Identification of Naturally Occurring Numerical Expressions in Arabic

Nizar Habash and Ryan Roth

Center for Computational Learning Systems, Columbia University
New York, NY, USA
{habash,ryanr}@ccls.columbia.edu

Abstract

In this paper, we define the task of *Number Identification* in natural context. We present and validate a language-independent semi-automatic approach to quickly building a gold standard for evaluating number identification systems by exploiting hand-aligned parallel data. We also present and extensively evaluate a robust rule-based system for number identification in natural context for Arabic for a variety of number formats and types. The system is shown to have strong performance, achieving, on a blind test, a 94.8% F-score for the task of correctly identifying number expression spans in natural text, and a 92.1% F-score for the task of correctly determining the core numerical value.

1. Introduction

Numerical Expressions (NUMs) permeate throughout text and speech presenting special challenges to natural language processing (NLP) applications. In addition to the typical issues NLP faces such as word ambiguity or morphological complexity, NUM challenges include: (a) their representing an infinite set of possible expressions and (b) their being represented in multiple script forms: digits (e.g., *168,000*), multi-word sequences (e.g., *one hundred and sixty-eight thousand*), or a mix of both (e.g., *168 thousand*). Different NLP applications have different needs when it comes to NUMs. In machine translation, source language NUMs may be normalized into a digital form that is used to generate them appropriately in the target language. However, speech recognition may expect the use of a language model that does not contain any digits, or where those digits have been consistently converted to word forms. And similarly, text-to-speech applications need to convert digits or mixed forms to words. Therefore, to be able to process NUMs in real contexts, we need tools to (a) determine the NUM span, and (b) convert its content into a normalized, digit-based form. Generating NUMs from a normalized form may be also needed but is an easier task by comparison.

In this paper, we define the task of Number Identification (NUMID) in natural context as including span determination and number normalization. We discuss the complexities of this task and we present a language-independent approach to quickly building a gold standard for evaluating NUMID by exploiting aligned parallel data. We also describe and evaluate a rule-based system for NUMID in Arabic.

In the next section, we describe previous work on NUMID. In Section 3., we describe the linguistic issues particular to Arabic NUMs. In Section 4., we describe a language-independent approach to building a gold NUMID tagged corpus. Finally, we present a rule-based system for NUMID for Arabic and we evaluate its performance in Sections 5. and 6., respectively.

2. Previous Work

Most of previous work on NUMID have focused on out-of-context number conversion from word form to digit form and vice versa. Such work has been done for many languages and is exclusively rule-based. Examples include English (Sproat, 2000), Swedish (Sigurd, 1973), Finnish (Karttunen, 2006) and, of special relevance here, Arabic (Al-Anzi, 2001; Dada, 2007). Particularly impressive is Bringert (2004)'s Numeral Translator, a demo applet which uses the Grammatical Function (GF) interpreter and numerals grammar to translate word NUMs among over 80 languages. However, in addition to being purely out-of-context number conversions, these approaches suffer in robustness: they often do not handle even small variations of input number format, perhaps because the authors have chosen to keep their system grammars small. For example, Bringert (2004)'s Numeral Translator can translate *twenty-five thousand six* into *[25006]*, but cannot parse *twenty-five thousand* ***and*** *six* or *twenty five thousand six* (hyphen removed), which are relatively common variations.

One exception to the above type of research is the MUC NUMEX effort on identification of number expressions in natural contexts. However, the NUMEX guidelines on NUMs are restricted only to monetary expressions and percentages (Grishman and Seundheim, 1996) and do not include normalization into a digit form.

In the work presented here, we extend the definition of NUMs in natural contexts to include multiple types including forms such as ordinals (e.g., [10th]) and plurals (e.g., [10s]). We also crucially extend the research on NUMID by including an evaluation of this task in naturally occurring contexts. Given the lack of gold standards for the NUMID task, we describe and validate an approach to create a gold standard using word-aligned parallel data. Finally, we describe a relatively simple rule-based approach to Arabic NUMID and evaluate it extensively. Our approach is designed to be robust in natural contexts by allowing (a) a mix of digits and words and (b) a wide range of simple variations in word and mixed NUMs.

3. Arabic Number Expressions

Compared to English, Arabic has a complex number system that interacts heavily with its complex morphology. Arabic numbers can vary by gender, definiteness and case and can take on a variety of clitics (prepositions, articles and even pronouns), e.g., سبعة *sbςħ* [1] 'seven' السبعة *Alsbςħ* 'the seven' and ولسبعتهم *wlsbςthm* 'and for the seven of them' are NUMs with the normalized value [7]. Moreover, Arabic verbs, nouns and adjectives inflect for number in addition to other features (Habash and Rambow, 2005). Arabic orthography uses *optional* diacritical marks which, if present, would help in disambiguating some NUMs, e.g., خمس *xms* can be خُمس *xums* 'one-fifth' or خَمس *xams* 'five'. Al-Anzi (2001) presents a good description of the word-based cardinal number system in Arabic (out of natural context). Dada (2007) also discusses in more detail issues of Arabic number-noun agreement, which we will not discuss here as they primarily pertain to generation, whereas our focus is on recognition (analysis). In naturally occurring data, we face additional issues not restricted to Arabic only: (a) ambiguity of words that can be numbers or non-numbers, e.g., ثانية *θAnyħ* is the noun 'second (unit of time)' and the NUM [2nd] (same ambiguity as in English); (b) use of a mix of digits and words to construct NUMs; and (c) variation of forms (orthographic alternatives, spelling errors, dialectal forms, and ungrammatical constructions). Additionally, we encounter other forms of numbers besides cardinal numbers such as ordinals and fractions.

In the work presented here, we address the forms of NUMs represented in Table 1. We handle some of the issues of form variations, but we leave part-of-speech (POS) and lexical ambiguity resolution to future work. We do not mark the morphological number of nouns (singular, dual or plural). Since our NUMID task for Arabic is intended for naturally-occurring text, we do not expect the text to be tokenized or diacritized in any way. However, we make use of some low-level morphological knowledge in our system (Section 5.).

4. Building a Gold Standard for Number Identification

A crucial resource to the evaluation of any tagging system is a *gold standard* to measure performance against. We did not have a ready-to-use gold standard for NUMID. And since building a gold standard can be a time-consuming effort, we describe here an approach for semi-automatically annotating a text corpus with NUMIDs by exploiting parallel corpora in the same spirit of work on projection over alignments (Yarowsky et al., 2001). We use a manually-aligned corpus of Arabic-English sentences from IBM (IBMAC)[2] (Ittycheriah and Roukos, 2005) although we believe the approach can be extended with some added noise to automatically-aligned data. In our approach, we *do not* include any information about Arabic numbers. The approach itself is language-pair independent, but the implementation needs some resources for the *other* language – in our case, English.

In the IBMAC corpus, Arabic punctuation and digits are mapped into their English equivalents.[3] Alif hamzated forms are normalized and so are Alif-maqsura and Yaa. All diacritics and tatweel/kashidas are removed. The reported inter-annotator agreement is around 93%. The data contained no explicit NUMID information other than marking the presence of digits. In one subset of the data (afa.align and treebank.align), digits were identified, e.g., 33.5 is represented as $num{33.5}. In the rest of the data (annahar.align.fw and ummah.align.fw), the digits were replaced with the token $NUM, and as such were not of use to us. In the subset of data we use (afa.align and treebank.align), there were 8818 sentences total, which we divide into test and devtest in Section 6.. An example of the information present in IBMAC is shown in Table 2. The first two columns specify the word positions (WPos) of words aligned together in the Arabic and English sentences. The last two columns show the actual aligned words.

Starting with the aligned data, we mark all adjacent English NUM words, e.g., *2.8* and *million* in the example in Table 2. We then use the alignments from English to Arabic to identify the spans of NUMs in Arabic. We also project back from Arabic to English, thus expanding the pairs of adjacent words referring to NUMs in Arabic and English. Of the identified sequences, we exclude all pairs where the English span ends up including non-number words. This ensures that dual nouns in Arabic (among other things) are excluded. We normalize the English NUM to a digit form using an English number normalization script we developed, and assign the normalization as the value of the Arabic NUM.

After this automatic step, we ran a manual check on all unique identified Arabic spans and values to verify well-formedness. The checks for spans and values were done independently. The manual corrections included correcting values, redefining spans, and removing incorrect spans resulting from ambiguous English terms, e.g., *half* (*part of a football match*) in *during the first half*, which can translate into the Arabic شوط *šwT* (not a number). The manual changes affected less than 10% of the automatic decisions.

To validate the quality of the gold standard, we manually corrected a sample of 400 sentences (containing 370 NUMs). Comparing the original sample to the corrected one, we achieve 100% precision and 98.4% recall (99.2% F-1). Most of the untagged cases responsible for the re-

[1] All Arabic transliterations are provided in the Habash-Soudi-Buckwalter transliteration scheme (Habash et al., 2007). This scheme extends Buckwalter's transliteration scheme (Buckwalter, 2002) to increase its readability while maintaining the 1-to-1 correspondence with Arabic orthography as represented in standard encodings of Arabic, i.e., Unicode, CP-1256, etc. The following are the only differences from Buckwalter's scheme (which is indicated in parentheses): Ā آ (|), Â أ (>), ŵ ؤ (&), Ǎ إ (<), ŷ ئ (}), ħ ة (p), θ ث (v), ð ذ (*), š ش ($), Ď ظ (Z), ς ع (E), γ غ (g), ý ى (Y), ã ً (F), ũ ٌ (N), ĩ ٍ (K).

[2] We thank IBM for making their hand-aligned data available to the research community.

[3] Eastern Arab countries such as Syria and Egypt use Indo-Arabic digits ٠١٢٣٤ *01234*, while western Arab countries such as Morocco and Tunisia use Arabic digits *01234* (same as those used in the West).

Type	Arabic		Value
Digits	14	14	14
Decimals	5,5	5,5	5.5
Percentages	82%	82%	82%
Number Words	سبعة	*sbςħ*	7
Number Strings	ثلاثمئة وواحد واربعين	*θlAθmŷħ wwAHd wArbςyn*	341
Mixed	63 الف	63 *Alf*	63000
Mixed	6.6 ملايين	6.6 *mlAyyn*	6600000
Mixed	12 بالمئة	12 *bAlmŷp*	12%
Ordinals	الثالث والستين	*AlθAlθ wAlstyn*	63rd
Plurals	المئات	*AlmŷAt*	100s
Simple Fractions	ثلثي	*θlθy*	2/3

Table 1: Examples of common NUM forms in naturally-occurring Arabic.

وقد تخطي عدد المتضررين من الاعصار ال 2,8 مليونا .			
wqd txTy ςdd AlmtDrryn mn AlAςSAr Al 2,8 mlywnA .			
The number of people harmed by the hurricane has surpassed 2.8 million .			
Arabic WPos	**English WPos**	**Arabic Phrase**	**English Phrase**
1	9	وقد *wqd*	has
2	10	تخطي *txTy*	surpassed
3	1,2,3	عدد *ςdd*	the number of
4	4,5	المتضررين *AlmtDrryn*	people harmed
5	6	من *mn*	by
6	7,8	الاعصار *AlAςSAr*	the hurricane
7	-1	ال *Al*	e_0
8	11	2,8 *2,8*	2.8
9	12	مليونا *mlywnA*	million
10	13	. *.*	.

Table 2: Example of a sentence aligned in the IBM hand-aligned corpus.

call error result from the Arabic number being aligned to a non-number in English, e.g. جانب واحد *jAnb wAHd* '*one* side' is translated as 'unilaterally.' This high degree of correctness makes us feel comfortable that this resource can be used to evaluate our NUMID performance.

5. Arabic Number Identification

In this section, we describe our algorithm for Arabic NUMID. The algorithm is divided into two stages: Span Identification (**SpanID**) and Number Normalization (**NumNorm**). An example of the NUMID process is shown in Figure 1. The **SpanID** step identifies word sequences that refer to NUMs, making sure to split apart any distinct NUMs that happen to be adjacent to each other. Upon completion, each of the marked spans is passed to the **NumNorm** step, which determines the numerical value associated with each span. Next, we describe the number lexicon we use. We follow it with a description of the two NUMID stages, **SpanID** and **NumNorm**.

5.1. Number Expression Lexicon

Both parts of the algorithm make use of a specially constructed lexicon, which lists over 400 forms of Arabic numerical terms and their associated values. Lexicon entries may include the definite determiner +ال *Al+* 'the'. However, we exclude other affixes, namely, the conjunctions +و *w+* 'and' and +ف *f+* 'so/then'; and the prepositions +ل *l+* 'to/for', +ب *b+* 'in/with' and +ك *k+* 'as/such'. The large

Type	Arabic		Value
Numbers	سبعة	*sbςħ*	[7]
Teens	ستة عشر	*stħ ςšr*	[16]
Plurals	عشرينات	*ςšrynAt*	[20s]
Dual Forms	مائتان	*mAŷtAn*	[200]
Scale Value	الف	*Alf*	[1000]
Scale Value	مليونا	*mlywnA*	[1000000]
Scale Value	مليارات	*mlyArAt*	[1000000000]
Non-numerics	فاصلة	*fASlħ*	[.]
Non-numerics	بالمئة	*bAlmŷħ*	[%]
Simple Fractions	نصف	*nSf*	1/2

Table 3: Examples of lexicon entries.

Input:

تواصل شركة XYZ النجاح المتميز الذي شهدته في فصليها الاول والثاني بافتتاحها اليوم الاثنين فصلها الثالث بقيمة 7.68 مليون يورو بارتفاع اكثر من 15 بالمئة ليصل سعر اسهمها الى خمسة وعشرين يورو .

twASl šrkħ XYZ AlnjAH Almtmyz Alðy šhdth fy fSlyhA AlAwl wAlθAny bAfttAHhA Alywm AlAθnyn fSlhA AlθAlθ bqymħ 7.68 mlywn ywrw bArtfAς Akθr mn 15 bAlmŷħ lySl sςr AshmhA Aly xmsħ wςšryn ywrw .

After Span Identification (SpanID):

twASl šrkħ XYZ AlnjAH Almtmyz Alðy šhdth fy fSlyhA <**num**>**AlAwl**</**num**> <**num**>**wAl*θ*Any**</**num**> *bAfttAHhA Alywm AlAθnyn fSlhA* <**num**>**Al*θ*Al*θ***</**num**> *bqymħ* <**num**>**7.68 mlywn**</**num**> *ywrw bArtfAς Akθr mn* <**num**>**15 bAlmŷ*ħ***</**num**> *lySl sςr AshmhA Aly* <**num**>**xms*ħ* wςšryn**</**num**> *ywrw* .

After Number Normalization (NumNorm):

twASl šrkħ XYZ AlnjAH Almtmyz Alðy šhdth fy fSlyhA <**num value="1st"**>**AlAwl**</**num**>
<**num value="2nd"**>**wAl*θ*Any**</**num**> *bAfttAHhA Alywm AlAθnyn fSlhA*
<**num value ="3rd"**>**Al*θ*Al*θ***</**num**> *bqymħ* <**num value="7680000"**>**7.68 mlywn**</**num**>
ywrw bArtfAς Akθr mn <**num value="15%"**>**15 bAlmŷ*ħ***</**num**> *lySl sςr AshmhA Aly*
<**num value="25"**>**xms*ħ* wςšryn**</**num**> *ywrw* .

Figure 1: Example of the NUMID process. Spans identified in the **SpanID** step are indicated with XML tags without *value*. The output of **NumNorm** places the determined values of NUMs inside the XML tags.

number of forms is necessary to accommodate the variety of ways in which NUMs are expressed in Arabic. Table 3 shows a few example lexicon entries.

We define a *scale value* to be a numerical term referring to a value of thousands, millions, billions, etc. Scale values offer important indications of a NUM's total value because of the grammatical rules which restrict the order in which they can appear in a NUM. Both **SpanID** and **NumNorm** make use of these indications. Arabic possesses several ways of expressing scale, including special forms to indicate duals, few and many, e.g., مليون *mlywn* is used with 1, 100 and 1000 *million(s)*; مليونين *mlywnyn* means '2 million'; ملايين *mlAyyn* is used with *a few (3-10)* millions; and مليونا *mlywnA*[4] is used with *many (11 to 99)* millions. Each of these forms is represented in the lexicon.

One interesting *exclusion* from our lexicon is the word الاثنين *AlAθnyn* which can mean both 'the+two' or 'Monday'. In a preliminary analysis of the kind of errors we were getting in our devtest, we found that more than half of the precision errors were due to this word being incorrectly assigned the value [2], while it was invariably 'Monday'. As a result, we removed it from the lexicon. All the results reported in Section 6. (baselines and our system) use the same lexicon.

5.2. Span Identification

SpanID is a crucial process for identifying NUMs in context and is implemented here as a two-step procedure. First, a line of input Arabic text is scanned for numerical terms which have entries in the lexicon. Any sequence of one or more consecutive numerical terms is collected into a span. Terms which are already digital (e.g., *1.234*) and ancillary terms such as فاصلة *fASlħ* 'a decimal point' are included as well. At the end of this stage, the algorithm will have generated a collection of potential NUM spans. We have one ad hoc rule that deletes a span around the word واحد *wAHd* when it is not part of a bigger number span; appearing by itself, this word is often ambiguous as '1', 'one/someone' or the adjective 'common/unique'. Since this is an ad hoc rule, it fails in some cases (Section 6.3.).

It is possible that the input text will have two or more NUMs adjacent to each other. If normalization is attempted on these cases, the result is that the two NUMs will be combined into a single, erroneous value. For this reason, we examine each span prior to normalization and split it if necessary. Span splitting is done by sequentially checking each word (**Y**) in the span for indicators of its connection with the preceding word (**X**); if no such indicator is found, the word is assumed to start a separate NUM, and the span is split between **X** and **Y**. The connection indicator rules we use are:

1. **Y** has a *wa*+ 'and' prefix, e.g., ثمانية وثلاثين *θmAnyħ wθlAθyn* '8 and 30' is [38]. This rule is ignored when both **X** and **Y** are *ordinal*, e.g., الاول والثالث *AlAwl wAlθAlθ* 'first and+third' is [1st] [3rd] not [4th].

2. **Y** is a decimal point or percentage sign (or words referring to those punctuation symbols), e.g., ستة بالمائة *stħ bAlmAŷħ* '6 %' is [6%].

3. **Y** follows a decimal point word, e.g., تسعة فاصلة ثلاثة *tsςħ fASlħ θlAθħ* '9 . 3' is [9.3].

4. **X** and **Y** together refer to a number between *11* and *19* (inclusive), e.g., سبعة عشر *sbςħ ςšr* '7 10' is [17].

5. **Y** is part of a string of "digit" words that follow a decimal, e.g., صفر فاصلة ثلاثة سبعة سبعة تسعة *Sfr fASlħ θlAθħ sbςħ sbςħ tsςħ* '0 . 3 7 7 9' is [0.3779].

[4]This is the singular indefinite accusative form of the word مليون *mlywn* 'million'. In Arabic this form is used as part of a special construction called تمييز *tamyiyz* 'specification' with numbers between 11 and 99 (Dada, 2007).

6. **Y** is a scale value word, while **X** is not, e.g., خمسين مليونا *xmsyn mlywnA* '50 1000000' is [50000000].

5.3. Number Normalization

After the spans are determined, they are normalized, i.e., translated into digital form. Prior to doing this, however, the span is examined for the presence of decimal points and percent words. In the case of decimals, the sections of the span before and after the decimal will be translated separately, and their output combined into single value afterward. Note that numbers following the decimal can be expressed as digits (e.g., ثلاثة فاصلة واحد خمسة *θlAθħ fASlħ wAHd xmsħ* [3.15]) or (less commonly) as a regular number (e.g., ثلاثة فاصلة خمسة عشر *θlAθħ fASlħ xmsħ ςšr* [3.15]); either format can be handled. If the span has a percent symbol or expression, the span is normalized without it, and a [%] is appended to the final value.

Once decimals and percentages are dealt with, the span can be translated. Often a span will consist of only one or two words; one common format is <*digits*><*scale word*> (e.g., 33.6 مليونا *33.6 mlywnA* [33600000]). For such simple cases, normalization only requires replacing numerical terms with their numerical values and multiplying/adding as appropriate. However, we must also consider the (much rarer) cases where a number is written out expressly, leading to a potentially large span.

Input:			
مائة واربعة عشر الفا واثنين وثمانين			
mAŷħ wArbςħ ςšr AlfA wAθnyn wθmAnyn			
Normalization Sequence:			
(100) (wa) (4) (10) (1000) (wa) (2) (wa) (80)			

Index	Number Type	Value	Stack
Initial	—	0	Empty
(100)	Number	100	Empty
(wa)	wa	0	100
(4)	Number	4	100
(10)	Number	14	100
(1000)	Scale	114000	Empty
(wa)	wa	0	114000
(2)	Number	2	114000
(wa)	wa	0	114000 2
(80)	Number	80	114000 2
End	End of Sequence	**114082**	Empty

Table 4: Example of number normalization. The **Value** and **Stack** columns show the contents of those variables after each element in the normalized sequence is examined. This input results in a returned value of [114082].

Table 4 shows an example of how such as span is processed. The normalization algorithm first replaces each word in the span with its associated value from the lexicon. Flags are also inserted into the span wherever the +و *wa+* 'and' prefix is used. Note that if any of the words in the span are ordinal or plural (as in [10s]) in form, the entire span is assumed to refer to an ordinal or plural value.

After this step, the span consists of a normalized sequence of numerical values, occasionally separated by *wa* flags. The algorithm then initializes two variables: a variable to hold the output value (**Value**) and a value stack (**Stack**). After initialization, each element of the sequence is examined in turn. Depending on the type of the element, one of several actions is taken:

1. If the element is a simple (non-scale) number, the element is added to **Value**.
2. If the element is *wa*, **Value** is pushed onto **Stack** and **Value** is set to zero.
3. If the element is a scale value, a value is popped from **Stack** (if it isn't empty) and is added to **Value**. **Value** is then multiplied by the scale value.
4. If the end of the sequence is reached, each element on **Stack** is popped and added to **Value**. The final **Value** is returned.

After the value is determined, it is adjusted as necessary to include ordinal or plural information if the original span was in those formats (e.g., [10] is changed to [10th] or [10s] as needed). Decimal recombination and percent appending is also performed.

6. Evaluation

We divide the gold data (8818 sentences) created in Section 4. into two sets. The first set, devtest, consists of 2,250 sentences containing 2,197 NUMs. The second set, test, consists of 6,568 sentences containing 6,236 NUMs. We use devtest to debug our system and do our error analysis. But test is kept hidden and only evaluated once. We evaluate our system on three tasks: **SpanID** (how accurately the system identifies which words form NUMs), Core-Match value determination (**Core-Match**), and Full-Match value determination (**Full-Match**). **Full-Match** is the most strict, in that it requires that the expression returned by the system match the gold expression perfectly in value, type and span. **Core-Match** only requires that the numerical values and spans match, and ignores type and other variations such as ordinal and plural markers, leading zeros, and percent symbols. For example, [3rd] and [3] would not match under **Full-Match**, nor would [02] and [2] or [15%] and [15]. Each of these would be counted as correct matches under the **Core-Match** task criteria.

In addition to the evaluation tasks, we define several methods of NUM span selection. These baselines show the importance of proper span identification. The simplest baseline (**Digits Only**) only tags digits as NUMs. **One Word** separates every numerical term into its own, one-word span. **Single Span** allows for multiple-word spans, but does not perform any span splitting to distinguish adjacent NUMs. Each of these baselines supersedes the previous; that is, a number expression correctly tagged by **Digits Only** will be correctly tagged by **One Word** and **Single Span**. The full system (as described in Section 5.) is labeled as **Our System**.

The precision, recall and F-scores for each combination of evaluation task and span selection method are shown in Table 5 and Table 6 for the devtest and test data sets, respectively.

devtest	SpanID			Core-Match			Full-Match		
Span Selection Method	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
Digits Only	88.4	60.9	72.1	85.5	58.9	69.8	84.8	58.5	69.2
One Word	77.2	86.2	81.5	75.1	83.8	79.2	73.8	82.4	77.8
Single Span	95.6	96.6	96.1	94.7	95.7	95.2	92.6	93.5	93.1
Our System	96.0	97.6	96.8	95.1	96.7	95.9	93.0	94.5	93.8

Table 5: **P**recision, **R**ecall and **F**-scores for each evaluation task and method of span selection (devtest set).

test	SpanID			Core-Match			Full-Match		
Span Selection Method	**P**	**R**	**F**	**P**	**R**	**F**	**P**	**R**	**F**
Digits Only	92.3	67.4	77.9	90.0	65.8	76.0	86.7	63.4	73.3
One Word	79.3	90.9	84.7	77.5	88.9	82.8	74.1	85.0	79.2
Single Span	92.6	96.0	94.3	90.0	93.3	91.6	86.1	89.3	87.7
Our System	92.7	96.9	94.8	90.2	94.2	92.1	86.2	90.1	88.1

Table 6: **P**recision, **R**ecall and **F**-scores for each evaluation task and method of span selection (test set).

6.1. Span Selection Method Comparison

The **Digits Only** baseline shows that significant percentage of the NUMs in both the devtest and test sets were already digital, and thus need no translation. The **Digits Only** method allows for relatively high precision (it is difficult to translate NUMs which are already digital incorrectly), but low recall. Note that the test set has a higher proportion of purely-digital numbers than the devtest set, and for this reason the **Digits Only** baseline actually performs better on test. The F-score steadily increases with each subsequent baseline, and implies that the types of NUMs encountered often consist of a single word or number, and that NUM spans occur next to each other somewhat infrequently. **Our System** provides significantly better F-score numbers on both data sets than any of the baseline span selection methods.

6.2. Evaluation Task Comparison

The high scores for the **SpanID** task provide an upper bound on the performance of the other two evaluation tasks, since if the span is not correctly identified, its value cannot be correctly deduced. Likewise, **Core-Match** is an upper bound on **Full-Match**. Considering **Our System**, the difference between **Core-Match** and **Full-Match** performance is not very large; this implies that the system does well in preserving non-digital information such as ordinality. In every case, **Our System** has better recall than precision, which indicates that it is more likely to tag a non-number expression as a NUM (a false positive) than to miss an existing NUM (a false negative). This is likely due to the significant number of Arabic terms which have multiple interpretations, some of which are numerical and some of which are not.

6.3. Error Analysis

In this section, we present an error analysis of all the errors in the devtest **Core-Match** evaluation. As shown in Table 5, our system attains high scores on this task: 95.1 precision (4.9 error) and 96.7 recall (3.3 error). We classified the errors into two categories by source as *gold errors* (i.e., resulting from bad gold standard) or *system errors* (i.e., resulting from bad system performance). Overall error contributions are presented in Table 7 in the rows marked *Total*. Precision errors are almost equally divided between gold and system (with system slightly larger). The majority of recall errors are gold based accounting for 2.58% (absolute) error compared with 1.6% recall error obtained in validation in Section 4.. More precision errors in gold are seen in the devtest (2.29% absolute) compared with validation check (none).

We further classify the different errors into four categories: **Wrong Value** (but *correct span*), **Wrong Span** (*over part of a valid* NUM), completely **Missed** (*valid*) NUM and incorrectly **Added** (*invalid*) NUM. Detailed error contributions are in Table 7.

Among gold errors, wrong values were the main culprit for both precision and recall. Examples of wrong values are

Gold Errors	**% of Precision Errors**	**% of Recall Errors**
Total	**46.8**	**78.1**
Wrong Value	25.7	38.4
Wrong Span	3.7	2.7
Missed NUM	17.4	-
Added NUM	-	37.0
System Errors	**% of Precision Errors**	**% of Recall Errors**
Total	**53.2**	**21.9**
Wrong Value	5.5	6.8
Wrong Span	3.7	5.5
Missed NUM	-	9.6
Added NUM	44.0	-

Table 7: Distribution of error types for the **Core-Match** task using the **Best System** on devtest. Gold Errors are caused by problems in the gold standard itself. System Errors are failures of our system. Precision errors indicate cases where the system predicted a NUM where the gold did not mark. Recall errors are the reverse.

cases where the numbers appearing in the English translation used to derive the gold standard were simply mistranslated or incorrectly inverted, e.g., *1700* being mapped to *17000* or *1973* to *3791*.

Gold missed and added NUMs are next. Examples of missed valid NUMs include unhandled roman numerals, e.g., *II*. Examples of added invalid NUMs include cases of the adjective واحد *wAHd* 'common/unique' which is commonly ambiguous with the number 'one'. These errors will feed into a revised version of the gold standard in the future by adding rules for handling missed cases and incorporating POS information if possible for added cases. Wrong spans were present in a very small number of cases.

As for system errors, the largest errors in precision and recall are added and missed NUMs, respectively. Invalid added NUMs are over 4 times as common as missed NUMs. Examples of common incorrectly added NUMs are the proper names باول *bAwl* 'Powell' and ثاني *θAny* 'Thani' which are ambiguous with the numbers '(in_)first' and 'second', respectively. Missed NUMs were primarily the word واحد *wAHd*, which we do not handle when it appears by itself since it *often* is not a number but rather a noun meaning 'someone' or an adjective meaning 'common/unique'. Wrong value errors result from spelling errors in the input data, e.g., ثلات *θlAt* instead of ثلاث *θlAθ* 'three'; or ambiguous cases such as خمس *xms* meaning 'five' or 'one-fifth'. POS ambiguity and lexical ambiguity are the main problems in our current system. We plan to address these problems by incorporating lexical and POS information in the future. Wrong span errors were again the smallest contributors to overall errors.

7. Conclusions

In this paper, we defined the task of *Number Identification* in natural context. We also presented and validated a language-independent semi-automatic approach to quickly building a gold standard for evaluating number identification systems by exploiting hand-aligned parallel data. Finally, we presented and extensively evaluated a robust rule-based system for number identification in natural context for Arabic for a variety of number formats and types. The system is shown to have strong performance, achieving, on a blind test, a 94.8% F-score for the task of correctly identifying number expression spans in natural text, and a 92.1% F-score for the task of correctly determining the core numerical value.

8. Future Work

In the future, we plan to improve the process to creating the gold standard. We will compare automatically aligned data to hand-aligned data and consider other languages. We also plan to build more sophisticated number identifiers that exploit POS and lexical resources to handle ambiguous cases in a manner similar to work in morphological disambiguation (Habash and Rambow, 2005), named entity recognition (Farber et al., 2008) and phrase-base chunking (Diab et al., 2004).

Acknowledgments

This work was funded under the DARPA GALE program, contract HR0011-06-C-0023.

9. References

Fawaz S. Al-Anzi. 2001. Sentential Count Rules for Arabic Language. *Computers and the Humanities*, 35(2):153–166.

Björn Bringert. 2004. Numeral Translator. http://www.cs.chalmers.se/˜bringert/gf/translate/.

Tim Buckwalter. 2002. Buckwalter Arabic morphological analyzer version 1.0. Linguistic Data Consortium, University of Pennsylvania. LDC Catalog No.: LDC2002L49.

Ali Dada. 2007. Implementation of Arabic numerals and their syntax in GF. In *Proceedings of the 2007 Workshop on Computational Approaches to Semitic Languages*, pages 9–16, Prague, Czech Republic.

Mona Diab, Kadri Hacioglu, and Dan Jurafsky. 2004. Automatic Tagging of Arabic Text: From Raw Text to Base Phrase Chunks. In *Proceedings of the North American Chapter of the Association for Computational Linguistics (NAACL)*, Boston, MA.

Benjamin Farber, Dayne Freitag, Nizar Habash and Owen Rambow. 2008. Improving NER in Arabic Using a Morphological Tagger. In *Proceedings of Language Resources and Evaluation Conference (LREC)*, Marrakech, Morocco.

Ralph Grishman and Beth Seundheim. 1996. Design of the MUC-6 evaluation. In *TIPSTER Text Program Phase II Workshop*, Vienna, VA.

Nizar Habash, Abdelhadi Soudi and Tim Buckwalter. 2007. On Arabic Transliteration. In A. van den Bosch and A. Soudi, editors. *Arabic Computational Morphology: Knowledge-based and Empirical Methods*, Springer, 2007.

Nizar Habash and Owen Rambow. 2005. Tokenization, Morphological Analysis, and Part-of-Speech Tagging for Arabic in One Fell Swoop. In *Proceedings of Association for Computational Linguistics (ACL)*, Ann Arbor, Michigan.

Abraham Ittycheriah and Salim Roukos. 2005. A maximum entropy word aligner for Arabic-English machine translation. In *Proceedings of Empirical Methods in Natural Language Processing Conference*, pages 89–96, Vancouver, Canada.

Lauri Karttunen. 2006. Numbers and Finnish Numerals. *SKY Journal of Linguistics*, 19:407–421.

Bengt Sigurd. 1973. From numbers to numerals and vice versa. In *Proceedings of the International Conference on Computational Linguistics*, Pisa, Italy.

Richard Sproat. 2000. Lextools: a toolkit for finite-state linguistic analysis. http://www.research.att.com/˜alb/lextools/synth.pdf.

David Yarowsky, Grace Ngai, and Richard Wicentowski. 2001. Inducing multilingual text analysis tools via robust projection across aligned corpora. In *Proceedings of Human Language Technology Conference*, pages 161–168.