Computational Linguistic Analysis of Earthquake Collections

December 10, 2014

Kenneth Reed Bialousz
Kevin Kokal
Kwamina Orleans-Pobee
Chris Wakeley

CS 4984 - Computational Linguistics Capstone
Virginia Tech, Blacksburg, VA 24061

Table of Contents

(View in PDF form to use hyperlinks)

Executive Summary

CS4984 is a newly-offered class at Virginia Tech with a unit based, project-problem learning curriculum. This class style is based on NSF-funded work on curriculum for the field of digital libraries and related topics; and in this class, is used to guide a student based investigation of computational linguistics. More specifically, this class is corpus linguistics oriented and explores the question:

What is the best summary that can be automatically generated for your type of event?

Corpus linguistics is a form of linguistic analysis focused on collections of "real word" texts known as corpora. In the scope of this class, corpora consist of web archives related to a variety of event types ranging from community events to fires. Summaries of these events consist of one or two paragraphs of well-written English text that describe the event. The generation of these summaries involves identifying the most critical pieces of information in natural language text, extracting the relevant information, and combining the extracted information to produce an informative and concise summary. The class is structured such that each event type and its related corpora are assigned to an individual group of students for investigation. It was the task of our group to investigate earthquake related events.

The specific problem this report addresses is the creation of a means to automatically generate a short summary of a corpus of articles about earthquakes. Such a summary should be best representative of the texts and include all relevant information about earthquakes. For our analysis, we operated on two corpora--one about a 5.8 magnitude earthquake in Virginia in August 2011, and another about a 6.6 magnitude earthquake in April 2013 in Lushan, China. Techniques used to analyze the articles include clustering, lemmatization, frequency analysis of n-grams, and regular expression searches.

The report first provides a manual for the use of the project's codebase followed by a manual for further development. The implementation and results of the project are then provided and discussed.

User Manual

The code base of the project can be found at the following URL:
https://github.com/atokop/compling

Requirements:
Much of the code is written in Python and thus requires a Python environment to execute. The following modules are required:

- numpy
- pyparsing
- dateutil
- six
- matplotlib

In addition to a Python environment, some code is written for a Hadoop cluster and uses the mapreduce programming model. Consequently, a Hadoop system or virtual machine is required for some portions of the code.

Running Instructions see each unit's directory on the provided GitHub repository above and then execute the corresponding script for each unit after making necessary imports.

Developer Manual

The code base of the project can be found at the following URL:
https://github.com/atokop/compling

U1

Two functions defined, import u1_analysis:

most_frequent(text)
Returns the 20 most frequent words in the given text

most_frequent_above_n(text, n)
Returns the 20 most frequent words in the given text with lengths greater than n

U2

Script is defined in small_analysis.py. Loads in YourSmall, lemmatizes, and provides frequency distribution

U3

Functions defined in u3.py

tagged_nouns(tagged_words)
Given words with their tags, returns all tagged words that are nouns

brown_tagged_sents()
Returns a training set and test set generated from the Brown corpus.

simple_pos_tagger(words)
Tags words with NLTK's default tagger, and returns a set without stop words and duplicates.

get_regexp_tagger()
Returns a tagger that tags words based on regex on their endings

sents_to_words()
Expands a list of sentences into one list containing all the words.

trigram_tag(sentences, default_tagger)
Tags a list of sentences, via backoff trigram tagging. If no default tagger is specified uses a regex tagger.

Script implementation in u3_ce.py. Creates a tagger and stores it as tagger.pkl

U4

Functions defined in u4.py

load_words(filename)

Given a filename, decodes contents via Unicode, then parses the content into a list of words.

load_clean_words

Like load_words, but cleans invalid characters out of words returned.

contains_word_feature_set(document_words, feature_words)

Creates a feature set for the set of words provided. The featureset consists of a set of Boolean flags, one for each word in feature words, that are true if the word is found in document words.

contains_word_feature_set_from_file(filename, feature_words)

Similar to *contains_word_feature_set* but takes in a filename instead of the document words.

top_words_by_frequency(words, n=20)

Returns the words with the highest frequency in the list of words given (default: top 20).

top_words_with_frequency(words, n=20)

Similar to *top_words_by_frequency* but returns the frequency of each word in a tuple with the word (default: top 20).

create_boolean_training_sets(create_featureset_func, tagged_dir)

Given the directory containing a training set of files that are tagged either positive or negative by appending '_pos.txt' or '_neg.txt', creates featuresets for and mixes up the tagged data to create a training set containing 75% of the tagged data and a testing set containing 25% of the data as a tuple.

Script in u4_execution.py

- Creates a featureset by determining most common words in the corpus
- Creates training and test sets from tagged data,
- Trains a Decision Tree Classifier on this data
- Trains a Naive Bayes Classifier on the training data
- Prints the accuracy of the Decision Tree on the test data
- Prints the accuracy of the Naive Bayes on the test data

U5

u5_execution.py

file_words(corpus_root)

Returns a list containing the list of words for each document in the corpus.

tag_files(file_list, tagger)

Given a list with document words for each documents, returns a list with the tagged document words for each document as tagged by the provided tagger.

sort_freq(tagged_words)

Given a list of words that are tagged, returns the same list of tagged words, sorted by how often they occur.

- Tags all words in all files using the Stanford NER tagger
- Expands the nested list of words into one list with all the tagged words
- Determines which tagged words are named entities
- Prints most common locations, organisations, and people from those named entities

U6

scrub.py

Usage: scrub.py file_to_be_cleaned.txt

Removes paragraphs with no mention of earthquakes from the file to be cleaned

U8

fill_template.py

most_common(lst)

Given a list, returns the element that occurs the most often.

find_magnitude(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in contexts corresponding to magnitude descriptions, using regex.

find_deaths(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in contexts corresponding to number of deaths descriptions, using regex.

find_injuries(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in

contexts corresponding to descriptions of the number injured, using regex.

find_epicenter(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in contexts corresponding to descriptions of the epicenter location, using regex.

find_date(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in contexts corresponding to descriptions of the date the event occurred, using regex. The date object returned is a 3-tuple of (day, month, year)

find_time(raw_collection)

Given the raw text of a collection describing an earthquake, determines the value that occurs most often in contexts corresponding to descriptions of the time the event occurred, using regex.

find_word(word, path_to_corpus, cutoff)

Given a word, the path to the corpus root directory, and a certain cutoff value (0-1), determines whether the fraction of files that the word occurs is higher than cutoff.

earthquake_template(corpus, path_to_corpus=None)

Given a corpus, and optionally the path to the corpus, determines various characteristics of the corpus using regexes, and returns a Python dictionary containing these values.

u8_execution.py

- Imports big collection and small collection
- Creates and prints the earthquake template for each

U9

summary.py

summarize(template)

Given a template (of the form returned by the function *earthquake_template* in U8), constructs a paragraph summary of the qualities of the earthquake.

u9_execution.py

- Loads in Big and Small collections
- Generates and prints a summary of each

Problem Statement

For this course, we were tasked with creating the means to automatically generate a short summary of a corpus of articles about earthquakes. Such a summary should be best representative of the texts and include all relevant information about earthquakes. For our analysis, we operated on two corpora--one about a 5.8 magnitude earthquake in Virginia in August 2011 (referred to as our “Big” collection), and another about a 6.6 magnitude earthquake in April 2013 in Lushan, China (referred to as our “Small” collection).

Outline of Approach

In computational linguistics, text summarization is the process of identifying the most critical pieces of information in natural language text and combining them to form an informative and concise summary.

An important part of text summarization can be lemmatization. Lemmatization is the process of grouping together different forms of a word which share the same linguistic lemma when considering them for analysis. This makes word lemmas with a high number of alternative forms be represented in the same proportion as word lemmas with a low number of alternative forms.

Additionally, part of speech tagging allows for more in-depth text analysis. Backoff tagging is a method used to tag words with their parts of speech for further analysis. Backoff tagging works by using multiple methods of tagging, starting with the most accurate but less applicable and shifting to less accurate but more applicable methods in order to tag each word with the highest accuracy possible.

To extract relevant pieces of information from natural language, one method commonly used is k-Means clustering. Clustering is a technique that starts by defining identifiable features of the text such as words or n-grams and the objects to be clustered, such as sentences or documents. Once a feature set is defined, the number of clusters must be decided upon; the larger the variation in the text, the more clusters that are typically chosen. The centroid of a cluster contains its most relevant features. The clustering process begins by defining the centroids of these clusters in a random manner. After the centroids are defined, which objects belong in which clusters is determined by finding the cluster to which each object is closest. After the objects have been assigned to a cluster, the features in the centroids of the clusters are recalculated to be the average feature set of the objects in the cluster. An iterative process of defining which objects belongs to which cluster and defining the centroids based upon the average feature set of the clusters continues until the process yields no additional reassignments. Clustering sentences of a corpus and then choosing sentences closest to a particular centroid can be used to extract sentences about a particular topic.

Regular expressions are a way to ambiguously define a string. Regular expressions can be used to search for strings containing specific words or pieces of information in order to obtain better results than a traditional exact search query.

Details of Approaches Taken

Frequency Analysis

Naive frequency analysis of the texts yields meaningful but noisy results. As shown in Table 1, the nine most common words of our Lushan earthquake corpus have little to do with the earthquake.

Word	the	in	to	of	and	a	s	on	for	china	earthquake
Frequency	14092	8953	8372	7339	6463	5441	4048	3677	3005	2719	2649

Table 1. Naive Frequency Analysis of Lushan Earthquake

To reduce the amount of noise in the frequency analysis, we filtered the results using a list of stopwords from the Natural Language Toolkit (NLTK). This stopword list contains over 2,400 words which provide very little contextual value and are present in natural language from a wide range of topics. The results of the most common words after this filtration are shown in Figure 2.

Word	china	earthquake	news	quake	sichuan	people	said	us	sikkim
Frequency	2719	2649	2310	2153	1482	1380	1339	1083	982

Table 2. Filtered Frequency Analysis of Lushan Earthquake

Filtered frequency analysis provides meaningful information about the contents of the corpus, however it doesn't provide the context of the information. To provide context to the information, we utilized NLTK to create a backoff part-of-speech tagger (Bird, "NLTK Processing with Python"). This tagger analyzes the n-grams of the word to determine its part of speech. The first layer of the backoff tagger analyzes the trigrams, followed by bigrams, and finally unigrams. If n-gram analysis isn't able to determine the part-of-speech of the word, the tagger backs off to using custom regular expressions to match common patterns in words such as *-ing* being indicative of the word being a gerund. If all else fails then the word is categorized as a noun, by default. From the tagged data, the most notable are the nouns, followed by the verbs. Table 3 shows the most common nouns in the Lushan earthquake.

Word	earthquake	news	quake	sichuan	people	comment	sikkim
Frequency	2717	2281	1965	1473	1382	952	948

Table 3. Lushan Earthquake Most Frequent Nouns

Corpora Cleaning

From the frequency analysis, the Lushan earthquake corpus has produced promising results, however the most popular nouns from the Virginia earthquake corpus consisted of *ad, blog,* and *business*. From these results, it was determined that our corpora were very noisy and contained information not related to earthquakes. To begin cleaning the corpora, trivially poor documents were omitted. Any documents shorter than a couple sentences were removed since most consisted of only noise. Then documents not even mentioning the word earthquake were removed as being completely off topic.

To aid in cleaning the corpora, a classifier was created to classify documents as relevant and irrelevant. To produce the classifier, we manually looked at a sample of 150 documents from the corpora and labeled them as relevant or irrelevant. We then trained three types of classifiers on the set of 150 files to recognize the relevancy of the file based on the presence of distinguishable words in the documents. Once the classifiers were trained, they were run against all documents to determine which files are relevant and which are not. Manual checking of the files showed that the best classifier (Decision Tree) had over 90% accuracy.

Collection	**Files**	**Words**	**Sentences**
YourSmall Raw	454	411,717	17,923
YourSmall Clean	281	103,996	4,094
YourBig Raw	4599	4,103,266	171,525
YourBig Clean	59	40,107	2010

Table 4. Effect of Cleaning on Corpus Size

Examining Named Entities

To consider named entities in our analysis, we made use of open source libraries. NLTK's built in named entity extractor is called *ne_chunk*. Table 5 shows some of the results of NLTK's named entity recognition on our Lushan earthquake corpus.

GPE	China	Chinese	India	Lushan	Sikkim	Boston
Frequency	2682	535	416	405	364	355

Person	Sichuan	Sikkim	Google	Xinhua	Video	Lushan
Frequency	817	384	186	176	133	118

Organization	CNN	Date	YouTube	Reuters	US	RSS	AP
Frequency	205	204	144	131	105	98	95

Table 5. NLTK.ne_chunk Results for Lushan Earthquake.

The global, political, and economic results of NLTK's named entity extractor are accurate and descriptive; however the people and organization results are not representative of the Lushan earthquake. Additionally, we utilized the Stanford Named Entity Recognizer (NER). Table 6 shows the results of the Stanford NER run against the Lushan earthquake corpus.

Location	China	Sikkim	Sichuan	India	U.S.	Province
Frequency	450	244	171	110	35	33

Person	Weibo	Renren	Sina	Kaixin	Li	David
Frequency	36	15	15	15	13	13

Organization	News	Home	Weibo	Business	India	Xinhua	World
Frequency	184	100	69	67	65	62	61

Table 6. Stanford NER Results for Lushan Earthquake

The Stanford NER was able to obtain better information about the location of the earthquake. Sichuan, the third location result, is located in Lushan and was the epicenter of the earthquake. While the Stanford NER was better at recognizing people's names than NLTK, the people extracted are not noteworthy in describing the earthquake. Additionally, the organizations acquired from both named entity recognizers were unhelpful in describing the earthquake.

Clustering

To cluster the text, we decided to use words as features and cluster sentences as the individual objects. After trying different numbers of clusters, we decided on twenty as the optimal number. We utilized Mahout for our k-means clustering which takes advantage of MapReduce for its processing. After manually observing the contents of the clusters, we chose six which contained the most relevant content and discarded the other fourteen as noise.

The first cluster which we chose characterized information about whom was affected by the Lushan earthquake and where the earthquake took place. Words such as *son, wife, hospital, family,* and *evacuated* are used in contexts which can help characterize potential victims. Table 7 contains the Mahout sentence vector which best characterizes the cluster.

Sentence: (30.450599999999998, ['lin:8.709', ' tent:7.292', ' his:7.018', ' son:6.746', ' wife:7.493', ' after:4.002', ' hospital:5.579', ' soon:6.358', ' 37:6.857', ' family:5.624', ' sat:6.645', ' who:4.894', ' zhou:7.844', ' big:5.861', ' chen:6.630', ' before:6.009', ' another:5.713', ' looking:6.818', ' said:3.683', ' tending:9.179', ' were:4.000', ' from:3.580', ' never:6.661', ' farmer:8.016', ' tianxiong:9.690', ' struck:4.948', ' earthquake:3.309', ' old:5.400', ' quake:3.409', ' between:6.135', ' i:4.289', ' ve:6.763', ' stretcher:9.402', ' saturday:4.293', ' day:5.190', ' evacuated:8.150', ' tents:5.854', ' lushan:4.335', ' lying:7.233', ' seen:6.485', ' three:5.574'])

Table 7. Relevant Victim Cluster

The second cluster chosen contains technical information relevant to the earthquake. Words such as *magnitude, geological,* and *aftershock* are contained in sentences which give detailed information about the structure of the earthquake. Table 8 contains the Mahout sentence vector which best characterizes the cluster.

Sentence: (47.557709000000003, [' field:7.323', ' oil:6.837', ' down:5.689', ' largest:6.614', ' later:6.526', ' reuters:5.531', ' revised:8.304', ' telephone:7.323', ' total:6.458',' china:3.364',' magnitude:4.333', ' 100:5.510', ' bifengxia:7.493', ' geological:6.018', ' output:7.744', ' provinces:7.205', ' spokesman:10.356', ' huge:6.471', ' office:6.960', ' safe:8.925', ' sichuan:3.862', ' sinopec:8.224', ' accounts:6.897', ' gas:9.354', ' initially:7.793', ' major:6.001', ' survey:5.976', ' houses:5.516', ' producing:8.486', ' refiner:8.391', ' more:3.521', ' puguang:8.709', ' put:6.645', ' unaffected:8.486', ' aftershock:10.545', ' background:8.016', ' us:4.147'])

Table 8**. Technical Information Cluster**

The third cluster contains information relevant to the amount of damage done including damage to builds and death toll. Table 9 shows the centroid vector and the sentence vector closest to the centroid. It contains words such as *damaged, hospital, died,* and *missing* which give insight into the impact of the earthquake.

centroid: ['also:4.792', ' have:3.759', ' government:7.052', ' hospital:5.579', ' damaged:5.490', ' building:8.331', ' school:6.569', ' buildings:8.784', ' four:6.088', ' all:3.947', ' secretariat:8.391', ' extensively:9.402', ' districts:7.698', ' press:5.590', ' headquarter:8.997', ' been:4.214', ' police:5.372']
Sentence: (14.130080999999999, ['have:3.759', ' died:5.985', ' people:3.668', ' another:5.713', ' 180:8.150', ' least:4.880', ' 24:5.427', ' missing:5.618'])

Table 9. Relevant Damage Cluster

The fourth cluster contains information regarding the location of the earthquake and contains words such as *epicentre, hits,* and *outskirts* which are typically accompanied by specific location information. Table 10 contains the Mahout sentence vector which best characterizes the cluster.

Sentence: (49.570920999999991, ['ago:5.943', ' epicentre:6.173', ' chengdu:5.409', ' morning:5.671', ' city:4.775', ' miles:8.511', ' outskirts:8.843', ' around:5.585', ' china:5.827', ' capital:5.547', ' close:5.505', ' its:4.792', ' magnitude:4.333', ' authorities:5.968', ' people:3.668', ' hits:5.579', ' province:4.253', ' western:6.897', ' rise:6.358', ' from:3.580', ' shallow:6.857', ' south:5.451', ' co:5.869', ' likely:6.370', ' over:4.594', ' struck:4.948', ' 7:4.319', ' area:4.755', ' earthquake:5.732', ' quake:3.409', ' sichuan:6.689', ' 8:4.677', ' casualties:6.358', ' uk:6.419', ' injured:6.111', ' least:4.880', ' provincial:6.097', ' depth:5.891', ' number:6.116', ' saturday:4.293', ' 150:9.133', ' county:4.651', ' telegraph:7.611', ' lushan:4.335', ' roughly:9.179', ' killed:4.660', ' more:4.980', ' magnified:8.016'])

Table 10. Relevant Location cluster

The fifth cluster contains information regarding the time of the earthquake. Words such as *gmt and pm* are used exclusively with time which is usually accompanied with information about the date. Table 11 contains the centroid vector of the cluster.

centroid: ['epicentre:4.116', ' local:1.743', ' s:1.880', ' 68:7.698', ' gmt:2.190', ' 41:2.190', ' pm:1.734', ' time:1.555', ' capital:5.547', ' sikkim:4.298', ' gangtok:5.720', border:2.026', ' 7:2.880', ' being:1.849', ' earthquake:1.103', ' india:1.438', ' occurred:2.100', ' quake:1.136', ' located:4.685', ' depth:3.927', ' epicenter:1.865', ' km:7.793', ' nepal:1.862', ' northwest:7.456']

Table **11. Relevant Time Cluster**

The sixth cluster contains contains information about the reason why and how the earthquake occurred. Table 12 contains the Mahout sentence vector which best characterizes the cluster. Words such as *tectonic* and *movement* usually are accompanied with context as to what caused the earthquake.

Sentence: (67.634175999999997, ['earthquakes:6.106', ' movement:8.391', ' seems:7.611', ' tectonic:7.955', ' build:7.898', ' causing:7.075', ' plate:8.591', ' up:4.362', ' stress:8.224'])

Table 12. Circumstantial Cluster

Compared to other approaches (units), clustering did not yield particularly accurate results, nor was the process as intuitive as others. The clusters listed above were our best results, but are in no way indicative of the average value of the clusters we received. Many clusters were not distinguishable in terms of relevance, rendering the clustering useless at times.

In regards to timings and speedup of clustering, we noticed that both our big collection and small collection took similar times to run as demonstrated in Table 13 below.

Big Collection Clustering Timing (minutes)	Small Collection Clustering Timing (minutes)
5.17535	5.1245

Table 13. Clustering Timings

After consulting the TAs about this phenomenon, we learned that this was most likely due to the actual processing time being insignificant compared to the overhead of running the clustering. This also implied that the size of our collections was too small to gain a significant advantage from utilizing the Hadoop cluster.

Regular Expressions

Our next approach was to construct a template containing critically important information for earthquakes. After some research into what factors are most important for earthquakes we appraised and used the template categories described in Table 14 (Jones, A Parent's Guide To Earthquakes). We decided that this template would be filled using regular expressions combined with frequency analysis. Regex searches were constructed by combining the results from various regexes that aimed to find relevant data that was surrounded by certain key indicators. See Table 14 for the regex searches used.

Category	**Regex(s) used**
Magnitude	'magnitude (?:of)?[0-9]\.?[0-9]*' '[0-9]\.?[0-9]* magnitude'
Deaths	'(?:[0-9]+ deaths)' '(?:[0-9]+ were killed)' 'killed (?:\w+){0,2}[0-9]+ people' '(?:[0-9]+ casualties)'
Injuries	'[0-9]+ injur(?:(?:ies)|(?:ed))' '(?:[0-9]+ were injured)' 'injured (?:\w+){0,2}[0-9]+ people'
Location	'((in|at)\s([A-Z][a-zA-Z]{4,}|[A-Z][a-zA-Z]{2,}\s[A-Z][a-zA-Z]{3,})|\s+[A-Z][a-zA-Z]{3,},\s[A-Z][a-zA-Z]{2,}\s[A-Z][a-zA-Z]{3,})'
Epicenter	'((in|at)\s([A-Z][a-zA-Z]{4,}|[A-Z][a-zA-Z]{2,}\s[A-Z][a-zA-Z]{3,})|\s+[A-Z][a-zA-Z]{3,},\s[A-Z][a-zA-Z]{2,}\s[A-Z][a-zA-Z]{3,})' NOTE: This regex is only applied to sentences containing the word 'epicenter'
Date (Month)	'(?:January|February|March|April|May|June|July|August|September|October|November|December)'
Date (Day)	'(?:January|February|March|April|May|June|July|August|September|October|November|December),?\s([0-9]{,2})[^0-9]'
Date (Year)	'(?:1|2)[0-9]{3}'
Time	'[0-9]{1,2}:[0-9][0-9](?::[0-9][0-9])?(?:\s?[apAP]\.?[mM]\.?)?'

Table 14. **Regex Searches**

All occurrences where appropriate data was found near an indicator were recorded, and the average and mode were determined for numerical data types (number of deaths, magnitude, etc.) We reported final values using the mode, since high frequency was our best indicator of accuracy.

Finding the mode of non-numeric data distributions also resulted in extremely accurate templates containing the relevant information from the corpus. The templates created are shown in Table 15. One thing to note is that while our results were extremely accurate, and the process was extremely fast to run, setting up correct regexes requires careful engineering. Some of our solutions were nested regular expressions which required multiple iterations of development before we found a combination that provided the best results. As such, these expressions may not be as flexible as our solutions for other units. The accuracy and speed obtained from this technique comes with a loss of flexibility; this solution, unlike the vast majority of the other approaches to extracting information from the corpus, can not be readily applied to a corpus with a different subject matter.

Big Corpus Earthquake Analysis:
Deaths: 140.0
Epicentre: Louisa
Magnitude: 5.8
Location: Virginia
Time: 1:51
Date: (u'23', u'August', u'2011')

Small Corpus Earthquake Analysis:
Deaths: 186.0
Epicentre: Lushan
Magnitude: 6.6
Location: Lushan
Time: 8:02
Date: (u'20', u'April', u'2011')

Table 15. **Template Results**

Results

Our final step was to construct a way to transform this simple template into a dynamic, grammatically correct English report. We wrote a Python script to analyze the contents of the template and built this report accordingly. We used the structure shown in Table 16, and our results are detailed in Table 17.

summary = 'On {0} {9}, {10} at {1}, a {2} magnitude earthquake struck {3}. The epicenter of the quake was located at {4}. There {5} aftershocks that followed the earthquake and {6} tsunami was caused by the earthquake. There {7} reports of landslides due to this earthquake. A total of {8} deaths occurred.'.format(template['date'][0], template['time'], template['magnitude'], template['location'], template['epicentre'], aftershock, tsunami, landslide, int(template['deaths']), template['date'][1], template['date'][2])

Table 16. Structure for Summary

YourBig**:**
On 23 August, 2011 at 1:51 p.m., a 5.8 magnitude earthquake struck Virginia. The epicenter of the quake was located at Louisa. There were aftershocks that followed the earthquake and no tsunami was caused by the earthquake. There are no reports of landslides due to this earthquake. A total of 140 deaths occurred.
YourSmall**:**
On 20 April, 2013 at 8:02 a.m., a 6.6 magnitude earthquake struck Lushan. The epicenter of the quake was located at Lushan. There were aftershocks that followed the earthquake and no tsunami was caused by the earthquake. There are reports of landslides due to this earthquake. A total of 186 deaths occurred.

Table 17. Final Summaries

We were very pleased with the quality of our results. We verified the accuracy of each report with our own research of the earthquakes. This result was by far the most comprehensive and accurate summary and it can be viewed as our crowning achievement for this course.

Potential Improvements

In regards to our most successful approach of filling a template using regular expression analysis and then constructing an English report based on this template, there are a few improvements that we would have liked to have made given more time. First of all, having an expanded template containing more useful earthquake information would enable an even more descriptive report to be constructed. In addition, combining the strategy of using regular expression with other techniques such as named entity recognition could improve our solution.

Spending more time with clustering also has the potential to produce valuable results. We encountered many obstacles during this phase of development and as a result, our solution was not as well written and useful as it could have been. We noticed that other groups had more success with this approach so it would be a good idea to revisit this strategy in the future.

It would be helpful to improve upon the way that the collections are retrieved before being analyzed by our system. Implementing a way to collect only relevant articles from the web would help our system focus on the textual analysis and it would allow the system to avoid periodic collection cleaning that needed to be performed. Further, it would be interesting to perform the strategies of the earliest units again on final, cleanest collections that we were using as input by the final units since it is likely that more useful results would be given by our solutions.

Lessons Learned

We learned early on that analyzing a “noisy” corpora and expecting meaningful results is unreasonable. At the beginning of most units, we made an effort to make our collections cleaner. We found that this greatly improved our results. In our first few units, we likely had false positives and false negatives due to the nature of our collections. However, as we progressively cleaned our corpora, we found that our solutions were more reliable.

Over the semester, we encountered many new technologies that were useful when developing solutions for this project. We found that approaching new technologies in an intelligent way is critical to efficiently learning and applying them. We typically approached the technologies in four distinct phases: Familiarize, Explore, Practice, and Utilize. In the first stage, we researched the tool and learned its most distinguishable properties. We then explored the technology in a more detailed way, examining its intricacies and beginning to take into account the problem we were trying to solve. We then practiced using this technology to solve simple related problems. For example, we completed some NLTK exercises in the course textbook when learning that technology. Finally, we applied the techniques we learned to the problem at hand. We found that this four stage approach increased our efficiency when learning new technologies.

We also realized the importance of parallelization in two areas of development. First, we found it quite important to split up work between our group members in a fair and efficient way. This required careful analysis of the driving question and developing an approach that could be broken into mutually exclusive parts to facilitate parallelization. In some units we also parallelized the execution of our solution. We accomplished this by using the MapReduce technology on a Hadoop file structure.

Similar to parallelization, documentation and reuse of code greatly increased our efficiency. Due to the incremental nature of our units, we quickly learned code reuse was invaluable. This allowed us to focus on new and improved aspects for solutions, building on old foundations. Code reuse also allowed us to have multi-level solutions which we found to be more reliable than single strategy approaches.

The combination of the above high level approaches to this course's driving question allowed us to implement a working solution which provided a meaningful summary of an earthquake event. The lessons we learned from this project will be immediately applicable to any design or programming projects that we encounter in the future.

Acknowledgements

We would like to thank Dr. Edward Fox (fox@vt.edu) and the rest of the CS4984 course instructors for creating and maintaining a detailed wiki which greatly aided learning and development:

- Xuan Zhang (xuancs@vt.edu)
- Tarek Kanan (tarekk@vt.edu)
- Mohamed Magdy (mmagdy@vt.edu)

We would also like to thank our sponsors:

- NSF DUE-1141209
- IIS-1319578

Finally, we would like to thank our classmates for continuously providing constructive feedback and support.

References

Bird, Steven, and Ewan Klein. Natural Language Processing with Python. Beijing: O'Reilly, 2009. Print.

Jones, Dr. Lucy. "Parent's Guide - A." *A Parent's Guide to Earthquakes*. U. S. Geological Survey. Web. 12 Dec. 2014. http://earthquake.usgs.gov/learn/kids/abc/parents/pa.html.

"Natural Language Toolkit." *Natural Language Toolkit — NLTK 3.0 Documentation*. NLTK Project, 21 Aug. 2014. Web. 8 Dec. 2014. http://www.nltk.org/

"Python 2.7.9rc1 Documentation." *Overview*. Python Software Foundation, 7 Dec. 2014. Web. 8 Dec. 2014. https://docs.python.org/2/.