Microsoft Business Intelligence with Numerical Libraries

A White Paper by Visual Numerics, Inc.
April 2008

Visual Numerics, Inc.
2500 Wilcrest Drive, Suite 200
Houston, TX 77042
USA
www.vni.com

Microsoft Business Intelligence with Numerical Libraries

by Visual Numerics, Inc.

Copyright © 2008 by Visual Numerics, Inc. All Rights Reserved
Printed in the United States of America

Publishing History:

April 2008

Trademark Information

Visual Numerics, IMSL and PV-WAVE are registered trademarks. JMSL, TS-WAVE, and JWAVE are trademarks of Visual Numerics, Inc., in the U.S. and other countries. All other product and company names are trademarks or registered trademarks of their respective owners.

The information contained in this document is subject to change without notice. Visual Numerics, Inc. makes no warranty of any kind with regard to this material, included, but not limited to, the implied warranties of merchantability and fitness for a particular purpose. Visual Numerics, Inc, shall not be liable for errors contained herein or for incidental, consequential, or other indirect damages in connection with the furnishing, performance, or use of this material.

Microsoft images reprinted with permission from Microsoft Corporation.

TABLE OF CONTENTS

Audience 4
Rationale 4
Background 5
Plug-in Architecture 8
Managed Plug-in Development 9
IMSL C# Library: ClusterKMeans Integration 9
Starting up 10
Metadata Changes (Metadata.cs) 10
Algorithm Changes (Algorithm.cs) 11
Training and Persistence of patterns 11
Persistence of Patterns 13
Prediction 13
Algorithm Navigator Changes (AlgorithmNavigator.cs) 13
Registering the Algorithm with Analysis Services 14
Debugging 15
Other Default Features for Third-Party Mining Algorithm Developers 16
The User Experience 16
Excel 2007 19
Conclusion 21
About the Author 21
References 22
Appendix A: Code Files 23

Audience

This paper is intended for Microsoft developers who are interested in integrating third-party data mining algorithms into the Microsoft SQL Server 2005 Analysis Services (SSAS). This paper will provide a high-level overview of the SSAS architecture and its managed plug-in development environment, and will demonstrate the development of a plug-in for an IMSL® C# Numerical Library cluster K-means cluster algorithm with code examples.

Rationale

In recent years, the amounts of data available to organizations and data storage capabilities have grown exponentially. As a result, many organizations are working to leverage this captured data to make better business decisions and gain a competitive advantage. Through Business Intelligence (BI) data analysis techniques ranging from classical data mining to advanced and predictive analytics, organizations are relying on data analysis for strategic direction. To support these efforts, software developers and IT professionals are being asked to incorporate advanced data analysis methods into data analysis applications.

Based on experience with many customers implementing advanced analytics, Visual Numerics has identified a growing need for organizations to integrate analytics with existing systems and data stores (e.g., data warehouses or data marts). Integration significantly improves time-to-analysis and reduces system complexity by bringing the analytics closer to the data versus the traditional extraction–analysis– loading methods. Microsoft SQL Server is a prime target for integrated analytics with SASS's plug-in capabilities allowing the analytics to be brought closer to the data and ultimately closer to the end-users of the data.

There are typically two types of users for integrated algorithms:

- Developers who use an algorithm to create a data mining model, check for model accuracy, and make predictions using the trained model.
- Client users who use the model created by the developer. For example, a Microsoft Excel 2007 user could fulfill the role of a client.

This paper will focus on the integration of an IMSL C# Library algorithm into a Microsoft BI environment. The same techniques can be applied to other third-party C# algorithms.

For more information about the IMSL C# Library, please visit the IMSL C# Library Product Page[1].

[1] http://www.imsl.com/products/imsl/cSharp/overview.php

Background

Microsoft SQL Server provides solutions for large-scale online transaction processing, data warehousing, and e-commerce applications. With recent additions it can also act as a BI platform for data integration, analysis, and reporting solutions. The following figure shows the relationship between the SQL Server 2005 components. For more information, refer to SQL Server Overview[2].

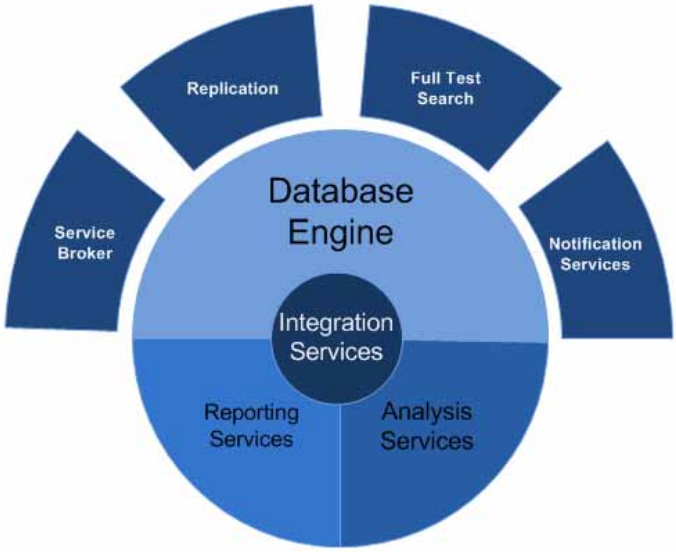


Figure 1. Microsoft SQL Server TechCenter and Relationship of Components

Additionally, SQL Server 2005 provides a SQL Management Studio to manage database objects and a BI development studio to develop BI solutions. These tools are based on Microsoft Visual Studio.

The SQL Server component that is the focus for integrating IMSL C# Library routines is "Analysis Services". Refer to Figure 2 below.

[2] http://technet.microsoft.com/en-us/library/ms166352.aspx

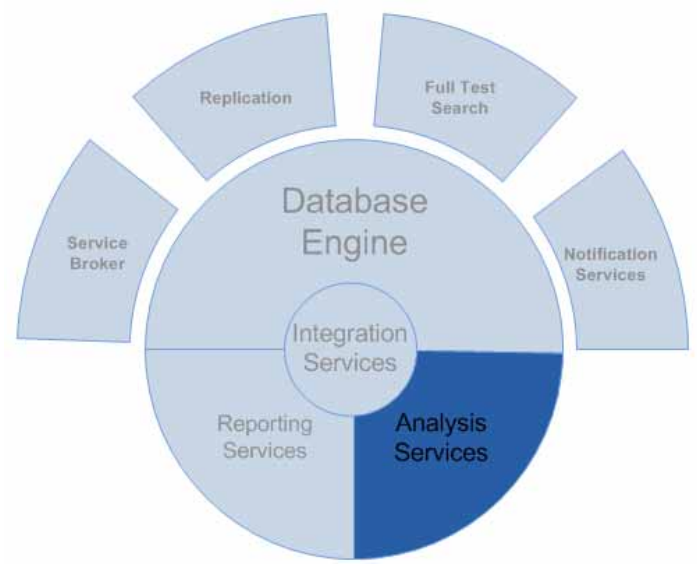


Figure 2. The SQL Server Analysis Services Component

"Analysis Services" is a Windows service that provides online analytical processing (OLAP) and data mining functionality through a combination of server and client technologies. By default, Microsoft Analysis Services provides several data mining algorithms but also allows third parties to integrate new algorithms into the Analysis Services framework. This extensibility allows for IMSL C# Library classes to be integrated in the SQL Server 2005 BI platform. For more information, see Figure 3 below or refer to the Add Custom Data Mining Algorithms to SQL Server 2005[3] article.

[3] http://technet.microsoft.com/en-us/library/aa964125.aspx

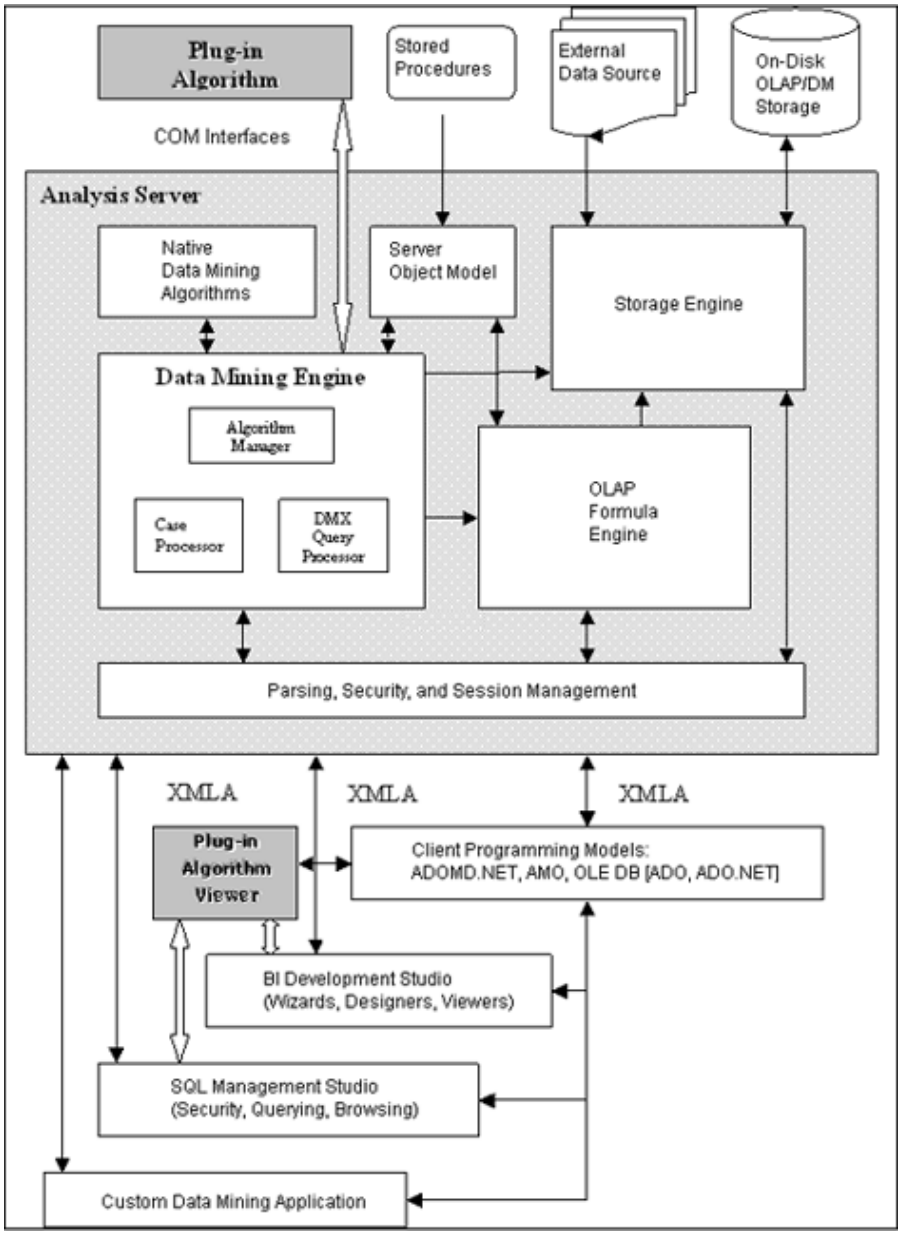


Figure 3. Data Mining Plug-in Architecture of SSAS 2005

In Microsoft Analysis Services, the integrated mining algorithms use the Unified Dimensional Model (UDM) to access data. The purpose of the UDM is to combine data from several data sources and expose it as virtual data. It creates a single version of the truth for customer data. The ability to create a UDM quickly in the Analysis Services framework allows developers to focus on the logic of their mining algorithm. For more information, refer to Figure 4 below on the Unified Dimensional Model[4].

[4] http://technet.microsoft.com/en-us/library/ms174783.aspx

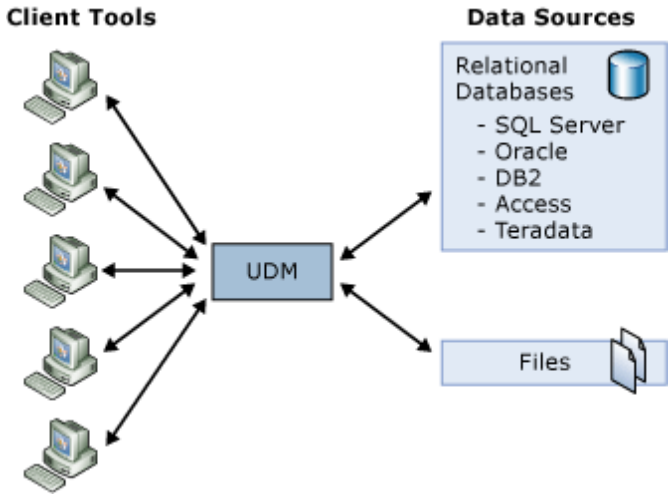


Figure 4. Unified Dimensional Model

Plug-in Architecture

The Data Mining engine communicates with the plug-in algorithms through a set of publicly available COM (Component Object Model) interfaces. However, the implementation of managed plug-ins requires the use of the DMPluginWrapper assembly. This freely available assembly implements the COM interfaces that are required for a plug-in and translates the interface calls into CLI-compliant calls. Figure 5 shows how calls into a managed plug-in are handled within Analysis Services.

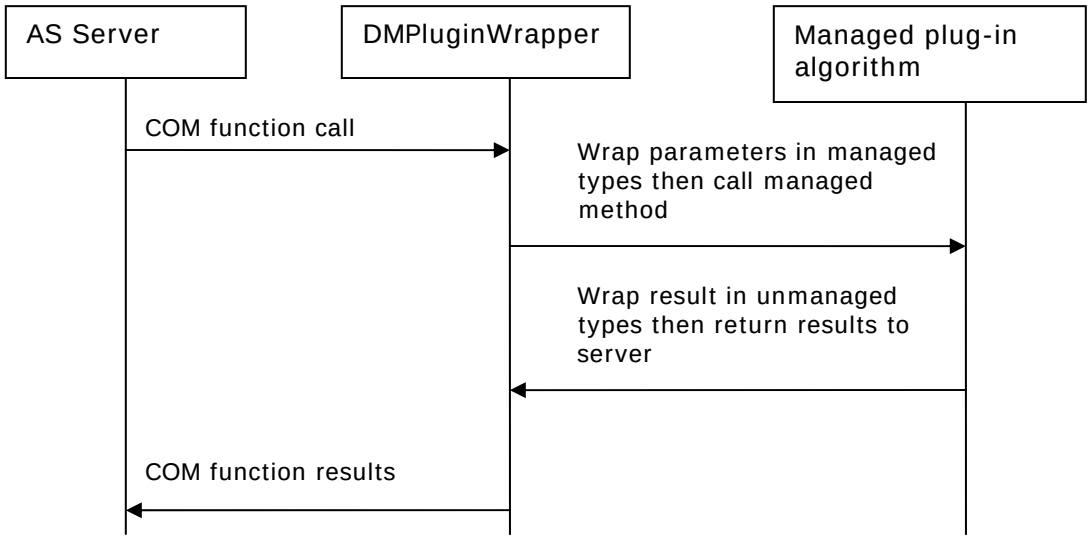


Figure5. Managed Plug-in Communication within SSAS

Managed Plug-in Development

Three classes need to be implemented to integrate a third party algorithm in SQL Server Analysis Services.

1. Metadata Class – This class is responsible for exposing the algorithm features and creates algorithm objects.
2. Algorithm Class – This class detects, persists, and uses patterns found in data.
3. Navigator Class – This class is responsible for displaying the patterns found by the Algorithm class.

For further detail, please refer to the Data Mining Managed Plug-in Algorithm API Tutorial[5] listed on http://www.sqlserverdatamining.com.

IMSL C# Library: ClusterKMeans Integration

A tutorial for constructing a managed plug-in algorithm provided by Microsoft has an example for integrating a simple algorithm in SQL Server Analysis Services. The rest of this section will explain the integration process for the ClusterKMeans class from the IMSL C# Library.

It is recommended that you follow the steps in the Data Mining Managed Plug-in Algorithm tutorial to create the shell plug-in. This stub code will be used as a template for developing the ClusterKMeans algorithm.

[5] http://www.sqlserverdatamining.com/ssdm/Home/Tutorials/tabid/57/Default.aspx

Starting up

1. Create a new folder called VNIClusterKMeans and copy the files and settings of the shell plug-in into the new folder. The shell plug-in is a solution created in Microsoft Visual Studio 2005.
2. Change all references of the Shell name to VNIClusterKMeans. This means renaming the solution, project, signature file, and any references in the project properties.
3. Make sure the project is signed and the post-build steps that register the assembly into the global assembly cache are listed in the project properties.
4. The solution should have two projects: the DMPluginWrapper and VNIClusterKMeans. In addition, VNIClusterKMeans should reference the DMPluginWrapper project. The DMPluginWrapper is a COM interop assembly that translates the COM calls from Analysis Services Server to the managed plug-in algorithm. It is freely available as part of the Data Mining Managed Plug-in Algorithm API for SQL Server 2005[6] download.

Note: The Metadata, Algorithm, and AlgorithmNavigator classes support many functions, but this document will only describe functions that need to be modified for ClusterKMeans.

Metadata Changes (Metadata.cs)

1. To make the managed code visible to the COM subsystem, decorate the Metadata class with the `[ComVisible (true)]` and `[Guid (<unique_id>)]`. In this case unique_id is obtained by selecting Tools -> Create GUID and copying the unique ID to the Metadata class. Your declaration should look like the following:

```
[ComVisible(true)]
[Guid("891DF04A-6B01-4125-B78E-C6DD8DB93471")]
[MiningAlgorithmClass(typeof(Algorithm))]
public class Metadata : AlgorithmMetadataBase
```

2. Add a constructor for the Metadata class. This constructor may call a function that declares any parameter that the user might be allowed to set before calling the algorithm. This usually happens from the BI development studio or from a client application such as Microsoft Excel. The following code allows users to set the `cluster_count` variable from client applications.

[6] http://www.microsoft.com/downloads/details.aspx?familyid=DF0BA5AA-B4BD-4705-AA0A-B477BA72A9CB

```
Public Metadata()
{
        Parameters = DeclareParameters();

}
Static public MiningParameterCollection DeclareParameters()
  {
      MiningParameterCollection parameters
              = new MiningParameterCollection();
      MiningParameter param;
      //  Sample  of  completely  populating  a  parameter  in
      constructor
      param = new MiningParameter(
                                "CLUSTER_COUNT",
                                "Number of Clusters",
                                "3",
                                "(0.0, ...)",
                                true,
                                true,
                                typeof(System.Int32));
      parameters.Add(param);
      return parameters;
     }
```

3. Change the `GetServiceName` function to return the name of the new algorithm, VNI_ClusterKMeans. Also change `GetDisplayName` and `GetServiceDescription` according to your algorithm.
4. Change `GetParametersCollection` to return the parameters.
5. Change `ParseParameterValue` to parse parameter values passed in by users.

Algorithm Changes (Algorithm.cs)

This class implements algorithm-specific tasks. It is responsible for training the algorithm, finding any patterns in the data and predicting values by making use of the trained algorithm.

Training and Persistence of patterns

The training for ClusterKMeans will have three phases:

First Phase

In the first phase, you will collect the data present in all training Cases. A Case is a data type within the Analysis Services framework. You can think of a Case as a row in a relational database. For more information, refer to the Microsoft Data Mining Help. During training, you will be presented one Case at a time. You will need to go through all of the Cases and create some sort of storage for all of the data present within each Case. The collected data will be formatted and used as an input argument to the ClusterKMeans routine. It should be noted that there is a loss of performance with the approach of collecting data from Cases. Usually, algorithms directly deal with the Cases

and do not have an intermediate step of setting up data to pass it to an algorithm. However, this transform allows us to take advantage of existing IMSL C# Library programming interfaces without any modifications.

The functions that you will need to override to accomplish the above task are the following:

- InsertCases – This function is the entry point for algorithm training. In this function, you will create a new CaseProcessor to process each Case.
- ProcessCase – This function deals with actually processing a Case. In this function, you will extract the data from the Case and store it in some sort of a container that can be retrieved at a later time. For the ClusterKMeans example, a VniStore object was used to store the data values. For more detail, please see "ClusterKmeans code" in the Appendix.

Second Phase

In the second phase, you will format the data collected in the first phase, execute the algorithm, define data patterns and associate data with each pattern.

The collected data needs to be formatted so that it can be used as an input argument to the algorithm. In the case of the ClusterKmeans, the data needs to be transformed into a two-dimensional array. See ClusterKmeans[7] documentation for further explanation of available arguments. Once the data is formatted, the algorithm can be executed. After the execution, you will work with the results from the algorithm to define data patterns. It is best to define an object to represent a pattern. For ClusterKMeans, a Cluster object (class) was used to represent a pattern. This class contains any information related to the pattern such as data and statistics. For example, if the ClusterKMeans detects three patterns, then you will have three Cluster objects to represent each of the detected patterns. Once the object is defined to represent a pattern, you will have to populate the object with the data associated with that specific pattern/cluster.

The function you will need to override or modify:

- InsertCases – Modify the source code to add the second phase that executes the algorithm and define patterns

For ClusterKMeans, a VNIStore object stores the data from the first phase and in the second phase executes the routine and associates the data with each detected pattern. For more detail, please see "ClusterKmeans code" in the Appendix.

Third Phase

In the third phase, you will be setting the statistics for each pattern or cluster. This includes setting the number of items in a pattern, min, max, variance, and probability for each attribute. You can think of an attribute as a column in a row of data. The point is to set the cluster distribution that will be used by the prediction method of the

[7] http://www.vni.com/products/imsl/cSharp/v50/manual/api/index.html

Analysis Services. To accomplish this task, you will need to add a function to your pattern object (Cluster) to update any related statistics. Please refer to the updateStats function in the Cluster class (see the Appendix for details).

Persistence of Patterns

The purpose of persistence is to save all of the required information so that it can be loaded at a later time. The SQL Server Analysis Services API provides a PersistenceWriter and PersistenceReader to accomplish these tasks. The Algorithm class should be used to save any global information, but the pattern-specific information should be delegated to the pattern class. For ClusterKMeans, the Cluster object is responsible for writing and loading pattern-specific information.

The functions you will need to override are SaveContent and LoadContent.

Prediction

In the Analysis Services paradigm, to *predict* means to return a histogram (distribution) for the target attribute. For ClusterKMeans, you will have to determine the cluster membership of the new data and then delegate the prediction task to that cluster which, in turn, returns the statistics from phase three of the model training process.

The functions you will need to override are the following:

- Predict – This function is reponsible for determining the cluster membership and delegating the prediction to that cluster.
- Cluster.predict – This function is responsible for returning the statistics determined in phase three of the training model.

Algorithm Navigator Changes (AlgorithmNavigator.cs)

This class is responsible for exposing the patterns detected by the plug-in algorithm. The SQL Server Analysis Services uses a Navigator object (this class) to expose the patterns. This object is in the form of a tree structure. Thus, it can use the notion of a current node to display node properties and also allows for switching between parent or child of the current node.

The implementation of the Navigator class depends on the Viewer that you will use for your detected patterns. By default, Microsoft provides several Viewers to display clusters, Naïve Bayes patterns, etc. For ClusterKMeans, the default Microsoft clustering viewer was used to display the detected patterns. The code to implement the Navigator object for the cluster viewer is available as an on-line example and is also listed in “A Tutorial For Constructing a Managed Plug-In Algorithm” (see reference). Since this code is available, the details are not listed in this section as there were no changes to the code. However, you may have to change parts of this code if a custom viewer is developed for your detected data patterns.

Besides overriding most of the Navigator class function according to your viewer type, the functions you will have to override are the following:

- MetaData.GetViewerType – Sets the viewer type used to display the data patterns.
- MetaData.GetServiceType – Describes the class of algorithms that includes your algorithm. For ClusterKmeans, it is ServiceTypeClustering.
- MetaData .GetSupportedStandardFunctions – Includes support for clustering specific functions.
- Algorithm.GetNavigator – Returns the navigator object. For ClusterKMeans, it returns the AlgorithmNavigator class.

Registering the Algorithm with Analysis Services

This step allows your algorithm to be used by the Analysis Services. To load your built assemblies into the Analysis services, it must be visible in the Global Assembly Cache (GAC). The post build commands in the project properties should perform this step; if you are having trouble, make sure the post build steps are accurate and point to a valid location. Once the assemblies are visible in the GAC then you will need to use the XMLA template provided in the online document "A Tutorial for Constructing a Managed Plug-In Algorithm" (see the Reference section in this white paper). Be sure to change the template accordingly to contain a description about your algorithm. The registration request using the XMLA file can be sent from the SQL Server Management studio**:**

1. Launch the SQL Server Management Studio.
2. Connect to the target Analysis Services server.
3. Choose File -> New -> Analysis Services XMLA Query.
4. Paste the XMLA statement.
5. Execute the statement.

Next, you will have to restart the Analysis Service. Select Control Panel -> Administrative Tools -> Services -> SQL Server Analysis Services (MSSQLSERVER) and restart the service. At this point your newly created algorithm should be available to all clients connecting to the Analysis Services.

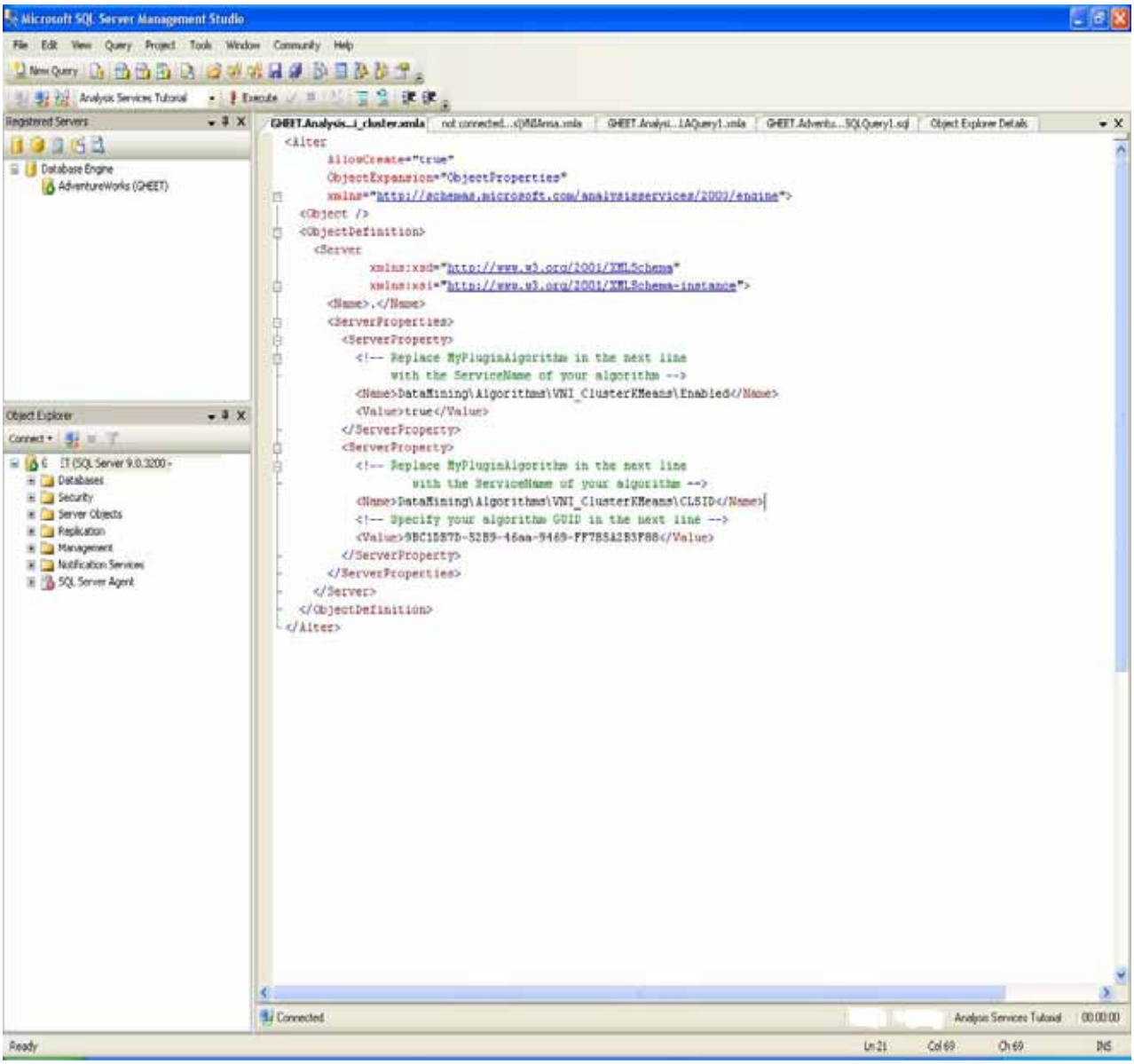


Figure 6. Enabling an algorithm to be used by the Analysis Services

Debugging

To debug your algorithm, you must first register it with the Analysis Services (see above). After registration, select Debug -> Attach to process it from the Visual Studio environment. You will be presented with the Attach To Process Dialog. In the Attach To text field, make sure managed code is selected. Under the Available processes, select the `msmdsrv.exe` process. After this selection, you should be in the Debug session, where you should be able to perform your normal debugging tasks. While in a debug session, a client application must use your algorithm for execution to stop at any valid breakpoints. Note that any modification to your algorithm will require it to be re-registered with the Analysis Services.

Other Default Features for Third-Party Mining Algorithm Developers

In addition to the UDM, there are several default features available to third-party data mining algorithm developers. The following is a list of a few features that might be beneficial for IMSL C# Library routines:

1. The integrated mining algorithms can be accessed as a Web service, since Analysis Services is a native XMLA (XML for Analysis) server that can be accessed by TCP or HTTP protocols.
2. Data mining results can be easily distributed through the SQL Server 2005 Reporting Services.
3. Enterprise deployment: multiple users, secure storage, access control, and easy deployment to a sharepoint server.
4. Interoperability with other data-mining products via PMML.
5. Automatic integration of your data mining algorithm within Excel 2007 allows the large Excel user base to directly access the mining algorithm using Excel's Data Mining add-ins.
6. A scalable training and querying engine.

The User Experience

This section provides a brief description for the user experience in the BI development studio and Excel.

Data Mining developers use the BI development studio to develop a model. Start by creating the Analysis Services project. The following figure shows the initial state of an Analysis Services project.

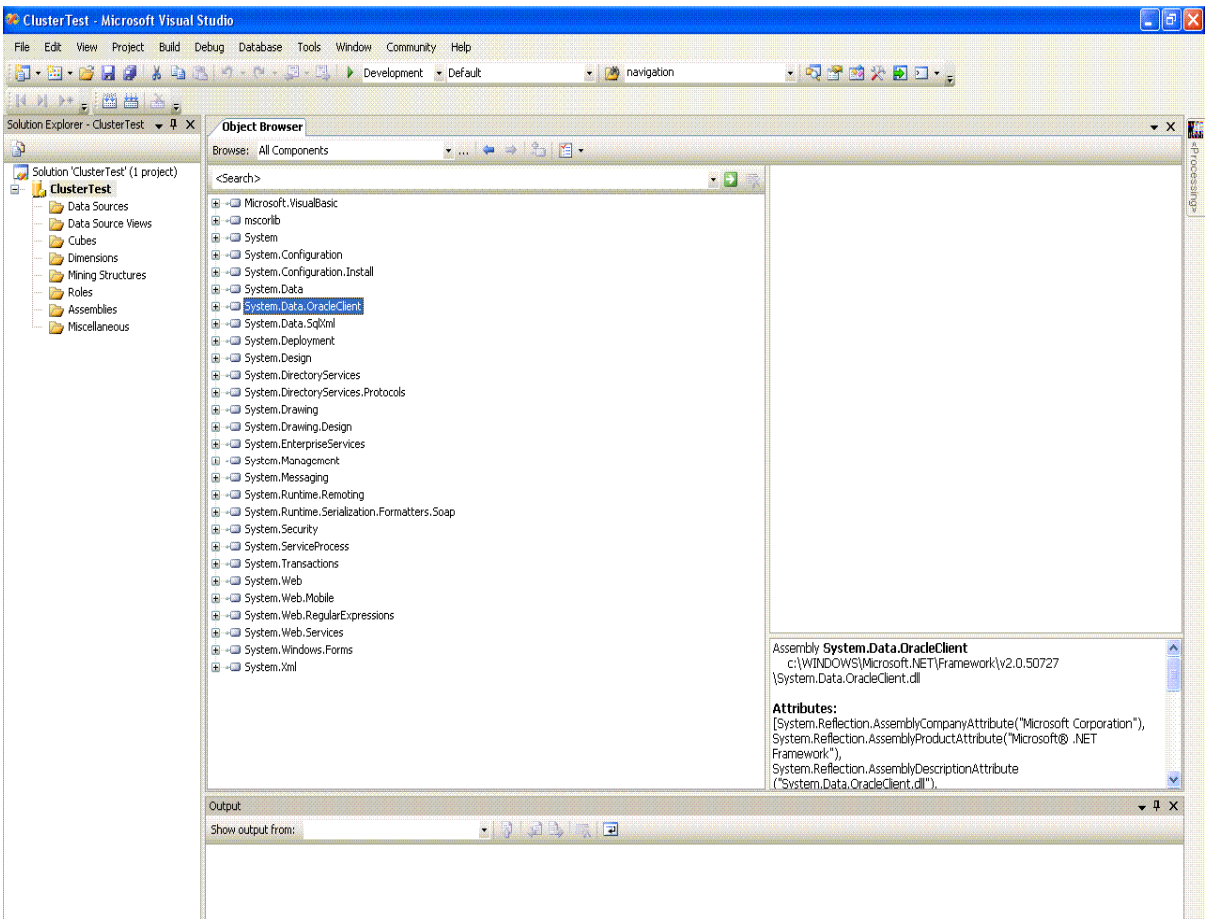


Figure 7. Initial State on an Analysis Services Project

Before you can start using your mining algorithm, you will need to define data sources and data source views. Right click on the Data Sources and follow the instructions presented by the wizard. Do the same for Data Source View. You can think of a data source as a database and the data source view as a table within the database. Next, right-click on the Mining structure, and your algorithm is automatically available in the list of available algorithms if the registration of algorithm was successful (see above).

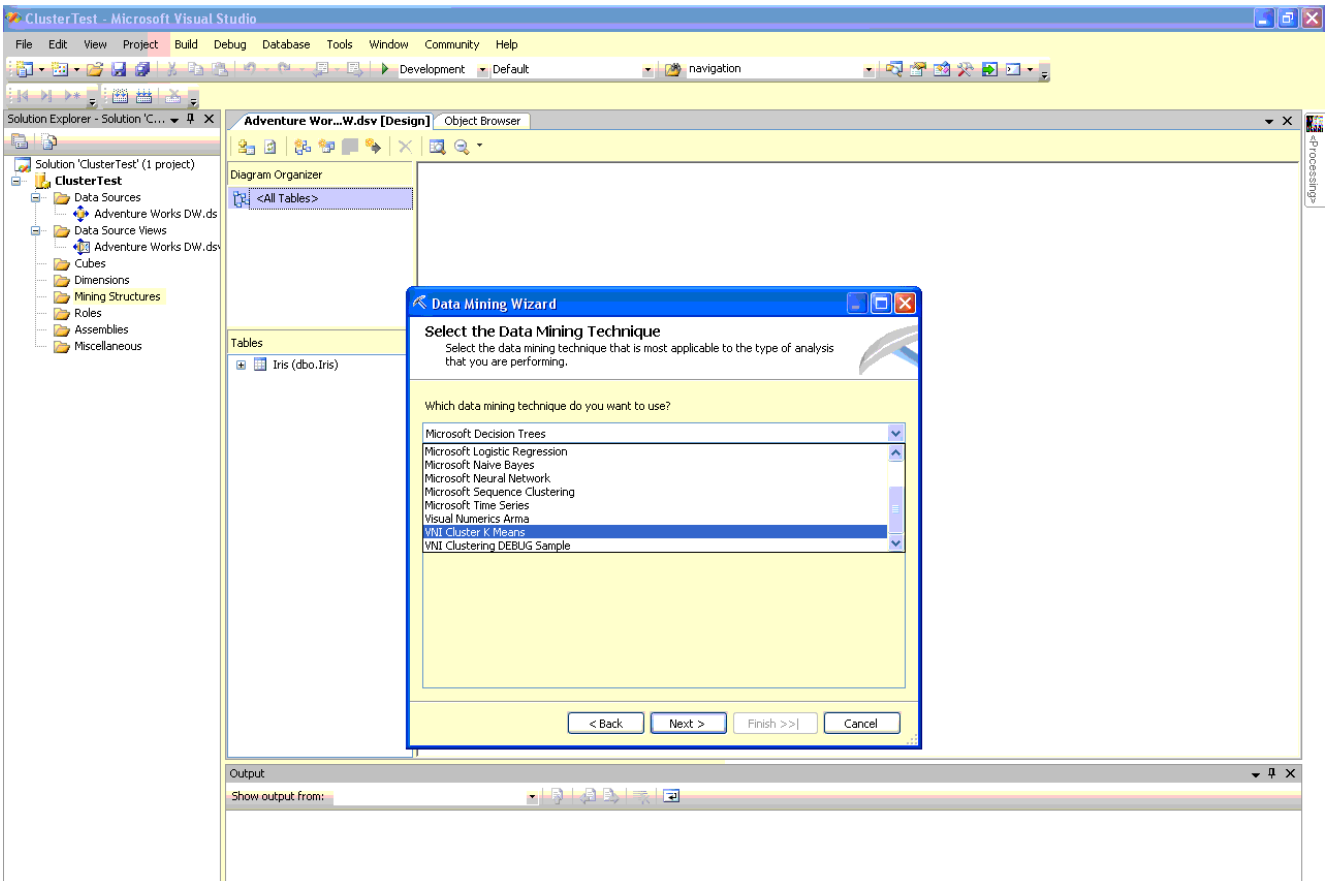


Figure 8. Data Mining Technique Selection Dialogue Box Showing VNI Cluster K-Means.

Follow the instructions presented by the Data Mining Wizard. Next, you will need to deploy the solution. After it is successfully deployed, you will be able to browse your model, view detected patterns and characteristics of each pattern, and check the accuracy of your model. Once the data mining developer is satisfied with the trained model, it can be used by clients (Excel) to find patterns and predict values using the trained model. The following figure displays the detected patterns.

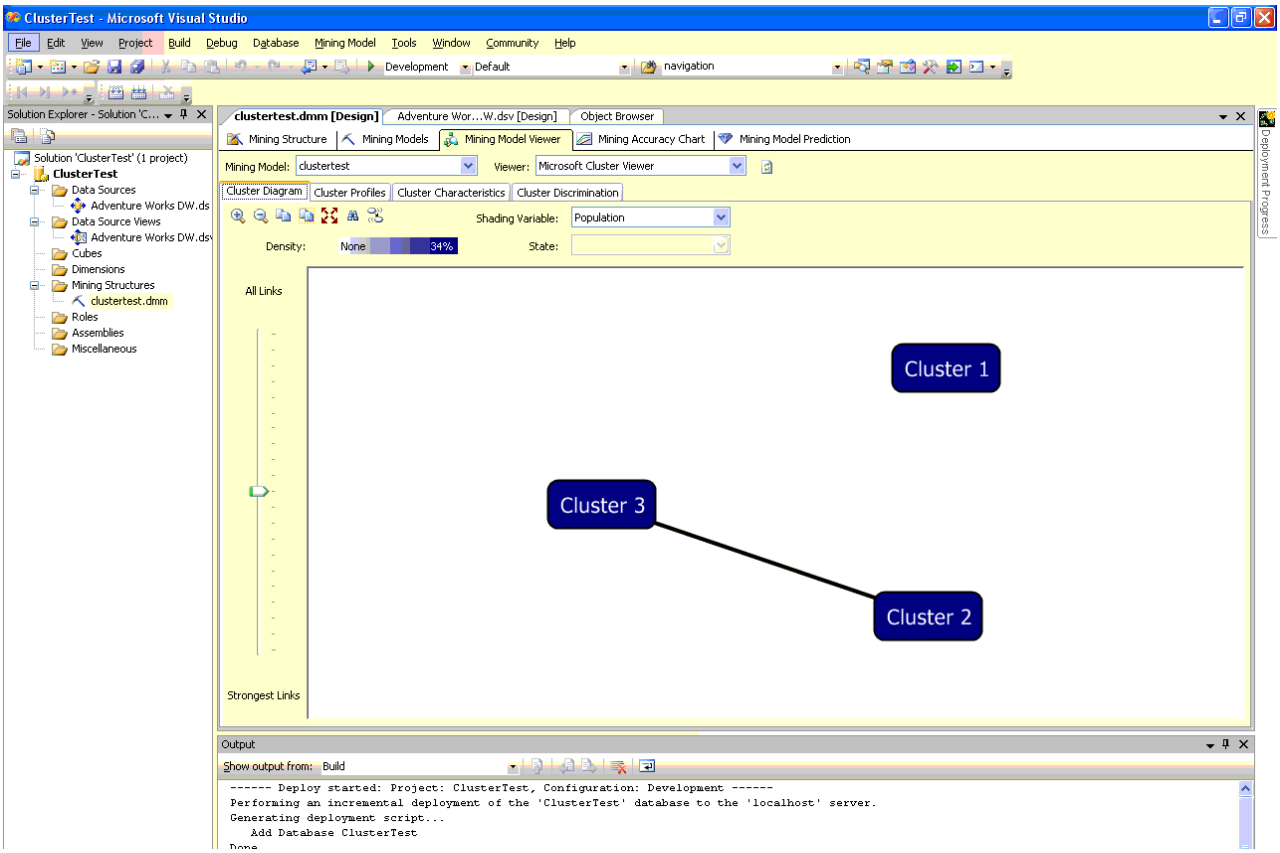
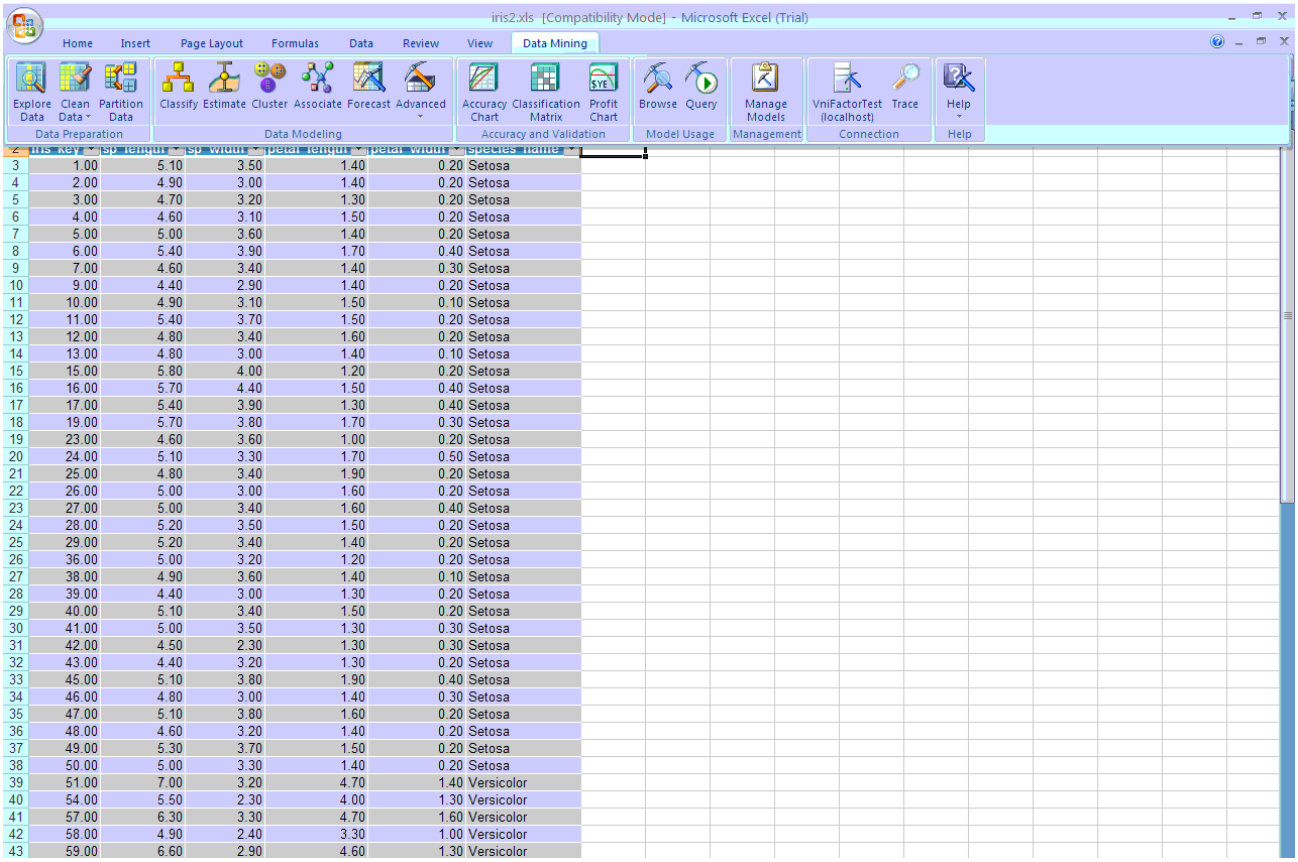


Figure 9. The Observed Patterns for Example

Excel 2007

The Data Mining add-ins for Excel 2007 allows users to either create a new model just like in the BI Development Studio or use an existing model that was created using the BI studio. The Data mining tab in Excel allows users to perform data preparation, data modeling, accuracy and validation, use existing model, and management. The following figure shows the data mining capabilities in Excel 2007.



	iris_key	sp_lenght	sp_width	petal_lenght	petal_width	species_name
3	1.00	5.10	3.50	1.40	0.20	Setosa
4	2.00	4.90	3.00	1.40	0.20	Setosa
5	3.00	4.70	3.20	1.30	0.20	Setosa
6	4.00	4.60	3.10	1.50	0.20	Setosa
7	5.00	5.00	3.60	1.40	0.20	Setosa
8	6.00	5.40	3.90	1.70	0.40	Setosa
9	7.00	4.60	3.40	1.40	0.30	Setosa
10	9.00	4.40	2.90	1.40	0.20	Setosa
11	10.00	4.90	3.10	1.50	0.10	Setosa
12	11.00	5.40	3.70	1.50	0.20	Setosa
13	12.00	4.80	3.40	1.60	0.20	Setosa
14	13.00	4.80	3.00	1.40	0.10	Setosa
15	15.00	5.80	4.00	1.20	0.20	Setosa
16	16.00	5.70	4.40	1.50	0.40	Setosa
17	17.00	5.40	3.90	1.30	0.40	Setosa
18	19.00	5.70	3.80	1.70	0.30	Setosa
19	23.00	4.60	3.60	1.00	0.20	Setosa
20	24.00	5.10	3.30	1.70	0.50	Setosa
21	25.00	4.80	3.40	1.90	0.20	Setosa
22	26.00	5.00	3.00	1.60	0.20	Setosa
23	27.00	5.00	3.40	1.60	0.40	Setosa
24	28.00	5.20	3.50	1.50	0.20	Setosa
25	29.00	5.20	3.40	1.40	0.20	Setosa
26	36.00	5.00	3.20	1.20	0.20	Setosa
27	38.00	4.90	3.60	1.40	0.10	Setosa
28	39.00	4.40	3.00	1.30	0.20	Setosa
29	40.00	5.10	3.40	1.50	0.20	Setosa
30	41.00	5.00	3.50	1.30	0.30	Setosa
31	42.00	4.50	2.30	1.30	0.30	Setosa
32	43.00	4.40	3.20	1.30	0.20	Setosa
33	45.00	5.10	3.80	1.90	0.40	Setosa
34	46.00	4.80	3.00	1.40	0.30	Setosa
35	47.00	5.10	3.80	1.60	0.20	Setosa
36	48.00	4.60	3.20	1.40	0.20	Setosa
37	49.00	5.30	3.70	1.50	0.20	Setosa
38	50.00	5.00	3.30	1.40	0.20	Setosa
39	51.00	7.00	3.20	4.70	1.40	Versicolor
40	54.00	5.50	2.30	4.00	1.30	Versicolor
41	57.00	6.30	3.30	4.70	1.60	Versicolor
42	58.00	4.90	2.40	3.30	1.00	Versicolor
43	59.00	6.60	2.90	4.60	1.30	Versicolor

Figure 10. Sample Data Loaded into Excel

Users can partition their Excel data into training and testing, create new models using a similar interface as in the BI studio, and use the testing data to query an existing model. For example, using the IMSL C# Library ClusterKMeans trained model with the test data on flower species, you can predict the species' name. The following figure shows the column mapping step in the Data Mining Query Wizard used to develop the query for predicting the flower species' name.

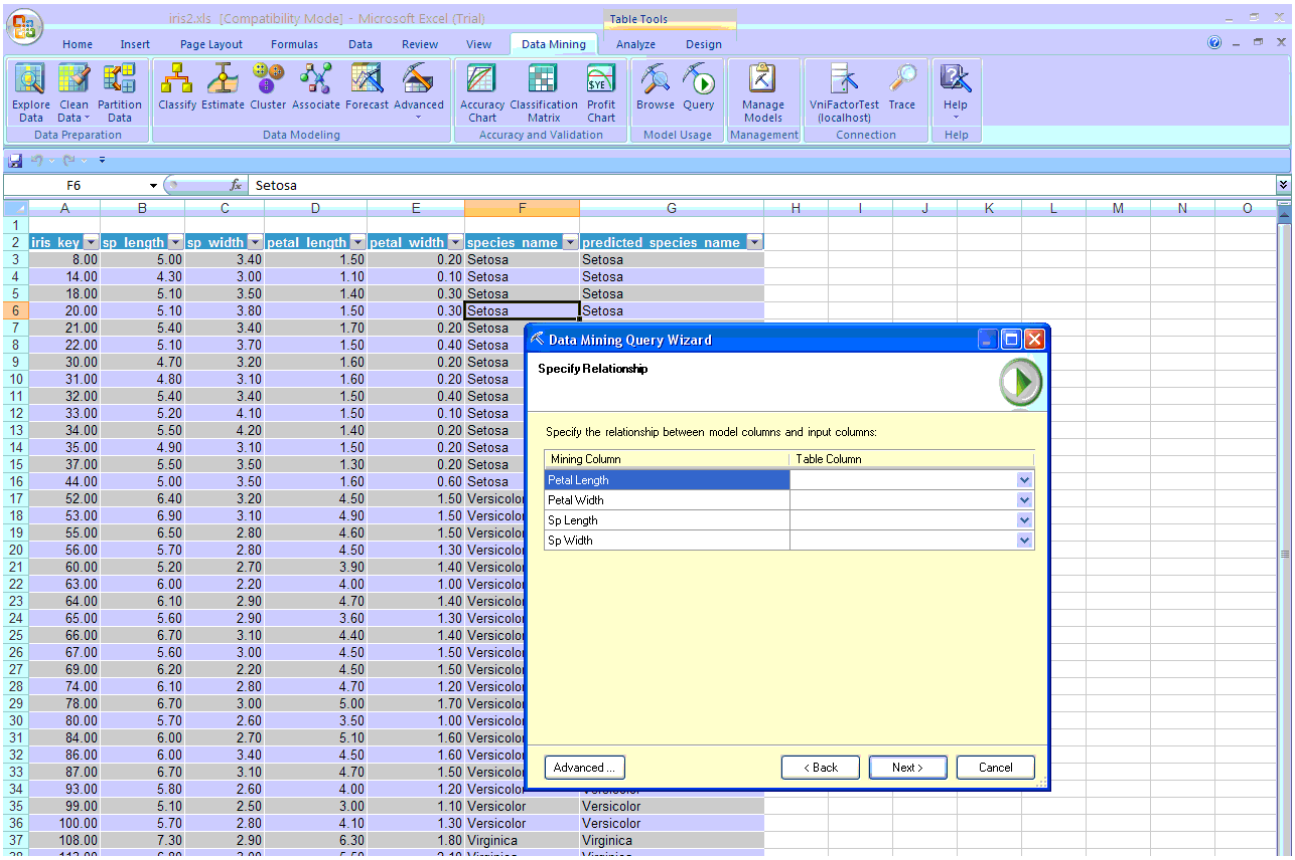


Figure 11. Data Mining Query Wizard Configuring Column Mapping.

Conclusion

The plug-in algorithm architecture in SQL Server 2005 Data Mining allows selected IMSL C# Library classes to take full advantage of the Microsoft BI platform (UDMs, enterprise solutions, etc.). Every IMSL C# Library routine that is a candidate for SQL Server Analysis Service integration will provide its own challenges, but the initial development should lend itself to reusable components that may be helpful in integrating other IMSL Library algorithms.

About the Author

Jasmit Singh is a Senior Consulting Engineering with Visual Numerics. Jasmit has worked at Visual Numerics since 2000 and has experience in areas ranging from C and Java programming to database and graphical programming. Prior to working with the Consulting Services group, Jasmit was a developer on the PV-WAVE product team. Originally from India and fluent in English and Hindi, Jasmit also has bachelor's degrees in Applied Mathematics and Computer Science from the University of Colorado, Boulder.

References

IMSL C # Numerical Library – Overview, technical documentation and evaluation CD available upon request.

Data Mining Managed Plug-in Algorithm API Tutorial[8] is a tutorial for constructing a managed plug-in algorithm.

Introduction to SQL Server 2005 Data Mining[9] is a brief introduction to Data Mining.

[8] http://www.sqlserverdatamining.com/ssdm/Default.aspx?tabid=94&Id=165
[9] http://technet.microsoft.com/en-us/library/ms345131.aspx

Appendix A: Code Files

VniClusterMetadata.cs

Expose the features of the ClusterKMeans algorithm

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Runtime.InteropServices;
using Microsoft.SqlServer.DataMining.PluginAlgorithms;
namespace VNI
{
    /* must create GUID number by executing
     * Tools->Create GUID and then use Copy and paste here
     * Only copy the unique number and disregard rest of the
     * numbers
    */
    [ComVisible(true)]
    [Guid("9BC1DB7D-52B9-46aa-9469-FF7B5A2B3F88")]
    [MiningAlgorithmClass(typeof(VniClusterKMeansAlgorithm))]
    public class VniClusterMetadata : AlgorithmMetadataBase
    {
        // Parameters
        protected MiningParameterCollection parameters;
        // modeling flag
        internal static MiningModelingFlag
                        MainAttributeFlag = MiningModelingFlag.CustomBase + 1;
        /* Paramater collection init */

        public VniClusterMetadata()
        {
            parameters = DeclareParameters();
        }
        static public MiningParameterCollection DeclareParameters()
        {
            MiningParameterCollection parameters
                    = new MiningParameterCollection();
            MiningParameter param;
```

```
        // Sample of completely populating a parameter in constructor
        param = new MiningParameter(
                                "CLUSTER_COUNT",
                                "Number of Clusters",
                                "3",
                                "(0.0, ...)",
                                true,
                                true,
                                typeof(System.Int32));
        parameters.Add(param);
        // Sample of populating a parameter after construction
        // When using this constructor, the following settings
        // are generated:
        //    - isRequired = false
        //    - isExposed = true
        //    - description = ""
        //    - defaultValue = ""
        //    - valueEnum = ""
        //parameters.Add(param);
        return parameters;
    }

    public override string GetDisplayName()
    {
        return "VNI Cluster K Means";
    }
    public override string GetServiceName()
    {
        return "VNI_ClusterKMeans";
    }
    public override string GetServiceDescription()
    {
        // Arma description
        return "computes K-means (centroid) Euclidean metric clusters for an input"+
               "data starting with initial estimates of the K cluster means.";
    }
    /* The service type enumeration value returned by this function describes the
     * class of algorithms that includes your algorithm, if any. For example, popular
     * classes of algorithms include Association Rules, Classification, and Clustering.
     * The sample returns ServiceTypeOther, because it does not really belong to any of
     * these classes.
```

```
 *
*/
public override PlugInServiceType GetServiceType()
{
    return PlugInServiceType.ServiceTypeClustering;
}
/* The viewer type string returned by this function indicates the tools which viewer
 * object should be instantiated to display the content of models trained with your
 * algorithm. If your algorithm content is similar to the content of built-in algorithms,
 * you can use one of the predefined (commented-out) strings. You can also build your own
 * custom viewer and return the identifier of that viewer. For details about how to do
 * this see “A tutorial for constructing a plug-in viewer”, at
 * http://msdn.microsoft.com/library/default.asp?url=/library/en-us/dnsql90/html/TutConPIV.asp
*/
public override string GetViewerType()
{
    //return MiningViewerType.MicrosoftAssociationRules;
    //return MiningViewerType.MicrosoftCluster;
    //return MiningViewerType.MicrosoftNaiveBayesian;
    //return MiningViewerType.MicrosoftNeuralNetwork;
    //return MiningViewerType.MicrosoftSequenceCluster;
    //return MiningViewerType.MicrosoftTimeSeries;
    //return MiningViewerType.MicrosoftTree;
    //return string.Empty;
    return MiningViewerType.MicrosoftCluster;
}
/* This is not used by the AS but exposed in the MINING_ALGORITHMS schema rowset */
public override MiningScaling GetScaling()
{
    return MiningScaling.Medium;
}
/* used by mining_algorithm schema rowset */
public override MiningTrainingComplexity GetTrainingComplexity()
{
    return MiningTrainingComplexity.Low;
}
public override MiningPredictionComplexity GetPredictionComplexity()
{
    return MiningPredictionComplexity.Low;
}
public override MiningExpectedQuality GetExpectedQuality()
```

```
{
    return MiningExpectedQuality.Low;
}
/* An algorithm supports data mining dimensions if the content of models trained
 * with that algorithm can be organized as a data mining dimension.
 * This sample returns false.
*/
public override bool GetSupportsDMDimensions()
{
    return false;
}
/* Support for drill-through operations is described in Section 10 of this document.*/
public override bool GetSupportsDrillThrough()
{
    return false;
}
public override bool GetDrillThroughMustIncludeChildren()
{
    return false;
}
/* Return true if your model is treating the case ID as a separate variable.*/
/* This sample returns false.*/
public override bool GetCaseIdModeled()
{
    return false;
}
/*
 * This informs the server of the statistics that need to be built before launching the
 * algorithm training. The MarginalRequirements enumeration fields may describe all statistics
 * (most common cases), statistics for input attributes only, for output attributes only, or no
 * statistics at all.
*/
public override MarginalRequirements GetMarginalRequirements()
{
    return MarginalRequirements.AllStats;
}
/*
 * This method returns the content types that are supported by this algorithm for input attributes.
 * All common types are supported by the managed plug-in.
*/
public override MiningColumnContent[] GetSupInputContentTypes()
```

```
{
    MiningColumnContent[] arInputContentTypes = new MiningColumnContent[]
     {
        MiningColumnContent.Discrete,
        MiningColumnContent.Continuous,
        MiningColumnContent.Discretized,
        MiningColumnContent.NestedTable,
        MiningColumnContent.Key
     };

    return arInputContentTypes;
}

/* This method returns the content types that are supported by this algorithm for
 * predictable attributes. All common types are supported by the managed plug-in.
*/
public override MiningColumnContent[] GetSupPredictContentTypes()
{
    MiningColumnContent[] arPredictContentTypes = new MiningColumnContent[]
     {
        MiningColumnContent.Discrete,
        MiningColumnContent.Continuous,
        MiningColumnContent.Discretized,
        MiningColumnContent.NestedTable,
        MiningColumnContent.Key
     };

    return arPredictContentTypes;
}
/* This method returns the list of standard Data Mining Extensions (DMX) functions
 * supported by this algorithm. Most standard functions can be supported without any
 * developer effort, once the AlgorithmBase.Predict function is implemented correctly.
*/
public override SupportedFunction[] GetSupportedStandardFunctions()
{
    SupportedFunction[] arFuncs = new SupportedFunction[] {
    // General prediction functions
    SupportedFunction.PredictSupport,
    SupportedFunction.PredictHistogram,
    SupportedFunction.PredictProbability,
    SupportedFunction.PredictAdjustedProbability,
```

```
        SupportedFunction.PredictAssociation,
        SupportedFunction.PredictStdDev,
        SupportedFunction.PredictVariance,
        SupportedFunction.RangeMax,
        SupportedFunction.RangeMid,
        SupportedFunction.RangeMin,
        SupportedFunction.DAdjustedProbability,
        SupportedFunction.DProbability,
        SupportedFunction.DStdDev,
        SupportedFunction.DSupport,
        SupportedFunction.DVariance,
        // content-related functions
        SupportedFunction.IsDescendent,
        SupportedFunction.PredictNodeId,
        SupportedFunction.IsInNode,
        SupportedFunction.DNodeId,
        // Cluster specific functions
        SupportedFunction.Cluster,
        SupportedFunction.ClusterDistance,
        SupportedFunction.ClusterPredictHistogram,
        SupportedFunction.ClusterProbability,
        SupportedFunction.PredictCaseLikelihood,
        SupportedFunction.DCluster,

        };

        return arFuncs;
    }


    /* This method performs a validation of the attribute set before training is launched.
     * For example, this method may ensure that at least one attribute is predictable, in
     * a classification algorithm.
    */
    public override void ValidateAttributeSet(AttributeSet attributeSet)
    {
        uint nCount = attributeSet.GetAttributeCount();

        int mainAttrs = 0;
        int inputAttrs = 0;
```

```
        for (uint nIndex = 0; nIndex < nCount; nIndex++)
        {
            bool thisAttIsInput = false;
            if ((attributeSet.GetAttributeFlags(nIndex) & AttributeFlags.Input) != 0)
            {
                inputAttrs++;
                thisAttIsInput = true;
            }

            MiningModelingFlag[] modelingFlags = attributeSet.GetModelingFlags(nIndex);
            for (int flagIndex = 0; flagIndex < modelingFlags.Length; flagIndex++)
            {
                if (modelingFlags[flagIndex] == MainAttributeFlag)
                {
                    if (!thisAttIsInput)
                    {
                        string strMessage = string.Format(
                                            "{0} can only be applied to an input attribute",
                                            GetModelingFlagName(MainAttributeFlag));
                        throw new System.Exception(strMessage);
                    }
                    mainAttrs++;
                }
            }
        }
    }
    public override AlgorithmBase CreateAlgorithm(ModelServices model)
    {
        return new VniClusterKMeansAlgorithm();
    }
    public override MiningParameterCollection GetParametersCollection()
    {
        if (parameters == null)
        {
            DeclareParameters();
        }
        return parameters;
    }
    public override object ParseParameterValue(
                                            int parameterIndex,
                                            string parameterValue)
```

```
{
    // This function should return an object containing the value of the parameter
    // NOTE!! the type of the object must exactly match the declared type of
    // parameter paramIndex
    object retVal = null;
    if (parameterIndex == 0)
    {
        // This is a value for PARAM1, which is Int32,
        // see DeclareParameters's implementation
        int dVal = System.Convert.ToInt32(parameterValue);
        retVal = dVal;
    }
    /* else if (parameterIndex == 1)
     {
         // This is a value for PARAM2, which is String,
         // see DeclareParameters's implementation
         string strVal = parameterValue;
         retVal = strVal;
     }*/
    else
    {
        throw new System.ArgumentOutOfRangeException("paramIndex");
    }
    return retVal;
}
/* Main atrribute flag or any custom flags*/
public override MiningModelingFlag[] GetSupModelingFlags()
{
    MiningModelingFlag[] arModelingFlags = new MiningModelingFlag[1];
    arModelingFlags[0] = MainAttributeFlag;
    //new MiningModelingFlag[] {
    //        MainAttributeFlag
    //    };

    return arModelingFlags;
}
/* name of teh main atrribute flag or any other custom name */
public override string GetModelingFlagName(MiningModelingFlag flag)
{
    if (flag == MainAttributeFlag)
    {
```

```
                return "VNI_MAIN";
            }
            else
            {
                throw new System.Exception("Unknown VNI modeling flag : " +
                flag.ToString());
            }
        }
    }
}
```

VniClusterKmeansAlgorithm.cs

This class implements Algorithm specific tasks.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SqlServer.DataMining.PluginAlgorithms;
using VNI;
using System.Diagnostics;
using Imsl.Stat;
using Imsl.Math;
using System.Collections;
/* The shell plug-in algorithm works in the following way:
 * • During training, it traverses all the cases once and sends progress notifications.
 * • The persisted content consists only of the number of cases and the time of processing.
 * This information does not constitute useful patterns, but it is a simple enough example
 * of how to use the persistence objects.
 * • The content has a single node, labeled “All”, which has the training set statistics
 * as node distribution.
 * • The prediction is ignoring the input and is based solely on the training set statistics.
*/
namespace VNI
{

    /// <summary>
    /// Persistence stuff
    /// </summary>
    enum VNIClusterPersistenceMarker
    {
```

```
        MainAttribute,
        Parameters,
        ClusterCount,
        ClusterDescription,
        ClusterDistribution

    }
    /// <summary>
    /// enumeration containing delimiters in
    /// the persisted content
    /// </summary>
    enum MyPersistenceTag
    {
        ShellAlgorithmContent,
        NumberOfCases
    };
    public class MyCaseProcessor : ICaseProcessor
    {
        protected VniClusterKMeansAlgorithm algo;

        public MyCaseProcessor(VniClusterKMeansAlgorithm algo)
        {
            this.algo = algo;
        }

        public void ProcessCase(long caseID, MiningCase inputCase)
        {
            // Check for cancel every 100 rows
            // Also, fire a progress notification every 100 rows, to avoid overloading the tracerowset
            if (caseID % 100 == 0)
            {
                algo.Context.CheckCancelled();
                algo.trainingProgress.Progress();
            }
            algo.trainingProgress.Current++;



            // This is the trivial clustering condition, see top of the file for
            // details
            //int destinationCluster = algo.InternalClusterMembership(inputCase);
```

```
            // Got the cluster membership
            switch (algo.ProcessingPhase)
            {
                case VniClusterKMeansAlgorithm.MainProcessingPhase:
                    algo.vniStore.addCase(inputCase);
                    //algo.Clusters[destinationCluster].PushCase(inputCase);
                    break;
                //case VniClusterKMeansAlgorithm.UpdateSupportPhase:
                    //algo.Clusters[destinationCluster].UpdateStats(inputCase);
                    //algo.vniStore.fillClusters(algo.Clusters);
                  //  break;
            }
        }
    }
    public class VniClusterKMeansAlgorithm : AlgorithmBase
    {
        // Mining parameters. Holds the training parameters
        // together with their values.
        protected MiningParameterCollection algorithmParams;

        // trace notifications during processing
        public TaskProgressNotification trainingProgress;

        // "Main" attribute (used in partitioning)
        protected System.UInt32 MainAttribute;
        protected double MainMean;  // mean of the main attribute, if continuous
        protected bool MainContinuous; // true if the main attribute is continous

        // Internal Clusters representation
        public InternalCluster[] Clusters;
        public int ProcessingPhase = 0;
        public const int MainProcessingPhase = 1;
        public const int UpdateSupportPhase = 2;
        public const int FinalPhase = 3;
        public VNIStore vniStore;
        // for right now, set it to 3 but pass in a paramter to determine number of clusters
        public int num_clusters = 0;


        public VniClusterKMeansAlgorithm()
```

```
{
    algorithmParams = VNI.VniClusterMetadata.DeclareParameters();
    MainAttribute = 0;
    MainContinuous = false;
    MainMean = 0.0;
    vniStore = new VNIStore(this);

}
// Optional override -- one does not HAVE TO override this
// The base.Initialize implementation does nothing, so it
// does not have to be invoked
protected override void Initialize()
{

    // Initialize the parameters with the default values
    this.algorithmParams["CLUSTER_COUNT"].Value = 3;

}

/*
    a.    The value specified by the user in deployment.
    b.    The default value (if none was specified by the user in training).
    c.    The best value automatically (heuristically) detected by the algorithm for
          the current training set.
*/
protected override object GetTrainingParameterActualValue(int paramOrdinal)
{
    return algorithmParams[paramOrdinal].Value;
}

public void ProcessCase(long caseId, MiningCase currentCase)
{
    // Make sure that the processing was not canceled
    this.Context.CheckCancelled();
    // increment the current value of the trace notification
    trainingProgress.Current++;

    if (caseId % 100 == 0)
    {
        // fire the trace every 100 cases, to avoid
        // performance impact
```

```
            trainingProgress.Progress();
        }
        // use the MiningCase here for actual training
    }
    /* Load/Save content is used for persistence of detected patterns */
    protected override void LoadContent(PersistenceReader reader)
    {
        // Load the main attribute
        reader.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.MainAttribute);
        reader.GetValue(out this.MainAttribute);
        reader.GetValue(out this.MainContinuous);
        reader.GetValue(out this.MainMean);
        reader.CloseScope();


        // Load the parameters
        reader.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.Parameters);
        foreach (MiningParameter param in this.algorithmParams)
        {
            string name;
            reader.GetValue(out name);
            if (name != param.Name)
            {
                throw new System.Exception("Corrupted file -- unrecognized parameter name : " + name);
            }

            if (param.Name == "CLUSTER_COUNT")
            {
                int dVal = 0;
                reader.GetValue(out dVal);
                param.Value = dVal;
            }

            /*if (param.Name == "PARAM2")
            {
                string sVal;
                reader.GetValue(out sVal);
                param.Value = sVal;
            }*/
        }
        reader.CloseScope();
```

```
        // Load the clusters
        reader.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterCount);
        int clusterCount = 0;
        reader.GetValue(out clusterCount);
        reader.CloseScope();


        Clusters = new InternalCluster[clusterCount];
        for (int nIndex = 0; nIndex < clusterCount; nIndex++)
        {
            Clusters[nIndex] = new InternalCluster(this);
            Clusters[nIndex].ClusterID = (ulong)nIndex;
            Clusters[nIndex].Description = BuildClusterDescription(nIndex);
            Clusters[nIndex].Load(ref reader);
        }
    }


    protected override void SaveContent(PersistenceWriter writer)
    {
        // Save the main attribute
        writer.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.MainAttribute);
        writer.SetValue(this.MainAttribute);
        writer.SetValue(this.MainContinuous);
        writer.SetValue(this.MainMean);
        writer.CloseScope();

        // Save the values of the known parameters
        writer.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.Parameters);
        foreach (MiningParameter param in this.algorithmParams)
        {
            writer.SetValue(param.Name);
            if (param.Name == "CLUSTER_COUNT")
            {
                int nVal = System.Convert.ToInt32(param.Value);
                writer.SetValue(nVal);
            }
        }
        writer.CloseScope();
```

```
        // Save the clusters
        writer.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterCount);
        writer.SetValue(Clusters.Length);
        writer.CloseScope();

        for (int iIndex = 0; iIndex < Clusters.Length; iIndex++)
        {
            Clusters[iIndex].Save(ref writer);
        }
    }
    protected override AlgorithmNavigationBase GetNavigator(
                                                bool forDMDimensionContent)
    {
        return new AlgorithmNavigator(this, forDMDimensionContent);
    }
    private void PrepareForProcessing(int numClusters)
    {
        /*//////////////////////////////////////////////////////////
         * Detect the main attribute
         * Look for the input attribute that has the MainAttributeFlag flag
        */
        UInt32 nAtt = 0;
        MainAttribute = AttributeSet.Unspecified;

        for (nAtt = 0; nAtt < this.AttributeSet.GetAttributeCount(); nAtt++)
        {
            MiningModelingFlag[] flags = this.AttributeSet.GetModelingFlags(nAtt);
            for (int flagIndex = 0; flagIndex < flags.Length; flagIndex++)
            {
                if (flags[flagIndex] == VniClusterMetadata.MainAttributeFlag)
                {
                    MainAttribute = nAtt;
                    Debug.Assert((AttributeSet.GetAttributeFlags(nAtt) & AttributeFlags.Input) != 0);
                    break;
                }
            }
        }

        if (MainAttribute == AttributeSet.Unspecified)
        {
```

```
            for (nAtt = 0; nAtt < this.AttributeSet.GetAttributeCount(); nAtt++)
            {
                if ((AttributeSet.GetAttributeFlags(nAtt) & AttributeFlags.Input) != 0)
                {
                    MainAttribute = nAtt;
                }
            }
        }

        Debug.Assert(MainAttribute != AttributeSet.Unspecified);
        MainContinuous = (AttributeSet.GetAttributeFlags(MainAttribute) & AttributeFlags.Continuous) !=
0;

        if (MainContinuous)
        {
            // Get the mean
            AttributeStatistics stats = this.MarginalStats.GetAttributeStats(MainAttribute);
            // Keep in mind that, for continuous attributes, the first state is missing and
            // the second state
            // contains the mean of the attribute
            Debug.Assert(stats.StateStatistics.Count == 2);
            Debug.Assert(stats.StateStatistics[1].Value.IsDouble);
            MainMean = stats.StateStatistics[1].Value.Double;
        }


        // Use the trainingParams and the marginal statistics here to infer the best number of clusters
        // This sample hard-codes this to 2
        Clusters = new InternalCluster[numClusters];

        for (int nIndex = 0; nIndex < numClusters; nIndex++)
        {
            // create the clusters
            Clusters[nIndex] = new InternalCluster(this);
            // set the internal node id property
            Clusters[nIndex].ClusterID = (ulong)nIndex;
            // Generally, the cluster should build its own description
            // In this case, the algorithm knows the main attribute, hence it
            // will build the description
            Clusters[nIndex].Description = BuildClusterDescription(nIndex);
        }
```

```
}
// Generally, the cluster should build it's own description
// In this case, the algorithm knows the main attribute, hence it will build the description
private string BuildClusterDescription(int nIndex)
{
    string strRet = string.Empty;

    //return "VNI Cluster " + nIndex.ToString();
    string attName = AttributeSet.GetAttributeDisplayName(MainAttribute, false);

    if (MainContinuous)
    {
        StateValue sVal = new StateValue();
        sVal.SetDouble(MainMean);
        object val = AttributeSet.UntokenizeAttributeValue(MainAttribute, sVal);
        if (nIndex == 0)
        {
            strRet = string.Format("{0} < {1}", attName, "99999");
        }
        else
        {
            strRet = string.Format("{0} >= {1} OR {0} = Missing", attName, val.ToString());
        }
    }
    else
    {
        StateValue sVal = new StateValue();
        sVal.SetIndex(1);
        object val = AttributeSet.UntokenizeAttributeValue(MainAttribute, sVal);
        if (nIndex == 0)
        {
            strRet = string.Format("{0} = {1}", attName, val.ToString());
        }
        else
        {
            strRet = string.Format("{0} NOT = {1}", attName, val.ToString());
        }
    }
    return strRet;
}
```

```
public int InternalClusterMembership(MiningCase mcase)
{
    /* check error to make mCase has same attributes as
     * trained cluster attributes
    */
    int member = -1;
    double[] varr = new double[this.AttributeSet.GetAttributeCount()];

    bool mcontinue = mcase.MoveFirst();
    while (mcontinue)
    {
        UInt32 attribute = mcase.Attribute;
        StateValue value = mcase.Value;
        if (value.IsDouble) /*continous */
        {
            varr[attribute] = value.Double;
            //attrList.Add(value.Double);
        }
        /* for every discrete column there will be a
         * index representing a state.  For example,
         * a column with values A,B,C will have 3 indices
         * A=1, B=2, c=3
         */
        if (value.IsIndex) /*discrete */
        {
            varr[attribute] = value.Double;
            //attrList.Add((double)value.Index);
        }
        if (value.IsMissing) /* missing values */
        {
            //attrList.Add(null);
        }
        mcontinue = mcase.MoveNext();
    }
    //double[] vals = (double[])attrList.ToArray(typeof(double));
    // use the Euclidean Distance to figure out tthe cluster
    // It is assumed that the input case has as many attributes
    // as the trained model.
    double [,] centers = this.vniStore.getCenters();
    double[] distance = new double[Clusters.Length];
    double esum = 0.0;
```

```
        for(int i = 0;i<distance.Length;i++)
        {
            esum = 0.0;
            for (int j = 0; j < varr.Length; j++)
            {
                esum += (varr[j] - centers[i, j]) * (varr[j] - centers[i, j]);
            }
            distance[i] = Math.Sqrt(esum);
        }
        double[] distcopy = new double[distance.Length];
        Array.Copy(distance, distcopy, distance.Length);
        Array.Sort(distcopy);
        for (int m = 0; m < distance.Length; m++)
        {
            if (distcopy[0] == distance[m])
            {
                member = m;
                break;
            }
        }
        return member;
    }
    /// <summary>
    /// Pseudo clustering method
    /// Returns 0 for the first cluster, 1 for the second
    /// </summary>
    /*public int InternalClusterMembership(MiningCase inputCase)
    {
        int nRet = 1;

        bool bContinue = inputCase.MoveFirst();
        while (bContinue)
        {
            if (inputCase.Attribute == MainAttribute)
            {
                if (MainContinuous)
                {
                    // Safety check
                    Debug.Assert(inputCase.Value.IsDouble || inputCase.Value.IsMissing);
                    if (inputCase.Value.IsDouble && (inputCase.Value.Double < MainMean))
                    {
```

```
                    // Belongs to the first cluster
                    nRet = 0;
                }
            }
            else
            {
                // Safety check
                Debug.Assert(inputCase.Value.IsIndex || inputCase.Value.IsMissing);
                if (inputCase.Value.IsIndex && (inputCase.Value.Index == 1))
                {
                    // Belongs to the first cluster
                    nRet = 0;
                }
            }
            break;
        }
        else
        {
            bContinue = inputCase.MoveNext();
        }
    }
    return nRet;
}*/
/* Begining of Case processing.  The PushCaseSet object allows us to interact
 * with CaseProcessor
*/
protected override void InsertCases(PushCaseSet caseSet, MiningParameterCollection trainingParams)
{
    // Initialize the internal cluster set
    // and the parameters
    LoadTrainingParameters(trainingParams);
    /* get the number of clusters specified by the user */
    num_clusters = (int) GetTrainingParameterActualValue(0);
    if (num_clusters == 0)
    {
        throw new System.ArgumentOutOfRangeException("num_clusters");
    }
    //prepare for processing (2 clusters)
    PrepareForProcessing(num_clusters);

    // switch to phase 1
```

```
ProcessingPhase = MainProcessingPhase;

// Main training loop
while (ProcessingPhase != FinalPhase)
{
    // Create a task progress  notification object, to send trace events
    trainingProgress = this.Model.CreateTaskNotification();
    trainingProgress.Total = (int)this.MarginalStats.GetTotalCasesCount();
    trainingProgress.Current = 0;
    switch (ProcessingPhase)
    {
        case MainProcessingPhase:
            trainingProgress.Format = "MainProcessingPhase: processing {0} out of {1}";
            bool bSuccess = true;
            try
            {
                trainingProgress.Start();

                MyCaseProcessor processor = new MyCaseProcessor(this);
                caseSet.StartCases(processor);
            }
            catch
            {
                bSuccess = false;
                throw;
            }
            finally
            {
                trainingProgress.End(bSuccess);
            }
            break;
        case UpdateSupportPhase:
            trainingProgress.Format = "Updating support: processing {0} out of {1}";
            this.vniStore.fillClusters(this.Clusters);
            break;
    }
    // Move to next processing phase
    ProcessingPhase++;
}

// Done with processing, call PostProcess on each cluster
```

```
        for (int nIndex = 0; nIndex < Clusters.Length; nIndex++)
        {
            Clusters[nIndex].UpdateStats();
        }
    }
    private void LoadTrainingParameters(MiningParameterCollection trainingParams)
    {
        // Copy the values of the parameters into this's collection of params
        foreach (MiningParameter param in trainingParams)
        {
            if (this.algorithmParams[param.Name] != null)
            {
                this.algorithmParams[param.Name].Value = param.Value;
            }
        }
    }
    protected override void Predict(MiningCase inputCase, PredictionResult predictionResult)
    {
        // Prediction means
        //  - determine the right cluster
        //  - perform cluster prediction
        int nCaseCluster = InternalClusterMembership(inputCase);
        Clusters[nCaseCluster].Predict(ref predictionResult);
    }
    protected override ClusterMembershipInfo[] ClusterMembership(
                        long caseID,
                        MiningCase inputCase,
                        string targetCluster)
    {
        // Fire a progress notification
        Model.EmitSingleTraceNotification("ClusterMembership ... ");

        int clIndex = InternalClusterMembership(inputCase);
        string caption = Clusters[clIndex].Caption;

        ClusterMembershipInfo[] ret = null;
        if (targetCluster.Length > 0)
        {

            int cltargetCluster = -1;
```

```
            for (int nIndex = 0; nIndex < Clusters.Length; nIndex++)
            {
                if (Clusters[nIndex].Caption.CompareTo(targetCluster) == 0)
                {
                    cltargetCluster = nIndex;
                    break;
                }
            }
            if (cltargetCluster == -1)
                return null;

            ret = new ClusterMembershipInfo[1];
            ret[0] = new ClusterMembershipInfo();
            ret[0].Caption = Clusters[cltargetCluster].Caption;
            ret[0].ClusterId = Clusters[cltargetCluster].ClusterID;
            ret[0].Distance = (cltargetCluster == clIndex) ? 0.0 : 1.0;
            ret[0].Membership = 1.0 - ret[0].Distance;
            ret[0].NodeUniqueName = Clusters[cltargetCluster].NodeUniqueName;
            return ret;
        }

        ret = new ClusterMembershipInfo[Clusters.Length];

        for (int nIndex = 0; nIndex < Clusters.Length; nIndex++)
        {
            ret[nIndex] = new ClusterMembershipInfo();
            ret[nIndex].Caption = Clusters[nIndex].Caption;
            ret[nIndex].ClusterId = Clusters[nIndex].ClusterID;
            ret[nIndex].Distance = (nIndex == clIndex) ? 0.0 : 1.0;
            ret[nIndex].Membership = 1.0 - ret[nIndex].Distance;
            ret[nIndex].NodeUniqueName = Clusters[nIndex].NodeUniqueName;
        }
        return ret;
    }

    protected override double CaseLikelihood(
                        long caseID, MiningCase inputCase, bool normalized)
    {
        // this sample does not compute the cluster distance,
        // so all cases are equally likely
        return 1.0;
```

```
        }
    }
}
```

AlgorithmNavigator.cs

Expose the patterns detected by the ClusterKMeans algorithm.

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.SqlServer.DataMining.PluginAlgorithms;
using VNI;
namespace VNI
{
    class AlgorithmNavigator : AlgorithmNavigationBase
    {
        VniClusterKMeansAlgorithm   algorithm;
        bool        forDMDimension;
        int         currentNode;

        public AlgorithmNavigator(VniClusterKMeansAlgorithm currentAlgorithm, bool dmDimension)
        {
            algorithm = currentAlgorithm;
            forDMDimension = dmDimension;
            currentNode = 0;
        }

        protected override bool MoveToNextTree()
        {
            // Single tree for this algorithm
            return false;
        }

        protected override int GetCurrentNodeId()
        {
            return currentNode;
        }

        protected override bool ValidateNodeId(int nodeId)
        {
```

```
        return (nodeId >= 0 && nodeId <= algorithm.Clusters.Length);
    }

    protected override bool LocateNode(int nodeId)
    {
        // The only valid node is 0
        if (!ValidateNodeId(nodeId) )
            return false;
        currentNode = nodeId;
        return true;
    }

    protected override int GetNodeIdFromUniqueName(string nodeUniqueName)
    {
        int nNode = System.Convert.ToInt32(nodeUniqueName);
        return nNode;
    }

    protected override string GetUniqueNameFromNodeId(int nodeId)
    {
        return nodeId.ToString("D3");
    }

    protected override uint GetParentCount()
    {
        switch (currentNode)
        {
            case 0:
                return 0;
            default:
                return 1;
        }
    }

    protected override void MoveToParent(uint parentIndex)
    {
        currentNode = 0;
    }

    protected override int GetParentNodeId(uint parentIndex)
    {
```

```
        return 0;
    }

    protected override uint GetChildrenCount()
    {
        switch (currentNode)
        {
            case 0:
                return (uint)algorithm.Clusters.Length;
            default:
                return 0;
        }
    }

    protected override void MoveToChild(uint childIndex)
    {
        if (currentNode == 0)
        {
            currentNode = (int)(childIndex + 1);
        }
    }

    protected override int GetChildNodeId(uint childIndex)
    {
        if (currentNode == 0)
        {
            return (int)(childIndex + 1);
        }
        return -1;
    }

    protected override NodeType GetNodeType()
    {
        // Root is Model, everything else is cluster
        if (currentNode == 0)
            return NodeType.Model;
        else
            return NodeType.Cluster;
    }

    protected override string GetNodeUniqueName()
```

```
{
    return GetUniqueNameFromNodeId(currentNode);
}

protected override uint[] GetNodeAttributes()
{
    // There is no association between a node and an attribute
    return null;// new uint[] { 1, 2 };
}

protected override double GetDoubleNodeProperty(NodeProperty property)
{
    double dRet = 0;

    double dTotalSupport = algorithm.MarginalStats.GetTotalCasesCount();
    double dNodeSupport = 0.0;
    switch (currentNode)
    {
        case 0:
            dNodeSupport = dTotalSupport;
            break;
        default:
            dNodeSupport = algorithm.Clusters[currentNode - 1].Support;
            break;
    }

    switch (property)
    {
        case NodeProperty.Support:
            dRet = dNodeSupport;
            break;
        case NodeProperty.Score:
            dRet = 0;
            break;
        case NodeProperty.Probability:
            dRet = dNodeSupport / dTotalSupport;
            break;
        case NodeProperty.MarginalProbability:
            dRet = dNodeSupport / dTotalSupport;
            break;
    }
```

```
        return dRet;
    }

    protected override string GetStringNodeProperty(NodeProperty property)
    {
        string strRet = "";

        switch (property)
        {
            case NodeProperty.Caption:
                {
                    // IMPORTANT: The caption of a node may be modified by admin
                    // with a statement like
                    // UPDATE Model.CONTENT SET NODE_CAPTION = 'Some cluster label'
                    // WHERE NODE_UNIQUE_NAME = '000001'
                    // The changes map is currently saved in the model, here is how to
                    // access it through the
                    // model services
                    strRet = algorithm.Model.FindNodeCaption(GetNodeUniqueName());
                    if (strRet.Length == 0)
                    {
                        // if empty, it was not found in the map
                        // generate the decsription
                        switch (currentNode)
                        {
                            case 0:
                                strRet = "All";
                                break;
                            default:
                                strRet = algorithm.Clusters[currentNode - 1].Caption;
                                break;
                        }
                    }
                }
                break;

            case NodeProperty.ConditionXml:
                // The condition for a case to fit into one node
                // should be represented here
                strRet = "";
                break;
```

```
                case NodeProperty.Description:
                    switch (currentNode)
                    {
                        case 0:
                            strRet = "All";
                            break;
                        default:
                            strRet = algorithm.Clusters[currentNode - 1].Description; break;
                    }
                    break;

                case NodeProperty.ModelColumnName:
                    strRet = "";
                    break;

                case NodeProperty.RuleXml:
                    switch (currentNode)
                    {
                        case 0: strRet = "<Rule>All</Rule>"; break;
                        default:
                            strRet = "<Cluster>" + algorithm.Clusters[currentNode - 1].Caption +
"</Cluster>";
                            break;
                    }
                    break;

                case NodeProperty.ShortCaption:
                    switch (currentNode)
                    {
                        case 0:
                            strRet = "All";
                            break;
                        default:
                            strRet = algorithm.Clusters[currentNode - 1].Caption;
                            break;
                    }
                    break;
            }
            return strRet;
        }
```

```
protected override AttributeStatistics[] GetNodeDistribution()
{
    switch (currentNode)
    {
        case 0:
            {
                // For the root node, return the marginal statistics of the whole mining model
                int attStats = (int)algorithm.AttributeSet.GetAttributeCount();
                AttributeStatistics[] marginalStats = new AttributeStatistics[attStats + 2];
                for (uint nIndex = 0; nIndex < attStats; nIndex++)
                {
                    marginalStats[nIndex] = algorithm.MarginalStats.GetAttributeStats(nIndex);
                }

                // Adding extra information in NODE_DISTRIBUTION, no string
                AttributeStatistics extraInfo = new AttributeStatistics();
                extraInfo.Attribute = AttributeSet.Unspecified;


                StateStatistics state = new StateStatistics();
                state.ValueType = MiningValueType.Intercept;
                state.Value.SetDouble(2.0);
                extraInfo.StateStatistics.Add(state);
                marginalStats[attStats] = extraInfo;

                // Adding extra information in NODE_DISTRIBUTION -- attribute value and
                // attribute name
                extraInfo = new AttributeStatistics();
                extraInfo.Attribute = AttributeSet.Unspecified;
                extraInfo.NodeId = "Any string here";
                state = new StateStatistics();
                state.ValueType = MiningValueType.Other;
                state.Value.SetIndex(124);
                extraInfo.StateStatistics.Add(state);
                marginalStats[attStats + 1] = extraInfo;

                return marginalStats;
            }
        default:
            // for the cluster nodes, return the distribution of the cluster
```

```
                    return algorithm.Clusters[currentNode - 1].Distribution;
                }
            }
        }
    }
```

Cluster.cs

An object used to represent the detected pattern (cluster).

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using Microsoft.SqlServer.DataMining.PluginAlgorithms;
using System.Collections;

namespace VNI
{
    // Internal Representation of a cluster
    // An instance of this class will represent a cluster detected by the plug-in algorithm.
    public class InternalCluster
    {

        private string nodeUniqueName;
        private string description;
        /* Each cluster will maintain the distribution of the attributes for all the
         * training cases that end up in that cluster.
        */
        private AttributeStatistics[] clusterDistribution;
        public VNIPatternAttribute[] vniatts;
        /* reference to the Algorithm object that detected this cluster */
        private VniClusterKMeansAlgorithm algo;

        // internal ID of the cluster
        private int clusterID;
        private int casesCount;
        ArrayList clusterValues;
        public InternalCluster(VniClusterKMeansAlgorithm parent)
        {
            algo = parent;
```

```
            // Allocate room for all the statistics
            // as well as for the cluster Prediction
            clusterDistribution = new AttributeStatistics[algo.AttributeSet.GetAttributeCount()];
            /* for each pattern we find in data there will be attributes belongning to that
             * patterns.  The VNIPatternattribute keeps track of each attribute in the pattern and
             * it's values and statistics
            */
            vniatts = new VNIPatternAttribute[algo.AttributeSet.GetAttributeCount()];
            for (uint nIndex = 0; nIndex < algo.AttributeSet.GetAttributeCount(); nIndex++)
            {

                ////////////////////////////////////
                // Distribution for this cluster
                clusterDistribution[nIndex] = new AttributeStatistics();

                vniatts[nIndex] = new VNIPatternAttribute();
                // determine the number of states
                uint statCount = algo.AttributeSet.GetAttributeStateCount(nIndex);

                // determine whether the attribute is continuous
                bool bContinuous = (algo.AttributeSet.GetAttributeFlags(nIndex) &
AttributeFlags.Continuous) != 0;

                clusterDistribution[nIndex].Attribute = nIndex;
                clusterDistribution[nIndex].Support = 0;
                clusterDistribution[nIndex].Min = 0.0;
                clusterDistribution[nIndex].Max = 0.0;
                clusterDistribution[nIndex].NodeId = string.Empty;
                clusterDistribution[nIndex].Probability = 0.0;


                for (int nStatIndex = 0; nStatIndex < statCount; nStatIndex++)
                {
                    StateStatistics stateStat = new StateStatistics();
                    if (nStatIndex == 0)
                        stateStat.Value.SetMissing();
                    else
                    {
                        if (bContinuous)
                        {
```

```
                    Debug.Assert(nStatIndex == 1);
                    stateStat.Value.SetDouble(0.0);
                }
                else
                    stateStat.Value.SetIndex((uint)nStatIndex);
            }

            stateStat.Probability = 0.0;
            stateStat.AdjustedProbability = 0.0;
            stateStat.ProbabilityVariance = 0.0;
            stateStat.Support = 0.0;
            stateStat.Variance = 0.0;

            clusterDistribution[nIndex].StateStatistics.Add(stateStat);
        }
    }
}
// Pushing cases into the cluster
// For discrete attributes, just increment the state support
// For continuous attributes, increment the state support and update Min and Max
// temporarily sum the values in the AttributeStatistics's Value field
public void PushCase(MiningCase inputCase)
{
    bool bContinue = inputCase.MoveFirst();
    casesCount++;

    while (bContinue)
    {
        UInt32 attribute = inputCase.Attribute;
        StateValue stateVal = inputCase.Value;
        AttributeStatistics attStat = this.clusterDistribution[attribute];

        bool bContinuous = (algo.AttributeSet.GetAttributeFlags(attribute) &
AttributeFlags.Continuous) != 0;

        if (bContinuous)
        {
            Debug.Assert(attStat.StateStatistics.Count == 2);
            // Continuous attribute
            bool first = attStat.StateStatistics[1].Support == 0.0;
```

```
            if (stateVal.IsMissing)
            {
                attStat.StateStatistics[0].Support += 1.0;
            }
            else
            {
                Debug.Assert(stateVal.IsDouble);
                double thisValue = stateVal.Double;
                double dSumSoFar = attStat.StateStatistics[1].Value.Double;
                // Increment the support for the non-missing state
                attStat.StateStatistics[1].Support += 1.0;
                attStat.StateStatistics[1].Value.SetDouble(dSumSoFar + thisValue);
                // The non-missing support for the attribute also gets incremented
                attStat.Support += 1.0;


                if (first)
                {
                    attStat.Min = thisValue;
                    attStat.Max = thisValue;
                }
                else
                {
                    if (attStat.Min > thisValue)
                        attStat.Min = thisValue;
                    if (attStat.Max < thisValue)
                        attStat.Max = thisValue;
                }
            }
        }
        else
        {
            // discrete attribute
            if (stateVal.IsMissing)
            {
                attStat.StateStatistics[0].Support += 1.0;
            }
            else
            {
                // Increment the support for the non-missing state
                Debug.Assert(stateVal.IsIndex);
```

```
                    attStat.StateStatistics[stateVal.Index].Support += 1.0;
                    // and also for the attribute
                    attStat.Support += 1.0;
                }
            }


            bContinue = inputCase.MoveNext();
        }
    }
    public void UpdateStats()
    {
        // determine the number of states

        //casesCount = algo.vniStore.getCaseCount();

        for (int i = 0; i < this.clusterDistribution.Length; i++)
        {
           uint statCount = algo.AttributeSet.GetAttributeStateCount((uint)i);
           AttributeStatistics attStat = this.clusterDistribution[i];
           bool bContinuous = (algo.AttributeSet.GetAttributeFlags((uint)i) &
AttributeFlags.Continuous) != 0;
           if (bContinuous)
           {
               casesCount = this.vniatts[i].getCount();
               attStat.StateStatistics[1].Support = 0.0;
               Debug.Assert(attStat.StateStatistics.Count == 2);
               double ExistingSupport = this.vniatts[i].getCount();
               attStat.StateStatistics[1].Support = ExistingSupport;
               /* sum of values in the cluster */
               attStat.StateStatistics[1].Value.SetDouble(vniatts[i].getSum());
               attStat.StateStatistics[1].Variance = vniatts[i].getVariance();

               attStat.Support = ExistingSupport;
               attStat.Min = vniatts[i].getMin();
               attStat.Max = vniatts[i].getMax();

                //double ExistingSupport = attStats.StateStatistics[1].Support;
                //double sumValues = attStats.StateStatistics[1].Value.Double;
                //double dExistingMiu = sumValues / this.casesCount;
                // Set the value for existing state. It is Miu (SUM/ExistingSupport)
```

```
                    attStat.StateStatistics[1].Value.SetDouble(vniatts[i].getSum() / ExistingSupport);


                    // Set Prob/AdjProb for existing state
                    attStat.StateStatistics[1].Probability = (ExistingSupport + 1.0) / (ExistingSupport +
attStat.StateStatistics.Count);
                    // smoothen the adjustProb
                    attStat.StateStatistics[1].AdjustedProbability =
attStat.StateStatistics[1].Probability;

                    // Set Prob/AdjProb for missing state ??
                    double MissingSupport = attStat.StateStatistics[0].Support;
                    attStat.StateStatistics[0].Probability = (MissingSupport + 1.0) / (ExistingSupport +
attStat.StateStatistics.Count);
                    // smoothen the adjustProb
                    attStat.StateStatistics[0].AdjustedProbability =
attStat.StateStatistics[0].Probability;

                    // Set Prob/AdjProb for the whole attribute
                    attStat.Probability = attStat.StateStatistics[1].Probability;
                    attStat.AdjustedProbability = attStat.StateStatistics[1].AdjustedProbability;
                }
                else   /* discrete */
                {
                    /* further sub divide the support according to discrete vars
                     * Red =1 , blue = 2, green =3.  Decide on how many reds, blues or
                     * greens there are in a cluster
                    */
                    ArrayList vals = this.vniatts[i].getDataValues();
                    int max = (int)vniatts[i].getMax();
                    /* discrete states start at 1 */
                    for (int k = 1; k <= max; k++)
                    {
                        /*loop through each vniatts values to set the support according to the value*/
                        foreach (Object attrobj in vals)
                        {
                            /* null means missing value */
                            if (attrobj != null)
                            {
                                if (k == (int)(double)attrobj)
                                {
```

```
                                    attStat.StateStatistics[(uint)k].Support += 1.0;
                                    attStat.Support += 1.0;
                                }

                            }
                        }
                    }
                    // discrete attribute, detect the most popular state and compute probabilities
                    double ExistingSupport = 0.0;
                    for (uint nStateIndex = 0; nStateIndex < statCount; nStateIndex++)
                    {
                        double dStateSupport = attStat.StateStatistics[nStateIndex].Support;
                        attStat.StateStatistics[nStateIndex].Probability = (dStateSupport + 1.0) /
(this.casesCount + statCount);
                        attStat.StateStatistics[nStateIndex].AdjustedProbability =
attStat.StateStatistics[nStateIndex].Probability;

                        if (nStateIndex > 0)
                            ExistingSupport += dStateSupport;
                    }
                    // set the attribute overall statistics
                    attStat.Probability = (ExistingSupport + statCount - 1.0) / (ExistingSupport +
statCount);
                    attStat.AdjustedProbability = attStat.Probability;
                }

            }



        }
        // Updating the statistics
        // Nothing to do for discrete or for Missing continuous
        // For continuous, need to compute the StdDev and Variance
        // Variance = SUM( Xi - Miu)^2 / N
        // We have SUM( Xi) in Value, hence Miu = Value/N
        // We'll increment here the Variance with (Xi - Miu)^2/N
        // also, we'll update the Value
        public void UpdateStats(MiningCase inputCase)
        {
            // Updating the statistics
```

```
            bool bContinue = inputCase.MoveFirst();

            while (bContinue)
            {
                UInt32 attribute = inputCase.Attribute;
                StateValue stateVal = inputCase.Value;
                AttributeStatistics attStat = this.clusterDistribution[attribute];

                bool bContinuous = (algo.AttributeSet.GetAttributeFlags(attribute) &
AttributeFlags.Continuous) != 0;

                if (bContinuous)
                {
                    if (!stateVal.IsMissing)
                    {
                        double ExistingSupport = attStat.StateStatistics[1].Support;
                        double Miu = attStat.StateStatistics[1].Value.Double / ExistingSupport;
                        double thisValue = stateVal.Double;

                        attStat.StateStatistics[1].Variance += ((thisValue - Miu) * (thisValue - Miu) /
ExistingSupport);
                    }
                }

                bContinue = inputCase.MoveNext();
            }
        }


        // Post processing the clusters
        // for continuous attributes:
        //      - missing state -- nothing  to do
        //      - non-missing state -- Value is currently SUM(Xi), divide by existing support to get Miu
        //   - decide the most likely state, missing or existing, for prediction
        //   - copy the existing probability, variace etc to the attribute statistics
        // for discrete attributes:
        //      - detect the most likely state for prediction
        //      - compute the attribute probability (ExistingSupport/NumCases)
        public void PostProcess()
        {
            for (uint nIndex = 0; nIndex < algo.AttributeSet.GetAttributeCount(); nIndex++)
```

```
            {
                // determine the number of states
                uint statCount = algo.AttributeSet.GetAttributeStateCount(nIndex);

                // determine whether the attribute is continuous
                bool bContinuous = (algo.AttributeSet.GetAttributeFlags(nIndex) &
AttributeFlags.Continuous) != 0;

                AttributeStatistics attStats = this.clusterDistribution[nIndex];
                if (bContinuous)
                {
                    double ExistingSupport = attStats.StateStatistics[1].Support;
                    double sumValues = attStats.StateStatistics[1].Value.Double;
                    double dExistingMiu = sumValues / this.casesCount;
                    // Set the value for existing state. It is Miu (SUM/ExistingSupport)
                    attStats.StateStatistics[1].Value.SetDouble(dExistingMiu);

                    // Set Prob/AdjProb for existing state
                    attStats.StateStatistics[1].Probability = (ExistingSupport + 1.0) / (this.casesCount +
attStats.StateStatistics.Count);
                    // smoothen the adjustProb
                    attStats.StateStatistics[1].AdjustedProbability =
attStats.StateStatistics[1].Probability;

                    // Set Prob/AdjProb for missing state
                    double MissingSupport = attStats.StateStatistics[0].Support;
                    attStats.StateStatistics[0].Probability = (MissingSupport + 1.0) / (this.casesCount +
attStats.StateStatistics.Count);
                    // smoothen the adjustProb
                    attStats.StateStatistics[0].AdjustedProbability =
attStats.StateStatistics[0].Probability;

                    // Set Prob/AdjProb for the whole attribute
                    attStats.Probability = attStats.StateStatistics[1].Probability;
                    attStats.AdjustedProbability = attStats.StateStatistics[1].AdjustedProbability;
                }
                else
                {
                    // discrete attribute, detect the most popular state and compute probabilities
                    double ExistingSupport = 0.0;
                    for (uint nStateIndex = 0; nStateIndex < statCount; nStateIndex++)
```

```
                    {
                        double dStateSupport = attStats.StateStatistics[nStateIndex].Support;
                        attStats.StateStatistics[nStateIndex].Probability = (dStateSupport + 1.0) /
(this.casesCount + statCount);
                        attStats.StateStatistics[nStateIndex].AdjustedProbability =
attStats.StateStatistics[nStateIndex].Probability;

                        if (nStateIndex > 0)
                            ExistingSupport += dStateSupport;
                    }

                    // set the attribute overall statistics
                    attStats.Probability = (ExistingSupport + statCount - 1.0) / (this.casesCount +
statCount);
                    attStats.AdjustedProbability = attStats.Probability;
                }
            }
        }
        public string NodeUniqueName
        {
            get
            {
                return nodeUniqueName;
            }
        }

        public ulong ClusterID
        {
            get
            {
                return (ulong)clusterID;
            }
            set
            {
                clusterID = (int)value;
                // Node Unique Name is 1-based, 0 is the root
                nodeUniqueName = (clusterID + 1).ToString("D3");
            }
        }

        public string Description
```

```
{
    get
    {
        return description;
    }
    set
    {
        description = value;
    }
}

public string Caption
{
    get
    {
        return "Cluster " + (clusterID + 1).ToString();
    }
}


public int Support
{
    get
    {
        return casesCount;
    }
}
public void Load(ref PersistenceReader reader)
{
    // Load cluster info
    reader.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterDescription);
    reader.GetValue(out nodeUniqueName);
    reader.GetValue(out description);
    reader.GetValue(out clusterID);
    reader.GetValue(out casesCount);
    int distLength = 0;
    reader.GetValue(out distLength);
    reader.CloseScope();

    clusterDistribution = new AttributeStatistics[distLength];
    for (int nIndex = 0; nIndex < distLength; nIndex++)
```

```
{
    // Save each dist
    reader.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterDistribution);
    clusterDistribution[nIndex] = new AttributeStatistics();
    AttributeStatistics attStats = clusterDistribution[nIndex];
    double dVal;
    uint uVal;
    reader.GetValue(out dVal); attStats.AdjustedProbability = dVal;
    reader.GetValue(out uVal); attStats.Attribute = uVal;
    reader.GetValue(out dVal); attStats.Max = dVal;
    reader.GetValue(out dVal); attStats.Min = dVal;
    reader.GetValue(out dVal); attStats.Probability = dVal;
    reader.GetValue(out dVal); attStats.Support = dVal;
    int statCount;
    reader.GetValue(out statCount);

    for (int nState = 0; nState < statCount; nState++)
    {
        StateStatistics stateStat = new StateStatistics();
        reader.GetValue(out dVal); stateStat.AdjustedProbability = dVal;
        reader.GetValue(out dVal); stateStat.Probability = dVal;
        reader.GetValue(out dVal); stateStat.ProbabilityVariance = dVal;
        reader.GetValue(out dVal); stateStat.Support = dVal;
        reader.GetValue(out dVal); stateStat.Variance = dVal;
        bool bIsMissing = false;
        reader.GetValue(out bIsMissing);
        if (bIsMissing)
        {
            stateStat.Value.SetMissing();
        }
        else
        {
            bool bIsIndex = false;
            reader.GetValue(out bIsIndex);
            if (bIsIndex)
            {
                uint indexVal;
                reader.GetValue(out indexVal);
                stateStat.Value.SetIndex(indexVal);
            }
            else
```

```
                    {
                        double dblVal;
                        reader.GetValue(out dblVal);
                        stateStat.Value.SetDouble(dblVal);
                    }
                }
                attStats.StateStatistics.Add(stateStat);
            }
        }
    }

    public void Save(ref PersistenceWriter writer)
    {
        // Save cluster info
        writer.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterDescription);
        writer.SetValue(nodeUniqueName);
        writer.SetValue(description);
        writer.SetValue(clusterID);
        writer.SetValue(casesCount);
        writer.SetValue(clusterDistribution.Length);
        writer.CloseScope();

        for (int nIndex = 0; nIndex < clusterDistribution.Length; nIndex++)
        {
            // Save each dist
            writer.OpenScope((PersistItemTag)VNIClusterPersistenceMarker.ClusterDistribution);
            AttributeStatistics attStats = clusterDistribution[nIndex];
            writer.SetValue(attStats.AdjustedProbability);
            writer.SetValue(attStats.Attribute);
            writer.SetValue(attStats.Max);
            writer.SetValue(attStats.Min);
            writer.SetValue(attStats.Probability);
            writer.SetValue(attStats.Support);
            writer.SetValue(attStats.StateStatistics.Count);

            for (int nState = 0; nState < attStats.StateStatistics.Count; nState++)
            {
                StateStatistics stateStat = attStats.StateStatistics[(uint)nState];
                writer.SetValue(stateStat.AdjustedProbability);
                writer.SetValue(stateStat.Probability);
                writer.SetValue(stateStat.ProbabilityVariance);
```

```
                writer.SetValue(stateStat.Support);
                writer.SetValue(stateStat.Variance);
                writer.SetValue(stateStat.Value.IsMissing);
                if (!stateStat.Value.IsMissing)
                {
                    writer.SetValue(stateStat.Value.IsIndex);

                    if (stateStat.Value.IsIndex)
                    {
                        writer.SetValue(stateStat.Value.Index);
                    }
                    else
                    {
                        writer.SetValue(stateStat.Value.Double);
                    }
                }
            }
        }
    }

    // Predict -- returns the most likely prediction in this cluster
    public void Predict(ref PredictionResult predictionResult)
    {
        // predictionResult contains the predictin options and
        // should be filled with the predicted values/stats
        AttributeGroup outputAttrs = predictionResult.OutputAttributes;
        outputAttrs.Reset();
        uint nAtt = AttributeSet.Unspecified;

        while (outputAttrs.Next(out nAtt))
        {
            // Periodically check whether the processing was cancelled
            algo.Context.CheckCancelled();

            // Build the prediction
            AttributeStatistics attStats = new AttributeStatistics();
            if (predictionResult.IncludeNodeId)
            {
                attStats.NodeId = this.NodeUniqueName;
            }
```

```
            attStats.Attribute = nAtt;
            attStats.Min = clusterDistribution[nAtt].Min;
            attStats.Max = clusterDistribution[nAtt].Max;
            attStats.Support = clusterDistribution[nAtt].Support;
            attStats.Probability = clusterDistribution[nAtt].Probability;
            attStats.AdjustedProbability = clusterDistribution[nAtt].AdjustedProbability;

            uint nStatesCount = (uint)clusterDistribution[nAtt].StateStatistics.Count;
            for (uint index = 0; index < nStatesCount; index++)
            {
                StateStatistics clusterStateStat = clusterDistribution[nAtt].StateStatistics[index];
                StateStatistics stateStat = new StateStatistics();

                stateStat.AdjustedProbability = clusterStateStat.AdjustedProbability;
                stateStat.Probability = clusterStateStat.Probability;
                stateStat.Support = clusterStateStat.Support;
                stateStat.Variance = clusterStateStat.Variance;
                stateStat.ProbabilityVariance = clusterStateStat.ProbabilityVariance;
                stateStat.Value = clusterStateStat.Value;

                attStats.StateStatistics.Add(stateStat);
            }

            predictionResult.AddPrediction(attStats);
        }

    }
    public AttributeStatistics[] Distribution
    {
        get
        {
            return clusterDistribution;
        }
    }
    public void addValues(ArrayList values)
    {
        clusterValues = values;
    }
    public ArrayList getValues()
    {
        return clusterValues;
```

```
        }
        public VNIPatternAttribute[] getVNIAtts()
        {
            return vniatts;
        }
    }
}
```

VniStore.cs

This class helps in data translations between Analysis services and IMSL cluster K Means routine.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Diagnostics;
using Microsoft.SqlServer.DataMining.PluginAlgorithms;
using System.Collections;
using Imsl.Stat;
using Imsl.Math;
namespace VNI
{
    /* This is a helper class that will assist in data translation between
     * Analysis services and IMSL C# libraries
    */
    public class VNIStore
    {
        private ArrayList caseList;
        /* reference to the Algorithm object that detected this cluster */
        private VniClusterKMeansAlgorithm algo;
        private ClusterKMeans kmean;
        private double[,] cases;
        private double[,] centers;
        public VNIStore(VniClusterKMeansAlgorithm parent)
        {
            caseList = new ArrayList();
            algo = parent;
        }
        /* function to execute.  This will depend on user
         * selection from the available algoritm list
        */
```

```
private void execute(String function,int cluster_count)
{
    int m = 0;
    int seeds_inc = caseList.Count / cluster_count;
    ArrayList list = translateData(0);
    cases = (double[,])list[0];
    double[,] cluster_seeds = new double[3,((double [])caseList[0]).Length];
    for (int i = 0; i < cluster_count; i++)
    {
        for (int j = 0; j < ((double [])caseList[0]).Length; j++)
        {
            cluster_seeds[m, j] = cases[i*seeds_inc, j];
        }
        m++;
    }
    kmean = new ClusterKMeans(cases, cluster_seeds);
    // translate data to what is expected by function
    // Initially, we will use ClusterKMeans.
}
public void addCase(MiningCase mcase)
{
    ArrayList attrList = new ArrayList();
    double[] varr = new double[algo.AttributeSet.GetAttributeCount()];
    bool mcontinue = mcase.MoveFirst();
    while (mcontinue)
    {
        /* use attribute to index into correct values*/
        UInt32 attribute = mcase.Attribute;
        StateValue value = mcase.Value;
        if (value.IsDouble) /*continous */
        {
            varr[attribute] = value.Double;
           //attrList.Add(value.Double) ;
        }
        /* for every discrete column there will be a
         * index representing a state.  For example,
         * a column with values A,B,C will have 3 indices
         * A=1, B=2, c=3
         */
        if (value.IsIndex) /*discrete */
        {
```

```
                varr[attribute] = value.Index;
                //attrList.Add((double)value.Index);
            }
            if (value.IsMissing) /* missing values */
            {
                //varr[attribute] = ;
                //attrList.Add(null);
            }
            mcontinue = mcase.MoveNext();
        }
        caseList.Add(varr);
    }
    /* translates the ArrayList of inputcases into arrays
     * for structures for IMSL c# routine.
     * Returns: an Arraylist of one element that contains the
     * array/object that is to be used by the C# routine.
     * 0 - use the caseList to figure out the array dimesion
     * 1-8 - use the caselist and make it into dimesions varying from
     *       one through 8
     * 9 -   use it for special data.
     *
     *
    */
    private ArrayList translateData(int dim)
    {
        switch (dim)
        {
            case 0:
                return getArrayFromCaseList();
                //break;
            case 1:
            case 2:
            case 3:
            case 4:
            case 5:
            case 6:
            case 7:
            case 8:
            case 9:
               break;
        }
```

```
        return null;
    }

    private ArrayList getArrayFromCaseList()
    {
        int rows = caseList.Count;
        if (caseList.Count == 0) return null;
        double[] attrlist;
        /* check the first element.  It should be an another arraylist
         * with size equal to number of attributes in the MiningCase
         * In other words, if table has 10 rows and 5 columns
         * then this array must have 5 elements.
         *
        */
        /* for right now, create a 2d array in this code
         * but we should have objects that convert this datalist to
         * 2D,3D,structure,etc that is required by CNL routine.  May be
         * a parent class that deal with main conversation and then
         * some subclasses that perform task specific conversions
        */
        double[,] data = new double[caseList.Count, ((double [])caseList[0]).Length];
        for(int i = 0;i<caseList.Count;i++){
            attrlist = (double[])caseList[i];
            for (int j = 0; j < attrlist.Length;j++)
            {
                /* null means missing value */
                data[i, j] = (double)attrlist[j];
            }
        }
        ArrayList rlist = new ArrayList();
        rlist.Add(data);
        return rlist;
    }
    public void fillClusters(InternalCluster[] clusters)
    {
        execute("ClusterKMeans",algo.num_clusters);
        centers = kmean.Compute();
        int[] cmember = kmean.GetClusterMembership();
        int[] nc = kmean.GetClusterCounts();

        // filter out cluster values for each cluster
```

```
            // basically setting up patterns with initial values
            // it will be used to set up attribute statistics that is used
            // in the prediction.
            for(int i = 0; i <= nc.Length ; i++)
            {
                // [] indices = new int[nc[i]];
                //int m = 0;
                for(int j = 0; j < cmember.Length ; j++){
                  if(cmember[j] == i+1){
                      double [] data = (double[])caseList[j];
                      /* add values for each attribute */
                      for (int m = 0; m < data.Length;m++)
                      {
                          clusters[i].vniatts[m].addDataValues(data[m]);
                          clusters[i].vniatts[m].setCount(nc[i]);
                      }
                  }
                }
            }
            /* set up statistics for each cluster according to attributes */
            for(int i = 0; i < nc.Length ; i++)
            {

            }

        }
        public double[,] getCenters()
        {
            return centers;
        }

        public int getCaseCount()
        {
            return caseList.Count;
        }


    }
}
```

VniPatternAttribute.cs

This class is used to represent an attribute in the detected pattern. A pattern may consist of one or multiple attributes.

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Collections;
namespace VNI
{
    /* Microsoft has the concept of Case. For example a table from a DB is a case.  The record in the
     * Case or table are called attribute set.  Each column in Case or table is called attribute.  In Data
     * mining, the task is to find patterns in your data.  A pattern is made up of attribute set.
     * For example,
     * in cluster analysis we might find 3 clusters and each cluster will have different set of attributes.
     * For each attribute in the pattern, we need to set up some basic statistics (min, max, variance,
     * etc).
     * This class will keep track of the basic statistics
    */
    public class VNIPatternAttribute
    {
        ArrayList dataValues;
        int count = 0;
        public VNIPatternAttribute()
        {
            dataValues = new ArrayList();
        }
        public double getMin()
        {
            if (dataValues.Count > 0)
            {
                double[] vals = (double[])dataValues.ToArray(typeof(double));
                Array.Sort(vals);
                return vals[0];
            }
            return 0;
        }
        public double getSum()
        {
            double sum = 0.0;
            foreach (Object attrobj in dataValues)
```

```
        {
            /* null means missing value */
            if (attrobj != null)
            {
                sum += (double)attrobj;
            }
        }
        return sum;

    }
    public double getMax()
    {
        if (dataValues.Count > 0)
        {
            double[] vals = (double[])dataValues.ToArray(typeof(double));
            Array.Sort(vals);
            return vals[vals.Length-1];
        }
        return 0;
    }
    public double getVariance()
    {
        double variance = 0.0;
        if (getCount() == 0)
        {
            return 0;
        }
        double ExistingSupport = getCount();
        double Miu = this.getSum() / ExistingSupport;

        foreach (Object attrobj in dataValues)
        {
            /* null means missing value */
            if (attrobj != null)
            {
                double thisValue = (double)attrobj;
                variance += (thisValue - Miu) * (thisValue - Miu);
            }
        }
        return variance / ExistingSupport;
    }
```

```
        public void addDataValues(double value)
        {
            dataValues.Add(value);
        }
        public int getCount()
        {
            return count;
        }
        public void setCount(int count)
        {
            this.count = count;
        }
        public ArrayList getDataValues()
        {
            return dataValues;
        }
    }
}
```