ClassMate:
A System for Automated Event Extraction from Course Websites

Ashutosh Kulkarni, Harry Robertson

Abstract

Websites contain a huge amount of time-critical data in highly unstructured and heterogeneous form. Information Extraction systems can extract relevant entities and relationships from these sites, and identify, classify and categorize them. In this paper, we present ClassMate, a complete system for extracting key course-related events from university course websites. ClassMate pipelines web data through a Named Entity Recognition module, an window-based event extractor, and a KMeans clustering-based classifier.

1 Introduction

1.1 Problem statement

Students in today's universities are required to follow a multitude of online sources of information in order to organize their daily lives and their studies. In particular, time-critical information pertaining to each of the student's classes (notably office hours, assignment deadlines, and exam times) is generally only available on that class's website. Even within a class's website, the information is often posted haphazardly, with no real standardization of presentation[1].

From a Natural Language Processing perspective, this corresponds to an Information Extraction problem. The information required by the student is contained in non-trivial text form in a large corpus of data (i.e. the course web pages). We wish to extract the relevant time-critical information entities (henceforth known as *events*), and classify them appropriately.

1.2 Related work

The problem of extracting and classifying useful information entities/relationships has been well studied in the NLP literature. The starting point for IE systems is generally Named Entity Recognition, which identifies and labels basic word groups of certain categories (such as "person" or "organization"). Several NER systems (which typically use word lists, rules, and some learning) are publicly available, notably ANNIE which we examine in more detail later in this paper.

As is usually the case for Machine Learning-type problems, approaches to relation identification and classification for input text data can be either supervised or unsupervised (or somewhere in between with "lightly supervised" methods). When labelled training data can be obtained, supervised learning approaches generally work best; in recent year Zhou

[1] Of course, while we take a very student-centric vision here, this is more generally representative of the problem faced by any active individual who has multiple time-critical online information sources.

et al in particular have had success by applying Support Vector Machines to the problem (see [3]). Other standard Machine Learning methods such as Naive Bayes can obviously also be applied, as can typical NLP approaches such as Maximum Entropy Markov models.

Without extensive labeled training data, lightly supervised methods can be applied. Notably one can use "seed" rules and examples of the relationships we wish to identify in order to bootstrap a system, which can then learn other rules by exploiting text similarity and other properties. Etzioni et al's KnowItAll system applies such approaches to fact extraction on the Internet with no hand-labelled training data (see [1]). Most recently, Michelson and Knoblock use an unsupervised reference-set learning-based approach to perform Information Extraction on Craigslist classified ads, with results which they claim are competitive with supervised reference-set approaches (see [5]).

A promising paradigm for event extraction, at least when the events tend to follow a typical script, is template filling. In such approaches one represents categories of events by prototypical templates, and then tries to match the slots of the template to sequences of text. Such approaches can go beyond simple rules to use statistical methods such as HMMs. Interestingly, evolved versions of these approaches have been applied by McCallum to a problem similar to the one at hand, that of extracting basic static course information from college websites (see [4]).

1.3 Our approach

The problem studied here has certain key difficulties which make it an interesting research, as well as just practical, problem:

- While there is obviously an abundance of course websites on which to test, there is no available corpus of annotated pages (suitable for use as training data).

- The events we wish to extract vary enormously in format, from proper English sentences ("I will be holding office hours tomorrow in Gates 248") to unstructured telegraph format ("TA hours: MW 9-10am, 248").

- The events we are concerned with are sufficiently general to not easily be fit by templates (e.g. the only essential characteristic of an *assignment due* event, in terms of word roles, is the presence of a date).

The first difficulty precludes the use of supervised IE methods, at least without the great investment in time and manpower required to label relevant training data. The second and third make simple handwritten rules or template-filling schemes much less effective. These characteristics motivated our decision to follow an unsupervised approach to solving the problem.

In this paper we present our solution to the student's problem, the ClassMate system. ClassMate is an end-to-end solution, taking as input a set of courses, and outputting a set properly extracted and classified time-critical events. To do this it processes the data through a sequence of modules: it extracts the data from the web, annotates it, extracts candidate events, computes features, clusters the events into categories, and finally labels the categories.

In the following section we present the architecture of the system and the key design and implementation decisions behind each of the main components. We then analyze the performance of each component, both quantitatively and qualitatively through example analysis. Finally we give directions for future work.

2 The ClassMate system

2.1 System architecture

We had two primary goals in building ClassMate: the first was to develop an end-to-end system capable of taking as input a set of courses, and giving as output all of the corresponding time-critical events, properly labeled and categorized. The second was to tackle genuine active research problems (namely unsupervised Information Extraction and text classification) as part of this task.

To achieve these goals, we built ClassMate following a pipeline architecture: the input data is processed and transformed by a linear succession of components. The key components are:

- Web crawler for obtaining HTML documents
- Named Entity Recognition (annotation of all words with basic information-type tags)
- Extraction of event entities
- Calculation of features for each event
- Clustering of the events
- Labeling of the clusters to obtain final categories

Our key architectural concern in building the system was to maintain modularity as much as possible, both to allow easy evaluation of the different components in relative isolation, and to be able to easily switch in and out different versions of each component without affecting the rest of the system. In particular, we separated most of the "business logic" (e.g. the feature specifications) from the main code by storing it in flat files and reading it into the program dynamically when necessary.

All but the first two components are fully implemented and integrated in our main Java code base. For the Crawler and Annotation components, we chose to use off-the-shelf solutions, both to save development time, and as they represent much less interesting research questions. In the following subsections we look at each component in more detail.

2.2 Web crawler

Statistical NLP systems need large amounts of training and testing data. Collecting data for the system was one of the major tasks. We wrote a simple python script and used freely available website crawling software (httrack) to collect data from University websites. We filtered out files other than the actual HTML content of the webpages since we planned to concentrate only on the HTML documents for this project. The data we downloaded contained approximately 3391 events (The number is not exact since it is calculated using output of ANNIE NER system which may not be 100% accurate) from 283 course webpages downloaded from Stanford as well as Gatech course websites. we specifically chose to download the webpages from computer science courses since those websites were updated frequently and had more and better information content.

2.3 Named Entity Recognition

2.3.1 ANNIE

The data collected from the crawler needed to be annotated with certain tags - most important of them being *date* along with *person*, *location* and possibly *email address*. We thus needed an NER system which was good at tagging dates since the accuracy of this system plays an important part in the overall performance of ClassMate. We chose ANNIE (A Nearly New information extraction system) based on performance analyses studied elsewhere.

ANNIE is a component of a bigger information extraction infrastructure named GATE (General Architecture for Text Engineering) designed for natural language information extraction (see [2] for further details on the system and its performance). It provides various sub-components like an English tokenizer, an extensible gazetteer, sentence splitter, POS tagger, orthographic coreferencer etc. These components can be used independently or sequentially.

We quickly confirmed that ANNIE is relatively good at tagging dates - both those explicitly formatted like "Wednesday", "February 2003", "14-06-08" as well as ones composed of phrases like "tomorrow", "today", "next week" etc. We ran ANNIE on all of the webpages downloaded by the crawler to get obtain annotated files.

2.3.2 Processing ANNIE output

The output of ANNIE is an xml file with information such as tag type, the rule which determined the tag, and features that were associated with the word. It also contains the original HTML tag information stored in a similar format to ANNIE's tag types. We thus needed to convert this format into a format that was usable by the event extraction system. We wrote a python script to parse this xml file and convert it into a suitable format.

This pre-processing stage went through several iterations, influenced by error analysis and performance evaluation of the ClassMate system. In the final version, we provide as much information as possible to the event extraction module to allow it to intelligently extract the events. Split tags (i.e. tags which define where a sentence is split: '.', ':' etc.) and HTML tags associated with the webpage were integrated along with other tags, in order to take maximal advantage of sentence structures and the HTML structure of each web page. More details on the iterations of the pre-processing stage are discussed in the error analysis and improvement iterations section.

2.4 Event Extraction

At the core of ClassMate is the Event Extractor. Indeed, in its purest form ClassMate simply performs Information Extraction following certain kinds of "templates" (i.e. class-related events). However, it is difficult to reduce the problem to a standard template filling due to the vaguely defined nature of the events we are concerned with (an assignment event, for example, has to have a date slot, but may or not have a location, a description, etc.) and the absence of detailed annotations for our data (the only useful tags ANNIE provides us with are *date*, *person* and *location*, and they are far from having perfect recall). We also completely lack training data, which would have allowed us to take a Machine Learning style / statistical approach to classifying items as events.

We therefore adopted a relatively home-grown and pragmatic approach to event extraction. We initially assume that every word sequence annotated by ANNIE as a *date* of some sort is part of an event[2]. For each date, we search for a window of text around the date (within certain min and max size limits) which is, if possible, delimited by splitters (e.g. punctuation which indicates a break in the text, or HTML tags such as "</table>").

This window-based approach is designed to give good recall at the expense of precision (with the hope that the "bad" events can be caught later). The event extractor contains several parameters (such as min and max window size, the splitters we considers, etc.) which we iteratively tuned essentially through trial and error.

[2]While this approach may seem fallacious, bogus events should be weeded out later on in the cluster step, as they will be classified as *other*

2.5 Classification

2.5.1 Features

In order to perform any kind of classification of the extracted events, we need to calculate numeric features to obtain a feature vector. We experimented with a variety of features. The essential ones we retained are:

- Binary features for the presence of each possible word (of course this leads to very sparse feature vectors).
- Binary features for the presence of each HTML tags.
- Weighted features for the presence of certain key words (such as "assignment", "office hours").
- Weighted features for the presence of certain key relevant tags (such as "location").

We set the sets of key words / tags and weights by hand, relying on our intuition of the form of the events we were looking for (we examine our feature development process in depth in the next Section).

Given more time, a more sophisticated and possibly more efficient approach would have been to learn these keywords and weights with some form of optimization algorithm, such as basic hill-climbing. This would have required much more processing time (as it requires repeating the classification task many times in order to learn the right weights), and also validation data (in order to evaluate the performance of the classifier with a given set of weights).

2.5.2 KMeans

Given the feature vectors, there is a plethora of possible algorithms for classifying the events into categories. The essential choice was between adopting a supervised, a lightly supervised, or an unsupervised approach. We chose to build an unsupervised classifier, both for the academic challenge (unsupervised classification is considerably harder and less studied / understood) and due to our lack of training data.

We implemented the KMeans clustering algorithm to cluster the events. The KMeans algorithm is summarized here:

1. Initialize K clusters randomly.
2. At each iteration, calculate the centroids of each cluster (i.e. the average of all the points currently in the cluster, which is the point which minimizes the sum of the squared distances to the points in the cluster).
3. Then allocate each point to the cluster whose centroid it is closest to.
4. Iterate until the clustering is stable (i.e. there are no changes in a whole iteration).

There are various parameters one can modify, the most obvious being the value of K. For the problem at hand, we wish to identify events in three categories (office hours, assignment, and exams), and there is an implicit fourth category (*other*). So $K = 4$ is the obvious choice. However we can, we can also fix $K > 4$, and consider that K-3 of the clusters all correspond to *other*. We experimented with this, and found that $K = 5$ did indeed give better overall results than $K = 4$. This is no doubt accounted for by the fact that our *other* category encapsulates a tremendous amount of diversity in the potential events, which certainly cannot be represented by one point in the feature space. So by allowing for several *other* clusters, the clusters for the categories we are interested in can correspond to more focused regions of the feature space, while the *other* centroids mop up the other points. However, once we introduced more specific bigram features, this conclusion

became a lot less clear; in its current state, it would appear that our system performs better with $K = 4$.

2.5.3 Derandomization

A major problem we encountered while experimenting with KMeans was the huge variance in the results obtained for different runs over the same data set. Indeed, the initial centroids of the clusters are chosen randomly, and different choices can lead to drastically different clusterings. This is related to another major problem with KMeans: the algorithm is only guaranteed to find a local minimum (with respect to the mean squared distances of points to their cluster's centroids).

To counter this, we attempted to "nudge" KMeans in the right direction, by intelligently choosing the initial cluster centroids. We use simple heuristics to identify a good candidate for each of our three categories, and set these three points to be the centroids of the first three clusters. We then deterministically (although this is not necessary) set the other centroids. This approach has the dual advantages of improving general classification performance and removing the random aspect of KMeans (which is not necessarily desirable, but does make results analysis simpler and reproducible).

2.5.4 Cluster labeling

For the clustering generated above to be usable, it is necessary to actually label the clusters to show which cluster corresponds to which category. To do this we again use very simple heuristics (the percentage of occurrences of certain keywords for each category in the events making up a cluster) to identify the best candidate cluster for each of the three categories. We label the remaining clusters as *other*.

Of course, the fundamental underlying problem is in what measure KMeans' LMS optimization criteria actually correspond to our categorization goal. This is clearly always a problem with unsupervised classification especially. We attempt to subsume it by employing extremely suggestive features (e.g. by giving a very weight to the presence of the bigram "office hours" in an event), but it is impossible to achieve perfect alignment of KMeans' goals with ours.

3 Results analysis

3.1 Experimental process

As we have seen, the ClassMate system pipelines course information through several successive information extraction components, and over-all system performance is therefore dependent on the combination of all these steps. However, for the purposes of analyzing and improving performance, it is more useful to experimentally isolate each primary component and evaluate the quality of its output assuming its input is perfect.

We spent the most time evaluating and iterating on the Event Extraction and KMeans components. While ANNIE is also pivotal in the success of our system, its performance is already proven and has been well studied elsewhere.

As our system is essentially unsupervised (in that no labeled training data is required), evaluation of our system is of course complicated by the corresponding lack of available test data. In order to obtain some measure of quantitative results for Event Extraction, we therefore manually extracted events from randomly selected webpages (in order to evaluate Recall), and hand-evaluated around 200 automatically extracted events (in order to evaluate Precision). For KMeans, we hand-classified around 700 automatically extracted events as *Office hours*, *Assignments*, *Exams* or *Other* and evaluated the percentage of correct classification by KMeans relative to these.

While the quantitative results thus obtained give some idea of system performance, our test sets are too small to provide a good degree of statistical confidence. We therefore actually relied more on error analysis on specific examples in order to find the strengths and areas of improvement for our system. We present some interesting examples of this below.

3.2 Error analysis and iterative improvement

Classmate system went through many iterations, each iteration improved the performance of the system based on the error analysis done on the previous versions. We present here a summary of the changes we went through.

Version 0.1 The first version of ClassMate included:

1. Pre-processed data with *person*, *location*, *date*, *address* and *other* tags
2. A fixed-sized window based event extractor
3. A randomized KMeans classifier (the initialization of the K centroids was done randomly)
4. A simple cluster labeling system

The output of this version was dissappointing, mainly because KMeans had such strong variance that the variations between runs dwarfed any benefits of the classification. To illustrate this, the table below shows the output of 3 consecutive runs of the randomized KMeans classifier.

Run	Cluster	Size
First	Office hours	914
	Assignments	453
	Exam	117
	Others	1907
Second	Office hours	158
	Assignments	103
	Exam	2259
	Others	871
Third	Office hours	690
	Assignments	107
	Exam	1672
	Others	922

We realized that in our system, the initialization of the cluster points can be done intelligently given that a small select set of features is very representative of each particular class (e.g. phrases like "Office hours", "final exam", "assignment due" are clear representatives of *Office hours*, *Exam* and *Assignment* events respectively). We thus used an intelligent intialization of KMeans centroids wherein each centroid was assigned according to a few initial features that were representative of each of these event classes. This approach improved general classification performance and also removed the random aspect of KMeans.

Version 0.2 Thus the next iteration included the deterministically initialized KMeans classifier descirbed above. The output of this system was stable and gave the same results for every run.

We found that the extracted events contained lot of sentence specific information that we could make use of. Specifically, many of the fixed-sized window extracted events contained parts of two different sentences, we could remove the unnecessary part of the next sentence by considering the delimiters in the sentence. Also, we realized that the dates could be subclassified as time, date and datetime

tags since many a times this type of differentiation was representative of a particular type. For example, publications always contained year and month of publication ("February 2003"), while announcements contained only the date at which they were posted ("04/06/2008"), and exam dates were of the type *datetime* (e.g. "Finals will be held on 5th June 2008 from 12:00 pm to 3:15 pm in Gates B01").

We therefore modified the pre-processing stage to contain these subclasses of dates and delimiters of the sentences. the event extraction system was also modified to take these cases into consideration, and have a variable sized window depending on the delimiters present.

Version 0.3 This noticeably improved the performance of system giving better precision for each of the classes. We noticed that we could make use of special HTML tags like "<table>", "<title>" and "<h1>" to maintain the structure in the data and make event extraction system better. We therefore added HTML tags extraction in the pre-processing step to preserve HTML tag information .

Thus our final version contains:

1. Pre-processed data with *person*, *location*, *date_date*, *date_datetime*, *date_time*, *address*, *split*, HTML tags (*table*, *td*, *tr*, *h1*, *h2*, *h3*, *title*) and *other* tags
2. An intelligent variable-sized window based event extractor
3. A deterministically initialized KMeans classifier
4. A simple cluster labeling system

3.3 Feature engineering process

To further improve our system's performance we continuously re-engineered our feature set.

We initially used the standard features of any information extraction system - word-presence features (one for every word in the corpus). The most obvious extension is to have representative words as the features - "office", "hours", "assignment", "homework", "due", "deadline" and "exam". We gave high weights to highly representative words (for example, the word "office" in an event would almost always mean an *office hour* event given that the corpus is limited to course web pages).

After analysing the output, we realized that events tended to have a specific set of tags associated with the words in it depending on its class. For example, consider the following office hour event:

> Ada Gavrilovska , KACB 3228 , 404 894 0387 ada @ cc Office Hours : Tue 1:30 - 2pm , Wed. 2 - 3pm ,

This event contains a person name, a location, an email address, and the timings of office hours. We realized that this set of tags could be representative of this class of events. We therefore added word tags as features.

We also noticed that bigram can be useful features too; for example a bigram "Office hours" present in the event almost always means that the event is an *office hours* event, much more so that if there is "office" or "hours" separately. There are however a few cases where this is not true, such as:

> Symposium Project 3 posted RPC handout Project 2 deadline extended until 11:59pm on Sunday , 3 / 16 Extra office hours Mon

This event contains the bigram "office hours" but still is not an office hour event since the bigram appears right at the end of the event window. We attempted to compensate for this by taking into account the location of the key words and tags within the event window. That said, bigrams such as "office hours", "due date", "final exam" are obviously extremely powerful features.

We also noticed that university buildings are generally referred to in a peculiar, university specific way (e.g. "Gates", "Hewlett" at Stanford University); when these words come in the context of the corresponding educational institution, these words will always mean a location. ANNIE on the other hand, being a generalized NER system, was not able to classify them as locations. We therefore extracted these words from the dataset manually and added these as specialized tag features. There is a similar case with words like "MW", "TTh", "MWF" which are abbreviations of "Monday-Wednesday", "Tuesday-Thursday", "Monday-Wednesday-Friday". These abbreviations are fairly commonly used on course web pages and should have the specific tag *date_date*. We therefore also added these abbreviations as specialized tag features.

Conversely, certain words (such as "et. al.", "IEEE" and "proceedings", which all tend to indicate papers) can help classify an event into the *other* event category, and thus make useful features.

3.4 Performance results

3.4.1 Classification performance

Evaluating a system with unsupervised learning approach is difficult since there is no gold data to compare results with. We manually went through 700 events in total (divided in the ratio proportional to the total number of events in each class) and calculated precision for each of the classes. The overall system performance including the errors in the output of ANNIE is given by the following table.

Event	Precision	Recall	F1 score
Office Hours	70%	93.3%	0.799
Exam dates	80%	93.02%	0.860
Due dates	88%	92.63%	0.902
Overall	79.33%	92.99%	0.854

We realize that our evaluation data is limited and the performance of the system in the real world would be lower than the overall performance mentioned in the table. We expect F1 score to be in the range of 0.75 - 0.80 for the real world situations. It is not surptising that our recall results are high, given that our feature set included features like the bigrams "office hours", "due dates", "final exams" with high weights. Therefore, for example, an *office hours* event getting misclassified even though it contained word "office hours" is very rare. Precision on the other hand was lower since few other events were also classified as *office hours* merely because the event window contained the text "office hours". For example consider the following example:

> Adam Steinberg (Ooyala) presents applied Computer Vision Friday at 11am . TA office hours

This event is misclassified as an *office hours* event, when it is actually just a lecture date, due to the presence of the bigram "office hours" in the excessively large window around the time "11am".,This is an example of classifier errors due to imprecision in upstream event extraction.

3.4.2 Ablation study

As we built ClassMate, we added features iteratively as we attempted to improve performance. In order to validate our decisions, we performed an ablation study on the final system: we evaluated classifier performance with keyword features (unigram and bigram) removed, with just key bigram features removed, and with just the key tag features removed. The results are shown in figure 1.

These results clearly show the huge importance of the keyword features; classification is basically random with them. This is not surprising given that it is essentially these features which inform KMeans of

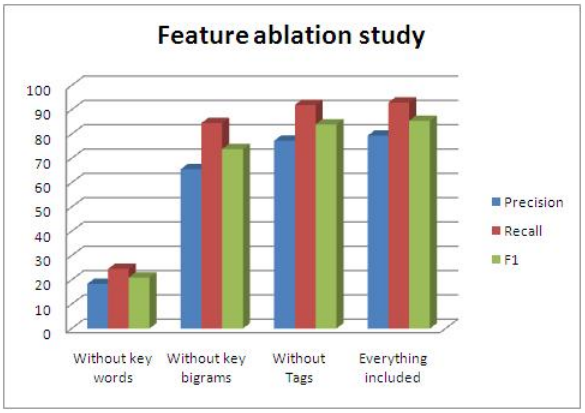

Figure 1: Classifier performance with assorted features ablated

our categorization criteria. Bigram features also provide a strong performance boost, of almost 10% to both Precision and Recall; we believe this is due to the importance of composite phrases such as "office hours" which cannot be captured just by the presence of all or some of their component parts separately. The key tag features, however, appear provide very little performance benefit. They do give a tangible improvement when bigram features (which we developed after) are used, but they clearly do not have any further synergy with the bigram features. Given how generic the tags provided by ANNIE are, it is not particularly surprising that the tag features are not very useful to the classifier; more fine grained tags (such as "room" or "course") would have been of more potential use.

3.4.3 Effect of tagging errors

In performing error analysis we observed that many classification errors were actually due to errors in ANNIE's tagging. For example, the subject name "Ling" is systematically classified as a *person*. Even worse, many numbers (such as the room number in "Klaus 2402") are misclassified as *dates*, which obviously leads to serious event extraction and classification errors. To quantify the effect of these errors, we reevaluated Precision and Recall while discarding events containing ANNIE-tagging errors, obtaining the following results:

Event	Precision	Recall	F1 score
Office Hours	90.9%	93.3%	0.920
Exam dates	80%	93.02%	0.860
Due dates	90.72%	92.63%	0.916
Overall	87.2%	92.9%	0.899

We note that Recall is pretty much unchanged when accounting for ANNIE errors. Precision for *office hours* in particular however is greatly boosted, no doubt due to the misclassification of many *locations* as *dates*.

3.4.4 Influence of data set size

Finally, one particularly interesting parameter is the effect of the overall data set size on system performance. Figure 2 shows the of performance when we use only 10% or 50% of our data set. It is interesting to note that, despite the fact that our algorithm does not use any supervized learning (which typically benefit from larger training data sets), performance improves significantly with larger data sets, albeit with diminishing returns. We hypothesize that this is due the smoothing effect that introducing more points into the feature space has on the point cloud considered by our classifier.

4 Conclusion and future work

In this paper we have presented ClassMate, a system which applies a variety of advanced Information Extraction techniques to extract and categorize time-critical information from university course websites. While ClassMate is clearly far from ready for deployment in a real-world setting, our results do show

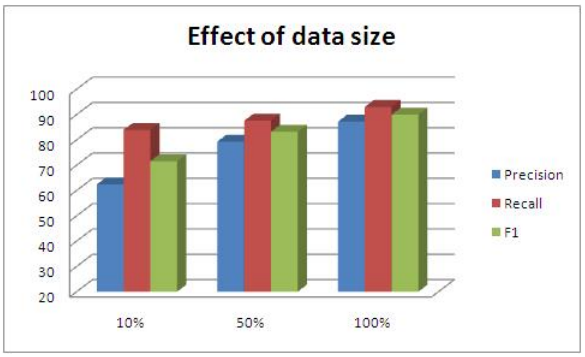

Figure 2: Effect of data set size on classifier performance

that an end-to-end solution is feasible. Our primary academic innovation, beyond the domain of application, is the use of unsupervised clustering techniques coupled with simple word-density heuristics to perform classification.

In future work we plan to improve our primitive event extraction algorithm. A template-based approach could provide much better Precision for event extraction, as our current system equates the presence of a date with an event, which is not always the case. While simple templates would probably provide low recall, we can alleviate this by boot-strapping a statistical algorithm to learn a variety of event templates by induction from our data sets, yielding high coverage (similarly to [4]). We can adapt some of our current hand-written features to form seed rules for boot-strapping such a system.

Another key problem which we have not tackled in the present work is how to situate the events our system extracts which do not have absolute dates (such as "Final exam tomorrow in Gates 104"). While placing all events in an absolute timeline would be challenging to say the least, we could utilize words indicating temporal relations (such as "before" and "tomorrow") to establish a partial ordering of events, which would be useful in itself, and could provide bounds on absolute dates.

References

[1] Oren Etzioni, Michael Cafarella, Doug Downey, Ana-Maria Popescu, Tal Shaked, Stephen Soderland, Daniel S. Weld, and Alexander Yates. Unsupervised named-entity extraction from the web: an experimental study. *Artif. Intell.*, 165(1):91–134, 2005.

[2] Hamish GATE: Cunningham, Diana Maynard, Kalina Bontcheva, and Valentin Tablan. A framework and graphical development environment for robust nlp tools and applications. In *Proceedings of the 40th Annual Meeting of the Association for Computational Linguistics (ACL)*, pages 168–175, 2002.

[3] Zhou GuoDong, Su Jian, Zhang Jie, and Zhang Min. Exploring various knowledge in relation extraction. In *ACL '05: Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 427–434, Morristown, NJ, USA, 2005. Association for Computational Linguistics.

[4] Andrew McCallum. Information extraction: distilling structured data from unstructured text. *Queue*, 3(9):48–57, 2005.

[5] Matthew Michelson and Craig A. Knoblock. Unsupervised information extraction from unstructured, ungrammatical data sources on the world wide web. *Int. J. Doc. Anal. Recognit.*, 10(3):211–226, 2007.