# COUGAR^2: AN OPEN SOURCE MACHINE LEARNING AND DATA MINING DEVELOPMENT PLATFORM

Abraham Bagherjeiran[1], Oner Ulvi Celepcikay, Rachsuda Jiamthapthaksin, Chun Sheng Chen, Vadeerat Rinsurongkawong, Seungchan Lee, Justin Thomas[2], Christoph F. Eick

Computer Science Department, University of Houston, Houston, TX 77204-3010

**Abstract**

This paper introduces Cougar^2, an innovative open source Java framework and toolset that assists researchers in designing, developing, and using machine learning and data mining algorithms. The primary mission for Cougar^2 is to provide an intuitive API to the research community with the abstraction and flexibility necessary to allow painless extension of the core framework. The Cougar^2 framework introduces and employs the Factory, Algorithm, and Model (FAM) paradigm in order to meet this objective. The FAM paradigm represents a novel combination of established object-oriented principles, design patterns, strategic abstraction, and domain knowledge geared for any machine learning or data mining task. In the spirit of the open source movement, Cougar^2 also provides facilities to make entire experiments (datasets, algorithms, and experimental configuration) available to the public for exact replication of research results. Cougar^2 has been used successfully for both state of the art spatial data mining research (regional knowledge discovery and clustering) and as the main development tool in a data mining graduate course over the past two years.

**Index Terms**

Machine Learning and Data Mining framework, open source development, region discovery, clustering algorithms

## I. Introduction

Advances in database and data acquisition technologies have generated an immense amount of data. Data mining and machine learning tools needed to analyze this immense data vary for each researcher, and often one COTS (Commercial-off-the-Shelf) product or self-developed algorithms do not meet the needs of the complete data mining task. Therefore there is a great need for open source development platforms where individuals can alter the machine learning and data mining tools based on their needs and contribute to both the academic and open source community at the same time.

[1] Now working for Yahoo Inc.
2 Now at the Johns Hopkins University Applied Physics Laboratory.

Typically, there are three problems encountered when working with open source data mining projects, especially those that involve a team of researchers: (1) individuals need to perform a large number of experiments that involve many parameter configurations that should be saved for future analysis and reuse; (2) a new algorithm often extends existing work which yields to inter-operability problems and significant effort is essentially wasted porting one algorithm into a different learning framework. (3) Former developers leave code without adequate documentation which makes it very difficult to reuse and improve. These problems require a unified framework that allows for implementing diverse categories of algorithms. Additionally, the framework should provide tools and utilities that facilitate the entire lifecycle of use: designing experiments, developing algorithms, and implementing software that lives beyond the lifetime of the project.

There are many data mining and machine learning open source projects available. The majority of them are implemented in Java for platform independence (e.g. Weka [13] and RapidMiner [11]) or in C/C++ for computational efficiency (e.g. malibu [10] and Orange [6]). Weka is a collection of machine learning algorithms for data mining tasks. It is one of the most popular open source software tool bundles and currently contains more than 150 algorithms. Developers can either add an algorithm in Weka or integrate Weka's classes into another system. RapidMiner provides a GUI for representing experiments in a tree-like structure called the operator tree and uses a multi-layered data view which represents data in memory similar to a database system view. Orange is a framework for machine learning and data mining that is implemented in C++ and Python. A graphical user interface is provided through a set of building blocks called widgets. Different open source tools have different goals. For example, RapidMiner focuses on fulfilling needs of end users and provides a GUI for the users. Unlike end-users, researchers are more interested in code validation, libraries/tools that aid algorithm construction, ease of implementing new algorithms, template structures, and code reusability.

In this paper, we introduce Cougar^2 [14] (spoken as Cougar-squared), an open source machine learning and data mining framework that provides intuitive APIs for researchers and developers. Cougar^2 provides tools not only to facilitate development but also to enhance experiment configuration. In a nutshell, Cougar^2 provides a development framework whose goal is to make code easy to write as well as to use. The core functionality consists of data representation, an algorithm design paradigm, and an experiment system. The design of algorithms in Cougar^2 is based on a 3-role model called *Factory, Algorithm and Model (FAM)*. A parallel experiment system allows configuration and consistency checking to be done before running the experiment on multiple machines, which greatly enhances the ability to run large-scale experiments. Users can save configured algorithms, trained models and experimental results and reuse them later without manually restoring object state. The framework also provides additional APIs and tools that assist implementation and data preprocessing. For instance, Cougar^2 enables a novel region discovery library that assists knowledge discovery in spatial data. Cougar^2 has proven to be a robust framework supporting the development of region discovery clustering algorithms; it has been used and extended as a development platform in graduate courses and in research conducted by our group for over two years.

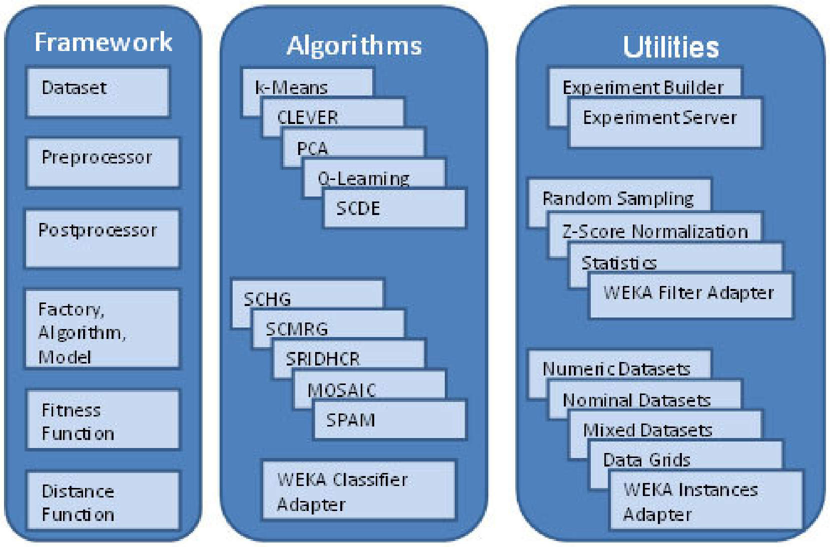

Fig. 1. Cougar^2 Platform Major Components

The remainder of the paper is organized as follows: Section 2 gives details on Cougar^2 system architecture and design decisions. Section 3 presents two main applications to illustrate how Cougar^2 is utilized in solving real-world problems. Section 4 concludes the paper.

## II. System Architecture

A key design goal of Cougar^2 is to provide minimal concrete functionality in contrast to other platforms that include functionality for all potential problems in their designs. Instead, the goal is to provide a core set of interfaces. This permits great flexibility but, more importantly, brings well-established design patterns and practices to a data mining library. The primary design areas are data representation, learning algorithms, and experiments.

### *A. Data Representation*

A fundamental challenge for any data mining library is representing data such that it is easy to access programmatically. The goals for a good design should be that the interface not to be so rigid that elaborate workarounds are necessary but not to be so general that errors can be made through misconfiguration. In Cougar^2, we have multiple interfaces for a dataset, depending on how the data are used in the software, based on the assumptions of the learning algorithm. For example, a linear regression algorithm should not have to handle the complexity of dealing with discrete-valued features. Therefore, there is no need to allow it access to such data. In contrast, a decision tree can handle multiple feature types; thus, it should accept a very general dataset interface allowing it to determine extra attribute information.

A dataset object encapsulates all information a caller needs to know about the data. Data are represented logically as a collection of examples, with each example having a fixed set of features. Associated with a feature is an object containing summary information (meta-data) such as a lookup table for discrete features or the numeric range for real-valued features. An empty dataset object is effectively a header that provides feature information. An example is indexed by an integer, but no guarantees are made about the continuity of the index. That is, an index uniquely identifies an example, but not all possible index values map to an example. This allows for running data-parallel algorithms on the same dataset. An example contains so-called real and meta-features. Real features are those available for use by a learning algorithm. Meta-features are additional features that are carried along with an example but not considered proper features, such as an identifier or a class label. Often a learning algorithm assumes the input has a consistent format, but other features such as an identifier which can be a string are carried along with the data and not to be used for learning.

The `Dataset` interface provides random access as well as an iterator to iterate over objects in the dataset. Recently in large scale data mining, there has been a trend toward streaming processing, such as in the map-reduce paradigm. Our interfaces are general enough to be easily incorporated into a streaming model using the existing Model pattern. The iterator access method integrates with the enhanced 'for loop' introduced in Java 5.0.

We contrast the design of the Cougar^2 Dataset collection of interfaces with those found in Weka. The Instances class in Weka contains all functionality related to data access. Because the Instances class is concrete and instantiated in nearly every other class in Weka, it is impossible to introduce an improvement to data access without affecting all other classes. In the development of an algorithm, it requires some programming effort to determine that the input object contains numbers rather than, say, strings or dates. In Cougar^2, the dataset interface provides functionality to copy, alter, and access data without any access to the underlying concrete class. A new interface can be utilized without affecting any other code. Different algorithms make different assumptions about the data; they should go through different interfaces. In fact, we have three interfaces for different kinds of data that can potentially be implemented in one large class. Examples (rows in the data matrix) in Weka are encapsulated in an Instance object, which adds an overhead for each example. Even moderately sized datasets exceed the memory of typical machines. In Cougar^2 there is no example class, all data access is through the appropriate interface. Because algorithms depend only on an interface, a storage class could be written so that data could reside on disk and be fetched into memory via indexes (similar to virtual memory).

## B. *Learning Algorithms*

The process of developing a model has an important and unique software lifecycle. Great time and effort is required to implement a learning algorithm. From a software perspective, the input to a learning algorithm is a dataset and some configuration parameters. The output is a standalone `Model` instance, where the model object, once trained, does not require the training data (`DataSet` instance) or on the learning algorithm (`Algorithm` instance) for run-time classification or prediction. In Cougar^2, these logical steps in the deployment of a model are decoupled into different software roles namely Factory, Algorithm, and Model (FAM) as depicted in Figure 2. To exemplify the FAM paradigm, sample code is given in Figure 3.

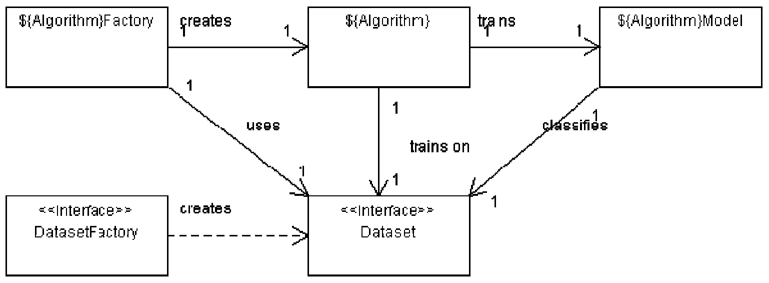

Fig. 2. Cougar^2 architecture harnessing the FAM paradigm

The factory object is a JavaBean that contains parameters for a learning algorithm, which can be serialized and reused later to train identical models. A factory is a wrapper around the constructor of the algorithm. An algorithm contains the necessary parameter validation code in the constructor. Separating the factory from the learning algorithm provides significant flexibility as well as code safety. Many learning algorithms have a wide variety of configuration parameters that can dramatically affect performance. A factory can be created containing default settings for parameters that can be reused for a wide variety of cases. The factories are safe because all parameter checking is done in the constructor. The factory will fail to build an improperly configured object. The learning algorithm, once properly configured, builds models. It is stateless in the sense that the algorithm object implements a single method, `call()`. The benefit of the stateless approach is that in a large experiment, the factory can be de-serialized and models can be built in parallel.

```
File dataFile = new File("data1.csv");      // The CSV data
File xmlFile = new File("data1_spec.xml");  // XML field spec
// Load the dataset using a dataset loader factory
CSVDatasetSpecDatasetFactory factory =
               new CSVDatasetSpecDatasetFactory();
factory.setDataFile(dataFile);
factory.setDatasetSpecFile(xmlFile);
Dataset data = factory.create();
// Create the factory that generates the CLEVER algorithm object
CleverFactory cleverFactory = new CleverFactory();
// The CLEVER algorithm supports a modular fitness function
AdditiveFitnessFunction fitnessFunction =
               new AdditiveFitnessFunction();
// Configure the algorithm
cleverFactory.setFitnessFunction(fitnessFunction);
cleverFactory.setP(50);
cleverFactory.setQ(20);
// Create the Clever algorithm object using dataset meta-data
Clever algorithm = cleverFactory.create(data);
// Generate the Model and classify the training set
CleverModel model = algorithm.call();
// Retrieve the results (region labels) from the model
NumericDataset result = model.getDataset();
```

Fig. 3. Sample code for using FAM paradigm Cougar^2

All configuration parameters, including the training data are provided through the constructor, allowing the algorithm to check configuration parameters against the data by only inspecting feature meta-data. Unlike most other libraries, a dataset object is provided in the constructor as opposed to the training method. An algorithm may have constraints on the data or may need to do some initialization. During this time, the algorithm does not need to access the examples, but some errors may occur. The algorithm class asserts that if the algorithm object can be created, the parameters and dataset are consistent. It is fail-fast, meaning that configuration errors are detected at object creation time; an inconsistent object cannot be created. The output of a learning algorithm is a model, which is designed both for post-experiment analysis and deployment in a standalone platform. In classification experiments, a model must predict the class label for incoming examples. Often in clustering, contents of the model are analyzed without the need for prediction. In either use case, the model can be serialized and reused without any functional dependency on the learning algorithm or the factory.

We compare the Cougar^2 and Weka lifecycles. In Weka, the FAM roles are combined into one class. In addition, Weka provides two methods for configuring a classifier, using JavaBeans accessor methods or using command line options. A classifier must support both. Because a classifier is a JavaBean, it must support the default constructor. Recently, tagging interfaces and annotations have been introduced to allow the user interface to distinguish classifiers that handle different types of data. Any necessary checking, however, is done in the train method. This presents significant safety risks, in that if there is a configuration error, it will be discovered when building the model. Thus, the only way to know if the model can be built is to finish building it. This causes problems when the algorithm takes a long time.

## C. *Experiments*

The ability to run experiments over a large parameter space is one of the primary use cases behind Cougar^2. It is also a key motivation behind the Factory Algorithm Model (FAM) paradigm. In data mining, some of the critical tasks are designing a new algorithm, then applying the algorithm to many datasets, and finally determining the effect of algorithm parameter changes. This usually involves testing a number of configurations that is exponential in the number of parameters. Supporting these kinds of experiments pose several challenges to most data mining libraries:

1. Configuring the learning algorithm
2. Duplicating the configuration for several datasets
3. Applying the model on the appropriate test set

4. Computing relevant performance metrics
5. Recording the configuration and resulting model for post-analysis
6. Supporting experiment resumption in the event of failure
7. Running experiments on a cluster of processors

Cougar^2 was designed to address these issues. A Factory object supports the first two requirements. A factory can create multiple copies of a learning algorithm that can run on different samples of data. Models can be serialized and de-serialized for post-experiment analysis. New performance metrics can be added and the model can re-score the examples.

Traditional file-based dependency checking packages such as make, rake, and ant are very well suited to running a graph of tasks on a single machine and re-running tasks as necessary. A research experiment, however, is often run and built in stages simultaneously with software implementation. One of the only concrete implementations included in the core package of Cougar^2 is the experiment package containing an experiment builder class. A user can build and change an experiment programmatically. In the underlying implementation, the state of an experiment is a versioned graph of dependencies. An experiment scheduler consults the versioned graph to determine which new parts of the experiments need to be run. The primary differences between the experiment builder package and file-based tools such as make are that whole sections of the graph can be altered without causing a re-run of the experiment and that the experiment system support parallel execution. Users can mark parts of the graph as invalid and all descendants will be scheduled to be re-run. This avoids the complication of having to touch files in a file-based dependency structure.

### D. *Software Engineering Aspects of Cougar^2*

Cougar^2 became an open source project in late 2006 and remains active today. All software is freely available under the GNU General Public License (GPL v 2.0). The GPL license was selected in the academic spirit to promote the free software movement, thus ensuring all derivations of the source code are always available to further the data mining and machine learning research fields. Cougar^2 is currently hosted on dev.java.net (Collabnet-based) which provides the project homepage, Subversion source code repository, mailing lists, electronic documentation, and issue tracker.

The Cougar^2 software engineering process is a mixture of open source, distributed development practices and Agile development practices. These practices, taken from eXtreme Programming (XP), include:

- Continuous Integration (using CruiseControl)
- Refactoring
- Incremental Releases
- Collective Ownership
- Sustainable Pace
- Test-Driven Development (TDD)

Arguably the most important of these practices is TDD using JUnit (xUnit framework for Java). TDD generates two primary benefits: well-designed class interfaces and high-quality, correct software. The former is a less-obvious byproduct of writing test-case code before any actual classes are implemented. In this fashion, the class interface is designed from the perspective of the class user. Consider a case of implementing a new linear classifier; the user is interested in analyzing the weights when using the classifier, which leads to introducing methods for configuring the necessary weights during testing. The latter benefit of software correctness is due to the more obvious fact that the software is being thoroughly and regularly tested.

## III. Applications and Case Studies

In this section, we demonstrate two main applications of Cougar^2 including spatial clustering algorithms and region discovery in spatial data mining; Cougar^2 is the only open source package that provides the libraries to support both applications. Generally, Cougar^2 provides many core classes and interfaces that guide and aid developers in effectively implementing algorithms so that developers can 1) speed up new algorithm implementation (since core framework classes and utility classes are given, developers can use them and focus more on the scope of their work), 2) reduce the number of errors (by performing Test-Driven Development), and 3) produce more reusable classes and experimental results (models).

## A. *Application I: Spatial Clustering Algorithms*

Generally, there are many clustering approaches that perform clustering by maximizing an objective function such as grid-based, representative-based, agglomerative-based, and density-based clustering. Since different clustering paradigms can have either implicit or explicit objective functions, Cougar^2 utilizes the fitness functions as an optional parameter. By treating fitness functions as a plug-in component, users can adapt clustering algorithms to group data in alternative ways (by using different fitness functions). There are diverse clustering algorithms implemented in Cougar^2 that allow for plug-in fitness functions. CLEVER [7] is a representative based clustering algorithm that uses representatives and randomized hill climbing. MOSAIC [2] is an agglomerative clustering algorithm that merges neighboring clusters greedily maximizing a fitness function. Finally, SCMRG [8] is a divisive grid-based clustering algorithm that uses multi-resolution grids to identify promising regions. Finally, there is another density based clustering algorithm implemented in Cougar^2, namely SCDE [9] that does not rely on any plug-in fitness function, but operates using density functions instead. In addition to the clustering algorithms, many fitness functions are implemented in Cougar^2 which achieve different knowledge discovery tasks, and are reusable among clustering algorithms such as co-location mining [5, 7], hotspot discovery [3], and regional correlation [1].

For spatial data mining, clustering is commonly performed by using spatial proximity. As discuss in Section 2.1, the `Dataset` interface in Cougar^2 is generalized so that clustering algorithms can treat spatial and non-spatial attributes separately. Therefore, distance functions efficiently calculate a spatial distance between objects while clustering algorithms evaluate clusters using non-spatial attributes. Cougar^2 also provides a library of distance functions that is also frequently used in constructing clustering algorithms.

## B. *Application II: Region Discovery in Spatial Data Mining*

Region discovery is one of the significant spatial data mining tasks that focus on discovering interesting regional patterns and spatial scope from large spatial datasets. One way to cope with region discovery is to treat the problem as a clustering problem; given a fitness function, a clustering algorithm seeks for a set of regions that maximize the fitness function which reflects the domain expert's notion of interestingness with respect to the spatial dataset analyzed. Eventually we can extract implicit but useful regional patterns from the regions obtained. Figure 4 depicts core interfaces involved in region discovery; they establish a particular instantiation of FAM in Cougar^2.

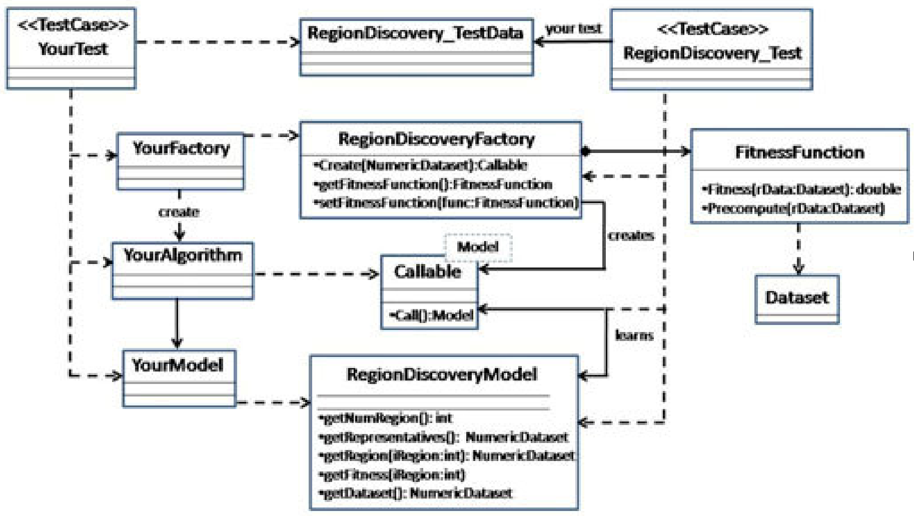
<<TestCase>>
YourTest
RegionDiscovery_TestData
your test
<<TestCase>>
RegionDiscovery_Test
YourFactory
RegionDiscoveryFactory
•Create(NumericDataset):Callable
•getFitnessFunction():FitnessFunction
•setFitnessFunction(func:FitnessFunction)
FitnessFunction
•Fitness(rData:Dataset): double
•Precompute(rData:Dataset)
create
creates
YourAlgorithm
Model
Callable
•Call():Model
Dataset
learns
YourModel
RegionDiscoveryModel
•getNumRegion(): int
•getRepresentatives(): NumericDataset
•getRegion(iRegion:int): NumericDataset
•getFitness(iRegion:int)
•getDataset(): NumericDataset

Fig. 4. Core interfaces in region discovery framework

Three new elements are introduced: `FitnessFunction`, an interface for plug-and-play fitness functions, `RegionDiscoveryModel` that provides a concrete model implementation specific to region discovery, and

`RegionDiscoveryFactory`, a specialized factory interface denoting configuration that is common to all region discovery algorithms. Outputs of region discovery typically consist of a set of clusters and its corresponding regional patterns. Merits of the region discovery framework are that conformity to `RegionDiscoveryFactory` enhances the algorithms to validate parameters before running them, whereas conformity to `RegionDiscoveryModel` enhances models reusability. Most classes in Cougar^2 implement `Serializable` to allow for object persistence.

The main focus when performing a region discovery task is on designing and constructing a fitness function and selecting an appropriate clustering algorithm. In addition, it requires running and comparing a lot of experiments and summarizing the results. We can broadly define three tasks performed in region discovery: 1) pre-processing, 2) identifying a fitness function representing a region discovery task, 3) applying a clustering algorithm that allows for the plug-in fitness function to generate a clustering. Cougar^2 provides some frequently used tools to facilitate the pre-processing step such as sampling and normalizing data, and providing an adapter interface to Weka's preprocessing utilties. In addition, Cougar^2 provides utilities for loading and storing experiment data: defining meta attributes describing spatial datasets in .xml format (both numeric and nominal features), reading data from .csv files or .arff files (Weka file format), and saving data and clustering models into files. For the second task, Cougar^2 provides structural classes to construct fitness functions in such a way that the fitness functions are reusable in any clustering algorithm. In addition, the fitness functions possess the additive property, thereby enhancing incremental computation. For the third task, there are several clustering algorithms available in Cougar^2 as discussed in the previous section.

Now, we exemplify four case studies of region discovery implemented in Cougar^2. The first case study is ***association rule mining and regional scoping***. In [3], we focus on discovering regional association rules that demonstrate associations between high/low Arsenic concentrations and other chemical concentrations. Regional scoping in [4] is further applied in order to identify the scope of the regions where the association rules are valid. The second case study is ***co-location mining***; we are interested in identifying regions where sets of attributes are co-located. In [5], we mine feature-based hotspots where extreme densities of deep ice and shallow ice co-locate on Mars. Other work in [7] discovers not only co-location regions but also regional co-location patterns between Arsenic and other chemical contamination in Texas water wells. The third case study is ***regional correlation pattern discovery using PCA***. In [1], we are interested in finding regional patterns in spatial datasets which may not exist globally. In particular, we apply a Principal Component Analysis (PCA)-based fitness function to discover regional correlation patterns—a strong correlation between a fatal disease and set of chemical concentrations in Texas water wells. Finally, ***change analysis*** analyzes how interesting regions are different in two different time frames. In [12], the region discovery framework is used in developing change analysis approaches and a set of change predicates are incorporated in analyzing changes in several locations on earth where deep earthquakes are in close proximity to shallow earthquakes.

## IV. Conclusion

In this paper, Cougar^2, an open source data mining and machine learning framework is presented that provides intuitive APIs for researchers and developers. These APIs are made available the research community with the abstraction and flexibility necessary so that extension of the core framework can be achieved easily.

The Cougar^2 framework introduces and employs the Factory, Algorithm, and Model (FAM) paradigm which enables developers to save and reuse both configurations and experimental results. FAM is a novel combination of established object-oriented principles, design patterns, and domain knowledge geared for any machine learning or data mining task. The framework also provides additional APIs and tools that assist implementation and data preprocessing. For instance, Cougar^2 enables a novel region discovery library that assists knowledge discovery in spatial data.

Additionally two main applications and four case studies in the areas of spatial clustering algorithms and region discovery in spatial data mining are provided to demonstrate the capabilities of Cougar^2 platform. Cougar^2 has proven to be a robust framework supporting the development of region discovery clustering algorithms; it has been used and extended as a development platform in graduate courses and in research conducted by our group for over two years.

## REFERENCES

[1] O. U. Celepcikay, and C. F. Eick, “A Regional Pattern Discovery Framework using Principal Component Analysis”, *in Proc. International Conference on Multivariate Statistical Modeling & High Dimensional Data Mining*, Kayseri, 2008.

[2] J. Choo, R. Jiamthapthaksin, C. Chen, O. Celepcikay, C. Giusti, and C. F. Eick, “MOSAIC: A Proximity Graph Approach to Agglomerative Clustering, *in Proc. 9th International Conference on Data Warehousing and Knowledge Discovery*, 2007

[3] W. Ding, C. F. Eick, J. Wang, and X. Yuan, “A Framework for Regional Association Rule Mining in Spatial Datasets”, *in Proc. IEEE International Conference on Data Mining*, 2006

[4] W. Ding, C. F. Eick, X. Yuan, J. Wang, and J.-P. Nicot, “On Regional Association Rule Scoping”, *in Proc. International Workshop on Spatial and Spatio-Temporal Data Mining*, 2007

[5] W. Ding, R. Jiamthapthaksin, R. Parmar, D. Jiang, T. Stepinski, and C. F. Eick, “Towards Region Discovery in Spatial Datasets”, *in Proc. Pacific-Asia Conference on Knowledge Discovery and Data Mining*, 2008

[6] J. Demsar, B. Zupan, G. Leban, and T. Curk, “Orange: From Experimental Machine Learning to Interactive Data Mining”, *in Proc. of the 8th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2004 (Demonstration paper)

[7] C. F. Eick, R. Parmar, W. Ding, T. Stepinki, and J.-P. Nicot, “Finding Regional Co-location Patterns for Sets of Continuous Variables in Spatial Datasets”, *in Proc. 16th ACM SIGSPATIAL International Conference on Advances in GIS*, 2008

[8] C. F. Eick, B. Vaezian, D. Jiang, and J. Wang, “Discovery of Interesting Regions in Spatial Datasets Using Supervised Clustering”, *in Proc. 10th European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2006

[9] D. Jiang, C. F. Eick, and C.-S. Chen, “On Supervised Density Estimation Techniques and Their Application to Clustering”, UH Technical Report UH-CS-07-09, short version appeared in Proc. 15th ACM International Symposium on Advances in Geographic Information Systems, 2007

[10] R. E. Langlois, and H. Lu, “Intelligible Machine Learning with Malibu”, *in Proc. of the 30th IEEE International Conference on Engineering in Medicine and Biology Society*, 2008