MSAN
Summer 2017
Natural Language Processing

Topic 03a: Practical IE (NLTK and MITIE)

Overview

NER with NLTK

IE with MITIE
- Download and Setup
- Test
- Python Integration
- Start MITIE
- Extract Named Entities
- Perform relationship extraction

Overview

Since Information Extraction is an important topic, one of the three things we need to learn how to do well to be good data scientists, it is important that we learn how to do it well. Practically speaking, this means not implementing an IE system from scratch, as we've discussed, but starting with an existing implementation and extending it for our purposes. There are a few systems which we can use, including: Stanford CoreNLP, NLTK and MITIE. Since the Stanford toolkit is written in Java (and we don't expect this in MSAN), let's understand it's an option and focus on NLTK and MITIE.

NER with NLTK

NLTK contains an NER toolkit. The name of the function is "ne_chunk." It requires the POS tags, which itself requires tokenised input. Assuming we have a corpus in a variable called "document," we can chain all the functions in one line of python:

```
chunks = ne_chunk(pos_tag(word_tokenize(document)))
```

Let's use our toy example corpus. Print the "`chunks`" variable. What is in it? Technically, "ne_chunk" is a kind of parser — a tool which reconstructs the structure of a sentence — so the output is an instance of nltk.Tree with "S" as the normal root node. More on parsing in the future.

Finding the named entities in a sentence (or document or corpus) involves parsing the nodes in the nltk.Tree instance and parsing nodes which do not have a 'ROOT' label. If we want the variable "named_entities" to store a list of named entities, the following may be useful:

```
named_entities = []
for node in chunks:
    if type(node) is Tree:
        if node.label() != 'ROOT':
            named_entities.append([node.label(),
node.leaves()])
```

If useful, use the code above to read our toy corpus and print a list which looks like:

```
ORGANIZATION ==> CNN
PERSON ==> Vili Fualaau
ORGANIZATION ==> CNN
PERSON ==> Mary Kay Fualaau
...
```

Use ner-nltk.py if you need help. Questions:

1) Did NLTK work?
2) How well did it work? Specifically:
    a) Was it perfect?
    b) If not, what errors did it make? How should we quantify its performance?

IE with MITIE

MITIE is the MIT implementation of an Information Extraction tool. Implementation started approximately 3 years ago and has been updated regularly since. By default, it tags PERSON, LOCATION, ORGANIZATION and MISC. The current release includes tools for performing named entity extraction and binary relation detection as well as tools for training custom extractors and relation detectors. It is available via GitHub at https://github.com/mit-nlp/MITIE.

Download and Setup

Download MITIE from GitHub. From command line:

```
git clone https://github.com/mit-nlp/MITIE.git
```

Once downloaded, compile the source code into binaries. Since MITIE is written in C++, an older (but still useful) tool called “make” is used to compile it. From command line:

```
make
```

We will also need to get the English-language MITIE trained models:

```
make MITIE-models
```

(Models for Spanish and German are also available.)

Test the implementation by executing the following on command line:

```
cat sample_text.txt | ./ner_stream
MITIE-models/english/ner_model.dat
```

… and you should see something like the following:

```
Loading MITIE NER model file...

time: 1.98sec
Now running NER tool...
A [ORGANIZATION Pegasus Airlines] plane landed at an [LOCATION
Istanbul] airport Friday after a passenger " said that there was a
bomb on board " and wanted the plane to land in [LOCATION Sochi] ,
[LOCATION Russia] , the site of the Winter Olympics , said officials
with [LOCATION Turkey] 's [ORGANIZATION Transportation Ministry] .

[PERSON Meredith Vieira] will become the first woman to host [MISC
Olympics] primetime coverage on her own when she fills on Friday
night for the ailing [PERSON Bob Costas] , who is battling a
continuing eye infection . " It 's an honor to fill in for him , "
[PERSON Vieira] said on TODAY Friday . " You think about the [MISC
Olympics] , and you think the athletes and then [PERSON Bob Costas] .
" " [PERSON Bob] 's eye issue has improved but he 's not quite ready
to do the show , " [ORGANIZATION NBC Olympics] Executive Producer
[PERSON Jim Bell] told TODAY . com from [LOCATION Sochi] on Thursday
.

From wikipedia we learn that [PERSON Josiah Franklin] 's son ,
[PERSON Benjamin Franklin] was born in [LOCATION Boston] . Since
wikipedia allows anyone to edit it , you could change the entry to
say that [LOCATION Philadelphia] is the birthplace of [PERSON
Benjamin Franklin] . However , that would be a bad edit since [PERSON
Benjamin Franklin] was definitely born in [LOCATION Boston] .
```

Note that the output is in bracketed BIO format and punctuation is separated from text. In other words, it performs tokenisation and NER labelling.

Test

Let's test this against our toy example, mary_kay_and_vili.txt. If you haven't already downloaded this input file, do it now and run the MITIE NER tool against our example file:

```
cat mary_kay_and_vili.txt | ./ner_stream \
    MITIE-models/english/ner_model.dat
```

Let's answer the same questions we had about NLTK's NER:

1) Did MITIE work?
2) How well did it work? Specifically:
    c) Was it perfect?
    d) If not, quantify and categorise the errors.

Python Integration

The MITIE implementation comes with examples in several languages, python included. The examples are located in the ./python/ directory. If we are happy with the NER model's performance, we can use it in our python scripts. There are a few steps in the NER pipeline:

1) Start MITIE
    a) Instantiate the model
    b) Read the text
    c) Tokenise the input
2) Extract named entities
3) Perform relationship extraction

Let's discuss these individually.

Start MITIE

In order to start MITIE, we first need to point to the base directory and import from there:

```
sys.path.append('/PATH_TO/mitielib')
from mitie import *
```

Obviously, we have to change "PATH_TO" with the operating system directory containing the MITIE installation.

Once we've imported MITIE, we can instantiate one of the NER models (in this case, named "ner") with:

```
ner = named_entity_extractor \
    ('/PATH_TO/MITIE-models/english/ner_model.dat')
```

This particular model includes four NER tags: PERSON, LOCATION, ORGANIZATION and MISC. We can get a list of possible tags for this model with:

```
possible_tags = ner.get_possible_ner_tags()
```

After we load the NER model, we need to read the data to operate on. If the name of the file is FILE_NAME, we load the data with:

```
load_entire_file(FILE_NAME)
```

… which returns a string with the entire file.

Let's use this against our toy example file, mary_kay_and_vili.txt. However, since we only load the file to tokenise it, we bundle the above with "tokenize" and assign it to a list, called tokens, as in the following:

```
tokens = tokenize(load_entire_file(FILE_NAME))
```

… where FILE_NAME is the name of the file containing the text we care about.

Extract Named Entities

Once we have the list of tokens, we can perform NER with a call to "extract_entities" like the following:

```
entities = ner.extract_entities(tokens)
```

Once complete, "entities" contains a list of tuples, ordered by their appearance in the source document, each of the tuples containing (in order): range, tag, confidence score and the text in question. For example, the following code prints the text and NER label for the text in question:

```
for entity in entities:
    print ' '.join(tokens[i].decode() for i in entity[0]),
    print '=', entity[1]
```

Note that the our ranges for entities are 0-based *start position* and *end position + 1*. For example, in the table below, representing the tokenised sentence fragment, "My flight to Miami this afternoon." the named entity "Miami" is in range (3, 4). Keep this in mind for the next steps.

0	*1*	*2*	*3*	*4*	*5*	*6*
My	flight	to	Miami	this	afternoon	.

Table 1: Example sentence fragment for NER

MITIE also allows us to train a custom NER model if, for example, if we want to specialise to a set of organisations, create new label types (time? money?), etc. In order to do so, we need to:

1) Provide a training corpus tagged with the location and types of named entities.
2) Pass the training corpus to an ner_trainer instance and train using the corpus.

There is an example at the MITIE site. Here is the core of what we need:

```
sample = ner_training_instance(['My', 'name', 'is', 'Sam',
'at',\
   '2017-06-11', '.'])
sample.add_entity(xrange(3, 4), "person")
sample.add_entity(xrange(5, 6), "DTT")
trainer = ner_trainer(

"../../MITIE-models/english/total_word_feature_extractor.dat")
trainer.add(sample)
ner = trainer.train()
```

Let's use this to see if we can recreate some potentially interesting results. Train with the following sentences (the first one is above):

```
My name is [PER Sam] at [DTT 2017-06-11].
The other day at work I saw [PER Brian Smith] from [ORG USF].
```

… and test with the following:

```
Paramore was manufactured in 2002-09-09.
```

What do we see? What can we infer about the features used to predict named entities based on this result?

Perform relationship extraction

When we think of Information Extraction from text, we think of relationships as being defined by a triple (pattern) in the form: (X α Y), where X and Y are named entities and α is a string of words that intervenes between X and Y. MITIE uses this as a model. The premises behind performing MITIE relationships are that: relationships are binary and are defined by the tokens between entities. The implication of having binary relationships is that we can limit our relationships to pairs of entities. The implication of having the relationship defined by intervening tokens is that we can limit to neighbouring pairs of named entities — i.e. we do not need to check $O(n^2)$ relationship pairs.

MITIE has a limited number of relations already defined. These include:

- Author writes Book
- Director directs Film
- Inventor invents Invention
- Person (born in / died in / interred in) Place
- Person is (Ethnicity, Nationality, Religion)

… along with many others. Let's use one of them (Person born in Place) against the toy example "eliot.txt," which is the text version of the first portion of the Wikipedia entry for Eliot Spitzer.

Firstly, we need the neighbouring pairs of entities we found in our NER stage:

```
entities = ner.extract_entities( \
    tokenize(load_entire_file(FILE_NAME)))
```

Once we have this, we can load the proper model (person.place_of_birth) and create pairs of entities for comparison:

```
rel_detector = \
    binary_relation_detector( \
    "MITIE-models/english/binary_relations/ \
    rel_classifier_people.person.place_of_birth.svm")
adjacent_entities = [(entities[i][0], entities[i+1][0]) \
```

```
        for i in xrange(len(entities)-1)]
```

… but we want to allow our named entities to appear in any order for the binary relation, so:

```
    adjacent_entities += [(r, l) for (l, r) in adjacent_entities]
```

In other words, we make all pairs bi-directional. Let's print this to see the pairs we generate. (Because this is a "born in" relationship, we can call the entities `person` and `place` even though those variables may refer to any type of named entity.)

```
    for person, place in adjacent_entities:
        person_name = ' '.join(tokens[i].decode() for i in person)
        place_name  = ' '.join(tokens[i].decode() for i in place)
        print person_name, 'and', place_name
```

If the above prints all adjacent pairs, we may notice that this code compares all pairs of (hypothesis) entities, even when those are pairs of names of people. This is not an accident. Recall that it's possible to get the text of a named entity correct even when the type is incorrect. Let's move on. We can use MITIE's "`extract_binary_relation`" function to retrieve the intervening tokens and score them. We specifically invoke this on the `rel_detector` variable we instantiated earlier:

```
    for person, place in adjacent_entities:
        relation = ner.extract_binary_relation(tokens, person,
    place)
        score = rel_detector(relation)
```

Let's print the person names, place names and scores. (Here's mitie_entities.py if we want a reference or need to catch up, but note that the paths are customised to a particular environment.) Higher scores indicate better matches to the model. We will want to use our training data to establish reasonable cutoffs for scores. In practice, the cutoff scores will vary according to our needs for tolerance for precision and recall. Noting that, we probably never want to consider any relation which scores below 0.

MITIE allows us to train our own binary relationships. The relationships have the same form (X `α` Y) as the ones provided by MITIE. Ideally, we want to have many examples to train on. For now, let's just use one example and one test instance. Let's use an example from the recent news:

> *An Egyptian court has acquitted Aya Hijazi, her husband and six other charity workers of all charges, after they were imprisoned for more than three years without trial.*

We'll tokenise the sentence and assign it to a variable:

```
    train_sentence = ['An', 'Egyptian', 'court', 'has',
        'acquitted', 'Aya', 'Hijazi', ',', 'her', 'husband',
        'and', 'six', 'other', 'charity', 'workers', 'of', 'all',
        'charges', ',', 'after', 'they', 'were', 'imprisoned',
        'for', 'more', 'than', 'three', 'years', 'without',
'trial',
        '.']
```

To train our model, we'll first need an instance of the named_entity_extractor.

```
ner = named_entity_extractor \
    ('/PATH_TO/MITIE-models/english/ner_model.dat')
```

Then we'll create an instance of binary_relation_detector_trainer — an object to train the binary relations we've used before. We should name the relationship. (In harmony with the now-defunct Freebase knowledge base, we use "people.person.court_acquittal" as our name.) We also need an instance of the named_entity_extractor from above.

```
rel_trainer = binary_relation_detector_trainer( \
  'people.person.court_acquittal', ner)
```

We'll need to know where our entities are, so we identify them manually using the highlighted position in our example sentence. In this case, "Egyptian court" is in positions (1-3) and "Aya Hijazi" is in position (5-7). This becomes a positive example for our model.

```
rel_trainer.add_positive_binary_relation(train_sentence, \
    xrange(1,3), xrange(5,7))
```

We'll also want to provide one or more negative examples which will help the training. We can use the contra-positive relations as the negative examples.

```
rel_trainer.trainer.add_negative_binary_relation(train_sentence
,\
    xrange(5,7), xrange(1,3))
```

Once we have all of these, we can instantiate a relationship detector from a trained model of this binary relation:

```
rel_detector = rel_trainer.train()
```

And that's it. We can save our model to disk using the save_to_disk function.

```
rel_detector.save_to_disk("rel_acquittal_classifier.svm")
```

Once we have this altogether, let's test our model. Will use a similar event which made news some time ago. This sentence is different.

In 2015, the Supreme Court of Cassation definitively acquitted Amanda Knox.

Let's assign this to a list ("test_sentence"?) and print out our score for this example against our model. Again, we manually identify the named entities here, but this should be automated by our code.

```
print 'Detection score for Knox being acquitted:', \
    rel_detector(ner.extract_binary_relation(test_sentence,
    xrange(3, 4), xrange(9, 13)))
```

(Use mitie_relations.py if we need a reference or need to catch up.)

Questions:

1) How would you characterise the score for this sentence?
2) What if we used the (original Wikipedia) wording for the test sentence:
   In 2015, Knox was definitively acquitted by the Supreme Court of Cassation.
3) What can we do to increase the performance of our model?