Replicating "Extracting Action and Event Semantics from Web Text"

Louis Carlin

Abstract

An important challenge in AI research is to build models which can encapsulate the behaviour of agents in a dynamically changing world. Events and actions are a formal framework which can be used to construct such models. However it is not feasible to construct large, sufficiently detailed collections of events and actions by hand. Thus the challenge is to build systems to automatically generate them. An approach to this problem via textual event extraction is presented in Sil et al. (2010). In this thesis I outline my attempt to replicate it and give a more thorough exposition of the proposed system. I was not able to achieve the reported level of performance and discuss some potential reasons.

1 Introduction

In order for an intelligent agent to make decisions it needs to be able to model its environment. Such a model must describe the effects of its actions (and possibly the actions of other agents or other events that may occur). Furthermore, it must be possible to use this model to *plan* by reasoning about sequences of possible actions and evaluating the ensuing state of the world.

Planning is a well studied problem and many tasks have been framed as planning problems to great effect (see 1.2). Posing a planning problem requires information about the state of the environment and dynamics of that state. One limit to the application of planning algorithms to general tasks is the "knowledge acquisition bottleneck" (Sil et al., 2010): it is expensive or even infeasible for humans to write models of the requisite complexity. To address this gap, attempts have been made to automatically generate this data. The obvious approach is to extract it from text; humanity's largest repository of knowledge.

Sil et al. (2010) describe a novel approach to solving this problem. Their system, known as 'PrePost', uses a combination of statistical and structural information — extracted from automatically downloaded web documents — to build simple representations of events and actions. The PrePost system is expanded upon in Sil and Yates (2011) to extract a more powerful representation and address problems the authors identify with word sense disambiguation and generalisation to hypernyms. Both papers leave a number of expositionary gaps and implementations of the systems are not publically available.

The purpose of this thesis is twofold: I aim to reproduce the results of the PrePost system as well as provide a more thorough exposition of the details of its implementation. In 1 I outline the problem, give a motivating example, and briefly discuss other approaches. I describe my work implementing the PrePost system in 2, including explanation of some background techniques as well as difficulties I faced. Finally in 3 I evaluate the performance of my implementation and discuss some potential limiting factors.

1.1 Events and Actions

Sil et al. (2010) give an informal definition of events as "observable phenomena that happen at a particular time and place". It is helpful to, in addition, think of events as always changing the state of the world in some way. Actions are simply events which are brought about by (rational) agents.

There are different ways of formally representing events. One representation popular in automated planning is STRIPS (originally used in the Stanford Research Institute Problem Solver). A STRIPS representation of an action consists of the arguments it takes, boolean predicates on those arguments that must be true before the action is taken (preconditions) and boolean predicates which are made true (add effects) or made false (delete effects) after the action has occured. Automated planners sequence STRIPS actions from a start state to a goal state by heuristic graph search.

PrePost relies on a simpler representation of events. Pre- and post-conditions are represented as single words and don't take arguments, and there is no distinction between add effects and delete effects. Thus they are less expressive than the predicative pre- and post-conditions of STRIPS. The authors build upon PrePost in Sil and Yates (2011) to extract full STRIPS representations.

		`awaken`	`insert`
STRIPS	**args:**	x	o, p
	pre:	$asleep(x)$	$object\#1(o)$, $opening\#\ 1(p)$, $\neg\ \text{in}(o, p)$
	add:	$awake(x)$	$\text{in}(o, p)$
	del:	$asleep(x)$	$\neg\text{in}(o, p)$
PrePost	**pre:**	*asleep*	*person, slot*
	post:	*awake*	in

Figure 1: Example STRIPS and PrePost representations of actions "awaken" and "insert". From Sil and Yates (2011).

1.2 Application to Narrative Generation

One planning problem that would greatly benefit from larger action datasets is narrative generation. Narrative generation is the task of generating fictional plots (fabula) and is not concerned with the related task of representing those plots in prose (syuzhet). We frame it as a planning problem as follows:

- Build a collection of characters, locations, and objects
- Specify ways in which these characters may interact with the world (ie actions)
- Choose a starting state and goal state

A planner then generates a story by sequencing together character actions from the starting state to the goal.

By itself this approach generates plots that are usually not very coherent or compelling. Long sequences of legal events may make sense to a planner but be incomprehensible to a human reader. Furthermore there is no reason for us to expect them to be "interesting" in the same way a well-written story is.

Riedl and Young (2010) address the problem of coherence by adding the constraint of *intentionality*. Under their framework any action a character takes must further that character's intentions. An intention is a model literal of the form (**intends** A f), where A is a character and f is a fact. Intentions are dynamic in the sense that they may arise as effects of actions. For example, a character **?woman** being betrayed by another character **?man** could establish the intention (**intends ?woman** (**dead ?man**)). Intentionality creates coherence by requiring characters' actions to be internally consistent. Haslum (2012) demonstrated that Reidl and Young's narrative planning problem can be compiled as a classical planning problem. This allows it to be solved much more efficiently using existing, highly optimised classical planners.

While the example Riedl and Young (2010) give consists of a small cast of 5 characters and a handfull of possible actions, human novels typically feature hundreds of characters and thousands of different interactions. Often actions that may seem inconsequential can play a central role in a plot. For example imagine a murder suspect signing something with his left hand being the clue a detective needs to identify him as the killer. An open-domain narrative generation system would need a collection of millions of actions to be able to model complex interactions like this in the same way human authors do. The only feasible way to compile a dataset this large is through automatic extraction.

1.3 Other Work

Most approaches to event extraction derive a temporal ordering of events but do not seek to model the underlying state of the world that allows for and arises from these events. Chambers and Jurafsky (2008) extract "narrative schemas", or sets of events that occur together. In its sequel (Chambers and Jurafsky, 2009) they extend this to also extract participant roles. The *event2event* system

(Martin et al., 2018) uses a recurrent encoder-decoder neural network to extract sequences of events from a corpus of Wikipedia movie plot summaries. Knowlywood (Tandon et al., 2015) is a knowledge base of human activities extracted from Hollywood scripts, including attribute data (participating agents, location, time of day) as well as temporal links to typical previous and next activities. Manikonda et al. present a naive approach to event extraction which assumes events occur in the order they are described.

These approaches all primarily extract *event–event* relationships rather than *event–state* relationships. Information about typical event sequences can be used for narrative generation (see for example Martin et al. (2018)), however it does not lend itself to classical planning. Without explicit pre- and post-conditions it is not possible to decide whether an action can be taken at a particular step in a plan and what the effects of that action are. Thus the problem PrePost and its successor attempt to solve is relatively unexplored.

2 PrePost

The PrePost system aims to extract events by identifying the pre- and post-conditions of given action words. Action words themselves are not automatically extracted, however this could easily be done using a list of transitive verbs for example. For each given action word the system collects web documents from google searches. Candidate pre- and post-conditions are identified using a statistical heuristic. Two binary classifiers are trained on a hand-labeled collection of actions, using features extracted from the collected documents. Both of these classifiers take as input a single (action, candidate) pair (A, C). Given a pair (A, C), the first classifier identifies whether C is a pre-condition of A. Similarly, the second classifier determines whether C is a post-condition of A.

Features are chosen to be independent of the specific choice of action word A. The intent is that they only indicate whether there is a causitive relationship between A and C. Thus the two trained classifiers should be able to identify pre- and post-conditions for previously unseen action words which have similar causitive relationships.

2.1 Data Labeling

Sil et al. (2010) randomly select "a set of 40 actions from the lexical units in the frames that inherit from the `Transitive_action` frame in FrameNet (Johnson et al., 2003)". They label each action with a list of pre-conditions and a list of post-conditions (of average length 4.2 and 3 respectively). Only one partial example of labeled pre- and post-conditions of an action is given (in discussing their results). They identify "knife", "sharp", and "person" as being pre-conditions, and "blood" and "bloodshed" as being post-condtions, of the action"cut". This contrasts with the more idealised system they describe where true generalised pre- and post-conditions are distinguished from candidates which are merely *sometimes* true before or after the action. For example they state that "ice" is not actually a

	`arresting`	`washing`	`kindling`
Pre	police, criminal, crime, warrant	dirty, stains, soap, water, detergent, laundry	sticks, paper, spark, firewood, matches, match, logs
Post	jail, charges, fines, caught	clean, washed	fire, blaze, flame, heat

Figure 2: Three example labeled pre- and post-conditions

precondition of "melt" (prefering the precondition "solid"). By this logic, "knife" and "person" are not true pre-conditions and "blood" and "bloodshed" are not true post-conditions of "cut" (one could envisage a chimpanzee cutting paper with scissors).

I chose a set of 40 actions in a similar way. However, I intentionally avoided actions which were not well described by the simplified PrePost representation due to having a very general meaning or multiple meanings. Where actions had one usage that was much more common than other (potentially more general) meanings, I chose the common meaning. Some examples of labeled actions are given in Figure 2. The full list can be found in my git repository[1].

2.2 Document Collection

For each labeled action word A, PrePost collects documents by using a search engine to search "(is|are|was|were) A-ing". The idea behind this is to "select documents where the word A is being used as an action, rather than as a noun or perhaps a sense of the word A that is not an action" (Sil et al., 2010).

In my system the python package `google-search`[2] was used to query google for the phrases "is A-ing", "was A-ing", etc. Here A-ing is taken to mean the present participle of A, so for example "juggle" becomes "juggling" and "cut" becomes "cutting". For each phrase 200 results were requested with a delay of 60 seconds every 10 results to avoid being timed out by google. The corresponding webpages were downloaded and parsed using `jusText`[3] to extract body text from the HTML. This removed most of the text that was unrelated to the content of the article, such as advertisements and navigational links. The downloaded documents were then checked to ensure they contained A, because some websites

[1] This is currently cecs-private https://gitlab.cecs.anu.edu.au/u6384109/prepost
[2] https://pypi.org/project/google-search
[3] https://pypi.org/project/jusText

would serve non-related texts such as requests to solve a CAPTCHA.

I chose to exclude the action "desiccate" from the dataset since my system only collected 28 documents for it. For the remaining actions there was an average of 418 documents collected with a minimum of 190 ("weaken") and a maximum of 561 ("demolish"). The system works relatively well.

2.3 Candidate Selection

PrePost selects potential preconditions and postconditions for an action A by computing the pointwise mutual information (PMI) between "A-ing" and every other word in the documents collected for A. PMI is a measure of how often two words appear in the same document, weighted by how many documents they appear in individually. If D is the set of documents then we define

$$PMI(A, C) = \log \frac{|\{d \in D | A, C \in d\}|}{|\{d \in D | A \in d\}||\{d \in D | C \in d\}|}$$

It is worth noting that PAPE1 defines PMI as

$$PMI(A, C) = \log \frac{|\{d \in D | A, C \in d\}|}{|D_A||\{d \in D | C \in d\}|}$$

where D_A is the documents collected as described above for the action word A. The difference here is that in their definition occurences of "A-ing" outside the documents collected for A are not counted. I could see no reason for this restriction and the former definition is used in Sil and Yates (2011) so I went with that.

To enable fast calculation of PMI my system precomputes a dictionary mapping each word to the set of documents they appear in. To exclude mispellings and other errors, words are first filtered using a list of around 100,000 english words[4]. From this dictionary PMI can be efficiently computed using Python's set cardinality and intersection methods. After the PMI between A and all other words has been computed the 500 words with highest PMI are selected as candidates.

Sil et al. (2010) report that their method was enough to ensure over 95% of labeled pre- and postconditions appeared among the selected candidates. In my experience this was not the case: only 5% of my labeled pre- and postconditions were recovered as candidates. The main cause of this was words that appeared very rarely. For a fixed action A, $\text{PMI}(A, C)$ is maximised by maximising $\frac{|\{d|A, C \in d\}|}{|\{d|C \in d\}|}$, thus words which appeared once in a document containing A and nowhere else were given the highest possible rating. This led to words like *aadvarks*, *encaenia*, and *pfennig* appearing as candidates for *murder*.

To address this I imposed minimums for the number of times a candidate word must appear in the document set and the number of documents it must appear

[4] Available in my repository or at https://cs.anu.edu.au/courses/comp1730/labs/data/wordlist.txt

	`arresting`	`washing`	`kindling`
PMI only	arresting	washing	kindling
	kibitz	uncaught	crossbreeding
	fossicking	pressurizes	vermis
	hairlocks	unhitching	outbred
	hairlock	rocketry	morbidities
	quilled	tetanies	inbreed
	ungloved	mullens	forelimbs
	presaging	mangers	hyperexcitable
	prefabbing	hollands	amygdalae
	subhumid	enjoin	focally
PMI and min frequency	arresting	washing	kindling
	detaining	detergents	kindled
	jailing	shampooing	amygdala
	detain	lather	twigs
	arrests	softener	kindle
	handcuffed	detergent	campfire
	handcuffs	washer	lobe
	lawfully	grime	alight
	deport	shampoos	firewood
	arrest	washers	epilepsy

Figure 3: Top ten selected candidates with and without minimum frequencies.

in[5]. This created a trade-off between generally widely appearing candidates — which had lower PMIs and thus benefited from higher choices of minimums — and more specific candidates. For example, "masseur" occurred only 35 times in the document set and in only 17 documents. If we lowered the minimums to allow this to be selected as a candidate for "massage" then the candidate "table" did not make the PMI ranking cut-off. I experimentally chose the minimums of 300 occurences and 25 documents. This resulted in 85% of my labeled pre- and post-conditions being recovered.

2.4 PMI Features

To classify C as a precondition of A or not the first feature used is $\text{PMI}(A, C)$. This feature gives a measure of the association between A and C and allows the classifier to identify words which appear more frequently with A as more likely to be pre- or post-conditions. However, candidate words may be highly associated with an action even if they are not true pre- or post-conditions. For example, "drying" appears often with "washing" and is neither a pre-condition nor a post-condition.

To address this issue Sil et al. construct a set of *feature words*. The idea is that for example words like "requires" or "before" may be more highly correlated with true action-precondition pairs than non-precondition pairs, regardless of how correlated the words in the pair are with each other. To measure this three way association between A, C, and a feature word F, they use three way PMI defined[6] as

$$PMI(A, C, F) = \log \frac{|\{d \in D | A, C, F \in d\}|}{|\{d \in D | A \in d\}||\{d \in D | C \in d\}||\{d \in D | F \in d\}|}$$

In theory this should, as Sil et al. (2010) put it, "allow the classifier to learn the nature of association between candidate words and action words".

Rather than manually constructing such a set of feature words they propose a system to automatically select them. First, "to ensure that [their] PMI statistics would have sufficient numbers of occurrences to work with" (Sil et al., 2010), they filter their dataset for words which appeared more than 500 times. They reported that this resulted in around 3000 candidate feature words. Using these candidate feature words they "computed χ^2 values by comparing each candidate feature word and [their] labeled pre and postconditions" (Sil et al., 2010). They then chose an experimental threshold on the χ^2 value which resulted in 161 feature words to use in extracting both pre- and post-conditions.

This part of Sil et al. (2010) was the most difficult to interpret. I selected candidate feature words using the same threshold of 500 which resulted in a

[5] This second condition was inspired by a word which appeared over 500 times in a Victorian era erotic novel, and almost nowhere else.

[6] As before PAPE1 actually defines it as

$$PMI(A, C, F) = \log \frac{|\{d \in D | A, C, F \in d\}|}{|D_A||\{d \in D | C \in d\}||\{d \in D | F \in d\}|}$$

similar amount of candidates (3419). Even after doing so over 99% of candidates feature words resulted in undefined PMI values for at least one point in my dataset. I interpreted the χ^2 test as a test of categorical independence. That is, whether the PMI values were statistically independent of the label given to the data. Sil et al. select the same set of feature words for both pre and postconditions. Since the idea is to train two separate classifiers I simplified things by selecting them separately.

Sil et al. do not make it explicit whether they select features using their entire dataset or only the training set. To avoid biasing performance estimates I restricted feature selection to the training set. A more thorough treatment could involve feature selection during cross-validation. See Ambroise and McLachlan (2002) for a discussion.

While χ^2 feature selection is a very standard technique it is somewhat unclear how Sil et al. use it. Normal χ^2 feature selections works on frequency tables or counts which must have non-negative values. However since PMI is a log of a number between 0 and 1 it is always negative. Using Python's `sklearn.chi2` (Pedregosa et al., 2011) I tried three different approaches to address these two issues.

Simply working without the log results in values that are always defined and non-negative. However, this resulted in an almost meaningless χ^2 test where all p-values were above 0.99. One possible cause is numerical instability in the Python implementation of the χ^2 statistic. Words that were selected using this method were highly dependent on the choice of training data. For example, the top ten words selected for a training set containing the action "juggling" included "jugglers", "juggle", "juggling", and "balls". Two disjoint training sets of five actions shared only 6/161 feature words. Thus I rejected this method.

My second approach was to exclude any undefined PMI values and then take the negative of the defined ones in calculating the χ^2 statistic. The intent was to then replace undefined PMI values during classification by an artificial "negative infinity" represented by a large negative number. This produced similarly data-dependent feature words which did not serve the intended purpose.

My third approach was to use a modified definition of three way PMI defined as

$$PMI'(A, C, F) = \log \frac{1 + |\{d \in D | A, C, F \in d\}|}{1 + |\{d \in D | A \in d\}||\{d \in D | C \in d\}||\{d \in D | F \in d\}|}$$

This was defined everywhere and I could compute χ^2 statistics using its negative which is always greater than zero. The resulting p-values for the 161 chosen feature words ranged from around 0.05 to 0.12; a huge improvement. Selected feature words included "and", "to", "with", "when", "before", "never" and "while" which from a human standpoint seem like much better choices. Using this method the two disjoint training sets compared above shared 62 feature words.

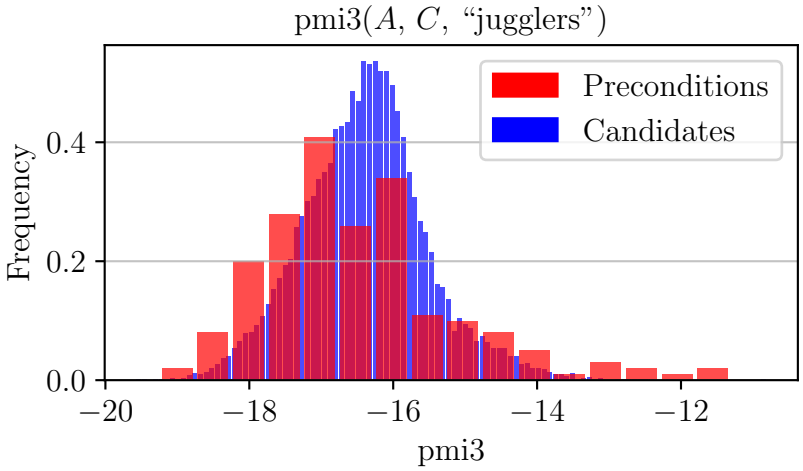

Figure 4: Normalised three way PMI frequencies for feature word “jugglers”, the highest ranked candidate feature word using the first method.

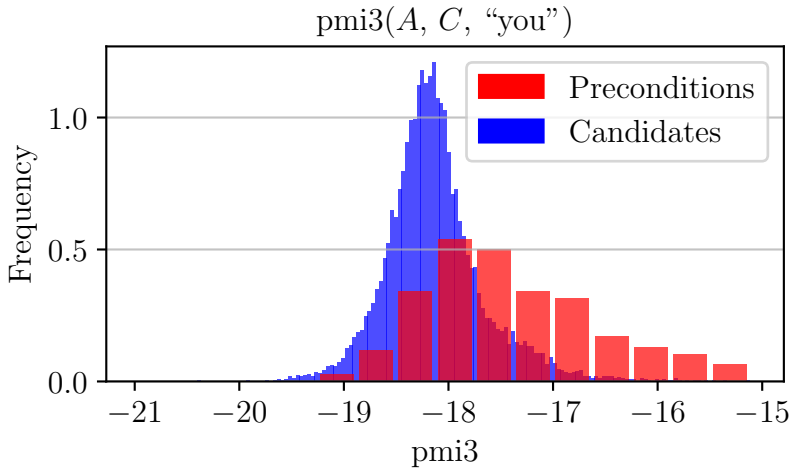

Figure 5: Normalised three way PMI frequencies for feature word “you”, the highest ranked candidate feature word using the third method. Note that the two distributions appear to differ, which would indicate dependence on the label.

2.5 Structural Information

The use of PMI only gives "coarse-grained, document-level features that ignore the significant structure in language and the prediction task" (Sil et al., 2010). Sil et al. (2010) identify that for many actions the arguments to the action are often its pre- and post-conditions. They give the example sentence

> "Kava is a highly prized medicinal herb whose primary benefit is *alleviating* anxiety and minor pain."

where the precondition "pain" acts as an argument to the predicate "alleviating". To allow their classifiers access to this structural information Sil et al. annotate their dataset with a Semantic Role Labeling (SRL) system and Coreference Resolution (Coref.) system and use this information to extract five different features.

2.5.1 Semantic Role Labeling

SRL refers to the task of labeling words or phrases with the semantic role they play in a sentence. There is no single standard set of labels used in SRL, however common labels include the verb, the actor/agent of the verb, and the patient (the thing which is acted on). The two sentences below are annotated using labels from PropBank (Palmer et al., 2005). Note that the semantic role of a word is distinct from its syntactic role. In the sentence

$$\underbrace{\text{The girl}}_{\text{A0}}\ \underbrace{\text{hit}}_{\text{V}}\ \underbrace{\text{the ball}}_{\text{A1}}$$

"The girl" is the subject of "hit" whereas in the passive formulation

$$\underbrace{\text{The ball}}_{\text{A1}}\ \text{was}\ \underbrace{\text{hit}}_{\text{V}}\ \text{by}\ \underbrace{\text{the girl}}_{\text{A0}}$$

it is the object. However, in both sentences "the girl" is the actor (A0).

Sil et al. used an SRL system developed in Huang and Yates (2010). Huang and Yates don't provide an out of the box implementation of their system so I searched for an alternative. I settled on SENNA (Collobert et al., 2011), which provided fast SRL and offered similar performance to the system developed by Huang and Yates. It is worth noting that there are more recent SRL systems which claim to achieve much higher performance, for example He et al. (2017).

In multi-clause sentences words often play multiple semantic roles. SENNA indicates this with a multicolumn output where each column corresponds to the roles played by words in a particular clause. Tags are in IOBES format. The prefix (I, O, B, E, or S) indicates what part of a tag they are (Interior, Outside, Beginning, End, or Singleton tag). The suffix is the semantic role. So for example "girl" in the above sentences would be tagged "E-A0".

I parsed the SENNA output into a graph structure where the nodes were words, tags, and clauses. Each word is linked to the tags it is labeled with and

Word	Predicates	Clause 1	Clause 2
How	-	B-AM-TMP	O
often	-	E-AM-TMP	O
you	-	S-A0	S-A0
should	-	S-AM-MOD	O
change	change	S-V	O
bed	-	B-A1	O
sheets	-	E-A1	O
to	-	B-AM-PNC	O
avoid	avoid	I-AM-PNC	S-V
bugs	-	I-AM-PNC	B-A1
and	-	I-AM-PNC	I-A1
mould	-	E-AM-PNC	E-A1

Figure 6: Example SENNA SRL output

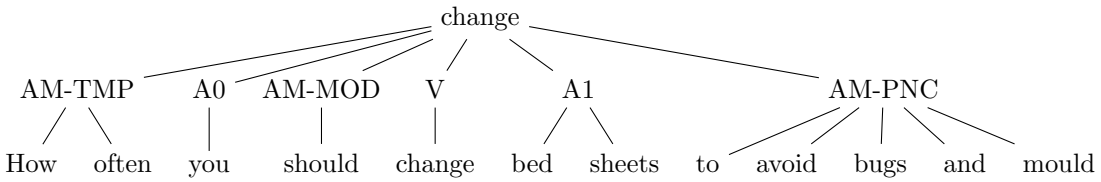

Figure 7: Graph representation of Clause 1

each tag is linked to the clause that contains it. Tags that denoted an argument or predicate were also marked with that information. In doing this I noticed a relatively rare bug: SENNA occasionally doesn't properly begin a tag with the B prefix (instead having the first word with an E prefix). Fortunately this exception could be accounted for.

2.5.2 Coreference Resolution

Coreference resolution is the task of identifying phrases in text which refer to the same entity. For example in

> "My sister has a pet dog. She takes him for walks."

the noun phrases "My sister" and "She" both refer to the same thing (similarly for "a pet dog" and "him").

Coreference resolution is a non-trivial task. One reason for this is that a coreference system must combine syntactic and semantic information (often with contextual knowledge) to correctly identify coreferent phrases. As an example consider the sentences

> "The bowling ball broke the glass table because it was heavy."

and

"The bowling ball broke the glass table because it was fragile."

Syntactically the two are identical. However, in the first sentence we can identify the pronoun "it" with "The bowling ball" since being heavy implies the ability to break rather than breakability. Similarly, in the second sentence "it" is identified with "the glass table" since being fragile implies breakability. Things are futher complicated through metaphor where phrases figuratively refer to the same thing ("But soft, what light through yonder window breaks? It is the east, and Juliet is the sun!") or in puns where the same phrase refers to different things ("Mr Leopold Bloom ate with relish the inner organs of beasts and fowls").

There are several publically available software packages which advertise out of the box coreference resolution. Sil et al. used Apache OpenNLP[7] (written in Java). I initially tried using OpenNLP but found that its coreference functionality lacked documentation and did not have a command line interface which made it difficult to integrate with my work (written in Python). Instead I used Stanford CoreNLP[8] which was better documented and offered a command line interface as well as wrappers for a variety of different languages.

CoreNLP's output contains a lot of irrelevant information so I used the Python wrapper[9] to reduce this to collections of coreferent phrases (and their sentence number, word number addresses). I also retained information about sentence and word character offsets to allow me to integrate the coreference information with the semantic role labelling of SENNA. I did this by augmenting the word/tag/clause graph I had constructed earlier with phrase nodes which linked to the words in the corresponding phrase, and any other coreferent phrases. This was actually a significant challenge as SENNA and CoreNLP tokenised (split plaintext into words) differently. For example SENNA would split a Java function call "Class.method()" into the tokens "Class", ".", "method", and "()", whereas CoreNLP would maintain the whole string as a single token "Class.method()". Using character offset information I identified strictly contained tokens, so in my graph"method" would be linked to any phrase CoreNLP identified as containing "Class.method()". I ignored tokens which overlapped but were not strictly contained in each other as these occurred relatively infrequently.

2.5.3 Structural features

From the structural annotations Sil et al. calculate five features for each action-candidate pair (A, C). The first three are determined only using SRL, whereas the last two rely on both SRL and coreference information. As discussed above, the features all indicate predicate-argument relationships between A and C. In Figures 8–12 the action "washing" is used as a recurring example to indicate typical distributions of these features. Inspecting these Figures we see many of the highly ranked features are pre- or post-conditions.

[7] https://opennlp.apache.org/
[8] https://stanfordnlp.github.io/CoreNLP/
[9] https://stanfordnlp.github.io/stanfordnlp/

Feature 1: Arg in

To capture the fact that pre- and post-conditions often appear as arguments to their action, Sil et al. calculate "how often the candidate word C appears as part of a phrase that is an argument to the action word A". In evaluating this (and the proceeding features) I counted the predicates in the form "A-ing" rather than "A", since that was the form selected for in document collection.

Candidate	**Counts**
machines	166
clothes	97
soda	71
dishes	61
washing	31
detergent	26
beaches	24
soap	23
powder	19
laundry	19
⋮	⋮
softener	0
lather	0

Figure 8: Highest ranked candidates for "washing" by **Arg in**

Feature 2: Arg anywhere

The second structural feature is a count of "how often the candidate word appears as an argument to any predicate at all". There is a commonly occurring ambiguity here where in a multi-clause sentence a single word might be the argument to multiple predicates. I took the literal interpretation and counted these multiple occurences.

Candidate	Counts
washing	2756
clothes	2026
bacteria	1883
wash	1344
soap	1257
machines	1235
wet	1062
cleaning	988
laundry	962
⋮	⋮
anxiously	5
scrubbed	5
adheres	2

Figure 9: Highest ranked candidates for "washing" by **Arg anywhere**

Feature 3: Arg near

The next three features were intended to provide less sparse counts. Sil et al. determine a distance based feature, counting "how often C appears as an argument near (less than 50 words away from) A".

Candidate	Counts
washing	5622
wash	901
clothes	634
machines	562
laundry	433
soda	318
soap	264
cleaning	260
detergent	235
⋮	⋮
sinful	0
pitting	0
eyelids	0

Figure 10: Highest ranked candidates for "washing" by **Arg near**

Feature 4: Coref 1

Using the coref. annotations they counted "how often C appears as an argument that is coreferential with an argument of A". Consider the sentences

> "The clothes were covered in mud after the storm. He was *washing* them."

Here "clothes" is taken to be a precondition[10] of "wash". While not directly an argument to "washing", "clothes" is coreferential with the argument "them". Thus this feature would count this relationship.

Candidate	Counts
washing	1110
nappies	691
soda	288
machines	139
clothes	119
dishes	103
drum	47
fishermen	35
laundry	32
⋮	⋮
lather	0
shampooing	0
stained	0

Figure 11: Highest ranked candidates for "washing" by **Coref 1**

Feature 5: Coref 2

The final structural feature counts "how often C appears as an argument to a predicate P which has another argument D which is coreferential with an argument of A." Consider the exampleSil et al. (2010)

> "Doctors need to $\overbrace{\textit{heal}}^{A}$ patients. $\overbrace{\text{They}}^{D}$ $\overbrace{\text{are}}^{P}$ the ones who need $\overbrace{\textbf{medicine}}^{C}$."

The candidate "medicine" (C) appears as an argument to the predicate "are" (P) which has another argument "they" (D) which is coreferential with an argument of "heal". Ideally this feature would allow us to use scenarios like this to identify that "medicine" is a pre-condition of "heal".

[10] Clothes aren't a true precondition of "washing" but in the majority of uses online, "washing" is used in the sense of laundry. See 2.1

Candidate	Counts
washing	11588
wash	2450
clothes	1308
nappies	1022
soap	826
laundry	725
shampoo	688
washed	681
detergent	653
machines	625
⋮	⋮
tubs	0
stained	0

Figure 12: Highest ranked candidates for "washing" by **Coref 2**

2.6 Experimental Setup

Sil et al. split their 40 labeled actions into a set of 5 training actions and 35 test actions. Words which are labeled as pre- or post-conditions are taken to be positive examples by their respective classifiers. For a given action, any word in its 500 candidates which is not labeled is taken as a negative example. They use a support vector machine, implemented in SVM-Light (Joachims, 1998), with radial basis function (RBF) kernel and default parameter settings.

I followed their test/train split and used scikit-learn's (Pedregosa et al., 2011) SVM with RBF kernel and default parameter settings. Training time was less than 30 seconds.

3 Results

Sil et al. evaluate the performance of their system with a precision-recall graph (Figure 13). This is generated by varying the threshold which their classifier uses to decide on its classification to trade off the two metrics. I was not able to achieve nearly the same level of precision-recall performance (See Figure 14). In fact, my implementation performs worse than the author's version restricted to only PMI features.

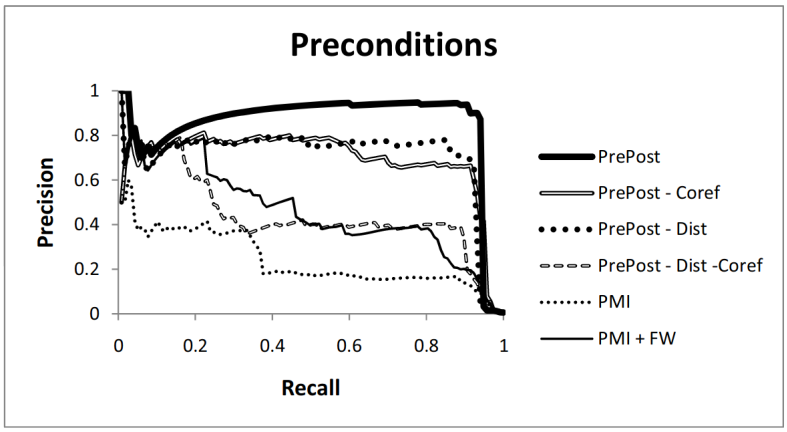

Figure 13: Precondition Precision-Recall (Sil et al., 2010)

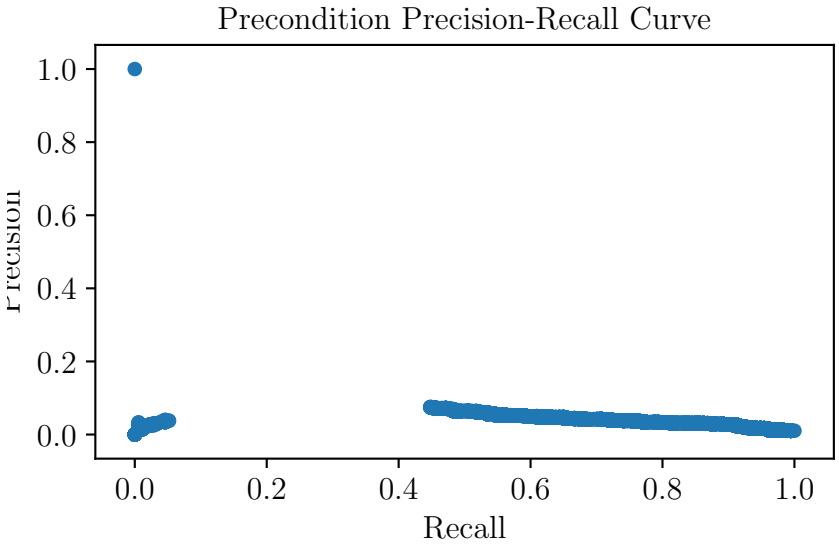

Figure 14: Precondition Precision-Recall for my full PrePost system.

Candidate	Probability
loose	0.28
friction	0.10
bird	0.03
sweat	0.02
overnight	0.01
pollution	0.01
cotton	0.01
excess	0.01
bowl	0.01
consumption	0.01
delicate	0.01
thoroughly	0.01
instructions	0.01
powder	0.01
dirt	0.01
machines	0.01
load	0.01
bacteria	0.01
mesh	0.01
shoes	0.01

Figure 15: 20 most likely preconditions for "washing" and their predicted probability of being a precondition. None of them are labeled preconditions. Interestingly the two highest candidates "loose" and "friction" are preconditions for one of the training actions "chafing".

On the training set my implementation achieves precision and recall over 0.99 (where 1 is a perfect score). This means that the classes are sufficiently separable and that the system is failing to generalise adequately. There are several possible reasons for this discrepancy.

One obvious suspect is my hand-constructed actions. Given the sparsity of labeled examples it is difficult to be sure that my labeling coincides with that of Sil et al. (2010). As previously mentioned, fewer of my labeled pre- and post-conditions appear among the selected candidates and I encountered a trade off caused by the minimum frequencies I introduced.

I had to make several decisions in interpreting the PMI features. My method for selecting words as feature words with χ^2 may have differed from theirs. They may also have dealt with undefined PMI in a different way.

It is possible that SENNA or CoreNLP performed substantially worse than their SRL and coreference systems. This could feasibly lead to a reduction in the quality of structural features, thus making them less useful in distinguishing examples.

The default parameters chosen by `sklearn`'s SVM implementation may have differed from those of SVM-light. In particular the two systems may have dif-

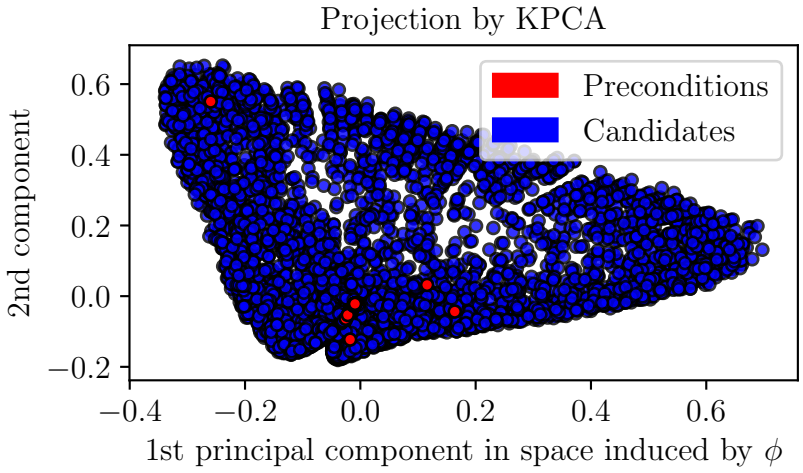

Figure 16: Dimension reduction of entire dataset using PCA with RBF kernel. In two dimensions the data is not linearly separable. We can see however that true preconditions are relatively clustered.

ferently dealt with the imbalanced dataset (`sklearn`'s default behaviour is to weight classes proportionally).

Finally, while I have performed many sanity checks on the individual components of my system it is difficult to inspect 20,000 datapoints, each with over 150 features. Thus there is always the chance that my system contains some fatal bug.

4 Conclusion

Sil et al. (2010) formulate the problem of event extraction and convincingly argues its importance in the context of classical planning. While their exposition lacks detail and I was ultimately unable to reproduce their results, the system they propose is the beginning of a plausible solution. Its largest flaw is failing to distinguish between multiple meanings a single action word may have (word sense disambiguation). Sil and Yates (2011) expand on this work substantially to include argument extraction and generalisation to hypernyms. There are a number of gaps in their work too, particularly in their treatment of word sense disambiguation. A follow-on project could be to more thoroughly diagnose the reasons for my implementation's poor performance as well as implement the extensions of Sil and Yates (2011).

References

Ambroise, C. and G. J. McLachlan
2002. Selection bias in gene extraction on the basis of microarray gene-expression data. *Proceedings of the national academy of sciences*, 99(10):6562–6566.

Chambers, N. and D. Jurafsky
2008. Jointly combining implicit constraints improves temporal ordering. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, Pp. 698–706. Association for Computational Linguistics.

Chambers, N. and D. Jurafsky
2009. Unsupervised learning of narrative schemas and their participants. In *Proceedings of the Joint Conference of the 47th Annual Meeting of the ACL and the 4th International Joint Conference on Natural Language Processing of the AFNLP: Volume 2-Volume 2*, Pp. 602–610. Association for Computational Linguistics.

Collobert, R., J. Weston, L. Bottou, M. Karlen, K. Kavukcuoglu, and P. Kuksa
2011. Natural language processing (almost) from scratch. *Journal of machine learning research*, 12(Aug):2493–2537.

Haslum, P.
2012. Narrative planning: Compilations to classical planning. *Journal of Artificial Intelligence Research*, 44:383–395.

He, L., K. Lee, M. Lewis, and L. Zettlemoyer
2017. Deep semantic role labeling: What works and what's next. In *Proceedings of the Annual Meeting of the Association for Computational Linguistics*.

Huang, F. and A. Yates
2010. Open-domain semantic role labeling by modeling word spans. In *Proceedings of the 48th Annual Meeting of the Association for Computational Linguistics*, Pp. 968–978. Association for Computational Linguistics.

Joachims, T.
1998. Making large-scale svm learning practical. Technical report, Technical report, SFB 475: Komplexitätsreduktion in Multivariaten ....

Johnson, C. R., M. Schwarzer-Petruck, C. F. Baker, M. Ellsworth, J. Ruppenhofer, and C. J. Fillmore
2003. Framenet: Theory and practice.

Manikonda, L., S. Sohrabi, K. Talamadupula, B. Srivastava, and S. Kambhampati
. Extracting incomplete planning action models from unstructured social media data to support decision making. *KEPS 2017*, P. 62.

Martin, L. J., P. Ammanabrolu, X. Wang, W. Hancock, S. Singh, B. Harrison, and M. O. Riedl
2018. Event representations for automated story generation with deep neural nets. In *Thirty-Second AAAI Conference on Artificial Intelligence*.

Palmer, M., D. Gildea, and P. Kingsbury
2005. The proposition bank: An annotated corpus of semantic roles. *Computational linguistics*, 31(1):71–106.

Pedregosa, F., G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay
2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

Riedl, M. O. and R. M. Young
2010. Narrative planning: Balancing plot and character. *Journal of Artificial Intelligence Research*, 39:217–268.

Sil, A., F. Huang, and A. Yates
2010. Extracting action and event semantics from web text. In *2010 AAAI Fall Symposium Series*.

Sil, A. and A. Yates
2011. Extracting strips representations of actions and events. In *Proceedings of the International Conference Recent Advances in Natural Language Processing 2011*, Pp. 1–8.

Tandon, N., G. De Melo, A. De, and G. Weikum
2015. Knowlywood: Mining activity knowledge from hollywood narratives. In *Proceedings of the 24th ACM International on Conference on Information and Knowledge Management*, Pp. 223–232. ACM.