JaTeCS, a Java library focused on automatic text categorization

Andrea Esuli (andrea.esuli@isti.cnr.it)
Tiziano Fagni (tiziano.fagni@isti.cnr.it)
Alejandro Moreo Fernández (alejandro.moreo@isti.cnr.it)

Istituto di Scienze e Tecnologie dell'Informazione (ISTI)
Italian National Research Council (CNR)
Pisa, Italy

March 25, 2016

Abstract

JaTeCS is an open source Java library focused on automatic text categorization. It covers all the steps of an experimental activity, from reading the corpus to the evaluation of the results. JaTeCS focuses on text as the central input, and its code is optimized for this type of data. As with many other machine learning (ML) frameworks, JaTeCS provides data readers for many formats and well-known corpora, NLP tools, feature selection and weighting methods, the implementation of many ML algorithms as well as wrappers for well-known external software (e.g., libSVM, SVM^{light}). JaTeCS also provides the implementation of methods related to text classification that are rarely, if never, provided by other ML framework (e.g., active learning, quantification, transfer learning).

1 Introduction

JaTeCS (Java Text Categorization System) is a Java framework that we developed in the past years as the main software tool to perform research on a broad range of automatic text categorization (ATC) problems. It is similar in spirit to other machine learning (ML) libraries like Weka [4], Scikit-learn [6] or Mallet [5] but it is more focused on text as the central input data. JaTeCS covers every stage required by a typical experimentation pipeline (data loading, feature extraction, dimensionality reduction, learning, evaluation). We now publish[1] it under the LGPL v3.0 license.
JaTeCS has been designed to support single/small group's day-to-day research on ML methods for ATC; its deployment scenario is single server with a multiple

[1] `https://github.com/jatecs/jatecs`

cores CPU and an average amount of memory; it has been developed to exploit this configuration, using multi-threaded processing and exploiting the intrinsic characteristics of information extracted from text (e.g., high dimensionality, sparseness, Zipfian distribution, multiple labels, hierarchical labeling...) in the design of its data structures. The code is divided into two components: a core library[2] that implements most of the tools needed to implement ATC methods, and a vast examples set[3] that provides ready-made, customizable applications, built on top of the core library, for a rich range of typical ATC tasks. JaTeCS support its expansion by abstracting, through interfaces, the typical tools and procedures used in ATC tasks, and providing a number of "template" implementations of typical ATC applications, in which components that implement the various interfaces (corpora readers, text processors, learners...) can be quickly plugged in.

2 Main features

JaTeCS handles the most common types of classification tasks, starting from simple binary classification, and including multi-label classification, multi-class single-label classification, and hierarchical classification. In hierarchical taxonomies, the learner is able to choose the selection policy of negative examples by choosing from *Siblings*, *All*, *BestGlobal*, and *BestLocal(k)* policies. Classification on multilingual (comparable or parallel) corpora is also supported, including an implementation of the Multilingual Domain Models technique.
Optimization of parameters is supported by the implementation of many well known exploration/validation methods, including *K-Fold cross validation* (simple or stratified), *leave-one-out* and *grid-search*.
JaTeCS is also able to handle advanced and uncommon NLP application types. The framework provides implementations of several methods having the aim to improve the quality of training data: Active Learning [7], Training Data Cleaning [2], Semi-Automated Text Classification [1], and Quantification [3]. *Transfer learning* is also supported by the library through the implementation of representation methods (e.g., Distributional Correspondence Indexing), or analytic tools (e.g., *proxy $\mathcal{A}$-distance*). The library implements *corpus readers* for many popular text corpora like Reuters-21578, RCV1-v2, RCV2, WipoAlpha, etc. The library also supports datasets in the popular SVM^{light} and Comma Separated Values (CSV) formats. A rich set of NLP tools (e.g., stop words removal, word or character n-grams, stemming, POS tagging, sentiment lexica) is included in JaTeCS. JaTeCS offers many dimensionality reduction tools through (i) filtering methods, implementing local or global feature selection with classic scoring functions like *Information Gain*, χ^2, etc.; and (ii) distributional semantic models, including Latent Semantic Analysis, and Random Projections.
JaTeCS implements many ML algorithms for classification, ordinal regression,

[2]Located in `src/main/java` in GitHub project repository.
[3]Located in `src/example/java` in GitHub project repository.

and clustering. It covers the most popular learning methods such as the Naïve Bayes, Rocchio, Logistic regression, k-NN, AdaBoost and boosting methods in general, and Support Vector Machines (SVMs). For ordinal regression, the library provides wrappers for ϵ-SVR and ν-SVR, as well as implementations of graph based methods, such as regression trees and D-DAGs. A highly customizable implementation of k-means is implemented for clustering. Committees (ensembles) and bagging methods are also implemented.

3 Data representation through IIndex structure

In JaTeCS, a processed textual dataset is made available to the learning algorithm through a *IIndex* interface that offers different views of the data. The *IIndex* is based on three basic entities: documents, features (extracted from documents), and categories (on which documents are labeled). Each entity has a corresponding database interface: *IDocumentDB*, *IFeatureDB*, and *ICategoryDB*. The *ICategoryDB* also models relations between categories, enabling the definition of hierarchical, preferential, and ordinal regression categorization scenarios. Relations between the basic entities are handled by four databases: *IContentDB*, that keeps track of how features occur in documents; *IWeightingDB*, that stores the relative importance of features with respect to documents (as computed by any given weighting function); *IClassificationDB*, that stores the labeling assigned to documents; and *IDomainDB*, which accounts for the set of valid features for each category, thus enabling category-dependent representations. These interfaces define methods that model recurrent queries in ATC problems, like "count all documents containing feature f" or "retrieve all documents from category c which also contain feature f". The following example lists the features which occur more than three times in documents belonging to a category:

```
// iterates on docs assigned to a category
for (int docID : index.getClassificationDB().getCategoryDocuments(cat)) {
   for (int featID : index.getContentDB().getDocumentFeatures(docID)) {
      // checks features satisfying the condition
      if(index.getContentDB().getDocumentFeatureFrequency(docID, featID)
          > 3)
         print(docID+":'"+index.getFeaturesDB().getFeatureName(featID)+"'");
         // prints 1:'the' 1:'is' 1:'said'...
}}
```

The *IIndex* can be extended with additional task-specific databases, e.g., a *ILanguageDB* is available to handle multilingual datasets information. JaTeCS provides an in-memory implementation of all the interfaces based on the high performance primitive collections of Trove library, as well as tools for data persistence.

4 A text quantification example in JaTeCS

As an example of JaTeCS usage, we show a sample code of text quantification. Quantification [3] is the problem of estimating the distribution of labels in a collection of unlabeled documents, when its distribution may substantially differ from the one of the training set. Though quantification processes a dataset as a single entity, the classification of single documents is the building block on which many quantification methods are built. JaTeCS implements the three best performing classification-based methods of [3] as well as their probabilistic versions. The implementation is independent from the underlying classification method, which acts as a plug-in component:

```
int folds = 50; IScalingFunction scaling = new LogisticFunction();
// any other learner can be plugged in
ILearner classificationLearner = new SvmLightLearner();

// learns six different quantifiers on training data (train is an IIndex
    object)
QuantificationLearner quantificationLearner = new QuantificationLearner(
   folds, classificationLearner, scaling);
QuantifierPool pool = quantificationLearner.learn(train);

// quantifies on test returning the six predictions (test is an IIndex
    object)
Quantification[] quantifications = pool.quantify(test);
// evaluates predictions against true quantifications
QuantificationEvaluation.Report(quantifications,test);
```

5 Conclusion

JaTeCS is an open source Java library specifically designed for ATC research. The framework builds around the notion of IIndex, a data structure that abstracts the text corpus and all of its components, facilitating the development of text-focused ML algorithms and the relative experimental activities. JaTeCS covers all the ML pipeline, from reading the corpus, to evaluating experiments. Any step is modeled with interfaces, allowing a quick plug-in like construction of pipelines. In addition to the typical set of text processing and ML tools that are also provided by similar frameworks, JaTeCS provides implementations for more complex text mining methods that are not usually supported by other frameworks.

References

[1] G. Berardi, A. Esuli, and F. Sebastiani. A utility-theoretic ranking method for semi-automated text classification. In *Proceedings of the 35th Interna-*

tional ACM SIGIR Conference on Research and Development in Information Retrieval, SIGIR '12, pages 961–970, New York, NY, USA, 2012. ACM.

[2] A. Esuli and F. Sebastiani. Training data cleaning for text classification. In *Advances in Information Retrieval Theory: Second International Conference on the Theory of Information Retrieval, ICTIR 2009 Cambridge, UK, September 10-12, 2009 Proceedings*, pages 29–41, Berlin, Heidelberg, 2009. Springer Berlin Heidelberg.

[3] G. Forman. Quantifying counts and costs via classification. *Data Min. Knowl. Discov.*, 17(2):164–206, Oct. 2008.

[4] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The weka data mining software: An update. *SIGKDD Explor. Newsl.*, 11(1):10–18, Nov. 2009.

[5] A. K. McCallum. Mallet: A machine learning for language toolkit. http://www.cs.umass.edu/ mccallum/mallet, 2002.

[6] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[7] S. Tong and D. Koller. Support vector machine active learning with applications to text classification. *J. Mach. Learn. Res.*, 2:45–66, Mar. 2002.