CS 5604: Information Storage and Retrieval Spring, 2015
Computer Science Dept. of Virginia Tech
Blacksburg, VA
Instructor: Dr. Edward Fox
Project Team: NER

Named Entity Recognition for IDEAL

Project Report

By Qianzhou Du, Xuan Zhang

May 10, 2015

Abstract

The term “Named Entity”, which was first introduced by Grishman and Sundheim, is widely used in Natural Language Processing (NLP). The researchers were focusing on the information extraction task, that is extracting structured information of company activities and defense related activities from unstructured text, such as newspaper articles. The essential part of “Named Entity” is to recognize information elements, such as location, person, organization, time, date, money, percent expression, etc. To identify these entities from unstructured text, some researchers called this sub-task of information extraction as “Named Entity Recognition” (NER). Now, NER technology has become mature and there are good tools to implement this task, such as the Stanford Named Entity Recognizer (SNER), Illinois Named Entity Tagger (INET), Alias-i LingPipe (LIPI), and OpenCalasi (OCWS). Each of these has some advantages and is designed for some special data. In this term project, our final goal is to build a NER module for the IDEAL project based on a particular NER tool, such as SNER, to apply NER to the Twitter and web pages data sets. This project report presents our work towards this goal, including literature review, requirements, algorithm, development plan, system architecture, implementation, user manual, and development manual. Further, results are given with regard to multiple collections, along with discussion and plans for the future.

Key Words: Named Entity Recognition, Information Extraction, Information Retrieval, MapReduce, Hadoop

Table of Contents

1. Overview

What is Named Entity Recognition? In the area of Natural Language Processing (NLP), the process of identifying and categorizing words into different classes has become very important, and this process is defined as **Named Entity Recognition** (NER) [10], which was first introduced by Grishman and Sundheim [6]. To be strict, a named entity is a group of consecutive words existing in a sentence, and it can represent some concepts, such as person, location, organization, time, date, money, percent expression, etc. For example, in the sentence "Martin Schulz of Germany has been the president of the European Parliament since January 2012", "Martin Schulz", "Germany" and "European Parliament" are person, location, and organization entities, respectively. Here we should note that there is no real consensus on the various types of entities. NER tools also identify numeric entities such as amounts of money, distances, or percentages, and hybrid entities such as dates (e.g., "January 2012" in the previous sentence). Recognizing and extracting such data is a fundamental task and a core process of NLP, mainly for two reasons. First, NER is directly used in many applied research domains [10]. For example, proteins and genes can be considered as named entities, and many work in medicine focus on the analysis of scientific articles to find out hidden relationships between them, and drive experimental research [11]. Second, NER is used to do preprocessing for more advanced NLP problems, such as relationship or information extraction [12]. Consequently, a number of tools have been developed to perform NER, such as the Stanford Named Entity Recognizer (SNER) [4], Illinois Named Entity Tagger (INET) [3], Alias-i LingPipe (LIPI) [9], and OpenCalasi (OCWS) [10].

Problems and Challenges: In spite of the high F1 measure value reported on the MUC-7 dataset, the problem of Named Entity Recognition is still far from being solved. Now the main efforts are directed to reducing the annotation labor by employing semi-supervised learning [13], robust performance across domains [3] and scaling up to fine-grained entity types [14]. During past few years, many projects have focused on crowdsourcing area, which is a promising the solution to obtain high-quality aggregate human judgments for supervised and semi-supervised machine learning approaches to NER [15]. Another interesting task is to identify "important expressions" in text and cross-link them to Wikipedia [16], which can be seen as an instance of extremely fine-grained named entity recognition, where the types are the actual Wikipedia pages describing the (potentially ambiguous) concepts.

An important objective of this project is to support the IDEAL project [20] (Integrated Digital Event Archiving and Library). IDEAL builds a digital library for archiving, retrieving, and analyzing event datasets, such as disasters, wars, elections, etc. The datasets managed by IDEAL include huge Twitter datasets (about 1 TB) and Web page datasets (about 11 TB), etc. Named entities are important features in the searching (e.g. ranking the search results) and analyzing of IDEAL project. Therefore, applying NER to huge tweet and web page collections is important utility for the IDEAL project.

However, little research is reported in term of its performance on Twitter data in the application of NER. In addition, how to scale the NER utility to large datasets is another challenge.

Our goal: As we mentioned above, there are a lot of works have been done about NER. However, these works also have some disadvantages respectively, such as they have a good performance on some datasets, but perform badly on others. So we challenge and improve the previous works on other data sets, especially the Twitter data sets where the writing style is not formal and the writing is lack of consideration about the grammar and syntax. **In this term project**, our work contains three parts: 1. We implement a NER prototype based on existing tool; 2. We scale our model into MapReduce model on the Hadoop cluster; 3. We explore applying NER to large Twitter datasets and corresponding web page datasets, evaluate the performance of our model, and propose suggestions on how to improve the method.

The organization of this report is like the following: Section 2 reviews the existing study around NER; Section 3 discusses the requirements with the NER module for the IDEAL project; Section 4 proposes the overall design of the solution, including the approach and corresponding algorithm; Section 5 introduces the process of the implementation and the evaluation; Section 6 serves as the user manual of this project; as an important part, Section 7 presents the information needed by the possible continuous work on this subject; and the remaining sections conclude the project and show acknowledgement.

2. Literature Review

We did a literature review from several textbooks, tutorials, and papers.

Textbook: After going through the IR textbook, we found no chapters or sections explicitly introduce how to do NER. The term NER is mentioned only two times in Chapter 10 and Chapter 15. We also checked other two famous NLP text books, "Foundation of Statistical NLP" and "Speech and Language Processing". Unfortunately, neither of them explains NER technology in detail. However, chapters 2, 13, 14, and 15 of the IR book are somewhat related to our task. Chapter 2 discusses the preprocessing jobs of information retrieval, such as tokenization, removing stop-word, and normalization. These tasks are prerequisites of NER, which might be helpful to improve its accuracy. On the other hand, classification is one of the important technologies to implement NER. In the IR book, chapters 13, 14, and 15 are related to classification. Chapter 13 introduces feature selection, chapter 14 focuses on document modeling, and Chapter 15 presents SVM classification algorithms. We may benefit from those chapters when we try to understand how NER works, and modules of the IR system.

Papers and Tutorials: We also found a few good papers about NER. A survey of existing NER technologies is presented by Nadeau and Sekine [1]. Zhou and Su [2] proposed a HMM-based chunk tagger for NER. Ratinov and Roth [3] investigated the common challenges and misconceptions in NER, and proposed corresponding solutions. They created INET that relies on

several supervised learning methods: hidden Markov models (HMM), multilayered neural networking and other statistical methods. The technology used by the Stanford NER, which is based on linear chain conditional fields, is introduced in [4]. LIPI [7] use n-gram character language model and is trained through HMM and conditional random field methods. The last tool we know is OCWS [8] that is a Web Service for NER. LIPI and OCWS also provide NLP services and we can get free licences of them for academic use. Regarding the application of NER, course reports [18] [19] introduce the experience of applying NER to storm disaster news.

3. Requirements

As the ultimate goal of our team, a NER module for the IDEAL project will be developed which will fulfill the following functions and performance requirements. An investigation over existing NER tools (such as Stanford NER, INET, LIPI, OCWS, NLTK, etc.) is to be conducted to find the best named entities associated with each given document. The NER module was constructed based on the results of investigation.

- *Functional Requirements*
  - ➢ NER function

  Given a document (web page or tweet) indexed in Solr, the NER module extracts the dates, times, locations, person names, organization names, and for any other special type of entity.
  - ➢ Result Indexing

  The extracted named entities should be exported to Solr or the HBASE, associated with the source document, and indexed by Solr. Multiple fields such as date, location, person, and organization should be created for each document.

- *Performance Requirements*

  A good F1-measure (say 80%) for named entity recognition is the goal of the NER module.

- *Dependencies*
  - ➢ Solr & Lucene

  The NER module definitely relies on the document collections imported and indexed by Solr.
  - ➢ Hadoop

  The NER module may rely on the Hadoop module in terms of huge collection analysis.

4. Design

4.1 Approach and Tools

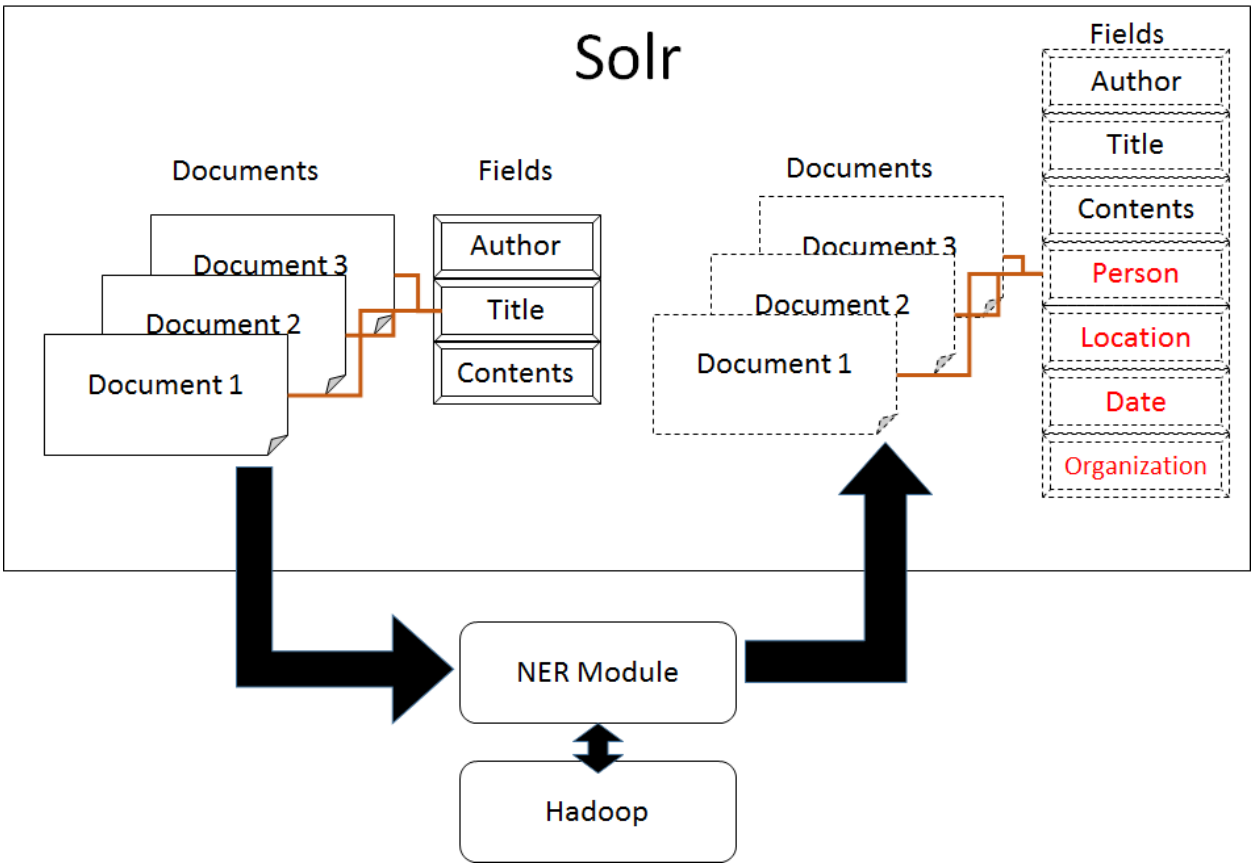

Fig.1 Architecture

The overall approach of this project is shown in Fig.1. A NER module is developed to get documents (tweets or web pages) from Solr, extract named entities (such as person, location, date, organization), and fill the entities into the new fields of these documents in Solr (or HBASE). The Hadoop platform is used to scale the NER utility when dealing with big collections.

For the NER module, by now we just use the baseline algorithm that is from Finkel's work [4]. This model can be treated as a Conditional Random Fields (CRFs) classifier and it provides a general implementation of linear chain CRFs sequence models. Here CRFs are a kind of statistical modelling method often applied in pattern recognition and machine learning, where they are used for structured prediction. In this hidden state sequence model CRFs, it uses the Viterbi algorithm, which is a dynamic programming algorithm, to infer the most likely hidden state sequence given the model and the input [17].

The Stanford NER tool is used for the implementation of the NER module. This tool is chosen due to its popularity and the high accuracy (F1>85% when applied to two famous datasets) claimed by the Stanford NLP group.

4.2 NER Algorithm

We have introduced what NER is in above statement. Now we will discuss how to implement it. First of all, let us consider this question: which kind of named-entity should the phrase of "New York" be assigned? If only based on this information, most of us will assign it "Location". However, if "Times" occurs after "New York", we should assign it "Organization". Moreover, if no "Times" after "New York" but there are more than 2 sports-related words occurring in the same sentence, it might be an indicator that "New York" is a sports team that is an "Organization". Thus, when we try to recognize a word, we also need to consider the context of this word. So a regular graphical model, such as Naive Bayes, is not enough. In our project, we will use the linear-chain CRF model, which can do NER according to the dependency on neighboring words, to implement our NER prototype.

4.2.1 Problem Definition

Now we will define this problem as follows:

Input: word_sequence = {X1, X2, X3, X4 … Xn} where 1,2,3,4 … n is the position of the words.

Output: tag_sequence = {Y1, Y2, Y3, Y4, … Yn} where item Yi is the tag assigned to Wi and tag might be "Location", "Person", "Organization", or "Date".

This problem of extracting entities from a word sequence of length n can be cast as a graphical model by introducing for each word, Xt, 1<= t <= n, a target variable Yt, which indicates the entity type of the word.

4.2.2 Introduction to CRF

To help us understand how linear-chain CRF works, we need to introduce the CRF first. CRF [22] is derived from Markov network. We can use the same undirected graph representation and parameterization to encode probability distribution. Compared to Markov network representation that encodes a joint distribution over X, the CRF **goal** is to get the conditional distribution P(Y|X), where Y is the a set of target variables (such as "Location", "Person", "Organization", "Date") and X is a set of observed variables (such as "New York", "Obama", etc.).

A conditional random field is an undirected graph H whose nodes correspond to XY; the network is annotated with a set of factors 1(D1) .....m(Dm) such as each DiX. The network encodes a conditional distribution as follows:

$$P(Y|X) = \frac{1}{Z(X)} P^{\sim}(Y,X)$$

$$P^{\sim}(Y,X) = \prod_{i=1}^{m} \phi_i(D_i)$$

$$Z(X) = \sum_{Y} P^{\sim}(Y,X)$$

Here P(Y,X) is the unnormalized measure and i(Di) is the factor which we can calculate by using log-linear model and Di is a scope we can get it from the training data. Each i(Di) can represent a dependency or a relation between some variables. Based on this, we can compute the conditional distribution over X.

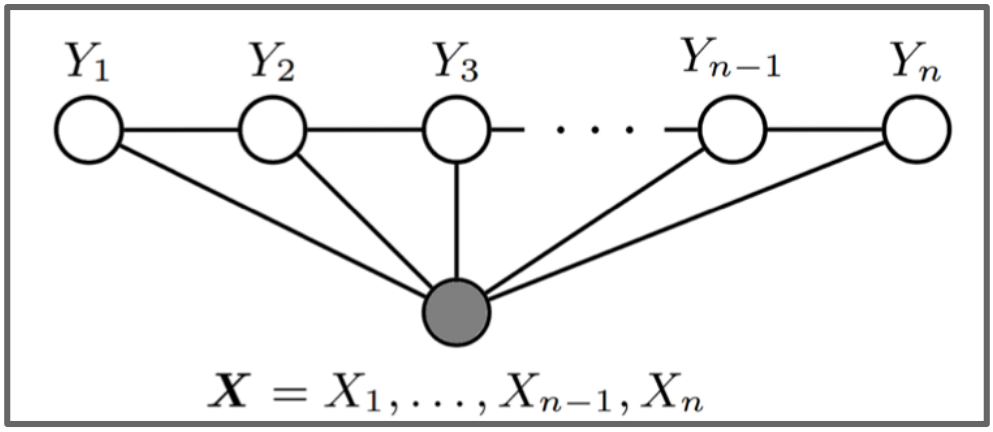

Fig.2 Linear-chain CRF Graph

4.2.3 Linear-chain CRF

Linear-chain CRF is a special and important type of CRF. As shown in Fig.2, it has two factors: one factor t1(Yt, Yt+1) to represents the dependency between neighboring target variables, and another factor t2(Yt,X1, X2, ... Xn)that represents the dependency between a target and its context in the word sequence. These two dependencies can support our recognition job very well. Because in this model, each state is a sequence, so here we need to use forward-backwards algorithm to calculate the probability.

4.2.4 How to choose the best tag sequence

The two former sections have introduced how to calculate the conditional distribution over X. So now we need to think about how to choose the best tag sequence. In our project, we will use a dynamic programming algorithm, Viterbi, to get the best sequence by maximizing the probability as follows.

$$
\begin{aligned}
y^* &= \arg\max_y P_w(y \mid x) \\
&= \arg\max_y \frac{\exp(wF(y,x))}{Z_w(x)} \\
&= \arg\max_y \exp(wF(y,x)) \\
&= \arg\max_y wF(y,x)
\end{aligned}
$$

4.3 Deliverables

The main programs and documents to be delivered are listed in the following lists. The detail of the deliverables are available in Section 7.4.

Programs:

- Java prototype of NER
- MapReduce program of NER
- AVRO converting programs

Document:

- Project report
- Presentation slides

5. Implementation

5.1 Plan

By 01/29: Finish the installation of Solr.
By 01/29: Submit Project 01 Report.
By 01/29: Import the sample CSV file and run some simple queries.
By 02/05: Import z_542_qb.csv data and sample web pages.
By 02/05: Submit Project 02 Report.
By 02/12: Implement a basic NER tool
By 02/12: Import the small collection and then run NER on the sample data.
By 02/12: Submit Project 03 Report
By 02/26: Use the script from Mohamed and get the tweets for our small collection and produce a directory structure that has a text file for each of the webpages that have a URL appearing in the set of tweets.

By 02/26: Design the schema for our output and document our input.
By 02/26: Document our basic architecture.
By 02/26: Submit Project 04 Report.
By 03/22: Document our NER algorithm.
By 03/22: Build a prototype and document it.
By 03/22: Run our prototype (Mapper) on the Hadoop cluster.
By 03/22: Submit Project 05 Report.
By 03/29: Load our small collection to the Hadoop cluster.
By 03/29: Run our prototype (Reducer) on the Hadoop cluster to extract named entities from original Small Tweet Collection.
By 03/29: Label the classes of the storm data set.
By 03/29: Submit Project 06 Report.
By 04/05: Working on the extracting seed URL from big collection.
By 04/05: Submit Project 07 Report.
By 04/14: Run our prototype on the Hadoop cluster to extract named entities from cleaned Big Tweet Collection and cleaned Small Tweet Collection.
By 04/20: We have extracted partial long URLs from big collection and also have crawled some web page data.
By 04/20: Submit the project09 report.
By 04/20: Collaborated with Hadoop team, we imported the NER result into the format that they required.
By 04/26: Submit the project10 report.
By 04/26: Finish all processing on the small collections.
By 05/06: Evaluate the performance of the NER module.
By 05/06: Write the final project.

5.2 Milestones and Deliverables

Shown in Table.1, there are 5 milestones in this project. As the first step, we implemented a java prototype based on Stanford NER. Next, the design of a MapReduce version of NER was done. The Mapper and Reducer were developed based on this design. And we updated the I/O interface of the Mapper and Reducer to support AVRO files. This report was completed in the end.

Table.1 Milestones and Deliverables

Date	Milestone	Deliverable
02/12	Implement a NER prototype based on Stanford NER	Java prototype
02/26	Finish the design of the NER module on the Hadoop platform	Design of NER on Hadoop
03/29	Implement the Mapper and the Reducer for	Mapper and Reducer for

	NER	NER
04/20	Implement the interface with AVRO files	NER Mapper and Reducer supporting AVRO file
05/06	Finish the final project report	Project report

5.3 Evaluation and Discussion

In order to evaluate the performance of the NER module, we randomly select 100 tweets and 20 webpages from the small collection ("winter_storm_S"), label the named entities in them by the 2 team members, compare the manual result with automatic result, and figure out the precision and recall of the NER module on the two datasets.

Table.2 and Table.3 show the NER accuracy regarding various entity types on the tweet dataset and web page dataset respectively. Here, "True positive" means the number of relevant entities successfully identified; "False negative" means the number of relevant entities failed to identify; "False positive" means the number of irrelevant entities wrongly identified as named entity. The precision and recall are calculated as below:

Precision = True Positive / (True Positive + False Positive)
Recall = True Positive / (True Positive + False Negative)

Table.2 NER Performance on Tweet Data Set

PEOPLE		**ORGANIZATION**		**LOCATION**		**DATE**	
True Positive	2	**True Positive**	15	**True Positive**	11	**True Positive**	28
False Negative	1	**False Negative**	21	**False Negative**	60	**False Negative**	45
False Positive	0	**False Positive**	24	**False Positive**	0	**False Positive**	1
Precision	100.00%	**Precision**	38.46%	**Precision**	100.00%	**Precision**	96.55%
Recall	66.67%	**Recall**	41.67%	**Recall**	15.49%	**Recall**	38.36%

From Table.2, we can see that very few PEOPLE entities (3 entities out of 100 tweets) in the storm tweet collection. A lot of false positive occurs to ORGANIZATION entities. The reason might be Stanford NER always consider a series of word with first letter capitalized as ORGANIZATION entities, which is not always the case. We can also see Stanford NER misses a big proportion of LOCATION and DATE entities in tweets when looking at the high false

negative values. One possible reason regarding location is, the Stanford NER can't identify new locations. The weird date format (such as 1200 PM) in the cleaned tweets may explain the large amount of DATE entities missed by Stanford NER.

Table.3 NER Performance on Web Page Data Set

PEOPLE		ORGANIZATION		LOCATION		DATE	
True Positive	0	**True Positive**	0	**True Positive**	4	**True Positive**	73
False Negative	40	**False Negative**	57	**False Negative**	131	**False Negative**	31
False Positive	0	**False Positive**	0	**False Positive**	0	**False Positive**	0
Precision	N/A	**Precision**	N/A	**Precision**	100.00%	**Precision**	100.00%
Recall	0.00%	**Recall**	0.00%	**Recall**	2.96%	**Recall**	70.19%

From Table.3, we can see Stanford NER fails to find most of the PEOPLE, ORGANIZATION, and LOCATION entities from noise-reduced web pages. All of these 3 types of entities have high false negative. A very important reason is, all the text in the noise-reduced web pages is normalized. However, the case of first letter is an essential feature for Stanford NER. Therefore, it misses most of the PEOPLE, ORGANIZATION, and LOCATION. The accuracy in term of DATE is fairly good.

Generally speaking, the accuracy of NER module on the tweet dataset and the web page dataset is astonishing, compared that (F1>85%) stated by Stanford NLP group. Our past project [21] proved the good accuracy of Stanford NER dealing with raw text. Therefore, we have suggestions as below regarding the evaluation results above:

- Avoid doing normalization during noise reduction.
- Avoid changing the date format during noise reduction.
- Train Stanford NLP for a particular document collection to enhance the accuracy.

5.4 Development Preparation

❖ Import sample data into Solr

➢ Configure solrconfig.xml and schema.xml as shown in Fig.3.

```
<requestHandler name="/update/csv" class="solr.UpdateRequestHandler" startup="lazy">
      <lst name="defaults">
       <str name="separator">,</str>
       <str name="header">true</str>
       <str name="encapsulator">"</str>
      </lst>
```

Fig.3 Setup solrconfig.xml

➢ For schema.xml: Add the fields' name in our CSV data as shown in Fig.4.

```
<field name="a" type="string" indexed="true" stored="true"/>
<field name="b" type="string" indexed="true" stored="true"/>
<field name="c" type="string" indexed="true" stored="true"/>
<field name="d" type="string" indexed="true" stored="true"/>
<field name="e" type="string" indexed="true" stored="true"/>
<field name="f" type="string" indexed="true" stored="true"/>
<field name="g" type="string" indexed="true" stored="true"/>
<field name="h" type="string" indexed="true" stored="true"/>
<field name="i" type="string" indexed="true" stored="true"/>
<field name="j" type="string" indexed="true" stored="true"/>
<field name="k" type="string" indexed="true" stored="true"/>
<field name="l" type="string" indexed="true" stored="true"/>
<field name="m" type="string" indexed="true" stored="true"/>
<field name="n" type="string" indexed="true" stored="true"/>
```

Fig.4 Setup schema.xml

❖ Import CSV data

The following command used is to import CSV data:

```
curl 'http://localhost:8983/solr/update?commit=true' --data-binary
@example/exampledocs/paris_shooting-First200.csv -H 'Content-
type:application/csv
```

Note: here we add a header (fields name, such as a,b,c,d ...) in the paris_shooting-First200.csv

After importing data, we have the data shown in Fig.5 saved in Solr.

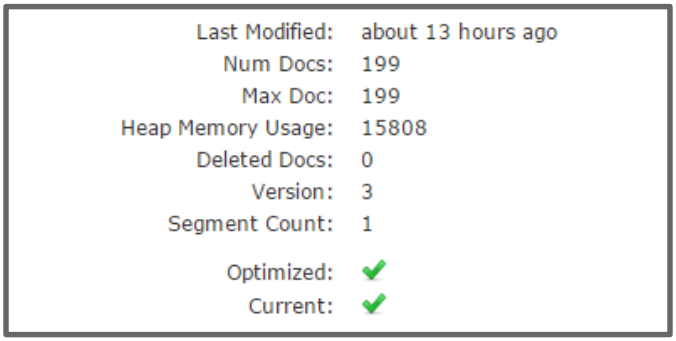

Fig.5 Statistics of Solr Data

Note: Here our data includes only 199 lines, because we delete one line as a test and 199 is not an importing data error.

Do an empty query as shown in Fig.6:

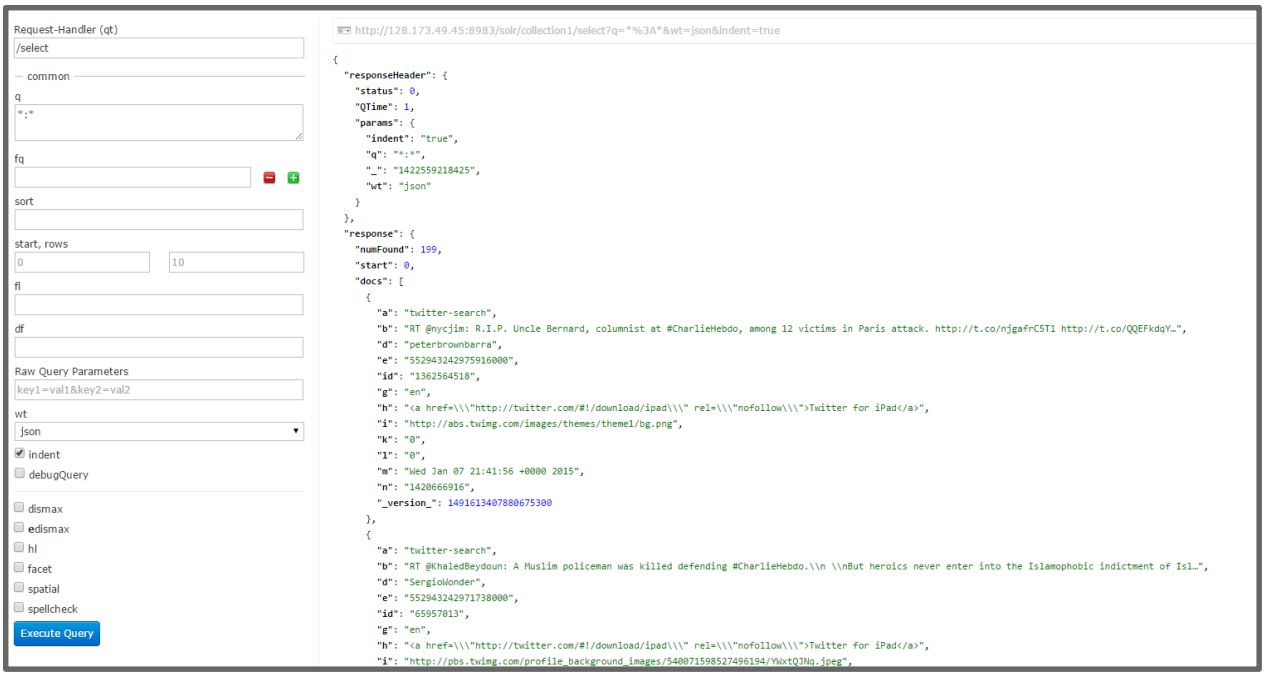

Fig.6 Query result after loading “paris_shooting-First200.csv”

❖ Import web pages folder into Solr
Run command as follows:

```
java -Dauto=yes -Drecursive=yes -jar post.jar webpages
```

Fig.7 shows the webpages data in Solr:

```
{
  "responseHeader": {
    "status": 0,
    "QTime": 2,
    "params": {
      "indent": "true",
      "start": "200",
      "q": "*:*",
      "_": "1423163458747",
      "wt": "json",
      "rows": "230"
    }
  },
  "response": {
    "numFound": 37163,
    "start": 200,
    "docs": [
      {
        "id": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/43.txt",
        "content_type": [
          "text/plain; charset=UTF-8"
        ],
        "resourcename": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/43.txt",
        "content": [
          " \n \n  \n  \n  \n  \n  \n \n  \r\n\tMQM strongly condemns the Paris Terror Attack
        ],
        "_version_": 1492204770228699100
      },
      {
        "id": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/30.txt",
        "content_type": [
          "text/plain; charset=UTF-8"
        ],
        "resourcename": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/30.txt",
        "content": [
          " \n \n  \n  \n  \n  \n  \n \n  Which countries still outlaw apostasy and blasphemy
        ],
        "_version_": 1492204770240233500
      },
      {
        "id": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/46.txt",
        "content_type": [
          "text/plain; charset=UTF-8"
        ],
        "resourcename": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/46.txt",
        "content": [
          " \n \n  \n  \n  \n  \n  \n \n  stephen strydom on Twitter: \"\"this is not a relig
        ],
        "_version_": 1492204770274836500
      },
      {
        "id": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/12.txt",
        "content_type": [
          "text/plain; charset=UTF-8"
        ],
        "resourcename": "/home/qianzhou/solr-4.9.1/example/exampledocs/webpages/12.txt",
        "content": [
          " \n \n  \n  \n  \n  \n  \n \n  BBC News - France sieges: Moment police raids both
        ],
```


Fig.7 Query result after loading webpages

- Import z_542_qb.csv
  Run command as follows:

```
curl 'http://localhost:8983/solr/update?commit=true' --data-binary
@example/exampledocs/z_542_qp.csv -H 'Content-type:application/csv
```

Fig.8 shows the z_542_qp.csv data in Solr:


```
"response": {
  "numFound": 37163,
  "start": 2000,
  "docs": [
    {
      "content": [
        "RT @counterfireorg: RT @pplsassembly: Excellent response by @jabberworks to #ParisShoo
      ],
      "name": "NumberMonkey69",
      "id": "553110254574370816",
      "description": "1420706735",
      "_version_": 1492290651312619500
    },
    {
      "content": [
        "Day of mourning in France after the #ParisShooting :( I was just there for work a coup
      ],
      "name": "lovelifetoday",
      "id": "553110251961327616",
      "description": "1420706735",
      "_version_": 1492290651312619500
    },
    {
      "content": [
        "RT @JaneFerguson5: Youngest suspect in #ParisShooting surrenders - more details emergi
      ],
      "name": "ummanais",
      "id": "553110249075257345",
      "description": "1420706734",
      "_version_": 1492290651312619500
    },
    {
      "content": [
        "RT @Swt_Sadhana: Lately I have realised that 'fear of consequences' is the only thing
      ],
      "name": "vickywiz",
      "id": "553110243010297857",
      "description": "1420706732",
      "_version_": 1492290651312619500
    },
```


Fig.8 Query result after loading Import z_542_qb.csv

➢ Import webpages_collections
Run command as follows:

```
java -Dauto=yes -Drecursive=yes -jar post.jar webpages_collections/
```

Fig.9 shows the webpages_collections data in Solr:

Fig.9 Query result after loading webpages_collections

6. User manual

The local prototype (developed in Java) implements the basic NER functions based on the Stanford NER tool [9]. In order to run this software, use the following command

For Linux:

NER.sh <text file> <NER destination file>

For Windows:

NER.bat <text file> <NER destination file>

Here, the "text file" is the path of the source text file, and "NER destination file" is the path of the file which will save the recognized named entities.

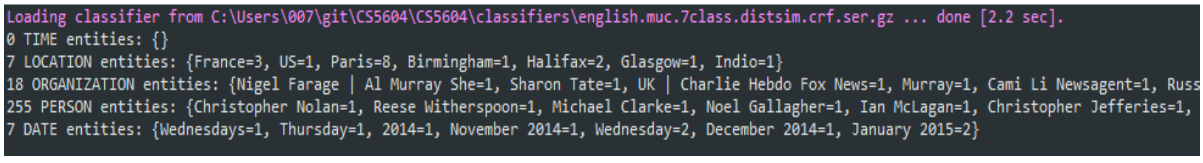

Fig.10 Snapshot of NER module (1st version)

The current version of our software can recognize 4 types of named entities (“person”, “location”, “organization”, and “date”), and figure out their term frequencies, given a text file. As shown in Fig.10, the output includes 4 types of entities, the number of types of entities, and the term frequency of an entity.

The usage of the NER prototype for Hadoop is introduced at the end of Section 7.2.

7. Developer manual

In this project, we are targeting on building a NER model on the data of Tweet and web pages. And the following four sections will introduce more details about our work step by step.

7.1 Project Development

7.1.1 Architecture

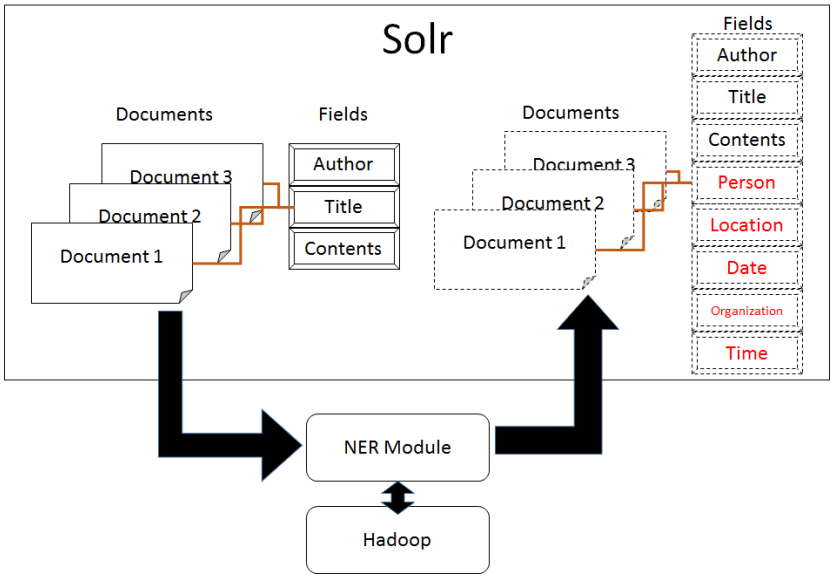

Fig.11 Architecture

As shown in Fig.11, the NER module gets documents (tweets or web pages) from Solr, extracting named entities such as person, location, date, organization, and time. The extracted entities are filled into the new fields of these documents in Solr. The Hadoop platform is used for the NER of big collections.

As mentioned before, we are using the baseline algorithm that is from Finkel’s work [4]. This model can be treated as a Conditional Random Fields (CRFs) classifier and it provides a general implementation of linear chain CRFs sequence models, which is introduced in Section 4.2.

7.1.2 Solr Schema

For every tweet or webpage, we plan to add the following fields in terms of named entities:

- Person
- Organization
- Location
- Date

These fields present the named entities extracted from a tweet or a web page.
Therefore, we design the corresponding Solr schema as shown in Fig.12, in order to index the fields.

```
<field name="person" type="string" indexed="true" stored="true"/>
  <field name="organization" type="string" indexed="true" stored="true"/>
  <field name="location" type="string" indexed="true" stored="true"/>
  <field name="date" type="string" indexed="true" stored="true"/>
```

Fig.12 Design in schema.xml

The value of each field will look like the following table. For each tweet or webpage, a field may have multiple values separated by double colons. For example, as shown in Table 4, if 3 person names (Obama, Putin, and Ban Ki-moon) occur in one webpage, they will be separated by double colons ("::") in the "person" field of this webpage.

Table.4 Examples of Extended Fields

Field	Value
person	Obama :: Putin :: Ban Ki-moon
organization	U.N. :: NATO
location	Washington, D.C. :: Moscow
date	Jan 1st 2015

7.1.3 Hadoop Interface

As shown in Fig. 13, the basic idea is using HDFS and Hadoop Streaming to build our NER model on the Hadoop Cluster. Hadoop Streaming uses a Mapper component to fetch text contents from the Input file on HDFS, segments file contents into small pieces, and distributes them to worker nodes. The NER operation for each piece of text, which may include some tweets or web pages, is done on each worker node. Each node produces an interim file which accommodates the named entities from that piece of text. After that, the Reducer component of Hadoop streaming gathers all the extracted named entities, and puts them in an Output file on HDFS.

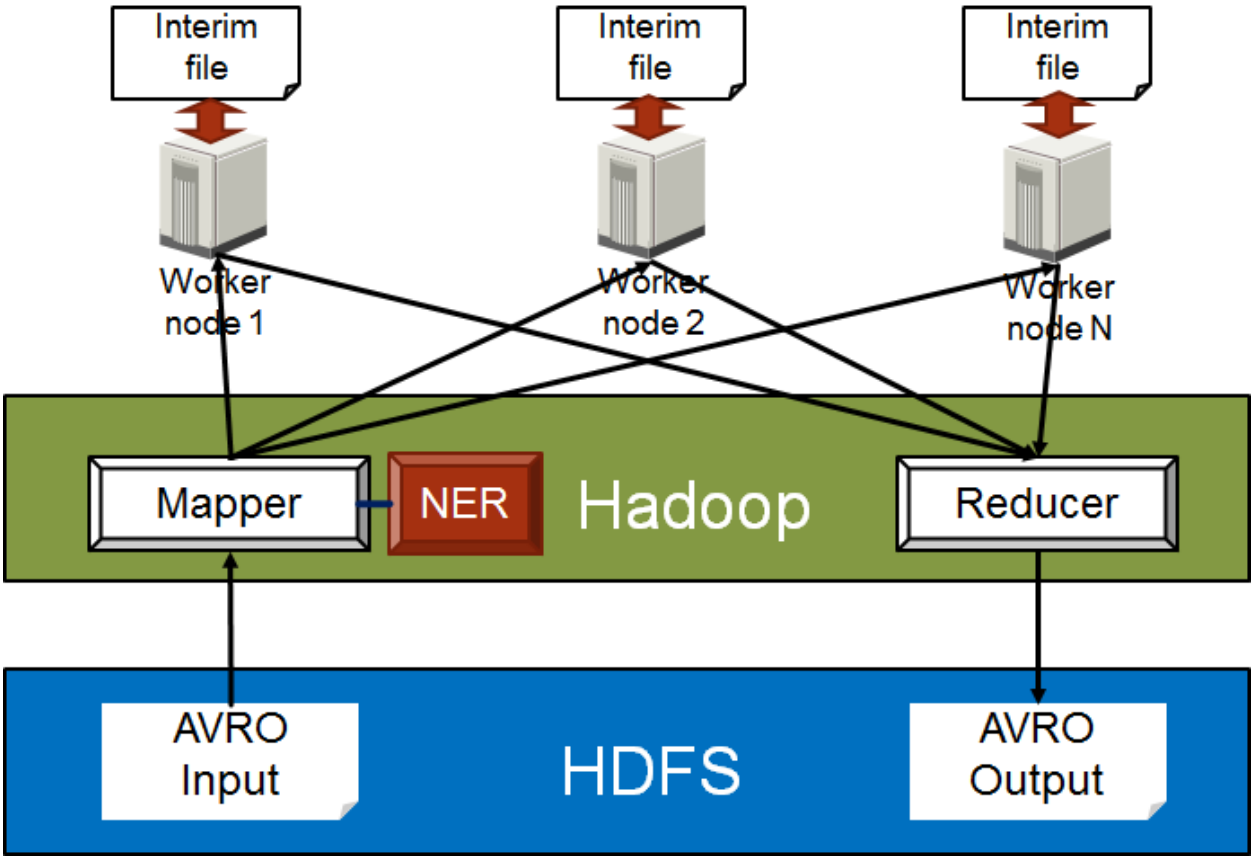

Fig.13 Tools & Environment on Hadoop Cluster

7.1.4 NER Model

We have introduced what NER is in above statement. Now we will discuss how to implement it. First of all, let us consider this question: which kind of named-entity should the phrase of "New York" be assigned? If only based on this information, most of us will assign it "Location". However, if "Times" occurs after "New York", we should assign it "Organization". Moreover, if no "Times" after "New York" but there are more than 2 sports-related words occurring in the same sentence, it might be an indicator that "New York" is a sports team that is an "Organization". Thus, when we try to recognize a word, we also need to consider the context of this word. So a regular graphical model, such as Naive Bayes, is not enough. In our project, we will use the linear-chain CRF model, which can do NER according to the dependency on neighboring words, to implement our NER prototype.

- **Problem Definition**

Now we will define this problem as follows:

Input: word_sequence = {X1, X2, X3, X4 … Xn} where 1,2,3,4 … n is the position of the words.

Output: tag_sequence = {Y1, Y2, Y3, Y4, … Yn} where item Yi is the tag assigned to Wi and tag might be "Location", "Person", "Organization", or "Date".

This problem of extracting entities from a word sequence of length n can be cast as a graphical model by introducing for each word, Xt, 1<= t <= n, a target variable Yt, which indicates the entity type of the word.

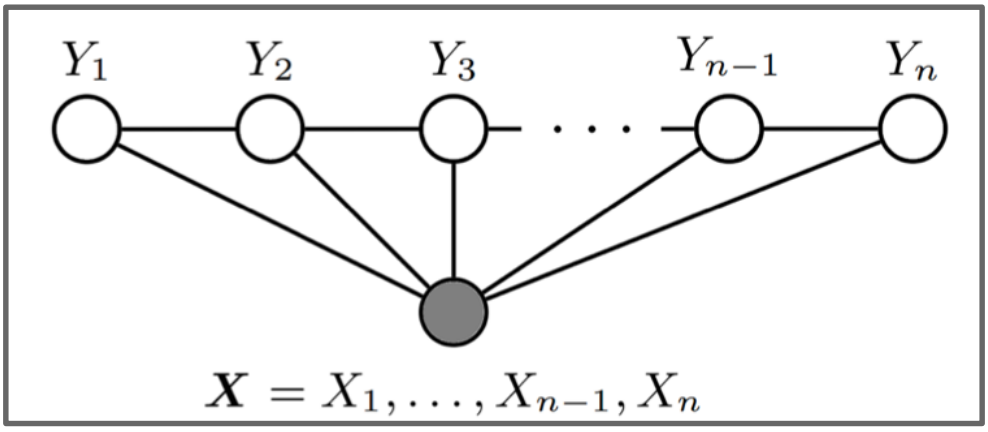

Fig.14 Linear-chain CRF Graph

➢ **Linear-chain CRF**

Linear-chain CRF is a special and important type of CRF. As shown in Fig. 14, it has two factors: one factor $\phi^1_{\square}(\square_{\square}, \square_{\square+1})$ to represents the dependency between neighboring target variables, and another factor $\phi^2_{\square}(\square_{\square}, \square_1, \square_2, \ldots \square_{\square})$that represents the dependency between a target and its context in the word sequence. These two dependencies can support our recognition job very well.

➢ **How to choose the best tag sequence**

The two former sections have introduced how to calculate the conditional distribution over X. So now we need to think about how to choose the best tag sequence. In our project, we will use a dynamic programming algorithm, Viterbi, to get the best sequence by maximizing the probability as follows.

$$
\begin{aligned}
y^* &= \arg\max_y P_w(y \mid x) \\
&= \arg\max_y \frac{\exp(wF(y,x))}{Z_w(x)} \\
&= \arg\max_y \exp(wF(y,x)) \\
&= \arg\max_y wF(y,x)
\end{aligned}
$$

7.2 Data Processing and Analysis

In our project, we have two types of data: Tweet and web pages. (Note: we have finished all processing of small collection) The TAs have given us the Twitter data in avro format. We extract the original URLs from tweets content by using the modified python scripts. And then we use Nutch to crawl the web pages data.

The tweet data is given to us, but it is not enough. We need extract the URLs from tweet content and crawl the web page data by using Nutch. First we extract the original URLs from the collection:

```
python short_to_long.py part-m-00000
```

And get the seed URL file: seedsURLs_part-m-00000.txt
Installing Nutch by following the tutorial.
Running Nutch to crawl the web page data and write them into sequence file "part-m-00000"
Note: We can do the same thing on both the small and big collections.

After the cleaning work of Noise Team, we need to learn how to deserialize the data from AVRO format:

First of all, we need to download four Jars, such as avro-1.7.7.jar, avro-tools-1.7.7.jar, jackson-core-asl-1.8.10.jar, and jackson-mapper-asl-1.8.10.jar.

Second, we need to create an avro schema file WebpageNoiseReduction.avsc and TweetNoiseReduction.avsc as follows:

```
{"type": "record", "namespace": "cs5604.webpage.NoiseReduction", "name": "WebpageNoiseReduction",
"fields": [{"type": "string", "name": "doc_id"}, {"default": null, "doc": "original", "type": ["null", "string"],
"name": "text_clean"}, {"default": null, "doc": "original", "type": ["null", "string"], "name": "text_original"},
{"default": null, "doc": "original", "type": ["null", "string"], "name": "created_at"}, {"default": null, "doc":
"original", "type": ["null", "string"], "name": "accessed_at"}, {"default": null, "doc": "original", "type": ["null",
"string"], "name": "author"}, {"default": null, "doc": "original", "type": ["null", "string"], "name": "subtitle"},
{"default": null, "doc": "original", "type": ["null", "string"], "name": "section"}, {"default": null, "doc":
"original", "type": ["null", "string"], "name": "lang"}, {"default": null, "doc": "original", "type": ["null", "string"],
"name": "coordinates"}, {"default": null, "doc": "original", "type": ["null", "string"], "name": "urls"}, {"default":
null, "doc": "original", "type": ["null", "string"], "name": "content_type"}, {"default": null, "doc": "original",
"type": ["null", "string"], "name": "text_clean2"}, {"default": null, "doc": "original", "type": ["null", "string"],
"name": "collection"}, {"default": null, "doc": "original", "type": ["null", "string"], "name": "title"}, {"default":
null, "doc": "original", "type": ["null", "string"], "name": "url"}, {"default": null, "doc": "original", "type": ["null",
"string"], "name": "appears_in_tweet_ids"}, {"default": null, "doc": "original", "type": ["null", "string"], "name":
"domain"}]}
```

```
{"type": "record", "namespace": "cs5604.tweet.NoiseReduction", "name": "TweetNoiseReduction", "fields":
```

```
[{"type": "string", "name": "doc_id"}, {"doc": "original", "type": "string", "name": "tweet_id"}, {"doc": "original", "type": "string", "name": "text_clean"}, {"doc": "original", "type": "string", "name": "text_original"}, {"doc": "original", "type": "string", "name": "created_at"}, {"doc": "original", "type": "string", "name": "user_screen_name"}, {"doc": "original", "type": "string", "name": "user_id"}, {"doc": "original", "type": ["string", "null"], "name": "source"}, {"doc": "original", "type": ["string", "null"], "name": "lang"}, {"doc": "original", "type": ["int", "null"], "name": "favorite_count"}, {"doc": "original", "type": ["int", "null"], "name": "retweet_count"}, {"doc": "original", "type": ["string", "null"], "name": "contributors_id"}, {"doc": "original", "type": ["string", "null"], "name": "coordinates"}, {"doc": "original", "type": ["string", "null"], "name": "urls"}, {"doc": "original", "type": ["string", "null"], "name": "hashtags"}, {"doc": "original", "type": ["string", "null"], "name": "user_mentions_id"}, {"doc": "original", "type": ["string", "null"], "name": "in_reply_to_user_id"}, {"doc": "original", "type": ["string", "null"], "name": "in_reply_to_status_id"}]}
```

And then we can compile the schema file:

```
java -jar /path/to/avro-tools-1.7.7.jar compile schema WebpageNoiseReduction.avsc .
```

```
java -jar /path/to/avro-tools-1.7.7.jar compile schema TweetNoiseReduction.avsc .
```

This step will generate a Java file: WebpageNoiseReduction.java including some useful functions, such as getId(), getCleanText(), and so on. We can compile WebpageNoiseReduction.java and get WebpageNoiseReduction.jar. (Note: we can do the similar thing on the tweet data, but we will repeat these steps for TweetNoiseReduction.java)

```
javac -classpath ./avro-1.7.7.jar:./avro-tools-1.7.7.jar:./jackson-core-asl-1.8.10.jar:./jackson-mapper-asl-1.8.10.jar WebpageNoiseReduction.java

jar cvf WebpageNoiseReduction.jar
```

Now we can write our code and use the functions from these Jars.

Deserializing:

```
File file = new File("part-m-00000.avro");
        DatumReader<WebpageNoiseReduction> userDatumReader = new
SpecificDatumReader<WebpageNoiseReduction>(WebpageNoiseReduction.class);
        DataFileReader<WebpageNoiseReduction> dataFileReader = null;
        try {
       dataFileReader = new DataFileReader<WebpageNoiseReduction>(file, userDatumReader);
    } catch (IOException e) {
    }
        int num = 0;
        sqoop_import_z_160 siz = null;
        try {
```

```
    while (dataFileReader.hasNext()) {
      // Reuse user object by passing it to next(). This saves
      // us from allocating and garbage collecting many objects for
      // files with many items.
            num ++;
      siz = dataFileReader.next(siz);
      System.out.println(siz.getId());
            if(num == 10){
                  break;
            }
    }
  } catch (IOException e) {
  }
```

Serializing:

```
WebpageNoiseReduction record1 = new webpage("December 08|December 11", None, "Roberto", None);
WebpageNoiseReduction record2 = new webpage("December 07|December 12", None, “Tom", None");
WebpageNoiseReduction record3 = new webpage("December 08|December 11", None, "Ronaldo", None);

DatumWriter<WebpageNoiseReduction> userDatumWriter = new SpecificDatumWriter<WebpageNoiseReduction>(sqoop_import_z_160.class);
DataFileWriter<WebpageNoiseReduction> dataFileWriter = new DataFileWriter<WebpageNoiseReduction>(userDatumWriter);
dataFileWriter.create(record1.getSchema(), new File("part-m-00000.avro"));
dataFileWriter.append(record1);
dataFileWriter.append(record2);
dataFileWriter.append(record3);
dataFileWriter.close();
```

In the next section , we will introduce the interface of Hadoop to implement serializing and deserializing.

We also run NER analysis on both datasets. The input and output are as follows:

{u'lang': u'English', u'user_id': u'479797173', u'in_reply_to_user_id': None, u'text_clean': u'RT Severe Snow Storm England 19623 RT if you secretly want this in winter 201415 ', u'created_at': u'1412891544', u'hashtags': None, u'coordinates': u'0.0,0.0', u'user_mentions_id': u'KasimAwan', u'text_original': u'RT @KasimAwan: Severe Snow Storm, England 1962/3. RT if you secretly want this in winter 2014/15, :) http://t.co/lQsTEi8SOV', u'tweet_id': u'520330967576494080', u'in_reply_to_status_id': None, u'source': u'twitter-search', u'urls': u'http://t.co/lQsTEi8SOV', u'retweet_count': None, u'doc_id': u'winter_storm_S--1', u'user_screen_name': u'CHARLESCOLBY1', u'favorite_count': None, u'contributors_id': None}
{u'lang': u'English', u'user_id': u'1027114068', u'in_reply_to_user_id': None, u'text_clean': u'RT Colorado Rockies winter storm Central Mtns 48 gt10000kft 13 snow900010000kft Pikes Peak814 ', u'created_at': u'1412891396', u'hashtags': u'COwx', u'coordinates': u'0.0,0.0', u'user_mentions_id': u'WilliamMScherer', u'text_original': u'RT @WilliamMScherer: Colorado Rockies winter storm. Central Mtns: 4"-8" >10,000\'kft. 1"-3" snow:9000\'-10000\'kft. Pikes Peak:8"-14"+. #COwx \xe2\u20ac\xa6', u'tweet_id': u'520330344818819073', u'in_reply_to_status_id': None, u'source': u'twitter-search', u'urls': None, u'retweet_count': None, u'doc_id': u'winter_storm_S--2', u'user_screen_name': u'gjeni_u', u'favorite_count': None, u'contributors_id': None}
{u'lang': u'English', u'user_id': u'1473102152', u'in_reply_to_user_id': None, u'text_clean': u'RT Colorado Rockies winter storm Central Mtns 48 gt10000kft 13 snow900010000kft Pikes Peak814 ', u'created_at': u'1412891347', u'hashtags': u'COwx', u'coordinates': u'0.0,0.0', u'user_mentions_id': u'WilliamMScherer', u'text_original': u'RT @WilliamMScherer: Colorado Rockies winter storm. Central Mtns: 4"-8" >10,000\'kft. 1"-3" snow:9000\'-10000\'kft. Pikes Peak:8"-14"+. #COwx \xe2\u20ac\xa6', u'tweet_id': u'520330141801926656', u'in_reply_to_status_id': None, u'source': u'twitter-search', u'urls': None, u'retweet_count': None, u'doc_id': u'winter_storm_S--3', u'user_screen_name': u'hokeekat', u'favorite_count': None, u'contributors_id': None}

Input

The command for running NER.

```
export LIBJARS=lib/avro-1.7.7.jar,lib/avro-mapred-1.7.7-hadoop2.jar,lib/stanford-corenlp-3.3.0.jar,lib/stanford-corenlp-3.3.0-models.jar,lib/stanford-ner.jar,lib/NER.jar

export HADOOP_CLASSPATH=lib/avro-1.7.7.jar:lib/avro-mapred-1.7.7-hadoop2.jar:lib/stanford-corenlp-3.3.0.jar:lib/stanford-corenlp-3.3.0-models.jar:lib/stanford-ner.jar:lib/NER.jar

hadoop jar lib/NER.jar -libjars ${LIBJARS} -files classifiers/english.muc.7class.distsim.crf.ser.gz winter_storm_S_AVRO/part-m-00000.avro testoutput-1
```

Command

The output is shown below: All possible words are listed after the corresponding labels, such as ‘ner_dates’, ‘ner_locations’, ‘ner_people’, and ‘ner_organization’. If no word is assigned to a particular named entity, the value is None after the corresponding label.

{u'ner_dates': u'December 08|December 11', u'ner_locations': None, u'doc_id': u'winter_storm_S--100052', u'ner_people': None, u'ner_organizations': u'NWS'}
{u'ner_dates': u'Monday', u'ner_locations': None, u'doc_id': u'winter_storm_S--100067', u'ner_people': None, u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--10010', u'ner_people': None, u'ner_organizations': None}
{u'ner_dates': u'Winter', u'ner_locations': None, u'doc_id': u'winter_storm_S--100133', u'ner_people': None, u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100148', u'ner_people': None, u'ner_organizations': u'Heavy'}
{u'ner_dates': u'Tuesday|Winter', u'ner_locations': None, u'doc_id': u'winter_storm_S--100214', u'ner_people':

```
None, u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100229', u'ner_people': None,
u'ner_organizations': u'ALERT Winter Storm Watch'}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--10025', u'ner_people': u'Blaine
Countys', u'ner_organizations': None}
{u'ner_dates': u'Tuesday|Thursday', u'ner_locations': None, u'doc_id': u'winter_storm_S--100283', u'ner_people':
None, u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100298', u'ner_people': None,
u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100364', u'ner_people': None,
u'ner_organizations': u'Heavy Snow Possible Winter Storm Watch|Northeast PA|Coal Region Endless Mtns'}
{u'ner_dates': u'Winter', u'ner_locations': None, u'doc_id': u'winter_storm_S--100379', u'ner_people': None,
u'ner_organizations': None}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100430', u'ner_people': None,
u'ner_organizations': u'Northeast Tue'}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100445', u'ner_people': None,
u'ner_organizations': u'Northeast Tue'}
{u'ner_dates': None, u'ner_locations': None, u'doc_id': u'winter_storm_S--100511', u'ner_people': None,
u'ner_organizations': u'Winter Storm Watch|Eastern Franklin Eastern Hampshire'}
{u'ner_dates': u'December 08|December 10', u'ner_locations': None, u'doc_id': u'winter_storm_S--100526',
u'ner_people': None, u'ner_organizations': u'NWS'}
{u'ner_dates': u'December 08|December 10', u'ner_locations': None, u'doc_id': u'winter_storm_S--100580',
u'ner_people': None, u'ner_organizations': u'NWS'}
{u'ner_dates': u'December 08|December 11', u'ner_locations': u'Binghamton', u'doc_id': u'winter_storm_S--
100595', u'ner_people': None, u'ner_organizations': u'NWS'}
```

Output

7.3 Code

This NER model is based on the linear-chain CRF model. We are showing the core function of NER as follows (the code is changed from the example in [9]):

```
public static Map<String, String> extractNamedEntities(String text)
                                throws IOException {

                        Map<String, String> result = new HashMap<String, String>();

                        List<String> personList = new ArrayList<String>();
                        List<String> orgList = new ArrayList<String>();
                        List<String> dateList = new ArrayList<String>();
                        List<String> locationList = new ArrayList<String>();

                        // Annotate every word in the file with the 7 named entity types
                        List<List<CoreLabel>> out = classifier.classify(text);
                        //List<List<String>> entitiesInSentences = new LinkedList<List<String>>();
                        List<String> entitiesInDoc = new LinkedList<String>();

                        /* Modified List<List<CoreLabel>> out, combine continuous words which belong to same
entity type*/
                        // For every sentence
                        for (int i = 0; i < out.size(); i++) {
```

```
                        String comWord = out.get(i).get(0).word();// word after combination
                          // indicate type of previous word
                        String tag = out.get(i).get(0).get(CoreAnnotations.AnswerAnnotation.class);

                        //for each word in this sentence
                        for (int j = 1; j < out.get(i).size(); j++) {
                                String word = out.get(i).get(j).word();
                                String category =
out.get(i).get(j).get(CoreAnnotations.AnswerAnnotation.class);

                                // combine continuous same-type words
                                if(category.equals(tag)) {
                                        comWord = comWord + " " + word;
                                        out.get(i).remove(j);
                                        j = j - 1;
                                        out.get(i).get(j).setWord(comWord);
                                }
                                else
                                        comWord = word;

                                tag = category;
                        }

                        //Put the identified named entities in the corresponding sets
                        for (CoreLabel cl : out.get(i)) {
                                String word = cl.word();
                                String category = cl.get(CoreAnnotations.AnswerAnnotation.class);
                                // System.out.println(word + " : " + category);

                                if(category.equals("PERSON")) {
                                        //Rule out the meaningless entities such as "Reuters"
                                        if(!Arrays.asList(STOP_ENTITIES).contains(word)){
                                                entitiesInDoc.add(word);
                                                personList.add(word);
                                        }
                                }
                                else if (category.equals("ORGANIZATION")) {
                                        if(!Arrays.asList(STOP_ENTITIES).contains(word)){
                                                entitiesInDoc.add(word);
                                                orgList.add(word);
                                        }
                                }
                                else if (category.equals("LOCATION")) {
                                        entitiesInDoc.add(word);
                                        locationList.add(word);
                                }
                                else if (category.equals("DATE")) {
                                        entitiesInDoc.add(word);
                                        dateList.add(word);
                                }
                        }
                }

                //Save the 4 kinds of entities in the final result
```

```
		StringBuffer buffer = new StringBuffer();

		int i=0;
		for(String term : personList){
			buffer.append(term);
			if(i != personList.size()-1)
				buffer.append(ENTITY_SEPARATOR);
			i++;
		}
		result.put(NERConstants.NAMED_ENTITY_PERSON, buffer.toString());

		i=0;
		buffer = new StringBuffer();
		for(String term : orgList){
			buffer.append(term);
			if(i != orgList.size()-1)
				buffer.append(ENTITY_SEPARATOR);
			i++;
		}
		result.put(NERConstants.NAMED_ENTITY_ORGANIZATION,buffer.toString());

		i=0;
		buffer = new StringBuffer();
		for(String term : locationList){
			buffer.append(term);
			if(i != locationList.size()-1)
				buffer.append(ENTITY_SEPARATOR);
			i++;
		}
		result.put(NERConstants.NAMED_ENTITY_LOCATION,buffer.toString());

		i=0;
		buffer = new StringBuffer();
		for(String term : dateList){
			buffer.append(term);
			if(i != dateList.size()-1)
				buffer.append(ENTITY_SEPARATOR);
			i++;
		}
		result.put(NERConstants.NAMED_ENTITY_DATE, buffer.toString());

		return result;
	}
```

Next we introduce how to run a Mapreduce job on a Hadoop cluster. In order to run the NER function on Hadoop, we need a driver class, which extends the "Configured" class of Hadoop, and implements the "Tool" interface. A main() function is needed, which is used to start the Hadoop job. The most important function of the driver class is run(), which sets the configuration of the Hadoop job. The configuration comprises of at least the following:

- Job class
- Job name

- Input and output path
- Mapper class
- Data types of Mapper's input and output parameters
- Combiner class
- Reducer class
- Output schemas

```
public class NERDriver extends Configured implements Tool {
        public int run(String[] rawArgs) throws Exception {
                Job job = new Job(super.getConf());
                job.setJarByClass(NERDriver.class);
                job.setJobName("NER");

                String[] args =
                 new GenericOptionsParser(rawArgs).getRemainingArgs();
                Path inPath = new Path(args[0]);
                Path outPath = new Path(args[1]);

                FileInputFormat.setInputPaths(job, inPath);
                FileOutputFormat.setOutputPath(job, outPath);
                outPath.getFileSystem(super.getConf()).delete(outPath, true);

                job.setInputFormatClass(AvroKeyInputFormat.class);
                job.setMapperClass(AvroNERMapper.class);
                AvroJob.setInputKeySchema(
                 job, sqoop_import_z_160.getClassSchema());
                job.setMapOutputKeyClass(Text.class);
                job.setMapOutputValueClass(Text.class);

                job.setCombinerClass(AvroNERCombiner.class);

                job.setOutputFormatClass(AvroKeyValueOutputFormat.class);
                job.setReducerClass(AvroNERReducer.class);
                AvroJob.setOutputKeySchema(
                 job, Schema.create(Schema.Type.STRING));
                AvroJob.setOutputValueSchema(job, Schema.create(Schema.Type.STRING));

                return (job.waitForCompletion(true) ? 0 : 1);
        }

        public static void main(String[] args) {
                int result;
                try {
                        result = ToolRunner.run(new NERDriver(), args);
                        System.exit(result);
                 } catch (Exception e) {
                         e.printStackTrace();
                 }
}
```

Mapper class

```
public static class AvroAverageMapper extends
        Mapper<AvroKey<sqoop_import_z_160>, NullWritable, Text, Text> {
         protected void map(AvroKey<sqoop_import_z_160> key, NullWritable value,
                        Context context) throws IOException, InterruptedException {
                CharSequence cs = key.datum().getId();
                Text tweetID = new Text(cs.toString());

                cs = key.datum().getText();
                String text = cs.toString();
                Map<String, String> map = NERUtil.extractNamedEntities(text);
                StringBuffer buffer = new StringBuffer();
                int i = 0;
                for (String entityType : map.keySet()) {
                        buffer.append(entityType + "||" + map.get(entityType));
                        if (i != map.size() - 1)
                                buffer.append("$$");
                        i++;
                }
                String namedEntities = buffer.toString();
                Text tNamedEntities = new Text(namedEntities);
                context.write(tweetID, tNamedEntities);
        }
  }
```

An essential class of Hadoop application is the Mapper class, which performs the core analysis on each data instance (a tweet or a web page in our case). The customized Mapper class extends the Mapper class of Hadoop, which implements the key function map(). In the NER case, the map() function gets the ID and content of each document (tweet or web page), applies StanfordNER to extract named entities from the contents, then takes the document ID and extracted named entities as the key and value of the output.

Reducer Class

```
public static class AvroNERReducer extends
                Reducer<Text, Text, AvroKey<String>, AvroValue<String>> {

          protected void reduce(Text key, Iterable<Text> value, Context context)
                        throws IOException, InterruptedException {
                String newKey = key.toString();
                String newValue = null;
                for(Text text : value){
                        newValue = text.toString();
                }
                context.write(new AvroKey<String>(newKey), new AvroValue<String>(newValue));
        }
  }
```

Another important class of Hadoop application is the Reducer class, which aims to combine the output of the Mapper class. There might be multiple values in term of a key, which we should merge into one. However, we assume the keys (tweet IDs) are never duplicated among multiple Map-Reduce jobs in this case. Therefore, the merging job is not really necessary. What the Reducer does is just getting the last value (also the single value) of each key, and writing the key and value into the final output.

3 instructions are needed to run the NER job on Hadoop:

- Create the LIBJARS environmental variable, which comprises the 3rd party .jar files needed by Stanford NER and AVRO file extraction. Here we uploaded the .jar files to the head node of the cluster (under a "lib" subfolder of current directory).
- Update the HADOOP_CLASSPATH environmental variable, make sure Hadoop copies these Jar files into its classpath.
- Start the Hadoop job. The first and most important argument is the .jar file (lib/NER.jar) which contains the driver class and other NER implementations. The "-libjars" parameter uses the LIBJARS environmental variable to specify the 3rd party .jar files, separated by comma. The "-files" argument specifies other needed files, such as the StanfordNER model file, also separated by comma. These files are also uploaded to the head node of the cluster in advance. The last two arguments are the input and output data in HDFS. In this case, we used the AVRO file of the winter storm, and output the extracted NER to a directory named "testoutput-1".

The NER output of the small tweets collection "Winter Storm" in **AVRO format** is place at "/user/cs5604s15_ner/shared/output.avro" of the HDFS. As shown in the above snapshot, each tweet has a key and a value. The key is the tweet ID, while the value is a string which contains named entities extracted from that tweet. The composition of the value string is the same with that in plain text. The Map-Reducer job produces 36 AVRO files under the "/user/cs5604s15_ner/testoutput-25" directory. Those files are further merged into one by running the following command.

```
avro-tools concat part-r-00000.avro  part-r-00001.avro part-r-00002.avro part-r-
00003.avro part-r-00004.avro part-r-00005.avro part-r-00006.avro part-r-00007.avro
part-r-00008.avro part-r-00009.avro part-r-00010.avro part-r-00011.avro part-r-
00012.avro part-r-00013.avro part-r-00014.avro part-r-00015.avro part-r-00016.avro
part-r-00017.avro part-r-00018.avro part-r-00019.avro part-r-00020.avro part-r-
00021.avro part-r-00022.avro part-r-00023.avro part-r-00024.avro part-r-00025.avro
part-r-00026.avro part-r-00027.avro part-r-00028.avro part-r-00029.avro part-r-
00030.avro part-r-00031.avro part-r-00032.avro part-r-00033.avro part-r-00034.avro
part-r-00035.avro winter_storm_S-Tweet-NamedEntities.avro
```

The MapReduce driver is updated to produce the corresponding AVRO file. The main change is the Reducer class, which parses the named entity string exported by the Mapper and write the named entities to AVRO file following the specified schema.

This class takes the TweetNER class as the key of the Map-Reduce output, while keeps the value as NULL. It parses the output of Mapper by splitting the named entity string, using "##" as separator. Then, the parsed named entities are saved into a TweetNER instance and written into the AVRO file.

The NER job for the "winter_storm_S" collection takes 6 minutes.
The NER job for the "storm_B" collection takes 9 minutes.

```
public static class AvroNERReducer extends
                Reducer<Text, Text, AvroKey<TweetNER>, NullWritable> {

        protected void reduce(Text key, Iterable<Text> value, Context context)
                        throws IOException, InterruptedException {
                TweetNER ne = new TweetNER();
                String sKey = key.toString();
                String sValue = null;
                for(Text text : value){
                        sValue = text.toString();
                }
                String[] array = sValue.split("##", -1);

                ne.setDocId(sKey);

                String dates = array[3].trim();
                if( dates.length() > 0)
                        ne.setNerDates(dates);

                String location = array[0].trim();
                if(location.length() > 0)
                        ne.setNerLocations(location);

                String organization = array[1].trim();
                if(organization.length() > 0)
                        ne.setNerOrganizations(organization);

                String people = array[2].trim();
                if(people.length() > 0)
                        ne.setNerPeople(people);

                context.write(new AvroKey<TweetNER>(ne), NullWritable.get());
```

```
        }
}
```

The AVRO files produced for the tweets in “winter_storm_S” collection and “storm_B” collection can be found under the “/user/cs5604s15_ner/shared” directory. The two files are “winter_storm_S-Tweet-NamedEntities.avro” and “storm_B-Tweet-NamedEntities.avro”.
The AVRO file produced for the web pages in “winter_storm_S” collection can be found under the “/user/cs5604s15_ner/shared” directory as well. The file is “winter_storm_S-Webpage-NamedEntities.avro”.

7.4 Inventory of all the files

- NER-Report-Final.docx
  Our final project report in docx format.
- NER-Report-Final.pdf
  Our final project report in pdf format.
- avro-schema.zip
  It includes four AVRO schema files as follows:
  - tweet_cleaned.avsc
    This schema is used for cleaned tweet AVRO data from Noise team.
  - webpage_cleaned.avsc
    This schema is used for cleaned web page AVRO data from Noise team.
  - ner_tweet_result.avsc
    This schema is used for tweet NER result AVRO data from NER team
  - ner_webpage_result.avsc
    This schema is used for web page NER result AVRO data from NER team.
- CS5604-NER.zip
  NER source code for Hadoop. You can learn more by reading comments.
  - NER.sh
    Run NER model on datasets.
  - hadoop.sh
    The commands to run the NER classes on Hadoop cluster.
    Edits are needed according to the environment.
  - src directory
    All java source files, such as the Mapper classes and Reducer classes as introduced in above sections.
  - lib directory.
    All jars files, such as stanford-ner.jar, stanford-corenlp-3.3.0.jar, and so on.
- ProjectPresentation.ppt
  Our final project presentation in ppt format.

- ProjectPresentation.pdf
  Our final project presentation in pdf format.

8. Conclusion

In this project, we learn the basic knowledge background of NER and have done a literature review for the previous works, such as Stanford NER, LIPI, OpenCalasi, etc. Our model is implemented based on Stanford NER and we parallelize our model on the Hadoop cluster. Also our model is expanded to support serializing and deserializing AVRO data. And then we randomly select 100 tweets and 20 webpages, label the named entities in them, compare the manual result with automatic result, and figure out the precision and recall of the NER module on the two datasets.

Acknowledgements

This project is supported by the US National Science Foundation through grant **IIS - 1319578**. This project would not have been possible to finish so smoothly without Instructor Dr. Fox, GTA Sunshin Lee and GRA Mohammed's help. Thank you for your help and your input did make a difference.

References

[1] Nadeau, David, and Satoshi Sekine. "A survey of named entity recognition and classification." *Lingvisticae Investigationes* 30.1 (2007): 3-26.

[2] Zhou, GuoDong, and Jian Su. "Named entity recognition using an HMM-based chunk tagger." *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2002.

[3] Ratinov, Lev, and Dan Roth. "Design challenges and misconceptions in named entity recognition." *Proceedings of the Thirteenth Conference on Computational Natural Language Learning*. Association for Computational Linguistics, 2009.

[4] Finkel, Jenny Rose, Trond Grenager, and Christopher Manning. "Incorporating non-local information into information extraction systems by Gibbs sampling." *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*. Association for Computational Linguistics, 2005.

[5] Apache Solr Quick Start Tutorial, http://lucene.apache.org/solr/quickstart.html, 2014

[6] Grishman Ralph, Sundheim B. "Message Understanding Conference - 6: A Brief History". In Proc. International Conference On Computational Linguistics.

[7] Alias-i (2008, February 22, 2013). LingPipe 4.1.0. Available: http://alias-i.com/lingpipe

[8] Thomson Reuters. (2008). Clais Web Service. Available: http://www.openclais.com

[9] Stanford NER. Available: http://nlp.stanford.edu/software/CRF-NER.shtml

[10] D. Nadeau, “Semi-Supervised Named Entity Recognition: Learning to Recognize 100 Entity types with Little Supervision,” Ottawa Carleton Institute for Computer Science, School of Information Technology and Engineering, University of Ottawa, 2007
[11] L. Tanabe, N. Xie, L. H. Thom, W. Matten, and W. J. Wilbur, “GENETAG: a tagged corpus for gene/protein named entity recognition,” BMC Bioinformatics, VOL 6, p. S3, 2005
[12] Bach, Nguyen, and Sameer Badaskar. "A review of relation extraction", Literature review for Language and Statistics II, 2007.
[13] Lin, DeKang, Xiaoyun Wu, “Phrase Clustering for discriminative learning” Annual Meeting of the ACL and JCNLP. pp. 1030-1038
[14] Changki Lee, Yi-Gyu Hwang, Hyo-Jung Oh, and Jeong Heo, “Fine-Grained Named Entity Recognition Using Conditional Random Fields for Questions Answering”, Information Retrieval Technology Lecture Notes in Computer Science Volume 4182, 2006, pp. 581-587
[15] http://en.wikipedia.org/wiki/Named-entity_recognition
[16] Ratinov, Lev, et al. "Local and global algorithms for disambiguation to wikipedia." Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1. Association for Computational Linguistics, 2011.
[17] L. R. Rabiner. “A tutorial on Hidden Markov Models and selected applications in speech recognition”. Proceedings of the IEEE, 77(2):257–286, 1989
[18] Acanfora, Joseph; Evangelista, Marc; Keimig, David; Su, Myron, “Natural Language Processing: Generating a Summary of Flood Disasters”, Course project report of “CS 4984 Computational Linguistics”, Virginia Tech, Blacksburg, VA, 2014, Available: https://vtechworks.lib.vt.edu/handle/10919/51134
[19] Crowder, Nicholas; Nguyen, David; Hsu, Andy; Mecklenburg, Will; Morris, Jeff, “Computational Linguistics Hurricane Group”, Course project report of “CS 4984 Computational Linguistics”, Virginia Tech, Blacksburg, VA, 2014, Available: https://vtechworks.lib.vt.edu/handle/10919/51136
[20] Fox, Edward; Hanna, Kristine; Shoemaker, Donald; Kavanaugh, Andrea; Sheetz, Steven. IDEAL project, Available: http://www.eventsarchive.org/
[21] Zhang, Xuan; Huang, Wei; Wang, Ji; Geng, Tianyu, “Unsupervised Event Extraction from News and Twitter”, Course project report of “CS 6604 Digital Library”, Virginia Tech, Blacksburg, VA, 2014, Available: https://vtechworks.lib.vt.edu/handle/10919/47954
[22] Lafferty, J., McCallum, A., & Pereira, F. C. (2001). Conditional random fields: Probabilistic models for segmenting and labeling sequence data.