Summarizing Fire Collections with Natural Language Processing

Michael Zamani, Hayden Lee, Michael Trujillo, Jordan Plahn

CS 4984: Computational Linguistics
Virginia Tech
Blacksburg, VA
Dr. Edward Fox
December 8, 2014

Table of Contents

Executive Summary

Throughout this semester, we were driven by one question: how do we best summarize a fire with articles scraped from the internet? We took a variety of approaches to answer it, incrementally constructing a solution to summarize our events in a satisfactory manner.

We needed a considerable amount of data to process. This data came in the form of two separate corpora: one involving the Bastrop County, Texas wildfires of 2011 and the other the Kiss nightclub fire of 2013 in Santa Maria, Brazil. For our “small” collection, the Texas wildfires, we had approximately 16,000 text files. For our “large” collection, the nightclub fire, we had approximately 690,000 text files. Theoretically, each text file contained a single news article relating to the event. In reality, this was *rarely* true. As a result, we had to perform considerable preprocessing of our corpora to ensure useful outcomes.

The incremental steps to produce our final summary took the form of 9 units to be completed over the course of the semester, with each building on the work of the previous unit. Owing to our lack of domain knowledge at the beginning of the semester (with either fires or natural language processing), we were provided considerable guidance to produce naive, albeit useful, initial solutions. In the first few units, we summarized our collections with brute force approaches: choosing the most frequent words as descriptors, manually generating words to describe the collection, selecting descriptive lemmas, and more. Most of these approaches are characterized by arbitrarily selecting descriptors based on frequency alone, with little consideration for the underlying linguistic significance.

From this, we transitioned to more intelligent approaches, attempting to utilize more fine grained techniques to remove extraneous information. We incorporated part-of-speech (POS) tagging to determine the speech type of a word, which allows us to select the most important nouns, for example. Using POS tagging, as well as an ever expanding stopword list, allowed us to remove much of the uninformative results. To further improve our collection, we needed a way to filter out more than just stopwords. In our case, we had many text files that were unrelated to our corpora topics, which could corrupt or skew our results. To accomplish this, we built a document classifier to determine if articles are relevant and mark them appropriately, allowing us to include only the relevant articles in our processing. Despite this, our collection still suffered from considerable noise.

In almost all of our units we employed various “big data” techniques and tools, including MapReduce and Mahout. These tools allowed us to process extremely large collections of data in an efficient manner. With these tools we could select the most relevant names, topics, and sentences, providing the framework for a summary of the entire collection. It is these insights that lead us to the final two sections of producing a summarization based on preconstructed templates of our event. Using a mixture of every technique we had learned we constructed paragraphs that summarized both fires we had in our collection.

For the final two units of our course, we were tasked with creating a paragraph summary of both the Texas Wildfire and the Brazil Nightclub Fire events. We began with a generic fire event template with a set of attributes that would be filling in with the best results we could extract. We made the decision early on to create separate templates for the more specific fire event types of wildfires and building fires, as there are some details which do not overlap among the two event types. In order to fill in our templates we created a process of extracting, refining and finally filling in our gathered results. In order to extract data from our corpora, we created a regular expression for each attribute type and stored any matches found. Next, using only the top 10 results for each attribute, we filtered results by parts of speech, constructed a simple grammar to modify the template according to our selected result, and conjugated any present tense verbs to past tense.

User's Manual

Requirements

To run the code in this project you will need the following items:

- A collection of documents that you want summarized
- The codebase of the project, which can be found at this url: https://github.com/mzamani1/NLP-Capstone
    - Code can be obtained by running the command (with Git installed):
      `> git clone git@github.com:mzamani1/NLP-Capstone.git`
- An understanding of basic Python scripting
- Access to a Hadoop cluster

Data Analysis

The codebase is organized into nine units representing the nine units completed in this course. Each unit comprises of various python scripts. In general, the scripts named mapper.py or reducer.py are scripts to be run on your Hadoop cluster with data stored on your cluster. Other scripts are to be run locally with local datasets.

Each Python script that acts on local data will use a variable to specify where your data is located. On a per-script basis this variable will need to be changed to point to the parent directory of your collection. Once this has been changed you can run the script.

To run each script locally enter the following command into a terminal (where *script_name.py* is the name of the script you want to run):

```
> python script_name.py
```

- The mapper.py and reducer.py files.
- A zipped archive of your corpora.
- The nltk.mod file (if you intend to use any NLTK functionality)
- An nltk_data.zip file (see repositiory)
- (Optional) An input.txt file containing a list of all filenames in your corpora.

Once these files are on the cluster, you will need to transfer your zipped corpora and input text file to the Hadoop file system using the following commands:

```
> hadoop fs -put archive.zip
> hadoop fs -put input.txt
```

When you are ready to run MapReduce, use the following command, replacing parameters with your own file names:

```
hadoop jar
/opt/cloudera/parcels/CDH/lib/hadoop-0.20-mapreduce/contrib/streaming/hadoo
p-streaming-2.2.0-mr1-cdh5.0.0-beta-1.jar -mapper mapper.py -reducer
reducer.py -input input_files/ -output outputDirectory -file mapper.py
-file reducer.py -file nltk.mod input.txt -cacheArchive
nltk_data.zip#nltkData -cacheArchive input_files.zip#input_files
```

Finally, depending on the `-output` parameter specified, you may fetch the results of your MapReduce job from the Hadoop FS by using the following command:

```
> hadoop fs -get output_folder_name
```

Developer's Manual

Github Repository

All code that was produced during the semester can be found at https://github.com/mzamani1/NLP-Capstone. The code is organized into the nine units that we worked on throughout the semester, as well as directories containing the text files that were deemed relevant by the classifier. All the code necessary to reproduce our results is available at the above link.

Dependencies

All of the code that we wrote was in Python, version 2.7.x. As a result, you will need to install the appropriate version of Python on your machine (Python 3.x will not work as expected).

We also make heavy use of NLTK, the Natural Language Toolkit built with Python. It will also need to be downloaded and configured. In addition, when running MapReduce jobs, we make use of the nltk.mod file that can be found on GitHub. There are further MapReduce dependencies that are listed within the appropriate Python files; most of them can be downloaded by the same name that appears in the import statement.

Hadoop

To run our MapReduce jobs, we heavily rely on the Apache Hadoop ecosystem. MapReduce specifically will be explained in more detail in the next section. For all of our computations, we run them on the Hadoop Distributed File System (HDFS), which stores data across machines, giving us extremely high-bandwidth access to them. All of our files for processing are placed on the Hadoop file system.

MapReduce

A large number of the computations we performed this semester relied on Hadoop MapReduce. MapReduce allows us to perform computations on large data sets because it relies on highly scalable, parallel algorithms. It utilizes a large number of computers/nodes (called a cluster), where the data to be processed is split into a number of chunks that are each sent to a different node. Each node then performs the "map" step, which applies some function to the data and then outputs it in some form to be redistributed during the "shuffle" step. From this, the data is fed to the "reduce" step, transforming it into the final output, which in our case was typically a count of the frequency of some phrase or word.

On the GitHub repository is a file mapreduce.md that contains the command we used to run our MapReduce jobs, as well as other explanations that might be useful.

Collection Cleaning

Many of our scripts for cleaning files and collection, including our classifier, are in the directories for the earlier units. They are all marked and include comments explaining what they accomplish. They can be modified for use with other collections that may need cleaning.

Summary of Results

Throughout this semester, we had the benefit of being provided a schedule that we had to follow. This made deadlines very clear and allowed us to structure our plan so that we hit all of the targets and developed our solution in an iterative manner. Below we present the schedule broken up into units with their approach, deliverables, outcomes, and difficulties described.

Unit 1

Driving question: *What is the best way to summarize with words or phrases considering frequencies?*

The first unit employed a very naive approach to summarize a text provided in the NLTK package, as well as the Class Event (flooding in Islip, NY). These collections served as practice since they are small and can be processed quickly. To summarize, we iterated through the articles in the collection and created a dictionary of each word and collocation that appeared, accumulating the count of each one. We placed arbitrary restrictions on the length and type of the word to filter out unnecessary results, like pronouns or stand alone numbers. For the Class Event, we took the top 10 results and found them to be good descriptors of the collection. Our top 10 words (and their corresponding frequency) were:

> [('Island', 101), ('Weather', 72), ('inches', 66), ('flooding', 56), ('August', 54), ('rainfall', 48), ('Wednesday', 39), ('record', 37), ('National', 30), ('flooded', 28)]
> Ten most frequent words in Class Event.

To finish, we brainstormed words that may describe our collection and looked them up in Solr to determine their frequency in the collection.

One thing we noticed is that we needed to find an effective way to filter out stopwords. By removing these we would not have to place a restriction on the length of the word, which may filter out short but descriptive words like “fire.”

Unit 2

Driving Question: *What is the best way to summarize with corpus characteristics, including synsets, with aid for scalability from Hadoop?*

To start Unit 2 we repeated the same steps that we performed in Unit 1 on our small collection, relating to the wildfire in Bastrop, Texas. We manually generated a list of words that we believed would be descriptive of our collection, listed below.

> *Your Words* = [Fire, Wildfire, Water, Damage, Ground, Burn, Town, Wind, Speed, Size, City, People, Home, Weather, Debris, Death, Smoke, State, Ash]
> Ten most frequent words in our small collection.

We believed these were representative words for any wildfire. To justify our beliefs, we determined the frequency of each word in our collection, which we could then compare against the frequency percentage of the words in other collections, indicating whether our words are or are not good descriptors of our collection. What we found was that our chosen words *did* occur more often in our collection than in a generic corpora, e.g. the Brown corpus, which lends credence to the belief that we will be able to describe our collection uniquely.

We performed further analysis on our small collection to get an understanding of the syntactic makeup of the documents. We found that just over 25% of all words in the collection are stopwords, indicating that effective stopword filtering is vital. Regarding the length of words in the collection, the average word was about 4 characters long, which we did not find to be statistically significant in subsequent units.

Finally, we utilized NLTK's built-in WordNet database that allowed us to utilize the synsets(*word*) function, which returns a list of all synonyms for the *word* argument that it accepts. We anticipated utilizing this to match synonyms for words like fire or damage when constructing a summary, but we found that it was often unnecessary.

As for challenges we faced, we noticed that when getting the list of synonyms for words, many of the results, and many of the top results, were words that we could not use. For example, the top synonym for "people" was "citizenry", but we couldn't anticipate any situations where a news article would use that word. So we came to believe the significant use of synonyms may be wasted effort when a large portion of the root words were irrelevant.

Unit 3

Driving Question: *What is the best way to summarize with nouns (and verbs?) by POS (part-of-speech) tagging?*

To begin Unit 3 we created a part-of-speech tagger: an algorithm that reads text and creates a tuple consisting of a word and its part of speech. Using the NLTK toolkit, we were able to expand on a simple single word tagger by creating a trigram tagger which could consider up to three words at a time; this approach was used as it yields more accurate results. Our POS tagger was trained using a combination of the Brown and Treebank tagged corpora bundled in NLTK.

Using the trigram tagger, we then found the most frequent nouns and verbs. The tagged text allowed us to filter out other parts-of-speech which include words ‘like’,’a’, and ‘the’, so a list of stopwords wasn’t needed. Verbs were included in our results as we believed that they can be descriptive.

In this unit we also started cleaning our collection. We kicked this off by removing all files under 5KB in the small collection and files under 1KB in our large collection. These sizes were chosen as most of the files removed contained website metadata, foreign language texts, and other irrelevant data. In our small collection we also removed all files where the term fire did not appear, again for the reason that the removed files were irrelevant.

Unit 4

Driving Question: *Improve summaries by cleaning up your corpus: classifying to eliminate noise, including n-grams as features.*

The first part of this unit was focused on creating a list of most frequent bigrams. To create a list efficiently on our entire collection, we began using MapReduce on the Hadoop cluster. To minimize wasted runtime on the cluster we first made an algorithm to find bigrams in our small collection and once we were confident with this algorithm we then ran it on our large cluster. This resulted in a list of 50 bigrams that represented the corpus, part of which can be seen below (Fig. 1). The full list is available in Appendix A.

Bigram	Frequency
fire department	594
rick perry	507
state park	435
officials said	377
destroyed home	341

Fig. 1: Five of the fifty bigrams

Along with finding bigrams we began creating a classifier with the aim of further cleaning up our corpus. We created a list of 300 manually classified documents that contained even proportions of positively and negatively tagged documents. Using this tagged collection we were able to extract a set of features (Fig. 2) that could be used to classify further documents using the NLTK library's built in "most informative features" algorithm.

Feature	Odds (True : False)
flames	21.3 : 1.0
fires	20.1 : 1.0
burned	20.0 : 1.0
firefighting	16.4 : 1.0
acres	15.3 : 1.0
drought	15.2 : 1.0
evacuate	12.8 : 1.0
evacuated	12.6 : 1.0
burn	12.3 : 1.0
wildfires	12.0 : 1.0

Fig. 2: The top ten features found in our small collection.

Using this feature set we experimented with three classification algorithms:

- Decision Trees
- Naive Bayes
- Maximum Entropy

The maximum entropy algorithm resulted in the best classification which we were able to test by running each algorithm on a test set. However, when adapting the classifier for our big collection we had to use the Naive Bayes classifier, largely due to technical difficulties with implementing maximum entropy on the Hadoop cluster (certain dependencies were not present on Hadoop, making it impossible to run the classifier). Once we were confident with our classifier we used it classify our small and large collections. The results of the classifiers are presented below:

Fold #	Maximum Entropy	Decision Tree	Naive Bayes
1	0.78	0.80	0.68
2	0.98	0.95	0.95
3	0.87	0.88	0.87
4	0.97	0.93	0.90
5	0.77	0.80	0.77
Overall	~0.874	~0.872	~0.834

Fig. 3: Classifier results

Unit 5

Driving Question: *What is the best way to summarize with names and entities?*

This unit involved experimenting with new techniques, namely finding named entities and chunking. We took advantage of the NLTK library's part-of-speech tagger and chunking algorithms. By tagging the corpus as named entities, we were able to work out that organizations, persons, locations, and dates were the four most important entities to examine given our topic of fires. We concluded this because we received either no results tagged as money, time or percent, or results that provided very little value and context, whereas those tagged as the four aforementioned named entities provided useful information even without context.

We used the Stanford Named Entity Recognition (NER) tagger on all of our collections. The Stanford NER tagger seemed to accurately classify named entities for the most part, with some notable exceptions, such as Obama being classified as an Organization, and Yahoo as a Person. Also, for our large collection, the most frequent named entities that we received were not very indicative of our collection, which likely has more to do with the quality of our corpus than the tagger itself. For our small collection there were a reasonable amount of relevant named entities in our top 40 most frequent words, such as Texas, Bastrop, Austin, and Fire.

Once we had the most common named entities we tested whether or not they were indicative of our corpus by using them to search on Solr, and evaluating the results of the search by their relevancy to our topics. Via this method we found that the top 10 words produced very relevant results, and the quality degraded after those as we moved down the list.

Unit 6

Driving Question: *What is the best way to summarize with topics?*

Unit 6 introduced two new concepts: Mahout and Latent Dirichlet Allocation (LDA) to help with summarizing with topics.

We originally performed the experiments with Mahout using a subset of our small collection that we had previously classified as relevant. In the case of classification, that meant the file contained some number of keywords that we had identified in a previous unit. The issue with this approach, especially when it came to our experiments with Mahout, is that a significant portion of the file was noise which ruined our results. So when we ran the experiment with our unmodified collection of relevant files, none of our keywords were listed among the top probabilities.

To remedy this, we tried to emulate the Class Event collection (i.e. construct a small number of extremely relevant articles), which had given us very good results. We took 15 files that had been classified as relevant, removed all of the noise (i.e. anything but the actual article), and then ran Mahout on it again. This yielded much better results, with words like fire, Bastrop, and wildfire achieving a high probability.

We also used the Python Gensim library to perform LDA. This worked great, providing topics that were very relevant to our corpus. Here are the top three topics for our small, 15 file collection:

> *Topic 1* : 0.006*texas + 0.005*news + 0.005*fire + 0.005*2011 + 0.003*ago + 0.003*us + 0.003*new + 0.003*people + 0.002*1 + 0.002*said
>
> *Topic 2* : 0.017*fire + 0.006*2011 + 0.005*september + 0.004*texas + 0.004*news + 0.003*us + 0.003*2010 + 0.003*firefighter + 0.003*firefighters + 0.003*new
>
> *Topic 3* : 0.006*news + 0.005*2011 + 0.005*fire + 0.005*september + 0.005*texas + 0.004*new + 0.003*us + 0.003*ago + 0.002*home + 0.002*said
>
> Three most frequent topics in our small collection.

The above results list the top 3 topics that describe our articles. As you can see, the major topics all include fire, the month the event occurred (September), and various other useful descriptions of the event.

Unit 7

Driving Question: *What is the best way to summarize with indicative sentences?*

Before we began Unit 7 we had to re-familiarize ourselves with Apache Mahout, the free implementation of scalable machine learning algorithms. We used Mahout for k-means clustering: the grouping of object sets similar to each other based on a feature. For our purpose we could use clustering to form k groups from our articles and from these k clusters obtain objects closest to the center.

In our implementation, we created a set of sentences we deemed relevant by checking if they contained at least one of our feature words. We then fed this sentence set to k-means, with k = 30 for our small collection and k = 65 for our big collection. Next we obtained the sentence closest to each cluster center and filtered out irrelevant clusters by checking if a sentence contained a feature or not. Our top sentences for each collection were then chosen manually.

> *['state:3.629', ' counties:4.476', ' bastrop:2.630', ' erupted:4.476', ' than:3.377', 'wildfires:3.629', ' largest:3.965', ' several:3.965', ' thousands:3.629', ' 40:4.476', 'houston:4.188', ' miles:3.629', ' some:3.272', ' 000:4.188', ' four:3.783', ' 1:3.783', 'days:3.629', ' people:4.943', ' strong:4.476', ' temperatures:4.476', ' destroying:5.923', 'spawned:4.476', ' winds:3.783', ' mix:4.476', ' about:4.256', ' 700:3.965', ' area:3.495', 'drought:4.476', ' nearly:4.188', ' perilous:4.476', ' burned:3.377', ' fire:1.725', ' forcing:5.923', ' straddling:4.476', ' historic:4.476', ' 22:4.476', ' blaze:3.629', ' evacuate:5.350', ' has:3.010', ' rural:4.476', ' says:3.783', ' 60:4.476', ' homes:3.074', ' killing:4.476', ' northwest:4.476', 'hot:3.783', ' service:3.965', ' three:3.629', ' week:3.629', ' acre:4.188', ' forest:3.495', 'more:3.010', ' 190:4.476']*
> A result from k-means using our Small Collection.

While our results seemed to accurately describe our event, there are a couple of areas that could have been improved if we had more time. One area is the sentence filtering after clustering. We just checked if a sentence contains a feature, but a better method would be to run a classifier on each sentence like we did in Unit 4. The other area that needed improvement was our final sentence selection, where we chose sentences manually. The solutions we brainstormed were to order sentences based on the date of the article, or by the article's source. Unfortunately these ideas would rely on parsing the proper date while excluding extraneous dates, or requiring an article's source to appear somewhere in the text.

Unit 8

Driving Question: *What is the best way to summarize with a filled-in template?*

With this unit we had a an answer to the driving question for the class: What is the best summary that can be automatically generated for a collection? In this unit we were tasked with obtaining the information needed to complete a template that was given to us:

> *On* <start time> *there was a fire started by* <cause> *in* <geographic location>*. This* <fuel> *fire grew to encompass* <area of impact> *causing* <loss of life> *and* <monetary damage> *in damages.* <firefighting measures>*. The fire was extinguished on* <end time> *after burning for* <end time – start time>*.* <closures> *were made unavailable for a period of time due to this fire. Compared to previous fires in the area this fire was* <severity>*.*
> The template supplied to us.

To accomplish our task we scanned our text files while using regular expression patterns to find information for each field, recording and counting each occurrence or a matched pattern. The method is simple, but by using regular expressions we could easily tune the expression to match our needs. Once the text has been scanned, we place the most frequent pattern match from each category and print them out with the template.

> *In* <start time>*, there was a fire started by* <cause> *in* <geographic location>*. This fire, fueled by* <fuel>*, grew to encompass* <area of impact>*,* <damage (land/homes)>*, and* <loss of life>*.* <firefighting measures>*.* <closures> *as a result of the fire. Compared to previous fires in the area this fire was* <severity>*.*
> Modified template used in this unit.

We decided to make a number of changes to our template to improve our summary. We removed two of the fields due to the difficulty of finding the proper information for them. The sentences in the original template were also reworded to match our results better. Although we had difficulty with the fields ‘firefighting measures’ and ‘severity’, we kept them as we were able to extract suitable results.

The following regular expressions were used for our Wildfire template:

Attribute	Regular Expression
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,}|(around|near)(\sS+){1,2}|(\S+\s){1,3}forest|(\S+\s){1,2}*

	(State|National)\sPark
Cause	\S+\s(lightning|drought(s)?|arson(s)?|negligence)
Fuel	\S{3,}\s(temperature(s)?|wind(s)?|brush|tree(s)?)(,(\s\S+){1,2})?
Damage	(damaged|destroyed|burned|desecrated|scorched|engulfed)(\s\S+){1,3}
Loss of Life	(\s[1-9][0-9]{0,2}\skilled)|((killed|killing)\s([1-9][0-9]{0,2}|\S+))|(\s[1-9][0-9]{0,2}(dead|(of\s)?deaths))
Closures	[1-9][0-9]{0,}\s(ranch(es)?|farm(s)?|road(s)?|home(s)?|highway(s)?|freeway(s)?|interstate(s)?)
Area of Impact	(([0-9]{1,3},){0,}[0-9]{1,3}|(\S+\sof))\s(square\smiles|hectares|acres|square\smeters|square\syards)
Firefighting	([0-9]+|(\S+\sof))\sfirefighters
Year	\s\d{4}
Month	(?:January|February|March|April|May|June|July|August|September|October|November|December)

Fig. 4: Regular expression for each attribute

> *In* September 2011*, there was a fire started by* temperatures, strong winds, and a historic drought *in* Bastrop*. This fire, fueled by* hot temperatures, strong winds*, grew to encompass* about 40 miles*,* fire which has burned thousands of acres*, and* has killed 2*.* 400 firefighters*.* Than 1700 homes and forcing some to evacuate *as a result of the fire. Compared to previous fires in the area this fire was* a devastating fire*.*
> The result from our small collection.

While we were happy with our results, there were areas we could improve upon, with the best result not always being grammatically correct. As mentioned above, we had difficulty with the fields ‘firefighting measures’ and ‘severity’. We could not find a common or repeated characteristic or word in sentences detailing firefighting measures. For severity, our trouble lied with how it was typically a subjective phrase, creating an infinite number of possible adjectives we could match. Despite how unscientific the term is, we accepted ‘devastating’ as it gives some description of the fire.

Unit 9

Driving Question: *What is the best way to summarize with an English report based on the filled-in template?*

As you can see from the driving question, this unit builds off of Unit 8. While Unit 8 was more focused on obtaining multiple forms of data, this unit focused on making the generated sentence grammatically correct.

> *In* <start time>*, there was a fire started by a* <cause> (in|at) <location>*. This fire, fueled by* <fuel>*, grew to encompass* <area of impact>*,* <damage land/homes>*, and* (ended up <loss of life> | <loss of life>)*.* <firefighting response> *responded to the wildfire.* <closures> *were affected as a result of the fire.*
> Template used for this unit, built off of the previous unit's template.

Since we were building off of Unit 8, we were able to make improvements on both our data collecting and our printed result. We refined our regular expressions to better match the ideal results for each attribute. We altered our template to accommodate our new regular expressions and to better fit our results. For example, we made a change from "The fire was <severity>" to "Compared to previous fires in the city it was a <severity>." To accompany these changes we also improved on our results processing, like making our results fit better with our template by changing the tense. This was accomplished with the use to NLTK and the third-party library Pattern.

The following Python code excerpt is an example of how we used the Pattern library to conjugate past tense verbs in a sentence:

```
firefightingResult = " ".join([conjugate(word, tense = "past") if tag == 'VBP' else word for word, tag in nltk.pos_tag(nltk.word_tokenize(firefightingResult))])
```

> *In* September 2011*, there was a fire started by a* historic drought in Bastrop*. This fire, fueled by* hot temperatures, strong winds*, grew to encompass* 33,000 acres, burned for several days*, and* ended up killing four*.* 400 firefighters *responded to the wildfire.* 700 homes *were affected as a result of the fire.*
> The result from our small collection.

Similarly, our final result for the Brazil Nightclub Fire event was the following:

> *In* January 2013 *there was a fire started by* indoor fireworks *in* Santa Maria*. This fire, fueled by* ignited foam*, grew* to the size of the building*,* engulfed the club *and* ended up killing 309*.* Firefighters worked to douse a fire at the Kiss Club*.* One exit was made unavailable for a period of time*. Compared to previous fires in the city it was* fast-moving*.*
> The result from our big collection.

Lessons Learned

Our work this semester generated a number of outcomes that will be useful for future work to reduce the number of issues that we faced this semester.

With a large number of techniques, technologies, and topics introduced this semester, including MapReduce, Mahout, LDA, and others, we were constantly faced with having to learn enough about them to work productively. What we found is that we routinely underestimated the amount of knowledge we needed to posses before making progress. For example, when using MapReduce, we tended to follow the provided examples without understanding what made it work effectively. We then spent a considerable amount of time debugging our scripts to make them run on Hadoop, typically because of some important aspect that we were missing. After working through a Udacity course on MapReduce, we noticed the time spent debugging decreased significantly. Since we were constantly fighting against deadlines to produce results, we fell into the same trap of rushing to implementation with Mahout, LDA, and others. We realize now that if we had spent more of that time delving into the theory, we would likely have reduced our total time spent.

As an extension of the above, we determined that our best use of time entailed one or two people on our team attempting to become domain “experts” for any given aspect of a unit, so that they could dive deeper into a particular aspect while the rest of us did the same with the other relevant parts of the unit. This meant that we typically had the same people do MapReduce for each unit to avoid the training efforts that would be required for others to reach the same ability. If any unit disproportionately focused on one technique, then those that weren’t the experts assumed the responsibilities of writing and compiling the reports. We found this particularly effective when we had less and less time to accomplish our goals.

With any technology involving large amounts of data, it is inevitable that a significant amount of time will be spent processing it while the developer waits idily for it to finish. The real time drain comes when a script is run on a huge dataset, processes for several minutes, and then fails because of a bug that could have been easily diagnosed had we run it on a small data set before. This taught us an important lesson: *always* test scripts (particularly MapReduce scripts) on a small, local data set to diagnose any trivial bugs before running them on our large datasets. We started doing this in later units and found it rapidly improved the iterative approach of our programming, while also dramatically reducing our idle time.

Finally, likely the largest takeaway of the semester, regards the composition of our corpus and the articles within it. Without fail we found that the idea of *garbage in, garbage out* held true, unequivocally. Below we present a couple of statistics about our corpora, with the intent of showing just how poor our articles were.

Collection	Avg. # lines per file before cleanup	Avg. # lines per file after cleanup	% Duplicate Files
Small	1001	14	78
Large	3846	56	85

Fig. 5: The first column, from left to right, is the collection represented, the average number of lines per text file before cleaning, after cleaning, and the percentage of our collections that were duplicate files.

As demonstrated, our corpus was, by and large, composed of duplicate files, completely irrelevant files, and files containing a small relevant article, flanked by thousands of lines of junk (i.e. JavaScript code, HTML, links, other articles, etc.). To make sure we got any results of significance, we had to strip out everything but the actual file. Without doing this, the results are completely ludicrous, typically having no relevance to the actual corpus topic. So, before doing *any* processing, we needed to make sure our corpus was clean, consisting only of relevant articles.

Conclusions

To discuss the conclusions a semester of work produces means to acknowledge the variety of pieces that came together to make it all possible. And such a discussion begins with the tools that we leveraged.

A major component of our worked involved the use of NLTK. Our experiences were mixed with it. We found the biggest positive was that it made our lives easier, with the large number of built in functions it provided. It made classifying documents relatively painless, with support for decision trees, naive bayes, and max entropy. Likewise, built-in support for parts of speech tagging, chunking, and tokenizing sentences came in handy time and time again. We found, though, that it wasn't a silver bullet. We utilized the Pattern library to give us enhanced capability to determine the tenses of words, conjugate them, and manipulate our results to properly fill in our template. So while NLTK is undoubtedly powerful, incorporating other libraries should be part of any due diligence in this field of work.

As this was such a new field for all of us, understanding the workflow was extremely important. Proper application of the appropriate tools, at the appropriate time, allowed us to progressively understand our corpus in more depth as we proceeded. The various technologies provided ever more clarity to our work. Our major disappointment, then, is that we felt we didn't utilize them as much as we should have in the final deliverable. While we returned insightful results in earlier units with these tools, we relied heavily on regular expressions and part of speech tagging for our final summary. While we got great results, we're not convinced this is an extensible solution to other corpora. In theory, our final summary should likely have utilized Mahout, LDA, and MapReduce, in addition to regular expressions and grammars.

Finally, we discovered the difficulty inherent in natural language processing. With language as varied as the number of people that use it, identifying universally applicable rules is impossible. There is no specific phrase that should be used in all situations, nor should there be. Linguistics is an area made beautiful by its infinite uniquities, a fact that makes NLP a problem that will never be fully solved.

Future Work

From this point, there are a variety of routes that can be taken to extend our work this semester. As mentioned, we felt that our final deliverable did not do justice to all that we learned this semester and fear that it may not be as extensible as we would like. Therefore, the logical next step is to more thoroughly incorporate the variety of tools we have at our disposal. For example, classifying our documents based on what part of the fire they describe, then running our various algorithms on those, may allow us to more accurately extract information pertaining to a certain topic, for example firefighting or damages.

In addition, we ran many of our tools on a small dataset which may have excluded files that had useful details about one or more aspects of the fire. So following this approach may be the best for future work:

1. Scan the corpus and delete all duplicates by whatever preferred heuristic
    a. Comparing file size down to the byte level yielded good duplicate identification
2. For each file, iterate through its contents and attempt to identify the block of text containing the article
    a. Remove all text on either side
    b. Try to preserve headlines and dates
3. Classify each document into relevant and not relevant and remove all irrelevant
4. Construct a small subset for quick tests

Beyond the applications of this knowledge to analyzing corpora of news articles, there are many more problems that could utilize similar methodology, such as scraping status updates on Facebook to determine mood at certain times of the day, or summarizing book chapters to produce an automated SparkNotes.

Acknowledgements

We would like to thank the following individuals and organizations for all of the assistance they provided and giving us enough guidance to allow us to complete this project.

The National Science Foundation, for providing the grant for this course.

Dr. Edward Fox
fox@vt.edu

Tarek Kanan
tarekk@vt.edu

Xuan Zhang
xuancs@vt.edu

Mohamed Magdy
mmagdy@vt.edu

References

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

Rehurek, Radim. "Introduction." Gensim: Topic Modelling for Humans. Gensim, 17 Nov. 2014. Web. 08 Dec. 2014.

"Clustering - K-means." A Tutorial on Clustering Algorithms. Polytechnic University of Milan, n.d. Web. 08 Dec. 2014.

Appendix A

Top 50 Bigrams

Bigram	Frequency	Solr Results
fire department	594	442
rick perry	507	361
state park	435	110
officials said	377	2464
destroyed home	341	22
aerial view	316	137
press conference	298	9129
healthy living	173	Unrelated
golf course	136	264
catholic church	131	1000
super user	129	Unrelated
huffpost super	129	Unrelated
high school	124	Unrelated
tropical storm	117	135
first time	110	Unrelated
insurance companies	108	694
official said	107	Unrelated
home depot	106	Unrelated
weird news	99	Unrelated
news pages	98	Unrelated
report abuse	97	Unrelated
health news	88	Unrelated
entertainment celebrity	88	Unrelated
small business	87	2585
social security	83	Unrelated
middle school	82	Unrelated
house fire	82	287
front page	82	Unrelated

music movies	81	Unrelated
style style	80	Unrelated
politics politics	80	Unrelated
news women	80	Unrelated
entertainment music	80	Unrelated
crime weird	80	Unrelated
find something	78	Unrelated
make sure	67	Unrelated
home page	59	Unrelated
damage caused	55	335
national guard	55	Unrelated
death penalty	54	Unrelated
associated press	54	118579
michele bachmann	53	Unrelated
comment please	53	Unrelated
people make	52	Unrelated
link link	52	Unrelated
good health	52	Unrelated
family history	52	Unrelated
copy link	52	Unrelated
user agreement	51	Unrelated
photo galleries	51	Unrelated