MIDDLE EAST

TECHNICAL UNIVERSITY

COMPUTER ENGINEERING

DEPARTMENT

Final Design Report

Wekarel

A Parallel Weka Implementation

- Myrzabek MURATALIEV – *1408665*
- Serdar DALGIÇ – *1448570*
- Haşim TİMURTAŞ – *1449149*
- Barış YÜKSEL – *1449826*

Table Of Contents

1. INTRODUCTION....5
1.1. Project Title....5
1.2. Problem Definition....5
1.3. Project Goals and Scope....5
2. DESIGN CONSTRAINTS AND LIMITATIONS....6
2.1. Time Constraint....6
2.2. Performance Issues....6
2.3. Reliability of Used Toolkits and Libraries....7
2.4. Experience on Data Mining and Parallelization Software....7
3. DETAILED DESIGN....8
3.1. General Information....8
3.2. Packages, Classes and Inheritance Diagrams....8
3.2.1. weka.core package....8
3.2.2. weka.clusterers package....13
3.2.3. weka.classifiers package....17
3.2.4. weka.classifiers.bayes package....19
3.3. Class Diagram....21
3.4. Parallelization Decisions....22
3.4.1. ParallelInstances class....23
3.4.2. Parallel Utils Class....25
3.4.3. ParallelKMeans class....26
3.4.4. ParallelClusterEvaluation class....28
3.4.5. NaiveBayesSimple class....28
3.4.6. ParallelEvaluation class....31
4. DATA DESIGN....31
4.1. Function Modelling....31
4.1.1. Level 0 of DFD....32
4.1.2. Level 1 of DFD....32
4.1.3. Explanation of Level 1 DFD....33
4.1.4. Level 2 DFD....34
4.1.5. Explanation of Level 2 DFD....35
5. GUI DESIGN....36

5.1. GUI of Weka....36
5.2. Interface Usage....36
5.3. Planned Additions to User Interface....39
6. SPECIFICATIONS AND DEPENDENCIES....40
6.1. Specifications....40
6.2. Dependencies....41
6.2.1. Hardware Dependencies....42
6.2.2. Software Dependencies....42
7. DEVELOPMENT SCHEDULE....51
7.1. Project Management....52
7.1.1. Proposal Report....52
7.1.2. Requirement Analysis Report....52
7.1.3. Initial Design Report....52
7.1.4. Final Design Report....53
7.1.5. Seminars....53
7.1.6. Intel HPC Workshop....53
7.1.7. Weka Homepage, Wiki and Mailing Lists....54
7.1.8. Communication with Weka Developers....54
7.2. Parallelization and Testing of the Classes....55
7.2.1. Paralellization of Instances Class....55
7.2.2. Paralellization of Utils Class....55
7.2.3. Paralellization of SimpleKMeans Class....55
7.2.4. Paralellization of ClusterEvaluation Class....56
7.2.5. Paralellization of SimpleNaiveBayes Class....56
7.2.6. Paralellization of Evaluation Class....56
8. Conclusion....56
9. References....58
10. Appendix A – Gannt Chart 1....60
11. Appendix B – Gannt Chart 2....61

1. INTRODUCTION

1.1. Project Title

Our project title is “Wekarel”.

1.2. Problem Definition

Weka (Waikato Environment for Knowledge Analysis) is a free software (licensed by GNU General Public License) and a popular suite of machine learning software written in Java, developed at the University of Waikato. Weka supports several standard data mining tasks, more specifically, data preprocessing, clustering, classification, regression, visualization, and feature selection. The Cluster panel gives access to the clustering techniques in Weka, e.g., the simple k-means algorithm. While using these clustering techniques, speed and accuracy are the two keywords that modern world is interested in. There is an attempt to create some parallel calculations for Weka, called “Weka-Parallel”[1] but only for being able to run the cross-validation portion of any given classifier rapidly. There is not any work on data/task-parallelism of cluster evaluation processes. A parallelized software with noticeable increase in computation time and accuracy is what today's data mining world needs.

1.3. Project Goals and Scope

Weka is a capable framework on data mining that has proven her abilities with receiving several awards.[2][3] It's one of the most common and trusted open source tools for data-mining and machine learning experiments. However, it lacks the gain of speed and accuracy that parallel evaluation will bring.. *Wekarel* is intended to fill the gap.

Wekarel is going to be a parallel implementation of popular machine learning software Weka, and will keen on achieving data and task parallelism while evaluating clusters and classifiers. *Wekarel* is going to be compatible with 3.4 version of Weka,

that is the latest stable version of Weka.

By doing this project, we want to achieve:

- Code the parallelized versions of commonly used middle-layer parts of Weka, in order to enable the parallelized versions of the algorithms in clusterers and classifiers packages.
- Design the coding part in a modular way (just as in Weka) in order to create a flexible environment.
- Write a detailed documentation on parallelization process and classes, by this means pave the way of those who want to write “parallel-computable” versions of algorithms that are compatible with our design. A “parallel-computable” K – Means algorithm is promised to be implemented and documented as an example.
- Design a usable GUI part for selecting computer number, parallelization method and algorithm. Integrate the design with the current Weka GUI.

As *Wekarel* is going to be like a fork of Weka that works in parallelization, that's why it is going to possess the **portability**, **reliability**, **usability** and **object oriented approach** inherited from original Weka.

2. DESIGN CONSTRAINTS AND LIMITATIONS

2.1. Time Constraint

Wekarel Project must be completed at the end of May. Detailed information about scheduling and timing will be given as a Gannt Chart in the *Appendix* Part.

2.2. Performance Issues

Our implementation is going to be a standardized solution for all kinds of multicore machines, for this purpose, NAR (The high performance computing system

environment in METU Computer Engineering Department) is going to be our testing environment in development process. Multicore and multiple processor machines improves the performance by cache hits (for multicore processors, cache memory on a single chip is shared also), hyper-threading (multiple ALU or FPU's parallelized on a single core) and very low communication cost. However, still the communication costs are big deal, although infiniband switches speeds up the process.

2.3. Reliability of Used Toolkits and Libraries

There is no need to reinvent the wheel and we can not write every single code of the project by ourselves. Previous written middle – layers , libraries and toolkits are essential to use in the development process.

Weka, itself, is a framework for machine learning software. It has ongoing development process, but with a history more than 10 years and worldwide known fame for it's success increases reliability of the software. Among Weka's still known bugs, we didn't coincide with anything related to the part we are going to deal with; but although there is a slight chance, there can always happen a surprise in big projects like this.

Another leg of our implementation consists of parallelization software. There are some alternatives for using as a paralellization software integrated with java platform (These alternatives are discussed in details in specifications and dependencies part of the report.) They can involve some kind of bugs that we may run across, but the usability and features these programs inherit aims us to choose these software.

2.4. Experience on Data Mining and Parallelization Software

Although all group members are interested in both of these topics, there is a slight concern on all four of us about our lack of experience on data mining and parallelization software. Having our accounts created on NAR affected us positively

on this constraint, however we still need time to have our experiences been risen up in due time.

3. DETAILED DESIGN

3.1. General Information

In Weka, each algorithmic models are grouped into different packages. For each package, there exists a different unique evaluator class; and for each class implementing the algorithms, there exists a specific algorithm builder class. In our project, we will focus on the parallelization of clustering algorithms (in weka.clusterers package) and classifying algorithms (in weka.classifiers package). We will parallelize the generic evaluator method of evaluator classes of these packages, and also will implement the parallelized version of one algorithm for each package because these generic evaluator classes are common for all other algorithms and showing the implementation of any algorithm would be counted as a proof of concept. We have selected the K – means clustering algorithm for clusterers package, and the simple naive – Bayes classifier algorithm for classifiers package. In addition to these, what we mean by parallellizing a class is that, we are going to create another class which inherits the original class, and override the methods we intend to parallelize.

3.2. Packages, Classes and Inheritance Diagrams

3.2.1. weka.core package

These classes are the most primitive ones that forms the skeleton of weka tool. The inner dynamics including handles for spesific classes are all included in this package.

3.2.1.1. Instance Class

It is the class for handling a single data item in a data set. All values (numeric,

date, nominal, or string) are internally stored as floating-point numbers. If an attribute is nominal (or a string), the stored value is the index of the corresponding nominal (or string) value in the attribute's definition.

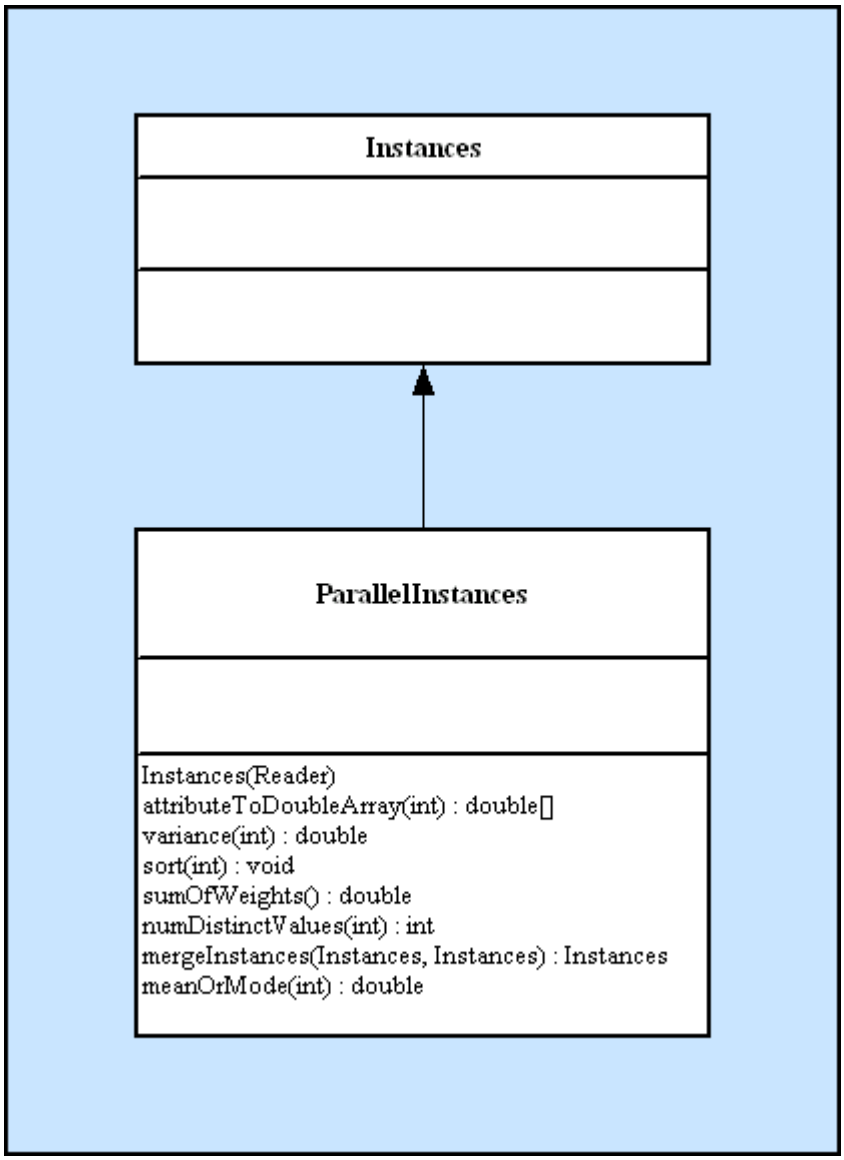
3.2.1.2. Attribute Class

It is the class for handling a single attribute in a dataset. A property of an attribute is once it has been created, it can't be changed.

3.2.1.3. Instances Class

Represents a group of objects of type Instance. This class for handling an ordered set of weighted instances. It is considered to be beneficial to parallelize some of the crucial methods of this class.

3.2.1.4. Inheritance Diagram for ParallelInstances Class



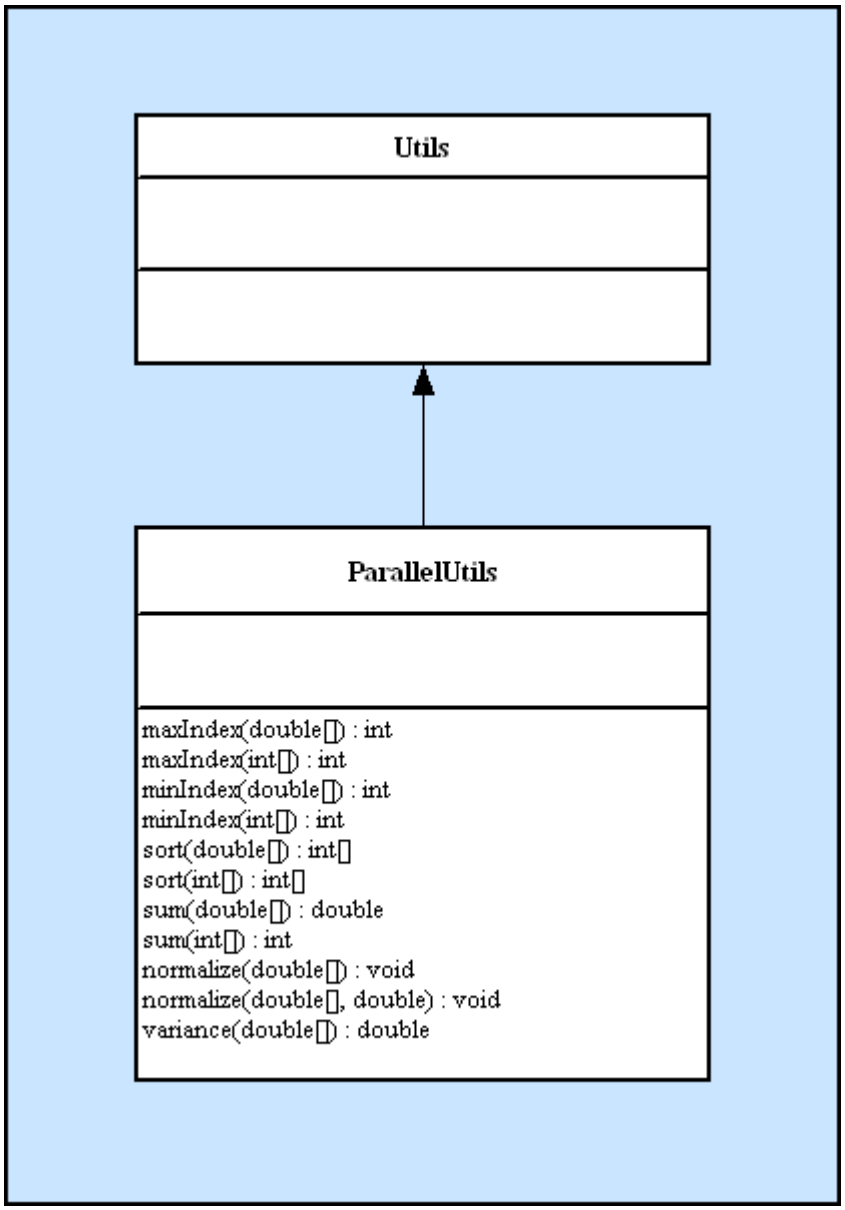
Below is the summary of the methods we are going to override in the parallelized version of the Instances class:

ParallelInstances	
`Double[] attributeToDoubleArray(int index)`	Gets the value of all instances in this dataset for a particular attribute.
`double variance(int attIndex)`	Computes the variance for a numeric attribute.
`void sort(int attIndex)`	Sorts the instances based on an attribute.
`double sumOfWeights()`	Computes the sum of all the instances' weights.
`int numDistinctValues(int attIndex)`	Returns the number of distinct values of a given attribute.
`Static Instances mergeInstances(Instances first, Instances second)`	Merges two sets of Instances together.
`double meanOrMode(int attIndex)`	Returns the mean for a numeric attribute, and mode for a nominal attribute as a floating-point value.
`Instances(Reader reader)`	Reads an ARFF file from a reader, and assigns a weight of one to each instance.

3.2.1.5 Utils Class

This is a class that some simple utility methods are implemented. Some examples are eq Method that tests equality of two integers, getArrayDimensions Method for acquiring array dimensions, maxIndex, minIndex methods for both double and ints, sum, sort methods..etc. Another interesting point about this class is some of the methods implemented here are rewritten(copied, pasted) to some spesific algorithms. We think it may be a trick to specialize methods according to the classes, instances, so that faster calls may happen.

3.2.1.6 Inheritance Diagram for ParallelUtils Class



Below is the summary of the methods we are going to override in the parallelized version of the Utils class:

ParallelUtils	
`int maxIndex(double[] doubles)`	Returns index of maximum element in a given array of doubles.
`int maxIndex(int[] ints)`	Returns index of maximum element in a given array of integers.

`int minIndex(double[] doubles)`	Returns index of minimum element in a given array of doubles.
`int minIndex(int[] ints)`	Returns index of minimum element in a given array of integers.
`int[] sort(double[] array)`	Sorts a given array of doubles in ascending order and returns an array of integers with the positions of the elements of the original array in the sorted array.
`int[] sort(int[] array)`	Sorts a given array of integers in ascending order and returns an array of integers with the positions of the elements of the original array in the sorted array.
`double sum(double[] doubles)`	Computes the sum of the elements of an array of doubles.
`int sum(int[] ints)`	Computes the sum of the elements of an array of integers.
`void normalize(double[] doubles)`	Normalizes the doubles in the array by their sum.
`void normalize(double[] doubles, double sum)`	Normalizes the doubles in the array using the given value.
`double variance(double[] vector)`	Computes the variance for an array of doubles.

3.2.1.5. Matrix Class

weka.core.Matrix is the deprecated package that is kept in the source code only for backwards compatibility. So the methods in this package and the constructors are all seen as deprecated. Instead of this, *weka.core.matrix.Matrix* package is used. This is a class for performing operations on a matrix of floating-point values. Also conversion of the matrix to Matlab strings, or simple strings are also handled in this class. For now, we plan to parallelize Matrix construction methods since it is one of the most suitable code snippets that can be parallelized in the source code.

3.2.1.6. Inheritance Diagram for ParallelMatrix Class

Below is the summary of the methods we are going to override in the parallelized version of the Matrix class:

ParallelMatrix	
`Matrix(double[] vals, int m)`	Construct a matrix from a 2-D array.
`Matrix(int m, int n, double s)`	Construct an m-by-n constant matrix.

3.2.2. weka.clusterers package

In this package there are classes for implementing clustering algorithms such as Cobweb and Classit clustering algorithms, Simple EM(expectation maximization), Farthest First Traversal Algorithm, Simple K – Means algorithm and so on. There are also classes that generates a general approach and are treated as a middle-layer for these implemented algorithms. These generative classes are going to be one of our main interests.

3.2.2.1 Clusterer class

It is the abstract class whose abstract methods are to be implemented by the classes generating the clustering algorithm (SimpleKMeans class in our proof of concept case). There are methods in this class for

- generating a clusterer and initializing all fields of the clusterer that are not being set via options
- classifying a given instance.
- predicting the cluster memberships for a given instance.
- returning the number of clusters
- creating a new instance of a clusterer given it's class name and (optional) arguments to pass to it's setOptions method.

3.2.2.2 ClusterEvaluation class

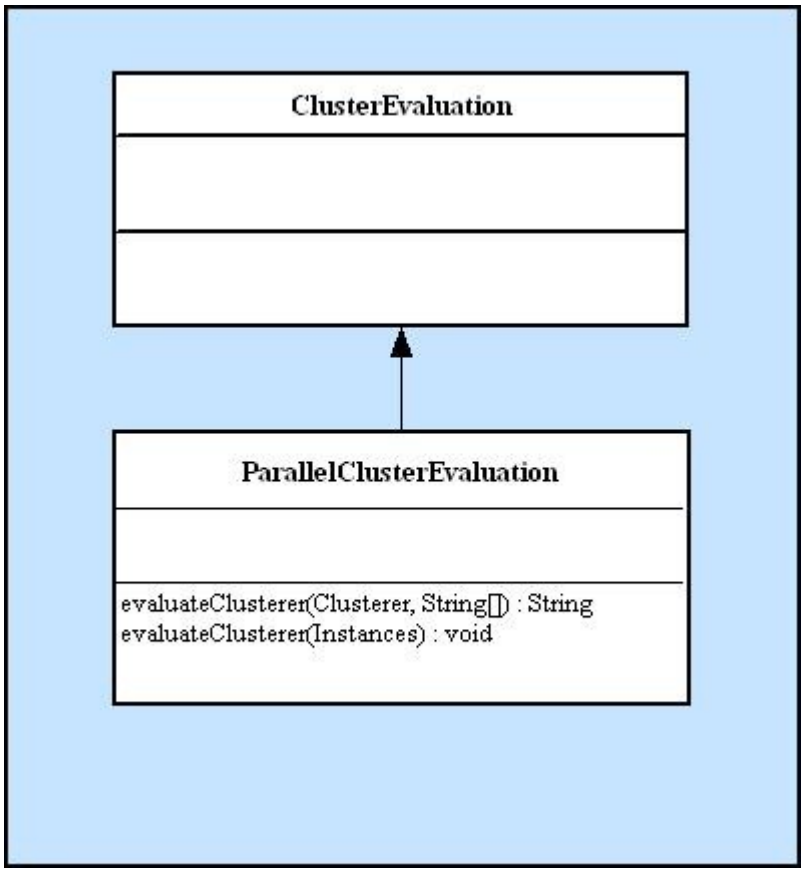
This is the class which evaluates the clustering model with the given parameters via its evaluateClusterer method. This is one of the key classes we are going to focus on parallelling. Arranging the data set and filling the parameters of the clusters before evaluating the algorithms happen within this class. The instances to cluster, the clusterer, the number of clusters found by the clusterer, the mapping of classes to clusters (for class based evaluation) are the variables that this class holds.

3.2.2.3 SimpleKMeans class

This is the class implementing the K-means algorithm. buildClusterer is the method which generates the algorithm for a given data set. There are variables for the number of clusters to generate, holding the cluster centroids, holding the standard deviations of the numeric attributes in each cluster, holding the frequency counts for the values of each nominal attribute for each cluster, the number of instances in each cluster and keeping track of the number of iterations completed before convergence in this class.

We are going to parallelize this class as a proof of concept.

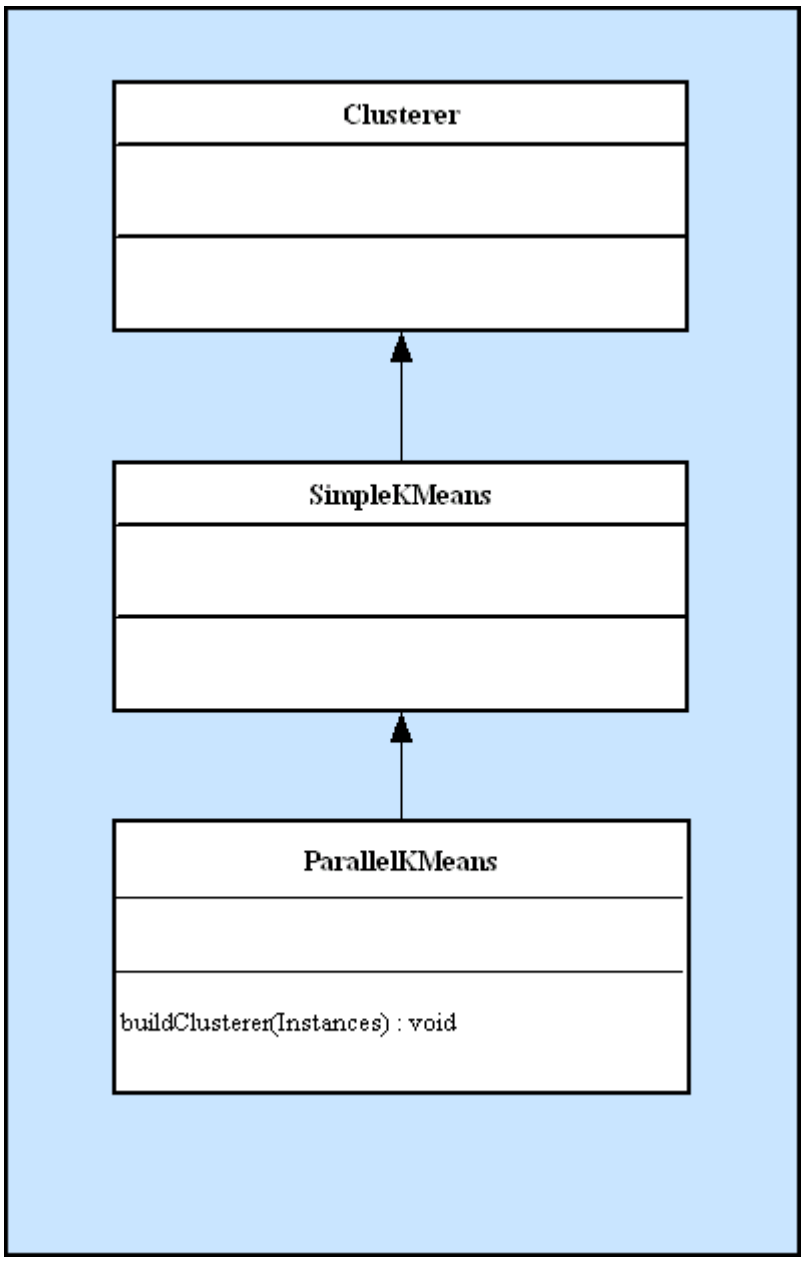
3.2.2.4 Inheritance Diagram for ParallelClusterEvaluation Class



Below is the summary of the methods we are going to override in the parallelized version of the ClusterEvaluation class:

ParallelClusterEvaluation	
`String evaluateClusterer(Clusterer clusterer, String[] options)`	Evaluates a clusterer with the options given in an array of strings.
`void evaluateClusterer(Instances test)`	Evaluate the clusterer on a set of instances.

3.2.2.5 Inheritance Diagram for ParallelKMeans Class



Below is the summary of the methods we are going to override in the parallelized version of the SimpleKMeans class:

ParallelKMeans	
`void buildClusterer(Instances data)`	Generates the clusterer for the algorithm.

3.2.3. weka.classifiers package

In this package there are classes for implementing classifiers. These classes vary on machine learning models to abstract utility classes for handling settings common to different types of classifiers. There are also classes that generates a general approach and are treated as a middle-layer for these classifiers. These generative classes are going to be one of our main interests.

3.2.3.1 Classifier Class

It is the abstract class whose abstract methods are to be implemented by the classes generating the classifying algorithm (NaiveBayesSimple class in our proof of concept case). There are methods in this class for

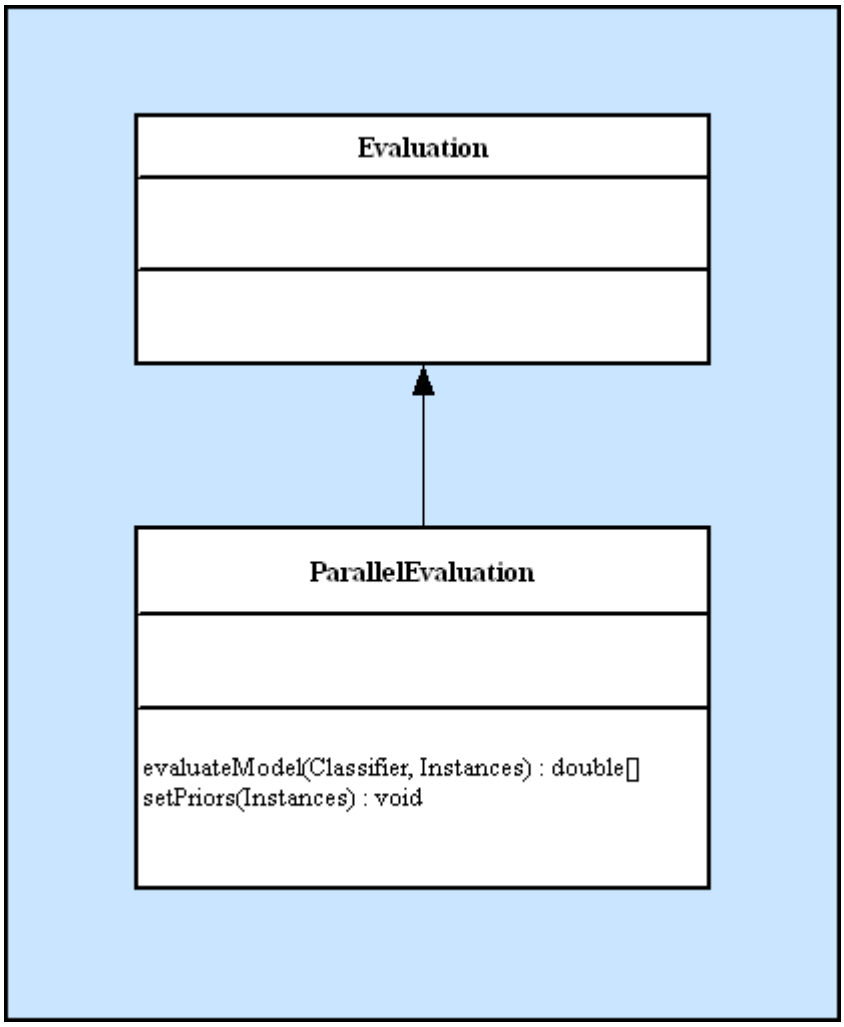
- generating a classifier and initializing all fields of the clusterer that are not being set via options
- classifying a given instance.
- predicting the cluster memberships for a given instance.
- returning the number of clusters
- creating a new instance of a clusterer given it's class name and (optional) arguments to pass to it's setOptions method.
- creating a deep copy of the given classifier using serialization.
- creating a given number of deep copies of the given classifier using serialization.
- getting the current settings of the Classifier.

3.2.3.2 Evaluation class

It is the class which evaluates the classifying model with the given parameters via its evaluateModel method. This is one of the key classes we are going to focus on parallelling. The number of classes, the weight of all incorrectly classified instances, the weight of all correctly classified instances, the property of the class whether it is nominal or numeric, the sum of class/predicted/squared predicted/squared class

values are all hold in this class. Arranging the data set and filling the parameters of the classifiers before evaluating the algorithms happen within this class.

3.2.3.3 Inheritance Diagram for ParallelEvaluation Class



Below is the summary of the methods we are going to override in the parallelized version of the Evaluation class:

ParallelEvaluation	
`double[] evaluateModel(Classifier classifier, Instances data)`	Evaluates the classifier on a given set of instances.
`void setPriors(Instances train)`	Sets the class prior probabilities.

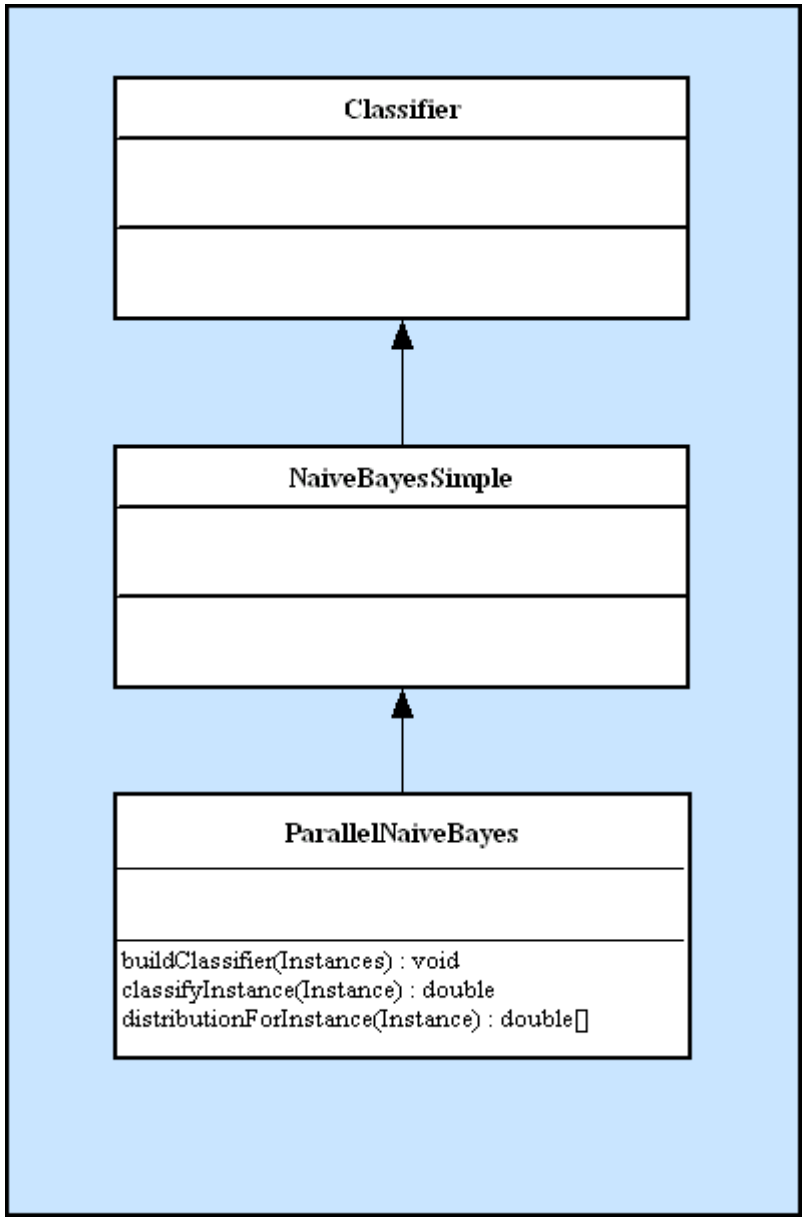
3.2.4. weka.classifiers.bayes package

This package is one depth under weka.classifiers package in the package hierarchy, and contains most of the bayesian algorithms. Base class for a Bayes Network classifier, a class for building and using a Complement class Naive Bayes classifier, a class for a Naive Bayes classifier using estimator classes, a class for building and using a simple Naive Bayes classifier and a class for a Naive Bayes classifier using estimator classes are the summaries of the classes that are included in this package.

3.2.4.1 NaiveBayesSimple class

The class implementing the naive Bayes algorithm. We are going to parallelize this class as a proof of concept for the classifiers. All the counts for nominal attributes, the means for numeric attributes, the standard deviations for numeric attributes, the prior probabilities of the classes, the instances used for training and constant for normal distribution are kept as variables in this class. buildClassifier is the method which generates the algorithm for a given data set.

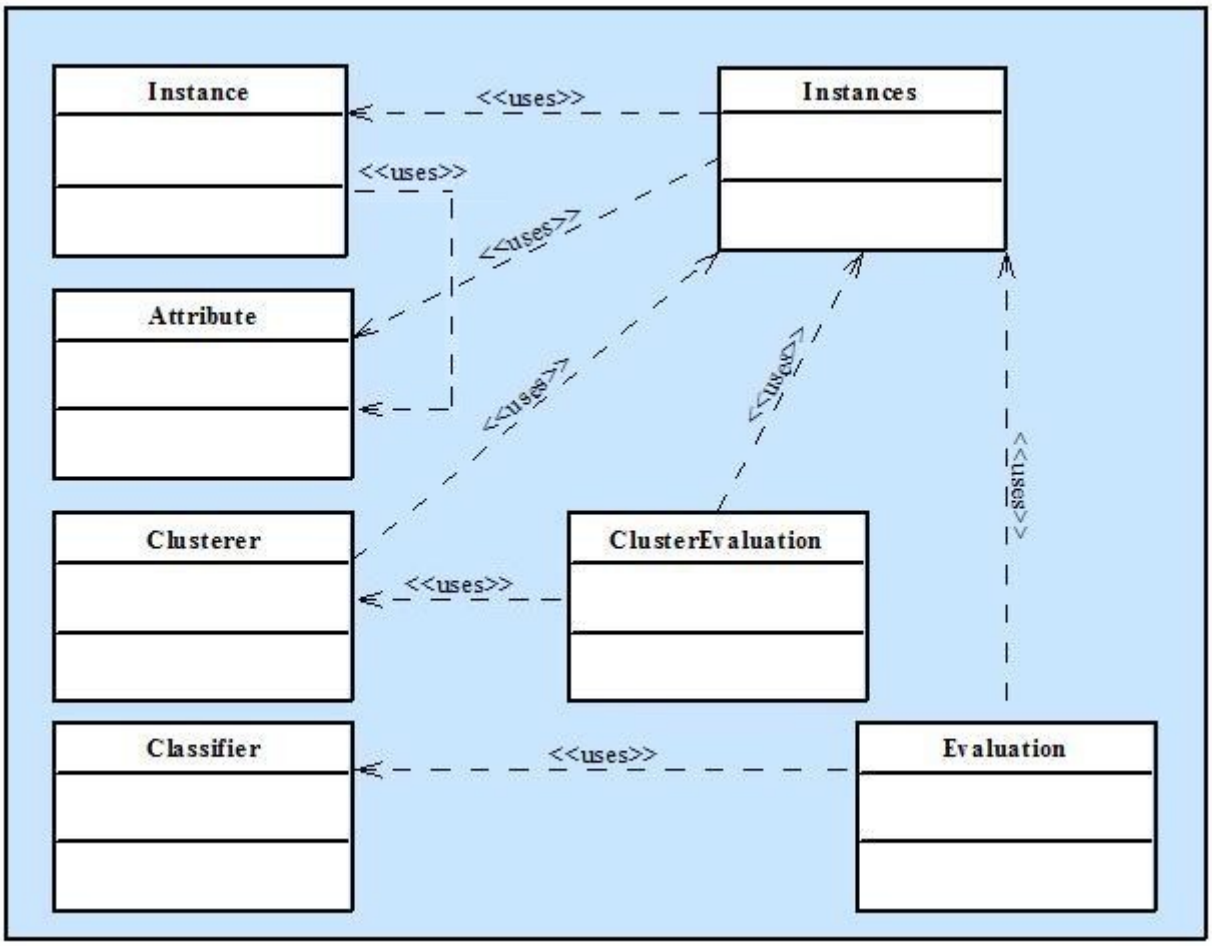
3.2.4.2 Inheritance Diagram for ParallelNaiveBayes Class



Below is the summary of the methods we are going to override in the parallelized version of the NaiveBayesSimple class:

ParallelNaiveBayes	
`void buildClassifier(Instances instances)`	Generates the classifier for the algorithm.
`double classifyInstance(Instance instance)`	Classifies the given instance.
`double[] distributionForInstance(Instance instance)`	Calculates the class membership probabilities for the given instance.

3.3. Class Diagram



This class diagram shows the dependencies between the classes (i.e. which class uses an instance of which class in its methods). The inherited classes are not shown in this diagram, since they have already been explained through the inheritance diagrams in 3.1.

The variables and methods of the classes in Weka are either available on the API documentation page of weka[4], or in the source code of Weka. We have tried to explain some of the variables and some of the methods while explaning related packages; however writing all of them in the class diagram is impossible to effectuate. There are large number of methods and variables in the classes shown in the diagrams. For these reasons, in the diagrams we choose not to write all of these methods and variables in the classes. This is the reason why the variables and methods of most of the classes are shown empty. We only showed the methods we

are going to override and modify in the parallel versions of these classes.

3.4. Parallelization Decisions

We will mostly use collective communicaton as the communication type in MPI, since most of the methods we are going to parallelize include iterations, calculations and similar kind of evaluations. It seems the most convenient strategy to involve all the nodes in the scope of a communication is to use collective communication. The data will be distributed from one node to all of the nodes, and will be gathered at the same node.

In most cases, we use data parallelism in the newly overridden classes. During the parallellization process, we intend to use task parallelism where neccessary. This is scheduled to be done in the second term, in the improvement task of the classes. In order to measure the improvement, we are going to use the time functions of Java. Also, we consider using Intel's Trace Analyzer and Collector software for measuring the performance. This software will be explained in details in the Software Dependencies section.

Parallelization of some of these methods & blocks might have a risk of decreasing the performance, so while coding, we may omit these. This has been one of the main concerns in our project design; we tried to overcome this issue, but without implementing and testing, one can not be sure whether the selected parallelization attempts may be profitable or not. The thing we mainly focused on while considering this issue is the magnitude of the loops i.e. the range of the loops. For huge amount of data, it is the most legitimate movement to distribute the data. However, for *for* loops running on small amount of data, distributing the data among the nodes may bring overheads and the result may be longer than the non-parallelized version of Weka.

So, *instance* is the keyword for our *data parallelization* decisions. Almost everywhere, where *instance* is mentioned and *for* loop is located, we realized that

there is a chance to distribute the huge amount of data among the nodes and this action will remarkably increase program's performance.

Below is the descriptions of the methods that we are going to parallelize and giving brief information about the methods.

3.4.1. ParallelInstances class

3.4.1.1. attributeToDoubleArray() method

Form of parallellization: data parallelism

For large datasets, the training data is split into different nodes, and each node has a number of instances. Then, these instances are gathered in the master node.

This method is called from the algorithms under the classifiers package, such as PaceRegresssion and ThresholdCurve.

3.4.1.2. variance() method

Form of parallellization: data parallelism

The data is dispersed into different nodes. Each node calculates the variance of the instances belonging to it. Then, the total calculation is collected in the master node.

This method is called from the algorithms under the classifiers and clusterers packages, such as LinearRegresssion and SimpleKMeans.

3.4.1.3. sort() method

Form of parallellization: data parallelism

The data is split into nodes. Each node applies the built-in sorting algorithm for the instances belonging to it. Then, k-way merge is applied for each node, and the whole sorted data is combined.

This method is called from the algorithms under the classifiers package, such as ADTree, DecisionStump and m5RuleNode.

3.4.1.4. sumOfWeights() method

Form of parallellization: data parallelism

The data is split into different nodes. Each node calculates the sum of the weights of the instances belonging to it. Then, the total value is summed up in the master node.

The data is first sorted in parallel, with the sorting algorithm above. Then, the data is scattered into different nodes. Each node calculates the sum of the weights of the instances belonging to it. Then, the total value is summed up in the master node.

This method is called from several classes under the classifiers package.

3.4.1.5. numDistinctValues() method

Form of parallellization: data parallelism

The data is first sorted in parallel, with the sorting algorithm above. Then, the data is scattered into different nodes. Each node calculates the number of distinct data. Then, the total count is calculated by adding up these values.

This method is called from RuleStats class under weka.classifiers.rules package.

3.4.1.6. mergeInstances() method

Form of parallellization: data parallelism

The data is scattered into different nodes. Then, k-way merge is applied, as in the sort() method, and the whole data is combined.

This method is not too essential one to parallelize, since there are no calls to this method from other methods except the main method of Instances class.

3.4.1.7. meanOrMode() method

Form of parallellization: data parallelism

The data is split into different nodes. Each node calculates the mean (or mode) values of the instances belonging to it. Then, the total value is summed up in the

master node.

This method is called from the algorithms under the classifiers and clusterers packages, such as LinearRegresssion, SimpleLinearRegresssion and SimpleKMeans.

3.4.1.8. Instances(Reader) constructor

Form of parallellization: data parallelism

The reading process of the data and inserting these values in an Instance object can be done in parallel. Assume there are k nodes available and n elements in the data set. Each node with rank r reads the data with index i where i mod k = r. Then, these sub-datasets can be collected to construct the whole data set.

3.4.2. Parallel Utils Class

3.4.2.1. maxIndex() & minIndex methods

Form of parallellization: data parallelism

First, the data is split into different nodes. Each node calculates the index of the min & max element belonging to it, and then the index of the min & max element of these min & max values are calculated one more time.

This method is called from several classes under the classifiers & clusterers package.

3.4.2.2. sort() method

Form of parallellization: data parallelism

This sorting algorithm is very similar to the sort() method under Instances class. Therefore, the same parallelization technique can be applied here.

This method is called from several classes under the classifiers & clusterers package.

3.4.2.3. sum() method

Form of parallellization: data parallelism

This algorithm is very similar to the sumOfWeights() method under Instances class. Therefore, the same parallelization technique can be applied here.

This method is called from several classes under the classifiers & clusterers package.

3.4.2.4. normalize() method

Form of parallellization: data parallelism

The data is split into different nodes. Each node normalizes the element in the given array, where this element is assigned to the given node. Then the whole array is collected at one node.

This method is called from several classes under the classifiers & clusterers package.

3.4.2.5. variance() method

Form of parallellization: data parallelism

This algorithm is very similar to the variance() method under Instances class. Therefore, the same parallelization technique can be applied here.

This method is not too essential one to parallelize, since there are no calls to this method from other methods except the main method of Utils class.

3.4.3. ParallelKMeans class

3.4.3.1. buildClusterer() method

Form of parallellization: data parallelism

In this method, where the K – means clustering algorithm is applied, there are several blocks of code which we intend to parallelize. Below are the most essential ones:

- ```
  for (int i = 0; i < instances.numInstances(); i++) {
        updateMinMax(instances.instance(i));
  }
  ```

Here, this code block updates minimum and maximum values of the attributes all data in a dataset. This part may be executed in parallel, where all elements in the data set are split into different nodes, and each node updates the min and max values of the elements belonging to it, and then these values are collected in the master node. As the whole data (every instance) enters into that region, it's essential to parallelize this method.

- ```
  for (i = 0; i < instances.numInstances(); i++) {
        Instance toCluster = instances.instance(i);
        int newC = clusterProcessedInstance(toCluster, true);
        if (newC != clusterAssignments[i]) {
              converged = false;
        }
        clusterAssignments[i] = newC;
  }
  ```

This part is also one of the code snippets that every instance(single piece of data) is traced through. `clusterProcessedInstance(toCluster, true)` clusters an instance that has been through the filters; after this, it is checked that whether the new assignment is equal to the old assignment; if not, K – means algorithm resumes recalculating new clusters. Parallelizing this code snippet will profit much for the time management.

- ```
  for (i = 0; i < instances.numInstances(); i++) {
        tempI[clusterAssignments[i]].add(instances.instance(i));
   }
  ```
  ```

This code snippet is used while updating the code snippets. It's used in cluster assignments as some of the data's belonging centroid may have changed and new values should be assigned according to the changes. No need to say, whole data enters that code region, and it's beneficial to distribute then gather the data while these calls are made.

3.4.4. ParallelClusterEvaluation class

3.4.4.1. evaluateClusterer() method

In this method, there seems to be no code blocks to parallelize, since the calls to this method have already been parallelized in one level below. However, the code segments can be divided into discrete parts so that each segment corresponds to a job, and these jobs can be split into nodes. In this way, task parallelism can be applied. We are considering these alternatives as assumptions state that these considerations may be reconsidered in the later versions of *Wekarel*.

This method is called from main functions of all classes under the clusterers package. That means this method is common in every weka.clusterers.* packages.

3.4.5. NaiveBayesSimple class

3.4.5.1. buildClassifier() method

Form of parallellization: data parallelism

- ```
  // Compute counts and sums
   Enumeration enumInsts = instances.enumerateInstances();
   while (enumInsts.hasMoreElements()) {
     Instance instance = (Instance) enumInsts.nextElement();
     if (!instance.classIsMissing()) {
       Enumeration enumAtts = instances.enumerateAttributes();
       attIndex = 0;
       while (enumAtts.hasMoreElements()) {
  ```
  ```

```
        Attribute attribute = (Attribute) enumAtts.nextElement();
        if (!instance.isMissing(attribute)) {
          if (attribute.isNominal()) {
            m_Counts[(int)instance.classValue()][attIndex]
            [(int)instance.value(attribute)]++;
           } else {
           m_Means[(int)instance.classValue()][attIndex] +=
instance.value(attribute);
           m_Counts[(int)instance.classValue()][attIndex][0]++;
           }
         }
        attIndex++;
      }
      m_Priors[(int)instance.classValue()]++;
    }
  }
```

In this part of the code, computations and summations of Simple NaiveBayes classifications are made. The Priors of the classValue's are estimated and loops are going through the data. Distributing the data among the nodes through the loops in this code snippet may be beneficial.

- ```
  // Compute standard deviations
  enumInsts = instances.enumerateInstances();
  while (enumInsts.hasMoreElements()) {
    Instance instance = (Instance) enumInsts.nextElement();
    if (!instance.classIsMissing()) {
  ```
  ```

```
      enumAtts = instances.enumerateAttributes();
      attIndex = 0;
      while (enumAtts.hasMoreElements()) {
        Attribute attribute = (Attribute) enumAtts.nextElement();
        if (!instance.isMissing(attribute)) {
          if (attribute.isNumeric()) {
           m_Devs[(int)instance.classValue()][attIndex] +=
  (m_Means[(int)instance.classValue()][attIndex]-
  instance.value(attribute))*
  (m_Means[(int)instance.classValue()][attIndex]-
  instance.value(attribute));
          }
        }
      attIndex++;
      }
    }
  }
```

Within this code snippet, standard deviations are calculated, among the instances with the related attributes. The loops within this code are planned to be parallelized.

This method is called from almost all classifier algorithm packages and classifier filters.

3.4.5.2. distributionForInstance() method

Form of parallellization: data parallelism

This method calculates the class membership probabilities for the given test instance. There are loops going through the whole data, and because of the reasons

stated before, we intend to apply parallelization in this method also.

This method is also called from almost all classifier algorithm *meta* packages and classifier filters.

3.4.6. ParallelEvaluation class

3.4.6.1. setPriors() method

Form of parallellization: data parallelism

The training data is split into nodes, and each node sets the class prior probabilities of the sub-dataset belonging to it. Then, the calculations and the objects used for the calculations are collected in the master node.

3.4.6.2. evaluateModel() method

Form of parallellization: data parallelism

In this method, each data in the data set is evaluated independently, by calling evaluateModelOnce method. If we split the data into nodes and each node evaluates the classifier on the instances belonging to the node itself, the parallelization can be done.

4. DATA DESIGN

4.1. Function Modelling

In this section, data flow between user and parallelized Weka system and data flow between modules of system will be described by diagrams upto reasonable levels. At the same time structured design of the data processing of the system will be explained to generate the general visualization of main processes.

4.1.1. Level 0 of DFD

Level 0 of DFD is shown in Figure 1; it is general overview of the system.

Level 0 DFD

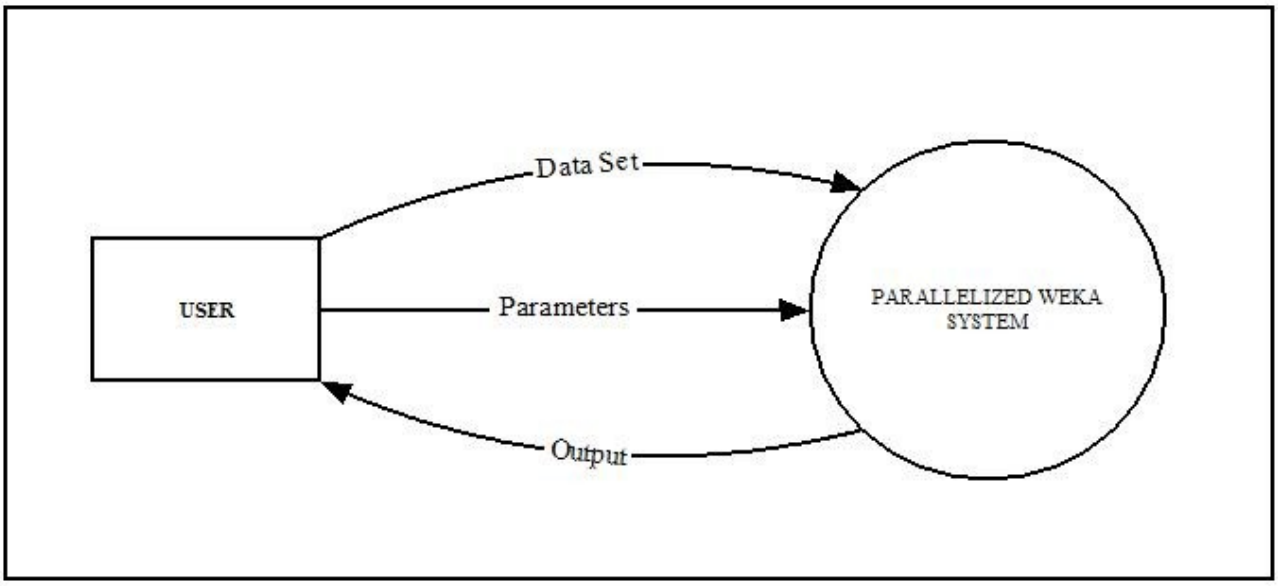


Figure 1

4.1.2. Level 1 of DFD

Level 1 of DFD is shown in Figure 2; it is more detailed view of Level 0 DFD. The system is divided into three processes; Preprocess the Data, Run the Algorithm and Generate the result.

Level 1 DFD

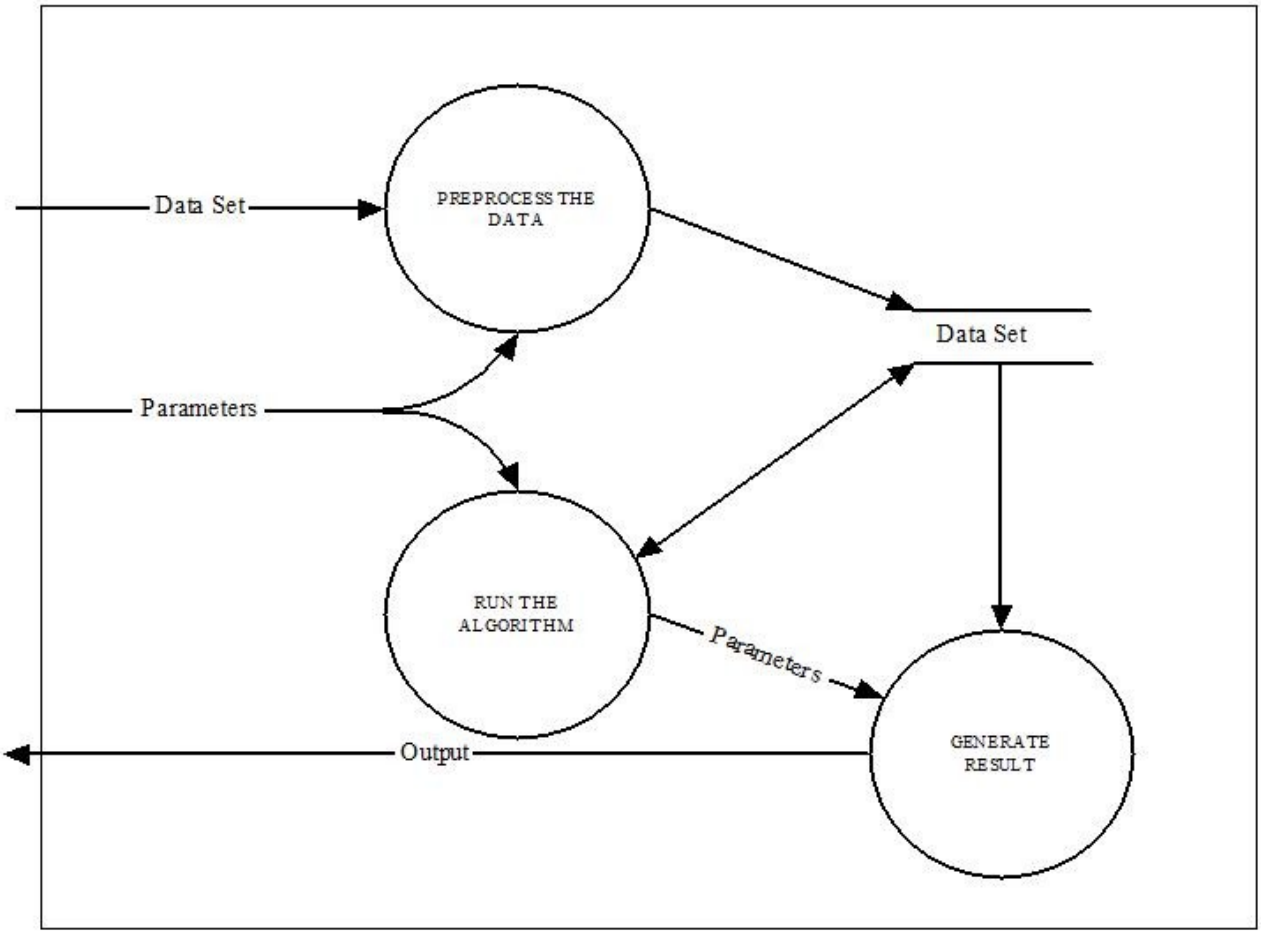


Figure 2

4.1.3. Explanation of Level 1 DFD

Preprocess the data: Initial data inspection is performed in this process. The dataset will be loaded to the system and attribute selecting or attribute filterings are done if needed. Discretization on numeric or continuous attributes is also will be done if techniques like association rule mining is going to be performed on dataset, since these techniques require only categorical data.

Run the algorithm: It is the main process in the system and actual parallelization is presented in this process. Obviously, this process is counted as a main part of this project. It is not easy to completely visualize the general structure of

this process in data flow diagram, since explanation of any part will be very near to the class and method level. The main interfaces and classes that are have to be inherited in constructing algorithm classes, such as instances class, are all modified in a way that they help the system to process paralelly. General overview of this process can also be obtained in the class diagram part of the Detailed Design part.

Generate result: Result is very important in Weka tool and system doesn't have any meaning without it. Not all algorithms emit same type of results so we need this process to generate the result according to the algorithms selected and according to the parameters given to the system by user.

4.1.4. Level 2 DFD

Level 2 of DFD is shown in Figure 3; it is more detailed view of Run the Algorithm process stated in Level 1 DFD. Other processes of Level 1 of DFD are not exploded in this level as they have no relation to the core of our project. Important and nearly most parts of the project will be included when we lay the Run the Algorithm process to the diagram, but general view and details will not be attainable since diagram just highlights flow of data through the system. Also the diagram presents the data flow of the parts where there will be cardinal changes when the system will be parallelized, so some parts of the system having negligible importance are not ephasized. Another thing to point out, as *'Wekarel'*s main job is to prepare middle layers for parallelizing main important algorithms of Weka tool; at the moment, clusterer and classifier parts of the system are accentuated as main tasks in this process. The Run the Algorithm process is divided into three processes; Build Algorithm, Evaluate Clusterer and Evaluate Classifier.

Level 2 DFD

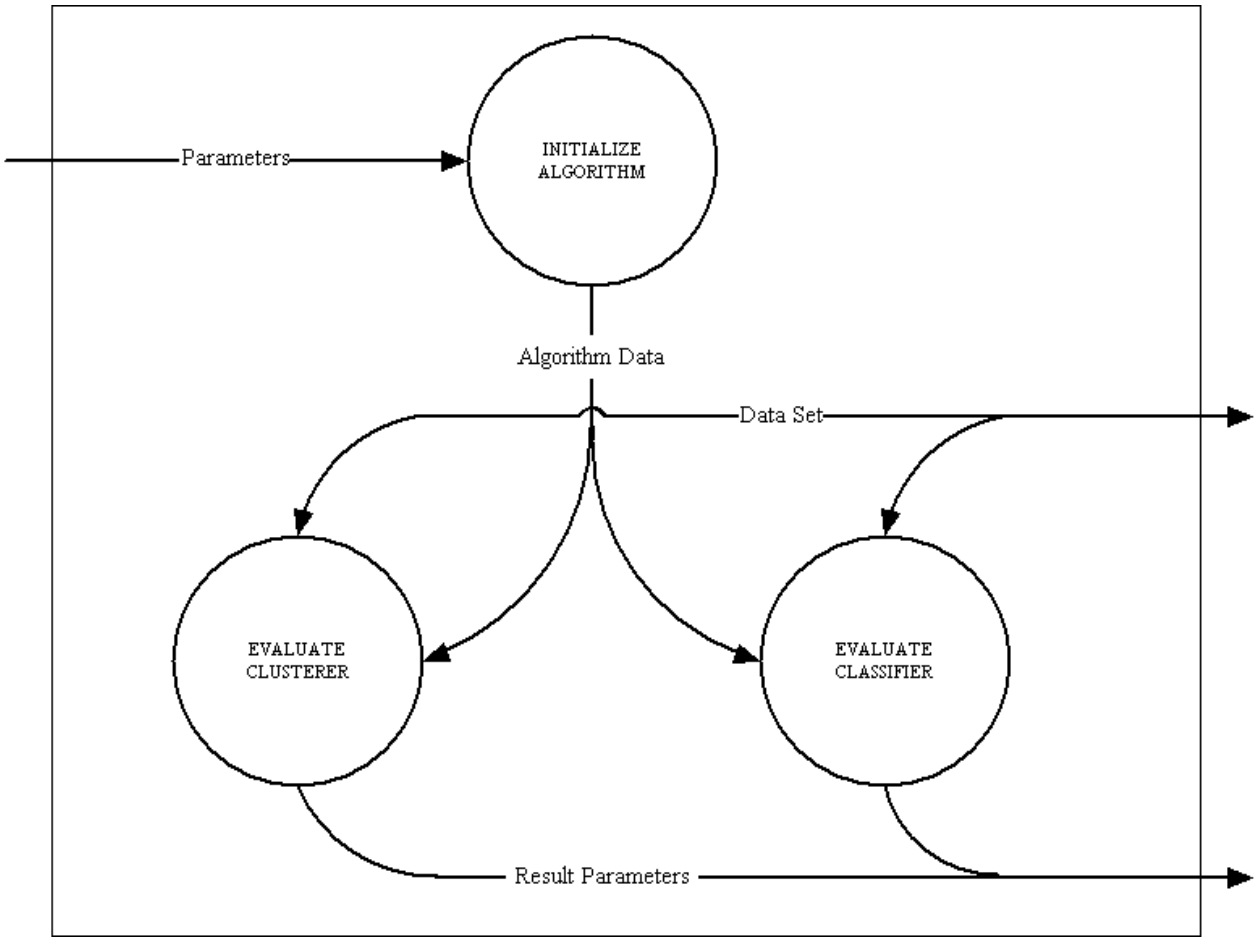


Figure 3

4.1.5. Explanation of Level 2 DFD

Initialize algorithm: This process initializes the algorithm stated in the parameters and performs the last operations which prepares the dataset to be evaluated. Some methods of Instances class which traces through the whole dataset will be performed parallelly when instance of this class is being initialized. After, evaluate clusterer or evaluate classifier process is triggered according to the type of the algorithm. Since projects main job is to prepare parallelized middle layer for main important algorithms of Weka tool, for the moment clusterer and classifier parts of system are aimed as a first tasks in this process.

Evaluate clusterer: All clustering algorithms are computed in this process and this is the one of the parts where paralellism will be conserved. Many classes and

methods shared by this class of algorithms will be parallelized. The listings of these classes and methods can be found in Detailed Design part of this report. Additionally, algorithm specific methods are also expected to run parallelly for this process to be counted fully parellelized.

Evaluate classifier: All classifying algorithms and their methods are computed in this process and this is the part where paralellism is conserved. All common functions of classifying algorithms will live changes to be able to operate paralelly. General overview of this process can also be obtained from the context of the Detailed Design section. One of the main algorithm specific methods that are additionally expected to run parallely can be buildClassifier() method.

5. GUI DESIGN

5.1. GUI of Weka

Talking about our Graphical User Interface Design, our interface will not have many differences compared to the interface of the original system. Because it is true that we are not adding many functionalities to the Weka tool. The only thing we are adding to the tool is an option, which gives an ability to operate in a parallel way. Another thing we can add to the interface is a small text box which passes the number of desired clusters from user to the system if one of the clustering algorithm is going to be computed. So all changes and designs that are going to be realized in project duration are not reflected in user interface part. The original Weka tool has Graphical User Interface which is not very complicated actually, but in first glance it may seem not very user friendly. Interface may slightly differ from version to version but parts representing general functionalities usually are not changed and figure below is user interface of 3.5.8 version.

5.2. Interface Usage

There are two panels of general functionalities that are going to live small

visual additions. To fully sense and visualize these additions, firstly, interface basics have to be familarized. All operations starts by loading the dataset to the system. Initially, by the “Open” button in the Preprocess tab and navigating to the directory containing the file dataset is loaded as in figure 5.

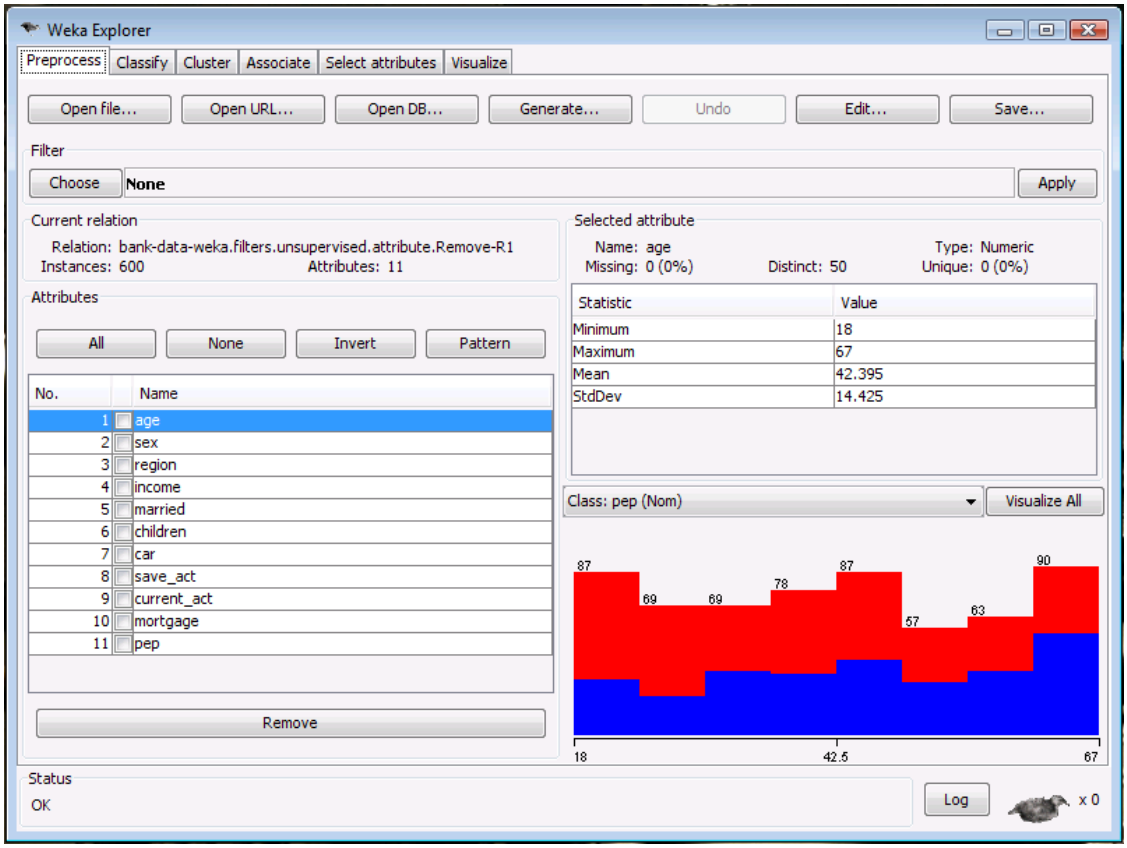


Figure 4 – Main Screen of Weka GUI

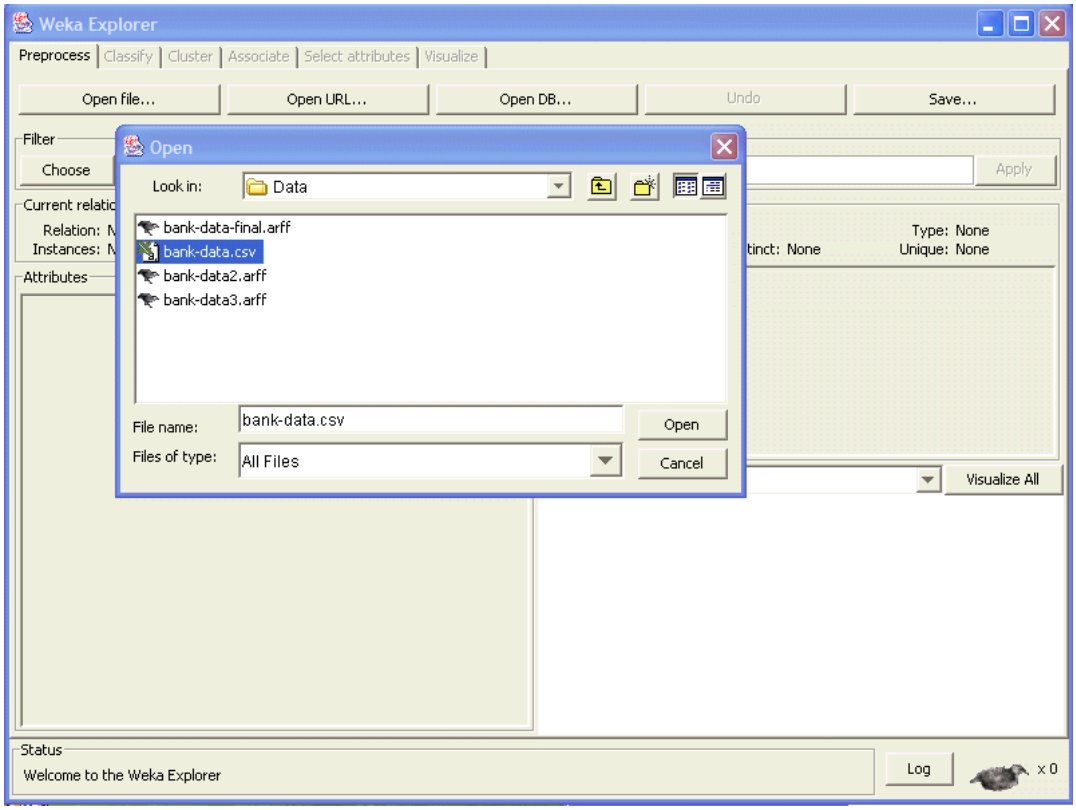


Figure 5 – How to Select a Data File

Then, if one of the classifier algorithm is going to operated, the Classify tab has to be selected and by "Choose" button in that tab desired algorithm is chosen. To gain proper results various parameters of algorithm is also have to be specified. These can be specified by clicking in the text box to the right of the "Choose" button. Assuming that J48 classifier is chosen to perform classification operation on dataset, default values of parameters in its interface is shown in figure below. After specifying parameters “Start” button is clicked to run the algorithm.

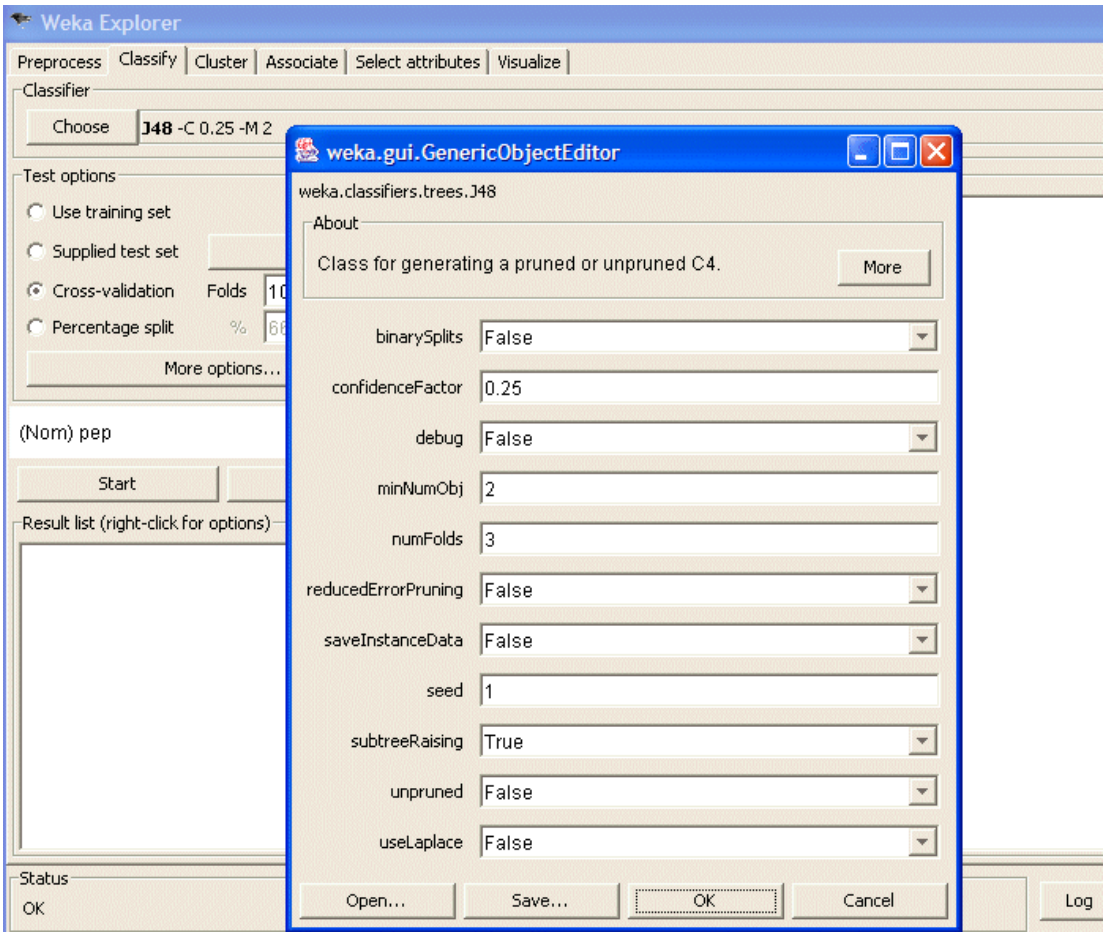


Figure 6 – Generating a classifier object

In clustering algorithm case, everything quite similar to the classifier algorithm initialization. Small and easy to understand differences exist on tab menus like parameters menu.

5.3. Planned Additions to User Interface

We will add some options to the Classify and Cluster tabs mainly in our interface design. These tabs will have an option which offers two ways to run an algorithms on datasets. One of them will be normal run of algorithms, in other words task is accomplished by one single process on one computing node and in another one parallelization will be used or task is distributed across different parallel computing nodes.

Figure 7 – Our planned improvements on WEKA Gui

In these tabs user will also be able to specify the number of computing nodes which will operate parallelly on running algorithm, if parallelization option is chosen. The default value of number of nodes will be 2, and can be incremented to a suitable number that we are going to identify after experimental tests. The figures above are rough visualizations of these options that will be placed to the tabs in user friendly way.

6. SPECIFICATIONS AND DEPENDENCIES

In this section we show the specifications , dependencies and the tools we use during development of our project *Wekarel.*

6.1. Specifications

In our project we use Java programming language so we must obey Java's syntax specifications, however this is not enough. We also need to obey the specifications of MPI and OpenMP or the java related counterparts of that technologies. Of course in future we may use some other technologies like java parallel, we should obey its specifications too. Since we use Weka and we will be very familiar with use of weka; we would like to mention about Attribute-Relation File Format (ARFF) which is special to Weka . Because obtaining data sets, using ARFF files is one of the most convenient ways.

6.1.1 Attribute-Relation File Format

An ARFF (Attribute-Relation File Format) file is an ASCII text file that describes a list of instances sharing a set of attributes. ARFF files have two distinct sections. The first section is the **Header** information, which is followed the **Data** information. The **Header** of the ARFF file contains the name of the relation, a list of the attributes (the columns in the data), and their types. An example header on a standard dataset looks like this:

```
%  Title: Iris Plants Database
   @RELATION iris

   @ATTRIBUTE sepallength  NUMERIC
   @ATTRIBUTE sepalwidth   NUMERIC
   @ATTRIBUTE petallength  NUMERIC
   @ATTRIBUTE petalwidth   NUMERIC
   @ATTRIBUTE class        {Iris-setosa,Iris-versicolor,Iris-virginica}
```

The **Data** of the ARFF file looks like the following:

```
@DATA
   5.1,3.5,1.4,0.2,Iris-setosa
   4.9,3.0,1.4,0.2,Iris-setosa
   4.7,3.2,1.3,0.2,Iris-setosa
   4.6,3.1,1.5,0.2,Iris-setosa
   5.0,3.6,1.4,0.2,Iris-setosa
   5.4,3.9,1.7,0.4,Iris-setosa
```

Lines that begin with a % are comments. The **@RELATION**, **@ATTRIBUTE** and **@DATA** declarations are case insensitive. More information about that file format can be found at the related wiki page[5].

6.2. Dependencies

For the dependencies, we introduce the tools we are going to use in our project development. We can divide our dependencies into two parts; hardware dependencies and software dependencies.

6.2.1. Hardware Dependencies

Related to our project, we will use the cluster named NAR in our department. Thanks to our instructors for supplying us that valuable hardware. To illustrate the capabilities of the NAR below some system properties of hardware is stated. For more detailed information you can visit hpc hardware website[6].

Computational and Storage capacity of NAR's system can be summarized as following:

- 46 x 2 = 92 CPUs
- 46 x 2 x 4 = 368 Cores
- 46 x 16 GB = 736 GB Memory
- 46 x 146 GB = 6.5 TB Local Disk (halved by RAID)
- 2 x 3 TB = 6 TB Common Storage Area (halved by RAID)

6.2.2. Software Dependencies

Here, we are going to mention about our software dependencies. The softwares we think to use so far are all open source and free softwares. We introduce the softwares we use in our project development process below.

a. Primary Softwares

Here, we introduce the primary software that we have investigated and we plan to use for the project development.

Weka Tool:

Weka is a collection of machine learning algorithms for data mining tasks. The algorithms can either be applied directly to a dataset or called from your own Java code. Weka contains tools for data pre-processing, classification, regression, clustering, association rules, and visualization. It is also well-suited for developing

new machine learning schemes.

Since Weka is a java project and we have chosen to study with java, java is the other tool that we need to use.

Weka can be imported as a project from Eclipse and Netbeans. It is beneficial to use that method while developing Weka and investigating the source code. Thanks to Weka Wiki; there are articles on how to add source code as a project in Eclipse[7] and Netbeans[8].

MPI:

Message Passing Interface (MPI) is a specification for an API that allows many computers to communicate with one another. It is used in computer clusters and supercomputers. MPI was created by William Gropp and Ewing Lusk and others.

MPI is a language-independent communications protocol used to program parallel computers. Both point-to-point and collective communication is supported. MPI "is a message-passing application programmer interface, together with protocol and semantic specifications for how its features must behave in any implementation." according to William Gropp. MPI's goals are high performance, scalability, and portability. MPI remains the dominant model used in high-performance computing today. (High-performance and scalable MPI over InfiniBand with reduced memory usage)

OpenMP:

The OpenMP (Open Multi-Processing) is an application programming interface (API) that supports multi-platform shared memory multiprocessing programming in C/C++ and Fortran on many architectures, including Unix and Microsoft Windows platforms. It consists of a set of compiler directives, library routines, and environment variables that influence run-time behavior.

Jointly defined by a group of major computer hardware and software vendors,

OpenMP is a portable, scalable model that gives programmers a simple and flexible interface for developing parallel applications for platforms ranging from the desktop to the supercomputer.

An application built with the hybrid model of parallel programming can run on a computer cluster using both OpenMP and Message Passing Interface (MPI).

OpenMP is an implementation of multi-threading, a method of parallelization whereby the master "thread" (a series of instructions executed consecutively) "forks" a specified number of slave "threads" and a task is divided among them. The threads then run concurrently, with the runtime environment allocating threads to different processors.[9]

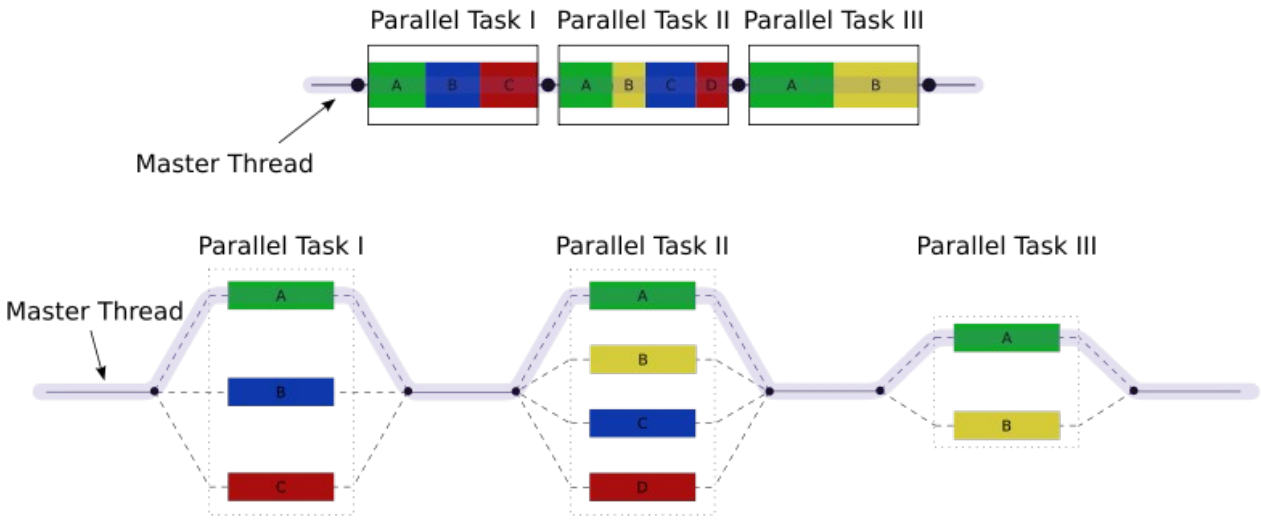


Figure 8 - An illustration of multithreading where the master thread forks off a number of threads which execute blocks of code in parallel

Java and Java Related Technologies:

During our project development process we will use java and we will benefit the capabilities of java such as default build in multi threading properties of java. We will also use some parallel programing technologies with java such as mpi and openmp there are already projects on this topic.

- Java and MPI

Although Java does not have an official MPI binding, there have been several attempts to bridge Java and MPI, with different degrees of success and compatibility. One of the first attempts was Bryan Carpenter's mpiJava, essentially a collection of JNI wrappers to a local C MPI library, resulting in a hybrid implementation with limited portability, which also has to be recompiled against the specific MPI library being used.

However, this original project also defined the mpiJava API (a de-facto MPI API for Java following the equivalent C++ bindings closely) which other subsequent Java MPI projects followed. An alternative although less used API is the MPJ API, designed to be more object-oriented and closer to Sun Microsystems' coding conventions. Other than the API used, Java MPI libraries can be either dependent on a local MPI library, or implement the message passing functions in Java, while some like P2P-MPI also provide Peer to peer functionality and allow mixed platform operation.

Some of the most challenging parts of any MPI implementation for Java arise from the language's own limitations and peculiarities, such as the lack of explicit pointers and linear memory address space for its objects , which make transferring multi-dimensional arrays and complex objects inefficient. The workarounds usually used involve transferring one line at a time and/or performing explicit de-serialization and casting both at the sending and receiving end, simulating C or FORTRAN-like arrays by the use of a one-dimensional array, and pointers to primitive types by the use of single-element arrays, thus resulting in programming styles quite extraneous from Java's conventions.

One major improvement is MPJ Express by Aamir Shafi. This project was supervised by Bryan Carpenter and Mark Baker. On commodity platform like Fast Ethernet, advances in JVM technology now enable networking programs written in Java to rival their C counterparts. On the other hand, improvements in specialized

networking hardware have continued, cutting down the communication costs to a couple of microseconds. Keeping both in mind, the key issue at present is not to debate the JNI approach versus the pure Java approach, but to provide a flexible mechanism for programs to swap communication protocols. The aim of this project is to provide a reference Java messaging system based on the MPI standard. The implementation follows a layered architecture based on an idea of device drivers. The idea is analogous to UNIX device drivers.[10]

In order to figure out how mpiJava works and how much does it assemble standard MPI, below you can see an easy Java sample named Hello:

```
      import mpi.* ;  // including the MPI library
// #include "mpi.h"  this is the C style  inclusion of MPI
class Hello
{
  static public void main(String[] args) throws MPIException
    {
     MPI.Init(args) ; // initialization of MPI environment
     //MPI_Init(&argc, &argv); this is C style initialization
    int myrank = MPI.COMM_WORLD.Rank() ; // getting rank number of each process
//MPI_Comm_rank(MPI_COMM_WORLD, &rank);  C style getting rank number
          if(myrank == 0)
          {
             char [] message = "Hello, there".toCharArray() ;
             MPI.COMM_WORLD.Send(message, 0, message.length, MPI.CHAR, 1, 99) ; // sending a message  with MPI call
```

```
            //MPI_Send(&num, 1, MPI_INT, next, tag, MPI_COMM_WORLD);
sending a message //with C style MPI call (arguments are garbage do not care them)
          }
          else
            {
             char [] message = new char [20] ;
             MPI.COMM_WORLD.Recv(message, 0, 20, MPI.CHAR, 0, 99) ; // recieving a message with MPI call
              //MPI_Recv(&num, 1, MPI_INT, from, tag, MPI_COMM_WORLD, &status); recieving a message with C style MPI call
            System.out.println("received:" + new String(message) + ":") ;
             }
        MPI.Finalize(); // finalizing MPI environment
        // MPI_Finalize(); finalizing MPI environment in C
        }
  }
```

Above we made **bold** the mpiJava calls to differentiate them from C like MPI calls. We also wrote the c style calls just below the counterpart mpiJava calls to easily see the difference of Java and C MPI calls. We have practiced some mpiJava program on NAR successfully. It relieved us about being afraid of having trouble that whether mpiJava would run on NAR or not. Thanks to our T.A Celebi Kocair for his interest and help about configuring mpiJava on NAR.

- Java and OpenMP

OpenMP is the widely used technology for parallel programing on shared memory architectures. However OpenMP does not directly support Java programing

language. Since Java is one of widely used programing language, naturally there has been projects that investigate relation of Java and OpenMP. JOMP is the mostly known and used technology.

JOMP is a research project whose goal is to define and implement an OpenMP-like set of directives and library routines for shared memory parallel programming in Java.

It is, of course, possible to write shared memory parallel programs using Java's *native threads model*, but a directive system has a number of advantages. It is considerably easier to use, less error prone and allows compatibility to be maintained with a sequential version.[11]

b. Alternative Technologies

In this part, we are going to mention about some alternative softwares that we may use in our future work.

Intel® Trace Analyzer and Collector: :

In order to analyze our project's performance and check MPI correctness for our development we mind to use Intel Trace Analyzer and Collector. To introduce that useful software we describe some of its properties below:

- Intuitive full color customizable GUI with many drill down view options
- Highly scalable with low overhead and efficient memory usage
- Easy run-time loading - or instrument an MPI application executable
- MPI Correctness Checking Library detects many types of errors in communication
- Integrated online help
- Easy installation and full documentation
- Full tracing and/or light-weight statistics gathering

- Runtime behavior displayed by function, process, thread, timelines or cluster or node
- Multiple types of filtering (functions, processes, messages) and aggregation
- Performance counter data recording can be displayed as timeline
- Automated MPI tracing and MPI Correctness Checking
- Generic distributed (non-MPI) and single process tracing
- Thread level tracing with traces created even if the application crashes

Intel Trace Analyzer and Collector provides us many useful functionalities. However we are mostly interested in MPI checking property. So we can analyze how we have fasten our development and where we have made bottleneck and logical errors.

Included in Intel Trace Analyzer and Collector is a unique MPI correctness checker to detect deadlocks, data corruption, or errors with MPI parameters, data types, buffers, communicators, point-to-point messages and collective operations. By providing checks at run-time, and reporting the errors as they are detected, the debugging process is greatly expedited. The correctness checker also allows debugger breakpoints to help in-place analysis but has a small enough footprint to allow use during production runs. The true benefit of the Intel Trace Analyzer and Collector Correctness Checker is the potential to scale to extremely large systems and the ability to detect errors even among a large number of processes. The checker can be set to view deadlocks regardless of fabric type.

By tracking data types and wrapping MPI calls, the requests and communicators can be reused from the trace collector. (The checking library is compiled from the source code of the performance data collection library.) The Analyzer is able to extremely rapidly unwind the call stack and use debug information to map instruction addresses to source code with and without frame pointers.

With both command line and GUI interfaces the user can additionally set up batch runs or do interactive debugging. The timeline view shows actual function calls and process interactions which highlights excessive delays or errors that stem from improper execution ordering.

Since Intel Trace Analyzer and Collector has been tested on nar. We can use it. However it is a commercial product and we can 30 day free of charge. Yet we hope that our department will help us about usage of that useful tool.

To illustrate how Intel Trace Analyzer and Collector works, We supply a figure below:

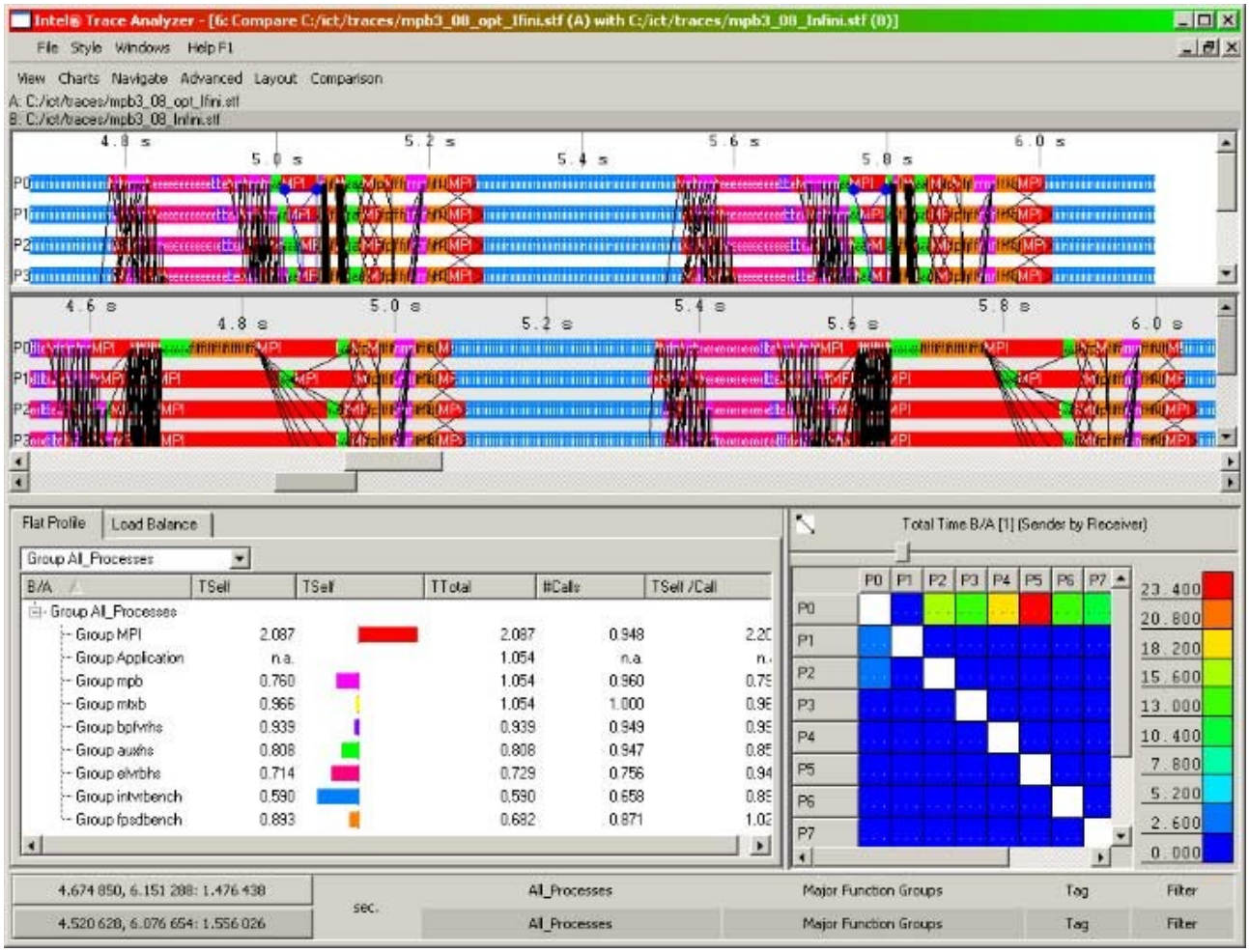


Figure 9 - Intel® Trace Analyzer and Collector

Hadoop :

Hadoop is a framework for running applications on large clusters built of commodity hardware. The Hadoop framework transparently provides applications both reliability and data motion. Hadoop implements a computational paradigm named Map/Reduce, where the application is divided into many small fragments of work, each of which may be executed or reexecuted on any node in the cluster. In addition, it provides a distributed file system (HDFS) that stores data on the compute nodes, providing very high aggregate bandwidth across the cluster. Both Map/Reduce and the distributed file system are designed so that node failures are automatically handled by the framework.[12]

MapReduce: Simplified Data Processing on Large Clusters :

MapReduce is a programming model and an associated implementation for processing and generating large data sets. Users specify a map function that processes a key/value pair to generate a set of intermediate key/value pairs, and a reduce function that merges all intermediate values associated with the same intermediate key.

Programs written in this functional style are automatically parallelized and executed on a large cluster of commodity machines. The run-time system takes care of the details of partitioning the input data, scheduling the program's execution across a set of machines, handling machine failures, and managing the required inter-machine communication. This allows programmers without any experience with parallel and distributed systems to easily utilize the resources of a large distributed system.[13]

7. DEVELOPMENT SCHEDULE

During the development of the initial design report, the tasks of Wekarel project have gradually arised. A tentative schedule, which gives information about our past, present and future work, has been defined and the well-balanced

assignments of the tasks are going to be made for the second term.

Our tasks can be categorized into two parts: Project management, which includes tasks related to the project documentation, and the paralellization & testing of the classes. We are also considering some alternative approaches to our parallelization work that we may consider and switch some of these methods in our future development roadline. These approaches can be found in the Alternative Technologies part under Software Dependencies title.

7.1. Project Management

The foundations of general software architecture is laid. Relevant class hierarchies and data design hierarchies are constructed. These design considerations are discussed in relevant sections in great detail. A yearlong development schedule has been prepared (See Gannt chart given in Appendix part). During these processes, some factors have helped our development line:

7.1.1. Proposal Report

A short report about project specifications about what we have aimed to achieve at the end of the term. The motivation, purpose and initial estimates and intentions are included in this report. The proposal report has been completed.

7.1.2. Requirement Analysis Report

A report including system requirement specifications represented formally. This report also includes information about our work packages, our literature survey and work calendar. The requirement analysis report has been completed.

7.1.3. Initial Design Report

A report including the formal specification of our system solution. This report includes descriptions of system modules, data flow, state diagrams, syntax

specifications, development schedule etc. depending on our project and methodology. This is the report that brought an experience on us about the Weka code and parallelization methods. Also, our initial design report is the documentary that enabled people like our TA, instructors and Weka developers to give us more feedback, especially on technical issues. The initial design report has been completed

7.1.4. Final Design Report

A report including a detailed information about the whole system, classes, parallelization methods and project plans. It is the latest documentation that warms up us before coding. This is the report you are reading.

7.1.5. Seminars

In order to be prepared for the project, our department has supplied us a series of seminars according to the schedule[14]. These seminars included parts about system basics & mpi basics, openmp, PBS internals, Anvanced MPI skills and Multithreading with Pthread examples.

7.1.6. Intel HPC Workshop

In order to warm up students about computational science methodologies and parallel programming tools Intel has conducted a 3 day workshop in our department between 7 and 9 January. One of our group students (Haşim) has attended the workshop. During the workshop instructors (Prof. Tom Murphy, chair of Contra Costa College Science program, Ass. Prof. Charlie Peck, Earlham College Computer Science, Werner Krotz Wogel,Intel Cluster Software Technologies group) focused on the following issues:

- Parallel, Distributed, and Grid Computing
- Using large scale computational resources
- MPI and OpenMP

- Intel toolchain and libraries related MPI and OpenMP
- GNU toolchain and libraries
- Performance Analysis
- Debugging with the GNU

During the workshop attendees also had a chance to make practice after each issue. Thanks to our department for supplying us such valuable events.

7.1.7. Weka Homepage, Wiki and Mailing Lists

Weka has a great homepage[15] giving detailed information about project, data mining and sources for getting help. In the “documentation” part of the site, links for tutorials on seperate topics, mailing list[16] and its archive, wiki located at Sourceforge.net are listed. Tracking the mailing list helped us catching the live development roadmap, getting latest news and grabbing interesting footnotes about the project. Weka – wiki[17] has also many beneficial articles explaining inner dynamics of Weka design.

These factors have been covered in requirement analysis report in details so only little information on them have been comprised.

7.1.8. Communication with Weka Developers

During the development process, we contacted with some of WEKA developers; Peter Reutemann and Mark Hall. Mr. Reutemann is the contributor who answers most of the questions in Weka Mailing List during our first semestre. He said about our project *“Nice work”* and directed us to the maintainer of WEKA code: Mark Hall. Mr. Hall inspected our initial design report and gave feedbacks to us. His opinion about our project is:

"I've taken a quick look over your design report. It is very well written and outlines your plans clearly. I must admit that the project sounds ambitious (but then I'm no expert on parallel methods). I do know that some ML techniques lend themselves naturally to parallel implementations (meta learning methods such as bagging spring to mind) and there are papers on parallel decision tree construction and rule learning. Parallelization of other learning techniques can be non-trivial and is the subject of current research."

His words brought encourage and happiness to us and thrilled our efforts.

7.2. Parallelization and Testing of the Classes

These tasks constitute the major part of what we are going to do next semester. The tasks are also mentioned in the Gantt Chart at the Appendix part B. It should be pointed out that the development schedule of the 2nd Semestre is tentative, with respect to TA and ET, it can change due time.

7.2.1. Paralellization of Instances Class

- Parallelization of overriden methods in ParallelInstances class : these methods include Instances(), attributeToDoubleArray(), variance(), sort(), sumOfWeights(), numDistinctValues(), mergeInstances() and meanOrMode().
- Testing & improvement of ParallelInstances class

7.2.2. Paralellization of Utils Class

- Parallelization of overriden methods in ParallelUtils class : these methods include maxIndex(), minIndex(), sort(), sum(), normalize() and variance().
- Testing & improvement of ParallelInstances class

7.2.3. Paralellization of SimpleKMeans Class

- Parallelization of buildClusterer() method

- Testing & improvement of ParallelKMeans class

7.2.4.Paralellization of ClusterEvaluation Class

- Parallelization of evaluateClusterer() method
- Testing & improvement of ParallelClusterEvaluation class

7.2.5.Paralellization of SimpleNaiveBayes Class

- Parallelization of distributionForInstance() method
- Parallelization of classifyInstance() method
- Parallelization of buildClassifier() method
- Testing & improvement of ParallelNaiveBayes class

7.2.6.Paralellization of Evaluation Class

- parallelization of evaluateModel() method
- parallelization of setPriors() method
- testing & improvement of ParallelEvaluation class

8. Conclusion

Initial design report was a milestone for our design project, as it was the most well – documented piece of paper we have written so far. Another advantage of initial design report is the contents revealing technical and practical information about the whole process. However, it wasn't enough to start coding. Some more detailed investigations were needed in order to start the coding part.

With these notions and intentions like deepening our interest on the Weka code and our project, we started writing final design report. More detail, tips and tricks have been mentioned in final design report.

This report is believed to be our guideline and the accelerator for the forthcoming works. We also believe that this report will give detailed information to those who are planning to contribute to Weka or any other Machine Learning Software's parallelization process.

There is not much that have been changed since initial design report. More additions and consciousness adjoined into the project plans as specs are more detailed and parallelization methods are deeply enrolled in this final design report; that's why final design report is believed to be our peak in designing – before – coding period.

We were unsure about where to parallellize and what kind of test to implement before we started writing initial design report. Detailed investigations for writing initial design report have helped us in seizing the situation in a better point of view. In addition to these investigations, more and more realizations while preparing the final design report increased our self-awareness about the project. Now, we are ready to start coding and improve our implementation according to the results of testing parameters.

Packages were determined and selected in initial design report. In the final design report we improved our design and added necessary information about the *to_be_parallelized* packages. Also, dfd diagrams are improved and the latest design decisiond for the data flow are reflected.

To summarize, we are proud to announce that our final design report is finished and we are ready to start the coding process of the parallelization attempt on WEKA, *Wekarel.* The latter parts of the project, source code, scicomp team blog and other features are going to be implemented as soon as possible at group's web site[18].

9. References

[1] http://weka-parallel.sourceforge.net/

[2] In 2005, Weka receives the SIGKDD Data Mining and Knowledge Discovery Service Award: Gregory Piatetsky-Shapiro (2005-06-28). "KDnuggets news on SIGKDD Service Award 2005". Retrieved on 2007-06-25.

[3] "Overview of SIGKDD Service Award winners" (2005). Retrieved on 2007-06-25.

[4] Weka API documentation – http://weka.sourceforge.net/doc

[5] Details about ARFF format can be found here:

http://weka.wiki.sourceforge.net/ARFF+(book+version)

[6] HPC hardware page: http://hpc.ceng.metu.edu.tr/sistem/hardware/

[7] How to setup a Weka environment in Eclipse:

http://weka.wiki.sourceforge.net/Eclipse

[8] How to setup a Weka environment in Netbeans:

http://weka.wiki.sourceforge.net/NetBeans

[9] http://en.wikipedia.org/wiki/OpenMP

[10] http://en.wikipedia.org/wiki/Message_Passing_Interface#cite_note-0

[11] http://www2.epcc.ed.ac.uk/computing/research_activities/jomp/index_1.html

[12] http://wiki.apache.org/hadoop/

[13] http://labs.google.com/papers/mapreduce.html

[14] http://web.ceng.metu.edu.tr/~ketenci/schedule.htm

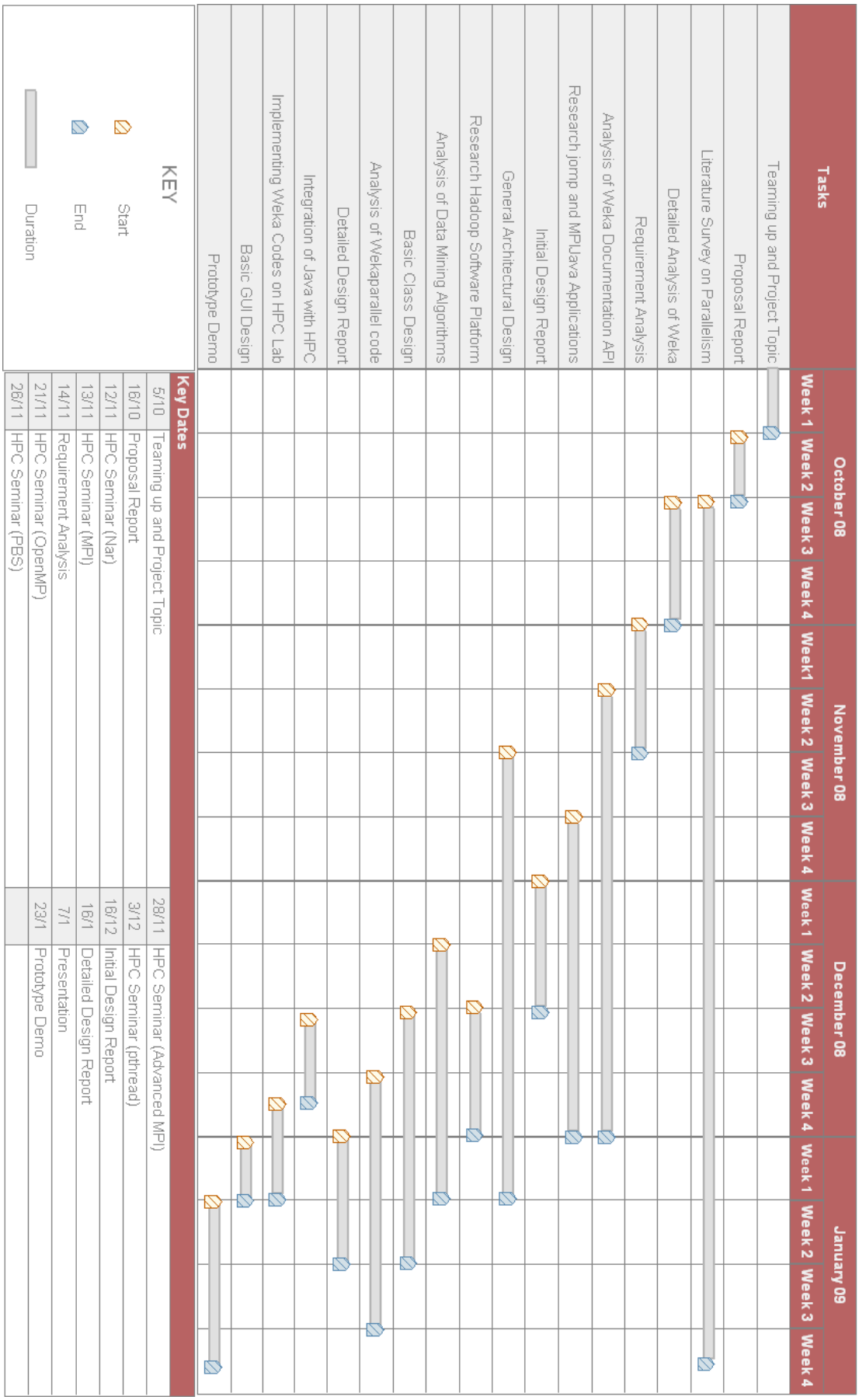
[15] Weka Homepage http://www.cs.waikato.ac.nz/~ml/weka/

[16] Weka Mailing List
https://list.scms.waikato.ac.nz/mailman/listinfo/wekalist

[17] Weka Wiki http://weka.wiki.sourceforge.net/

[18] SciComp Group's Web Page http://senior.ceng.metu.edu.tr/2009/scicomp/

10. Appendix A – Gannt Chart 1



11. Appendix B – Gannt Chart 2