JCLAL: A Java Framework for Active Learning

User Manual and Developer Documentation

Last update on November 24, 2014.

CONTENTS

1. INTRODUCTION

Nowadays, is common to find modern applications that have a small number of labeled examples in union with a large number of unlabeled examples. These types of datasets have motivated the development of new machine learning techniques. The Semisupervised Learning (SL) [1] and Active Learning (AL) [2] are the two main areas of research that are concerned with the development of learning algorithms capable of construct a predictive model from labeled and unlabeled 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 machine learning area. The most of these tools are restricted to the Supervised Learning and Unsupervised Learning problems. However, software tools for the AL and SL problems are scarce.

The most libraries for AL have been designed to solve specific problems. They are restricted to use Support Vector Machine (SVM) as base classifier, and they are not designed for an easy use, modification, adaptation and extension. Consecuently, the AL researchers lose a considerable time in the implementation of the existing query strategies, in order to compare the state-of-the-art AL methods with their proposals.

The list of software tools for AL which appears in the web page maintained by Burr Settles [1] evidences that the development of software tools which accelerate the experimentation process in this research area is almost null.

The above situation motivated the development of the framework JCLAL, a Java Class Library for Active Learning which includes the most significant state-of-the-art AL methods. Also, JCLAL has an easy use, and allows to the developers adapt, modify and extend the framework according to their needs.

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

1.1 What is JCLAL?

JCLAL is inspired in the arquitecture of JCLEC [3, 4, 5] which is a framework for evolutionary computation developed by KDIS group of the University of Córdoba, Spain. Therefore, JCLAL inherits the most of the advantage of the JCLEC framework.

JCLAL provides a high-level software environment to do any kind of AL method. JCLAL architecture follows strong principles of object oriented programming and design patters, where abstractions are represented by loosely coupled objects and where it is common and easy to reuse code.

[1] http://active-learning.net

JCLAL provides an environment that its main features are:

- *Generic.* JCLAL is able to execute any kind of AL method. Through a flexible classes structure, JCLAL provides the posibility of including new AL methods, as well as adapt, modify or extend the framework according to the developers's needs.
- *User friendly.* JCLAL possesses several mechanisms that offer a user friendly programming interface. JCLAL allows to users execute an experiment through a configuration file 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.* JCLAL 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 structure that follows good object oriented and generic programming principles was designed. 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 of JCLAL 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 query strategies for single-label and multi-label learning paradigms. For next versions, we hope to provide query strategies related with multi-instance and multi-label-multi-instance learning paradigms.

Up to date, the single-label query strategies that JCLAL provides are: Entropy Sampling, Least Confident and Margin Sampling which belong to the Uncertainty Sampling category. The Vote Entropy and Kullback Leibler Divergence query strategies which belong to Query By Committe category. In the Expected Error Reduction category, the Expected 0/1-loss and Expected Log-Loss query strategies are included. One query strategy which belongs to Variance Reduction family. Additionally, the Information Density framework is also provided. The references where appear for the first time these single-label query strategies can consulted on [2]. On the other hand, JCLAL provides the following multi-label query strategies: Binary Minimum [6], Max Loss [7], Mean Max Loss [7], Maximal Loss Reduction with Maximal Confidence [8], Confidence-Minimun-NonWeighted [9] and Confidence-Average-NonWeighted [9].

JCLAL implements two AL scenarios, Stream-Based Selective Sampling and Pool-Based Sampling. JCLAL provides the interfaces and abstract classes for implementing batch-mode AL methods and other types of oracles. Furthermore, JCLAL has a simple manner of defining new stopping criterion which may vary with respect to the problem. JCLAL contains a set utilities, e.g. algorithms for random number generations, sort algorithms, sampling methods, methods to compute the Area Under the Learning Curve, classes to manipulate and graphically configure an XML configuration file. Also, a plug-in which permits to integrate JCLAL with Weka's explorer is provided.

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 or source files from SourceForge.net:

  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/`.

- Download source files from SVN.

  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.

  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.

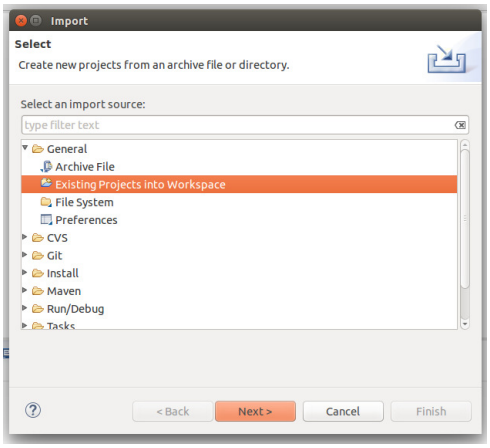
  Anonymous Mercurial access is also available via SourceForge. To access the source code repository you can use one of the following ways:

  - 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:

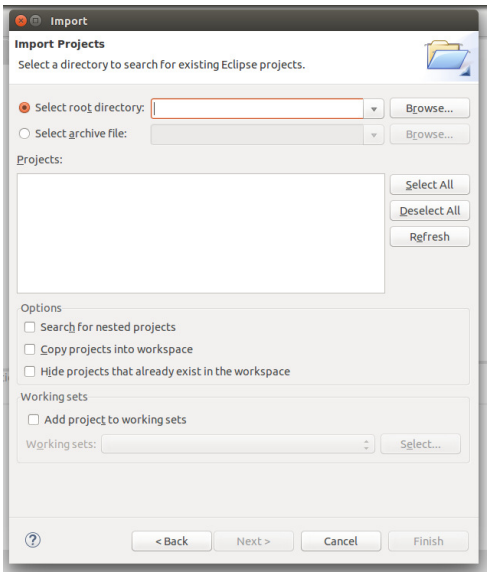
```
hg clone http://hg.code.sf.net/p/jclal/hg jclal-hg
```

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

  You can download JCLAL and import it as Eclipse project. After you have downloaded the JCLAL framework and unzip the folder into the location that you want, click on File → Import in the main menu of the Eclipse IDE.

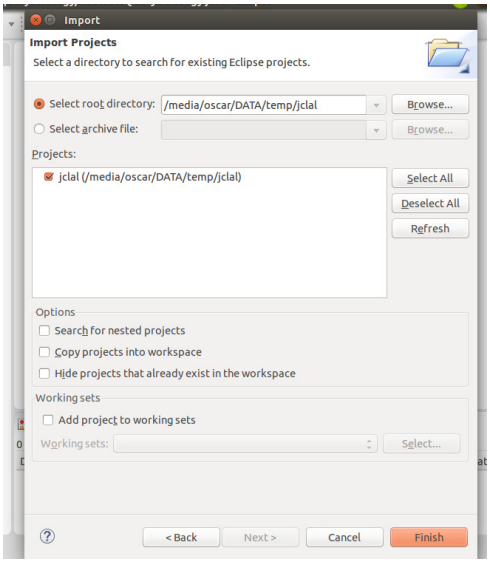


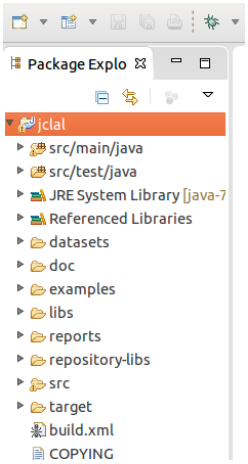
  Click on Next button and click on Browse button and find the folder where the JCLAL framework were unzipped.



  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". Click on the "Finish" button.

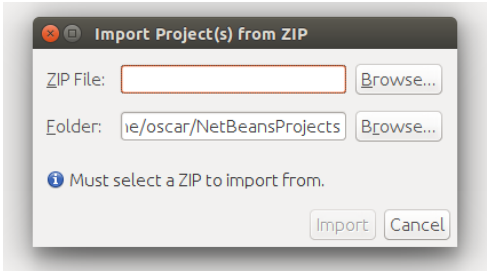
  Finally, the jclal project and its libraries are imported.



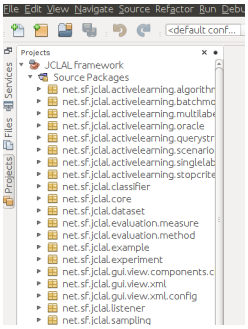


- Download source files and import as project in NetBeans IDE.

  You can also download JCLAL and import it as NetBeans project. After you have downloaded the JCLAL framework and copy the zip file into the location that you want, click on File → Import Project → From Zip.



  Click on the "Browse" button and find the zip file of JCLAL framework. Click on "Import" button. Finally, the JCLAL project and its libraries are imported.



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`".
   A jar file named "jclal-1.0.jar" will be created.
   For cleaning the project just type "`ant clean`".

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-1.0.jar" will be created.

   For cleaning the project just type "`mvn clean`".

2.3 Configuring an experiment

The JCLAL framework allows to users execute an experiment through an configuration file 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.

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. Up to date, the HoldOut and k-fold cross validation methods are supported. A sample of the HoldOut 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, only one file dataset can 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 labeled and unlabeled sets: The labeled and unlabeled sets can be declared in several ways. The users can opt to extract the labeled and unlabeled sets from the training set, for doing it a sampling method must be declared. On the other hand, the labeled and unlabeled sets could be passed directly as parameters. A sample of the case when the train set is splitted into labeled and unlabeled 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>
```

- 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 set of instances may be selected) from the unlabeled set. Afterward, an oracle labels the selected instance and the instance is added to the labeled set and removed from the unlabeled set. A sample of the AL algorithm is shown:

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

- Stopping criterions: this parameter determines when the experiment must be stopped. The JCLAL framework provides several ways to define stopping criterions. The users can define new stopping criterions acording their needs. The maximun number of iterations must be declared at least in order to guarantee that the AL process finishes. Once a stopping criterion is reached, the AL algorithm returns its results. A sample of 100 iteration is shown:

```
<max-iteration>100</max-iteration>
```

- Active learning scenario: this parameter establishes the scenario used in the AL process. 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. To date, the most significant query strategies that have appeared in the single-label and multi-label contexts are supported. Also, the base classifier to use must 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>
```

- The oracle: this parameter establishes the oracle used in the AL process. To date, two type of oracle are supported. A Simulated Oracle that reveals the hidden classes of a selected unlabeled instance, and a Console Human Oracle that iteratively asks to the user for the class of a selected unlabeled instance. The users can define new types of oracles acording their needs. A sample of the Simulated Oracle is shown:

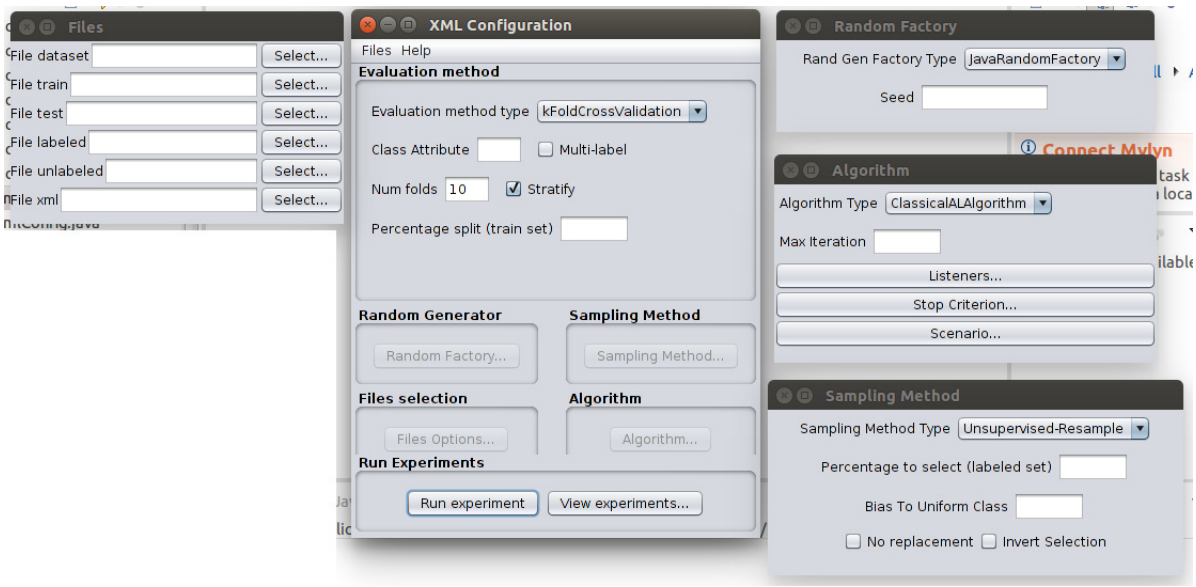
```
<oracle type="net.sf.jclal.activelearning.oracle.SimulatedOracle"/>
```

- Listener: a listener to display the results of the AL process should be determined from the package net.sf.jclal.listener. 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 printed in the console too. A window where the user can visualize the learning curve of the AL process is showed, also the user can 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>false</report-on-console>
    <report-title>Example</report-title>
    <show-window>true</show-window>
 </listener>
```

The JCLAL framework provides much more XML tags to configure an experiment file. In the JCLAL API reference can be consulted the tags that nowadays are supported by JCLAL. In the "examples" folder, which is included into the downloaded file of JCLAL framework, appears several examples of experiments of single-label and multi-label query strategies.

Additionally, in the package net.sf.jclal.gui.view.xml there is a class which allows to configure an experiment through a GUI. This user interface is simple and it can be extended in order to include new features.



2.4 Executing an experiment

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

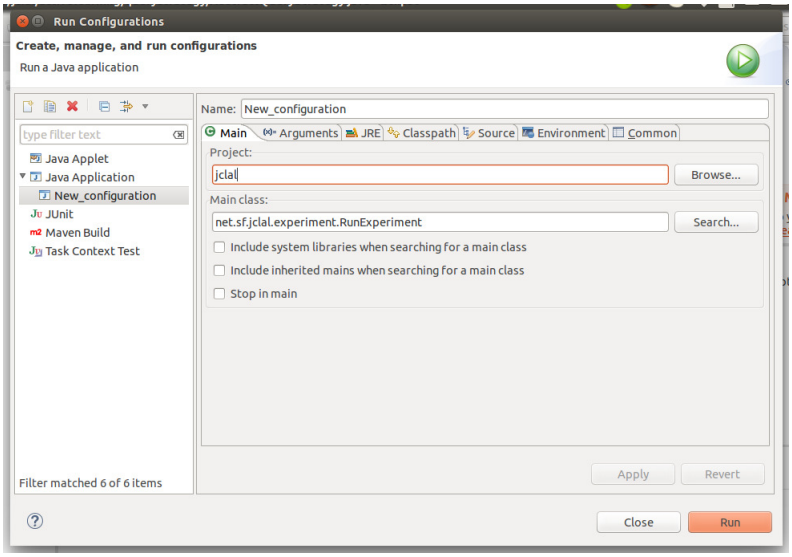
- Using JAR file

You can execute any experiment using the JAR file. For instance, you could run the Entropy Sampling query strategy, using the following command:

```
java -jar jclal-1.0.jar -cfg="examples/SingleLabel/EntropySamplingQueryStrategy.cfg"
```

- Using Eclipse IDE

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



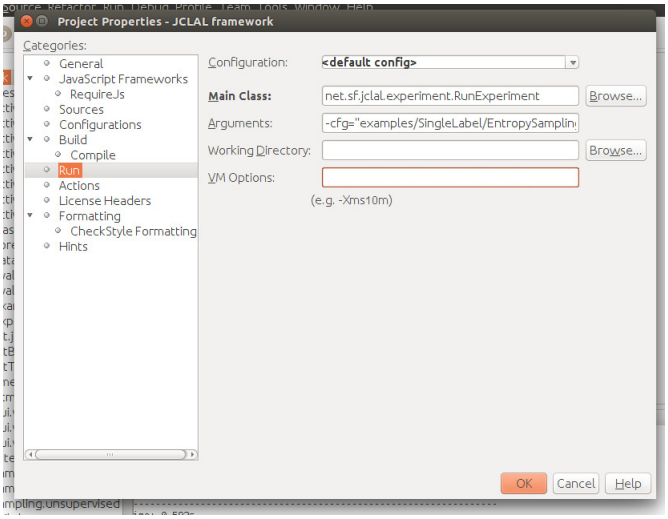

```
Project jclal
Main class: net.sf.jclal.experiment.RunExperiment
Program arguments: -cfg="examples/SingleLabel/EntropySamplingQueryStrategy.cfg"
```

Finally, we execute the AL method by clicking on the "Run" button. The user could use any of the example configuration files which are included in the JCLAL framework.

- Using NetBeans IDE

  Click on Project properties → Run
  Set the main class: net.sf.jclal.experiment.RunExperiment
  Set the program arguments: -cfg="examples/SingleLabel/EntropySamplingQueryStrategy.cfg

  Finally, we execute the AL method by clicking on the "Run" button.



2.5 Study cases

Next, two examples of experiments are showed to illustrate how to use the JCLAL framework.

2.5.1 Margin sampling query strategy

The configuration file comprises a series of parameters required to run the algorithm. This configuration file implements the Margin Sampling query strategy [2]. Below, the configuration file is shown.

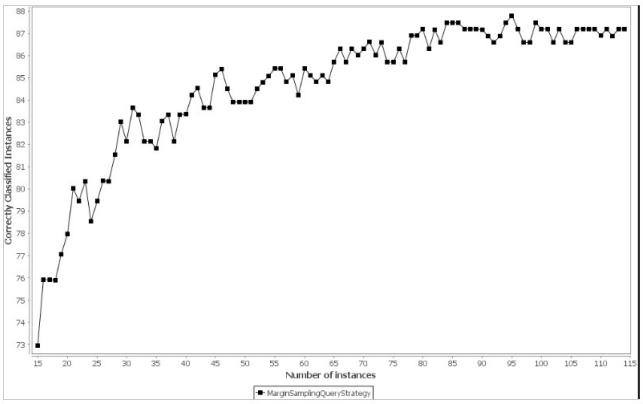
```
<experiment>
 <process evaluation-method-type="net.sf.jclal.evaluation.method.kFoldCrossValidation">
  <rand-gen-factory seed="9871234" type="net.sf.jclal.util.random.RanecuFactory"/>
  <file-dataset>datasets/ecoli.arff</file-dataset>
  <stratify>true</stratify>
  <num-folds>10</num-folds>
  <sampling-method type="net.sf.jclal.sampling.unsupervised.Resample">
   <percentage-to-select>5.0</percentage-to-select>
   <no-replacement>false</no-replacement>
   <invert-selection>false</invert-selection>
  </sampling-method>
  <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
   <max-iteration>100</max-iteration>
   <listener type="net.sf.jclal.listener.GraphicalReporterListener">
    <report-title>margin</report-title>
    <report-frequency>1</report-frequency>
    <report-directory>reports/ecoli</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="net.sf.jclal.classifier.SMO"/>
     </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 labeled set and the rest of the instances form the unlabeled 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:

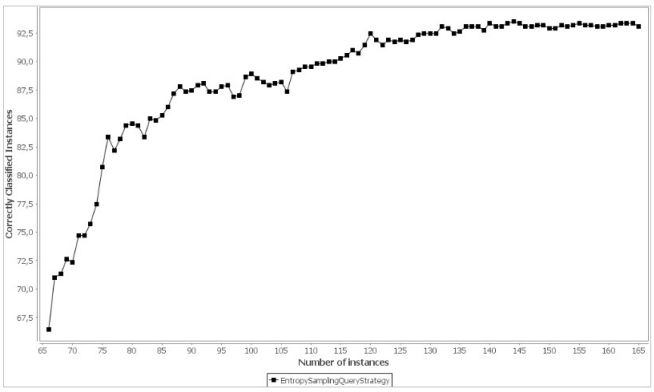


2.5.2 Entropy sampling query strategy

This configuration file implements the Entropy Sampling query strategy [2]. Below, the configuration file 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>
   <no-replacement>false</no-replacement>
   <invert-selection>false</invert-selection>
  </sampling-method>
  <algorithm type="net.sf.jclal.activelearning.algorithm.ClassicalALAlgorithm">
   <max-iteration>100</max-iteration>
   <listener type="net.sf.jclal.listener.GraphicalReporterListener">
    <report-title>entropy</report-title>
    <report-frequency>1</report-frequency>
    <report-directory>reports/mfeat</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 labeled set and the rest of the instances form the unlabeled set. A pool-based sampling scenario with the Entropy Sampling strategy is used. The Naive Bayes algorithm is used as base classifier. The learning curve obtained of running the experiment is shown:



2.6 Using JCLAL in Weka's explorer

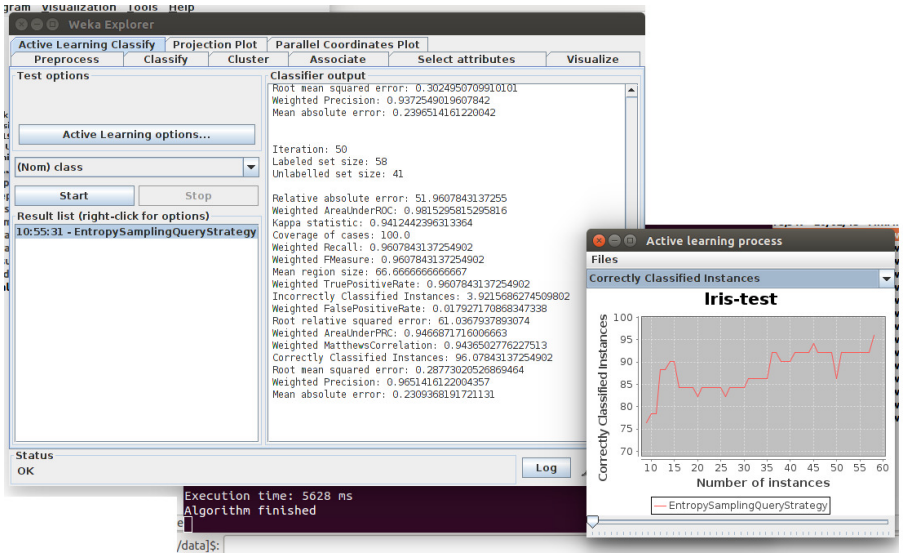
Weka [10] has become one of the most popular DM (Data Mining) workbenchs and its success in researcher and education communities is due to its constant improvement, development, and portability. This tool, developed in the Java programming language, comprises a collection of algorithms for tackling several DM tasks as data pre-processing, classification, regression, clustering and association rules.

To use JCLAL 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.



3. DOCUMENTATION FOR DEVELOPERS

This chapter is developer-oriented and describes the structure of the JCLAL framework, the API reference, how to implement new query strategies, and the unit tests validation.

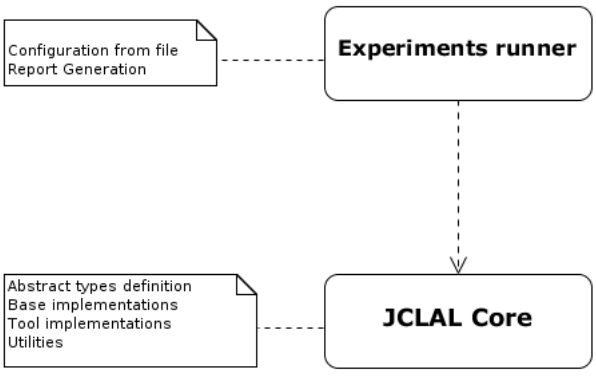
3.1 Overview

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, JCLAL 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 versions, we hope to include query strategies related with multi-instance and multi-label-multi-instance learning paradigms.

JCLAL uses the ARFF data set format. An ARFF (Attribute-Relation File Format) 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 include the Jakarta Lucene[2] library which allows convert an index file of Lucene into a arff dataset. Thus, the AL researchers in domains as text classification can use the JCLAL 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 experiments's results. JFreeChart is the most widely used chart library for Java and it is 100% free.

Two layers comprise the JCLAL architecture represented in following figure:



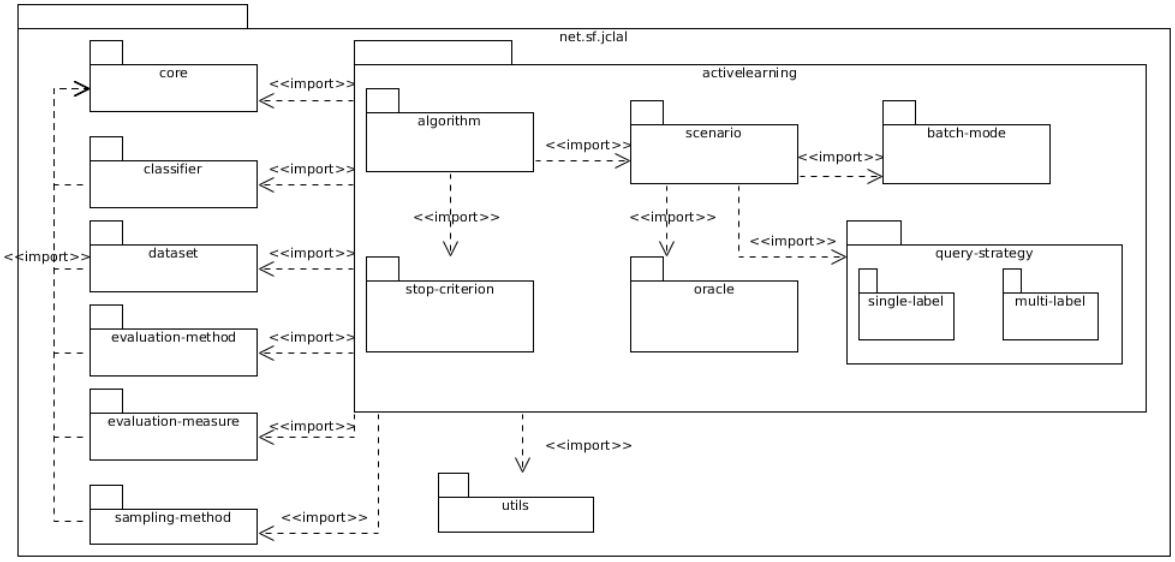
The core system is the lower layer. It has the definition of interfaces, abstract classes, its base

[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/

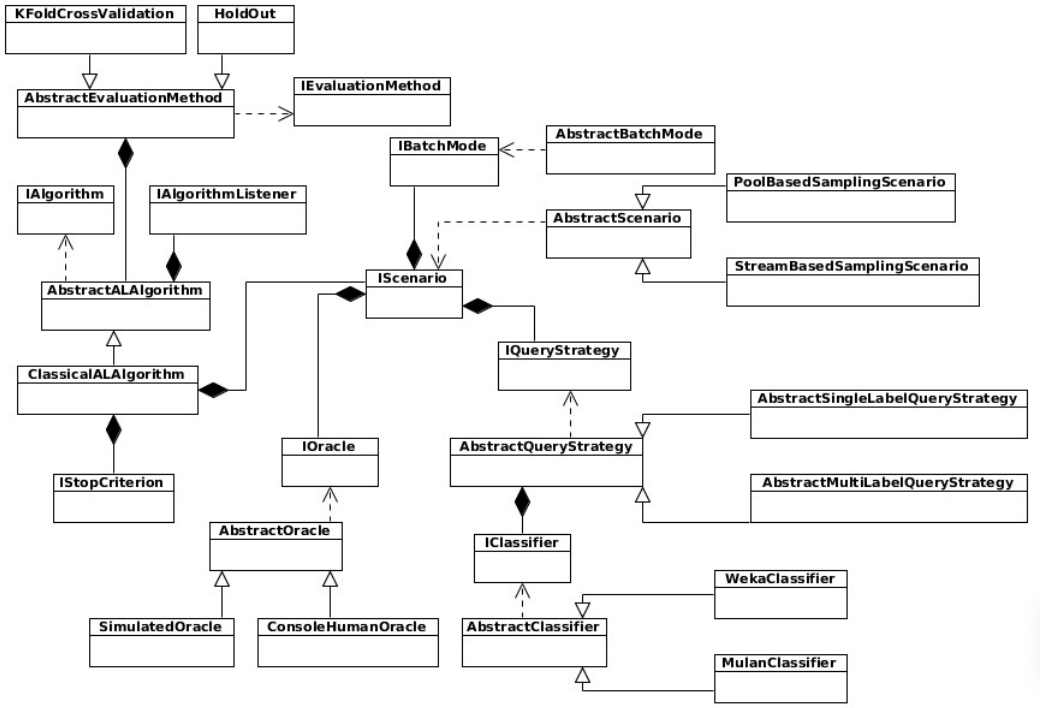
implementations and some software modules that provide all the functionality to the system. Over the core layer it is the experiments runner system in which a job is a experiment of AL defined by means of a configuration file.

3.2 Packages

The structure of the JCLAL 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 of JCLAL.



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



3.2.1 Package net.sf.jclal.core

This package contains the basic JCLAL interfaces.

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 selecting more than one unalebeled instance in each iteration.
- 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 that represents a stopping criterion.
- IStopCriterion: Interface for defining stopping criterions.
- ISystem: Interface for representing the system in an abstract way.

3.2.2 Package net.sf.jclal.activelearning.algorithm

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

Classes:

- 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.

Classes:

- 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 the classes which implement the state-of-the-art query strategies on the single-label learning paradigm.

Classes:

- 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.
- QueryByCommitte: Abstract class which represents the query strategies which 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 the classes which implement the state-of-the-art query strategies on the multi-label learning paradigm.

Classes:

- 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.

3.2.6 Package net.sf.jclal.activelearning.scenario

This package contains the classes which define the AL scenarios.

Classes:

- 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 which define the behaviour of the Batch-mode AL methods.

Classes:

- 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.

Classes:

- AbstractClassifier: Abstract class which defines the basic methods that must be implemented by any base classifier.
- WekaClassifier: Wrapper for using classifiers from Weka. This class inherits of AbstractClassifier.
- MulanClassifier: Wrapper for using classifiers from Mulan. This class inherits of AbstractClassifier.

3.2.9 Package net.sf.jclal.experiment

This package contains the classes used for running an experiment.

Classes:

- RunExperiment: Class which executes an experiment stored in a configuration file. This class represents the access point of the JCLAL framework.
- ExperimentBuilder: Interpretation of the XML configuration files.
- Experiment: Class which represents an experiment.

3.2.10 net.sf.jclal.listener

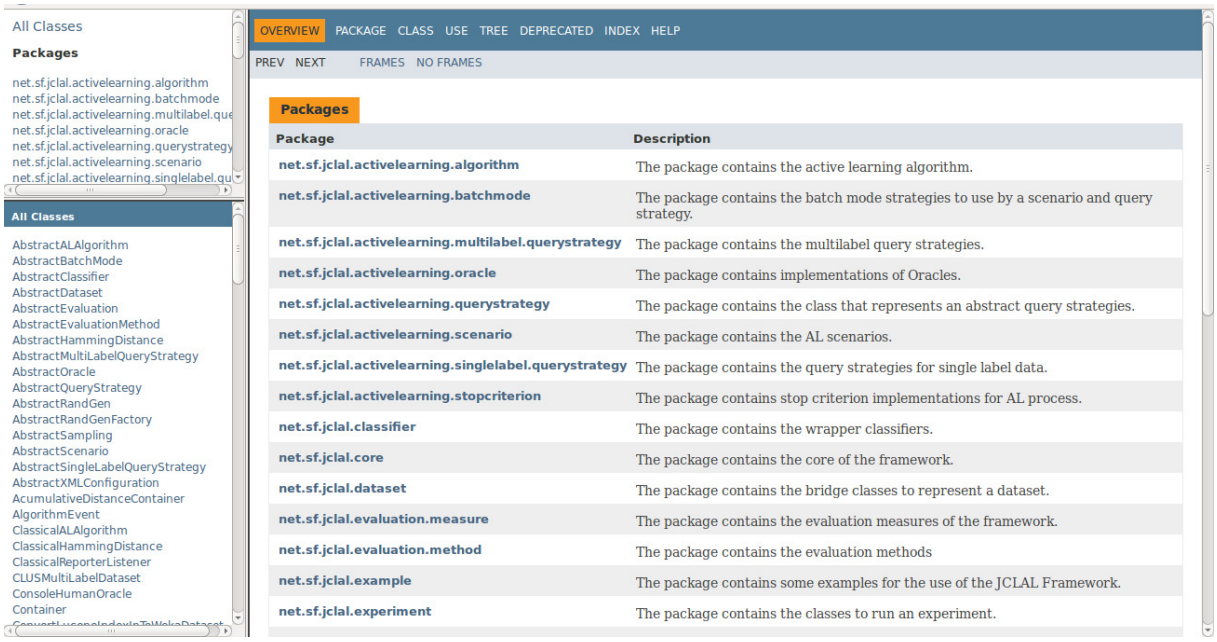
This package defines the listeners to obtain reports of the AL process.

Classes:

- 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.

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 JCLAL project.



3.4 Requirements and availability

JCLAL is available under the GNU GPL license. JCLAL requires Java 1.7, Apache commons logging 1.1, Apache commons collections 3.2, Apache commons configuration 1.5, Apache commons lang 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.

3.5 Implementing new query strategies

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

Suppose that we want to implement the Margin Sampling query strategy. Margin Sampling queries the unlabeled 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 unlabeled 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 unlabeled 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 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>
            <max-iteration>45</max-iteration>
            <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>
```

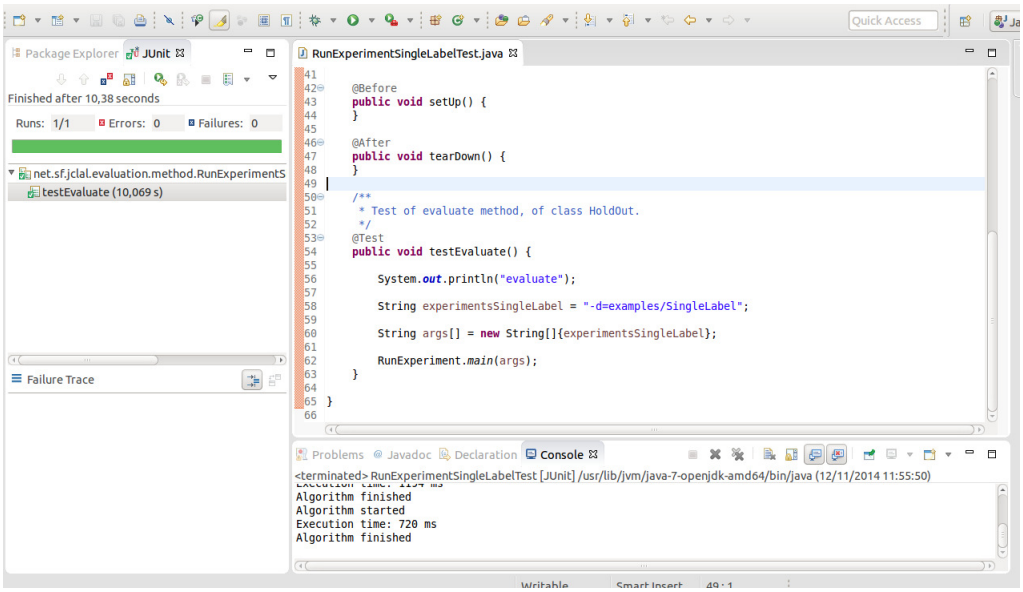
In this case, a Hold Out evaluation method is used where Iris dataset is split into training (66%) and test set (34%). An unsupervised sampling method is used to extract the 10% of the training set as the labeled set and the rest of the instances form the unlabeled set. The classical AL algorithm is used and a listener is defined. The Pool-based scenario is used, only one unlabeled instance is selected in each iteration. The Margin Sampling query strategy uses as base classifier the Naive Bayes implementation that comes in Weka library.

Finally, for running the experiment we can follow some of the ways described in the section 2.4.

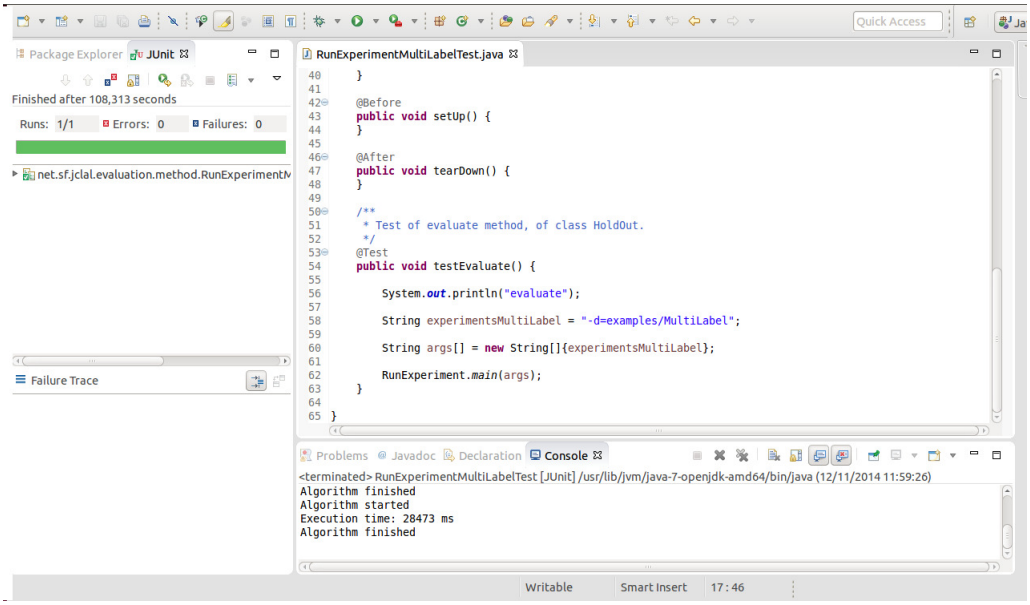
3.6 Running unit tests

In this section you will find the results of running unit tests over the query strategies available in the JCLAL framework. 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.

- Unit test for Single-label query strategies



- 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 literature survey," Computer Sciences Technical Report 1648, University of Wisconsin–Madison, 2009.

[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. M. Luna, J. L. Olmo, and S. Ventura, "JCLEC Meets WEKA!," *Lecture Notes on Computer Science*, vol. 6678, pp. 388–395, 2011.

[5] 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.

[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*, pp. 2207–2210, 2004.

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