Web Retrieval and Mining

Programming Assignment 2

資工四 黃柏智
b01902068

執行:

chmod a+x ./compile.sh
chmod a+x ./naivebayes.sh
chmod a+x ./EM.sh
./compile.sh (可以不用執行 compile.sh)
./naivebayes.sh -i directory -o output file [-n labeled-data size]
./EM.sh -i directory -o output file [-n labeled-data size]

Naive Bayes Classifier:

在 Naive Bayes 中，在給定一些 parameters 的情況下 (我們叫它 θ)，能求出 probability distribution。而 probability distribution 中，包含了許多 components，每個 component 就是 θ 的一個 disjoint subset (在這次作業中每個 topic 可視為是一個 component)。
接著，會依序求出每個 unlabeled documents 從這些 components 生成的機率分別是多少，進而認定該 documents 從哪個 components 生成的機率越高。根據 “Text Classification from Labeled and Unlabeled Documents using EM” 這篇論文，其公式可以寫成：

$$P(d_i|\theta) = \sum_{j=1}^{|\mathcal{C}|} P(c_j|\theta)P(d_i|c_j;\theta).$$

其中 $P(c_j \mid \theta)$ 代表著該 component 的 mixture weights，
而 $P(d_i \mid c_j;\theta)$ 代表著某一個 document 從特定 component 生成的機率，又可表示成：

$$P(d_i|c_j;\theta) = P(|d_i|)\prod_{k=1}^{|d_i|} P(w_{d_{i,k}}|c_j;\theta).$$

其中 $P(w_{d_{i,k}} \mid c_j ; \theta)$ 代表著 document 中每一個 word 在給定的 component 中的機率

因此我們可以得知，要使用 Naive Bayes，就必須求出 $P(c_j|\theta)$ 以及 $P(d_i \mid c_j;\theta)$，而要求出 $P(d_i \mid c_j;\theta)$ 就必須求出 $P(w_{d_{i,k}} \mid c_j ; \theta)$。根據論文的 section 4.2，

$$P(c_j|\hat{\theta}) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} P(y_i = c_j|d_i)}{|\mathcal{C}| + |\mathcal{D}|}$$

$$P(w_t|c_j;\hat{\theta}) = \frac{1 + \sum_{i=1}^{|\mathcal{D}|} N(w_t, d_i)P(y_i = c_j|d_i)}{|V| + \sum_{s=1}^{|V|}\sum_{i=1}^{|\mathcal{D}|} N(w_s, d_i)P(y_i = c_j|d_i)}$$

根據這兩個公式我們可以發現，實作 naive bases 關鍵點是**求出每個 topic 中每一個 unique word 的 word count**。我試過幾種定義 word 的方式：
第一種是先把某一段文字中，用 regular expression 找出所有連續的英文單字並轉成小寫。假設在 topic A 下有一段文字長這樣：“In article <114127@bu.edu>”，那根據第一種方法，會得到 [‘in’, ‘article’, ‘bu’, ‘edu’]，這四個 unique 的字，這四個字在 A 中的 word count 就會加 1。
第二種定義的方法是，對某一段文字，先用空白把這段文字切成好幾個部分，再把每個部分中，不屬於英文字母的部分去除。假設在 topic A 下有一段文字長這樣：“In article <114127@bu.edu> jaeger@buphy.bu.edu <ma”，那根據第二種方法，會得到 [‘in’, ‘article’, ‘buedu’, ‘jaegerbuphybuedu’, ‘ma’] 這五個 unique 的字，這五個字在 A 中的 word count 就會加 1。
根據測試的結果，第二種方法做出來的結果會比第一種方法，正確率高上 2~3%，因此我採用第二種。
當算出每個 topic 中所有 unique 的字分別出現了幾次，就能實作出 Naive Bayes Classifier 了。

EM Algorithm:

Naive Bayes 需要使用 labeled data，但現實中，取得 labeled data 的 cost 遠遠比取得 unlabeled data 要高，也更困難。因此需要依靠 EM algorithm 把 unlabeled data 也考慮進來。事實上 EM algorithm 也是 based on Naive Bayes Classifier，上面我求出了每個 topic 的 language model，也就是在每個 topic 中，所有 unique 的字分別出現了幾次，接著對 unlabeled data 中所有的 document，分別求出該 document 最可能由哪個 topic generate 出來，並把該 document 加入該 topic 的 language model 中。要注意的是這個加入的動作並不是永久加入的，在 EM algorithm 中的每個 iteration 中，會先決定每一個 unlabeled document 最可能由哪個 topic generate，然後把分別把每一個 unlabeled document 加到該 document 最有可能屬於的 topic 中。接著用這個 iteration classify 出來的結果去計算下一個 iteration 中，每一個 unlabeled document 應該屬於哪個 topic，直到收斂。

在 EM algorithm 中每個 iteration 可以表示成這樣：

```
while True:
    E-step: 根據 naive bayes 求出每一個 unlabeled document 應該要分到哪個 topic 中
        if 收斂:
            結束 EM algorithm
        else:
            M-step: 根據 E-step classify 出的結果，把每個 document 分別
                    加到最有可能 generate 該 document 的 topic 的
                    language model (LM) 中
```

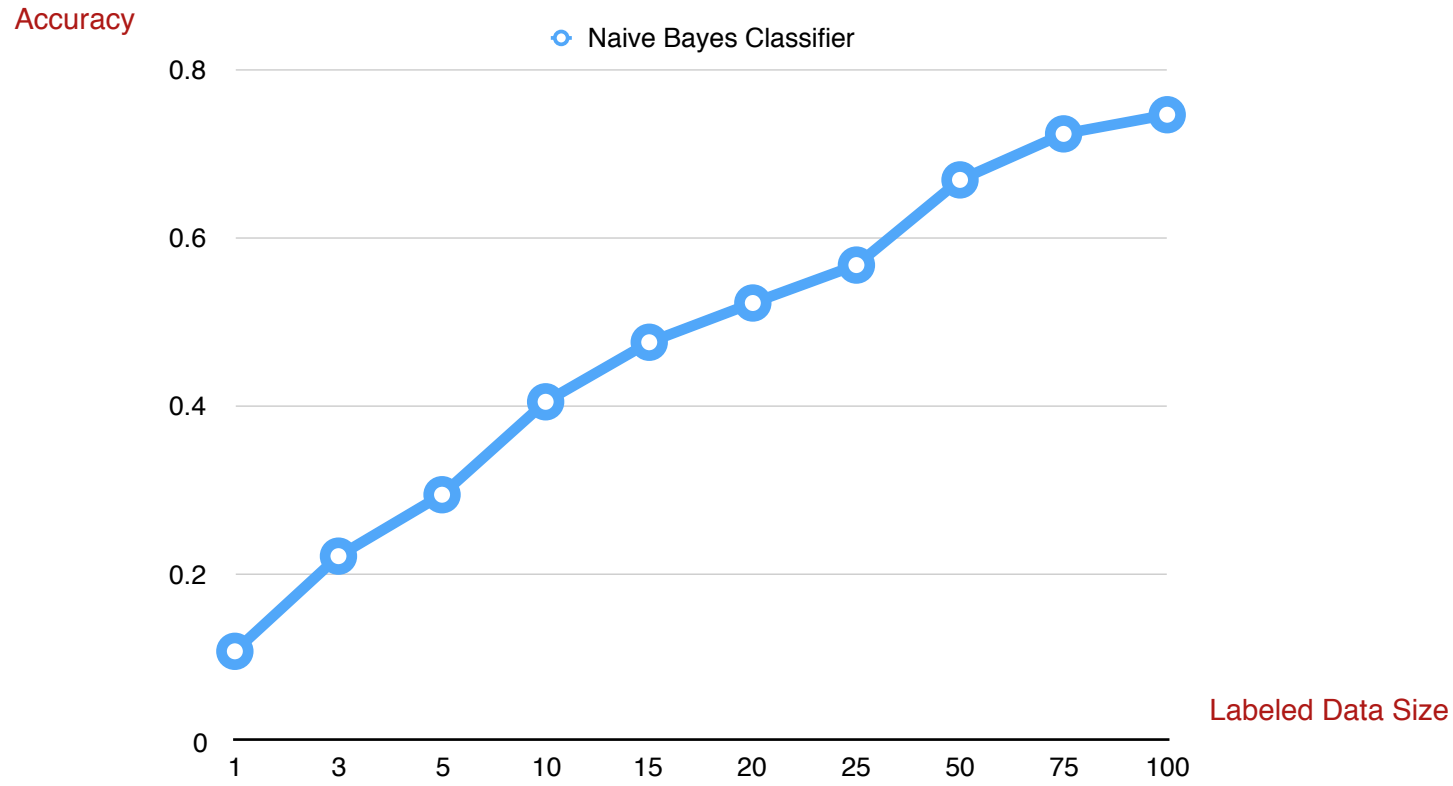
當某個 iteration 中沒有任何 document 的 topic 改變，我才判斷其收斂。

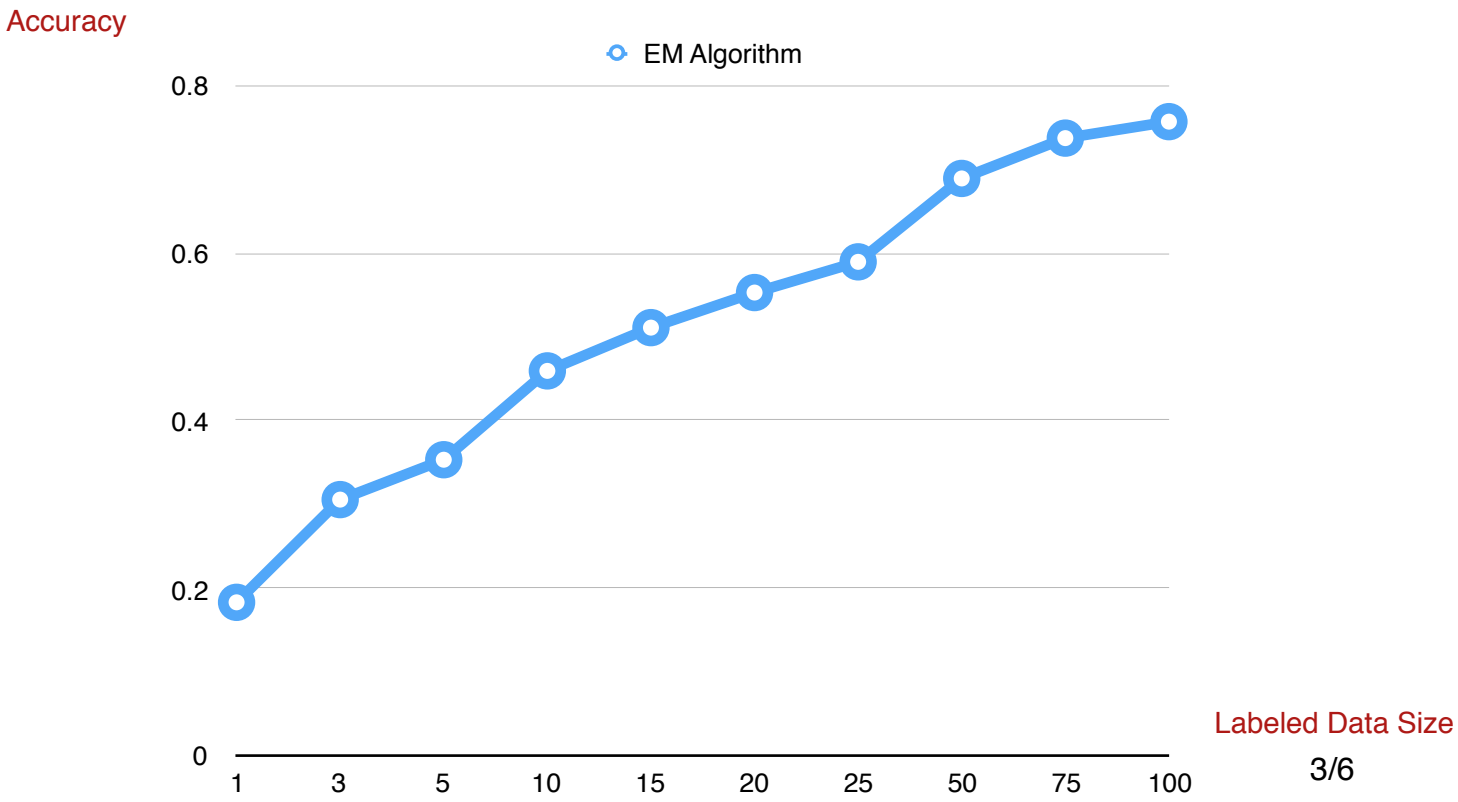
Results of Experiments:

- Result of two methods

Naive Bayes Classifier: 當使用全部的 labeled data，準確率是 0.74668223803
EM algorithm: 當使用全部的 labeled data，準確率是 0.75676823442

- Analysis on data's size and performance





- Details of experiments

(不同 labeled data-size 的 Round 0 結果就是 Naive Bayes Classifier 的結果)

EM Algorithm Raw Data:
data size: 100
    Round: 0 Accuracy: 0.74668223803
    Round: 1 Accuracy: 0.756449729271
    Round: 2 Accuracy: 0.756662066037
    Round: 3 Accuracy: 0.75676823442
    Round: 4 Accuracy: 0.75676823442
data size: 75
    Round: 0 Accuracy: 0.723856035673
    Round: 1 Accuracy: 0.737127083555
    Round: 2 Accuracy: 0.737445588704
    Round: 3 Accuracy: 0.737233251938
    Round: 4 Accuracy: 0.737127083555
    Round: 5 Accuracy: 0.737127083555
data size: 50
    Round: 0 Accuracy: 0.668966981633
    Round: 1 Accuracy: 0.687864953817
    Round: 2 Accuracy: 0.68903280603
    Round: 3 Accuracy: 0.689138974413
    Round: 4 Accuracy: 0.68903280603
    Round: 5 Accuracy: 0.68903280603
data size: 25
    Round: 0 Accuracy: 0.567363839049
    Round: 1 Accuracy: 0.590614714938
    Round: 2 Accuracy: 0.591570230385
    Round: 3 Accuracy: 0.590402378172
    Round: 4 Accuracy: 0.589553031107
    Round: 5 Accuracy: 0.589553031107
data size: 20
    Round: 0 Accuracy: 0.522136107867
    Round: 1 Accuracy: 0.548465866865
    Round: 2 Accuracy: 0.552075591889
    Round: 3 Accuracy: 0.552712602187
    Round: 4 Accuracy: 0.552924938953
    Round: 5 Accuracy: 0.552924938953
data size: 15
    Round: 0 Accuracy: 0.47531585094
    Round: 1 Accuracy: 0.50695402909
    Round: 2 Accuracy: 0.51077609088
    Round: 3 Accuracy: 0.511094596029
    Round: 4 Accuracy: 0.511094596029
data size: 10

Round: 0 Accuracy: 0.404289202675
Round: 1 Accuracy: 0.449304597091
Round: 2 Accuracy: 0.456524047139
Round: 3 Accuracy: 0.458328909651
Round: 4 Accuracy: 0.458965919949
Round: 5 Accuracy: 0.459390593481
Round: 6 Accuracy: 0.459496761864
Round: 7 Accuracy: 0.459496761864
data size: 5
Round: 0 Accuracy: 0.293449410765
Round: 1 Accuracy: 0.337190784584
Round: 2 Accuracy: 0.347595286124
Round: 3 Accuracy: 0.351311179531
Round: 4 Accuracy: 0.352585200127
Round: 5 Accuracy: 0.353222210426
Round: 6 Accuracy: 0.353434547192
Round: 7 Accuracy: 0.353540715575
Round: 8 Accuracy: 0.353540715575
data size: 3
Round: 0 Accuracy: 0.220193226457
Round: 1 Accuracy: 0.266694978235
Round: 2 Accuracy: 0.284106593057
Round: 3 Accuracy: 0.29185688502
Round: 4 Accuracy: 0.296528293874
Round: 5 Accuracy: 0.299607176983
Round: 6 Accuracy: 0.301836713027
Round: 7 Accuracy: 0.303110733624
Round: 8 Accuracy: 0.304490922603
Round: 9 Accuracy: 0.305127932902
Round: 10 Accuracy: 0.305446438051
Round: 11 Accuracy: 0.305658774817
Round: 12 Accuracy: 0.305871111583
Round: 13 Accuracy: 0.305977279966
Round: 14 Accuracy: 0.305977279966
data size: 1
Round: 0 Accuracy: 0.106699224971
Round: 1 Accuracy: 0.140354602399
Round: 2 Accuracy: 0.156916870156
Round: 3 Accuracy: 0.166047351099
Round: 4 Accuracy: 0.17443465336
Round: 5 Accuracy: 0.179106062215
Round: 6 Accuracy: 0.18101709311
Round: 7 Accuracy: 0.182928124005
Round: 8 Accuracy: 0.18345896592
Round: 9 Accuracy: 0.18345896592

Some techniques in implementation and their impact:

• Vocabulary building

在前面有提過，我是一個一個把檔案讀進來，然後先用空白符號做 split，切出來的每一個字，再去除掉不是英文字母的符號，接著做 lowercase，最後再把做好的 words 依序加進一個 global 的變數中，如果 words 已經出現在該變數中就不必加入，最後就能得到公式中所需要的參數 - vocabulary size (也就是 |V|)

$$P(w_t|c_j;\hat{\theta}) = \frac{1+\sum_{i=1}^{|\mathcal{D}|} N(w_t,d_i)P(y_i=c_j|d_i)}{|V|+\sum_{s=1}^{|V|}\sum_{i=1}^{|\mathcal{D}|} N(w_s,d_i)P(y_i=c_j|d_i)}$$

解析檔案的 function 大概類似這樣：

```
def parseWord(fileContent):
    content = fileContent.split()
    for i in range(0, len(content)):
        vocab = re.findall('[a-zA-Z]+', content[i])
        content[i] = ''.join([k for k in vocab])
        content[i] = content[i].lower()
    return content
```

此外，我也有試著拿掉在整個 corpus 出現太多的字 (拿掉前幾名的字)，但成效不進反退，而且很花時間，最後不採用。

• Smoothing for unseen words

論文中 $P(w_{d_{i,k}} \mid c_j ; \theta)$ 的公式已經有做 smoothing，但我有針對其再做修正：分子的 “1” 以及 分母的 “|V|”，我都會再乘上一個 smoothing 的參數 S，也就是 1 會變成 1*S，|V| 會變成 |V|*S

根據多次測試的結果，S = 0.027 時的 performance 會最好

Observations in the experiment:

根據上面 EM Algorithm 的結果可以得到一些結論

- labeled data size 取越高，初始的 accuracy 也就越高。大致上呈現線性變化。

- labeled data size 取越高，花費越少的 iteration 即可收斂，反之要經過更多次 iteration 才能收斂。(EX: size 100: 4次, size 1: 9次)

- labeled data size 越低，越需要利用 unlabeled data，跑 EM Algorithm 的進步幅度明顯大於 labeled data size 高的組別

  EX:

  size 1: 10.6% -> 18.3%

  size 100: 74.6% -> 75.6%

- 通常 Round 0 到 Round 1 的進步幅度最大，也就是在原本的 Naive Bayes 下做的第一個 iteration，會讓準確率提高最多。後面的 iteration 的進步幅度通常較小。