# Extracting and Displaying Temporal and Geospatial Entities from Historical Articles

<table>
<thead>
<tr>
<th>Journal:</th>
<th><em>The Computer Journal</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Manuscript ID:</td>
<td>COMPJ-2012-12-0833.R1</td>
</tr>
<tr>
<td>Manuscript Type:</td>
<td>Original Article</td>
</tr>
<tr>
<td>Date Submitted by the Author:</td>
<td>n/a</td>
</tr>
<tr>
<td>Complete List of Authors:</td>
<td>Chasin, Rachel; Massachusetts Institute of Technology, Computer Science Woodward, Daryl; University of Colorado, Colorado Springs, Electrical and Computer Engineering Witmer, Jeremy; MITRE, Kalita, Jugal; University of Colorado at Colorado Springs, Department of Computer Science</td>
</tr>
<tr>
<td>Key Words:</td>
<td>Information Extraction, Temporal Extraction, Geospatial Entity Extraction</td>
</tr>
</tbody>
</table>
Extracting and Displaying Temporal and Geospatial Entities from Articles on Historical Events

Rachel Chasin\textsuperscript{1}, Daryl Woodward\textsuperscript{2}, Jeremy Witmer\textsuperscript{3}, and Jugal Kalita\textsuperscript{4}

\textsuperscript{1}Department of Computer Science, Massachusetts Institute of Technology, Cambridge, MA, 02139; \textsuperscript{2}Department of Electrical and Computer Engineering, University of Colorado, Colorado Springs, CO 80918; \textsuperscript{3}The MITRE Corporation, 49185 Transmitter Road, San Diego, CA 92152-7335; \textsuperscript{4}Department of Computer Science, University of Colorado, Colorado Springs, CO 80918

Email: \textsuperscript{1}rchasin@mit.edu, \textsuperscript{2}daryl.woodward@gmail.com, \textsuperscript{3}jwitmer@mitre.org, \textsuperscript{4}jkalita@uccs.edu

This paper discusses a system that extracts and displays temporal and geospatial entities in text. The first task involves identification of all events in a document followed by identification of important events using a classifier. The second task involves identifying named entities associated with the document. In particular, we extract geospatial named entities. We disambiguate the set of geospatial named entities and geocode them to determine the correct coordinates for each place name, often called grounding. We resolve ambiguity based on sentence and article context. Finally, we present a user with the key events and their associated people, places, and organizations within a document in terms of a timeline and a map. For purposes of testing, we use Wikipedia articles about historical events, such as those describing wars, battles, and invasions. We focus on extracting major events from the articles, although our ideas and tools can be easily used with articles from other sources such as news articles. We use several existing tools such as Evita, Google Maps, publicly available implementations of SVM, HMM and CRF, and the MIT SIMILE Timeline.

Keywords: Information Extraction, Temporal Extraction, Geospatial Entity Extraction, Natural Language Processing

Received 00 January 2009; revised 00 Month 2009

1. INTRODUCTION

The amount of user-generated content on the Internet increases significantly every day. Whether they be short news articles or large scale collaborations on Wikipedia, textual documents harbor an abundance of information, explicit or implicit. Relationships between people, places, and events within a single text document are sometimes difficult for a reader to extract—meanwhile the thought of digesting and integrating information from a whole series of lengthy articles is daunting. Consequently, the need for techniques to automatically extract, disambiguate, organize and visualize information from raw text is increasing in importance. In this paper, we are interested in mining documents to identify important events along with the named entities to which they are related. The objective is to create a basis of a system that combines automatically extracted temporal, geospatial, and nominal information into a large database in a form that facilitates further processing and visualization.

This paper describes an interesting, state-of-the-art system combining state-of-the-art techniques. We believe that the article will be instructive for anyone trying to set up an operational system with the complex functionalities described. Instead of focusing on a subproblem, this paper’s focus is on an overall system. The paper describes a well-crafted combination of state-of-the-art techniques (many of them embodied in off-the-shelf toolkits) into an interesting working system. The two main foci of this paper are to accurately and automatically acquire various pieces of event-related information and to visualize the extracted data. Visualization is an important post-process of information extraction as it can offer a level of understanding not inherent in reading the text alone. Visualization is ideal for users who need
quick, summary information from an article. Possible domains for obtaining the information used in our process are numerous and varied, however we have identified two primary sources for rich amounts of reliable information. These domains are online news archives and Wikipedia.

For the research reported in this paper, we work with Wikipedia articles on historical events because of the availability of high quality articles in large numbers. Figure 1 shows a portion of a Wikipedia article that has been marked up for event references and named entities. One of the most influential features of the Internet is the ability to easily collaborate in generating information. Wikis are perfect examples. A wiki offers an ideal environment for the sharing of information while allowing for a form of peer review that is not present elsewhere. Wikipedia is only one of many corpora that can be mined for knowledge that can be stored in machine-readable form.

A system such as the one we report in this paper may have exciting and practical applications. An example would be the assistance in researching World War II. One would be able to select a time period, perhaps 1939-1940. Then one would be able to select an area on a map—perhaps Europe. One could choose a source of information, such as the World War II article on Wikipedia and all of the articles linked on that page. The result may be an interactive map and timeline that plays between the time period of interest, showing only events on the timeline that occurred in the selected region. Meanwhile markers would fade in and out on the map as events begin and end—each one clickable for a user to retrieve more information. The people and organizations involved in a particular battle could be linked to articles about them. The system described in this paper collects the information necessary and although the visualization is not as sophisticated, we describe the basis for such a system here.

It should be noted that although we extract information from Wikipedia articles, all information mining in our system is based on raw text which allows our system to work with historical articles, books and memoirs, and even newspaper articles describing currently unfolding events. Thus, it is possible to analyze text on the same topic from different sources, and find the amount of similarity or dissimilarity or inconsistency among the sources at a gross level. The existing visualization can currently enable a human to make such distinctions, e.g., when the same event may appear at different places on a timeline.

2. RELATED RESEARCH

Our overarching goal is to build a system that extracts, stores and displays important events and named entities, from Wikipedia articles on historical events. It should be noted that our methods do not limit the system to this source and the event and named entity extraction works directly from plain text.

Research on event extraction has often focused on identifying the most important events in a set of news stories [1, 2, 3]. 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 [4] to be used. In the 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 [5]. Another issue that has been addressed is event coreference, determining which descriptions of events in a document refer to the same actual event [6, 7].

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 interval notation and 13 relations [8] and variations on it using points instead, to a constraint structure for the medical domain in [9]. A recent way to represent the relations is a set of XML tags called TimeML [10]. Software has been developed [11, 12] that identifies events and time expressions, and then generates relations among them. Most events are generally represented by verbs in the document, but also include some nouns. The link or relation generation is done differently depending on the system, but uses a combination of rule-based components and classifiers.

We are also interested in named entity recognition (NER), i.e., the extraction of words and strings of text within documents that represent discrete concepts, such as names and locations [13]. Successful approaches include Hidden Markov Models, Conditional Random Fields, Maximum Entropy models, Neural Networks, and Support Vector Machines (SVM). The extraction of named entities continues to invite new methods, tools, and publications.

SVMs [14] have shown significant promise for the task of NER. Starting with the CoNLL 2002 shared task on NER, SVMs have been applied to this problem domain with improving performance. Bhole et al. report extracting named entities from Wikipedia and relating them in time using SVMs [15]. In the biomedical domain, Takeuchi and Collier have demonstrated that SVMs outperform HMMs for NER [16]. Lee et al. [17] and Habib [18, 19, 20] have used SVMs for the multi-class NER problem. Isozaki and Kazawa [21] show that chunking and part-of-speech tagging also provide performance increases for NER with SVMs, both in speed and in accuracy. Dakka and Cucerzan [22] demonstrate high accuracy using SVMs for locational entities in Wikipedia articles. Mansouri et al. demonstrate excellent performance on NER [23] using what are called fuzzy SVMs. Thus, as attested by many published papers, SVMs provide good performance on NER tasks. Hidden Markov Models
The Battle of <LOCATION>Fredericksburg</LOCATION> was <EVENT>fought</EVENT> in December 11-15, 1862, in and around <LOCATION>Fredericksburg</LOCATION>, between General <PERSON>Robert E. Lee</PERSON>'s <ORGANIZATION>Confederate Army of Northern Virginia</ORGANIZATION> and the <ORGANIZATION>Union Army of Potomac</ORGANIZATION>. Maj. General <PERSON>Ambrase E. Burnside</PERSON> commanded the <MISC>Union</MISC> army's futile frontal <EVENT>assault</EVENT> on December 13 against the entrenched <MISC>Confederate</MISC> defenders on the heights behind the city. Figure 1 highlights a Wikipedia article for event references and named entities.
place where aspects of the work being discussed in this paper are pursued. Geographical Information Systems (GIS) is one such area. A GIS is a system designed to capture, store, manipulate, analyze, manage and present geographic information. Traditionally, such information has been used in cartography, resource management and scientific work, and in an increasing manner with mobile computing platforms like cell phones and tablets for mapping and location-aware searches and advertisements. Books by Rigaux et al [39] and Clarke [40] present the main theoretical concepts behind these systems, namely spatial data models, algorithms and indexing methods. As in many fields, with tremendous increase in the amount of geographic information being available in recent years, techniques from statistical analysis, data mining, information retrieval and machine learning are becoming tools in the quest to discover useful patterns and nuggets of geographic knowledge and information. Development and use of geographic ontologies to facilitate geographic information retrieval is an active area of research [41, 42, 43, 44]. Fu et al. [45] describe the use of an ontology for deriving the spatial footprints of a query, focusing on queries that involve spatial relations. Martins [46] describes issues in the development of a location-aware Web search engine. He creates a geographic inference graph of the places in the document and uses the PageRank algorithm [47] and the HITS algorithm [48] to associate the main geographical contexts/scopes for a document. Martins [46] explores the idea of returning documents in response to queries that may be geographically interesting to the individual asking the query. This means queries must be assigned geographic scopes as well and the geographic scope of the query should be matched with the geographic scopes of the documents returned. It also discusses ranking search results according to geographical significance.

3. OUTLINE OF OUR WORK

The work reported in this paper is carried out in several phases. First, we extract all events from a document. Next, we identify significant events in the text. We follow by identifying associated named entities in the text. We process the geospatial entities further for disambiguation and geolocation. Finally, we create a display with maps and timelines to represent the information extracted from the documents. The activities we carry out in the system reported in this paper are listed below.

- Extract events from texts.
- Identify important events from among the events extracted.
- Extract temporal relations among identified important events.
- Extract named entities in the texts including those associated with important events.
- Disambiguate locational named entities and geocode them.
- Create a display with a map and timeline showing relationships among important events.

We discuss each one of these topics in detail in the rest of the paper. Figure 2 shows the overall processing architecture and segments of the system, including the visualization sub-system. The operation of the entire system is based on an initial NER processing of the document, to extract PER (people), LOC (location), and ORG (organization) named entities. Temporal information, document structure, and NE information is used to identify important events. In parallel, the LOC named entities are geocoded and grounded to a single specific location. If the LOC entities are ambiguous, document information is used to ground the LOC entity to the correct geospatial location. Once the LOC and event processing is complete, all of the information is stored in a database. We have also developed a map and timeline front-end that allows users to query documents and visualize the geospatial and temporal information on a map and timeline.

4. EXTRACTING EVENTS FROM TEXTS

Automatically identifying events in a textual document enables a higher level of understanding of the document than only identifying named entities.

The usual approach to identifying events in a text is by providing a set of relations along with associated patterns that produce their natural language realization, possibly in a domain dependent manner. For example, [49, 50, 51] are able to provide answers to queries about individual facts or factoid questions using relation patterns. Others [52, 53] take a deeper approach that converts both the query text and the text of candidates for answers into a logical form and then use an inference engine to choose the answer text from a
set of candidates. Still others [54] use linguistic contexts to identify events to answer factoid questions. For non-factoid questions, where there are usually no unique answers and which may require dealing with several events, pattern matching does not work well [55, 56].

Evita [57] does not use any pre-defined patterns to identify the language of event occurrence. It is also not restricted to any domain. Evita combines linguistic and statistical knowledge for event recognition. It is able to identify grammatical information associated with an event-referring expression such as tense, aspect, polarity and modality. Linguistic knowledge is used to identify local contexts such as verbal phrases and to extract morphological information. The tool has a pre-processing stage and uses finite-state machine based and Bayesian-based techniques.

We run the Evita program on the article, which labels all possible events in the TimeML format. 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 establish temporal relations in later processing steps; these properties include tense and modality. Specifically, Evita places XML “EVENT” tags around single words defining events; these may be verbs or nouns, as seen in Figures 1, 3 and 4. 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.

For event extraction, there were a few other alternatives although we found that for our task EVITA’s results were comparable. EVITA is easily downloadable and usable as well. The other systems include TRIPS-TRIOS [58, 59] and TIPSem [60]. TRIPS-TRIOS parses the raw text using a parser that produces a deep logical (LF) from text and then extracts events by matching semantic patterns using about 100 hand-coded extraction patterns that match against parsed phrases. TIPSem is uses Conditional Random Fields [26, 61].

5. IDENTIFYING IMPORTANT EVENTS

When storing events in our database, we currently associate an event with its containing sentence, saving the entire line. Each event/sentence is classified as important or not using a classifier trained with a set of mostly word-level and sentence-level features.

We experimented with several SVM-based classifiers to identify important events/sentences. We also experimented with choices of features for the classifiers. Several features are concerned 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). On the other hand, some features relate to the sentence as a whole (for example, presence of negation and presence of digits). The features are based on practical intuition. For example, low distance from a named entity, especially from a named entity that is mentioned often, probably indicates importance because a reader cares about events that the named entity is involved in. This is opposed to more generic nouns, or abstract nouns which generally means less importance. For example, consider the sentence from the Wikipedia article on the Battle of Gettysburg: “Lee led his army through the Shenandoah Valley to begin his second invasion of the North.” This sentence describes an important event and Lee is a named entity that occurs often in this article. Intuitively speaking, negation is usually not associated with importance since we care about what happened, not what did not. Grammatical aspect, in particular, progressive, seems less important because it tends to pertain to a state of being, not a clearly defined event. For example, consider another sentence from the Battle of Gettysburg article: “The 26th North Carolina (the largest regiment in the army with 839 men) lost heavily, leaving the first day’s fight with around 212 men.” This has “leaving” with progressive aspect and this sentence does not really depict an event, more a state. Longer words are sometimes more descriptive and thus more important while nouns and verbs are usually more important than adjectives. Digits are also a key factor of importance as they are often part of a date or statistic. Similar features have been used by other researchers when searching for events or important events or similar tasks [5, 62, 58, 59].

We also experimented with features that relate to the article as a whole such as the position of a sentence in the document although we did not find these as useful. We use similarity of the event word to article “keywords.” These keywords are 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, words such as “war,” “conflict,” and “fought” are often used to specify salient topics. 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 from [63]. As explained in the documentation for this Perl module,2

---

1http://timeml.org/site/tarsqi/toolkit/download.html

2http://search.cpan.org/dist/WordNet-Similarity/doc/intro.pod
The Battle of Fredericksburg was fought December 11-15, 1862, in and around Fredericksburg, Virginia, between Robert E. Lee’s Confederate Army of Northern Virginia and the Union Army of Potomac, commanded by Maj. General Ambrose E. Burnside. The Union army’s futile frontal assault on December 13 against the entrenched Confederate defenders on the heights behind the city is remembered as one of the most one-sided battles of the American Civil War, with Union casualties more than twice as heavy as those suffered by the Confederates.

FIGURE 3. Portion of an article with events tagged by EVITA (words followed by e_i). We show only the event tags here.

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

FIGURE 4. A portion of the XML representation of Figure 3.

it 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. Given two word senses, this module looks at their WordNet glosses and computes the amount of similarity. It implements several semantic relatedness measures including the ones described by Leacock and Chodorow [64], Jiang and Conrath [65], Resnik [66], [67], Hirst and St-Onge [68], Wu and Palmer [69], and the extended gloss overlap measure by Banerjee and Pedersen [70].

After initial experimentation, we changed some word-level features to sentence-level features by adding, averaging, or taking the maximum of the word-level features for each event in the sentence. A feature added was based on TextRank [71], an algorithm developed by Mihalcea and Tarau that ranks sentences based on importance and is used in text summarization. TextRank works using the PageRank algorithm developed by Page et al. [47]. 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 to rank 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 gives a weight of essentially 0 if either sentence is shorter than a threshold of 10 words which we chose based on average sentence length in our corpora. For all others, it calculates the distance between the sentences using a simple approach. Each sentence is treated as a bag of words. The words are stemmed. Then, we simply compute how many words are the same between the two sentences. The similarity is then chosen as the sum of the sentence lengths divided by their distance. Named entities are also considered more important to the process than before. Instead of just asking if the sentence contains one, 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 in the sentence (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.

The final set of features used can be seen in Table 1.

<table>
<thead>
<tr>
<th>Important Event Classifier Features</th>
</tr>
</thead>
<tbody>
<tr>
<td>Presence of an event in the perfective aspect</td>
</tr>
<tr>
<td>Percent of events in the sentence with class “occurrence”</td>
</tr>
<tr>
<td>Digit presence</td>
</tr>
<tr>
<td>Maximum length of any event word in the sentence</td>
</tr>
<tr>
<td>Sum of the Named Entity ‘weights’ in the sentence (NE weight being the number of times this NE was mentioned in the article divided by the number of all NE mentions in the article)</td>
</tr>
<tr>
<td>Negation presence in the sentence</td>
</tr>
<tr>
<td>Number of events in the sentence</td>
</tr>
<tr>
<td>Percent of events in the sentence that are verbs</td>
</tr>
<tr>
<td>Position of the sentence in the article normalized by the number of sentences in the article</td>
</tr>
<tr>
<td>Maximum similarity of any event word in the sentence</td>
</tr>
<tr>
<td>Percent of events in the sentence that are in some past tense</td>
</tr>
<tr>
<td>TextRank rank of the sentence in the article, divided by the number of sentences in the article</td>
</tr>
<tr>
<td>Number of “to be” verbs in the sentence</td>
</tr>
</tbody>
</table>

TABLE 1. A final list of features for classifying events as important or not

6. IDENTIFYING TEMPORAL RELATIONS

We experimented with three different ways to identify temporal relations among events in a document. The first approach used the TARSQI toolkit to identify temporal relations among events. The second approach was the development of a large number of regular expressions. We also downloaded Heideltime [72] and integrated it with our system. Heideltime is also a regular expression based system, written in a
6.1. Using TARSQI to Extract Occurrence Times

We use the TARSQI Toolkit (TTK) to create event-event and event-time links (called TLINKs) in addition to identifying events in a document. These include attributes like type of expression (DATE, TIME, etc.) and value. Some events are anchored to time expressions, and some to other events. These are represented by the TLINK XML tag, which has attributes including the type of relation. An example of each kind of TLINK is shown in Figure 5. The first represents 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.

6.2. Using Regular Expressions to Extract Occurrence Times

Our second approach applied regular expressions to extract times. The results of using extensive regular expressions versus using TTK showed that the regular expressions pull out many more complete dates and times. For example, in the text given in Figure 1 and Figure 3, TTK 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 on regular expressions rather than TTK. The regular expressions in our program are built from smaller ones representing months, years, seasons, and so on. We use 24 total combinations of smaller expressions, some of which recursively tested groups they matched for sub-expressions. One example of an expression is:

(early | late | mid | mid | middle | the end of | the middle of | the beginning of | the start of )?(Winter | Spring | Summer | Fall | Autumn)
(of )? ( ((([0-9])|((1-9)))
([0-9] [0-9]) | (?[0-9] [0-9])
which finds expressions like “[By] early fall 1862 [...]” a type of phrase that occurs in the corpus. A flaw present in our program and not in TTK 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.

To anchor time expressions to real times—specifically to a year—we use 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 the difference in the month if available. For example, a month that is more than five months after the month of the anchoring time is statistically 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 possible.

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 empirical evidence; 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. For example, the first sentence of the passage in Figure 6 sets a start point, June 9, for the subsequent events. The year for this expression would be calculated using a previously found time expression, possibly the first one in the article. There are various sentences in the passage following this one that do not contain time expressions, some of which may be considered important enough to place on the timeline—for example, “Alfred Pleasonton’s combined arms force of two cavalry divisions (8,000 troopers) and 3,000 infantry, but Stuart eventually repulsed the Union attack.” In context, a human can tell that this also takes place on June 9; the system places it (correctly) between June 9 (the previously found expression) and June 15, the next one that appears.

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.

7. EXTRACTING NAMED GEOSPATIAL AND OTHER NAMED ENTITIES

The next goal is to extract all geospatial named entities from the texts of documents. In particular, we are interested in extracting NEs that coincide with the important events or sentences identified in the documents. We experimented with three approaches to extracting geospatial named entities from the texts of documents. The first approach uses an SVM which is trained to extract geospatial named entities. The second approach uses an HMM-based named entity recognizer. The third approach uses a CRF-based (Conditional Random Field) named entity recognizer. A by-product of the second and third approaches is that we obtain non-geospatial named entities as well.

The reason for experimenting with the three methods is that we wanted to evaluate if training an SVM on just geospatial entities would work better than off-the-shelf HMM or CRF tools. Precision, recall and F-measure are standard metrics used to evaluate the performance of a classifier, especially in natural language processing and information retrieval. After a trained classifier has been used to classify a set of objects, the fraction of objects correctly classified is termed precision. Recall computes the fraction of objects that belong to a specific class in the test dataset and are correctly classified by the classifier. When there are several classes under consideration, precision and recall can be obtained for each class and averaged. Precision and recall are often inversely proportional to each other and there is normally a trade-off between these two ratios. The F-measure mixes the properties of the previous two measures as the harmonic mean of precision and recall [73, 74]. If we want to use only one accuracy metric as an evaluation criterion, F-measure is the most preferable.

7.1. SVM-based Extraction of Geospatial Entities

This effort is discussed in details in [75, 76, 77]. A Support Vector Machine (SVM) [78, 79, 14] is an example of a type of supervised learning algorithm called classifiers. A classifier finds the boundary function(s) between instances of two or more classes. A binary SVM works with two classes, but a multi-class SVM can learn the functions for the discriminant or separator functions for several classes at the same time. A binary SVM attempts to discover a wide-margin separator between two classes. The SVM can learn this separator, but to do so, it must be presented with examples of known instances from the two classes. Shown a number of such known instances, the SVM formulates the problem of classification as solving a complex optimization problem. To train an SVM, each instance must be described in terms of a number of attributes or features. A multi-class SVM is a generalization of the binary SVM.

The desired output of this stage of processing is a set of words and strings that may be place names. To define the features describing the geospatial NEs, we draw primarily from the work done for word-based feature selection by [18, 19, 20]. They used lexical, word-shape, and other language-independent features to train an SVM for the extraction of NEs. Furthermore, the work
The first major action of the campaign took place on June 9 between cavalry forces at Brandy Station, near Culpeper, Virginia. The 9,500 Confederate cavalrymen under Stuart were surprised by Maj. Gen. Alfred Pleasonton’s combined arms force of two cavalry divisions (8,000 troopers) and 3,000 infantry, but Stuart eventually repulsed the Union attack. The inconclusive battle, the largest predominantly cavalry engagement of the war, proved for the first time that the Union horse soldier was equal to his Southern counterpart.

By mid-June, the Army of Northern Virginia was poised to cross the Potomac River and enter Maryland. After defeating the Federal garrisons at Winchester and Martinsburg, Ewell’s Second Corps began crossing the river on June 15.

FIGURE 6. Flow of time in a few sentences from the Battle of Gettysburg article

done by Cucerzan et al. [80] on extracting NEs from Wikipedia provided further features. For feature set selection, we postulate that each word in the geospatial NE should be independent, and that the training corpus should be split into individual words. A word can be labeled as beginning, middle or end of a named entity for multi-word named entities. After testing various combinations of features, we used the following features for each word to generate the feature vectors for the SVM, with equal weighting on all features:

- Word length: alphanumeric character count, ignoring punctuation marks
- Vowel count, consonant count, and capital count in the word
- First capital: if the word begins with a capital letter
- If the word contains two identical consonants together
- If the word ends with a comma, a period, a hyphen, a semi-colon, or a colon
- If the word begins or ends with a quote
- If the word begins or ends with a parenthesis
- If the word is plural, or a single character
- If the word is alphanumeric, purely numeric, purely text, all uppercase, or all lowercase
- If the word has a suffix or prefix
- If the word is a preposition, an article, a pronoun, a conjunction, an interrogative or an auxiliary verb.

7.2. HMM-based Approach Using LingPipe

Hidden Markov Models (HMMs) have shown excellent results in the task of named entity recognition. An HMM [81] is also a type of classifier that can learn to discriminate between new and unknown instances of two (or more classes), once it has been trained with known and labeled examples of the classes. Unlike an SVM, an HMM is a sequence classifier, in the sense that an HMM classifies individual objects into classes but in doing so, it takes into account characteristics of objects or entities that occur before and after the current entity that is being classified. An HMM is a graphical model of learning that visualizes a sequence as a set of nodes, with transitions among them. Transitions have associated probabilities. Learning involves finding a sequence of “hidden” states that best explains a certain emitted sequence.

The idea of sequence classification is useful in many natural language processing tasks because the classification of an instance (e.g., a word as describing an event or not) may depend on the nature of words that occur before or after it in a sentence. An SVM can also take into account what comes before and after the object being classified, but the idea of sequence has to be handled in an indirect manner. One must note that sequencing does not always make sense, but in some situations, it is natural and important. Klein et al. [24] and Zhou and Su [25] have demonstrated excellent performance in extracting named entities using HMMs. We chose to use the HMM implemented by the LingPipe library for NER.

7.3. CRF-based Approach

We also used the named entity extraction tool designed by the Stanford Natural Language Processing Group. This package uses a conditional random field (CRF) for named entity recognition (NER) which achieved very high performance for all named entities when tested on the CoNLL 2003 test data [82]. The Stanford Web site states that the CRF was trained on “data from CoNLL, MUC6, MUC7, and ACE”.

These datasets originate from news articles from the U.S. and U.K., making it a strong model for most English domains.

CRFs [26, 61] provide a supervised learning framework to build statistical models using sequence data. A CRF is an undirected graphical model that defines a single log-linear distribution over label sequences given a particular observation sequence. HMMs make certain strong independence assumptions among neighboring samples, which can be relaxed by CRFs. CRFs allow encoding of known relationships among observations in terms of likelihoods and construct consistent interpretations. The linear chain CRF is popular in natural language processing for predicting as sequence of labels for a sequence of input

http://alias-i.com/lingpipe/demos/tutorial/ne/read-me.html

http://nlp.stanford.edu/

$^5$Stanford CRF - README.txt
8. GROUNDING GEOSPATIAL NAMED ENTITIES

Each of the three learning based methods extract a set of candidate geospatial NEs from the article text. For each candidate string, the second objective was to decide whether it was a geospatial NE, and to determine the correct (latitude, longitude), or (φ, λ) coordinate pair for the place name in context of the article, which is often referred to as grounding the named entity.

To resolve the candidate NE, a lookup is made using Google Geocoder[67]. If the entity reference resolves to a single geospatial location, no further action is required. Otherwise, the context of the place name in the article, a novel data structure and a rule-driven algorithm are used to decide the correct spatial location for the place name.

Our research targeted the so-called context locations, defined as “the geographic location that the content of a web resource describes” [83]. Our task is close to that of word sense disambiguation, as defined by Cucerzan [80]; only that we consider the geospatial context and domain instead of the lexical context and domain. Sehgal et al. demonstrate good results for geospatial entity resolution using both spatial (coordinate) and non-spatial (lexical) features of the geospatial entities [84]. Zong et al. demonstrate a rule-based method for place name assignment, achieving a precision of 88.6% on disambiguating place names in the United States, from the Digital Library for Earth System Education (DLESE) metadata [85].

While related to the work done by Martins et al. in [86], we apply a learning based method for the initial NER task, and only use the geocoder/gazetter for lookups after the initial candidate geospatial NEs have been identified. We also draw on our earlier work in this area, covered in [77].

The primary challenge to the resolution and disambiguation is that multiple names can refer to the same place, and multiple places can have the same name. The statistics for these place name and reference overlap are given in Table 2, from [87]. The table demonstrates that the largest area with ambiguous place names is North and Central America, the prime areas for our research. Most of the name overlaps are in city names, and not in state/province names. These statistics are further supported by the location name breakdowns on a few exemptual individual Wikipedia articles in Table 3. The details in this table are broken down into countries, states, foreign cities, foreign features, US cities, and US features, where “features” are any non-state or city geospatial features, such as rivers, hills, and ridges. The performance of the NER extraction and grounding system is indicated. The Battle of Gettysburg, with over 50% US city names, many of which overlap, resulted in the lowest performance. The Liberty Incident in the Mediterranean Sea, with less than 10% US city and state names combined, showed significantly better performance.

8.1. Google Geocoder

We used Google Geocoder as the gazetteer and geocoder for simplicity, as much research has already been done in this area. Vestavik and D’Roza et al. provide an excellent overview of existing technology in this area [88, 89]. Google Geocoder provides a simple REST-based interface that can be queried over HTTP, returning data in a variety of formats.

For each geospatial NE string submitted as a query, Google Geocoder returns 0 or more placemarks as a result. Each placemark corresponds to a single (φ, λ) coordinate pair. If the address string is unknown, or another error occurs, the geocoder returns no placemarks. If the query resolves to a single location, Google Geocoder returns a single placemark. If the query string is ambiguous, it may return more than one placemarks. A country code can also be passed as part of the request parameters to bias the geocoder toward a specific country. Google Geocoder returns locations at roughly four different levels, Country, State, City, and Street/Feature, depending on the address query string and the area of the world.

In our system, the address string returned from Google Geocoder is separated into the four parts, and each part is stored separately, with the coordinates. The coordinates are also checked against the existing set of locations in the database for uniqueness. For our system, coordinates had to be more than 1/10th of a mile apart to be considered a unique location.

8.2. Location Tree Data Structure

The output from the Google Geocoder is fed through our location tree data structure, which, along with an algorithm driven by a simple set of rules, orders the locations by geospatial division, and aids in the disambiguation of any ambiguous place name.

The location tree operates at four levels of hierarchy, Country, State/Province, City, and Street/Local Geographic Feature. Each placemark response from the geocoder corresponds to one of these geospatial divisions. A separate set of rules governs the insertion of a location at each level of the tree. For simple additions to the tree, the rules are given below.

1. If the location is a country, add it to the tree.
2. If the location is a state or province, add it to the tree.

---


Google Geocoder was chosen because of accessibility and accuracy of the data, as this research does not focus on the creation of a geocoder.
For Review Only

EXTRACTING AND DISPLAYING TEMPORAL AND GEOSPATIAL ENTITIES FROM ARTICLES ON HISTORICAL EVENTS

TABLE 2. Places With Multiple Names and Names Applied to More Than One Place in the Getty Thesaurus of Geographic Names

<table>
<thead>
<tr>
<th>Continent</th>
<th>% places with multiple names</th>
<th>% names with multiple places</th>
</tr>
</thead>
<tbody>
<tr>
<td>North &amp; Central America</td>
<td>11.5</td>
<td>57.1</td>
</tr>
<tr>
<td>Oceania</td>
<td>6.9</td>
<td>29.2</td>
</tr>
<tr>
<td>South America</td>
<td>11.6</td>
<td>25.0</td>
</tr>
<tr>
<td>Asia</td>
<td>32.7</td>
<td>20.3</td>
</tr>
<tr>
<td>Africa</td>
<td>27.0</td>
<td>18.2</td>
</tr>
<tr>
<td>Europe</td>
<td>18.2</td>
<td>16.6</td>
</tr>
</tbody>
</table>

TABLE 3. Place Name Details for Example Wikipedia Articles. All numbers are percentages.

<table>
<thead>
<tr>
<th>Article name</th>
<th>Foreign features</th>
<th>US Features</th>
<th>Foreign Cities</th>
<th>US Cities</th>
<th>States</th>
<th>Countries</th>
<th>F-measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battle of Gettysburg</td>
<td>0</td>
<td>31</td>
<td>1</td>
<td>5</td>
<td>51</td>
<td>15</td>
<td>2</td>
</tr>
<tr>
<td>War of 1812</td>
<td>3</td>
<td>18</td>
<td>5</td>
<td>30</td>
<td>8</td>
<td>36</td>
<td>81.6</td>
</tr>
<tr>
<td>World War II</td>
<td>20</td>
<td>0</td>
<td>17</td>
<td>1</td>
<td>0</td>
<td>62</td>
<td>87.0</td>
</tr>
<tr>
<td>Liberty Incident</td>
<td>25</td>
<td>1</td>
<td>10</td>
<td>6</td>
<td>1</td>
<td>57</td>
<td>90.5</td>
</tr>
</tbody>
</table>

FIGURE 7. Example Location Tree Showing Duplicated Node Count

3. If the location is a city in the US, and the parent node in the tree is a state, add it to the tree.
4. If the location is a city outside the US, add it to the tree.
5. If the location is a street or feature inside the US, and the parent node is a state or city, add it to the tree.
6. If the location is a street or outside the US, and the parent node is either a country or city, add it to the tree.

Rules 3 and 5 are US-specific because Google Geocoder has much more detailed place information for the US than other areas of the world, requiring extra processing.

Any location that does not match any rule is placed on a pending list. Each time a new location is added, the tree is re-sorted to ensure that the correct hierarchy is maintained, and each location on the pending list is re-checked to see if it now matched the rules for insertion.

If the location returned from Google Geocoder has a single placemark and matches the rules, it is placed in the tree. If the location returned corresponds to multiple coordinates, another set of calculations is required before running insertion rule set. First, any ambiguous locations are placed on the pending list until all other locations are processed. This ensures that we have the most complete context possible, and that all non-ambiguous locations are in the tree before the ambiguous locations are processed.

Once we finish processing all the non-ambiguous locations in the candidate list, we have a location tree, and a set of pending locations, some of which could fit more than one place on the tree. As each node is inserted into the tree, a running count of nodes and repeated nodes are kept. Using this information, when each ambiguous location is considered for insertion, a weighting is placed on each possible node in the tree that could serve as the parent node for the location. As an example, we consider the location “Cambridge”. This could either be Cambridge, MA, USA, or Cambridge, UK. The tree shown in Figure 7, demonstrates a pair of possible parent leaf nodes to “Cambridge”, with the occurrence count for each node in the article text. The location “Cambridge” can be added underneath either leaf node in the tree. The weight calculation shown in Equation 1 determines the correct node. The weight for node \( n \) is determined by summing the insertion count for all the parent nodes of \( n \) in the tree (up to three parents), and dividing by the total insertion count for the tree. A parent node is set to zero if the tree does not contain that node. The insertion count for each node is the total number of occurrences of the location represented by that node in the original article, so the insertion count for the whole tree is equal to the number of all location occurrences in the article.

\[
Weight_n = \frac{(Ct_{\text{country}} + Ct_{\text{state}} + Ct_{\text{city}})}{Ct_{\text{total}}} \tag{1}
\]

Calculating weight for the two possible parent nodes of “Cambridge”, we use 3 as the insertion count for UK, and 4 for the insertion count for Massachusetts, 2 from the Massachusetts node and 2 from the USA node.
9. EXPERIMENTS, RESULTS AND ANALYSIS

For this research, we downloaded the English language pages and links database from the June 18, 2008 dump of Wikipedia. This download provides the full text of all the pages in the English Wikipedia at the time of creation. This full text includes all wiki and HTML markup. The download contains approximately 7.2 million pages. A selection of 90 pages about battles and wars were obtained and hand-tagged. The pages were obtained by automatically crawling a Wikipedia index of battles and dismissing extremely short articles. These pages resulted in approximately 230,000 words of data, an average of 7% of which are geospatial named entities. The pages also provided a total of 5000 sentences. Table 4 shows a list of selection of articles we work with.

It is necessary to pre-process the text of the articles before it can be used for information extraction. We remove all infoboxes. For Wikipedia-style links, the link text is removed, and replaced with either the name of linked page or with the caption text for links with a caption. Links to external sites are removed. Certain special characters and symbols that do not add any information are removed as they tend to cause exceptions and problems in string processing, e.g., HTML tags. The characters we remove do not include those that would appear in names such as accented characters, commas, periods, etc. The cleaned primary text of each page is split into an array of individual words, and the individual words are hand-tagged for various purposes as necessary.

### 9.1. Extracting Important Events

We use supervised learning for extracting important events from the historical documents. For this purpose, we require a set of hand-tagged articles indicating whether each event word identified by EVITA is part of an important event. If an event word is part of an important event, the sentence containing it is considered important because the word contributes to this importance. Because there is no good objective measure of importance, we asked multiple volunteers to tag the articles to avoid bias. The subjectivity was a real problem, however. The most basic guideline

<table>
<thead>
<tr>
<th>Page Title</th>
<th>Word Ct.</th>
<th>Sentence Ct.</th>
<th>NE Ct.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Battle of Antietam</td>
<td>6167</td>
<td>400</td>
<td>260</td>
</tr>
<tr>
<td>Battle of Britain</td>
<td>10611</td>
<td>512</td>
<td>229</td>
</tr>
<tr>
<td>Battle of the Bulge</td>
<td>7614</td>
<td>339</td>
<td>265</td>
</tr>
<tr>
<td>Battle of Chancellorsville</td>
<td>2979</td>
<td>137</td>
<td>133</td>
</tr>
<tr>
<td>Battle of Chantilly</td>
<td>1170</td>
<td>50</td>
<td>56</td>
</tr>
<tr>
<td>Battle of Chickamauna</td>
<td>2000</td>
<td>99</td>
<td>108</td>
</tr>
<tr>
<td>American Civil War</td>
<td>7961</td>
<td>397</td>
<td>298</td>
</tr>
<tr>
<td>First Barbary War</td>
<td>1906</td>
<td>88</td>
<td>97</td>
</tr>
<tr>
<td>Battle of Fredericksburg</td>
<td>2354</td>
<td>111</td>
<td>77</td>
</tr>
<tr>
<td>Battle of Gettysburg</td>
<td>5289</td>
<td>256</td>
<td>274</td>
</tr>
<tr>
<td>First Gulf War</td>
<td>11562</td>
<td>895</td>
<td>674</td>
</tr>
<tr>
<td>Korean War</td>
<td>10570</td>
<td>462</td>
<td>495</td>
</tr>
<tr>
<td>Mexican-American War</td>
<td>5822</td>
<td>275</td>
<td>472</td>
</tr>
<tr>
<td>War</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Operation Eagle</td>
<td>1869</td>
<td>60</td>
<td>56</td>
</tr>
<tr>
<td>Attack on Pearl Harbor</td>
<td>3414</td>
<td>188</td>
<td>198</td>
</tr>
<tr>
<td>Battle of Shiloh</td>
<td>4899</td>
<td>253</td>
<td>157</td>
</tr>
<tr>
<td>Spanish-American War</td>
<td>4229</td>
<td>194</td>
<td>296</td>
</tr>
<tr>
<td>Battle of Vicksburg</td>
<td>3369</td>
<td>178</td>
<td>119</td>
</tr>
<tr>
<td>The Whiskey Rebellion</td>
<td>1475</td>
<td>65</td>
<td>49</td>
</tr>
<tr>
<td>World War II</td>
<td>6775</td>
<td>331</td>
<td>661</td>
</tr>
</tbody>
</table>
was to judge whether one 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 usually be resolved using a majority vote. Thirty articles were 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.

If we took one annotator’s tagging as ground truth and measure every other annotator’s accuracy against that, the average inter-annotator F1-score came out as 42.6%, indicating great disagreement. Using an alternative approach, we labeled a sentence as 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 average F1-score between annotators for this new method rose to 66.8% (lowest being 47.7% and highest being 78.5%).

By using data gathered by multiple annotators and filtering it in such a way, we are able to concentrate on events that multiple people can agree are important. This is our attempt to both avoid overly focusing and overly diluting our training data. Adding more participants can impact our training data in a variety of ways that depends on the like-mindedness of the individuals; the addition could increase our confidence in a particular event being important, add to or drop events from acceptable confidence levels, etc. Depending on our annotators, training data could represent anything from a very focused method of importance identification to a noisy over-saturated list of events. Overall the human training aspect is necessary and has the potential to be improved, likely by adding a short phase of mass-tagging potentially followed by periodic additions to the training corpus. Although we currently use one generic set of training data which we apply to text from any source, another option is to develop various models trained on different sets of text to achieve better performance across sources, i.e., utilizing metadata along with the training text to allow classification of the text based on time period, structure, source, etc. We can generate multiple models trained on different subsets of our tagged texts, grouped by their characteristics and when we apply the system to a new source of text, we can use metadata, if it is available for the text, to choose the most relevant model.

Ultimately, we trained a binary SVM using a radial basis function kernel using the LIBSVM software [90]. The parameters for the kernel and cost constant were found with one of LIBSVM’s built-in tools. SVMs were trained and tested using 10-fold 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 \(c\), \(g\), 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 \(g\) were reduced 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 5. 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

<table>
<thead>
<tr>
<th>(c)</th>
<th>(g)</th>
<th>(w)</th>
<th>F-score</th>
</tr>
</thead>
<tbody>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>1</td>
<td>29.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>2</td>
<td>46.1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>2</td>
<td>46.9</td>
</tr>
<tr>
<td>1.0</td>
<td>0.25</td>
<td>2</td>
<td>47.5</td>
</tr>
<tr>
<td>1.0</td>
<td>0.125</td>
<td>2</td>
<td>48.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.0625</td>
<td>2</td>
<td>49.0</td>
</tr>
<tr>
<td>1.0</td>
<td>0.03125</td>
<td>2</td>
<td>48.0</td>
</tr>
<tr>
<td>1.0</td>
<td>1.0</td>
<td>3</td>
<td>46.2</td>
</tr>
<tr>
<td>1.0</td>
<td>0.5</td>
<td>3</td>
<td>48.9</td>
</tr>
<tr>
<td>1.0</td>
<td>0.25</td>
<td>3</td>
<td>50.1</td>
</tr>
<tr>
<td>1.0</td>
<td>0.125</td>
<td>3</td>
<td>51.0</td>
</tr>
<tr>
<td>(p:41.06066, r:75.0522, f:51.0428)</td>
<td>1.0</td>
<td>0.0625</td>
<td>3</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>2</td>
<td>45.2</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>2</td>
<td>47.1</td>
</tr>
<tr>
<td>0.5</td>
<td>0.25</td>
<td>2</td>
<td>47.5</td>
</tr>
<tr>
<td>0.5</td>
<td>0.125</td>
<td>2</td>
<td>48.2</td>
</tr>
<tr>
<td>0.5</td>
<td>0.0625</td>
<td>2</td>
<td>48.1</td>
</tr>
<tr>
<td>0.5</td>
<td>1.0</td>
<td>3</td>
<td>48.0</td>
</tr>
<tr>
<td>0.5</td>
<td>0.5</td>
<td>3</td>
<td>49.2</td>
</tr>
<tr>
<td>0.5</td>
<td>0.25</td>
<td>3</td>
<td>50.55</td>
</tr>
<tr>
<td>0.5</td>
<td>0.125</td>
<td>3</td>
<td>51.0</td>
</tr>
<tr>
<td>(p:41.14866, r:74.7909, f:51.03596)</td>
<td>0.5</td>
<td>0.0625</td>
<td>3</td>
</tr>
<tr>
<td>(p:41.353864, r:73.9689, f:51.01095)</td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
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.

9.2. Experiments in Extracting Occurrence Times

Testing the times is significantly more difficult, since the intervals generated by this program are certainly different than the ones intended by the article and representing events with times is difficult even for human annotators. We use two evaluation methods for extracted times.

First, spot-checking showed strengths of our regular-expression based time extraction method when time expressions are explicit. Two undergraduates examined a random pick of several dozen times generated automatically. We found that approximately 80% of the automatic extractions were correct if there is an explicit time expression.

When a time of occurrence is not given explicitly, we use a measure proposed by Ling and Weld in 2010 [12] 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. The range of temporal entropies obtained is shown in Figure 9. It is also 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. The TE from these results is primarily between 5 and 15, with 1400 sentences that had temporal expressions having TE in this range, and the majority of this close to 10. A TE obtained is shown in Figure 9. It is also 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. The TE from these results is primarily between 5 and 15, with 1400 sentences that had temporal expressions having TE in this range, and the majority of this close to 10. A TE.

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. An example of this error can be seen in the USS Liberty article. This article’s span was found to be the one day June 8, 1967. Regular expressions on the sentence “She was acquired by the United States Navy ... and began her first deployment in 1965, to waters off the west coast of Africa.” find ‘began’ and ‘1965’ and it is concluded that this event starts in 1965 and ends at the end of the whole event, in 1967. This is incorrect. Worse, a sentence closely following this one begins “On June 5, at the start of the war,” and is guessed to be in the 1965 because the correct year, 1967, was not mentioned again between the sentences. This error then propagates through at least one more sentence. Similar examples succeed, however, as in the Battle of Gettysburg article: “The two armies began to collide at Gettysburg on July 1, 1863, as Lee urgently concentrated his forces there” is given date July 1, 1863, and the next sentence with a time expression “On the third day of battle, July 3, fighting resumed on Culps Hill...” uses 1863 as a year. A useful but impractical additional test, which we have not performed, would be human examination 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 integrate this test’s results into the former results.

9.3. Experiments in Geospatial Named Entity Extraction

We performed extensive experiments in extracting geospatial named entities. We used three different methods for extracting and geocoding geospatial named entities. Our goal was to see if an especially trained and tuned SVM will perform better than HMM or CRF approaches, particularly for geospatial names. More details can be found in [76, 75, 77, 91].

9.3.1. Using Machine Learning for Geospatial Named Entity Extraction

We downloaded a number of previously tagged data sets to provide the training material for the SVM. We also hand-tagged a data set on our own. Here are the relevant data sets.

- Spell Checker-Oriented Word List (SCOWL)\(^{10}\) to provide a list of English words.
- CoNLL 2003 shared task dataset on multi-language NE tagging\(^{11}\) containing tagged named entities for PER, LOC, and ORG, in English.
- Reuters Corpus from NIST:\(^{12}\) 15,000 selected words from the corpus were hand-tagged for LOC entities.
- CoNLL 2004 and 2005 shared task datasets, on Semantic Role Labeling\(^{13}\) tagged for English LOC NEs.
- Geonames database of 6.2 million names of cities, counties, countries, land features, rivers and lakes.\(^{14}\)

\(^{10}\)http://downloads.sourceforge.net/wordlist/scowl-6.tar.gz
\(^{11}\)http://www.lsi.upc.es/conll2003/seer/
\(^{12}\)http://trec.nist.gov/data/reuters/Reuters.html
\(^{13}\)http://www.lsi.upc.es/~aricconll/st04/st04.html
\(^{14}\)http://www.geonames.org/export/
To generate the SVM training corpus, we combined our hand-tagged Reuters corpus and the three CoNLL datasets. We extracted those records in the 2.5 million word Geonames database in either the P or A Geonames codes (Administrative division and Population center), which provided the names of cities, villages, countries, states, and regions. Combined with the SCOWL wordlist, which provided negative examples, this resulted in a training set of 1.2 million unique words.

We used LibSVM to train an SVM based on the feature vectors and two parameters, $C$ and $\gamma$, which are the cost and degree coefficient of the SVM kernel function. After using a grid search, the optimal parameters were determined to be $C = 8$ and $\gamma = 0.03125$. We selected a radial basis function kernel, and applied it to the C-SVC, the standard cost-based SVM classifier. The SVM was trained using feature vectors based on the training corpus, requiring 36 hours to complete. Our goal was to obtain high recall while maintaining good precision. Table 6 shows the precision, recall, and f-measure numbers for training corpuses created with different subsets of the training data. The performance numbers are the results from the Wikipedia testing corpus.

After selecting the optimal training corpus, we optimized the feature set for the SVM. Table 7 shows a breakdown of the feature sets we test separately. We found that the list presented earlier provided the best performance by close to 10% in f-measure over any other combination of features. With this setup, we were successful in training the SVM to return high recall numbers, recognizing 99.8% of the geospatial named entities. Unfortunately, this was at the cost of precision. The SVM returned noisy results with only

### TABLE 6. Training Corpus Combination Results for the NER SVM

<table>
<thead>
<tr>
<th>Corpus</th>
<th>Precision</th>
<th>Recall</th>
<th>F</th>
</tr>
</thead>
<tbody>
<tr>
<td>R</td>
<td>0.38</td>
<td>0.63</td>
<td>0.47</td>
</tr>
<tr>
<td>R + C03 + C04 + C05</td>
<td>0.45</td>
<td>0.68</td>
<td>0.54</td>
</tr>
<tr>
<td>R + C03 + C04 + C05 + SCW</td>
<td>0.49</td>
<td>0.75</td>
<td>0.59</td>
</tr>
<tr>
<td>Wiki</td>
<td>0.65</td>
<td>0.34</td>
<td>0.44</td>
</tr>
<tr>
<td>Wiki + R + C03 + C04 + C05 + SCW</td>
<td>0.60</td>
<td>0.55</td>
<td>0.57</td>
</tr>
<tr>
<td>Geo + R + C03 + C04 + C05 + SCW</td>
<td>TC</td>
<td>TC</td>
<td>TC</td>
</tr>
<tr>
<td>FGeo + R + C03 + C04 + C05 + SCW</td>
<td>0.51</td>
<td>0.99</td>
<td>0.67</td>
</tr>
</tbody>
</table>

### TABLE 7. Feature Set Combination Results for the NER SVM

<table>
<thead>
<tr>
<th>Feature Set</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>SHP+PUN+HYP+SEP</td>
<td>0.44</td>
</tr>
<tr>
<td>SHP+CAP+PUN+HYP+SEP</td>
<td>0.47</td>
</tr>
<tr>
<td>CAP+VOW+CON</td>
<td>0.51</td>
</tr>
<tr>
<td>SHP+CAP</td>
<td>0.54</td>
</tr>
<tr>
<td>SHP+CAP+VOW+CON</td>
<td>0.56</td>
</tr>
<tr>
<td>SHP+CAP+VOW+CON+AFF+POS</td>
<td>0.58</td>
</tr>
<tr>
<td>SHP+CAP+VOW+CON+PUN +HYP+SEP+AFF+POS</td>
<td>0.67</td>
</tr>
</tbody>
</table>

**FIGURE 9.** Temporal Entropy graph - the temporal entropies were sorted in increasing order for plotting. Temporal entropy is in log(seconds/days)
51.0% precision, but the noise was processed out during the NE geocoding process, to obtain our final results.

As described earlier, we also performed geospatial NER using an HMM-based tool from LingPipe and a CRF-based tool from Stanford Natural Language Processing Group.

### 9.3.2. Comparison of Geospatial NER using Machine Learning Methods

Table 8 shows the raw results from the geospatial NE recognition using the three learning methods along with the results after processing through geospatial resolution and disambiguation (discussed in the following section). The NER Results show the accuracy of the NERs before any further processing. The Resolved Results in Table 8 specifically show the performance in correctly identifying location strings and geocoding the locations. A string correctly identified by the NER process is one that exactly matched a hand tagged ground truth named entity.

Figure 10 shows a more detailed breakdown of the precision, recall, and f-measure for a subset of the articles processed. A handful of articles with foreign names, such as the article on the Korean War, brought down the average results for all three, but to a larger extent for the SVM and a little for the HMM, compared to the other two methods. This is most likely due to the fact that our training data contained a limited number of foreign names, and the HMM had trouble recognizing these strings as LOC named entities.

The SVM produces a lot more traffic to the geocoder compared to the HMM or CRF method. Table 9 shows the decrease in the number of candidate location NEs extracted by the HMM over the SVM for some of the articles in the test corpus. It also shows the number of these NEs that successfully geocoded and were disambiguated. Although the HMM or the CRF often extracted less than half as many potential NEs as the SVM, the final results came out similar. The HMM demonstrates better performance than the SVM in the longer articles, and worse performance on shorter articles. The f-measure for some of these articles are pictured in Figure 10 in which the three lowest and three highest scoring articles of the HMM’s results are shown side by side. For the HMM, the lowest scoring articles were the articles with the fewest potential NEs in the text, and the highest had the most potential NEs.

We see in Table 8 that as far as f-measure after resolution goes, all three approaches are comparable in performance, although our especially trained SVM has high recall as tuned. However, the traffic to the Geocoder is a lot less for the HMM or CRF method. Our system allows a choice of all three methods as options although CRF is the default due to its fastest training speed.

#### 9.3.3. Disambiguation of Geospatial Named Entities

We found that there are problems with the context-based disambiguation in North America in areas where the city and feature names have significant overlap. The ultimate result of this is that places with globally unique city and feature names perform the best, as there is no need to run the context disambiguation algorithms. We also found that the worst areas in North America are the Northeast, and the Southwest US where there are significant overlap between place names in the US and place names in Mexico. In general, our context disambiguation algorithm performs at 78%, but in these areas of the country, the performance drops to around 64.5%, due to the significant place name overlap.

#### 9.3.4. Geocoding and Caching Geocoded Results

Once a list of geospatial names are identified, they are fed into Google Geocoder which is a service that attempts to resolve each name as an object with various attributes, including a latitude and longitude. The geocoder is accessed via an online API and can return multiple results if there are multiple places in Google’s catalog that match the provided name. Geocoding success is defined as correctly resolving a string to a single location in the context of the document by the end of our post-processing of the geocoder’s results.

We also perform post-processing on the Geocoder results. We find that the geocoder performs much better on land-based locations than ocean and sea-based. This is primarily because land tends to be better divided and named than the ocean. This problem is found primarily in articles about naval battles, like the Battle of Midway article. We also find that Google Geocoder has much better data in its gazetteer for the US and Western Europe than other areas of the world. While we are able to resolve major cities in Asia and Africa, smaller-scale features are often returned as “not found”. Russian and Eastern Asian locations also introduce the occasional problem of other character sets in Unicode characters. We find that the articles in our corpus do not refer to very many features at smaller than a city level in general, so these geocoding problems do not significantly impact our performance. Historical location names cause some small issues with the geocoder, but our research demonstrates that, for North America and Europe at least, Google Geocoder has historical names in its gazetteer going back at least 300 years.

It should be noted that we cache geocoded results
TABLE 9. Hand-Tagged Articles—Potential Location NEs

<table>
<thead>
<tr>
<th>Article</th>
<th>HMM Extracted Potential NEs / Grounded NEs</th>
<th>SVM Extracted Potential NEs / Grounded NEs</th>
<th>HMM Precision</th>
<th>SVM Precision</th>
<th>HMM Recall</th>
<th>SVM Recall</th>
</tr>
</thead>
<tbody>
<tr>
<td>Chancellorsville</td>
<td>149/47</td>
<td>566/75</td>
<td>0.7015</td>
<td>0.8621</td>
<td>0.4947</td>
<td>0.7895</td>
</tr>
<tr>
<td>Gettysburg</td>
<td>327/115</td>
<td>1209/117</td>
<td>0.7718</td>
<td>0.7267</td>
<td>0.6319</td>
<td>0.6429</td>
</tr>
<tr>
<td>Korean War</td>
<td>625/328</td>
<td>2167/331</td>
<td>0.9371</td>
<td>0.6910</td>
<td>0.8700</td>
<td>0.8780</td>
</tr>
<tr>
<td>War of 1812</td>
<td>752/384</td>
<td>2173/408</td>
<td>0.9165</td>
<td>0.8518</td>
<td>0.7370</td>
<td>0.7831</td>
</tr>
<tr>
<td>World War 2</td>
<td>668/464</td>
<td>1641/448</td>
<td>0.9915</td>
<td>0.9124</td>
<td>0.8609</td>
<td>0.8312</td>
</tr>
</tbody>
</table>

FIGURE 10. Comparing SVM and HMM results for a Subset of Wikipedia Articles

which decreases the number of online geocoding queries by an average of 70%. Google Geocoder limits the speed at which a public user can make consecutive requests so we add a delay of 0.1 seconds between queries. By reducing our online accesses by such a large percentage, we end up saving seconds processing each article. More detailed performance evaluations still need to be made to more clearly identify how well the cache has improved efficiency. Since Google Geocoder outputs have not yet been disambiguated by our system when cached, the cache holds information across the processing of multiple articles. In initial development, it was the single final disambiguated output that was saved into the cache rather than raw geocoder output. This caused problems when two completely different geospatial entities with the same name were referenced in two different articles that would have been distinguished correctly given raw geocoder output. We have since fixed this problem by immediately saving the list of results returned by the geocoder into the cache, rather than the specific location identified after disambiguation. Unfettered access to Google Geocoder costs several tens of thousands of dollars per year.

10. VISUALIZATION

The overall objective of the research presented in this paper is to create a structured dataset that represents the events, entities, temporal data, and grounded locations contained in a corpus of unstructured text. While this is, of course, very valuable for many things, our follow-up objective is the visualization of this structured data from the text. We create a bifurcated GUI that provides the visualizations for the geospatial and event/temporal information. We generate a combination map and timeline using the Google Maps API\textsuperscript{15} and the MIT SIMILE Timeline\textsuperscript{16}.

The map interface provides a clickable list of events on the right, with an associated map of the events. Clicking on a location on the map or in the sidebar shows an infobox on the map that provides more details about the location, including the sentence in which it appears. Figure 12 shows this interface, and Figure 13 shows the clicked map. The second part of the interface is a timeline with short summaries of the important events. Clicking on any of the events brings up an infobox with a more detailed description of the event.

\textsuperscript{15}http://code.google.com/apis/maps/documentation/javascript/

\textsuperscript{16}http://www.simile-widgets.org/timeline/
shown in Figure 14.

10.1. Timeline Hot Zones

Key to the timeline visualization is the concept of “hot zones”, or clusters of important events on the timeline. An example set of hot zones is shown in Figure 15. In the figure, the orange bars represent three different hot zones where multiple events have been identified to occur at about the same time. Blue markers, along with the text of the event itself, are aligned vertically in each zone. Events that were not identified to reside in a hot zone are staggered outside of the orange bars.

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 times. Long stretches of low differences indicate dense time intervals, indicating a hot zone. It is not necessary for the differences to be zero, just low, so a measure is needed for how low is acceptable. We choose 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 $Q3 + 3 \times (Q3 – Q1)$ or less than $Q1 – 3 \times (Q3 – Q1)$. The latter quantity is usually zero, so this ends up removing high outliers. This average is the threshold below which a difference is called low. We also ensure that too many hot zones are not generated, so we choose a threshold for the number of consecutive differences that is low for the interval to be a hot zones; we choose a threshold of 10, based on what is appropriate for the dates in the majority of articles we process.

10.2. Article Processing Flow

The website both allows a user to search for and visualize articles, and serves to control the processing of new articles as they are requested. Figure 16 shows the processing flow of articles. While the system constantly queues and processes articles, articles that have not been previously processed that are requested by a user are immediately moved to the top of the queue for processing. A user can also choose to re-process an article that has already been processed, if it has been too long since it was last processed as the article content may change in Wikipedia.

11. CONCLUSIONS AND FUTURE WORK

In this paper, we have presented details of a system that puts together many tools and approaches to produce a working system that is useful and interesting. Of course, the work presented here can be improved in many ways.

The performance of the important event classifier could be improved upon in the future. It is possible that more features would help. Ideas for sentence features that were not implemented included calculating the current features for the adjacent sentences. Dependency relations obtained from dependency parsers can be used as well. Classifiers other than SVMs could be explored. Since the task is similar to text summarization, methods used for summarization could be adapted as well. Assuming we perform extractive summarization, the sentences that are selected for the summary for a document can be considered important; these in turn are assumed to contain references to important events. If the number of sentences selected for the summary can be controlled, it will give us control over the number or percentage of sentences selected, and hence, events considered important. Calculating the features to train the classifier takes more time than desirable. 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 extraction of temporal expressions that are explicit 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. Although they work well, most of the algorithms we used here are naïve and do not make use of existing tools. The regular expression extractor works fairly well, but needs expansion to handle relative expressions like “Two
FIGURE 14. Event infobox in the visualization, after clicking “Richmond, VA, USA”

FIGURE 15. Timeline for Mexican-American War, containing 3 hot zones. Key to the timeline visualization is the concept of “hot zones”, or clusters of important events on the timeline. An example set of hot zones is shown in this figure. In the figure, the orange bars represent three different hot zones where multiple events have been identified to occur at about the same time. Blue markers, along with the text of the event itself, are aligned vertically in each zone. Events that were not identified to reside in a hot zone are staggered outside of the orange bars.

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 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 [12], which temporally relates events within one sentence. We need to improve performance when time expressions are implicit. Errors tend to build up and carry over from sentence to sentence; some kind of check or reset condition would help accuracy.

The extracted information is stored in a MYSQL database at this time. We would like to convert it into an ontology based on OWL or RDF so that it can be easily integrated into programs for effective use. For example, this may enable us to allow searches for text content to be filtered by geographic area. The search could then be performed in two steps. The first would be a standard free text search, for articles about the topic. That list could then be further filtered to those articles with appropriate locations associated with them. Reversing this paradigm, the location data provided by the system could also allow location-centric search.

The visualization could also be improved with changes to SIMILE’s code for displaying a timeline,
ACKNOWLEDGEMENT

The work reported in this paper has been partially supported by NSF grant ARRA: 0851783. Rachel Chasin and Daryl Woodward are undergraduate researchers supported by this NSF Research Experience for Undergraduates (REU) Grant. This paper is a much extended version of [92].

REFERENCES


Extracting and Displaying Temporal and Geospatial Entities from Articles on Historical Events


