Event and Temporal Information Extraction towards Timelines of Wikipedia Articles

Rachel Chasin
Department of Computer Science
University of Colorado at Colorado Springs
Colorado Springs, Colorado 80918

***Abstract*—This paper explores the task of creating a timeline for historical Wikipedia articles, such as those describing wars, battles, and invasions. It focuses on extracting only the major events from the article, particularly those associated with an absolute date. Existing tools extract all possible events, while we write tools to identify time expressions and anchor them in real time. From this set of all events, we identify the major ones using a classifier. We then place these events on a timeline and label them with a time interval as small as possible. The timeline is integrated into an online user interface that displays named entities for each event and finds its locations on a map; we separately list named entities and locations mentioned in the article but not around any event.**

I. Introduction

EVENT and temporal information extraction from plain text is a crucial task for natural language processing and knowledge management, particularly in the tasks of summarization and question-answering. Topic summarization must pick out the important events in one or more stories to yield the best summary with the least extraneous information. Question-answering tools must be able to answer queries about dates, durations, and even relative times, whether in natural language or with a set query type ("When did the Civil War end?", "How long was the Battle of Gettysburg?", "How many years were between the Civil War and World War I?").

The possible domains for temporal information extraction are numerous and varied; it is especially useful in the news domain and for patient reports in the medical domain. This paper discusses its use in documents that describe historical events. The problem it addresses can be roughly broken down into two large components: extracting only important events, and relating them via the temporal expressions in the document so they can be viewed on a timeline.

II. Related Research

Even setting aside the problem of automating the process, identifying and representing the temporal relations in a document is a daunting task for humans. Several structures have been proposed, from Allen's 1983 interval notation and 13 relations [1] and variations on it using points instead, to a constraint structure for the medical domain in [2] (patient hospital reports). The most recent way to represent the relations is a set of XML tags called TimeML.

Rachel Chasin is an undergraduate at Massachusetts Institute of Technology, Cambridge, MA, 02139 e-mail: rchasin@mit.edu.

Software has been developed ([3], [4]) that identifies events and time expressions, and then generates relations among them. Events generally consist of most verbs in the document, but also include some nouns. The link generation is done differently depending on the system, but uses a combination of rule-based components and classifiers.

Research on event extraction has often focused on identifying the most important events in a set of news stories ([1], [5]). The advantage of this domain is that important information is usually repeated in many different stories, all of which are being examined. This allows algorithms like TF-IDF to be used. In this Wikipedia project, there is only one document per specific topic, so these algorithms cannot be used. There has also been research into classifying events into specific categories and determining their attributes and argument roles by [6] (submitted to the ACE task in 2005). Another issue addressed by the ACE task is event coreference, determining which descriptions of events in a document refer to the same actual event.

III. Our Approach

The task of making a timeline for a Wikipedia article lends itself well to separation into two subtasks - identifying whether an event is important or not, and putting these events on a timeline. We focused on each of these separately. After the timeline data was generated, we used the existing software Simile[1] to graphically represent it, and worked with a group researching the identification and mapping of geospatial entities to combine the two visualizations.

A. Important Events

For determining important events, we first run the EVITA program[2] described in [7] on the article, which labels all possible events in the TimeML format. Specifically, it places XML "EVENT" tags around single words defining events; these may be verbs or nouns, as seen in Figures 1a and 1b. Each event has a class; in this case, "fought" is an occurrence. Other classes include states and reporting events. For our purposes, occurrences will be the most important because those are the kinds of events generally shown on timelines.

[1]Simile was originally authored by David Franois Huynh and is available at http://www.simile-widgets.org/timeline/.

[2]The entire TARSQI Toolkit is freely downloadable upon email request; see: http://timeml.org/site/tarsqi/toolkit/download.html

EVITA recognizes events by first preprocessing the document to tag parts of speech and chunk it, then examining each verb, noun, and adjective for linguistic (using rules) or lexical (using statistical methods) properties that indicate it is an event. Instances of events also have more properties that help temporal relations to be established in later processing steps; these properties include tense and sometimes modality.

The Battle of Fredericksburg , fought_e1 in and around Frederick
15 , 1862_t3 , between General Robert E . Lees Confederate Army o
Potomac , commanded_e2 by Maj . Gen . Ambrose E . Burnside , is
battles_e4 of the American Civil War . The Union Army suffered_e5
December 13 against entrenched Confederate defenders on the hei
their campaign_e8 against the Confederate capital of Richmond .

Portion of an article with events tagged by EVITA (words followed by e_i)

```
The Battle of Fredericksburg, <EVENT
class="OCCURRENCE" eid="e1">fought</EVENT>
in and around Fredericksburg, Virginia [...]
```

Figure 1b: A portion of the XML representation of Figure 1a.

For this task, an event is considered to be the sentence containing it. These sentences will eventually be used as the text for each event on the timeline. Each event/sentence is classified as important or not using a classifier trained with a set of mostly word-level and sentence-level features. On the first attempt at classification, one classifier used numerical features and another used purely binary features, translating the former into the latter by putting them in or out of ranges. The numerical classifier used 16 features and the purely binary one used 19. The numerical classifier's features are listed in Table 1 and the purely binary classifier used the same ones with numerical features split into ranges. Many features had to do with the characteristics of the event word in the sentence (part of speech, grammatical aspect, distance from a named entity, etc.) or the word by itself (length). Some had to do with the sentence as a whole (for example, presence of negation and presence of digits).

Two were also related to the article as a whole - position of the sentence in the document, and similarity of the event word to article "keywords." These keywords were taken as the first noun and verb or first two nouns of the first sentence of the article. These were chosen because the first sentence of these historical narratives often sums up the main idea and will often therefore contain important words. In the articles of our corpus, these are often "war," "conflict," or "fought," for example. An event word's similarity to one of these words may having a bearing on its importance. The decision to use two keywords helps in case one of the words is not a good keyword; only two are used because finding similarity to a keyword is expensive in time. Similarity is measured using the "vector pairs" measure from the WordNet::Similarity Perl module, proposed in [8]. This calculates the similarity of the vectors representing the glosses of the words to be compared. This measure was chosen because it was one of the few that can calculate similarity between words from different parts of speech, which was necessary for this feature.

Important Event Classifier Features, version 1
Distance (characters) from nearest named entity
Digit presence in sentence
Negation presence in sentence
Position (by token) of event word in sentence
Position of sentence in article
Length of event word
Similarity of event word to keywords
Event word is capitalized
Event word is noun
Event word is verb
Event word is in past tense
Event word is in infinitive
Event word has no tense (for nouns)
Event word has perfective aspect
Event word has positive polarity
Event word is of the event class "occurrence"

Table 1: A list of features for classifying events as important or not

Upon training and testing, SVMs using these features performed poorly, particularly in precision (see Experiments), so we altered the approach. Because events are considered to be the sentences containing them, that is, we are trying to decide which sentences are important based on the events EVITA extracts from them, we changed the set of features to apply to sentences. The new set of features discarded some features that did not make sense to apply to the sentence; included the same sentence-level features as well as some new ones; and changed some word-level to sentence-level features by adding, averaging, or taking the maximum of the word-level features for each event in the sentence. The new features can be seen in Table 2.

The most different feature added was a feature based on TextRank ([9]), an algorithm developed by Mihalcea that ranks sentences based on importance and is used in text summarization. TextRank works using the PageRank algorithm developed by Google. While PageRank works on web pages and the links among them, TextRank treats each sentence like a page, creating a weighted, undirected graph whose nodes are the document's sentences. Edge weights between nodes are determined using a function of how similar the sentences are. After the graph is created, PageRank is run on it which ranks the nodes in order of essentially how much weight points at them (taking into account incident edges and the ranks of the nodes on the other sides of these edges). Thus sentences that are in some way most similar to most other sentences get ranked highest. We wrote our own implementation of TextRank with our own function for similarity between two sentences. Our function automatically gave a weight of essentially 0 if either sentence was shorter than a certain threshold (we chose 10 words). For all others, it

calculated the "edit distance" between the sentences, treating words (rather than characters) as the units to be compared and calling two words equal if their stems are equal. The similarity was then chosen as the sum of the sentence lengths divided by their edit distance.

Named entities were also considered more important to the process than before. Instead of just asking if the sentence contained one, the some measure of the importance of the named entity is calculated and taken into account for the feature. This is done by counting the number of times the named entity is mentioned (people being equal if their last names are equal, and places being equal if their most specific parts are equal). This total is then normalized for the number of named entities in the article.

Important Event Classifier Features, final version
Presence of an event in the perfective aspect
Percent of events in the sentence with class "occurrence"
Digit presence
Maximum length of any event word in the sentence
Sum of the Named Entity 'weights' in the sentence (NE weight being the number of times this NE was number of mentioned in the article divided by the all NE mentions in the article)
Negation presence in the sentence
Number of events in the sentence
Percent of events in the sentence that are verbs
Position of the sentence in the article normalized by the number of sentences in the article
Maximum similarity (as previously described) of any event word in the sentence
Percent of events in the sentence that are in some past tense TextRank rank of the sentence in the article, divided by the number of sentence in the article
Number of "to be" verbs in the sentence

Table 2: A final list of features for classifying events as important or not

B. Temporal Relations

The rest of TTK creates event-event and event-time links (called TLINKs). Times are identified by GUTime, another part of the toolkit, which marks them with "TIMEX3" XML tags. These include attributes like type of expression (DATE, TIME, etc.) and value (when possible to tell). A time identified in Figure 1a was "1862." Some events, then, will be anchored to time expressions, and some to other events. These are represented by the TLINK XML tag, which has attributes like the type of relation. There are other links, called SLINKs, that describe modal relations between events, but these are not as relevant to the task, though perhaps useful for filtering out events that get tagged but do not actually happen in the narrative (ex. what someone thought would happen). An example of each kind of TLINK is shown in Figure 2. The first represents that fact that the event with ID 1 is related to the time with ID 3 by "before"; the second also represents "before," between events 5 and 6.

```
<TLINK eventInstanceID="ei1" lid="l9"
origin="CLASSIFIER 0.995194" relType="BEFORE"
relatedToTime="t3"/>
<TLINK eventInstanceID="ei5" lid="l10"
origin="CLASSIFIER 0.999577" relType="BEFORE"
relatedToEventInstance="ei6"/>
```

Figure 2: Two TLINKs.

In fact, almost all the TLINKs we encountered when trying out the TTK linking program were "before" links, and they gave little to no more information reiterating the order of the events in the text. While this is generally accurate, since it can be done without the program, we decided not to use TTK for the temporal relation processing.

As a step before beginning this work, we tried a simple approach just using regular expressions to extract times. The results of using extensive regular expressions versus using the GUTime part of the TTK showed that the regular expressions pull out many more (complete) dates and times. For example, in Figure 1a, GUTime only finds 1862, while the regular expressions would find a range from December 11 1862 to December 15 1862. Because of this, we decided to use our own program based in regular expressions rather than GUTime. A flaw present in our program and not in GUTime is its ability to only pick out one time expression (point or interval) per sentence. This is consistent with our current view of events as sentences, although it would arguably be better to duplicate the sentence's presence on a timeline while capturing all time expressions present in it. We do not think it would be overly difficult to implement this change although it might cause problems for the extraction of expressions that have parts that can be far away from each other in the sentence.

GUTime also attempts to compute real values for the times in the ISO8601 standard, but usually does it in relation to the document creation time; this is useful for news articles, for which a creation time is provided, but is detrimental in our case, as it assumes the current date. Instead, to anchor time expressions to real times - specifically to a year - we have used a naive algorithm that chooses the previous anchored time's year. We heuristically choose the most probable year out of the previous anchored time's year and the two adjacent to it by looking at difference in the month, if it is given. For example, a month that is more than five months earlier than the anchoring's time's month is probably in the next year rather than the same year (with "The first attack occurred in December 1941. In February, the country had been invaded," it is probably February 1942). In addition to the year, if the time needing to be anchored lacks more fields, we fill them with as many corresponding fields from the anchoring time as it has.

Each time expression extracted is considered to have a beginning and an end, at the granularity of days (though we do extract times when they are present). Then the expression "December 7, 1941" would have the same start and end point,

while the expression ”December 1941” would be considered to start on December 1 and end on December 31. Similarly, modifiers like “early” and “late” change this interval according to common sense; for example, “early December” corresponds to December 1 to December 9. While these endpoints are irrelevant to a sentence like, “The Japanese planned many invasions in December 1941,” it is necessary to have exact start and end points in order to plot a time on a timeline. Thus while the exact start and end days are often arbitrary for meaning, they are chosen by the extractor.

Some expressions cannot be given a start or end point at all. For example, “The battle began at 6:00 AM” tells us that the start point of the event is 6:00AM but says nothing about the end point. Any expression like this takes on the start or end point of the interval for the entire event the article describes (for example, the Gulf War). This interval is found by choosing the first beginning time point and first ending time point that are anchored directly from the text, and is logically probable to find the correct span. While this method is reasonable for some expressions, many of them have an implicit end point somewhere else in the text, rather than stretching until the end of the article’s event. It would likely be better to instead follow the method we use for giving times to sentences with no times at all, described below.

After initial extraction of time expressions and finding a tentative article span, we discard times that are far off from either end point, currently using 100 years as a cutoff margin. This helps avoid times that are obviously irrelevant to the event, as well as expressions that are not actually times but look like they could be (for example, “The bill was voted down 166-269” which looks like a year range). We also discard expressions with an earlier end date than start date, which helps avoid the latter problem.

Despite the thoroughness of the patterns we look for in the text, the majority of sentences still have no times at all. However, they may still be deemed important by the other part of our work, so must be able to be displayed on a timeline. Here we exploit the characteristic of historical descriptions that events are generally mentioned in the order in which they occurred. For a sentence with no explicit time expression, the closest (text-wise) sentence on either side that does have a time expression is found, and the start times of those sentences are used as the start and end time of the unknown sentence. There are often many unknown sentences in a row; each one’s position in this sequence is kept track of so that they can be plotted in this order, despite having no specific date information past the interval, which is the same for them all. Sometimes the start and end time we get in this manner are invalid because the end time is earlier than the start time. In this case, we look for the next possible end time (the start time of the next closest sentence with a time expression). If nothing can be found, we use a default end point. This default end point is chosen as the last time in the middle N of the sorted times, where N is some fraction specified in the program (currently chosen as 1/3). We do this because times tend to be sparse around the ends of the list of sorted times, since there are often just a few mentions of causes or effects of the article’s topic.

IV. Experiments

A. Important Events

Given tagged data, the “important event extraction” was simple to test since it is a classification problem. It required a set of Wikipedia articles that were hand-tagged as to whether each event word identified by EVITA was part of an important event. An event word being part of an important event meant that the sentence containing it was important and that it contributed to this importance. Because there is no good objective measure of importance (except, perhaps, presence on a manually generated timeline from some other source), we asked multiple volunteers to tag the articles to avoid bias. The subjectivity was a real problem, however. We gave the annotators guidelines for what events were and were not important, but there was still a lot of confusion and disagreement. The most basic guideline was to judge whether you would want the event on a timeline of the article. Other guidelines included to not tag it if it was more of a state than an action, and to tag all event words that referred to the same event. Each article was annotated by more than one person so that disagreements could be resolved using a majority vote. There were 13 articles annotated in total and the breakdown of numbers of annotators was: 1 article annotated by 5 people, 4 articles annotated by 4 people, 6 articles annotated by 3 people, and 2 articles annotated by 2 people.

Originally, different words tagged as events by EVITA were considered different events even if they were in the same sentence. During this stage, an event was labeled important for the classifier if at least half of its annotators tagged it as such. Inter-annotator agreement was calculated pair-wise and averaged over all pairs of annotators of an article. One annotator’s tagging was taken as ground truth, and each other annotator’s accuracy measures were calculated against that. Over all articles, all “ground truth” annotators for that article, and all annotators compared against them, the average F1-score was 42.6%, indicating great disagreement.

An SVM using a radial basis function kernel was trained on 11 articles, with 2 left for testing. This yielded precision, recall, and f-measure scores. The 11 training articles consisted of 5602 event words and the 2 testing articles consisted of 3227 event words (1503 and 1724). The SVM was trained with the LIBSVM software ([10]) using a binary classifier with a radial basis function kernel. The parameters for the kernel and cost constant were found with one of LIBSVM’s built-in tools.

After the initial training and testing, the SVM simply performed as a majority-class classifier, since the number of negative examples - events tagged as unimportant - grossly outnumber the positive examples. This obviously yielded high accuracy and high precision, but almost 0 recall.

Two common methods for combatting sample inequality of classes were considered and tried. One was SMOTE ([11]), Synthetic Minority Over-sampling Technique, presented by Chalwa in 2002. This technique reads in positive examples and creates artificial positive examples close to the real ones. This is better than simple oversampling because it creates different points rather than repeating the same points (although it does end up doing some repetition). We used SMOTE to

generate enough new positive examples so that the numbers of positive and negative examples were approximately equal for each article. The other method to prevent simple majority-class classification was weighting the costs of misclassifying positive and negative examples differently. By penalizing misclassified positive examples more than negative ones, the SVM gets trained to make fewer of those errors, increasing recall at the expense of precision. Following [12], we tried both of these methods together, but found that oversampling with SMOTE in addition to having different costs ended up with slightly lower scores. Further, for the numerical feature SVM (as opposed to the pure binary feature SVM), the best cost ratio with oversampled data was just 1. The precision-recall data points for different conditions are given in Figure 3. The different data sets represent the two test articles, each compared with predictions from an SVM that was or was not ("unsmoted") trained on oversampled data.

The results of testing the modified SVM showed low precision and medium to high recall. For the SVM with binary features, the best case was with data that was not oversampled, using a positive example cost to negative example cost ratio of 3. The average F-score between the two testing articles was 33.4% in this case, with precisions of 22.0% and 25.8%, and recalls of 54.0% and 56.8%. The SVM with numeric features performs similarly, with the highest average F-score being 33.3% on data that was not oversampled with a cost ratio of 2. The precisions for this are 29.0% and 27.8%, and the recalls are 40.9% and 39.5%.

We thought these low scores partially stemmed from the fact that annotators may have considered the same sentences important but tagged different event words within them, thus not giving accurate test data. Since we had told them to decide whether sentences were important or not and we were planning to only display sentences on the article timeline, we decided to classify at sentence level as well.

The annotated data was again processed to give a test set of important and unimportant sentences. A sentence was labeled important for the classifier if either (1) some event in the sentence was marked important by every annotator, or (2) at least half of the events in the sentence were marked important by at least half of the annotators. Using this criteria, over the 13 articles 696 sentences were labeled important and 1823 unimportant. The pairwise inter-annotator agreement was recalculated for this new method of labeling and the average F-score between annotators rose to 66.8% (lowest being 47.7% and highest being 78.5%).

As described above, a different set of features was chosen and calculated for each of the 13 articles. SVMs were trained and tested using 10-fold cross-validation. Rather than breaking up the data into training and test sets by article, this time the 2519 sentences were divided into 10 equal sets plus one remainder set that was not used in training or testing during cross-validation.

To determine the best SVM model, the parameter space was partially searched for cost of missing examples (c), the parameter gamma of the radial basis kernel function (g), and the weight ratio of cost for missing positive examples to cost for missing negative examples (w). For any given choice of

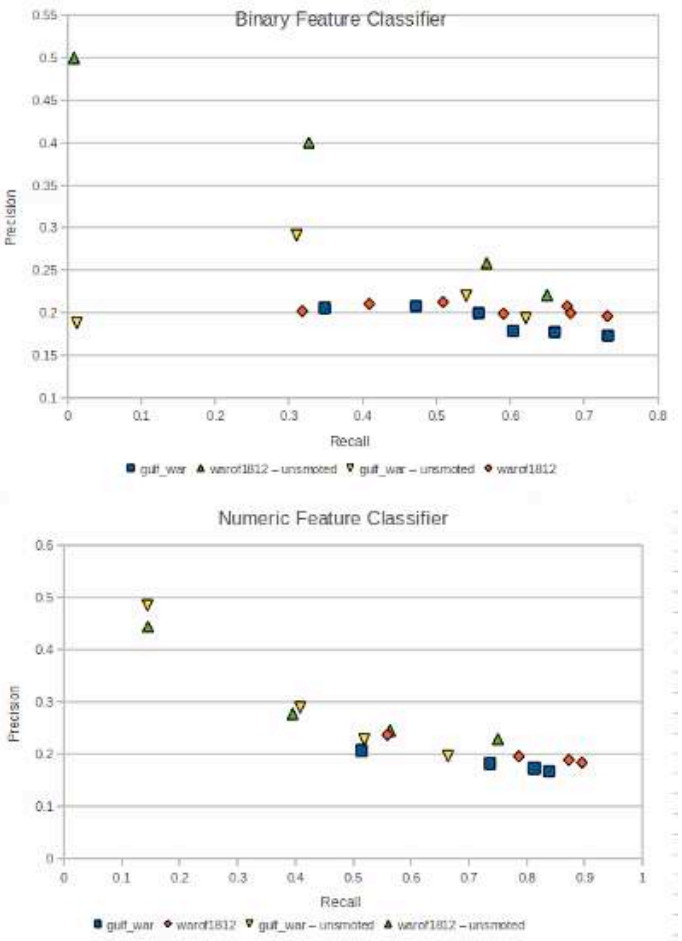

Figure 3: Precision and recall for original SVM results

c, g, and w, an SVM was trained on each cross-validation training set and tested on the corresponding test set, using libsvm. The resulting F-scores were averaged over the different sets and recorded. The search space was explored starting at c=1.0, g=1.0, w=1. c and g went down by powers of 2 and w was incremented. This portion of the space was explored because in the original classification SVM training, the optimal c and g (as found by a libsvm tool) were usually 0.5 and 0.5. Fixing c and w, the next g would be chosen until F-scores no longer went up. w was usually set to 2 almost immediately because of low scores for w=1. This was repeated for w up to 3 (as 4 produced slightly poorer results), and then the next value of c was chosen. The results of these tests are shown in Table 3. 0.5 and 0.5 were optimal values for the event-word-level classifiers, however, so a broader search may have been beneficial.

The best results were at 51.0% F-score and came from a few different choices of parameters. The set c=1.0, g=0.125, w=3 was chosen and the final model was trained with those parameters on the entire data set.

c	g	w	F-score
1.0	1.0	1	29.0
1.0	1.0	2	46.1
1.0	0.5	2	46.9
1.0	0.25	2	47.5
1.0	0.125	2	48.0
1.0	0.0625	2	49.0
1.0	0.03125	2	48.0
1.0	1.0	3	46.2
1.0	0.5	3	48.9
1.0	0.25	3	50.1
1.0	0.125	3	51.0
(p:41.06066, r:75.0522, f:51.0428)			
1.0	0.0625	3	50.9
0.5	1.0	2	45.2
0.5	0.5	2	47.1
0.5	0.25	2	47.5
0.5	0.125	2	48.2
0.5	0.0625	2	48.1
0.5	1.0	3	48.0
0.5	0.5	3	49.2
0.5	0.25	3	50.55
0.5	0.125	3	51.0
(p:41.14866, r:74.7939, f:51.03596)			
0.5	0.0625	3	51.0
(p:41.35864, r:73.9689, f:51.01095)			

Table 3: SVM parameter searching test results (averaged over cross-validation sets)

B. Temporal Relations

Testing the times is significantly more difficult, since the intervals generated by this program are certainly different than the ones intended by the article representing events with times is difficult even for human annotators. Instead, we will use a measure proposed by Ling and Weld in 2010 ([4]) that they term "Temporal Entropy" (TE). This indirectly measures how large the intervals generated are, smaller, and therefore better, ones yielding smaller TE. Different Wikipedia articles have different spans and time granularities, and therefore TE varies greatly among them. For example, a war is usually measured in years while a battle is measured in days. An event whose interval spans a few months in a war article should not be penalized the way that span should be in a battle article. It is then be necessary to normalize the TE. To do this, we divide the length of the event's interval in seconds by the length in days of the unit that is found for display on the timeline as described in the visualization section.

Temporal entropy does not give all the information, however. Spot-checking of articles reveals that many events - particularly those whose times were estimated - are not in the correct interval at all. A useful but impractical additional test, which we have not performed, would be human examination

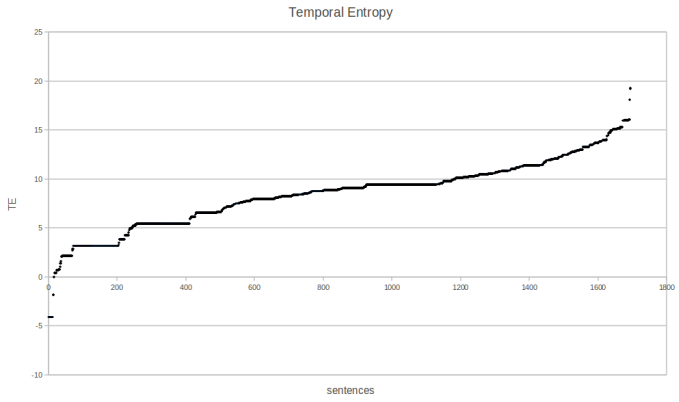

Figure 4: Temporal Entropy graph - the temporal entropies were sorted in increasing order for plotting. Temporal entropy is in log(seconds/days)

of the results to tell whether each event's assigned interval includes or is included in its actual interval. Then those that fail this test could be given maximum temporal entropy, to integreate this test's results into the former results.

V. Web Visualization and Relation to Geospatial and Other Named Entities

A website is maintained where a user can view already processed Wikipedia articles or request the processing of new ones. The website allows users to view several aspects of the article. An example of what a user sees is in Figure 5. One aspect is the timeline of events whose creation has been the subject of this paper. Clicking any event on the timeline displays an infobox with further information about it, including the full sentence. The other major aspect is a map (via the Google Maps API) and list of locations the article mentions. These locations are processed using a separate system, and are the results of running a named entity recognizer (currently the Stanford NER) on the article and then geocoding and disambiguating the results. People and organizations, as extracted by the named entity recognizer, are also recorded. The locations, people, and organizations keep track of which sentences they are associated with, and can therefore be related to the timeline events. An example of this is seen in Figure 6, which shows a sample event infobox that includes associated locations, people, and organizations. If the user clicks on a listed location, the map centers on and zooms to that location. If the user clicks a location marker on the map, an infobox with the location's name and original context is displayed, as seen in Figure 7. Pages and their links to events and entities are stored in a database; the text is stored in one place and upon a user's request to view or process an article, the server generates the necessary data.

For the visual representation of the timeline, we use the Simile Timeline software, which allows the inclusion of a timeline on a website. The timeline is set up in javascript, which specifies the number of bands and their paramters, such as scale and center date. It also allows highlighting and magnification of certain regions ("hot zones"). These features

are crucial to a readable timeline. Data is provided, in our case, in an XML format. Both the javascript and XML depend on the article, and they are generated upon a user's request to view a timeline. Most of the expensive computing is done when putting an article's information into the database, so a request to just view is fast.

Events that contain time expressions are simple to plot, as they have always a given start and end date. The events whose endpoints were guessed are placed in the interval they were given but are displayed at a subinterval that keeps them in order with respect to other such events in that interval. Then, the times that the timeline shows are far from certain, but the order is more likely to be correct.

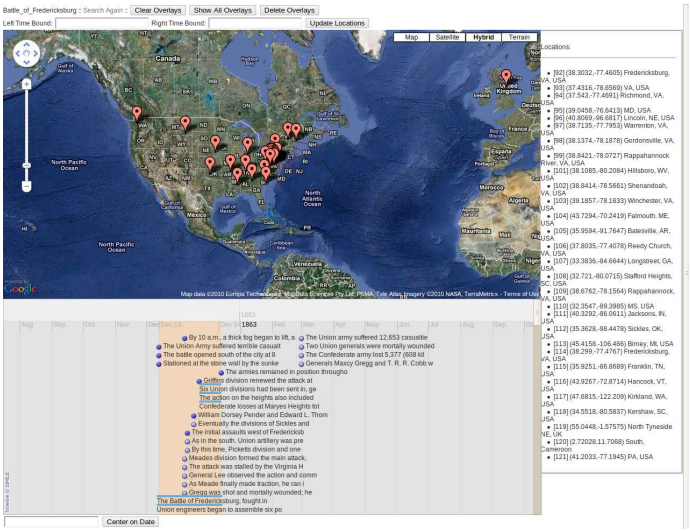

Figure 5: Example of an article's visualization

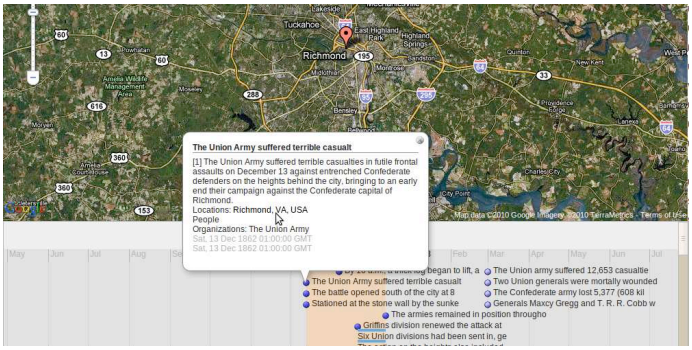

Figure 6: Event infobox in the visualization, after clicking "Richmond, VA, USA"

The identification of "hot zones" is an important part of the process of visualization, because the articles often give some background and after-effects of their main events, but most of the sentences cluster in one or more important intervals. We originally only created one hot zone and made it the "middle of the article times" interval described previously. This was not optimal, because times often cluster into multiple, densely populated sections. To generate the endpoints for more than one hot zone, we sort the times for an article and examine the time differences (in days) between consecutive

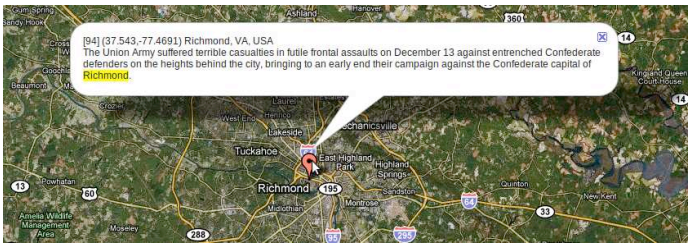

Figure 7: Geospatial entity infobox in the visualization

times. Long stretches of low differences indicate dense time intervals. An example of these differences graphed for an article with obvious clusters is shown in Figure 8. Figure 9 shows part of its corresponding timeline. It is not necessary for the differences to be zero, just low, so a measure was needed for how low was acceptable. We chose to remove outliers from the list of time differences and then take the average of the new list. To remove outliers, we proceed in a common manner and calculate the first and third quartiles ($Q1$ and $Q3$) and then remove values greater than $Q3 + 3 * (Q3 - Q1)$ or less than $Q1 - 3 * (Q3 - Q1)$. The latter quantity was usually zero, so this ended up removing high outliers. This average was the threshold below which a difference was called low. We also had to ensure that too many hot zones were not generated, so we chose a threshold for the number of consecutive differences that had to be low for the interval to be a hot zones; we chose 10, based on what we felt would be appropriate for a timeline.

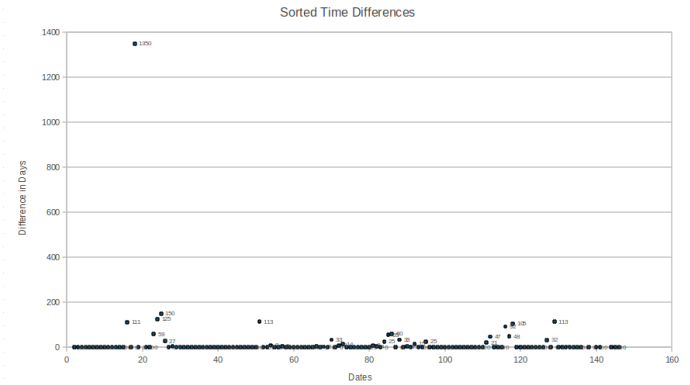

Figure 8: Differences between consecutive times in the sorted time list for Mexican-American War

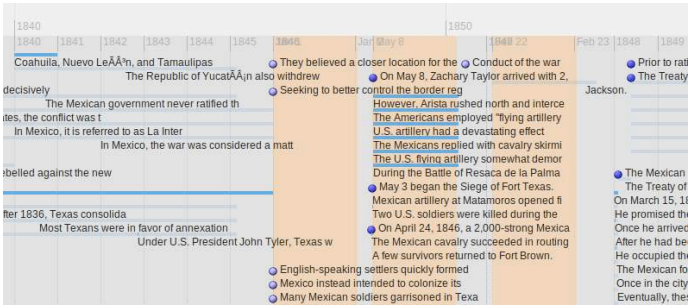

Figure 9: Timeline for Mexican-American War, containing 3 hot zones

VI. Conclusions

The difficulty in classifying important events may lie with the features we chose or with the idea that one can even classify events as "important" or "not important." From the performance improvement for both machine and humans when switching from words to sentences, it seems that it is much more natural to ask about important sentences. This moves into the territory of text summarization, and it might be more pertinent to rank the sentences than to put them into two classes.

The temporal relation extraction turns out to work fairly well for this domain of historical narratives because the sentences are often ordered in the text similarly to their temporal order. In another domain, even one rich in temporal information like biographies, it might not do as well. Further, due to the algorithms for anchoring times to years and for giving times to sentences without them, errors tend to build up and carry over a lot; some kind of check or reset condition, if one could be developed, would have helped accuracy.

Finding and displaying links among events, people, organizations, and locations succeeded since the events are just considered sentences. The visualization of this has good usability and is a useful tool to people analyzing the articles.

VII. Future Work

The mediocre performance of the important event classifier could be improved upon in the future. It is possible that more features would help it better classify. Ideas for sentence features that were not implemented included calculating the current features for the adjacent sentences. Classifiers other than SVMs were also considered, and this option could be explored. Since the task is similar to text summarization, methods like TF/IDF that are used for summarization could be adapted to one document and tried out. It is also possible that the definition of "important" is not objective or specific enough, and further investigation into this would be useful.

While classifying more events positively tends to increase scores, visualizing so many is not desirable. To reduce the number that get selected, we could reduce the number that are classified at all by doing preprocessing on the sentences. We suggest running TextRank on the sentences originally and then picking some fraction of the top ranked sentences to classify.

The classifier also takes more time than is desirable to calculate the features. Long articles can take 2-3 minutes, and since these articles are being processed while users wait for them, the current run times are not optimal. Using multithreading on processing articles would make some improvement. There are three specific features that take the most time to calculate - similarity, named entity weight, and TextRank rank. Experimenting with removing any of these features while preserving accuracy could make a more efficient classifier.

The place for the most improvement is likely the temporal portion of the process. Most of the algorithms we use here are naive and do not make use of existing tools. The regular expression extractor works fairly well, but needs expansion to handle relative expressions like "Two weeks later, the army invaded." To implement this with the same accuracy that the extractor currently has should not be hard, as when the expression is recognized, the function that it implies (in this example, adding two weeks) can be applied to the anchoring time. It also cannot currently handle times B.C.E. More difficult would be a change to the program to have it extract multiple expressions per sentence (beyond the range-like expressions it already finds). If events are ever considered smaller than sentences, this is crucial. Having events smaller than sentences would also allow the use of Ling and Weld's Temporal Information Extraction system ([4]), which temporally relates events within one sentence.

The visualization could also be improved with some changes to Simile's code for displaying a timeline, particularly the width of the timeline, which does not automatically adjust for more events. Because the important event classifier positively identifies a lot of sentences that are often close in time, the listing of events on the timeline overflows the space, so more must be allotted even for sparser articles. This could be improved with automatic resizing or zooming. Another improvement would be to create event titles that better identify the events since they are currently just the beginnings of the sentences. Finally, more features could be added to the visualization, especially in terms of filtering by time or location range.

References

[1] J. F. Allen, "Maintaining knowledge about temporal intervals," *Commun. ACM*, vol. 26, no. 11, pp. 832–843, November 1983. [Online]. Available: http://dx.doi.org/10.1145/182.358434

[2] L. Zhou, G. B. B. Melton, S. Parsons, and G. Hripcsak, "A temporal constraint structure for extracting temporal information from clinical narrative." *J Biomed Inform*, September 2005. [Online]. Available: http://dx.doi.org/10.1016/j.jbi.2005.07.002

[3] M. Verhagen and J. Pustejovsky, "Temporal processing with the tarsqi toolkit," in *22nd International Conference on on Computational Linguistics: Demonstration Papers*. Manchester, United Kingdom: Association for Computational Linguistics, August 2008, pp. 189–192.

[4] X. Ling and D. Weld, "Temporal information extraction," in *Proceedings of the Twenty-Fourth Conference on Artificial Intelligence (AAAI-10)*, Atlanta, GA, July 2010.

[5] R. Swan and J. Allan, "Extracting significant time varying features from text," in *CIKM '99: Proceedings of the eighth international conference on Information and knowledge management*. New York, NY, USA: ACM, 1999, pp. 38–45. [Online]. Available: http://dx.doi.org/10.1145/319950.319956

[6] D. Ahn, "The stages of event extraction," in *Proceedings of the Workshop on Annotating and Reasoning about Time and Events*. Sydney, Australia: Association for Computational Linguistics, July 2006, pp. 1–8. [Online]. Available: http://www.aclweb.org/anthology-new/W06/W06-0901.bib

[7] R. Saur, R. Knippen, M. Verhagen, and J. Pustejovsky, "Evita: a robust event recognizer for qa systems," in *Proceedings of the conference on Human Language Technology and Empirical Methods in Natural Language Processing*. Vancouver, British Columbia, Canada: Association for Computational Linguistics, October 2005, pp. 700–707.

[8] T. Pedersen, S. Patwardhan, and J. Michelizzi, "Wordnet::similarity: measuring the relatedness of concepts," in *HLT-NAACL '04: Demonstration Papers at HLT-NAACL 2004 on XX*. Morristown, NJ, USA: Association for Computational Linguistics, 2004, pp. 38–41.

[9] R. Mihalcea and P. Tarau, "Textrank: Bringing order into texts," in *Proceedings of EMNLP 2004*, D. Lin and D. Wu, Eds. Barcelona, Spain: Association for Computational Linguistics, July 2004, pp. 404–411.

[10] C.-C. Chang and C.-J. Lin, *LIBSVM: a library for support vector machines*, 2001, software available at http://www.csie.ntu.edu.tw/ cjlin/libsvm.

[11] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "Smote: Synthetic minority over-sampling technique," *Journal of Artificial Intelligence Research*, vol. 16, pp. 321–357, 2002. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.18.5547

[12] R. Akbani, S. Kwek, and N. Japkowicz, "Applying support vector machines to imbalanced datasets," in *Machine Learning: ECML 2004*, 2004, pp. 39–50. [Online]. Available: http://www.springerlink.com/content/pa57eam5t5dkem4h