CSCI 572 Homework1

Team Member:

Chenyin Sang, 2812623778

Dongni Zhao, 3282822727

Hongzhi Li, 1322106452

Wenhai Zhu, 9262786977

1. Dataset

About the subset data, there are 430 tsv files, and each is about 10MB. The data in these files is split into lines. Each line is the information about one job. Each piece of information is split into different contents by tab. There are 20 different columns in each line. These are postedDate, location, department, title, empty(this one is added by us accidently to avoid a useless tab), salary, start, duration, jobtype, applications, company, contactPerson, phoneNumber, faxNumber, location, latitude, longitude, firstSeenDate, url, lastSeenDate.

After inspecting the data, we found that, for each line, there is always an empty column. So in order that data and the columns are the same, we named this column "empty".

Before we began to write code, we also looked at these files. From the file names, we can see that each file's content might be crawled on a certain day. Every day the crawler crawls the website and stores it into a single file. Each file represents the employment information that exists on the website on that day. As most job information won't change, say, in a week or even a month, there must be many duplicated jobs. To detect two duplicate jobs is not difficult. Each line is short and we know the meaning of each piece of information, like which is company name, which is job type. If two jobs are duplicate, some columns must be the exactly the same and some others should be nearly similar. The big problem is that there are 430 tsv files and total data is about 4G. This is huge and we need some fast algorithm to decide duplication. The detailed algorithm is in part 6.

2. TSVParser

Develop a TSVParser to take in a tsv file and to create structured XHTML output. In the parser, create the function parser (InputStream, ContentHandler, Metadata, ParseContext). In this function, first get the charset, content type and content encoding, then store these information in metadata. Secondly, call startdocument() function to write the head of xhtml file. Then, start to write the table tags, write the column headers in first row as well. Since these tsv files are separated by tab, read every line of the tsv file, split the line by tab. Finally, process the special characters in every record, like &, ", ', >, <, then using element, startelement and endelement function to write the table of every tsv. In the main function, just call the TSVParser to parse the tsv file and use java I/O to write the XHTML files.

In the tsv file, the value of the fifth column is empty, so the head of this column named "empty". Also there are two fields name "location", so the second one changed to "place". Since there are some languages other than English, like Spanish, Estonian, these characters

need to be formatted. Also structure xhtml is very stricter, according to observation, some special characters belong to undefined entities.

3. JSONTableContentHandler

Develop a JSONTableContentHandler to take over the XHTML file and using ToJSONContentHandler to transform XHTML to JSON. The main method here is to extract value from "<tr>" and "</tr>". Such as: <tr><td>....</td></tr>, handler sets separators between <tr> and </tr>. And then, handler sets separator between each <td> and </td> to retrieve each value of elements.

The process is below:

(1) retrieve the data from the dataset by separators

(2) add tag value in front

(3) add punctuations to transform into JSON format

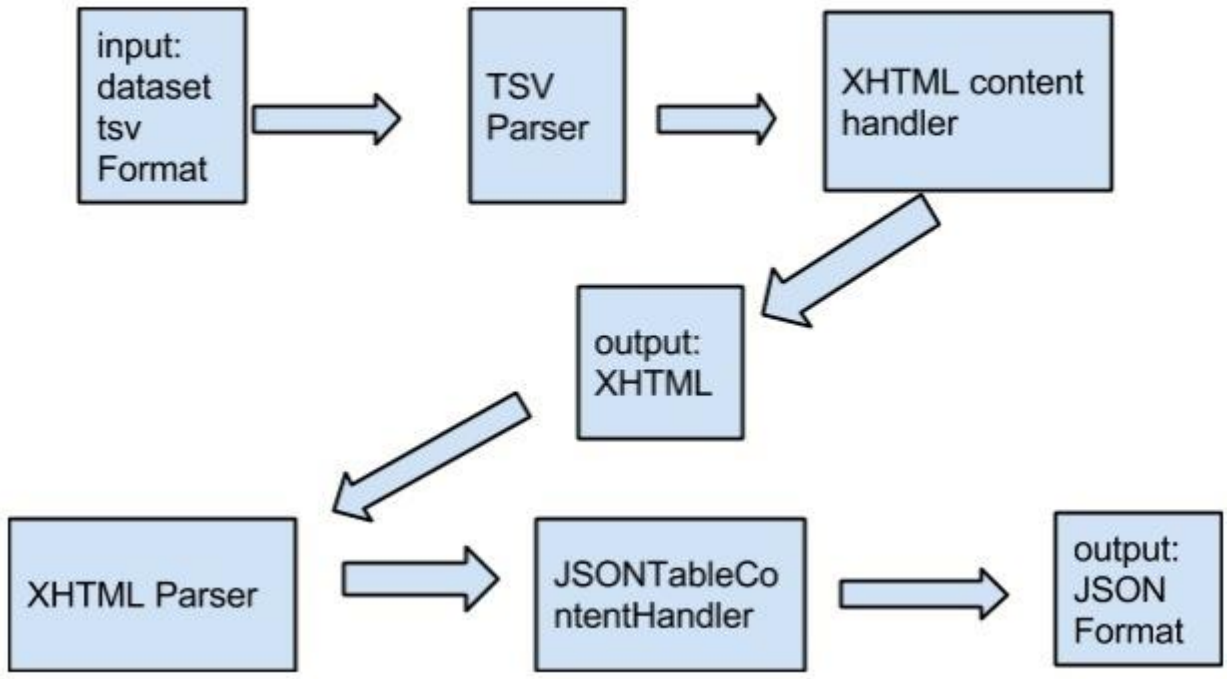
(4) write JSON format string in an individual file

4. Simple Crawler

Combine first one and second one. Since the output type of TSV parser is string, convert the output to inputstream, saving time to write XHTML files on the disk, call the XHTMLParser and JSONTableContentHandler to generate the json file. However json has some special character rules, we improved the function to write the json file. One more backslash need to be added before backslash. Finally, use Java I/O to write the json files on the disk.

There are **12483611** json files.

The procedure is as below:



5. ETLLIB

Etllib is a very convenient and useful tools to convert tsv files into json files. It has two useful commands/functions: tsvtojson and repackage.

Tsvtojson is used to convert one tsv file into one json file. This json file will contains multiple job information. Then we need repackage to split the one json file into different json files, each contains only one job.

We write a program to convert each tsv files into json files. We got **12483611** json files in total. The problem when converting is it needs time and spaces. It is about 20 – 30s to process one tsv file. And it will generate about 24000 json files. With the increasing of number of json files, the process will be slower. We were trying to store all json files into one directory, but when it processed about 200 tsv files, the speed is too slow. So we split the 430 tsv files into 29 directories and process the converting separately.

I think there are some aspects that could be applied to improve etllib. The first one is it would be more helpful if there is examples about how to use the commands like tsvtojson and repackage, so we don't need to try different combinations of the parameter. Second, it is better to give the detailed error information when we wrongly use the command. For example, in the tsvtojson commands, there is a parameter "-j", which is the file name of the generated json file. If the file exists, this command won't work, but it only gives the usage of whole command. It would be better if it prints "file xxx.json exits, please delete it first". We will know why it is wrong. Third, about the repackage command, it would be better if it has a parameter which could specify the directory to store the generated json files. Otherwise thousands files will be generated in the same directory of the code.

6. Deduplicate

(1) how do we distinguish a job as duplicate?

A typical job has multiple features to describe it, but not all of them are helpful to decide a duplicate job. So we selected 6 features (location, title, department, title, company, jobtype) to filter the same jobs. While comparing two jobs, to be consider as the same, some of the six features need to be exact the same whereas the others could be just similar. After discussion, we agreed on that two duplicate jobs must have the same location ,title, department, company and job type. In terms of the feature 'application', only if it reaches the similarity threshold, we regard it as matched.

(2)Algorithm to get rid of duplicate job

Consider that we are dealing with millions of jobs, the efficiency of the algorithm is the most critical issue has to be tackled. We want the deduplication process can be done in several minutes rather than hours or days, so the algorithm should be O(n) time complexity which means the indexing is O(1) time complexity. After comparing some approaches, we think simhash algorithm works well in similarity comparision. Therefore, we combined those five job features that have be exactly matched and generate fingerprint from it. Then we build hash table base on this fingerprint. Once two jobs have the same fingerprint, we move forward to compare the features that can be approximate matched. As long as similarity reaches the threshold, we say these two jobs are duplicate.

Algorithm:

For each job record:

exactString = combine features that have to be exact the same

similarString = extract features that can be almost the same

generate fingerprint base on exactString

if fingerprint exist in hash table:

    generate fingerprint base on similarString

    if similarity of fignerPrint > Threshold:

        detect duplicate job

    else:

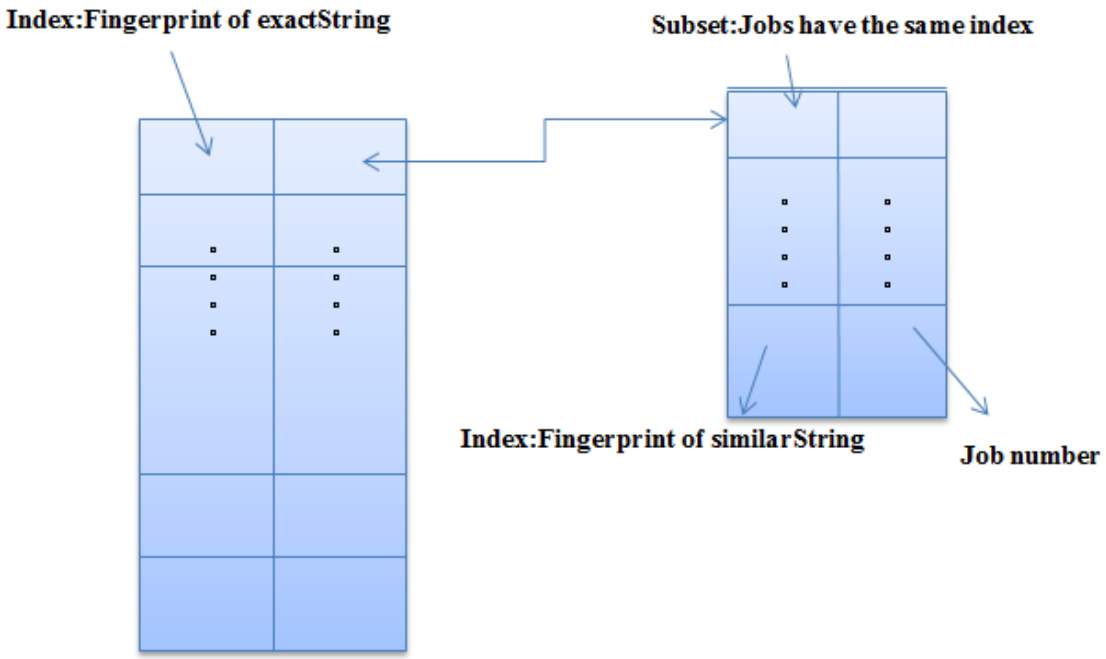
        add job to this subset

else:

    Add one more index and add job to its subset

End for loop

We applied **0.6** for the threshold of similarity because the strings extracted from the features are quite short and difference in just one word can decrease the similarity by **40%**. For example, two string "Aquellos interesados enviar su CV axxxx@xxxxxxxx.xxx" and "Aquellos interesados enviar su CV axxxx@xxxxxxxx.xxx. indicando en el asunto" are almost the same, but their similarity is only 0.64. Thus, after tens of test, we set the threshold to 0.6. Below is the data structure of our algorithm:



Data Structure

We referenced some resources on the website:

SimHash introduction: http://matpalm.com/resemblance/simhash/

SimHash implementation: https://github.com/sangelone/python-hashes

SimHash introduction and implementation: http://gemantic.iteye.com/blog/1701101

(3)Result

We implemented the algorithm both in Java and Python. For Python code, the unique jobs are **261480**. For Java code the unique jobs are **261711**.

Compared to the original dataset which has **12483611** job records, approximately 98% of jobs are duplicate. And the reason to this is obvious, the job information is captured every day and most of the jobs remain there for months. Moreover, the recruiters are likely to post their jobs in multiple job boards. As we can tell from the duplicate job files, most of the information are exact the same except the website url or last seen day.

(4)Our improvement

At first we separate the procedure of converting tsv to json and the procedure of deduplication, which means after we get all json files and then use deduplicate algorithm to read all json files and then deduplicate. This took us a long time to finish the whole test, which about 3 hours.

Later, we found that we can deduplicate at the time of converting tsv to json. We don't need to read all json files again from disk. This will save us lots of time. With this methods, we use 40-50mins.

7. Labor division and further improvement

Dongni Zhao is responsible for the implementation and test of JSONTableContentHandler.

Chenyin Sang is responsible for the implementation and test of TSVParser.

Dongni Zhao and Chenyin Sang worked together to implement the crawler.

Wenhai Zhu is responsible for the implementation and test of ETLlib code and deduplication in Java.

Hongzhi Li is responsible for the implementation and test of deduplication in Python, and dataset test in Java and Python.

There are a few aspects that we can improve our algorithms:

(1) We can use multiple threads to make it run faster.

(2) We can build index on the fingerprint.