ARTIFICIAL INTELLIGENCE RESEARCH LABORATORY
CENTER FOR COMPUTATIONAL INTELLIGENCE, LEARNING, AND DISCOVERY
DEPARTMENT OF COMPUTER SCIENCE
IOWA STATE UNIVERSITY

INDUS[1] [2] [3]

User Guide

REVISION HISTORY

O4/04/2010 Initial Version (Neeraj Koul)
05/21/2010 Included Licensing Information (Neeraj Koul)
04/05/2011 Added ILF for Linked Data section (Harris Lin)

[1] INDUS includes open source prototype implementations of algorithms and software for learning classifiers from large, semantically disparate, distributed data sources. This work was funded in part by the National Science Foundation through grants NSF IIS 0219699 to Vasant Honavar (Iowa State University), and NSF IIS 0711356 to Vasant Honavar (Iowa State University) and Doina Caragea (Kansas State University).
[2] INDUS is distributed under open source GPL license. INDUS relies on several open source software packages that are freely available under open source license or similar licensing terms.
[3] INDUS has been designed and implemented by Neeraj Koul, with contributions from Jie Bao, Jyotishman Pathak, Adrian Silvescu, Jun Zhang, Remy Younes, John Leacox under the supervision of Dr. Vasant Honavar at Iowa State University and by Roshan Chetri and Rohit Parimi under the supervision of Dr. Doina Caragea at Kansas State University.

Table of Contents

INDUS

INDUS (INtelligent Data Understanding System), is a *federated*, *query-centric* system for knowledge acquisition from distributed, semantically heterogeneous data . INDUS is composed of two different components, *INDUS Learning Framework* (ILF) and *INDUS Integration Framework (IIF).* The Learning framework (ILF) learns predictive models from a single data source (stored in a RDBMS such as MySQL) using SQL count queries. The Integration Framework allows a user or an application to view a collection of physically distributed, autonomous, semantically heterogeneous data sources (regardless of location, internal structure and query interfaces) as though they were a collection of tables structured according to an ontology supplied by the user. The Integration Framework employs ontologies and inter-ontology mappings to answer user queries against distributed, semantically heterogeneous data sources without the need for a centralized data warehouse or a common global ontology. Consequently the IIF can be used in conjunction with ILF to learn predictive models from semantically disparate data sources. However, both of them are available as separate applications and can also be run individually.

INDUS Learning Framework

Overview

The Indus learning framework is a suite of machine learning algorithms that learn from datasets using statistical queries. This framework is particularly useful in the following scenarios:

- When the data set is huge and it cannot be fit into memory (e.g. when for a massive arff file , WEKA runs out of memory)
- When access to underlying data instances is not available (due to considerations such as security or cost) but the datasources provides some statistics (like count queries)

ILF currently supports two kinds of data sources: propositional data (stored in a database table or an ARFF file) and linked data (stored in an RDF database or an RDF file). For propositional data, the framework provides learning Naive Bayes and Decision Trees classifiers; for linked data, the framework provides learning Relational Bayesian classifiers. The framework has been written so that it can be extended to include more classifiers that are amenable to the statistical queries approach.

Availability

INDUS is an open source project that is freely available . The software is provided as is without any warranty (explicit or implied). The software for the ILF component can be downloaded from the following location (http://code.google.com/p/induslearningframework/).

Using INDUS Learning Framework for Propositional Data

System Requirements

In order to run INDUS Learning Framework on your computer, you should have Java 1.6.0 or above installed in your system. INDUS requires a database to store ARFF files (See section on Configuration Files regarding how to set access to this database). The current system has been tested using a MySQL database.

Install Howto

Download the latest distribution (e.g. *induslearningframework_0.1.1.zip*) from the open source site and extract the contents to a folder in your system. The extracted folder will contain the learning framework jar (iif_0.1.1.jar), the java source code folders (src,extension_src), an examples folder, and a lib folder that contains *dependant* jars that the application needs to run.The jars in the lib folder are provide for ease of use and *Please ensure you have the proper licenses for the dependant jars.* The list of dependant jars is

- WEKA (weka.jar) for some utility functions - http://www.cs.waikato.ac.nz/ml/weka/)
- DBCP commons - http://commons.apache.org/dbcp/
  - Commons Pool http://commons.apache.org/pool/ (since DBCP requires it)
- Appropriate JDBC connector (e.g. mysql-connector-java-5.0.7-bin.jar)
- See Section on Use with INDUS Integration Framework for additional jars (if you plan to use the integration framework)

Configuration

The application uses JDBC to connect to the database that contains the training dataset for the classifier to build. The information required to make the connection via JDBC is read from a configuration file called *database.properties.* This configuration file contains the following fields (as name value pairs):

```
1.  DataSource.driverClassName=<JDBC DRIVER>
2.  DataSource.url=<URL for database containing training dataset>
3.  DataSource.username=<User Name>
4.  DataSource.password=<Password>
5.  dbPoolMinSize=<value>
6.  dbPoolMaxSize=<value>
```

In addition if a classifier is to be build from training data in a ARFF file, the same database (in *database.properties*) will be used to create a table in which the ARFF file is stored. The configuration file is to be placed in the same directory in which the application is started (see section on running the ILF). Typically this is the same directory that will contain the ILF jar (ilf.jar)

An example database.properties is given below

```
DataSource.driverClassName=com.mysql.jdbc.Driver
DataSource.url=jdbc:mysql://localhost/db_research
DataSource.username=ilf_user
DataSource.password=ilf_password
dbPoolMinSize=30
dbPoolMaxSize=70
```

Command Line Execution

When executing ILF from command line the -classpath option is to use to point to dependant jars (alternatively environment variable CLASSPATH can be set to point to the list of dependant jars) .

To run the Naïve Bayes classifier used the following command

java -classpath < semi colon seperated list of dependent jars and ilf_<version>.jar> airldm2.classifiers.bayes.NaiveBayesClassifier [options]

To run the Decision Tree classifier used the following command

java -classpath < semi colon seperated list of dependent jars and ilf.jar> airldm2.classifiers.trees.Id3SimpleClassifier [options]

The following options are supported

- **-a** The training file is to be read from an arff file and inserted into database.
- **-b** The training instances are in a relational database.
- **-trainTable** The Name of the table which contains training instances (used in conjunction with flag -b).
- **-testFile** The arff file against which to test (used in conjunction with flag –a or flag –b)
- **-trainFile** The arff file which contains training instances (use in conjunction with flag -a).
- **-?** Handle missing values

Note: The ILF is restricted to working with nominal attributes

A few example of the command line execution of the ILF are given below

1. *java -classpath ilf_0.1.1.jar;lib/mysql/mysql-connector-java-5.0.7-bin.jar;lib/weka-3.5.6/weka.jar;lib/apache/commons-dbcp-1.2.2.jar;lib/apache/commons-pool-20070730.jar;lib/junit/junit-4.5.jar;airldm2.classifiers.trees.Id3SimpleClassifier -a -trainFile examples/sample/HouseVotesTrain.arff -testFile sample/HouseVotesTest.arff*

   This command builds a decision tree classifier from the training data in the file *examples/sample/HouseVotesTrain.arff* and then evaluates the performance of the built classifier against the test data in *examples/sample/HouseVotesTest.arff*. This command results in the training data (contained in the *examples/sample/HouseVotesTrain)* being inserted into a table (whose name corresponds to the relation name in the arff file) in SQL database. Then the classifier is build by interacting with this table using SQL queries.

2. *java -classpath ilf_0.1.1.jar;lib/mysql/mysql-connector-java-5.0.7-bin.jar;lib/weka-3.5.6/weka.jar;lib/apache/commons-dbcp-1.2.2.jar;lib/apache/commons-pool-*

20070730.jar;lib/junit/junit-4.5.jar; airldm2.classifiers.trees.Id3SimpleClassifier -b -trainTable votes_train -testFile examples/sample/HouseVotesTest.arff

This command builds a *decision tree* classifier from the training data in the *votes_train* table and then evaluates the performance of the built classifier against the test data in *examples/sample/HouseVotesTest.arff. Note that the attributes in the votes_train table should correspond to the attributes described in the test arff file.* As mentioned earlier the information in *database.properties* is used to connect to the database that contains the *votes_train* table.

3. *java -classpath ilf_0.1.1.jar;lib/mysql/mysql-connector-java-5.0.7-bin.jar;lib/weka-3.5.6/weka.jar;lib/apache/commons-dbcp-1.2.2.jar;lib/apache/commons-pool-20070730.jar;lib/junit/junit-4.5.jar;; airldm2.classifiers.trees.Id3SimpleClassifier –a –trainFile examples/sample/HouseVotesTrain.arff -testFile examples/sample/HouseVotesTest.arff -?*

   Same as above but handle missing values by replacing them by the most possible value.

Running IIF using a script

Since running the IIF involves a batch file (*ilf.bat*) has been included in the *dist folder* for ease of running the IIF. To run the Naïve Bayes classifier using the batch file , use the following command:
ilf naive-bayes [options]
To run the Decision Tree classifier used the following command:
ilf decision-tree [options]
The *options* are as described earlier. A few examples of running the ILF using the batch file are given below

1. ilf decision-tree -a -trainFile examples/sample/HouseVotesTrain.arff -testFile examples/sample/HouseVotesTest.arff

2. ilf decision-tree -b -trainTable votes_train -testFile examples/sample/HouseVotesTest.arff

3. ilf decision-tree -a -trainFile examples/sample/HouseVotesTrain.arff -testFile examples/sample/HouseVotesTest.arff -?

(***Note**: Please modify the batch file appropriately in case you want to change the versions of the dependant jar files from those included in the distribution*)

API Integration

In addition to running from the command line , the ILF can be integrated into your application. It can build a specific classifier (Naïve Bayes or Decision Tree) for the data contained in an arff file (or a database) and return the built classifier. The built classifier can also be evaluated on a test data and the *Confusion Matrix* returned can be further processed by your application. The code snippet below shows an example how to integrate Naïve Bayes in your application.

```
import airldm2.core.datatypes.relational.SingleRelationDataDescriptor;
import airldm2.core.datatypes.relational.RelationalDataSource;
import  airldm2.util.SimpleArffFileReader;
import airldm2.classifiers.Evaluation
import weka.classifiers.evaluation.ConfusionMatrix;
import weka.core.Utils;

........
.......

String[] options= {"-b",  "-trainTable", "votes_train", "-
testFile","sample/HouseVotesTrain.arff"};


String trainTableName = Utils.getOption("trainTable", options);
String testFile = Utils.getOption("testFile", options);

NaiveBayesClassifier classifier = new NaiveBayesClassifier();

SingleRelationDataDescriptor  desc = null;


SimpleArffFileReader readTest = new SimpleArffFileReader(testFile);
LDTestInstances testInst = readTest.getTestInstances();
desc = (SingleRelationDataDescriptor )testInst.getDesc();

SSDataSource dataSource = new RelationalDataSource(trainTableName);
// Create a Large DataSet Instance and set its descriptor and source
LDInstances trainData = new LDInstances();
trainData.setDesc(desc);
trainData.setDataSource(dataSource);

ConfusionMatrix matrix = Evaluation.evlauateModel2(classifier, trainData,
testInst, options);
System.out.println(matrix.toString("===Confusion Matrix==="));
```

Running Test Example

Make sure you have an RDMS (e.g mySQL) instance running on your system. Create a database (say db_research) and a user (say ilf_user) that has permissions to query and create tables in the created database. Download the Ilf.zip from the open source location mentioned. The zip file contains ilf.jar, the dependant jars (weka, DBCP commons, JDBC driver), a sample example and the configuration file database.properties. Unzip the file to a location in your directory. If you are running an RDBMS instance besides mySQL, please add the relevant JDBC driver to

this directory and modify *database.properties* accordingly. If you are using mySQL (running on localhost) the database.properties would look like

```
DataSource.driverClassName=com.mysql.jdbc.Driver
DataSource.url=jdbc:mysql://localhost/db_research
DataSource.username=ilf_user
DataSource.password=ilf_password
```

The classifier for arff data in the sample folder can be built by executing the following from the command line (from the folder where you extracted the distribution) .

java -classpath ilf_0.1.1.jar;lib/mysql/mysql-connector-java-5.0.7-bin.jar;lib/weka-3.5.6/weka.jar;lib/apache/commons-dbcp-1.2.2.jar;lib/apache/commons-pool-20070730.jar;lib/junit/junit-4.5.jar;- airldm2.classifiers.trees.Id3SimpleClassifier –a –trainFile sample/HouseVotesTrain.arff -testFile sample/HouseVotesTest.arff

Alternatively the included batch file can be used to run the above command as follows :

ilf decision-tree –a –trainFile sample/HouseVotesTrain.arff -testFile sample/HouseVotesTest.arff

Note: If you are unable to build and evaluate the classifier, ensure that you are able to connect to the database and the user in *database.properties* has the correct password and has permissions to create and query tables.

Use with INDUS Integration Framework

The Learning Framework can only be used with a single datasource. To allow the Learning framework to build classifiers over multiple disparate data sources, the Indus Integration Framework is used to present the multiple datasources as a single data source. The use of this extension is made possible by supporting an additional *-indus* option (in conjunction with the –b option). In addition the option *indus_base* is used and it is followed by the directory that contains configuration files required by the Indus Integration Framework (refer INDUS Integration Framework documentation). Again you can set up the CLASSPATH explicitly or use the –classpath option with java (Note you also need to include the jars that the indus integration framework needs – in the distribution they are under lib/iif/lib folder)

Below is an example to build Naïve Bayes classifier using the said option.

java -classpath ilf_0.1.1.jar;lib/mysql/mysql-connector-java-5.0.7-bin.jar;lib/weka-3.5.6/weka.jar;lib/apache/commons-dbcp-1.2.2.jar;lib/apache/commons-pool-20070730.jar;lib/junit/junit-4.5.jar;lib/iif/iif_0.1.1.jar;lib/iif/lib/commons-lang-2.2.jar;lib/iif/lib/junit-4.4.jar;lib/iif/lib/log4j-1.2.15.jar;lib/iif/lib/mysql-connector-java-5.0.7-bin.jar;lib/iif/lib/Zql.jar;lib/iif/lib/pellet_jars/aterm-java-1.6.jar;lib/iif/lib/pellet_jars/owlapi-

bin.jar;lib/iif/lib/pellet_jars/owlapi-src.jar;lib/iif/lib/pellet_jars/pellet-cli.jar;lib/iif/lib/pellet_jars/pellet-core.jar;lib/iif/lib/pellet_jars/pellet-datatypes.jar;lib/iif/lib/pellet_jars/pellet-dig.jar;lib/iif/lib/pellet_jars/pellet-el.jar;lib/iif/lib/pellet_jars/pellet-explanation.jar;lib/iif/lib/pellet_jars/pellet-jena.jar;lib/iif/lib/pellet_jars/pellet-modularity.jar;lib/iif/lib/pellet_jars/pellet-owlapi.jar;lib/iif/lib/pellet_jars/pellet-pellint.jar;lib/iif/lib/pellet_jars/pellet-query.jar;lib/iif/lib/pellet_jars/pellet-rules.jar;lib/iif/lib/pellet_jars/pellet-test.jar;lib/iif/lib/pellet_jars/relaxngDatatype.jar;lib/iif/lib/pellet_jars/servlet.jar;lib/iif/lib/pellet_jars/xsdlib.jar;lib/iif/lib/matheval/meval.jar; airldm2.classifiers.bayes.NaiveBayesClassifier -b -indus -indus_base examples/indus_example-2 -trainTable details -testFile examples/indus_example-2/movieTest.arff

Notice this option assumes that the required configuration files required by the Indus Integration Framework are set appropriately (refer Indus Integration Framework section below). The distribution includes an *examples/indus_example-2* that has the various configuration files for the command above .

Scripts to run with INDUS Integration Framework

Again for ease of use the batch file (ilf.bat) can be used to run the ILF with IIF. The command to run ILF with the INDUS Integration Framework are:
ilf **integration** naive-bayes [options]
ilf **integration** decision-tree [options]

Note the *integration* flag to indicate that the classifier should use the INDUS Integration Framework. The *options* are as stated before .

Below is an example to build Naïve Bayes classifier (uses the Integration Framework) by executing the batch file:
ilf integration naive-bayes -b -indus -indus_base examples/indus_example-2 -trainTable details -testFile examples/indus_example-2/movieTest.arff

Using INDUS Learning Framework for Linked Data

System Requirements

ILF requires an RDF database to store linked data (See section on Configuration Files regarding how to set access to this database). The current system has been tested using a Virtuoso database (see .docs/virtuoso/README.txt for instructions on how to set up a Virtuoso database).

Configuration

ILF provides two ways to connect to the RDF database that contains the training dataset as well as the test dataset for the classifier to build – an SPARQL endpoint or a local RDF database

connection. The information required to make the connection is read from a configuration file called *rdfstore.properties.* This configuration file contains the following fields (as name value pairs):

```
1. DataSource.sparqlEndpoint=<SPARQL endpoint URL>
2. DataSource.url=<Local RDF database URL>
3. DataSource.username=<User Name>
4. DataSource.password=<Password>
```

Specify Line 1 for an SPARQL endpoint connection, and specify Line 2-4 for a local RDF database connection. If both are specified, the SPARQL endpoint will be used by default.

An example rdfstore.properties is given below

```
DataSource.sparqlEndpoint=http://localhost:8890/sparql
DataSource.url=jdbc:virtuoso://localhost:1111/charset=UTF-8/log_enable=2
DataSource.username=dba
DataSource.password=dba
```

Command Line Execution

When executing ILF from command line the -classpath option is to use to point to dependant jars (alternatively environment variable CLASSPATH can be set to point to the list of dependant jars) .

To run the Relational Bayesian classifier used the following command

java -classpath <semi colon seperated list of dependent jars and ilf_<version>.jar> airldm2.classifiers.rl.RelationalBayesianClassifier [options]

The following options are supported

- **-desc** The name of the descriptor file that describes the attributes and target types – see rbc_example/README.txt for the descriptor format.
- **-testGraph** The name of the context (RDF named graph) which contains the test triples.
- **-trainGraph** The name of the context (RDF named graph) which contains the training triples.

INDUS Integration Framework

Overview

The INDUS Integration Framework (IIF) allows users to virtually combine data that is physically spread across many data sources and run queries on that data as though it existed in a single database. The individual data sources can be Ontology Extended Data Sources and the system is able to understand queries using ontological constructs such as equivalent class, superclass and subclass respectively. The user poses the queries in SQL like syntax (INDUS SQL) which overloads the *equalto, greater than* and *less than brackets* ("=", "<" and ">") to imply equivalent class, subclass and superclass respectively. Additionally the system can be configured to imply "=" in INDUS SQL as a IS-A relationship existing in the ontology. The system uses a reasoner to handle queries with ontological constructs. The current system can be configured to use Pellet (http://clarkparsia.com/pellet) for the reasoning task (in addition to a custom reasoner developed in our group).

Availability

The IIF component of INDUS is available as an open sourced hosted at (http://code.google.com/p/indusintegrationframework/). The software is provided as is without any warranty (explicit or implied). The application is available as a zip file which includes the IIF jar file along with some dependant jars and some sample examples. The dependant jars are made available for ease of use and please ensure you have the relevant licenses as applicable.

Using INDUS Integration Framework

System Requirements

In order to run INDUS Integration Framework on your computer, you should have Java 1.6.0 or above installed in your system. INDUS requires a database to store temporary results (See section on Configuration Files regarding how to set access to this database). The temporary results are deleted when they are no longer needed. In order to run the examples that are part of the current release, a MySQL database has to be installed on the user's computer.

Install Howto

Download the latest distribution (e.g. indusintegrationframework_0.1.1.zip) from the download section of the open source site and extract the contents to a location on your system. The extracted folder will contain the Indus Integration Framework jar (e.g. iif_0.1.1.jar), few sample examples and a *lib* directory that includes a the dependant jars that are used by *Indus* Integration Framework jar. *Please ensure you have the proper licenses for the dependant jars.* The list of dependant jars is
Zql.jar - SQL Parser (**http://www.gibello.com/code/zql/**)
meval.jar - Math Evaluator (**http://lts.online.fr/dev/java/math.evaluator/**)

Pellet – Pellet OWL API Implementation (**http://clarkparsia.com/pellet**)
Log4j - (**http://logging.apache.org/log4j/**)
MySQL drivers for JDBC connectivity (**http://www.mysql.com/products/connector/**)
Apache Commons Lang (**http://commons.apache.org/lang/**)

.

Command Line Execution

java -classpath <dependant jars, Indus Integration Framework jar>
org.iastate.ailab.qengine.core.QueryEngine <configuration_directory>.

The <*configuration_directory*> is a folder that contains the various configuration files that are needed to run the example. Please see the section on IIF Configuration files for information related to the configuration files that are to be included in the <*configuration_directory*>. Some user's may find it easier/useful to check the configurations files included in an example available with the distribution (See section on Running Sample Example)

For ease of use a script (iif.bat) has been included with the distribution that can be used to run the IIF on a windows system. The syntax of running the script is

iif <*configuration_directory*>

Running Sample Example

The example included with the distribution (*examples/indus_example-2)* simulates combining movie information stored in two different datasources (named netflix and blockbuster in the example) from a user point of view. The datasources have different attribute value hierarchies (in OWL syntax) associated with them and mappings are used to resolve the semantic heterogeneity. In order to run the example, the databases *netflix* and *blockbuster* needs to be set up and additionally OWL files need to be made available at the appropriate URIs. Navigate to the *setup* folder under the directory where you extracted the zip file and run '*indus_test_database.sql'* file in proper MySQL environment. This script will set up necessary databases and users required for the demo examples to run successfully. *Note* if Pellet is used as a reasoner (correspondingly OWL syntax for the ontologies and mappings), the OWL files should be available at the URI being referred . To do so the *setup* contains a folder *movieExample* that contains the necessary OWL files. Please copy the folder in the **www** directory of web server (e.g. apache) running on your system. Once this is done you should be able to navigate to the following locations:
http://localhost/movieExample/Blockbuster.owl

http://localhost/movieExample/Netflix.owl

http://localhost/movieExample/userMovies.owl

http://localhost/movieExample/mapBlockBusterCategory.owl

http://localhost/movieExample/mapBlockBusterCategory.owl

Now perform the following steps to run the example

Step 1: open the command prompt.

Step 2: cd to the directory to which you extracted the distribution. This folder also contains some examples which can be used to test the system and to understand the scope of this project.

Step 3: Type the following command on the command line giving an example folder name as a command line argument.

java -classpath iif_0.1.1.jar;lib/commons-lang-2.2.jar;lib/junit-4.4.jar;lib/log4j-1.2.15.jar;lib/mysql-connector-java-5.0.7-bin.jar;lib/Zql.jar;lib/pellet_jars/aterm-java-1.6.jar;lib/pellet_jars/owlapi-bin.jar;lib/pellet_jars/owlapi-src.jar;lib/pellet_jars/pellet-cli.jar;lib/pellet_jars/pellet-core.jar;lib/pellet_jars/pellet-datatypes.jar;lib/pellet_jars/pellet-dig.jar;lib/pellet_jars/pellet-el.jar;lib/pellet_jars/pellet-explanation.jar;lib/pellet_jars/pellet-jena.jar;lib/pellet_jars/pellet-modularity.jar;lib/pellet_jars/pellet-owlapi.jar;lib/pellet_jars/pellet-pellint.jar;lib/pellet_jars/pellet-query.jar;lib/pellet_jars/pellet-rules.jar;lib/pellet_jars/pellet-test.jar;lib/pellet_jars/relaxngDatatype.jar;lib/pellet_jars/servlet.jar;lib/pellet_jars/xsdlib.jar;lib/matheval/meval.jar; org.iastate.ailab.qengine.core.QueryEngine examples/indus-example-2

If the command line argument is valid, we will see the "indus" prompt. A query related to the example is to be entered at the "indus" prompt.

The above example can also be easily run using the included script as

iif examples/indus_example-2

On Similar lines the other examples included in the distribution can be run.

API Integration

In addition to running from the command, the IIF can be integrated into your application. It can be used to build a specific classifier which can be used in your application as desired.

The following snippet of code indicates how to integrate into your application (once the configuration files have been set as specified in the Configuration file). The relevant classes are

- org.iastate.ailab.qengine.core.QueryEngine
- org.iastate.ailab.qengine.core.QueryResult

```
String baseDir; //  contains the directory that points to indus.conf

baseDir = System.getProperty("user.dir") + File.separator + "config";
QuerEngine engine = new QueryEngine(baseDir); //configure the Engine


 String query1 = "select COUNT(*) from EMPLOYEETABLE;";
 QueryResult result1 = Engine.execute execute (query1);  //Execute a
 query

 String query2 = "select COUNT(*) from EMPLOYEETABLE where position >
 'redshirt';";

 QueryResult result2 = Engine.execute(query2); //Execute Another Query
```

Configuration Files

The INDUS Integration Framework (IIF) reads from a number of configuration files the information it requires to query multiple ontology extended data sources from the perspective of a user. The configuration files includes a *schemaMap* configuration file to resolve schema heterogeneity, a *ontoMap* configuratuon file to specify the relations between the ontologies used by the datasource and the user and *a DTree* configuration file specifies how the multiple datasources combine to form a user view of the data. In addition a system wide configuration file (indus.conf) specifies information to access the database that is used to store any temporary results

System Configuration File

This is the main configuration file that sets up the INDUS Integration framework. This file has to be named **indus.conf**. The possible key/value pairs that can exist in this file are

Key	Value	Comments
driver_<rdbms>	<driver_class>	Examples of <rdbms>: mysql, postgress, oracle Examples of <driver_class>: com.mysql.jdbc.Driver, org.postgresql.Driver The current version has been tested with mysql
dbname_indus	<DB_name>	Name of the database in which INDUS will store temporary results
hostname_indus	<URL_to_DB>	Location of a database in which INDUS will store temporary results
dbtype_indus	<DB_type>	Type of the database in which INDUS will store

		temporary results. It must match <rdbms> for one of the driver keys provided above.
<DB_name>	<username>;<password>	A user name (with read access) and password must be provided for every database that are to be integrated
<view_filename>	<view.txt>	Path to the file containing configuration files for view.
<class_name_key>	<class_name>	Only applicable for developers who want to add custom implementations of certain components (in lieu of defaults) . This feature allows for the dynamic loading of classes other than the defaults. The possible strings for <class_name_key> can be found in: org.iastate.ailab.qengine.core.factories.solution.SolutionCreator The value of <class_name> should be the fully qualified class name of a class that implements the same interface as <class_name_key>.
equivalent_flag	FALSE or TRUE	Set the below flag to 'TRUE' to change default behavior of '=' in SQL_indus from 'IS-A' to 'equivalent' relation. Currently supported when Pellet is used as reasoner. Ignored for default reasoner which always interprets '=' as equivalent relation
world_semantics	<open_or_closed>	Future use. Currently Ignored.

The following key/value pairs in indus.conf to configure INDUS with Pellet as a Reasoner (check out indus.conf for indus-example-2 in the distribution)

```
viewImplClass=org.iastate.ailab.qengine.core.PelletViewImpl
reasonerImplClass=org.iastate.ailab.qengine.core.reasoners.impl.PelletReasonerImpl
queryTransformerImplClass=org.iastate.ailab.qengine.core.PelletQueryTransFormer
```

User Configuration Files

These includes configurations files to describe to data sources, ontologies (if any) associated with the attributes in the data sources and schema and ontology mappings to resolve schema and data content heterogeneity. Once these files have been set, the user can query over the disparate data sources as if it is a single datasource

1. The UserView File (view.txt)

This file gives a name to a user view and provides the paths to the files that describe the DTree, SchemaMap, and the Ontology Map associated with this user view (See the appendix for an example of a user view file.). The various possible key/value pairs that can exist in this file:

Key	Value	Comments
UserViewName	<name_of_userview>	Name of this userview
DTreeFile	<path_to_dtreefile>	Path to the Data Agrregation Tree File
SchemaMap	<path_to_schema_map>	Path to the Schema Map File
OntologyMap	<path_to_ontology_map>	Path to the Ontology Map File. This field is required if the reasoner used is in house reasoner.
OntMapProperties	<name_of_mappingsfile>	Name of the properties file which has the locations of the mappings files. extension (<name>.properties). This field is required if the reasoner is pellet reasoner.

The Data Aggregation Tree, Schema Map, and Ontology Map files are described below.

2. DTreeFile (tree.xml)

The DTreeFile specifies how the multiple data sources combine to form the user view of the data. In addition it contains information to configuration file that describe the schema of the data sources as well as the ontologies associated with them

It is an XML file with following structure (See the Appendix for an example of a DTreeFile)

Elements	Attributes	Subelements	Comments
Tree		Node	This is the root element.
Node	name, fragmentationType	dataSourceDescriptor, children, dbInfo	If the node is a leaf node, it must contain the subelements *dbInfo* and *dataSourceDescriptor*, otherwise it must contain the subelement *children* except for the root node, which always has the element *dataSourceDescriptor*.
dataSourceDescriptor	File		
Children	joinTable, joincol	Node	There must be one or more *node* subelements. When the fragmentation of the subelement *node*s are vertical, the *children*

			element must contain the attributes *joinTable* and *joincol*.
dbInfo	host, type		

Description of Attributes

Element: Attribute	**Value**	**Comments**
node: name	<unrestricted>	This is used to uniquely identify an element *node*. This *name* must match a *name* attribute of the *datasource* element in the Data Source Descriptor file.
node: fragmentationType	<horizontal_or_vertical>	The fragmentation type of the element *node*. The value must be of the following: horizontal, vertical
dataSourceDescriptor: file	<path_to_desc>	Relative path to the file describing the data source of the superelement *node*.
children: joinTable	<table_name>	The name of the table containing the *joincol* attribute for this *children* element.
children: joincol	<column_name>	The name of the column on which the data will be joined. If the value of *joincol* is the same for two fragmented pieces of data, then each fragment is referring to the same data as a whole.
dbInfo: host	<URL_of_DB>	Location of database described by the superelement *node*.
dbInfo: type	<DB_type>	Type of the database described by the superelement *node*. It must match <rdbms> for one of the driver keys provided above.

3. DataSource Descriptor

The Data Source Descriptor file is an XML file that describes the tables, columns, and types of data in those columns for either the virtual database that exists at the root node or for actual databases at leaf nodes. If the attribute a column is associated with an ontology, it also containts links to the file (or database) containing the ontology.

Here is the structure of a Data Source Descriptor file (see appendix for an example) :

Element: Attribute	**Value**	**Comments**

node: name	<unrestricted>	This is used to uniquely identify an element *node*. This *name* must match a *name* attribute of the *datasource* element in the Data Source Descriptor file.
node: fragmentationType	<horizontal_or_vertical>	The fragmentation type of the element *node*. The value must be of the following: horizontal, vertical
dataSourceDescriptor: file	<path_to_desc>	Relative path to the file describing the data source of the superelement *node*.
children: joinTable	<table_name>	The name of the table containing the *joincol* attribute for this *children* element.
children: joincol	<column_name>	The name of the column on which the data will be joined. If the value of *joincol* is the same for two fragmented pieces of data, then each fragment is referring to the same data as a whole.
dbInfo: host	<URL_of_DB>	Location of database described by the superelement *node*.
dbInfo: type	<DB_type>	Type of the database described by the superelement *node*. It must match <rdbms> for one of the driver keys provided above.

The description of the attributes is as follows

Element: Attribute	**Value**	**Comments**
datasource: name	<unrestricted>	This is used to uniquely identify an element *datasource*. This *name* must match a *name* attribute of the *node* element in the DTreeFile..
table: name	<table_name>	Name of the table
column: name	<col_name>	Name of the column
column: type	<col_type>	The type of the data in this column.
operation: type	<op_type>	Fixed value AVH
operation: base	<op_base>	*base* is the Uri to be appended to the values of the attribute to form concepts in the ontology
operation: ontology	<op_ontology>	The ontology associated with the attribute

operation: ontLoc	<path_to_ontLoc>	The relative path to the file that describes the ontology.

The **ontLoc** points to a file that contains an attribute value hierarchy that can be associated with an attribute in the data source. The attribute value hierarchy can be specified in OWL syntax (the system can be extended to support a user specified format) .

4. Schema Map

The Schema Map is a Java Properties file where the keys are actual table and column names that exist in real databases and their values are the table and column names from the user view (root DTree node) to which they should be mapped. The key/value pairs should have the following structure

<datasourcename>.<tablename>.<attributename> =
<userdatasource>.<usertablename>.<userattributename>

I In the case when a conversion function is required to convert the domain of an attribute in the user view to domain of an matched attribute in the datasource view the appropriate expression is added using the following syntax

<datasourcename>.<tablename>.<attributename> =
<userdatasource>.<usertablename>.<userattributename> ,exp1,exp2

where exp1 and exp2 are expressions with variable x . The exp1 is used to convert a value from the user view to datasource view and exp2 from datasource view to user view.

An example is shown below (refer examples/config-example-1 in the distribution)

DS1.DS1_Table.compensation=DS1_DS2_DS3.EMPLOYEETABLE.benefits,x+1,x-1

5. ontmap.properties File

The ***ontmap.properties*** file contains the location of the ontology mapping file which has the mappings between an attribute in user view ontology to an attribute in data source view ontology. Each entry in the ontmap.properties is a key/value pair of the following format

<UserviewDatabaseName>.<UserViewTableName>.<UserViewColumnName>|<DatasourceDatabaseName>.<DatabaseTableName>.<DatasourceColumnName> = <URL of the Mappings file>

The directory were the downloaded zip file was extracted contains a sample *ontmap.properties* file in the config-example-5 folder.

Note that the mapping file is in OWL Syntax

Custom Format (Ontology and Ontology Mappings)

Although the recommend Ontology Format is OWL, the system also supports a custom format. This has been done to demonstrate that the system can be extended.
The syntax for the custom format of the ontology is:
Three key/value pairs proceeded by a single semicolon (;). These are for information purposes only

The three key/value pairs are:

Key	Value	Comments
typename	<attributename>Type	attributeName is the column in the table with which ontology is used
subTypeOf	AVH	fixed
ordername	ISA	fixed

Following the key/value pairs is an optional alias that would be used to shorten the URIs for each item in the ontology. The alias is a key/value pair that is preceded by the string "xmlns:" (without quotes). Finally, the structure of the ontology begins with URI of the ontology followed by a open curly brace ({). Each line between the curly braces is the URI to an item in the ontology. If using an alias, use double periods (..) to denote the use of the alias. The rest of the line should complete the URI for an item of the ontology. All items in the ontology that are refered to in the Ontology Map file must be present. After the last URI, there must be a closing curly brace (}). Multiple ontologies can be present in a single file. Comments begin with the pound sign (#) or double forward slashes (//) and continue until the end of the line. The appendix includes an ontology in the custom format.

The system also supports a custom format for specifying ontology mappings. The ontology mappings are specified as key/value pairs with the following syntax

```
<targetdatasource>@<datasource_ontologyid>@<datasource_classid>
= [EQUAL|SUPER|SUB]@<userview_ontologyid>@<userview_classid>
```

The <*datasource_classid*> is the URI of a concept in the ontology with URI <*datasource_ontologyid*> that is associated with an attributed in the <*targetdatasource*>. Similarly <*userview_classid*> is the URI of a concept in the ontology with URI <*userview_ontologyid*> that is associated with an attributed in the *User View*. The keyword EQUAL, SUPER and SUB specifies equivalent class, superclass and subclass relationship between the concept with URI <*datasource_classid*> *and* between the concept with URI <*userview_classid*>. Please see appendix for an example.

Assumptions and Limitations

- The ILF is restricted to handling attributes with nominal values.

- The IIF ontologies are restricted to attribute value hierarchies.
- Schema Mappings are restricted to being one to one.
- The supported ontology mappings are restricted subclass, superclass and equivalent class relations

License

References

The following is a list of papers (partial) that have been published as a part of INDUS. Users may find it worthwhile to go through the papers to understand the system better.

1. Koul, N., and Honavar, V. (2009). Design and Implementation of a Query Planner for Data Integration. In:Proceedings of the IEEE Conference on Tools with Artificial Intelligence. IEEE Press. pp. 214-218
2. Koul, N., Caragea, C., Bahirwani, V., Caragea, D., and Honavar, V. (2008). Using Sufficient Statistics to LearnPredictive Models from Massive Data Sets. Proceedings of the ACM/IEEE/WIC Conference on Web Intelligence (WI-2008). pp. 923-926.
3. Bao, J., Caragea, D., and Honavar, V. (2006). A Tableau-based Federated Reasoning Algorithm for Modular Ontologies. ACM / IEEE / WIC Conference on Web Intelligence, Hong Kong, pp. 404-410.
4. Caragea, D., Zhang, J., Bao, J., Pathak, J., and Honavar, V. (2005). Algorithms and Software for Collaborative Discovery from Autonomous, Semantically Heterogeneous, Distributed, Information Sources. Invited paper. In: Proceedings of the Conference on Algorithmic Learning Theory. Lecture Notes in Computer Science. Vol. 3734. Berlin: Springer-Verlag. pp. 13-44.
5. Zhang, J., Caragea, D. and Honavar, V. (2005). Learning Ontology-Aware Classifiers. In: Proceedings of the 8th International Conference on Discovery Science. Springer-Verlag Lecture Notes in Computer Science. Singapore. Vol. 3735. pp. 308-321. Berlin: Springer-Verlag
6. Caragea, D., Pathak, J., and Honavar, V. (2004). Learning Classifiers from Semantically Heterogeneous Data. In: Proceedings of the International Conference on Ontologies, Databases, and Applications of Semantics (ODBASE 2004), Springer-Verlag Lecture Notes in Computer Science vol. 3291. pp. 963-980.
7. Caragea, D., Silvescu, A., and Honavar, V. (2004). A Framework for Learning from Distributed Data Using Sufficient Statistics and its Application to Learning Decision Trees. International Journal of Hybrid Intelligent Systems. **Invited Paper. Vol 1. pp. 80-89**

Appendix

Sample *View File* - using OWL syntax for ontology mappings (view.txt)

```
UserViewName=movies
DTreeFile=tree.xml
SchemaMap=schemamap.txt
OntMapProperties=ontmap.properties
```

Sample DTreeFile (tree.xml)

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<tree>
<node name="movies" fragmentationType="horizontal">
      <dataSourceDescriptor file="userviewdesc.xml" />
      <children>
            <node name="netflix" fragmentationType="horizontal">
                  <dbInfo type="mysql" host="localhost"
DRIVER="org.postgresql.Driver" datasource="netflix" />
```

```
                  <dataSourceDescriptor file="netflix_desc.xml" />
            </node>
            <node name="blockbuster" fragmentationType="horizontal">

                              <dbInfo type="mysql" host="localhost"
DRIVER="org.postgresql.Driver" datasource="blockbuster" />
                              <dataSourceDescriptor
file="blockbuster_desc.xml" />
            </node>
      </children>
</node>
</tree>
```

Sample DataSource Descriptor (e.g. userviewdesc.xml):

```
<?xml version="1.0" encoding="ISO-8859-1" ?>
<descriptors>
 <descriptor>
<datasource name="movies">
<table name="details">
  <column name="title" type="varchar" />
  <column name="genre"  type="varchar" >
    <operation type="AVH" base=
"http://localhost/movieExample/userMovies.owl" ontology =
"http://localhost/movieExample/userMovies.owl" ontLoc="userMovies.owl" />
  </column>
  <column name="reviews" type="varchar"/>
   <column name="Class" type="varchar"/>
  </table>
  </datasource>
  </descriptor>
  </descriptors>

  </descriptors>
```

Sample SchemaMap file (schemamap.txt)

```
#netflix database mappings
netflix.table1.movie=movies.details.title
netflix.table1.area=movies.details.genre
netflix.table1.stars=movies.details.reviews
netflix.table1.buy=movies.details.Class

#blockbuster database mappings
blockbuster.table2.name=movies.details.title
blockbuster.table2.category=movies.details.genre
blockbuster.table2.rating=movies.details.reviews
blockbuster.table2.watch=movies.details.Class
```

Sample ontmap.properties file:

```
#Reference to Ontolgies Containing Mappings
movies.details.genre|netflix.table1.area = http://localhost/movieExample/mapNetflixArea.owl
```

```
movies.details.genre|blockbuster.table2.category =
http://localhost/movieExample/mapBlockBusterCategory.owl
```

Sample ontology mappings file – Using OWL (mapNetflixArea.owl):

```
<?xml version="1.0"?>
<!DOCTYPE rdf:RDF [
    <!ENTITY owl "http://www.w3.org/2002/07/owl#" >
    <!ENTITY xsd "http://www.w3.org/2001/XMLSchema#" >
    <!ENTITY owl2xml "http://www.w3.org/2006/12/owl2-xml#" >
    <!ENTITY rdfs "http://www.w3.org/2000/01/rdf-schema#" >
    <!ENTITY View "http://localhost/movieExample/View.owl#" >
    <!ENTITY Netflix "http://localhost/movieExample/Netflix.owl#" >
    <!ENTITY rdf "http://www.w3.org/1999/02/22-rdf-syntax-ns#" >
    <!ENTITY userMovies "http://www.ailab.iastate.edu/userMovies.owl#" >
]>
<rdf:RDF xmlns="http://localhost/userMovies/mapNetflixArea.owl#"
     xml:base="http://localhost/userMovies/mapNetflixArea.owl"
     xmlns:owl2xml="http://www.w3.org/2006/12/owl2-xml#"
     xmlns:Netflix="http://localhost/movieExample/Netflix.owl#"
     xmlns:View="http://localhost/movieExample/View.owl#"
     xmlns:xsd="http://www.w3.org/2001/XMLSchema#"
     xmlns:userMovies="http://www.ailab.iastate.edu/userMovies.owl#"
     xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#"
     xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
     xmlns:owl="http://www.w3.org/2002/07/owl#">
    <owl:Ontology rdf:about="">
        <owl:imports
rdf:resource="http://localhost/movieExample/Netflix.owl"/>
        <owl:imports
rdf:resource="http://localhost/movieExample/UserMovies.owl"/>
    </owl:Ontology>
    <!--

////////////////////////////////////////////////////////////////////////
//////////
    //
    // Classes
    //

////////////////////////////////////////////////////////////////////////
//////////
     -->
    <!-- http://localhost/movieExample/Netflix.owl#Action -->

    <owl:Class rdf:about="&Netflix;Action">
        <owl:equivalentClass rdf:resource="&userMovies;Action"/>
    </owl:Class>
    <!-- http://localhost/movieExample/Netflix.owl#ChickFlicks -->

    <owl:Class rdf:about="&Netflix;ChickFlicks">
        <owl:equivalentClass rdf:resource="&View;RomanticComedy"/>
    </owl:Class>
    <!-- http://localhost/movieExample/Netflix.owl#Comedy -->
```

```
    <owl:Class rdf:about="&Netflix;Comedy">
        <owl:equivalentClass rdf:resource="&View;Comedy"/>
    </owl:Class>
        <!-- http://localhost/movieExample/Netflix.owl#Drama -->

    <owl:Class rdf:about="&Netflix;Drama">
        <owl:equivalentClass rdf:resource="&View;Drama"/>
    </owl:Class>
        <!-- http://localhost/movieExample/Netflix.owl#ScienceFiction -->

    <owl:Class rdf:about="&Netflix;ScienceFiction">
        <owl:equivalentClass rdf:resource="&View;ActionAdventure"/>
    </owl:Class>
        <!-- http://localhost/movieExample/Netflix.owl#Spoof -->

    <owl:Class rdf:about="&Netflix;Spoof">
        <owl:equivalentClass rdf:resource="&View;Spoof"/>
    </owl:Class>
    <!-- http://localhost/movieExample/Netflix.owl#War -->

    <owl:Class rdf:about="&Netflix;War">
        <owl:equivalentClass rdf:resource="&View;WarAction"/>
    </owl:Class>
    <!-- http://localhost/movieExample/View.owl#ActionAdventure -->

    <owl:Class rdf:about="&View;ActionAdventure"/>
    <!-- http://localhost/movieExample/View.owl#Comedy -->

    <owl:Class rdf:about="&View;Comedy"/>
    <!-- http://localhost/movieExample/View.owl#Drama -->

    <owl:Class rdf:about="&View;Drama"/>

    <!-- http://localhost/movieExample/View.owl#RomanticComedy -->
    <owl:Class rdf:about="&View;RomanticComedy"/>
     <!-- http://localhost/movieExample/View.owl#Spoof -->
    <owl:Class rdf:about="&View;Spoof"/>
    <!-- http://localhost/movieExample/View.owl#WarAction -->

    <owl:Class rdf:about="&View;WarAction"/>
        <!-- http://www.ailab.iastate.edu/userMovies.owl#Action -->

    <owl:Class rdf:about="&userMovies;Action"/>
</rdf:RDF>
```

Sample *View* file –custom format for ontologies (view.txt)

```
UserViewName=DS1_DS2_DS3
DTreeFile=tree.xml
SchemaMap=schemamap.txt
OntologyMap=ontomap.txt
```

Sample *ontology* file (custom format):

```
;typename=positionType
;subTypeOf=AVH
;ordername=ISA
xmlns:n0=www.ailab.iastate.edu/indus/uView/ont0
n0:positionType_AVH{
..\positionType_AVH\grad
..\positionType_AVH\grad\M.S
..\positionType_AVH\grad\ph.D
..\positionType_AVH\undergraduate
..\positionType_AVH\undergraduate\fe
..\positionType_AVH\undergraduate\fe\redshirt
..\positionType_AVH\undergraduate\so
..\positionType_AVH\undergraduate\jun
..\positionType_AVH\undergraduate\se
}

;typename=statusType
;subTypeOf=AVH
;ordername=ISA
xmlns:n1=www.ailab.iastate.edu/indus/ds/ont1
n1:statusType_AVH{
..\statusType_AVH\graduate
..\statusType_AVH\graduate\mast
..\statusType_AVH\graduate\doct
..\statusType_AVH\undergraduate
..\statusType_AVH\undergraduate\freshman
..\statusType_AVH\undergraduate\sophomore
..\statusType_AVH\undergraduate\junior
..\statusType_AVH\undergraduate\senior
}
```

Sample ontmap.txt file:

```
DS1@www.ailab.iastate.edu/indus/ds/ont1/statusType_AVH@_junior=
EQUAL@www.ailab.iastate.edu/indus/uView/ont0/positionType_AVH@_jun
DS1@www.ailab.iastate.edu/indus/ds/ont1/statusType_AVH@_graduate=
SUPER@www.ailab.iastate.edu/indus/uView/ont0/positionType_AVH@_M.S

DS2@www.ailab.iastate.edu/indus/ds/ont2/typeType_AVH@_junior=
SUPER@www.ailab.iastate.edu/indus/uView/ont0/positionType_AVH@_redshirt

DS2@www.ailab.iastate.edu/indus/ds/ont2/typeType_AVH@_fresh=
SUB@www.ailab.iastate.edu/indus/uView/ont0/positionType_AVH@_undergraduate
```