每回合的新增資料，這些檔案大小相當可觀)

```
java -cp NER.jar TriTraining.TrainModel -iIter 1 -strEvaluation None -bRemoveFiles False
```

3. Evaluation

- 目的：在執行 Training.TrainModel 時若沒有一併計算效能，可利用此程序於完成後計算效能，因此相關參數與說明同 Testing.Evaluation
- 參數：

  -strEvaluation <Exact, Partial, All, or None>
  -bRemoveFiles <True or False>
- 注意：
  - -strEvaluation：TriTraining 執行完畢後，是否直接計算效能
  - -bRemoveFiles：TriTraining 執行完畢後，是否移除每一回合的相關檔案 (例如每回合的新增資料，這些檔案大小相當可觀)

```
java -cp NER.jar TriTraining.Evaluation -strEvaluation All -bRemoveFiles True
```

4. TriTraining Script

Script 說明：要執行 Tri-Training 時，可執行下列一系列的 Scripts

```
REM Tri-Training
java -cp NER.jar PreProcessing.Segmentation -strInput U.txt -strOutput_S U_S.txt -strType Training
-bFilterNoneECJK false -bFilterEllipsis false
java -cp NER.jar TriTraining.Initialization -strInput_L Corpus_08_L.txt -strInput_L_F Corpus_08_F.txt
-strInput_U U_S.txt -strSample Bootstrap
java -cp NER.jar TriTraining.TrainModel -iIter 1 -strEvaluation None -bRemoveFiles False
java -cp NER.jar TriTraining.Evaluation -strEvaluation All -bRemoveFiles True
```

四、 Customize (客製化)

4.1. Customize Feature

目的：新增一個長度為 1 的 Feature 要修改三個部份如下：

- Global\Features.java
  增加一個 function: public String getFeature_CustomFeature(String strInput)
  Input 是一個長度為 1-3 的字串，依據使用者的邏輯回傳"0" or "1"，以下為範例：
  首先假設已將客製資料放入 CustomFeature.txt，並載入存於 hsCustomFeature 中

```
129     public String getFeature_CustomFeature(String strInput)
130     {
131         if(strInput.length() > 0 && hsCustomFeature.contains(strInput))
132         {
133             return "1";
134         }
135         else
136         {
137             return "0";
138         }
139     }
```

getFeature_CustomFeature Function Example

- Global\Features.java：增加 Line 268

```
strFeature = strWord_1
  + Setting.strCRFTraining_Separator + getFeature_CommonBefore(strWord_1) // CommonBefore_1
  + Setting.strCRFTraining_Separator + getFeature_CommonBefore(strWord_2) // CommonBefore_2
  + Setting.strCRFTraining_Separator + getFeature_CommonBefore(strWord_3) // CommonBefore_3
  + Setting.strCRFTraining_Separator + getFeature_EntityPrefix(strWord_1) // EntityPrefix_1
  + Setting.strCRFTraining_Separator + getFeature_EntityPrefix(strWord_2) // EntityPrefix_2
  + Setting.strCRFTraining_Separator + getFeature_EntityPrefix(strWord_3) // EntityPrefix_3
  + Setting.strCRFTraining_Separator + getFeature_EntitySuffix(strWord_1) // EntitySuffix_1
  + Setting.strCRFTraining_Separator + getFeature_EntitySuffix(strWord_2) // EntitySuffix_2
  + Setting.strCRFTraining_Separator + getFeature_EntitySuffix(strWord_3) // EntitySuffix_3
  + Setting.strCRFTraining_Separator + getFeature_CommonAfter(strWord_1)  // CommonAfter_1
  + Setting.strCRFTraining_Separator + getFeature_CommonAfter(strWord_2)  // CommonAfter_2
  + Setting.strCRFTraining_Separator + getFeature_CommonAfter(strWord_3)  // CommonAfter_3
  + Setting.strCRFTraining_Separator + getFeature_EngNum(strWord_1)       // English + Number
  + Setting.strCRFTraining_Separator + getFeature_Symbol(strWord_1)       // Symbol
  + Setting.strCRFTraining_Separator + getFeature_CustomFeature(strWord_1);   // Custom Feature
```

新增一個 Feature

- Root\template_custom.txt 修改 template，新增一個 feature

```
# Custom Feature, Length = 1
U330:%x[-2,15]
U331:%x[-1,15]
U332:%x[0,15]
U333:%x[1,15]
U334:%x[2,15]
U335:%x[-2,15]/%x[-1,15]
U336:%x[-1,15]/%x[0,15]
U337:%x[0,15]/%x[1,15]
U338:%x[1,15]/%x[2,15]
U339:%x[-2,15]/%x[-1,15]/%x[0,15]
U340:%x[-1,15]/%x[0,15]/%x[1,15]
U341:%x[0,15]/%x[1,15]/%x[2,15]
```

修改 CRF++ Template 檔，新增一個 Feature

五、 FAQ

- Out of Memory：若收集的 Corpus 較大時，可能會出現 out of memory 的錯誤訊息，此為 Java Virtual Machine 在預設值下沒有足夠的記憶體處理這個 Corpus，因此可以透過指定記憶體參數給予較多的記憶體後，即可順利執行。例：java -cp NER.jar → java –Xms8g –Xmx8g -Xss256k -cp NER.jar

Appendix A (附錄 A)

- CRF++設定

Windows 平台的使用者，可自 CRF++官方網站下載 CRF++，並將 crf_learn.exe、crf_test.exe 以及 libcrfpp.dll 三個檔案放置於本套件下的 crfpp 目錄(建議)，若要放置於其

他路徑，則切記修改 config.ini 中 CRF++相關設定。

```
[CRF++]
CRFLocation = crfpp ; the location of crf_learn.exe/crf_tset.exe and libcrfpp.dll
```

Linux 平台的使用者，請先至 CRF++官方網站下載 CRF++ 原始碼，並於 Linux 下編譯，方可正常執行，編譯步驟請參考官方網站。編譯完成後請於 console 下執行 crf_learn -h 與 crf_tset -h，測試是否正確編譯。

```
$ crf_learn
CRF++: Yet Another CRF Tool Kit
Copyright (C) 2005-2013 Taku Kudo, All rights reserved.

Usage: crf_learn [options] files
  -f, --freq=INT                use features that occuer no less than INT(default 1)
  -m, --maxiter=INT             set INT for max iterations in LBFGS routine(default 10k)
  -c, --cost=FLOAT              set FLOAT for cost parameter(default 1.0)
  -e, --eta=FLOAT               set FLOAT for termination criterion(default 0.0001)
  -C, --convert                 convert text model to binary model
  -t, --textmodel               build also text model file for debugging
  -a, --algorithm=(CRF|MIRA)    select training algorithm
  -p, --thread=INT              number of threads (default auto-detect)
  -H, --shrinking-size=INT      set INT for number of iterations variable needs to    be optimal before
considered for shrinking. (default 20)
  -v, --version                 show the version and exit
  -h, --help                    show this help and exit
```

crf_learn -h Linux 下的輸出結果

Reference

[1] Chien-Lung Chou, Chia-Hui Chang, Ya-Yun Huang, Boosted Web Named Entities Recognition via Tri-Training, Transactions on Asian and Low-Resource Language Information Processing, Volume 16 Issue 2, 2016.

[2] Chien-Lung Chou, Chia-Hui Chang: Named Entity Extraction via Automatic Labeling and Tri-training: Comparison of Selection Methods. AIRS 2014: 244-25.