DKPro TC: A Java-based Framework for Supervised Learning Experiments on Textual Data*

Johannes Daxenberger†, Oliver Ferschke†‡, Iryna Gurevych†‡ and Torsten Zesch§‡

† UKP Lab, Technische Universität Darmstadt
‡ Information Center for Education, DIPF, Frankfurt
§ Language Technology Lab, University of Duisburg-Essen
http://www.ukp.tu-darmstadt.de

Abstract

We present DKPro TC, a framework for supervised learning experiments on textual data. The main goal of DKPro TC is to enable researchers to focus on the actual research task behind the learning problem and let the framework handle the rest. It enables rapid prototyping of experiments by relying on an easy-to-use workflow engine and standardized document preprocessing based on the Apache Unstructured Information Management Architecture (Ferrucci and Lally, 2004). It ships with standard feature extraction modules, while at the same time allowing the user to add customized extractors. The extensive reporting and logging facilities make DKPro TC experiments fully replicable.

1 Introduction

Supervised learning on textual data is a ubiquitous challenge in Natural Language Processing (NLP). Applying a machine learning classifier has become the standard procedure, as soon as there is annotated data available. Before a classifier can be applied, relevant information (referred to as *features*) needs to be extracted from the data. A wide range of tasks have been tackled in this way including language identification, part-of-speech (POS) tagging, word sense disambiguation, sentiment detection, and semantic similarity.

In order to solve a supervised learning task, each researcher needs to perform the same set of steps in a predefined order: *reading input data*, *preprocessing*, *feature extraction*, *machine learning*, and *evaluation*. Standardizing this process is quite challenging, as each of these steps might vary a lot depending on the task at hand. To complicate matters further, the experimental process is usually embedded in a series of configuration changes. For example, introducing a new feature often requires additional preprocessing. Researchers should not need to think too much about such details, but focus on the actual research task. DKPro TC is our take on the standardization of an inherently complex problem, namely the implementation of supervised learning experiments for new datasets or new learning tasks.

We will make some simplifying assumptions wherever they do not harm our goal that the framework should be applicable to the widest possible range of supervised learning tasks. For example, DKPro TC only supports a limited set of machine learning frameworks, as we argue that differences between frameworks will mainly influence runtime, but will have little influence on the final conclusions to be drawn from the experiment. The main goal of DKPro TC is to enable the researcher to quickly find an optimal experimental configuration. One of the major contributions of DKPro TC is the modular architecture for preprocessing and feature extraction, as we believe that the focus of research should be on a meaningful and expressive feature set. DKPro TC has already been applied to a wide range of different supervised learning tasks, which makes us confident that it will be of use to the research community.

DKPro TC is mostly written in Java and freely available under an open source license.[1]

2 Requirements

In the following, we give a more detailed overview of the requirements and goals we have identified for a general-purpose text classification system. These requirements have guided the development of the DKPro TC system architecture.

* This is the preprint of a paper accepted for presentation at the 52nd Annual Meeting of the Association for Computational Linguistics (ACL 2014) in Baltimore, MD, USA. Please refer to the published version for citation.

[1] http://dkpro-tc.googlecode.com

		Label **Single-label**	Label Label Label **Multi-label**	0.75 **Regression**
	**Document Mode**	· Spam Detection · Sentiment Detection	· Text Categorization · Keyphrase Assignment	· Text Readability
	**Unit/Sequence Mode**	· Named Entity Recognition · Part-of-Speech Tagging	· Dialogue Act Tagging	· Word Difficulty
	**Pair Mode**	· Paraphrase Identification · Textual Entailment	· Relation Extraction	· Text Similarity

Table 1: Supervised learning scenarios and feature modes supported in DKPro TC, with example NLP applications.

Flexibility Users of a system for supervised learning on textual data should be able to choose between different machine learning approaches depending on the task at hand. In supervised machine learning, we have to distinguish between approaches based on classification and approaches based on regression. In classification, given a document $d \in D$ and a set of labels $C = \{c_1, c_2, ..., c_n\}$, we want to label each document d with $L \subset C$, where L is the set of relevant or true labels. In single-label classification, each document d is labeled with exactly one label, i.e. $|L| = 1$, whereas in multi-label classification, a set of labels is assigned, i.e. $|L| \geq 1$. Single-label classification can further be divided into binary classification ($|C| = 2$) and multi-class classification ($|C| > 2$). In regression, real numbers instead of labels are assigned.

Feature extraction should follow a modular design in order to facilitate reuse and to allow seamless integration of new features. However, the way in which features need to be extracted from the input documents depends on the the task at hand. We have identified several typical scenarios in supervised learning on textual data and propose the following *feature modes*:

- In *document mode*, each input document will be used as its own entity to be classified, e.g. an email classified as wanted or unwanted (spam).
- In *unit/sequence mode*, each input document contains several units to be classified. The units in the input document cannot be divided into separate documents, either because the context of each unit needs to be preserved (e.g. to disambiguate named entities) or because they form a sequence which needs to be kept (in sequence tagging).
- The *pair mode* is intended for problems which require a pair of texts as input, e.g. a pair of sentences to be classified as paraphrase or non-paraphrase. It represents a special case of multi-instance learning (Surdeanu et al., 2012), in which a document contains exactly two instances.

Considering the outlined learning approaches and feature modes, we have summarized typical scenarios in supervised learning on textual data in Table 1 and added example applications in NLP.

Replicability and Reusability As it has been recently noted by Fokkens et al. (2013), NLP experiments are not replicable in most cases. The problem already starts with undocumented preprocessing steps such as tokenization or sentence boundary detection that might have heavy impact on experimental results. In a supervised learning setting, this situation is even worse, as e.g. feature extraction is usually only partially described in the limited space of a research paper. For example, a paper might state that "n-gram features" were used, which encompasses a very broad range of possible implementations.

In order to make NLP experiments replicable, a text classification framework should (i) encourage the user to reuse existing components which they can refer to in research papers rather than writing their own components, (ii) document all performed steps, and (iii) make it possible to re-run experiments with minimal effort.

Apart from helping the replicability of experiments, reusing components allows the user to concentrate on the new functionality that is specific to the planned experiment instead of having to reinvent the wheel. The parts of a text classification system which can typically be reused are

preprocessing components, generic feature extractors, machine learning algorithms, and evaluation.

3 Architecture

We now give an overview of the DKPro TC architecture that was designed to take into account the requirements outlined above. A core design decision is to model each of the typical steps in text classification (reading input data and preprocessing, feature extraction, machine learning and evaluation) as separate *tasks*. This modular architecture helps the user to focus on the main problem, i.e. developing and selecting good features.

In the following, we describe each module in more detail, starting with the workflow engine that is used to assemble the tasks into an experiment.

3.1 Configuration and Workflow Engine

We rely on the DKPro Lab (Eckart de Castilho and Gurevych, 2011) workflow engine, which allows fine-grained control over the dependencies between single tasks, e.g. the pre-processing of a document obviously needs to happen before the feature extraction. In order to shield the user from the complex "wiring" of tasks, DKPro TC currently provides three pre-defined workflows: *Train/Test*, *Cross-Validation*, and *Prediction* (on unseen data). Each workflow supports the feature modes described above: document, unit/sequence, and pair.

The user is still able to control the behavior of the workflow by setting parameters, most importantly the sources of input data, the set of feature extractors, and the classifier to be used. Internally, each parameter is treated as a single dimension in the global *parameter space*. Users may provide more than one value for a certain parameter, e.g. specific feature sets or several classifiers. The workflow engine will automatically run all possible parameter value combinations (a process called *parameter sweeping*).

3.2 Reading Input Data

Input data for supervised learning tasks comes in myriad different formats which implies that reading data cannot be standardized, but needs to be handled individually for each data set. However, the internal processing should not be dependent on the input format. We therefore use the Common Analysis Structure (CAS), provided by the Apache Unstructured Information Management Architecture (UIMA), to represent input documents and annotations in a standardized way.

Under the UIMA model, reading input data means to transform arbitrary input data into a CAS representation. DKPro TC already provides a wide range of readers from UIMA component repositories such as DKPro Core.[2] The reader also needs to assign to each classification unit an *outcome* attribute that represents the relevant label (single-label), labels (multi-label), or a real value (regression). In unit/sequence mode, the reader additionally needs to mark the units in the CAS. In pair mode, a pair of texts (instead of a single document) is stored within one CAS.

3.3 Preprocessing

In this step, additional information about the document is added to the CAS, which efficiently stores large numbers of stand-off annotations. In pair mode, the preprocessing is automatically applied to both documents.

DKPro TC allows the user to run arbitrary UIMA-based preprocessing components as long as they are compatible with the DKPro type system that is currently used by DKPro Core and EOP.[3] Thus, a large set of ready-to-use preprocessing components for more than ten languages is available, containing e.g. sentence boundary detection, lemmatization, POS-tagging, or parsing.

3.4 Feature Extraction

DKPro TC ships a constantly growing number of feature extractors. Feature extractors have access to the document text as well as all the additional information that has been added in the form of UIMA stand-off annotations during the preprocessing step. Users of DKPro TC can add customized feature extractors for particular use cases on demand.

Among the ready-to-use feature extractors contained in DKPro TC, there are several ones extracting grammatical information, e.g. the plural-singular ratio or the ratio of modal to all verbs. Other features collect information about stylistic cues of a document, e.g. the number of exclamations or the type-token-ratio. DKPro TC is able to extract n-grams or skip n-grams of tokens, characters, and POS tags.

Some feature extractors need access to information about the entire document collection, e.g. in

[2] http://dkpro-core-asl.googlecode.com

[3] http://hltfbk.github.io/Excitement-Open-Platform/

order to weigh lexical features with tf.idf scores. Such extractors have to declare that they depend on collection level information and DKPro TC will automatically include a special task that is executed before the actual features are extracted. Depending on the feature mode which has been configured, DKPro TC will extract information on document level, unit- and/or sequence-level, or document pair level.

DKPro TC stores extracted features in its internal feature store. When the extraction process is finished, a configurable *data writer* converts the content from the feature store into a format which can be handled by the utilized machine learning tool. DKPro TC currently ships data writers for the Weka (Hall et al., 2009), Meka[4], and Mallet (McCallum, 2002) frameworks. Users can also add dedicated data writers that output features in the format used by the machine learning framework of their choice.

3.5 Supervised Learning

For the actual machine learning, DKPro TC currently relies on Weka (single-label and regression), Meka (multi-label), and Mallet (sequence labeling). It contains a task which trains a freely configurable classifier on the training data and evaluates the learned model on the test data.

Before training and evaluation, the user may apply dimensionality reduction to the feature set, i.e. select a limited number of (expectedly meaningful) features to be included for training and evaluating the classifier. DKPro TC uses the feature selection capabilities of Weka (single-label and regression) and Mulan (multi-label) (Tsoumakas et al., 2010).

DKPro TC can also predict labels on unseen (i.e. unlabeled) data, using a trained classifier. In that case, no evaluation will be carried out, but the classifier's prediction for each document will be written to a file.

3.6 Evaluation and Reporting

DKPro TC calculates common evaluation scores including accuracy, precision, recall, and F_1-score. Whenever sensible, scores are reported for each individual label as well as aggregated over all labels. To support users in further analyzing the performance of a classification workflow, DKPro TC outputs the confusion matrix, the actual predictions assigned to each document, and a ranking of the most useful features based on the configured feature selection algorithm. Additional task-specific reporting can be added by the user.

As mentioned before, a major goal of DKPro TC is to increase the replicability of NLP experiments. Thus, for each experiment, all configuration parameters are stored and will be reported together with the classification results.

4 Tweet Classification: A Use Case

We now give a brief summary of what a supervised learning task might look like in DKPro TC using a simple Twitter sentiment classification example. Assuming that we want to classify a set of tweets either as "emotional" or "neutral", we can use the setup shown in Listing 1. The example uses the Groovy programming language which yields better readable code, but pure Java is also supported. Likewise, a DKPro TC experiment can also be set up with the help of a configuration file, e.g. in JSON or via Groovy scripts.

First, we create a workflow as a `BatchTaskCrossValidation` which can be used to run a cross-validation experiment on the data (using 10 folds as configured by the corresponding parameter). The workflow uses `LabeledTweetReader` in order to import the experiment data from source text files into the internal document representation (one document per tweet). This reader adds a UIMA annotation that specifies the gold standard classification outcome, i.e. the relevant label for the tweet. In this use case, preprocessing consists of a single step: running the `ArkTweetTagger` (Gimpel et al., 2011), a specialized Twitter tokenizer and POS-tagger that is integrated in DKPro Core. The feature mode is set to document (one tweet per CAS), and the learning mode to single-label (each tweet is labeled with exactly one label), cf. Table 1.

Two feature extractors are configured: One for returning the number of hashtags and another one returning the ratio of emoticons to tokens in the tweet. Listing 2 shows the Java code for the second extractor. Two things are noteworthy: (i) document text and UIMA annotations are readily available through the JCas object, and (ii) this is really all that the user needs to write in order to add a new feature extractor.

The next item to be configured is the `WekaDataWriter` which converts the internal fea-

[4] http://meka.sourceforge.net

```
BatchTaskCrossValidation batchTask = [
    experimentName: "Twitter-Sentiment",
    preprocessingPipeline: createEngineDescription(ArkTweetTagger), // Preprocessing
    parameterSpace: [ // multi-valued parameters in the parameter space will be swept
        Dimension.createBundle("reader", [
            readerTrain: LabeledTweetReader,
            readerTrainParams: [LabeledTweetReader.PARAM_CORPUS_PATH, "src/main/resources/tweets.txt"]]),
        Dimension.create("featureMode", "document"),
        Dimension.create("learningMode", "singleLabel"),
        Dimension.create("featureSet", [EmoticonRatioExtractor.name, NumberOfHashTagsExtractor.name]),
        Dimension.create("dataWriter", WekaDataWriter.name),
        Dimension.create("classificationArguments", [NaiveBayes.name, RandomForest.name])],
    reports: [BatchCrossValidationReport], // collects results from folds
    numFolds: 10];
```

Listing 1: Groovy code to configure a DKPro TC cross-validation BatchTask on Twitter data.

```
public class EmoticonRatioFeatureExtractor
extends FeatureExtractorResource_ImplBase implements DocumentFeatureExtractor
{
    @Override
    public List<Feature> extract(JCas annoDb) throws TextClassificationException {
        int nrOfEmoticons = JCasUtil.select(annoDb, EMO.class).size();
        int nrOfTokens = JCasUtil.select(annoDb, Token.class).size();
        double ratio = (double) nrOfEmoticons / nrOfTokens;
        return new Feature("EmoticonRatio", ratio).asList();
    }
}
```

Listing 2: A DKPro TC document mode feature extractor measuring the ratio of emoticons to tokens.

ture representation into the Weka ARFF format. For the classification, two machine learning algorithms will be iteratively tested: a Naive Bayes classifier and a Random Forest classifier. Passing a list of parameters into the parameter space will automatically make DKPro TC test all possible parameter combinations. The classification task automatically trains a model on the training data and stores the results of the evaluation on the test data for each fold on the disk. Finally, the evaluation scores for each fold are collected by the `BatchCrossValidationReport` and written to a single file using a tabulated format.

5 Related Work

This section will give a brief overview about tools with a scope similar to DKPro TC. We only list freely available software, most of which is open-source. Unless otherwise indicated, all of the tools are written in Java.

ClearTK (Ogren et al., 2008) is conceptually closest to DKPro TC and shares many of its distinguishing features like the modular feature extractors. It provides interfaces to machine learning libraries such as Mallet or libsvm, offers wrappers for basic NLP components, and comes with a feature extraction library that facilitates the development of custom feature extractors within the UIMA framework. In contrast to DKPro TC, it is rather designed as a programming library than a customizable research environment for quick experiments and does not provide predefined text classification setups. Furthermore, it does not support parameter sweeping and has no explicit support for creating experiment reports.

Argo (Rak et al., 2013) is a web-based workbench with support for manual annotation and automatic analysis of mainly bio-medical data. Like DKPro TC, Argo is based on UIMA, but focuses on sequence tagging, and it lacks DKPro TC's parameter sweeping capabilities.

NLTK (Bird et al., 2009) is a general-purpose NLP toolkit written in Python. It offers components for a wide range of preprocessing tasks and also supports feature extraction and machine learning for supervised text classification. Like DKPro TC, it can be used to quickly setup baseline experiments. As opposed to DKPro TC, NLTK lacks a modular structure with respect to preprocessing and feature extraction and does not support parameter sweeping.

Weka (Hall et al., 2009) is a machine learning framework that covers only the last two steps of DKPro TC's experimental process, i.e. machine learning and evaluation. However, it offers no dedicated support for preprocessing and feature generation. Weka is one of the machine learning frameworks that can be used within DKPro TC for actual machine learning.

Mallet (McCallum, 2002) is another machine

learning framework implementing several supervised and unsupervised learning algorithms. As opposed to Weka, is also supports sequence tagging, including Conditional Random Fields, as well as topic modeling. Mallet can be used as machine learning framework within DKPro TC.

Scikit-learn (Pedregosa et al., 2011) is a machine learning framework written in Python. It offers basic functionality for preprocessing, feature selection, and parameter tuning. It provides some methods for preprocessing such as converting documents to tf.idf vectors, but does not offer sophisticated and customizable feature extractors for textual data like DKPro TC.

6 Summary and Future Work

We have presented DKPro TC, a comprehensive and flexible framework for supervised learning on textual data. DKPro TC makes setting up experiments and creating new features fast and simple, and can therefore be applied for rapid prototyping. Its extensive logging capabilities emphasize the replicability of results. In our own research lab, DKPro TC has successfully been applied to a wide range of tasks including author identification, text quality assessment, and sentiment detection.

There are some limitations to DKPro TC which we plan to address in future work. To reduce the runtime of experiments with very large document collections, we want to add support for parallel processing of documents. While the current main goal of DKPro TC is to bootstrap experiments on new data sets or new applications, we also plan to make DKPro TC workflows available as resources to other applications, so that a model trained with DKPro TC can be used to automatically label textual data in different environments.

Acknowledgments

This work has been supported by the Volkswagen Foundation as part of the Lichtenberg-Professorship Program under grant No. I/82806, and by the Hessian research excellence program "Landes-Offensive zur Entwicklung Wissenschaftlich-ökonomischer Exzellenz" (LOEWE) as part of the research center "Digital Humanities". The authors would like give special thanks to Richard Eckhart de Castilho, Nicolai Erbs, Lucie Flekova, Emily Jamison, Krish Perumal, and Artem Vovk for their contributions to the DKPro TC framework.

References

S. Bird, E. Loper, and E. Klein. 2009. *Natural Language Processing with Python*. O'Reilly Media Inc.

R. Eckart de Castilho and I. Gurevych. 2011. A Lightweight Framework for Reproducible Parameter Sweeping in Information Retrieval. In *Proc. of the Workshop on Data Infrastructures for Supporting Information Retrieval Evaluation*, pages 7–10.

D. Ferrucci and A. Lally. 2004. UIMA: An Architectural Approach to Unstructured Information Processing in the Corporate Research Environment. *Natural Language Engineering*, 10(3-4):327–348.

A. Fokkens, M. van Erp, M. Postma, T. Pedersen, P. Vossen, and N. Freire. 2013. Offspring from Reproduction Problems: What Replication Failure Teaches Us. In *Proc. ACL*, pages 1691–1701.

K. Gimpel, N. Schneider, B. O'Connor, D. Das, D. Mills, J. Eisenstein, M. Heilman, D. Yogatama, J. Flanigan, and N. Smith. 2011. Part-of-speech tagging for Twitter: annotation, features, and experiments. In *Proc. ACL*, pages 42–47.

M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. Witten. 2009. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18.

A. McCallum. 2002. MALLET: A Machine Learning for Language Toolkit.

P. Ogren, P. Wetzler, and S. Bethard. 2008. ClearTK: A UIMA toolkit for statistical natural language processing. In *Towards Enhanced Interoperability for Large HLT Systems: UIMA for NLP workshop at LREC*, pages 32–38.

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. 2011. Scikit-learn: Machine Learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

R. Rak, A. Rowley, J. Carter, and S. Ananiadou. 2013. Development and Analysis of NLP Pipelines in Argo. In *Proc. ACL*, pages 115–120.

M. Surdeanu, J. Tibshirani, R. Nallapati, and C. Manning. 2012. Multi-instance multi-label learning for relation extraction. In *Proc. EMNLP-CoNLL*, pages 455–465.

G. Tsoumakas, I. Katakis, and I. Vlahavas. 2010. Mining Multi-label Data. *Transformation*, 135(2):1–20.