資料工程 **Final Project**

404410030 資工四 鄭光宇

系統需求

要執行這支程式，系統必須具備：

- gcc, g++
- make
- apache2.0
- php7.0
- php7.0-mbstring

編譯

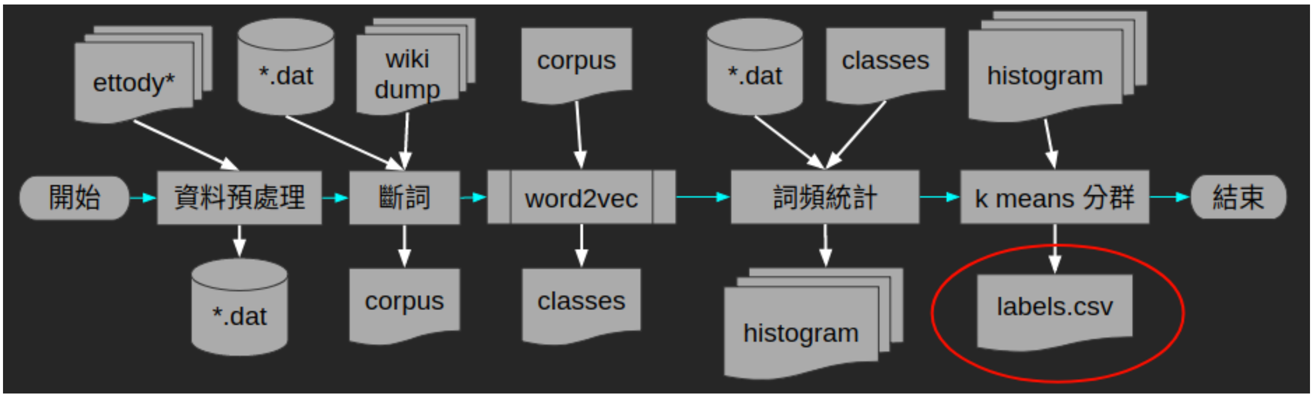
```
make
```

執行

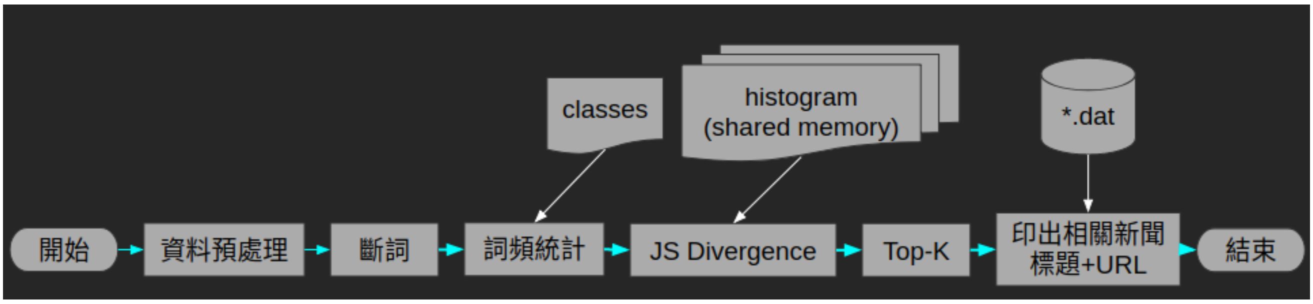
先把 ettoday 資料放在 makefile 同目錄下 data 資料夾 wiki dump 放在 wiki_data 資料夾

```
make # 編譯
make data_cleaning # 對 ettoday, wiki dump資料清理
make run_preprocessing # 訓練詞向量，得到 100 類相似詞
make compile_kmeans # 編譯 kmeans 程式
./word_count # 得到每篇新聞內文詞頻
./feature_extracter # 產生 kmeans 訓練資料
# 分成 13 群，容忍值 1e-4，最大迭代次數 200 次
# 用不同初始值跑 3 次，選較好的 centroids
../kmeans/kmeans ../.db/word_count/features.bin 13 1e-4 200 3
```

詞類別會被存放在 `./.db/word2vec/classes.txt` 文章分群類別會被存放在 `./labels.csv` 全部文章的 histogram 存放在 `./.db/word_count/features.bin`



把專案目錄軟連結到 Apache 能識別到的地方， 之後執行 `./loadData2Shm` 將文章 histogram 的 binary 載入 shared memory，方便快速查詢 再使用瀏覽器開啟 `index.php`，輸入文章查詢類似文章。



資料

使用 ettoday0~ettoday5 並且以：

，。；：「」等標點符號斷句。

實作細節

專案目錄結構

```
FINAL_PJ
├── closeShm.c
├── compute_copy_range.cpp
├── data_cleaning.sh
├── feature_extracter.cpp
├── find_news_by_id.cpp
├── flat_hash_map.hpp
├── gen_corpus.cpp
├── gen_wiki_corpus.cpp
├── index.php
├── jieba_word_count.hpp
├── kmeans
│   ├── kmeans.c
│   ├── kmeans.h
│   ├── main.c
│   └── makefile
├── loadData2Shm.c
```

```
├── makefile
├── ngram_word_count.hpp
├── parser.c
├── put_data_here
├── read_histogram.hpp
├── record_structure.h
├── record_structure_io.c
├── retrive_topN.cpp
├── Trie.hpp.patch
├── word_count.cpp
├── wstringcvt.hpp
├── 資料工程 Final Project.md
└── 資料工程 Final Project.pdf

1 directory, 28 files
```

實作細節

除了 word2vec、斷詞以外，其他都自己使用 C/C++ 實作， 儘量使用 OpenMP 平行化。 網頁部份使用 Apache2.0 + PHP7.0， 呼叫編譯好的 C++ 程式做新聞預處理、比較、查詢。

Word2Vec

使用 Google 提供的 C 版本 word2vec，800 多行的程式碼，很簡潔很強大。
https://github.com/tmikolov/word2vec

斷詞

使用結巴 (jieba) 斷詞，這裡使用 runtime 較高效的 `cppjieba` https://github.com/yanyiwu/cppjieba

histogram 差異計算

使用 Jensen-Shannon divergence 的平方計算兩 histogram 分佈差異 若要對文章長度變化不敏感，只在乎詞頻的相對關係，可以選擇對詞頻 histogram normalize。

```
double hist_intersection(const float *P, const float *Q, unsigned int cols) {
    double P_M = 0.0;
    double Q_M = 0.0;
    double JSD = 0.0;
    for (unsigned int i=0; i<cols; ++i) {
        float M_i = (P[i]+Q[i]) / 2.0 + 1e-6;
        P_M += P[i]*log((P[i]+1e-6) / M_i);
        Q_M += Q[i]*log((Q[i]+1e-6) / M_i);
    }
    JSD = (P_M+Q_M) / 2.0;
    return JSD*JSD;
}

double hist_intersection_normalized(const float *P, const float *Q, unsigned int cols)
{
    double P_M = 0.0;
    double Q_M = 0.0;
```

```
    double JSD = 0.0;
    double P_S = 1e-6;
    double Q_S = 1e-6;
    for (unsigned int i=0; i<cols; ++i) {
        P_S += P[i];
        Q_S += Q[i];
    }
    for (unsigned int i=0; i<cols; ++i) {
        float p=P[i]/P_S, q=Q[i]/Q_S;
        float M_i = (p+q) / 2.0 + 1e-6;
        P_M += p*log((p+1e-6) / M_i);
        Q_M += q*log((q+1e-6) / M_i);
    }
    JSD = (P_M+Q_M) / 2.0;
    return JSD*JSD;
}
```

K-means 程式碼節錄

1. 隨機選取 K 個初始 centroids
2. 分配每個資料點到最近的 centroid，形成 k 個 cluster（很容易平行化）
3. 計算每個 cluster 內資料點平均值，得到新的 centroids
4. 重複 2, 3 直到收斂，或達到指定迭代次數

複雜度為 O(tNDK), N 為資料筆數, D 為資料維度, K 為 cluster 數, t 為迭代次數。實際上有可能跑到不好的局部最佳解，所以通常會選擇不同的初始值多跑幾次。

```
double kmeans_intersec_int(unsigned int **data, unsigned int **return_labels, double
***return_centroid, int rows, int cols, int K, double tol, int max_iter, char verbose)
{
    double mean_centroid_d = DBL_MAX;
    double **centroids = NULL;
    double **new_centroids = NULL;
    unsigned int *lab_counts=NULL;
    unsigned int iter_counter=0;
    int *labels=NULL;
    centroids = (double**)malloc(sizeof(double*)*K);
    new_centroids = (double**)malloc(sizeof(double*)*K);
    labels = (int*)malloc(sizeof(int)*rows);
    lab_counts = (int*)malloc(sizeof(int)*K);

    for (int i=0; i<K; ++i) centroids[i] = (double*)malloc(sizeof(double)*cols);
    for (int i=0; i<K; ++i) new_centroids[i] = (double*)malloc(sizeof(double)*cols);

    // initialize (randomly pick k samples wo replacement)
    for (int i=0; i<K; ++i) {
        int l=rand()%rows;
        // check repetition
        char fail;
        do {
            fail=0;
            for (int j=0; j<i; ++j)
```

```
                if (lab_counts[j]==l) {
                    fail=1;
                    l=rand()%rows; // pick another
                    break;
                }
        } while(fail);
        lab_counts[i] = l;
        for (int j=0; j<cols; ++j) {
            centroids[i][j] = (double)data[l][j];
        }
    }

    iter_counter=0;
    while(iter_counter<max_iter && mean_centroid_d>tol) {
        // determine labels
        #pragma omp parallel for shared(data, centroids, labels) schedule(static,1)
        for (int i=0; i<rows; ++i) {
            unsigned int best_l=0;
            double min_d=DBL_MAX;
            for (int k=0; k<K; ++k) {
                double d = hist_intersection(data[i], centroids[k], cols);
                if (d<min_d) {
                    min_d = d;
                    best_l = k;
                }
            }
            labels[i] = best_l;
        }
        // determine new centroids
        for (int k=0; k<K; ++k) memset(new_centroids[k], 0x00, sizeof(double)*cols);
        memset(lab_counts, 0x00, sizeof(int)*K);
        for (int i=0; i<rows; ++i) {
            unsigned int l = labels[i];
            ++lab_counts[l];
            for (int j=0; j<cols; ++j) {
                new_centroids[l][j] += data[i][j]; // sum
            }
        }
        for (int k=0; k<K; ++k) {
            for (int j=0; j<cols; ++j) new_centroids[k][j] /= (double)
(lab_counts[k]+1e-8); // mean
        }
        mean_centroid_d = 0;
        for (int k=0; k<K; ++k) {
            mean_centroid_d += hist_intersection_f(centroids[k], new_centroids[k],
cols);
        }
        mean_centroid_d /= (double)K;
        // assign new centroids to centroids
        for (int k=0; k<K; ++k) {
            double *ptr = centroids[k];
            centroids[k] = new_centroids[k];
            new_centroids[k] = ptr;
```

```
        }
        ++iter_counter;
        if (verbose) fprintf(stderr, "[%d/%d] d:%.4lf\n", iter_counter, max_iter,
mean_centroid_d);
    }

    double *intra_distance = NULL;
    intra_distance = (double*)malloc(sizeof(double)*K);
    memset(intra_distance, 0x00, sizeof(double)*K);

    for (int i=0; i<rows; ++i) {
        unsigned int k = labels[i];
        intra_distance[k] += hist_intersection(data[i], centroids[k], cols);
    }

    double mean_intra_distance = 0.0;
    double max_intra_distance = 0.0;
    for (int k=0; k<K; ++k) {
        if (intra_distance[k]>max_intra_distance) max_intra_distance =
intra_distance[k];
        mean_intra_distance += intra_distance[k];
        intra_distance[k] /= (double)(lab_counts[k]+1e-8);
    }
    mean_intra_distance /= (double)rows;
    if (verbose) {
        fprintf(stderr, "mean data-centroid distance:\n");
        for (int k=0; k<K; ++k) {
            fprintf(stderr, "%3d: %.4f\n", k, intra_distance[k]);
        }
        fprintf(stderr, "average: %.4f\n", mean_intra_distance);
        fprintf(stderr, "maximum: %.4f\n", max_intra_distance);
        fprintf(stderr, "each class count:\n");
        for (int k=0; k<K; ++k) {
            fprintf(stderr, "%3d: %10d\n", k, lab_counts[k]);
        }
    }
    free(intra_distance); intra_distance=NULL;
    if (verbose) {
        fprintf(stderr, "centroid-centroid distance:\n");
        fprintf(stderr, "            ");
        for (int i=0; i<K; ++i) fprintf(stderr, "%12d", i);
        fprintf(stderr, "\n");
        for (int i=0; i<K-1; ++i) {
            // print upper traingle matrix
            fprintf(stderr, "%12d", i);
            for (int j=0; j<=i; ++j)  fprintf(stderr, "            ");
            for (int j=i+1; j<K; ++j) fprintf(stderr, "%12.3f",
hist_intersection_f(centroids[i], centroids[j], cols));
            fprintf(stderr, "\n");
        }
        fprintf(stderr, "\n");
    }
```

```
    for (int k=0; k<K; ++k) free(new_centroids[k]);

    free(lab_counts);
    free(new_centroids);

    if(return_centroid!=NULL) *return_centroid = centroids;
    else free(centroids);
    if(return_labels!=NULL)   *return_labels   = labels;
    else free(labels);

    return mean_intra_distance;
}
```

Top K

1. 假設資料為 A[N，要找到 top K 個最小元素
2. 初始化容量為 K 的 max-heap H
3. push A[0...K-1] -> H
4. 依序檢查 A[K...N-1] 中的資料 a_i , i: K -> N-1
5. 如果 a_i < H.top, H.pop(), H.push(a_i)
6. 最後 H 中會留下 top K 個最小元素

複雜度大約 O(N log K)

```
for (int i=0; i<topN; ++i) max_heap.push({distances[i], i});
for (unsigned int i=topN; i<n_rows; ++i) {
    if (distances[i]<max_heap.top().first) {
        max_heap.pop();
        max_heap.push({distances[i], i});
    }
}
topN = max_heap.size();
for (int i=topN-1; i>=0; --i) {
    topN_id[i] = max_heap.top().second;
    max_heap.pop();
}
```

Copy-append model

有 A, B 兩個檔案，我們要找到一連串操作 δ，使得： A + δ -> B δ 允許的操作有：從A複製貼上、加上新內容

使用 Copy-append model 去偵測兩篇文章整段相同的部份 虛擬碼：

```
k=4 # or 5 or 6, 7, 8, ...
# S 為 A 的 k-gram index, N 為 B 的 string length
i=0
while i<N:
    b = B[i:min(i+K, N)] # B 的 K-gram
    if b not in S: # binary search / hash table
        put(append,b)
        i+=b.length
    else:
        (l,m) = longest_match(A,i,S,B)
        put(copy,l,m)
        i+=l
```

C++ 實現，longest match 的部份主要使用 Z function 還沒有與 K-gram index + Boyer Moore 的方法比較速度 但實際跑起來蠻快的

```
inline std::pair<int,int> find_longest_match(const unsigned char *fileA, const
std::vector<int> &shortcut_index, const unsigned char *fileB, int A_len, int B_len, int
*z) {
    using std::min;
    using std::max;
    fileA += shortcut_index[0];
    A_len -= shortcut_index[0];
    memset(z, 0x00, sizeof(int)*(A_len+B_len));
#define s(i) (i<B_len?fileB[i]:fileA[i-B_len])
    // From Eddy's codebook:
    int l=0, r=0;
    z[0]=A_len+B_len;
    for (int i=1; i<A_len+B_len; ++i) {
        int j = max(min(z[i-l],r-i),0);
        while(i+j<A_len+B_len&&s(i+j)==s(j)) ++j;
        z[i] = j;
        if (i+z[i]>r) r=i+z[i], l=i;
    }
#undef s
    int M=B_len;
    int L=z[M];
    for (auto v : shortcut_index) {
        int m = v+B_len-shortcut_index[0];
        int l = z[m];
        if (l>L) {
            L = l;
            M = m;
        }
    }
    if (L>B_len) L = B_len; // Although this happen, the value of M is correct.
    M = M-B_len+shortcut_index[0];
    return {M,L};
}

std::string copy_append_encoder(const unsigned char *fileA, const unsigned char *fileB,
int A_len, int B_len, unsigned int K_size) {
```

```
    using namespace std;
    int B_index=0;
    ska::flat_hash_map<string, vector<int> > kgramA;
    string delta;
    int *z_buffer = new int[A_len+B_len+5];

    /* Generate Kgram index for fileA */
    for (int i=0; i<A_len; ++i) {
        int l = min((int)(i+K_size), A_len) - i;
        string ss((char*)(fileA+i), l);
        if(kgramA.count(ss)==0) kgramA.insert({ss, vector<int>(1, i)});
        else kgramA[ss].push_back(i);
    }

    string lastKgramB;
    int last_msg_length = 0;

    while (B_index<B_len) {
        int l = min((int)(B_index+K_size), B_len) - B_index;
        int m;
        string kgramB((char*)(fileB+B_index), l);
        if (kgramA.count(kgramB)==0) {
            string append_msg;
            if (last_msg_length>0) delta.resize(delta.length()-last_msg_length);
            lastKgramB += kgramB;
            append_msg += "a " + to_string(lastKgramB.length());
            append_msg += "\n" + lastKgramB;
            delta += append_msg;
            last_msg_length = append_msg.length();
        } else {
            /* Find longest match between A and B[B_index:] */
            pair<int,int> ML = find_longest_match(fileA, kgramA[kgramB], fileB+B_index,
A_len, B_len-B_index, z_buffer);
            m = ML.first;
            l = ML.second;
            string copy_msg;
            copy_msg += "c " + to_string(m);
            copy_msg += ","  + to_string(l);
            copy_msg += "\n";
            delta += copy_msg;
            lastKgramB.clear();
            last_msg_length = 0;
        }
        B_index += l;
    }
    lastKgramB.clear();
    if (z_buffer!=NULL) delete[] z_buffer; z_buffer=NULL;
    return delta;
}
```

實作上將連續的 append 操作變成同一個操作，避免浪費字元數。

考慮到使用者可能比較在意文章「出自」那一些段落，所以在伺服器上改為只偵測複製的操作，並找出複製的 unique 區段。

```
std::vector< std::pair<int,int> > copy_detect(const unsigned char *fileA, const
unsigned char *fileB, int A_len, int B_len, unsigned int K_size) {
    using namespace std;
    int B_index=0;
    ska::flat_hash_map<string, vector<int> > kgramA;
    vector< pair<int,int> > ret;
    int *z_buffer = new int[A_len+B_len+5];

    /* Generate Kgram index for fileA */
    for (int i=0; i<A_len; ++i) {
        int l = min((int)(i+K_size), A_len) - i;
        string ss((char*)(fileA+i), l);
        if(kgramA.count(ss)==0) kgramA.insert({ss, vector<int>(1, i)});
        else kgramA[ss].push_back(i);
    }

    while (B_index<B_len) {
        int l = min((int)(B_index+K_size), B_len) - B_index;
        int m;
        string kgramB((char*)(fileB+B_index), l);
        if (kgramA.count(kgramB)>0) {
            /* Find longest match between A and B[B_index:] */
            pair<int,int> ML = find_longest_match(fileA, kgramA[kgramB], fileB+B_index,
A_len, B_len-B_index, z_buffer);
            m = ML.first;
            l = ML.second;
            ret.push_back({m,m+l}); // [, )
        }
        B_index += l;
    }
    if (z_buffer!=NULL) delete[] z_buffer; z_buffer=NULL;
    return ret;
}

inline std::vector< std::pair<int,int> > summary_ranges(std::vector< std::pair<int,int>
> &ranges) {
    using namespace std;
    vector< pair<int,int> > ret;
    sort(ranges.begin(), ranges.end());
    ranges.push_back({INT_MAX,INT_MAX}); // 哨兵
    int l=ranges[0].first, r=ranges[0].second;
    for (int i=1; i<ranges.size(); ++i) {
        if (ranges[i].first<=r) r = max(r, ranges[i].second);
        else {
            ret.push_back({l,r});
            l = ranges[i].first;
            r = ranges[i].second;
        }
    }
    ranges.pop_back();
```

```
        return ret;
    }
```

實驗結果

實驗環境：

- Ubuntu 16.04 64-bit
- 32GB DDR4 (2400MHz)
- AMD Ryzen 5 1600 (6C12T)
- 5400rpm HDD

分群效果

資料距離群心的平均距離 (JSD^2)

	data-centroid	count
0	3152.1657	57238
1	1422.4215	42418
2	1422.4490	22251
3	111797.1769	157
4	1189.1763	85518
5	2667.7802	53586
6	9599.0289	2698
7	12069.3628	5496
8	4196.4556	9458
9	99240.8710	107
10	13561.2694	1829
11	10511.0250	5945
12	2159.9833	243577

群心間距離

centroid-centroid distance:

	0	1	2	3	4	5	6	7	8	9	10	11	12
0		1726.177	2635.470	574997.263	2847.557	779.596	9773.077	29298.529	13485.311	994810.870	15664.413	13011.640	885.770
1			230.995	714038.255	4045.732	1718.443	26484.825	36188.242	22616.659	1063529.988	31050.319	20550.180	951.453
2				619361.984	7325.204	1875.415	20703.235	25512.895	25759.213	927342.939	25631.757	16833.333	1290.238
3					867918.455	514452.040	171303.461	134522.862	583342.601	102481.399	166575.413	294752.677	659091.197
4						5195.936	43592.105	79396.880	4924.774	1358128.557	55902.481	44777.198	4659.396
5							10833.258	16775.143	18346.265	873512.894	21049.818	13267.452	491.653
6								7570.148	32179.436	478678.231	7548.072	13524.838	21390.045
7									81353.877	204036.603	10266.032	4717.326	22604.782
8										1129539.647	53287.469	60263.092	25981.084
9											459350.232	422082.587	950550.873
10												9483.434	24666.488
11													10903.596

觀察

肉眼能觀察出來的大概有以下幾類：

財經新聞 (0) 體育新聞 (1,2) 長文 (3) 股市相關公告-短文 (4) 股市相關公告-長文 (8) 政治新聞 (5) 甄嬛傳系列文章/原創長文 (9) 3C新聞 (10) *(num) 代表某個 cluster

有些 cluster 特別容易混淆，例如 財經新聞(0) 與 政治新聞 (5) 有些離群值，其群心與其他類別隔非常遠，肉眼閱讀文章也能發現用詞有極大差異，例如 (9) 而有些 cluster 應該處於同一類，例如 (1) (2) 同樣是體育新聞，可能是因為 cluster 數量設太大。

文章查詢效果

記者洪聖壹／西班牙巴塞隆納報導

BlackBerry 今天（2/25）上午在 MWC 2014 會場當中宣布與鴻海集團結盟，確認將於今年四月份，在印尼推出首款合作新機 BlackBerry Z3，未來將由鴻海集團負責硬體生產，黑莓則會把資源放在軟件研發，雙方還將推出代號為「Classic」的 Q20 黑莓機。

BlackBerry 執行長程守宗與鴻海董事長郭台銘兩人今天共同宣布合作成果，並發表了 BlackBerry Z3，配置了全觸控螢機身，搭載 BlackBerry 10.2.1作業系統，但是空機售價如同今年一月份傳出的一樣，將以不到 200 美元的價格，於四月份先在印尼推出。

顯示前幾筆： 50 比較文章長度 Go!

標題	差異度	重複的內文
MWC2014／黑莓、鴻海聯手推合作新機BlackBerryZ3	163.92	[1] 記者洪聖壹／西班牙巴塞隆納報導 [2] 2/25）上午在 [3] 會場當中 [4] 集團結盟，確認將於今年四月份，在印尼推出首款合作新機 [5] Z3，未來將由鴻海集團負責硬體生產，黑莓則會把資源放在軟件研發，雙方還將推出代號為「Classic」的 [6] 行長程守宗與鴻海董事長 [7] 今天共同宣 [8] 果，並發表了 [9] Z3，配置了全觸控螢機身，搭載 [10] 10.2.1作業系統，但是空機售價如同今年一月份傳出的一樣，將以不到 [11] 的價格，於四月份先在印尼推出
LG第二代彎曲手機GFlex2傳將於2015年初亮相	185.72	[1] 記者洪聖壹／
蝴蝶飛向歐洲！HTC將於IFA展出8核手機與二代蝴蝶機	189.36	[1] 記者洪聖壹／
快訊／Google宣佈推出搭載原生Android4.2的GALAXYS4	204.39	[1] 記者洪聖壹／
前臺灣三星總經理杜偉昱將出任德誼數位轉戰蘋果通路	221.48	[1] 記者洪聖壹／
終於要來了！HTC自拍神器re將於12月2日登場	233.50	[1] 記者洪聖壹／

如上圖，可以輸入文章查詢類似文章，輸出前 N 筆最像的文章標題與 URL，並顯示它與欲查詢文章的 JSD^2 作為差異度。最後列出與查詢的內文有整段重複的部份。

在我的電腦上大約 1~2 秒一個查詢 在系上工作站大約 3~6 秒一筆查詢

GitHub

程式碼： https://github.com/peter0749/Data-Engineering/tree/master/FINAL_PJ

https://github.com/peter0749/Data-Engineering/tree/master/copy_append_model