Writing a custom NameFinder model in OpenNLP

Though we have various NER models available in OpenNLP, but entity extraction doesn’t end here with the existing one only. We may need to find the entities based on Clinical, Biological, Sports, Banking domain etc.

So should we restrict ourselves with the models already provided?  - No,

We can build our own Name Finder model.  Steps required doing this are: Get the sample training dataset, build the model and test it.

**What type of data should we have for training a model:**

Sentences should be separated with new line character (\n).  Values should be separated from <Start> and <END> tags with a space character.

<START:medicine> Augmentin-Duo <END> is a penicillin antibiotic that contains two medicines - <START:medicine> amoxicillin trihydrate <END> and <START:medicine> potassium clavulanate <END>. They work together to kill certain types of bacteria and are used to treat certain types of bacterial infections.

You can refer a sample [dataset](https://github.com/mccraigmccraig/opennlp/blob/master/src/test/resources/opennlp/tools/namefind/AnnotatedSentencesWithTypes.txt) for example. Training data should have at least 15000 sentences to get the better results.

Model can be trained via command line tool as well as Java Training API :

**Command Line tool :**

There are various argument which you need to pass while building the model as follows :

$ opennlp TokenNameFinderTrainer

Usage: opennlp TokenNameFinderTrainer[.bionlp2004|.conll03|.conll02|.ad] [-resources resourcesDir] \

               [-type modelType] [-featuregen featuregenFile] [-params paramsFile] \

               [-iterations num] [-cutoff num] -model modelFile -lang language \

               -data sampleData [-encoding charsetName]

Arguments description:

        -resources resourcesDir

                The resources directory

        -type modelType

                The type of the token name finder model

        -featuregen featuregenFile

                The feature generator descriptor file

        -params paramsFile

                training parameters file.

        -iterations num

                number of training iterations, ignored if -params is used. Default value is 100.

        -cutoff num

                minimal number of times a feature must be seen, ignored if -params is used. Default value is 5.

        -model modelFile

                output model file.

        -lang language

                language which is being processed.

        -data sampleData

                data to be used, usually a file name.

        -encoding charsetName

                encoding for reading and writing text, if absent the system default is used.

Now lets say, we want to build a model “en-ner-drugs.bin” for data “drugsDetails.txt” in English language.

$opennlp TokenNameFinderTrainer -model en-ner-drugs.bin -lang en -data drugsDetails.txt -encoding UTF-8

Now we’ll see,  how can we train the same model using JAVA API.

**Steps** :

* Open a sample data stream
* Call the NameFinderME.train method
* Save the TokenNameFinderModel to a file

Here is the example.

**import** java.io.BufferedOutputStream;

**import** java.io.FileInputStream;

**import** java.io.FileOutputStream;

**import** java.io.IOException;

**import** java.nio.charset.Charset;

**import** java.util.Collections;

**import** java.util.HashMap;

**import** opennlp.tools.namefind.NameFinderME;

**import** opennlp.tools.namefind.NameSample;

**import** opennlp.tools.namefind.NameSampleDataStream;

**import** opennlp.tools.namefind.TokenNameFinderModel;

**import** opennlp.tools.util.ObjectStream;

**import** opennlp.tools.util.PlainTextByLineStream;

**public** **class** DrugsClassifierTrainer {

**static** String *onlpModelPath* = "en-ner-drugs.bin";

                // training data set

**static** String *trainingDataFilePath* = "D:/NLPTools/Datasets/drugsDetails.txt";

**public** **static** **void** main(String[] args) **throws** IOException {

                                Charset charset = Charset.*forName*("UTF-8");

                                ObjectStream<String> lineStream = **new** PlainTextByLineStream(

**new** FileInputStream(*trainingDataFilePath*), charset);

                                ObjectStream<NameSample> sampleStream = **new** NameSampleDataStream(

                                                                lineStream);

                                TokenNameFinderModel model = **null**;

                                HashMap<String, Object> mp = **new** HashMap<String, Object>();

**try** {

                                                model = NameFinderME.*train*("en", "drugs", sampleStream, Collections.<String,Object>                                                                                                                emptyMap(),100,4);

                                } **finally** {

                                                sampleStream.close();

                                }

                                BufferedOutputStream modelOut = **null**;

**try** {

                                                modelOut = **new** BufferedOutputStream(**new** FileOutputStream(*onlpModelPath*));

                                                model.serialize(modelOut);

                                } **finally** {

**if** (modelOut != **null**)

                                                                modelOut.close();

                                }

                }

}

Above code will generate the “en-ner-drugs.bin” model.

Now you are all set to use this model for finding entity like other NER models…!!!!!!!!

For more details, you can go through [OpenNLP Documentation](https://opennlp.apache.org/documentation/1.5.3/manual/opennlp.html" \l "tools.namefind.training" \t "_blank).

http://nishutayaltech.blogspot.in/2015/07/writing-custom-namefinder-model-in.html

# ****Open NLP Name Finder Model Training****

**Named Entity Recognition**

The Name Finder is used to detect entities like person, location, date, money, organization time and date in the text. These entities are detected using trained model. The model is specific to the language and entity type.

The Open NLP provides following pre-trained name finder models.

1. en-ner-location.bin
2. en-ner-money.bin
3. en-ner-organization.bin
4. en-ner-percentage.bin
5. en-ner-person.bin
6. en-ner-date.bin

These are trained on various freely available corpora.

## Open NLP Name Finder Training

To detect custom entities from Name Finder APIs, we have to train the models for requires entity and language specific.

### ****Training Tool****

Open NLP provides command line tool to train the models.

1. To train the model we have to provide the data in Open NLP name finder training format, which is one sentence per line.
2. We can use other format also to train the model, the sentence must be tokenized and contain spans which mark the entities.
3. Documents within train file are separated by empty lines which trigger the reset of the adaptive feature generators.

Sample training data for medical entity (**en-ner-medical.train file**):

|  |
| --- |
| <START:disease> Cancer <END> is the uncontrolled growth of abnormal cells anywhere in a body.These abnormal cells are termed <START:disease> cancer <END> cells, <START:disease> malignant <END> cells, or <START:disease> tumor <END> cells. |

**Command to train data :**

|  |
| --- |
| $ opennlp TokenNameFinderTrainer -model en-ner-medical.bin -lang en -data en-ner-medical.train -encoding UTF-8 |

Additionally it’s possible to specify the number of iterations, the cutoff.

### Training API

We can train the name finder using training API. Open NLP recommends to use the training API instead of the command line tool.

To train the model we have to follow the three steps:

* Read data from train file (Input Stream)
* Train the data using NameFinderME.train method (Training process)
* Store the model (TokenNameFinderModel) to a file (Output Stream)

public String train(String lang, String entity,

            InputStream taggedCorpusStream, OutputStream modelStream) {

          Charset charset = Charset.forName(CHAR\_ENCODING);

    try {

        ObjectStream<String> lineStream = new PlainTextByLineStream(

                taggedCorpusStream, charset);

        ObjectStream<NameSample> sampleStream = new NameSampleDataStream(

                lineStream);

        TokenNameFinderModel model;

        OutputStream modelOut = null;

        try {

            model = NameFinderME.train(lang, entity, sampleStream, null);

            modelOut = new BufferedOutputStream(modelStream);

            if (model != null) {

                model.serialize(modelOut);

            }

            return entity + " model trained successfully";

        } catch (Exception ex) {

            ex.printStackTrace();

        } finally {

            sampleStream.close();

            if (modelOut != null) {

                modelOut.close();

            }

        }

    } catch (Exception e) {

        e.printStackTrace();

    }

    return "Something goes wrong with training module.";

}

public String train(String lang, String entity, String taggedCoprusFile,

        String modelFile) {

    try {

        return train(lang, entity, new FileInputStream(taggedCoprusFile),

            new FileOutputStream(modelFile));

    } catch (Exception e) {

        e.printStackTrace();

    }

    return "Something goes wrong with training module.";

}

Call the “train” method of above code from main method :

train("en", "medical", "/home/Opennlp/en-ner-custom-medical.train","/home/Opennlp/en-ner-custom-medical.bin");

It will create model for disease entity in /home/Opennlp directory

http://blog.thedigitalgroup.com/sagarg/2015/10/30/open-nlp-name-finder-model-training/