Content extraction and search using Apache Tika – Employment Postings Dataset contributed via DARPA XDATA

Renu Kankamedala
Aldrin Rodrigues
Neha Ahuja
Poushali Banerjee

CSCI 572 HW 1
Prof. Chris Mattman

October 6, 2014
University of Southern California

Table of Contents:

- Introduction
- Methodology
- Results

other TODO's

1. Change hard coded file paths in DeduplicationJSON. (Aldrin)
2. Create a readme.txt with any instructions for grader, any external libraries (Renu)
3. Deduplication - used Jaro Winkler Similarity measure for Location2 column - java code for that function is taken from code.google.com

----

Introduction

In this project we create a local search engine of the Employment postings by cleansing, transforming, and developing an algorithm for ranking the job postings from http://www.computrabajo.com affiliate sites that primarily serve Mexico and South American countries. We have worked with a subset of the data for the most except where mentioned.

Methodology

The data is given in tsv format which is messy and difficult to understand, so first we transform the TSV files into Java Script Object Notation (JSON) files, and then from there into individual JSON posting files per job.

The TSV Parser:

In the parser, we reach each line and extract the column values. We then use the WriteoutContentHandler(-1) to get rid of 1000000 limit for writing characters
and xhtml content handler for creating xhtml tags. To start a xhtml document we used xhtml.startDocument() which by default creates the namespace, html and body tags. Then we create a table using xhtmlcontenthandler's startElement and endElement functions to create different table tags. We used .characters() function to print the values and the writer function to write the xhtml output to a file.

TSV to Json:

Once we have the xhtml file we used html cleaner to parse through the html tags and retrieve the values. Similar to XHTMLContentHandler we created JsonContentHandler and overrided the functions like startDocument, endDoument, startEle, endEle accordingly and added few new methods like startArray, startJsonAttribute for different elements in Json. Thus we created JSON files corresponding to the XHTML table rows (one file per row).

Crawler:

After we had all the components for translating tsv files into JSON, we developed a program which fetches all the TSV files from the respective folder and converts to xhtml using TSV parser and the xhtml content is passed to JSon content handler to create json files.

ETLLIB Crawler:

This is an API developed by Chris Mattmana and team to munge and prepare JSON, TSV and other data using Apache Tika and JSON parsing/loading for ETL via Apache OODT (or other libs) into Apache Solr. This was already provided to us and we installed the library without tika and ran the necessary commands for translating tsv to json.

We also wrote some code in python that would iterate through JSON files and count the number of files produced. Details about how we used the etllib and other code to achive our results are explained in readme.txt (attached).

Deduplication:

With respect to web crawling, de-duplication refers to the identification of identical and nearly identical web pages and indexing only a single version to return as a search result. In this part we attempted to remove any duplicate file/records from our JSOn results. For this we followed the following methodology:

First, by analyzing the dataset we decide columns like Company, Department, Location2, Url, Title deduce uniqueness to a record. The same company can have a requirement in other department They can have requirement for same company and department in different locations and we used Jaro Winkler string similarity measure to compare two locations. Jaro winkler is good when the two fields have their prefix matching. Ex: San Francisco and San Francisco City. We used Jaro Winkler Similarity measure for Location2 column - java code for that function is taken from code.google.comTitle, the Name of the position and Url - If the url's match we need not check for match between any other columns.

Algorithm:

```
if company name matches{
        if url matches{
                return true;
        }else if dept name matches {
                        if title matches {
                                if location2 columns are not null {
                                        if (JaroWinkler similarity score >= 0.8) {
                                                return true;
                                        }
                                } else if location2 column is null in both records {
                                        return true;
                                }
                        }
                }
}
```

Results

- Count of Json Jobfiles from our TSVParser: 12,483,611
- Count of Json Jobfiles from ETLLIB software: 12483611
- Deduplication (explained above)

- Number of job files produced with deduplication enabled:
- Number of job files produced without deduplication when run on the entire data set (extra credit): 119423497

Analysis