cTAKES 3.0 - POS Tagger

{scrollbar} 65%Contents of this Page2 Menu cTAKES 3.0 to Include

Overview of POS Tagger

This project provides a UIMA wrapper around the popular OpenNLP part-of-speech tagger. The UIMA examples project provides a default wrapper from which we have borrowed liberally. We have created our own wrapper so that it will work better with our type system and to add features and supporting components. Additionally, both the OpenNLP package and the UIMA examples OpenNLP wrappers lack documentation for how to generate training data, build a part-of-speech tagging model, and build a tag dictionary. The latter in particular can be confusing if you are new to OpenNLP.

A part-of-speech tagging model is included with this project.

The model derives from a combination of GENIA, Penn Treebank (Wall Street Journal) and anonymized clinical data per Safe Harbor HIPAA guidelines. Prior to model building, the clinical data was deidentified for patient names to preserve patient confidentiality. Any person name in the model will originate from non-patient data sources.

Building a model

Obtaining training data

There are a variety of sources of part-of-speech data that may be useful for training a part-of-speech tagger. We have used the following three sources for training a part-of-speech tagger for clinical data:

Mayo Clinic part-of-speech corpus

This is a corpus owned and maintained by Mayo Clinic. Unfortunately, because of legal and privacy issues it is not currently available for distribution. However, a part-of-speech model based on this data is released.

If you'd like to use your own algorithms on the Mayo Clinic corpus, please contact clinicalnlp@mayo.edu for its availability.

GENIA

GENIA is a literature mining project in molecular biology from University of Tokyo. Its corpus, a collection of biomedical literature, has been annotated with POS tags. You can download a copy of its POS corpus version 3.02p that we used to build our model from Topics.

Penn Treebank

The Penn Treebank project annotates naturally-occurring text for linguistic structure. Penn Treebank also annotates text with part-of-speech tags. To obtain a copy of Release 2 from which we built our model, refer to Release 2.

Formatting training data

The format of a training data file expected by OpenNLP tools should have one sentence per line, with each "word" immediately followed by "_" and the word's part-of-speech tag, which is then followed by a space. An example snippet from one line of training data is shown below.
Example 4.2. POS tagger training data

the_DT stories_NNS about_IN well-heeled_JJ communities_NNS and_CC

We have provided a script to convert GENIA data to OpenNLP part-of-speech data. To create a training data file from the GENIA corpus:

- Remove the following lines from GENIAcorpus3.02.pos.xml:

line 2: <?xml-stylesheet type="text/css" href="gpml.css" ?>
line 3: <!DOCTYPE set SYSTEM "gpml.merged.dtd">
line 5: <import resource="GENIAontology.daml" prefix="G"></import>
java -cp*<classpath>*data.pos.training.GeniaPosTrainingDataExtractor GENIAcorpus3.02.pos.xml*<genia-pos-training-data>*

Where ***<genia-pos-training-data>*** is a file that the converted training data will be written into. For the few cases in Genia where tokens contain white space, these are simply ignored and not added to the training data file.

- We do not have scripts that we can share for converting Penn Treebank version 2 into OpenNLP-formatted training data. However, there are many libraries that are available that can be used to parse the Penn Treebank. Two suggestions are:
  - opennlp.tools.parser.ParserEventStream API from OpenNLP library
- Stanford parser

Another strategy is to take the output of the chunker training data as detailed in the section called
"Prepare Penn Treebank training data" from the Chunker component.
and convert it to the correct format.

What if my text contains underscores?
No problem. OpenNLP splits the word from the tag using the last underscore. However, there will be difficulties if your data uses an underscore as a part-of-speech tag.

What if I have a "token" that contains a space?
This is a problem. OpenNLP will not be able to handle a token that contains a space in it. GENIA, for example, contains 108 occurrences of spaces inside tokens. The white space must be removed from these tokens or ignored (see above).

Creating a model

java -cp *<classpath>* opennlp.tools.postag.POSTaggerME *<training-data> <model-name> iterations cutoff*
Where

- *<training-data>** is an OpenNLP training data file.
- *<model-name>** is the file name of the resulting model. The name should end with either .txt (for a plain text model) or .bin.gz (for a compressed binary model).
- *<iterations>** determines how many training iterations will be performed. The default is 100.
- *<cutoff>** determines the minimum number of times a feature has to be seen to be considered for inclusion in the model. The default cutoff is 5

The iterations and cutoff arguments are, taken together, optional, that is, you should provide both or provide neither.

Building a tag dictionary

One thing that can be confusing about the OpenNLP part-of-speech tagger is that there are two data structures with similar sounding names, Dictionary and TagDictionary. In short, the Dictionary construct is one that can and should be ignored while the TagDictionary is one that needs a bit of attention.

A tag dictionary is used when tagging text, not during the training of a POS model.

We have provided a mechanism for creating a tag dictionary. It can be run with the following command:
java -cp*<classpath>*edu.mayo.bmi.uima.pos_tagger.TagDictionaryCreator*<training-data><tag-dictionary>case-sensitive*
Where

- *<training-data>** is a file containing pos-of-speech tagged training data
- *<tag-dictionary>** the file name of the resulting tag dictionary
- *<case-sensitive>** is either 'true' or 'false' depending on whether the tag dictionary should be case sensitive or not.

For relevant material about the difference between Dictionary and TagDictionary refer to the following forum threads:

- https://sourceforge.net/forum/forum.php?thread_id=1720863&forum_id=9943
- https://sourceforge.net/forum/forum.php?thread_id=1894043&forum_id=9943
- DefaultPOSContextGenerator.getContext(int, Object[], String[])] API

OpenNLP provides a default tag dictionary for the English part-of-speech model called tag.bin.gz which can be downloaded from
}+http://opennlp.sourceforge.net/models/english/parser/tagdict+
. You should use this tag dictionary only if you are using the model from
+http://opennlp.sourceforge.net/models/english/parser/tag.bin.gz+.

If you want to use the tag dictionary in a case insensitive way, then entries in the tag dictionary which are not all lowercased will be ignored because the tag dictionary fails to lowercase entries read in from the file. It only lowercases the words that are compared against the dictionary when "CaseSensitive" is set to false. Therefore, if you want the tag dictionary to be used in a case insensitive way, be sure to build the tag dictionary using 'false' as the third argument.

Analysis engines (annotators)

POSTagger.xml

The file desc/POSTagger.xml provides a descriptor for the POSTagger analysis engine which is the UIMA component we have written that wraps the OpenNLP part-of-speech tagger. It calls "org.apache.ctakes.postagger.POSTagger", whose Javadoc API provides further information on the parameters of this descriptor.

Parameters

PosModelFile
the file that contains the part-of-speech tagging model

TagDictionary
the file that contains the tag dictionary (if available)

CaseSensitive
determines whether to use the TagDictionary in a case sensitive way or not.

POSTaggerAggregate.xml

The descriptor desc/POSTaggerAggregate.xml defines a pipeline for part-of-speech tagging that creates all the necessary inputs (e.g. token and sentence annotations). It needs the same three parameters as
POSTagger.xml
does.

POSTaggerCPE.xml

The file desc/POSTaggerCPE.xml provides an XML-specification of a collection processing engine (CPE).
Start UIMA CPE GUI.
java -cp<*classpath*>org.apache.uima.tools.cpm.CpmFrame

- Open this file.
- Set the parameters for the collection reader to point to a local collection of files that you want part-of-speech tagged.
- Set the parameters for the POSTagger as appropriate for your environment.
- Set the output directory of the XCAS Writer CAS Consumer.

The results of running the pipeline are written to the output directory as XCAS files. These files can be viewed in the CAS Visual Debugger.

Evaluating a POS tagger

There are two ways a POS tagger should be evaluated:
(1) Use gold standard tokens. Run the POS tagger using gold standard tokens and calculate the percentage of part-of-speech labels that have been correctly assigned.
An example is given in Example below called "Evaluate a POS tagger using gold standard tokens"
(2) Use generated tokens.

Evaluate a POS tagger using gold standard tokens

If this is gold standard sentence:
The_DT major_JJ inducible_JJ protein_NN complex_NN that_WDT binds_VBZ ._.

And if this is the output for that sentence:
The_DT major_JJ inducible_NN protein_NN complex_NN that_WDT binds_VBD ._.

By the second method, the accuracy should be 6/8 = 75%.

- Use tokenizer generated tokens
- Run the tokenizer and use this as input to the POS tagger.
- In this scenario, we calculate F-measure in the following way:

true positive (TP)
a token that has the correct boundary and part-of-speech label

false positive (FP)
a tagged token that does not have the correct boundary and/or part-of-speech label

- false negative (FN)
  a token in the gold standard data that was not correctly generated by the tokenizer/POS tagger

An example is given in

deeppink

Example below called "Evaluate a POS tagger using generated tokens"

Evaluate a POS tagger using generated tokens

If this is gold standard sentence:
This_DT complex_NN is_VBZ not_RB cyclosporin_JJ -sensitive_JJ ._.

And if this is the output for that sentence:
This_DT complex_JJ is_VBZ not_RB cyclosporin-sensitive_JJ ._.

TP = 4, FP = 2, and FN = 3
F-measure = (2 * recall * precision) / (precision + recall) = (2 * TP) /
(2*TP + FP + FN) = (2 * 4) / (2*4 + 2 + 3) = 8 / 13 = .615

In fact, if you do the evaluation this way for the "gold standard tokens" evaluation, then you will get the same answer as the accuracy calculation given above.