

First Person Narrative Story Extraction and Retrieval

Reid Swanson

University of Southern California
The Institute for Creative Technologies
swansonr@ict.usc.edu

1 Introduction

The ability to acquire, communicate and transfer complex knowledge is one of the fundamental characteristics that make us human. It should be no surprise that knowledge representation and management is seen as a fundamental problem in Artificial Intelligence research. There is certainly no requirement that a machine encode this knowledge the same way humans do. However, any decisions in representation must be made with great care if there is any hope of computer agents being able to communicate effectively and coherently in a human environment. Like many other problems finding the right level of granularity for representation is a difficult open problem. Schank & Abelson [18] argue that, despite longstanding beliefs, propositional knowledge is not the primary mechanism for knowledge storage or communication in humans. Rather stories, the first person narrative accounts of events, are much more fundamental and offer a more flexible structure for manipulation. For example, stories about similar events share many structural elements often referred to as the script or skeleton. Slight perturbations in the telling of a story can be used to embellish the same story to convey different moral outcomes or elicit different emotional responses from the listener. Multiple views are also possible from a single story offering different interpretations based on the role of the teller. The ability to pass knowledge through the telling of such a rich representation makes the value of a large collection of stories immediately obvious. In fact story telling is seen as an important process in the management sciences for assess-

ing performance and directions for change. Another important application, particularly useful to the projects at the Institute for Creative Technologies, is the integration of story collections into training technologies and simulations. Johnson et al. [9] and Gordon [3] have separately shown ways to integrate story collections into a working environment to improve the training and performance of individuals.

Despite the inescapable prevalence of stories around us an effective means for collecting and managing large-scale repositories has yet to be developed. Several manual collection programs have been tried, in specialized domains, such as the WPA Life Histories Collection by the Library of Congress (memory.loc.gov/ammem/wpaintro/). Unfortunately large-scale manual collection endeavors such as these are astronomically expensive and require an extraordinary amount of time to put together. Another collection strategy is through a guided interview process where a trained expert stimulates the conversation in a way to maximize the story content told. If non-fictional stories are not a requirement then a third possible way to acquire a collection of stories is simply to hire a staff of writers. This approach, too, is impractical for very large collections and often lacks the breadth of details that are found in real world experiences.

Every year sees a vast increase in the number of people with online access. This incredible rise of users has created a critical mass that has materialized into large online communities and an explosion of personal content being exchanged through the popularity of Weblogs. With over 70 million accessible Weblogs vast amounts of stories are

likely to be publicly available already. There is no other place where such a diverse collection of stories in nearly any domain can be found. However, despite these stories being out in the open there is still no easy way to identify, manage or use them. In fact, through analysis of over 100 Weblogs, only about 17% of the text contained in them is actually pertaining to a story. The vast majority of user driven content is opinion, complaints or other non-narrative statements. Filtering these sites by hand poses the same problems as other manual collection strategies. However, web data is already in a machine-readable format that is easily adaptable for high performance machine learning approaches in the Computational Linguistics community.

In this paper I will describe the components used to update and improve the Story Upgrade project at the Institute for Creative Technologies. This project is geared towards aiding content developers and writers of immersive training environments by giving them access to a wide range of stories on topics directly related to their interests. There are two major goals for this system. The first is to improve the quality of fictional scenarios that are authored for immersive training environments at ICT. The hope is that the breadth of stories returned by the Story Upgrade system can provide much richer details, more expansive story branches and topics that are up to date and relevant to the needs of the user. Despite offering even richer content this system should enable the development time and cost to be drastically reduced by minimizing the labor the author must do from scratch. There are three major components to the new Story Upgrade system that will be described in this paper. I will first summarize the work done to improve the story extraction. Second I will discuss work done in semantic role labeling. Third I will describe the Story Upgrade system and how the two previous components were used to try to improve the original Story Upgrade system.

1.1 Story Upgrade Version 1

Story Upgrade is a search engine designed to turn a boring story into a collection of interesting ones. The idea is that the user has a sample story or experience in mind but would like to find other more interesting stories about similar events. These stories could then be used to enhance fictional stories

through sampling and recombination. Gordon & Ganesan [5] showed that it was possible to harvest large collections of stories from the Web. Their original approach was intended to extract stories from conversational speech data using an off the shelf automatic speech recognition system to first transcribe the data into text streams. The text was classified as story or non-story using a Naïve Bayes classifier over fixed length windows of text. Classification of this sort is deficient, however, because it does not account for the clear sequential regularities of story text. Story text is not evenly distributed within the document, but is grouped together such that story text is much more likely to be located near other story text. Two steps were taken to try to account for this distribution of sequences. The first was finding an optimal window width, which was found to be approximately 50 words. The other was to interpret the classification output as confidence values and smooth the results using a mean-average function. This essentially has the effect of increasing the confidence of blocks that surround high confidence intervals and lowering those that do not. Unfortunately the low performance of the speech recognition doomed this approach from the start, although, this same technique was then applied on hand-derived transcriptions and ultimately on Weblog data with much greater success.

This classifier was then used to amass a large database of stories from the web. Over the course of 373 days a web crawler downloaded the story content of over 400,000 Weblogs and resulted in a collection of over 1 billion words of story data. This data set was then indexed for searching using Indri (<http://www.lemurproject.org/indri/>), a publicly available search engine toolkit. A simple web interface was set up to allow users to quickly find hundreds of stories relevant to their interests.

2 Story Extraction Revised

Like the previous approach, the initial discovery of story text is taken to be a binary classification problem, which allows the use of high performance machine learning techniques. However, several modifications to the pipeline are made to try to leverage more sophisticated techniques for processing the data. This section of the paper will summarize the relevant work done with Gordon

and Cao [4] that will elaborate on these techniques further. First I will describe the new preprocessing steps applied to the data followed by a brief discussion of the Support Vector Machines classifying algorithm that was used in place of Naïve Bayes. This will then be followed by a description of the feature sets used for classification and finally the improved smoothing techniques used.

2.1 Preprocessing Internet Weblogs

The raw HTML data downloaded from the web is not easily adapted for extracting features suitable for use with the SVM classifier. In an ideal situation we would like to extract only Weblog entries on a particular page and be able to maintain the meta-information about each post, such as the author and entry date. However, given an arbitrary HTML page this is not a trivial task. A primary reason this is particularly difficult is because there are no standard formats for Weblogs and each software application for posting content can format the page in an arbitrary way. Although there have been techniques developed to try to learn these associations automatically, they can be quite complex and are a research topic in their own right. However, for this application a much simpler technique should suffice; stripping the HTML markup and keeping only the plain text between tags. This approach is simple, fast and will work on any HTML page. The downside is that we lose any related information to the posts. Although this is unfortunate for some uses of the data it is unlikely that these features would significantly affect the performance of story classification. Slightly more concerning than losing the structural information is that non content text, like advertisements and link names, also creep into the data. This does pose some problems, but for the most part the machine learning algorithm should be able to easily classify these segments correctly as non story text.

As a classification problem the (cleaned up) data needs to be split up into chunks that can be independently labeled as story or non story. Choosing the right granularity is an important aspect that will determine the overall performance of the system. In Gordon & Ganesan a sliding window of 50 words was taken to be the unit of classification. This has the benefit of exploiting the distribution of story text. However, it ignores natural sentence

boundaries that may prove useful for identifying the beginning and ending of stories. Working with sentence boundaries also allows for more complex features such as syntactic information. To try to leverage these features the additional step of sentence delimited delimited the text using a maximum entropy toolkit, MXTERMINATOR [17], was taken. A more complete description of these features will be given a little further on.

A new annotated corpus was created for the development and evaluation of the new story extraction system. Using the collection of over 400,000 Weblogs found in the previous work 150 random sites were downloaded and preprocessed as described. Each sentence was then either marked as a story or not based on the following definition from Gordon & Ganesan:

“The definition of a story is somewhat ambiguous. Generally, the stories that people tell are about events that have happened in the past. Accordingly, people use a lot of past tense verbs (e.g. said, went, gave) when telling stories. However, not all descriptions of events that happened in the past count as stories. Stories give descriptions of specific events that actually occurred, not generalizations over multiple events or times. Stories generally have a sequential structure to them, providing a description of events that happened one after another. Collectively, these events are composed to create a complete narrative. Finally, stories usually have some point to them: the reason that the person is telling the story in the first place. Sometimes stories are truly pointless, though, but some message is usually still conveyed.”

To assess the degree in which people's assessments agreed on actual data Gordon & Ganesan performed an inter-rater reliability study. They found using this definition a Kappa score of 0.68 could be achieved. Of these 150 Weblogs 100 were used for training and testing and 50 were held out for development and parameter tuning.

2.2 Support Vector Machines

There are several machine learning algorithms that are commonly used in natural language processing systems. Support Vector Machines (SVM) are one of these methods that typically perform among the best in a wide variety of tasks. Recently through the use of new extensions that will be described in

more detail later, they have also become more attractive for exploiting higher level syntactic features. For this work the SVM toolkit SVM-Light [8] was chosen as a replacement for the Naïve Bayes learning algorithm in Gordon & Ganesan. Like many other classification algorithms training examples are treated as a class label and feature vector pair, where the class label can either be +1 for a positive example or -1 for a negative example. The SVM learning algorithm then uses these example to find the linear hyperplane that separates the example by the minimum margin. In the cases where a linear hyperplane is unlikely to separate the data well a kernel function is used to transform the data into a higher dimensional space. Polynomial and the Radial Basis Function are two common kernel functions.

Most of the kernel functions used in practice have additional parameters that need to be set in advance. Although many problems share similarities, often even slight differences can have wide ranging effects on the performance of a particular set of parameter values. Because it is difficult to know in advance which kernel or parameter values will perform well it is usually necessary to search for suitable values. To find these values the approach outlined by Hsu et al. [7] was applied. As suggested the RBF kernel was chosen, because of the limited number of parameters, and a segment of held out data roughly one quarter the size of the training/test corpus, was used to select the kernel parameters. A grid search was performed and the c and γ values yielding the highest average results over a 10-fold cross validation were chosen.

2.3 Encoding Text as Feature Vectors

There are infinitely many ways the text could be encoded as a feature vector for use with the SVM machine-learning algorithm. In this section I will present five encodings that were tried in this experiment. For most of these approaches a variant of a bag-of-words encoding was used. In this representation a word or short sequences of words (n-grams) are used as the component in the feature vector irrespective of the position in the sentence. In the simplest case the value of the feature represents existence of the feature in the sentence, e.g. 1.0 if the word or n-gram is present and 0.0 if it is not. Slightly more complex representations will be de-

scribed further into this section. Another common theme throughout most of the representations is a method for capturing the context in which the sentence occurs. Sentences in stories do not occur in isolation and the determination of whether an utterance is part of a story cannot solely be determined by the words in that utterance. Capturing long distance relationships in most NLP tasks is a difficult problem and is no different in this case. To at least gain some useful information an analogous idea to n-grams was applied at the sentence level. Not only was each n-gram in the current sentence (the one to be classified) used as a feature, but also each n-gram in the previous and next sentence as well. This can be seen as similar to the Gordon & Ganesan approach of using a window of text, although in this case the larger window is being used to help classify a more finely grained segment. Additionally each n-gram was given a distinct marker for identifying which sentence it came from (previous, current, next) that also gives the machine-learning algorithm more information on which to discriminate.

Version 1: The first version of the system forms the basis for four of the feature sets. This version follows the basic guidelines as outlined above. It uses a simple bag-of-words encoding using n-grams of length 1, 2 and 3 from the previous, current and next sentence. Although using n-grams is relatively straightforward a few other decisions are of concern. Primarily determining exactly what to count as a word. In English space characters are usually a first approximation for tokenizing the text. However, this is not usually sufficient mainly because of punctuation. Ignoring the issue and using these tokens is not necessarily wrong, but has the effect of significantly increasing the size of one's vocabulary. One generally does not have enough training data however, and to avoid this data sparsity problem a typical approach is to lower case all the words and remove all the punctuation. In this case removing punctuation does not seem like the best approach, because the use of exclamation points and question marks can give useful clues. A UNIX sed script (www.cis.upenn.edu/~treebank/tokenizer.sed) containing a set of regular expressions was used to tokenize the text so that word boundaries could be easily calculated. Although the results are not pub-

lished here, including punctuation improved the performance by several percent in all n-gram based versions of the system. Additionally all strings of digits were replaced by a single token and all characters repeated more than three times were replaced by a single instance.

Version 2: One of the reasons n-grams are a natural choice for feature representation in this domain, and are hard to beat, is because they are able to capture the characteristics of the genre that are apparent in many different lexical tokens. For example, phrases like “and then”, “while”, and “during” as well as the existence of personal pronouns such as “I” are highly correlated with story text. However, only accounting for existence misses important information about story text. It is not unreasonable to think that many segments of non story text also contain these words also. The difference lies in the increased relative frequency that occurs in story text. *Version 2* uses the same features as version 1 but in this case the value of each feature is based on the frequency of occurrence in the sentence. To prevent high valued features from dominating the learning algorithm the frequencies were normalized to the sentence length so the values always range from 0 to 1. Additionally it has been empirically shown that distributing the values on a logarithmic scale can also improve performance.

Versions 3 & 4: Using an n-gram encoding on the lexical items poses several significant problems with the data. One of the problems is that the number of features grows exponentially with respect to n. If the vocabulary of the language is V the number of possible n-grams is $|V|^n$. Although many high value features occur between spans of text that have many words in between, typically values of n larger than 3 become impractical because the number of features makes training the model intractable and even when it is possible it often leads to over-fitting the data. Even if these issues are overcome the frequency statistics of such large n-grams are likely to be unreliable except in cases where huge amounts of training data are available, since the vast majority of (n>2)-grams are only seen a handful of times, if at all. One method for alleviating this problem is by excluding all n-grams occurring fewer than a specified number of times. For this work a frequency threshold of two

words was used. This does help to reduce the amount of time to train and test, but is only one small partial solution to finding high value n-gram features. Consider the sentence “Kent went bass fishing everyday on our trip.” This sentence poses two similar problems. One likely potential high value feature of this sentence is the bigram “Kent went”. However, despite people going places frequently this information will be lost because Kent is unlikely to occur again diluting the frequency of “going” events. Similarly “bass fishing” is probably fairly common, but in web data so are misspellings, like the one above, and so once again the statistics of the events we are interested in are watered down. The approach taken by versions 3 & 4 to address these issues is to replace the words by their corresponding part-of-speech. This effectively reduces the vocabulary from tens of thousands to only a few dozen. Although we lose the lexical information we are able to generalize more effectively and obtain better statistics. It also reduces the vocabulary so much it affords the possibility having reliable statistics for even longer n-grams. In version 3 we used the MXPOST part-of-speech toolkit [16] and extracted n-grams of length 4. *Version 4* combined the lexical n-grams as in *version 2* and the part-of-speech n-grams in *version 3*.

Version 5: There are many approaches to improving the generality of the features, reducing data sparsity and increasing the context in which decisions are made, of which using part of speech tags are one option. More recently the use high-level syntactic features has shown great promise in a variety of NLP applications through the use of tree kernels incorporated into SVM machine learning methods [12]. The basic idea behind tree kernels is to provide a measure of similarity between two trees by comparing the number of matching sub-trees. This value can replace the result of the traditional kernel function and be directly incorporated in the same SVM classifier. There are two primary approaches; the SubTree kernel and the SubSet kernel. The SubTree kernel generates a feature for every sub-tree in the original parse, where a sub-tree is a tree rooted at a non terminal node and whose leaves are terminals of the original parse. The SubSet kernel is also a sub-tree in the original parse but the leaves may be non-terminals.

Version	Sentence-level performance						Story-level performance					
	precision		recall		F-score		precision		recall		F-score	
	raw	smooth	raw	smooth	raw	smooth	raw	smooth	raw	smooth	raw	smooth
1. n-grams	.267	.241	.263	.876	.242	.352	.039	.211	.139	.911	.054	.329
2. n-grams w/frequency	.465	.464	.329	.606	.371	.509	.150	.298	.308	.683	.176	.399
3. POS n-grams	.460	.406	.314	.703	.348	.496	.134	.275	.363	.730	.183	.367
4. n-grams+POS w/frequency	.466	.497	.329	.455	.371	.463	.130	.267	.304	.594	.171	.347
5. tree kernels	.234		.319		.246		.044		.180		.065	
6. GG05		.302		.829		.414		.215		.654		.287

Table 1: Comparative performance of story extraction approaches (with best performance in boldface)

For our experiments we used the SubSet kernel because it offers a more general feature set than the SubTree kernel.

2.4 Smoothing

Although SVM classification usually out performs Naïve Bayes, SVM suffers from the same sequence labeling problem. Often the language of a particular sentence, or even a small window of sentences, is not enough to determine whether it is a story sentence or not. For example, sentences that bridge two events together may be an opinion or statement of fact that by itself is not story-like at all, but is in fact essential to the continuity of the story as a whole. In this section I will describe a new smoothing technique that attempts to improve upon the mean-average method used in the previous version. It should be noted that this definition of smoothing is not what is usually meant in NLP. Smoothing in NLP typically refers to the redistribution of probability mass from events that are seen to events that are never seen. This is extremely important for models that multiply probabilities together, such as n-grams, to prevent unlikely, but possible, events from never occurring. Although what is described here is somewhat analogous, since confidence mass is being redistributed, it is not exactly the same thing.

The benefit of the mean-value function is that it will give more confidence to sentences that occur near story text and give less confidence to those that do not. However, it does not have two properties that we would also like. First is that it gives equal weight to all sentences surrounding the target sentence even though more distant sentences probably have less influence than close ones do.

Second, in our development set we noticed that the classification tends to underestimate the confidence of story text across the board. To address these deficiencies in the mean-average function a Gaussian and Laplacian of the Gaussian (LoG) were tried instead. These methods are commonly used in the image processing community for edge detection and the shapes of these curves can be manipulated to control the confidence mass each surrounding sentence as well as amplifying (or dampening) the confidence levels universally.

2.5 Results

The results are summarized in Table 1. The performance of the feature sets are evaluated at both the sentence level and at the story level. At the sentence level precision, recall and F-score are calculated as one would expect. Precision is the number of correctly labeled story sentences over the total number of sentences the system labeled as story. Recall is the number of correctly labeled sentences over the total number of story sentences possible. F-score is the equally weighted harmonic mean between precision and recall. The story level metrics are slightly more complicated. Story precision is calculated by computing the number of correct sentences of story predicted to the total story sentences predicted. In the case that a predicted sequence of sentences overlaps with more than one actual story the results are averaged. Conversely story recall is the amount of overlap between the correct predictions in a predicted sequence versus the total amount of story possible per actual story sequence. The F-score is again the harmonic mean between the two.

Although both Gaussian and LoG smoothing were tried on the data set, the Gaussian function consistently performed 2-3% above LoG and so only the Gaussian results are presented in the table. Feature set 2 is generally the best overall system for both sentence and story level F-score. There is some variability if one prefers either precision or recall however. Although the results are not completely comparable the new system also outperforms the old system (GG05) both at the story and sentence level. Unfortunately due to problems with the tree kernel software we were unable to get complete results. Even though the partial results were not competitive with the best feature sets there does still seem to be value in pursuing these approaches further.

3 Semantic Role Labeling

Discourse is full of states, actions and events involving relationships between people, things and other entities. Although flat features, like n-grams, are often convenient because of their high performance at low computational and complexity cost, they offer very little help in discovering these relationships in text. However, these relationships are essential components for any deep natural language understanding. This is especially true for the Story Upgrade system in which the temporal sequence of events and the roles of the actors in the events are absolutely critical in a correct interpretation of the story. For example, perhaps you are interested in stories about wine tasting in Napa Valley. A couple of key elements in this story archetype include a person tasting wine and a person visiting a winery. Relying only on n-grams, documents containing the word wine will most likely trigger a high likelihood that a document is relevant to your search. However, this approach will lead to many highly ranked irrelevant text passages because many other documents such as wine retailers and restaurants also have a high correlation with this word but do not share much of the same semantic content.

Even though the tree kernel approach did not work as well as expected for story classification, using syntactic information provides another level of structure that contains potentially useful information not available at the lexical level. Syntactic in-

formation is also one method for obtaining even deeper semantic information about the meaning of the text. One type of semantic information particularly useful is semantic roles. These types of relationships explicitly describe what roles the participants play in the discourse. Loosely speaking they can identify the subject and objects of a particular verb. Often these relationships are implicit in the syntax and can be made explicit through transformation, as is done here. This type of information is also extremely difficult to capture at the word level because these relationships are not always local and may be of arbitrary length. A highly reliable system that can identify these semantic roles could enable a host of new approaches for natural language processing applications and will later be exploited for improvements in the Story Upgrade system. In this section of the paper I will summarize work done with Andrew Gordon [6] in generalizing semantic role annotations using the syntactic similarity between verbs to improve the performance when little or no training data for a verb is available.

The task of semantic role labeling is to assign all the thematic roles for a target verb in a sentence. PropBank [14] is one corpus that provides over 100,000 parse trees that have been annotated with semantic role information. The following is an example for the verb give in the style of a PropBank annotation:

[ARGO John][give.01 gave][ARG1 a book][ARG2 to Mary][ARGM in the morning]

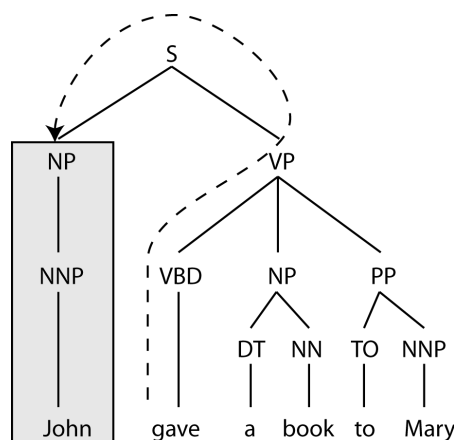


Figure 1: An example parse tree path from the predicate *gave* to the argument NP *John*, represented as \uparrow VBD \uparrow VP \uparrow S \downarrow NP

For each word sense a verb PropBank defines a role set that specifies the semantic frame for that verb's sense. In this example the target verb *gave* is an instance of the roleset *give.01* meaning it is the first sense of the verb *give*. Here *John* is the “giver”, *Mary* is the “entity given to” and *the book* is the “thing given”. To try to remain as theory neutral as possible PropBank opts to use generic terms ARG0, ARG1, ARGN for labeling the arguments. For each roleset there is an associated XML entry that describes in more detail what is meant by each argument label. In PropBank there are two primary types of arguments; numbered arguments, as above, which tend to be the core required arguments and argument modifiers denoted by ARGM. Previous work [13][15] has used this corpus to create high performance automatic semantic role labeling systems. Although these systems achieve high levels of precision and recall on this corpus, they suffer from problems typical of machine learning approaches. One of the big problems is the availability of training data that is usually a limiting factor, as well as the ability to extend to novel contexts that have never been seen before. For example, there are 617 examples for the verb *want*, 6 for *desire* and 0 for *yearn*. For similar reasons these approaches often sharply degrade in performance on out of domain texts. One can alleviate these issues by annotating more training data, although this will invariably become prohibitively expensive because there will always be unseen verbs, phrases, contexts and out of domain documents that will prevent a system from reaching optimum performance. A more robust approach would be to generalize the behavior across verbs by leveraging the inherent syntactic similarity between verb classes. Although the verb *yearn* does not appear in the corpus, it is very likely a verb that behaves syntactically similar to it does appear. Our approach uses this idea, a form of the distributional hypothesis, to generalize the process of assigning semantic role information by using the training data from syntactically similar verbs. When labeling a target verb, such as *yearn*, instead of using the training data available for that roleset, of which there might not be any, we use surrogate training data from one or more syntactically similar verbs. The first step in our approach was to find and create an ordered list of the most syntactically

similar verbs for each roleset. Second we aligned the role labels between the target roleset and each syntactically similar verb. Finally we evaluated the results using a simple semantic role-labeling algorithm that uses parse tree path frequencies to assign roles to nodes in the tree.

3.1 Parse Tree Paths

Parse tree paths are essential to both our algorithm and our determination of syntactically similar verbs. Parse tree paths were first used for SRL by Gildea and Jurafsky [2] as one syntactic feature to capture the relationship between a predicate and its argument in the parse tree. This relationship is represented through a sequence of up transitions from the predicate and at least one down transition to a novel branch in the tree. In this case the predicate is equivalent to the target roleset, e.g. *give.01*, and the argument of this roleset is a span of words in the sentence. The span of words can equally be represented by a node in the parse tree that dominates this and only this span of words. Figure 1 gives an example parse tree with its path highlighted visually as well as a textual sequence of up and down transitions. Parse tree paths are an effective encoding because of their ability to generalize well across syntactically similar sentences regardless of the underlying lexical items. For example many different argument spans of text such as “John” or “The man standing behind the door” can be recognized by only a single parse tree path.

3.2 Syntactic Similarity

Another key part of our method for generalizing semantic role information is being able to identify and quantify the similarity between two verbs. We hypothesize that verbs appearing in similar contexts will behave similarly with regard to their semantic role assignments. If this is true then the amount of data needed to label accurately the arguments of any arbitrary verb become significantly reduced. In this section I will describe our approach for computing the syntactic similarity between verbs using parse tree paths as an approximation for syntactic context.

Our approach is similar to Lin's work [10] of extracting collocations using dependency path information. To apply this technique to our work it was necessary to utilize a much larger corpus and one that was distinct from PropBank, but still in a similar domain. The Gigaword corpus [11] is a collection of newswire text from four major sources totaling over one billion words of textual data and was used in our experiments for the purpose of determining syntactic similarity. We pre-processed this data by extracting the body text from news stories and applied MXTERMINATOR to identify sentence boundaries.

After preprocessing the data we computed the similarity as follows. For each of the 3,257 verbs in PropBank we identified all the sentences in Gigaword that contained an inflection of that verb. Charniak's August 2005 parser [1] was then used to automatically parse these sentences. Unfortunately we did not have access to the computational resources to parse all the sentences. Instead we focused on parsing only the first 100 sentences for

<i>Verb pairs (instances)</i>		<i>Cosine</i>
bind (83)	bound (95)	0.950
plunge (94)	tumble (87)	0.888
dive (36)	plunge (94)	0.867
dive (36)	tumble (87)	0.866
jump (79)	tumble (87)	0.865
fall (84)	fell (102)	0.859
intersperse (99)	perch (81)	0.859
assail (100)	chide (98)	0.859
dip (81)	fell (102)	0.858
buffet (72)	embroil (100)	0.856
embroil (100)	lock (73)	0.856
embroil (100)	superimpose (100)	0.856
fell (102)	jump (79)	0.855
fell (102)	tumble (87)	0.855
embroil (100)	whipsaw (63)	0.850
pluck (100)	whisk (99)	0.849
acquit (100)	hospitalize (99)	0.849
disincline (70)	obligate (94)	0.848
jump (79)	plunge (94)	0.848
dive (36)	jump (79)	0.847
assail (100)	lambaste (100)	0.847
festoon (98)	strew (100)	0.846
mar (78)	whipsaw (63)	0.846
pluck (100)	whipsaw (63)	0.846
ensconce (101)	whipsaw (63)	0.845

Table 2. Top 25 most syntactically similar pairs of the 3257 verbs in PropBank. Each verb is listed with the number of inflection instances used to calculate the cosine measurement.

each verb. In total 324,461 sentences with an average of 99.6 sentences per verb were parsed. To facilitate the similarity computation we adopted a feature vector representation of the syntactic context where each component of the vector represented the relative frequency of a particular parse tree path. For each sentence we extracted all possible parse tree paths for the target verb and maintained a count of each path over all the sentences in the set. For example all the possible parse tree paths for the tree in Figure 1 are as follows:

1. $\uparrow\text{VBD}\uparrow\text{VP}\uparrow\text{S}\downarrow\text{NP}$
2. $\uparrow\text{VBD}\uparrow\text{VP}\uparrow\text{S}\downarrow\text{NP}\downarrow\text{NNP}$
3. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{NP}$
4. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{NP}\downarrow\text{DT}$
5. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{NP}\downarrow\text{NN}$
6. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{PP}$
7. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{PP}\downarrow\text{TO}$
8. $\uparrow\text{VBD}\uparrow\text{VP}\downarrow\text{PP}\downarrow\text{NNP}$

To quantify the similarity between verbs we used the cosine measure, a common vector based distance metric. Although we also tried several other popular methods such as Chi-square, Euclidean and Manhattan distance cosine was the least sensitive to variations in sample size between the verbs. To facilitate the evaluation we computed the pairwise similarity between each pair of verbs in PropBank producing 3,257 ranked lists of 3,257 verbs. Table 2 lists the top 25 most syntactically similar pairs of verbs seen in PropBank.

3.3 A Simple SRL Algorithm

We developed a simple algorithm for assigning semantic roles to nodes in a parse tree based on the frequency of parse tree paths seen in the training data. A labeler was constructed for a particular roleset by counting the number of parse tree paths between the verb and its arguments over all the training data for that roleset. For example Table 3 shows the top 10 most frequent parse tree paths and corresponding argument label for the roleset *want.01*. In order to assign labels to an un-annotated parse tree we traverse the table in the order of highest frequency. We look to see if the parse tree path of the current table entry is found in the tree; if it is then we assign the argument label to the node in the tree at the end of the path. Additionally we invalidate all entries in the table with the same

argument label (e.g. ARG0) and any entry that has a parse tree path that is a sub-path of the one just found. We then consider the next valid argument in the list and repeat the process until there are no more valid entries in the table. One exception to this process is dealing with the modifying ARGM argument types. Although numbered arguments (e.g. ARG0, ARG1 ... ARGN) are generally considered essential and unique many ARGM arguments are possible for a rolest. In the above algorithm only numbered arguments are invalidated allowing for multiple ARGM assignments.

3.4 Alignment

Our approach involves using data from surrogate rolesets that each have their own role definitions. There is no guarantee that the role labels in the surrogate training data correspond to the same labels as the target rolest. In order to ensure that the training data from the surrogate data is consistent with the target rolest we used the following alignment algorithm to find a mapping between argument labels of the target and surrogate rolesets. For example, if for some reason we do not have a lot of training data for the rolest *taking* in our corpus, but we do have a sufficient amount for *give* then we would like to know that the “taker” corresponds to the “giver”, the “entity taken from” corresponds to the “entity given to” and “the thing taken” corresponds to “the thing given”.

In order to simplify this process several assumptions are made. First, we only consider rolesets that contain the same number of core arguments candidates for alignment (and also for surrogate data). Second our approach requires that at least one fully annotated parse tree be available for the target role-

Count	Argument	Parse tree path
189	ARG0	↑VBP↑VP↑S↓NP
159	ARG1	↑VBP↑VP↓S
125	ARG0	↑VBZ↑VP↑S↓NP
110	ARG1	↑VBZ↑VP↓S
102	ARG0	↑VB↑VP↑VP↑S↓NP
98	ARG1	↑VB↑VP↓S
96	ARG0	↑VBD↑VP↑S↓NP
79	ARGM	↑VB↑VP↑VP↓RB
76	ARG1	↑VBD↑VP↓S
43	ARG1	↑VBP↑VP↓NP

Table 3: Top 10 most frequent parse tree paths for arguments of the PropBank want.01 rolest, based on 617 annotations

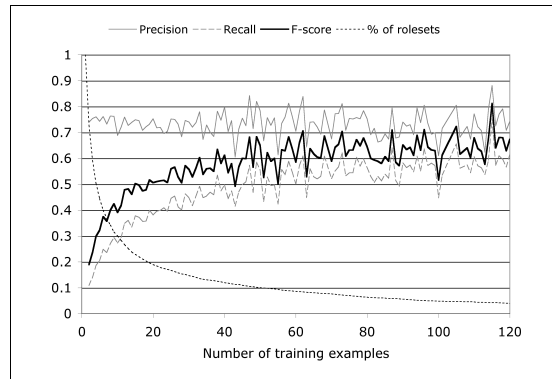


Figure 2. Performance of our semantic role labeling approach on PropBank rolests

set so that a complete mapping can be made. A fully annotated parse tree is one in which all core arguments have actually been assigned. For example the rolest for *give.01* states that ARG0 is the giver, ARG1 is the “entity given to” and ARG2 is “the thing given”. The example in Figure 1 represents a fully annotated piece of data for this rolest because all three of the core arguments are assigned. It is possible however that some annotated sentences do not have all the core arguments given in the rolest definition. For example:

[ARG0 John][give.01 gave][ARG1 the book][ARGM away] is missing one of the core arguments and would not be considered a fully annotated sentence.

The alignment process for a particular rolest begins with its ordered list of syntactically similar verbs. We apply two steps to augment this list with the information necessary to allow the use of arbitrary amounts of surrogate training data for our se-

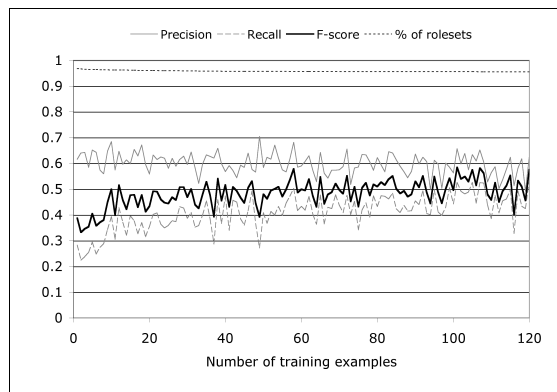


Figure 3. Performance of our semantic role labeling approach on PropBank rolests using various amounts of surrogate training data

semantic role-labeling algorithm. The list of syntactically similar verbs contains all the verbs in PropBank, however to satisfy our first simplification constraint we must expand the verbs to their corresponding rolesets and then filter any roleset that does not have the required number of arguments. The second step involves finding the mapping between arguments of the target roleset and the surrogate roleset. We do this by building a role labeler, as described above, for each surrogate roleset in the list using whatever data is available for that roleset. This labeler is then applied to the one fully annotated example sentence for the target roleset as if it were a test example of the target roleset. Then we compare the two labeling instances and note if there is an overlap in the assignments. If a constituent is annotated in the example for the target and the same constituent is annotated by the surrogate labeler then the mappings are recorded along with the other information in the ordered list. The roleset is removed from the list if a mapping is not found between the target and surrogate roleset. After these steps are performed for each element in the similarity list we obtain an ordered list for each verb in PropBank, with at least one fully annotated training example, that is similar to the following:

- | | |
|-------------------|---------|
| 1. orchestrate.01 | 1:1 |
| 2. chase.01 | 0:0 1:1 |
| 3. unearth.01 | 0:0 1:1 |
| 4. snub.01 | 1:1 |
| 5. erect.01 | 0:0 1:1 |

This list allows us to use as much training data as we would like by simply conglomerating the training data available for as many rolesets in the list as we would like. Although to test this approach we limit ourselves to the rolesets given in PropBank, however, it would work equally well for any novel verb provided it was given one fully annotated example.

3.5 Results

We postulated that the surrogate data could improve the results for the rolesets with little or no training data. We performed two major experiments, using PropBank for our training, test and source of surrogate data, to evaluate the results of our hypothesis. First we evaluated how well our simple semantic role-labeling algorithm could per-

form with varying amounts of “real” training data. To do this we collected the frequencies of parse tree paths using incrementally more training data. We started by using just one example to train and tested on the remaining data, then used two examples until all but one example was left for testing. Figure 2 shows the results of this experiment. The graph indicates learning curves for the precision, recall and F-score as a function of how many training examples the role labeler was trained on. Precision is defined as the number of correct assignments divided by the number of assignments the system returned. Recall is the number of correct assignment out of the number of possible correct assignment. The F-score is defined as the equally weighted harmonic mean of the precision and recall. Precision, recall and F-score are the predominant lines in the graph. Precision is fairly constant, although dips slightly, as more training data is introduced. Recall dramatically increases for small amounts of training data but plateaus around 120 training examples. The F-score is sandwiched between precision and recall. Also shown in the graph is the percentage of rolesets that have the number of training examples given on the x-axis. So, despite the fact our system is capable of achieving an F-score near 0.7, it is only able to do so for less than 5% of the rolesets.

Second we ran an analogous set of experiments using surrogate training data instead. As described previously we built a table for each fully annotated roleset that contained a ranked list of similar verbs along with an argument mapping. Unfortunately only 2,858 of the 4250 rolesets in PropBank have one fully annotated example and only 2,807 of these were seen at least once in the Gigaword corpus. We therefore used this set of 2,807 rolesets for our test data. For each of these rolesets various amounts of surrogate training data was used by incrementally descending the list of similar verbs and building a role labeler as described in section 3.3 using all the available surrogate data for that verb. It is important to note that the training data for the target roleset was not included in these experiments. This artificially decreases the overall performance of a “real world” semantic role labeler utilizing this approach but gives us a better idea of the contribution of the surrogate data.

The results of this experiment are displayed in Figure 3. There are two interesting things to note in relation to the performance using the real data. First, the graph is much flatter. The level of performance starts much higher and does not increase as rapidly. However, it also does not reach the same peak in performance and is consistently 10-15% lower than “real” data once the plateau is reached. Second and more importantly the curve showing the percent of rolesets with the available data now encompasses nearly all the rolesets considered. In this case 95% of the rolesets can reach an F-score between 0.5 and 0.6, where as using real data only 10%-20% have enough data.

4 Story Upgrade Version 2

Story Upgrade is a system intended to enable the retrieval of highly relevant stories in order to facilitate and enhance content creation and scenario development for training simulations and games. Stories that portray a convincing account of real world possibilities and present realistic decisions are an integral part of any training simulation that hopes to have a significant impact. Over the past several years the Institute for Creative Technologies has developed several large-scale training simulations that have been instrumental in teaching and fostering leadership skills. Although these projects have been successful in producing highly compelling environments it takes a lot of time to author the scenarios from scratch and is extremely expensive. The extensive time it takes also inhibits training simulations tailored to current events that could more adequately address the immediate needs of an organization. In this section I will describe the initial stages of an approach I am developing. It aims to address these issues by providing an easy to use search interface and provides relevant, real world stories based on the author's search criteria.

The search interface is designed as a web utility that provides input for the user to give a story title along with a brief example story. The idea is that the user, for example the content author, can think of a simple boring story about the desired type of scenario but would like to fill out the details and get new ideas from real people's actual experiences. This is essentially a three step process that

will be more fully described in the following sections. First the user inputs his or her search criteria, the boring story, and a standard web search is done to retrieve Weblogs likely to have relevant stories on them. Second the story text is extracted from each Weblog and third the stories are ranked by relevance and returned to the user.

4.1 Web Search

One possible approach to finding candidate Weblogs is to first crawl the web, download the data locally and index it with a standard search engine toolkit. This is the approach taken by the earlier version of Story Upgrade, although it poses several management problems. For example it requires active crawling and maintenance of the data in order to have up to date Weblog entries. This requires a large amount of hardware and takes a long time. In this version of Story Upgrade a different strategy was employed. Each search request by the user was first preprocessed by sentence delimiting the boring story. For each sentence a query was made via the Google Blog Search website using the words of the sentence as keywords. These searches were used to collect a set of candidate documents that were broadly related to the topic at hand. Each of the candidate documents was then preprocessed and the stories extracted as described in section 2.

To try to improve the quality of the initial set of candidate documents semantic role labeling was incorporated in the following way. Each sentence in the original user query was labeled with its semantic roles and only words contained within a bracketed label were sent to Google Blog Search as a key word. This essentially used the semantic role labels as a filter for finding highly relevant words with the hope of restricting the set of returned documents to even more relevant topics.

4.2 Ranking

The second step is essentially unchanged. Each returned Weblog was preprocessed and the story extracted in the manner described in section x. Unlike the previous version however, the stories were ranked using a feature vector representation and the cosine distance metric. Analogous to the representation in the story extraction section the document is most simply represented as a bag of words.

Term frequency-inverse document frequency (TFIDF) is a common, but effective, way of weighting the components of the document vector. TFIDF is a method for computing how important a term is in a document. The intuition is that frequently occurring words, the term frequency, should be given additional weight. However, words that occur in many documents, the inverse document frequency, are unlikely to provide any discriminative power and should counter balance the term frequency. This leads to a common formula for computing the TFIDF value:

$$tf_i = \frac{n_i}{\sum_k n_k}$$

$$idf_i = \log \frac{|D|}{|\{d : t_i \in d\}|}$$

$$tfidf = tf \cdot idf$$

Where n_i is the number of times term i was seen in the document. $|D|$ is the total number of documents and $|\{d : t_i \in d\}|$ is the number of documents term i is found. Again, much like the story extraction work it is easy to extend this representation to include larger n -gram sequences. In this work unigrams, bigrams and trigrams were all included in the vector representation.

This approach also offered an opportunity to apply semantic role labeling to try to improve the search rankings. Instead of using generic n -grams a different feature vector representation was also tried. Each sentence of both the original query and of the candidate documents were sentence delimited and labeled with their semantic roles. N -grams were then extracted as features like before, however each n -gram was also encoded with its semantic role and was excluded if it was not labeled.

5 Discussion

Each component of the original Story Upgrade system has been improved and several new components have been added to improve the overall system's quality. Unfortunately quantitative evaluations of the new system have not yet been performed and so no solid conclusions can be drawn. Some tentative conclusions are none the less still possible. The advantages of using Google Blog Search versus indexing a large section of the web

locally seem to outweigh the disadvantages. Although using this approach increases the time to return a search result, less hardware resources are needed to maintain the system. Additionally this approach always returns the most up to date results, which can be very advantageous in designing pertinent training simulations. Unfortunately the resources were not available to put together a comprehensive evaluation corpus to test the relevancy rankings of the system so it is impossible to say for sure which feature sets and methods performed the best. Qualitatively the best rankings seemed to come from simple n -grams (up to trigrams), while using the SRL features seemed to have little or slightly negative effects. Unfortunately none of the feature sets tried seemed to return a high quality set of rankings. Although some relevant documents do appear near the top, many of the top documents are highly irrelevant. For example, searching for stories about a road trip to Santa Barbara returns many documents about Santa Barbara, driving, and cars. Some of these are highly relevant but many of the documents are about Santa Barbara politics, car shows, and other completely irrelevant topics. This is not surprising because n -grams can only capture an extremely small context representing an even smaller semantic component of an entire document. Additionally there is no guarantee that what we as humans consider semantically relevant words have the highest TFIDF values. Proper names, for example, will receive extremely high values because they are unlikely to occur in many documents. Using the semantic role labels was one attempt to try to filter out some of these irrelevant, yet "high value", words but without much success. Although a more thorough analysis is required it seems that an even deeper representation capable of modeling some form of discourse analysis will be necessary to truly discriminate based on the semantics of a document.

There are several directions for future research in this area. First and foremost before any more work is done on story search an evaluation corpus needs to be developed. Once this is done a closer look at the effects of high level features on ranking relevance such as syntactic structure and semantic roles is warranted. There are also several aspects to story extraction that are worth investigating. Despite the availability of a relatively large training

corpus, its quality is questionable. Although the average Kappa was sufficiently high at 0.68, the individual scores of 0.66, 0.80, 0.85, 0.72, and 0.40 indicate a high standard deviation (0.176) casting some doubt on the robustness of the corpus. It would therefore be beneficial to apply a more rigorous development process to create a more consistent data set. Along with a more robust data set it would be interesting to see if a richer discourse analysis could be learned instead of a simple binary story/non-story classification. Finally with the availability of a large scale database of stories it would also be interesting to apply unsupervised learning techniques to try to induce this story structure or to learn the common features between story types.

References

- [1] Charniak, E. 2000. A maximum-entropy-inspired parser, Proceedings NAACL-ANLP, Seattle.
- [2] Gildea, D. and Jurafsky, D. 2002. Automatic Labeling of Semantic Roles. *Computational Linguistics* 28:3, 245-288.
- [3] Gordon, A. 2004. Authoring branching storylines for training applications. Proceedings of the sixth international conference of the learning sciences (ICLS-04), Santa Monica, CA.
- [4] Gordon, A., Cao, Q., & Swanson, R. (2007) Automated Story Capture From Internet Weblogs. Proceedings of the Fourth International Conference on Knowledge Capture, October 28-31, 2007, Whistler, BC.
- [5] Gordon, A. & Ganesan, K. 2005. Automated story capture from conversational speech. Third international conference on knowledge capture (KCAP-05), Banff, Canada.
- [6] Gordon, A. & Swanson, R. (2007) Generalizing semantic role annotations across syntactically similar verbs. Proceedings of the 2007 meeting of the Association for Computational Linguistics (ACL-07), Prague, Czech Republic, June 23-30, 2007.
- [7] Hsu, C., Chang, C. & Lin, C. 2003. A practical guide to support vector classification. Available online at <http://www.csie.ntu.edu.tw/~cjlin/libsvm/>
- [8] Joachims, T. 1999. Making large scale SVM learning practical. In Schölkopf, Burges, & Smola (eds) *Advances in Kernel Methods - Support Vector Learning*. Cambridge, MA: MIT Press.
- [9] Johnson, C., Birnbaum, L., Bareiss, R., & Hinrichs, T. 2000. War stories: Harnessing organizational memories to support task performance. *Intelligence* 11(1):16-31.
- [10] Lin, D. 1998. Automatic Retrieval and Clustering of Similar Words. COLING-ACL, Montreal.
- [11] Linguistic Data Consortium. 2003. English Gigaword. Catalog number LDC2003T05. Available from LDC at <http://www ldc.upenn.edu>.
- [12] Moschitti, A. 2006. Making tree kernels practical for natural language learning. In Proceedings of the Eleventh International Conference on European Association for Computational Linguistics.
- [13] Moschitti, A., Pighin, D. and Basili, R. 2006. Semantic Role Labeling via Tree Kernel joint inference. Proceedings of CoNLL, New York.
- [14] Palmer, M., Gildea, D., and Kingsbury, P. 2005. The Proposition Bank: An Annotated Corpus of Semantic Roles. *Computational Linguistics* 31(1):71-106.
- [15] Pradhan, S., Ward, W., Hacioglu, K., Martin, J., and Jurafsky, D. 2005. Semantic role labeling using different syntactic views. Proceedings ACL-2005, Ann Arbor, MI.
- [16] Ratnaparkhi, A. 1996. A maximum entropy model for part-of-speech tagging. Proceedings EMNLP, University of Pennsylvania.
- [17] Reynar, J. & Ratnaparkhi, A. 1997. A maximum entropy approach to identifying sentence boundaries. Proceedings of ANLP, Washington, D.C.
- [18] Schank, R. & Abelson, R. 1995. Knowledge and Memory: The Real Story. In R. Wyer (ed) *Knowledge and Memory: The Real Story*. Mahwah, NJ: Lawrence Erlbaum Associates, pp. 1-85.