EXTREME COMPUTING GROUP

Defining the future.

Microsoft

MapReduce on Windows Azure

Daytona – Outlier Detection algorithm user guide

Table of Contents

1 Implementing Outlier Detection using Daytona 3

2 Attribute Value Frequency technique 3

2.1 Algorithm steps for Daytona 4

2.1.1 Step 1 – Calculate Column Level Frequency 4

2.1.2 Step 2 - Calculate Row Level Frequency 5

2.2 Implementation Details....................5

2.2.1 Daytona Assemblies....................5

2.2.2 Controller 5

2.2.3 Step #1 - Combiner and Reducer 12

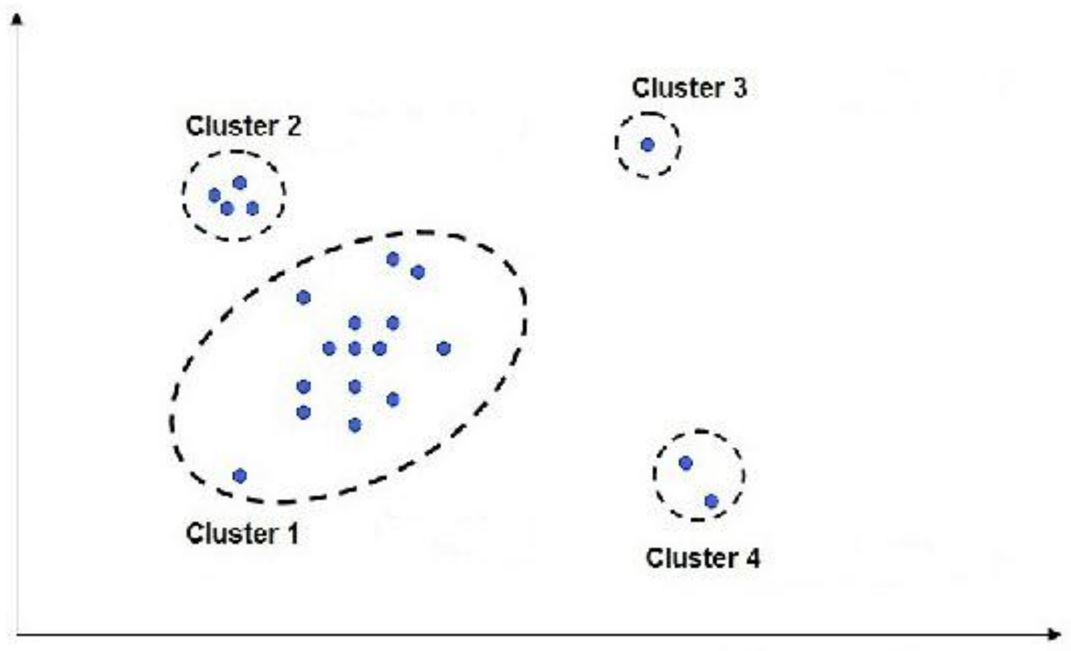
2.2.4 Step #2 - Mapper 13

2.2.5 Step #2 - Reducer and Combiner 19

2.3 Running the Sample 21

1 IMPLEMENTING OUTLIER DETECTION USING DAYTONA

Outlier detection approaches concentrate on detecting patterns that occur infrequently in the dataset, in contrast to traditional data mining techniques that attempt to find patterns that occur frequently in the data. Application examples where the discovery of outliers is useful include identifying irregular credit card transactions, indicating potential credit card fraud, or patients who exhibit abnormal symptoms due to their suffering from a specific disease or ailment. Outliers can occur by chance in any distribution, but they are often indicative *either* of measurement error or that the population has a heavy-tailed distribution. In the former case one wishes to discard them or use statistics that are robust to outliers, while in the latter case they indicate that the distribution has high kurtosis and that one should be very cautious in using tools or intuitions that assume a normal distribution.



In most larger samplings of data, some data points will be further away from the sample mean than what is deemed reasonable. This can be due to incidental systematic error or flaws in the theory that generated an assumed family of probability distributions, or it may be that some observations are far from the center of the data. Outlier points can therefore indicate faulty data, erroneous procedures, or areas where a certain theory might not be valid. However, in large samples, a small number of outliers are to be expected (and not due to any anomalous condition).

2 ATTRIBUTE VALUE FREQUENCY TECHNIQUE

The attribute value frequency (AVF) algorithm is a simple and fast approach to detect outliers in categorical data, which minimizes the scans over the data, without the need to create or search through different combinations of attribute values or itemsets. It is intuitive that outliers are those points which are infrequent in the dataset is one whose each and every value is extremely irregular (or infrequent). The infrequent-ness of an attribute value can be measured by computing the number of times this value is assumed by the corresponding attribute in the dataset.

Let's assume that there are n points in the dataset, x_i, i=1...n, and each data point has m attributes. We can write xi = [x_{i1},..., x_{il}, ..., x_{im}], where x_{il} is the value of the l-th attribute of xi. Following the reasoning given above, the AVF score below is a good indicator of deciding of whether xi is an outlier:

$$\text{AVF Score } (X_i) \; = \frac{1}{m} \sum_{l=1}^{m} f(x_{il}) \qquad (1)$$

Where f(x_{il}) is the number of times the l-th attribute value of x_i appears in the dataset. A lower AVF score means that it is more likely that the point is an outlier. Since (1) is essentially a sum of m positive numbers, the AVF score is minimal when each of the sum's terms is individually minimized. Thus, the 'ideal' outlier as defined above will have the minimum AVF score. The minimum score will be achieved when every value in the data point occurs just once. Once AVF score is calculated for all the points, the k outliers returned are the points with the smallest AVF scores.

AVF approach is inherently easy to parallelize using the MapReduce paradigm and proves to be a high speed, scalable method for categorical data.

The paper which describes this technique and the algorithm in depth can be found here.

2.1 ALGORITHM STEPS FOR DAYTONA

MapReduce AVF technique is relatively simple to develop using Daytona and highly scalable with respect to the number of cluster nodes. Given a dataset in CSV format, the algorithm functionality can be broken into the following high-level steps:

1. The original AVF algorithm calculates the AVF over each input record independently, making it amenable to easy parallelization. If the AVF can be expressed in terms of the MapReduce model, then the algorithm can have the benefits of automatic load balancing and fault tolerance, with no additional effort from the user's perspective.
2. Outliers in the input dataset are found by using Attribute Value Frequency AVF.
3. It is implemented as a two step process. In step 1, frequency of the values in the respective columns specified by the user is calculated. In step 2, AVF value for each and every row is calculated and then the rows (number of outliers i.e. rows is taken as an input parameter from the user) with the least AVF value is returned as output.
4. User should give names of the columns which will be used to calculate AVF value based on which outliers are determined. The columns can contain any kind of data.

2.1.1 STEP 1 – CALCULATE COLUMN LEVEL FREQUENCY

1. Daytona runtime controller splits the input dataset into multiple partitions. For each partition, a map task will be created.
2. The map task (Mapper) will read each record in the partition, output all the attributes individually.
    a. Key – Index and value of the column separated by underscore.
    b. Value – Integer 1.

3. The output of map task are sent to combiner which will merge the duplicate keys (if any) produced by mapper and computes the sum of the values.
4. The output of combiner is partitioned using the formula 'HashCode(Key) % NoOfReduceTasks' and sent to the corresponding reduce task.
5. The reduce task (reducer) again merges the duplicate keys (if any), computes the sum of values and produce one key-value pair.
    a. Key - Index and value of the column separated by underscore.
    b. Value – Integer representing the frequency of occurrence of each attribute in its respective column.
6. Reducer output is uploaded to a blob file.

2.1.2 STEP 2 - CALCULATE ROW LEVEL FREQUENCY

1. Controller splits the input dataset into multiple partitions. For each partition, a map task will be created.
2. The map task (mapper) will read each record in the partition, compute the AVF value.
    a. Key – Integer 1.
    b. Value – An object of AttributeValueFrequencyRecord class which has input data row and its AVF value.
3. The output of map task are sent to combiner which will merge the duplicate keys (if any) produced by mapper and sends only those many number of rows specified by user with least AVF value.
4. The output of combiner is partitioned using the formula 'HashCode(Key) % NoOfReduceTasks' and sent to the corresponding reduce task.
5. The reduce task (reducer) again merges the duplicate keys (if any), sends only those many number of rows specified by user with least AVF value and produce one key-value pair.
    a. Key – Input datarow.
    b. Value – AVF value of the row.

2.2 IMPLEMENTATION DETAILS

2.2.1 Daytona Assemblies

The sample makes use of the following assemblies.

Assembly	***Purpose***
Research.MapReduce.Core.dll	Contains the base classes, interfaces for implementing the different components (mapper, reducer, writer etc) required to run a job and API for creating and submitting jobs.
Research.MapReduce.Library.dll	Contains commonly used implementation of data partitioning, key partitioning and output writing.

2.2.2 Controller

Controller is responsible for validating the input parameters, formulating the job definitions and submitting them for execution.

2.2.2.1 PARAMETER VALIDATION

Parameter	*Parameter Type*	*Description*	*Requirement*	*Validation*
StorageConnectionString	string	Connection string required to connect to the Azure storage used for input/output.	Mandatory	Valid azure storage connection string format.
InputDataLocation	string	URI pointing to the blob containin the input data in CSV format.	Mandatory	Valid blob URI format. Should exist in the storage account specified by the parameter 'StorageConnectionString'.
OutputDataLocation	string	Name of the blob container to which datarows with AVF value will be written to.	Mandatory	1. Valid blob container name. 2. If it already exists, then should be accessible so that it can cleaned during initialization.
RequestedNumberOfMappers	int	The number of mappers the caller has requested to use for running a job.	Mandatory	Should be greater than zero.
RequestedNumberOfReducers	int	The number of Reducers the caller has requested to use for running a job.	Mandatory	Should be greater than zero.
ColumnNames	string	The names of the columns (in CSV format) whose values are to be used to calculate AVF value.	Mandatory	Should be greater than zero.
NumberOfOutliers	string	Number of outliers i.e. output datarows	Mandatory	Should be greater than zero.

2.2.2.2

2.2.2.3 JOB DEFINITION

The following code snippet shows how the outlier detection controller creates a job definition.

```
var jobConfiguration = new JobConfiguration();

var dataPartitioner = new BlobCsvPartitioner(CloudClient, inputBlob.Uri.ToString(),
RequestedNumberOfMappers);



jobConfiguration.MapperType = typeof(FindAttributesCountMapper);
```

```
//// Combiner implementation is the same as Reducer.

jobConfiguration.CombinerType = typeof(FindAttributesCountCombiner);

jobConfiguration.MapOutputStorage = MapOutputStoreType.Local;

jobConfiguration.KeyPartitioner = typeof(HashModuloKeyPartitioner<string, int>);

jobConfiguration.ReducerType = typeof(FindAttributesCountReducer);


jobConfiguration.ExceptionPolicy = new TerminateOnFirstException();

jobConfiguration.JobTimeout = TimeSpan.FromMinutes(10);


jobConfiguration.JobParameters.Add("CsvHeader", dataPartitioner.Header);

jobConfiguration.JobParameters.Add("ColumnNames", this.ColumnNames);


var job = new Job(jobConfiguration, this);

job.DataPartitioner = dataPartitioner;

job.RecordWriter = new BlobWriter<string, int>

{

CloudClient = this.CloudClient,

ContainerName = outputBlobContainerName,

KeyValueDelimiter = ","

};

job.NoOfReduceTasks = this.RequestedNumberOfReducers;


return job;
```

```
var jobConfiguration = new JobConfiguration();

var dataPartitioner = new BlobCsvPartitioner(CloudClient, inputBlob.Uri.ToString(),
RequestedNumberOfMappers);


jobConfiguration.MapperType = typeof(FindOutliersMapper);

//// Combiner implementation is the same as Reducer.

jobConfiguration.CombinerType = typeof(FindOutliersCombiner);

jobConfiguration.MapOutputStorage = MapOutputStoreType.Local;
jobConfiguration.KeyPartitioner = typeof(HashModuloKeyPartitioner<int, AttributeValueFrequencyRecord>);

jobConfiguration.ReducerType = typeof(FindOutliersReducer);



jobConfiguration.ExceptionPolicy = new TerminateOnFirstException();

jobConfiguration.JobTimeout = TimeSpan.FromMinutes(10);



jobConfiguration.JobParameters.Add("CsvHeader", dataPartitioner.Header);

jobConfiguration.JobParameters.Add("ColumnNames", this.ColumnNames);

jobConfiguration.JobParameters.Add("NumberOfOutliers",
this.NumberOfOutliers.ToString(CultureInfo.InvariantCulture));

jobConfiguration.JobParameters.Add("StorageConnectionString", this.StorageConnectionString);

jobConfiguration.JobParameters.Add("FindAttributesCountOutputContainerName",
findAttributeCountOutputContainerName);



var job = new Job(jobConfiguration, this);

job.DataPartitioner = dataPartitioner;

job.RecordWriter = new BlobWriter<string, string>

{

CloudClient = this.CloudClient,

ContainerName = outputBlobContainerName,

IgnoreKey = true,

ColumnHeaders = dataPartitioner.Header + ",AVF Score"

};

job.NoOfReduceTasks = this.RequestedNumberOfReducers;



return job;
```

‘BlobCsvPartitioner’ is used to split the input CSV data equivalently as per the number of mappers specified by the user.

Column header of the input file and column names specified by the user is passed as parameters to the first job. Column header of the input file, column names & number of outliers specified by the user, storage connection string and output container of step 1 is passed as parameters to the second job.

2.2.2.4 JOBS SUBMISSION AND FINAL RESULT

Following code snippet shows how the outlier detection controller submits the jobs for execution.

```
var findAttributeCountOutputContainerName = "find-attribute-count-" + Guid.NewGuid().ToString("N");

Helper.GetInitializedContainer(this.CloudClient, findAttributeCountOutputContainerName);

var findAttributeCountJob = this.GetJobForFindAttributeCount(inputBlob,
findAttributeCountOutputContainerName);

var task = Task.Factory.StartNew(() => findAttributeCountJob.Run());



task.Wait();



var findOutliersJob = this.GetJobForFindOutliers(inputBlob, outputBlobContainer.Name,
findAttributeCountOutputContainerName);

findOutliersJob.Run();

this.Results.Add("OutputContainer", outputBlobContainer.Name);
```

2.2.2.5 STEP #1 - MAPPER

Following code snippet shows the FindAttributesCountMapper.

```
    /// <summary>

    /// Split the attributes.

    /// </summary>

    public sealed class FindAttributesCountMapper : IMapper<int, string, string, int>

    {

        /// <summary>

        /// Used to store the indexes of the data columns in the input file.

        /// </summary>

        private int[] columnIndexes;



        /// <summary>

        /// Used to store the number of user specified columns.

        /// </summary>

        private int numberOfUserSpecifiedColumns;
        /// <summary>
        /// Configure Mapper.
        /// </summary>
        /// <param name="context">MapContext to get the user inputs.</param>
        public void Configure(MapContext<int, string> context)
        {
            if (context == null)
            {
                throw new ArgumentNullException("context", Resources.MapContextCannotBeNull);
            }

            if (string.IsNullOrWhiteSpace(context.JobParameters["CsvHeader"]))
            {
                throw new ArgumentNullException(Resources.CsvHeaderCannotBeEmpty);
            }

            if (string.IsNullOrWhiteSpace(context.JobParameters["ColumnNames"]))
            {
                throw new ArgumentNullException(Resources.ColumnNamesCannotBeEmpty);
            }

            var columnNames = context.JobParameters["ColumnNames"].Split(',').Select(x =>
x.Trim().ToUpperInvariant()).ToArray();
            this.numberOfUserSpecifiedColumns = columnNames.Length;

            // Get all the column headers.
            var metadata = context.JobParameters["CsvHeader"];
            var columnHeaders = metadata.Split(',').Select(x => x.Trim().ToUpperInvariant()).ToArray();
            this.columnIndexes = new int[this.numberOfUserSpecifiedColumns];
            int columnIndex;
            var columnsNotFound = new StringBuilder();
            for (int index = 0; index < this.numberOfUserSpecifiedColumns; index++)
            {
                columnIndex = Array.IndexOf(columnHeaders, columnNames[index]);
                if (columnIndex >= 0)
                {
                    this.columnIndexes[index] = columnIndex;
                }
                else
                {
                    columnsNotFound.Append(columnNames[index]);
                    columnsNotFound.Append(", ");
                }
            }

            if (columnsNotFound.Length > 0)
            {
                columnsNotFound.Remove(columnsNotFound.Length - 2, 2);
                throw new ArgumentException(columnsNotFound + " could not be found.");
            }
        }

        /// <summary>
        /// Map method sends the attribute value in each user selected column with count as 1.
        /// </summary>
        /// <param name="key">Integer key.</param>
        /// <param name="value">Comma separated column values.</param>
        /// <param name="context">MapContext to get the user inputs.</param>
        /// <returns>Filtered datarows.</returns>
        public IEnumerable<KeyValuePair<string, int>> Map(int key, string value, MapContext<int, string>
context)
        {
            var stringvalues = value.Split(',');
            int columnIndex;
            string stringvalue;


            for (int index = 0; index < this.numberOfUserSpecifiedColumns; index++)

            {

                columnIndex = this.columnIndexes[index];

                stringvalue = stringvalues[columnIndex];

                yield return new KeyValuePair<string, int>(index.ToString() + "_" +
stringvalue.ToUpperInvariant(), 1);

            }

        }

    }
```

The mapper retrieves and saves the input parameters during configuration.

For each attribute in the input data record specified by the user(‘value’), the mapper returns a output. Attribute value and integer 1 are returned as key and value respectively.

2.2.3 STEP #1 - COMBINER AND REDUCER

Following code snippet shows the FindAttributesCountReducer which also serves as the combiner.

```
    /// <summary>

    /// Finds the frequency of attributes of all the partitions.

    /// </summary>

    public sealed class FindAttributesCountReducer : IReducer<string, int, string, int>

    {

        /// <summary>

        /// Configure Reducer.

        /// </summary>

        /// <param name="context">ReduceContext to get the user inputs.</param>

        public void Configure(ReduceContext<string, int> context)

        {

            ////No configuration is required.

        }
        /// <summary>
        /// No records are filtered in this method.
        /// </summary>
        /// <param name="key">Integer key.</param>
        /// <param name="values">Values with the same key.</param>
        /// <param name="context">ReduceContext to get the user inputs.</param>
        /// <returns>output of match filter operator.</returns>
        public IEnumerable<KeyValuePair<string, int>> Reduce(string key, IEnumerable<int> values,
ReduceContext<string, int> context)
        {
            yield return new KeyValuePair<string, int>(key, values.Sum());
        }
    }
```

Both Combiner and Reducer sums the values. This gives us the frequency of each and every attribute in its respective column. In case of combiner, it will give the frequency for a particular partition. In case of reducer, it will give the frequency for the whole input dataset i.e. for all the partitions. They return key and sum as the output. Using these frequencies, AVF value for all the rows is calculated in the second step and the rows with least AVF values are considered as outliers.

2.2.4 STEP #2 - MAPPER

Following code snippet shows the FindOutliersMapper.

```
    /// <summary>
    /// Calculate AVF value.
    /// </summary>
    public class FindOutliersMapper : IMapper<int, string, int, AttributeValueFrequencyRecord>
    {
        /// <summary>
        /// Used to store the indexes of the data columns in the input file.
        /// </summary>
        private int[] columnIndexes;


        /// <summary>
        /// Used to store the number of user specified columns.
        /// </summary>
        private int numberOfUserSpecifiedColumns;

        /// <summary>
        /// Used to store the count of each attribute value.
        /// </summary>
        private Dictionary<string, int> frequencies = new Dictionary<string, int>();

        /// <summary>
        /// Configure Mapper.
        /// </summary>
        /// <param name="context">MapContext to get the user inputs.</param>
        public void Configure(MapContext<int, string> context)
        {
            if (context == null)
            {
                throw new ArgumentNullException("context", Resources.MapContextCannotBeNull);
            }

            if (string.IsNullOrWhiteSpace(context.JobParameters["CsvHeader"]))
            {
                throw new ArgumentNullException(Resources.CsvHeaderCannotBeEmpty);
            }

            if (string.IsNullOrWhiteSpace(context.JobParameters["ColumnNames"]))
            {
                throw new ArgumentNullException(Resources.ColumnNamesCannotBeEmpty);
            }

            if (string.IsNullOrWhiteSpace(context.JobParameters["StorageConnectionString"]))
            {
                throw new ArgumentNullException(Resources.StorageConnectionStringCannotBeEmpty);
            }
            if
(string.IsNullOrWhiteSpace(context.JobParameters["FindAttributesCountOutputContainerName"]))

            {

                throw new
ArgumentNullException(Resources.FindAttributesCountOutputContainerNameCannotBeEmpty);

            }


            // Get all the column headers.

            var metadata = context.JobParameters["CsvHeader"];

            var columnHeaders = metadata.Split(',').Select(x => x.Trim().ToUpperInvariant()).ToArray();


            var columnNames = context.JobParameters["ColumnNames"].Split(',').Select(x =>
x.Trim().ToUpperInvariant()).ToArray();

            this.numberOfUserSpecifiedColumns = columnNames.Length;

            this.columnIndexes = new int[this.numberOfUserSpecifiedColumns];

            int columnIndex;

            var columnsNotFound = new StringBuilder();


            for (int index = 0; index < this.numberOfUserSpecifiedColumns; index++)

            {

                columnIndex = Array.IndexOf(columnHeaders, columnNames[index]);

                if (columnIndex >= 0)

                {

                    this.columnIndexes[index] = columnIndex;

                }

                else

                {

                    columnsNotFound.Append(columnNames[index]);

                    columnsNotFound.Append(", ");

                }

            }


            if (columnsNotFound.Length > 0)
            {
                columnsNotFound.Remove(columnsNotFound.Length - 2, 2);
                throw new ArgumentException(columnsNotFound + " could not be found.");
            }

            var storageConnectionString = context.JobParameters["StorageConnectionString"];
            var findAttributesCountOutputContainerName =
context.JobParameters["FindAttributesCountOutputContainerName"];

            CloudClient cloudClient = null;

            if (!string.IsNullOrWhiteSpace(storageConnectionString))
            {
                cloudClient = new CloudClient(storageConnectionString);
            }

            var findAttributesCountOutputContainer =
cloudClient.BlobClient.GetContainerReference(findAttributesCountOutputContainerName);
            try
            {
                findAttributesCountOutputContainer.FetchAttributes();
            }
            catch (StorageClientException e)
            {
                if (e.ErrorCode == StorageErrorCode.ResourceNotFound)
                {
                    throw new ArgumentException(Resources.OutputBlobContainerInvalidErrorMessage);
                }
                else
                {
                    throw;
                }
            }
            this.DownloadAttributesCount(findAttributesCountOutputContainer);

        }



        /// <summary>

        /// Map method to filter datarows based on the value of a column using regular expression.

        /// </summary>

        /// <param name="key">Integer key.</param>

        /// <param name="value">Comma separated column values.</param>

        /// <param name="context">MapContext to get the user inputs.</param>

        /// <returns>Filtered datarows.</returns>

        public IEnumerable<KeyValuePair<int, AttributeValueFrequencyRecord>> Map(int key, string value,
MapContext<int, string> context)

        {

            var stringvalues = value.Split(',');

            int columnIndex;

            string stringvalue;

            int avf = 0;



            for (int index = 0; index < this.numberOfUserSpecifiedColumns; index++)

            {

                columnIndex = this.columnIndexes[index];

                stringvalue = stringvalues[columnIndex];

                avf += this.frequencies[index.ToString() + "_" + stringvalue.ToUpperInvariant()];

            }



            yield return new KeyValuePair<int, AttributeValueFrequencyRecord>(1, new
AttributeValueFrequencyRecord(value, avf));

        }



        /// <summary>

        /// Used to attributes with their frequency of occurence.

        /// </summary>

        /// <param name="line">ColumnIndex (separated by _) AttributeValue (separated by ,)
Frequency.</param>
```

```
        /// <param name="property">ColumnIndex (separated by _) AttributeValue</param>
        /// <param name="frequency">Frequency of occurence.</param>
        private static void GetFrequencies(string line, out string property, out int frequency)
        {
            string[] parts = line.Split(',');
            property = parts[0];
            frequency = Int32.Parse(parts[1], CultureInfo.InvariantCulture);
        }

        /// <summary>
        /// Download centroids from blob container and add them to canopyCentres variable.
        /// </summary>
        /// <param name="findAttributesCountOutputContainer">Output blob container of phase 1.</param>
        private void DownloadAttributesCount(CloudBlobContainer findAttributesCountOutputContainer)
        {
            var findAttributesCountBlobs = findAttributesCountOutputContainer.ListBlobs();
            var findAttributesCountBlobsCount = findAttributesCountBlobs.Count();
            string property;
            int frequency;

            for (int blobIndex = 0; blobIndex < findAttributesCountBlobsCount; blobIndex++)
            {
                var blob = (CloudBlockBlob)findAttributesCountBlobs.ElementAt(blobIndex);
                using (TextReader tr = new StreamReader(blob.OpenRead()))
                {
                    string line;
                    while ((line = tr.ReadLine()) != null)
                    {
                        GetFrequencies(line, out property, out frequency);
                        if (this.frequencies.ContainsKey(property))
                        {
                            this.frequencies[property] += frequency;
                        }
                        else
                        {
                            this.frequencies[property] = frequency;
                        }
                    }
                }
            }
        }
    }
```

The mapper retrieves and saves the input parameters during configuration.

For each input record (‘value’), the mapper calculates AVF value. Integer 1 and an object of AttributeValueFrequencyRecord which consists of input row and AVF value are returned as key and value respectively.

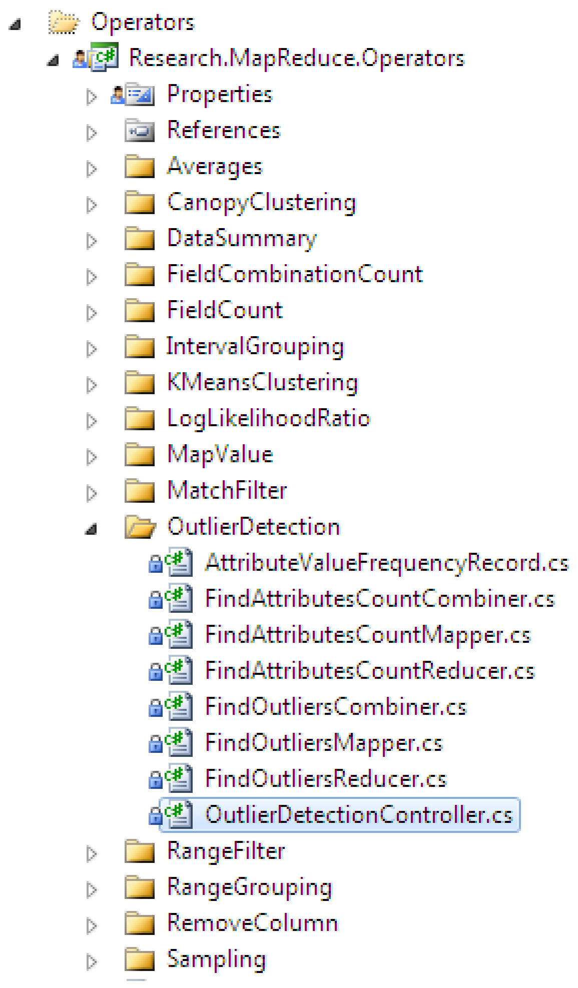
2.2.5 STEP #2 - REDUCER AND COMBINER

Following code snippet shows the FindOutliersReducer which also serves as the combiner.

```
    /// <summary>
    /// Return the final outliers.
    /// </summary>
    public sealed class FindOutliersReducer : IReducer<int, AttributeValueFrequencyRecord, string, string>
    {
        /// <summary>
        /// Number of outliers need to be displayed in the output.
        /// </summary>
        private int numberOfOutliers;

        /// <summary>
        /// Configure Reducer.
        /// </summary>
        /// <param name="context">ReduceContext to get the user inputs.</param>
        public void Configure(ReduceContext<int, AttributeValueFrequencyRecord> context)
        {
            if (context == null)
            {
                throw new ArgumentNullException("context", Resources.MapContextCannotBeNull);
            }

            if (!int.TryParse(context.JobParameters["NumberOfOutliers"], out this.numberOfOutliers) ||
this.numberOfOutliers <= 0)
            {
                throw new ArgumentException(Resources.NumberOfOutliersShouldBeGreaterThanZero);
            }
        }

        /// <summary>
        /// No records are filtered in this method.
        /// </summary>
        /// <param name="key">Integer key.</param>
        /// <param name="values">Values with the same key.</param>
        /// <param name="context">ReduceContext to get the user inputs.</param>
        /// <returns>output of match filter operator.</returns>
        public IEnumerable<KeyValuePair<string, string>> Reduce(int key,
IEnumerable<AttributeValueFrequencyRecord> values, ReduceContext<int, AttributeValueFrequencyRecord>
context)
        {
            return values.OrderBy(v =>
v.AttributeValueFrequencyScore).Take(this.numberOfOutliers).Select(v => new KeyValuePair<string,
string>(key.ToString(CultureInfo.InvariantCulture), v.ToString()));
        }
    }
```

Once AVF score is calculated for all the points, the k outliers returned are the points with the smallest AVF scores. The input values are sorted in the ascending order of their AVF scores and the first n outliers i.e. rows are returned as output. Combiner returns outliers of a particular partition. Reducer returns final set of outliers for the whole input dataset i.e. for all the partitions. To return only the first n outliers, the reducer count is restricted to one.

2.3 RUNNING THE SAMPLE



Operators
Research.MapReduce.Operators
Properties
References
Averages
CanopyClustering
DataSummary
FieldCombinationCount
FieldCount
IntervalGrouping
KMeansClustering
LogLikelihoodRatio
MapValue
MatchFilter
OutlierDetection
AttributeValueFrequencyRecord.cs
FindAttributesCountCombiner.cs
FindAttributesCountMapper.cs
FindAttributesCountReducer.cs
FindOutliersCombiner.cs
FindOutliersMapper.cs
FindOutliersReducer.cs
OutlierDetectionController.cs
RangeFilter
RangeGrouping
RemoveColumn
Sampling

Open the solution 'Research.MapReduce.sln' in a new instance of visual studio 2010 and build it.

2.3.1.1 DAYTONA RUNTIME DEPLOYMENT

Please refer to the document for 'Daytona-Setup Guide' to deploy the Daytona runtime onto an azure service.

2.3.1.2 UPLOAD INPUT

The input dataset should be in CSV format and must be uploaded to an azure blob.

2.3.1.3 SUBMIT

All the assemblies corresponding to the project will be uploaded to a dedicated azure blob container as a single logical unit named 'Package' and then submitted for execution.

```
            if (args.Length == 0)
            {

                Console.WriteLine(@"Usage:- -InputDataLocation::<value> -OutputDataLocation::<value> -
Mappers::<value> -Reducers::<value> -ColumnNames::<value> -NumberOfOutliers::<value>");

                return;

            }



            Dictionary<string, string> controllerArgs = new Dictionary<string, string>();

            foreach (string arg in args)

            {

                string[] argArray = arg.Split(new string[] { "::" },
StringSplitOptions.RemoveEmptyEntries);

                controllerArgs.Add(argArray[0].TrimStart('-'), argArray[1]);

            }



            string masterConnectionString = ConfigurationManager.AppSettings["MasterConnectionString"];

            if (string.IsNullOrEmpty(masterConnectionString) ||
string.IsNullOrWhiteSpace(masterConnectionString))

            {

                throw new InvalidOperationException("MasterConnectionString configuration missing.");

            }



            MapReduceClient client = new MapReduceClient("outlierdetection", "outlierdetection-" +
Guid.NewGuid(), typeof(OutlierDetectionController).AssemblyQualifiedName, controllerArgs);

            client.Submit(masterConnectionString);

            Console.WriteLine("Job submitted successfully...");

            Console.WriteLine("Waiting for completion...");

            bool concluded = client.WaitForCompletion(masterConnectionString, TimeSpan.FromMinutes(10),
TimeSpan.FromSeconds(5));

            if (concluded)

            {

                if (client.Succeeded)

                {

                    Console.WriteLine("Job completed successfully. Following are the results:");

                    foreach (KeyValuePair<string, string> result in client.Results)

                    {
                        Console.WriteLine("Result Name: '{0}' and Result Value: '{1}'", result.Key, 
result.Value);

                    }

                }

                else

                {

                    Console.WriteLine("Job failed due to: \n {0}", client.FailReason);

                }

            }

            else

            {

                Console.WriteLine("Wait timeout elapsed.");

            }


            Console.WriteLine("Press <ENTER> to exit.");

            Console.ReadLine();
```

The following keys in the app.config must be set to appropriate values as explained below.

```
<appSettings>

    <add key="MasterConnectionString" 
value="DefaultEndpointsProtocol=http;AccountName=<AccountName>;AccountKey=<AccountKey>"/>

<add key="StorageConnectionString" 
value="DefaultEndpointsProtocol=http;AccountName=<AccountName>;AccountKey=<AccountKey>"/>

  </appSettings>
```

- **MasterConnectionString:** This should be the same as the storage account used by the Daytona runtime deployment.
- **StorageConnectionString:** This should be the storage account intended for uploading input dataset and writing output of the job(s) as well as the final result. It can be same as 'MasterConnectionString' if both the accounts belong to the current user. This value is automatically set as the value of the controller parameter 'StorageAccountString' by 'MapReduceClient' which looks for this specific key in the 'appSettings' and sends it as controller argument irrespective of whether the user has specified any controller arguments or not.