Brainy: A Machine Learning Library

Michal Konkol

Natural Language Processing Group
Department of Computer Science and Engineering
University of West Bohemia
Univerzitni 8
306 14 Plzen
Czech Republic
nlp.kiv.zcu.cz
konkol@kiv.zcu.cz

Abstract. Brainy is a newly created cross-platform machine learning library written in Java. It defines interfaces for common types of machine learning tasks and implementations of the most popular algorithms. Brainy utilizes a complex mathematical infrastructure which is also part of the library. The main difference compared to other ML libraries is the sophisticated system for feature definition and management. The design of the library is focused on efficiency, reliability, extensibility and simple usage. Brainy has been extensively used for research as well as commercial projects for major companies in Czech Republic and USA. Brainy is released under the GPL license and freely available from the project web page.

Keywords: Machine learning, software library.

1 Introduction

Machine learning is a branch of artificial intelligence which studies computer systems with the ability to learn without being explicitly programmed. Such systems are very different from standard rule-based systems where the knowledge is hand coded by humans. The machine learning systems have their strengths and weaknesses. On one hand, they can handle very complex problems, that are intractable by standard rule-based systems. On the other hand, the knowledge learned by machine learning system is almost never perfect and the rule-based systems can perform better for simple problems.

Machine learning is used for wide variety of tasks all around us. These tasks include natural language processing, weather forecasting, stock value forecasting, earthquake prediction, medicine decision making and many others. Generally, machine learning can be used for any complex problem where no other solution performs well.

There exist different paradigms of learning. The basic one is *supervised learning*. In supervised learning the training data are provided along with annotations which are telling the algorithm the right answers. The algorithm then tries to generalize the knowledge acquired from training data and also to still give the maximum of good answers. When the algorithm generalizes badly, it can often reproduce right answers for training data, but performs poorly for unseen data.

The second important paradigm is *unsupervised learning*. In unsupervised learning, the algorithm receives only data (without answers) and tries to find patterns in the data. There exists more learning paradigms, but the majority of tasks uses the presented two.

Machine learning represents a wide variety of algorithms. We will briefly describe the basic groups of these algorithms. The first group of algorithms is the *regression* group. Regression algorithms are designed for problems where the predicted variable is real valued. It analyzes the training data where the values are already annotated and then tries to find optimal values for unseen data. For example in the weather forecasting domain this algorithm can be used to predict the precipitation based on weather radar information.

A *classification* group is for problems where the output is categorical. These algorithms require some examples for each category and tries to find optimal decision boundary between these categories. An example from weather domain can be classification of days into sunny and cloudy categories. Another group closely related to classification is *sequence labelling*. It addresses problems where the category of the classified object depends not only on the data for this object but also on categories assigned to objects in its vicinity. This happens for example in industry where defective products are often produced in short series.

Another important group is *clustering*. Clustering can be seen as unsupervised learning version of classification. It analyzes data and assigns them to one of categories. Some algorithms need a predefined number of categories, some can choose the number of categories based on the data. Clustering can be used for example to automatically categorize news articles by topic.

All machine learning algorithms use the data to learn. The data for different domains are completely different and a universal representation have to be used. Machine learning introduces an abstraction called *feature* for this purpose. Each feature represents one property of the data object and translates it into numerical form. All features extracted from a data object form a *feature vector*. Features are often defined by *feature templates* (often also called features). One feature template is often responsible for many features. A typical feature template used in the natural language processing domain is a word and features are words themselves as “the” or “day”. The group of features (or feature templates) used for some task is called *feature set*.

Two groups of machine learning algorithms are specialized on choosing optimal feature set. *Feature selection* takes features defined by user as input and removes features with smallest impact on the result. Feature set can be often heavily reduced without loosing performance. *Feature induction* (sometimes extraction or selection) also starts with features from user, but it also tries to create combinations of these features and tries to select an optimal set. This task is much harder than simple feature selection, because the number of features can easily reach million and the number of combinations is huge.

Training of machine learning algorithms often requires a lot of time and resources. The training algorithms are often very sophisticated and are based on linear algebra, statistics and optimization. It is necessary to use highly optimized mathematical algorithms to reduce required resources. A naive implementation usually makes even smaller problems intractable.

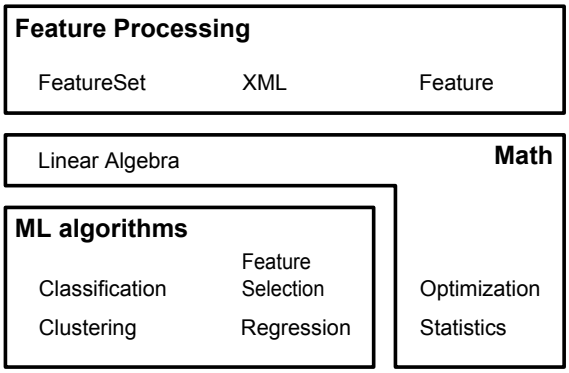
Brainy deals with all presented tasks and problems. The following sections give a basic overview of Brainy rather than an in-depth description. It should provide enough information to decide, whether Brainy is interesting for you.

2 Architecture

The framework consists of three main components – feature management, machine learning algorithms and mathematics (mainly statistics, optimization and linear algebra). We will briefly describe the interactions of the components and then each component more deeply.

Each machine learning task needs some data to learn. These data are usually represented as feature vectors. For this purpose, we have defined interfaces for matrices and vectors. These matrices and vectors serves as unified data interface between the machine learning and the feature management parts of the library. The feature management part defines interfaces for features and feature set. The machine learning part processes the data represented as vectors and matrices. It is heavily supported by the mathematical infrastructure. An overview is given by fig. 1.

Fig. 1. Main components of the library.



2.1 Feature management

The feature management is often overlooked in machine learning libraries. In our library we define interfaces for both features and feature set. The `Feature` interface represents a feature template and provides well-defined methods for changing data representation from user defined objects to numerical values. Each feature template then returns a vector indexed from 0 to n, where n_f is number of features defined on the template. Therefore the features are independent of each other.

The `FeatureSet` class manages the features. It processes all the user objects, combines the vectors from individual features into the final feature vector and creates a matrix representing the data from feature vectors. The feature set can be defined by an XML file. The file contains a list of features and their parametrization. This allows fast experimentation with multiple feature sets without changing the source code. It also helps you to keep track of your experiments and reproduce the results.

2.2 Mathematical infrastructure

The first part of the mathematical infrastructure is linear algebra. The machine learning algorithms can be often vectorized – transformed into vectors and matrices combined using standard linear algebra operations. We have defined interfaces for matrices and vectors. Different implementations of matrices allow very efficient execution, e.g. using sparse/dense matrices with different implementations, running in parallel, etc. It is easy to extend the library with new algorithms thanks to the support of linear algebra primitives.

Optimization algorithms form another part. We have defined interfaces for mathematical functions (e.g. `Function` for simple function, `DiffFunction` for differentiable function) and a `Minimizer` interface. For new method you only need to implement the cost function and use one of our implementations of Minimizer. We are usually using L-BFGS minimizer or some non-trivial version of gradient descent.

We have implemented interfaces for more mathematical primitives (e.g. random distributions, distances, similarities, etc.) and provided implementations for commonly used variants.

2.3 Machine learning algorithms

The main part of machine learning library are, of coarse, machine learning algorithms. We have defined interfaces for all major tasks – regression, classification, sequence labelling and clustering. We will describe these interfaces and their usage in section 3. All algorithms of the same type share the same interface and that allows easy experimentation with different algorithms and their combinations.

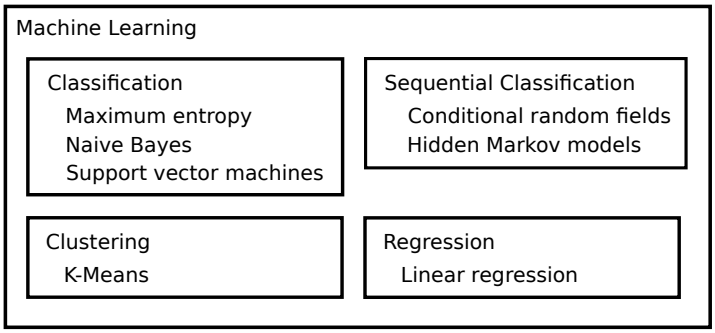
Fig. 2 shows detailed view of the machine learning component with listing of selected algorithms.

3 User view

In this section, we will briefly describe the library from the user view. For detailed description see the tutorial on the project website (sec. 6).

First step for any machine learning task is preparation of data. Our machine learning library supports two approaches. The machine learning algorithms work solely with matrices and vectors. The first possibility is to directly create these matrices using any way that fits your needs. The second possibility is to use our feature management system.

Fig. 2. The machine learning component.



3.1 Feature management

The feature management system supports easy experimentation with features. The basic interface is `Feature` which represents a feature template. With feature template we mean set of very similar features with the same semantics, e.g. in the field of natural language processing the feature template 'word' consists of features for individual words (current word is 'wood', 'steel', etc.).

The `Feature` interface has two important methods. The `train()` method is used for training the feature from training data, e.g. the previously mentioned 'word' feature needs to learn the words. The `extractFeature()` method translates the user objects to numeric representation.

After defining features you can create a feature set (class `FeatureSet`). The feature set can be created programatically or by XML configuration file. The XML file is very useful for testing multiple feature sets. Listing 1 shows both ways of feature set creation and its usage. Note that `FeatureSet` class is generic. The generic type represents the object you want to classify – in our example the `String[]` represents a document. The `trainingObjects` object is a list-like structure with objects for classification.

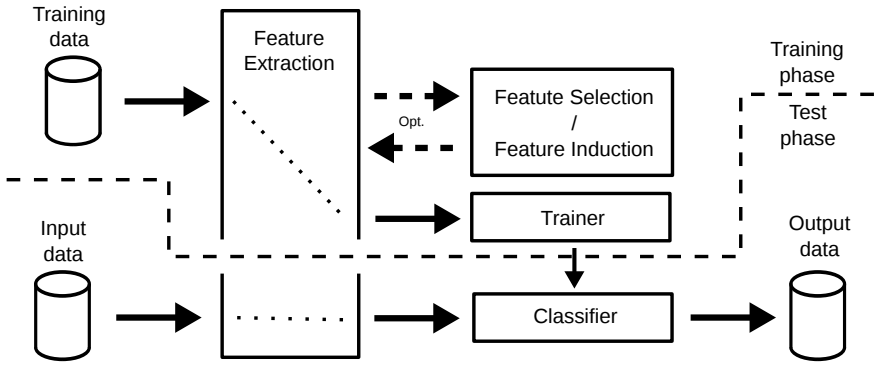
Listing 1. Creation of feature sets.

```
FeatureSet<String[]> set = new FeatureSet<String[]>();
set.add(new WordFeature(0));
set.train(trainingObjects);

set = new FeatureSet<String[]>("myFeatureSet.xml");
set.train(trainingObjects);

DoubleMatrix data = set.getData(trainingObjects);
IntVector labels = set.getLabels(trainingObjects);
```

3.2 Classification

The schema of classification task is on fig. 3. After data preparation a classifier trainer object is created. This object represents a method for training of chosen classifier, e.g. maximum entropy [1] classifier can be trained by L-BFGS [2] method. The trainer object then returns a classifier object, which is ready to classify unseen data.

Fig. 3. The classification task.



Listing 2 shows an example code for classifier training and usage. We have used a maximum entropy classifier, where the `MaxEntLBFGSTrainer` is the trainer class and `MaxEnt` is the classifier class. As we said previously the data can be prepared in multiple ways. Before classification you need to create `Results` object, which is filled by results of the classifier. This allows you to reuse this object, e.g. when you are searching for optimal parameters of the classifier.

Listing 2. Creation and usage of maximum entropy classifier.

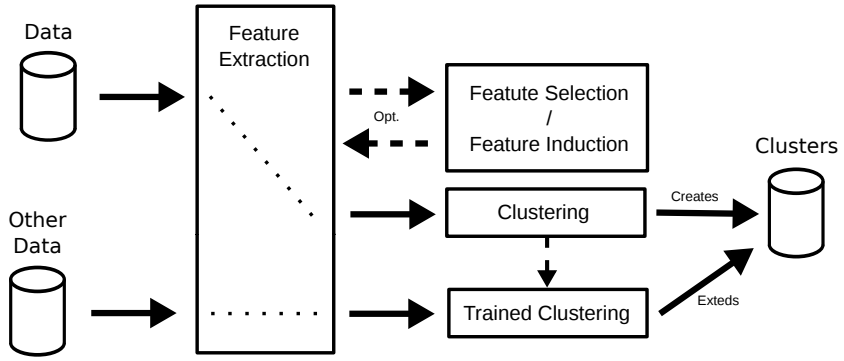
```
DoubleMatrix trainingData = ...;
IntVector trainingLabels = ...;
SupervisedClassifierTrainer trainer = new MaxEntLBFGSTrainer();
Classifier classifier = trainer.train(trainingData,
    trainingLabels, numLabels);

DoubleMatrix data = ...;
Results results = BasicResults.create(numLabels,data.columns());
classifier.classify(data, results);
```

3.3 Clustering

The scheme for clustering is on fig. 4. The objects used for clustering have different semantics. The clustering method is represented by single object which implements

the `Clustering` interface, e.g. object of `K-Means` class. This object clusters the data. Some methods also support an optional function – they return an object which represents a trained version of the clustering method. This object also implements the `Clustering` interface and is able to add additional (previously unseen) objects to the previously created clusters without the need of clustering all data objects from start. In the case of K-Means [3] the trained version computes the distances between additional vectors and centroids and adds them to the cluster with shortest distance.

Fig. 4. The clustering task.



An example of clustering can be seen on listing 3.

Listing 3. Creation and usage of K-Means clustering.

```
DoubleMatrix data = ...;
Clustering kmeans = new KMeans(means.length,
    new EuclideanDistance());
Clustering trainedKmeans = kmeans.cluster(data, results);

DoubleMatrix otherData = ...;
trainedKmeans.cluster(data, results);
```

4 Related systems

In this section we will briefly describe libraries and frameworks with similar focus. We have done some artificial tests on them and compared with our library. The tests had almost the same results for all libraries (comparing the same algorithms). These tests represents only simple problems. Their results prove that the libraries are not faulty and their performance is more or less the same for simple problems, but do not show well the differences between algorithms on real and more complex problems. The proper way

of comparing these libraries would be testing on multiple real problems from multiple fields, but it is out of the scope of this article.

The most similar libraries are Mallet[1] [4] and Java-ML[2] [5]. They are both machine learning libraries written in Java. The biggest difference between these libraries and our library is the architecture. The interfaces for machine learning tasks are different. They heavily differ in the way they prepare and represent the data. These differences in architecture forces the user to use different concepts and the appropriate library should be chosen based on the task, compatibility with other systems and personal preference.

Weka[3] [6] and Apache Mahout[4] are another machine learning frameworks worth mentioning, but they differ in purpose compared to the previously mentioned libraries. The Weka itself states "Weka is a collection of machine learning algorithms for data mining tasks.", so it is not primarily intended as a general machine learning library. The standard usage is through GUI and CLI, while our library provides only API. The Weka framework generally uses higher level of abstraction then our library.

Apache Mahout is focused on very big problems. The framework is based on the Apache Hadoop framework for distributed computing. Our library can be parallelized to some extent, but it is limited by one cluster. Hadoop provides much more complex infrastructure for distributed computing then our library, e.g. node error recovery, etc. So the main difference is the basic concept or targets of the library.

5 Verification

All algorithms provided by the library are tested on very simple machine learning tasks using the jUnit framework. The library is also used by students for assignments and theses.

The library was extensively used for research by the Natural Language Processing Group at University of West Bohemia. So far, the library was used in two main directions of research – named entity recognition (NER) and sentiment analysis (SA).

The NER tool is based on Brainy and GATE[5]. The first version of our NER system was based on the maximum entropy classifier [7]. Current version is based on conditional random fields and is a state-of-the-art method for Czech [8]. We are working on multilingual NER, which has already achieved exceptional results. The NER tool is currently tested by two major companies in Czech republic – Seznam.cz, a.s[6] (a majority search engine) and ČTK[7] (national news agency established by law).

Our sentiment analysis research is also heavily based on Brainy. The research was focused on social media SA [9], semantic spaces in SA [10], or a new model for SA based on the target context [11].

[1] http://mallet.cs.umass.edu
[2] http://java-ml.sourceforge.net
[3] http://www.cs.waikato.ac.nz/ml/weka/
[4] http://mahout.apache.org
[5] http://gate.ac.uk
[6] http://www.seznam.cz
[7] http://www.ctk.eu

Brainy is also used in a commercial project for Owen Software Ltd. and in the High Precision Stemmer[8], which is a unsupervised language-independent stemmer.

This section shows that the library can achieve state-of-the-art results in multiple research fields and it can be used for commercial projects.

6 Availability and requirements

The library is written in Java and thus should be usable on any platform with Java Virtual Machine. The minimal version of Java is 1.6. It is necessary to use the 64 bit version of Java for non-trivial applications because of memory requirements.

The library is available from the project web page[9]. It is released under the GPLv3[10] license.

7 Conclusion and future work

We have implemented a Java machine learning library called Brainy. Brainy was already used in research and commercial projects. It is released under GPL license.

The library provides many advanced algorithms, data structures and utilities. The infrastructure of the library allows quick implementation of new algorithms with standard interfaces. The library is designed for experimentation as well as for production systems.

The library is under active development. In the near future we are going to add our own implementation of various types of neural networks and graphical models. We also want to create a framework for standard machine learning pipelines and their configuration.

Acknowledgements

This work was supported by grant no. SGS-2013-029 Advanced computing and information systems, by the European Regional Development Fund (ERDF). Access to the MetaCentrum computing facilities provided under the program “Projects of Large Infrastructure for Research, Development, and Innovations” LM2010005, funded by the Ministry of Education, Youth, and Sports of the Czech Republic, is highly appreciated.

References

1. Berger, A.L., Pietra, V.J.D., Pietra, S.A.D.: A maximum entropy approach to natural language processing. Comput. Linguist. **22** (March 1996) 39–71
2. Malouf, R.: A comparison of algorithms for maximum entropy parameter estimation. In: Proceedings of the 6th Conference on Natural Language Learning - Volume 20. COLING-02, Stroudsburg, PA, USA, Association for Computational Linguistics (2002) 1–7

[8] http://liks.fav.zcu.cz/HPS/

[9] home.zcu.cz/∼konkol/brainy.php

[10] http://www.gnu.org/licenses/gpl-3.0-standalone.html

3. Lloyd, S.: Least squares quantization in pcm. IEEE Trans. Inf. Theor. **28**(2) (September 2006) 129–137
4. McCallum, A.K.: Mallet: A machine learning for language toolkit. (2002)
5. Abeel, T., Van de Peer, Y., Saeys, Y.: Java-ml: A machine learning library. J. Mach. Learn. Res. **10** (June 2009) 931–934
6. Hall, M., Frank, E., Holmes, G., Pfahringer, B., Reutemann, P., Witten, I.H.: The weka data mining software: An update. SIGKDD Explor. Newsl. **11**(1) (November 2009) 10–18
7. Konkol, M., Konopík, M.: Maximum entropy named entity recognition for czech language. In Habernal, I., Matoušek, V., eds.: Text, Speech and Dialogue. Volume 6836 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2011) 203–210
8. Konkol, M., Konopík, M.: Crf-based czech named entity recognizer and consolidation of czech ner research. In Habernal, I., Matoušek, V., eds.: Text, Speech and Dialogue. Volume 8082 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 153–160
9. Habernal, I., Ptáček, T., Steinberger, J.: Sentiment analysis in czech social media using supervised machine learning. In: Proceedings of the 4th Workshop on Computational Approaches to Subjectivity, Sentiment and Social Media Analysis, Atlanta, Georgia, Association for Computational Linguistics (June 2013) 65–74
10. Habernal, I., Brychcín, T.: Semantic spaces for sentiment analysis. In Habernal, I., Matoušek, V., eds.: Text, Speech, and Dialogue. Volume 8082 of Lecture Notes in Computer Science. Springer Berlin Heidelberg (2013) 484–491
11. Brychcín, T., Habernal, I.: Unsupervised improving of sentiment analysis using global target context. In: Proceedings of the International Conference Recent Advances in Natural Language Processing RANLP 2013, Hissar, Bulgaria, INCOMA Ltd. Shoumen, BULGARIA (September 2013) 122–128