A RapidMiner framework for protein interaction extraction

Timur Fayruzov[1], George Dittmar[2], Nicolas Spence[2], Martine De Cock[1],
Ankur Teredesai[2]

[1]Ghent University, Ghent, Belgium
[2]University of Washington, Tacoma, USA

Abstract

During the last 10 years researchers have proposed many approaches to automatically extract protein-protein interactions (PPIs) from scientific papers. However, the lack of a unified implementation and evaluation framework complicates the development of new PPI methods and makes the comparison of existing methods difficult. In this paper we present such a framework that is built as an extension of RapidMiner. Next to providing a platform for evaluating and comparing different text mining methods for PPI extraction, the framework can also be leveraged to build a standalone application that can accumulate the mined interaction data and make it available to biologists for querying. We illustrate the utility of the developed framework with a prototype of a protein interaction search engine.

1 Introduction

Biologists regularly search PubMed[1] to find relevant papers that outline new gene and protein interactions relevant for their research. This task is extremely cumbersome today since the amount of information available when querying for protein names to look for interactions is simply overwhelming. The PubMed index itself contains entries for over 16 million references, most of which contain either abstracts or full texts, and it grows at a tremendous rate of several thousands of publications per week[2]. This makes it impossible for any biologist to keep track of every new result relevant to his study by manually browsing through all new entries. Existing conventional information retrieval tools provide little assistance with this task; for example, the search features provided by PubMed do not allow a user to search specifically for interactions between proteins. The co-occurence based methods of mainstream search engines generate a large amount of superfluous results which do contain the mentioned proteins but not the sought-after interactions between them, resulting in low precision.

The text mining community fully recognizes this problem and is spending a lot of effort on solving the tasks of named entity recognition (NER, i.e., automatically detecting words in abstracts or full texts that correspond to protein names) and protein-protein interaction detection (PPID, i.e., automatically deciding whether a sentence containing two protein names describes an interaction between those proteins or not), see e.g. [1]. However, most of the developed methods do not follow any standard specification, which means that they are implemented in different languages, with different data preprocessing techniques and for different platforms. Additionally, some researchers evaluate their systems on non publicly available datasets, which makes it difficult to reproduce their experiments. Moreover, the NER and PPID components that are needed to build a full interaction extraction pipeline are often developed independently of each other in a research context and thus are not compatible. This hampers the use of the developed methods in the real world, e.g. to build standalone applications containing the NER and PPID methods for the benefit of the biological community.

The objective of this work is to take the first steps towards a flexible unified text mining framework for extracting gene and protein interactions based on RapidMiner architectural principles. This framework aims to provide an easy way:

[1] http://www.ncbi.nlm.nih.gov/pubmed/

[2] http://www.nlm.nih.gov/pubs/factsheets/medline.html

1. to build and evaluate PPI extraction systems, and to simplify the use of different datasets for training and testing purposes
2. to build a stand-alone application server that is capable of processing and accumulating PPI data continuously, and disclosing this data to the user through a query mechanism

In Section 2 we describe the problem of PPI extraction in more detail, as well as a domain specific data structure that is central to our approach. In Section 3 we present our RapidMiner based framework for PPI extraction that can be used both for evaluation purposes (Section 3.2) as well as to build a stand-alone application (Section 3.1). The framework is designed in such a way that a developer can easily plug in his own text mining methods; we illustrate this in Section 4 with the description of a protein interaction search engine called PRISE that is built using our RapidMinder based framework for PPI extraction.

2 Protein-protein interaction extraction

To explain the PPI extraction task in more detail let us consider the following example sentence:

> "In the shaA mutant, sigma(H)-dependent expression of spo0A and spoVG at an early stage of sporulation was sensitive to external NaCl."

This sentence contains 4 protein names, namely *shaA*, *sigma(H)*, *spo0A* and *spoVG*. The first step of the PPI extraction task is to identify these names in the text. Assume that the 4 proteins in this sentence were successfully recognized by a NER method. These proteins can be combined into 6 unordered pairs, namely *shaA-sigma(H)*, *shaA-spo0A*, *shaA-spoVG*, *sigma(H)-spo0A*, *sigma(H)-spoVG*, and *spo0A-spoVG*. A protein pair is a positive instance if the original sentence expresses an interaction between the members of this pair, and a negative instance if they just co-occur in the sentence. In the example above, there are two positive instances, namely *sigma(H)-spo0A* and *sigma(H)-spoVG*, while the other 4 instances are negative. The second step in PPI extraction is to detect which pairs are positive instances; this phase is called PPID.

Supervised machine learning techniques can be used to address both NER and PPID. In general, for the NER task, a 'window' of several words around a candidate protein name or even a whole sentence is represented as a vector of features, such as the words themselves, orthographic features (capitalization, hyphenation, alphanumeric characters), greek letters, punctuation etc. These feature vectors are further used to train a classifier based on a conventional machine learning technique. The results of NER can then be used to solve the PPID task.

To address the PPID problem, one needs to build representations for candidate interactions, i.e., for protein pairs. Such a protein pair representation should include information (features) from the sentence that can be used to distinguish between positive and negative instances. The difference with the NER task is that in PPID one already has a fixed pair of terms in a sentence and the task is not to detect an entity name, but to extract a (potentially implicit) relationship between these terms. This means that the strategy to build a feature vector for this task is likely to be different and potentially can include more elaborate features such as syntactic dependencies between words or other structural information. After the feature vector construction though, just like in the case of NER, a conventional machine learning based classifier can be used to complete the PPID task.

As we will see in the following section, a system to solve the PPI extraction problem as a whole can be decomposed into independent modules that generally fit well in the RapidMiner paradigm. The main difference between the NER and PPID tasks from the structural point of view is the way to obtain the feature vector representation of an entity/interaction. Thus abstracting the operations that convert a sentence into a feature vector representation gives us a flexible framework that is suitable to build both NER and PPID components. To this end we have developed a set of RapidMiner operators that simplify this task and that provide extra functionalities specific to PPI extraction. Before describing the actual operators, let us review some of the RapidMiner terminology and the design choices we made to build our framework.

`Operator` and `OperatorChain` are terms used in RapidMiner to denote units that perform certain operations, such as reading data from a file, training a classifier, etc. An `OperatorChain` can contain other operators as its parts. A `Process` is a special case of an `OperatorChain` that is used to denote

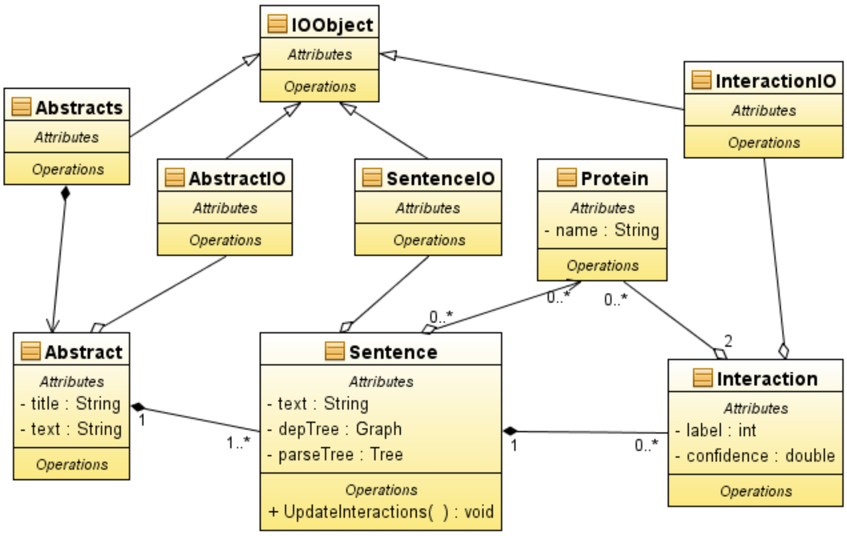

Figure 1: Diagram of the domain data structure. An abstract can contain 1 or more sentences; a sentence can contain 0 or more proteins and 0 or more interactions; an interaction involves exactly 2 proteins. The additional wrapper classes (with IO suffix) are intended to pass data between RapidMiner operators.

a set of operators aimed to solve a certain task, e.g. PPI extraction. Every `Operator` has input and output ports that are used to pass data around and produce new data as the result of the `Operator`'s execution. Typically, data in RapidMiner is stored in a specific data structure called `ExampleTable` that is connected to a persistent storage (database) containing all instances. The actual operators use `ExampleSets` which are views over an `ExampleTable` that may represent partial information from this `ExampleTable`. An `ExampleSet` consists of `Examples` that usually correspond to feature vectors representing instances. Moreover, it is possible to pass user-defined objects between operators if these objects extend the `IOObject` class. This is an important feature that we use in our framework as it allows us to build a domain model and operators that act directly on this model as we explain below.

To better accommodate the PPI extraction domain we have developed our own data structure to represent the textual data on different granularity levels, rather than convert it to RapidMiner's `ExampleTable` immediately. RapidMiner assumes that every instance in the dataset is represented as an `Example`, but merely loading text in a RapidMiner `Process` does not provide us with a desirable feature vector. Moreover, as we do not want to restrict a developer to any certain set of features, we want to provide a flexible mechanism for text to feature vector conversion that a developer can implement depending on his needs.

The main blocks of information we use are texts (article abstracts), sentences, gene and protein names, and interactions themselves. A class diagram for our data structure is shown in Figure 1. An abstract can contain 1 or more sentences, a sentence may contain 0 or more proteins, and 0 or more interactions, and an interaction should contain exactly 2 proteins. To pass the data between operators we have implemented additional wrapper classes (with IO suffix) that can be passed between the RapidMiner operators.

Another design choice is to use an external database to store the interaction related data. It is important to distinguish between RapidMiner's internal database used to store `ExampleTables` and the database we use. In RapidMiner, a database is a medium to store the dataset loaded in a `Process`, and this database is used to operate with data exclusively within this `Process`, i.e. it is a transient storage[3]. In our case, to build a stand-alone PPI extraction application, we need a persistent storage that contains the same information no matter what the `Process`'s structure is.

[3] `http://rapid-i.com/wiki/index.php?title=Data_core`

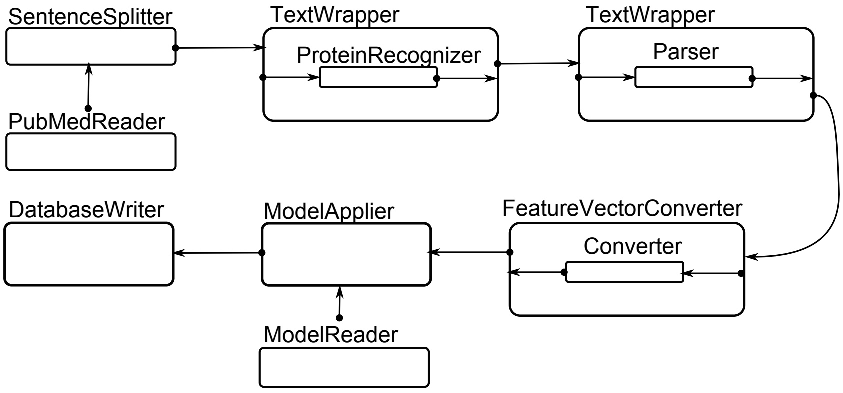

Figure 2: Diagram of a PPI extraction system that reads articles from Medline (PubMed) and stores the mined interactions in a database. The labels correspond to RapidMiner operators.

3 RapidMiner extension for PPI

3.1 Application mode

The starting point for PPI extraction are articles from the Medline text database. PubMed provides a query interface to this library that allows to retrieve information from Medline[4]. To access this data we have built a PubmedReader operator that queries the online service with parameters such as keywords and date range, and provides a list of abstracts as output. These are stored in an instance of the data structure from Figure 1. The initial text mining step to perform next is to split the text of these abstracts into sentences. To perform this preprocessing step one can use an existing sentence splitter that takes text as input and outputs a corresponding list of sentences. To encapsulate this component we have built a SentenceSplitter operator that takes an instance of the data structure containing a list of Abstract objects as input and returns the same data structure instance that additionally contains a list of Sentence objects for every abstract.

The next step in the application pipeline is to determine the entities (gene and protein names) which are potential interaction arguments. Here again one can use an existing NER tool that processes texts sentence by sentence and annotates protein names. To this end we have developed a RapidMiner operator TextWrapper that has a parameter Granularity which regulates the granularity level of the iterator that reads the data (per abstract, per sentence or per (potential) interaction). This operator is connected to the ProteinRecognizer operator that encapsulates an external NER tool. ProteinRecognizer converts the format produced by the NER tool to the data structure we use, which now contains abstracts with sentences that contain annotated gene and protein names. If furhter syntactic preprocessing needs to be done, a natural language parser can be plugged in. Typically, parsers accept a sentence and output a parsing structure for this sentence, in this case a parser can also be encapsulated with TextWrapper to ensure correct input/output processing.

For the PPID step we do not follow the 'wrapping' strategy we used for NER and sentence splitting. Instead, we offer the developer the ability to take advantage of readily available classifiers implemented in RapidMiner, as well as provide him with a mechanism to prepare arbitrary feature vectors as it was discussed in Section 2. To this end we need to support the conversion of a candidate interaction to a representation as a feature vector. Note that typically more features beyond mere terms are induced from the text for complex information extraction tasks. The induced features may range from n-grams, lemmas, POS (part-of-speech) tags to more complex ones such as syntactic roles and dependency trees. To perform this conversion we have developed a FeatureVectorConverter operator chain that takes the instance of data representation structure as input and processes it candidate interaction by candidate interaction in its inner chain, converting every protein pair to an Example object that represents a feature vector for this protein pair. The operator Converter in the inner chain that performs the actual conversion should be implemented by the developer. In the application mode we assume that the classifier

[4] http://www.nlm.nih.gov/databases/databases_medline.html

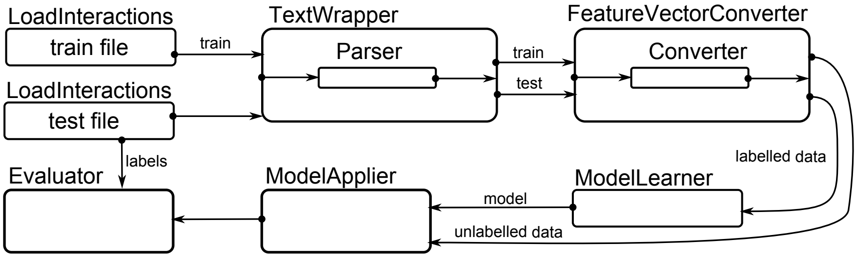

Figure 3: Diagram of the evaluation of a PPID method on an annotated benchmark dataset.

was trained elsewhere (see Section 3.2) and thus it is being read by a standard `ModelReader` operator and applied by a `ModelApplier` operator.

To complete the picture we have implemented a `DatabaseWriter` operator that provides access to the underlying interaction database. The input of this operator is the domain data structure that is being written to the database.

3.2 Evaluation mode

Our RapidMiner framework for PPI extraction can be used for evaluation purposes too. In this section we discuss an evaluation pipeline for PPID as shown in Figure 3; evaluation of NER follows a similar path.

Instead of pulling in unseen articles from Medline, in the evaluation mode one typically has annotated data that is used both for training and evaluation. One of the problems with annotated datasets is that they can come in different formats. Recently, a unified XML format for PPID was proposed in [8]. We have implemented a `LoadInteractions` operator that reads such an XML file and produces an instance of the domain data structure from Figure 1 as output. Alternatively, a developer can read the interactions from the domain database using the operator `DatabaseReader` or implement an operator that converts a proprietary data format in this data structure for further processing. Typically, PPID benchmark datasets contain texts that are split into sentences with interactions, gene and protein names already annotated. This means that the output of this operator is a fully populated instance of the data structure, as opposed to in the application case in Section 3.1 where several preprocessing steps were needed to arrive at this stage. The subsequent feature vector construction step is the same as in Section 3.1.

The next steps involve training and evaluating a classifier. RapidMiner provide a number of popular classification algorithms that can be readily applied through learner operators as well as allow for inclusion of new algorithms by implementing custom learner operators for them. In Figure 3 we abstract from any particular algorithm and denote the learner as `ModelLearner`. The input to a learner is an `ExampleSet` and the output is RapidMiner `Model` that stores the parameters of the model learned by a classifier. This model can be further applied to classify new instances by means of `ModelApplier` operator. The `Evaluator` operator takes predictions from `ModelApplier` along with actual labels of instances and computes various evaluation metrics such as accuracy, and classification error values.

4 Use case

We have used the RapidMiner framework described above to build a PRotein Interaction Search Engine (PRISE)[5] that continuously scans PubMed entries and provides a web interface for querying the extracted interactions. To build the application pipeline we have used an implementation of a sentence splitter provided in the OpenNLP[6] framework and a NER tool called ABNER [9].

For PPID, we used the shortest path between two proteins in the dependency tree for candidate interaction representation as it was first proposed in [5]. However, dependency trees are not in our domain model, thus we needed to include an extra step to parse the sentences. We implemented a

[5] `http://prise.insttech.washington.edu`
[6] `http://opennlp.sourceforge.net/`

Parser operator that wraps the Stanford Parser [6] and enriches our data structure with parse and dependency trees, which potentially allow us to extract not only shortest paths but also POS tags, chunks, phrases etc. To build a classifier we used a TreeSVM implementation [7] of the SVMLight classifier [4]. TreeSVM takes trees in the Penn Treebank notation as input, thus the Converter operator merely extracts the shortest path from the sentence dependency tree for a given protein pair and returns an Example that contains a shortest path in the Penn Treebank notation as a single feature. Further, the Converter straightforwardly extends the feature vector with different types of shortest paths such as those that contain POS as nodes and those that contain only syntactic dependencies as it was described in [2].

Note that having a tree as a feature vector generally poses a problem in machine learning frameworks such as Weka [3] as trees are difficult to fit into the concept of feature vectors implemented there. However, in the modular framework built with RapidMiner we isolate this problem in one component, namely the classifier itself, and keep the rest of the pipeline untouched, which means that we can switch to another application setup by merely changing the Converter operator and the classifier.

5 Conclusions

In this paper we have examined the possibility of using RapidMiner to build a text mining pipeline for protein interaction extraction. The idea of such a pipeline fits well into the RapidMiner architecture. The implementation we devised significantly reduces the effort needed to substitute classifiers, use different document representations in feature space and perform experiments with different datasets.

References

[1] Special Section on BioCreative II.5, *IEEE/ACM Transactions on Computational Biology and Bioinformatics*, 7(3), 2010.

[2] T. Fayruzov, M. De Cock, C. Cornelis, and V. Hoste. Linguistic feature analysis for protein interaction extraction. *BMC Bioinformatics*, 10(1):374, 2009.

[3] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten. The Weka data mining software: an update. *SIGKDD Explorations*, 11(1):10–18, 2009.

[4] T. Joachims. Making large-scale support vector machine learning practical. pages 169–184, 1999.

[5] S. Kim, J. Yoon, and J. Yang. Kernel approaches for genic interaction extraction. *Bioinformatics*, 24:118–126, 2008.

[6] Klein, D., and Manning, C. D. Accurate unlexicalized parsing. In *ACL '03: Proceedings of the 41st Annual Meeting on Association for Computational Linguistics* (2003), pp. 423–430.

[7] A. Moschitti. Efficient convolution kernels for dependency and constituent syntactic trees. *ECML*, 2006.

[8] S. Pyysalo, A. Airola, J. Heimonen, J. Bjorne, F. Ginter, and T. Salakoski. Comparative analysis of five protein-protein interaction corpora. *BMC Bioinformatics, special issue*, 9(Suppl 3):S6, 2008.

[9] B. Settles. ABNER: an open source tool for automatically tagging genes, proteins and other entity names in text. *Bioinformatics*, 21(14):3191–3192, 2005.

[10] S. Van Landeghem, Y. Saeys, B. De Baets, and Y. Van de Peer. Extracting protein-protein interactions from text using rich feature vectors and feature selection. *SMBM 08*, 2008.