人工智慧概論 HW2 報告

110550088 李杰穎

April 9, 2022

1 Preprocessing

在本次作業中，我主要實作了以下幾種 preprocessing 的方法，條列如下：

1. 將英文大寫轉成小寫: 使用 Python 內建的 `lower()` 函式，將字串文字全部轉成小寫
2. 移除 stopwords: 利用 nltk 提供的 stopwords 列表，將 stopwords 從字串中移除。
3. 移除 `<br />` HTML tag: 使用 Python 內建的 `replace()` 函式，將 `<br />` 用一個空白取代
4. 移除標點符號: 利用 for 迴圈檢查每一個 char 是否為標點符號，如果非標點符號則將其加進一個 list，最後再使用 `“”.join()` 來將 list 內元素轉為字串。檢查標點符號的部分則使用內建之 `string.punctuation` 來檢查。
5. Stemming (使用 nltk 內建之 SnowballStemmer): Stemming (詞幹提取) 是一種將詞彙去除後綴的方式。將單字進行 stemming 會讓模型不用處理額外的訊息，以下是一些經過 SnowballStemmer 處理後的單字。
   cared → care, university → univers, fairly → fair, easily → easili, singing → sing, sings → sing, sung → sung, singer → singer, sportingly → sport

在進行 preprocessing 時，會依照上面排列的順序進行這五個步驟。

下方為一英文句子通過以上 preprocessing 後的句子。

“It is a truth universally acknowledged that
 a single man in possession of a good fortune must be in want of a wife.” → “truth univers acknowledg singl man possess good fortun must want wife”

我也會在下文討論各 preprocessing 的方法對於最終的 F1-Score 的影響。

2 Implement the bi-gram language model

本部分介紹 bi-gram model 的實作，因為大部分實作細節都可以在繳交的程式碼中看到，故在此我只大致說明實作內容。

1. 計算各 unigram 及 bigram 的出現頻率: 由於在計算 $P(w_i|w_{i-1})$ 時，需要同時知道 bigram 及 unigram 的出現頻率，故先 iterate 所有 document 並利用 dict 來統計出現的頻率。
2. 利用上一個步驟的頻率求出 $P(w_i|w_{i-1})$: 左述之條件機率可由下式得到:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i)}{\text{count}(w_{i-1})} \tag{1}$$

3. 將所有算出的機率利用 Python 內建的 dict 資料結構，存為 `model[`w_{i-1}`][`w_i`]` 的形式，而 feature 則儲存各 bigram 的頻率 (即為 `feature[(`w_{i-1}`, `w_i`)]`)

可以發現若 Eq. 1 中分子及分母若為 0，則會使機率的計算發生問題，於是我在此使用 Add-1 (Laplace) Smoothing 來避免以下問題，Add-1 Smoothing 的具體式子如下:

$$P(w_i|w_{i-1}) = \frac{\text{count}(w_{i-1}, w_i) + 1}{\text{count}(w_{i-1}) + |V|} \tag{2}$$

其中 $|V|$ 為 unique vocabulary 的數量。在使用 Add-1 Smoothing 後，就不會出現先前機率出現 0 或無限大的情形。

3 Implement feature selection and conversion

我主要使用使用了兩種 feature selection 的方法，第一種方法是按 bigram 的出現頻率進行排序。第二種則是 Chi-Square Test 進行 feature selection。第一種方式較為簡單，在此不在贅述。接下來會介紹 Chi-Square Test 的實做細節。

3.1 Chi-Square Test Feature Selection

χ^2 是一個可以計算出特定 feature 對於最終答案 dependent 程度的演算法，對於二元分類問題其計算方式如下：

$$
\begin{aligned}
e_{00}(\text{bigram}) &= \frac{sum_{neg}(sum_{pos} + sum_{neg} - (N_{pos}(\text{bigram}) + N_{neg}(\text{bigram})))}{sum_{pos} + sum_{neg}} \\
e_{01}(\text{bigram}) &= \frac{sum_{pos}(sum_{pos} + sum_{neg} - (N_{pos}(\text{bigram}) + N_{neg}(\text{bigram})))}{sum_{pos} + sum_{neg}} \\
e_{10}(\text{bigram}) &= \frac{sum_{neg}(N_{pos}(\text{bigram}) + N_{neg}(\text{bigram}))}{sum_{pos} + sum_{neg}} \\
e_{11}(\text{bigram}) &= \frac{sum_{pos}(N_{pos}(\text{bigram}) + N_{neg}(\text{bigram}))}{sum_{pos} + sum_{neg}}
\end{aligned}
$$

$$
\begin{aligned}
\chi^2(\text{bigram}) = &\frac{(sum_{neg} - N_{neg}(\text{bigram}) - e_{00}(\text{bigram}))^2}{e_{00}(\text{bigram})} \\
&+ \frac{(sum_{pos} - N_{pos}(\text{bigram}) - e_{01}(\text{bigram}))^2}{e_{01}(\text{bigram})} \\
&+ \frac{(N_{neg}(\text{bigram}) - e_{10}(\text{bigram}))^2}{e_{10}(\text{bigram})} \\
&+ \frac{(N_{pos}(\text{bigram}) - e_{11}(\text{bigram}))^2}{e_{11}(\text{bigram})}
\end{aligned}
\tag{3}
$$

其中 sum_{pos} 為 positive 句子中的 bigram 總和，sum_{neg} 為 negative 句子中的 bigram 總和。$N_{pos}(\text{bigram})$ 為 bigram 在 positive 句子出現的次數，$N_{neg}(\text{bigram})$ 為 bigram 在 negative 句子出現的次數。

透過以上方式可以計算出特定 bigram 的 χ^2 score。χ^2 score 越高，代表此 bigram 對於結果越 dependent，也就代表此 bigram 越有價值。

計算出所有 bigram 的 χ^2 score 後，我們就可以將各 bigram 透過 χ^2 score 從小排到大，並取出前 `feature_num` 個 bigram 做為 GaussianNB 的輸入。

4 Perplexity

透過 Eq. 2，我們即可以計算一個 bigram 模型的 perplexity，perplexity 的計算方式如下:

$$\text{Perplexity} = 2^{-\text{entropy}}, \text{where entropy} = \frac{1}{N}\sum_{i=0}^{N}\log_2(P(w_i|w_{i-1})) \tag{4}$$

當我們使用測試資料去測試模型時，算出的 perplexity 越低時，代表模型認為測試資料中的句子出現的機率越高。也就是說當一個語言模型的 perplexity 越低時，模型的 performance 越好。

5 Experiments

5.1 Preprocessing 相關實驗

此部分會探討不同 preprocessing 的方法對於 perplexity, F1-score, precision 及 recall 的影響。

6 Discussion

6.1