Learning Machine: design and implementation

Marko Grobelnik, Dunja Mladenić
Dpt. for Intelligent Systems, J.Stefan Institute,
Jamova 39, 1000 Ljubljana, Slovenia
Phone: (+386)(61) 1773 272, Fax: (+386)(61) 1258-158
E-mail: Marko.Grobelnik@ijs.si, Dunja.Mladenic@ijs.si

Abstract

We present design and implementation of the machine learning system called **Learning Machine**. The system enables usage of different machine learning techniques on large sets of data, with a special part that enables direct incorporation of text data collected from the Web. The system incorporates a whole range of components used in machine learning research and application on real-world data, used in data mining, text-learning, learning on Web. The system is design in object oriented paradigm to enable easy combination of different components.

1 Introduction

Machine learning research and application on real-world data requires tools supporting different learning techniques and data transformations. Our system **Learning Machine** currently incorporates several machine learning algorithms, different feature scoring and selection methods, model quality estimates and special modules for handling text data and collecting data from the Web.

Our aim is to enable simple incorporation and testing of new research approaches, efficient handling of large data sets necessary for data mining, efficient collection of Web documents from Internet and simple combination and extension of different components used in machine learning approaches. **Learning Machine** is designed in object oriented paradigm and consists of the number of modules designed to separate the lower level definitions and operations (components) from the higher level components and to group similar components together.

2 Basic modules

All basic modules are included in a module called `base` that is used from most of the higher-level modules. Basic modules implement two groups of the basic building blocks. The first group contains basic definitions, data types, file handling and data structures.

basic definitions `bd` Contains character constants, basic type definitions, error and warning messages, assertions, basic operator and definition of basic relational operators, simple smart pointers.

files `fl` Contains check-sum for reading/writing files, input/output stream, input/output file, input/output memory, random access file.

basic data types `dt` Contains calculation of random numbers, character array, string, void, reference, Boolean, character, integer, float.

data structures `ds` Contains smart pointer with input/output, reference count, pair of values, key data, vector, 2D-vector, stack, queue, hash table, record file, function.

The second group contains definition of different size bit-sets, environmental variables and extended files that are system dependent, extended mathematical functions, lexical analysis and properties definition:

bit-sets `bits` one byte, four bytes, flag-set, bit32-set, bit-set,

environment `env` environment variables,

extended files `xfl` find file utility,

extended mathematics `xmath` sqr, statistical moments,

lexical analysis `lx` lexical characters, lexical symbols, lexical input-symbols, lexical input/output,

properties `pp` property value, property.

3 Functional modules

Functional modules implement higher-level operations needed for different kinds of the system usage. Some modules are specific for application area, for example, html-modules that are used in text-learning. Other modules are common to different areas, for example, domain module that is used for storing domain data. Some functional modules are rather independent (eg., `cgi`, see Figure 3), while other are more complex and include some other, usually simpler functional modules (eg., `dm` includes `exset` and `dmhd`, see Figure 3).

3.0.1 Collecting and handling Web data

Collecting and handling Web data expects as input a set of Web-page addresses (URLs) that point to documents and outputs a structure containing data to be used as machine learning examples. Module `webtrv` enables getting not only Web-page with a given URL but also all other Web-pages accessible from the required Web-page. This module is used by `yahoo` module in order to get Web-pages included in Yahoo hierarchy.

For each URL a document is fetched and stored together with its URL using `web` module. The set of documents is stored as memory base using `webmb` module. First URL is parsed using `url` module and request for document is put to Web-server on Internet using `sock` module. Response of the contacted Web-server is parsed using `http` module and an appropriate action taken. For example, in case that Web-server responded with a message that requested document was moved to some other address (redirection message) request to a new address is send.

In the next phase, each successfully fetched Web-document is given to `html` module for parsing html-documents. The result of parsing is a structure described in module `hc`. This structure is used in module `tbhc` to form a table of examples. This table is the connection between collecting and handling Web data and machine learning.

Learning from text documents available on the Web first requires mechanisms for automated collection of documents given their location on the Web. It further requires text data processing and representation in a format suitable for the application of machine learning methods. Our system incorporates following functional modules for collecting documents from the Web and handling text documents.

sockets `sock` Containing methods for working with asynchrone sockets including socket events handling, socket host information handling and methods for transferring data using sockets.

uniform resource locator `url` Containing the original URL and parsing it to extract components (scheme, host, port, path, search, fragment identifier), handles absolute and relative URLs.

common gateway interface `cgi` CGI input that reads from `stdin` and stores into hash table `(key name, string values)`, cgi output that gets page title, font, content, html-document background as input and outputs generated html-document to `stdout`.

http lexical `http` Containing http lexical characters and methods for parsing http request and response.

world wide web `web` Containing web-page transferring given URL, web-page description, its loading and saving, adding to database.

web memory base `webmb` Containing web-pages, for each page a file name on the local disk and a list of URLs pointing to the page, getting and adding pages to the database, walking through the whole database, URL redirection handling.

web travel `webtrv` Containing mechanisms for getting all web-pages accessible from the given set of web-pages whose URL contains required substring, several examples that use the mechanism included: ARTFL Roget thesaurus, Thesaurus.com Roget thesaurus, IJS, CIA, HomeNet, CMU Personal WebWatcher.

information retrieval stop words `irsw` Containing three versions of English stoplist used in information retrieval to filter out common/uninformative English words.

Yahoo `yahoo` Containing structures used to represent and handle data collected from the hierarchical structure of Web-pages based on Web-page topic called Yahoo: word, document, data base, Yahoo word distribution and distribution data base.

html lexical `html` Containing html lexical characters and methods used for parsing html-text.

html coded data `hc` Word represented as `(wordId that is a pointer to hash table, flags capturing context and html-tags)`, document represented as vector of words, base containing:

- word hash table containing `(word-string, frequency in all documents)`,
- document hash table containing `(URL, pointer to document)`,
- a bunch of smaller hash tables containing stop-list, flags for each word, continuous html-tags that don't break content, break html-tags, skip html-tags.

3.0.2 Machine learning methods

Input for machine learning methods is description of some problem domain that usually includes set of examples. The result of the application of machine learning algorithm is some knowledge model that is usually tested on new examples. Our system enables description of examples with different kind of variables (eg., discrete valued variables), usage of different machine learning algorithms and knowledge representation, different ways of model testing. All these is supported with object oriented design that enables flexible combination of system components. For example, the same decision tree induction algorithm can be used with different attribute scoring methods.

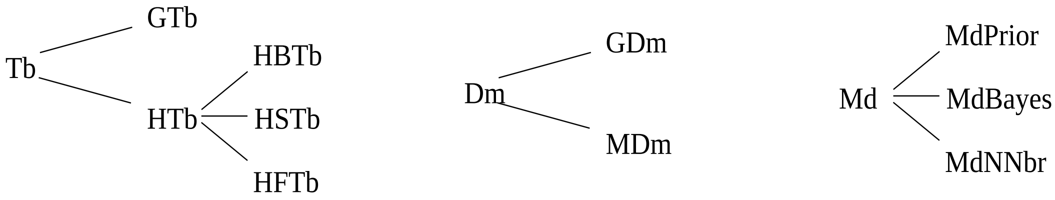

Figure 1: Derivation of different classes from an abstract class. Example is given for table `Tb`, domain `Dm` and model `Md`.

Machine learning part of the system consists of modules that cover: representation and handling of input data (`tbval`, `tb`, `tbhc`), machine learning domain description (`dmhd`, `dm`, `exset`), interface between examples defined in domain and machine learning algorithms that use examples and representation, generation and testing of knowledge model (`md`, `mdtr`, `mg`, `ts`). Figure 1 shows example of inheritance using abstract classes for table containing input data, machine learning domain and model generated in the learning process. Interface between examples given in domain and examples needed in machine learning consists of modules for retrieving values of variables that describe examples (`valret`), calculating distribution for different variables from examples (`valds`), distance metrics working on pair of examples (`mtr`). What follows is a brief description of each module.

table value `tbval` Table value represented as `(tag, value)`, where `tag` can have some of the following values: `unknown, unimportant, inapplicable, discrete,` `integer, float`, table value selector represented as `(operator, value)` eg.$a_1 \leq 5$, table value set implemented as an abstract class for table value set expression and table value set selector.

table `tb` Containing table data type, abstract classes and general classes (see Figure 1):

- table data type (fixed means that range of a variable is predefined),
- table variable implemented as an abstract class containing `(name, type)`, table tuple implemented as an abstract class containing a tuple of variables (one example), table implemented as an abstract class containing a vector of tuples and capable of adding variables and adding tuples
- generic table variable represented as `(name, type, active value type set)`, generic table tuple `(name, value vector)`, generic table `(variable vector, tuple vector)` with additional hash tables for variable and tuple names.

table for html representation `tbhc` HTML-coded table variable `(name, type)`, html-coded table containing html-base and translation tables for mapping variables into words and tuples into documents, and three classes derived from html-coded table `HTb`:

- `HBTb` html-coded binary table where tuples are represented as bit-sets (size_of(tuple) = #words bits),
- `HSTb` html-coded sequence table where tuples are represented as vectors of sorted integer (`worId`) thus taking into account sparseness in tuple representation (size_of(tuple) = document_length * size_of(integer)),
- `HFTb` html-coded frequency table where tuples are represented as vectors of pairs `(wordId, frequency)` (size_of(tuple) = document_length * 2 * size_of(integer)).

Using `HBTb` enables faster access but takes more space than using `HSTb`. For example, for a document set containing 100000 different words the size_of(tuple) for `HBTb` is $100000/8 = 12.5Kb$ and for `HSTb` would be the same only if the average length of document is 3125 words.[1]

domain header `dmhd` Domain header variable `(name, type)`, domain header implemented as an abstract class, general domain header containing two vectors of domain header variables one for classes and the other for attributes

domain definition `dm` Containing:

- domain implemented as an abstract class `Dm` containing name, load/save methods, methods for operations on class, attributes and examples, domain property that can be either file property (name, type, memory representation) or reference (pointer to memory),

[1] The average length of Web documents we used in text-learning experiments is not more than 1000 words.

- domain check of some domain characteristics (eg. having discrete class),
- `GDm` general domain containing table, vector of variables used as classes, vector of variables used as attributes, vector of tuple numbers used as examples,
- `MDm` multiple domain containing vector of tables, the three vectors (as in general domain) for classes, attributes, examples with the additional information about the table number added to classes and attributes.

example set `exset` Example set implemented as an abstract class, generic example set containing vector of examples `(number, weight)` and capable of generating random split for a given hold-out testing fraction size and split into k folders for $k - fold$ cross validation testing.

value retriever `valret` Value retriever implemented as an abstract class containing domain header, domain value retriever containing domain and methods for getting attribute value from example and getting/setting example number, overloaded value retriever handling changes of domain during learning and containing value retriever and mapping from new attributes into old attributes.

value distribution `valds` Value distribution for one variable `(type, weight, tag weights, discrete value weights, float value moments)`, class discrete value distribution containing weight for each class value, class attribute discrete value distribution containing weight sum, class discrete weight vector, class attribute discrete weight vector.

metric `mtr` Metric implemented as an abstract class containing property for selecting metric `Mtr`, derived metric classes:

- `MtrBlock` city-block metric $\sqrt{\sum_a dist_a(x_a, y_a)}$, where $dist(x, y) = |x - y|$,
- `MtrEucl` Euclidean metric $\sqrt{\sum_a dist_a(x_a, y_a)}$, where $dist(x, y) = (x - y)^2$,
- `MtrVdm` value difference metric $\sqrt{\sum_a dist_a(x_a, y_a)}$, where $dist_a(x, y) = \sum_c |P_{c|a}(x) - p_{c|a}(y)|^q$, usually $q = 1$ or $q = 2$,
- `MtrCos` vector angle cosine distance $1 - \frac{\sum_a x_a * y_a}{\sqrt{\sum_a x_a^2 \sum_a y_a^2}}$.

model of induced knowledge `md` Model implemented as an abstract class `Md` containing name, methods for getting domain header and classifying examples, derived model classes:

- `MdPrior` model of prior probability containing domain header, class identifier, prior distribution of classes,
- `MdBayes` model of naive Bayessian classifier containing domain header, class identifier, distribution of classes and attribute values,
- `MdNNbr` model of nearest neighbor classifier containing domain, example set, distance metric, class identifier, number of neighbors to use in classification.

model tree `mdtr` Decision node implemented as an abstract class `DNd` and derived decision node classes:

- `DndLeaf` decision node leaf containing model,
- `DNdABr` decision node attribute branch containing test-attribute, attribute values for each branch, branches,
- `MdTree` model tree containing domain header, class identifier, decision model.

model generator `mg` Model generator implemented as an abstract class `Mg` containing properties, methods for getting domain, methods for getting model, derived model generator classes:

- `MgPrior` prior model calculating prior class discrete distribution
- `MgBayes` naive Bayes model calculating class-attribute discrete distribution
- `MgNNbr` nearest neighbor model storing data directly from domain
- `MgId3` ID3 decision tree model inducing decision tree

testing model `ts` test model getting domain, examples and model as input and calculating model quality estimate (classification accuracy, probability accuracy, information score, geometric mean of recall, geometric mean of precision, recall, precision, probability recall, probability precision, squared error in counting) (see Figure 2), test model generator implemented as an abstract class, test model generator with cross validation, test model generator with hold out

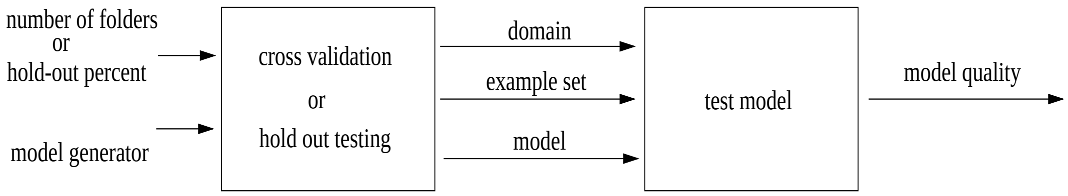

Figure 2: Data flow for estimating model quality using cross-validation or hold-out testing scheme.

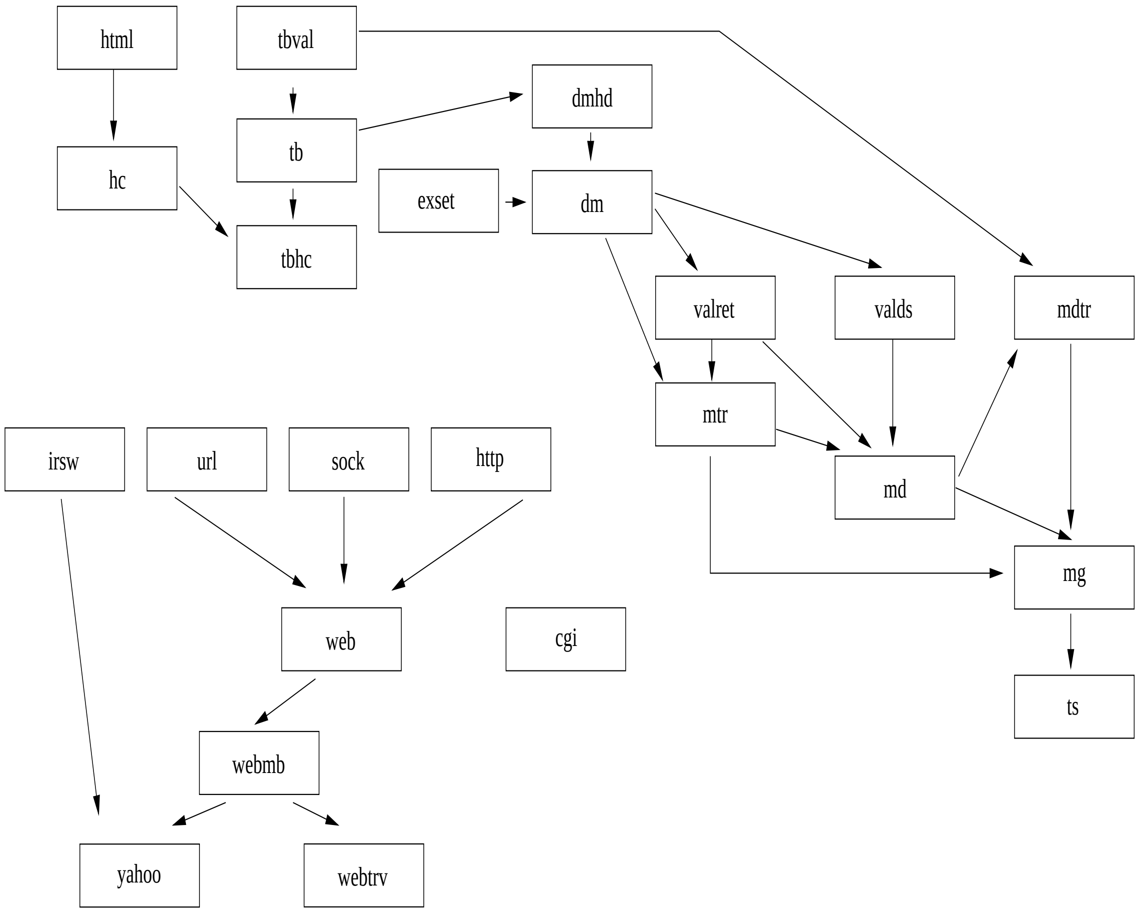

Figure 3: Functional modules use each other through inclusion mechanism (eg. `hc` includes `hlx`). Additionally to the given connections, all functional modules include `base` that includes all basic modules.