**[Day-01-Lecture-07]**

## 관련 논문

> **권기성, 최운호, 김동건 (2022), "문학 작품의 거리 측정을 활용한 야담의 이본 연구",「한국고전연구 57집」87~120쪽, 한국고전연구학회.**

- 이번에는 정렬을 단락 단위로 생성하고 연구용으로 만들어 놓은 정렬 결과를 만드는 과정입니다.
- 중간 단계는 하나 생략하고 결과 중심으로 보겠습니다.

**[폴더] okssw_05 > pairs_00**

- 이 폴더에 제공하는 파일은 작품집 별로 대응되는 단락을 서로 짝을 맞추어 놓은 파일들입니다.
- 이 파일들에서 어떻게 문자 단위 정렬과 대응이 이루어지는지 단계별로 확인해 보도록 하겠습니다.

**[폴더] okssw_05 > pairs_01**

- 이 폴더에서는 edit_dist_align.py 를 활용해서 대응되는 파일의 aligned text를 생성해 냅니다.
- backtracing & alignment는 다음 사이트의 코드를 참조했습니다.

  > https://giov.dev/2016/01/minimum-edit-distance-in-python.html

- 이 코드를 실행하기 위해서는 다음 패키지를 설치해야 합니다.

```
pip install numpy
pip install tabulate
pip install argparse
```

- 사용법은 다음과 같습니다.

```
python --input DKIM_ZKES.txt --output DKIM_ZKES.align
```

```
#!python
#
#!-*-code=utf-8-*-
#
#
# edit_dist_align.py
#
# backtracking & alignment
#
#  - https://giov.dev/2016/01/minimum-edit-distance-in-python.html
#
#

```

```python
import numpy as np
import tabulate as tb
import argparse

def wagner_fischer(word_1, word_2):
    n = len(word_1) + 1 # couting empty string
    m = len(word_2) + 1 # couting empty string

    # init. D(istance) matrix

    D = np.zeros(shape=(n, m), dtype=np.int)
    D[:, 0] = range(n)
    D[0, :] = range(m)

    # B is the backtrack matrix. At each index, it contains a triple
    # of booleans, used as flag.
    # If B(i, j) = (1, 1, 0) for example,
    # the distance computed in D(i, j) came from a deletion or a
    # substitution. This is used to compute backtracking later.

    B = np.zeros(shape=(n, m), dtype=[("del", 'b'), ("sub", 'b'), ("ins", 'b')])

    B[1:, 0] = (1, 0, 0)
    B[0, 1:] = (0, 0, 1)

    for i, l_1 in enumerate(word_1, start=1):
        for j, l_2 in enumerate(word_2, start=1):
            deletion = D[i-1, j] + 1
            insertion = D[i, j-1] + 1
            substitution = D[i-1, j-1] + (0 if l_1 == l_2 else 2)

            mo = np.min([deletion, insertion, substitution])

            B[i, j] = (deletion==mo, substitution==mo, insertion==mo)
            D[i,j] = mo
    return D, B

def naive_backtrace(B_matrix):
    i, j = B_matrix.shape[0] - 1, B_matrix.shape[1] - 1
    backtrace_idxs = [(i, j)]

    while (i, j) != (0, 0):
        if B_matrix[i, j][1]:
            i, j = i-1, j-1
        elif B_matrix[i, j][0]:
            i, j = i-1, j
        elif B_matrix[i, j][2]:
            i, j= i, j -1
        backtrace_idxs.append((i, j))

    return backtrace_idxs

def align(word_1, word_2, bt):

```

```
    aligned_word_1 = []
    aligned_word_2 = []
    operations = []

    backtrace = bt[::-1]    # make it a forward trace

    for k in range(len(backtrace) - 1):
        i_0, j_0 = backtrace[k]
        i_1, j_1 = backtrace[k+1]

        w_1_letter = None
        w_2_letter = None

        # either substitution or no_op
        if i_1 > i_0 and j_1 > j_0:
            if word_1[i_0] == word_2[j_0]:
                w_1_letter = word_1[i_0]
                w_2_letter = word_2[j_0]
                op = " "
            #cost increased: substitution
            else:
                w_1_letter = word_1[i_0]
                w_2_letter = word_2[j_0]
                op = "s"
        # insertion
        elif i_0 == i_1:
            w_1_letter = " "
            w_2_letter = word_2[j_0]
            op = "i"
        # j_0 == j_1, deletion
        else:
            w_1_letter = word_1[i_0]
            w_2_letter = " "
            op = "d"

        aligned_word_1.append(w_1_letter)
        aligned_word_2.append(w_2_letter)
        operations.append(op)

    return aligned_word_1, aligned_word_2, operations

def make_table(word_1, word_2, D, B, bt):
    #w_1 = word_1.upper()
    #w_2 = word_2.upper()
    w_1 = word_1
    w_2 = word_2

    w_1 = "#" + w_1
    w_2 = "#" + w_2

    table = []
    # table formatting in emac, you probably don't need this line
    table.append(["<r>" for _ in range(len(w_2)+1)])
    table.append([""] + list(w_2))

```

```
    max_n_len = len(str(np.max(D)))

    for i, l_1 in enumerate(w_1):
        row = [l_1]
        for j, l_2 in enumerate(w_2):
            v, d, h = B[i, j]
            direction = ("⇑" if v else "") + ("⇖" if d else "") + ("⇐" if h else "")
            dist = str(D[i,j])

            cell_str = "{direction} {star}{dist} {star}".format(direction=direction, star=" *"[((i,j) in bt)], dist=dist)
            row.append(cell_str)
        table.append(row)

    return table


def get_lv_table(w1, w2):
    word_1 = w1
    word_2 = w2

    D, B = wagner_fischer(word_1, word_2)
    bt = naive_backtrace(B)

    edit_distance_table = make_table(word_1, word_2, D, B, bt)

    print("Edit Distance with Backtrace:")
    print("#+ATTR_HTML: :border 2:rules all : frame border :style text-align: right")
    print(tb.tabulate(edit_distance_table, stralign="right", tablefmt="orgtbl"))

def get_alignment(w1, w2):
    word_1 = w1
    word_2 = w2

    D, B = wagner_fischer(word_1, word_2)
    bt = naive_backtrace(B)

    alignment_table = align(word_1, word_2, bt)

    #print("\nAlignment")
    #print(tb.tabulate(alignment_table, tablefmt="orgtbl"))

    return alignment_table


def make_alignment(w1, w2):
    word_1 = w1
    word_2 = w2

    D, B = wagner_fischer(word_1, word_2)
    bt = naive_backtrace(B)

```

```
    edit_distance_table = make_table(word_1, word_2, D, B, bt)
    alignment_table = align(word_1, word_2, bt)

    print("Edit Distance with Backtrace:")
    print("#+ATTR_HTML: :border 2:rules all : frame border :style text-
align: right")
    print(tb.tabulate(edit_distance_table, stralign="right",
tablefmt="orgtbl"))

    print("\nAlignment")
    print(tb.tabulate(alignment_table, tablefmt="orgtbl"))

def make_process(str_f_in, str_f_out):


    f_out = open(str_f_out, "w", encoding="utf-8")


    n_linecnt = 0
    with open(str_f_in, "r", encoding="utf-8") as f_in:
        for bline in f_in:
            bline = bline.strip("\n")
            cur_data = bline.split("\t")
            str_src = cur_data[5]
            str_target = cur_data[11]
            bscr = cur_data[0]
            btarget = cur_data[6]


            print("#seqpair" + "\t" + bline, file=f_out)
            aligned_result = get_alignment(str_src, str_target)
            print("#seqsrc" + "\t" + "\t".join(aligned_result[0]),
file=f_out)
            print("#seqtar" + "\t" + "\t".join(aligned_result[1]),
file=f_out)
            print("#seqopr" + "\t" + "\t".join(aligned_result[2]),
file=f_out)

            n_linecnt += 1
            print("current line: {0}".format(n_linecnt))


    f_out.close()



if __name__ == "__main__":

    # argument parsing
    argpar = argparse.ArgumentParser(description="edit_dist_align.py --
input infile_name --output outfile_name")

    argpar.add_argument('--input', required=True, help="Input File Name to
be aligned")
    argpar.add_argument('--output', required=True, help="Output File Name")
```

```
222
223     args = argpar.parse_args()
224
225     make_process(args.input, args.output)
```

---

**[폴더] okssw_05 > pairs_02 > html_arc**

- 이 폴더에서 실제 정렬된 파일들을 검토해 보도록 합시다.