
JCLAL: A Java Framework for Active Learning

Version 1.1

USER MANUAL AND DEVELOPER DOCUMENTATION

Last update on May 5, 2016.

CONTENTS

| | | |
|----------|---|-----------|
| 1 | Introduction | 5 |
| 1.1 | What is JCLAL? | 6 |
| 1.2 | What to expect in future releases of JCLAL? | 7 |
| 1.3 | Available AL libraries on the Internet | 8 |
| 2 | User manual | 13 |
| 2.1 | Download and install | 13 |
| 2.2 | Compilation | 16 |
| 2.3 | Configuring an experiment | 16 |
| 2.4 | Executing an experiment | 18 |
| 2.5 | Study cases | 20 |
| 2.5.1 | Margin sampling query strategy | 20 |
| 2.5.2 | Entropy sampling query strategy | 21 |
| 2.5.3 | Actual deployment example | 22 |
| 2.5.4 | Scalability of the JCLAL framework | 23 |
| 2.6 | Using JCLAL in WEKA's explorer | 25 |
| 3 | Developer documentation | 27 |
| 3.1 | Overview | 27 |
| 3.2 | Packages | 27 |
| 3.2.1 | Package net.sf.jclal.core | 29 |
| 3.2.2 | Package net.sf.jclal.activelearning.algorithm | 29 |

| | | |
|---------------------|--|-----------|
| 3.2.3 | Package net.sf.jclal.querystrategy | 30 |
| 3.2.4 | Package net.sf.jclal.singlelabel.querystrategy | 30 |
| 3.2.5 | Package net.sf.jclal.multilabel.querystrategy | 30 |
| 3.2.6 | Package net.sf.jclal.activelearning.scenario | 31 |
| 3.2.7 | Package net.sf.jclal.activelearning.batchmode | 31 |
| 3.2.8 | Package net.sf.jclal.classifier | 31 |
| 3.2.9 | Package net.sf.jclal.experiment | 32 |
| 3.2.10 | Package net.sf.jclal.listener | 32 |
| 3.3 | API reference | 32 |
| 3.4 | Requirements and availability | 33 |
| 3.5 | Implementing new features | 33 |
| 3.5.1 | Implementing new query strategies | 33 |
| 3.5.2 | Implementing a new evaluation method | 36 |
| 3.5.3 | Implementing a new oracle | 37 |
| 3.5.4 | Implementing a new stop criterion | 38 |
| 3.5.5 | Implementing a new listener | 38 |
| 3.6 | Running unit tests | 39 |
| Bibliography | | 41 |

1. INTRODUCTION

It is well known that for obtaining good performance levels in learning systems, the supervised learning algorithms must often be trained on hundreds (even thousands) of labelled instances. Sometimes these labels come at little or no cost. However, for numerous supervised learning tasks, labelled instances are very difficult, time-consuming, or expensive to obtain [1]. Consequently, nowadays is common to find real-world problems that involve a small number of labelled examples in union with a large number of unlabelled examples. These problems have motivated the development of new machine learning techniques.

The Semi-Supervised Learning (SSL) [2] and Active Learning (AL)¹ [1] are the two main areas of research that are concerned with the development of algorithms for learning predictive models from labelled and unlabelled data at the same time. The main hypothesis of AL states: *if the learning algorithm can choose the data from which it learns, then the labelling effort and the cost of training an accurate model will be reduced* [1].

Particular successful applications of AL methods include text categorization, image classification, protein structure prediction, natural language processing, information retrieval, information extraction and many more. The AL methods learn an accurate model by means of an iterative process. In each iteration, a query strategy selects the most informative instances from the unlabelled set. An oracle labels the selected instances, the instances are added to the labelled set, and the predictive model is induced. Figure 1.1 shows an example of the AL cycle with a pool of unlabelled instances.

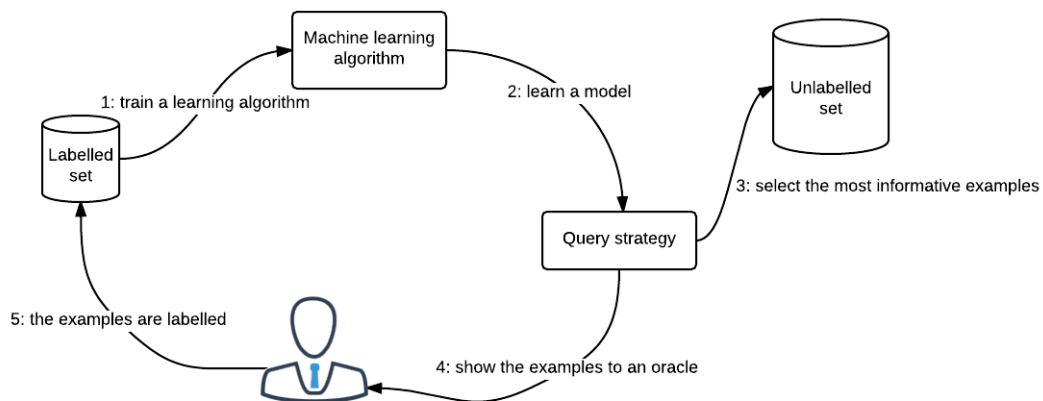


Figure 1.1: The active learning cycle.

Currently, there are several software tools which assist the experimentation process and development of new algorithms in the data mining and machine learning areas, such as Rapid Miner², WEKA³, Scikit-

¹Also known as query learning or optimal experimental design in the statistics literature.

²<https://rapidminer.com/>

³<http://www.cs.waikato.ac.nz/ml/weka/>

learn⁴, Orange⁵, KEEL⁶, etc. However, these tools are focused to Supervised and Unsupervised Learning problems.

Some libraries and independent code that implement AL methods can be found on the Internet (see Section 1.3), such as Vowpal Wabbit, DUALIST, Active-Learning-Scala, TexNLP and LibAct. The Active-Learning-Scala and LibAct libraries are mainly focused to AL, they implement several AL strategies that have been proposed in the literature. On the other hand, Vowpal Wabbit, DUALIST and TexNLP have been designed for a different purpose, but they also include some AL methods. To date, and in our opinion, there has been insufficient effort towards the creation of a computational tool mainly focused to AL. In our view, a good computational tool is not only a tool which includes the most relevant AL strategies, but also one that is extensible, user-friendly, interoperable, portable, etc.

The above situation motivated the development of the JCLAL framework (Java Class Library for Active Learning). JCLAL is an open source software for researchers and end-users to develop AL methods. It includes the most relevant AL strategies that have been proposed in single-label and multi-label learning paradigms. JCLAL is user-friendly. It provides the necessary interfaces, classes and methods to develop any AL method.

This manual assumes that readers have previous knowledge about AL. For non-expert readers, we recommend to consult the technical report of AL literature which can be downloaded at <http://active-learning.net>.

What is JCLAL?

JCLAL is inspired in the architecture of JCLEC [3, 4] which is a framework for evolutionary computation developed by the KDIS research group of the University of Córdoba, Spain. JCLAL provides a high-level software environment to perform any kind of AL method. It has an architecture that follows strong principles of object-oriented programming, where it is common and easy to reuse code. The main features of the library are the following:

- *Generic*. Through a flexible class structure, the library provides the possibility of including new AL methods, as well as the ability to adapt, modify or extend the framework according to developer's needs.
- *User friendly*. The library has several mechanisms that offer a user friendly programming interface. It allows the user to execute an experiment through a XML configuration file, as well as directly from Java code. We recommend that the users run the experiments through a configuration file. This is the fastest and easiest way to perform an experiment.
- *Portable*. The library has been coded in the Java programming language. This ensures its portability between all platforms implementing a Java Virtual Machine.
- *Elegant*. The library has a coherent class structure that follows good object oriented and generic programming principles.
- *Interoperable*. The use of XML provides a common ground for tools development and to integrate the framework with other systems.

⁴<http://scikit-learn.org>

⁵<http://orange.biolab.si/>

⁶<http://sci2s.ugr.es/keel/>

- *Open Source.* The source code is free and available under the GNU General Public License (GPL). Thus, it can be distributed and modified without any fee. It is hosted at SourceForge, GitHub, OSSRH repository provided by Sonatype, and Maven Central Repository.

JCLAL aims to bring the benefits of machine learning open source software [5] to people working in the area of AL. The library offers several state-of-the-art AL strategies for single-label and multi-label learning paradigms (see Table 1.1). In next releases, we hope to provide AL strategies related with multi-instance and multi-label-multi-instance learning paradigms.

| Single-label strategies | Multi-label strategies |
|-----------------------------|--|
| Entropy Sampling | Binary Minimum [6] |
| Least Confident | Max Loss [7] |
| Margin Sampling | Mean Max Loss [7] |
| Vote Entropy | Maximal Loss Reduction with Maximal Confidence [8] |
| Kullback Leibler Divergence | Confidence-Minimum-NonWeighted [9] |
| Expected 0/1-loss | Confidence-Average-NonWeighted [9] |
| Expected Log-Loss | Max-Margin Prediction Uncertainty [10] |
| Variance Reduction | Label Cardinality Inconsistency [10] |
| Information Density | Information Density |

Table 1.1: AL strategies included in JCLAL. More information about the single-label AL strategies can be found in [1].

The Stream-Based Selective Sampling and Pool-Based Sampling scenarios are supported. JCLAL provides the interfaces and abstract classes for implementing batch-mode AL methods and other types of oracle. Furthermore, the library has a simple manner of defining new stopping criteria which can change according to the problem. The library contains a structure which allows a set of listeners to simply define the events of an algorithm. The AL methods can be tested using the following evaluation methods: Hold-Out, k -fold cross validation, 5X2 cross validation and Leave-One Out. A method for actual deployment is also provided. The library contains a set of utilities, e.g. algorithms for random number generation, sort algorithms, sampling methods and methods to compute, for example, AUC and non-parametric statistical tests.

What to expect in future releases of JCLAL?

We are studying to include some new features that come with the JDK8, e.g. the use of the Stream API that allows to create streams from collections, files and other sources. The Stream API enables lazy operations and parallel data processing.

We are also working in a JCLAL's module for distributed computing using Spark. Spark is well suited for highly iterative algorithms, e.g. machine learning algorithms, that require multiple passes over a dataset, as well as reactive applications that quickly respond to user queries by scanning large in-memory datasets. Spark allow us to use fast learning algorithms that comes with the Spark MLlib library, as well as partitioning the data. Consequently, the time needed in some processes, such as scoring the unlabelled data and training the learning algorithms, can be considerably reduced.

On the other hand, we are working in a user-friendly GUI that allows to configure the experiments, and we are also working in a more advanced plug-in for WEKA's explorer. In next releases, we hope to include strategies related with multi-instance and multi-label-multi-instance learning paradigms.

Available AL libraries on the Internet

The most relevant libraries including AL methods are listed. Apologies if any other existing libraries have not been included in this list.

Vowpal Wabbit (VW)

- *URL.* https://github.com/JohnLangford/vowpal_wabbit/wiki
- *Description.* VW is a library to build fast learning algorithms, and it is useful as a platform for research and experimentation. VW contains several optimization algorithms with the baseline being sparse gradient descent on a loss function (several functions are available). VW allows to resolve several types of problems, such as Binary Classification, Regression, Multiclass and Multi-label Classification, Cost-Sensitive Multiclass Classification and Sequence Predictions. Although the main purpose of VM library is not focused to AL, it also allows to build the models by interacting with teachers.
- *User friendly.* VW is easy to use. VW includes some features that enable the user to experiment with actual deployments, e.g. building a classifier with the interaction of users by using teachers. VW is well documented, and the author provides several examples and tutorials.
- *Extensibility.* The VW's code is easy to extend.
- *Programming Language.* VW has been implemented in C++, and it also provides some interfaces to work with C#, Python, Java (through JNI) and R languages.
- *Portability.* The author guarantees the portability of VW library for Microsoft Windows, MacOS X and Linux distributions.
- *License.* VW is released under a modified BSD license.
- *AL methods.* The Importance Weighted AL (IWAL) strategy is implemented. VW library supports Stream-based AL.
- *Scalability.* It can be effectively applied on learning problems with high dimensionality. The size of the set of features is bounded independently of the amount of training data using the hashing trick.

DUALIST

- *URL.* <https://github.com/burrsettles/dualist>
- *Description.* DUALIST is a practical tool to expedite annotation/learning in NLP tasks. It is a utility software to do AL with instances and semantic terms. It uses some methods from AL and SSL areas to build text-based classifiers.
- *User friendly.* DUALIST includes a Web-based GUI that enables the user to experiment with actual deployments. DUALIST is poorly documented. However, a demo video is provided by the author.
- *Extensibility:* The class structure is extensible.
- *Programming Language.* DUALIST requires Java and Python to work properly. It is provided with most of the dependencies it needs to work, being Play! Web framework for running the Web-based GUI the only exception.

- *Portability.* All platforms that supports JVM and Python.
- *License.* This software is released under the Apache License v2.0.
- *AL methods.* DUALIST implements the most known query strategies based on uncertainty sampling, such as Least Confident, Margin Sampling and Entropy Sampling strategies. It supports Stream-based and Pool-based AL.
- *Scalability.* DUALIST is developed to run on a single computer and loads all data into the main memory. It is robust for hundreds of thousands of instances and features on modern hardware, but it may be difficult to use beyond that (exposed by the DUALIST 's author).

Active-Learning-Scala

- *URL.* <https://github.com/active-learning/active-learning-scala>
- *Description.* This is an AL library for Scala language based on mls (machine-learning-scala) and elm-scala (extreme-learning-machine-scala) frameworks. It also supports classifiers that come from WEKA [11] and CLUS⁷ frameworks.
- *User friendly.* It is poorly documented. Examples and tutorials about the use of Active-Learning-Scala library are not provided.
- *Extensibility.* The library is designed for an easy extension of AL strategies.
- *Programming Language.* The library has been implemented in the Scala programming language.
- *Portability.* All platforms that supports the JVM.
- *License.* This software is released under the GNU General Public License.
- *AL methods.* Active-Learning-Scala includes a large list of AL strategies, such as strategies that belong to Uncertainty Sampling, Query by Committee, Density Weighted, Expected Model Changed and Expected Error Reduction categories. It supports Pool-based and Stream-based AL.
- *Scalability.* The authors do not give evidences about the scalability of this library. However, the Scala language has some features that make it scalable and the active-learning-scala library could benefit of these features.

TexNLP

- *URL.* <https://github.com/utcompling/texnlp>
- *Description.* TexNLP (Texas Natural Language Processing tools) library implements some SSL algorithms combined with AL methods. TexNLP allows to learn Hidden Markov Models for tagging.
- *User friendly.* Documentation about how to build and run this software is available. However, TexNLP is not user-friendly (exposed by the TexNLP's authors). TexNLP is poorly documented. Examples and tutorials about the use of the library are not provided.
- *Extensibility.* The library is designed for an easy extension.
- *Programming Language.* TexNLP has been implemented in Java. It also provides some interfaces for working with Python.

⁷<http://clus.sourceforge.net>

- *Portability.* All platforms that support a JVM.
- *License.* This software is released under the GNU General Public License.
- *AL methods.* Labels can be predicted for new sequences using the Viterbi algorithm. Some AL strategies that belong to the uncertainty sampling category are implemented.
- *Scalability.* The authors don't give evidences about the scalability of this software.

Active Semi-Supervised Learning

- *URL.* http://www.cs.cmu.edu/~zhuxj/pub/semisupervisedcode/active_learning
- *Description.* In fact, this is not even a software, but merely a research code. It implements an AL strategy on top of SSL and it supports multi-class classification.
- *User friendly.* The code is easy to understand and use. The code is not documented.
- *Extensibility.* The code is not easy to extend.
- *Programming Language.* The code was written in Matlab.
- *Portability.* All platforms that support Matlab.
- *License.* None.
- *AL methods.* A version of Expected Error Reduction strategy is implemented on a Pool-based AL scenario.
- *Scalability.* No scalable.

LibAct

- *URL.* <https://github.com/ntucllab/libact>
- *Description.* LibAct is, to the best of our knowledge, the most recent software for AL. The package implements several popular AL strategies. It also includes the AL by learning meta-strategy that allows to automatically learn the best strategy on the fly.
- *User friendly.* It is a Python package designed to make AL easier for real-world users. Models can be easily obtained by interfacing with models in the Scikit-learn framework. A software usage tutorial is not provided and there are very few examples.
- *Extensibility.* The library is designed for an easy extension of AL strategies, models and labellers.
- *Programming Language.* LibAct has been coded in Python.
- *Portability.* All platforms that support Python.
- *License.* This software is released under a modified BSD license.
- *AL methods.* LibAct implements several query strategies, such as uncertainty sampling (Least Confident, Margin Sampling and Entropy Sampling), variance reduction, QUIRE strategy (AL by QUerying Informative and Representative Examples), Query by committee, HintSVM (Hinted Support Vector Machine) and ALBL (AL by Learning). It supports Pool-based AL scenario.
- *Scalability.* The authors do not give evidences about the scalability of this software.

Generally speaking, there are several libraries available on the Internet which include AL methods. A small number of these libraries are mainly focused to AL, e.g. the Active-Learning-Scala and LibAct libraries. Most software have been designed for a different purpose and some AL methods have been included into them, e.g. the VW, DUALIST and TexNLP libraries.

Regarding to the libraries that are mainly focused to AL and which are compatible with Java language, the most relevant is Active-Learning-Scala library. The Active-Learning-Scala library was implemented in the Scala language, and also gives some compatibility with classifiers that come from WEKA framework. However, other types of AL methods are still difficult to find in the libraries, e.g. batch-mode AL strategies, multi-label AL strategies and multi-instance AL strategies.

2. USER MANUAL

This chapter describes the process for end-users and developers to download, install and use the JCLAL framework.

Download and install

The JCLAL framework can be obtained in one of the following ways, which are adapted to the user's preferences:

- Download runnable jar file and source files from SourceForge.net or GitHub:

The runnable jar file provides the user with the fastest and easiest way to execute an experiment, whereas the source files allow developers to implement personalized AL methods.

Download the files provided at <http://sourceforge.net/projects/jclal/files> or <https://github.com/ogreyesp/JCLAL>.

- Download source files from SVN by SourceForge.

Anonymous SVN access is also available via SourceForge. To access the source code repository you could use one of the following ways:

- Browse source code online (<http://sourceforge.net/p/jclal/svn/HEAD/tree/>) to view the project's directory structure and files.
- Check out source code with a SVN client using the following command:

```
svn checkout svn://svn.code.sf.net/p/jclal/svn/ jclal-svn
```

- Download source files from GIT by SourceForge.

Anonymous GIT access is also available via SourceForge. To access the source code repository you could use one of the following ways:

- Browse source code online (<http://sourceforge.net/p/jclal/git/ci/master/tree/>) to view the project's directory structure and files.
- Check out source code with a GIT client using the following command:

```
git clone git://git.code.sf.net/p/jclal/git jclal-git
```

- Download source files from Mercurial by SourceForge.

Anonymous Mercurial access is also available via SourceForge. To access the source code repository you could use one of the following ways:

- Browse source code online (<http://sourceforge.net/p/jclal/hg/ci/default/tree/>) to view the project's directory structure and files.
- Check out source code with a Mercurial client using the following command:

```
hg clone http://hg.code.sf.net/p/jclal/hg jclal-hg
```

- Download source files from GitHub.

Anonymous GIT access is also available via GitHub. To access the source code repository you could use one of the following ways:

- Browse source code online (<https://github.com/ogreyesp/JCLAL>) to view the project's directory structure and files.
- Check out source code from the following url:

```
https://github.com/ogreyesp/JCLAL.git
```

- Download source files, jar file, API docs and POM file from Maven Central Repository.

JCLAL is also available via Maven Central Repository. Download JCLAL from the following urls:

```
http://search.maven.org/remotecontent?filepath=net/sf/jclal/jclal/1.1/  
http://mvnrepository.com/artifact/net.sf.jclal/jclal/1.1
```

- Download source files, jar file, API docs and POM file from Sonatype OSSRH repository.

JCLAL is also available via Sonatype OSSRH repository. Download JCLAL from the following url:

```
https://oss.sonatype.org/#nexus-search;quick~jclal
```

- Download source files and import as project in Eclipse IDE.

You can download JCLAL and import it as Eclipse project from SourceForge. After you have downloaded the JCLAL framework and unzip the folder into the location that you want:

- Steep one. Click on “File” → “Import” in the main menu of the Eclipse IDE.
- Steep two. Click on “Next” button and click on “Browse” button and select the folder where the JCLAL framework were unzipped.
- Steep three. In the area of “Projects” will appear the JCLAL project, click on the checkbox “jclal”. If you want to copy the project into your workspace click on the checkbox “Copy projects into workspace”.
- Steep four. Click on the “Finish” button. Finally, the JCLAL project and dependence libraries are imported.

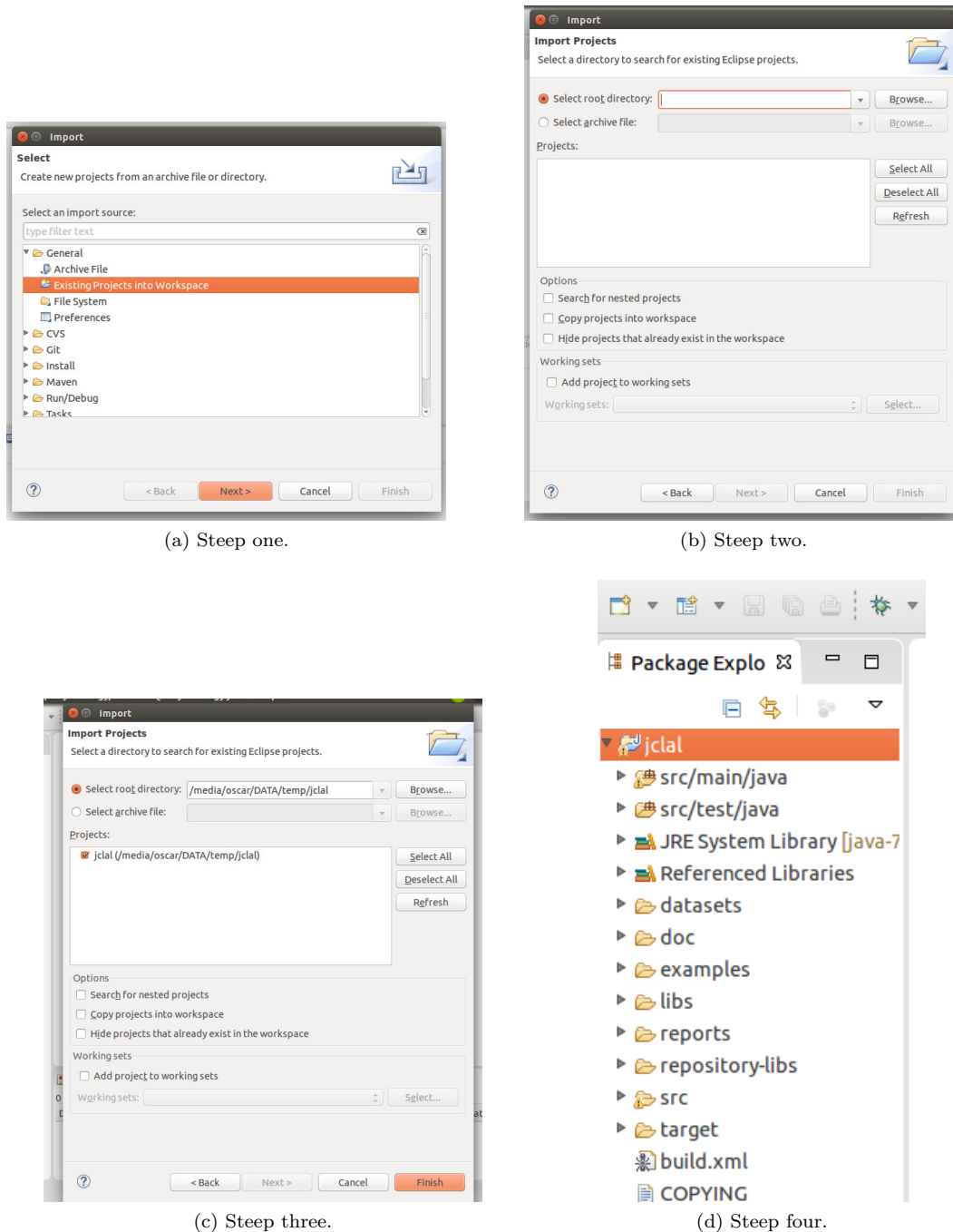


Figure 2.1: Importing JCLAL as Eclipse project.

- Download source files and import as project in NetBeans IDE.

You can also download JCLAL and import it as NetBeans project from SourceForge. After you have downloaded the JCLAL framework and copy the zip file into the location that you want:

- Steep one. Click on “File” → “Import Project” → “From Zip”.
- Steep two. Click on the “Browse” button and find the zip file of the JCLAL framework. Click

on “Import” button. Finally, the JCLAL project and dependence libraries are imported.

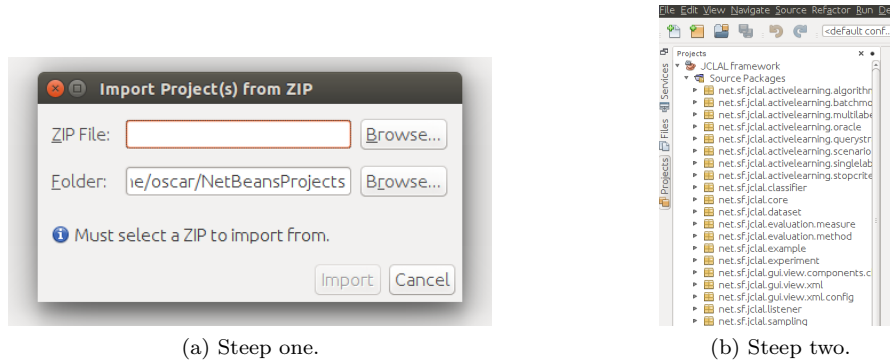


Figure 2.2: Importing JCLAL as NetBeans project.

For users that use a different IDE, such as IntelliJ IDEA, please consult the information about how to import a project in this IDE (<https://www.jetbrains.com/help/idea/2016.1>).

We also provide the JCLAL framework in form of packages for being installed in the most relevant operating systems.

Compilation

The source code can be compiled as follows:

1. Using the ant compiler and the `build.xml` file.
Open a terminal console located in the project path and type:
 - “`ant`” to build the jar file. A jar file named “`jclal-1.1.jar`” will be created.
 - “`ant clean`” for cleaning the project.
2. Using the maven compiler and the `pom.xml` file.
Open a terminal console located in the project path and type:
 - “`mvn initialize`” to initialize the project setup and repositories.
 - “`mvn install`” to build the jar file. A jar file named “`jclal-1.1.jar`” will be created.
 - “`mvn clean`” for cleaning the project.

Configuring an experiment

The library allows users to execute an experiment through an XML configuration file as well as directly from Java code. We recommend that the users run the experiments through an XML configuration file, owing to it is the easiest and fastest manner. A configuration file comprises a series of parameters required to run an algorithm. The most important parameters are described as follows:

- The evaluation method: this parameter establishes the evaluation method used in the learning process. In the JCLAL-1.1 version, the hold out, k -fold cross validation, leave one out cross validation, 5x2 fold cross validation and a method for an actual deployment are supported. A sample of the hold out evaluation method is shown:

```
<process evaluation-method-type="net.sf.jclal.evaluation.method.HoldOut">
```

- The dataset: the dataset to be used in the experiment can be declared in several ways. For example, in the case of using the HoldOut method only one file dataset may be declared and the evaluation method splits the dataset into the train and test set using a sampling method. On the other hand, the train and test sets could be passed directly as parameters. A sample of the case when only one file is passed is shown:

```
<file-dataset>datasets/iris/iris.arff</file-dataset>
<percentage-split>66</percentage-split>
```

- Labelled and unlabelled sets: labelled and unlabelled sets can be declared in several ways. The users can opt to extract the labelled and unlabelled sets from the training set, in this case a sampling method must be declared. On the other hand, the labelled and unlabelled sets could be passed directly as parameters. A sample of the case when the training set is divided into labelled and unlabelled sets is shown:

```
<rand-gen-factory seed="1299961164" type="net.sf.jclal.util.random.RanecuFactory"/>
<sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
  <percentage-to-select>10</percentage-to-select>
</sampling-method>
```

In this case, the 10% of the training set is randomly selected as the labelled set, the rest of the instances form the unlabelled set.

- AL algorithm: this parameter establishes the AL protocol used in the learning process. Currently, the classical AL algorithm is supported, i.e. in each iteration a query strategy selects the most informative instance (a batch of instances may be selected) from the unlabelled set. Afterwards, an oracle labels the selected instance, the instance is added to the labelled set and removed from the unlabelled set. A sample of the AL algorithm is shown:

```
<algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
```

- Stopping criteria: this parameter determines when the experiment must be stopped. The JCLAL framework provides several ways to define stopping criteria. Users can define new stopping criteria according with their needs. The maximum number of iterations must be declared to guarantee that the AL process finishes. A sample of 100 iteration is shown:

```
<stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
  <max-iteration>100</max-iteration>
</stop-criterion>
```

- AL scenario: this parameter establishes the scenario used in the AL process. Currently, the Pool-based sampling and Stream-based sampling scenarios are provided. A sample of a scenario is shown:

```
<scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
```

- Query strategy: this parameter establishes the query strategy used in the AL process. Currently, the most significant query strategies that have appeared in the single-label and multi-label settings are provided. The base classifier to use must also be defined. A sample of a single-label query strategy is shown:

```
<query-strategy type="net.sf.jclal.activelearning.singlelabel.querystrategy.
EntropySamplingQueryStrategy">
  <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
    <classifier type="weka.classifiers.bayes.NaiveBayes"/>
  </wrapper-classifier>
</query-strategy>
```

In this case, we use the Entropy Sampling strategy with Naive Bayes as a base classifier.

- The oracle: this parameter denotes the oracle used in the AL process. Two types of oracle are provided. A Simulated Oracle that reveals the hidden classes of a selected unlabelled instance, and a Console Human Oracle that iteratively asks users for the class of a selected unlabelled instance. The users can define new types of oracles according to their needs. A sample of the Simulated Oracle is shown:

```
<oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
```

- Listeners: a listener to display the results of the AL process can be defined. In the following example, a report is made for each iteration. Furthermore, the report is stored on a file named “Example”, and the results are also printed in the console. A window where the user can visualize the learning curve of the AL process is showed. The user can also select the evaluation measure to plot.

```
<listener type="net.sf.jclal.listener.GraphicalReporterListener">
  <report-frequency>1</report-frequency>
  <report-on-file>true</report-on-file>
  <report-on-console>true</report-on-console>
  <report-title>Example</report-title>
  <show-window>true</show-window>
</listener>
```

The library provides more XML tags to configure an experiment file. In the JCLAL API reference can be consulted the tags that are currently supported. In the “examples” folder, which is included into the downloaded file of the JCLAL framework, several examples of experiments are provided.

Executing an experiment

Once the user has created a XML configuration file which specifies the parameters and settings for the experiment, there are several ways to execute the experiment.

- Using the JAR file

You can execute any experiment using the JAR file. For instance, you can run the Entropy Sampling query strategy with the Hold Out evaluation method using the following command:

```
java -jar jclal-1.1.jar -cfg "examples/SingleLabel/HoldOut/EntropySamplingQueryStrategy.cfg"
```

In the “examples” folder, which is included into the downloaded file of JCLAL framework, several examples of experiments of single-label and multi-label query strategies are provided. The following command executes all the experiments contained into a folder.

```
java -jar jclal-1.1.jar -d "examples/SingleLabel/HoldOut"
```

In this case, all the experiments contained in the “examples/SingleLabel/HoldOut” folder will be conducted.

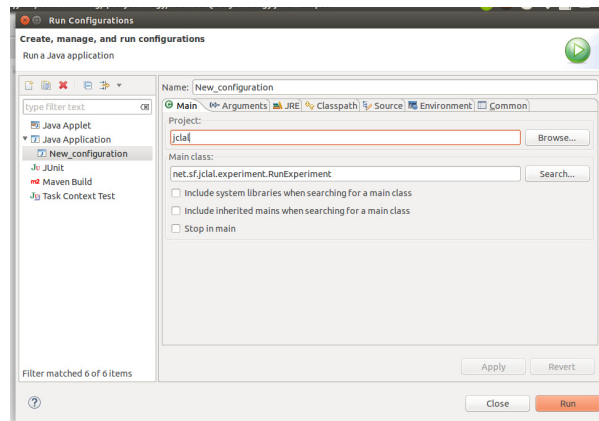


Figure 2.3: Executing an experiment with the Eclipse IDE.

- Using Eclipse IDE

Click on “Run” → “Run Configurations” will create a new launch configuration as Java Application.

```
Project jclal
Main class: net.sf.jclal.experiment.RunExperiment
Program arguments: -cfg "examples/SingleLabel/HoldOut/EntropySamplingQueryStrategy.cfg"
```

Finally, the experiment will be executed by clicking on the “Run” button. The user can use any of the example configuration files which are included in the library.

- Using NetBeans IDE

Click on “Project properties” → “Run”

Set the main class:

```
net.sf.jclal.experiment.RunExperiment
```

Set the program arguments:

```
-cfg "examples/SingleLabel/HoldOut/EntropySamplingQueryStrategy.cfg"
```

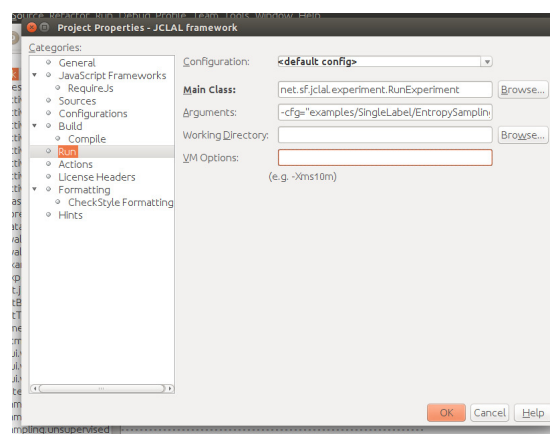


Figure 2.4: Executing an experiment with the NetBeans IDE.

Finally, the experiment will be executed by clicking on the “Run” button. For users that use a different IDE, such as IntelliJ IDEA, please consult the information about how to run a project in this IDE.

Study cases

Next, four examples of experiments are showed to illustrate how to use the library.

Margin sampling query strategy

The configuration file comprises a series of parameters required to run an algorithm. Below, a configuration file that implements the Margin Sampling query strategy is shown.

```
<experiment>
<process evaluation-method-type="net.sf.jclal.evaluation.method.kFoldCrossValidation">
  <rand-gen-factory seed="9871234" type="net.sf.jclal.util.random.RanecuFactory"/>
  <file-dataset>datasets/ecoli.arff</file-dataset>
  <stratify>true</stratify>
  <num-folds>10</num-folds>
  <sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
    <percentage-to-select>5.0</percentage-to-select>
  </sampling-method>
  <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
    <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
      <max-iteration>100</max-iteration>
    </stop-criterion>
    <listener type="net.sf.jclal.listener.GraphicalReporterListener">
      <report-title>margin</report-title>
      <report-frequency>1</report-frequency>
      <report-directory>reports</report-directory>
      <report-on-console>false</report-on-console>
      <report-on-file>true</report-on-file>
      <show-window>true</show-window>
    </listener>
    <scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
      <batch-mode type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
        <batch-size>1</batch-size>
      </batch-mode>
      <oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
      <query-strategy type="net.sf.jclal.activelearning.singlelabel.querystrategy.
MarginSamplingQueryStrategy">
        <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
          <classifier type="weka.classifiers.function.SMOsync"/>
        </wrapper-classifier>
      </query-strategy>
    </scenario>
  </algorithm>
</process>
</experiment>
```

In this case, a 10-fold cross validation evaluation method is used. In each fold, the 5% of the training set is selected to construct the labelled set and the rest of the instances form the unlabelled set. A Pool-based Sampling scenario with the Margin Sampling strategy is used. A Support Vector Machine is used as a base classifier.

After the experiment concluded, a folder containing the report for the experiment's output is created. A summary report which comprises information about the induced classifier and several performance measures is created. The report file also provides the runtime of the AL process.

The JCLAL provides a GUI utility (`net.sf.jclal.gui.view.components.chart.ExternalBasicChart`) to visualize the learning curves of the AL methods.

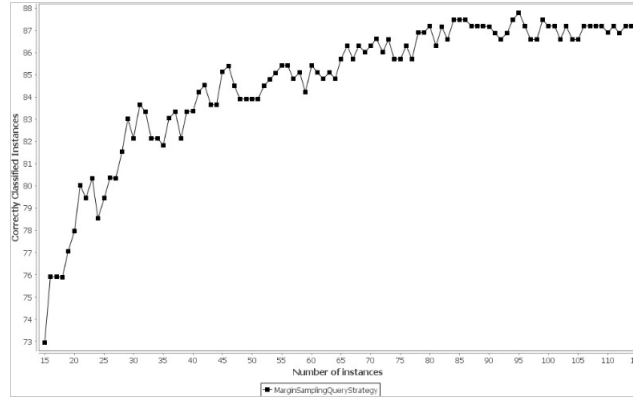


Figure 2.5: Results of the Margin Sampling strategy.

Entropy sampling query strategy

Below, a configuration file that implements the Entropy Sampling query strategy is shown.

```
<experiment>
<process evaluation-method-type="net.sf.jclal.evaluation.method.HoldOut">
  <rand-gen-factory seed="9871234" type="net.sf.jclal.util.random.RanecuFactory"/>
  <file-dataset>datasets/mfeat-pixel.arff</file-dataset>
  <percentage-split>66.0</percentage-split>
  <sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
    <percentage-to-select>5.0</percentage-to-select>
  </sampling-method>
  <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAAlgorithm">
    <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
      <max-iteration>100</max-iteration>
    </stop-criterion>
    <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.UnlabeledSetEmpty"/>
  </algorithm>
  <listener type="net.sf.jclal.listener.GraphicalReporterListener">
    <report-title>entropy</report-title>
    <report-frequency>1</report-frequency>
    <report-directory>reports</report-directory>
    <report-on-console>false</report-on-console>
    <report-on-file>true</report-on-file>
    <show-window>true</show-window>
  </listener>
  <scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
    <batch-mode type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
      <batch-size>1</batch-size>
    </batch-mode>
    <oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
    <query-strategy type="net.sf.jclal.activelearning.singlelabel.querystrategy.
EntropySamplingQueryStrategy">
      <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
        <classifier type="weka.classifiers.bayes.NaiveBayes"/>
      </wrapper-classifier>
    </query-strategy>
  </scenario>
</process>
</experiment>
```

In this case, a Hold Out evaluation method is used. The 66% of the dataset form the training set and the rest of the instances form the test set. The 5% of the training set is randomly selected to construct the labelled set and the rest of the instances form the unlabelled set. A Pool-based Sampling scenario with the Entropy Sampling strategy is used. The Naive Bayes algorithm is used as a base classifier. Two stop criteria are defined, the first one is related to the maximum number of iterations, and the second one stops the experiment in case that the unlabelled set is empty.

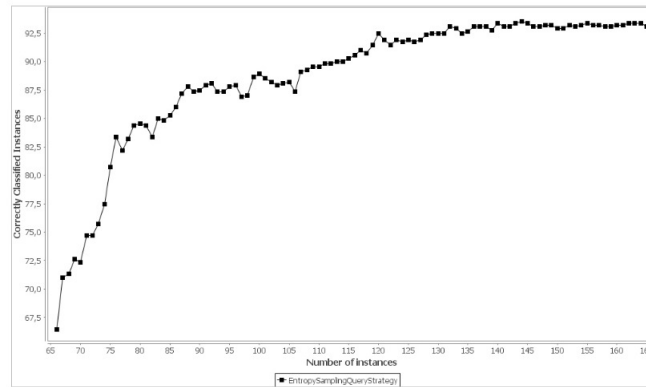


Figure 2.6: Results of the Entropy Sampling strategy.

Actual deployment example

We have included a real-usage scenario in the JCLAL-1.1 version, where the user provides a small labelled set from which the initial classifier is trained, and an unlabelled set for determining the unlabelled instances that will be query in each iteration. In each iteration, the selected unlabelled instances are showed to the user, the user labels the instances and they are added to the labelled set. This real-usage scenario allows to obtain the set of labelled instances at the end of the session for further analysis. Below, the configuration file is shown.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<experiment>
  <process evaluation-method-type="net.sf.jclal.evaluation.method.RealScenario">
    <file-labeled>datasets/abalone-labeled.arff</file-labeled>
    <file-unlabeled>datasets/abalone-unlabeled.arff</file-unlabeled>
    <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
      <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
        <max-iteration>10</max-iteration>
      </stop-criterion>
      <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.UnlabeledSetEmpty"/>
      <listener type="net.sf.jclal.listener.RealScenarioListener">
        <informative-instances>reports/real-scenario-informative-data.txt</informative-instances>
      </listener>
      <scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
        <batch-mode type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
          <batch-size>1</batch-size>
        </batch-mode>
        <oracle type="net.sf.jclal.activelearning.oracle.ConsoleHumanOracle"/>
        <query-strategy type="net.sf.jclal.activelearning.singlelabel.querystrategy.EntropySamplingQueryStrategy">
          <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
            <classifier type="weka.classifiers.bayes.NaiveBayes"/>
          </wrapper-classifier>
        </query-strategy>
      </scenario>
    </algorithm>
  </process>
</experiment>
```

In this case a “ConsoleHumanOracle” object is used, therefore JCLAL continuously asks users for the class of the selected unlabelled instance. Next, it is showed how JCLAL interacts with the user.

```
Human oracle.
What is the class of this instance?
Instance: 0.26,0.205,0.07,0.097,0.0415,0.019,0.0305,?

Index: Class name
```

```

-----
0: 1
1: 2
2: 3
3: 4
4: 5
5: 6
6: 7
7: 8
8: 9
9: 10
...
...
...
Type the index of the class or type -1 if you want to skip this instance
index >>

```

At the end of the experiment, the instances that were labelled are saved into the file “reports/real-scenario-informative-data.txt”.

Scalability of the JCLAL framework

Large-scale datasets push the scalability limits of class libraries. When designing the library, we try to follow a flexible and extensible object oriented design that does not damage the efficiency of the framework. Nevertheless, we are continuously optimizing the code.

In this JCLAL-1.1 version, we have worked to parallelize some operations, like the evaluation of the unlabelled instances, the testing process of the constructed model, the execution of experiments, etc. The parallelization of the training of models depends of the capability of the classifiers, that come from WEKA [11], MULAN [12] u other adopted library, of being parallelized.

On the other hand, we have supported the use of incremental learners to avoid the retraining of classifiers from scratch with all labelled instances. This supposes a considerable reduction of the time needed for retraining the classifiers. We have included the use of classifiers that come with the MOA framework (Massive Online Analysis) [13], which is a software environment for implementing algorithms for online learning.

Next, an example of an experiment to measure the scalability of JCLAL is showed. This experiment was conducted on p53 Mutant dataset by using Margin Sampling as strategy with Naive Bayes as a base classifier. The experiment was performed on Ubuntu 14.04 64-bit system with an Intel Core i7 2.67 GHz and 16 GB of RAM.

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<experiment>
  <process evaluation-method-type="net.sf.jclal.evaluation.method.kFoldCrossValidation">
    <rand-gen-factory seed="124321453" type="net.sf.jclal.util.random.RanecuFactory"/>
    <stratify>true</stratify>
    <num-folds>10</num-folds>
    <file-dataset>datasets/p53-mutants.arff</file-dataset>
    <sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
      <percentage-to-select>10</percentage-to-select>
    </sampling-method>
    <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAAlgorithm">
    <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
      <max-iteration>50</max-iteration>
    </stop-criterion>
    <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.UnlabeledSetEmpty"/>
    <listener type="net.sf.jclal.listener.ClassicalReporterListener">
      <report-title>margin-sampling-parallel</report-title>
      <report-frequency>1</report-frequency>
    </listener>
  </process>
</experiment>

```

```

    <report-directory>reports</report-directory>
    <report-on-console>false</report-on-console>
    <report-on-file>true</report-on-file>
  </listener>
  <scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
    <batch-mode type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
      <batch-size>15</batch-size>
    </batch-mode>
    <oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
    <query-strategy type="net.sf.jclal.activelearning.singlelabel.querystrategy.MarginSamplingQueryStrategy">
      <parallel>true</parallel>
      <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
        <parallel>true</parallel>
        <classifier type="weka.classifiers.bayes.NaiveBayes"/>
      </wrapper-classifier>
    </query-strategy>
  </scenario>
</algorithm>
</process>
</experiment>

```

In JCLAL 1.1, a new XML tag named “<parallel>” was included into the query strategy configuration. This XML tag indicates to the library that the scoring of unlabelled instances will be performed in parallel mode. The same XML tag can also be included into the base classifier configuration. In this case the testing process of the base classifier is performed in parallel mode. Up to now, the “<parallel>” tag is only available for single-label query strategies. We are working to parallelize more tasks into the library.

Suppose that the above configuration file is saved into a file named “Mutant-Margin-parallel.cfg” which is located in the “examples/SingleLabel” folder. For running the experiment with 4 CPUs just type:

```
java -jar jclal-1.1.jar -cores-per-processor 4 -cfg examples/SingleLabel/Mutant-Margin-parallel.cfg
```

Figure 2.8 shows the results of the time needed for selecting the unlabelled instances in each iteration. The figure shows that a good speed-up is obtained as the number of CPUs used is increased.

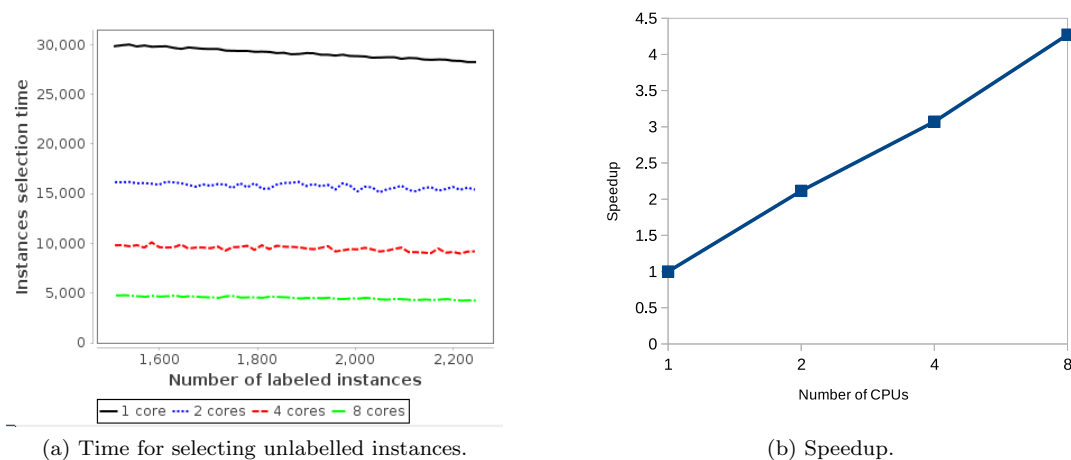


Figure 2.7: Experiment conducted in the p53 Mutant dataset.

Using JCLAL in WEKA's explorer

WEKA [11] has become one of the most popular data mining tool, and its success in researcher and education communities is due to its constant improvement, development, and portability. This tool, developed in the Java programming language, comprises a collection of algorithms for tackling several tasks as data pre-processing, classification, regression, clustering and association rules.

To use the library in WEKA, download the JCLAL-WEKA plug-in at <https://sourceforge.net/projects/jclal/files/jclal-1.0/jclal-weka-1.0.zip/download>. After downloading the zip file, for installing the JCLAL plug-in locate the folder where WEKA is installed and just type:

```
java -classpath weka.jar/ weka.core.WekaPackageManager -offline -install-package <PathToZip>
```

where PathToZip is the path of the JCLAL-WEKA plug-in. The plug-in only works with the JCLAL-1.0 version. We are working on a more user-friendly and advanced plug-in.

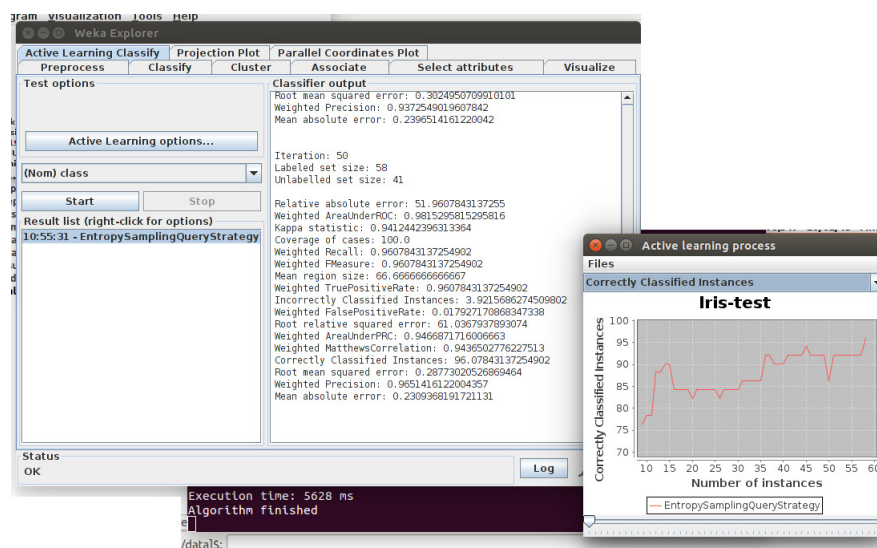


Figure 2.8: Plug-in for using the library into WEKA's explorer.

3. DEVELOPER DOCUMENTATION

This chapter is developer-oriented and describes the structure of the framework, the API reference, and the unit tests validation. Some examples that show how to extend the framework are provided.

Overview

JCLAL framework is an open source software and it is distributed under the GNU General Public License. It is constructed with a high-level software environment, with a strong object oriented design and use of design patterns, which allow to the developers reuse, modify and extend the framework. Currently, the library uses WEKA [11] and MULAN [12] libraries to implement the most significant query strategies that have appeared on single-label and multi-label learning paradigms. In next releases, we hope to include strategies related with multi-instance and multi-label-multi-instance learning paradigms. JCLAL also includes the MOA library [13] that allows to use incremental learning algorithms as base classifiers.

JCLAL uses the ARFF (Attribute-Relation File Format) data set format. An ARFF file is a text file that describes a list of instances sharing a set of attributes. The ARFF format was developed by the Machine Learning Project at the Department of Computer Science of The University of Waikato. The ARFF datasets are the format used by WEKA, MULAN and MOA libraries¹.

JCLAL includes the Jakarta Lucene² library which allows convert an index file of Lucene into a ARFF dataset. Thus, the researchers in domains as text classification could use the framework. Lucene is an open source project, implemented by Apache Software Foundation and distributed under the Apache Software License. JCLAL uses the JFreeChart³ library to visualize experiment results.

Two layers comprise the JCLAL architecture. The core system is the lower layer. It has the definition of interfaces, abstract classes, its base implementations and some software modules that provide all the functionality to the system. The experiments runner system is over the core layer, in which a job is an experiment defined by means of a configuration file.

Packages

The structure of the library is organized in packages. In this section we describe the main packages and the main classes, whereas the next section shows the API reference.

¹Information about how is formed an ARFF file can be consulted in WEKA webpage.

²<http://lucene.apache.org/>

³<http://www.jfree.org/jfreechart/>

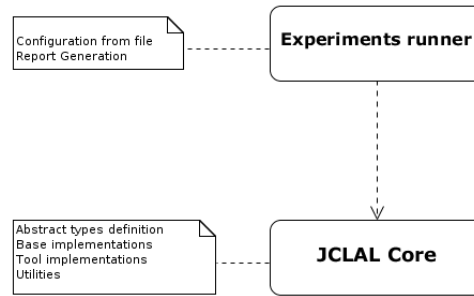


Figure 3.1: Logical layers of the JCLAL framework.

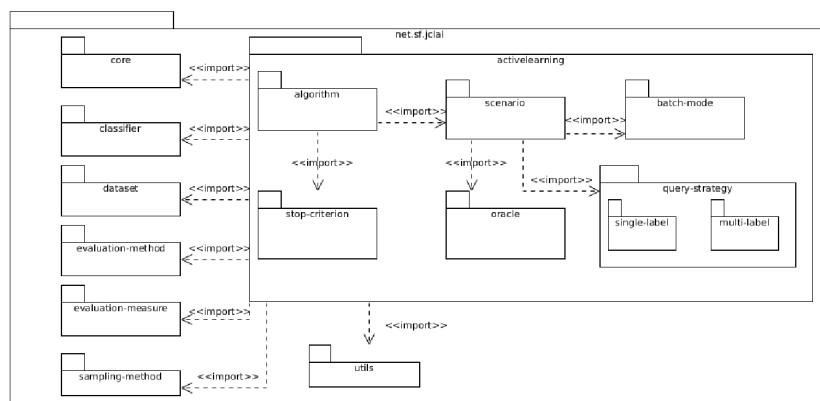


Figure 3.2: Package diagram of the JCLAL framework.

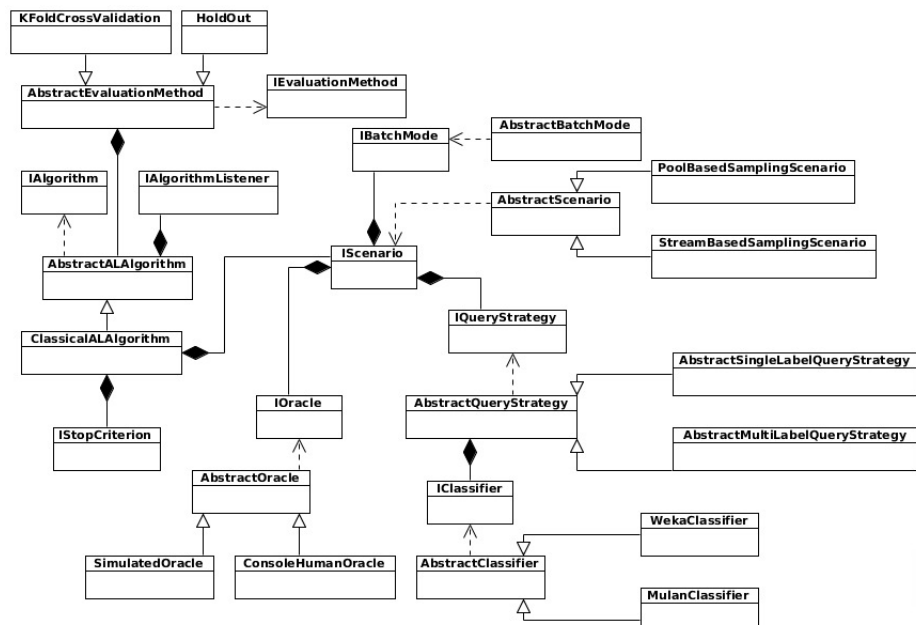


Figure 3.3: Class diagram of the main classes of JCLAL.

Package `net.sf.jclal.core`

This package comprises the basic interfaces.

- `IAlgorithm`: interface that provides the main steps and methods that must contain any AL algorithm. By implementing this interface is possible to extend the framework to different AL contexts.
- `IAlgorithmListener`: it takes charge of picking up all the events related to the algorithm execution (algorithm started, iteration completed and algorithm finished), in order to react depending on the event fired.
- `AlgorithmEvent`: it represents the events that happen during the algorithm execution.
- `IQueryStrategy`: interface that provides the methods that must be implemented by any query strategy.
- `IScenario`: interface that provides the methods that must be implemented by any AL scenario.
- `IBatchMode`: interface that provides the methods for performing batch-mode AL.
- `IClassifier`: interface that provides the methods that must be implemented by any wrapper classifier.
- `IConfigure`: interface that provides the methods that must be implemented by any class configurable from an XML file.
- `IDataset`: interface that represents a dataset.
- `IEvaluation`: interface that represents an evaluation metric.
- `IEvaluationMethod`: interface that represents an evaluation method.
- `IOracle`: interface that provides the methods that must be implemented by any oracle.
- `ISampling`: interface that represents a sampling method.
- `IStopCriterion`: interface for defining a stopping criterion.
- `ISystem`: interface for representing the system in an abstract way.
- `IRandGen`: interface that represents a random number generator method.
- `IRandGenFactory`: interface that represents a random generator factory.
- `ITool`: interface that represents a object that performs a task.
- `JCLAL`: Root for the JCLAL class hierarchy.

Package `net.sf.jclal.activelearning.algorithm`

This package contains the classes which allow us to redefine the behaviour of an AL algorithm and adapt the framework to an specific domain.

- `AbstractALAlgorithm`: abstract class which defines the basic methods that must be implemented by any AL algorithm.
- `ClassicalALAlgorithm`: class that inherits of `AbstractALAlgorithm` and represents the classical iterative AL algorithm.

Package `net.sf.jclal.querystrategy`

This package comprises the classes which define the query strategies.

- `AbstractQueryStrategy`: abstract class which defines the basic methods that must be implemented by any query strategy.

Package `net.sf.jclal.singlelabel.querystrategy`

This package comprises several state-of-the-art single-label query strategies.

- `AbstractSingleLabelQueryStrategy`: abstract class which defines the basic methods that must be implemented by any single-label query strategy. It inherits of the `AbstractQueryStrategy` class.
- `DensityDiversityQueryStrategy`: implementation of the Information Density framework.
- `UncertaintySamplingQueryStrategy`: abstract class which represents the query strategies that belong to Uncertainty Sampling category.
- `EntropySamplingQueryStrategy`: implementation of the Entropy Sampling query strategy.
- `MarginSamplingQueryStrategy`: implementation of the Margin Sampling query strategy.
- `LeastConfidentSamplingQueryStrategy`: implementation of the Least Confidence query strategy.
- `QueryByCommittee`: abstract class which represents the query strategies that belong to Query By Committee category.
- `KullbackLeiblerDivergenceQueryStrategy`: implementation of the Kullback Leibler Divergence query strategy.
- `VoteEntropyQueryStrategy`: implementation of the Vote Entropy query strategy.
- `ErrorReductionQueryStrategy`: abstract class which represents the query strategies that belong to Expected Error Reduction category.
- `ExpectedCeroOneLossQueryStrategy`: implementation of the Expected 0/1-Loss query strategy.
- `ExpectedLogLossQueryStrategy`: implementation of the Expected Log Loss query strategy.
- `VarianceReductionQueryStrategy`: implementation of the Variance Reduction query strategy.
- `RandomSamplingQueryStrategy`: implementation of the Random Sampling query strategy.

Package `net.sf.jclal.multilabel.querystrategy`

This package comprises several state-of-the-art multi-label query strategies.

- `AbstractMultiLabelQueryStrategy`: abstract class which defines the basic methods that must be implemented by any multi-label query strategy. It inherits of the `AbstractQueryStrategy` class.

- `MultiLabel3DimensionalQueryStrategy`: implementation of the Confidence-Minimum-NonWeighted and Confidence-Average-NonWeighted query strategies.
- `MultiLabelBinMinQueryStrategy`: implementation of the Binary Minimum query strategy.
- `MultiLabelMaxLossQueryStrategy`: implementation of the Max Loss query strategy.
- `MultiLabelMeanMaxLossQueryStrategy`: implementation of the Mean Max Loss query strategy.
- `MultiLabelMMCQueryStrategy`: implementation of the Maximum loss Reduction with Maximal Confidence query strategy.
- `MultiLabelDensityDiversityQueryStrategy`: implementation of the Information Density framework for multi-label learning.
- `MultiLabelMMUQueryStrategy`: implementation of the Max-Margin Uncertainty Sampling strategy.
- `MultiLabelLCIQueryStrategy`: implementation of the Label Cardinality Inconsistency strategy.

Package `net.sf.jclal.activelearning.scenario`

This package comprises the classes which represent the AL scenarios.

- `AbstractScenario`: abstract class which defines the basic methods that must be implemented by any AL scenario.
- `PoolBasedSamplingScenario`: implementation of the Pool-based Sampling scenario.
- `StreamBasedSelectiveSamplingScenario`: implementation of the Stream-based Selective Sampling scenario.

Package `net.sf.jclal.activelearning.batchmode`

This package includes the classes for performing batch-mode AL methods.

- `AbstractBatchMode`: abstract class which defines the basic methods that must be implemented by any batch-mode AL method.
- `QBestBatchMode`: implementation of the myopic “q-best instances” batch-mode approach.

Package `net.sf.jclal.classifier`

This package comprises the classes for the base classifiers.

- `AbstractClassifier`: abstract class which defines the basic methods that must be implemented by any base classifier.

- `WekaClassifier`: wrapper for using WEKA's classifiers. This class inherits of `AbstractClassifier`.
- `MulanClassifier`: wrapper for using MULAN's classifiers. This class inherits of `AbstractClassifier`.
- `MOAWrapper`: wrapper for using MOA's classifiers as WEKA's classifiers. This class inherits of `weka.classifiers.AbstractClassifier`.
- `MOAClassifier`: wrapper for using MOA's classifiers. This class inherits of `AbstractClassifier`.
- `ParallelBinaryRelevance`: implementation of the Binary Relevance approach in parallel mode.
- `BinaryRelevanceUpdateable`: implementation of the Binary Relevance to use incremental learners as binary classifiers.
- `WekaComitteClassifier`: class that represents a committee of WEKA's classifiers.
- `WekaClassifierThread`: this class executes a WEKA's classifier in a different thread.

Package `net.sf.jclal.experiment`

This package comprises the classes used for running an experiment.

- `RunExperiment`: class which executes an experiment via a configuration file. This class represents the access point of the library.
- `ExperimentBuilder`: class that interprets an XML configuration files.
- `Experiment`: class which represents an experiment.

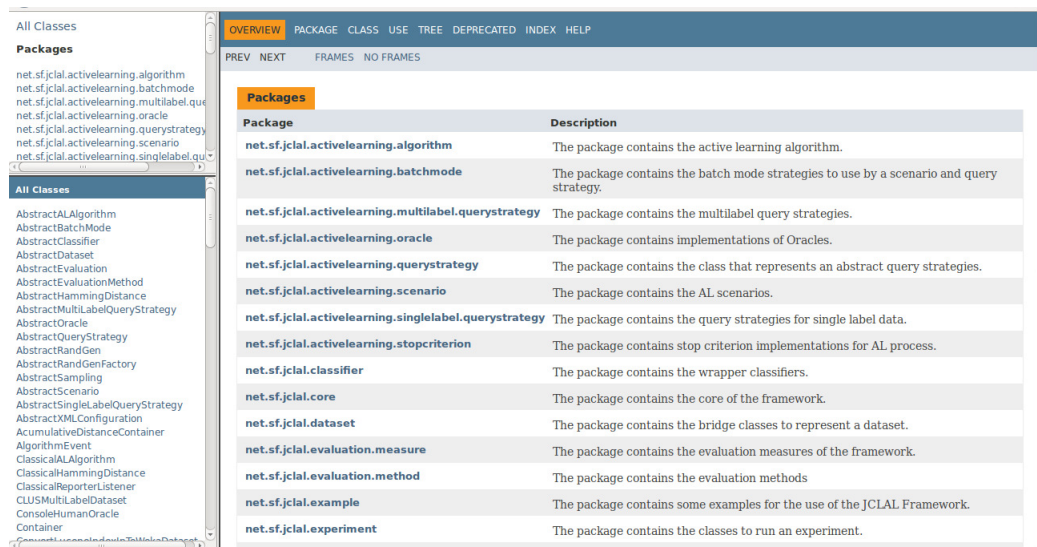
Package `net.sf.jclal.listener`

This package defines the listeners to obtain reports of the AL process.

- `ClassicalReporterListener`: class that reports the experiment results over a file and/or console.
- `GraphicalReporterListener`: class that inherits of `ClassicalReporterListener`. The experiment's results can be visualized by means of charts.
- `RealScenarioListener`: this class is a listener for a actual scenario. This class inherits of `ClassicalReporterListener`.

API reference

API documentation includes information about the code, the parameters and the content of the functions. For further information, please consult the API reference which is available in the downloaded file of the library.



| Package | Description |
|---|---|
| net.sf.jclal.activelearning.algorithm | The package contains the active learning algorithm. |
| net.sf.jclal.activelearning.batchmode | The package contains the batch mode strategies to use by a scenario and query strategy. |
| net.sf.jclal.activelearning.multilabel.querystrategy | The package contains the multilabel query strategies. |
| net.sf.jclal.activelearning.oracle | The package contains implementations of Oracles. |
| net.sf.jclal.activelearning.querystrategy | The package contains the class that represents an abstract query strategies. |
| net.sf.jclal.activelearning.scenario | The package contains the AL scenarios. |
| net.sf.jclal.activelearning.singlelabel.querystrategy | The package contains the query strategies for single label data. |
| net.sf.jclal.activelearning.stopcriterion | The package contains stop criterion implementations for AL process. |
| net.sf.jclal.classifier | The package contains the wrapper classifiers. |
| net.sf.jclal.core | The package contains the core of the framework. |
| net.sf.jclal.dataset | The package contains the bridge classes to represent a dataset. |
| net.sf.jclal.evaluation.measure | The package contains the evaluation measures of the framework. |
| net.sf.jclal.evaluation.method | The package contains the evaluation methods |
| net.sf.jclal.example | The package contains some examples for the use of the JCLAL Framework. |
| net.sf.jclal.experiment | The package contains the classes to run an experiment. |

Figure 3.4: API reference of the JCLAL framework.

Requirements and availability

The software is available under the GNU GPL license. The library requires Java 1.7, Apache commons logging 1.1, Apache commons collections 3.2, Apache commons configuration 1.5, Apache commons lang 2.4, Jakarta Lucene 2.4, JFreeChart 1.0, WEKA 3.7, MULAN 1.4, MOA 14.04 and JUnit 4.10 (for running tests). There is also a mailing list and a discussion forum for requesting support on using or extending JCLAL (<http://sourceforge.net/projects/jclal/>).

Implementing new features

In this section, several examples about how to implement new features in JCLAL are showed.

Implementing new query strategies

Next, two examples of implementing new query strategies are showed.

Implementing Margin Sampling strategy

Implementing new query strategies into JCLAL is an easy task. A developer only has to create one Java class using the class structure that the library provides.

Suppose that we want to implement the Margin Sampling query strategy. Margin Sampling queries the unlabelled instance which has the minimal difference among the first and second most probable class under the current model. Margin Sampling is based in the fact that unlabelled instances with large

margins means that the classifier has little doubt in differentiating between the two most likely class. On the other hand, instances with small margins are more ambiguous for the current classifier.^[1]

The Margin Sampling query strategy belongs to the Uncertainty Sampling category. Therefore, we define a class that extends of the abstract class `UncertaintySamplingQueryStrategy` for inheriting the methods that distinguish this type of query strategies. Below, the `MarginSamplingQueryStrategy` class is shown:

```
public class MarginSamplingQueryStrategy extends UncertaintySamplingQueryStrategy {

    public MarginSamplingQueryStrategy() {

        setMaximal(false);
    }

    @Override
    public double utilityInstance(Instance instance) {

        double[] probs = distributionForInstance(instance);

        //determine the class with the highest probability
        int ind1 = Utils.maxIndex(probs);
        double max1 = probs[ind1];
        probs[ind1] = 0;

        //determine the second class with the highest probability
        int ind2 = Utils.maxIndex(probs);
        double max2 = probs[ind2];

        return max1 - max2;
    }
}
```

In the class constructor we state that the query strategy is minimal by means of the `setMaximal` function, i.e. it selects the instance that has the minimal difference among the first and second most probable class under the current model.

```
public MarginSamplingQueryStrategy() {

    setMaximal(false);
}
```

Next, we must override the `utilityInstance(Instance instance)` function. This function receives as parameter an unlabelled instance and returns the utility (uncertainty) of the instance. The `distributionForInstance(Instance instance)` function returns the probabilities for each class according to the current model. Once the current model returns the probabilities for each class, the first and second class with the highest probabilities are determined and the difference between the probabilities is returned.

```
@Override
public double utilityInstance(Instance instance) {

    double[] probs = distributionForInstance(instance);

    //determine the class with the highest probability
    int indexMax = Utils.maxIndex(probs);
    double max1 = probs[indexMax];
    probs[indexMax] = 0;

    //determine the second class with the highest probability
    int indexMax2 = Utils.maxIndex(probs);
    double max2 = probs[indexMax2];
```

```

    return max1 - max2;
}

```

With the definition of `MarginSamplingQueryStrategy` class is enough to execute an experiment using this query strategy. For running an experiment with the Margin Sampling query strategy, we construct and save a configuration file with the following information:

```

<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<experiment>
  <process evaluation-method-type="net.sf.jclal.evaluation.method.HoldOut">
    <rand-gen-factory seed="1299961164" type="net.sf.jclal.util.random.RanecuFactory"/>
    <file-dataset>datasets/iris/iris.arff</file-dataset>
    <percentage-split>66</percentage-split>
    <sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
      <percentage-to-select>10</percentage-to-select>
    </sampling-method>
    <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
      <listener type="net.sf.jclal.listener.GraphicalReporterListener">
        <report-frequency>1</report-frequency>
        <report-on-file>true</report-on-file>
        <report-on-console>true</report-on-console>
        <report-title>Example-MarginSampling</report-title>
        <show-window>true</show-window>
      </listener>
      <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.MaxIteration">
        <max-iteration>45</max-iteration>
      </stop-criterion>
      <stop-criterion type="net.sf.jclal.activelearning.stopcriteria.UnlabeledSetEmpty"/>
      <scenario type="net.sf.jclal.activelearning.scenario.PoolBasedSamplingScenario">
        <batch-mode type="net.sf.jclal.activelearning.batchmode.QBestBatchMode">
          <batch-size>1</batch-size>
        </batch-mode>
        <query-strategy type="example.MarginSamplingQueryStrategy">
          <wrapper-classifier type="net.sf.jclal.classifier.WekaClassifier">
            <classifier type="weka.classifiers.bayes.NaiveBayes"/>
          </wrapper-classifier>
        </query-strategy>
        <oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
      </scenario>
    </algorithm>
  </process>
</experiment>

```

In this case, a Hold Out evaluation method is used. The Iris dataset is split into training (66%) and test (34%) sets. An unsupervised sampling method is used to extract the 10% of the training set as the labelled set and the rest of the instances form the unlabelled set. The classical AL algorithm is used and a listener is defined. The Pool-based scenario is used, only one unlabelled instance is selected in each iteration. The Margin Sampling query strategy uses as a base classifier the Naive Bayes implementation that comes with WEKA library. Finally, for running the experiment we can follow some of the ways described in the Section 2.4.

Implementing a “different” query strategy

Suppose that, we want to create a single-label query strategy that selects the unlabelled instance which is the most different with respect to labelled instances. A new class named `NearestInstance` is created.

```

public class NearestInstance extends AbstractSingleLabelQueryStrategy {
  ..
}

```

In the constructor of the class we specify that the query strategy is maximal, which means that it selects the unlabelled instance with the highest score.

```
public NearestInstance() {
    setMaximal(true);
}
```

We must also define the `utilityInstance(Instance instance)` method. We have used the cosine distance to compute the distance between the unlabelled instance that is being evaluated and the instances that belong to the labelled set.

```
@Override
public double utilityInstance(Instance instance) {

    CosineDistance cos = new CosineDistance(getLabelledData().getDataset());

    double total = 0;

    for (Instance current : getLabelledData().getDataset()) {
        total += cos.distance(instance, current);
    }

    return total/getLabelledData().getNumInstances();
}
```

After creating the class `NearestInstanceStrategy`, if we want to run this query strategy, we must create a XML configuration file such as we did in the past example.

Implementing a new evaluation method

Suppose that, we want to implement an evaluation method, where the user provides a small labelled set from which the initial classifier is trained and an unlabelled set. Optionally, we may specify a test set for testing the model. For the implementation of this method, we coded the class named `RealScenario` that inherits of `AbstractEvaluationMethod` class.

```
public class RealScenario extends AbstractEvaluationMethod {
    ...
}
```

We must define the method `evaluate()` for setting how the AL algorithm is evaluated. We load the data, pass the data loaded as arguments to the AL algorithm, and we finally execute the algorithm.

```
@Override
public void evaluate() {
    //Load data
    loadData();

    IAlgorithm algorithmCopy = getAlgorithm().makeCopy();

    //Set the labelled set
    algorithmCopy.setLabeledDataSet(getLabeledDataset());
    //Set the unlabelled set
    algorithmCopy.setUnlabeledDataSet(getUnlabeledDataset());

    //Set the test set
    if(getTestDataset()!=null)
```

```

algorithmCopy.setTestDataSet(getTestDataset());

// Executes the algorithm
algorithmCopy.execute();
}

```

If we want to create a XML configuration file for running an experiment with this new method, we have to specify the following XML tags in the evaluation method configuration.

```

...
<process evaluation-method-type="net.sf.jclal.evaluation.method.RealScenario">
  <file-labeled>datasets/abalone-labeled.arff</file-labeled>
  <file-unlabeled>datasets/abalone-unlabeled.arff</file-unlabeled>
...

```

Implementing a new oracle

Create a new oracle is an easy task. We only have to create a new class that inherits of **AbstractOracle** class. Suppose that, we want to ask users about the class of the selected unlabelled instances. The users labelled the instances and they are added to the labelled set. To do that, we create a new class named **ConsoleHumanOracle**.

```

public class ConsoleHumanOracle extends AbstractOracle {
...
}

```

We must define the `labelInstances(IQueryStrategy queryStrategy)` method. In this method, the selected unlabelled instances will be labelled by the users. Firstly, we get the indexes of the unlabelled instances that were selected by the query strategy. Secondly, the users are requested about the classes of the unlabelled instances, their responses are given by means of the console. Finally, the classes of the instances are saved.

```

@Override
public void labelInstances(IQueryStrategy queryStrategy) {
    // Object to read from the console
    Scanner scanner = new Scanner(new BufferedInputStream(System.in));

    ArrayList<Integer> selected = queryStrategy.getSelectedInstances();

    for (int i : selected) {
        Instance instance = queryStrategy.getUnlabelledData().instance(i);

        System.out.println("\nWhat is the class label of this instance?");

        System.out.println("Instance:" + instance.toString() + "\n");

        int classV = scanner.nextInt();

        instance.setClassValue(classV);
    }
}

```

For more details, you can consult the **ConsoleHumanOracle** class that comes implemented in the JCLAL framework.

Implementing a new stop criterion

Stop criteria are very important since they indicate when stopping the execution of an experiment. Create a new stop criterion is an easy task. You only have to create a new class that implements the `IStopCriterion` interface. Suppose that, we want to create a stop criterion related to the maximum number of iterations to perform. To do that, we create a new class named `MaxIteration` that implements the `IStopCriterion` interface.

```
public class MaxIteration implements IStopCriterion{

    /**
     * Max of iterations
     */
    private int maxIteration = 50;

    @Override
    public boolean stop(IAgorithm algorithm) {
        return ((ClassicalALAlgorithm) algorithm).getIteration() >= maxIteration;
    }
}
```

The `IStopCriterion` interface only has one method, `stop(IAgorithm algorithm)`. This method receives as parameter the algorithm executed, and returns whether the algorithm must be stopped or not.

If we want to set the maximum number of iterations by means of an XML configuration file, we must implement the `IConfigure` interface and override the `configure(Configuration settings)` method.

```
public class MaxIteration implements IStopCriterion, IConfigure{
    ...

    @Override
    public void configure(Configuration settings) {

        // Set max iteration
        int maxIterationT = settings.getInt("max-iteration", maxIteration);
        maxIteration= maxIterationT;

    }
    ...
}
```

Implementing a new listener

Listeners take charge of picking up all the events related to the algorithm execution (algorithm started, iteration completed and algorithm finished), in order to react depending on the event fired. Suppose that, we want to create a listener that saves into a file the last instances that were labelled by an oracle. We create a class named `LastLabelledInstanceListener` that inherits of `IAgorithmListener` interface.

```
public class LastLabelledInstanceListener implements IAgorithmListener{

    /**
     * Object for writing the file.
     */
    private BufferedWriter writer;

    ...
}
```

We must define the `algorithmStarted(AlgorithmEvent event)` method. In this method, we prepare the file where the instances will be saved.

```
@Override
public void algorithmStarted(AlgorithmEvent event) {
    File keep = new File("path-of-file");
    keep.createNewFile();
    writer = Files.newBufferedWriter(keep.toPath(),
        Charset.defaultCharset(), StandardOpenOption.CREATE, StandardOpenOption.APPEND);
}
```

We must define the `iterationCompleted(AlgorithmEvent event)` method. In this method, we write in the file the last instances that were labelled.

```
@Override
public void iterationCompleted(AlgorithmEvent event) {

    ClassicalALAlgorithm alg = (ClassicalALAlgorithm) event.getAlgorithm();

    ArrayList<String> lastInstances = ((AbstractOracle) alg.getScenario().getOracle()).getLastLabeledInstances();

    for (String last : lastInstances) {
        writer.append(last).append("\n");
    }
}
```

Finally, we must define the `algorithmFinished(AlgorithmEvent event)` method. In this method, we close the buffer.

```
@Override
public void algorithmFinished(AlgorithmEvent event) {
    writer.flush();
    writer.close();
}
```

Running unit tests

A unit test is a piece of code written by a developer that tests a specific functionality in the code. The JUnit framework was used for running unit tests. You can find the unit tests into the “test” folder of the JCLAL framework.

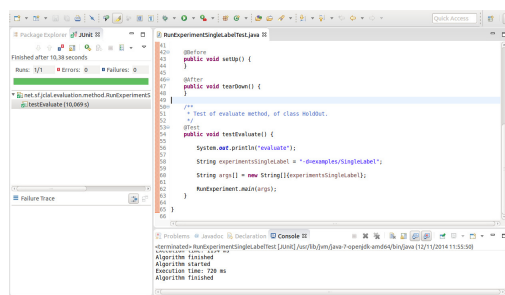


Figure 3.5: Example of a unit test for Single-label query strategies.

BIBLIOGRAPHY

- [1] B. Settles, *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool, 1st ed., 2012.
- [2] X. Zhu, “Semi-supervised learning literature survey,” Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison, 2005.
- [3] S. Ventura, C. Romero, A. Zafra, J. A. Delgado, and C. Hervás, “JCLEC: A Java Framework for Evolutionary Computation,” *Soft Computing*, vol. 12, pp. 381–392, 2007.
- [4] A. Cano, J. Luna, A. Zafra, and S. Ventura, “A Classification Module for Genetic Programming Algorithms in JCLEC,” *Journal of Machine Learning Research*, vol. 1, pp. 1–4, 2014.
- [5] S. Sonnenburg, M. L. Braun, C. S. Ong, S. Bengio, L. Bottou, G. Holmes, Y. LeCun, K.-R. Muller, F. Pereira, C. E. Rasmussen, G. Ratsch, B. Scholkopf, A. Smola, P. Vincent, J. Weston, and R. Williamson, “The need for open source software in machine learning,” *Journal of Machine Learning Research*, vol. 8, pp. 2443–2466, 2007.
- [6] K. Brinker, *From Data and Information Analysis to Knowledge Engineering*, ch. On Active Learning in Multi-label Classification, pp. 206–213. Springer, 2006.
- [7] X. Li, L. Wang, and E. Sung, “Multi-label SVM active learning for image classification,” in *International Conference on Image processing, ICIP’04*, vol. 4, pp. 2207–2210, IEEE, 2004.
- [8] B. Yang, J. Sun, T. Wang, and Z. Chen, “Effective Multi-Label Active Learning for Text Classification,” in *Proceedings of the 15th International Conference on Knowledge Discovery and Data Mining*, (Paris, France), pp. 917–926, ACM, 2009.
- [9] A. Esuli and F. Sebastiani, “Active Learning Strategies for Multi-Label Text Classification,” in *ECIR 2009, LNCS 5478* (M. B. et al., ed.), pp. 102–113, Springer-Verlag Berlin Heidelberg, 2009.
- [10] X. Li and Y. Guo, “Active Learning with Multi-Label SVM Classification,” in *Proceedings of the 23th International joint Conference on Artificial Intelligence*, pp. 1479–1485, AAAI Press, 2013.
- [11] M. Hall, E. Frank, G. Holmes, B. Pfahringer, P. Reutemann, and I. H. Witten, “The WEKA data mining software: An Update,” in *SIGKDD explorations*, vol. 11, pp. 10–18, 2009.
- [12] G. Tsoumakas, E. Spyromitros-Xioufis, J. Vilcek, and I. Vlahavas, “MULAN: A Java Library for Multi-Label Learning,” *Journal of Machine Learning Research*, vol. 12, pp. 2411–2414, 2011.
- [13] A. Bifet, G. Holmes, R. Kirkby, and B. Pfahringer, “MOA: Massive Online Analysis,” *Journal of Machine Learning Research*, vol. 11, pp. 1601–1604, 2010.