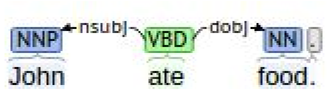
Approach

This document tries to explain the approach I have used to create the tool. The drawbacks of this approach and the possible improvements have also been discussed.

This is a very simple and easy to implement approach for extracting meaningful triplets limited to the form (subject - verb - object) from sentences. The tool makes use of dependency parsing to extract the entities.

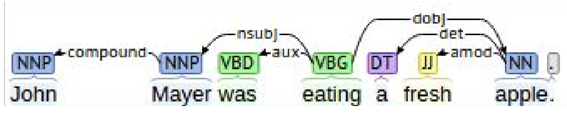
Let us go through some examples for better understanding. We will go on increasing the complexity of sentences.

1) **John ate food.**



This is a very simple sentence containing one verb/action/predicate “ate” and two arguments subject “John” and direct-object “food”. All the three entities contain only one word and verb is the root of the dependency tree. All the arguments are easily identifiable from the dependency labels. So the meaningful triplet would be (John, ate, food).
In this way, we make use of dependency labels to identify entities and their roles in the sentence, and then we take out all the subject-verb-object triples.

2) (John Mayer) (was eating) (a fresh apple) .



Here, all the entities subject, verb and object consist of multiple words. The three entities are (John Mayer), (was eating) and (a fresh apple). These group of words form a meaningful units or chunks or a phrases (NP,VP,NP) in terms of constituency parsing. If we consider just the words which are directly attached with the verb with relation subject and object, the triplet formed (Mayer, eating, apple) does not seem meaningful and ungrammatical. It also ignores a lot of meaningful information. The triplet we would like to get is (John Mayer, was eating, a fresh apple)

1) One way to extract these multiword entities, is to make use of constituency parser or a simple chunker. A constituency parser assigns the following structure to the sentence:

```
(ROOT
  (S
    (NP (NNP John) (NNP Mayer))
    (VP (VBD was)
      (VP (VBG eating)
        (NP (DT a) (JJ fresh) (NN apple))))))
```

If we consider the level above the POS tags, we get following phrases:

- NP: John Mayer

- VP : was eating a fresh apple
- VP: eating a fresh apple
- NP: a fresh apple

It is difficult to extract the entities we expected from this.
Whereas a syntactic chunker would produce exactly the same entities as we wanted:

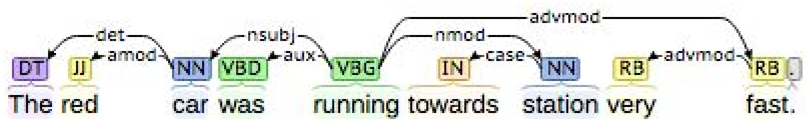
([John_NNP Mayer_NNP])
<: was_VBD eating_VBG :>
([a_DT fresh_JJ apple_NN])

But there are some limitations associated with the chunker, that we would get to see in the next example.

2) Our approach makes use of dependency relations to combine the related words to form a single meaningful entity.

- First we find the words which provide important information about the verb. In this example, “eating” is the main verb and “was” is the supporting/auxiliary verb which provides information about tense. We also join adverb with the main verb (though, whether to combine adverb with verb to form one entity depends on the application). We will see this in coming examples. Similarly for a noun, we combine the compounds, attach the determiner, adjectives and adverbs. All of these terms are identified by the dependency relation labels. (similar to the case of verb, combining adjective with noun is a choice of user, as the modifiers convey some additional information which may or may not be necessary in applications).
- Once we combine the words to get meaningful entities, we find their relations with the verb (subject, direct-object, nmod, etc) and find out meaningful triplets.

3) The red car was running towards station very fast.



Possible triplets:

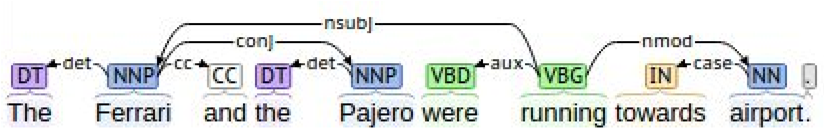
- The car, was running, towards station
- The red car, was running, towards station
- The car, was running very fast, towards station
- The red car, was running very fast, towards station

Different types of triplets are possible depending on the choice of adverb-verb adjective-noun strategy discussed in the previous example. Our tool provides only the the last triplet. It can be modified easily to produce other triplets also.

A chunker would produce the following chunks: (The red car) (was running) (towards station) (very fast).

One thing to notice in this example is that the main verb chunk and the adverb chunk, modifying the verb are separated by another chunk which is an object of the verb. So connecting them would not be possible by just looking at the chunks. Using dependency relations, we find out the adverb nodes (“advmod” relation with verb) and combine them with the verb to form “was running very fast”.

4) The Ferrari and the Pajero were running towards airport. (Conjunction in noun-subject)



Again, it is the matter of choice or sometimes depends on the context, if the nouns in conjunction should be separated or not. The approach we are following, is rule based. So any number of rules can be added to make the system sophisticated. But in general, it is not possible to write rules which are exhaustive to cover all the possibilities in a language.

In the above example, three possible triplets are:

- the Ferrari, was running, towards airport
- the Pajero, was running, towards airport
- the Ferrari and the Pajero, were running, towards airport

The selection of triplets depends on whether we want to keep the property of togetherness intact. In our approach, we combine such nouns to produce a single entity. It could be modified easily to behave differently, though.

Characteristics:

The system uses Universal Dependencies Labels while parsing and use them to combine words and form entities. There are around 40-45 types of relations in the universal dependency framework. The whole system is based around the rules created to handle this variety of dependency relations and all kinds of possible structures of sentences. The current system looks at only few important relations and a small set of grammatical structures. We have not considered complex structures like sentences with multiple verbs in conjunction or long sentences consisting of multiple clauses. As the rules are hard-coded, more rules can be added easily to handle complex cases.