

Kuzman Ganchev
Department of Computer Science
University of Pennsylvania

Georgi Georgiev
Ontotext AD

ABSTRACT

The Edlin toolkit provides a machine learning framework for linear models, designed to be easy to read and understand. The main goal is to provide easy to edit working examples of implementations for popular learning algorithms. The toolkit is very brief, consisting of 40 Java classes with a total of about 2200 lines of code, of which about 20% are I/O and driver classes for examples. A version of Edlin has been integrated as a processing resource for the GATE architecture, and has been used for gene tagging, gene name normalization, named entity recognition in Bulgarian and biomedical relation extraction.

HIGHLIGHTS



- EDUCATION:** Easy to understand, hands on experience with working implementation.
- RESEARCH:** Easy to modify, easy to run experiments with modified learning algorithms.
- INDUSTRY:** State of the art learning, GATE integration, limited dependence on libraries, available in public maven repositories.

TRUTH IN ADVERTISING

Edlin is designed with a programmer in mind. Edlin is not a replacement for other toolkits such as Mallet, Weka, NLTK, LingPipe which target mostly end-users of learning algorithms.

- No graphical user interface; Edlin does not have any GUI: the programmer is intended to read and edit the code. A user who wants to try some learning algorithms on their data without programming would be better served by e.g. Weka.
- I/O and feature construction are left to the user; Edlin does not have a feature generation pipeline similar to Mallet's. The user has to implement feature extraction.

LINEAR MODELS

Let: $x \in X$ denote an input example and $y \in Y$ range over possible labels for x .

LINEAR MODELS: $h(x) = \arg \max_y f(x, y) \cdot w$

where $f(x, y)$ is a feature function and w is a parameter vector
· denotes inner product.

Examples
Naive Bayes
Maximum Entropy
Perceptron
MIRA
AdaBoost
Conditional Random Fields

The feature function $f(x, y)$ is specified by the user, the parameter vector w is learned.

BRIEF, LEGIBLE IMPLEMENTATION

```

public LinearTagger batchTrain(ArrayList<SequenceInstance> trainingData){
    LinearTagger w = new LinearTagger(xAlphabet, yAlphabet, fxy);
    LinearTagger theta = null;
    if (performAveraging)
        theta = new LinearTagger(xAlphabet, yAlphabet, fxy);
    for (int iter = 0; iter < numIterations; iter++) {
        for (SequenceInstance inst : trainingData) {
            int[] yhat = w.label(inst.x);
            // if y = yhat then this update won't change w.
            StaticUtils.plusEquals(w.w, fxy.apply(inst.x, inst.y));
            StaticUtils.plusEquals(w.w, fxy.apply(inst.x, yhat), -1);
            if (performAveraging)
                StaticUtils.plusEquals(theta.w, w.w, 1);
        }
    }
    if (performAveraging) return theta;
    return w;
}
  
```

FIGURE: As an example, we have the implementation of the Perceptron algorithm, with and without averaging (training procedure is 13 lines of code). The number of lines of code for some learning algorithms and implementations are shown below.

Learning Algorithm	Edlin LOC	Mallet LOC	Weka LOC
Perceptron	43	—	273
Naive Bayes	61	221+160	164
Maximum Entropy	97	175+265	—
MIRA	79	—	—
AdaBoost	141	53+101	377
CRF	182+68	207+1513+219	—
structured Perceptron	37+68	—	—
structured MIRA	77+12+68	—	—

TABLE Counts of lines of code (excluding comments and blank lines) for implementations of different learning algorithms in Edlin as well as the Mallet toolkit and Weka.

GATE INTEGRATION

GATE is a framework for engineering NLP applications along with a graphical development environment for developing components. GATE divides language processing resources into language resources, processing resources, and graphical interfaces. We have integrated a version of Edlin into the GATE framework as a set of processing resources, by defining interfaces in Edlin for training, classification, and sequence tagging. These interfaces are used to communicate between Edlin's machine learning implementations and the concrete implementations of tagger and classifier processors in GATE. The integration allows Edlin to be used for robust, complex text processing applications, relying on GATE processors such as tokenizers, sentence splitters and parsers, to preprocess the data. The integration also makes it easy to pipeline Edlin-trained linear models using the GATE infrastructure for processing pipelines. Since Edlin has very readable code, this makes it easy for a researcher or engineer to try a modified learning algorithm if they already use the GATE framework.

EXAMPLE APPLICATION: BIOMEDICAL RELATION EXTRACTION

Application: BioNLP 2009 shared task.

goal: relation extraction from biomedical text

"... phosphorylation of [TRAF2] ..." →
event:Phosphorylation; trigger:phosphorylation theme:TRAF2

Provided: gene annotations, list of event types

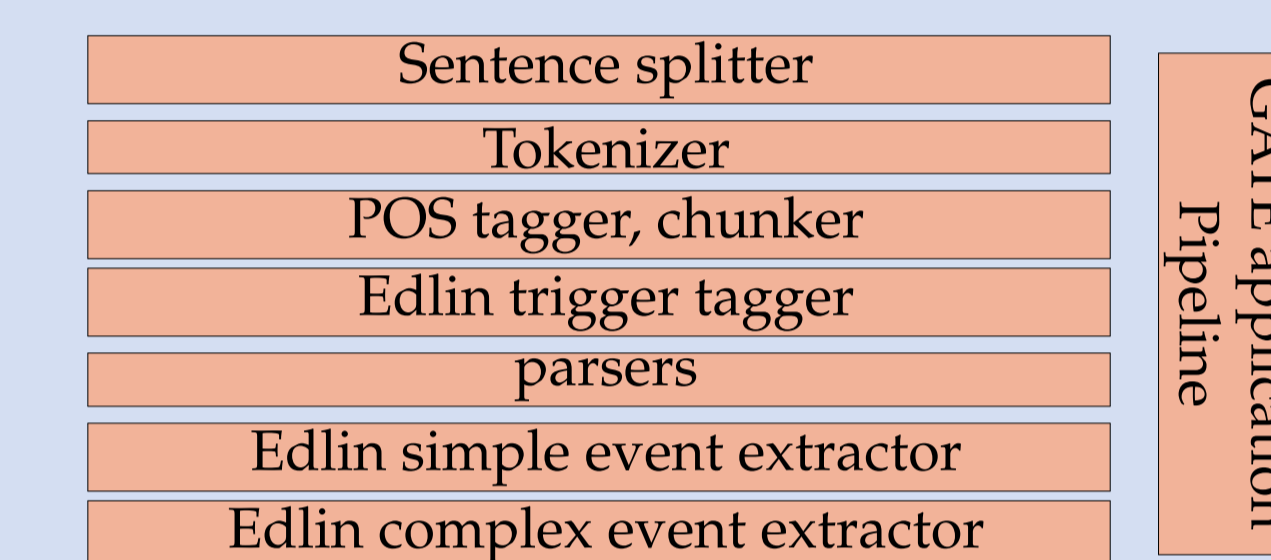


FIGURE: Our event extraction pipeline, stringing together different GATE text processors. The first stages of the pipeline as well as the parsers are included in order to create features useful for later stages.

1 Gene and Trigger Tagging

Trigger chunks are words and phrases that describe the events linking proteins. For example "binds" is such a trigger word that would link two or more genes in a Binding event.

We used the Edlin GATE integration described in to create one GATE processing resource that trains an Edlin linear sequence model and another that uses that Edlin sequence model to tag trigger chunks.

Both processors work in a pipeline with GATE preprocessors including a tokenizer, sentence splitter, POS tagger and chunker. Because Edlin represents linear models trained using different algorithms in the same way it was easy for us to compare different learning algorithms for the task. For this application tagger recall is an upper bound on system performance, and used MIRA with a loss function designed to achieve high recall since that performed best.

2 Relation Extraction

We separate the process of relation extraction into two stages: in the first stage, we generate events corresponding to relations between a trigger word and one or more proteins (simple events), while in the second stage, we generate events that correspond to relations between trigger words, proteins and simple events (we call the new events complex events).

The training of an Edlin linear model and classification using that model are again done using the Edlin-GATE integration, and are integrated in a GATE pipeline that now also includes dependency and phrase-structure parsers.

The uniform representation of linear models allowed us to compare different learning methods. We compared max entropy, perceptron and one-best MIRA, and again chose MIRA with a loss function designed to increase recall, since getting high recall was the most challenging part of the task. Finally, this tunable loss function was appealing for us because it allows application-specific tuning. For example, a search might require high recall, but high precision might be more important for adding relations to a knowledge base.

The trigger tagging stage uses an Edlin GATE processor trained using one-best MIRA. We employ also a maximum entropy constituency parser (OpenNLP) and a dependency parser (MST parser). These components are also represented as GATE processors. In the last stage of the pipeline we use two components, one for simple and one for complex events, based on the classification version of one-best MIRA algorithm implemented in Edlin and used as GATE processors.