CSCI-B 659: APPLYING ML TECHNIQUES IN CL Assignment 1 - Vectorization and NLP

Khandokar Md. Nayem (knayem@iu.edu)

March 2, 2018

Task 1

Stanford CoreNLP Server launch

Stanford CoreNLP is a library of processing human language and extract useful linguistic features (e.g. properties and relations) from the natural languages. Stanford CoreNLP offeres a wide range of analysis tools, like parts of speech, whether they are names of companies, people, etc. (Named entities), normalize dates, times, and numeric quantities (Lemmas), mark up the structure of sentences in terms of phrases and syntactic dependencies (Dependency parse and Constituent parse), indicate which noun phrases refer to the same entities (Coreference) and more.

We launch Stanford CoreNLP server and access the interface activating the following annotators,

- Parts-of-speech
- Named entities
- Dependency parse
- Lemmas
- Constituent parse
- Coreference

The given sample sentence pair is,

> John met Susan in the mall. She told him that she is traveling to Europe next week.

This is the output we get from Stanford CoreNLP server.

— Text to annotate —

John met Susan in the mall. She told him that she is traveling to Europe next week.

— Annotations —

parts-of-speech × named entities × dependency parse × lemmas × constituency parse × coreference ×

— Language —

English

Submit

Part-of-Speech:

1	John	met	Susan	in	the	mall	.				
	NNP	VBD	NNP	IN	DT	NN					
2	She	told	him	that	she	is	traveling	to	Europe	next	week	.
	PRP	VBD	PRP	IN	PRP	VBZ	VBG	TO	NNP	IN	NN	

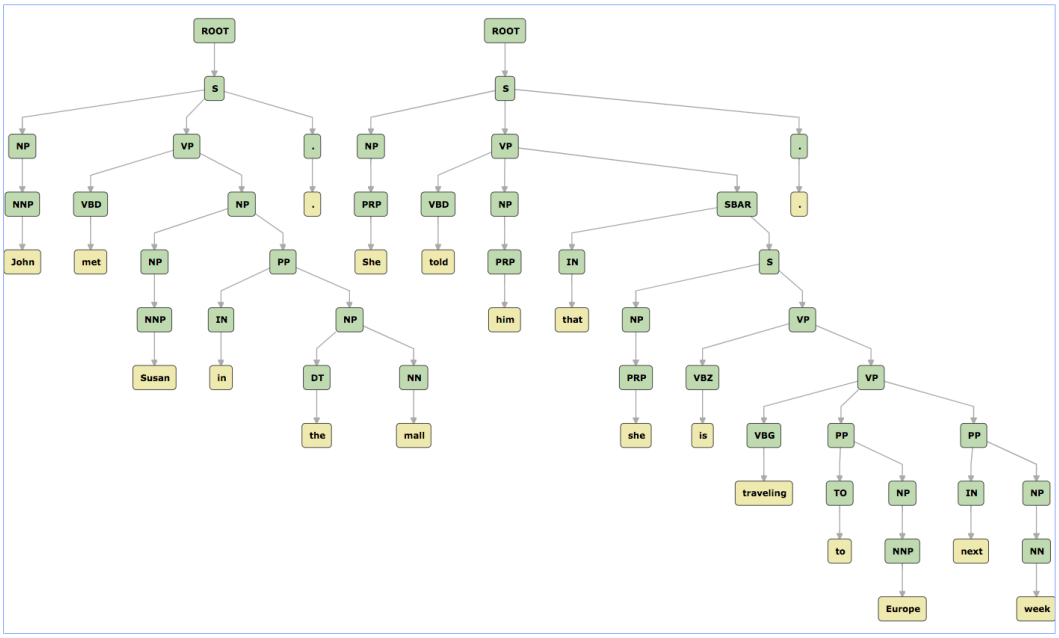
Lemmas:

1	John	met	Susan	in	the	mall	.				
	John	meet	Susan	in	the	mall					
2	She	told	him	that	she	is	traveling	to	Europe	next	week	.
	she	tell	he	that	she	be	travel	to	Europe	next	week	

Named Entity Recognition:

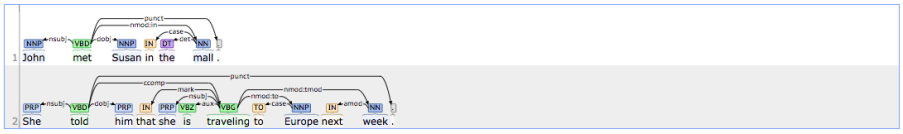
1	John	met	Susan	in	the	mall	.				
	PERSON		PERSON								
2	She	told	him	that	she	is	traveling	to	Europe	next	week	.
									LOCATION	DATE 2018-W10	

Constituency Parse:



Basic Dependencies:

Enhanced++ Dependencies:



Coreference:

Task using NLP features

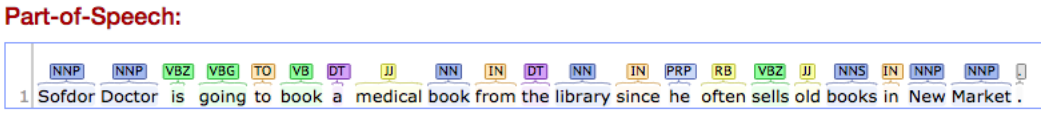
Now a days, most of the natural language related tasks use statistical analytic on natural language, but human language is inherently ambiguous and too much flexible to represent in simple statistical models. So using NLP features is a smart and logical way to model human language. We identify a specific task which will definitely be benefited from these features, that is **Machine Translation (MT)**. We want to translate one human language to another language without no intervention of human. To restrain ourselves, we want to translate English to Bangla using Machine Translation. Here is a sample english to bangla translation done by google translate.

This translation is quite wrong. The correct Bangla translation should be similar of সফদর ডাক্তার লাইব্রেরি থেকে একটি মেডিকেল বই ধরে রাখতে যাচ্ছেন, কারন তিনি প্রায়ই নিউ মার্কেটে পুরাতন বই বিক্রি করেন।

So linguistic (NLP) features are a must to achieve better accuracy and performance. We feel like features as Parts-of-speech, Named entities, Dependency parse, Lemmas, Constituent parse can be useful in machine translation. Sennrich R. and Haddow B. mention this in their work [RB15] and other works [NC17, CvG$^+$14]. Now we want to describe why we think that these features can be useful and the strategies to vectorize these features.

Parts-of-speech (POS)

Parts-of-speech gives a lot of information about a word and its neighbor words (nouns are preceded by determiners and adjectives, verbs by nouns) and about the grammatical structure around the word (nouns are generally part of noun phrases), which makes part-of-speech tagging an important component of parsing. In machine translation, parts-of-speech of the word in one language generally preseves in another language too. But word position varies due to different sentence making structure over language to language. So parts-of-speech will help to get the context of the word as same word can be used differently in the same sentence. Part-of-speech annotation of the above sentence is,

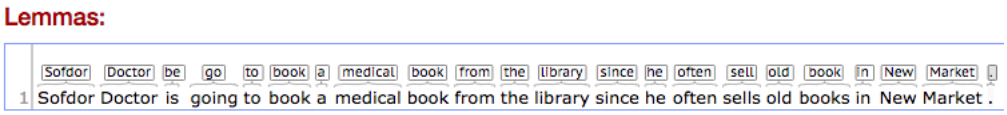


CoreNLP POS stategy In Stanford CoreNLP, there are 36 possible POS tag for an English word. So we make an one hot vector encoding of size 36 and put 1 in the corresponding POS tag of the vector. One important point is that, POS is a word level

feature. So to incorporate this in vectorizing a sentence, we add all the vectors into a single vector with size $(1, 36)$ which represents the count of POS for a sentence.

Lemma

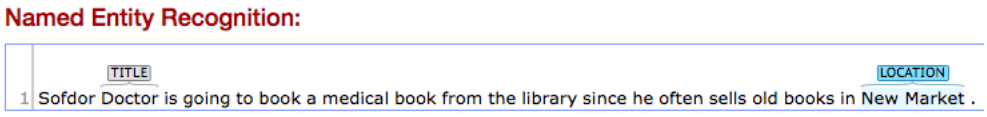
Lemma is important to know the base word over different words. Using lemma, models can learn inflectional variants are semantically related, and represent them as similar points in the continuous vector space. In machine translation, lemma will help us to use a word in various variants in other language which is very likely to be preserved. This will also help to minimize the dimension of words in a sentence. Lemma annotation of the above sentence is,



CoreNLP Lemma stategy In Stanford CoreNLP, we merge the words who share same lemma into one vector and create a single parts-of-speech vector for all such words. By this number of word vector is minimized, yet preserve the different POS tags for the word. Then we add all the word vectors into a single vector with size $(1, 36)$ which represents the count of POS and Lemma for a sentence.

Named entities

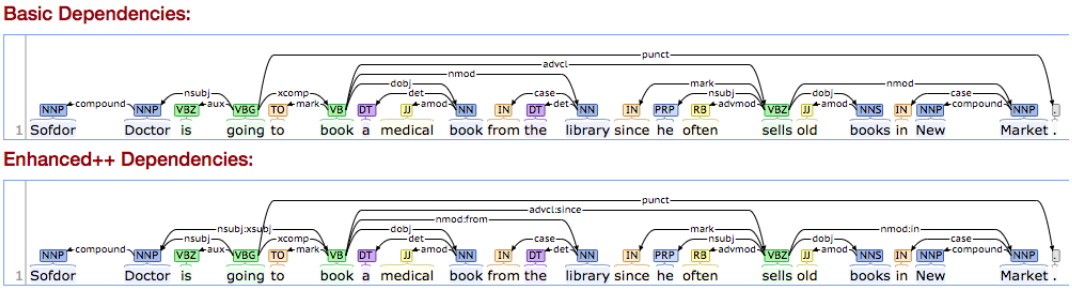
Named entities recognizes entities such as, person and company names, etc. in a sentence. Common named entities are used in Stanford coreNLP is PERSON, LOCATION, ORGANIZATION, MISC, MONEY, NUMBER, ORDINAL, PERCENT, DATE, TIME, DURATION, SET. It is useful in machine translation as it gives the entities which will be almost same for both the language. Such as the name of place would be same for all language. So it is an important feature for machine translation as it will reduce the complexity by directly mapping the named entities to the other language. Named entity annotation of the above sentence is,



CoreNLP Lemma stategy In Stanford CoreNLP, there are 23 possible named entities tag. So we make an one hot vector encoding of size 23. We know that an entity can be of multiple tokens (e.g. New Market, New York). So first we merge those entities in a single word. This will further minimize the word dimensional. Then put 1 in the corresponding named entities tag of the vector. Then we add all the word vector and created a sentence vector with size (1,23) which represents the count of all the named entities for that sentence.

Dependency parse

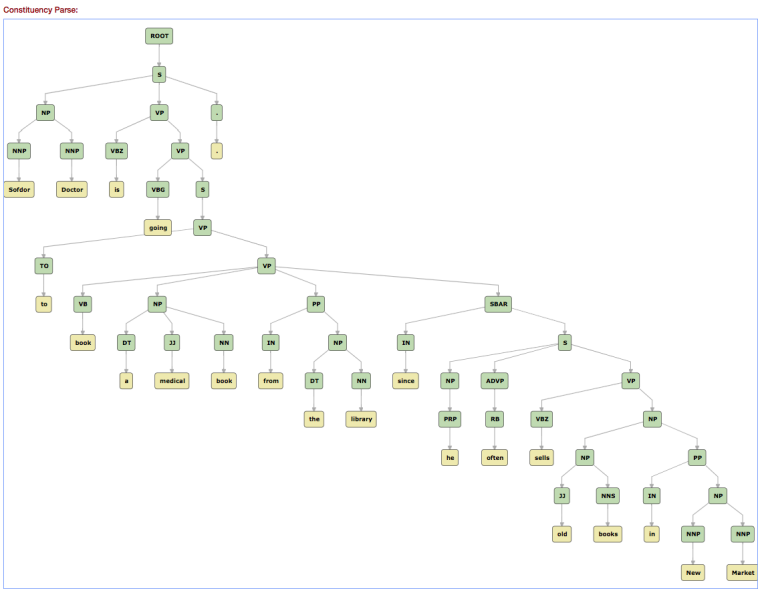
Dependency parse tree shows the word to word dependencies and relations which is very crucial in machine translation. Across language, dependencies between words are very unlikely to change, rather their order and grammar change. As an example in context of our above sentence, the dependency between subject and verb in English (e.g. *sells* — *(nsubj)* → *he*) will also preserve in Bangla (e.g. বিক্রি — (nsubj) → তিনি). Dependency parse annotation of the above sentence is,



CoreNLP Dependency parse stategy We extracted all the basic dependency relations (triples) between the words in a sentence. There are total 52 dependency relation generalized in CoreNLP. We create a zero vector of size 52 and count all each types of relations for all the words in the sentence. This is used as a feature vector for dependency relation of a sentence.

Constituent parse

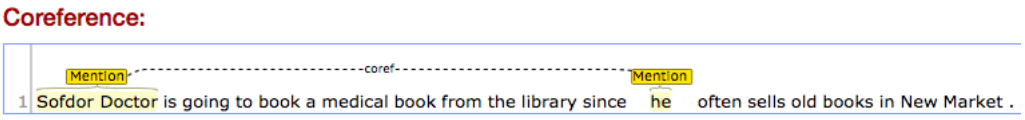
Constituent parse tree is another very important feature to break down a sentence in its grammatical structure. We believe grammatical structure changes over language to language but the construction of general tags remains same. As an example in context of our above sentence, a noun phase in English (e.g. *a medical book*) will interpreter as a noun phase in Bangla also (e.g. একটি মেডিকেল বই). Constituent parse annotation of the above sentence is,



CoreNLP Constituent parse stategy We extracted the basic constituent tags of a sentence. There are total 8 basic constituent generalized in CoreNLP. We create a zero vector of size 8 and count all each types of constituent present in the sentence. This is used as a feature vector for constituent of a sentence.

Coreference

Coreference is an important feature that relates persons ans sexs over sentences and complex structure. Coreference parse annotation of the above sentence is,



CoreNLP Coreference parse stategy In our specific machine translation model, where we want to translate to Bangla, this feature is not that important. Unlike French and many other language, pronoun of Bangla are unisex. As an example, সফদর ডাক্তার is a male person, but the coreferenced pronoun (shown in figure above) তিনি can be both male and female. So right now, we ignore this NLP feature in vectorization of a sentence.

Task 2

Spacy

Spacy is used to extract the features from the same sentence. In Spacy the features Constituent parse and Coreference are not avaiable. In Spacy we use the following

features,

- Parts-of-speech
- Named entities
- Dependency parse
- Lemmas

The given sample sentence pair is,

> John met Susan in the mall. She told him that she is traveling to Europe next week.

This is the output we get from Spacy.

```
Text    POS
-------------
John --->PROPN
met --->VERB
Susan --->PROPN
in --->ADP
the --->DET
mall --->NOUN
. --->PUNCT
She --->PRON
told --->VERB
him --->PRON
that --->ADP
she --->PRON
is --->VERB
traveling --->VERB
to --->ADP
Europe --->PROPN
next --->ADJ
week --->NOUN
. --->PUNCT
```

```
Text    Lemma
-------------
John --->john
met --->meet
Susan --->susan
in --->in
the --->the
mall --->mall
. --->.
She --->-PRON-
told --->tell
him --->-PRON-
that --->that
she --->-PRON-
is --->be
traveling --->travel
to --->to
Europe --->europe
next --->next
week --->week
. --->.
```

```
Text    Entity
-------------
John --->PERSON
Susan --->PERSON
Europe --->LOC
next week --->DATE
```

```
Text    Dependency
-------------
John --->nsubj
met --->ROOT
Susan --->dobj
in --->prep
the --->det
mall --->pobj
. --->punct
She --->nsubj
told --->ROOT
him --->dobj
that --->mark
she --->nsubj
is --->aux
traveling --->ccomp
to --->prep
Europe --->pobj
next --->amod
week --->npadvmod
. --->punct
```

Now we describe the strategy fo vectorization and compare the output for each feature of Spacy with Stanford coreNLP output.

Part-of-speech (POS)

Part-of-speech is important as we stated earlier. Here is the part-of-speech output of Spacy for the same sentence using in CoreNLP,

```
Text    POS
-------------
Sofdor --->PROPN
Doctor --->PROPN
is --->VERB
going --->VERB
to --->PART
book --->VERB
a --->DET
medical --->ADJ
book --->NOUN
from --->ADP
the --->DET
library --->NOUN
since --->ADP
he --->PRON
often --->ADV
sells --->VERB
old --->ADJ
books --->NOUN
in --->ADP
New --->PROPN
Market --->PROPN
. --->PUNCT
```

Spacy Part-of-speech stategy An one hot vector of size 19 with all the universal POS tag in English language for each word. For each word we detect the POS and put 1 in the corresponding block of the vector. Then to represent the sentence vector we add all the vectors into a single vector with size $(1, 19)$ which represents the count of POS for a sentence.

Difference in POS (Spacy vs CoreNLP) In Spacy we have only 19 POS compared to 36 POS of coreNLP. The output of Spacy is slightly different than Stanford coreNLP. In Spacy the word *"to"* is labeled as both ADP(adposition) and PART(particle) whereas in coreNLP it is labeled as 'TO' for both the time.

Lemma

In Spacy, the output of lemma would be like this:

```
Text    Lemma
-------------
Sofdor --->sofdor
Doctor --->doctor
is --->be
going --->go
to --->to
book --->book
a --->a
medical --->medical
book --->book
from --->from
the --->the
library --->library
since --->since
he --->-PRON-
often --->often
sells --->sell
old --->old
books --->book
in --->in
New --->new
Market --->market
. --->.
```

Spacy Lemma stategy Same procedure to vectorize lemma like CoreNLP is used in Spacy. We get the lemma for each token using the tag *token.lemma_*. We merge the words who share same lemma into one word and create parts-of-speech vector for each word here too. Then we merge all these one hot vector into a single vector to represent the whole sentence.

Difference in Lemma (Spacy vs CoreNLP) The output for lemma is same for both the Spacy and CoreNLP except for the word *"he"*. Spacy labeled it as PRON but coreNLP labeled it as HE. CoreNLP makes more sense in this regard. Also CoreNLP maintains case, where as Spacy lowercases all of the lemmas.

Named entities

The result of named entities in Spacy is below,

```
Text    Entity
-------------
Sofdor Doctor --->ORG
New Market --->GPE
```

Spacy Named entities stategy To vectorize the named entities, we take the vector of size 22 which is the number of named entities supported by Spacy. Then for each word we create one hot vector and put 1 into the corresponding named entity block if the word is a named entity other wise 0. Then we add all the word vector and created a sentence vector with size $(1, 22)$ which represents the count of all the named entities for that sentence.

Difference in Named entities (Spacy vs CoreNLP) Spacy gives 22 types of Named entities compared to 23 of Stanford coreNLP. From the list of the Spacy Named entities, we can see that the tags for named entities are different from Stanford coreNLP but they covered almost same types of entities.

Dependency parse

The output of Dependency parse in Spacy is,

```
Text    Dependency
-------------
Sofdor --->compound
Doctor --->nsubj
is --->aux
going --->ROOT
to --->aux
book --->xcomp
a --->det
medical --->amod
book --->dobj
from --->prep
the --->det
library --->pobj
since --->mark
he --->nsubj
often --->advmod
sells --->advcl
old --->amod
books --->dobj
in --->prep
New --->compound
Market --->pobj
. --->punct
```

Spacy Dependency parse stategy To vectorize dependency parse tree, we first extract the dependency relation between the words using *token.dep_* tag. we also get

the triples relation for each word from the dependency module. There are total 52 dependency relation and we create a zero vector of size 52 and count all each types of relations for all the words in a sentence. This is the feature vector for dependency relation of a sentence.

Difference in Dependency parse (Spacy vs CoreNLP) Basic Dependency parse of these both are identical. But enhanced++ Dependencies of CoreNLP gives more insightful dependencies. For our application, basic dependency seems good enough.

Task 3

We read JSON object from server and vectorize using coreNLP (code attached in separate file). Also we do same vectorization using Spacy (code attached in separate file).

By using CoreNLP, we make vectors of POS+Lemma $\rightarrow (1, 36)$, Entity $\rightarrow (1, 52)$, Dependency $\rightarrow (1, 52)$, Constituent $\rightarrow (1, 8)$, total $(1, 119)$ sized vector for each sentence.

By using Spacy, we make vectors of POS+Lemma $\rightarrow (1, 19)$, Entity $\rightarrow (1, 22)$, Dependency $\rightarrow (1, 52)$, total $(1, 93)$ sized vector for each sentence.

Task 4

We have uploaded separate files for Spacy vectors and fastText. And for fastText, we use supervised learning similiarity maching.

References

[CvG+14] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio. Learning Phrase Representations using RNN Encoder-Decoder for Statistical Machine Translation. *ArXiv e-prints*, June 2014.

[NC17] J. Niehues and E. Cho. Exploiting Linguistic Resources for Neural Machine Translation Using Multi-task Learning. *ArXiv e-prints*, August 2017.

[RB15] Sennrich R and Haddow B. Linguistic input features improve neural machine translation. 1:83–91, 2015.