JCLAL: A Java Framework for Active Learning

Version 1.1

User Manual and Developer Documentation

Last update on March 2, 2016.

CONTENTS

1. INTRODUCTION

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 types of problems have motivated the development of new machine learning techniques. The Semisupervised Learning (SSL) [1] and Active Learning (AL) [2] are the two main areas of research that are concerned with the development of learning algorithms capable of learning a predictive model from labelled and unlabelled data at the same time. The AL has as main hypothesis that, if the learning algorithm can choose the data from which it learns, then it will perform better with a less cost [2].

To date, there are several frameworks or class libraries which assist the experimentation process and development of new algorithms in the data mining and machine learning areas, such as Rapid Miner[1], WEKA[2], Scikit-learn[3], Orange[4], KEEL[5], etc. However, the most of these tools are more focused on the Supervised Learning and Unsupervised Learning problems. Toolkits for the AL are still scarce.

We can find some libraries and independent codes in internet that implement AL methods (see Section 1.2), releasing to the researchers of implementing all methods that they need from scratch. However, to the best of our knowledge, up to date there is not a substantial effort oriented to the creation of a computational tool mainly focused on AL, e.g. a tool as important as WEKA is to the Supervised Learning community. In our consideration, a good computational tool is not only a tool which includes the most relevant AL strategies, but it should also be extensible, user-friendly, interoperable, portable, etc.

The above situation motivated the development of the JCLAL framework, a Java Class Library for Active Learning which includes the most significant state-of-the-art AL methods. JCLAL is user-friendly, and allows to the developers adapt, modify and extend the framework according to their needs. It includes the most relevant strategies that have been proposed in single-label and multi-label learning paradigms.

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

1.1 What is JCLAL?

JCLAL is inspired in the architecture of JCLEC [3, 4] which is a framework for evolutionary computation developed by KDIS research group of the University of Córdoba, Spain. Therefore, the library inherits the most of the advantages of the JCLEC framework.

The library provides a high-level software environment to do any kind of AL method. The architecture

[1] https://rapidminer.com/
[2] http://www.cs.waikato.ac.nz/ml/weka/
[3] http://scikit-learn.org
[4] http://orange.biolab.si/
[5] http://sci2s.ugr.es/keel/

follows strong principles of object oriented programming and design patterns, where abstractions are represented by loosely coupled objects and where it is common and easy to reuse code.

JCLAL provides an environment that its main features are:

- *Generic.* The library is able to execute any kind of AL method. Through a flexible classes hierarchy, the library provides the possibility of including new AL methods, as well as adapt, modify or extend the framework according to the developer's needs.

- *User-friendly.* JCLAL has several mechanisms that offer a user friendly programming interface. It allows to the users execute an experiment through a configuration file in XML format, as well as directly from a Java code. We recommend that the users run the experiments through a configuration file, owing to it is the easier, intuitive and faster manner.

- *Portable.* The library has been coded in the Java programming language that ensures its portability between all platforms that implement a Java Virtual Machine.

- *Elegant.* A coherent class hierarchy that follows good object oriented and generic programming principles was designed.

- *Interoperable.* The use of XML file format provides a common ground for tools development and to integrate the framework with other systems.

- *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.

JCLAL offers several state-of-the-art active learning strategies for single-label and multi-label learning paradigms. For next releases, we hope to provide strategies related with multi-instance and multi-label-multi-instance learning paradigms.

Currently, the following single-label AL strategies that are provided [2]: Entropy Sampling, Least Confident and Margin Sampling which belong to the Uncertainty Sampling category. The Vote Entropy and Kullback Leibler Divergence strategies which belong to Query By Committee category. In the Expected Error Reduction category, the Expected 0/1-loss and Expected Log-Loss strategies are included. One AL strategy belongs to Variance Reduction family. Additionally, the Information Density framework is also provided. More information about all these single-label AL strategies can be found in [2].

On the other hand, the following multi-label AL strategies are provided: Binary Minimum [5], Max Loss [6], Mean Max Loss [6], Maximal Loss Reduction with Maximal Confidence [7], Confidence-Minimun-NonWeighted [8], Confidence-Average-NonWeighted [8], Max-Margin Prediction Uncertainty [9] and Label Cardinality Inconsistency [9].

JCLAL implements two AL scenarios, Stream-Based Selective Sampling and Pool-Based Sampling. The library provides the interfaces and abstract classes for implementing batch-mode AL methods and other types of oracles. Furthermore, it has a simple manner of defining new stopping criteria which may vary with respect to the problem. It contains a set of utilities, e.g. algorithms for random number generations, sort algorithms, sampling methods, methods to compute the Area Under the Learning Curve, non-parametric statistical tests, etc.

1.2 Brief comparison with other AL libraries

These are the most relevant libraries that include AL methods that we can find in Internet. Apologies if any other existing library has not been included in the list.

Vowpal Wabbit (VW)

- *Description.* VW library is about of building fast learning algorithms, and it's 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 in AL, it allows to build the models by interacting with teachers.
- *User friendly.* VW is easily usable. VW includes some features that allows actual user experiments, e.g. building a classifier with the interaction of users by using teachers. VW is well documented, examples and tutorials are provided.
- *Extensibility.* The VW's code is easily extensible.
- *Programming Language.* VW has been implemented in C++, and it also provides some interfaces for working from C#, Pyhton, 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.
- *Scalability.* It can be effectively applied on learning problems with high dimensionality. The size of the set of features is bounded independent of the amount of training data using the hashing trick.
- *Active learning methods.* The IWAL (Importance Weighted Active Learning) strategy is implemented. VW library supports stream-based AL.

DUALIST

- *Description.* DUALIST is a practical tool to expedite annotation/learning in NLP tasks. It is a utility software for AL with instances and semantic terms. It uses some methods from AL and SSL areas to build text-based classifiers at interactive speed.
- *User friendly.* This includes a Web-based GUI that enables actual user experiments. DUALIST is poorly documented. However, a demo video is provided by the author.
- *Extensibility*: The hierarchy of classes is extensible.
- *Programming Language.* DUALIST requires Java and Python to work properly. It ships with most of the dependencies it needs to work, the only exception being the Play! Web framework for running the Web-based GUI.
- *Portability.* All platforms that supports the JVM and Python.
- *License.* This software is released under the Apache License v2.0.

- *Active learning methods.* DUALIST implements the most known query strategies based in uncertainty sampling, such as least confident, margin sampling and entropy sampling. It supports stream-based and pool-based AL.
- *Scalability.* DUALIST is written to run on a single computer and loads all data into memory. It is robust for hundreds of thousands of instances and features on modern hardware, but may be difficult to use beyond that (exposed by the DUALIST 's author).

active-learning-scala

- *Description.* This is an Active Learning 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 and CLUS frameworks.
- *User friendly.* It is poorly documented, not examples and tutorial about the use of active-learning-scala library are provided.
- *Extensibility.* The library is designed for an easy extension of active learning strategies.
- *Programming Language.* active-learning-scala has been implemented in Scala programming language.
- *Portability.* All platforms that supports the JVM.
- *License.* This software is released under the GNU General Public License.
- *Active learning methods.* active-learning-scala includes a large list of active learning 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 active learning and stream-based active learning.
- *Scalability.* Not evidences are given by the authors about the scalability of this library. However, the Scala languages has some features that make it scalable and the active-learning-scala library could profit of this Scala language feature.

TexNLP

- *Description.* TexNLP (Texas Natural Language Processing tools) library implements SSL algorithms combined with AL methods. TexNLP supports supervised and semi-supervised learning of Hidden Markov Models for tagging, and standard supervised Maximum Entropy Markov Models.
- *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, not examples and tutorials about the usage of the software are provided.
- *Extensibility.* The library is designed for an easy extension.
- *Programming Language.* TexNLP has been implemented in Java language. It also provides some interfaces for working with the Python language.
- *Portability.* All platforms that support the JVM.
- *License.* This software is released under the GNU General Public License.

- *Active learning 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.* Not evidences are given by the authors about the scalability of this software.

Active Semi-Supervised Learning

- *Description.* In fact it is not even a software, but merely a research code. It implements an active learning strategy on top of SSL and 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.
- *Active learning methods.* A version of Expected Error Reduction strategy is implemented. Pool-based active learning is supported.
- *Scalability.* Not scalable.

LibAct

- *Description.* To the best of our knowledge, LibAct is the most recent contribution software for active learning. The package not only implements several popular AL strategies, but also features the AL by learning meta-strategy that allows the machine 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. Very few examples and not tutorial of the usage of the software are provided.
- *Extensibility.* The library is designed for an easy extension of AL strategies, models and labellers.
- *Programming Language.* LibAct has been implemented in Pyhton language.
- *Portability.* All platforms that support Python.
- *License.* This software is released under a modified BSD license.
- *Active learning methods.* LibAct implements several query strategies, such as uncertainty sampling (least confident, margin sampling and entropy sampling), variance reduction, QUIRE strategy (Active Learning by QUerying Informative and Representative Examples), Query by committee, HintSVM (Hinted Support Vector Machine) and ALBL (Active learning by learning). It supports pool-based AL.
- *Scalability.* Not evidences are given by the authors about the scalability of this software.

As we can see, there are several libraries available in Internet which include AL methods. A very small number of libraries are mainly focused on active learning, they have been designed for working and researching on AL area, e.g. the active-learning-scala and LibAct libraries. Most libraries discussed above 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 on active learning and which are compatible with the 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 support to classifiers that come from the popular 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.

1.3 What to expect in future releases of JCLAL?

We are studying to include in JCLAL 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 supports lazy operations, process data in parallel with no additional work from the developers, pulls elements from a data source on-demand and passes them to a pipeline of operations for 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. By using Spark allow us to support fast learning algorithms that comes from 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. For next releases, we hope to include strategies related with multi-instance and multi-label-multi-instance learning paradigms.

2. DOCUMENTATION FOR USERS

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

2.1 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 to the user the fastest and easiest way of executing 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/archive/master.zip`.

- Download source files from SVN by SourceForge.

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

  - Browse source code online (`http://sourceforge.net/p/jclal/svn/HEAD/tree/`) to view this 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 can use one of the following ways:

  - Browse source code online (`http://sourceforge.net/p/jclal/git/ci/master/tree/`) to view this 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 can use one of the following ways:

  - Browse source code online (`http://sourceforge.net/p/jclal/hg/ci/default/tree/`) to view this 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 can use one of the following ways:

  - Browse source code online (`https://github.com/ogreyesp/JCLAL`) to view this 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.

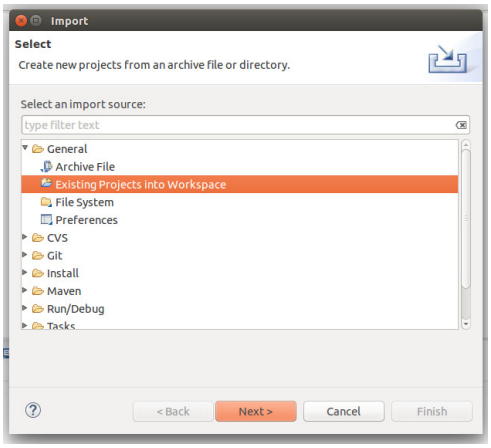
  JCLAL is also available via Sonatype OSSRH repository. Download JCLAL from the following url:

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

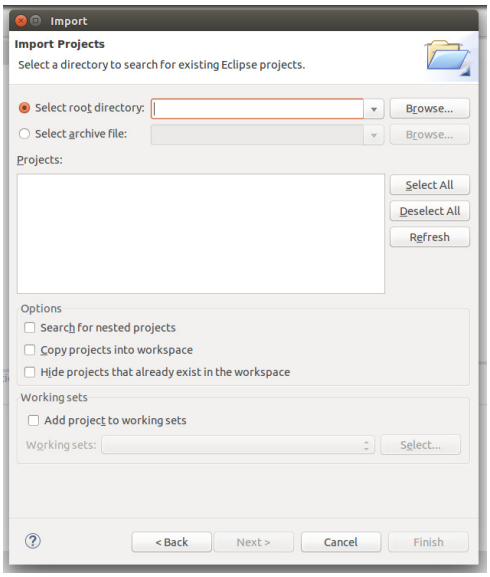
- Download source files and import as project in Eclipse IDE.

  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:

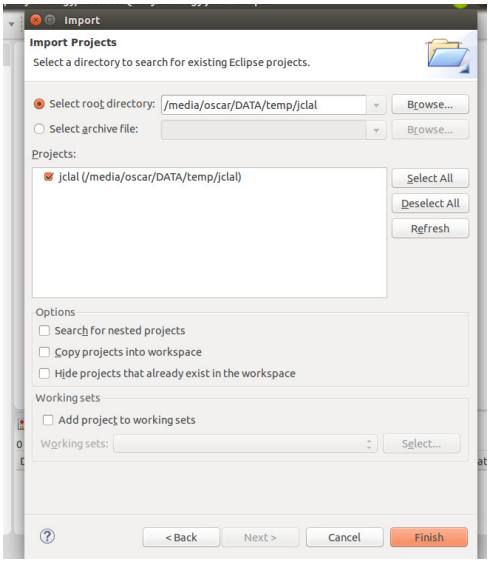
  - Steep one. Click on File → Import in the main menu of the Eclipse IDE.
  - Steep two. Click on Next button and click on Browse button and find the folder where the JCLAL framework were unzipped.
  - Steep three. In the area of "Projects" will appear the JCLAL project, click on the checkbox "jclal", 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 its libraries are imported.



(a) Steep one.



(b) Steep two.



(c) Steep three.

(d) Steep four.

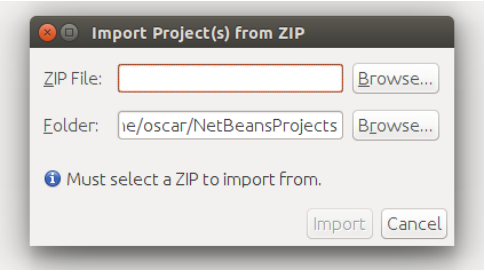
Figure 2.1: Importing JCLAL as Eclipse project.

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

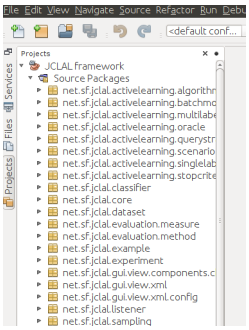
  You can also download JCLAL and import it as NetBeans project from SourceForge. After you have downloaded the JCLAL framework and copy the zip file into the location that you want:

  – 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 its libraries are imported.



(a) Steep one.



(b) Steep two.

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.

We also provide the JCLAL framework in form of packages for being installed in the most relevant operating systems. A `.deb` package is provided for DEB-based systems like Debian, Ubuntu and LinuxMint. A `.rpm` package is provided for RPM-based systems like RedHat, Fedora and OpenSuse. A `.osxpkg` package is provided for MacOS X. A Windows installer is also provided.

2.2 Compilation

The source code can be compiled in two ways:

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.

2.3 Configuring an experiment

The JCLAL framework 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 a XML configuration file, owing to it is the easier, intuitive and faster manner.

A configuration file comprises a series of parameters required to run an experiment. 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 use in the experiment can be declared in several ways. For example, in the case of using the HoulOut method only one file dataset could 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>
```

- The labelled and unlabelled sets: The 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, for doing it 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.

- The active learning algorithm: this parameter establishes the AL protocol used in the learning process. Up to date, the classical AL algorithm is supported, i.e. in each iteration a query strategy selects the most informative instance (a batch of instances can be selected) from the unlabelled set. Afterwards, an oracle labels the selected instance and 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. The users can define new stopping criteria according with their needs. The maximum number of iterations must be declared at least in order 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>
```

- Active learning scenario: this parameter establishes the scenario used in the AL process. Up to date, the Pool-based sampling and Stream-based sampling scenaries 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. Up to date, the most significant query strategies that have appeared in the single-label and multi-label settings are provided. The base classifier to use must be also 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 base classifier.

- The oracle: this parameter establishes 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 should be defined. In the following example, a report is made every one 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 of single-label and multi-label query strategies are provided.

2.4 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 could run the Entropy Sampling query strategy by using 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. If you want to execute all experiments contained into a folder use the following command:

```
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 performed.

- Using Eclipse IDE

  Click on Run → Run Configurations. Create a new launch configuration as Java Application.

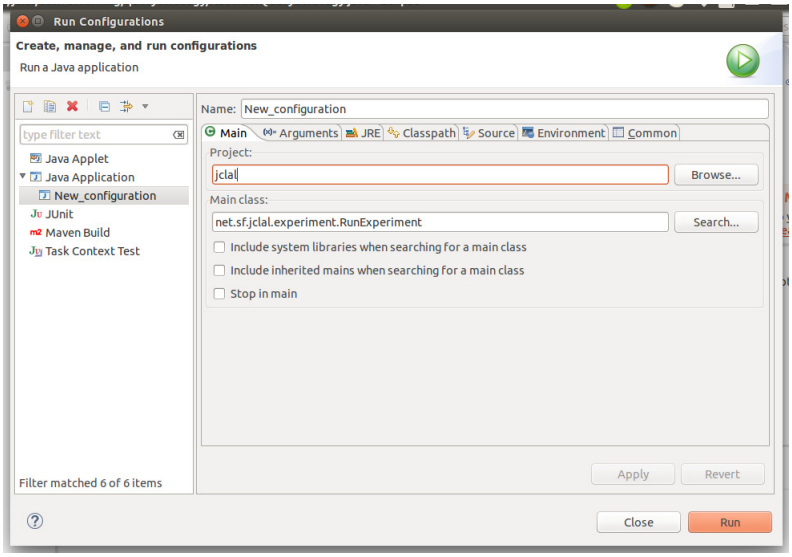


Figure 2.3: Executing an experiment with Eclipse IDE.

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

Finally, we execute the experiment by clicking on the "Run" button. The user could 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"
```

  Finally, we execute the AL method 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.

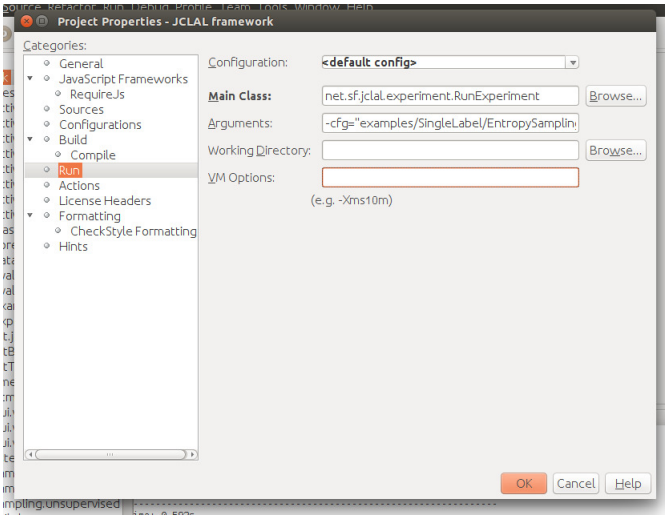


Figure 2.4: Executing an experiment with NetBeans IDE

2.5 Study cases

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

2.5.1 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 base classifier.

After the experiment is run, a folder containing the report for the experiment's output is created. A summary report which comprises information about the classifier inducted 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 method. The learning curve obtained of running the experiment is shown:

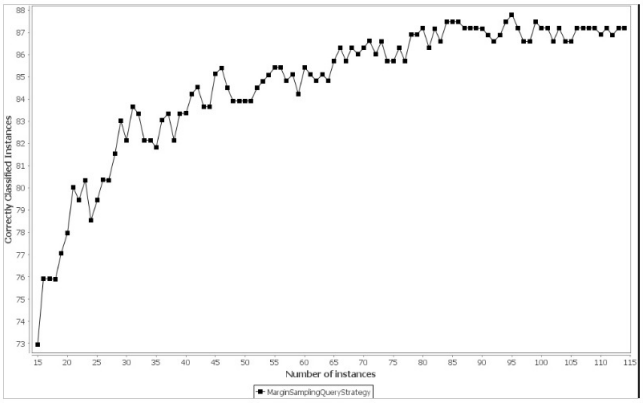


Figure 2.5: Results of Margin Sampling strategy.

2.5.2 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.ClassicalALAlgorithm">
   <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"/>
   <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>
  </algorithm>
 </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 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 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 shows the learning curve obtained of running the experiment.

2.5.3 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 unlabelled instances selected 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">
```

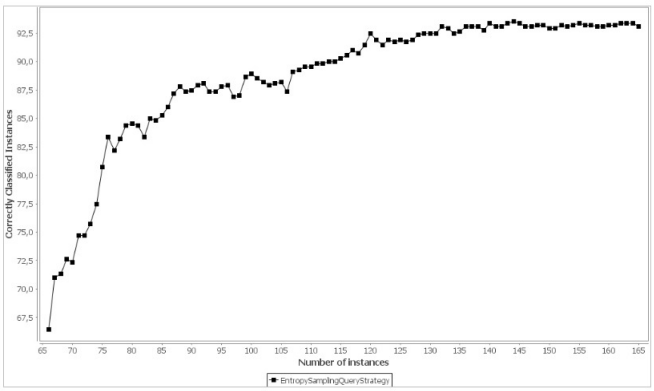


Figure 2.6: Results of Entropy Sampling strategy.

```
    <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".

2.5.4 Scalability of the JCLAL framework

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

In this JCLAL-1.1 version, we have worked to parallelize some operations, e.g. 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, MULAN 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 supported the use of classifiers that come from MOA framework (Massive Online Analysis), 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 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.

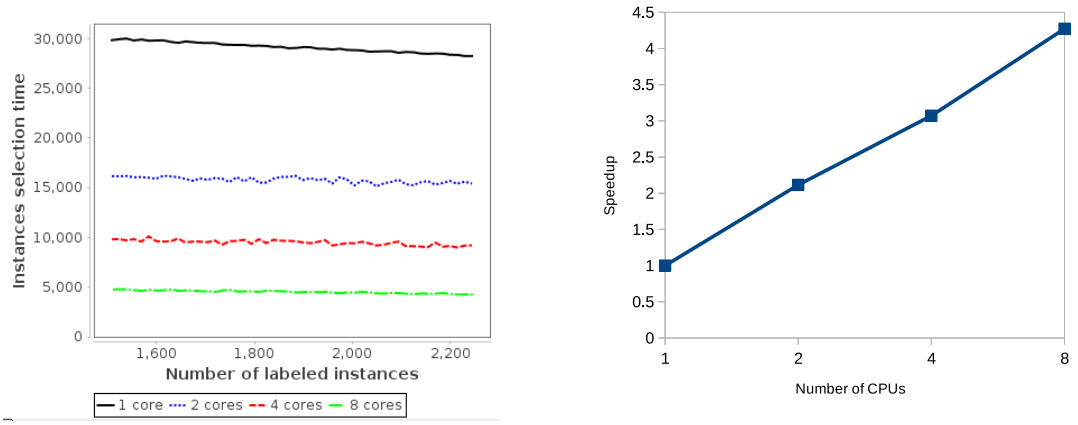
```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
 <experiment>
   <process evaluation-method-type="net.sf.jclal.evaluation.method.kFoldCrossValidation">
     <rand-gen-factory seed="124321453" type="net.sf.jclal.util.random.RanecuFactory"/>
     <stratify>true</stratify>
     <num-folds>10</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.ClassicalALAlgorithm">
     <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>
        <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>
```

See that a new XML tag "<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 for parallelizing 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 2.7 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.



(a) Time for selecting unlabeled instances.

(b) Speedup.

Figure 2.7: Experiment conducted in the p53 Mutant dataset.

2.6 Using JCLAL in WEKA's explorer

WEKA [10] has become one of the most popular data mining workbench 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 DM tasks as data pre-processing, classification, regression, clustering and association rules.

To use the library in WEKA, download the JCLAL-WEKA plug-in from `http://sourceforge.net/projects/jclal/files/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.

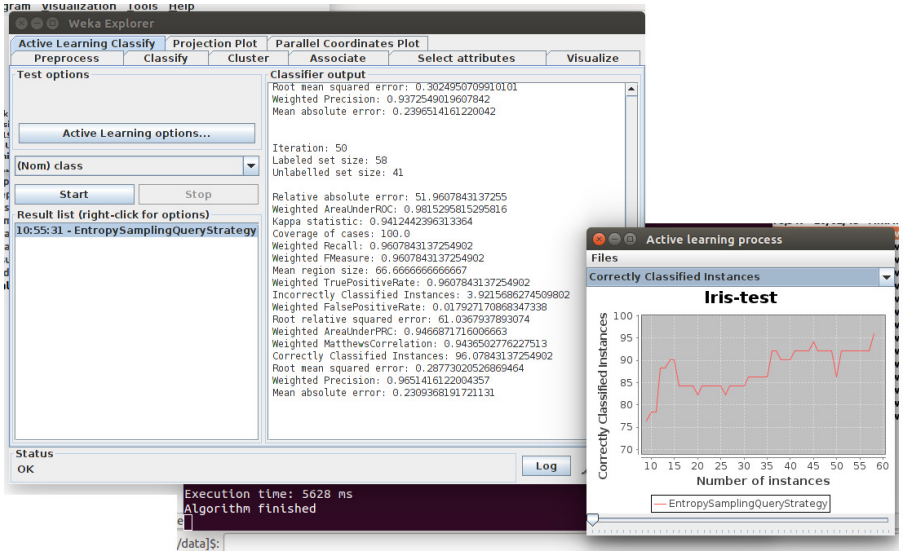


Figure 2.8: Plug-in for using the library into WEKA's explorer.

3. DOCUMENTATION FOR DEVELOPERS

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.

3.1 Overview

JCLAL framework is 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. Up to date, the library uses WEKA [10] and MULAN [11] libraries to implement the most significant query strategies that have appeared on single-label and multi-label learning paradigms. For next releases, we hope to include strategies related with multi-instance and multi-label-multi-instance learning paradigms.

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 the WEKA and MULAN libraries [1].

JCLAL includes the Jakarta Lucene[2] library which allows convert an index file of Lucene into a ARFF dataset. Thus, the researchers in domains as text classification can use the framework. Lucene is an open source project, implemented by Apache Software Foundation and distributed under the Apache Software License. JCLAL uses the JFreeChart[3] library to visualize the experiment's results. JFreeChart is the most widely used chart library for Java and it is free.

Two layers comprise the JCLAL architecture represented in following figure:

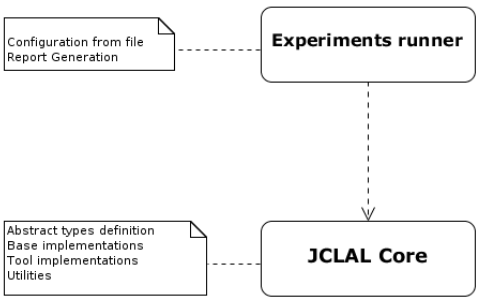


Figure 3.1: Logical layers of the JCLAL framework.

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. Over the

[1] Information about how is formed an ARFF file can be consulted in WEKA webpage
[2] http://lucene.apache.org/
[3] http://www.jfree.org/jfreechart/

core layer it is the experiments runner system in which a job is a experiment of AL defined by means of a configuration file.

3.2 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 presents the API reference. The following figure shows the diagram of the main packages.

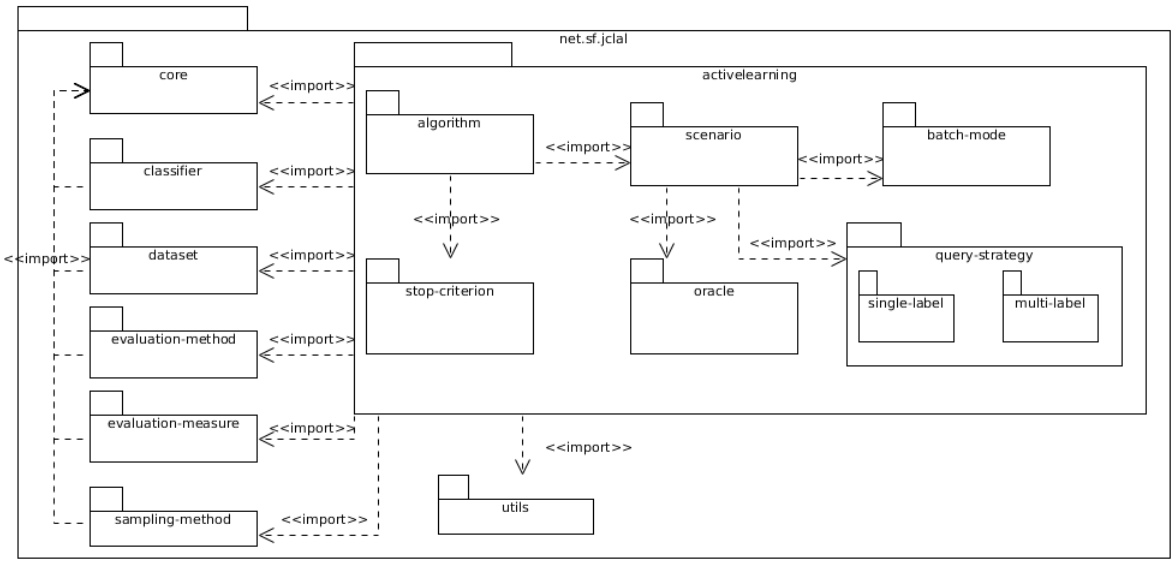


Figure 3.2: Package diagram of the JCLAL framework.

The following figure shows the diagram of the main classes of the JCLAL framework.

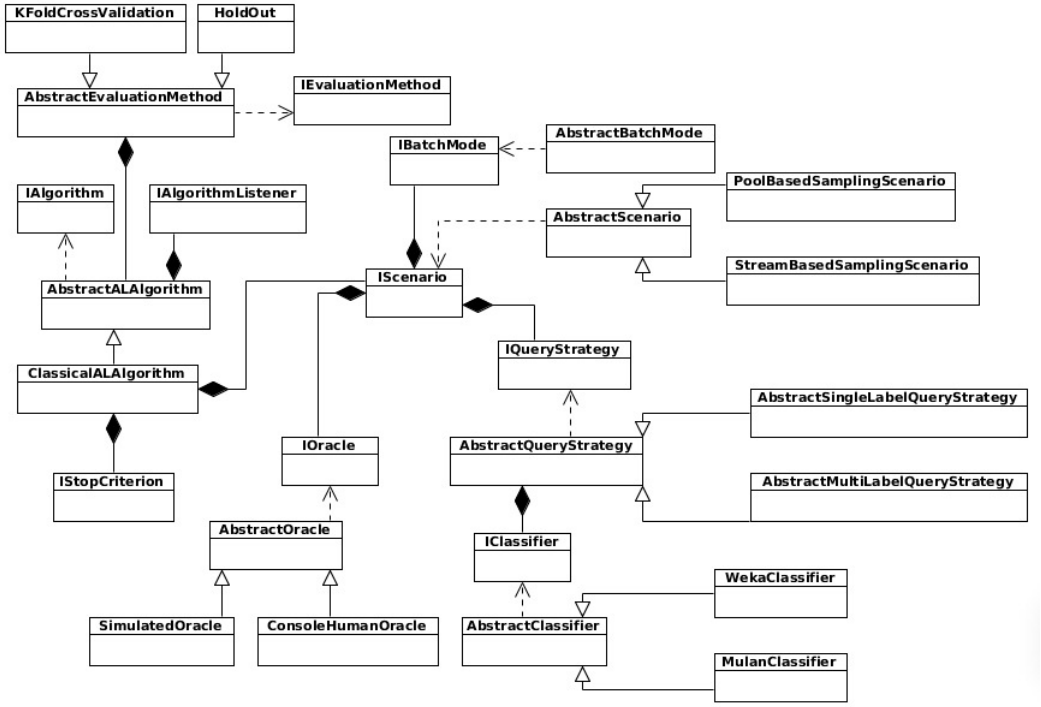


Figure 3.3: Class diagram of the main classes of JCLAL.

3.2.1 Package net.sf.jclal.core

This package contains the basic JCLAL 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 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 active learning.
- 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.

3.2.2 Package net.sf.jclal.activelearning.algorithm

This package contains the classes which allow to redefine the behaviour of an AL algorithm and adapt the framework to a concrete 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.

3.2.3 Package net.sf.jclal.querystrategy

This package contains the classes which define the query strategies.

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

3.2.4 Package net.sf.jclal.singlelabel.querystrategy

This package contains 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.

3.2.5 Package net.sf.jclal.multilabel.querystrategy

This package contains 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.

3.2.6 Package net.sf.jclal.activelearning.scenario

This package contains the classes which define 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.

3.2.7 Package net.sf.jclal.activelearning.batchmode

This package contains 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.

3.2.8 Package net.sf.jclal.classifier

This package contains 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 for using 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 thread.

3.2.9 Package net.sf.jclal.experiment

This package contains the classes used for running an experiment.

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

3.2.10 Package net.sf.jclal.listener

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

- ClassicalReporterListener: class that reports over a file and/or console the experiment's results.
- GraphicalReporterListener: class that inherits of ClassicalReporterListener and in addition 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 Classical-ReporterListener.

3.3 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.

3.4 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 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/).

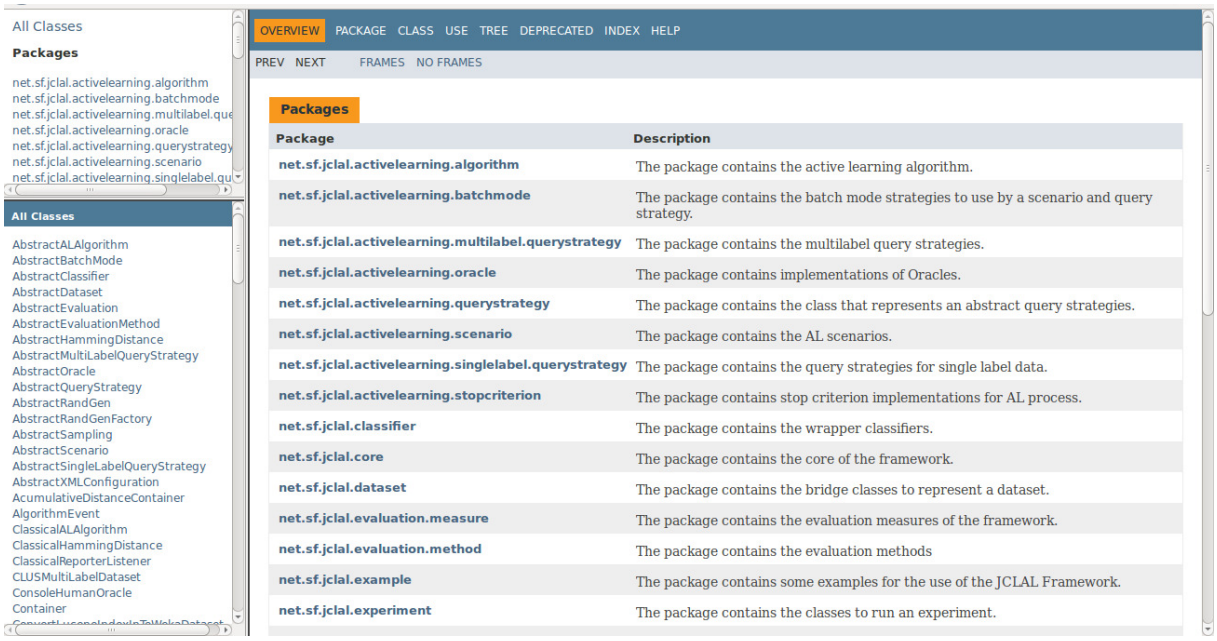


Figure 3.4: API reference of the JCLAL framework.

3.5 Implementing new features

3.5.1 Implementing new query strategies

Implementing Margin Sampling strategy

Implementing new and personalized query strategies into JCLAL is an easy task. A developer only has to create one Java class using the 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.[2]

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 `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[ind1];
        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 for JCLAL can execute an experiment using this query strategy. Note that, the scenario, oracle and AL algorithm are not necessary to implement.

For running an experiment with the Margin Sampling query strategy, we construct and store 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 set (34%). 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 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 as the most informative instance the unlabelled instance which is the most different with respect to the labelled instances.

We create a new class named `NearestInstanceStrategy` that inherits of `AbstractSingleLabelQueryStrategy`.

```
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. See that, 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 you want to run this query strategy, you must create a XML configuration file as we did in the past example.

3.5.2 Implementing a new evaluation method

The class hierarchy of JCLAL is quite flexible that allows to include new features, modify or extend existing ones, with a very low cost. Suppose that, we want to include an evaluation method, where the user provides a small labelled set from which the initial classifier is trained and an unlabelled set. Optionally, we can 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();
}
```

After creating the class `RealScenario`, if you want to create a XML configuration file for running an experiment with this new method, you can specify the following XML tags in your 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>
...
```

3.5.3 Implementing a new oracle

Create a new oracle is very simple. You 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, ask users the classes of the unlabelled instances, their responses are taken from 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 ins = 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.

3.5.4 Implementing a new stop criterion

Stop criteria are very important since they indicates when stop the execution of an experiment. Create a new stop criterion is very simple. 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(IAlgorithm algorithm) {
  return ((ClassicalALAlgorithm) algorithm).getIteration() >= maxIteration;
}
}
```

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

If you want set the maximum number of iterations by means of a XML configuration file, you 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;

}
...
}
```

3.5.5 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 the oracle. We create a class named `LastLabelledInstanceListener` that inherits of `IAlgorithmListener` interface.

```
public class LastLabelledInstanceListener implements IAlgorithmListener{

/**
 * Object for writing in 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. When the algorithm starts there is nothing to save.

```
@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 save into 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)`. In this method, we close the buffer.

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

3.6 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.

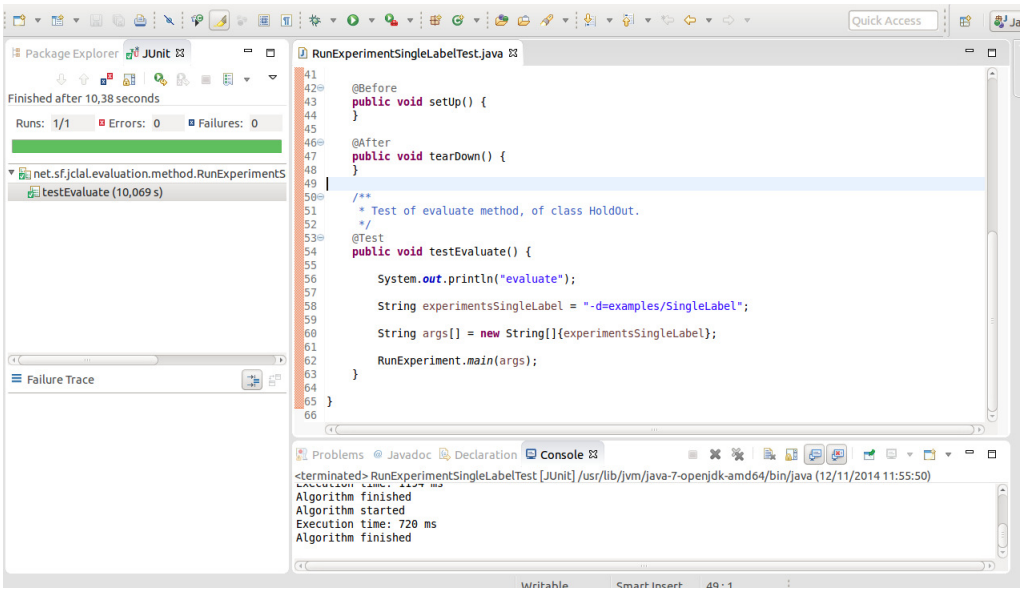


Figure 3.5: Unit test for Single-label query strategies.

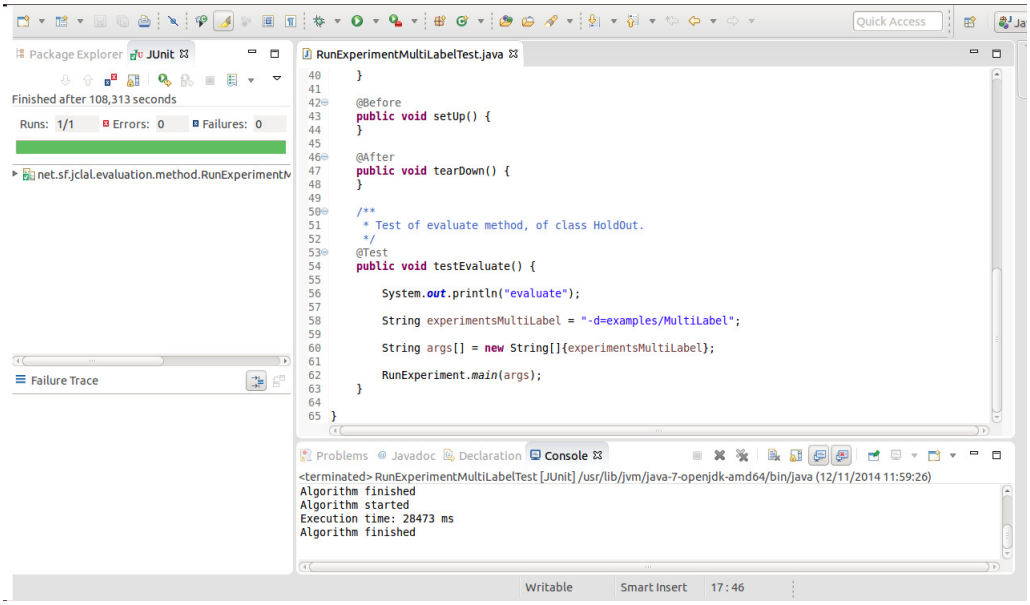


Figure 3.6: Unit test for Multi-label query strategies.

BIBLIOGRAPHY

[1] X. Zhu, "Semi-supervised learning literature survey," Tech. Rep. 1530, Computer Sciences, University of Wisconsin-Madison, 2005.

[2] B. Settles, *Active Learning*. Synthesis Lectures on Artificial Intelligence and Machine Learning, Morgan & Claypool, 1st ed., 2012.

[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] K. Brinker, *From Data and Information Analysis to Knowledge Engineering*, ch. On Active Learning in Multi-label Classification, pp. 206–213. Springer, 2006.

[6] X. Li, L. Wang, and E. Sung, "Multi-label SVM active learning for image classification," in *International Conference on Image processing*, pp. 2207–2210, 2004.

[7] B. Yang, J. Sun, T. Wang, and Z. Chen, "Effective Multi-Label Active Learning for Text Classification," in *KDD-2009*, (Paris, France), ACM, 2009.

[8] 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.

[9] X. Li and Y. Guo, "Active learning with multi-label svm classification," in *Proceedings of the Twenty-Third international joint conference on Artificial Intelligence*, pp. 1479–1485, AAAI Press, 2013.

[10] R. R. Boucaert, E. Frank, M. Hall, G. Holmes, B. Pfahringer, P. Reutemannr, and I. H. Witten, "WEKA-Experiences with a Java Open-Source Project," *Journal of Machine Learning Research*, vol. 11, pp. 2533–2541, 2010.

[11] 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.