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)

```
from my_util import exec_time

from datetime import datetime
import re
import regexp14
import regexp16
import regexp18
import regexpOther
import copy

@exec_time.printer
def main():
  regs14 = regexp14.get()
  regs16 = regexp16.get()
  regs18 = regexp18.get()
  regsOther = regexpOther.get()
  max_cnt = 0
  fis = open('./01_new_shibo_join_kv.csv', 'r')
  fos = open('./02_new_shibo_join_kv2' + ( '' if max_cnt <= 0 else '_' + str(max_cnt) ) + '.csv', 'w')
  f_removed = open('./02_new_shibo_join_removed_words' + ( '' if max_cnt <= 0 else '_' + str(max_cnt) ) + '.tsv', 'w')

  cnt = 0
  for sbh in sbh_data_generator(fis):
    cnt += 1
    if cnt == max_cnt: break
    rowid = sbh['rowid']
    values = sbh['value']
    # 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　II死因　14,10: 死亡原因　II期間　14,12: 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
    # for sbhnum, sbhnum2 in [('14', '12'), ('14', '16'), ('16', '4'), ('16', '8'), ('17', '1'), ('18', '1'), ('99', '1')]:
    # 18,1: その他付言　99,1: 備考
    for sbhnum, sbhnum2 in [('18', '1'), ('99', '1')]:
      col = copy.deepcopy(values[sbhnum][sbhnum2])
      for k in col.keys():
        if k=='colnum': continue
        value = col[k]
        if len(value):
          org_value = value
          # 14用の正規表現でパース
          for r in regs14:
            # 14の区切り位置を[col14]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 16用の正規表現でパース
          for r in regs16:
            # 16の区切り位置を[col16]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 18用の正規表現でパース
          for r in regs18:
            # 18の区切り位置を[col18]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # 上記以外用の正規表現でパース
          for r in regsOther:
            # 区切り位置を[colxx_0]で置換する
            t = fw_reg(r[0]).subn(r[1], value)
            if t[1]: value = t[0]
          # スプリッタ（[colxx]）で切り分け
          value_arr = fw_reg(r'(\[col[^\]]+\])([^(\[$)]*)').findall(value)
          if value_arr:
            len14_1 = len(values['14']['1']) - 1
            len14_2 = len(values['14']['2']) - 1
            len14_3 = len(values['14']['3']) - 1
            len14_4 = len(values['14']['4']) - 1
            len14_5 = len(values['14']['5']) - 1
            len14_6 = len(values['14']['6']) - 1
```

```
            len14_7 = len(values['14']['7']) - 1
            len14_8 = len(values['14']['8']) - 1
            len14_9 = len(values['14']['9']) - 1
            len14_10 = len(values['14']['10']) - 1
            len14_12 = len(values['14']['12']) - 1
            len14_14 = len(values['14']['14']) - 1
            len14_16 = len(values['14']['16']) - 1
            len16_4 = len(values['16']['4']) - 1
            len16_8 = len(values['16']['8']) - 1
            len18_1 = len(values['18']['1']) - 1
            # 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
            # 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。
            splited_value_regexp = '(' + fw_reg(r'\[col[^\]]+\]').sub(r')
(.*?', value) + ')'
            splited_value_arr = re.findall(splited_value_regexp, org_value)
            for i, (spliter, v) in enumerate(value_arr):
              # 連結前の整形
              tmp = fw_reg(r'^(は)?「(.*)」(である)?(。)?$').search(v)
              if tmp: v = tmp.groups()[1]
              v = fw_reg(r'( (続く) )?\n?$').sub('', v)
              if spliter == '[col18_top]':
                sbh['value']['18']['1'][len18_1] = v
                len18_1 += 1
              elif spliter == '[col18]':
                sbh['value']['18']['1'][len18_1 + 100] = v
                len18_1 += 1
              #elif spliter == '[col16_4]':
              # sbh['value']['16']['4'][len16_4] = v
              # len16_4 += 1
              elif spliter == '[col16]' or spliter == '[col16_8]':
                sbh['value']['16']['8'][len16_8] = v
                len16_8 += 1
              elif spliter == '[col14_1]':
                sbh['value']['14']['1'][len14_1] = v
                len14_1 += 1
              elif spliter == '[col14_2]':
                sbh['value']['14']['2'][len14_2] = v
                len14_2 += 1
              elif spliter == '[col14_3]':
                sbh['value']['14']['3'][len14_3] = v
                len14_3 += 1
              elif spliter == '[col14_4]':
                sbh['value']['14']['4'][len14_4] = v
                len14_4 += 1
              elif spliter == '[col14_5]':
                sbh['value']['14']['5'][len14_5] = v
                len14_5 += 1
              elif spliter == '[col14_6]':
                sbh['value']['14']['6'][len14_6] = v
                len14_6 += 1
              elif spliter == '[col14_7]':
                sbh['value']['14']['7'][len14_7] = v
                len14_7 += 1
              elif spliter == '[col14_8]':
                sbh['value']['14']['8'][len14_8] = v
                len14_8 += 1
              elif spliter == '[col14_9]':
                sbh['value']['14']['9'][len14_9] = v
                len14_9 += 1
              elif spliter == '[col14_10]':
                sbh['value']['14']['10'][len14_10] = v
                len14_10 += 1
              elif spliter == '[col14_12]':
                sbh['value']['14']['12'][len14_12] = v
                len14_12 += 1
              elif spliter == '[col14_14]':
                sbh['value']['14']['14'][len14_14] = v
                len14_14 += 1
              elif spliter == '[col14_16]':
                sbh['value']['14']['16'][len14_16] = v
                len14_16 += 1
              else: # ここまでに合致しないスプリッタの文言は捨てる
                f_removed.write(rowid + '\t' + spliter + '\t' +
```

```
splited_value_arr[0][i+1].rstrip('\n') + '\n')
            # 切り分けた先頭は元の欄に残す
            try:
              sbh['value'][sbhnum][sbhnum2][k] = fw_reg(r'^([^\
[]+)').search(value).group() + '\n'
            except AttributeError:
              sbh['value'][sbhnum][sbhnum2][k] = '\n'
    sbh_data_writer(fos, sbh)

  fis.close()
  fos.close()
  f_removed.close()

def sbh_data_generator(fis):
  l = fis.readline()
  bef_rowid = l.split(',')[0]
  sbh = {'rowid': bef_rowid, 'value': {}}
  while l!='':
    rowid, colnum, sbhnum, sbhnum2, colrownum, value = l.split(',')
    if bef_rowid != rowid:
      yield sbh
      sbh = {'rowid': rowid, 'value': {}}
    # キーを変更してdictにする
    # sbh.update({'rowid': rowid, 'value': {'sbhnum': sbhnum, 'value':
{'sbhnum2': sbhnum2, 'sbhnum': sbhnum, 'value': {sbhrownum: value }}}})
    # sbh.update({'rowid': rowid, 'value': {sbhnum: {sbhnum2: {'colnum':
colnum, colrownum: value }}}})
    if not sbhnum in sbh['value']: sbh['value'][sbhnum] = {}
    if not sbhnum2 in sbh['value'][sbhnum]: sbh['value'][sbhnum][sbhnum2] =
{'colnum': colnum}
    sbh['value'][sbhnum][sbhnum2][colrownum] = value
    bef_rowid = rowid

    l = fis.readline()
  yield sbh

def sbh_data_writer(fos, sbh):
  # rowid     : new_shibo_join.tsvの先頭カラム、行ごとのID
  # colnum    : new_shibo_join.tsvのカラム番号
  # sbhnum    : 死亡診断書の欄番号。new_shibo_join.tsvには存在しない。
  # 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。
  # colrownuma: 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。
  # value     : new_shibo_join.tsvの2カラム目以降の値。
  rowid = sbh['rowid']
  out_arr = []
  for sbhnum, sbhnum_col in [(str(sbhnum), sbh['value'][str(sbhnum)]) for
sbhnum in sbh['value'].keys()]:
    if sbhnum == 'rowid': continue
    for sbhnum2, sbhnum2_col in [(sbhnum2, sbhnum_col[sbhnum2]) for sbhnum2
in sbhnum_col.keys()]:
      colnum = sbhnum2_col['colnum']
      for colrownum, value in [(colrownum, sbhnum2_col[colrownum]) for
colrownum in sbhnum2_col.keys()]:
        if colrownum == 'colnum': continue
        out_arr.append([rowid, colnum, sbhnum, sbhnum2, str(colrownum),
value])
  out_arr.sort(key=lambda item: (int(item[1]), int(item[4])))

  bef_colnum = out_arr[0][1]
  l = ','.join(out_arr[0])
  for l_arr in out_arr[1:]:
    if bef_colnum != l_arr[1]:
      fos.write(fw_reg(r'( (続く) )?\n?$').sub('', l) + '\n')
      l = ','.join(l_arr)
      bef_colnum = l_arr[1]
    else:
      l = l.rstrip('\n') + l_arr[5]
  fos.write(l)

fw_reg_cache = {}
def fw_reg(p):
  """
  初めて取得するパターンはコンパイルして返す。
```

```
  初めてでないパターンはキャッシュから返す。
  """
  try:
    return fw_reg_cache[p]
  except KeyError:
    r = re.compile(p)
    fw_reg_cache[p] = r
    return r


if __name__ == '__main__':
  main()
```

備考欄前処理プログラム　(regexp14.py)

```
def get():
  return [[r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ア)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_1]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ア)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_2]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(イ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_3]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(イ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_4]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ウ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_5]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(ウ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_6]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(エ)+([」）])?(の|　|）)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_7]'],
    [r'（?ｌ４）?([ーーー「(])?(ｌ|1|I|i)?([「(])?(エ)+([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14_8]'],
    [r'（?ｌ４）?([ーーー「(])?(２|2|Ⅱ|ⅱ)+([」）])?(の|　)?(死因)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?(傷病名)?',r'[col14_9]'],
    [r'（?ｌ４）?([ーーー「(])?(２|2|Ⅱ|ⅱ)+()?([」）])?(死亡までの|の|　)?(期間|年月日|年月|年|月日|月|日)+(欄|ラン|らん)?(続き|つづき)?(は)?([　、。・:，ーー）])?(傷病名)?',r'[col14_10]'],
    [r'（?ｌ４）?(欄中?)?([ーーー，、])?(の|　)?(解剖)+(欄|ラン|らん|　)?(主要所見|所見)?(の|　)?(続き|つづき)?(は)?([　、。・:，ーー）])?', r'[col14_16]'],
    [r'（?ｌ４）?(欄中?)?([ーーー，、])?(の|　)?(手術|手術部位及び主要所見追加)+(?!解剖|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)?([　、。・:，ーー）])?', r'[col14_12]'],
    [r'（?ｌ４）?(欄中?)?([ーーー「(，、])?(の|　)?(手術期間|手術年月日|手術年月|手術年|手術月日|手術月|手術日|手術施行日)+(欄)?([」）])?(の|　)?(続き|つづき)?(は)?([　、。・:，ーー）])?', r'[col14_14]'],
    [r'(（ｌ４）)+(欄中?)?(の|　)?([ーーー(，、])?(期間の記載|死亡したとき)?(続き|つづき)?(は)?([　、。・:，ーー）])?',r'[col14]'],
    [r'（ｌ４）(ｌ|1|I|i|２|2|Ⅱ|ⅱ)?([ーーー，、])?(「(ア)直接原因」|「(イ)（ア）の原因」|「(ウ)（イ）の原因」|「(エ)（ウ）の原因」|「I欄に影響を及ぼした傷病名等」)?(の|　)?(「死亡までの期間」)+(　)?',r'[col14]'],
    [r'（?ｌ４）?(手術・解剖|手術解剖)+(　)?',r'[col14]'],
    [r'（?ｌ４ｌ(ｌ|２|３|４|５)?）?(欄|ラン|らん)?(続き|つづき)?(続)?(は)?([　、。・:，ーー）])?',r'[col14]']]#分類できないものはこのキーに置く
    #記号控え([　、。・:，ーー）])

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s) for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')


  print('==========---------- 終了')

```

備考欄前処理プログラム　(regexp16.py)

```
def get():
  return [[r'（?１６）?(欄中?)?(　)?）?([の（])?(続き|続|つづき|追記)([　、・:，ーー
ー）])?', r'[col16]'],#１６の情報として抽出
    [r'（?１６）?(欄中?)?([ーー一の「])?(傷害|障害|死亡)?(発生|が発生した|した)+(日時|時
刻|時分|とき|時間)+(欄)?([の（）」])?(続き|続|つづき|追記|時分)?(は)?([　、・:，ーー
ー）])?', r'[col16_4]'],#傷害が発生した時刻として抽出
    [r'（?１６）?(欄中?)?([ーー一の「])?(傷害|障害|死亡)?(発生|が発生した|した)+(日|時)+
(欄)?([の（）」])?(続き|続|つづき|追記|)?(は)?([　、・:，ーー一）])?',
r'[col16_4]'],#傷害が発生した時刻として抽出
    [r'（１６）(欄中?)?(「)?(手段及び状況|状況|手段および状況)?(欄)?([の（」　])?(続き|
続|つづき|追記)?(において|は)?([　、・:，ーー一）])?', r'[col16_8]'],#手段及び状況とし
て抽出と結合
    [r'（?１６０５）?([の（])?(続き|続|つづき|追記)?([は）])?([　、・:，ーー一])?',
r'[col16_8]'],#手段及び状況として抽出と結合
    [r'「?１６枠外」?([　、・:，ーー一])?', r'[col16]'],#１６の情報として抽出
    [r'（１６追加）', r'[col16]'], #１６の情報として抽出
    [r'１６[　）：欄]', r'[col16]'],#１６の情報として抽出
    [r'１６(欄)?(ーー一)',r'[col16]'],#１６の情報として抽出
    [r'(外因死)?(の)?(追加事項の)?：?(手段及び状況)+(欄)?(について)?(の)?(続き|続|つづ
き|追記)?(は)?([　、。・:，ーー一）])?',r'[col16_8]']]#手段及び状況として抽出と結合

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')


  print('==========---------- 終了')
```

備考欄前処理プログラム　(regexp18.py)

```
def get():
  return [[r'１８続き、', r'[col18_top]'],
    [r'（?１８）?(欄)?(　)?）?([の（])?(続き|続|つづき|追記|：)([　、・:，ーー—）])?', r'[col18]'],
    [r'（?１８）?(欄)?([ーー—])?(その他|その他特に付言すべき事柄)(欄)?([の（）])?(続き|続|つづき|追記)?([　、・:，ーー—）])?', r'[col18]'],
    [r'（１８(欄)?）(欄)?([の（])?(続き|続|つづき|追記)?([　、・:，ーー—）])?', r'[col18]'],
    [r'（?１８０１）?([の（])?(続き|続|つづき|追記)?([　、・:，ーー—])?', r'[col18]'],
    [r'「?１８枠外」?([　、・:，ーー—])?', r'[col18]'],
    [r'（１８欄その他特に付言すべきことがら）', r'[col18]'],
    [r'１８その他特に付言すべきことがらの続きは、', r'[col18]'],
    [r'（１８追加）', r'[col18]'],
    [r'１８[　）；欄]', r'[col18]'], ]

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s) for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')

  print('==========---------- 終了')
```

備考欄前処理プログラム (regexpOther.py)

```
def get():
  return [# （3）生年月日
    [r'（3）(欄|続き)?', r'[col03_0]'],
    [r'(?<!1)3(続き|続|つづき)[ 、・:，ーー]?', r'[col03_0]'],
    [r'（3(続き|続|つづき)）', r'[col03_0]'],
    # （4）死亡したとき
    [r'（4）死亡したとき[ 、・:，ーー]?', r'[col04_0]'],
    [r'（4）(欄|続き)?', r'[col04_0]'],
    [r'(?<!1)4(続き|続|つづき)[ 、・:，ーー]?', r'[col04_0]'],
    [r'（4(続き|続|つづき)）', r'[col04_0]'],
    # （5）死亡したところ
    [r'（5）(欄|続き)?', r'[col05_0]'],
    [r'(?<!1)5(続き|続|つづき)[ 、・:，ーー]?', r'[col05_0]'],
    [r'（5(続き|続|つづき)）', r'[col05_0]'],
    # （6）住所
    [r'（6）(欄|続き)?', r'[col06_0]'],
    [r'(?<!1)6(続き|続|つづき)[ 、・:，ーー]?', r'[col06_0]'],
    [r'（6(続き|続|つづき)）', r'[col06_0]'],
    # （7）本籍
    [r'（7）(欄|続き)?', r'[col07_0]'],
    [r'(?<!1)7(続き|続|つづき)[ 、・:，ーー]?', r'[col07_0]'],
    [r'（7(続き|続|つづき)）', r'[col07_0]'],
    # （8）（9）死亡した人の夫または妻
    [r'（8）(欄|続き)?', r'[col08_0]'],
    [r'(?<!1)8(続き|続|つづき)[ 、・:，ーー]?', r'[col08_0]'],
    [r'（8(続き|続|つづき)）', r'[col08_0]'],
    [r'（9）(欄|続き)?', r'[col09_0]'],
    [r'(?<!1)9(続き|続|つづき)[ 、・:，ーー]?', r'[col09_0]'],
    [r'（9(続き|続|つづき)）', r'[col09_0]'],
    # （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
    [r'（10）(欄|続き)?', r'[col10_0]'],
    [r'10(続き|続|つづき)[ 、・:，ーー]?', r'[col10_0]'],
    [r'（10(続き|続|つづき)）', r'[col10_0]'],
    [r'（11）(欄|続き)?', r'[col11_0]'],
    [r'11(続き|続|つづき)[ 、・:，ーー]?', r'[col11_0]'],
    [r'（11(続き|続|つづき)）', r'[col11_0]'],
    # （12）（13）死亡したところ、及びその種別
    [r'（?12）?[ーー]施設名称?(続き|続|つづき)[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）死亡したところ[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）死亡したとき[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12）(欄|続き)?', r'[col12_0]'],
    [r'12(続き|続|つづき)[ 、・:，ーー]?', r'[col12_0]'],
    [r'（12(続き|続|つづき)）', r'[col12_0]'],
    [r'（13）死亡したところ[ 、・:，ーー]?', r'[col12_0]'],
    [r'（13）死亡したとき[ 、・:，ーー]?', r'[col12_0]'],
    [r'（13）(欄|続き)?', r'[col13_0]'],
    [r'13(続き|続|つづき)[ 、・:，ーー]?', r'[col13_0]'],
    [r'（13(続き|続|つづき)）', r'[col13_0]'],
    #（17）生後1年未満で病死した場合の追加事項
    [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]'],
    [r'（17）(欄|続き)?', r'[col17_0]'],
    [r'17(続き|続|つづき)[ 、・:，ーー]?', r'[col17_0]'],
    [r'（17(続き|続|つづき)）', r'[col17_0]'],
  ]

def _assert(r, s):
  import re
  # [print(re.subn(rr[0], rr[1], s)[1]) for rr in r]
  print('開始[' + s + ']')
  result = ''
  for rr in r:
    t = re.subn(rr[0], rr[1], s)
    if t[1]:
      if len(result.replace('0', '')): print('経過[' + s + ']')
      s = t[0]
      result += '1'
    else:
      result += '0'
  # l = list(filter(lambda result: result[1] > 0, [re.subn(rr[0], rr[1], s)
for rr in r]))
  print('結果[' + s + ']\n')
  # print([result[0] + '\n' for result in l])
```

```
  return len(result.replace('0', ''))


if __name__ == '__main__':
  r = get();

  print('開始 ----------==========')

  print('==========---------- 終了')
```

機械学習用データセット作成プログラム (TF・IDF)

```
from datetime import datetime
from gensim import corpora
from gensim import models
from gensim import matutils
import MeCab
import math
import re
import sys
import os

def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
    return add + math.log(1.0 * totaldocs / docfreq, log_base)

def arg_below():
    import argparse
    p = argparse.ArgumentParser()
    p.add_argument('-a', '--anybelow', help='単語出現回数下限の指定、未入力の場合
Noneで引数を取り、100で処理')
    a = p.parse_args()
    return a.anybelow

def main(any_below = 100, mode = 0):
    max_cnt = 0
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']開始しま
す。')
    #単語の出現回数の下限、Noneの場合は100
    below = int(100 if any_below == None else any_below)
    #print(below)

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = ''
    if any_below != None:
        dirname = 'TFIDF'+ ( '' if below <= 0 else '_' + str(below) ) + '/'
        #os.makedirs(dirname,mode=511,exist_ok=True)

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./' + dirname + '05' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '_sogyouretu.csv','w')
    fos2 = open('./' + dirname + '05' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '_total_wordnum.txt','w')

    #作成済み辞書の読み込み先リンク
    gdic_fname = './' + dirname + 'new_shibo_join.dict'

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き開
始')
    #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
    rownum_list,tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き終
了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成開始')
    #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
    if mode == 0:
        #トークンリストを辞書変換
        tokens_gdic = tokenslist2dic(tokens_list)
        #辞書情報を保存
        #save_gdic(tokens_gdic, gdic_fname)
    elif mode == 1:
        #辞書情報の読み込み
        tokens_gdic = load_gdic(gdic_fname)

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
開始')
    #辞書をbag-of-words形式のリスト（corpus）に変換
```

```
    tokens_corpus = doc2bow(tokens_gdic, tokens_list)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成開始')
    test_model = models.TfidfModel(tokens_corpus,wglobal=new_idf)

    corpus_tfidf = test_model[tokens_corpus]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成終了')

    #0回出現の単語についても重要度0を出力する
    #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for c in id_list]
    #辞書の総単語数を取得
    total_wordnum = 0
    for word,wordid in tokens_gdic.token2id.items():
        total_wordnum += 1
    fos2.write(str(total_wordnum))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力開始')
    for rowid,doc in zip(rownum_list,corpus_tfidf):
        lil =[]
        for word in doc:
            lil.append((word[0],word[1]))

        fos.write(data_make(rowid,lil))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力終了')

    fis.close()
    fos.close()
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']終了します。')

#出力の形式を整えるメソッド
def data_make(rowid,lil):
    return rowid + '\t['+','.join([sogyouretu(kv) for kv in lil])+']\n'

#tfidf値を疎行列の形にして返すメソッド
def sogyouretu(kv):
    return str(kv[0])+':'+str(kv[1])

    #for rowdic,rowid in zip(new_dic,id_list):
        #print(rowid+'\t',','.join([str(k)+','+str(v) for k,v in sorted(rowdic.items())]))
        #print(rowid+'\t',','.join([str(v) for k,v in rowdic.items()]))
        #fos.write(rowid+'\t'+','.join([str(v) for k,v in sorted(rowdic.items())])+'\n')

    """
    #出現0回の文字について重要度0を入れない出力
    texts_tfidf = []
    for rowid,doc in zip(id_list,corpus_tfidf):
        text_tfidf = []
        for word in doc:
            text_tfidf.append(word[1])
        texts_tfidf.append(text_tfidf)
        print(rowid+'\t',','.join([str(i) for i in text_tfidf]))
    """

#作成済みdictionaryを読み込むメソッド
def load_gdic(fname):
    return corpora.Dictionary.load(fname)

#作成したdictionaryを保存するメソッド
def save_gdic(tokens_gdic, fname):
    tokens_gdic.save(fname)

#形態素解析にMeCabを使用し、トーカナイザーを取得するメソッド
def get_tokenizer():
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')
```

```
129
130 #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
131 def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
132     cnt = 1
133     l = fis.readline();
134     rownum_list = []
135     tokens_list = []
136     while (cnt != max_cnt and l != ''):
137         rownum, sentences = remove_tagwords(l).split('\t')
138         rownum_list.append(rownum)
139         #トーカナイズ（1文章を単語ごとに分割した一次元配列）
140         tokens = tokenizer.parse(sentences).split()
141         #トークンリスト（複数行の文章、単語の二次元配列）
142         tokens_list.append(tokens)
143         cnt += 1
144         l = fis.readline();
145
146     return (rownum_list, tokens_list)
147
148 re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
149 #解析データの項目名を単語から除去
150 def remove_tagwords(sentences):
151     return re_tagwords.sub('', sentences)
152
153 #トークンリストを辞書変換するメソッド
154 def tokenslist2dic(tokens_list):
155     return corpora.Dictionary(tokens_list)
156
157 #辞書をbag-of-words形式のリストに変換するメソッド
158 def doc2bow(tokens_gdic, tokens_list):
159     return list(map(tokens_gdic.doc2bow, tokens_list))
160
161 if __name__ == '__main__':
162     any_below = arg_below()
163     main(any_below, 1)
164
165
166
```

機械学習用データセット作成プログラム (LSI)

```
from datetime import datetime
from gensim import corpora
from gensim import models
from gensim import matutils
import MeCab
import math
import re
import os

def new_idf(docfreq, totaldocs, log_base=2.0, add=1.0):
    return add + math.log(1.0 * totaldocs / docfreq, log_base)

def main(mode = 0):
    max_cnt = 0
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']開始しま
す。')

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = 'LSI/'
    os.makedirs(dirname,mode=511,exist_ok=True)

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt) )
+ '05_sogyouretu.csv','w')
    fos2 = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt)
) + '05_total_wordnum.txt','w')

    #作成済み辞書の読み込み先リンク
    gdic_fname = './new_shibo_join.dict'

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き開
始')
    #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
    rownum_list,tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']分かち書き終
了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成開始')
    #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
    if mode == 0:
        #トークンリストを辞書変換
        tokens_gdic = tokenslist2dic(tokens_list)
        #辞書情報を保存
        #save_gdic(tokens_gdic, gdic_fname)
    elif mode == 1:
        #辞書情報の読み込み
        tokens_gdic = load_gdic(gdic_fname)

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']dictionary
作成終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
開始')
    #辞書をbag-of-words形式のリスト（corpus）に変換
    tokens_corpus = doc2bow(tokens_gdic, tokens_list)
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']corpus作成
終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成開
始')
    test_model = models.TfidfModel(tokens_corpus,wglobal=new_idf)

    corpus_tfidf = test_model[tokens_corpus]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']model作成終
了')

    #0回出現の単語についても重要度0を出力する
    #new_dic = [{wordid:0 for word,wordid in dictionary.token2id.items()} for
```

```
c in id_list]
    #辞書の総単語数を取得
    """
    total_wordnum = 0
    for word,wordid in tokens_gdic.token2id.items():
        total_wordnum += 1
    fos2.write(str(total_wordnum))
    """
    #LSI modelによる次元圧縮したcorpusの作成
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI開始')
    lsi_model = models.LsiModel(corpus_tfidf, id2word=tokens_gdic,
num_topics=200)
    lsi_model.save('lsi_topics200.model')
    #lsi_model = models.LsiModel.load('lsi_topics200.model')
    corpus_lsi = lsi_model[corpus_tfidf]
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']LSI終了')

    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
開始')
    vec_size = 0
    for rowid,doc in zip(rownum_list,corpus_lsi):
        lil =[]
        for word in doc:
            lil.append((word[0],word[1]))

        if len(lil) != 0:
            vec_size = len(lil)
        fos.write(data_make(rowid,lil))
    fos2.write(str(vec_size))
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']ベクトル出力
終了')

    fis.close()
    fos.close()
    fos2.close()
    print('['+ datetime.now().strftime('%Y-%m-%d %H:%m:%S.%f') + ']終了しま
す。')

#出力の形式を整えるメソッド
def data_make(rowid,lil):
    return rowid + '\t['+','.join([sogyouretu(kv) for kv in lil])+']\n'

#tfidf値を疎行列の形にして返すメソッド
def sogyouretu(kv):
    return str(kv[0])+':'+str('{:f}'.format((kv[1]+1)/2))

    #for rowdic,rowid in zip(new_dic,id_list):
        #print(rowid+'\t',','.join([str(k)+','+str(v) for k,v in
sorted(rowdic.items())]))
        #print(rowid+'\t',','.join([str(v) for k,v in rowdic.items()]))
        #fos.write(rowid+'\t'+','.join([str(v) for k,v in
sorted(rowdic.items())])+'\n')

    """
    #出現０回の文字について重要度０を入れない出力
    texts_tfidf = []
    for rowid,doc in zip(id_list,corpus_tfidf):
        text_tfidf = []
        for word in doc:
            text_tfidf.append(word[1])
        texts_tfidf.append(text_tfidf)
        print(rowid+'\t',','.join([str(i) for i in text_tfidf]))
    """

#作成済みdictionaryを読み込むメソッド
def load_gdic(fname):
    return corpora.Dictionary.load(fname)

#作成したdictionaryを保存するメソッド
def save_gdic(tokens_gdic, fname):
    tokens_gdic.save(fname)

#形態素解析にMeCabを使用し、トーカナイザーを取得するメソッド
```

```
def get_tokenizer():
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')

#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
def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
    cnt = 1
    l = fis.readline();
    rownum_list = []
    tokens_list = []
    while (cnt != max_cnt and l != ''):
        rownum, sentences = remove_tagwords(l).split('\t')
        rownum_list.append(rownum)
        #トーカナイズ（1文章を単語ごとに分割した一次元配列）
        tokens = tokenizer.parse(sentences).split()
        #トークンリスト（複数行の文章、単語の二次元配列）
        tokens_list.append(tokens)
        cnt += 1
        l = fis.readline();

    return (rownum_list, tokens_list)

re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
#解析データの項目名を単語から除去
def remove_tagwords(sentences):
    return re_tagwords.sub('', sentences)

#トークンリストを辞書変換するメソッド
def tokenslist2dic(tokens_list):
    return corpora.Dictionary(tokens_list)

#辞書をbag-of-words形式のリストに変換するメソッド
def doc2bow(tokens_gdic, tokens_list):
    return list(map(tokens_gdic.doc2bow, tokens_list))

if __name__ == '__main__':
    main(1)
```

機械学習用データセット作成プログラム (Word2Vec)

```
import sys
#sys.path.append('/home/bmiwork/ITEC')
from my_utils import exec_time

import MeCab
import re
from gensim.models import word2vec
from gensim import corpora
#from gensim import matutils
import numpy as np
import os

@exec_time.printer
def main(mode = 0):
    max_cnt = 0

    #疎行列ベクトル出力先ディレクトリ作成
    dirname = 'WORD2VEC/'
    os.makedirs(dirname,mode=511,exist_ok=True)

    #トーカナイザを取得
    tokenizer = get_tokenizer()

    #読み込みファイルと出力ファイル
    fis = open('./new_shibo_join_concat.tsv','r')
    fos = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt) )
+ '05_sogyouretu.csv','w')
    fos2 = open('./'+ dirname + ( '' if max_cnt <= 0 else '_' + str(max_cnt)
) + '05_total_wordnum.txt','w')

    #作成済みmodelの出力先リンク
    #vec_fname = './new_shibo_join' + ( '' if max_cnt <= 0 else '_' +
str(max_cnt) ) + '.vec.pt'
    vec_fname = "./WORD2VEC/entity_vector.model.bin"

    #対象文書の形態素を格納
    rownum_list, tokens_list = fetch_tokenslist(fis, tokenizer, max_cnt)
    #print(tokens_list[0])

    if mode == 0:
        #word2vecのmodelに形態素を学習
        model = fit_word2vec(tokens_list)
        #modelを保存
        save_model(model, vec_fname)
    elif mode == 1:
        model = load_model(vec_fname)

    #各行の文書のベクトルを計算して出力
    write_vector(fos, fos2, model, rownum_list, tokens_list)

    fis.close()
    fos.close()
    fos2.close()

#word2vec学習モデルの読み込み
def load_model(fname):
    return word2vec.KeyedVectors.load_word2vec_format(fname, binary=True)

#word2vec学習モデルの保存
def save_model(model, fname):
    model.wv.save_word2vec_format(fname, binary=True)

@exec_time.printer
def write_vector(fos, fos2, model, rownum_list, tokens_list):
    #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
    num_features = 200
    for rowid, doc in zip(rownum_list,tokens_list):
        #print(rowid)
        #print(doc[0])
        #単語を200次元のベクトルとして出力したもの
        """
        vec_list = []
        for word in doc:
```

```
            if word in model.key_to_index:
                vec = model[word]
                vec_list.append(vec)
        fos.write(data_make(rowid, doc, vec_list))
        """
        #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
        if doc != []:
            feature_vec = np.zeros((num_features), dtype= "float32")
            for word in doc:
                if word in model.key_to_index:
                    feature_vec = np.add(feature_vec, model[word])
                    #feature_vec = np.add(feature_vec, model.get_vector(word,
norm=True))
            if len(doc) > 0:
                feature_vec = np.divide(feature_vec, len(doc))
        else:
             feature_vec = []
        #print(feature_vec)
        fos.write(data_make2(rowid, feature_vec, num_features))
    fos2.write(str(num_features))

#出力の形式を整えるメソッド
def data_make(rowid, doc, vec_list):
    return rowid + '\t['+ ','.join([sogyouretu(wordid,vec) for wordid, vec in
zip(doc, vec_list)])+']\n'

#出力の形式を整えるメソッド
def data_make2(rowid, feature_vec, vec_cnt):
    cnt_list = []
    for num in range(vec_cnt):
        cnt_list.append(num)
    if feature_vec != []:
        return rowid + '\t['+ ','.join([sogyouretu2(v,i) for v,i in
zip(feature_vec, cnt_list)])+']\n'
    else:
        return rowid + '\t[]\n'

#ベクトルを疎行列形式に整えるメソッド
def sogyouretu(wordid, vec):
    return str(wordid)+':'+str(vec)

#ベクトルを疎行列形式に整えるメソッド
def sogyouretu2(vec, cnt):
    return str(cnt)+':'+str('{:f}'.format((vec+10)/20))
    #return str(cnt)+':'+str(vec)

@exec_time.printer
def get_tokenizer():
    #形態素解析にMeCabを使用する
    return MeCab.Tagger('-r /etc/mecabrc -Owakati')

@exec_time.printer
def fetch_tokenslist(fis, tokenizer, max_cnt = 0):
    cnt = 1
    l = fis.readline()
    rownum_list = []
    tokens_list = []
    while (cnt != max_cnt and l != ''):
        rownum, sentences = remove_tagwords(l).split('\t')
        #if(sentences != ''):
        rownum_list.append(rownum)
        #トーカナイズ（1文章を単語ごとに分割した一次元配列）
        tokens = tokenizer.parse(sentences).split()
        #ストップワードの除去
        tokens = filter_stopwords(tokens)
        #トークンリスト（複数の文章、単語の二次元配列）
        tokens_list.append(tokens)
        cnt += 1
        l = fis.readline();
    return (rownum_list, tokens_list)

re_tagwords = re.compile(r'(【手術】)|(【解剖】)|(【状況】)|(【母体】)|(【付言】)|
(【備考】)|\n')
```

```
def remove_tagwords(sentences):
    return re_tagwords.sub('', sentences)

stopword_list = None
def filter_stopwords(tokens):
    global stopword_list
    if stopword_list is None:
        stopword_list = []
        #ストップワードの読み込み
        fis = open('./stopwords.txt','r')
        re_comment = re.compile(r'^\s*(#.*|)$')
        l = fis.readline()
        while l:
            if not re_comment.match(l):
                stopword_list.append(l.rstrip('\n'))
            l = fis.readline()
        fis.close()
    return [word for word in tokens if word not in stopword_list]

@exec_time.printer
def tokenslist2dic(tokens_list):
    return corpora.Dictionary(tokens_list)

@exec_time.printer
def fit_word2vec(tokens_list):
    #word2vecによる単語の学習
    return word2vec.Word2Vec(tokens_list)

if __name__ == '__main__':
    main(1)


```

```
#!/usr/bin/env python3
# -*- coding: utf-8 -*-

#=================================================#
#==                                             ==#
#==                 [embedding.py]              ==#
#==                   version.1.1               ==#
#==                   2022/03/28                ==#
#==                                             ==#
#=================================================#

import pandas as pd
import numpy as np
import argparse
from argparse import RawTextHelpFormatter
import ast
import MeCab
from gensim.models.doc2vec import Doc2Vec, TaggedDocument
import tensorflow_hub as hub
import tensorflow_text
import re
import os

import ssl
ssl._create_default_https_context = ssl._create_unverified_context

# Embeddingクラス作成
class Embedding(object):
  def __init__(self, rawdata_path, delete_symbol, params, mode, output_path):
    self.rawdata_path = rawdata_path      # 入力ファイルのパス
    self.delete_symbol = delete_symbol    # 形態素解析時に削除する記号
    self.params = params                        # doc2vecを使用する際の引数(辞書
型)
    self.mode = mode
    self.output_path = output_path

  def check_mode(self):
    '''
    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
る。
    params['dm']を設定していない場合は、modeがparams['dm']の引数となる
    '''
    if self.mode==2:
      print('mode: Universal Sentence Encoder')

    elif self.mode==0 or self.mode==1:
      if 'dm' not in self.params:
        self.params['dm']=self.mode
        if self.mode==0:
          print('mode: doc2vec PV-DBOW')
        elif self.mode==1:
          print('mode: doc2vec PV-DM')

      elif 'dm' in self.params:
        if self.params['dm'] == self.mode:
          if self.mode==0:
            print('mode: doc2vec PV-DBOW')
          elif self.mode==1:
            print('mode: doc2vec PV-DM')
        else:
          raise KeyError('modeとparams["dm"]が一致していません. mode: {},
params["dm"]: {}'.format(self.mode, self.params['dm']))

    else:
      raise ValueError('--mode 0~2から選んでください {0: doc2vec PV-DBOW, 1:
doc2vec PV-DM, 3: Universal Sentence Encoder}')


  def get_data(self):
    '''
    rawdataの読み込み。入力ファイルは一列目にid、二列目にテキストデータを含む、hedderおよ
びindex_colの無い.tsvファイル。
    '''
```

```
    # 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
    src = pd.read_csv('{}'.format(self.rawdata_path), delimiter='\t',
index_col=None, header=None, names=['id', 'data'])
    # テキストデータがNaNのサンプル行を削除
    src_data = src.dropna(how='any')

    sentences = []
    for text in src_data['data']:
        # テキスト内の任意の記号を削除
        text = re.sub(r'{}'.format(self.delete_symbol), '', text)

        # 各行のテキストを１要素(リスト)としてsetencesに二次元リストとして格納
        text_list = text.split(' ')
        sentences.append(text_list)

    return src, sentences

  def doc2vec(self, sentences):
    '''
    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。
    '''
    token = []
    tokernizer = MeCab.Tagger('-Owakati')     # Mecabの形態素解析の引数設定

    # setences内の各要素(テキスト)を形態素解析
    for s in sentences:
      token.append(tokernizer.parse(s[0]).split())

    # 以下処理過程の参考
    # 例 src_data.iloc[0,1] : 【付言】ＤＮＡ検査にて本人確認。
    # 例 sentences[0] : ['付言ＤＮＡ検査にて本人確認']
    # 例 token[0] : ['付言', 'ＤＮＡ', '検査', 'にて', '本人', '確認']

    # tokenをdoc2vecに読み込ませる形式に変換
    documents = [TaggedDocument(doc, [i]) for i, doc in enumerate(token)]

    #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
    import hashlib
    hashfxn = lambda x: int(hashlib.md5(str(x).encode()).hexdigest(), 16)

    # doc2vecモデル構築
    model = Doc2Vec(documents,hashfxn=hashfxn, **self.params)
    vectors = [model.docvecs[i] for i in range(len(sentences))]

    return vectors

  def UniversalSentenceEncoder(self, sentences):
    '''
    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。
    '''
    # tensorflow hubから学習済みモデルの読み込み
    USE_model = hub.load(
    "https://tfhub.dev/google/universal-sentence-encoder-multilingual/3")  #
2022/01時点, latest version:はver.3
    vectors_all = USE_model(sentences)
    vectors = [i for i in vectors_all]

    return vectors

  def output(self, src, vectors):
    '''
    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。
    '''
    src_data = src.dropna(how='any')
    src_data_vec = src_data.assign( vecs = vectors)
    df = pd.merge(src, src_data_vec, how="left", on = "id")
    df = df.loc[:, ['id','vecs']]

    # output_pathを任意指定した時のみファイルを出力する
    if self.output_path=='None':
        pass
```

```
    else:
        output_path = self.output_path
        df.to_csv(output_path, sep='\t', header=False, index=False)

    return df

def main():
  parser = argparse.ArgumentParser(prog='embedding',
                       description='----------------------------------------
----------------------------------\n'
                       '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'
                       '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
ログラム。\n'
                       'doc2vecまたはuniversal setence encoderを用いてベクトル化す
る。\n'
                       '_____________________________________________________
----------------------\n',
                       usage='$ python3 embedding.py [-m] [-rp] [-ds] [-
params] [-op]\n',
                       formatter_class=RawTextHelpFormatter,
                       add_help=True)
  parser.add_argument('--mode', '-m', type=int, required=True,
                          help='モデルの選択\n'
                          'mode:{0: doc2vec PV-DBOW, 1: doc2vec PV-DM, 2:
Universal Sentence Encoder}')
  parser.add_argument('--rawdata_path', '-rp', type=str, required=True,
                          help='入力ファイルのパス. \n'
                          '入力ファイルは一列目にid、二列目にテキストデータを含む、
hedderおよびindex_colの無い.tsvファイル')
  parser.add_argument('--delete_symbol', '-ds', type=str, default='[【】、。
「」（）]',
                          help='形態素解析時の削除したい記号\n'
                          '削除したい記号をカッコ内に入れ[ ]、str型で渡す\n'
                          'e.g. [PROGRAM] --delete_symbol "[【】、。「」（）]"')
  parser.add_argument('--doc2vec_params', '-params', type=str, default="
{'vector_size':2}",
                          help='doc2vecに与える引数をstr型で入力する. 引数はモジュー
ル内で辞書型に変換される.\n'
                          '''e.g. [PROGRAM] -params "{'vector_size' : 2,
'window': 2, 'min_count': 1, 'epochs': 10}"''')
  parser.add_argument('--output_path', '-op', type=str, default='None',
                        help='出力ファイルのパス. \n'
                        '出力ファイルは一列目にid、二列目にベクトルデータを含む、
hedderおよびindex_colの無い.tsvファイル\n'
                        '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
い')
  args = parser.parse_args()

  # 引数受け取り
  mode = args.mode
  rawdata_path = args.rawdata_path
  delete_symbol =args.delete_symbol
  params = ast.literal_eval(args.doc2vec_params)
  output_path = args.output_path

  # インスタンス作成
  E = Embedding(rawdata_path, delete_symbol, params, mode, output_path)
  # modeとparamsの確認
  E.check_mode()

  # 入力データ(rawdata)読み込み
  src, sentences = E.get_data()

  # doc2vecによる実行
  if mode==0 or mode==1:
    vectors = E.doc2vec(sentences)
  # USEによる実行
  elif mode==2:
    vectors = E.UniversalSentenceEncoder(sentences)

  # 出力用データフレーム作成
  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)

```
import pandas as pd
import xgboost as xgb
import numpy as np
import pickle
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from matplotlib import pyplot as plt
from sklearn.metrics import confusion_matrix, roc_curve,
precision_recall_curve, auc
from sklearn.model_selection import GridSearchCV
from scipy.sparse import lil_matrix
from datetime import datetime
import sys
import os

def arg_parse():
    import argparse
    p = argparse.ArgumentParser()
    p.add_argument('-a', '--anyfolder', help='疎行列ベクトル参照先フォルダ')
    p.add_argument('-g', '--gpuid', help='使用するGPUのID（デフォルトは0')
    p.add_argument('-y', '--year', help='学習用データ年')
    a = p.parse_args()
    return (a.anyfolder, a.gpuid, a.year)

def main(any_folder='.', gpuid=0, year=2020):
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] ' +
__file__  + ' start')

    # 引数の検証
    if not any_folder: any_folder = '.'
    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 + ']')
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 出力先フォ
ルダ[./' + any_folder + '/RESULT/' + year + '/]')
    #os.makedirs('./RESULT/' + any_folder, exist_ok=True)
    # 疎行列行列数ファイル読み込み
    # rows = 81688   # 行数
    # cols = 1579 + 2736  # 次元数
    fis = open('../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_rowcolnum_' + year + '.txt', 'r')
    rows = int(fis.readline())
    cols = int(fis.readline())
    fis.close()
    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) + ']')

    # 疎行列ファイルを読み込み
    smatrix_fname = '../07_付帯情報Embedding/' + any_folder +
'/learningData_smatrix_' + year + '.txt'
    X, y = load_sparse_matrix(smatrix_fname, rows, cols)
    # 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
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
shuffle=True, random_state=42, stratify=y)

    # 学習用の0.8で学習
    # 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。
    # ----- Best Estimator
    clf = xgb.XGBClassifier(base_score=0.5, booster='gbtree',
colsample_bylevel=1,
                colsample_bynode=1, colsample_bytree=1, eta=0.25, gamma=0,
                gpu_id=gpuid, importance_type='gain',
interaction_constraints='',
                learning_rate=0.25, max_delta_step=0, max_depth=14,
                min_child_weight=0.05, monotone_constraints='()',
                n_estimators=2000, n_jobs=1, num_parallel_tree=1,
random_state=0,
                reg_alpha=0, reg_lambda=1, scale_pos_weight=1, subsample=1,
                tree_method='gpu_hist', use_label_encoder=False,
                validate_parameters=1, verbosity=None, eval_metric='logloss')
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] fit
start')
    clf.fit(X_train, y_train)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] fit
```

```
end')

    # モデルを保存
    pickle.dump(clf, open('./' + any_folder + '/' + year + '/model.pkl',
'wb'))
    clf = pickle.load(open('./' + any_folder + '/' + year + '/model.pkl',
'rb'))

    # テスト用の0.2で検証
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
start')
    # pred = clf.predict(X_test)
    pred_p = clf.predict_proba(X_test)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] predict
end')

    # 結果出力
    report = []
    for i in range(21):
        t = (i/20)
        pred = (pred_p[:, 1] > t).astype(int)
        print('-----===== 閾値[' + str(t) + '] ==========')
        accuracy = print_accuracy(y_test, pred)
        tp, fp, fn, tn, precision, recall = print_confusion_matrix(y_test,
pred)
        report.append([t, accuracy, tn, fp, fn, tp, precision, recall, tp+fp,
(tp+fp)/(tp+fp+fn+tn)])
        write_id_tf(X[:, 0], y_test, pred, t, any_folder, year)

    if any_folder == 'TFIDF_100':
        print_importances(clf, any_folder, year)

    y_score = pred_p[:,1]
    make_pr_curve_pict(y_test, y_score, any_folder, year)
    make_roc_curve_pict(y_test, y_score, any_folder, year)

    write_report(report, any_folder, year)
    write_pred_score(X[:, 0], y_test, pred_p[:, 1], any_folder, year)

    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] ' +
__file__  + ' end')

def load_sparse_matrix(fname, rows, cols):
    # 疎行列ファイルをオープン
    fis = open(fname, 'r')

    X_lil = lil_matrix((rows, cols - 1), dtype=float)
    # y_lil = lil_matrix((rows, 1), dtype=float)
    y = [0] * rows

    # 疎行列ファイルの読み込み
    l = fis.readline()
    rownum = 0
    while l != '':
        l_smatrix = sogyoretu2smatrix(l.rstrip('\n'))
        for k, v in l_smatrix.items():
            if k != cols - 1:
                if len(v) != 0:      # DOC2VEC 疎行列データエラー対策
                    X_lil[rownum, k] = float(v)
                else:
                    print("[ERROR]: '' can not be converted to float @" +
str(rownum))
                    X_lil[rownum, k] = 0
            else:
                ## y_lil[rownum, 0] = float(v)
                y[rownum] = float(v)

        l = fis.readline()
        rownum += 1

    fis.close()

    # sparseからnumpy.ndarrayに変換
```

```
    X = X_lil.toarray()
    # y = y_lil.toarray()
    return X, y

def sogyoretu2smatrix(sogyoretu):
    """
    01_sogyouretu.csvの疎行列表記文字列を疎行列のdictionaryに変換する。
    """
    # 先頭末尾の[]を外す
    # ,でスプリットしkvセットに切り分る
    # :でスプリットしてkeyとvalueを分ける
    return {int(kv.split(':')[0]): kv.split(':')[1] for kv in
sogyoretu[1:-1].split(',') if kv != ''}

def print_accuracy(act, pred):
    """
    精度を出力する。
    """
    acc = accuracy_score(act, pred)
    print('----- Accuracy')
    print(acc)
    return acc

def print_confusion_matrix(act, pred):
    """
    混合配列を出力する。
    """
    cm = {}
    cm[(0,0)] = 0
    cm[(0,1)] = 0
    cm[(1,0)] = 0
    cm[(1,1)] = 0
    for t, p in zip(act, pred):
        cm[(t, p)] += 1
    print('----- Confusion Matrix')
    print('(act, pred): ', cm)
    print('----- Confusion Matrix')
    print(pd.DataFrame(confusion_matrix(act, pred, labels=[0, 1]), columns=
["pred_0", "pred_1"], index=["act_0", "act_1"]))

    p, r = print_pr(cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)])
    return (cm[(1,1)], cm[(0,1)], cm[(1,0)], cm[(0,0)], p, r)

def print_pr(tp, fp, fn, tn):
    p = tp/(tp+fp) if (tp+fp)!=0 else 1
    print('----- Precision')
    print(str(p))
    r = tp/(tp+fn) if (tp+fn)!=0 else 1
    print('----- Recall')
    print(str(r))
    return (p, r)

def print_importances(clf, any_folder, year):
    """
    重要度を出力する。
    """
    import os
    from gensim import corpora

    dict_fname = '../07_付帯情報Embedding/' + any_folder +
'/new_shibo_join.dict'
    tokens_dict = None
    if os.path.exists(dict_fname) and os.path.getsize(dict_fname):
        tokens_dict = corpora.Dictionary.load(dict_fname)

    icdcode_fname = '../06_機械学習用基本データ/ICDList/acme_kari_code_sort_' +
year + '.txt'
    icdcodes = ['certificateKey', '年齢', '性別']
    fis = open(icdcode_fname, 'r')
    lines = 0
    l = fis.readline()
    while(l != ''):
        icdcodes.append(l.rstrip('\n'))
```

```
        l = fis.readline()
        lines += 1

    fis.close()

    icdcodes.append('手術フラグ(1)')
    icdcodes.append('手術の部位及び所見(116)')
    icdcodes.append('（手術）備考欄への記載(1)')
    icdcodes.append('手術日(8)')
    icdcodes.append('解剖フラグ(1)')
    icdcodes.append('解剖の部位及び所見(116)')
    icdcodes.append('（解剖）備考欄への記載(1)')
    icdcodes.append('死因の種類(2)')
    icdcodes.append('傷害が発生したとき(8)')
    icdcodes.append('傷害が発生したとき(5)')
    icdcodes.append('傷害が発生したところの種別(1)')
    icdcodes.append('傷害が発生したところその他の記述(40)')
    icdcodes.append('傷害発生場所(8)')
    icdcodes.append('傷害発生場所(12)')
    icdcodes.append('傷害発生場所(18)')
    icdcodes.append('手段及び状況(120)')
    icdcodes.append('（傷害）備考欄への記載(1)')
    icdcodes.append('「生後1年未満での病死」の病態・異状の詳細(84)')
    icdcodes.append('備考欄への記載(1)')
    icdcodes.append('その他付言すべき事柄(60)')
    icdcodes.append('備考欄外字有無(1)')
    icdcodes.append('備考欄(1024)')

    lines += 25
    imps = sorted(enumerate(clf.feature_importances_), key=lambda kv: -kv[1])
    print('----- Feature Importances')
    for k, v in imps:
        print(str(k) + ',' + (icdcodes[k] if k<lines else tokens_dict[k-
lines] if tokens_dict else '-')  + ',' + str(v))

def make_pr_curve_pict(y_test, y_score, any_folder, year):
    # PR取得
    precision, recall, thresholds = precision_recall_curve(y_true=y_test,
probas_pred=y_score)
    # 結果をプロット
    plt.figure(1, figsize=[12.8, 9.6], dpi=100)
    plt.figure(1)
    plt.plot(recall, precision, label=(any_folder if any_folder != '.' else
'precision_recall curve') + ' (AUC = %0.3f)' % auc(recall, precision))
    for i in range(21):
        close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
        plt.plot(recall[close_point], precision[close_point], 'o')
     # ラベルなどを追加しファイル出力
    plt.plot([0,1], [1,1], linestyle='--', label='ideal line')
    plt.legend()
    plt.xlabel('recall')
    plt.ylabel('precision')
    plt.set_title=('P-R Curve')
    fname = './' + any_folder + '/' + year + '/PR-Curve.png'
    plt.savefig(fname)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def make_roc_curve_pict(y_test, y_score, any_folder, year):
    # ROC取得
    fpr, tpr, thresholds = roc_curve(y_true=y_test, y_score=y_score)
    # 結果をプロット
    plt.figure(2, figsize=[12.8, 9.6], dpi=100)
    plt.figure(2)
    plt.plot(fpr, tpr, label=(any_folder if any_folder != '.' else 'roc
curve') + ' (AUC = %0.3f)' % auc(fpr, tpr))
    for i in range(21):
        close_point = np.argmin(np.abs(thresholds - (i * 0.05)))
        plt.plot(fpr[close_point], tpr[close_point], 'o')
    # ラベルなどを追加しファイル出力
    plt.plot([0,0,1], [0,1,1], linestyle='--', label='ideal line')
    plt.plot([0, 1], [0, 1], linestyle='--', label='random prediction')
    plt.legend()
    plt.ylabel('false positive rate(FPR)')
```

```
    plt.xlabel('false positive rate(FPR)')
    plt.ylabel('true positive rate(TPR)')
    plt.set_title=('ROC Curve')
    fname = './' + any_folder + '/' + year + '/ROC-Curve.png'
    plt.savefig(fname)
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_report(report, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_report.csv'
    fos = open(fname, 'w')
    for l in list(zip(*report)):
        fos.write(','.join([str(f) for f in l]) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_id_tf(ids, act, pred, ikichi, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_id_tf_ikichi_' + str(ikichi) + '.csv'
    fos = open(fname, 'w')
    for i, a, p in list(zip(ids, act, pred)):
        fos.write( '{:.0f},{:.0f},{:.0f},{}'.format(i, a, p, a==p) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

def write_pred_score(ids, act, pred_score, any_folder, year):
    fname = './' + any_folder + '/' + year +
'/fit_and_predict_xgboost_pred_score.csv'
    fos = open(fname, 'w')
    for i, a, p in list(zip(ids, act, pred_score)):
        fos.write( '{:.0f},{:.0f},{:.5f}'.format(i, a, p) + '\n')
    fos.close()
    print('[' + datetime.now().strftime('%Y-%m-%d %H:%M:%S.%f') + '] 結果を出力
しました。[' + fname + ']')

if __name__ == '__main__':
    any_folder, gpuid, year = arg_parse()
    main(any_folder, gpuid, year)


```