Machine learning is a vast field and has a broad range of applications including natural language processing, medical diagnosis, search engines, speech recognition, game playing and a lot more. A number of machine learning algorithms have been developed for different applications. However no single machine learning algorithm can be used appropriately for all learning problems. It is not possible to create a general learner for all problems because there are varied types of real world datasets that cannot be handled by a single learner. In this chapter we present an evaluation of various state-of-the-art machine learning algorithms using WEKA (Waikato Environment for Knowledge Analysis) for a real world learning problem- credit approval used in banks. Section 5.1 provides description about the components and working of WEKA. Section 5.2 describes the learning problem and the dataset that we have used in our experiments. In Section 5.3 we have explained the machine learning methods that we have evaluated. Section 5.4 provides description about our experimental setup and procedure and finally Section 5.5 shows the conclusion and the result. The work described in this chapter has been previously published [Khan and Quadri, 2012b].

5.1. Introduction

WEKA (http://www.cs.waikato.ac.nz/ml/weka/) is an open source software which consists of a collection of state-of-the-art machine learning algorithms and data preprocessing tools. It has been developed at the University of Waikato in New Zealand. It is designed in such a way that allows users to try all machine learning algorithms on new datasets easily. The WEKA system is written in Java. It can be used for a variety of tasks. It provides an implementation of state-of-the-art machine learning algorithms that we can apply to our datasets for extracting information about the data or we can apply several algorithms to our dataset for comparing their performance and choosing one for prediction. It also provides a number of tools for data preprocessing i.e. transforming datasets and analyzing the resulting classifier. Such tools are called filters. Thus the main focus of WEKA is on the learning methods and the filters. There are two ways in which we can invoke these methods: either by using command line options or by using the interactive graphical user interface. In our experiments we have used graphical user interface of WEKA because it is much more convenient. We have used WEKA 3.7.7.
5.1.1. WEKA- Interfaces

There are several ways by which we can access the functionality of WEKA. These are various interfaces and the simple CLI. Interfaces of WEKA include the Explorer, Experimenter and the Knowledge Flow.

5.1.1.1. Explorer

It is the most important graphical user interface in WEKA. Figure 5.1 shows the explorer interface. It consists of various tabs that are used for different tasks. First tab is the “Preprocess” tab. It is used for loading the datasets and transforming the datasets using filters. As shown in the figure datasets can be loaded as a file, from a URL or from databases using queries. WEKA allows files with specific formats e.g. ARFF, CSV, LibSVM’s format, and C4.5’s format.

Figure 5.1: WEKA Explorer Interface showing Preprocess Tab
After data is loaded it can be transformed by using various data preprocessing tools i.e. filters. Various discretization methods can be used for transforming these datasets or for dividing a dataset into training and testing sets using the appropriate filters.

Next is the “Classify” tab as shown in Figure 5.2. Through this tab we can use various classification and regression algorithms and applied to our preprocessed datasets. Classification algorithms typically produce decision trees or rules, while regression algorithms produce regression curves or regression trees. For a learning algorithm, the classify panel by default performs cross validation on the dataset that has been prepared in the Preprocess panel to estimate predictive performance. Other than cross-validation, test set can also be used. In that case we need to provide a test dataset separately. This panel also enables users to evaluate the resulting models, both numerically through statistical estimation and graphically through visualization of the data and examination of the model. This panel also allows us to visualize classifier errors, margin curve, threshold curve and so on. Moreover, it can visualize prediction errors in scatter plots, and also allows evaluation via ROC curves and other “threshold curves”. Models can also be saved and loaded in this panel.

![WEKA Explorer Interface showing Classify Tab](image)

**Figure 5.2: WEKA Explorer Interface showing Classify Tab**
Apart from supervised classification algorithms, WEKA also provides unsupervised algorithms such as clustering and association algorithms. The third tab “Cluster” provides access to the clustering algorithms and the fourth tab “Associate” enables users to access algorithms for learning association rules. In the “Cluster” tab we can run a clustering algorithm on the data that has been loaded in the “Preprocess” panel.

The last two tabs are “Select attributes” and “Visualize”. “Select attributes” tab is used for identifying the most predictive attributes in the data. This tab has a lot of algorithms and evaluation criteria used for identifying the most important attributes in a dataset. It allows the users to access various methods for measuring the utility of attributes, and for finding attribute subsets that are predictive of the data. Robustness of the selected attribute set can be validated via a cross-validation-based approach.

Visualize tab is used for analyzing data visually. This presents a color-coded scatter plot matrix, and users can then select and enlarge individual plots. It is also possible to zoom in on portions of the data, to retrieve the exact record underlying a particular data point, and so on.

5.1.1.2. Experimenter

As shown in Figure 5.3, “Experimenter” is another interface of WEKA. As stated earlier, it is not possible to have a single machine learning method that works for all learning problems efficiently. Also there is no way to determine which learning method will work efficiently for a given problem at the beginning. For this purpose it is better to compare the performance of machine learning methods on various criteria. This interface is used for this purpose. Although it can also be done interactively in the “Explorer” interface, however “Experimenter” interface automates this process. This makes it easy to run the classification and regression algorithms with different parameter settings on a corpus of datasets, collect performance statistics, and perform significance tests on the results. Experiments can involve multiple algorithms that are run across multiple datasets; for example, using repeated cross-validation. Experiments can be saved in either XML or binary form. Saved experiments can also be run from the command-line. The Experimenter interface is not used much often by data mining practitioners as other WEKA’s interfaces. This interface makes identification of a suitable algorithm for a particular
dataset or collection of datasets easier once the initial experiments have been performed in the Explorer.

![Weka Experiment Environment](image)

**Figure 5.3: WEKA Experimenter Interface**

### 5.1.1.3. Knowledge Flow

When we load a dataset in the “Explorer” interface, the entire dataset is loaded into the main memory for processing. It means that problems involving large datasets are not suitable for this method. In other words, “Explorer” interface does not allow for incremental learning and is only used for small to medium sized problems. However, some incremental algorithms are implemented that can be used to process very large datasets. One way to apply these is through the command-line interface, which gives access to all features of the system. An alternative, more convenient, approach is to use the second major graphical user interface, called “Knowledge Flow” which enables users to specify a data stream by graphically connecting components representing data sources, preprocessing tools, learning algorithms, evaluation
methods, and visualization tools. Its data flow model enables incremental updates with processing nodes that can load and preprocess individual instances before feeding them into appropriate incremental learning algorithms. It also provides nodes for visualization and evaluation.

5.1.2. Datasets
As stated in Section 1 of Chapter 1, the datasets used by machine learning algorithms consists of a number of instances that are represented using the same set of features. In supervised learning the instances are given with known labels (the corresponding correct outputs) in contrast to unsupervised learning, where instances are unlabeled. Table 5.1[Kotsiantis, 2007] shows instances with known labels.

Table 5.1: Example of a Dataset

<table>
<thead>
<tr>
<th>Case</th>
<th>Feature 1</th>
<th>Feature 2</th>
<th>....</th>
<th>Feature n</th>
<th>Class</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>xxx</td>
<td>x</td>
<td></td>
<td>xx</td>
<td>Good</td>
</tr>
<tr>
<td>2</td>
<td>xxx</td>
<td>x</td>
<td></td>
<td>xx</td>
<td>Good</td>
</tr>
<tr>
<td>3</td>
<td>xxx</td>
<td>x</td>
<td></td>
<td>xx</td>
<td>Bad</td>
</tr>
<tr>
<td>...</td>
<td>...</td>
<td>...</td>
<td></td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

WEKA applies its learning methods to a dataset and analyzes its output to extract information about the data. WEKA accepts the data in specific formats e.g. ARFF, CSV, LibSVM’s format, and C4.5’s format as stated earlier.

5.1.2.1. Preparing Datasets
The data that are has been collected for being used in the experiments can be stored anywhere e.g. in databases or spreadsheets. As we know WEKA supports some particular formats of data therefore we first need to convert the data into a suitable format before loading it in WEKA. The format we used for our experiments is ARFF format. The process of converting data into ARFF format is explained below.

Suppose we have our data in a spreadsheet program say MS Excel as shown in Figure 5.4. In order to convert it to ARFF format we first save it as a comma-separated file i.e. in CSV format. Then we load this CSV file in a text processor say MS Word as shown in Figure 5.5.
### Figure 5.4: Data in Excel spreadsheet

<table>
<thead>
<tr>
<th>ID</th>
<th>age</th>
<th>sex</th>
<th>region</th>
<th>income</th>
<th>married</th>
<th>children</th>
<th>car</th>
<th>save_act</th>
<th>current_act</th>
<th>mortgage</th>
<th>prep</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>48</td>
<td>FEMALE</td>
<td>INNER_CIT</td>
<td>17564</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>3</td>
<td>40</td>
<td>MALE</td>
<td>TOWN</td>
<td>10661.1</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
<td>MALE</td>
<td>INNER_CIT</td>
<td>10575.4</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>5</td>
<td>23</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>20375.4</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>6</td>
<td>57</td>
<td>FEMALE</td>
<td>RURAL</td>
<td>10575.4</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>7</td>
<td>57</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>20375.4</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>8</td>
<td>22</td>
<td>MALE</td>
<td>RURAL</td>
<td>8777.07</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
</tr>
<tr>
<td>9</td>
<td>58</td>
<td>MALE</td>
<td>TOWN</td>
<td>26506.6</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>10</td>
<td>37</td>
<td>FEMALE</td>
<td>SUBURBAN</td>
<td>25364.3</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>11</td>
<td>54</td>
<td>MALE</td>
<td>TOWN</td>
<td>24321.1</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>12</td>
<td>66</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>59803.9</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>13</td>
<td>52</td>
<td>FEMALE</td>
<td>INNER_CIT</td>
<td>90508.8</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>14</td>
<td>44</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>37353.8</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>15</td>
<td>66</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>55047.7</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
<td>YES</td>
</tr>
<tr>
<td>16</td>
<td>36</td>
<td>MALE</td>
<td>RURAL</td>
<td>10478.6</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>17</td>
<td>38</td>
<td>FEMALE</td>
<td>INNER_CIT</td>
<td>23421.1</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>18</td>
<td>37</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>17729.8</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>19</td>
<td>46</td>
<td>FEMALE</td>
<td>SUBURBAN</td>
<td>40316</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>20</td>
<td>62</td>
<td>FEMALE</td>
<td>INNER_CIT</td>
<td>26909.2</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>21</td>
<td>31</td>
<td>MALE</td>
<td>TOWN</td>
<td>25922.8</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>22</td>
<td>61</td>
<td>FEMALE</td>
<td>INNER_CIT</td>
<td>57880.7</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>23</td>
<td>50</td>
<td>MALE</td>
<td>TOWN</td>
<td>18499.3</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>24</td>
<td>54</td>
<td>MALE</td>
<td>INNER_CIT</td>
<td>38446.6</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>25</td>
<td>27</td>
<td>FEMALE</td>
<td>TOWN</td>
<td>10531.8</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
<tr>
<td>26</td>
<td>37</td>
<td>MALE</td>
<td>INNER_CIT</td>
<td>12043.9</td>
<td>NO</td>
<td>NO</td>
<td>YES</td>
<td>YES</td>
<td>NO</td>
<td>NO</td>
<td>NO</td>
</tr>
</tbody>
</table>

### Figure 5.5: Data after loading in MS Word

...
In this file the rows of the original spreadsheet have been converted into lines of text, and the elements are separated from each other by commas. After that we have to convert the first line in which there are names of attributes into the header structure that makes up the beginning of an ARFF file. This is done by specifying the name of the dataset using @relation tag, the names, types, and values of each attribute are defined by @attribute tags, and @data tag is added before the data section of the file. This is shown in Figure 5.6.

![Figure 5.6: Data after adding tags](image)

After this we have to save this file with “Text Only with Line Breaks” as the file type. In this way, our data in spreadsheet gets converted into a format compatible with WEKA.

5.1.2.2. Training sets and Tests sets

In order to test the efficiency of our learning models we use training and test sets. We split our data into these two sets. The data used to construct or discover a predictive relationship are called the training data set. A test set is a set of data that is independent of the training data, but that follows the same probability distribution as the training data. The training set or the seen
data is used to build the model i.e. determine its parameters and the test set or the unseen data is used to measure its performance (holding the parameters constant). In supervised learning, the training set or the “gold standard” consists of both the input data as well as the correct/expected output i.e. the class values, and the test set is the data that we are going to apply to our method to test its efficiency. This set doesn’t have the output class values.

Sometimes another set called the validation set is also used in addition to training and test sets to tune the model. It is used to estimate how good your model has been trained. It cannot be used for testing.

**5.1.2.3. Using the training and test sets in WEKA**

WEKA allows us to use the dataset in a number of ways in our experiments. We can perform cross-validation, percentage split or we can use the supplied test set option. For using the “supplied test set” option we need to split our dataset into appropriate quantities of training and test sets. We first show how cross-validation works and then the process of splitting the dataset.

**Cross-Validation**

In cross-validation, mutually exclusive and same-sized subsets are created by dividing the training set. For each subset the classifier is trained on the union of all the other subsets. Using this technique the error rate of the classifier is calculated by the average of the error rate of each subset. WEKA allows us to specify how many folds we want to specify and usually we use 10 folds. In k-fold cross-validation, the data is randomly divided into k folds (subsets) of equal size. Then train the model on k−1 folds, use one fold for testing. This process is repeated k times so that all folds are used for testing. Finally, average performance is computed on the k test sets. This process helps in effectively using all the data for both training and testing [Keller, 2002].

**Splitting the datasets**

As stated earlier, for using supplied test set in WEKA we need to split our dataset into training and test sets. In the “Explorer” interface, we first load our dataset in the “Preprocess” panel. This is done either by loading an ARFF file or CSV file. We can also load our dataset directly from a URL or database. In our example, we have loaded the dataset using a URL as shown in Figure 5.7.
Figure 5.7: Loading Dataset from URL

Figure 5.8: Using the Randomize filter
Next we have to split our dataset into two sets, 30% testing and 70% training. To do this we first randomize the dataset by choosing Randomize filter as shown in Figure 5.8. This creates a random permutation. Next we apply RemovePercentage filter on our dataset keeping percentage as 30 and we save the dataset as a training set. This is shown in Figure 5.9.

![Figure 5.9: Using RemovePercentage filter.](image)

Next we undo the change and again apply the same filter but changing the invertSelection option to “True” as shown in Figure 5.10. This picks the rest of the dataset i.e. 30% and is saved as a testing set.

This way our dataset gets divided into 30% testing and 70% training set. Next to use our sets in the experiments we choose the training set and move to the “Classify” panel and choose the procedure that we have to use and start the experiment. After that we apply the same procedure on our testing set to check what it predicts on the unseen data. For that, we select "supplied test set" and choose the testing dataset that we created. We run the algorithm again and we notice the differences in the confusion matrix and the accuracy.
5.2. Learning problem and the Dataset used in our experiments

In our experiments we used credit approval problem used in banks for evaluating the efficiency of the state-of-the-art machine learning algorithms.

5.2.1. Understanding the problem

A financial institution, e.g. a bank, gives its customers an amount of money and expects them to pay it back in installments along with interest. This amount of money is called credit. However, before approving any credit application, it is necessary for the bank to make sure that the customer will pay the whole amount back. The bank should be able to predict in advance the risk associated with a loan. It is done for making sure that the bank will make a profit and that the customers get a loan within his or her financial capacity. This calls for a need to find out efficient
methods for automatic credit approval that can help the authorities in assessing credit applications effectively.

5.2.1.1. Risk involved in credit approval

Here the risk involved refers to the risk of loss to the financial institution if they lend the money to the customers who fail to pay the amount back [K.H. Ng, 1996]. The reason for this default can be anything like inability, unwillingness, etc. The bank should be able to predict in advance the risk associated with a loan. It is necessary for the lenders to calculate the probability of risk involved so that they can make correct decisions regarding the approval of the credit.

5.2.1.2. Credit evaluation method

Credit evaluation or credit scoring [Hand, 1998] is an evaluation system that is used for improving or increasing the abilities of the credit lenders in deciding the probability of the credit risk of a customer. In this method, risk is calculated by the bank on the basis of the amount of credit and the information about the customer. The information about the customer includes data that the bank has access to and is relevant in calculating financial capacity of the customer. This data consists of income, savings, collaterals, profession, age, past financial history, and so forth. The bank has a record of past loans containing such customer data and whether the loan was paid back or not. From this data of particular applications, the aim is to infer a general rule coding the association between a customer’s attributes and his risk. That is, the machine learning system fits a model to the past data to be able to calculate the risk for a new application and then decides to accept or refuse it accordingly.

This process can be carried out in two ways. The first is called deductive credit scoring in which points are assigned to relevant customer attributes. These points are then used to form a credit score. The experience of the credit professionals is used to select the relevant attributes, determine the points and calculate the credit scores. Another type of credit scoring is empirical credit scoring in which the past data about the customers is analyzed and used to construct the scoring models. This is done by using appropriate algorithms for identifying characteristics relating to the credit risk of customers. These scoring models are then used to calculate the credit risk of new customers [Liu, 2001].

Bank professionals then use these credit scores to indicate the level of the credit risk and then decide accordingly whether to approve the credit to the customers or not and at what interest rate
the credit should be approved. For the low risk customers, the chances of getting the credit at lower interest rates and on longer repayment terms are higher. However, if the risk of the customers is high but lower than the cut-off credit risk, the customer is not disqualified from getting the credit but in this case the bank professionals review the customer application more carefully before deciding whether to approve or deny the credit request. If the credit is approved in case of such customers, it is given on higher interest rates and shorter repayment terms as compared to the low-risk customers.

5.2.1.3. Automating the process
The above stated processes i.e. credit scoring and approval can be carried out more efficiently if they are done automatically using computers. Automatic scoring and approval helps in gathering the necessary information quickly and speeds up the process of evaluation and determining whether to approve or deny credit applications. Automating this process does not mean that it can take place of the credit professional but it can help in making rapid decisions. The credit applications that are identified as good credit risk and those as bad credit risk may be automatically approved, or denied, while those of intermediate risk may still be passed to credit analysts for more detailed review before deciding whether to approve or deny credit. This can reduce the number of credit applications that need more detailed review and reduce the wastage of time, thus allowing credit analysts to concentrate only on those credit applications that are difficult or important.

5.2.2. Description of the Dataset used
The dataset ([http://www.hakank.org/weka/credit.arff](http://www.hakank.org/weka/credit.arff)) that we used for our experiments for evaluating the learning algorithms was provided originally by Quinlan in his studies of ID3 and C4.5 system in 1987 and 1992, to induce decision trees for assessing credit card applications. It is the Australian Credit Approval dataset from UCI Repository of Machine Learning Databases and Domain theories ([http://archive.ics.uci.edu/ml/datasets.html](http://archive.ics.uci.edu/ml/datasets.html)). The dataset consists of 15 attributes and a class label attribute. Before being made available to use, all the names and values of the attributes were changed to meaningless symbols to protect the confidentiality of the data. The values that the “class” attribute can take are + (positive) and – (negative). The attributes of the dataset are continuous, nominal with small numbers of values and nominal with larger numbers of values. The dataset consists of 490 instances with 44.5% being positive (credit approved), 55.5% being negative (credit denied) and 5% having missing values. Table 5.2 shows
the attribute names and attribute types of the dataset and Table 5.3 shows distributions of “+” and “-” values.

Table 5.2: Australian Credit Approval Dataset

<table>
<thead>
<tr>
<th>Attribute</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>A1</td>
<td>nominal</td>
</tr>
<tr>
<td>A2</td>
<td>continuous</td>
</tr>
<tr>
<td>A3</td>
<td>continuous</td>
</tr>
<tr>
<td>A4</td>
<td>nominal</td>
</tr>
<tr>
<td>A5</td>
<td>nominal</td>
</tr>
<tr>
<td>A6</td>
<td>nominal</td>
</tr>
<tr>
<td>A7</td>
<td>nominal</td>
</tr>
<tr>
<td>A8</td>
<td>continuous</td>
</tr>
<tr>
<td>A9</td>
<td>nominal</td>
</tr>
<tr>
<td>A10</td>
<td>nominal</td>
</tr>
<tr>
<td>A11</td>
<td>continuous</td>
</tr>
<tr>
<td>A12</td>
<td>nominal</td>
</tr>
<tr>
<td>A13</td>
<td>nominal</td>
</tr>
<tr>
<td>A14</td>
<td>continuous</td>
</tr>
<tr>
<td>A15</td>
<td>continuous</td>
</tr>
<tr>
<td>Class</td>
<td>nominal</td>
</tr>
</tbody>
</table>

Table 5.3: Class Distribution

<table>
<thead>
<tr>
<th>Class</th>
<th>No. of instances</th>
</tr>
</thead>
<tbody>
<tr>
<td>+</td>
<td>218</td>
</tr>
<tr>
<td>-</td>
<td>272</td>
</tr>
</tbody>
</table>

The “class” attribute can take two values i.e. “+” and “-” as stated earlier. The two values represent the low-risk and high-risk customers here. For low-risk customers, “class” attribute takes “+” value meaning credit can be approved for such customers and vice-versa for high-risk customers. This makes our learning problem a classification problem.
WEKA provides a number of classification algorithms that are accessible from the “Classify” tab as stated earlier. Hence our experiments use this dataset for evaluation of various classification learning algorithms. For our experiments we divided our dataset into training and test sets by the same procedure as described in Section 5.1.

5.3. Learning Methods Chosen For Evaluation

As discussed above, the learning problem that we have used in our experiments is a classification problem. Therefore, we have used WEKA’s classification algorithms for evaluation of the chosen dataset. In our experiments, we have chosen 10 learning algorithms from 6 different types. These are given below:

- **Rule based**
  - Zero R
  - One R
- **Bayes Rule**
  - NaiveBayes
  - NaiveBayesUpdateable
- **Functions**
  - Multilayer Perceptron
- **Lazy Learners**
  - KStar (K*)
- **Tree Based**
  - J48
  - RandomForest
- **Meta-Algorithm**
  - AdaBoostM1
  - Bagging

The sections that follow first explain each of these learners and then show their performance evaluation.
5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

5.3.1. ZeroR and OneR

Both of these algorithms are rule-based algorithms. A rule-based algorithm uses rules to make deductions or choices. The classification method uses an algorithm to generate rules from the sample data. These rules are then applied to new data. OneR (One Rule) is a simple classification algorithm that generates a one-level decision tree. OneR classifier infers simple and accurate, classification rules from a set of instances. Performance studies of OneR classifier have shown that it produces rules that are only slightly less accurate than state-of-the-art learning schemes. It produces rules that are easy to interpret. OneR is also capable of handling missing values and numeric attributes showing adaptability despite simplicity. The OneR algorithm creates one rule for each attribute in the training data. It then selects the rule with the smallest error rate as its ‘one rule’. It determines the most frequent class for each attribute value for creating a rule for an attribute. The most frequent class is simply the class that appears most often for that attribute value. A rule is simply a set of attribute values bound to their majority class; one such binding for each attribute value of the attribute the rule is based on. The error rate of a rule is the number of training data instances in which the class of an attribute value does not agree with the binding for that attribute value in the rule. OneR selects the rule with the lowest error rate. In the event that two or more rules have the same error rate, the rule is chosen at random. In the implementation of WEKA, the OneR algorithm picks the rule with the highest number of correct instances, not lowest error rate, and does not randomly select a rule when error rates are identical. Zero Regression (ZeroR) is a pseudo-regression method that always builds models with cross-validation coefficient Q2=0. In the framework of this method the value of a property/activity is always predicted to be equal to its average value on the training set. This method is usually used as a reference point for comparing with other regression methods. ZeroR is the simplest classification method which relies on the target and ignores all predictors. ZeroR classifier simply predicts the majority category (class). Although there is no predictability power in ZeroR, it is useful for determining a baseline performance as a benchmark for other classification methods. The idea behind the ZeroR classifier is to identify the most common class value in the training set. It always returns that value when evaluating an instance. It is frequently used as a baseline for evaluating other machine learning algorithms.
5.3.2. NaiveBayes and NaiveBayesUpdateable

The Naive Bayes [Murphy, 2006] algorithm is based on conditional probabilities. It uses Bayes' Theorem. It is a formula that calculates a probability by counting the frequency of values and combinations of values in the historical data. Bayes' Theorem finds the probability of an event occurring given the probability of another event that has already occurred. If B represents the dependent event and A represents the prior event, Bayes' theorem can be stated as follows.

\[
\text{Prob}(B \text{ given } A) = \frac{\text{Prob}(A \text{ and } B)}{\text{Prob}(A)}
\]

To calculate the probability of B given A, the algorithm counts the number of cases where A and B occur together and divides it by the number of cases where A occurs alone. A naive Bayes classifier is a simple probabilistic classifier based on applying Bayes' theorem with strong (naive) independence assumptions. In simple terms, a naive Bayes classifier assumes that the presence (or absence) of a particular feature of a class is unrelated to the presence (or absence) of any other feature, given the class variable. An advantage of the naive Bayes classifier is that it only requires a small amount of training data to estimate the parameters (means and variances of the variables) necessary for classification.

NaiveBayesUpdateable is a class for a Naive Bayes classifier using estimator classes. This is the updateable version of NaiveBayes. This classifier will use a default precision of 0.1 for numeric attributes when buildClassifier is called with zero training instances.

5.3.3. MultiLayer Perceptron

It is a classifier that uses back propagation to classify instances. This network can be built by hand, created by an algorithm or both. The network can also be monitored and modified during training time. The nodes in this network are all sigmoid (except for when the class is numeric in which case the output nodes become unthresholded linear units). A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate output. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called back propagation for training the network. MLP is a modification of the standard
linear perceptron and can distinguish data that is not linearly separable. It is an artificial neural network generally used for classification or approximation. The MLP consists of a feed-forward network of neurons which map input vectors to output vectors. Each artificial neuron consists of a linear combination of weighted inputs which is passed through a non-linear activation function to produce the neuron’s output. It is an extension of the perceptron in that it has at least one hidden layer of neurons. Layers are updated by starting at the inputs and ending with the outputs. Each neuron computes a weighted sum of the incoming signals, to yield a net input, and passes this value through its sigmoidal activation function to yield the neuron's activation value. Unlike the perceptron, an MLP can solve linearly inseparable problems [Steinwender and Bitzer, 2003].

### 5.3.4. J48 and Random Forest

Both these algorithms are decision tree based algorithms. A decision tree is a predictive machine-learning model that decides the target value (dependent variable) of a new sample based on various attribute values of the available data. The internal nodes of a decision tree denote the different attributes, the branches between the nodes tell us the possible values that these attributes can have in the observed samples, while the terminal nodes tell us the final value (classification) of the dependent variable.

The attribute that is to be predicted is known as the dependent variable, since its value depends upon, or is decided by, the values of all the other attributes. The other attributes, which help in predicting the value of the dependent variable, are known as the independent variables in the dataset. J4.8 algorithm is WEKA’s implementation of C4.5 decision tree learner. C4.5 is an algorithm used to generate a decision tree developed by Ross Quinlan [Quinlan, 1993]. C4.5 is an extension of Quinlan's earlier ID3 algorithm. The decision trees generated by C4.5 can be used for classification, and for this reason, C4.5 is often referred to as a statistical classifier. The J48 Decision tree classifier follows the following simple algorithm. In order to classify a new item, it first needs to create a decision tree based on the attribute values of the available training data. So, whenever it encounters a set of items (training set) it identifies the attribute that discriminates the various instances most clearly. This feature that is able to tell us most about the data instances so that we can classify them the best is said to have the highest information gain. Now, among the possible values of this feature, if there is any value for which there is no ambiguity, that is, for which the data instances falling within its category have the same value for
the target variable, then we terminate that branch and assign to it the target value that we have obtained. For the other cases, we then look for another attribute that gives us the highest information gain. Hence we continue in this manner until we either get a clear decision of what combination of attributes gives us a particular target value, or we run out of attributes. In the event that we run out of attributes, or if we cannot get an unambiguous result from the available information, we assign this branch a target value that the majority of the items under this branch possess.

Random forest is a powerful new approach to data exploration, data analysis, and predictive modeling. RandomForest implements Breiman’s random forest algorithm (based on Breiman and Cutler’s original Fortran code) for classification and regression. It can also be used in unsupervised mode for assessing proximities among data points. Random forest (or random forests) is an ensemble classifier that consists of many decision trees and outputs the class that is the mode of the classes output by individual trees. The algorithm for inducing a random forest was developed by Leo Breiman [Breiman, 2001] and Adele Cutler, and "Random Forests" is their trademark. The term came “from random decision forests” that was first proposed by Tin Kam Ho of Bell Labs in 1995. A random forest is a collection of CART-like trees following specific rules for tree growing, tree combination, self-testing, and post-processing [Steinberg et al., 2004]. It is unexcelled in accuracy among current algorithms. It runs efficiently on large data bases. It can handle thousands of input variables without variable deletion. It gives estimates of what variables are important in the classification. It generates an internal unbiased estimate of the generalization error as the forest building progresses. It has an effective method for estimating missing data and maintains accuracy when a large proportion of the data are missing. It has methods for balancing error in class population unbalanced data sets. Generated forests can be saved for future use on other data. Prototypes are computed that give information about the relation between the variables and the classification. It computes proximities between pairs of cases that can be used in clustering, locating outliers or (by scaling) give interesting views of the data. The capabilities of the above can be extended to unlabeled data, leading to unsupervised clustering, data views and outlier detection. It offers an experimental method for detecting variable interactions.
5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

5.3.5. KStar (K*)

K* is one of the lazy learning methods. Lazy learning methods or memory-based methods learn the structure of a domain by storing learning examples with their classification [Van den Bosch et al. 1996]. The domain model that results from a lazy learning process is able to generalize by using a predefined distance function. When the domain model is required to give the classification for an unseen domain element then it will use the distance function for finding the stored example that is closest to this unseen example. K* is an instance-based learner. Instance-based learners classify an instance by comparing it to a database of pre-classified examples. The fundamental assumption is that similar instances will have similar classifications. The question lies in how to define “similar instance” and “similar classification”. The corresponding components of an instance-based learner are the distance function which determines how similar two instances are, and the classification function which specifies how instance similarities yield a final classification for the new instance. In addition to these two components, IBL algorithms have a concept description updater which determines whether new instances should be added to the instance database and which instances from the database should be used in classification. For simple IBL algorithms, after an instance has been classified it is always moved to the instance database along with the correct classification. More complex algorithms may filter which instances are added to the instance database to reduce storage requirements and improve tolerance to noisy data [Cleary and Trigg, 1995]. K* is an instance-based classifier, that is the class of a test instance is based upon the class of those training instances similar to it, as determined by some similarity function. It differs from other instance-based learners in that it uses an entropy-based distance function. The use of entropy as a distance measure has several benefits. Amongst other things it provides a consistent approach to handling of symbolic attributes, real valued attributes and missing values.

5.3.6. AdaBoostM1 and Bagging

Bootstrap aggregating (bagging) and boosting are useful techniques to improve the predictive performance of tree models. Boosting may also be useful in connection with many other models, e.g. for additive models with high-dimensional predictors; whereas bagging is most prominent for improving tree algorithms. Boosting is a general method for improving the performance of any learning algorithm. In theory, boosting can be used to significantly reduce the error of any
“weak” learning algorithm that consistently generates classifiers which need only be a little bit better than random guessing. Despite the potential benefits of boosting promised by the theoretical results, the true practical value of boosting can only be assessed by testing the method on “real” learning problems. AdaBoost [Freund and Schapire, 1996] is a boosting algorithm developed by Freund and Schapire that can be used to significantly reduce the error of any learning algorithm that consistently generates classifiers whose performance is a little better than random guessing. AdaBoostM1 is a version of AdaBoost algorithm. Bagging is based on an idea of making various samples of the training set. A classifier is generated for each of these training set samples by a selected machine learning algorithm. In this way, for k variations of the training set we get k particular classifiers. The result will be given as a combination of individual particular classifiers.

5.4. Experimental Setup

In this section we show the performance evaluation of all the learning algorithms discussed above. We show their evaluation on the dataset chosen i.e. Credit Dataset. As already stated, we carried our experiments using WEKA. It provides a number of measures of evaluation that can be used to check the performance of the algorithms. When an experiment is run, results are displayed on “Classifier Output” area. This area has several sections showing different results. First is run information. It is a list of information giving the learning scheme options, relation name, instances, attributes and test mode that were involved in the process. After that classifier model (full training set) is displayed. It is a textual representation of the classification model that was produced on the full training data. Then a summary is shown that shows a list of statistics summarizing how accurately the classifier was able to predict the true class of the instances under the chosen test mode. A detailed accuracy by class, which is a more detailed per-class break down of the classifier’s prediction accuracy, is shown. Lastly, confusion matrix shows how many instances have been assigned to each class. Elements show the number of test examples whose actual class is the row and whose predicted class is the column. The evaluation measures that we used to compare the learners are number of correctly classified instances, time taken to build the model, F-Measure. For a particular label of interest, we are provided with a set of actual positives (e.g., objects that belong to that label) contained within the data set. The model then makes a set of predicted positives (e.g., the objects it assigns to that label) for the same data set.
The actual and predicted label groupings can be thought of as indicator variables, and their cross product results in four important values: tp (the number of true positives), fp (false positives), tn (true negatives), and fn (false negatives). A basic evaluation measure is accuracy = tp+tn / tp+fp+tn+fn. Basically, this measure represents the fraction of objects that the model labels correctly. In some problems, however, the data may be highly skewed, e.g., there might be nine times as many negative objects as positives. In a case like this, accuracy is a poor evaluation measure because a model that labels everything negative will still have accuracy = 0.9. In these situations, it is common instead to use precision, $P = \frac{tp}{tp+fp}$, the fraction of predictions that are correct, and recall, $R = \frac{tp}{tp+fn}$, the fraction of actual positives that are correctly predicted. Because of the inherent trade-off between precision $P$ and recall $R$, a summary statistic called the F-Measure = $2 * \frac{P * R}{P+R}$ is commonly used when both are considered equally important.

Before using our dataset in the experiments, we used the method discussed in Section 5.1 for splitting it into 70% training set and 30% test set. First we loaded the actual dataset into the WEKA from URL (http://www.hakank.org/weka/credit.arff). Then after applying the splitting procedure, we saved both these sets as separate files, trainingcredit.arff and testingcredit.arff. For all experiments we used these two files. Figure 5.11 shows the actual dataset, Figure 5.12 shows trainingcredit.arff file and Figure 5.13 shows testingcredit.arff file.
5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

Figure 5.12: trainingcredit.arff file loaded in WEKA

Figure 5.13: testingcredit.arff file loaded in WEKA
5.4.1. Experimental Procedure

- In our experiments, for each learner, we first load `trainingcredit.arff` file into WEKA through “Preprocess” panel.

- Then in the “Classify” panel we choose the classification algorithm to be implemented and start the analysis using 10-fold cross validation.

- After that we load the file `testingcredit.arff` using the “Supplied test set option” and then start the analysis again.

- Finally, we analyze the results on the basis of the evaluation measures discussed above.

- The same process is repeated for all the classification algorithms that are to be evaluated.

5.4.2. Experimental Results

![Figure 5.14: Results of J48 on trainingcredit.arff](image)

Figure 5.14: Results of J48 on `trainingcredit.arff`
5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

Figure 5.15: Results of J48 on testingcredit.arff

Results of J48:

Correctly Classified Instances (%) = 85.7143
Incorrectly Classified Instances (%) = 14.2857
Kappa Statistic = 0.71
Mean Absolute Error = 0.1817
F-Measure = 0.837
Time taken to build the Model = 0.01 seconds
Figure 5.16: Results of RandomForest on testingcredit.arff

Results of RandomForest:

Correctly Classified Instances (%) = 84.3537
Incorrectly Classified Instances (%) = 15.6463
Kappa Statistic = 0.6835
Mean Absolute Error = 0.2445
F-Measure = 0.824
Time Taken to build the Model = 0.05 seconds
Results of ZeroR:

Correctly Classified Instances (%) = 56.4626
Incorrectly Classified Instances (%) = 43.5374
Kappa Statistic = 0
Mean Absolute Error = 0.4931
F-Measure = 0
Time Taken to build the Model = 0 seconds

Figure 5.17: Results of ZeroR on testingcredit.arff
Figure 5.18: Results of OneR on testingcredit.arff

**Results of OneR:**

- Correctly Classified Instances (%) = 85.034
- Incorrectly Classified Instances (%) = 14.966
- Kappa Statistic = 0.703
- Mean Absolute Error = 0.1497
- F-Measure = 0.845
- Time Taken to build the Model = 0 seconds
Results of NaiveBayes:

Correctly Classified Instances (%) = 79.5918
Incorrectly Classified Instances (%) = 20.4082
Kappa Statistic = 0.5727
Mean Absolute Error = 0.21
F-Measure = 0.732
Time Taken to build the Model = 0.01 seconds
Figure 5.20: Results of NaiveBayesUpdateable on testingcredit.arff

Results of NaiveBayesUpdateable:

Correctly Classified Instances (%) = 79.5918
Incorrectly Classified Instances (%) = 20.4082
Kappa Statistic = 0.5727
Mean Absolute Error = 0.21
F-Measure = 0.732
Time Taken to build the Model = 0.01 seconds
Figure 5.21: Results of AdaBoostM1 on testingcredit.arff

**Results of AdaBoostM1:**

- Correctly Classified Instances (%) = 84.3537
- Incorrectly Classified Instances (%) = 15.6463
- Kappa Statistic = 0.689
- Mean Absolute Error = 0.211
- F-Measure = 0.837
- Time Taken to build the Model = 0.04 seconds
Results of Bagging:

Correctly Classified Instances (%) = 84.3537
Incorrectly Classified Instances (%) = 15.6463
Kappa Statistic = 0.6879
Mean Absolute Error = 0.2196
F-Measure = 0.835
Time Taken to build the Model = 0.05 seconds
5 EVALUATING MACHINE LEARNING TECHNIQUES FOR EFFICIENCY

Figure 5.23: Results of MultiLayerPerceptron on testingcredit.arff

Results of MultiLayerPerceptron:

Correctly Classified Instances (%) = 86.3946
Incorrectly Classified Instances (%) = 13.6054
Kappa Statistic = 0.7252
Mean Absolute Error = 0.1508
F-Measure = 0.848
Time Taken to build the Model = 0.01 seconds
Figure 5.24: Results of KStar on testingcredit.arff

Results of KStar:

Correctly Classified Instances (%) = 71.4286
Incorrectly Classified Instances (%) = 28.5714
Kappa Statistic = 0.4017
Mean Absolute Error = 0.2896
F-Measure = 0.625
Time Taken to build the Model = 0 seconds
5.5. Conclusion

In this section we show the results in the form of charts and tables. Figure 5.25 shows the comparison of all the algorithms with respect to the time taken to build the model.

![Figure 5.25: Time chart of algorithms](image)

Figure 5.26 shows the comparison based about the number of correctly classified instances by each learning algorithm.

![Figure 5.26: Comparison of Algorithms By Percentage Of Correct Instances](image)
In Table 5.4 we have summarized three main measures of evaluation for each algorithm i.e. time taken to build the model, number of correctly classified instances, and F-Measure.

Table 5.4: Comparison of algorithms

<table>
<thead>
<tr>
<th>Algorithms</th>
<th>Time taken to build model (sec)</th>
<th>Correctly Classified Instances (%)</th>
<th>F-Measure</th>
</tr>
</thead>
<tbody>
<tr>
<td>J48</td>
<td>0.01</td>
<td>85.7143</td>
<td>0.837</td>
</tr>
<tr>
<td>RandomForest</td>
<td>0.05</td>
<td>84.3537</td>
<td>0.824</td>
</tr>
<tr>
<td>ZeroR</td>
<td>0</td>
<td>56.4626</td>
<td>0</td>
</tr>
<tr>
<td>OneR</td>
<td>0</td>
<td>85.034</td>
<td>0.845</td>
</tr>
<tr>
<td>NaiveBayes</td>
<td>0.01</td>
<td>79.5918</td>
<td>0.732</td>
</tr>
<tr>
<td>NaiveBayesUpdateable</td>
<td>0.01</td>
<td>79.5918</td>
<td>0.732</td>
</tr>
<tr>
<td>KStar</td>
<td>0</td>
<td>71.4286</td>
<td>0.625</td>
</tr>
<tr>
<td>MultiLayerPerceptron</td>
<td>0.01</td>
<td>86.3946</td>
<td>0.848</td>
</tr>
<tr>
<td>AdaBoostM1</td>
<td>0.04</td>
<td>84.3537</td>
<td>0.837</td>
</tr>
<tr>
<td>Bagging</td>
<td>0.05</td>
<td>84.3537</td>
<td>0.835</td>
</tr>
</tbody>
</table>

It shows that RandomForest and Bagging take maximum amount of time to build the model i.e.0.05 seconds. Next highest is 0.04 taken by AdaBoostM1. NaiveBayes, NaiveBayesUpdateable and MultiLayerPerceptron take 0.01 seconds and the remaining take 0 seconds to build the model. In terms of second measure of evaluation, MultiLayerPerceptron has the highest percentage of correctly labeled instances and has the best F-Measure among all. Hence, we conclude that MultiLayerPerceptron has performed better than all the other classifiers in the analysis of our dataset.