Computational Linguistics - CS 4984

Generating an Intelligent Human-Readable Summary of a Shooting Event from a Large Collection of Webpages

December 2014

Arjun Chandrasekaran
Saurav Sharma
Peter Sulucz
Jonathan Tran

Virginia Tech
Blacksburg, VA

Table of Contents

List of Tables

List of Figures

Abstract

We describe our approach to generating summaries of a shooting event from a large collection of webpages related to the event. We work with two separate events - a shooting at a school in Newtown, Connecticut and another at a mall in Tucson, Arizona. Our corpora of webpages are inherently noisy and contain a large amount of irrelevant information. In our approach, we attempt to clean up our webpage collection by removing all irrelevant content. For this, we utilize natural language processing techniques such as word frequency analysis, part of speech tagging and named entity recognition to identify key words in our news events. Using these key words as features, we employ classification techniques to categorize each document as relevant or irrelevant. We discard the documents classified as irrelevant. We observe that to generate a summary, we require some specific information that enables us to answer important questions such as "Who was the killer?", "Where did the shooting happen?", "How many casualties were there?" and so on. To enable extraction of these essential details from news articles in our collection, we design a template of the event summary with slots that pertain to information we would like to extract from the corpus. We designed regular expressions to extract a number of 'candidate' values for the template slots. Using a combination of word frequency analysis and specific validation techniques, we choose the top candidate for each slot of our template. We use a grammar based on our template to generate a human readable summary of each event. We utilize the Hadoop MapReduce framework to parallelize our workflow, along with the NLTK language processing library to simplify and speed our development. We learned that a variety of different methods and techniques are necessary in order to provide an accurate summary for any collection. It is seen that cleaning poses an incredibly difficult yet necessary task when attempting to semantically interpret data. We found that our attempt to extract relevant topics and sentences using the topic extraction method Latent Dirichlet Allocation and k-means clustering did not result in topics and sentences that were indicative of our corpus. We demonstrate an effective way of summarizing a shooting event that extracts relevant information by using regular expressions and generates a comprehensive human-readable summary utilizing a regular grammar. Our solution generates a summary that includes key information needed in understanding a shooting event such as: the shooter(s), date of the shooting, location of the shooting, number of people injured and wounded, and the weapon used. This solution is shown to work effectively for two different types of shootings: a mass murder, and an assassination attempt.

Background

A requirement to receive a degree of Computer Science at Virginia Tech is to take a 4000 level Capstone course. What differentiates the capstone courses from other 4000 level courses is that capstone courses synthesize and integrate skills and knowledge acquired throughout the Computer Science curriculum and focus on design experiences including teamwork, communication, and presentation.

The capstone course this report covers is Computational Linguistics or CS 4984. The course is also known as Natural Language Processing (NLP). The focus is to give students the opportunity to engage in active learning while working to create summaries of collections about events using modern technologies, algorithms, as well as linguistic techniques. A set of toolkits are provided and methods and techniques are used to achieve the goal of summarizing these large collections.

Overview

The main objective of the Computational Linguistics course is to utilize various tools, methods, and algorithms to ultimately create a summary of a collection of documents. The tools that we used to process the collections and large data sets included Cloudera virtual machines, an 11-node Hadoop cluster, Solr databases, Mahout, and the Natural Language Toolkit (NLTK) libraries. Most of the development was done in Python (Python, n.d.)

Each section below introduces a set of driving topics that approach the end goal of generating an easy-to-read and grammatically correct summary of an event in English. Each of these sections include concepts, techniques, and tools that are used to analyze statistical data and extract key information from the large sets of data.

In each section we explain the tools we used, our approach, and the outcomes. Our results are analyzed and discussed and related back to our main goal of generating a summary. We describe pitfalls that may have impacted our final result and provide a brief explanation of possible improvements for future applications.

Section 1: Identifying Key and Indicative Words

Best Summary Using frequencies

Our initial approach used a simple cleaned collection of articles about record rainfall in Islip, New York. We tokenized the text by word and created a frequency distribution. The theory behind this approach was that frequently occurring words are likely to be indicative of the content of our corpus. This approach is at the mercy of punctuation and capitalization, as words with different capitalizations, and even words with punctuation, are not equal (when examined by code). Many words that occurred frequently were articles and prepositions that did not indicate any importance about the collection set.

The collection was processed to remove noise, such as special characters and punctuation. The words that were added as unique elements in the set were combined by lower casing and stemming the original words. The words that appeared often but were not indicative were considered stop-words. These are words like (the, and, or, there). The NLTK default stop-words list filtered out the majority of the words that we found to be irrelevant. We made an additional list to filter out irrelevant words that appeared often in our collection. We operated on noise-filtered and whitespace delimited words to generate a frequency distribution, which we sorted by occurrence.

The class event contained thirteen text files about a flood on Long Island in New York. Our results show the top words (Table 1) and phrases (Table 2).

Words									
york	news	record	islip	flooding	inches	long	rain	weather	island

Table 1 Most common words from Islip Corpus

Phrases					
across long	flooding across	historic inches	island historic	long island	text flooding

Table 2 Most common phrases from Islip Corpus

Upon observing the results, we found flaws such as lower casing every word when the article contained proper nouns. It is worth noting that our custom stop-word list applies only to documents obtained from the same webpages as our collection. A way to improve our stop-words list is to include a wider range of collections spanning the common words found in webpages. Along with finding the frequency distribution, we can add weight to the words using term frequency-inverse document frequency (tf-idf) to determine the relevance of a word. Using tf-idf, we would only retain words with a high relevance weighting.

Synonyms

We combined the Brown, Reuters, Words, and State of the Union corpora from NLTK to obtain a frequency distribution of all the words contained within. We designed our own stop-words list based on the frequently occurring words throughout the corpora.

We attempted to reduce noise by removing non-alphanumeric characters, and limiting a word length to be less than 17 characters and greater 2. The downside to this was that we likely lost titles and dates, which might have been relevant to our event. We also lemmatized and lower-cased words to improve our frequency analysis.

We analyzed our small event to create word frequency (Figure 1 Word Frequency), most frequent words (Figure 3), and word pair frequency (Figure 4) distributions.

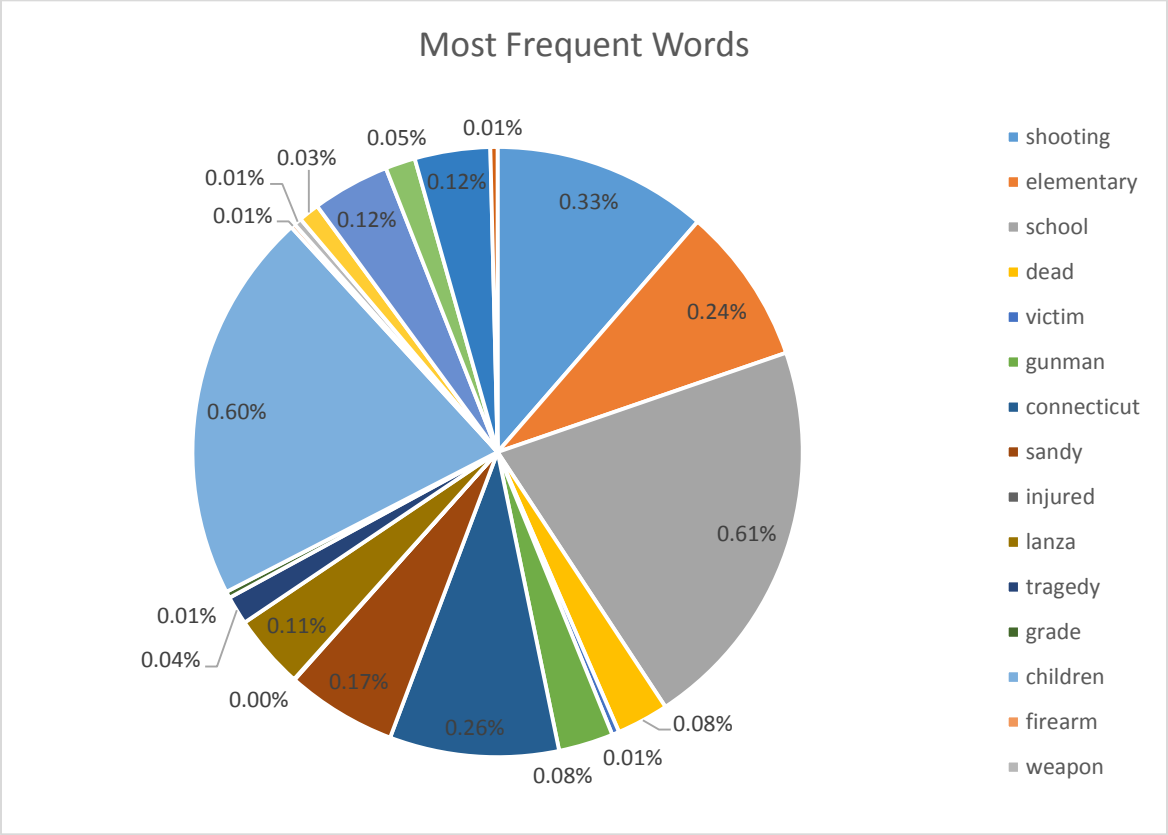

Figure 1 Word Frequency

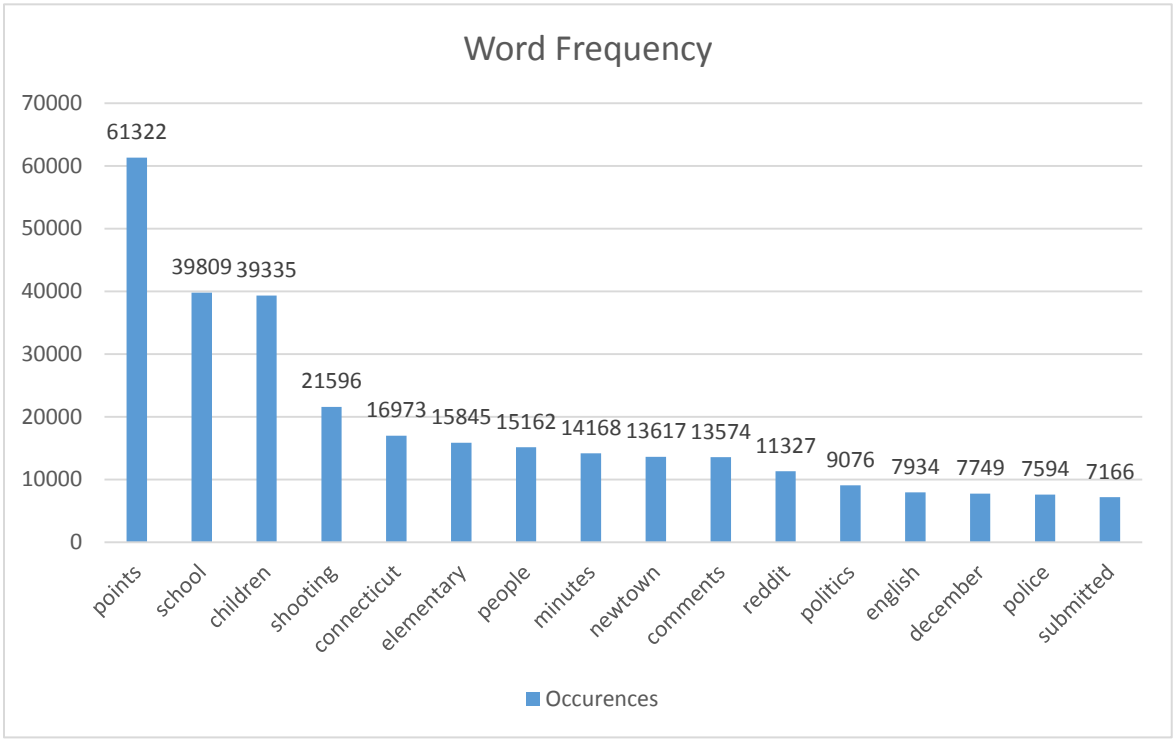

Figure 3 Most Frequent Words

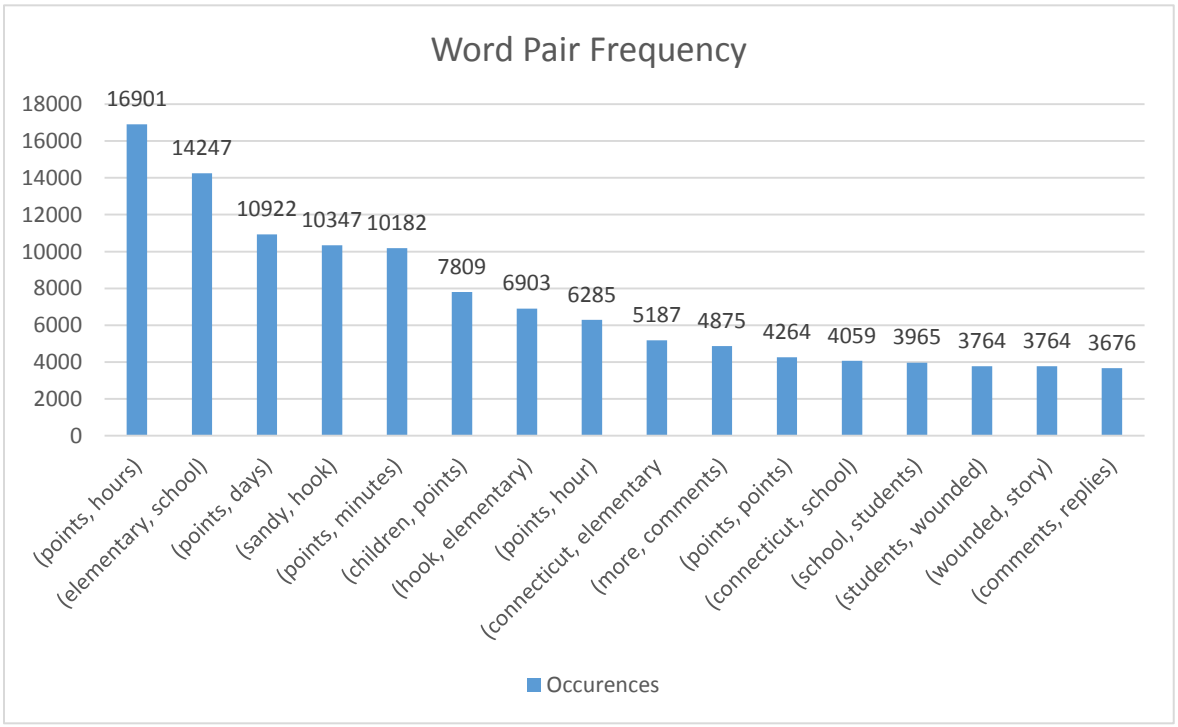

Figure 2 Word Pair Frequency

We observed that removing stop-words gave us a more relevant list of indicative words for our events. A noteworthy point is that the choice of stop-words must be comprehensive and proper for the specific event.

After doing a frequency analysis on collocations, we found that the most frequently occurring collocations were more descriptive and relevant than single words. We learned that though single word frequency distribution is a helpful way to get a grasp on what the event is about, it cannot provide as much context as collocations.

Upon observing our results, we found top words and collections that were irrelevant to our events. We noticed that in our small collection, we had a lot of articles from www.reddit.com, which consistently contain the same words and word pairs. Better cleaning of the collection could have helped avoid this issue.

Part of Speech Tagging

We used the Punkt tokenizer (nltk, n.d.) on the raw text to split our articles into sentences and tagged the words with the nltk.pos_tag tagger (Perkins, n.d.).

We picked out the most relevant nouns, verbs, and adjectives from a list of the most frequently occurring parts of speech (Table 3).

Small Collection			**Big Collection**		
point	N	60369	**news**	N	161035
hour	N	31356	**add**	V	126209
school	N	30776	**video**	N	94389
day	N	25501	**comment**	N	71473
news	N	23123	**say**	V	69249
gun	N	16347	**view**	N	68173
say	V	14839	**get**	V	63477
connecticut	N	14596	**home**	N	57251
minute	N	14134	**Tucson**	N	54526
shoot	V	13689	**arizona**	N	54058
people	N	12157	**day**	N	53770
get	V	11736	**queue**	N	53576
go	V	11379	**year**	N	51605
post	N	10401	**use**	V	51605
sandy	N	10359	**time**	N	50557
hook	N	9791	**january**	N	49404
child	N	9757	**right**	N	49328
make	V	9623	**people**	N	49171

Table 3 Most frequent verbs and nouns

We determined that a majority of the words in Table 3 were in fact indicative of our collections after doing a Solr search using the words in the query and finding that all articles returned were relevant.

It should be noted that although the majority of the data was relevant for our small collection, our big collection resulted in many irrelevant words. These irrelevant words occur often in the webpages composing our big collection. A flaw we noticed with the Punkt Tokenizer was that it did not consistently differentiate between periods as sentence breaks and periods after abbreviations/titles. It was also unable to split a sentence that did not contain whitespace after the period.

Section 2: Sanitizing Data

Classification of Documents

At this point, something that we focused heavily on was eliminating noise. This was the first time we had encountered the idea of separating good data from bad data at a much higher level than just words. We relied on a classification method to do the identification. The idea behind this was that we would manually tag a training set with positive and negative files, and classify each file in our collection as positive (relevant) or negative (noise).

The features for our classifier were manually selected words from the indicative words that we found in previous units. We found that SVM was the best performing classifier, but since it could not be used on the cluster, we chose to use the second-best performing classifier, namely the DecisionTreeClassifier for both the small and big collections. The classifier was run on the cluster for both collections.

We manually tagged 342 positive and negative training files in our small collection in order to classify the entire small collection. In more detail, our small collection training included 5 folds of training and testing with 274 files being trained and 68 files being tested. Table 4 represents average accuracies over 5 folds and the total time taken for all 5 folds of cross validation.

Creating a training set for the big collection proved to be a much more difficult problem than we anticipated. From the largest 1,000 files, we chose 77 files for our training set with a ratio of 1:3 positive to negative. This was because we only found 20 relevant files in our selection. We chose a small training set because we did not want to saturate the training data with irrelevant files. We had 5 folds of training and testing with 62 files being trained and 15 files tested. Table 4 represents the average accuracy over 5 folds and the total time taken for all 5 folds of cross validation.

Classifier	Accuracy	Time taken to train (s)
Naïve Bayes	0.9383	2.48
Decision Tree	0.9619	12.06
MaxEntropy	0.931	241.83
SVM	0.9938	4.99

Table 4 Small Collection, Classifier results

Classifier	Accuracy	Time taken to train (s)
Naïve Bayes	0.8039	5.75
Decision Tree	0.9167	6.76
MaxEntropy	0.9566	597.41
SVM	1	6.02

Table 5 Large Collection, Classifier results

These results seemed suspicious enough to us to do some manual verification. For our small collection, we did this by randomly choosing 50 files that were classified as relevant and 50 that were classified as noise. The results of our analysis were as follows:

Small Collection
Accuracy = 0.96
Precision = 0.92
Recall = 1

Table 6 Small collection, manual classifier rating

For our big collection, we did the same and our results showed that:

Big Collection
Accuracy = 0.57
Precision = 0.14
Recall = 1

Table 7 Large collection, manual classifier rating

Overall, we saw that our classifier performed very well in differentiating relevant documents from noisy ones for our small collection but not for our big. We postulated that this happened because we had a very small number of relevant documents to train our classifier, namely 20 documents among 30,000. As shown in Table 7, the reason our precision was so low was because we designed our feature vectors to result in maximum recall. We hypothesized that our big collection classifier would perform better with a more equal distribution of positive and negative files.

Section 3: Categorizing and Clustering

Named Entities

From this point onwards, we started only operating on the "positive" files that we discussed in section 2.

The Stanford Named Entity Recognizer tags entities based on their grammatical structure. We had a choice between using the Stanford NER and the NLTK NER, and decided upon using the former as it seemed to perform better with our dataset. We first used the Stanford NER tagger with the default English model, and then decided to use a custom training set to get more accurate results.

Using the Stanford NER tagger with the default English model, we found that Connecticut was correctly tagged as a location. This was correctly followed by the town: Newtown. Ideally, we should have also found "Sandy Hook" as the location, but both "Sandy" and "Hook" were tagged as organizations. Google was correctly tagged as an organization but was irrelevant to our data. "School" was incorrectly tagged as an organization; it should instead have been a location. Also, it is worth noting that Connecticut and Newton were tagged as organizations in addition to locations. This tagger tagged "Lanza" as a person, who was indeed the shooter involved in our event. All in all, the Stanford NER tagger with the default English model tagged some important features of our collection, but often seem to give them the incorrect tag.

Using the Stanford NER tagger with a custom training set, we were able to get more accurate results. All person entities that were tagged were indeed real names, with the top two being "lanza" and "adam". Our entity search for time returned "friday", "saturday", and "december". Our custom tagger performed very well for the location entity, yielding results describing the location of the shooting, such as "school", "connecticut", "newton", "conn.", "hook", "sandy" etc. Something to note is the fact that "newtown" was misspelled as "newton" eight times. This is notable because it brings in the perspective that the even the "good" data we were given was in itself not entirely accurate.

Table 8 and Table 9 show more comprehensive results of the default Stanford NER tagger and our custom tagger with respect to the location and person entities:

CUSTOM STANFORD NER		**STANFORD NER**	
school	2577	connecticut	1362
connecticut	1638	newtown	757
newtown	869	u.s.	599
conn.	329	conn.	275
hook	168	washington	91
sandy	166	ohio	75
control	79	east	74
washington	45	city	71
detroit	45	new	63
huffington	31	america	56
congressman	29	mississippi	54
elementary	22	united	48
company	18	hollywood	48

congress	10	san	45
newton	8	states	44

Table 8 Small Collection, Stanford NER comparison of results for location between standard training set, and custom training set.

CUSTOM STANFORD NER		**STANFORD NER**	
lanza	217	lanza	339
adam	197	obama	264
obama	33	adam	246
ryan	16	sandy	140
nancy	9	hook	129
vance	4	john	102
paul	4	edwards	97
summers	3	kasich	84
puja	3	paul	82
parti	3	jennifer	69
nather	3	christina	67
lanzas	3	scarlett	63

Table 9 Small Collection, Stanford NER comparison of results for location between standard training set, and custom training set.

As visible from the results, this approach gave us a much better representation of the content of the collection when compared to using just word frequencies or POS tagging. The results obtained from our named entity recognition improved upon previous methods in providing unique indicative information for our collections.

		topic 2	
topic 1			
WORDS	**PROB**	**WORDS**	**PROB**
school	0.0175	days	0.0232
shooting	0.0085	ago	0.0159
elementary	0.0073	point2	0.0094
hook	0.0071	permalinksavecontextfull	0.0088
news	0.0068	2.0000	0.0076
sandy	0.0067	ago	0.0099
newtown	0.0067	you	0.0072
conneconnecticut	0.0057	I	0.0070
2012	0.0050	all	0.0056
from	0.0043	points1	0.0054
december	0.0041	reddit	0.0052

Table 10 Small Collection, best topics

Latent Dirichlet Allocation

A topic model is a statistical model that aids in the discovering of abstract "topics" occurring in a collection (Blei). If a document is about a particular topic, we would expect that certain words would appear in the document more or less frequently. Latent Dirichlet Allocation (LDA) is a way of automatically discovering the topics contained in sentences, representing documents as mixtures of topics. We utilized LDA via Mahout, which is a library of scalable machine-learning algorithms that sits on top of Hadoop.

Before running anything, we iterated on the cleaning done in our collection by removing multiple newlines as well as foreign language words. We decided not to remove stop words because we wanted to maintain full sentence context. We then ran LDA on our cleaned collections using Mahout and obtained strong results for 3 topics for our small collection and 5 topics for our big collection. A summary of the top words from each topic follows in Table 10 and Table 11:

topic 4		topic 5	
WORDS	**PROB**	**WORDS**	**PROB**
tucson	0.0140	add	0.0307
arizona	0.0115	queue	0.0268
giffords	0.0096	added	0.0263
shooting	0.0075	views	0.0232
news	0.0071	1.0000	0.0100
gabrielle	0.0057	ago	0.0099
from	0.0051	video	0.0093
about	0.0048	2.0000	0.0090
who	0.0047	you	0.0086
local	0.0045	your	0.0083
killed	0.0042	0.0000	0.0077

Table 11 Large Collection, best topics

It is worth addressing that our decision to not remove stop-words resulted in them showing up in our output data.

We then chose a random number (K=7) words from the large collection and ran Solr queries. We ran the highest weighted words from the 4th topic for our big collection: Tucson, Arizona, Giffords, shooting, news, Garbielle, and from. These words did a great job in summing up our big collection and the Solr searches returned results with 100% accuracy. This was not surprising to us as these words answer the 'who', 'what', and 'where' regarding the content of our collection.

Overall, we found that LDA was a very effective tool to extract related words from our collections, in spite of our collections being noisy. Something that could have been done better would have been to identify which obtained topics were relevant. We could have used a set of words from either the feature set that we used to train our classifier, or our most frequent words from our indicative words from earlier units, or even a combination of the two. After doing this, we could have chosen the top N words that are required for our summary.

Clustering

Clustering is an unsupervised machine learning method for grouping together a set of related objects based on distance from a group centroid. We implemented a widely used clustering algorithm called k-means to separate the sentences in our corpus into related groups. K-means involves representation of data as an n-dimensional "feature vector". Figure 4 shows a number of groups (clusters) of data that are separated into three (in general, k) groups. After an initialization that fixes the number of clusters (k) and the initial centroids for these k clusters, k-means iteratively determines the points belonging to each cluster (based on least distance from cluster centroid) and computes the centroid locations of the k-clusters. While k-means is guaranteed to converge, this convergence may just be a local optimum and not the expected global optimum. Figure 5 is an example of an initialization that results in a local optimum, which is clearly not the desired global optimum.

Thus, from an implementation point of view, apart from the input feature vector, there are some important parameters: number of clusters, cluster size, and initial locations of cluster centroids that need to be carefully chosen. We implemented k-means on our vectorized data on the HDFS using Mahout's k-means (Apache, n.d.).

Choice of features

During our initial investigations, we experimented with the Mahout clustering algorithm to see whether words that appear in similar contexts would be grouped in the same cluster when we chose words as features to k-means. In the clustered output, we noticed that there were a few clusters with related words grouped together but these words were not among the 'Top Terms' (words closest to the cluster centroid). The clusters that did have similar words among the 'Top Terms' had a few good words along with some irrelevant words. We did not obtain satisfactory results from this naïve approach. A better approach to find similar words would have been to look at n-grams, rather than separate words. Thus, for our task of categorizing sentences from our corpus, we first filtered out candidate sentences which were likely to be indicative of our corpus. We placed each of these sentences in a separate file that were input to Mahout's vectorization utility (Apache). These sentences were then broken down by Mahout into vectors using tf-idf (Buckley) weights - which gave us our feature-vectors. We expected that our output would give us a set of clustered documents that each had a sentence with words similar to the other documents in the cluster.

Choice and initialization of K

In an implementation of k-means, an important parameter to consider is the choice of k. This is straightforward for cases in which the output is known to belong to one of k classes beforehand. In cases where not much is known about the number of desired output categories, the choice of k becomes trickier. A bad choice of k could cause the algorithm to produce sub-optimal output, even in cases where there are (for humans) an obvious number of clusters. If we initialize the number of clusters to 2 or 5 in a case such as Figure 5, the output may not make much sense. Figure 5 explores a case where even if k is chosen properly, we may still run into local optima.

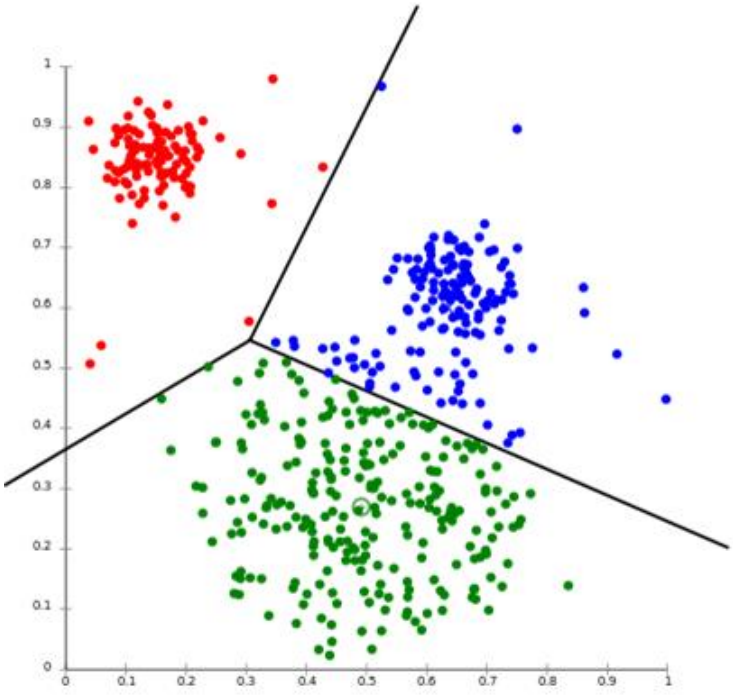

Figure 4 Representation of K-means clusters (K-means Clustering, n.d.)

We were unsure of how many categories of sentences there were. Thus, we decided not to specify the number of clusters manually using Mahout's (-k option). We instead used Mahout's canopy clustering which automatically initialized the number of clusters and also the location of the centroids.

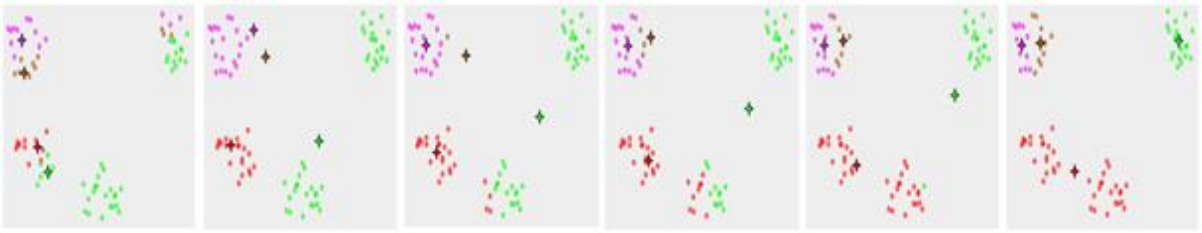

Figure 5 An typical example of K-Means Convergence

A typical example of the k-means convergence to a local minimum. In this example, the result of k-means clustering (the right figure) contradicts the obvious cluster structure of the data set. The small circles are the data points, the four ray stars are the centroids (means). The initial configuration is on the left figure. The algorithm converges after five iterations presented on the figures, from the left to the right. The illustration was prepared with the Mirkes Java applet. (K-means Clustering, n.d.). Figure source: (K-means Clustering, n.d.)

We specified two parameters that helped Mahout's canopy clustering in picking an optimal value for 'k'. The two parameters 't1' and 't2' determined the size of a cluster and distance between two clusters or overlap between two clusters. Fixing these two parameters enabled canopy to compute a value for 'k' and the initial locations of the cluster centroids. After extensive experimentation with different configurations of t1 and t2, we came up with the optimum cluster size and number of clusters.

From our experiments on cluster size, we learned that by having:

a) Larger t1 and t2, we had fewer clusters with big cluster sizes and (in general) a large number of points (sentences) assigned to a few clusters. However, using bigger t1 and t2 values caused memory issues.
b) Smaller t1 and t2, we had many clusters that were of small sizes. This resulted in a large number of clusters that contained a small number of similar indicative sentences.

Due to Hadoop memory issues, we used the latter approach.

Selecting a cluster

From the set of clusters that we obtained as output, we chose our pool of 'good' clusters by looking in our cluster centroids for a combination of:

1. The most frequent words that occurred in our collections which were indicative of corpus content.
2. Named entities in sentences obtained from our custom Stanford NER.

We noted that the topics from LDA were not very helpful for our task of selecting relevant sentences that characterized our corpus.

Finding the best sentences

Sentences whose feature-vectors have least distance from the centroid of the cluster are the sentences that best represent the cluster.

We further used a combination of approaches to choose a number of sentences from a cluster that could characterize our corpus:

1. When we looked at the number of points in a cluster, we found that the higher the number of points, the likelier it is for the cluster to be 'good' and contain relevant sentences. This is a valid assumption because we filtered regularly occurring noise (stop-words). Thus, the remaining regularly occurring sentences which had many related sentences were directly from our articles.
2. In addition, we noticed that we must not consider all the sentences from a cluster – rather just ones that were closer to the cluster centroid. In spite of our cleaning, we noticed that sentences away from relevant cluster centroids were not of use to us. We used the implementation to compute distance from centroid provided by the TA in our own code to obtain the closest sentences to the centroid.
3. We combined ideas from 1 and 2 and in our code implemented a weighted selection of sentences based on cluster size and distance from centroid.

In summary, there are two parameters that we can choose to create a pool of good sentences:

a) Size of cluster
b) Distances from centroid of sentences

Below are our results from a selected cluster for each collection:

SmallEvent
'A 28th person, believed to be Nancy Lanza, found dead in a house in town, was also believed to have been shot by Adam Lanza.'
'Twenty six people were shot dead, including twenty children, after a gunman identified as Adam Lanza opened fire at Sandy Hook Elementary School.'
‘The shooter's mom was also a kindergarten teacher there and was the target."
'Authorities say gunman Adam Lanza killed his mother at their home on Friday and then opened fire inside the Sandy Hook Elementary School in Newtown, killing 26 people, including 20 children, before taking his own life.'
'The gunman also died at the scene, and a 28th body was found elsewhere.'
The gunman also killed his mother and himself.Sen.
Incident is among worst mass shootings in U.S. history.
'A gunman at Virginia Tech University killed 33, including himself, in 2007.'

Table 12 Small event, Clustering results

From these selected sentences, we can see that our method performed well in extracting sentences that have to do with our event of Newtown shootings, but not fantastically on keeping them strictly related to our event. We have a lot of sentences related to gunmen and shootings, for example, the Virginia Tech shootings. This cluster focuses on the casualties of the shooting incident.

BigEvent
'I am praying for Gabrielle and all of those who love her.'
‘4Suspect in attack on congresswoman acted alone911 call: "I do believe Gabby Giffords was hit.'
'"I was in shock," he said, describing his reaction to the shooting.'
'EDC Gear 1What Holster to Buy?'
'"Still, LeMole stresses that the danger for Giffords\' is far from over.'
'Diana Lopez from the Tucson Police Department.'

Table 13 Big Collection, Clustering results

We see results that are not good for our chosen parameter values and cluster sizes. This is because the ratio of relevant content to noise is skewed in our big collection. So, the parameters, cluster sizes and even cleaning methods that we use for our collections do not perform well enough for our big collection. We see that for our big collection, the cluster contents are not as similar in content as they were for our small event.

We can improve our overall clustering by selecting features based on which we cluster our sentences. In this implementation, we clustered entire sentences with words weighted by tf-idf as features. An extension would be to use phrases, or other sentence patterns as features. We could even go a step further and choose only a limited number of words or phrases around which clusters are built.

For example, we may specify, “people injured” or “shooter was found” as features and perform clustering.

Section 4: Generating Paragraph Summaries

Populating Templates

In order to write a summary of an event, we require at least a few essential facts. We determined what facts these were and represented how they would be placed in our summary by building a template with the required facts represented by slots. The template which we used is shown below.

On< date >, there was a shooting at <where>. <Shooter(s) name> attacked the<place of shooting> at <time of day>, by opening fire in a <where>. The <shooter(s)> killed<number of victims> and injured<number of victims> before <did shooter die or get caught>. The victims were <age range>, and were <targeted or random>.

The <shooter(s)> used a <type of weapon>, and shot<number of rounds>. They had intended on <bigger plans> following the shooting. The <shooter(s) > spent <time planning>, and planned using<resources used to plan>. The reason behind the attack was <motive>.

Our aim is to extract appropriate words and phrases that fit the slots. The sub-tasks involved in extracting appropriate values for the slots are shown in Figure 6.

Identifying patterns of words or phrases around a slot

The first thing that we identified during slot filling was the type of value that fit the slot. A slot may be filled by numeric values (e.g., number of people killed), or by text (e.g. name of the killer) or by a combination of both (e.g., Date and Day). After these are identified, we needed to observe occurrences of these values in our collection of articles. We studied these occurrences using a Python script that looked through our collection and returned the sentences in which our selected word or phrase appeared. This information is similar to the information that the NLTK concordance function returns. Based on our study, we listed the most common context in which words occur, namely the phrases preceding and following the word, specific mention of named entities, etc. Using this information, we built our regular expressions.

Creating Regular Expressions

We built Python regular expressions based on the context information of a slot and tested it using Solr. We refined our regular expressions to filter out invalid matches while maintaining generality, so they could be applied to multiple documents. Our focus was more on recall than on precision because of the large sizes of our collections. We focused on extracting all the possible candidates present in our collections that could potentially fill our slots.

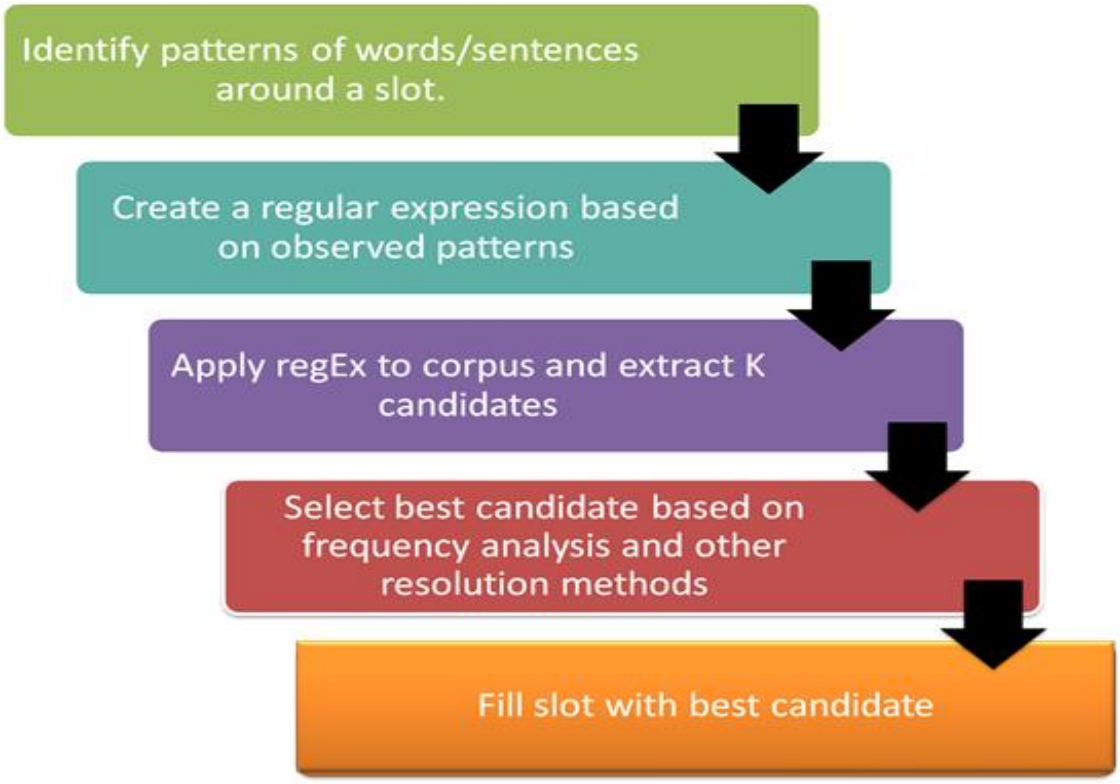

Figure 6 Template filling procedure

Applying Regular Expressions to our corpus

We were able to correctly fill most slots but were unable to extract abstract information (e.g. the killer's motives and whether the killer was apprehended) as abstract information was generally not just one word or a simple single phrase.

Slots that address questions related to motive and plans present a challenge as there are many ways to represent them. The challenge with slots that address questions related to motive and plans is that there are many ways in which news articles present them in sentences. It is difficult to identify a pattern for a regular expression that exactly matches and extracts the killer's motive or plans. For e.g, we noticed that some news articles mentioned, “the killer was found ….” while some others mentioned “Adam Lanza was found dead ...”. To enable matching both of these statements to a single regular expression, we tried an iterative slot filling approach (shown in Figure 7) to extract suitable candidates for these slots.

In the first iteration of regular expression matching and slot filling, we extracted and selected our best candidate for a few slots, for e.g. the killer's name. We used this information to construct regular expressions that matched other slots, for e.g. "(killer's name) was found". In our second iteration we extracted candidates for these slots and selected the best candidates.

Selecting the best candidate from K candidates

We used two methods to select the best candidate from a set of K candidates. First, we used frequency analysis to shortlist the most frequently occurring candidate for each slot. We observed that

in all but a few special cases, the right candidate for the slot was the most frequently occurring candidate. Thus, for the majority of slots, we simply chose the candidate with the highest frequency of occurrence.

For some special cases, e.g. the killer's name, we found that the correct candidate was the fourth most commonly occurring, whereas the three most frequent matches ended up being random English words. For the cases where the most frequently occurring candidate was not the right selection, we employed some resolution techniques.

To identify the killer's name from the list of matched phrases, we verified whether both the words of the candidate were proper nouns using our previously built Trigram tagger. If both words were indeed proper nouns, we inferred that it was a person's name, more specifically, the killer's name.

Another slot for which we employed a resolution method was 'date'. Since most articles were written on the days following the actual shooting incident, going just by frequency gave dates after the actual incident. We noticed, however, that many articles contained sentences such as 'Monday's shooting has caused outrage …' and '… aftermath of Monday's shootings…' This style of mentioning the 'day' of the shooting event was quite prevalent throughout our corpus. For the purpose of readability, we will refer to the date that an article was written as the 'absolute date'. With pointers (Jurafsky & Martin, 2009), we designed regular expressions to extract both the frequently occurring absolute dates and the 'day' of the shooting.

Once we had the most frequently occurring absolute date and the most frequently occurring day of the shooting, we approached a temporal resolution in a few different ways. One approach was to find the closest date before the most frequently occurring absolute date that mentioned the most frequently occurring day. The other approach was to look for absolute dates which had the same day in them as the most frequently occurring day i.e. we looked for articles written on the same day as the shooting. The latter method worked well since most of our articles were web news articles and thus, were often written immediately in response to the incidents. We implemented the latter method for temporal resolution to extract the correct date of the shooting.

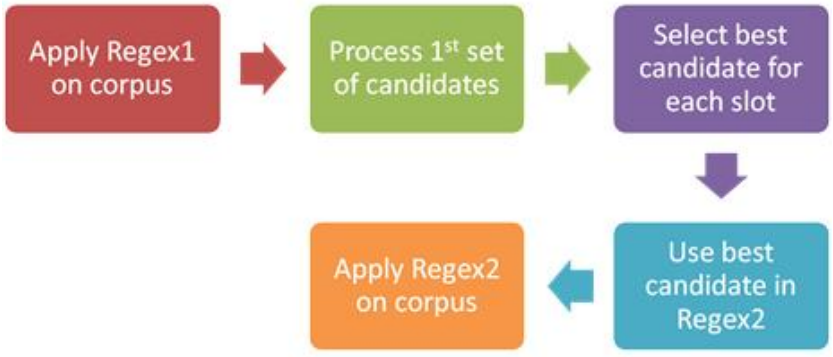

Figure 7 Advanced template filling procedure

Understanding relationships

To create meaningful summaries, we needed to understand the circumstances of the events. Up until this point, we had a superficial fact-driven summary which answered the questions of who (is the killer?), when (did the shooting happen?), where (did it happen?), how (type of weapon?) and what (number of injuries?).

Our approach, however, could be vastly improved with a better understanding of the subject involved. Only with a better understanding can we hope to answer questions like, 'What was the killer's motive?', 'Did the killer have elaborate plans?', 'Who was the killer's primary target?' We can quickly notice that these questions:

1. Are open-ended.
2. Have answers that do not have a fixed pattern or format (unlike the previous ones we answered above)
3. Could have answers that are entire sentences or even span multiple sentences.

We felt that in order to answer these open-ended questions, we first needed to identify the subject sentences and their relationship with the objects of the sentences. Understanding the relationship between subjects and objects gives us the ability to be free from depending on regularly occurring patterns, allowing us to extract abstract information such as the motivation of the killer, his plans, etc.

We were able to correctly fill most slots but were unable to extract abstract information (e.g., the killer's motives and whether the killer was apprehended) as that required more than matching just a single word or phrase. To extract relationships between the subject and objects in a sentence, we used the ReVerb Relationship Extractor (Washington, n.d.), which was developed by the University of Washington. ReVerb is an open Information Extraction (IE) system which aims to improve the coherence and relevance of the extracted information. The input to ReVerb is a sentence and its output is a 3-tuple in the form of: subject, relationship, and object. ReVerb is able to identify relationships between multiple pairs of subjects and objects. Upon using ReVerb with our collections, we ended up with incoherent, irrelevant, and overall unsatisfactory results. However, we noticed that ReVerb performed satisfactorily on simpler sentences such as (a), and sentence parts separated by commas such as (b) and (c) as shown in the examples below.

Slots that address questions related to motives and plans presented a challenge because they had many different representations in our collections. We found it difficult to identify a pattern for a regular expression that exactly matched and extracted the killer's motive and plans. For example, we noticed that some news articles mentioned "the killer was found" while others mentioned "Adam Lanza was found". To provide a single regular expression that would encompass both of these statements, we utilized an iterative slot filling approach (shown in Figure 7) to extract suitable candidates.

The first iteration of regular expression matching and slot filling involved extracting and selecting the best candidates for a few slots such as the killer's name. We then constructed regular expressions that matched slots that utilized previously found information. For example, the slot "(killer's name) was found" utilized knowing the killer's name. In the second iteration we extracted candidates for these slots and selected the best.

Sentences	Subject	Relation	Object
(a) Garage doors were ripped off homes .	garage doors	rip off	Homes
(b) An apartment complex was badly shattered , a school set ablaze , and a nursing home was left in ruins .	a nursing home	leave in	Ruins
(c) Residents were kept out of a large swath of West , where search and rescue teams continued to pick through the rubble .	residents	keep out of	a large swath of west
(d) He was off duty at the time but responded to the fire to help , according to a statement from the city of Dallas .	the time	respond to	the fire
(e) At West Intermediate School , which was close to the blast site , all of the building 's windows were blown out , as well as the cafeteria .	west intermediate school	be to	the blast site

Table 14 Output from ReVerb

The algorithm did not produce accurate results for long, complex sentences present in news articles such as (e) and (f).

As part of future work, we could test the Stanford relation extractor (Multi-instance Multi-label Relation Extraction, n.d.) to get a deeper understanding and improve our ability to answer abstract questions.

Grammar

To produce a well-written and informative summary of our corpus, we created a regular grammar which could be easily filled, was general, and created an informative, well-structured summary. Our goal was to easily convey the “who”, “what”, “where”, “when”, and "how" to the reader, in a “tweet” style paragraph. We created a handwritten summary of our smaller collection, keeping in mind that we should only use facts that are obtainable in a big data analysis, and could be validated programmatically. Our summary for our small can be found below:

On the morning of Saturday, December 15, Adam Lanza opened fire in Connecticut. The gunman fired 100 rounds out of his rifle. 27 children lost their lives. 2 of the victims were hurt, and are being treated for their injuries. The victims were between the ages of 6 and 7.

Numbers, dates, and names were simple to include, since they are consistent and behave well while analyzing regular grammars and big collections of data. We did encounter problems when trying to extract and include more abstract facts, such as the shooter's motive, whether or not the attack was premeditated, and if there was a specific target in the event. Large collections cannot be relied on to have consistently written abstract facts, so extracting them by frequency analysis is generally unsuccessful. Because of this, we developed a simple and straightforward regular grammar:

Non-Terminals	***Terminals***
Summary	*<intro> <weapondescription> <killed> <wounded> <agerange>*
Intro	*On the <time> of <date>, <shooter> opened fire in <location>*
weapondescription	*The <gunman plurality> fired <number> rounds out of his <weapon>.*
killed	*<number> (children|people) lost their lives.*
wounded	*<number> of the victims were hurt, and are being treated for their injuries.*
agerange	*The victims were between the ages of <number> and <number>*
gunman plurality	*<word> <word>*
weapon	*<word> <weapon>*
Time	*(morning|afternoon|night|evening)*
Date	*<day of week>, <month> <number>*
day of week	*(monday | tuesday | wednesday | thursday | friday | saturday | sunday)*
Month	*(january | february | march | april | may | june | july | august | september | october | november | december)*
Shooter	*<word> <word>*
Location	*<word>*
Number	*[0 -9]+*
Word	*[a-z]+*

Table 15 Conext free grammer for creating a summary

Using our grammar, we were able to generate the following summaries:

Newtown School Shooting:

> *On the morning of saturday, december 15, adam lanza opened fire in connecticut. The gunman fired 100 rounds out of his rifle. 27 children lost their lives. 2 of the children were hurt, and are being treated for their injuries. The victims were between the ages of 6 and 7.*

Tucson Gabrielle Giffords Shooting:

> *On the night of sunday, january 9, jared lee opened fire in tucson. The suspect fired 5 rounds out of his rifle. 6 people lost their lives. 32 of the people were hurt, and are being treated for their injuries. The victims were between the ages of 40 and 50.*

It is important to note that the Newtown School Shooting occurred on Friday, December 14th as opposed to Saturday. Also, 20 children and 6 adults were killed, rather than 27 children. The Tucson shooting also happened the day before, on Saturday January 8, and only 14 people were injured. We noticed that most new articles relating to these events were posted on the day following the event, causing issues with our frequency analysis. It is also important to recognize the missing capitalization of proper nouns. Our collections were normalized to aid in frequency analysis and simplify regular expression, so results were all in lower case. Capitalization of proper nouns would be simple to handle in code, so we decided to focus on improving results under the time constraints.

Lessons Learned

Aside from group dynamics and understanding the technologies required to complete this project we learned several important lessons. Code reuse and simplicity, along with good organization and file structure is essential for efficient group collaboration. We found out how important it is to have clean data in order to extract semantic understanding. We also learned how difficult it is to clean data to an optimal level without losing important information. Additionally we found out how difficult it is and how many processes are involved in solving a problem of this scope. In doing so, we learned the importance of allocating enough time to iterate over and improve potential solutions.

Conclusion

We found the impact that noise can have on interpreting data surprising. Following this, we learned how difficult it can be to properly and consistently clean data. In addition, we learned that doing something as complex as summarizing a large collection requires a multitude of techniques. We painfully found the amount of experimentation and iteration required in attempting to answer an open-ended research question.

Due to time limitations, we were unable to reduce noise to a level that we deemed satisfactory. A weakness of our summarization approach is that it was fairly ad-hoc. The extraction of information needed for filling out our template was entirely based on regular expressions and frequency analysis. Thus, we were unable to get a proper semantic understanding of the data we extracted. Furthermore, answering abstract and more difficult questions about our event was done simply by using a regular grammar. We are unlikely to be able to produce a coherent and accurate summary for different shooting events. As a direct result of all of this, our summary was relatively short and lacking.

If we had more time, we would have invested additional effort in classifying files to further clean our collection. Future work could improve our solution by using a context-free grammar in generating a summary to account for ambiguity. It would also be worthwhile to implement a better extraction method that utilizes dependencies for robust semantic recognition.

In conclusion, we demonstrated an effective way of summarizing a shooting event that extracts key information from using regular expressions and generates a human-readable and comprehensive summary utilizing a regular grammar.

Developers Manual

Attached Code

To anybody interested in expanding or improving our work, we have included the following resources. Relies on Python 2.7.8

- mapper.py
  - The map script used on the Hadoop cluster. This file contains the regular expressions which are used to extract the data from the collections.
  - Input: List of files to process. Opens each one of these files, from a Hadoop cache archive named “eventdata”
  - Output: Writes the regex results to standard out in the format <result>_<regex> 1
- reducer.py
  - Standard frequency based reducer on the cluster. Takes sorted input from the mapper, and reduces based on frequency.
- parse.py
  - Python which parses the output from the mapper and reducer. Does not run on Hadoop cluster. Relies on data being sorted by frequency (most frequent at top of file). Contains the implementation of the regular grammar, along with filtering techniques.
  - Output: The most frequent values which were found, along with the filled grammar.
- TrigramTagger.pkl
  - A python pickled version of our trigram tagger, which is used in parse.py. Contains an NLTK backoff tagger.

Timings

Job	Timing
Run 1	74 seconds
Run 2	70 seconds
Run 3	100 seconds
Average:	81.3 seconds

Table 16 Collection Small Hadoop job timings

Job	Timing
Run 1	231 seconds
Run 2	213 seconds
Run 3	226 seconds
Average:	223.3 seconds

Table 17 Collection Big Hadoop job timings

Acknowledgments

We would like to thank:

Dr. Edward A. Fox

fox@vt.edu

Xuan Zhang

xuancs@vt.edu

Tarek Kanan

tarekk@vt.edu

Mohamed Magdy Gharib Farag

mmagdy@vt.edu

With support from NSF DUE-1141209 and IIS-1319578

References

Apache. (n.d.). *Apache Wiki*. Retrieved from Solr Query Syntax: https://wiki.apache.org/solr/SolrQuerySyntax

Apache. (n.d.). Creating Vectors from text. *http://mahout.apache.org/users/basics/creating-vectors-from-text.html*.

Apache. (n.d.). *Hadoop*. Retrieved from Hadoop Streaming: http://hadoop.apache.org/docs/r1.1.2/streaming.html

Apache. (n.d.). *k-Means clustering - basics*. Retrieved from Mahout: https://mahout.apache.org/users/clustering/k-means-clustering.html

Blei, D. M. (n.d.). *Introduction to Probabilistic Topic Model.*

Buckley, G. S. (n.d.). Term-weighting approaches in automatic text retrieval. *Information Processing & Management*.

Chambers, N., Wang, S., & Jurafsky, D. (n.d.). *Classifying Temporal Relations Between Events*. Retrieved from Stanford: http://web.stanford.edu/~jurafsky/acl07-chambers.pdf

Jurafsky, D., & Martin, J. H. (2009). *Speech and Language Processing: An Introduction to Natural Language Processing, Computational Linguistics, and Speech Recognition.* Prentice Hall.

K-means Clustering. (n.d.). Retrieved from Wikipedia: http://en.wikipedia.org/wiki/K-means_clustering

Madnani, N. (n.d.). Retrieved from Getting Started on Natural Language Processing with Python: http://desilinguist.org/pdf/crossroads.pdf

Multi-instance Multi-label Relation Extraction. (n.d.). Retrieved from The Stanford Natural Language Processing Group : http://nlp.stanford.edu/software/mimlre.shtml

nltk. (n.d.). *nltk tokenize package*. Retrieved from NLTK 3.0 documentation: http://www.nltk.org/api/nltk.tokenize.html

Perkins, J. (n.d.). *Part of Speech Tagging with NLTK Part 4 – Brill Tagger vs Classifier Taggers*. Retrieved from StreamHacker: http://streamhacker.com/2010/04/12/pos-tag-nltk-brill-classifier/

Python. (n.d.). *Python Documentation*. Retrieved from https://docs.python.org/2.7/

Ramage, D., Hall, D., Nallapati, R., & Manning, C. D. (n.d.). *The Stanford Natural Language Processing Group.* Retrieved from http://nlp.stanford.edu/pubs/llda-emnlp09.pdf

S. Bird, E. K. (n.d.). *Natural Language Processing with Python.* Retrieved from http://www.nltk.org/book_1ed/

scikit. (n.d.). *scikit learn*. Retrieved from 1.8 Decision Trees: http://scikit-learn.org/stable/modules/tree.html

Stanford. (n.d.). *The Stanford Natural Language Processing Group* . Retrieved from Named Entity Recognition (NER) and Information Extraction (IE): http://nlp.stanford.edu/ner/

Washington. (n.d.). *Open Information Extraction Software*. Retrieved from Reverb: http://reverb.cs.washington.edu/