ソフトウェア工学最終レポート

201811319 永崎遼太[1]

2020.12.28

[1]筑波大学情報学群情報科学類 3 年

目 次

1　対象ソフトウェアプロダクト (プロセス) の説明

ソフトウェアサイエンス主専攻実験「Web 文書分類」にて作成した、ナイーブベイズ (NB) 法により入力文書のクラスを決定するソフトウェアプロダクトを対象とする。

NB 法では、あらかじめ単語の生起確率を求めておく訓練ステップと、求めた生起確率を利用して、入力文書のクラスを決定する分類ステップに分けて考えることができる。

手続き

訓練ステップ 教師文書を利用して各クラス、各単語について $P(w_j, c_i)$ を求める

分類ステップ 入力文書 x に対して、分類クラスを決定する。

1. 入力文書の各単語に対して $N(w_j, x)$ を求める
2. 入力文書の各単語に対して、訓練フェーズの結果を参照して $P(w_j, c_j)$ を得る
3. $N(w_j, x)$ と $P(w_j|c_i)$ の情報から分類クラスを決定する

訓練ステップ、分類ステップをそれぞれ以下の nb_train.py、nb_classification.py で実装している。

```
import MeCab as mc
from collections import Counter
import ast
import math
##### 訓練ステップ
##### 教師文書を利用して、各クラス、各単語についてp(w_j, c_i)を求める
# ----- ----- ----- ----- ----- # 関数・の定義delta
# 特徴抽出結果を受け取り辞書に変換
def str_dic(x):
    d = x
    d1 = '{'+'"'+d.replace(':', '":').replace(' ', ', "') + '}'
    d2 = ast.literal_eval(d1)
    return d2
# smoothing
delta = 1
# ----- ----- ----- # クラスについてcleanerP(wj|ci)を求める
# 教師文書を読み込み
tid = 0
space = {}
for line in open('train.cleaner_tf.txt', 'r'):
    line = line.replace(" \n", "\n")
    line_ = str_dic(line)
    space[tid] = line_
    tid += 1
# 各単語の生起頻度を登録
pwj_cleaner = {}
for id in space.keys():
    for val in space[id].keys():
        if (val not in pwj_cleaner):
            pwj_cleaner[val] = space[id][val]
```

```
        else:
            pwj_cleaner[val] = pwj_cleaner[val]+space[id][val]
# クラスの全単語数を計算cleaner
all_words_cleaner = 0
for id in pwj_cleaner.keys():
    tmp = pwj_cleaner[id] + delta
    all_words_cleaner = all_words_cleaner + tmp
# 最終的なP(wj|ci)を計算
for id in pwj_cleaner.keys():
    tmp = pwj_cleaner[id] + delta
    pwj_cleaner[id] = tmp / (all_words_cleaner)
# P(wj|ci)をファイルに出力
with open('pwj_cleaner.txt', 'w') as f:
  print(pwj_cleaner, file=f)
# ----- ----- ----- # # クラスについてmp3playerP(wj|ci)を求める
# 教師文書を読み込み
tid = 0
space = {}
for line in open('train.mp3player_tf.txt', 'r'):
    line = line.replace(" \n", "\n")
    line_ = str_dic(line)
    space[tid] = line_
    tid += 1
# 各単語の生起頻度を登録
pwj_mp3player = {}
for id in space.keys():
    for val in space[id].keys():
        if (val not in pwj_mp3player):
            pwj_mp3player[val] = space[id][val]
        else:
            pwj_mp3player[val]=pwj_mp3player[val]+space[id][val]
# クラスの全単語数を計算mp3player
all_words_mp3player = 0
for id in pwj_mp3player.keys():
    tmp = pwj_mp3player[id] + delta
    all_words_mp3player = all_words_mp3player + tmp
# 最終的なP(wj|ci)を計算
for id in pwj_mp3player.keys():
    tmp = pwj_mp3player[id] + delta
    pwj_mp3player[id] = tmp / (all_words_mp3player)
# P(wj|ci)をファイルに出力
with open('pwj_mp3player.txt', 'w') as f:
  print(pwj_mp3player, file=f)
```

Listing 1: nb_train.py

```
import ast
import nb_train
import math
##### 分類ステップ
##### 入力文書に対して、分類クラスを決定するx
# ----- ----- ----- ----- ----- # 関数の定義
# 特徴抽出結果を受け取り辞書に変換
def str_dic(x):
    d = x
    d1 = '{'+'"'+d.replace(':', '":').replace(' ', ', "')+'}'
    d2 = ast.literal_eval(d1)
    return d2
# ----- ----- ----- ----- ----- # 1
# 入力文書の各単語に対して、N(w_j, x)を求める
tid = 0
nwjx = {}
```

```
for line in open('test_tf.txt', 'r'):
    line = line.replace(" \n", "\n")
    line_ = str_dic(line)
    nwjx[tid] = line_
    tid += 1
# ----- ----- ----- ----- ----- # 2
# 入力文書の各単語に対して、訓練フェースの結果を参照してp(w_j, c_i)を得る
# 訓練フェーズの結果を参照
pwj_cleaner = nb_train.pwj_cleaner
pwj_mp3player = nb_train.pwj_mp3player
all_words_cleaner = nb_train.all_words_cleaner
all_words_mp3player = nb_train.all_words_mp3player
# クラスに関してcleanerP(wj|ci) を求める
for id in nwjx.keys():
    for val in nwjx[id].keys():
        if (val not in pwj_cleaner):
            pwj_cleaner[val] = 1 / all_words_cleaner
# クラスに関してmp3playerP(wj|ci) を求める
for id in nwjx.keys():
    for val in nwjx[id].keys():
        if (val not in pwj_mp3player):
            pwj_mp3player[val] = 1 / all_words_mp3player
# ----- ----- ----- ----- ----- # 3
#  N(w_j, x)とp(w_j, c_i)の情報から分類クラスを決定する
for id in nwjx.keys():
    val_cleaner = 0
    val_mp3player = 0
    for val in nwjx[id].keys():
        tmp = nwjx[id][val]* math.log(pwj_cleaner[val])
        val_cleaner = val_cleaner + tmp
    for val in nwjx[id].keys():
        tmp = nwjx[id][val]* math.log(pwj_mp3player[val])
        val_mp3player = val_mp3player + tmp
    output_class='cleaner'if val_cleaner > val_mp3player else '
    mp3player'
    print( output_class )
```

Listing 2: nb_classification.py

train.cleaner_tf.txt、train.mp3player_tf.txt という文書を教師文書とし、test_tf.txt というファイルを分類対象のファイルとしている (各ファイルの内容についての説明は割愛)。

2　対象ソフトウェアプロダクト (プロセス) の問題

作成したプログラムでは、まず nb_train.py を実行し、その結果を参照して nb_classification.py を実行するという手順で NB 法を実行する。

交差検定法により実装した NB 法の性能を評価しようとした時、分割されたデータに対して NB 法を繰り返し実行する必要性が生じる。作成したプログラムのままでは、NB 法を繰り返し実行するという処理が困難である。

また、文書を読み込むという機能を逐一プログラムを書いて実行していたり、それぞれで str_df() 関数を作成していたりなど、冗長な部分が見られる。

3　プロダクト (プロセス) の改善

両方のプログラムで必要とされる機能を取り出し、関数としてまとめた新たなプログラム my_function.py を作成した。

```
import ast
num_split = 5
# ----- ----- ----- ----- ----- #
# 特徴抽出結果を受け取り辞書に変換
def str_dic(x):
    d = x
    d1 = '{' + '"' + d.replace(':', '":').replace(' ', ', "').
    replace('"":"', '":"') + '}'
    d2 = ast.literal_eval(d1)
    return d2
# ----- ----- ----- ----- ----- #
# 特徴抽出結果を読み込むtf
def read_tf(text):
    tid = 0
    space = {}
    for line in text:
        line = line.replace(" \n", "\n")
        line_ = str_dic(line)
        space[tid] = line_
        tid += 1
    return space
```

Listing 3: my_function.py

作成したプログラム内の機能を抽象化して関数として再実装し、呼び出せるようにする。それぞれを改善した結果が以下に示す nb_train2.py、nb_classification2.py である。

```
import ast
import my_function
##### 訓練ステップ
##### 教師文書を利用して、各クラス、各単語についてp(w_j, c_i)を求める
# smoothing
delta = 1
# ----- ----- ----- # P(wj|ci)を求める
# あるクラスの教師文書の各単語の生起頻度を計算
def train_count_pwj(data):
    pwj_class = {}
    for id in data.keys():
        for val in data[id].keys():
            if (val not in pwj_class):
                pwj_class[val] = data[id][val]
            else:
                pwj_class[val] = pwj_class[val]+data[id][val]
    return pwj_class
# あるクラスの教師文書の全単語数を計算
def train_count_word(pwj_class):
    all_words_class = 0
    for id in pwj_class.keys():
        tmp = pwj_class[id] + delta
        all_words_class = all_words_class + tmp
    return all_words_class
# 最終的なP(pwj|ci)を計算
def train_pwj_class(pwj_class, all_words_class):
    for id in pwj_class.keys():
```

```
28          tmp = pwj_class[id] + delta
29          pwj_class[id] = tmp / (all_words_class)
30      return pwj_class
```

Listing 4: nb_train2.py

```
1  import ast
2  import math
3  import my_function
4  ##### 分類ステップ
5  ##### 入力文書に対して、分類クラスを決定するx
6  # ----- ----- ----- ----- ----- # 1
7  # 入力文書の各単語に対して、N(w_j, x)を求める
8  # ----- ----- ----- ----- ----- # 2
9  # 入力文書の各単語に対して、訓練フェースの結果を参照してp(w_j, c_i)を得る
10 # あるクラスに関してP(wj|ci) を求める
11 def pwj_class(test_data, pwj_class, all_words_class):
12     for id in test_data.keys():
13         for val in test_data[id].keys():
14             if (val not in pwj_class):
15                 pwj_class[val] = 1 / all_words_class
16     return pwj_class
17 # ----- ----- ----- ----- ----- # 3
18 #  N(w_j, x)とp(w_j, c_i)の情報から分類クラスを決定する
19 def decide_class(test_data, pwj_class1, pwj_class2):
20     output_class = {}
21     tid = 1
22     for id in test_data.keys():
23         val_class1 = 0
24         val_class2 = 0
25         for val in test_data[id].keys():
26             tmp = test_data[id][val]* math.log(pwj_class1[val])
27             val_class1 = val_class1 + tmp
28         for val in test_data[id].keys():
29             tmp = test_data[id][val]* math.log(pwj_class2[val])
30             val_class2 = val_class2 + tmp
31         output_class_ = 'pos' if val_class1 > val_class2 else '
    neg'
32         output_class[tid] = output_class_
33         tid += 1
34     return output_class
```

Listing 5: nb_classification2.py

4　改善のまとめと今後の展望

機能を抽象化して関数として再実装することで、機能を繰り返し実行することが容易になり、またプログラムの冗長性を改善することが可能になった。今回作成したプログラムを用いて交差検定法を実行するプログラムを実際に作成すると、正しく動作をすることが確認できたので、改善を正しく行うことができたことが確かめられた。