Data Categorization using OpenNLP

# Heading

library is **a machine learning based toolkit** for the processing of natural language text.

It supports the most common NLP(natural language processing) tasks, such as tokenization, sentence segmentation, part-of-speech tagging, named entity extraction, chunking, parsing, coreference resolution and categorization. These tasks are usually required to build more advanced text processing services.

You can also go through [categorization using mahout](http://hanishblogger.blogspot.in/2013/03/machine-learning-categorization-with.html) to see how to categorize data using mahout. Data categorization with OpenNLP is another approach with more accuracy and performance rate as compared to mahout.

OpenNLP has a **Document Categorizer** api which can classify text into predefined categories.

It is based on **maximum entropy framework**.

*Note: There is no pre-built model for document categorizer under OpenNLP project that can be used for running sample program.*

ML data-categorization always comes in two major phases: learning (training) and application (testing).

During **learning** phase a set of pre-labeled data is used to train the model and during **testing** phase a part of pre-labeled data is used to test the model. Then finally we can use that trained model to classify any data.

**Run sample classification example on openNLP**

Prerequisites:

* Download openNLP tar file. Lets say **apache-opennlp-1.5.2-incubating-bin.tar.gz**

    Set environment variable OPENNLP\_HOME which refers to where opennlp lives.

* To use the OpenNLP Tools  in maven java project define the following dependency:

|  |
| --- |
| <dependency>  <groupId>org.apache.opennlp</groupId>  <artifactId>opennlp-tools</artifactId>  <version>1.5.2</version> </dependency> |

**1. Training**

The Document Categorizer can be trained on annotated training material. The data must be in OpenNLP Document Categorizer training format.

Format of training file is: one document per line, containing category and text separated by a whitespace.

You can train the model by using either training tool or using java training api.

*Training Tool*

|  |
| --- |
| $OPENNLP\_HOME/bin/opennlp DoccatTrainer -encoding UTF-8 -lang en -data <PATH OF TRAINING FILE> -model <LOCATION TO PUT OUTPUT MODEL>/en-doccat.bin |

OpenNLP inbuilt indexes the training data and then use that indexed data to train the model.

*Training API*

|  |
| --- |
| String modelFilePath= **. . .**             String trainingFile= **. . .**         // Instance of openNLP's default model class         DoccatModel model = null;         InputStream dataIn = null;         try {             dataIn = new FileInputStream(trainingFile);             ObjectStream<String> lineStream = new PlainTextByLineStream(dataIn,                     "UTF-8");             ObjectStream<DocumentSample> sampleStream = new DocumentSampleStream( lineStream);             // "en" is language code of English.             model = DocumentCategorizerME.train("en", sampleStream);         } catch (IOException e) {             log.error("Failed to read or parse training data, training failed",e);         } finally {             if (dataIn != null) {                 try {                     // free the memory resources.                     dataIn.close();                 } catch (IOException e) {                     log.warn(e.getLocalizedMessage());                 }             }         }         OutputStream modelOut = null;         try {             modelOut = new BufferedOutputStream(new FileOutputStream(modelFilePath));             model.serialize(modelOut);         } catch (IOException e) {             log.error("Failed to save model at location "+modelFilePath);         } finally {             if (modelOut != null) {                 try {                     modelOut.close();                 } catch (IOException e) {                     log.error("Failed to correctly save model. Written model might be invalid.");                 }             }         } |

Now you will see that model has been created at specified location.

**2. Testing**

OpenNLP DocumentCategorizerEvaluator API can be used to evaluate accuracy of trained model.

*Testing API*

|  |
| --- |
| String modelFilePath= **. . .**  InputStream is = new FileInputStream(modelFilePath);  DoccatModel model = new DoccatModel(is);  DocumentCategorizerME myCategorizer = new DocumentCategorizerME(model);  DocumentCategorizerEvaluator evaluator = new DocumentCategorizerEvaluator(myCategorizer);  String category= **. . .**  String content= **. . .**  DocumentSample sample = new DocumentSample(category, content);  evaluator.evaluteSample(sample);  double result = evaluator.getAccuracy();  System.out.println("Accuracy = " + result); |

In above sample api we can also create list of DocumentSample with large amount of testing data and then pass to evaluator which will give more appropriate result for model accuracy.

**3. Classifying**

OpenNLP Document Categorizer can classify text into predefined categories. To perform classification we will need a maxent model - these are encapsulated in the DoccatModel class of OpenNLP tools.

Now you can run the classifier using model which you  have created in first step.

*Classifying API*

|  |
| --- |
| String modelFilePath= **. . .**  InputStream is = new FileInputStream(modelFilePath);  DoccatModel model = new DoccatModel(is);  DocumentCategorizerME myCategorizer = new DocumentCategorizerME(model);  double[] outcomes = myCategorizer.categorize("string  to classify");  String category = myCategorizer.getBestCategory(outcomes); |

Thus you can categorize your data into predefined categories. Accuracy of  result totally depends on your training data.

If your training data is well categorized then you would be able to achieve up to 85% accuracy. :)