chap2023.py

```python
# cos類似度と極性分類の確信度から類似文書を表示する。知識情報演習3に独自の工夫を追加。
# クエリに対して文書のスコアを出すプログラム(0 <= SCORE <= 1)


import math
import pandas as pd
import re
from janome.tokenizer import Tokenizer
from asari.api import Sonar
import os
from IPython.terminal.prompts import Token
import stopwords


sonar = Sonar()
t = Tokenizer()

#query = 'モラント 心配'
#query = 'ウォリアーズ 放出'
#query = 'プレイオフ ファイナル'
#query = 'W杯 日本'
query = 'オフシーズン ドラフト'


argv = query.split()
argc = len(argv)

# クエリ表示
print(argv)
# ストップワードリストを取得
stoplist = stopwords.stopwords()

# cos類似度を計算 ----------------------------------
def cos(argv):
    index_file = "index/index.txt"

    idf_scores = {}
    tfidf_scores = {}
    query_tf = {}
    query_tfidf = {}
    query_words = {}
    ranking_docs = {}

    #tfidf_scoresからデータフレームを作成
    f = open(index_file, 'r')

    for line in f:
        line = line.rstrip()
        split_line = line.split('\t')
        word = split_line[0]
        doc = split_line[1]
        idf = float(split_line[2])
        tfidf = float(split_line[3])

        if word not in tfidf_scores:
            tfidf_scores[word] = {}
```

```
            tfidf_scores[word][doc] = tfidf
        else:
            tfidf_scores[word][doc] = tfidf

        if word not in idf_scores:
            idf_scores[word] = {}
            idf_scores[word] = idf
        else:
            idf_scores[word] = idf

    tfidf_table = pd.DataFrame(tfidf_scores)
    tfidf_table = tfidf_table.fillna(0)


    #クエリに対する処理
    #argv = '吾輩は猫である'
    query_file = 'query'

    pattern = re.compile(r"^[ -—]$")


    #tokens = t.tokenize(argv[1])
    tokens = []
    for i in range(argc):
        list = t.tokenize(argv[i])
        tokens.append(list)
    flat_t = [item for sublist in tokens for item in sublist]

    flat_token = []
    for token in flat_t:
        tmp = token.surface
        judge = pattern.match(tmp)
        flat_token.append(tmp)

        if judge:
            continue
        if tmp in stoplist:
            continue

        if tmp in query_words:
            query_words[tmp] += 1
        else:
            query_words[tmp] = 1



    #クエリtfidf_scoresからクエリのデータフレームを作る
    for index_word in idf_scores:
        query_tf[index_word] = {}
        query_tfidf[index_word] = {}
        query_tf[index_word][query_file] = 0
        query_tfidf[index_word][query_file] = 0

    for query_word in query_words:
        if query_word in query_tf:
            query_tf[query_word][query_file] += 1

    for query_word in query_words:
        if query_word in query_tf:
```

```
            query_tfidf[query_word][query_file] = float(query_tf[query_word][query_file]) * float(idf_scores[query_word])

    query_table = pd.DataFrame(query_tfidf)

    #クエリが索引後辞書の中でどの文書名に属すのかを確認し、対象文書を同定
    for query_word in query_words:
        if query_word in tfidf_scores:
            for doc in tfidf_scores[query_word]:
                ranking_docs[doc] = 1

    #cos類似度計算
    for doc in ranking_docs:
        numerator = 0
        doc_vec = tfidf_table.loc[doc]
        query_vec = query_table.loc[query_file]
        for i in range(len(query_vec.values)):
            i_value = query_vec.values[i] * doc_vec.values[i]
            numerator = numerator + i_value

        denominator = 0
        query_value = 0
        doc_value = 0
        for i in range(len(query_vec.values)):
            query_value += query_vec.values[i]**2
            doc_value += doc_vec.values[i]**2

        denominator = math.sqrt(query_value) * math.sqrt(doc_value)
        cosine = numerator / denominator

        ranking_docs[doc] = cosine

    return ranking_docs.items() #sorted(ranking_docs.items(), key=lambda x:x[1], reverse=True)



# 文書のポジティブ/ネガティブを変数scaledで0から1で表示 ------------------------------
def text_emotions():
    data = []
    for filename in os.listdir('text'):
        doc_box = []
        f = open('text/' + filename, 'r')
        for row in f:
            row = row.rstrip()
            doc_box.append(row)
        result_doc = ''.join(doc_box)
        res = sonar.ping(text=result_doc)
        classes = res['classes']
        positive_confidence = next((item['confidence'] for item in classes if item['class_name'] == 'positive'), 0)
        negative_confidence = next((item['confidence'] for item in classes if item['class_name'] == 'negative'), 0)

        scaled = (positive_confidence - negative_confidence + 1) / 2
        data.append(scaled)
    return data


# クエリのポジティブ/ネガティブを変数scaledで-1から1で表示 ------------------------------
def query_emotions(argv):
```

```
    query = ' '.join(argv)
    res = sonar.ping(text=query)
    classes = res['classes']
    positive_confidence = next((item['confidence'] for item in classes if item['class_name'] == 'positive'), 0)
    negative_confidence = next((item['confidence'] for item in classes if item['class_name'] == 'negative'), 0)
    scaled = (positive_confidence - negative_confidence + 1) / 2
    return scaled



# ポジネガの検索式を定義 ----------------------------------------
# クエリと文書のポジネガが違かったら、距離を調整(0.5倍)
def dis_emotions(qemo, temo):
    if qemo >= 0.5:
        if temo >= 0.5:
            dis = abs(qemo - temo)
        else:
            dis = abs(qemo - temo) * 0.5
    elif qemo < 0.5:
        if temo <= 0.5:
            dis = abs(qemo - temo)
        else:
            dis = abs(qemo - temo) * 0.5
    scaled = 1 - dis*2
    return scaled



# 検索スコアのためのアルゴリズムを定義 ----------------
# score = cos類似度*1/3 + ポジネガ確信度*1/3
# 0 < score < 1 に調整
def score(cos, emo):
    return cos*(2/3) + emo*(1/3)



# クエリと文書のポジネガの距離を計算 -----------
te = text_emotions()
qe = query_emotions(argv)

emo_dis = []
counter = 1
for i in te:
    tmp = dis_emotions(qe, i)
    emo_dis.append(['doc{}'.format(counter),tmp])
    counter += 1


# 表示 ----------------------------------------
score_pair = []
cosinfo = cos(argv)
coslist = list(cosinfo)
for i in range(len(cosinfo)):
    doc_score = score(coslist[i][1], emo_dis[i][1])
    score_pair.append(['独自スコア: doc{} '.format(i+1), doc_score])

for i in sorted(score_pair, key=lambda x: x[1], reverse=True):
```

```
231     print(i)
```