D13 詞幹/詞條提取：Stemming and Lemmatization

簡報閱讀　範例與作業　問題討論

重要知識點
詞幹/詞條提取目的
詞幹/詞條提取優缺點
詞幹提取常見做法

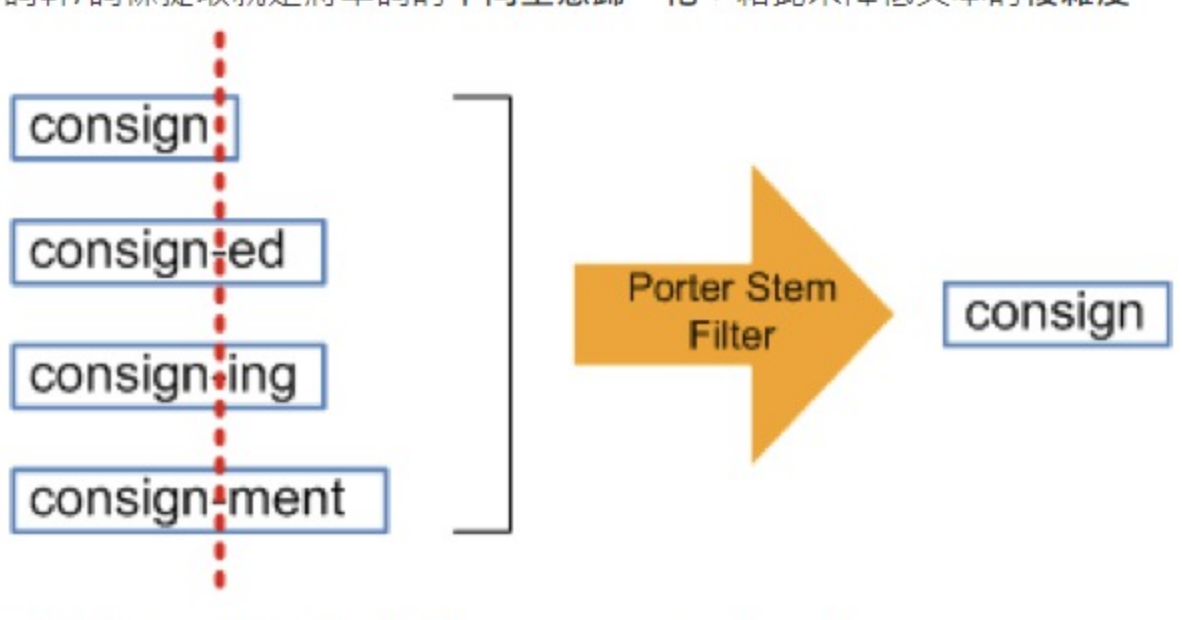
重要知識點

- 了解詞幹/詞條 提取的目的與原理
- 了解詞幹/詞條 提取的優缺點

詞幹/詞條提取目的

在英文語句中，同一個單詞的拼法可能會隨著**時態**、**單複數**、**主被動等狀況**而有所改變，如 speaking / speak ｜ cats / cat，然而其所要表達的語意並沒有太大的不同。

詞幹/詞條提取就是將單詞的**不同型態歸一化**，藉此來降低文本的**複雜度**。



資料來源：去字尾的方法，使用Porter Stemming Algorithm

詞幹/詞條提取優缺點

優點：

- 能夠大幅降低單詞數量：例如我們在建立 Bag of words 時使用 OneHotEncoding 的方式將字串轉為向量，如果在很大的訓練文本上使用而不經過任何詞幹提取的預處理，將會形成一串相當稀疏的向量。
- 降低資料複雜度，加快模型訓練速度：如 speak / speaking 想表達的語意相當接近，如果將兩者都轉為 speak，模型就能較快速學習到其語意訊息。

缺點：

- 失去部分訊息：降低資料複雜度往往是一體兩面的，以上述 speaking / speak 為例，有可能某些預測需要利用到時態的訊息，當我們將兩者都轉為 speak 輸入模型時， speak-ing 中 ing 的訊息也被我們移除。

詞幹提取常見做法

- Stemming
- Lemmatization

兩者的目標都相同，就是要取出**單詞的最小單位(詞幹/詞條)**，然而在拆解的原理上並不相同。

詞幹提取常見做法整理

常見方法	特色
Porter stemmer	運用既定的規則移除常見單詞字尾。
Snowball stemmer	根據使用者對Porter stemmer提出的問題加以優化。
Lemmatization	需字典尋找單詞原型，並可以利用POS tagging的訊息。

Stemming

Stemming 是兩者中較為簡單的作法(詞幹提取)，通常為 "**Heuristic process**"，簡單來說就是**制定規則**來拆解單詞(rule-based)，像是看到 ing/ed 就去除。

常見方法又可以分為兩種：
Porter stemmer / Snowball stemmer

Porter stemmer

為較為傳統的作法，運用既定的規則移除常見單詞字尾，相較於現今其他作法，顯得較不實用，不過仍然可以用來當作 stemming algorithm 的 benchmark(基準)。

Snowball stemmer

也稱為 Porter2，原理仍然相同(rule-based)，單根據使用者對 Porter stemmer 提出的問題加以優化。

Stemming 常見問題

Overstemming / Understemming

Overstemming

從字面上我們就可以理解 Overstemming 是指過多的 "Stemming"，意思就是去除太多字尾，導致剩餘的字根無法完整的表達原單詞的訊息，又或是不同語意的單詞經過處理後變為相同的字根。

For ex.
university, universal, universities, and universe
Stemming 完都成為 universi

就上述例子而言，儘管我們可以寫更多規則來獲取正確結果，然而這種情況被解決卻也可能造成其他種錯誤出現。

Understemming

換言之，Understemming 指的是字尾去除得不夠乾淨，同一個語意的單詞可能被拆解為不同的字根，如此不只失去了 stemming 的原意還有可能造出更多意義不明的字根。

Lemmatization

- Stemming 顧名思義是取出單詞的 **Stem (詞幹)**
- Lemmatization 則是取出單詞的 **Lemma**，Lemma 為語言學的用詞，可以翻譯為詞條、詞元、詞首等等，**其意思為字的元型**。

Lemmatization 的定義為：
Return the base or dictionary form of a word

其大致上與 Stemming 相同，然而相較於 Stemming ， Lemmatization 是需要有字典來尋找單詞的原型，除此之外 Lemmatization 也可以利用 POS tagging 的訊息來給出更正確的答案。

For ex.
　　amusing

Stemming : 返回 amus，然而其單詞原型應該為 amuse

```
In [102]: print('Stemming amusing : {}'.format(ps.stem('amusing')))

          Stemming amusing : amus
```

```
print('Stemming amusing : {}'.format(ps.stem('amusing')))
```

Lemmatization : 返回 amuse

```
In [104]: print('lemmatization amusing : {}'.format(lemmatizer.lemmatize('amusing',pos = 'v')))
          lemmatization amusing : amuse
```

```
print('lemmatization amusing : {}'.format(lemmatizer.lemmatize('amusing',pos = 'v')))
```

Stemming or Lemmatization

通常來說 Lemmatization 相較於 Stemming 會是更好的選擇，然而 Stemming 的優勢在於其所需運算較少，因此"速度較快"。

方 法	使用場景
Stemming	假設我們希望運算速度較快而且在該任務上兩者精度並沒有明顯差異
Lemmatization	假設我們時間充裕且希望得到較高精度的結果

參考資料

現今 SOTA 模型中較少用到 Stemming / Lemmatization 的技術，取而代之的是運用 NLP 模型(ex. BERT)來進行單詞拆解，常見如 Wordpiece，其將文字拆成字根、字首和字尾來保留更多語意訊息，並且也能有效降低字典中單詞數量。

網站：WordPiece

一文讀懂BERT中的WordPiece

完整機器學習實現代碼GitHub
歡迎轉載，轉載請註明出處https://www.cnblogs.com/huangyc/p/10223075.html
歡迎溝通交流： 339408769@qq.com

0. 目錄

- 1. 前言
- 2. WordPiece原理
- 3. BPE算法
- 4. 學習資料
- 5. 總結

回到頂部

1. 前言

2018年最火的論文要屬google的BERT，不過今天我們不介紹BERT的模型，而是要介紹BERT中的一個小模塊WordPiece。

下一步：閱讀範例與完成作業