

# 別添資料

本統括・分担研究報告書全体に係る別添資料を示す。  
順に、以下の構成となっている。

- 備考欄前処理プログラムソース（主要な部分抜粋）
  - applyregexp.py （以下の処理の実行）
    - ◇ regexp14.py （”死亡の原因” 用）
    - ◇ regexp16.py （”外因死の追加事項” 用）
    - ◇ regexp18.py （”その他特に付言すべきことがら” 用）
    - ◇ regexpOther.py （上記以外の全て用）
  
- 機械学習用データセット作成プログラムソース
  - TFIDF
  - LSI
  - Word2Vec
  - Doc2Vec (PV-DM / PV-DBOW)
  
- 分類器学習プログラムソース
  - fit\_and\_predict\_xgboost.py

## 備考欄前処理プログラム (apply\_regexp.py)

```
1 from my_util import exec_time
2
3 from datetime import datetime
4 import re
5 import regexp14
6 import regexp16
7 import regexp18
8 import regexp0ther
9 import copy
10
11 @exec_time.printer
12 def main():
13     regs14 = regexp14.get()
14     regs16 = regexp16.get()
15     regs18 = regexp18.get()
16     regs0ther = regexp0ther.get()
17     max_cnt = 0
18     fis = open('./01_new_shibo_join_kv.csv', 'r')
19     fos = open('./02_new_shibo_join_kv2' + ( ' ' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '.csv', 'w')
20     f_removed = open('./02_new_shibo_join_removed_words' + ( ' ' if max_cnt <= 0
else '_' + str(max_cnt) ) + '.tsv', 'w')
21
22     cnt = 0
23     for sbh in sbh_data_generator(fis):
24         cnt += 1
25         if cnt == max_cnt: break
26         rowid = sbh['rowid']
27         values = sbh['value']
28         # 14,1: 死亡原因 Iア死因 14,2: 死亡原因 Iア期間 14,3: 死亡原因 Iイ死因 14,4:
死亡原因 Iイ期間 14,5: 死亡原因 Iウ死因 14,6: 死亡原因 Iウ期間 14,7: 死亡原因 Iエ
死因 14,8: 死亡原因 Iエ期間 14,9: 死亡原因 I死因 14,10: 死亡原因 I期間 14,12: 死
死亡原因 手術 14,14: 死亡原因 手術年月日 14,16: 死亡原因 解剖 16,4: 障害が発生したとこ
ろ その他 16,8: 手段及び状況 17,1: 生後1年未満 詳細 18,1: その他付言 99,1: 備考
29         # for sbhnum, sbhnum2 in [( '14', '12'), ( '14', '16'), ( '16', '4'), ( '16',
'8'), ( '17', '1'), ( '18', '1'), ( '99', '1')]:
30         # 18,1: その他付言 99,1: 備考
31         for sbhnum, sbhnum2 in [( '18', '1'), ( '99', '1')]:
32             col = copy.deepcopy(values[sbhnum][sbhnum2])
33             for k in col.keys():
34                 if k=='colnum': continue
35                 value = col[k]
36                 if len(value):
37                     org_value = value
38                     # 14用の正規表現でパース
39                     for r in regs14:
40                         # 14の区切り位置を[col14]で置換する
41                         t = fw_reg(r[0]).subn(r[1], value)
42                         if t[1]: value = t[0]
43                     # 16用の正規表現でパース
44                     for r in regs16:
45                         # 16の区切り位置を[col16]で置換する
46                         t = fw_reg(r[0]).subn(r[1], value)
47                         if t[1]: value = t[0]
48                     # 18用の正規表現でパース
49                     for r in regs18:
50                         # 18の区切り位置を[col18]で置換する
51                         t = fw_reg(r[0]).subn(r[1], value)
52                         if t[1]: value = t[0]
53                     # 上記以外用の正規表現でパース
54                     for r in regs0ther:
55                         # 区切り位置を[colxx_0]で置換する
56                         t = fw_reg(r[0]).subn(r[1], value)
57                         if t[1]: value = t[0]
58                     # スプリッタ ([colxx]) で切り分け
59                     value_arr = fw_reg(r'([\col[^\]]+])([^\($)]*)').findall(value)
60                     if value_arr:
61                         len14_1 = len(values['14']['1']) - 1
62                         len14_2 = len(values['14']['2']) - 1
63                         len14_3 = len(values['14']['3']) - 1
64                         len14_4 = len(values['14']['4']) - 1
65                         len14_5 = len(values['14']['5']) - 1
66                         len14_6 = len(values['14']['6']) - 1
```

```

67 len14_7 = len(values['14']['7']) - 1
68 len14_8 = len(values['14']['8']) - 1
69 len14_9 = len(values['14']['9']) - 1
70 len14_10 = len(values['14']['10']) - 1
71 len14_12 = len(values['14']['12']) - 1
72 len14_14 = len(values['14']['14']) - 1
73 len14_16 = len(values['14']['16']) - 1
74 len16_4 = len(values['16']['4']) - 1
75 len16_8 = len(values['16']['8']) - 1
76 len18_1 = len(values['18']['1']) - 1
77 # OPTIMIZE: いったん組み直して正規表現化することでスプリッタを復元する
78 # NOTE: 行ごとに固有の正規表現を組むためfw_regを使用しない。
79 splitted_value_regexp = '(' + fw_reg(r'\[col[^\]]+\)').sub(r')
(*.*?', value) + ')'
80 splitted_value_arr = re.findall(splitted_value_regexp, org_value)
81 for i, (splitter, v) in enumerate(value_arr):
82     # 連結前の整形
83     tmp = fw_reg(r'^((は)?「(.*)」(である)?(。)?$)').search(v)
84     if tmp: v = tmp.groups()[1]
85     v = fw_reg(r'((続<))?\n?$').sub('', v)
86     if splitter == '[col18_top]':
87         sbh['value']['18']['1'][len18_1] = v
88         len18_1 += 1
89     elif splitter == '[col18]':
90         sbh['value']['18']['1'][len18_1 + 100] = v
91         len18_1 += 1
92     #elif splitter == '[col16_4]':
93     # sbh['value']['16']['4'][len16_4] = v
94     # len16_4 += 1
95     elif splitter == '[col16]' or splitter == '[col16_8]':
96         sbh['value']['16']['8'][len16_8] = v
97         len16_8 += 1
98     elif splitter == '[col14_1]':
99         sbh['value']['14']['1'][len14_1] = v
100        len14_1 += 1
101    elif splitter == '[col14_2]':
102        sbh['value']['14']['2'][len14_2] = v
103        len14_2 += 1
104    elif splitter == '[col14_3]':
105        sbh['value']['14']['3'][len14_3] = v
106        len14_3 += 1
107    elif splitter == '[col14_4]':
108        sbh['value']['14']['4'][len14_4] = v
109        len14_4 += 1
110    elif splitter == '[col14_5]':
111        sbh['value']['14']['5'][len14_5] = v
112        len14_5 += 1
113    elif splitter == '[col14_6]':
114        sbh['value']['14']['6'][len14_6] = v
115        len14_6 += 1
116    elif splitter == '[col14_7]':
117        sbh['value']['14']['7'][len14_7] = v
118        len14_7 += 1
119    elif splitter == '[col14_8]':
120        sbh['value']['14']['8'][len14_8] = v
121        len14_8 += 1
122    elif splitter == '[col14_9]':
123        sbh['value']['14']['9'][len14_9] = v
124        len14_9 += 1
125    elif splitter == '[col14_10]':
126        sbh['value']['14']['10'][len14_10] = v
127        len14_10 += 1
128    elif splitter == '[col14_12]':
129        sbh['value']['14']['12'][len14_12] = v
130        len14_12 += 1
131    elif splitter == '[col14_14]':
132        sbh['value']['14']['14'][len14_14] = v
133        len14_14 += 1
134    elif splitter == '[col14_16]':
135        sbh['value']['14']['16'][len14_16] = v
136        len14_16 += 1
137    else: # ここまでに合致しないスプリッタの文言は捨てる
138        f_removed.write(rowid + '\t' + splitter + '\t' +

```

```

139 splited_value_arr[0][i+1].rstrip('\n') + '\n')
140     # 切り分けた先頭は元の欄に残す
141     try:
142         sbh['value'][sbhnum][sbhnum2][k] = fw_reg(r'^([\^
143         [!+)]).search(value).group() + '\n'
144     except AttributeError:
145         sbh['value'][sbhnum][sbhnum2][k] = '\n'
146     sbh_data_writer(fos, sbh)
147
148     fis.close()
149     fos.close()
150     f_removed.close()
151
152 def sbh_data_generator(fis):
153     l = fis.readline()
154     bef_rowid = l.split(',')[0]
155     sbh = {'rowid': bef_rowid, 'value': {}}
156     while l!='':
157         rowid, colnum, sbhnum, sbhnum2, colrownum, value = l.split(',')
158         if bef_rowid != rowid:
159             yield sbh
160             sbh = {'rowid': rowid, 'value': {}}
161             # キーを変更してdictにする
162             # sbh.update({'rowid': rowid, 'value': {'sbhnum': sbhnum, 'value':
163             {'sbhnum2': sbhnum2, 'sbhnum': sbhnum, 'value': {sbhrownum: value }}}})
164             # sbh.update({'rowid': rowid, 'value': {sbhnum: {sbhnum2: {'colnum':
165             colnum, colrownum: value }}}})
166         if not sbhnum in sbh['value']: sbh['value'][sbhnum] = {}
167         if not sbhnum2 in sbh['value'][sbhnum]: sbh['value'][sbhnum][sbhnum2] =
168         {'colnum': colnum}
169         sbh['value'][sbhnum][sbhnum2][colrownum] = value
170         bef_rowid = rowid
171
172     l = fis.readline()
173     yield sbh
174
175 def sbh_data_writer(fos, sbh):
176     # rowid      : new_shibo_join.tsvの先頭カラム、行ごとのID
177     # colnum     : new_shibo_join.tsvのカラム番号
178     # sbhnum     : 死亡診断書の欄番号。new_shibo_join.tsvには存在しない。
179     # sbhnum2    : 死亡診断書の欄番号の枝番号。new_shibo_join.tsvには存在しない。
180     # colrownum : 死亡診断書の欄番号、枝番号ごとの内容が複数行に渡る場合の行番号。
181     # value      : new_shibo_join.tsvの2カラム目以降の値。
182     rowid = sbh['rowid']
183     out_arr = []
184     for sbhnum, sbhnum_col in [(str(sbhnum), sbh['value'][str(sbhnum)]) for
185     sbhnum in sbh['value'].keys()]:
186         if sbhnum == 'rowid': continue
187         for sbhnum2, sbhnum2_col in [(sbhnum2, sbhnum_col[sbhnum2]) for sbhnum2
188     in sbhnum_col.keys()]:
189             colnum = sbhnum2_col['colnum']
190             for colrownum, value in [(colrownum, sbhnum2_col[colrownum]) for
191     colrownum in sbhnum2_col.keys()]:
192                 if colrownum == 'colnum': continue
193                 out_arr.append([rowid, colnum, sbhnum, sbhnum2, str(colrownum),
194     value])
195     out_arr.sort(key=lambda item: (int(item[1]), int(item[4])))
196
197     bef_colnum = out_arr[0][1]
198     l = ','.join(out_arr[0])
199     for l_arr in out_arr[1:]:
200         if bef_colnum != l_arr[1]:
201             fos.write(fw_reg(r'(\s*)?(続<))?\n?$').sub(' ', l) + '\n')
202             l = ','.join(l_arr)
203             bef_colnum = l_arr[1]
204         else:
205             l = l.rstrip('\n') + l_arr[5]
206     fos.write(l)
207
208 fw_reg_cache = {}
209 def fw_reg(p):
210     """
211     初めて取得するパターンはコンパイルして返す。

```

```
203  初めてでないパターンはキャッシュから返す。
204  """
205  try:
206      return fw_reg_cache[p]
207  except KeyError:
208      r = re.compile(p)
209      fw_reg_cache[p] = r
210      return r
211
212
213  if __name__ == '__main__':
214      main()
215
216
217
```

# 備考欄前処理プログラム (regexp14.py)

```

1 def get():
2     return [[r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(ア)+( [ ] ) ])?(の| |) )?
3     (死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: , ---) ])?',r'[col14_1]'],
4     [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(ア)+( [ ] ) ])?(死亡までの|
5     の| )?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: ,
6     ---) ])?',r'[col14_2]'],
7     [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(イ)+( [ ] ) ])?(の| |) )?(死因)+(
8     欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: , ---) ])?',r'[col14_3]'],
9     [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(イ)+( [ ] ) ])?(死亡までの|
10    の| )?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: ,
11    ---) ])?',r'[col14_4]'],
12    [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(ウ)+( [ ] ) ])?(の| |) )?(死因)+(
13    欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: , ---) ])?',r'[col14_5]'],
14    [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(ウ)+( [ ] ) ])?(死亡までの|
15    の| )?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: ,
16    ---) ])?',r'[col14_6]'],
17    [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(エ)+( [ ] ) ])?(の| |) )?(死因)+(
18    欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: , ---) ])?',r'[col14_7]'],
19    [r' (?14) ?([----- 「 ( )?(1|1|I|i)?([ 「 ( )?(エ)+( [ ] ) ])?(死亡までの|
20    の| )?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: ,
21    ---) ])?',r'[col14_8]'],
22    [r' (?14) ?([----- 「 ( )?(2|2|㍻|㍺)+( [ ] ) ])?(の| )?(死因)+(欄|ラン|らん)?
23    (続き|つづき)?(は)?([ 、。・: , ---) ])?(傷病名)?',r'[col14_9]'],
24    [r' (?14) ?([----- 「 ( )?(2|2|㍻|㍺)+( )?( [ ] ) ])?(死亡までの|の| )?(期間|年
25    月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([ 、。・: , ---) ])?(傷病
26    名)?',r'[col14_10]'],
27    [r' (?14) ?(欄中)??([----- , ])?(の| )?(解剖)+(欄|ラン|らん| )?(主要所見|所
28    見)?(の| )?(続き|つづき)?(は)?([ 、。・: , ---) ])?', r'[col14_16]'],
29    [r' (?14) ?(欄中)??([----- , ])?(の| )?(手術|手術部位及び主要所見追加)+(?!解
30    剖|期間|年月日|年月|年|月日|月|日|施行日)(欄|ラン|らん)?(の| )?(続き|つづき)?(は)?
31    ([ 、。・: , ---) ])?', r'[col14_12]'],
32    [r' (?14) ?(欄中)??([----- 「 ( , ])?(の| )?(手術期間|手術年月日|手術年月|手術
33    年|手術月日|手術月|手術日|手術施行日)+(欄)?( [ ] ) ])?(の| )?(続き|つづき)?(は)?
34    ([ 、。・: , ---) ])?', r'[col14_14]'],
35    [r' ( (14) )+(欄中)??(の| )?( [----- ( , ])?(期間の記載|死亡したとき)?(続き|つづ
36    き)?(は)?([ 、。・: , ---) ])?',r'[col14]'],
37    [r' (14) (1|1|I|i|2|2|㍻|㍺)?([----- , ])?(「 (ア) 直接原因」 | 「 (イ) (ア) の
38    原因」 | 「 (ウ) (イ) の原因」 | 「 (エ) (ウ) の原因」 | 「I欄に影響を及ぼした傷病名等」)?
39    (の| )?(「死亡までの期間」)+( )?',r'[col14]'],
40    [r' (?14) ?(手術・解剖|手術解剖)+( )?',r'[col14]'],
41    [r' (?141(1|2|3|4|5)? )?(欄|ラン|らん)?(続き|つづき)?(続)?(は)?
42    ([ 、。・: , ---) ])?',r'[col14]']]#分類できないものはこのキーに置く
43    #記号控え([ 、。・: , ---) ]
44
45 def _assert(r, s):
46     import re
47     # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
48     print('開始[' + s + ']')
49     result = ''
50     for rr in r:
51         t = re.subn(rr[0], rr[1], s)
52         if t[1]:
53             if len(result.replace('0', '')): print('経過[' + s + ']')
54             s = t[0]
55             result += '1'
56         else:
57             result += '0'
58     # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
59     for rr in r]))
60     print('結果[' + s + ']\n')
61     # print([result[0] + '\n' for result in l])
62     return len(result.replace('0', ''))
63
64 if __name__ == '__main__':
65     r = get();
66
67     print('開始 -----')
68
69     print('===== 終了')
```

## 備考欄前処理プログラム (regexpl6.py)

```

1 def get():
2     return [[r' (?1 6) ?(欄中?)?( )? ?([の ( )]? (続き|続|つづき|追記)([、・:、--
3     -) ])?', r'[col16]'], #16の情報として抽出
4     [r' (?1 6) ?(欄中?)?([---の「」)?(傷害|障害|死亡)?(発生|が発生した|した)+(日時|時
5     刻|時分|とき|時間)+(欄)?([の ( )「」]? (続き|続|つづき|追記|時分)?(は)?([、・:、--
6     -) ])?', r'[col16_4]'], #傷害が発生した時刻として抽出
7     [r' (?1 6) ?(欄中?)?([---の「」)?(傷害|障害|死亡)?(発生|が発生した|した)+(日|時)+
8     (欄)?([の ( )「」]? (続き|続|つづき|追記|時分)?(は)?([、・:、---) ])?',
9     r'[col16_4]'], #傷害が発生した時刻として抽出
10    [r' (1 6) (欄中?)?(「」)?(手段及び状況|状況|手段および状況)?(欄)?([の ( )「」]? (続き|
11    続|つづき|追記)?(において|は)?([、・:、---) ])?', r'[col16_8]'], #手段及び状況とし
12    て抽出と結合
13    [r' (?1 6 0 5) ?([の ( )]? (続き|続|つづき|追記)?(は)?([、・:、---) ])?',
14    r'[col16_8]'], #手段及び状況として抽出と結合
15    [r' 「?1 6 枠外」?([、・:、---) ])?', r'[col16]'], #16の情報として抽出
16    [r' (1 6 追加) ', r'[col16]'], #16の情報として抽出
17    [r' 1 6 [ ] ; 欄]', r'[col16]'], #16の情報として抽出
18    [r' 1 6 (欄)?(---)', r'[col16]'], #16の情報として抽出
19    [r' (外因死)?(の)?(追加事項)? : ?(手段及び状況)+(欄)?(について)?(の)?(続き|続|つづ
20    き|追記)?(は)?([、。・:、---) ])?', r'[col16_8]'] #手段及び状況として抽出と結合
21
22 def _assert(r, s):
23     import re
24     # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
25     print('開始[' + s + ']')
26     result = ''
27     for rr in r:
28         t = re.subn(rr[0], rr[1], s)
29         if t[1]:
30             if len(result.replace('0', '')): print('経過[' + s + ']')
31             s = t[0]
32             result += '1'
33         else:
34             result += '0'
35     # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
36     for rr in r]))
37     print('結果[' + s + ']\n')
38     # print([result[0] + '\n' for result in l])
39     return len(result.replace('0', ''))
40
41 if __name__ == '__main__':
42     r = get();
43
44     print('開始 -----')
45
46     print('----- 終了')
```

## 備考欄前処理プログラム (regexp18.py)

```

1 def get():
2     return [[r'18続き、', r'[col18_top]'],
3             [r'(?18)?(欄)?( )?( [の ( ])?(続き|続|つづき|追記|:)( [、・:、---) ])?',
4             r'[col18]'],
5             [r'(?18)?(欄)?( [--- ])?(その他|その他特に付言すべき事柄)(欄)?( [の ( ])?(続き|
6             続|つづき|追記)?( [、・:、---) ])?', r'[col18]'],
7             [r'(18(欄)?(欄)?( [の ( ])?(続き|続|つづき|追記)?( [、・:、---) ])?',
8             r'[col18]'],
9             [r'(?1801)?( [の ( ])?(続き|続|つづき|追記)?( [、・:、---) ])?',
10            r'[col18]'],
11            [r'「?18枠外」?( [、・:、---) ])?', r'[col18]'],
12            [r'(18その他特に付言すべきことから)', r'[col18]'],
13            [r'18その他特に付言すべきことからの続きは、', r'[col18]'],
14            [r'(18追加)', r'[col18]'],
15            [r'18[ ) ;欄]', r'[col18]'], ]
16
17 def _assert(r, s):
18     import re
19     # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
20     print('開始[' + s + ']')
21     result = ''
22     for rr in r:
23         t = re.subn(rr[0], rr[1], s)
24         if t[1]:
25             if len(result.replace('0', '')): print('経過[' + s + ']')
26             s = t[0]
27             result += '1'
28         else:
29             result += '0'
30     # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
31     for rr in r]))
32     print('結果[' + s + ']\n')
33     # print([result[0] + '\n' for result in l])
34     return len(result.replace('0', ''))
35
36 if __name__ == '__main__':
37     r = get();
38
39     print('開始 -----')
40
41     print('===== 終了')
42
43

```

## 備考欄前処理プログラム (regexpOther.py)

```
1 def get():
2     return [# (3) 生年月日
3         [r'(3)(欄|続き)?', r'[col03_0]'],
4         [r'(?<1)3(続き|続|つづき)[、・:、---]?', r'[col03_0]'],
5         [r'(3(続き|続|つづき))', r'[col03_0]'],
6         # (4) 死亡したとき
7         [r'(4)死亡したとき[、・:、---]?', r'[col04_0]'],
8         [r'(4)(欄|続き)?', r'[col04_0]'],
9         [r'(?<1)4(続き|続|つづき)[、・:、---]?', r'[col04_0]'],
10        [r'(4(続き|続|つづき))', r'[col04_0]'],
11        # (5) 死亡したところ
12        [r'(5)(欄|続き)?', r'[col05_0]'],
13        [r'(?<1)5(続き|続|つづき)[、・:、---]?', r'[col05_0]'],
14        [r'(5(続き|続|つづき))', r'[col05_0]'],
15        # (6) 住所
16        [r'(6)(欄|続き)?', r'[col06_0]'],
17        [r'(?<1)6(続き|続|つづき)[、・:、---]?', r'[col06_0]'],
18        [r'(6(続き|続|つづき))', r'[col06_0]'],
19        # (7) 本籍
20        [r'(7)(欄|続き)?', r'[col07_0]'],
21        [r'(?<1)7(続き|続|つづき)[、・:、---]?', r'[col07_0]'],
22        [r'(7(続き|続|つづき))', r'[col07_0]'],
23        # (8) (9) 死亡した人の夫または妻
24        [r'(8)(欄|続き)?', r'[col08_0]'],
25        [r'(?<1)8(続き|続|つづき)[、・:、---]?', r'[col08_0]'],
26        [r'(8(続き|続|つづき))', r'[col08_0]'],
27        [r'(9)(欄|続き)?', r'[col09_0]'],
28        [r'(?<1)9(続き|続|つづき)[、・:、---]?', r'[col09_0]'],
29        [r'(9(続き|続|つづき))', r'[col09_0]'],
30        # (10) (11) 死亡したときの世帯の主な仕事と死亡した人の職業・産業
31        [r'(10)(欄|続き)?', r'[col10_0]'],
32        [r'10(続き|続|つづき)[、・:、---]?', r'[col10_0]'],
33        [r'(10(続き|続|つづき))', r'[col10_0]'],
34        [r'(11)(欄|続き)?', r'[col11_0]'],
35        [r'11(続き|続|つづき)[、・:、---]?', r'[col11_0]'],
36        [r'(11(続き|続|つづき))', r'[col11_0]'],
37        # (12) (13) 死亡したところ、及びその種別
38        [r'(?12)?[---]施設名称?(続き|続|つづき)[、・:、---]?', r'[col12_0]'],
39        [r'(12)死亡したところ[、・:、---]?', r'[col12_0]'],
40        [r'(12)死亡したとき[、・:、---]?', r'[col12_0]'],
41        [r'(12)(欄|続き)?', r'[col12_0]'],
42        [r'12(続き|続|つづき)[、・:、---]?', r'[col12_0]'],
43        [r'(12(続き|続|つづき))', r'[col12_0]'],
44        [r'(13)死亡したところ[、・:、---]?', r'[col12_0]'],
45        [r'(13)死亡したとき[、・:、---]?', r'[col12_0]'],
46        [r'(13)(欄|続き)?', r'[col13_0]'],
47        [r'13(続き|続|つづき)[、・:、---]?', r'[col13_0]'],
48        [r'(13(続き|続|つづき))', r'[col13_0]'],
49        # (17) 生後1年未満で病死した場合の追加事項
50        [r'(17)欄?欄妊娠・分娩時における母体の病態又は異状は?', r'[col17_0]'],
51        [r'(17)(欄|続き)?', r'[col17_0]'],
52        [r'17(続き|続|つづき)[、・:、---]?', r'[col17_0]'],
53        [r'(17(続き|続|つづき))', r'[col17_0]'],
54    ]
55 ]
56
57 def _assert(r, s):
58     import re
59     # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
60     print('開始[' + s + ']')
61     result = ''
62     for rr in r:
63         t = re.subn(rr[0], rr[1], s)
64         if t[1]:
65             if len(result.replace('0', '')): print('経過[' + s + ']')
66             s = t[0]
67             result += '1'
68         else:
69             result += '0'
70     # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
71     # for rr in r]))
72     print('結果[' + s + ']\n')
73     # print([result[0] + '\n' for result in l])
```

```
73 return len(result.replace('0', ''))
74
75
76 if __name__ == '__main__':
77     r = get();
78
79     print('開始 -----')
80
81     print('===== 終了')
82
83
84
85
86
```

# 機械学習用データセット作成プログラム (TF・IDF)

```
1 from datetime import datetime
2 from gensim import corpora
3 from gensim import models
4 from gensim import matutils
5 import MeCab
6 import math
7 import re
8 import sys
9 import os
10
11 def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
12     return add + math.log(1.0 * totaldocs / docfreq, log_base)
13
14 def arg_below():
15     import argparse
16     p = argparse.ArgumentParser()
17     p.add_argument('-a', '--anybelow', help='単語出現回数下限の指定、未入力の場合
18     Noneで引数を取り、100で処理')
19     a = p.parse_args()
20     return a.anybelow
21
22 def main(any_below = 100, mode = 0):
23     max_cnt = 0
24     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 開始しま
25     す。')
26     #単語の出現回数の下限、Noneの場合は100
27     below = int(100 if any_below == None else any_below)
28     #print(below)
29
30     #疎行列ベクトル出力先ディレクトリ作成
31     dirname = ''
32     if any_below != None:
33         dirname = 'TFIDF'+ ( ' ' if below <= 0 else '_' + str(below) ) + '/'
34         #os.makedirs(dirname,mode=511,exist_ok=True)
35
36     #読み込みファイルと出力ファイル
37     fis = open('./new_shibo_join_concat.tsv','r')
38     fos = open('./' + dirname + '05' + ( ' ' if max_cnt <= 0 else '_' +
39     str(max_cnt) ) + '_sogyoretu.csv','w')
40     fos2 = open('./' + dirname + '05' + ( ' ' if max_cnt <= 0 else '_' +
41     str(max_cnt) ) + '_total_wordnum.txt','w')
42
43     #作成済み辞書の読み込み先リンク
44     gdic_fname = './' + dirname + 'new_shibo_join.dic'
45
46     #トークナイザを取得
47     tokenizer = get_tokenizer()
48
49     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 分かち書き開
50     始')
51     #トークンリスト（複数行の文章、単語の二次元配列）の読み込み
52     rownum_list,tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
53     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 分かち書き終
54     了')
55
56     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] dictionary
57     作成開始')
58     #トークンリストを辞書とするか、既存の辞書を読み込むか、モードで切り替え
59     if mode == 0:
60         #トークンリストを辞書変換
61         tokens_gdic = tokenslist2dic(tokens_list)
62         #辞書情報を保存
63         #save_gdic(tokens_gdic, gdic_fname)
64     elif mode == 1:
65         #辞書情報の読み込み
66         tokens_gdic = load_gdic(gdic_fname)
67
68     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] dictionary
69     作成終了')
70
71     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] corpus作成
72     開始')
73     #辞書をbag-of-words形式のリスト (corpus) に変換
```

```

65 tokens_corpus = doc2bow(tokens_gdic, tokens_list)
66 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] corpus作成
終了')
67
68 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] model作成開
始')
69 test_model = models.TfidfModel(tokens_corpus, wglobal=new_idf)
70
71 corpus_tfidf = test_model[tokens_corpus]
72 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] model作成終
了')
73
74 #0回出現の単語についても重要度0を出力する
75 #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for
c in id_list]
76 #辞書の総単語数を取得
77 total_wordnum = 0
78 for word,wordid in tokens_gdic.token2id.items():
79     total_wordnum += 1
80 fos2.write(str(total_wordnum))
81 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] ベクトル出力
開始')
82 for rowid,doc in zip(rownum_list, corpus_tfidf):
83     lil = []
84     for word in doc:
85         lil.append((word[0], word[1]))
86
87     fos.write(data_make(rowid, lil))
88 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] ベクトル出力
終了')
89
90 fis.close()
91 fos.close()
92 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 終了しま
す。')
93
94 #出力の形式を整えるメソッド
95 def data_make(rowid, lil):
96     return rowid + '\t['+', '.join([sogyoretu(kv) for kv in lil])+']\n'
97
98 #tfidf値を疎行列の形にして返すメソッド
99 def sogyoretu(kv):
100     return str(kv[0])+':'+str(kv[1])
101
102 #for rowdic,rowid in zip(new_dic,id_list):
103     #print(rowid+'\t', ','.join([str(k)+' '+str(v) for k,v in
sorted(rowdic.items())]))
104     #print(rowid+'\t', ','.join([str(v) for k,v in rowdic.items()]))
105     #fos.write(rowid+'\t', ','.join([str(v) for k,v in
sorted(rowdic.items())])+'\n')
106
107     """"
108     #出現0回の文字について重要度0を入れない出力
109     texts_tfidf = []
110     for rowid,doc in zip(id_list, corpus_tfidf):
111         text_tfidf = []
112         for word in doc:
113             text_tfidf.append(word[1])
114             texts_tfidf.append(text_tfidf)
115         print(rowid+'\t', ','.join([str(i) for i in text_tfidf]))
116     """"
117
118 #作成済みdictionaryを読み込むメソッド
119 def load_gdic(fname):
120     return corpora.Dictionary.load(fname)
121
122 #作成したdictionaryを保存するメソッド
123 def save_gdic(tokens_gdic, fname):
124     tokens_gdic.save(fname)
125
126 #形態素解析にMeCabを使用し、トークナイザーを取得するメソッド
127 def get_tokenizer():
128     return MeCab.Tagger('-r /etc/mecabrc -Owakati')

```



# 機械学習用データセット作成プログラム (LSI)

```
1 from datetime import datetime
2 from gensim import corpora
3 from gensim import models
4 from gensim import matutils
5 import MeCab
6 import math
7 import re
8 import os
9
10 def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
11     return add + math.log(1.0 * totaldocs / docfreq, log_base)
12
13 def main(mode = 0):
14     max_cnt = 0
15     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 開始しま
16 す。')
17     #疎行列ベクトル出力先ディレクトリ作成
18     dirname = 'LSI/'
19     os.makedirs(dirname,mode=511,exist_ok=True)
20
21     #読み込みファイルと出力ファイル
22     fis = open('./new_shibo_join_concat.tsv','r')
23     fos = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt) )
24 + '05_sogyoretu.csv','w')
25     fos2 = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt)
26 ) + '05_total_wordnum.txt','w')
27
28     #作成済み辞書の読み込み先リンク
29     gdic_fname = './new_shibo_join.dict'
30
31     #トークナイザを取得
32     tokenizer = get_tokenizer()
33
34     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 分かち書き開
35 始')
36     #トークンリスト（複数行の文章、単語の二次元配列）の読み込み
37     rownum_list,tokens_list = fetch_tokenlist(fis, tokenizer, max_cnt)
38     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] 分かち書き終
39 了')
40
41     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] dictionary
42 作成開始')
43     #トークンリストを辞書とするか、既存の辞書を読み込むか、モードで切り替え
44     if mode == 0:
45         #トークンリストを辞書変換
46         tokens_gdic = tokenslist2dic(tokens_list)
47         #辞書情報を保存
48         #save_gdic(tokens_gdic, gdic_fname)
49     elif mode == 1:
50         #辞書情報の読み込み
51         tokens_gdic = load_gdic(gdic_fname)
52
53     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] dictionary
54 作成終了')
55
56     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] corpus作成
57 開始')
58     #辞書をbag-of-words形式のリスト (corpus) に変換
59     tokens_corpus = doc2bow(tokens_gdic, tokens_list)
60     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] corpus作成
61 終了')
62
63     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] model作成開
64 始')
65     test_model = models.TfidfModel(tokens_corpus,wglobal=new_idf)
66
67     corpus_tfidf = test_model[tokens_corpus]
68     print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + '] model作成終
69 了')
70
71     #0回出現の単語についても重要度0を出力する
72     #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for
```

```

c in id_list]
63 #辞書の総単語数を取得
64 """
65 total_wordnum = 0
66 for word,wordid in tokens_gdic.token2id.items():
67     total_wordnum += 1
68 fos2.write(str(total_wordnum))
69 """
70 #LSI modelによる次元圧縮したcorpusの作成
71 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI開始')
72 lsi_model = models.LsiModel(corpus_tfidf, id2word=tokens_gdic,
num_topics=200)
73 lsi_model.save('lsi_topics200.model')
74 #lsi_model = models.LsiModel.load('lsi_topics200.model')
75 corpus_lsi = lsi_model[corpus_tfidf]
76 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI終了')
77
78 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
開始')
79 vec_size = 0
80 for rowid,doc in zip(rownum_list,corpus_lsi):
81     lil = []
82     for word in doc:
83         lil.append((word[0],word[1]))
84
85     if len(lil) != 0:
86         vec_size = len(lil)
87         fos.write(data_make(rowid,lil))
88     fos2.write(str(vec_size))
89 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
終了')
90
91 fis.close()
92 fos.close()
93 fos2.close()
94 print('[ '+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']終了しま
す。')
95
96 #出力の形式を整えるメソッド
97 def data_make(rowid,lil):
98     return rowid + '\t['+', '.join([sogyoretu(kv) for kv in lil])+']\n'
99
100 #tfidf値を疎行列の形にして返すメソッド
101 def sogyoretu(kv):
102     return str(kv[0])+':'+str('{:f}'.format((kv[1]+1)/2))
103
104 #for rowdic,rowid in zip(new_dic,id_list):
105     #print(rowid+'\t',','.join([str(k)+' '+str(v) for k,v in
sorted(rowdic.items())]))
106     #print(rowid+'\t',','.join([str(v) for k,v in rowdic.items()]))
107     #fos.write(rowid+'\t'+','.join([str(v) for k,v in
sorted(rowdic.items())])+'\n')
108
109 """
110 #出現0回の文字について重要度0を入れない出力
111 texts_tfidf = []
112 for rowid,doc in zip(id_list,corpus_tfidf):
113     text_tfidf = []
114     for word in doc:
115         text_tfidf.append(word[1])
116     texts_tfidf.append(text_tfidf)
117     print(rowid+'\t',','.join([str(i) for i in text_tfidf]))
118 """
119
120 #作成済みdictionaryを読み込むメソッド
121 def load_gdic(fname):
122     return corpora.Dictionary.load(fname)
123
124 #作成したdictionaryを保存するメソッド
125 def save_gdic(tokens_gdic, fname):
126     tokens_gdic.save(fname)
127
128 #形態素解析にMeCabを使用し、トークナイザーを取得するメソッド

```

```

129 def get_tokenizer():
130     return MeCab.Tagger('-r /etc/mecabrc -Owakati')
131
132 #解析データを読み込み分かち書きしてlistを作成するメソッド
133 def fetch_tokenlist(fis, tokenizer, max_cnt = 0):
134     cnt = 1
135     l = fis.readline();
136     rownum_list = []
137     tokens_list = []
138     while (cnt != max_cnt and l != ''):
139         rownum, sentences = remove_tagwords(l).split('\t')
140         rownum_list.append(rownum)
141         #トークナイズ (1文章を単語ごとに分割した一次元配列)
142         tokens = tokenizer.parse(sentences).split()
143         #トークンリスト (複数行の文章、単語の二次元配列)
144         tokens_list.append(tokens)
145         cnt += 1
146         l = fis.readline();
147
148     return (rownum_list, tokens_list)
149
150 re_tagwords = re.compile(r'([【手術】]|([【解剖】]|([【状況】]|([【母体】]|([【付言】]|
([【備考】]|)\n'))))')
151 #解析データの項目名を単語から除去
152 def remove_tagwords(sentences):
153     return re_tagwords.sub('', sentences)
154
155 #トークンリストを辞書変換するメソッド
156 def tokenlist2dic(tokens_list):
157     return corpora.Dictionary(tokens_list)
158
159 #辞書をbag-of-words形式のリストに変換するメソッド
160 def doc2bow(tokens_gdic, tokens_list):
161     return list(map(tokens_gdic.doc2bow, tokens_list))
162
163 if __name__ == '__main__':
164     main(1)
165
166
167

```

## 機械学習用データセット作成プログラム (Word2Vec)

```
1 import sys
2 #sys.path.append('/home/bmiwork/ITEC')
3 from my_utils import exec_time
4
5 import MeCab
6 import re
7 from gensim.models import word2vec
8 from gensim import corpora
9 #from gensim import matutils
10 import numpy as np
11 import os
12
13 @exec_time.printer
14 def main(mode = 0):
15     max_cnt = 0
16
17     #疎行列ベクトル出力先ディレクトリ作成
18     dirname = 'WORD2VEC/'
19     os.makedirs(dirname,mode=511,exist_ok=True)
20
21     #トークナイザを取得
22     tokenizer = get_tokenizer()
23
24     #読み込みファイルと出力ファイル
25     fis = open('./new_shibo_join_concat.tsv','r')
26     fos = open('./'+ dirname + ( ' ' if max_cnt <= 0 else '_' + str(max_cnt) )
+ '05_sogyoretu.csv','w')
27     fos2 = open('./'+ dirname + ( ' ' if max_cnt <= 0 else '_' + str(max_cnt)
) + '05_total_wordnum.txt','w')
28
29     #作成済みmodelの出力先リンク
30     #vec_fname = './new_shibo_join' + ( ' ' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '.vec.pt'
31     vec_fname = "./WORD2VEC/entity_vector.model.bin"
32
33     #対象文書の形態素を格納
34     rownum_list, tokens_list = fetch_tokenlist(fis, tokenizer, max_cnt)
35     #print(tokens_list[0])
36
37     if mode == 0:
38         #word2vecのmodelに形態素を学習
39         model = fit_word2vec(tokens_list)
40         #modelを保存
41         save_model(model, vec_fname)
42     elif mode == 1:
43         model = load_model(vec_fname)
44
45     #各行の文書のベクトルを計算して出力
46     write_vector(fos, fos2, model, rownum_list, tokens_list)
47
48     fis.close()
49     fos.close()
50     fos2.close()
51
52 #word2vec学習モデルの読み込み
53 def load_model(fname):
54     return word2vec.KeyedVectors.load_word2vec_format(fname, binary=True)
55
56 #word2vec学習モデルの保存
57 def save_model(model, fname):
58     model.wv.save_word2vec_format(fname, binary=True)
59
60 @exec_time.printer
61 def write_vector(fos, fos2, model, rownum_list, tokens_list):
62     #単語ベクトルを平均した値を文書ベクトルとして扱った場合の出力
63     num_features = 200
64     for rowid, doc in zip(rownum_list,tokens_list):
65         #print(rowid)
66         #print(doc[0])
67         #単語を200次元のベクトルとして出力したもの
68         """"
69         vec_list = []
70         for word in doc:
```

```

71         if word in model.key_to_index:
72             vec = model[word]
73             vec_list.append(vec)
74         fos.write(data_make(rowid, doc, vec_list))
75         """"
76         #docが空でない場合のみベクトルの平均を作成、空の場合は空のリストを作成
77         if doc != []:
78             feature_vec = np.zeros((num_features), dtype= "float32")
79             for word in doc:
80                 if word in model.key_to_index:
81                     feature_vec = np.add(feature_vec, model[word])
82                     #feature_vec = np.add(feature_vec, model.get_vector(word,
norm=True))
83             if len(doc) > 0:
84                 feature_vec = np.divide(feature_vec, len(doc))
85             else:
86                 feature_vec = []
87             #print(feature_vec)
88             fos.write(data_make2(rowid, feature_vec, num_features))
89             fos2.write(str(num_features))
90
91 #出力の形式を整えるメソッド
92 def data_make(rowid, doc, vec_list):
93     return rowid + '\t['+ ', '.join([sogyoretu(wordid,vec) for wordid, vec in
zip(doc, vec_list)])+']\n'
94
95 #出力の形式を整えるメソッド
96 def data_make2(rowid, feature_vec, vec_cnt):
97     cnt_list = []
98     for num in range(vec_cnt):
99         cnt_list.append(num)
100     if feature_vec != []:
101         return rowid + '\t['+ ', '.join([sogyoretu2(v,i) for v,i in
zip(feature_vec, cnt_list)])+']\n'
102     else:
103         return rowid + '\t[]\n'
104
105 #ベクトルを疎行列形式に整えるメソッド
106 def sogyoretu(wordid, vec):
107     return str(wordid)+':'+str(vec)
108
109 #ベクトルを疎行列形式に整えるメソッド
110 def sogyoretu2(vec, cnt):
111     return str(cnt)+':'+str('{:f}'.format((vec+10)/20))
112     #return str(cnt)+':'+str(vec)
113
114 @exec_time.printer
115 def get_tokenizer():
116     #形態素解析にMeCabを使用する
117     return MeCab.Tagger('-r /etc/mecabrc -Owakati')
118
119 @exec_time.printer
120 def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
121     cnt = 1
122     l = fis.readline()
123     rownum_list = []
124     tokens_list = []
125     while (cnt != max_cnt and l != ''):
126         rownum, sentences = remove_tagwords(l).split('\t')
127         #if(sentences != ''):
128         rownum_list.append(rownum)
129         #トークナイズ (1文章を単語ごとに分割した一次元配列)
130         tokens = tokenizer.parse(sentences).split()
131         #ストップワードの除去
132         tokens = filter_stopwords(tokens)
133         #トークンリスト (複数の文章、単語の二次元配列)
134         tokens_list.append(tokens)
135         cnt += 1
136         l = fis.readline();
137     return (rownum_list, tokens_list)
138
139 re_tagwords = re.compile(r'([【手術】]|([【解剖】]|([【状況】]|([【母体】]|([【付言】]|
([【備考】])|\n'))

```

```

140 def remove_tagwords(sentences):
141     return re_tagwords.sub('', sentences)
142
143 stopword_list = None
144 def filter_stopwords(tokens):
145     global stopword_list
146     if stopword_list is None:
147         stopword_list = []
148         #ストップワードの読み込み
149         fis = open('./stopwords.txt','r')
150         re_comment = re.compile(r'^\s*(#.*|)$')
151         l = fis.readline()
152         while l:
153             if not re_comment.match(l):
154                 stopword_list.append(l.rstrip('\n'))
155             l = fis.readline()
156         fis.close()
157     return [word for word in tokens if word not in stopword_list]
158
159 @exec_time.printer
160 def tokenslist2dic(tokens_list):
161     return corpora.Dictionary(tokens_list)
162
163 @exec_time.printer
164 def fit_word2vec(tokens_list):
165     #word2vecによる単語の学習
166     return word2vec.Word2Vec(tokens_list)
167
168 if __name__ == '__main__':
169     main(1)
170
171

```

# 機械学習用データセット作成プログラム (Doc2Vec (PV-DM / PV-DBOW))

```
1 #!/usr/bin/env python3
2 # -*- coding: utf-8 -*-
3
4 #=====#
5 #== #==#
6 #== [embedding.py] #==#
7 #== version.1.1 #==#
8 #== 2022/03/28 #==#
9 #== #==#
10 #=====#
11
12 import pandas as pd
13 import numpy as np
14 import argparse
15 from argparse import RawTextHelpFormatter
16 import ast
17 import MeCab
18 from gensim.models.doc2vec import Doc2Vec, TaggedDocument
19 import tensorflow_hub as hub
20 import tensorflow_text
21 import re
22 import os
23
24 import ssl
25 ssl._create_default_https_context = ssl._create_unverified_context
26
27 # Embeddingクラス作成
28 class Embedding(object):
29     def __init__(self, rawdata_path, delete_symbol, params, mode, output_path):
30         self.rawdata_path = rawdata_path # 入力ファイルのパス
31         self.delete_symbol = delete_symbol # 形態素解析時に削除する記号
32         self.params = params # doc2vecを使用する際の引数(辞書
33 型)
34         self.mode = mode
35         self.output_path = output_path
36
37     def check_mode(self):
38         """
39         modeの確認。doc2vecを使用する際は、modeとparams['dm']が一致していないとエラーとな
40 る。
41         params['dm']を設定していない場合は、modeがparams['dm']の引数となる
42         """
43         if self.mode==2:
44             print('mode: Universal Sentence Encoder')
45
46         elif self.mode==0 or self.mode==1:
47             if 'dm' not in self.params:
48                 self.params['dm']=self.mode
49                 if self.mode==0:
50                     print('mode: doc2vec PV-DBOW')
51                 elif self.mode==1:
52                     print('mode: doc2vec PV-DM')
53
54             elif 'dm' in self.params:
55                 if self.params['dm'] == self.mode:
56                     if self.mode==0:
57                         print('mode: doc2vec PV-DBOW')
58                     elif self.mode==1:
59                         print('mode: doc2vec PV-DM')
60                 else:
61                     raise KeyError('modeとparams["dm"]が一致していません。 mode: {},
62 params["dm"]: {}'.format(self.mode, self.params['dm']))
63
64         else:
65             raise ValueError('--mode 0~2から選んでください {0: doc2vec PV-DBOW, 1:
66 doc2vec PV-DM, 3: Universal Sentence Encoder}')
67
68     def get_data(self):
69         """
70         rawdataの読み込み。入力ファイルは二列目にid、三列目にテキストデータを含む、headerおよ
71 びindex_colの無いtsvファイル。
72         """
```

```

69 # 入力ファイル形式が異なる場合は、read_csv()の引数設定を変更する
70 src = pd.read_csv('{}'.format(self.rawdata_path), delimiter='\t',
index_col=None, header=None, names=['id', 'data'])
71 # テキストデータがNaNのサンプル行を削除
72 src_data = src.dropna(how='any')
73
74 sentences = []
75 for text in src_data['data']:
76     # テキスト内の任意の記号を削除
77     text = re.sub(r'{}'.format(self.delete_symbol), '', text)
78
79     # 各行のテキストを1要素(リスト)としてsentencesに二次元リストとして格納
80     text_list = text.split(' ')
81     sentences.append(text_list)
82
83 return src, sentences
84
85 def doc2vec(self, sentences):
86     '''
87     doc2vecによるベクトル抽出。Mecab を用いて形態素解析した後、モデルを作成する。
88     '''
89     token = []
90     tokernizer = MeCab.Tagger('-0wakati') # Mecabの形態素解析の引数設定
91
92     # sentences内の各要素(テキスト)を形態素解析
93     for s in sentences:
94         token.append(tokernizer.parse(s[0]).split())
95
96     # 以下処理過程の参考
97     # 例 src_data.iloc[0,1] : 【付言】DNA検査にて本人確認。
98     # 例 sentences[0] : ['付言DNA検査にて本人確認']
99     # 例 token[0] : ['付言', 'DNA', '検査', 'にて', '本人', '確認']
100
101     # tokenをdoc2vecに読み込ませる形式に変換
102     documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(token)]
103
104     #hash関数の指定 モデルの再現性を得るためにハッシュ値を固定
105     import hashlib
106     hashfxn = lambda x: int(hashlib.md5(str(x).encode()).hexdigest(), 16)
107
108     # doc2vecモデル構築
109     model = Doc2Vec(documents, hashfxn=hashfxn, **self.params)
110     vectors = [model.docvecs[i] for i in range(len(sentences))]
111
112     return vectors
113
114 def UniversalSentenceEncoder(self, sentences):
115     '''
116     Universal Sentence Encoderによるベクトル抽出。形態素解析は行わずモデルを構築し、
512次元ベクトルを出力する。
117     '''
118     # tensorflow hubから学習済みモデルの読み込み
119     USE_model = hub.load(
120     "https://tfhub.dev/google/universal-sentence-encoder-multilingual/3") #
2022/01時点, latest version:はver.3
121     vectors_all = USE_model(sentences)
122     vectors = [i for i in vectors_all]
123
124     return vectors
125
126 def output(self, src, vectors):
127     '''
128     id とベクトルを結合し、出力用データフレームを作成。output_pathが"None"の場合はファイ
ル出力されない。
129     '''
130     src_data = src.dropna(how='any')
131     src_data_vec = src_data.assign( vecs = vectors)
132     df = pd.merge(src, src_data_vec, how="left", on = "id")
133     df = df.loc[:, ['id', 'vecs']]
134
135     # output_pathを任意指定した時のみファイルを出力する
136     if self.output_path=='None':
137         pass

```

```

138     else:
139         output_path = self.output_path
140         df.to_csv(output_path, sep='\t', header=False, index=False)
141
142     return df
143
144 def main():
145     parser = argparse.ArgumentParser(prog='embedding',
146                                     description='-----\n'
147                                     'PROGRAM NAME:下流工程を考慮したテキストの特徴抽出プログラム\n'
148                                     '概要: 自然言語(日本語)で記載されたテキストが持つ特徴を抽出するプ
149                                     'rogram.\n'
150                                     'doc2vecまたはuniversal setence encoderを用いてベクトル化す
151                                     'る.\n'
152                                     '-----\n',
153                                     usage='$ python3 embedding.py [-m] [-rp] [-ds] [-
154                                     params] [-op]\n',
155                                     formatter_class=RawTextHelpFormatter,
156                                     add_help=True)
157     parser.add_argument('--mode', '-m', type=int, required=True,
158                         help='モデルの選択\n'
159                         'mode:{0: doc2vec PV-DBOW, 1: doc2vec PV-DM, 2:
160                         Universal Sentence Encoder}')
161     parser.add_argument('--rawdata_path', '-rp', type=str, required=True,
162                         help='入力ファイルのパス.\n'
163                         '入力ファイルは一行目にid、二列目にテキストデータを含む、
164                         hedderおよびindex_colの無い.tsvファイル')
165     parser.add_argument('--delete_symbol', '-ds', type=str, default='[ [] \.。
166                         「」 () ]',
167                         help='形態素解析時の削除したい記号\n'
168                         '削除したい記号をカッコ内に入れ[ ]、str型で渡す\n'
169                         'e.g. [PROGRAM] --delete_symbol "[ [] \.。 「」 () ]"')
170     parser.add_argument('--doc2vec_params', '-params', type=str, default="
171     {'vector_size':2}",
172                         help='doc2vecに与える引数をstr型で入力する。引数はモジュー
173                         ル内で辞書型に変換される.\n'
174                         "'e.g. [PROGRAM] -params \"{'vector_size' : 2,
175                         'window': 2, 'min_count': 1, 'epochs': 10}\"")
176     parser.add_argument('--output_path', '-op', type=str, default='None',
177                         help='出力ファイルのパス.\n'
178                         '出力ファイルは一行目にid、二列目にベクトルデータを含む、
179                         hedderおよびindex_colの無い.tsvファイル\n'
180                         '引数を与えない、または"None"を与えるとファイルは出力されな
181                         い')
182     args = parser.parse_args()
183
184     # 引数受け取り
185     mode = args.mode
186     rawdata_path = args.rawdata_path
187     delete_symbol = args.delete_symbol
188     params = ast.literal_eval(args.doc2vec_params)
189     output_path = args.output_path
190
191     # インスタンス作成
192     E = Embedding(rawdata_path, delete_symbol, params, mode, output_path)
193     # modeとparamsの確認
194     E.check_mode()
195
196     # 入力データ(rawdata)読み込み
197     src, sentences = E.get_data()
198
199     # doc2vecによる実行
200     if mode==0 or mode==1:
201         vectors = E.doc2vec(sentences)
202     # USEによる実行
203     elif mode==2:
204         vectors = E.UniversalSentenceEncoder(sentences)
205
206     # 出力用データフレーム作成
207     df = E.output(src, vectors)

```

```
198 | if __name__ == '__main__':  
199 |     main()
```

# 分類器学習プログラム (fit\_and\_predict\_xgboost.py)

```
1 import pandas as pd
2 import xgboost as xgb
3 import numpy as np
4 import pickle
5 from sklearn.model_selection import train_test_split
6 from sklearn.metrics import accuracy_score
7 from matplotlib import pyplot as plt
8 from sklearn.metrics import confusion_matrix, roc_curve,
precision_recall_curve, auc
9 from sklearn.model_selection import GridSearchCV
10 from scipy.sparse import lil_matrix
11 from datetime import datetime
12 import sys
13 import os
14
15 def arg_parse():
16     import argparse
17     p = argparse.ArgumentParser()
18     p.add_argument('-a', '--anyfolder', help='疎行列ベクトル参照先フォルダ')
19     p.add_argument('-g', '--gpuid', help='使用するGPUのID (デフォルトは0)')
20     p.add_argument('-y', '--year', help='学習用データ年')
21     a = p.parse_args()
22     return (a.anyfolder, a.gpuid, a.year)
23
24 def main(any_folder='.', gpuid=0, year=2020):
25     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] ' +
__file__ + ' start')
26
27     # 引数の検証
28     if not any_folder: any_folder = '.'
29     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 入力元フォルダ[../07_付帯情報Embedding/' + any_folder + ']')
30     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 出力先フォルダ[./ + any_folder + '/RESULT/' + year + '/]')
31     #os.makedirs('./RESULT/' + any_folder, exist_ok=True)
32     # 疎行列行列数ファイル読み込み
33     # rows = 81688 # 行数
34     # cols = 1579 + 2736 # 次元数
35     fis = open('../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_rowcolnum_' + year + '.txt', 'r')
36     rows = int(fis.readline())
37     cols = int(fis.readline())
38     fis.close()
39     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 対象行数
[' + str(rows) + ' ] 対象ベクトル数[' + str(cols) + ' ]')
40
41     # 疎行列ファイルを読み込み
42     smatrix_fname = '../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_' + year + '.txt'
43     X, y = load_sparse_matrix(smatrix_fname, rows, cols)
44     # test_size=0.2 全体のうち0.8を学習用、0.2をテスト用に分割する
45     X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=True, random_state=42, stratify=y)
46
47     # 学習用の0.8で学習
48     # NOTE: tree_method、gpu_idの設定をすることでGPUが使用されるようになる。
49     # ----- Best Estimator
50     clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
51         colsample_bynode=1, colsample_bytree=1, eta=0.25, gamma=0,
52         gpu_id=gpuid, importance_type='gain',
interaction_constraints='',
53         learning_rate=0.25, max_delta_step=0, max_depth=14,
54         min_child_weight=0.05, monotone_constraints=('',),
55         n_estimators=2000, n_jobs=1, num_parallel_tree=1,
random_state=0,
56         reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
57         tree_method='gpu_hist', use_label_encoder=False,
58         validate_parameters=1, verbosity=None, eval_metric='logloss')
59     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] fit
start')
60     clf.fit(X_train, y_train)
61     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] fit
```

```

end')
62
63 # モデルを保存
64 pickle.dump(clf, open('./' + any_folder + '/' + year + '/model.pkl',
65 'wb'))
66 clf = pickle.load(open('./' + any_folder + '/' + year + '/model.pkl',
67 'rb'))
68
69 # テスト用の0.2で検証
70 print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
start')
71 # pred = clf.predict(X_test)
72 pred_p = clf.predict_proba(X_test)
73 print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
end')
74
75 # 結果出力
76 report = []
77 for i in range(21):
78     t = (i/20)
79     pred = (pred_p[:, 1] > t).astype(int)
80     print('-----==== 閾値[' + str(t) + '] =====')
81     accuracy = print_accuracy(y_test, pred)
82     tp, fp, fn, tn, precision, recall = print_confusion_matrix(y_test,
pred)
83     report.append([t, accuracy, tn, fp, fn, tp, precision, recall, tp+fp,
(tp+fp)/(tp+fp+fn+tn)])
84     write_id_tf(X[:, 0], y_test, pred, t, any_folder, year)
85
86 if any_folder == 'TFIDF_100':
87     print_importances(clf, any_folder, year)
88
89 y_score = pred_p[:,1]
90 make_pr_curve_pict(y_test, y_score, any_folder, year)
91 make_roc_curve_pict(y_test, y_score, any_folder, year)
92
93 write_report(report, any_folder, year)
94 write_pred_score(X[:, 0], y_test, pred_p[:, 1], any_folder, year)
95
96 print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] ' +
__file__ + ' end')
97
98 def load_sparse_matrix(fname, rows, cols):
99     # 疎行列ファイルをオープン
100     fis = open(fname, 'r')
101
102     X_lil = lil_matrix((rows, cols - 1), dtype=float)
103     # y_lil = lil_matrix((rows, 1), dtype=float)
104     y = [0] * rows
105
106     # 疎行列ファイルの読み込み
107     l = fis.readline()
108     rownum = 0
109     while l != '':
110         l_smatrix = sogyoretu2smatrix(l.rstrip('\n'))
111         for k, v in l_smatrix.items():
112             if k != cols - 1:
113                 if len(v) != 0: # DOC2VEC 疎行列データエラー対策
114                     X_lil[rownum, k] = float(v)
115                 else:
116                     print("[ERROR]: ' + str(rownum) +
117
118                     X_lil[rownum, k] = 0
119             else:
120                 ## y_lil[rownum, 0] = float(v)
121                 y[rownum] = float(v)
122
123         l = fis.readline()
124         rownum += 1
125
126     fis.close()
127
128 # sparseからnumpy.ndarrayに変換

```

```

126 X = X_lil.toarray()
127 # y = y_lil.toarray()
128 return X, y
129
130 def sogyoretu2smatrix(sogyoretu):
131     """
132     01_sogyoretu.csvの疎行列表記文字列を疎行列のdictionaryに変換する。
133     """
134     # 先頭末尾の[]を外す
135     # ,でスプリットしkvセットに切り分る
136     # :でスプリットしてkeyとvalueを分ける
137     return {int(kv.split(':')[0]): kv.split(':')[1] for kv in
sogyoretu[1:-1].split(',') if kv != ''}
138
139 def print_accuracy(act, pred):
140     """
141     精度を出力する。
142     """
143     acc = accuracy_score(act, pred)
144     print('----- Accuracy')
145     print(acc)
146     return acc
147
148 def print_confusion_matrix(act, pred):
149     """
150     混合配列を出力する。
151     """
152     cm = {}
153     cm[(0,0)] = 0
154     cm[(0,1)] = 0
155     cm[(1,0)] = 0
156     cm[(1,1)] = 0
157     for t, p in zip(act, pred):
158         cm[(t, p)] += 1
159     print('----- Confusion Matrix')
160     print('(act, pred): ', cm)
161     print('----- Confusion Matrix')
162     print(pd.DataFrame(confusion_matrix(act, pred, labels=[0, 1]), columns=
["pred_0", "pred_1"], index=["act_0", "act_1"]))
163
164     p, r = print_pr(cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)])
165     return (cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)], p, r)
166
167 def print_pr(tp, fp, fn, tn):
168     p = tp/(tp+fp) if (tp+fp)!=0 else 1
169     print('----- Precision')
170     print(str(p))
171     r = tp/(tp+fn) if (tp+fn)!=0 else 1
172     print('----- Recall')
173     print(str(r))
174     return (p, r)
175
176 def print_importances(clf, any_folder, year):
177     """
178     重要度を出力する。
179     """
180     import os
181     from gensim import corpora
182
183     dict_fname = '../07_付帯情報Embedding/' + any_folder +
'/new_shibo_join.dict'
184     tokens_dict = None
185     if os.path.exists(dict_fname) and os.path.getsize(dict_fname):
186         tokens_dict = corpora.Dictionary.load(dict_fname)
187
188     icdcode_fname = '../06_機械学習用基本データ/ICDList/acme_kari_code_sort_' +
year + '.txt'
189     icdcodes = ['certificateKey', '年齢', '性別']
190     fis = open(icdcode_fname, 'r')
191     lines = 0
192     l = fis.readline()
193     while(l != ''):
194         icdcodes.append(l.rstrip('\n'))

```

```

195     l = fis.readline()
196     lines += 1
197
198     fis.close()
199
200     icdcodes.append('手術フラグ(1)')
201     icdcodes.append('手術の部位及び所見(116)')
202     icdcodes.append('(手術)備考欄への記載(1)')
203     icdcodes.append('手術日(8)')
204     icdcodes.append('解剖フラグ(1)')
205     icdcodes.append('解剖の部位及び所見(116)')
206     icdcodes.append('(解剖)備考欄への記載(1)')
207     icdcodes.append('死因の種類(2)')
208     icdcodes.append('傷害が発生したとき(8)')
209     icdcodes.append('傷害が発生したとき(5)')
210     icdcodes.append('傷害が発生したところの種類別(1)')
211     icdcodes.append('傷害が発生したところその他の記述(40)')
212     icdcodes.append('傷害発生場所(8)')
213     icdcodes.append('傷害発生場所(12)')
214     icdcodes.append('傷害発生場所(18)')
215     icdcodes.append('手段及び状況(120)')
216     icdcodes.append('(傷害)備考欄への記載(1)')
217     icdcodes.append('「生後1年未満での病死」の病態・異状の詳細(84)')
218     icdcodes.append('備考欄への記載(1)')
219     icdcodes.append('その他付言すべき事柄(60)')
220     icdcodes.append('備考欄外字有無(1)')
221     icdcodes.append('備考欄(1024)')
222
223     lines += 25
224    imps = sorted(enumerate(clf.feature_importances_), key=lambda kv: -kv[1])
225     print('----- Feature Importances')
226     for k, v inimps:
227         print(str(k) + ', ' + (icdcodes[k] if k<lines else tokens_dict[k-
lines] if tokens_dict else '-') + ', ' + str(v))
228
229 def make_pr_curve_pict(y_test, y_score, any_folder, year):
230     # PR取得
231     precision, recall, thresholds = precision_recall_curve(y_true=y_test,
probas_pred=y_score)
232     # 結果をプロット
233     plt.figure(1, figsize=[12.8, 9.6], dpi=100)
234     plt.figure(1)
235     plt.plot(recall, precision, label=(any_folder if any_folder != '.' else
'precision-recall curve') + ' (AUC = %0.3f)' % auc(recall, precision))
236     for i in range(21):
237         close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
238         plt.plot(recall[close_point], precision[close_point], 'o')
239     # ラベルなどを追加しファイル出力
240     plt.plot([0,1], [1,1], linestyle='--', label='ideal line')
241     plt.legend()
242     plt.xlabel('recall')
243     plt.ylabel('precision')
244     plt.set_title('P-R Curve')
245     fname = './' + any_folder + '/' + year + '/PR-Curve.png'
246     plt.savefig(fname)
247     print([' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')
248
249 def make_roc_curve_pict(y_test, y_score, any_folder, year):
250     # ROC取得
251     fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_score)
252     # 結果をプロット
253     plt.figure(2, figsize=[12.8, 9.6], dpi=100)
254     plt.figure(2)
255     plt.plot(fpr, tpr, label=(any_folder if any_folder != '.' else 'roc
curve') + ' (AUC = %0.3f)' % auc(fpr, tpr))
256     for i in range(21):
257         close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
258         plt.plot(fpr[close_point], tpr[close_point], 'o')
259     # ラベルなどを追加しファイル出力
260     plt.plot([0,0,1], [0,1,1], linestyle='--', label='ideal line')
261     plt.plot([0, 1], [0, 1], linestyle='--', label='random prediction')
262     plt.legend()

```

```

263 plt.xlabel('false positive rate(FPR)')
264 plt.ylabel('true positive rate(TPR)')
265 plt.set_title('ROC Curve')
266 fname = './' + any_folder + '/' + year + '/ROC-Curve.png'
267 plt.savefig(fname)
268 print('[ ' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 結果を出力
しました。[ ' + fname + ' ]')
269
270 def write_report(report, any_folder, year):
271     fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_report.csv'
272     fos = open(fname, 'w')
273     for l in list(zip(*report)):
274         fos.write(','.join([str(f) for f in l]) + '\n')
275     fos.close()
276     print('[ ' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 結果を出力
しました。[ ' + fname + ' ]')
277
278 def write_id_tf(ids, act, pred, ikichi, any_folder, year):
279     fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_id_tf_ikichi_' + str(ikichi) + '.csv'
280     fos = open(fname, 'w')
281     for i, a, p in list(zip(ids, act, pred)):
282         fos.write( '{:.0f},{:.0f},{:.0f},{:.0f}'.format(i, a, p, a==p) + '\n')
283     fos.close()
284     print('[ ' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 結果を出力
しました。[ ' + fname + ' ]')
285
286 def write_pred_score(ids, act, pred_score, any_folder, year):
287     fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_pred_score.csv'
288     fos = open(fname, 'w')
289     for i, a, p in list(zip(ids, act, pred_score)):
290         fos.write( '{:.0f},{:.0f},{:.5f}'.format(i, a, p) + '\n')
291     fos.close()
292     print('[ ' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + ' ] 結果を出力
しました。[ ' + fname + ' ]')
293
294 if __name__ == '__main__':
295     any_folder, gpuid, year = arg_parse()
296     main(any_folder, gpuid, year)
297
298

```