The Dragon Toolkit Developer Guide

Xiaohua (Davis) Zhou, Xiaodan (Tom) Zhang, Xiaohua (Tony) Hu

Data Mining and Bioinformatics Lab
College of Information Science & Technology, Drexel University
3141 Chestnut Street, Philadelphia, PA 19104, USA

xiaohua.zhou@drexel.edu

(Version 1.3, June 3, 2007)

1. Overview

The Dragon Toolkit is a Java-based development package for academic use in information retrieval (IR) and text mining (TM, including text classification, text clustering, text summarization, and topic modeling). It is tailored for researchers who work on large-scale IR and TM and prefer Java programming. Moreover, different from Lucene and Lemur, it provides built-in supports for semantic-based IR and TM. The dragon toolkit seamlessly integrates a set of NLP tools, which enable the toolkit to index text collections with various representation schemes including words, phrases, ontology-based concepts and relationships. However, to minimize the learning time, we intentionally keep the package small and simple. The toolkit does not have some features including distributed IR and cross-language IR which is a part of Lemur toolkit.

Another important feature of the toolkit is its scalability. Unlike many text mining tools such as Weka, the dragon toolkit is specially designed for large-scale application. The toolkit uses sparse matrix to implement text representations and does not have to load all data into memory in the running time. Therefore, it can handle hundred thousands of documents with very limited memory.

2. Installation

Download Dragon Toolkit Core Library (dragontool.jar) and other necessary libraries (see Table 1). Then add the path of all these jar files to the CLASSPATH environment variable. Download necessary NLP data (see Table 2) and uncompress them to the "nlpdata" directory. In order to let the toolkit locate the data, it is required to set an environment variable called **DRAGONTOOL**, which links to the directory containing the “nlpdata” directory.

The dragon toolkit needs Java 2 Runtime Environment 1.4 or later version. The toolkit consumes a large volume of memories, especially when using UMLS and MeSH ontologies because we load the ontology into the memory. For this reason, Please use the following memory allocation options when running programs based on the dragon toolkit.

```
java -mx1000000000 -oss10000000000 your java class
```

Table 1. The list of external java libraries

Library Name	Jar Files/Source Code	Used for
Hepple POS Tagger	heptag.jar (339K)	dragon.nlp.tool.HeppleTagger
MedPost POS Tagger	medpost.jar (20K)	dragon.nlp.tool.MedPostTagger
SVM-light 6.01	svmlight.jar (15K) libsvmlight.so (Linux, 127K) svmlight.dll (Windows, 133K)	dragon.ir.classification.SVMLightClassifier
libsvm	libsvm.jar (44K)	dragon.ir.classification.LibSVMClassifier
Java WordNet (JWNL)	jwnl.jar (168K) utlities.jar (24K) commons-logging.jar (26K)	dragon.nlp.tool.WordNetDidion
Link Grammar Parser	linkgrammar.jar (8K) linkgrammar.dll (Windows, 240K) liblinkgrammar.so (Linux, 704K)	dragon.nlp.tool.LinkGrammar
GATE Library	gate.jar (3118K) junit.jar (119K)	dragon.nlp.tool.Annie

	jdom.jar (144K) gnu-regexp-1.0.8.jar (23K) jasper-compiler-jdt.jar (896K)	
Cmmons HttpClient	commons-httpclient-3.0.jar (273K) commons-codec-1.3.jar (46K) commons-logging.jar (26K)	dragon.util.HttpUtil
DB2 Database	db2jcc.jar (1034K) db2jcc_license_cisuz.jar (3K) db2jcc_license_cu.jar (1K)	dragon.util.DBUtil
MSSQL Database	msbase.jar (281K) mssqlserver.jar (66K) msutil.jar (58K)	dragon.util.DBUtil
Java Excel	jxl.jar (645K)	dragon.ir.model.ModelExcelWriter

Table 2. The list of NLP data

Data Name	Compressed Data Package	Used for
UMLS 2004AA Version	umls.rar (79M)	dragon.nlp.ontology.umls
MeSH Ontology	mesh.rar (7M)	dragon.nlp.ontology.mesh
POS Tagger	tagger.rar (330K)	dragon.nlp.tool.HeppleTagger dragon.nlp.tool.MedPostTagger
English Lemmatiser	lemmatiser.rar (413K)	dragon.nlp.tool.lemmatiser
Annie NER	gate.rar (2.5M)	dragon.nlp.tool.Annie
JWNL	wordnet.rar (9M)	dragon.nlp.tool.WordNetDidion
Link Grammar Parser	linkgrammar.rar (252K)	dragon.nlp.tool.LinkGrammar
ROUGE	rouge.rar (2K)	dragon.ir.summarize.ROUGE
Misc for Examples	exp.rar (2M)	Examples

3. Development

3.1 Architecture

A typical route for the development of text retrieval and mining applications is illustrated in Figure 1. First of all, it is required to prepare a collection of machine-readable documents. Second, natural language processing (NLP) is applied to each article in the collection. The frequently used NLP techniques include tokenization, part of speech (POS) tagging, lemmatization and stemming, phrase extraction, and ontological concept extraction. Third, all extracted words, phrases, or ontological concepts are saved often in sparse matrices for future use. Fourth, text data mining tasks such as retrieval, clustering, classification, and summarization are run on indexed data. Last, the retrieval or mining results are evaluated.

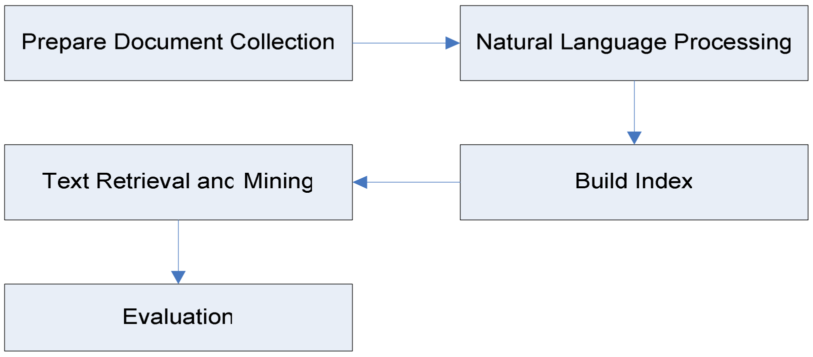
Prepare Document Collection
Natural Language Processing
Text Retrieval anc Mining
Build Index
Evaluation

Figure 1. Illustration of Text Mining Application Development

3.2 Conventions

Before stepping into the details of each step, I will introduce the conventions briefly, which is very helpful to understand the source code.

The development of applications follows the principle of object-oriented design (OOD). Any complex application will be split into a set of simple components (objects). Take the example of text retrieval task. To finish this task, one needs to pick up two components: a searcher and an evaluator at the top level. The searcher is further decomposed into four sub components: index, feedback, smoother, and query. One can choose different feedback and smoother to implement different searching methods.

To screen the details of various implementations, an interface is created for each set of objects with similar functions. Each interface defines a set of methods and variables. Although a specific class has a unique implementation of the interface, they often share many common implementations. For this reason, an abstract class, which implements the common parts, is created for each interface. In future, one can quickly implement an interface by extending the abstract class instead of coding from the scratch.

4. Functionality and Resource

4.1 Document Collection

Text retrieval and mining often deal with a collection of documents rather than a single document. The dragon toolkit provides various tools to collect, prepare, and read collections. All packages and classes with the prefix "dragon.onlinedb" are related to this line of functionalities.

The unit of a collection is Article. An article contains title, abstract, body, Meta, and so on. A collection reader is an agent to read articles from a collection. Often, collection readers work with article parser together. Collection readers usually identify the boundary and extract the article from the collection and the article parser further parse the extracted raw text into an article.

In this version, five types of collection readers have been implemented. The basic collection put all articles in a single text file and each line stands for an article. The array collection put selected articles into an array list. The simple collection treats each file as an article and the whole folder as a collection. The TREC-styled collection stores the whole collection in multiple files which could be in any sub-folders and each file contains multiple articles delimited by the tag "DOC". The last type of collection is online collection which is resided on an online database or a website. The implemented online collection readers include PubMed and Amazon Online Review readers.

The dragon toolkit also includes various article parsers. The basic article parser uses tab to delimit different fields of an article. The simple article parser simply treats all content of a text file as the body of an article. The SGM-styled news article parsers can parse most articles in TREC and TDT collections. For more miscellaneous article parsers, please refer to the API documentation.

4.2 Natural Language Processing

The NLP package provides the functionalities of extracting various concepts or relationships from articles during indexing. The concepts can be individual words, multiword phrases, proper nouns, and even ontological concepts. In order to facilitate the extraction of concepts other than individual words, the toolkit integrates various NLP tools such as stemmers, part of speech taggers, sentence parsers, phrase extractors, and named entity recognizers. The toolkit also supports the extraction of ontological concepts. It integrates WordNet, a generic Word ontology and implements two ontologies in domain of biomedicine, UMLS and MeSH,

For the convenience of natural language processing, the toolkit provides four basic data structures, Document, Paragraph, Sentence, and Word, to parse and tokenize the content of an article. The extractors and other supporting NLP tools use these four basic data structures to share data.

The toolkit defines three types of concept extractors. The first is token extractor, which extracts a sequence of individual words from a sentence or a document. The second is phrase extractor, namely extracting multiword phrases from a sentence or a document. The phrase extractor needs a phrase dictionary as input; the phrase dictionary could be automatically built by phrase tools such as Xtract. The third is term extractor, which extracts ontological terms from a sentence or a document.

4.3 Document Indexing

Indexing is a procedure of converting natural languages into structured and compacted format. The indexed data can feed various text mining applications such as text retrieval, clustering, classification, summarization, and topic modeling. The most frequently used indexing format is matrix. In other words, a collection of articles will be finally converted to a doc-term matrix and a term-doc matrix plus some basic statistics.

The dragon toolkit uses its own sparse matrix technique. For more information about sparse matrix, please read Section 6 and related API documentations. The toolkit provides three types of indexing: basic indexing, sequence indexing, and sentence indexing. The basic indexing converts extracted concepts into an integer-based index and save the indices of unique concepts and their frequency in a doc-term matrix; the inversed document frequency is saved in a term-doc matrix. The sentence indexing is almost the same as the basic indexing except that it splits an article into sentences first and then treats each sentence as a document for indexing. In other words, in the doc-term matrix resulted by the sentence indexing, each document actually denotes a sentence. The sentence indexing can be used for sentence-level summarizations. The sequence indexing simply converts each word to an integer-based index and record a sequence of indices. It may be used for any sequence-sensitive applications. According to whether the indexing is saved into hard disk, each type can be further classified into disk-based indexing and online indexing. The online indexing saves all information in memory and thus fits small collections.

Table 3. The explanations of files generated by the disk-based basic indexing

Filename	Format	Description
termkey.list	text	Mapping between term name and term index
dockey.list	text	Mapping between document key and document index
termdoc.matrix	binary	The term-document matrix
termdoc.index	binary	The index file of term-document matrix
docterm.matrix	binary	The document-term matrix
docterm.index	binary	The index file of document-term matrix
termindex.list	binary	The statistics of terms in the collection
docindex.list	binary	The statistics of documents in the collection
relationdoc.matrix	binary	The relation-document matrix
trelationdoc.index	binary	The index file of relation-document matrix
docrelation.matrix	binary	The document-relation matrix
docrelation.index	binary	The index file of document-relation matrix
relationindex.list	binary	The statistics of relations in the collection
rawsentence.collection	text	A list of sentences, for sentence indexing only
rawsentence.index	text	The index of positions of sentences in the collection

One can call an index reader to get the information of an indexed collection. Usually, disk-based index readers accept the indexing directory as input argument while online index readers accept an indexer and a collection as input arguments.

4.4 Text Mining and Evaluation

The dragon toolkit supports text retrieval, clustering, classification, summarization, and topic modeling. Usually, it is required to index documents prior to running these applications. After running these applications, one can simply call the corresponding evaluation program to evaluate the retrieval or mining results. The toolkit has implemented popular evaluation metrics for the above-mentioned applications.

The toolkit provides a well-defined framework for text retrieval. The first step is to generate a query from the topic descriptions (such as TREC Topic files). Please refer to the package of dragon.ir.query for query generation. The second step is to create a searcher. Since there are so many different retrieval models, the toolkit creates an interface called Smoother to hide the implementation details of different models. Thus, the routine for searching is the same for different models. One can simply call a full rank searcher or a partial rank searcher. The toolkit has implemented various language model smoothing methods as well as traditional probabilistic and vector space models. Pseudo-relevance feedback and query expansion are two frequently used techniques for improving the effectiveness of IR. One can call a feedback searcher or an expansion searcher to incorporate these two techniques, respectively. The details of the feedback approaches and query expansion approaches are encapsulated into the implantations of two interfaces, Feedback and Expansion, respectively. To evaluate the IR performance using TREC protocol, please call dragon.ir.search.evaluate.TrecEva.

The toolkit implements two common clustering approaches, the agglomerative algorithm and the K-Means algorithm. These two approaches have many variants in terms of similarity measures. The toolkit encapsulates the details of different similarity measures into the implementations of two interfaces, Doc

Distance and Cluster Model, respectively. The Doc Distance interface computes the distance between two documents and is designed for agglomerative clustering approaches. The Cluster Model interface computes the distance between a document and a cluster or the generative probability of a document by a cluster model. To evaluate the clustering quality, please call dragon.ir.clustering.ClusteringEva.

The toolkit includes a well-defined framework for text classification. A classifier should implement the interface called Classifier. Because feature selection is very important to text classification, an interface called Feature Selection is also created and therefore feature selection method is separated from the classification algorithm. One can test different feature selection methods without changing the classifier. To evaluate the classification quality, please call dragon.ir.classification.ClassficationEva.

The toolkit supports generic multi-document summarization. The summarizer should implement the interface called Generic Multi-Document Summarizer. Given a collection of articles, the summarizer should return a summary with given maximum length. To evaluate the quality of the machine-generated summary, please call ROUGE. The dragon toolkit implemented and tested the N-gram metric of ROUGE.

The topic modeling has been a hot research topic in recent years. The dragon toolkit implements three the-state-of-the-art topic models, the aspect model, the LDA model, and the simple mixture model.

Although the toolkit provides evaluation APIs, it is still time consuming to write a control program to load the testing data and human judgments, run the text mining applications, compare the mining results with human judgment, and finally print out the evaluation result. For this reason, the toolkit provides an evaluation program for each text mining tasks including text retrieval, classification, clustering and summarization. What one has to do is to prepare a XML-based configuration file to feed the evaluation program. Take the example of text classification. One just sets the classifier, the collection, the document labels, and the evaluation mode in the configuration file. To learn how to write a configuration file, please refer to Section 5 and examples on the homepage of the dragon toolkit.

4.5 List of All Resources

All available resources are listed in Table 4 for your reference.

Table 4. The list of all available resources

Resource Name	Class or Package
Sparse matrix	Dragon.matrix
Power method	Dragon.matrix.vector.PowerMethod
HITS algorithm	Dragon.matrix.vector.HITS
Singular value decomposition	Dragon.matrix.factorize.SVD
Non-negative matrix factorization	Dragon.matrix.factorize.NMF
Conditional Random Field	Dragon.ml.seqmodel
TREC collection reader and parsers	Dragon.onlinedb.trec
PubMed query	Dragon.onlinedb.pubmed
Amazon customer reviewer query	Dragon.onlinedb.amazon
CiteULike query	Dragon.onlinedb.citeulike
Google Search Engine	Dragon.onlinedb.searchengine.GoogleEngine
ISI article parser	Dragon.onlinedb.isi.ISIArticleParser
20 Newsgroup article	Dragon.onlinedb.dm.NewsgroupArticle
Reuters article parser	Dragon.onlinedb.dm.ReutersArticleParser
Simple collection reader	Dragon.onlinedb.SimpleCollectionReader
Simple article parser	Dragon.onlinedb.SimpleArticleParser
Basic collection reader	Dragon.onlinedb.BasicCollectionReader
Basic article parser	Dragon.onlinedb.BasicArticleParser
Annie named entity recognizer	Dragon.nlp.tool.Annie
English lemmatiser	Dragon.nlp.tool.lemmaister.EngLemmatiser
WordNet	Dragon.nlp.tool.WordNetDidion
Porter stemmer	Dragon.nlp.tool.PorterStemmer
Hepple tagger	Dragon.nlp.tool.HeppleTagger
Medpost tagger	Dragon.nlp.tool.MedPostTagger
Brill Tagger	Dragon.nlp.tool.BrillTagger
Xtract (phrase extractor)	Dragon.nlp.tool.xtract
Phrase extraction program	Dragon.config.PhraseExtractAppConfig

UMLS ontology	Dragon.nlp.ontology.umls
MeSH ontology	Dragon.nlp.ontology.mesh
Token extractor	Dragon.nlp.extract.BasicTokenExtractor
Phrase extractor	Dragon.nlp.extrctor.BasicPhraseExtractor
Ontological term extractor	Dragon.nlp.extractor.BasicTermExtractor
Concept pair extractor	Dragon.nlp.extractor.BasicTripleExtractor
Naive Bayesian classifier	Dragon.ir.classification.NBClassifier
NB with semantic smoothing	Dragon.ir.classification.SemanticNBClassifier
Nigam active learning	Dragon.ir.classification.NigamActiveLearning
SVM-light 6.01 classifier	Dragon.ir.classification.SVMLightClassifier
LibSVM classifier	Dragon.ir.classification.LibSVMClassifier
Doc Frequency feature selector	Dragon.ir.classification.featureselection.DocFrequencySelector
Chi Square feature selector	Dragon.ir.classification.featureselection.ChiFeatureSelector
Mutual info. feature selector	Dragon.ir.classification.featureselection.MutualInfoFeatureSelector
Info. Gain feature selector	Dragon.ir.classification.featureselection.InfoGainFeatureSelector
Loss-based multi-class decoder	Dragon.ir.classfication.multiclass.LossMultiClassDecoder
One-versus-all code matrix	Dragon.ir.classification.multiclass.OVACodeMatrix
Pairwise code matrix	Dragon.ir.classification.multiclass.AllPairCodeMatrix
Classification evaluation API	Dragon.ir.classification.ClassificationEva
Classification evaluation program	Dragon.config.ClassificationEvaAppConfig
Hierarchical clustering	Dragon.ir.clustering.HierClustering
K-Means	Dragon.ir.clustering.BasicKMean
Link-based K-Means	Dragon.ir.clustering.LinkKMean
Cosine distance	Dragon.ir.clustering.docdistance.CosineDocDistance
Euclidean distance	Dragon.ir.clustering.docdistance.EuclideanDocDistance
KL-divergence distance	Dragon.ir.clustering.docdistance.KLDivDocDistance
Cosine cluster model	Dragon.ir.clustering.clustermodel.CosineClusterModel
Euclidean cluster model	Dragon.ir.clustering.clustermodel.EuclideanClusterModel
Multinomial cluster model	Dragon.ir.clustering.clustermodel.MultinomialClusterModel
Clustering evaluation API	Dragon.ir.clustering.ClusteringEva
Clustering evaluation program	Dragon.config.ClusteringEvaAppConfig
Aspect model	Dragon.ir.topicmodel.AspectModel
LDA model	Dragon.ir.topicmodel.GibbsLDA
Simple mixture model	Dragon.ir.topicmodel.SimpleMixtureModel
Topic model writer	Dragon.ir.topicmodel.ModelExcelWriter
LexRank summarizer	Dragon.ir.summarize.LexRankSummarizer
ROUGE	Dragon.ir.summarize.ROUGE
Text Summarization Evaluation	Dragon.config.SummarizationEvaAppConfig
Two-stage smoothing	Dragon.ir.search.smooth.TwoStageSmoother
TF-IDF retrieval model	Dragon.ir.search.smooth.TFIDFSmoother
Okapi retrieval model	Dragon.ir.search.smooth.OkapiSmoother
Pivoted norm model	Dragon.ir.search.smooth.PivotedNormSmoother
JM smoothing	Dragon.ir.search.smooth.JMSmoother
Absolute discount smoothing	Dragon.ir.search.smooth.AbsoluteDiscountSmoother
Dirichlet smoothing	Dragon.ir.search.smooth.DirichletSmoother
Translation-based smoother	Dragon.ir.search.smooth.QueryFirstTransSmoother
Model-based feedback	Dragon.ir.search.feedback.GenerativeFeedback
Minimum divergence feedback	Dragon.ir.search.feedback.MinDivergenceFeedback
Information-flow feedback	Dragon.ir.search.feedback.InformationFlowFeedback
Relation-based feedback	Dragon.ir.search.feedback.RelationTransFeedback
Phrase-based feedback	Dragon.ir.search.feedback.PhraseTransFeedback
Rocchio feedback	Dragon.ir.search.feedback.RocchioFeedback
Basic Query generator	Dragon.ir.query.BasicQueryGenerator
Phrase-based query expansion	Dragon.ir.query.PhraseQEGenerator
Query generating program	Dragon.config.QueryAppConfig
TREC evaluation API	Dragon.ir.search.evaluate.TrecEva
IR evaluation program	Dragon.config.RetrievalEvaAppConfig
Basic Indexer	Dragon.ir.index.BasicIndexer
Basic sentence indexer	Dragon.ir.index.sentence.BasicSentenceIndexer
Basic sequence indexer	Dragon.ir.index.sequence.BasicSequenceIndexer
Indexing program	Dragon.config.IndexAppConfig

Basic index reader	Dragon.ir.index.BasicIndexReader
Online index reader	Dragon.ir.index.OnlineIndexReader
Basic sentence index reader	Dragon.ir.index.sentence.BasicSentenceIndexReader
Online sentence index reader	Dragon.ir.index.sentence.OnlineSentenceIndexReader
Basic sequence index reader	Dragon.ir.index.sequence.BasicSequenceIndexReader
Online sequence index reader	Dragon.ir.index.sequence.OnlineSequenceIndexReader
Translation program	Dragon.config.TranslationAppConfig
Cooccurrence program	Dragon.config.CooccurrenceAppConfig

5. Configuration File

In addition to programming APIs, the dragon toolkit also offers a XML-based interface to end users. One can simply define an object resource or an application in a XML-based configuration file. This mechanism makes the development easier and faster. It also becomes very convenient to keep and share experiment configurations.

```
<?xml version="1.0"?>
<configure>
    <porterstemmer type="lemmatiser" id="1"/>
    <englemmatiser type="lemmatiser" id="2">
        <param name="indexlookupoption" value="false"/>
        <param name="disableverbadjective" value="true"/>
    </englemmatiser>
    <medposttagger type="tagger" id="1">
        <param name="directory" value="nlpdata/tagger"/>
    </medposttagger>
    <heppletagger type="tagger" id="2">
        <param name="directory" value="nlpdata/tagger"/>
    </heppletagger>
    <basictokenextractor type="conceptextractor" id="1">
        <param name="subconceptoption" value="false"/>
        <param name="lemmatiser" type="lemmatiser" value="1"/>
        <param name="notworddelimitor" value=""/>
        <param name="filteroption" value="true"/>
        <param name="conceptfilter" type="conceptfilter" value="1"/>
        <basicconceptfilter type="conceptfilter" id="1">
            <param name="stoplistfile" value="nlpdata/exp/rijsbergen.stopword"/>
        </basicconceptfilter>
    </basictokenextractor>
    <svmclassifier class="dragon.ir.classification.SVMClassifier" type="classifier" value="1">
        <param name="xxx" value="xxx"/>
    </svmclassifier>
</configure>
```

Figure 2 Example of Configuration File

The configuration file has a root node called configure. The root node can contain multiple object nodes. An object node stands for an object or an application. An object node has two required attributes (type and id) and an optional attribute called class. The type of the object node is often the interface the object implements and the id is an integer to distinguish different object nodes with the same type. Thus the combination of type and id should be unique. The node name is often the class name (without package modifier). The toolkit locates an object by its type and id and determines how to load the object according to the node name. An object node may have multiple parameters which is either a simple data type (e.g. double, integer, boolean and string) or an object. If the parameter denotes an object, it has an optional attribute called type to indicate the type of the object and its value must be an integer denoting the id of the object. According to the type and id, the toolkit can locate the object in the configuration file. The search of the object specified by a parameter follows the order below: (1) it searches the object within the current object node first; (2) if not found, it searches the object within the parent node of the current object node; if still not found, it searches the object within the root node. In a short, the toolkit applies a recursive mechanism to the configuration of objects and applications. Therefore, one can define complicated objects and applications in a reusable and compact manner.

The dragon toolkit provides well-defined framework for various text mining applications. It allows developers to create their own algorithms or applications as long as the new class implements the corresponding interface defined by the toolkit. For example, one can create a SVM classifier by implementing the interface called dragon.ir.classification.Classifier. In this case, the toolkit does not know how to load and configure the SVM classifier. Fortunately, the dragon toolkit provides a solution to this problem.

First, one should use the attribute of the object node called "class" to declare the class name of the new object (see Figure 2). Second, one must define a public and static method which returns the defined object according to the given configuration in the implementation class. The method name begins with the verb "get" and is followed by the class name (without package modifier). See figure 3 for the example of SVM classifier.

```
public static SVMClassifier getSVMClassifier (ConfigureNode node){
    // implementation details
    …
}
```

Figure 3 The example of the definition of the method for creating an object

It is worth noting that the node name, node type and parameter name are case insensitive. However, the class attribute and the parameter value are case sensitive.

We are very sorry that we can not provide detailed specifications regarding parameter settings for different objects and applications at this moment. There are two ways one can learn the parameter settings. One is to learn from the configuration files provided by the six examples. These configuration files almost cover all types of objects. The other way is to read the source code under the package dragon.config.

6. Sparse Matrix

Colt is a well-known Java-based package for sparse matrix. However, it does not fit text mining applications for two reasons. First, a frequently used operation in text mining applications is to get all terms and their frequency in a given document; colt is not efficient to execute this operation. Second, colt uses hash method to implement sparse matrix and all data are kept in memory; thus it is not suitable for large text collections.

The dragon toolkit uses its own technique to implement sparse matrix. Basically, the toolkit provides three types of sparse matrix, flat sparse matrix, super sparse matrix, and giant sparse matrix. The flat sparse matrix is very similar to the colt matrix. It loads all data into memory and thus very fast, but fit for small dataset only. The super sparse matrix and giant sparse matrix reside on files in disk. Both have two files. One is matrix file (the extension is "matrix") and the other is index file (the extension is "index"). The index file is actually redundant and can be generated according to the matrix file. The super sparse matrix loads index into the memory and caches given number of recent –read rows. The giant sparse matrix loads nothing into the memory except caching the last-read row. One can select appropriate sparse matrix for your own applications according to the reading pattern, collection size, available physical memory as well as the expectation on reading speed.