A Supervised Approach To The Interpretation Of Imperative To-Do Lists

Paul Landes
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA
plande2@uic.edu

Barbara Di Eugenio
Department of Computer Science
University of Illinois at Chicago
Chicago, IL 60607, USA
bdieugen@uic.edu

Abstract

To-do lists are a popular medium for personal information management. As to-do tasks are increasingly tracked in electronic form with mobile and desktop organizers, so does the potential for software support for the corresponding tasks by means of intelligent agents. While there has been work in the area of personal assistants for to-do tasks, no work has focused on classifying user intention and information extraction as we do. We show that our methods perform well across two corpora that span sub-domains, one of which we released.

1 Introduction

To-do lists are pervasive and offer a concise representation of tasks that need to be accomplished (Bellotti et al., 2004; Gil et al., 2012), with studies showing that up to 60% of the population use them (Jones and Thomas, 1997). To-do tasks are often expressed as short utterances and written in list form. Examples of to-do task are *"Buy swimsuit"*, *"Call mom"*, and *"Hotel reservation"*.

With the growing popularity of personal assistants, such as Apple's voice-based Siri, the need for understanding and executing tasks such as those embodied in to-do tasks continues to grow; conversely, there is great potential in automating the resolution of to-do tasks given the plethora of mobile and desktop applications that currently exist.

This paper focuses on describing several models we have constructed to interpret to-do tasks. As far as we know, there has been little research in parsing and classifying to-do tasks, which entirely consists of short requests couched either as imperative sentences or as fragments not containing a verb; neither imperatives nor fragments are as frequent in corpora as other forms in which a user's intention can be expressed, as we will discuss in Section 2.

Execution of these tasks can be achieved by invoking an appropriate *intelligent agent* (IA), which will act upon the task (Gil et al., 2012). For example, *"Call mom"*, would yield: {agent}=⟨*call*⟩, {arguments}=⟨*contact=mom*⟩ and would look up `mom` as contact and dial via the phone application. IAs are described in Section 3.

The challenges with processing to-do tasks are:

- **Short length of utterances**, which yield poor results with current methods (Han and Baldwin, 2011).
- **Missing head verb**, which makes classification more difficult if missing. For example, *"Hotel reservation"* lacks the head verb, *schedule*.
- **Disambiguation of named entities**, which include products to purchase, persons with whom to communicate, companies, etc.
- **Processing commands in imperative form**, for which in most systems are well formed and don't contain many named entities.

In this paper, we discuss our approach to mapping a to-do task to a corresponding IA; this includes extracting its *argument*s and classifying them as concerns their types. We will describe the corpus we built, the preprocessing steps we devised, and the features we used to inform our classifiers.

Finally, we will demonstrate the generality of our approach by applying it to a compiled and annotated corpus with a partially separate domain.

2 Related Work

From a linguistic point of view, to-do tasks are either expressed with (short) imperative sentences or with fragments that do not contain a verb. Imperatives have not been studied very extensively in NLP, since they (not surprisingly) do not occur frequently in corpora of "standard" English, such as newspaper articles. For example, the PARC 700 dependency bank (King et al., 2003) is a random sample of 700 utterances taken from the Wall Street Journal corpus (Marcus et al., 1994) and contains only seven utterances containing at least one imperative.

However, even in datasets that focus on dialogue and tasks to be performed, imperatives do not occur that often. To start with, as (Levinson, 1983) notes, the imperative is rarely used to issue requests in English. This seems to be borne out even in interactions with personal assistants, even if, beyond anectotal evidence, distributional analyses of such data are not widely available.

One such analysis is provided by (Tur et al., 2014), who analyzes interactions between users and an entertainment personal assistants. They found that 31.21% of such utterances start with a VB tag (an upper bound for imperatives), another 31.64% with tags representative of NPs and wh-NPs (the remaining 37% of utterances is left unspecified): so even when communicating with an entertainment personal assistants, users use imperative sentences only about $\frac{1}{3}$ of the time (this frequency is not rare, but much lower than one would expect given the circumstances). In their work on evaluating the performance of personal assistants such as Siri, Google Now and Cortana, Jiang et al. (2015) show that 67% of user action types are commands, but they don't specify the proportion of imperatives within those.

A substantial percentage of to-do tasks are expressed as fragments, such as *"Hotel reservation"*. The literature on fragments is also rather sparse, other than as concerns ellipsis (Kempson et al., 2015)–but it is not clear that these fragments are in fact ellipsis. Even tweets, the shortest of today's social media language, are often grammatically ill-formed, contain a main verb.

If we now turn to the interpretation of to-do tasks, Gil et al. (2012) explored different kinds of intelligent assistance for to-do lists. Gil et al. (2012) provides a manual data entry to categorize the to-do task and Jiang et al. (2015) yields a performance evaluation, however, neither automatically categorize the IA as we do. Likewise (Tur et al., 2014) distinguishes between requests that are covered by the current interpreter and those that are not, but does not interprets them in anyway.

In this work, we also tackle argument extraction. [1] Our method of extracting verbs and their arguments is similar in spirit to efforts like OpenIE (Fader et al., 2011) and NeLL (Mitchell et al., 2015). Our work is specifically targeted at to-do lists where sentences are very short and telegraphic. OpenIE and NeLL learn from a large repository of knowledge, whose language is not telegraphic, and mostly well formed.

Earlier work in argument extraction includes processing sequences of executable actions for the Windows operating system by Branavan et al. (2009). Other argument extraction work includes efforts in deep learning by Meerkamp et al. (2017) for boosting precision of existing extraction systems. However, our work covers a broader domain of general English to-do tasks and does not fit in any existing method of tagging arguments.

The approach taken by Ghani et al. (2006) is the most similar to ours with regards to their argument extraction for textual product descriptions. However, their argument set is restricted to a predefined set where our model learns the arguments to extract.

3 Corpora

This work involves two to-do task data sets: the first is a proprietary corpus[2] and the sec-

[1] Argument extraction can be considered equivalent to slot filling as defined in many spoken dialogue systems. We follow e.g. Propbank (Kingsbury and Palmer, 2002) in using the term "arguments".

[2] Regretfully, we can not share the proprietary annotated corpus.

Intelligent Agent	Description	Corpus A Count	Corpus B Count
buy	Assists in buying goods.	480	52
service	Do It Yourself type tasks	284	46
self-improve	Self Improvement/Help	183	4
school-work	Task related to school	158	8
contact	Email, SMS or call	101	47
call	Makes a phone call via OS	97	19
email	Emails a contact via OS	60	12
calendar	Make an appointment	55	22
pay-bill-online	Online bill pay	54	17
find-service	Procure services	42	27
print	Print out a document	23	4
postal	Send mail by snail mail	20	11
plan-meal	Cook or gather ingredients	17	7
find-travel	Reserve transportation	18	10
text-sms	Sends SMS text messages	19	19
Total used		1,611	305
search	Search data on the device	14	-
find-activity	Abstract activity	14	-
reminder	Creates a reminder	12	-
repair	Repair a broken object	11	-
travel	Errands involving travel	8	-
return	Return an item to a retailer	6	-
rent-media	Rent a movie	4	-
find-food	Find food online or call	4	-
sell	Sell or get rid of an object	3	-
scan	Scan a document	2	-
how-to	Learn an activity	1	-
Total annotations	1,690	305	

Table 1: Intelligent Agent distribution in the corpora.

ond is a publicly available corpus we have built and released[3].

The corpus consists of IA and argument annotations. The IA type was formulated from an initial analysis of the corpus and several iterations of annotation guideliness that led us to the final list given in Table 1. The argument annotations are tokens associated with an IA annotation and provide additional context helpful in task resolution. A complete list of arguments are given in Table 2 and described in more detail in Section 3.3.

3.1 Corpus A

Corpus A was collected from two sources: publicly available online sources (see Appendix A for a full list of sources) and private sector data. We annotated a random sample of 3,169 to-do tasks with one utterance per task and doubly annotated 1,342 to compute the kappa score. We then divided the 3,169, into usable *non-exceptions* (1,690) and unusable *exceptions* (1,479) to-do tasks. A task was considered as an exception for one of the following reasons:

- the to-do task itself is ambiguous (i.e. *"flowers"*–plant them or buy them?)
- language is not English (i.e. *"compra flores"*) or meaningless (i.e. *"mkmkmk"*)
- illegal activity (i.e. *"buy drugs"*)
- generic professional (i.e. *"first quarter presentation"*)

The 1,690 non-exception tasks were annotated with IA labels and their respective arguments. IAs with less than 15 utterances were pruned as anything less proved to be insufficient training data. This left 1,611 tasks with utterances used for training and testing.

3.2 Corpus B

We created a Corpus B composed of 102 volunteer contributed personal to-do tasks and 498 Trello[4] to-do tasks with IA annotations. A subset of this data, including 68 volunteer and 218 Trello scraped to-do tasks, was used to test and train the model. The Trello data was sourced from public boards that allow for redistribution and the volunteers agreed to release their data for public distribution. This

[3] https://github.com/plandes/todo-task/

[4] http://trello.com

Argument	Description	Count	Argument	Description	Count
item	The item to buy	1415	size	Product size	13
contact	Who to buy for	637	department	Department related product	12
subject	Subject of email	166	amount	Item amount	11
complete	Homework to complete	144	holiday	What to buy for	9
topic	What to discuss	127	message	Message to send	9
when	Event time	118	mail type	Type of mail to send	8
title	Title of the event	80	recipe	Meal name	6
retailer	Where to buy	51	register	Register for a class	6
type	Type of event	49	gender	Gender of product	5
brand	Proper noun of purchase item	40	ingredients	Recipe ingredients	5
document	Document to print	38	channel	Method to get the media	5
quantity	Number of items	32	origin	Depart location	3
meal	Which meal	29	depart-when	Time to depart	3
location	Where to buy	28	artist	Item artist	3
duration time	How long to use the service	19	purchase	Material	3
special occasion	Holiday	19	number	Contact phone	3
color	Product color	17	arrive-when	Time to arrive	1
term	Search term	16	notes	Free form notes	1
event	Event name	16	action	What to learn to do	1
destination	Origin location	14	body	Body of email	1

Table 2: Argument distribution in the Corpus B.

corpus does not contain argument annotations given the purpose was to provide a way to reproduce IA classification results.

3.3 Argument Annotations

There is a zero to many relationship between IAs and arguments. For example, *"grocery store"* is tagged as a `buy` IA with no arguments.

Table 2 lists the argument annotations and their distribution in the corpus. Note that some arguments span multiple IAs while others are specific to a particular IA. An additional `descriptor list` annotation was provided to address edge cases where existing modifiers were insufficient. For example, the descriptor list would be populated with a URL since there is no corresponding argument for `calendar appointment`.

A fully annotated task of *"Get new sweater for John before Christmas"* is shown in Figure 1 and has the annotation {agent}=⟨*buy*⟩, {arguments}=⟨*item=sweater, person=John, holiday=Christmas*⟩.

An inter-coder agreement metric was computed to get an idea of how consistent the corpus was being annotated using Fleiss' kappa (Fleiss, 1971).

Early efforts were made to create a consistent corpus with sufficient overlap and annotation guidelines, in the face of subtle differences. This work proved worthwhile as the coder agreement of IA classification produced an agreement 0.679 over 848 tasks annotated by two annotators.

The Corpus B contains no argument annotations.

4 Corpus Processing

In earlier experiments, significant propagation errors in the pipeline were found that resulted in poor IA and argument classification due to incorrect *part of speech* (POS) tagger *main verb*s (described in Section 5.2). To overcome, this we added a pre-processing step to enhance the Stanford Parser and created features from the parsed utterances to train and test the models. This process includes:

1. Extend the Stanford *Named entity recognition* (NER). First, we compiled named entity lists that are supplied to the named entity recognition system at parse time and then we parsed the annotations.

2. Build the *first verb* model. To ameliorate errors caused from issues of parsing short utterances, the POS tagger was enhanced by creating the first verb model.

3. Apply the first verb model to the corpus.

This prevents significant error propagation by correcting the first POS tag created in the previous step. This correction is seen later as IA and argument models utilize the main verb parsed at this step.

4.1 Extending Named Entity Recognition

NER proved to be crucial as it provides additional context for classification. Two sets of features were created using both the NER (Finkel et al., 2005) and the Stanford TokensRegex (Chang and Manning, 2014). The latter was enhanced to include a set of static word lists (NER lists) generated from Wikidata (Wikidata, 2015), Open Product Data (Data, 2016), and the term lists.

The Wikidata lists were created with a set of SPARQL (Prudhommeaux and Seaborne, 2006) queries that included a primary term, and for some lists, a sublist term. The term lists also were annotated with three levels of categories, gender for human names, and list type (i.e. `modifiers`, `attributes` and `products`). These NER lists were used to create a Stanford TokensRegex formatted regular expression input file and subsequently used during parse time for each to-do task. The combined Wikidata and the term list produced a total of 85,777 unique entities.

4.2 Building the First Verb Model

The most crucial error made by the POS tagger was incorrectly tagging the first token of utterances as non-verbs. The input to the first verb model are features of the parsed utterances given by the Stanford Parser and the class is the main verb, which is used by the IA and argument models. Out of 1,690 utterances, 653 (38.6%) have the main verb annotation in the Corpus A, which is similar to the initial word utterance relative POS tag frequences (31.21%) of the Tur et al. (2014) VPA and web search datasets.

4.2.1 Bootstrap the Model

For the first parsing of the corpus this error was corrected by reassigning the POS tag of the initial token using the following criteria:

a) identified as a present tense verb tag in WordNet[5] (Miller, 1995) and

b) identified as not a color as in "*yellow* curry" is a noun and not a verb as in "*yellow* your teeth from coffee".

Each annotation included a token with a verb POS tag that differentiated each to-do task across IAs, which was in turn used as a feature. This was used to test the accuracy of the procedure to replace the POS tag with a verb tag. However, this method proved to be detrimental for to-do tasks containing homonyms.

4.2.2 Build the Rule Based Model

The Stanford POS tagger performed with an F-measure of 0.88. A rule set classifier (Frank and Witten, 1998) was created using WEKA[6] (Hall et al., 2009) brought the F-measure up to 0.92 using the following features in addition to the features created in bootstrap model described in Section 4.2.1:

a) the POS tags of the first token from both models:
   - a_1) the Stanford POS tag
   - a_2) the first verb model POS tag (see Section 4.2.1)

b) sentences containing one word

c) NER token spans greater than 1.

After creating the rule based model as just described, we applied the rule based model to correct the first POS tag of all of the utterances in the corpus using the Stanford Parser. The input to this step are the utterances and annotations from the corpus and the output is a tree of parsed items, which are used as features for the IA and argument.

[5] `https://wordnet.princeton.edu`
[6] `http://www.cs.waikato.ac.nz/~ml/weka/`

$_{\text{agent=buy}}$Get new $_{\text{item}}$sweater for $_{\text{person}}$John before $_{\text{holiday}}$Christmas.

Figure 1: Example semantic role labeled utterance

token	pos	prop-bank	dep	head-dep	function-tag
Get	VB	get.01	-	-	-
new	JJ	-	dobj	-	-
sweater	NN	-	root	A1	PPT
for	IN	-	dobj	-	-
John	NNP	-	prep	-	-
before	IN	-	root	-	AM-TMP
Christ-mas	NNP	-	prep	-	-

Table 3: Semantic Role Labeler example classification for *"Get new sweater for John before Christmas"* (see Figure 1).

5 Build the Classification Models

In this section we cover the techniques to process to-do tasks, classify the IA, and perform argument extraction.

5.1 Argument Extraction

The argument model is trained first as the IA model uses its labels as features.

A semantic role labeler (SRL) was used for many of the features in the argument model. There are one or more uses of a verb depending on the context, and each use of the verb has its own argument set. For this work, Proposition Bank (Palmer et al., 2005) was used for the operative verb of the utterance.

The SRL and POS tagging of the example given in Figure 1 is shown in Table 3. The `head-dep` label and `function-tag` represent how a parent and child token node are related in a head dependency tree (De Marneffe et al., 2006). This example's columns are further explained in Table 4.

The IA model was trained and tested (see Section 7) using WEKA on a per-IA basis. The ClearNLP[7] (Choi, 2014) semantic role labeling parser that was trained on Proposition Bank verb classes was used to provide features to the argument model. These feature's descriptions are listed in Table 4.

All feature set permutations were used in a ten-fold cross validation with the highest performing sets listed in Table 5. Results are given by IA in Table 8, which do not include techniques that failed (i.e. raw word vectors for the word w_n).

[7] http://github.com/emorynlp/nlp4j

Feature	Description
dep	SRL dependency of parent, which is always relative to a root verb
head-dep	Proposition Bank argument (i.e. `A1`)
propbankid	hashcode of the Proposition Bank identified by the semantic role labeler (i.e. `buy.01`)
list-type	the term list attribute (i.e. `name`, `attribute`)–see Section 4.1
ner-tag	Stanford's NER entity (i.e. `PERSON`, `ORGANIZATION`)
pos	w_n POS tag (i.e. `VB`, `NN`)
prev-pos	w_{n-1} POS tag
next-pos	w_{n+1} POS tag
stopword	if w_n a stop word–common/filler words with little meaning (i.e. `for`, `the`, `a`)
tm-ner-tag	*Taskmatch* NER list entity (i.e. `HOLIDAY`))
next-tm-ner-tag	w_{n+1} *Taskmatch* NER list entity
index	the token's index into the sentence (for word w_n token-index=n)

Table 4: Argument features where w_n is the Nth word in the to-do task utterance.

5.2 Intelligent Agent Classification

The main verb feature is the head root node of the dependency tree. If there is no parsed head node the POS tag of the first token of the utterance is used. For example, *"Buy swimsuit"* correlates closely to the `buy` IA. The main verb was the first feature when creating the model, and by itself, provided an impressive accuracy 55% by itself.

Set Name	Features	Description
pos	pos, next-pos, prev-pos	current and surrounding part of speech
NER/list/dep	ner-tag, index, list-type, dep	Stanford Parser named entity, the term list list and SRL dependency
head-dep	head-dep	Proposition Bank argument
TNER	tm-ner-tag, next-tm-ner-tag	*Taskmatch* current and next word named entity
stop	stopword	common/filler words with little meaning

Table 5: Argument feature sets (see Table 4 for more each feature's description).

We used the lemmatized form of the token for word count and cosine similarity features. Let c_{wa} = `Count(w,a)` be the count of word

Id	Classifier	Features	Precision	Recall	F1
1	Baseline	N/A	0.10	0.31	0.15
2	LogitBoost	*verb* + *TNER*	0.57	0.55	0.51
3	NearestNeighbor	CS_s + *verb* + *TNER* + *NER* + WC_a	0.56	0.57	0.56
4	LogitBoost	WC_a + *verb* + *TNER*	0.67	0.66	0.65
5	LogitBoost	CS_s + *verb* + *TNER*	0.68	0.67	0.67
6	BayesNet	CS_s + *verb* + *TNER* + *NER* + WC_a	0.67	0.66	0.65
7	LogitBoost	CS_s + *verb* + *TNER* + *NER* + WC_a	0.70	0.70	**0.69**

Table 6: Intelligent Agent classification results on Corpus A. The main verb is denoted as *verb*, *Taskmatch NER* as *TNER*, WC_a and CS_s is given in Section 5.2.

w for IA a and C_a be the set of word counts per IA such that $c_{wa} \in C_a$. We limit C to contain the highest n frequency counts with $n = |C_a|$ and hold n constant for all IAs as a hyper parameter. We use the word count aggregation across C_a as feature:

$$WC_a = \sum_{c \in C_a} c \tag{1}$$

Significant performance gains were achieved by increasing n from 5 to 15 with the WC_a feature. Now we define a mapping from word to a word distribution over C marginalizing over the word frequency:

$$q(w, a) = \frac{c_{wa}}{WC_a} \tag{2}$$

For example, for the `buy` IA utterances *"Purchase a shirt. Iron shirt."*: $C_{\text{buy}} = \{c_{\text{purchase}} = 1, c_{\text{a}} = 1, c_{\text{iron}} = 1, c_{\text{shirt}} = 2\}$ and $q(purchase, \texttt{buy}) = 1/4$, $q(a, \texttt{buy}) = 1/4$, $q(iron, \texttt{buy}) = 1/4$, $q(shirt, \texttt{buy}) = 2/4$.

In addition, word vector cosine distance (Mikolov et al., 2013) was calculated with Word2vecJava (Ko, 2015) using the the English Wikipedia pre-trained word vector data set (Mahoney, 2006). The word vector library was used by summing over the token cosine similarity and weighting it with the word frequency distribution from equation 2.

The cosine similarity feature is created by calculating the MLE across all IAs A to create cosine similarity (CS) for each sentence S:

$$CS_s = \underset{a \in A}{\text{argmax}} \sum_{w_c \in C_a} \sum_{w \in S} q(w_c, a) \cdot \texttt{cos}(w_c, w_s)$$

This feature contributed to a 5 point increase in F-measure in all results reported in Table 6.

6 An Illustrative Example

To illustrate how to-do tasks are handled we will use the example *"new christmas sweater for john"*. Once we receive the utterance the following happens:

1. Tokenize and sentence chunk, and POS tag the utterance using the modified version of the Stanford POS Tagger.
2. Create a head tree and tag tokens with Proposition Bank data using the semantic role labeler.
3. Classify IA as *"buy"* using the word counts, named entities and word vectors (see Section 5.2).
4. For each token in the utterance using the parameter model for IA *"buy"* classify an argument "sweater" as *"item"*, "john" as *"person"* and "christmas" as *"holiday"*.

Classifier	Features	Precision	Recall	F1
LibSVM	CS_s + *verb* + *TNER* + *NER* + WC_a	0.67	0.57	0.53
NearestNeighbor	CS_s + *verb* + *TNER* + *NER* + WC_a	0.57	0.56	0.56
RandomForest	CS_s + *verb* + *TNER* + *NER* + WC_a	0.66	0.66	0.66
J48	CS_s + *verb* + *TNER* + *NER* + WC_a	0.73	0.69	0.68
PART	CS_s + *verb* + *TNER* + *NER* + WC_a	0.69	0.69	0.69
NaiveBayes	CS_s + *verb* + *TNER* + *NER* + WC_a	0.72	0.69	0.70
LogitBoost	CS_s + *verb* + *TNER* + *NER* + WC_a	0.72	0.71	**0.71**

Table 7: IA classification results on Corpus B.

Intelligent Agent	Classifier	Features	Precision	Recall	F1
school-work	NaiveBayes	pos + NER/list/dep + head-dep	0.52	0.56	0.53
self-improvement	J48	pos + NER/list/dep + head-dep	0.60	0.59	0.56
plan-meal	IBk	pos + NER + list + dep + head-dep	0.66	0.65	0.64
find-travel	LogitBoost	pos + index + TNER + list + dep + head-dep	0.69	0.70	0.69
buy	SMO	pos + index + TNER + list + dep + head-dep	0.72	0.73	**0.72**
calendar	BayesNet	pos + index + TNER + list + dep + head-dep	0.73	0.73	0.72
postal	NaiveBayes	pos + NER/list/dep + head-dep	0.64	0.74	0.69
email	RandomForest	pos + index + TNER + list + dep + head-dep	0.77	0.77	0.77
contact	NaiveBayes	pos + NER + list + dep + head-dep	0.77	0.77	0.77
print	DecisionTable	pos + index + TNER + list + dep + head-dep	0.81	0.79	0.79
find-activity	NearestNeighbor	pos + index + TNER + list + dep + head-dep	0.81	0.79	0.79
search	J48	pos + NER + list + dep + head-dep	0.82	0.81	0.80
text-sms	NBTree	pos + NER + list + dep + head-dep	0.85	0.84	0.84
find-service	Bagging	pos + index + TNER + list + dep + head-dep	0.85	0.84	0.85
call	KStar	pos + NER + list + dep + head-dep	0.86	0.87	0.86
pay-bill-online	NBTree	pos + index + TNER + list + dep + head-dep	0.88	0.88	**0.88**

Table 8: Argument classification results per IA on Corpus A. See Table 5 for feature descriptions.

5. Concatenate contiguous tagged tokens of same argument type. In this example there are none, but if the example used "blue sweater", both would be tagged as *"item"* and be returned as one argument.

7 Results

Since testing was exhaustive, only noteworthy performance results for the IA and argument models are given. All subsets of reported features along with hyperparameter tuning was tried. The IA classification and baseline results are given in Table 6 and argument classification results are given in Table 8. Note that the argument classification results are for arguments and classify over all IAs, including the low count IAs shown in Table 1.

7.1 Intelligent Agent Classification

The baseline was created from the majority IA class (see Table 6).

A ten-fold cross validation was used on the IA model. Many classifiers and feature combination sets were tested. Additive Logistic Regression using Decision Stump Boosting[8] (Friedman et al., 1998) had the highest performance. The $\tilde{\chi}^2$ was computed between all classifiers with #7 showing a significant performance improvement increase over #2 - #4 with $p < 0.01$ in Table 6.

Results from the Corpus B (see Section 3.2) show a very similar pattern to those of Corpus A as shown in Table 7.

7.2 Argument Classification

The argument classification results are given in Table 8 for each respective IA and show a wide F-measure variance. The model was trained and tested over a high variance of argument occurrences as shown in Table 1 with some IAs covering many more annotations than others (i.e. "Buy" was the majority IA consisting of 28.5% of the task annotations). Another reason for the wide distribution in results is the ambiguous nature of some IAs. For example, `self-improve` could be anything from *study*, *school work* or *physical exercise*.

[8] `http://weka.sourceforge.net/doc.dev/weka/classifiers/meta/LogitBoost.html`

8 Conclusion

Classifying to-do tasks with good performance from upstream parsed data is a tractable problem. Using Argument extraction to aide in automating to-do task items is possible using the methods outlined in this work.

Bootstrapping methods for a NER with product lists using semi-supervised methods was used by Putthividhya, Pew and Junling (2011). Similarly, there is sufficient motivation by using our NER lists for exploring generation of entities using similar methods.

We focused on the IA and argument, but more work is needed to classify a category of the task, which identifies the theme of the action or its product attribute (Ghani et al., 2006) as a node in product taxonomy (i.e. *"buy dress"* $\{dress\} \rightarrow (apparel, women)$).

References

Victoria Bellotti, Brinda Dalal, Nathaniel Good, Peter Flynn, Daniel G Bobrow, and Nicolas Ducheneaut. 2004. What a to-do: studies of task management towards the design of a personal task list manager. In *Proceedings of the SIGCHI conference on Human factors in computing systems*, pages 735–742. ACM.

Satchuthananthavale RK Branavan, Harr Chen, Luke S Zettlemoyer, and Regina Barzilay. 2009. Reinforcement learning for mapping instructions to actions. 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 1-Volume 1*, pages 82–90. Association for Computational Linguistics.

Angel X Chang and Christopher D Manning. 2014. Tokensregex: Defining cascaded regular expressions over tokens. Technical report, Technical Report CSTR 2014-02, Department of Computer Science, Stanford University.

Jinho D Choi. 2014. *Optimization of natural language processing components for robustness and scalability*. Ph.D. thesis, University of Colorado Boulder.

Open Product Data. 2016. Open product data. http://product-open-data.com.

Marie-Catherine De Marneffe, Bill MacCartney, Christopher D Manning, et al. 2006. Generating typed dependency parses from phrase structure parses. In *Proceedings of LREC*, volume 6, pages 449–454.

Anthony Fader, Stephen Soderland, and Oren Etzioni. 2011. Identifying relations for open information extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1535–1545. Association for Computational Linguistics.

Jenny Rose Finkel, Trond Grenager, and Christopher Manning. 2005. Incorporating non-local information into information extraction systems by gibbs sampling. In *Proceedings of the 43rd Annual Meeting on Association for Computational Linguistics*, pages 363–370. Association for Computational Linguistics.

Joseph L Fleiss. 1971. Measuring nominal scale agreement among many raters. *Psychological bulletin*, 76(5):378.

Eibe Frank and Ian H. Witten. 1998. Generating accurate rule sets without global optimization. In *Fifteenth International Conference on Machine Learning*, pages 144–151. Morgan Kaufmann.

J. Friedman, T. Hastie, and R. Tibshirani. 1998. Additive logistic regression: a statistical view of boosting. Technical Report Vol. 28, No. 2, The Annals of Statistics, Stanford University.

Rayid Ghani, Katharina Probst, Yan Liu, Marko Krema, and Andrew Fano. 2006. Text mining for product attribute extraction. *ACM SIGKDD Explorations Newsletter*, 8(1):41–48.

Yolanda Gil, Varun Ratnakar, Timothy Chklovski, Paul Groth, and Denny Vrandecic. 2012. Capturing common knowledge about tasks: Intelligent assistance for to-do lists. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 2(3):15.

Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H Witten. 2009. The weka data mining software: an update. *ACM SIGKDD explorations newsletter*, 11(1):10–18.

Bo Han and Timothy Baldwin. 2011. Lexical normalisation of short text messages: Makn sens a# twitter. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies-Volume 1*, pages 368–378. Association for Computational Linguistics.

Jiepu Jiang, Ahmed Hassan Awadallah, Rosie Jones, Umut Ozertem, Imed Zitouni, Ranjitha Gurunath Kulkarni, and Omar Zia Khan. 2015. Automatic Online Evaluation of Intelligent Assistants. In *Proceedings of the 24th International Conference on World Wide Web*, WWW '15, pages 506–516, Republic and Canton of Geneva, Switzerland. International World Wide Web Conferences Steering Committee.

Stephen R Jones and Peter J Thomas. 1997. Empirical assessment of individuals' personal information management systems. *Behaviour & Information Technology*, 16(3):158–160.

Ruth Kempson, Ronnie Cann, Eleni Gregoromichelaki Arasheshghi, and Matthew Purver. 2015. Ellipsis. In *The Handbook of Contemporary Semantic Theory*, chapter 4. John Wiley & Sons.

Tracy Holloway King, Richard Crouch, Stefan Riezler, Mary Dalrymple, and Ronald M. Kaplan. 2003. The PARC 700 dependency bank. In *Proceedings of 4th International Workshop on Linguistically Interpreted Corpora (LINC-03) at EACL 2003*.

Paul Kingsbury and Martha Palmer. 2002. From treebank to propbank. In *LREC*, pages 1989–1993.

Andrew Ko. 2015. Word2vecJava. https://github.com/medallia/Word2VecJava.

Stephen C Levinson. 1983. Pragmatics (cambridge textbooks in linguistics).

Matt Mahoney. 2006. Rationale for a large text compression benchmark.

Mitchell Marcus, Grace Kim, Mary Ann Marcinkiewicz, Robert MacIntyre, Ann Bies, Mark Ferguson, Karen Katz, and Britta Schasberger. 1994. The penn treebank: annotating predicate argument structure. In *Proceedings of the workshop on Human Language Technology*, pages 114–119. Association for Computational Linguistics.

Philipp Meerkamp and Zhengyi Zhou. 2017. Boosting information extraction systems with character-level neural networks and free noisy supervision. In *Proceedings of the 2nd Workshop on Structured Prediction for Natural Language Processing*, pages 44–51.

Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. 2013. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

George A Miller. 1995. Wordnet: a lexical database for english. *Communications of the ACM*, 38(11):39–41.

T. Mitchell, W. Cohen, E. Hruschka, P. Talukdar, J. Betteridge, A. Carlson, B. Dalvi, M. Gardner, B. Kisiel, J. Krishnamurthy, N. Lao, K. Mazaitis, T. Mohamed, N. Nakashole, E. Platanios, A. Ritter, M. Samadi, B. Settles, R. Wang, D. Wijaya, A. Gupta, X. Chen, A. Saparov, M. Greaves, and J. Welling. 2015. Never-ending learning. In *Proceedings of the Twenty-Ninth AAAI Conference on Artificial Intelligence (AAAI-15)*.

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

Eric Prudhommeaux and Andy Seaborne. 2006. Sparql query language for rdf. w3c working draft, 4 october 2006. *SPARQL Query Language for RDF*.

Duangmanee Pew Putthividhya and Junling Hu. 2011. Bootstrapped named entity recognition for product attribute extraction. In *Proceedings of the Conference on Empirical Methods in Natural Language Processing*, pages 1557–1567. Association for Computational Linguistics.

Gokhan Tur, Anoop Deoras, and Dilek Hakkani-Tür. 2014. Detecting out-of-domain utterances addressed to a virtual personal assistant. In *Fifteenth Annual Conference of the International Speech Communication Association*.

Wikidata. 2015. Q5488768 — wikidata. http://www.wikidata.org/w/index.php?title=Q5488768&oldid=8079009.

A Corpus Sources

Corpus Ato-do list corpus was taken from the following locations:

- http://msippey.tadalist.com/lists/public/155420
- https://wiki.itap.purdue.edu/display/INSITE/Ta-Da+List+Research
- https://www.rememberthemilk.com/help/?ctx=basics.publish.publishlistpublic

The public to-do list corpus was taken from the following location:

- https://trello.com