# QLectives – Socially Intelligent Systems for Quality

# Project no. 231200

## Instrument: Large-scale integrating project (IP)

## Programme: FP7-ICT

# Deliverable D4.4.2

# Algorithms for detecting and handling HQ-MD (High Quality metadata) and Spam metadata

**Submission date: 2010-08-31**

**Start date of project: 2009-03-01** **Duration: 48 months**

**Organisation name of lead contractor for this deliverable: IRT**


| Project co-funded by the European Commission within the Seventh Framework Programme (2007-2013) | | |
|---|---|---|
| **Dissemination Level** | | |
| **PU** | Public | **x** |

| | | |
|---|---|---|
| **PP** | Restricted to other programme participants (including the Commission Services) | |
| **RE** | Restricted to a group specified by the consortium (including the Commission Services) | |
| **CO** | Confidential, only for members of the consortium (including the Commission Services) | |

# Document information

### 1.1 Authors

| Author | Organisation | E-mail |
|---|---|---|
| J. Groh | IRT | Groh@irt.de |
| I. Hegedűs | USZ | Ihegedus@inf.u-szeged.hu |
| R. Ormándi | USZ | Ormandi@inf.u-szeged.hu |
| M. Guelbahar | IRT | Guelbahar@irt.de |
| A. Zistler | IRT | Zistler@irt.de |

### 1.2 Other contributors

| Name | Organisation | E-mail |
|---|---|---|
| | | |

### 1.3 Document history

| Version# | Date | Change |
|---|---|---|
| V0.1 | | Starting version, template |
| V0.2 | 31-5-2010 | Definition of ToC |
| V0.3 | 14-6-2010 | First draft version |
| V0.4 | 26-7-2010 | Draft version |
| V0.5 | 30-07-2010 | Final Draft version sent to deliverables committee for approval |
| V0.7 | 11-08-2010 | Revised Final Draft version |
| V1.0 | 31-08-2010 | Approved version to be submitted to EU |

### 1.4 Document data

| | |
|---|---|
| **Keywords** | QLectives, Metadata, Quality |
| **Editor address data** | |

| | | |
|---|---|---|
| | Name: | Alois Zistler |
| | Partner: | IRT |
| | Address: | Floriansmuehlstrasse 60,<br>D-80939 Muenchen, Germany |
| | Phone: | +49 89 323 99 234 |
| | Fax: | +49 89 323 99 200 |
| | E-mail: | Zistler@irt.de |
| **Delivery date** | August 2010 | |

### 1.5 Distribution list

| **Date** | **Issue** | **E-mail** |
|---|---|---|
| | Consortium members | QLECTIVES@LIST.SURREY.AC.UK |
| | Project officer Jose Fernandez-Villacanas | Jose.FERNANDEZ-VILLACANAS@ec.europa.eu |
| | EC archive | INFSO-ICT-231200@ec.europa.eu |

# QLectives Consortium

This document is part of a research project funded by the ICT Programme of the Commission of the European Communities as grant number ICT-2009-231200.

**University of Surrey (Coordinator)**
Department of Sociology / Centre for Research in Social Simulation
Guildford GU2 7XH
Surrey
United Kingdom
Contact person: Prof. Nigel Gilbert
E-mail: n.gilbert@surrey.ac.uk

**Technical University of Delft**
Department of Software Technology
Delft, 2628 CN
Netherlands
Contact Person: Dr Johan Pouwelse
E-mail: j.a.pouwelse@tudelft.nl

**ETH Zurich**
Chair of Sociology, in particular Modelling and Simulation,
Zurich, CH-8092
Switzerland
Contact person: Prof. Dirk Helbing
E-mail: dhelbing@ethz.ch

**University of Szeged**
MTA-SZTE Research Group on Artificial Intelligence
Szeged 6720, Hungary
Contact person: Dr Mark Jelasity
E-mail: jelasity@inf.u-szeged.hu

**University of Fribourg**
Department of Physics
Fribourg 1700
Switzerland
Contact person: Prof. Yi-Cheng Zhang
E-mail: yi-cheng.zhang@unifr.ch

**University of Warsaw**
Faculty of Psychology
Warsaw 00927, Poland
Contact Person: Prof. Andrzej Nowak
E-mail: nowak@fau.edu

**Centre National de la Recherche Scientifique, CNRS**
Paris 75006,
France
Contact person: Dr. Camille ROTH
E-mail: camille.roth@polytechnique.edu

**Institut für Rundfunktechnik GmbH**
Munich 80939
Germany
Contact person: Christoph Dosch
E-mail: dosch@irt.de

# QLectives introduction

QLectives is a project bringing together top social modellers, peer-to-peer engineers and physicists to design and deploy next generation self-organising socially intelligent information systems. The project aims to combine three recent trends within information systems:

- **Social networks** - in which people link to others over the Internet to gain value and facilitate collaboration
- **Peer production** - in which people collectively produce informational products and experiences without traditional hierarchies or market incentives
- **Peer-to-Peer systems** - in which software clients running on user machines distribute media and other information without a central server or administrative control

QLectives aims to bring these together to form Quality Collectives, i.e. functional decentralised communities that self-organise and self-maintain for the benefit of the people who comprise them. We aim to generate theory at the social level, design algorithms and deploy prototypes targeted towards two application domains:

- **QMedia** - an interactive peer-to-peer media distribution system (including live streaming), providing fully distributed social filtering and recommendation for quality
- **QScience** - a distributed platform for scientists allowing them to locate or form new communities and quality reviewing mechanisms, which are transparent and promote

The approach of the QLectives project is unique in that it brings together a highly inter-disciplinary team applied to specific real world problems. The project applies a scientific approach to research by formulating theories, applying them to real systems and then performing detailed measurements of system and user behaviour to validate or modify our theories if necessary. The two applications will be based on two existing user communities comprising several thousand people - so-called "Living labs", media sharing community tribler.org; and the scientific collaboration forum EconoPhysics.

# Executive Summary

This document describes the combination of two approaches for detecting and handling high quality metadata and spam metadata. Both solutions are based on self-learning algorithms - analogous to email-junk filters – allowing for a classification, respectively prediction of the quality of each metadata record. The solutions presented are generic and based also on preceding work within this work package, namely the generic metadata model (D4.4.1). Thanks to the generic approach by deploying self-learning algorithms in conjunction with features, a high degree of flexibility has been achieved, allowing for alteration of functionality and identification of use-cases on application layers chosen at later stages within the project - following the QLectives concept of several empirical cycles during the project period.

The introduction chapter emphasises the importance of high quality metadata in high quality – related environments, such as the QLectives application area. It underlines the key role of metadata in search and discovery of content, in particular for so-called “User Generated Content (UGC)” corresponding metadata, which may be available in diverse levels of quality - ranging from detailed, correct descriptive data to missing or even false metadata by intention. To ensure incorruptible high quality search results, means are needed to evaluate the trustworthiness of the metadata elements. The introduction ends by outlining the successive steps of work package WP4.4 with the corresponding deliverables.

Following, a twofold approach is drawn out in chapter 2, respectively chapter 3.

In chapter 2 we describe the first part of our solution, which consists of the algorithms managing *static* metadata. This approach is particularly targeted on initial (static) metadata, which has not been (or never will be) changed since its first publication.

Chapter 3 describes a solution to handle *dynamic* metadata, by evaluating modifications on and different versions of metadata, especially to detect malicious, destructive editions by users.

The conclusions and references are presented in chapter 4 and chapter 5.

# Contents

# 1 Introduction

Metadata, commonly known as “data about data” or “information about data” plays a substantial role when it comes to definition, search and retrieval of digital items. In principle, the more precise and the more detailed metadata is available, the better approaches for search and discovery of content can be applied.

Generally, in broadcast domains metadata can be assumed to be correct and precise. It is available in high quality, since broadcasters are responsible for both the production of metadata and its delivery to the end user – via dedicated transmission channels - without any alteration by third parties. In domains of user generated content (UGC) e.g. YouTube [1], or P2P [2] networks, where metadata is generated and edited by users, it may be of high quality too, but on average, it is missing, not detailed enough, or even completely false and misleading. If so, we usually talk about “low quality metadata”, “junk metadata” or “spam metadata” – depending on the reasons and motivation of the people that provided the metadata not being of high quality.

The presence of detailed and correct metadata plays a fundamental role for improved quality search and discovery facilities, and therefore is essential for the success of QMedia. Without correct and detailed metadata, an adequate search mechanism can not properly be applied. In the simplest case, metadata may be just the title of the file itself. In the worst case, with metadata being wrong or even fake, the system would return mismatching search results, respectively recommendation results, and might cause users to download files they are not interested in, thereby wasting both time and bandwidth. QLectives aims to go beyond just file finding within the scope of what common P2P networks do offer. Since the QMedia use-cases will deal predominately with user-generated content, descriptive information about the content will have to be provided by users. Hence, the solutions provided by QLectives should allow for entering, editing and collecting metadata by users, and additionally, enable users to contribute descriptive information, e.g. by adding tags or rating clips. Consequently, gathered metadata will exist in different degrees of quality. The system needs to *apply algorithms individually*, in order to dynamically rate metadata based on various given parameters, and detect and separate spam metadata from high quality metadata.

Of course, besides the presence of adequate and correct metadata (high quality metadata), the quality of a *search result* depends also on the implementations of adequate search engines, and – in case of a recommender – additionally in interaction with the user's individual profile. Anyway, to do so, it is crucial that it is ensured that solely high quality metadata is applied and misleading metadata within the system (introduced by destructive users) remains unused or erased.

In this deliverable we introduce approaches to evaluate the quality of metadata in a self-learning and therefore fully user-adjusted manner. The work is based and makes use of the generic metadata framework, as presented in Deliverable *D4.4.1 "Generic Metadata Model"* [3].

This task is dedicated to find optimal and still flexible ways for evaluating the quality of metadata - but not the quality of the content described by this metadata. Furthermore, the procedure of collecting metadata e.g. by asking the user to rate content, or by even reviewing other users along with a suitable crediting and rating system, is outside the scope of this deliverable. Nevertheless, as depicted in Figure 1, the identification of appropriate use-cases on application-layer is a project-objective towards the next WP4.4 *Deliverable D4.4.3 "Implementation / integration guidelines for HQ-MD in QMedia v2".*

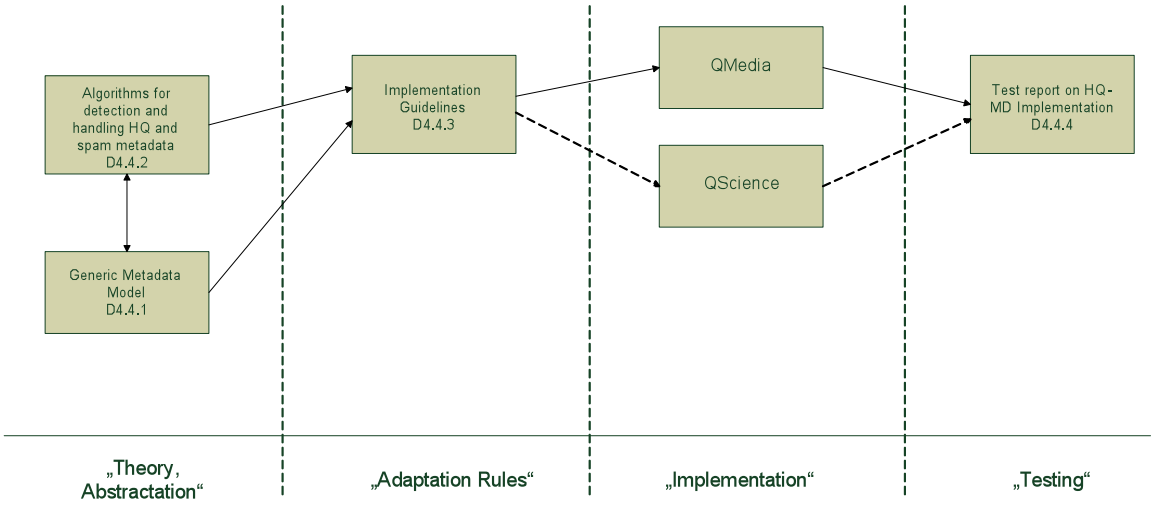


**Figure 1: Scheduling & Interdependencies of tasks in WP4.4**

D4.4.3, due in month 24 after project start, will deal with a set of implementation guidelines with respect to the QMedia-platform, thereby taking into account P2P-specific environments, requirements and possibilities of user (inter-)action.

Finally, Deliverable *D4.4.4 "Test report on HQ-MD implementation"*, due at the end of the project period, will report the test results from the living lab implementation.

# 2 Algorithms for static metadata rating

This section describes the approach and the QLectives-specific solution for static or initially provided user generated metadata. Algorithms rating dynamic metadata and metadata changes will be described in section 3.

## *2.1 Background*

In order to be able to evaluate the static rating approach, preconditions for possible experiments with real-world metadata were collected, and existing tools and data sources to make use of have been analysed. Then, an environment for metadata acquisition and management was set up, which also offers support for development, installation and testing of algorithms to generate quality ratings on the aggregated metadata. Since all libraries used within this task (for metadata aggregation, metadata handling, machine-learning) are implemented in Java [4] Eclipse [5] was the most suitable environment to choose.

## *2.2 Metadata aggregation*

TV-Anytime [6] metadata, describing audio and video content derived from various metadata sources, was chosen to be the data input to the machine learning algorithms, due to several reasons:

- IRT has access to metadata retrieval tools that support TV-Anytime
- The QLectives Generic Metadata Model [3] offers adapter classes towards TV-Anytime
- The QMedia implementation will support the TV-Anytime metadata format

The retrieval of the required data sets was realised by making use of IRTs metadata acquisition tools, which allow access to several metadata sources, such as:

- DVB-IP [7] servers: IP-based sources for broadcast metadata and content, providing professional metadata describing multimedia content

- YouTube backend: a typical Web2.0 metadata source, providing user generated metadata describing multimedia content
- UPnP [8] servers: servers residing in the DLNA [9] home network, providing access to both professional and user generated metadata, describing multimedia content
- Podcast [10] servers: servers residing in the open internet, providing access to metadata and content, providing professional metadata describing multimedia content

Although all metadata sources mentioned above provide metadata in different metadata formats, and classifications of content (i.e. media genres) in various ways, IRTs metadata integration service, which translates metadata formats as well as media classifications into a unified overlay format, allowed acquisition and translation of metadata from several sources to the TV-Anytime metadata format (and its classifications, respectively).

The metadata integration service was designed and implemented within the scope of the iNEM4U FP7 project [11] and details about the solution can be found at [12].

## *2.3 Self-learning algorithms*

### 2.3.1 Background

In order to be able to realise quality rating of metadata in a manner both personalised and self-adapting to future demands, the decision for the basic algorithm design fell onto self-learning algorithms (machine learning algorithms). This allows flexibility concerning user preferences, changing user likes and dislikes, as well as concerning upcoming metadata trends and requirements. The system remains dynamic, as new feature sets, which are being used by the self-learning algorithms, can be created and added to the system at any point in time.

A machine-learning algorithm generally requires:

- *Data samples*
- A definition of a *set of features* of the data and
- A definition of *class labels* of the data

Each data sample is represented as a record of its feature values and its class label value. Data samples are then grouped into *sample sets*, where one set comprises the "training" data, and another set comprises the data used for "testing" the state of the learned algorithms. With most machine-learning systems, testing data samples can also be tested individually, and not only as part of a set. WEKA [13] supports continuous-valued or symbolic-valued features, called numeric or nominal attributes, respectively. The WEKA implementation [14]**Error! Reference source not found.** is a machine-learning system that comprises an extensive set of machine learning tools that can be accessed via a stand-alone, GUI-based integrated environment program, thereby using file-based data exchange mechanisms, or via an *Application Programming Interface* ("API") for the use by other programs or class implementations. The latter way allows interfacing the QLectives testing applications with algorithms that were developed with the WEKA machine-learning system. The objective of the task covering quality rating for static metadata was defined as applying a WEKA based machine learning algorithm to metadata sets that are structured according to the TV-Anytime metadata specification.

Both WEKA and IRT's TV-Anytime tools are available as code libraries for the Java programming language, so the natural choice was to implement the static rating setup in Java.

### 2.3.2 Approach

One part of the task was to create an application that designs an interconnection mechanism for, on one side, sources of data that have features and, on the other side, machine-learning mechanisms. The interconnection mechanism does not need to take into account any dynamic nature of data, since it was intended for static data evaluation, but it should at least not complicate future extensions to dynamic data.

The easiest way to connect the (TV-Anytime) metadata sets to the machine-learning

algorithms offered via the WEKA API would have been by writing an application that simply uses both, the WEKA API and the TV-Anytime API. But in order to allow various implementers to make use of the QLectives metadata rating system, a design that does not restrictively tie QLectives, TV-Anytime and WEKA together was aimed at, maintaining the choice for other metadata formats as well as other machine learning systems. This abstraction additionally asserts consistency with the objective of deliverable D4.4.1, to define a generic metadata model that is independent from specific data format definitions. Following a layer-oriented design approach was preferred, as known from software engineering design principles. It allows QLectives to re-use system components at a later stage of the project, when all supported content sources, metadata formats, and machine-learning systems to be supported have been decided on.

At first, a new software project for the Eclipse Java development environment was set up. In conjunction with the tools used for D4.4.1 ("Enterprise Architect", a Software Engineering application) it is easy to integrate the classes defined there with those programmed in Java in Eclipse, as both-way Java code import and export is supported. Only the package naming was changed to be more Java compliant: The new project packages base is *"eu.qlectives"*, where the object types for machine learning reside in the sub-package *"eu.qlectives.rating.machinelearning"*.

### 2.3.3 Connecting data and machine learning systems

The essence of the connection between a machine learning system and its data lies in what was already mentioned in section 2.3.1, namely the notions of (data) samples, features, class labels and values (of features and class labels). This leads to making the necessary object types (classes and interfaces) reflect these notions and naming them *"Sample"* and *"Feature"*; for the third type we chose the name "Variable", not "Value", since it actually only provides the *access* to the value that is to be abstracted, not the value itself (in other words, a *"ValueAccessor"*). A separate type *"ClassLabel"* is not necessary, since all its properties are covered by *"Feature"* already, as we will see soon.

These three object types form a *machine learning abstraction layer* inside the QLectives software framework that provides enough abstraction to enable the interconnection of

any machine learning system to any data source (and, as "added value", any data sink, too), as depicted in Figure 2.

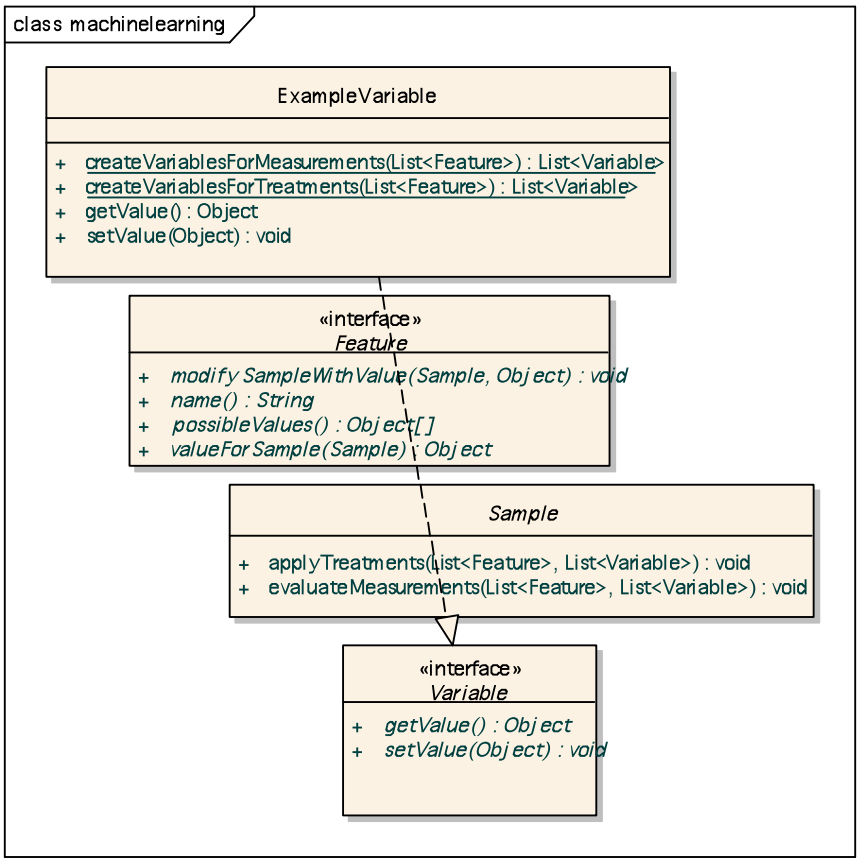


**Figure 2: The machine learning abstraction layer**

*"eu.qlectives.rating.machinelearning.Sample"* is (in Java terms) an *abstract* class, representing an abstraction of a data sample for a machine learning mechanism. Its intention is to be used in conjunction with *"Feature"*, in order to represent an *attributed data set*.

Implementations encapsulate the domain-specific data, but not any kind of behaviour. The associated domain-specific features, by "knowing" the *"Sample"* implementation type, can then bind to it and use its data to calculate or modify feature values for the data accordingly. *"Sample"* offers no methods that need to be implemented in sub-classes, but a set of public convenience methods:

- *"void evaluateMeasurements(List<Feature> features, List<Variable>*

*targetVariables)"*

This method allows the features to determine the measurement values, one value for each feature object, for this sample and stores them in the corresponding variable objects.

- *"void applyTreatments(List<Feature> features, List<Variable> sourceVariables)"*
  This method retrieves the treatment values, one value for each feature object, for this sample from the corresponding variable objects, and lets the features apply them.

*"eu.qlectives.rating.machinelearning.Feature"* is an interface representing an abstraction of a measurement (*read*) or treatment (*write*) function to be performed on a data sample for a machine learning mechanism. Its intention is to be used in conjunction with "Sample", in order to represent an attributed data set.

For *training* the machine learning mechanism, only the *read* function will be needed. Alternatively, a *write* function can be realised if any data generation or modification process is desired prior to the analysis. Implementations of only a reading behaviour are suggested to be named *"...Measurement"*, implementations of only a writing behaviour are suggested to be named *"...Treatment"*.

Machine learning mechanisms that identify classes (that is, classifiers) can represent the class labels as an appropriate *"Feature"*, too. Implementations encapsulate the domain-specific behaviour, but not data. They may assume to "know" the associated domain-specific *"Sample"* implementation type and can then bind to it and use its encapsulated data. The relevant methods to be implemented by implementations are:

- *"Object[] possibleValues()*"
  This method returns the set of legal values, or "null" if the values are not enumerable.
- *"Object valueForSample(Sample sample)*"
  This method retrieves the measurement value for the sample.
- *"void modifySampleWithValue(Sample sample, Object value)"*
  This method will apply the treatment value to the sample.

*"eu.qlectives.rating.machinelearning.Variable"* is an interface representing an abstraction of the (numeric or other) type of value of the *"cross-point"* of:

- One of the features and
- One of the samples in a data set for a machine learning mechanism.

Implementations encapsulate the learning-mechanism-specific access for values. More specifically, when a *"Feature"* has determined a measurement value for a *"Sample"*, the *"Variable"* object is the place where the value is written to, and when a *"Feature"* requires a treatment value for a *"Sample"*, the *"Variable"* object is the place where the value can be obtained. An implementation for an array-based or object-based mechanism should "know" an index or a reference, respectively, for "its" sample representation and "its" feature representation in the learning mechanism. The relevant methods to be implemented by implementations are:

- *"void setValue(Object value)"*

  This method sets the treatment value that is applied by a *"Feature" to a "Sample"*.

- *"Object getValue()"*

  This method retrieves the measurement value that is determined by a *"Feature"* from a *"Sample"*.

Applying the QLectives machine-learning abstraction layer is being done by implementing the appropriate abstract methods, as depicted in Figure 3.

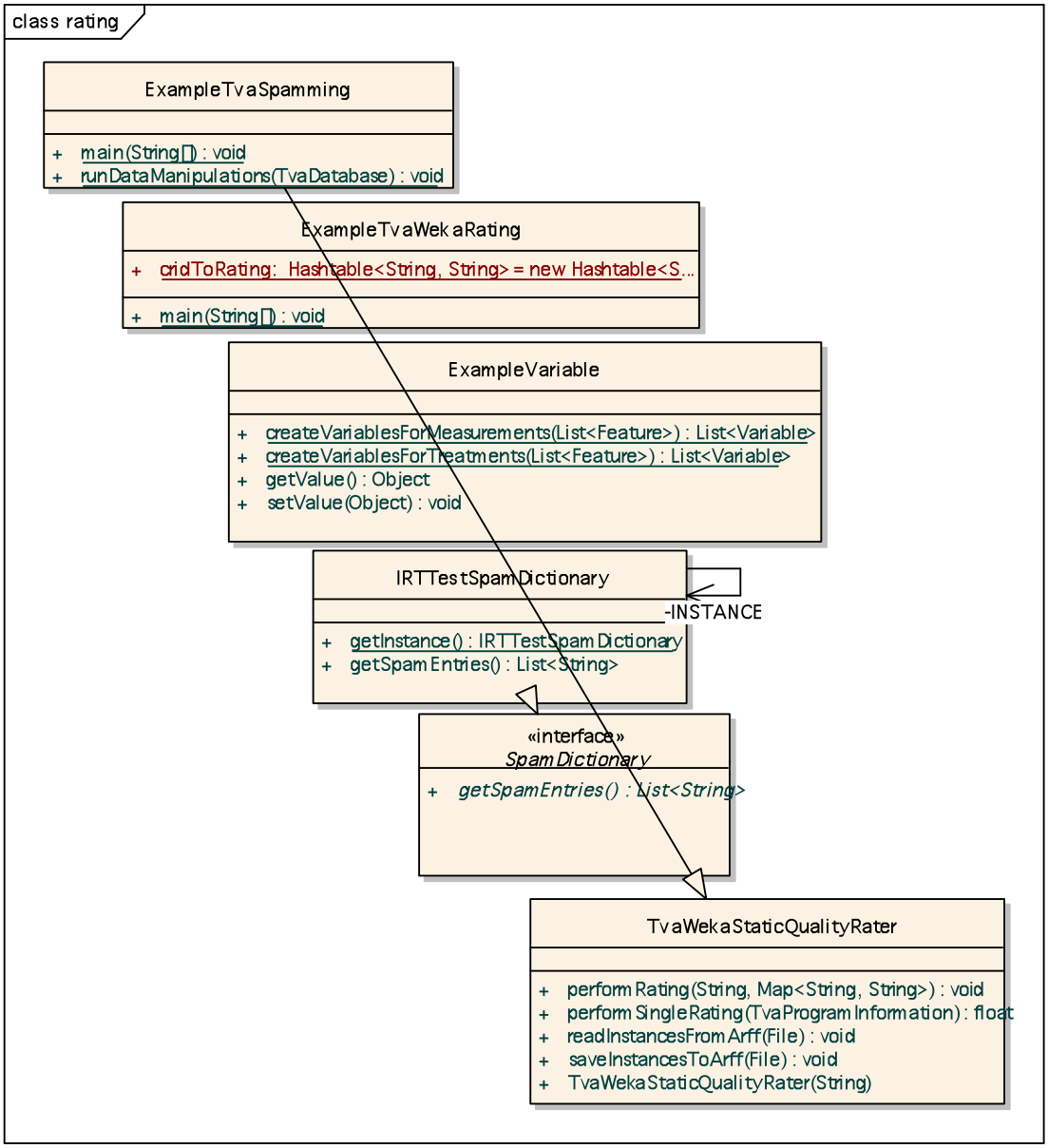


**Figure 3: Application of the abstraction layers onto "real" ratings**

In this specific case of implementation, the *"feature dimension"* and the *"sample dimension"* of the data set are concretized in TV-Anytime-specific classes, and the *"cross-point value accessor"* of the data set is concretized in a WEKA-specific class. Since such concretizations belong to the tasks of an application, they are not part of the QLectives framework, but of our test application and thus reside in the package branch *"de.irt.qlectives.rating"*. Within the implemented interconnection mechanism, *"Sample"* translates information between a class *"Instance"* in the WEKA domain and a class "*TvaProgramInformation"* in the TV-Anytime domain, and *"Feature"* translates

information between a class *"Attribute"* in the WEKA domain and, again, *"TvaProgramInformation"* in the TV-Anytime domain.

*"de.irt.qlectives.rating.weka.WekaVariable"* is an implementation of *"Variable"*, thereby encapsulating a WEKA "Instance" reference and *"Attribute"* index combination. Measurement and treatment *"Features"* are assumed to always use either *"Double"* or *"String"* values, corresponding to WEKA numeric or nominal attributes, respectively.

*"de.irt.qlectives.rating.tva.TvaProgramInformationSample"* is an implementation of *"Sample"* encapsulating a *TV-Anytime ProgramInformation*.

### 2.3.4 Implementation of “Feature”

Seven exemplary TV-Anytime-specific features have been designed and implemented within this task, as depicted in Figure 4:

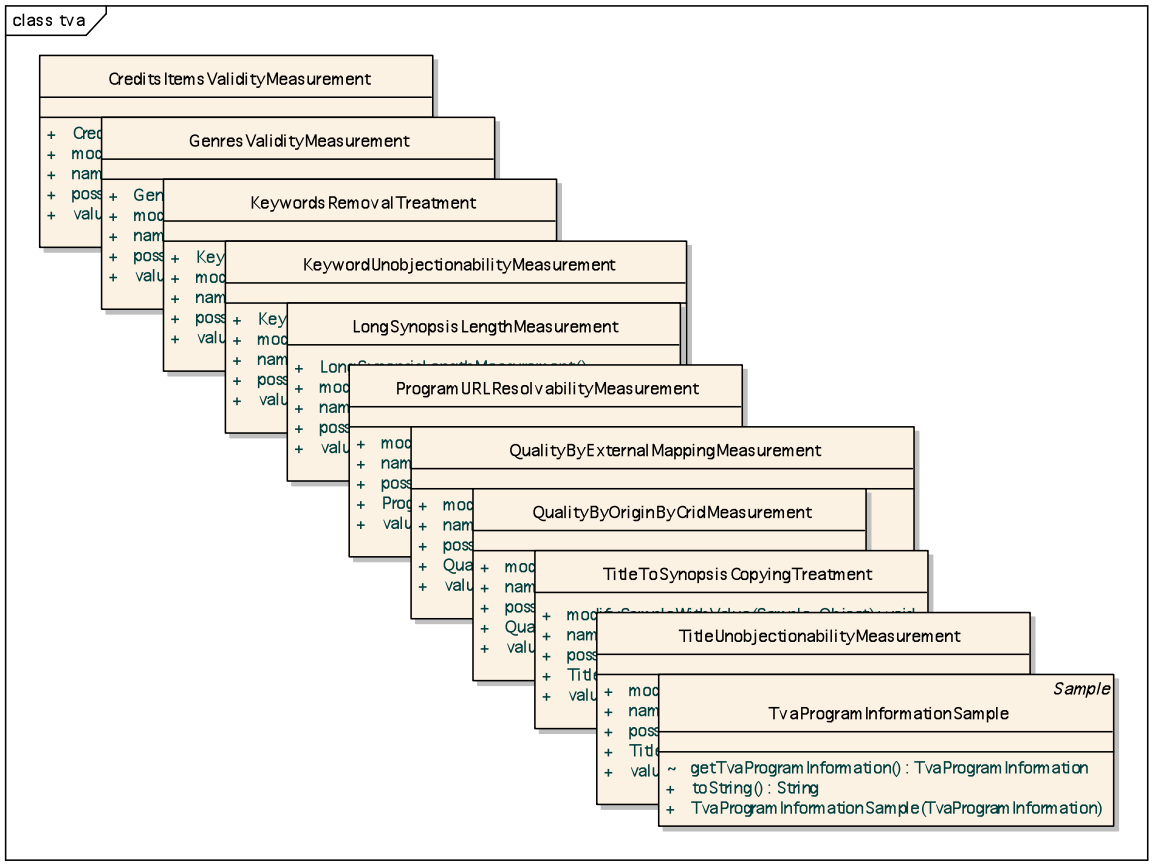


**Figure 4: The TV-Anytime specific features**

- *"de.irt.qlectives.rating.tva.CreditsItemsValidityMeasurement"*

  This is a measurement feature that evaluates whether the program information contains filled credits items (such as “actor”, “producer”, “stage director” etc).

- *"de.irt.qlectives.rating.tva.GenresValidityMeasurement"*

  This is a measurement feature that evaluates whether the genre types in the program information have valid *hrefs* (means, they are valid genre references).

- *"de.irt.qlectives.rating.tva.KeywordUnobjectionabilityMeasurement"*

  This is a measurement feature that evaluates whether the program information contains no objectionable words in the keywords, by making use of several SPAM dictionaries.

- *"de.irt.qlectives.rating.tva.LongSynopsisLengthMeasurement"*

  This is a measurement feature that evaluates the synopsis length, as well as its quality (by making use of several SPAM dictionaries) inside the program information element.

- *"de.irt.qlectives.rating.tva.ProgramURLResolvabilityMeasurement"*

  This is a measurement feature that evaluates whether the program information contains an associated schedule event with a resolvable program locator / URL and not just “any URL”, as known from SPAM web sites, which often offer links for un-subscription of newsletters, which usually almost never resolve into a valid un-subscribe web-site.

- *"de.irt.qlectives.rating.tva.TitleUnobjectionabilityMeasurement"*

  This is a measurement feature that evaluates whether the program information contains no objectionable words in the title, by making use of several SPAM dictionaries.

The *class label* itself is a "Feature", too:

- *"de.irt.qlectives.rating.tva.QualityByOriginByCridMeasurement"*

  is a provisional source for quality values for classification, for illustrative purpose. It tests whether the TV-Anytime sample was derived from YouTube (medium quality assumed) or professional broadcasters (high quality assumed). There are eleven discrete quality classes designated by a zero to ten stars "String" value.

### 2.3.5 Setting up the System

*"de.irt.qlectives.rating.TvaWekaStaticQualityRater"* (see Figure 3) is an example for binding the abstractions in *"eu.qlectives.rating.machinelearning"* to the TV-Anytime media metadata format and the WEKA machine learning library. These bindings are performed by *"TvaProgramInformationSample"* and *"WekaVariable"* (see Figure 5), respectively, in combination with a number of *"Features"* in *"de.irt.qlectives.rating.tva"*, all of which are consistent with the assumption of *"WekaVariable"* that values are either of type *"Double" or "String"*.

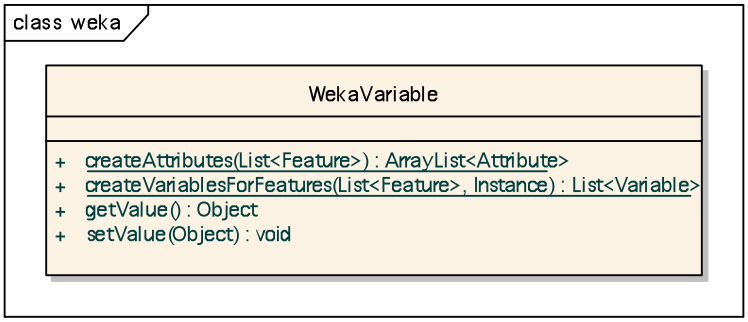


**Figure 5: A WEKA variable**

### 2.3.5.1 Data sets

Aggregated TV-Anytime data are imported from XML files into a *"TvaDatabase"* object, provided by the TV-Anytime code library we used. This is the purpose of the method *"createTvaDatabase()"*. The *"TvaDatabase"* object can then be queried for *"TvaProgramInformation"* objects, each of which represents one specific TV programme, video clip, or other sort of multimedia item.

### 2.3.5.2 Information transfer

The method *"TvaWekaStaticQualityRater.createWekaInstances()"* performs the essential steps in a loop that iterates over all data samples. In the following code excerpt it can be seen how information from a TV-Anytime-specific *"TvaProgramInformation"* object is transferred via the QLectives-specific *"Sample"* abstraction to a WEKA-specific *"Instance"* object:

```
// the multimedia item iterator
TvaProgramInformation tvaProgramInformation =
              tvaProgramInformationIterator.next();
// create a new sample from the current item
TvaProgramInformationSample sample = new
         TvaProgramInformationSample(tvaProgramInformation);
// create a new Instance
Instance instance = new DenseInstance(features.size());
instance.setDataset(instances);
// create WEKA variables
List<Variable> variables =
     WekaVariable.createVariablesForFeatures(features, instance);
// evaluate the current item, applying features
sample.evaluateMeasurements(features, variables);
// add this instance to all existing instances
```

**Figure 6: Applying TV-Anytime samples to the machine-learning system**

### 2.3.5.3 Features

The already outlined seven features that have been created and applied were fitted to some properties that were expected from the available data from the exemplary data sources. As they were intended to only show that a static rating using the abstraction layer works in principle, they were not assumed to necessarily be the best choice for quality ratings in the QLectives "Living Labs". Since *"Feature"* objects are designed to be "pluggable" functional code elements, they will be easily replaceable in a later stage of QLectives when concrete data properties are known.

### 2.3.5.4 Class labels

Because the existing data sources were lacking *"quality"* attribution, which the machine learning system could have been trained to, the *"quality"* class label was replaced by another feature derived from the data, namely one that discriminates the metadata origin: For the sake of simplicity it was programmed to assume that metadata originating from broadcasters is of *"high quality"* and metadata originating from YouTube is of *"medium quality"*, with some minor further refinement. By design, class labels - as well

as features - could be designed in both ways, a server-client-based as well as a Peer-to-Peer – based manner.

### 2.3.5.5 Algorithms

The algorithms part is used as an exemplary function. *"Learner"* (see Figure 7) is a class that encapsulates a WEKA-specific data set, and a selection of one or more classifier algorithms from the WEKA collection. It serves as the *access point* to the machine learning system used by the test application to invoke the training and testing.

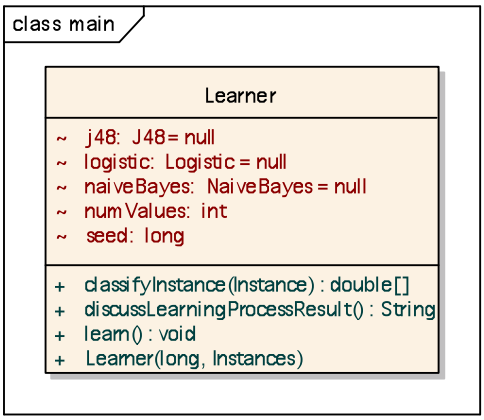


**Figure 7: The Learner implementation**

The method *"classifyTestSet()*" evaluates a set of aggregated TV-Anytime data, different from the training data set, against the trained algorithm and returns the result in a mapping table object, as depicted in Figure 8. This map contains pairs of a unique programme identifier (CRID [6]) and a quality rating value (1 to 10 stars).

### 2.3.6 Connecting the Generic Metadata Model

Figure 8 shows the structure of a basic metadata quality rating process, combining the Generic Metadata Model and the algorithms for handling static metadata.

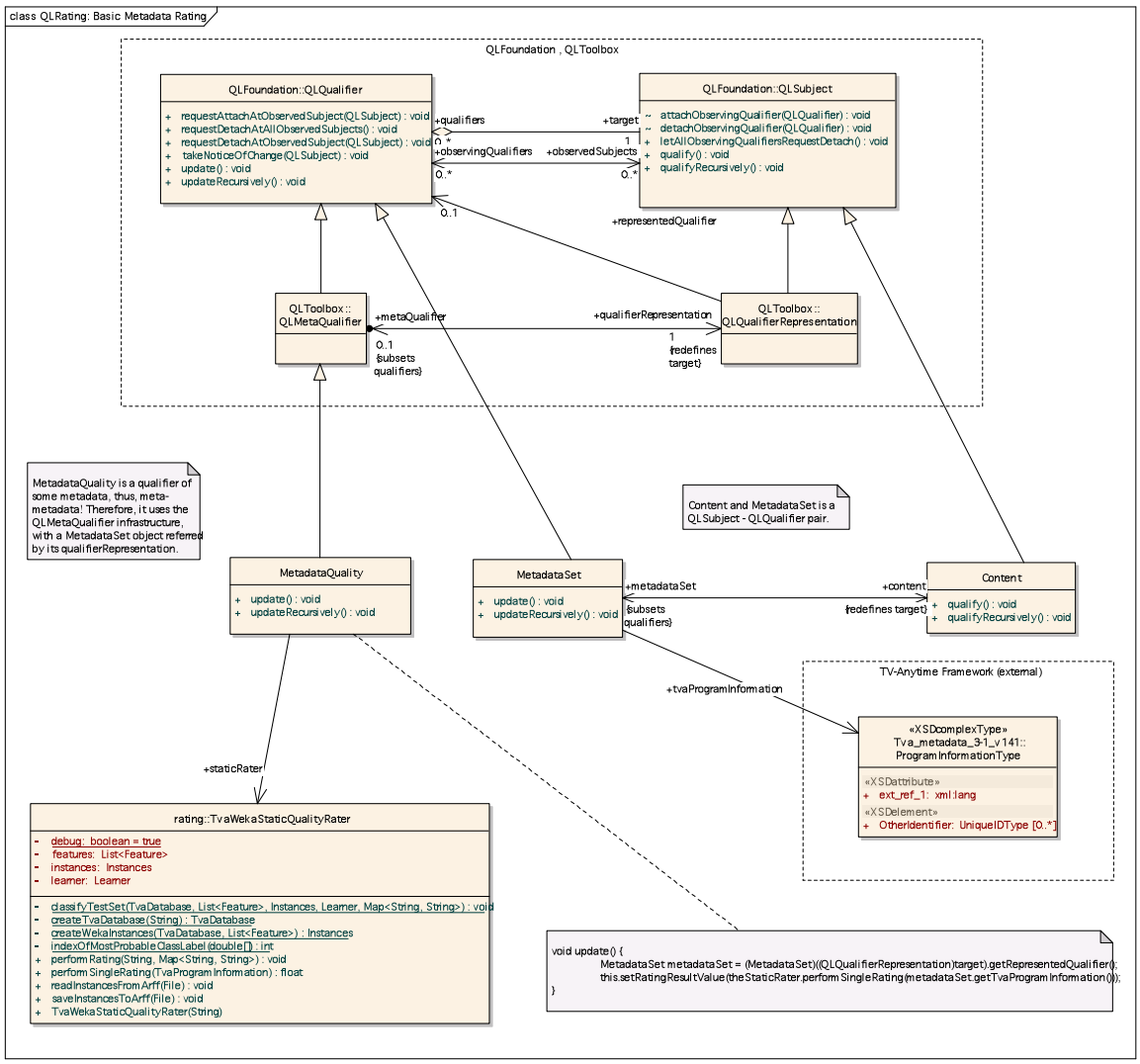


**Figure 8: Structure of a Metadata rating**

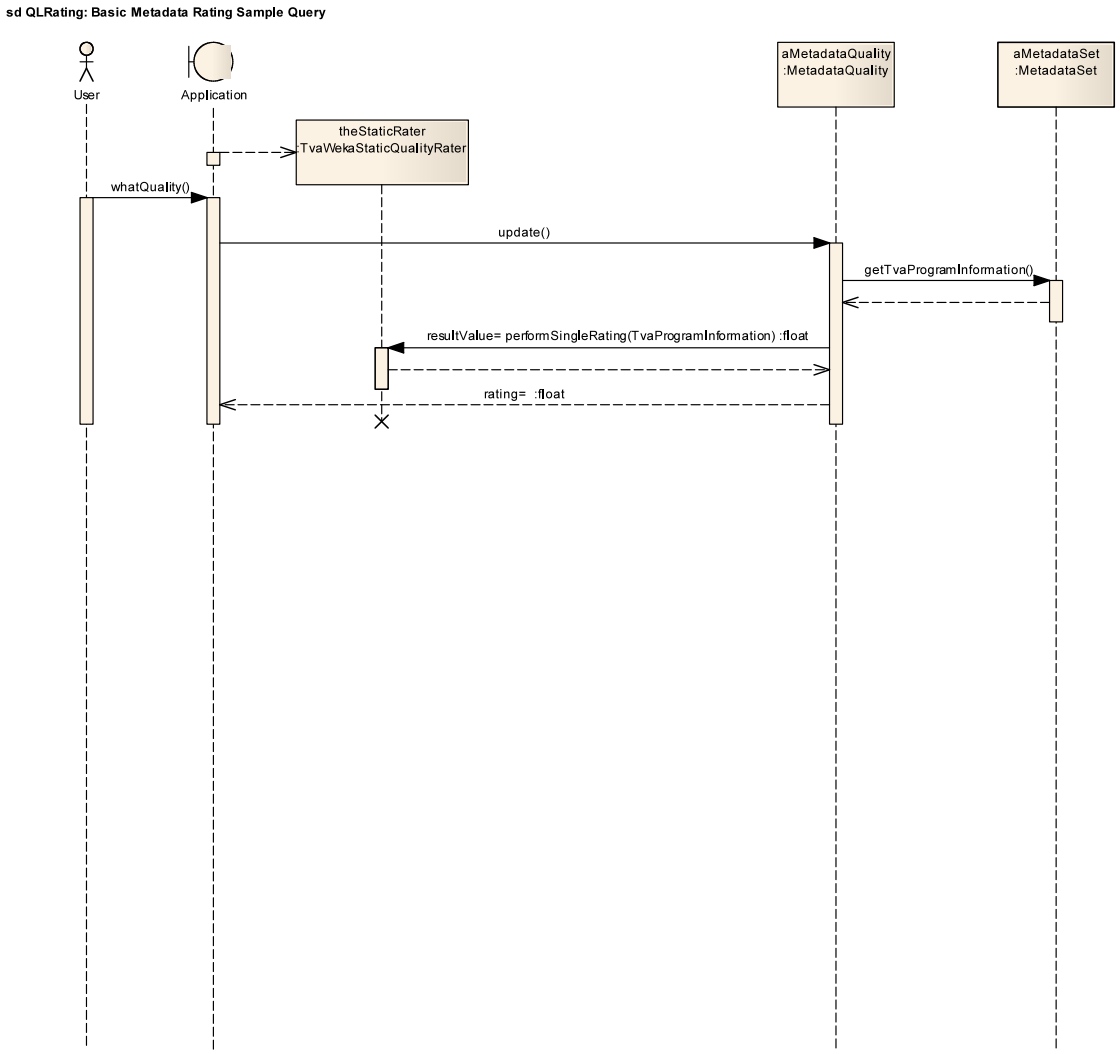


**Figure 9: Sequence diagram outlining steps for the quality rating process**

Figure 9 depicts the necessary steps that will be taken within the process of the metadata quality rating, based on the static metadata solution. At first, the user triggers the application to request the calculation of the quality of a metadata description: Note that, depending on the implementation of the application, this requesting process could also occur automatically. The application asks its reference to the *TvaWekaStaticQualityRater* to *update its knowledge* about the data. The quality is received from the *MetadataSet*, and forwarded to the self-learning algorithms that perform the calculation of the individual rating of the data. The received rating value is being returned to the application, which then displays the result in whatever format to the user (i.e. a number of stars out of n possible stars, a percentage value etc.).

## 2.4 Summary algorithms for static metadata

The QLectives machine learning abstraction layer is a powerful and future-proof light-weight tool that can be used to interconnect machine learning systems and data sources. It is designed in a data format – independent manner, allowing an interconnection with the Generic Metadata Model (D4.4.1), as well as with existing metadata formats, such as the TV-Anytime metadata format. Its functionality was tested successfully and is ready to become part of forthcoming QLectives software for the envisaged "Living Labs". The implementation is attached to this Deliverable, being part of the software bundles (as Java sources and in UML [15]).

# 3 Algorithms for Dynamic Metadata

“Dynamic metadata is a special type of metadata that is editable by numerous users, without centralized control”. From this definition, the following problem arises naturally: if anybody can edit the data (insert, delete or change) without a (centralized) controlling mechanism, it is very easy to add spam or even false data. *Detection* of this type of change is an important issue and challenging task of the artificial intelligence.

Metadata in QLectives, particularly QMedia will be editable by users as well. The detection of spam or false information in dynamic metadata in QLectives is a task similar to the detection and evaluation of changes made in Wikipedia [16] pages, since a dataset in Wikipedia is also editable by every page visitor. An approach for QLectives to maintain and even improve the quality of dynamically changeable metadata has many similarities to the detection and evaluation of edits in Wikipedia. Due to the huge amount of metadata being available in Wikipedia datasets compared to the amount of data being available in QMedia at this stage of the project, the dynamic approach has been developed, tested, evaluated and improved based on test data sets retrieved from the Wikipedia Encyclopaedia.

The section will be structured as follows. First, what is considered as “Detection & Evaluation of Dynamic Metadata” (DEDM) will be described, and thereby the task defined precisely. Second, an architectural overview of the approach will be given, and each architectural “building block” of the proposal will be presented in detail. Third, the experimental results will be presented, and finally, this section ends up with a summary of the task.

## *3.1 Evaluation of changes in Metadata*

### 3.1.1 The Metadata vandalism problem

First of all we define what “vandalism of Metadata” means. In this scope, vandalism is to be seen as any addition, removal, or change of content made in a deliberate attempt to compromise the integrity of Metadata. Common types of vandalism, in example in the Wikipedia domain, are the addition of obscenities or crude humour, page blanking, and

the insertion of nonsense into articles. Put another way, a vandalism edit is an edit made with bad intentions.

Any good-faith effort of a user or system to improve the Metadata, even if misguided or ill-considered, is not vandalism. Even harmful edits that are not explicitly made in bad faith are not to be seen as vandalism. For example adding a controversial personal opinion to an article is not vandalism, although reinserting it despite multiple warnings can be disruptive.

Vandalism has always been one of the major problems related to user-generated Metadata, yet there are only few automatic countermeasures. Instead, volunteers spend their time in reverting vandalism edits – time, which is not spent on improving other parts of i.e. the Wikipedia Encyclopaedia.

The task of DEDM is described as follows: Given a set of edits on Metadata elements (Wikipedia articles), separate the ill-intentioned edits from the well-intentioned edits. The goal of this task is to research and develop new, reliable ways to detect vandalism edits. In the next part, the task of the above mentioned challenge will be defined more precisely.

### 3.1.2 The Detection & Evaluation of Dynamic Metadata Task

In this subsection, further information about the DEDM will be given. This task is a classification challenge, so there was given a sample set that contains manually labelled samples from Wikipedia, and an evaluation set in which the labels were not presented. The sample set contains 15 000 labelled samples (edits) in which 944 samples are classified as “vandalism”. The sample set shows the following schema: there is a description about the edits that contains:

- the name or the IP number of the editor,
- the old and the new version id of the edited article/page,
- a URL that shows the differences between the versions,
- a comment that was written by the editor,
- the title of the article and further information about the edition and the annotators.

Furthermore, the given compressed file contains the pages belonging to the version identifiers and the class labels as well. The evaluation corpus follows the schema of the sample set (without the class labels of course), and contains more than 300 000 samples. The task is to predict the labels of the evaluation corpus based on the information described above.

The required output of the classification is the following: each line contains a given evaluation sample that is identified by the version IDs. The classified class label and the class probability follow the IDs.

```
26864258 27932250 V 0.92
28689695 87188208 R 0.50
85047080 85047157 V 0.67
80637222 91249168 R 0.43
```

For the easier understanding

- the first column is the edit's old revision ID.
- the second column is the edit's new revision ID.
- the third column denotes whether the edit vandalism (V) or regular (R), as determined by the classifier.
- the fourth column denotes the classifier's confidence. This value should be normalized to the interval [0,1], where 0 means the classifier is absolutely confident that the edit is regular, and 1 means that the classifier is absolutely confident the edit is vandalism.

The evaluation metric was the AUC, that is the *A*rea *U*nder the ROC (*R*eceiver *O*perating *C*haracteristic) *c*urve [17]. This measurement needs a natural order on the classified elements. The so called ROC curve is based on this order and the correct class labels. An AUC score denotes the probability that a classifier tells apart a randomly sampled vandalism edit from a randomly sampled regular edit, i.e. it measures the discrimination of the classifier. An optimal vandalism classifier achieves an AUC of 1, which means that it always correctly tells apart vandalism from regular editing, whereas a classifier with an AUC of 0.5 is not better than flipping a coin or in other words just choosing a random class label. As a rule of the thumb: an AUC above 0.9 is very good to excellent, an AUC between 0.8 and 0.9 is good, while anything below 0.8 is fair to poor.

## 3.2 The DEDM approach

As mentioned in the previous subsection, the task of the DEDM is to build a model which can assign a regular/vandalism label for each Metadata edit. For doing this the 15,000 manually labelled training samples that were collected and labelled can be used. In this solution, the machine-learning based approach was chosen, which divides the processing into the already mentioned two major phases: the *training* and the *prediction* phases. The elementary processing steps of these phases and the connections between them are shown in the Figure 10. The roles of these processing steps are described in the following to subsections.

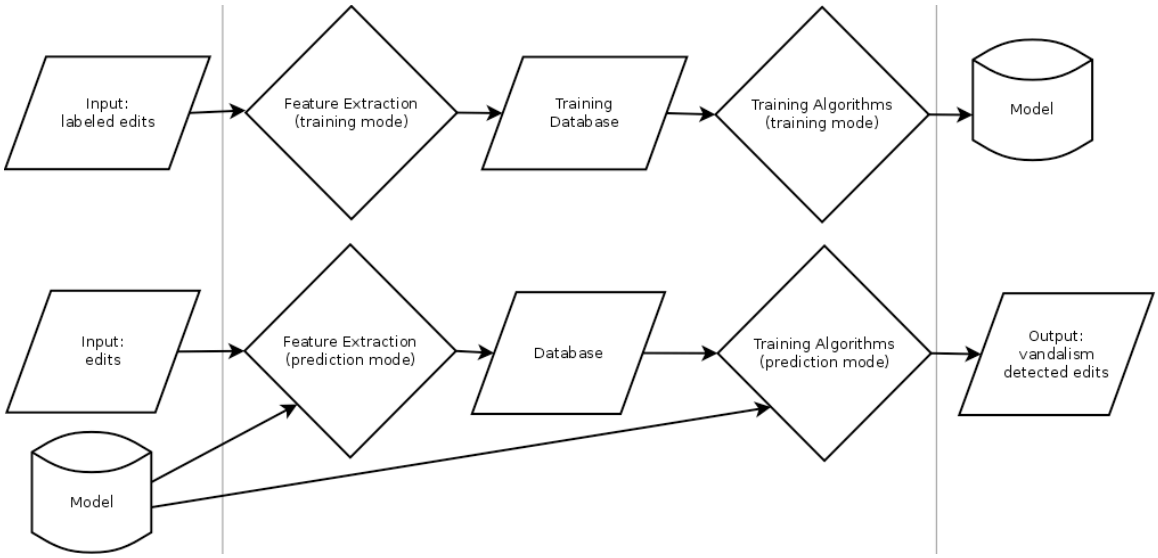


**Figure 10: Processing steps (top: training phase, bottom: prediction phase)**

### 3.2.1 Training Phase

The role of this phase is to build a *model* which can be used for predicting the labels of Metadata edits. These edits can remain unseen during the training phase. The model can be viewed as a *dense description of the knowledge* which can be extracted or captured from the training database.

A model can be considered as a good model when the probability of the correct labelling of the unseen edits is high. In a later section, the selection of the final model will be discussed.

1. Input:

It contains all the necessary data which was released by the shared task organizers, but in a more compact format. More precisely, the input is a single file which contains all the necessary information of an edit in one line in which the records are separated by TAB characters. First each line contains a key which is a unique ID – generated by us – for the certain edit, then it contains the text parts which were involved in current edit. For each text part, its role is also stored, i.e. whether the current text part was added, deleted or changed. This was the base data format of the processing change. All the text parts which remained untouched by the edits have been ignored from this format.

```
1111    327913606    328101741    vandalism    <c>Weyyyyyyyyyy
1141    329032895    329032928    regular <c>Rabbt    <c>Rabbit
1237    269896246    328119209    vandalism    <a>Dorothy Dalgliesh owns all
the other schools!
1325    326774153    326788380    regular <a>[[Category:Ultra Records artists]]
```

2. Feature Extraction (training mode):

The feature extraction processing step is a modular, complex subsystem of our system. The task of this module is to generate structured data from the unstructured textual edits. This is the key point where prior knowledge can be injected into the system. Technically, the Feature Extraction subsystem consists of a class hierarchy and an engine which can initialize and run the preconfigured instances of the Feature Extraction interface.

The “training mode” modifier in the title of this processing step indicates that features are being extracted from the training database, which means that the correct class labels are being retrieved, and the modules of this step can use this class information as well i.e. they can optimize based on the class labels or they can observe the distribution of the class labels.

The detailed discussion of the concrete features which were implemented in the system can be found later in section 3.2.3.

3. Training Database:

The format of the training database is the ARFF [18] format which is a widely used, WEKA related file format.

4. Training Algorithm (training mode):

The task of the training algorithm is crucial, as its goal is to build the model from the structured training database. In the field of machine-learning, a number of different training approaches which are based on different ideas have been introduced. In this work, some of the existing algorithms have been chosen. The implementations of these standard algorithms are available from many open sources. Additionally, existing implementations from the WEKA library have been used. Most of the algorithms which were used are available in this package, only one of them, a voting based meta-classifier was unavailable, but has been implemented as an extension of the WEKA library within this task. A short description of the applied algorithms can be found in section 3.2.3.

5. Model:

The model is a training algorithm dependent description of the knowledge which was extracted during the training. This can be viewed as the compressed format of the information which is available in the training database. This information or knowledge is used to predict the label of the unseen edits in the prediction phase.

### 3.2.2 Prediction phase

After producing a model in the training phase, one can predict the labels of *any* Metadata edits. The scenario of this process is shown under the prediction phase label in Figure 10.

1. Feature Extraction (prediction mode):

In this step, almost the same processes described in the training phase have been performed, except that in this case the modules cannot use the labels of the edits – they are considered as default values e.g. regular – and have to produce a compatible feature set (same features in the same ranges) than was used in the building of the model which is also input of the processing step.

2. Database:

The output of the feature extraction (prediction mode) is a structured dataset written in ARFF format, but here the class labels are missing, since they can be unknown.

3. Training Algorithm (prediction mode):

The task of this step is to *apply* the previously learnt model and predict the class labels of the Metadata edits i.e. to decide whether an edit is “vandalized” or not.

4. Output:

Since the WEKA based classification methods have been used, the output of the prediction was a simple ARFF file which contains the predicted and possible correct class labels of each edit (as shown in Figure 11). To produce the final expected format, a simple script that generated the expected format described in the previous section has been used.

.

```
@relation WikiVandalismTrain

@attribute UpperCaseStatisticDifferences numeric
@attribute NonLatterStatisticComments numeric
@attribute RepeatedCharSequences {false,true}
@attribute BadWords {false,true}
@attribute CommentStatistic {deleted,nothing,comment}
@attribute titl {n,a,d,c}
@attribute categori {n,a,d,c}
@attribute thi {n,a,d,c}
…
@attribute ClassLabel {regular,vandalism}

@data
{1 0.162791,2 0.04,3 0.2,6 nothing,7 0.000117,10 0.125,11 0.125,35 a}
{1 1,3 0.416667,6 nothing,7 0.000117,8 ip,9 Germany}
{7 0.000117,8 ip,9 Ireland,10 1,11 1,13 1,14 1,116 vandalism}
…
```

**Figure 11 : Prediction output**

After this brief overview of the system model of the solution, the proposed feature set and the used algorithms will be presented in detail.

### 3.2.3 The Feature Set

As mentioned above, the “Achilles’ heel” of the machine learning algorithms and methods is the correct feature set that has a high representation power belonging to the class labels. The features for this task have been selected intuitively and in a traditional way. The features come from two types of attributes, specially numeric and nominal attributes. The numeric attribute is a kind of feature that has in most cases continual numeric values, and the nominal attribute has a discrete and predefined value set. Furthermore, some of the features use dictionaries to calculate the result value for a sample. The following enumeration describes the features that have been developed:

1. Balanced-VSM:

This feature is a specialization of the commonly used and traditional Vector Space Model (VSM). The VSM is a simple bag of word statistic, namely *a document is represented as a vector and the dimension of the vector is the number of different words in the corpus*. If the actual document contains a word then the corresponding vector element is 1, otherwise 0. So there is a vector for each document. In our case one "document" was an edit, in other words a difference between the old and the new version of the Metadata element.

The simple VSM has been extended in the following way. In the VSM the vectors contain just 0 or 1. In our case we used 4 possible values as a vector element: *n, a, c* and *d*. The semantic meaning of these values comes from the properties of the edits, in this case:

- when the edit does not contain the word, then the value is *n*.
- when the word is in an added sequence, then the value is *a*.
- when the word is in a removed sequence, then the value is *d*.
- when the word is in a changed sequence, then the vale is *c*.

An extended version of the above described VSM has been developed within the task, the "Balanced-VSM". The name "balanced" comes from above mentioned extension, the features have been boosted. More precisely, the VSM has been built from a constrained sample set. This sample set was made from all of the vandalism samples and randomly chosen regular samples of the same size of vandalism samples. Then the VSM features have been weighted with the corresponding InfoGain scores (information gain [19]) is a measurement of the information gain of a feature with respect to the class). Iteratively, by re-sampling the regular samples, the most beneficial features were evolved.

2. CharacterStatistic:

This feature has a numeric value set and calculates the ratio of the upper case letter occurrences and the ratio of the characters that are neither letters, nor digits and nor white spaces.

3. CommentStatistic:

Making comments to the modification is permitted for each user who edits Metadata elements. With this attribute it is feasible to measure that a user makes, deletes or does nothing with the comment. So this feature has a nominal value set, and has three different available feature values that are *"deleted"*, *"nothing"* and *"comment"*. The value is

- *deleted* in case the comment of the edit is deleted,
- *comment* if the user has written into the comment field,
- *nothing* in all other cases.

4. RepeatedCharSequences:

One indication of vandalism is when somebody just repeats a sort string e.g. "asdasdasdasdasd". For this reason, the modification and the comments needed to be scanned in order to find and sort many occurring repeats, and mark these samples.

5. UserNameOrIP:

The user who edits Metadata can register him and choose a nick name or just edit the article. In the second case the nick name field will be filled with the IP number of the editor's computer. So, it comes to the decision adding a feature that describes whether a user is registered or not. In case of not, the country of the user will be identified based on the IP number, and added in replace of the nick name.

6. ValidWordRatio:

In this attribute, dictionaries have been deployed in order to create the feature values. Two different dictionaries have been used: a simple one that was exported from an English language dictionary, and another one that contains pejorative English expressions. The value of the feature is the ratio of the number of words in the two dictionaries.

## Technical Issues on Feature Extraction Subsystem

In this section, the technical details and the structural build up of the implementation will be outlined. The code was written in the Java programming language, and the advantages of Java have been exploited. First of all, this language is object oriented, and Java's inheritance mechanisms can easily be used to describe the hierarchy of the two different types of features: the numerical ranged and the nominal ranged ones. Each feature representation comes from these base classes, as depicted in the class diagram in Figure 12.

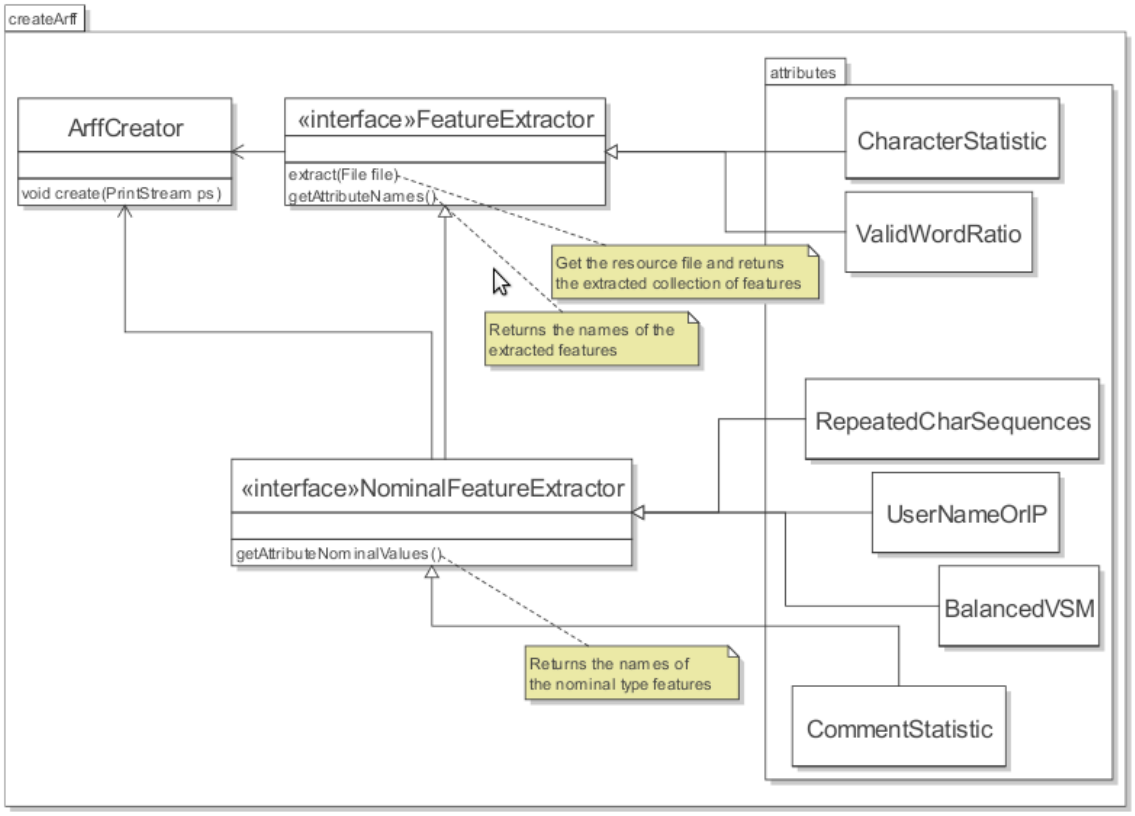


**Figure 12: The class diagram of the feature extraction subsystem**

Furthermore, the feature extractor engine has been implemented by adapting the reflection technique that is supported by Java, since the feature extraction object instances can be created and parameterized dynamically. This makes the system easily useable and extendable.

In the next section, the applied learning algorithms will be described in more detail.

### 3.2.4 The used training algorithms

Initially, some experiments took place, applying the following learning algorithms. For all of them their existing WEKA implementations have been used, except for the last one:

1. Support Vector Machine (SVM):

   Primarily, the SVM [20] was worked out for linear two-class classifications with margin, where margin means the minimal distance from the separating hyperplane to the closest data points. The SVM learning machine seeks for an optimal separating hyperplane, where the margin is maximal. An important and unique feature of this approach is that the solution is based only on those data points which are at the margin. These points are called “support vectors”. The linear SVM can be extended to a nonlinear one, when the initial problem is transformed into a *feature space* using a set of nonlinear basis functions. In the feature space – which can be very high dimensional – the data points can be separated linearly. A big advantage of the SVM is that it is not necessary to implement this transformation and to determine the separating hyperplane within the possibly very-high dimensional feature space. Instead, a kernel representation can be used, where the solution is written as a weighted sum of the values of certain kernel functions, evaluated at the support vectors.

2. Naïve Bayes:

   The *Naïve Bayes classifier* [21] is a simple probabilistic classifier based on applying Bayes' theorem with strong independence assumptions. The model of the classifier is a conditional model over a dependent class variable and conditional on several feature variables. The model parameters (i.e. class priors and feature probability distributions) can be approximated with relative frequencies from the training. The *Naïve Bayes classifier* applies this estimation model with a decision rule which picks the hypothesis that is most probable. This is known as the *maximum a posteriori* or MAP decision rule.

3. J48:

The *J48* [22] decision tree learning algorithm has also been used within this task, as it is a widely applied method in data mining. In a decision tree, each interior node corresponds to an attribute; an edge to a child represents a possible value or an interval of values of that variable. A *leaf* represents the most probable class label given the values of the variables represented by the path from the root. A tree can be learned by splitting the source set into subsets based on an attribute value test. This process is repeated on each derived subset in a recursive manner. The recursion stops either when further splitting is not possible or when the same classification can be applied to each element of the derived subset.

4. Logistic Regression:

The Logistic Regression [23] is another well studied machine learning method. It seeks to maximize the conditional probability of classes, subject to feature constraints (observations). This is performed by weighting features so as to maximize the likelihood of data.

5. Weighted Voting Metaclassifier:

This classifier combines several underlying classifier algorithms based on a weighted aggregation. In the WEKA environment each classification method can predict class membership probabilities as well. In this classifier we consider for each class the class membership probabilities of each underlying classifier and sum them up and multiply them with a predefined constants belonging to the corresponding classifier.

These are the chosen training algorithms. The reason why the selection fell onto the described combination of them is that these algorithms adopt quite different approaches, since they can capture the information stored in the training database in as many ways as possible.

## 3.3 Experimental Results

Several experiments have been performed during the developing period of the solution. In this section we describe the evaluations performed, as well as our final results measured on the released evaluation set.

### 3.3.1 Basic concepts of the evaluations

As mentioned earlier, our base evaluation metric was the AUC score, but additionally, we measured the F-measure and the Accuracy scores as well.

Since we only knew the correct class labels for the *training se*t, we could use this information for evaluation. But here we had to take care of avoid the *overfitting* [24] during the model building since we performed the 10-fold cross validation technique [25]. This is a well-known technique which divides the original training set into almost equal sized and characterized subsets, and iteratively changes the roles of these subsets in the following way: in the *i*th iteration, the *i*th subset is the evaluation set and the union of the remaining 9 sets is the training set. In this way we receive 10 different results from each iteration step. We can get the final, aggregated result as the mean of these 10 elementary scores. In our experiments *all* scores come from the above described evaluation process (10-fold cross validated scores).

### 3.3.2 Feature Sets and Training Algorithms

Our first evaluation focused on the examination of the relation of the defined feature sets to the chosen training algorithms. We defined some feature sets as subsets of the previously defined feature ranges and performed some learning-evaluation phases applying different training algorithms. The results are summarized in Table 1.

| Feature set | Classification method → | Most Frequent Class Classifier | J48, M=10 | Naïve Bayes | Logistic Regression | SVM, Linear Kernel, C=1 |
|---|---|---|---|---|---|---|
| Balanced VSM, dimension=100 | Accuracy | 0,8990 | 0,8999 | 0,8990 | 0,8996 | 0,9040 |
| | F-Measure (Class=V) | 0,0000 | 0,0900 | 0,1310 | 0,2660 | 0,2100 |
| | ROC Area (Class=V) | 0,4990 | 0,5230 | **0,7220** | **0,8130** | 0,5590 |
| Balanced VSM, dimension=100, using stop word list | Accuracy | 0,8728 | 0,9316 | 0,8164 | 0,8599 | 0,8766 |
| | F-Measure (Class=V) | 0,0000 | 0,7000 | 0,1650 | 0,2130 | 0,1660 |
| | ROC Area (Class=V) | 0,4990 | **0,8680** | 0,7750 | **0,8430** | 0,5430 |
| Balanced VSM, dimension=100 + all other features | Accuracy | 0,9386 | 0,9438 | 0,9105 | 0,9436 | 0,9449 |
| | F-Measure (Class=V) | 0,0000 | 0,3360 | 0,2350 | 0,3720 | 0,3070 |
| | ROC Area (Class=V) | 0,4990 | 0,8320 | **0,8580** | **0,8730** | 0,5960 |
| Balanced VSM, dimension=100, using stop word list + all other features | Accuracy | 0,9386 | 0,9460 | 0,9263 | 0,9422 | 0,9439 |
| | F-Measure (Class=V) | 0,0000 | 0,3790 | 0,2720 | 0,3550 | 0,2700 |
| | ROC Area (Class=V) | 0,4990 | 0,8280 | **0,8830** | **0,8870** | 0,5820 |

**Table 1: Overall results measured on different feature sets**

In Table 1, the rows show the used feature sets. Their meanings are summarized as follows:

- Balances VSM, dimension=100:
  The feature set in this case consists of only the Balanced VSM based features where the dimension of the VSM is 100.
- Balances VSM, dimension=100, using stop word list:
  In this case we applied exactly the same method as in the previous case, except that here we extended the VSM building process with the use of a “word blacklist” which filtered out the meaningless words from the VSM.
- Balances VSM, dimension=100 + all other features:
  Here we extended the simple VSM based feature set (without using the previously introduced stop word list) with all of the previously defined features.
- Balanced VSM, dimension=100, using stop world list + all other features:
  In this case we used exactly the same feature set as in the previous one, expect that the during the VSM building we again used the “word blacklist”.

For each feature set we show three different performance measures, namely the Accuracy, F-Measure and the AUC of Vandalism class. As we mentioned earlier, all of them are 10-fold cross validation scores. The reason for setting the dimension of the Balanced VSM to 100 is based on results from preliminary experiments, not shown in this report.

For each feature set we applied the 5 different classification methods and we have emphasized the best two AUC results of them using the colour red. Here, 4 of them are the previously introduced classification methods, and the so-called *"Most Frequent Class Classifier"* is a simple baseline classifier which classifies each sample using the most frequent class label seen in the training database.

As one can see, in the case of each feature set the *probabilistic based* learning algorithms achieve almost the *best results*, since they can give more reasonable class label probabilities – and the AUC performance measure uses this probabilities intensively – than those (J48, SVM) which are based on *discriminative models*. So we conclude that in case of any feature set – and AUC measurement – the most reasonable models are the probability based ones. However, in case of the last feature set (Balanced VSM, dimension=100, using stop world list + all other features), the J48 algorithm - a clearly discriminative model based approach - also shows pretty good AUC results.

In the case of the last feature set, the fact that one of the discriminative models could achieve a similar result to the probability based approaches indicates that this feature set is quite stable. Therefore, for further experiments we used this feature set.

### 3.3.3 Weighted Voting Based Classification

Several training algorithms in the experiments have been applied. Two of the algorithms (J48, SVM) are based on discriminative models, signifying these algorithms try to find a separating bound between the training samples coming from different classes. Other algorithms (*Naïve Bayes*, *Logistic Regression*) are based on probability based models i.e. they try to model the underlying probability distributions of the samples deriving from different classes.

Our observation that these algorithms are working in completely different ways led us to the assumption that maybe the algorithms based on different approaches produced different errors. This naturally formed the idea that the *combination* of these different approaches should be given a try. Hence, we chose the three best algorithms, namely the J48, the *Naïve Bayes* and the *Logistic Regression*, and built the previously introduced *Weighted Voting Metaclassifier* on top of these three algorithms. The only questions arising are the following:

- how should we determine the weights of the underlying classifiers,
- are the found weights optimal on the evaluation set?

The discussion of these questions can be found in the next subsection.

### 3.3.4 Fine - tuning the parameters

We decided to use the 10-fold AUC scores as the optimality measurement of a weight setting. But first we did not know whether this optimality criterion is optimal on a separated evaluation set, or whether it causes “overfitting”. For checking the validity of our optimization process, we designed a two steps evaluation method: First we performed a 5-fold cross validation based splitting, and for each iteration loop of this method we considered the training set as an optimization set. We performed a 10-fold cross validation based optimization of the parameters on that set and we observed whether this selection is optimal on the remaining 5-fold based evaluation set or not. This optimization was performed over the complete parameter space. The summary of our optimization for the first 5-fold iteration is depicted in Figure 13.

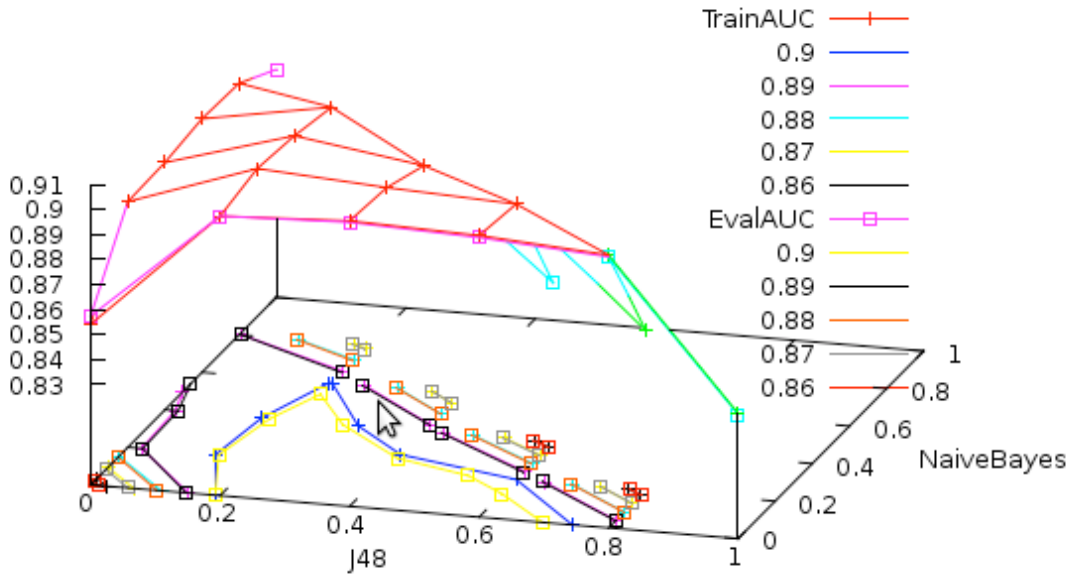


**Figure 13: The AUC results belonging to different weightings**

Obviously, the results of the evaluation set and the results of the optimization sets are correlated. Since we can state that this optimization criterion is valid, we found that the optimal weighting of the algorithms is the following:

| Name of the underlying classifier | Weight |
|---|---|
| J48 | 0.30 |
| Naïve Bayes | 0.09 |
| Logistic Regression | 0.61 |

**Table 2: Optimal weighting for Weighted Voting Metaclassifier**

The achieved AUC 10-fold cross validation based score of the optimally *weighted metaclassifier*, comparable to those values presented in Table 1, is 0.9129. This result is significantly higher than the best achieved score in Table 1. Thus, we used this combined model for making our final predictions on the evaluation set.

As a final prediction, we performed a completely new training-prediction circle using the full training set and our preliminary observation during the training phase and we predicted the class labels of the official evaluation set during the prediction phase. The used training set was the *"Balanced VSM, dimension=100, using stop world list + all other features"* feature set. The applied model was the *Weighted Voting Metaclassifier* using the parameterization presented in Table 2.

The results achieved an AUC score of **0.87669** on the evaluation set.

Although our final result on the evaluation set is poorer than measured in our experiments, the difference is statistically not significant. We have to notice that we tried to hold our solution as automatic as possible, since we tried to avoid using any prior knowledge (except a general stop word list, which is static) or applying manual interactions. Better results might be achievable when eliminating these assumptions.

## 3.4 Summary algorithms for dynamic metadata

The experiments in the field of detecting vandalism and spam in dynamic metadata have been carried out within the DEDM task. Although the solution achieved a good, but not excellent performance, we feel that our work has a strong contribution. This contribution is twofold.

1. First, we developed a strong feature representation for the task which can be built in a fully automatic manner and some of these features are pretty complex e.g. the Balanced VSM representation. This is a novel extension of the basic VSM representation and suitable for learning tasks where the class labels have a highly unbalanced distribution.
2. Second, we successfully combined classification methods in a weighted manner, where the weights have been optimized as hyper parameters. For this reasons we plan to make a scientific publication from our results soon.

Besides that, especially the combination of several types of algorithms seems to be the optimal way to go, when it comes to the implementation of solutions for detecting high-quality and spam metadata, especially in Peer-to-Peer environments. Details related to implementation in distributed environments will become part of D4.4.3, the implementation guidelines for HQ and Spam Metadata. The implemented software is attached to this Deliverable, being part of the software bundles (as Java sources and in UML).

# 4 Conclusions

In this document, the combination of two solutions to evaluate the quality of metadata has been presented. The first approach is designed to analyse the quality of *static metadata* entries (mainly initial entries), whereas the second one is developed to be applied for *dynamic metadata*. The so-called “dynamic approach” analyses modifications of metadata entries by users (or applications). This is particularly relevant to detect decreases in quality of metadata, especially false entries by intent, or in other words vandalism. Both solutions are deploying self-learning algorithms (analogous to email-junk filters) which – after the system has been trained – can predict the quality levels of metadata sets in self-adjusting ways. For best results, the *QLectives self-learning metadata rating system should combine both*, the static as well as the dynamic solution. This gives us on hand a powerful and future-proof light-weight tool providing a high degree of flexibility, since it is designed in a data format- and application implementation-independent manner by deploying self-learning algorithms. The system concept presented in this document interconnects perfectly with the Generic Metadata Model (D4.4.1), as well as with existing metadata sources and formats, such as DVB-IP with the TV-Anytime metadata format. Thanks to the given flexibility, alterations and adaptations at the QMedia platform functionality and the identification of use-cases on the application layer - even at later stages, bearing in mind the empirical cycles of the “Living Labs”- is enabled.

Finally, the functionality of both solutions has been tested successfully with reasonable amounts of test data. Several features, candidates to be implemented later on in the QMedia p2p software, have been developed for setting up and testing the system. The features already tested - amongst others yet to be developed - will be presented in the next deliverable D4.4.3 *“Implementation / integration guidelines for HQ-MD in QMedia v2”*. Once implemented in QMedia on application layer, the test results of the quality of metadata in the Living Lab will be presented in the last deliverable of WP4.4 (at the end of the project) in D4.4.4 *“Test report on HQ-MD implementation”*.

The implementations (static and dynamic solution) are attached to this Deliverable, being part of the software bundles (both, in UML and Java).

# 5 References

[1] YouTube website: http://www.youtube.com

[2] Peer-to-peer networks: http://searchnetworking.techtarget.com/sDefinition/0,,sid7_gci212769,00.html

[3] QLectives Deliverable D4.4.1 “Generic Metadata Model” http://qlectives.eu/docs/D4.4.1.pdf

[4] Java (Sun) website: http://java.sun.com

[5] Eclipse framework: http://www.eclipse.org

[6] TV-Anytime Forum: http://www.tv-anytime.org

[7] DVB-IP standards: http://www.dvb.org

[8] Universal Plug and Play (UPnP): http://www.upnp.org

[9] Digital Living Network Alliance (DLAN): http://www.dlna.org

[10] Podcast definition: http://en.wikipedia.org/wiki/Podcast (August 2010)

[11] iNEM4U website: http://www.inem4u.eu

[12] iNEM4U Deliverable D3.4, Metadata Services: https://doc.telin.nl/dsweb/View/Collection-16387

[13] WEKA website: http://www.cs.waikato.ac.nz/ml/weka

[14] WEKA development at sourceforge: http://sourceforge.net/projects/weka

[15] Unified Modelling Language: http://www.uml.org

[16] Wikipedia: http://www.wikipedia.org/

[17] Fawcett, T. "ROC Graphs: Notes and Practical Considerations for Researchers", Technical Report HPL-2003-4.

[18] ARFF database format: http://www.cs.waikato.ac.nz/~ml/weka/arff.html

[19] http://en.wikipedia.org/wiki/Information_gain_in_decision_trees (August 2010)

[20] Burges, C.J.C.: A tutorial on support vector machines for pattern recognition. Data Mining and Knowledge Discovery 2(2) (1998) 121–167

[21] Rish, Irina: An empirical study of the naive Bayes classifier. IJCAI 2001 Workshop on Empirical Methods in Artificial Intelligence. (2001)

[22] Quinlan, J.R.: C4.5: Programs for Machine Learning. Morgan Kaufmann, San Francisco (1993)

[23] Berger, A.L., Pietra, S.D., Pietra, V.J.D.: A maximum entropy approach to natural language processing. Computational Linguistics 22, 39–71 (1996)

[24] C. Bishop, Pattern Recognition and Machine Learning (Information Science and Statistics). Springer, August 2006.

[25] http://en.wikipedia.org/wiki/Cross-validation_%28statistics%29 (August 2010)

# 6 Annex A

The source code listed below creates the *"Weighted Voting Meta-classifier"*, which was implemented by this task as an extension of the WEKA library.

```
public class WeightedVote extends AbstractClassifier implements Classifier {

  private static final long serialVersionUID = 3360871868283712190L;
  protected Classifier[] classifiers = new Classifier[3];
  protected double[] weights = new double[]{0.3, 0.09, 0.61};


  public WeightedVote() throws Exception {
    classifiers[0] = AbstractClassifier.forName("weka.classifiers.trees.J48",
Utils.splitOptions("-M 10"));
    classifiers[1] =
AbstractClassifier.forName("weka.classifiers.bayes.NaiveBayes",
Utils.splitOptions(""));
    classifiers[2] =
AbstractClassifier.forName("weka.classifiers.functions.Logistic",
Utils.splitOptions(""));
  }


  public WeightedVote(double w1, double w2) throws Exception {
    weights[0] = w1;
    weights[1] = w2;
    weights[2] = 1.0 - w1 - w2;

    classifiers[0] = AbstractClassifier.forName("weka.classifiers.trees.J48",
Utils.splitOptions("-M 10"));
    classifiers[1] =
AbstractClassifier.forName("weka.classifiers.bayes.NaiveBayes",
Utils.splitOptions(""));
    classifiers[2] =
AbstractClassifier.forName("weka.classifiers.functions.Logistic",
Utils.splitOptions(""));
  }


  public void buildClassifier(Instances data) throws Exception {
    for (int i = 0; i < classifiers.length; i ++) {
      classifiers[i].buildClassifier(data);
    }

  }


  public double[] distributionForInstance(Instance instance) throws Exception {
    double[] finalProbs = new double[instance.numAttributes()];
    for (int j = 0; j < classifiers.length; j ++) {
      double[] probs = classifiers[j].distributionForInstance(instance);
      for (int i = 0; i < finalProbs.length && i < probs.length; i ++) {
        finalProbs[i] += weights[j] * probs[i];
      }
```

```
    }
    return finalProbs;
  }


  public Capabilities getCapabilities() {
    Capabilities result = super.getCapabilities();
    result.disableAll();

    // attributes
    result.enable(Capability.NOMINAL_ATTRIBUTES);
    result.enable(Capability.NUMERIC_ATTRIBUTES);

    // class
    result.enable(Capability.NOMINAL_CLASS);

    return result;
  }
}
```

# 7 Annex B

Here, the directory listing of all software parts (the content of the attached file "D442_Software.zip") of this Deliverable are outlined:

- „QLectives.UML.D442.QLFeatures.xml"
  - UML representation of the features
- „QLectives.UML.D442.QLRating.xml"
  - UML representation of the rating solutions
- „UML_EA.DTD"
  - The Document Type Description (DTD), according to which the UML representations have been created.
- "QL.D442.staticMD.zip"
  - "StaticMetadata" - the Eclipse Java project of the solution for static & initial metadata
    - "doc"
      - Javadoc representation / documentation of the implementation
    - "lib"
      - Libraries required by the Java project
    - "res"
      - Resources (SPAM dictionaries) required by the Java project
    - "src"
      - Java source code of the project implementation
    - "xml"
      - The static metadata, used for teaching and testing of the Java project implementation
    - "qualityResults.xml"

- An exported example results list, created by the test runner implementation

- "QL.D442.dynamicMD.zip"
  - "DynamicMetadata" - the Eclipse Java project of the solution for dynamic metadata
    - "bin"
      - Binaries of the Java project
    - "lib"
      - Libraries required by the Java project
    - "maps"
      - Resources required by the Java project
    - "resources"
      - Resources required by the Java project
    - "src"
      - Java source code of the project implementation
    - "dynamicData.2100.zip"
      - Test data for the Java project