MIDDLE EAST TECHNICAL UNIVERSITY

COMPUTER ENGINEERING DEPARTMENT

Initial Design Report

Wekarel

A Parallel Weka Implementation

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

Table Of Contents

1. INTRODUCTION....4
    1.1. Project Title....4
    1.2. Problem Definition....4
    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....6
    2.4. Experience on Data Mining and Parallelization Software....7
3. DETAILED DESIGN....7
    3.1. General Information & Parallelization Decisions....7
    3.2. Packages, Classes and Inheritance Diagrams 8
        3.2.1. weka.core package....8
        3.2.2 weka.clusterers package....10
        3.2.3. weka.classifiers package....13
        3.2.4. weka.classifiers.bayes package....15
    3.3. Class Diagram....17
4. DATA DESIGN....18
    4.1. Function Modelling....18
        4.1.1 Level 0 of DFD....18
        4.1.2 Level 1 of DFD....19
        4.1.3. Explanation of Level 1 DFD....20
        4.1.4. Level 2 DFD....21
        4.1.5. Explanation of Level 2 DFD....22
5. GUI DESIGN....23
    5.1. GUI of Weka....23
    5.2. Interface Usage....23
    5.3. Planned Additions to User Interface....26
6. SPECIFICATIONS AND DEPENDENCIES....26
    6.1 Specifications....27
    6.2 Dependencies....28
        6.2.1 Hardware Dependencies....28
        6.2.2 Software Dependencies....28

7. DEVELOPMENT SCHEDULE....34
7.1 Project Management....35
7.1.1 Proposal Report 35
7.1.2 Requirement Analysis Report 35
7.1.3 Initial Design Report 35
7.1.4 Seminars 35
7.1.5 Weka Homepage, Wiki and Mailing Lists :....36
7.2 Parallelization and Testing of the Classes....36
7.2.1 Paralellization of Instances Class....36
7.2.2 Paralellization of SimpleKMeans Class....36
7.2.3 Paralellization of ClusterEvaluation Class....37
7.2.4 Paralellization of SimpleNaiveBayes Class....37
7.2.5 Paralellization of Evaluation Class....37
8. Conclusion....37
9. References....38
10. Appendix A – Gannt Chart 1....40
11. Appendix B – Gannt Chart 2....41

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 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 spesifications 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 & Parallelization Decisions

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 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.

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 that kind of evaluations. It seems the most convenient strategy to involve all the nodes in the scope of a communication is to use collective communation. The data will be distributed from one node to all of the nodes, and will be gathered at the same node.

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.

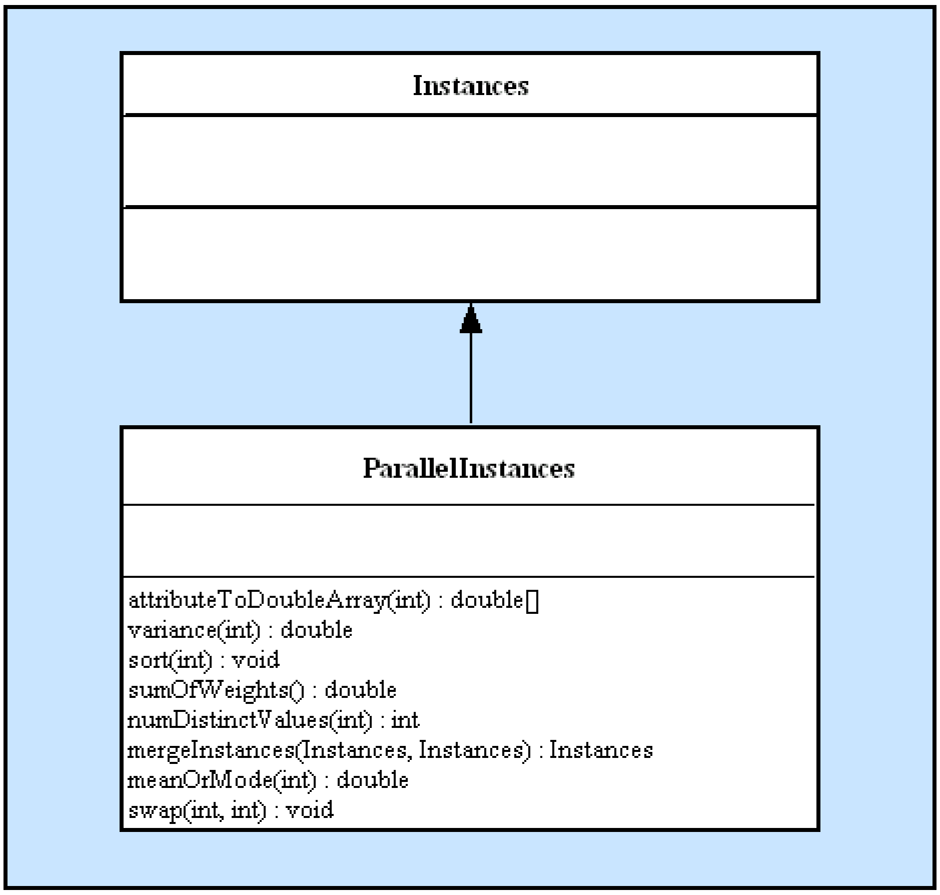
3.2.1.2. Attribute Class

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.
`void swap(int i, int j)`	Swaps two instances in the set.
`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.

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

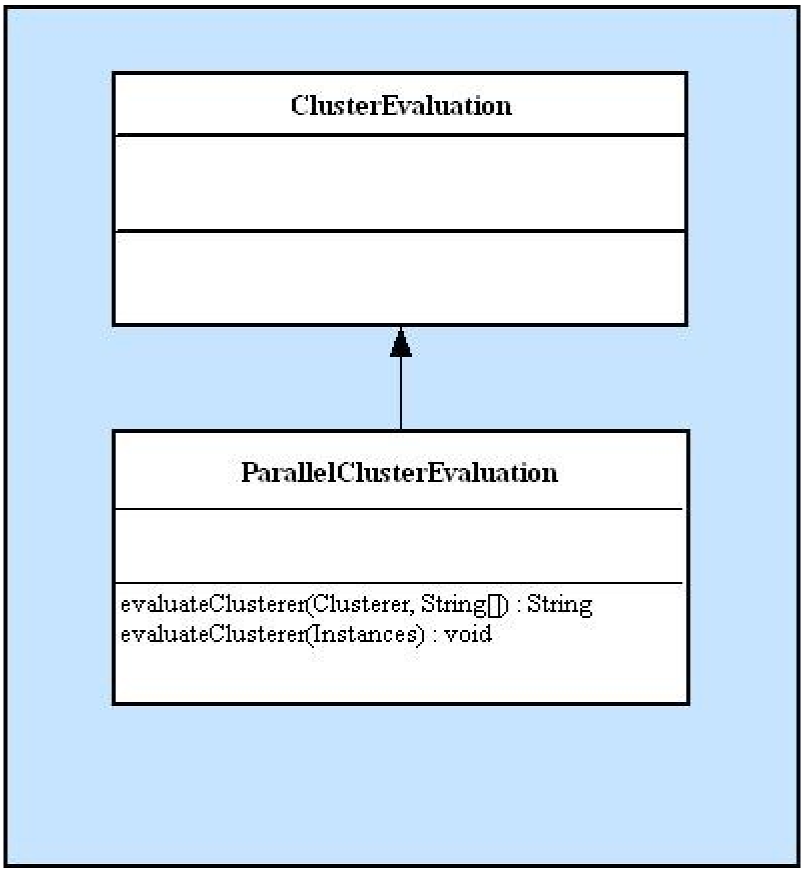
This is the class which evaluates the clustering model with the given parameters via its evaluateClusterer method. This is one of the key classes we are going to focus on parallelling. Arranging the data set and filling the parameters of the clusters before evaluating the algorithms happen within this 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.

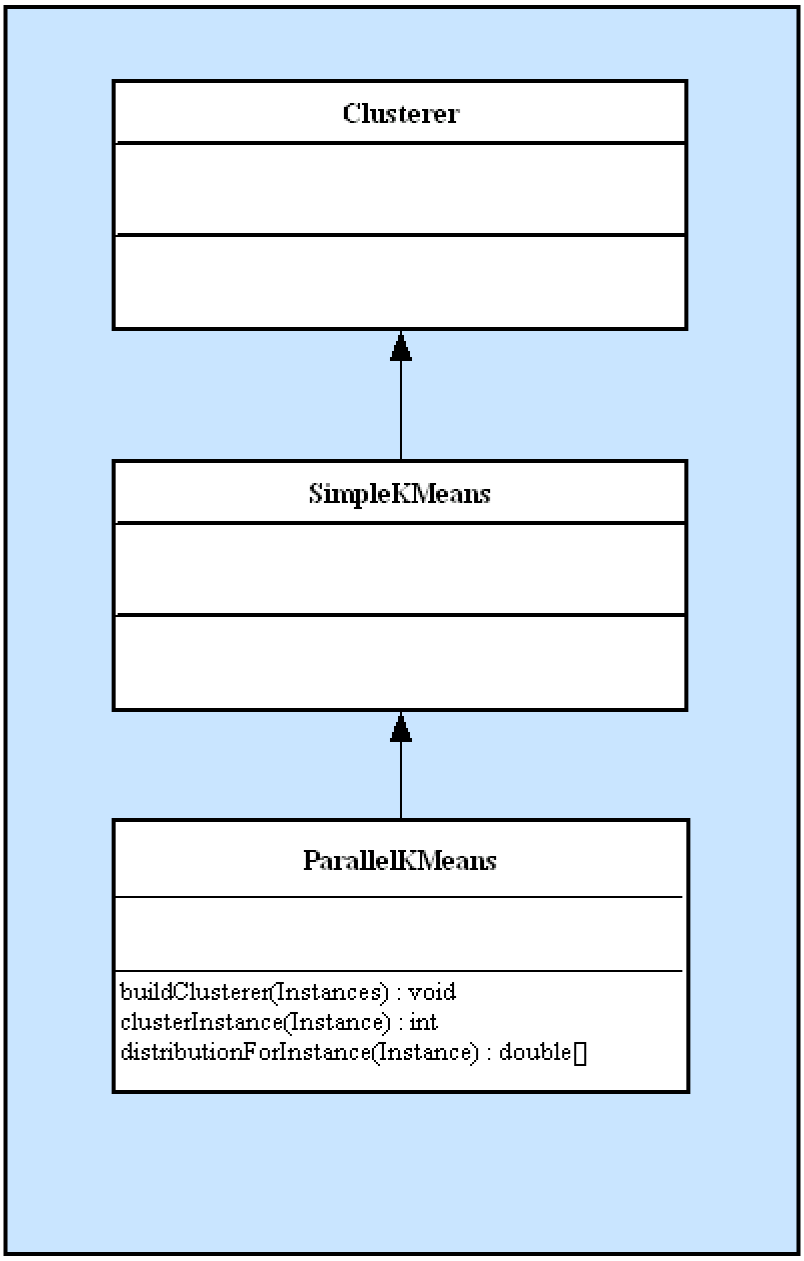
3.2.2.4 Inheritance Diagram for ParallelClusterEvaluation Class


ClusterEvaluation
ParallelClusterEvaluation
evaluateClusterer(Clusterer, String[]) : String
evaluateClusterer(Instances) : void

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


Clusterer
SimpleKMeans
ParallelKMeans
buildClusterer(Instances) : void
clusterInstance(Instance) : int
distributionForInstance(Instance) : double[]

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.
`int clusterInstance(Instance instance)`	Classifies the given instance.
`double[] distributionForInstance(Instance instance)`	Predicts the cluster memberships for a given instance.

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

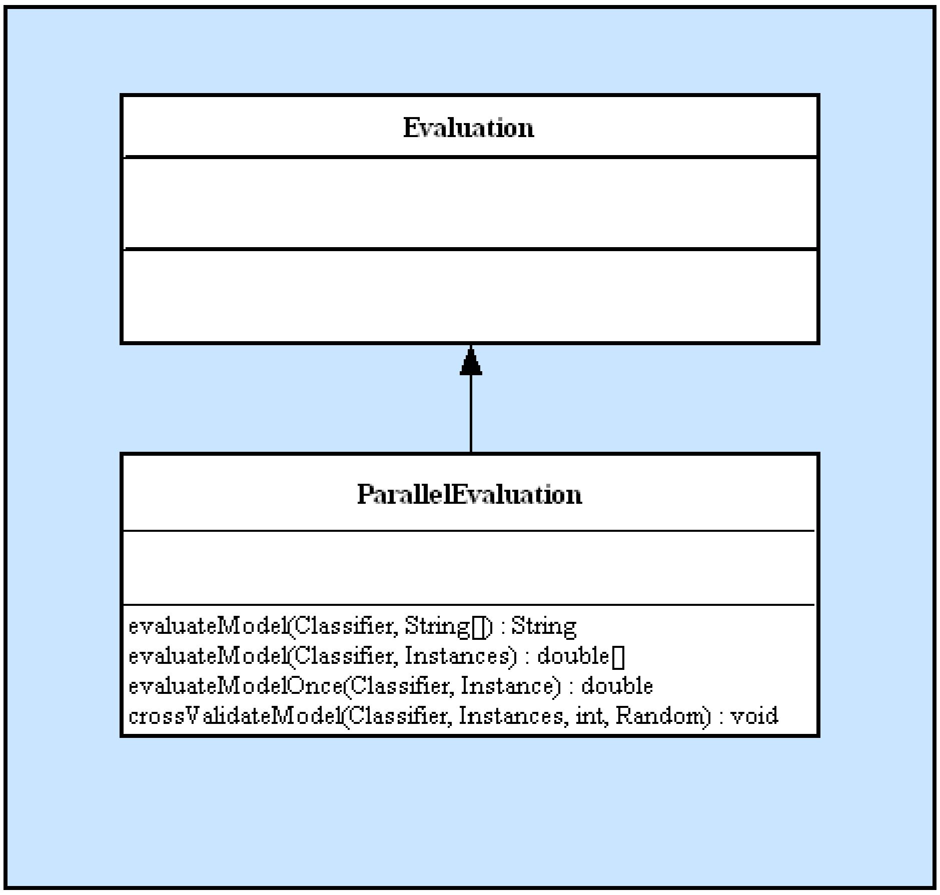
It is the abstract class whose abstract methods are to be implemented by the classes generating the classifying algorithm (NaiveBayesSimple class in our proof of concept case). There are methods in this class for

- 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


Evaluation
ParallelEvaluation
evaluateModel(Classifier, String[]) : String
evaluateModel(Classifier, Instances) : double[]
evaluateModelOnce(Classifier, Instance) : double
crossValidateModel(Classifier, Instances, int, Random) : void

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

ParallelEvaluation	
`void crossValidateModel(Classifier classifier, Instances data, int numFolds, Random random)`	Performs a cross-validation for a classifier on a set of instances.
`String evaluateModel(Classifier classifier, String[] options)`	Evaluates a classifier with the options given in an array of strings.
`double[] evaluateModel(Classifier classifier, Instances data)`	Evaluates the classifier on a given set of instances.
`double evaluateModelOnce(Classifier classifier, Instance instance)`	Evaluates the classifier on a single instance.

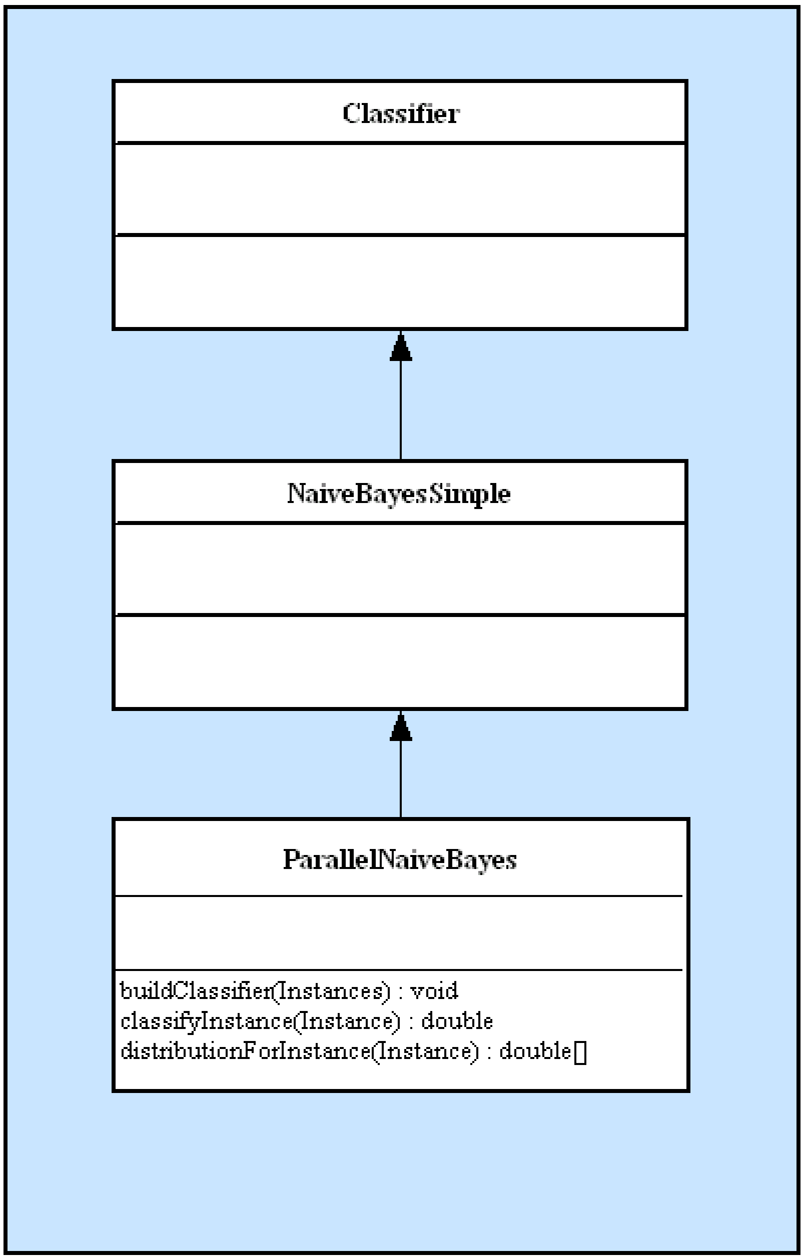
3.2.4. weka.classifiers.bayes package

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.

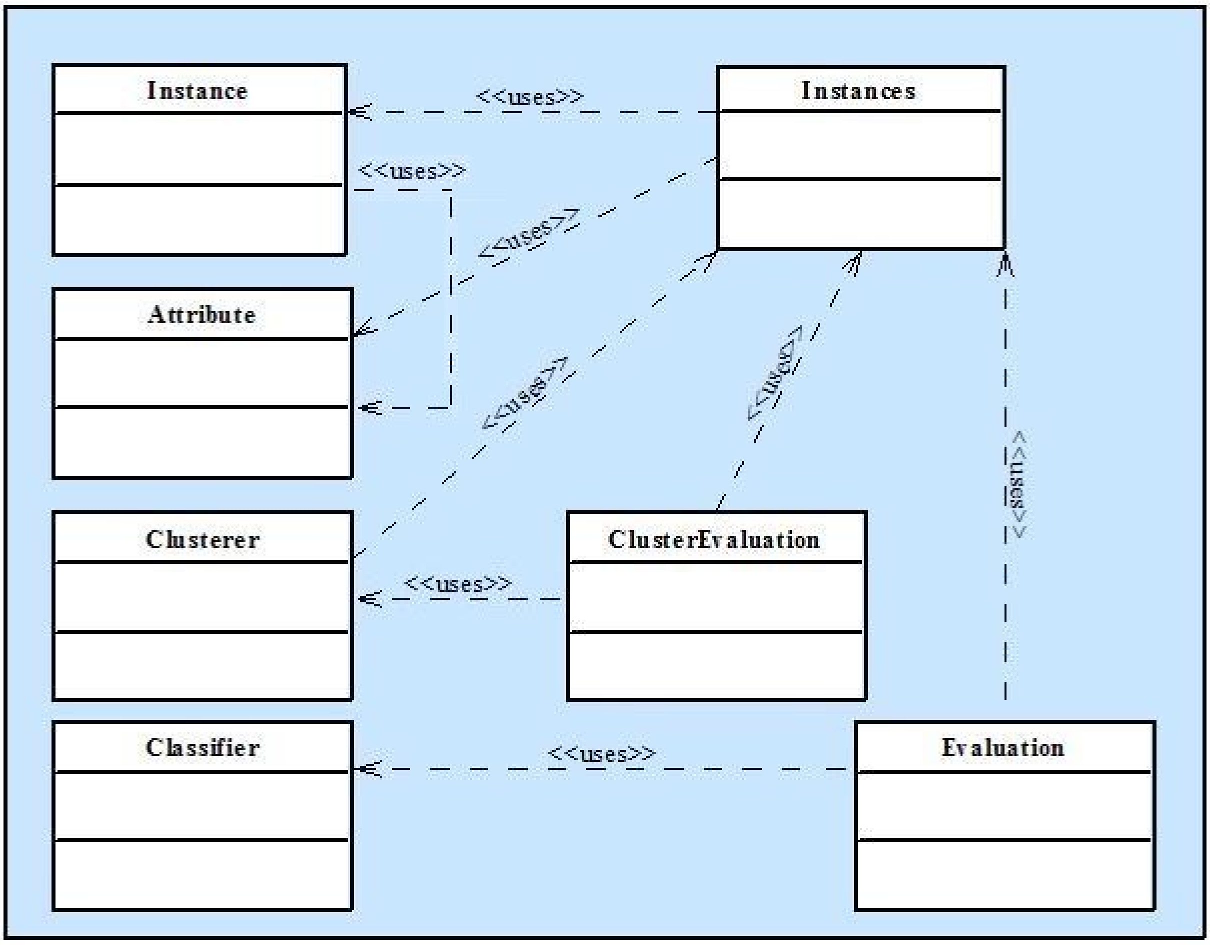
3.2.4.2 Inheritance Diagram for ParallelNaiveBayes Class


Classifier
NaiveBayesSimple
ParallelNaiveBayes
buildClassifier(Instances) : void
classifyInstance(Instance) : double
distributionForInstance(Instance) : double[]

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.

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

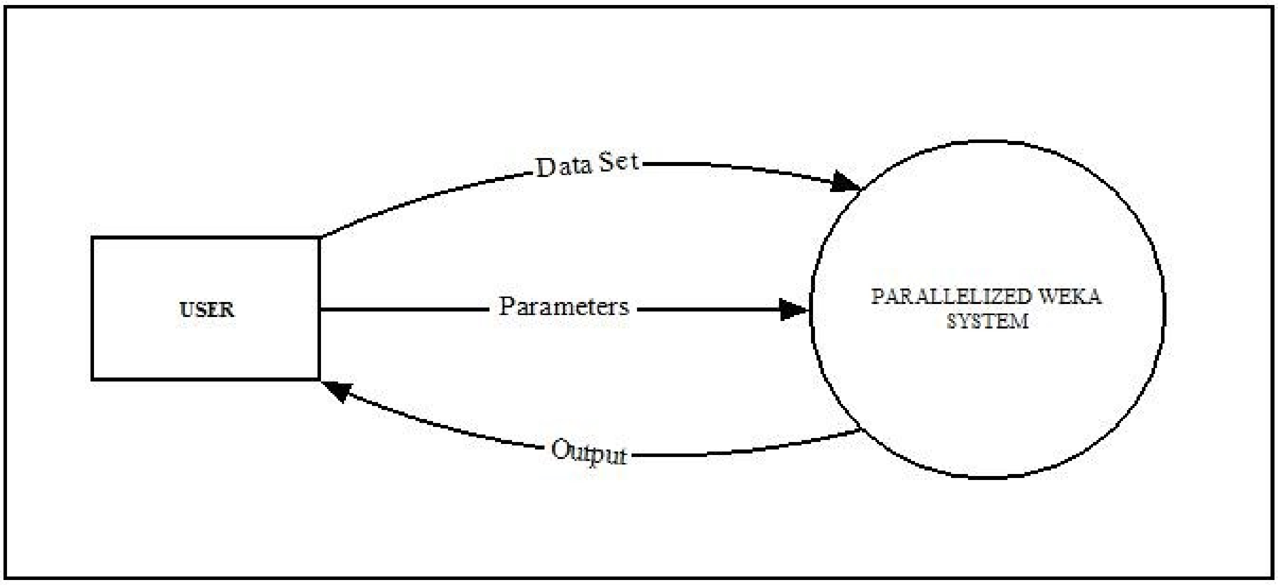


Figure 1

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

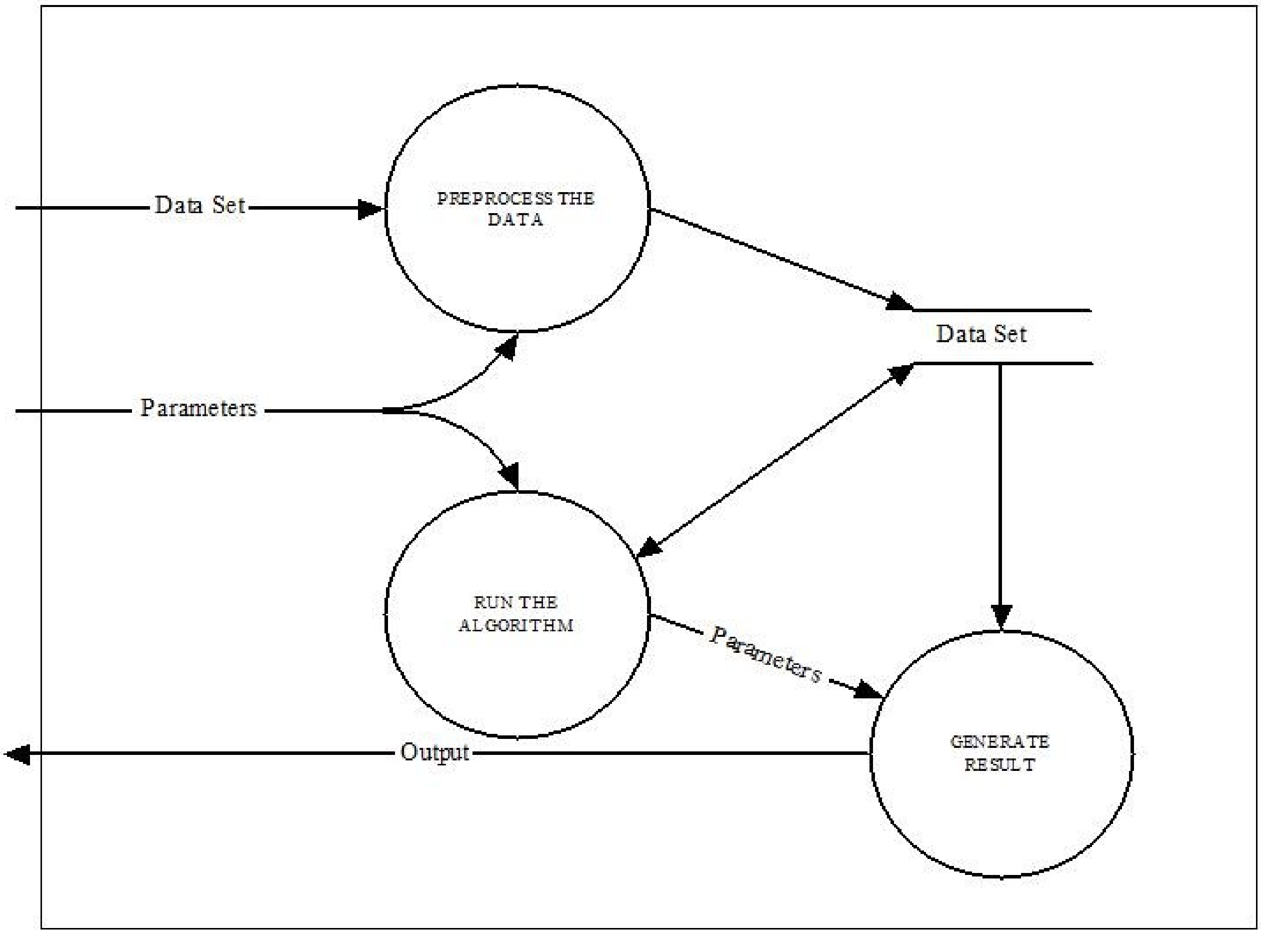


Figure 2

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 the some parts of the system having negligible importance are not ephasized. The Run the Algorithm process is divided into three processes; Build Algorithm, Evaluate Clusterer and Evaluate Classifier.

Level 2 DFD

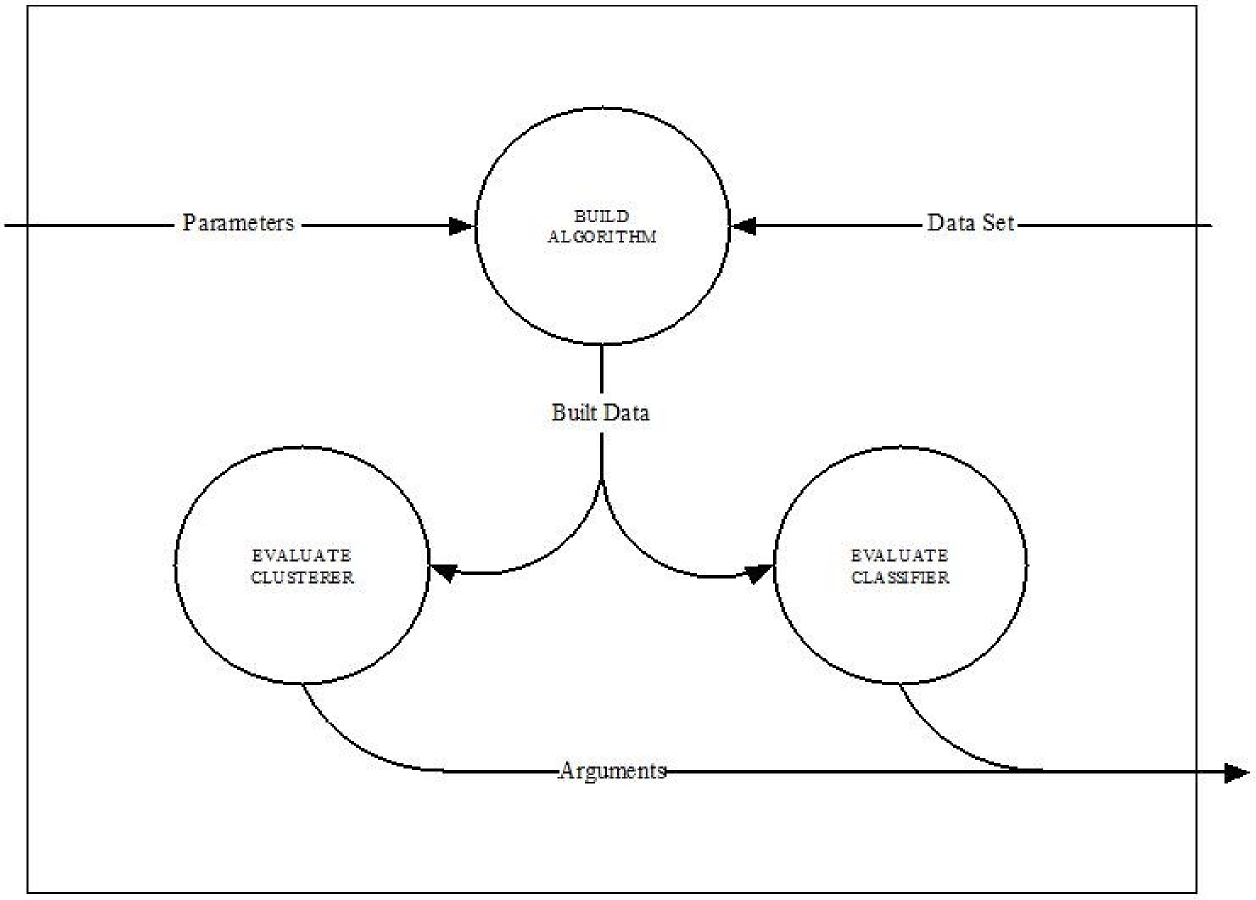


Figure 3

4.1.5. Explanation of Level 2 DFD

Build data set: This process initializes the algorithm stated in the parameters and performs the last operations which prepares the dataset to be evaluated. 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.

Evaluate classifier: All classifying algorithms are computed in this process and this is the another 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.

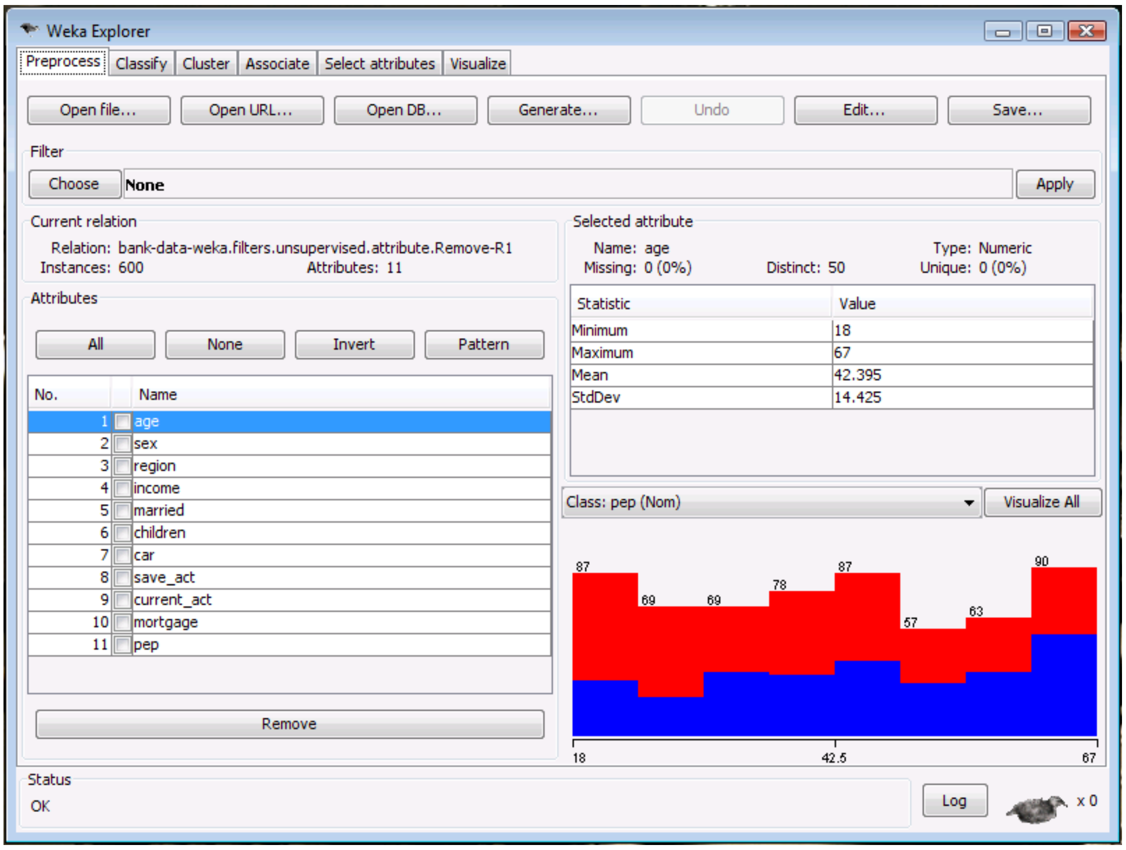
5. GUI DESIGN

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.



Filter: None

Current relation
Relation: bank-data-weka.filters.unsupervised.attribute.Remove-R1
Instances: 600 Attributes: 11

Attributes

No.	Name
1	age
2	sex
3	region
4	income
5	married
6	children
7	car
8	save_act
9	current_act
10	mortgage
11	pep

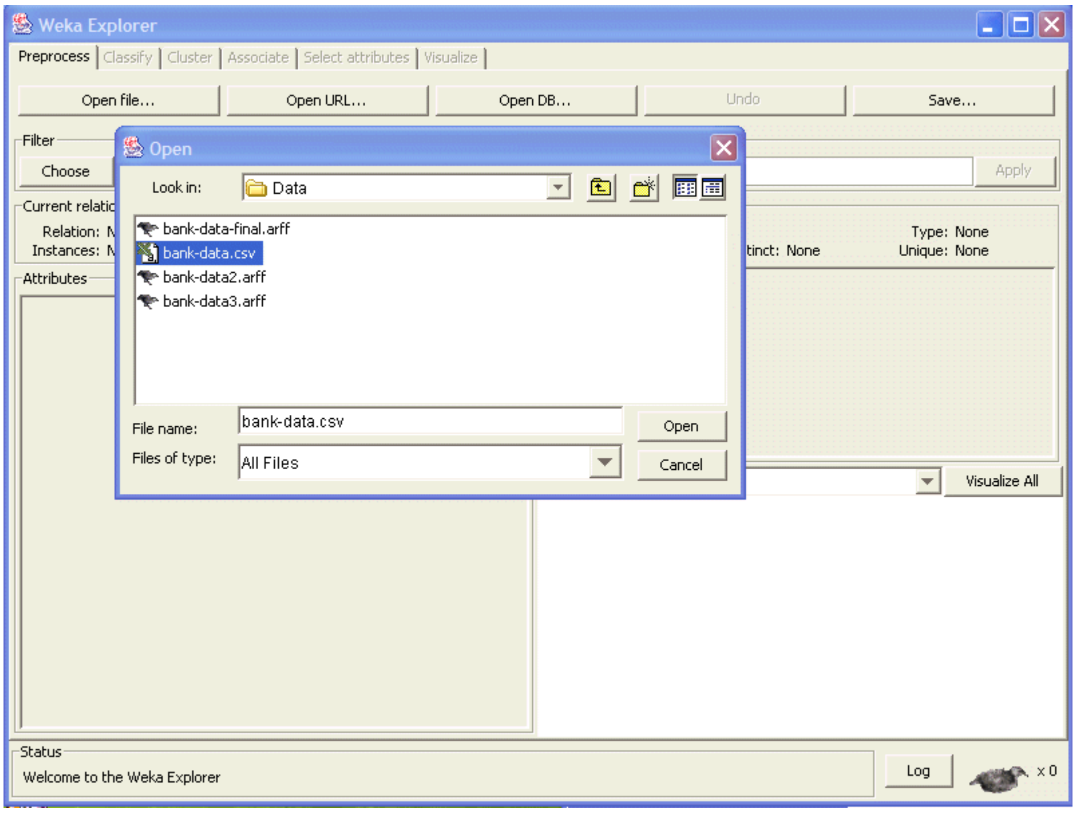
Selected attribute
Name: age Type: Numeric
Missing: 0 (0%) Distinct: 50 Unique: 0 (0%)

Statistic	Value
Minimum	18
Maximum	67
Mean	42.395
StdDev	14.425

Class: pep (Nom)

Status
OK

Figure 4 – Main Screen of Weka GUI



Open
Look in: Data

bank-data-final.arff
bank-data.csv
bank-data2.arff
bank-data3.arff

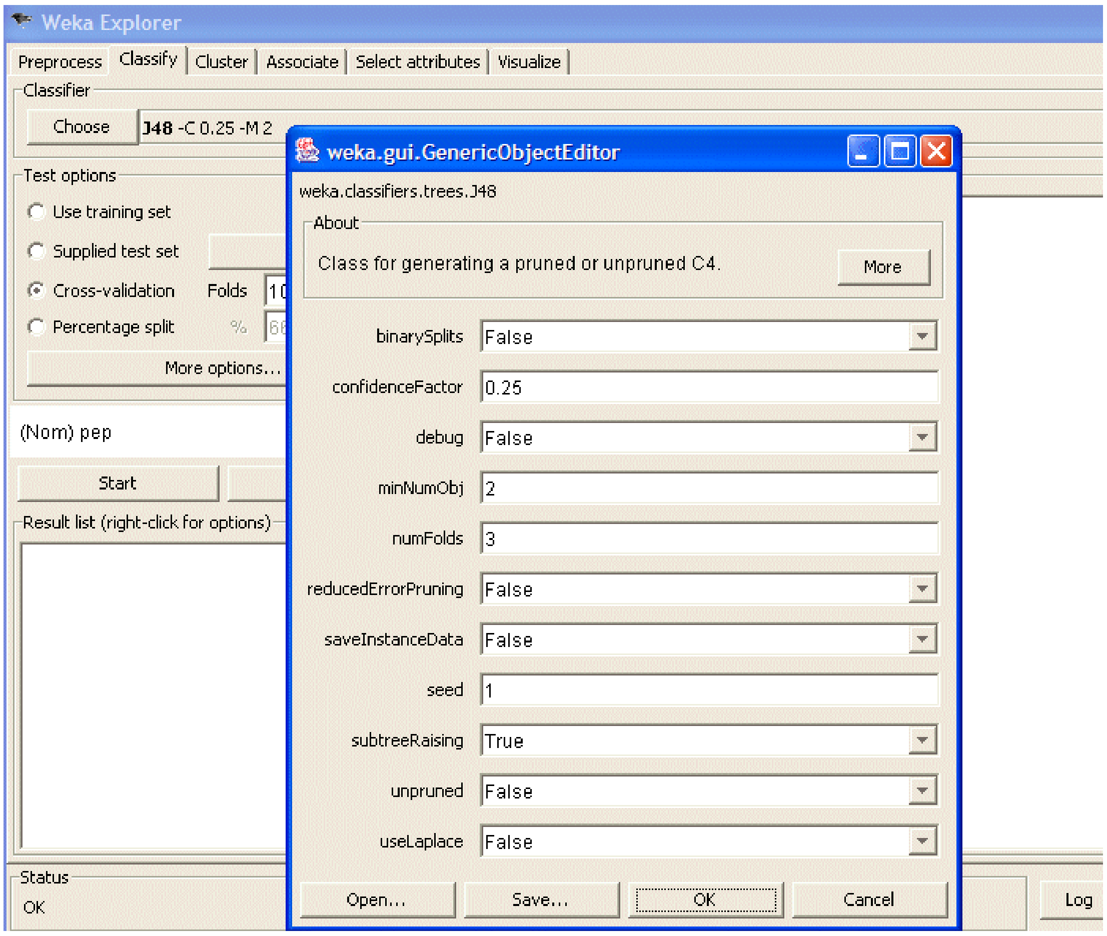
File name: bank-data.csv
Files of type: All Files

tinct: None Type: None
Unique: None

Status
Welcome to the Weka Explorer

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.



J48 -C 0.25 -M 2

Use training set
Supplied test set
[✓ Cross-validation] Folds 10
Percentage split % 66

(Nom) pep

OK

weka.classifiers.trees.J48

Class for generating a pruned or unpruned C4.

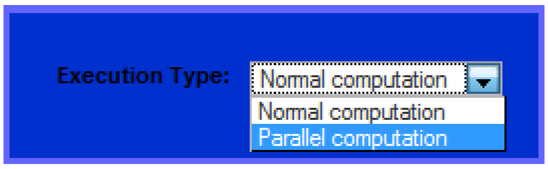
binarySplits	False
confidenceFactor	0.25
debug	False
minNumObj	2
numFolds	3
reducedErrorPruning	False
saveInstanceData	False
seed	1
subtreeRaising	True
unpruned	False
useLaplace	False

Figure 6 – Generating a classifier object

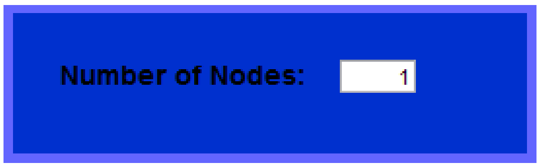
In clustering algorithm case, everything quite similar to the classifier algorithm initialization. Small and easy to understand differences exist on tab menus like parameters menu.

5.3. Planned Additions to User Interface

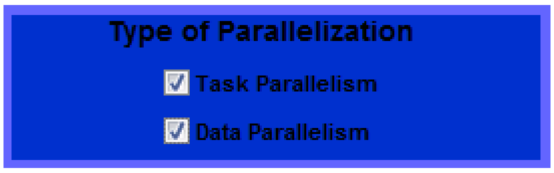
We will add some options to the Classify and Cluster tabs mainly in our interface design. These tabs will have an option which offers two ways to run an algorithms on datasets. One of them will be normal run of algorithms, in other words task is accomplished by one single process on one computing node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



Execution Type: Normal computation



Number of Nodes: 1



Type of Parallelization

✓ Task Parallelism

✓ Data Parallelism

In these tabs user will also be able to specify the number of computing nodes which will operate parallely on running algorithm, if parallelization option is chosen. Additionally, the system may provide a simple menu where user will have a chance to specify the type of parallelism for the system to use when performing task. The options will be Task parallelism and Data parallelism and if parallel computation is expected from the system, user have to choose one of the options or both of them at the same time. 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]

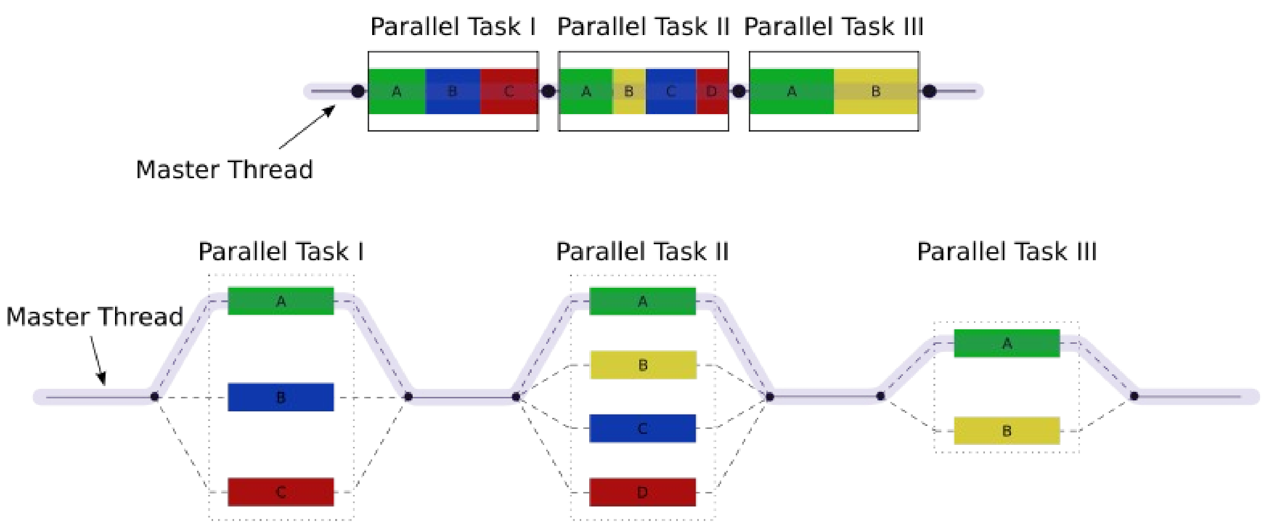


Figure 7 - An illustration of multithreading where the master thread forks off a number of threads which execute blocks of code in parallel

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]

- 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.

<u>Java Parallel Processing Framework :</u>

JPPF is an open source Grid Computing platform written in Java that makes it easy to run applications in parallel, and speed up their execution by orders of magnitude. Write once, deploy once, execute everywhere!

Some properties of Java Parallel Processing

- A JPPF grid can be up and running in minutes
- Simple programming model that abstracts the complexity of distributed and parallel processing
- Highly scalable, distributed framework for the parallel execution of cpu-intensive tasks
- Seamless integration with leading J2EE application servers
- Graphical and programmatic tools for fine-grained monitoring and administration
- Fault-tolerance and self-repair capabilities ensure the highest level of service and reliability
- A set of fully documented sample applications of JPPF to real-life problems
- JPPF@Home screensaver leverages idle enterprise resources
- Runs on any platform that supports Java ®
- Very flexible and business-friendly open source licensing

This technology may change our study framework. Because it is an alternative to MPI, OpenMP (for our scope they are mpiJava and JOMP) together.

<u>Hadoop :</u>

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 we are currently working on it.

7.1.4 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.

7.1.5 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.

7.2 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 attributeToDoubleArray(), variance(), sort(), sumOfWeights(), numDistinctValues(), mergeInstances(), meanOrMode() and swap() .
- Testing & improvement of ParallelInstances class

7.2.2 Paralellization of SimpleKMeans Class

- Parallelization of distributionForInstance() method
- Parallelization of clusterInstance() method

- Parallelization of buildClusterer() method
- Testing & improvement of ParallelKMeans class

7.2.3 Paralellization of ClusterEvaluation Class

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

7.2.4 Paralellization of SimpleNaiveBayes Class

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

7.2.5 Paralellization of Evaluation Class

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

8. Conclusion

We benefit much from requirement analysis report so when we started initial design report, we were aware of the importance and benefits of initial design report. Design is most probably the keypoint of a project and a good design is the crucial way to success; in this manner, we tried to prepare this document. 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 requirement analysis report.

More additions and consciousness adjoined into the project plans as specs are more detailed in initial design report; that's why we consider initial design report to be more beneficial than requirement analysis report.

We were unsure about where to parallellize and what kind of test to implement before we started writing this document. Detailed investigations for writing initial design report have helped us in seizing the situation in a better point of view. Now, we are aware of hints and design spirit of the Weka source code and how to parallelize it.

Packages are determined and selected in initial design report. We used requirement analysis DFD's in order to analyze the data flow and determining the inner dynamics of packages.

To summarize, this initial design report is a keypoint in our development schedule arrangements and design plans. We believe this design plan will guide through the end of the project and with the feedbacks from course coordinators, the design would be upgraded to a better level that will enrich our minds and shape our future plans.

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

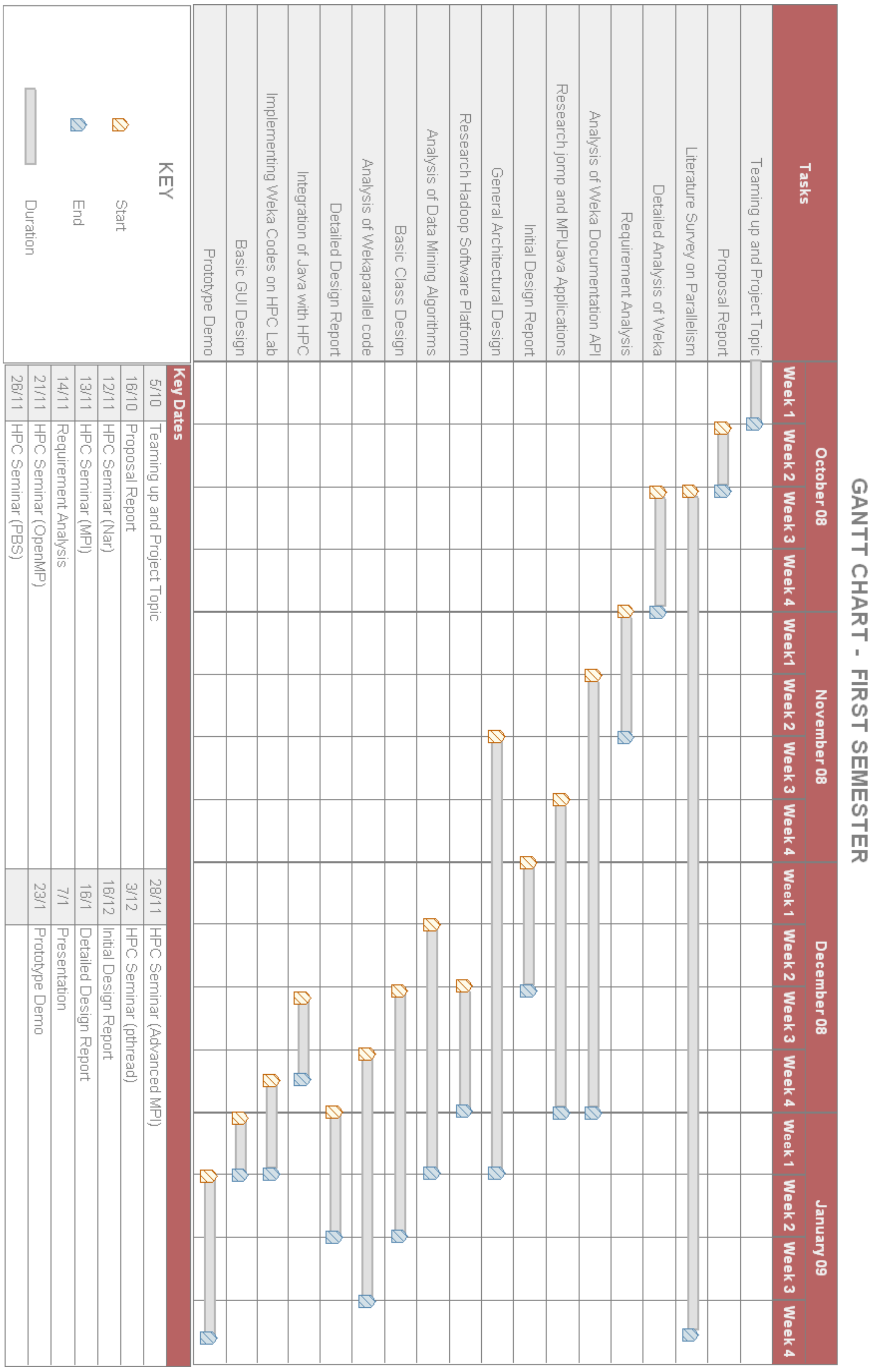
[14] http://web.ceng.metu.edu.tr/~ketenci/schedule.htm

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

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

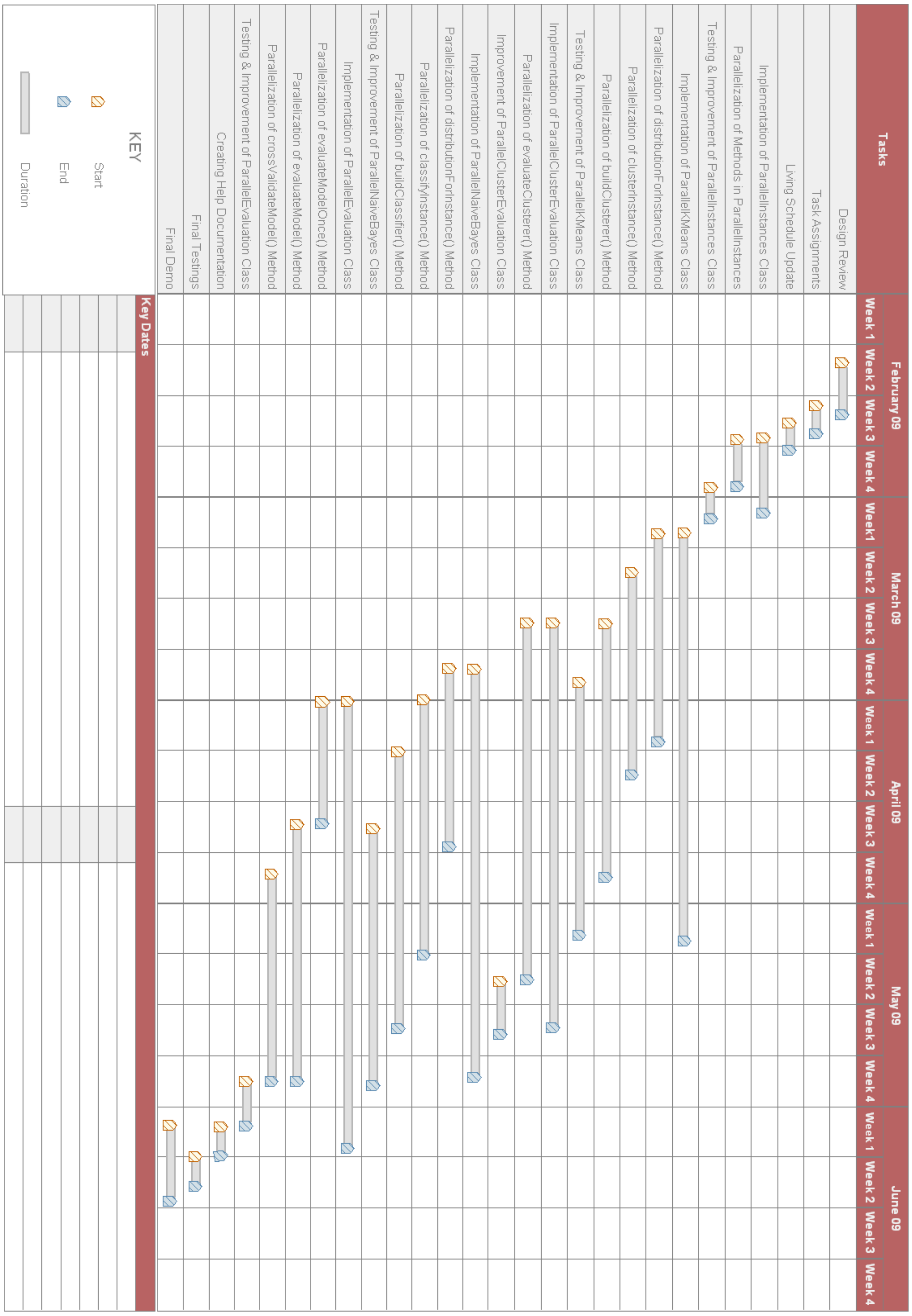
[17] Weka Wiki http://weka.wiki.sourceforge.net/

10. Appendix A – Gannt Chart 1



Key Dates			
5/10	Teaming up and Project Topic	28/11	HPC Seminar (Advanced MPI)
16/10	Proposal Report	3/12	HPC Seminar (pthread)
12/11	HPC Seminar (Nar)	16/12	Initial Design Report
13/11	HPC Seminar (MPI)	16/1	Detailed Design Report
14/11	Requirement Analysis	7/1	Presentation
21/11	HPC Seminar (OpenMP)	23/1	Prototype Demo
26/11	HPC Seminar (PBS)		

11. Appendix B – Gannt Chart 2


GANTT CHART - SECOND SEMESTER
Tasks
February 09
March 09
April 09
May 09
June 09
Week 1
Week 2
Week 3
Week 4
Week1
Design Review
Task Assignments
Living Schedule Update
Implementation of ParallelInstances Class
Parallelization of Methods in ParallelInstances
Testing & Improvement of ParallelInstances Class
Implementation of ParallelKMeans Class
Parallelization of distributionForInstance() Method
Parallelization of clusterInstance() Method
Parallelization of buildClusterer() Method
Testing & Improvement of ParallelKMeans Class
Implementation of ParallelClusterEvaluation Class
Parallelization of evaluateClusterer() Method
Improvement of ParallelClusterEvaluation Class
Implementation of ParallelNaiveBayes Class
Parallelization of distributionForInstance() Method
Parallelization of classifyInstance() Method
Parallelization of buildClassifier() Method
Testing & Improvement of ParallelNaiveBayes Class
Implementation of ParallelEvaluation Class
Parallelization of evaluateModelOnce() Method
Parallelization of evaluateModel() Method
Parallelization of crossValidateModel() Method
Testing & Improvement of ParallelEvaluation Class
Creating Help Documentation
Final Testings
Final Demo
KEY
Start
End
Duration
Key Dates