Decision Trees: A comparison of various algorithms for building Decision Trees

Vaibhav Mohan

April 18, 2013

Abstract

Decision Trees are a decision support tool that contains tree like graph of decisions and the possible consequences. They are commonly used in different real world scenarios ranging from operations research to classifying a specie in a phylum given its features.

The Decision Tree is implemented using traditional ID3 algorithm as well as an evolutionary algorithm for learning decision trees in this paper. The Traditional Algorithm for learning decision trees is implemented using information gain as well as using gain ratio. Each variant is also modified to combat over-fitting using pruning. The Evolutionary Algorithm is implemented with fitness proportionate and rank based as their selection strategy. The algorithm is also implemented to have complete replacement and elitism as replacement strategy. The two algorithms are compared based on their accuracy, precision and recall by varying the aforementioned parameters on the datasets taken from UCI Machine Learning repository[2]. The time taken for learning the Decision Tree by each algorithm corresponding to each setting is also compared in this paper.

1 Introduction

Classification is the problem of identifying a category for the given instance whose category is unknown. The classifier tries to classify the given unknown instance based on the data it learns from the training set and features it sees in the instance for which prediction is to be done. This problem arises in various fields ranging from operation research to robotics to making financial decisions. There are wide varieties of classification problems in machine learning domain, all of which cannot be solved using one technique [13]. Therefore, for proving that the results which we are getting from one kind of technique is good enough for us makes it indispensable that we compare the results with other techniques for the given problem. Whilst doing this, we come across the various performance aspects of the algorithm i.e. where it would fail and where it can do remarkably well in classifying the data.

One of the most popular technique for classifying data in Machine Learning domain is Decision Trees. The advantage of using Decision Trees in classifying the data is that they are simple to understand and interpret. Decision tree have been well studied and widely used in the knowledge discovery and decision support system. These trees approximate discrete-valued target functions as trees and are widely used practical method for inductive reference[9]. Each line present in the datasets is known as the instance. The instance contains the label and a vector of features present in it. The Decision Trees examines the feature of given instance and comes to a conclusion on what label to assign based on the values present for the various features of that particular instance. Each node in the decision tree is either a decision node or a leaf node. This classifier resembles tree data structure as each decision can have two outcomes, thereby making a binary decision tree that culminates in a label corresponding to each set of given features.

Algorithms, such as ID3, often use heuristics that tends to find short decision trees[9, 11], however finding the shortest decision tree is a hard optimization problem[6]. Genetic Algorithms (GAs) are inspired by the real world process of evolution[9, 11, 7]. GAs have been used to construct short and near-optimal decision trees. In order to utilize genetic algorithms, decision trees must be represented as chromosomes on which genetic operators such as mutation and crossover can be applied. Genetic Algorithms have been used in two ways for finding the near-optimal decision trees. One way is that they can be used to construct decision trees in a hybrid or preprocessing manner[8]. The other way is to apply them directly to decision trees[10]. In this paper, we implement decision trees using traditional ID3 algorithm as well as Genetic Algorithm. The comparison of performance of both the algorithms is done in this paper.

The remainder of paper is organized as follows. Section 2 defines the decision trees and the various algorithms used in implementing decision trees.

Section 3 comprises of details of the parameters chosen pertaining to implemented algorithms and experimental methods used with them. It also talks about the datasets used for performing the experiment. Section 4 presents the results obtained after performing the experiment. Section 5 talks about the interpretation of the data which we got from the experiment. Finally, Section 6 concludes this work.

2 Learning Decision Trees

The data classification is a two step process. The first step is training the classifier using training data set. The second step involves predicting the labels for the unknown datasets (or testing datasets). The decision tree is a flow chart like tree structure where each node denotes a test on an attribute and each branch denotes an outcome of the test. The leaf nodes present in the decision trees represents classes or class distributions. This comes under the training step. Now in order to classify an unknown instance, the attribute values of instance are tested against decision tree and a path is traced from root to leaf node which holds the class prediction for that sample. This comes under the prediction step. The decision trees can easily be converted into classification rules.

Decision tree falls under supervised learning techniques as we have known labels in the training data set in order to train the classifier. The various algorithms that are implemented in this paper are discussed in the subsections given below.

2.1 Traditional Methods

The traditional algorithm for building decision trees is a greedy algorithm which constructs decision tree in top down recursive manner. A typical algorithm for building decision trees is given in figure 1.

The algorithm begins with the original set **X** as the root node. it iterates through each unused attribute of the set **X** and calculates the information gain (IG). The information gain is calculated by deducting conditional entropy of the given attribute with the total entropy. The formulas needed to calculate information gain along with the formula for calculating information gain is given in the subsection given below.

The algorithm then chooses to split on the feature that has the highest

```
function BuildDecisionTree($data, labels$)
    if all labels are same then
        return leaf node for that label
    else
        calculate information gain of all the features
        choose the feature with highest information gain for splitting
        left = BuildDecisionTree(data with f=0, labels with f=0)
        right = BuildDecisionTree(data with f=1, labels with f=1)
        return Tree(f, left, right)
    end if
end function
```

Figure 1: A recursive algorithm for building decision tree

information gain. The Set X is then split by the feature obtained in the previous step to produce the subset of data depending on the value of feature. The decision tree is then built recursively until every element in the subset belongs to the same class, in which case, a terminal node is added to the decision tree with a class label same as the class all its elements belong to.

Entropy and Information Gain

The information gain from the attribute test on the set of instances **X** is the expected reduction in entropy. The algorithm computes the information gain of each feature and then chooses the feature with highest information gain for splitting. The formulas needed to calculate information gain along with formula for calculating information gain is given below.

The total entropy is given by the formula:

$$H(X) = -\sum_{i=1}^{n} p(x_i)\log(p(x_i)), \tag{1}$$

The conditional entropy is given by the formula:

$$H(Y|X) = -\sum_{i=1}^{m}\sum_{j=1}^{n} p(y_i, x_j)\log(\frac{p(y_i, x_j)}{p(x_j)}), \tag{2}$$

and finally, the information gain (IG) is given by the formula:

$$IG(Y|X) = H(Y) - H(Y|X), \tag{3}$$

The information gain is equal to total entropy for an attribute we have a unique class label for each of the given attribute values.

Information gain is a good measure for deciding the relevance of attribute in general. However if we have an attribute that can take large number of distinct values, then splitting feature based on the information gain is not prudent. For example, consider an attribute that contains the instance number of each instance. Now if we use information gain heuristics, this attribute will give the highest information gain value which would depict that we can classify the samples perfectly. This would make the classifier to over fit the training data. Now when we see an unknown instance with value of instance number that is not present in the training set, the classifier would fail to predict the label for it. In order to overcome this problem, we use a new heuristics that uses gain ratio for deciding which feature to split on.

Gain Ratio

The gain ratio biases the decision tree against considering attributes with higher number of distinct values thereby solving the drawback of the information gain. Using gain ratio heuristics for choosing best feature to split upon avoids over-fitting of training data. The gain ratio of an attribute is calculated by dividing its information gain with its information value. The formula for calculating information value is given below:

$$IV(Y|X) = -\sum_{j=1}^{n} p(y, x_j) \log(p(y, x_j)), \tag{4}$$

The gain ratio (GR) is then given by:

$$GR(Y|X) = \frac{IG(Y|X)}{IV(Y|X)}, \tag{5}$$

Therefore we use gain ratio instead of information gain when searching for best feature to split upon, avoiding the problem posed by using information gain heuristics.

Pruning

In case of traditional algorithms for learning decision trees, when decision trees are built, many of the branches may reflect noise or outliers in the training data. In order to combat this over-fitting, we attempt to identify and remove such branches with the goal of improving classification accuracy on the unseen data. This is called tree pruning.

After the decision tree is made using the aforementioned algorithm, we then traverse on each node of the tree and perform a statistical significance test. We initially assume that there is no underlying pattern. Then actual data is used for calculating the extent of deviation from a perfect absence of pattern. Significant pattern is present in the data if degree of deviation is statistically unlikely. We then calculate the probability, that under null hypothesis, a sample of size $v = n + p$ would exhibit the observed deviation from the expected distribution of positive and negative examples[12]. The deviation is measured by comparing actual number of positive and negative examples in each subset, p_k and n_k, with the expected number of positives and negatives, $\hat{p_k}$ and $\hat{n_k}$:

The $\hat{p_k}$ is given by -

$$\hat{p_k} = p \times \frac{\hat{p_k} + \hat{n_k}}{p + n}, \hat{n_k} = n \times \frac{\hat{p_k} + \hat{n_k}}{p + n}, \tag{6}$$

The total deviation is then given by:

$$\Delta = \sum_{k=1}^{d} \frac{(p_k - \hat{p_k})^2}{\hat{p_k}} + \frac{(n_k - \hat{n_k})^2}{\hat{n_k}}, \tag{7}$$

We then examine the value of Δ to see if it confirms or rejects the null hypothesis. With this pruning, the over-fitting of training data can be avoided thereby increasing the accuracy of predictions.

2.2 Genetic Algorithms

Genetic Algorithms (GAs) are adaptive heuristics search algorithm based on the evolutionary ideas of natural selection ad genetics. They are a rapidly growing area of artificial intelligence and are inspired by Darwin's theory about the evolution - survival of the fittest. GAs represent an intelligent exploitation of a random search used to solve optimization problems. Though they are randomized, they exploit historical information to direct the search into the region of better performance within the search space. The idea behind using GAs to solve the search problem in large state-space is that they are good at navigating the state-space and find near-optimal solution which could not have been found otherwise.

We generally start with a population of candidate solution, also called individuals. These individuals are evolved towards the better solution exploring the state-space of the given problem. Each individual is represented using genome encoding. The fitness function is defined such that it takes an individual as its input and gives its fitness value. Therefore the key issues to think about while programming a GA is its encoding and fitness function. The fitness function should be chosen wisely so that it can award fitness score to the individuals even when all the individuals perform badly. This needs to be done because performance of all the individuals is poor at the beginning of the search and even then the fitness function needs to drive the search towards the part of state-space that contains better solution. A meticulous decision about choosing the population size also needs to be done. We should not choose very small population size as it would be insufficient to cover the state-space for finding the solution. Increasing the population size also increases the search time of GA. We have to hand tune this parameter by running the algorithm on various population size so as to determine what should be the good value of population size for our problem.

Once the initial population is chosen, we then use selection schemes for selecting the individuals that are fittest among them. There are various techniques such as fitness proportionate and rank based strategy for selecting the fittest individuals. Once this is done, genetic operators such as mutation and crossover is applied on them to generate the new generation and fitness of new individuals is calculated. It is important that genetic operators are defined in such a way so that once they give an individual after mutation or crossover, the fitness function would still be able to calculate their fitness. Otherwise the search could not be continued. After new individuals are generated, we use some replacement strategy such as complete replacement or elitism so as to choose the initial population for the next iteration. These steps are performed till the fitness of the population converges i.e. fitness of two subsequent generation does not change, or the specified number of generations have reached.

It is also possible that the best performing individual would be lost if we are completely replacing the old individuals with new ones. In order to overcome this, the algorithm remembers the best individual seen so far.

Genetic Algorithms for Decision Trees

From the discussion above, it is clear that we need an encoding, initial population size, fitness function, genetic operators such as mutation and crossover, selection strategy and a replacement strategy in order to solve a problem using GAs. The following subsections discusses about how encoding is done, what are the selection/replacement strategies used, how genetic operators are implemented and how the fitness function is chosen in order to generate decision trees using GAs.

1. **Encoding:** A tree based encoding scheme is used in this implementation for encoding the population. The idea behind using tree based encoding scheme is that trees can directly be encoded and manipulated. The tree consists of a root node which has set of child nodes. These nodes can then either have a leaf node or a decision node. The decision nodes have decision variables associated with them while leaf nodes contains the class label. The tree contains child nodes corresponding to each possible outcome. This is a variable length encoding as the length of encoding is dependent on depth/branching factor of the tree.

2. **Crossover:** The crossover in this encoding scheme is straightforward. We traverse the trees and randomly choose a node for each and swap them to get two different trees. Swapping the two sub-trees allows for large change in behavior preserving good behavior generated by a sub-tree.

3. **Mutation:** Mutation is done by randomly choosing a node in the tree and turning it into leaf node. Mutation is also done by traversing tree until a leaf node is reached and then turning that leaf node into a randomly chosen decision node. The idea behind this is that we just need a valid solution and not necessarily a good solution after this operation. For example even if we use same attribute as decision criteria twice in a given path through the tree, it is still a valid solution.

4. **Selection Strategy:** Selection strategy is needed in order to choose the fittest individuals from the population that are fit for crossover and mutation. We have implemented two types of selection strategy viz. **fitness proportionate** and **rank based**.

   In case of **fitness proportionate** selection strategy, the fitness is assigned to all the individuals using fitness function which is used for associating a probability of selection with each individual. If f_i is the fitness of individual i in the population, then the probability of it being selected is given by:

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_j}, \text{where } N \text{ is the number of individuals in the population.} \tag{8}$$

The **rank order** based selection strategy assigns rank to each individual depending on their fitness. The fittest individuals are then selected with probability p for mutation and probability $1-p$ for crossover where p must be less than 0.2. This is because in general we choose lesser individuals for mutation than crossover.

5. **Replacement Strategy:** Replacement strategy is needed for choosing how many individuals do we need to keep for next generation. We have implemented two types of selection strategy viz. **complete replacement** and **elitism**. In case of **complete replacement** replacement strategy, we replace the entire population again with new randomly generated individuals while we keep an amount of individuals for using as next generation population in case of **elitism** replacement strategy.

6. **Fitness Function:** The fitness function is used by GAs to calculate the fitness of an individual. We have implemented three kind of fitness function based on accuracy, precision and recall given by the individual decision tree on the training data. The fitness function takes an individual as its inputs and gives its fitness as output. In this case our input is a decision tree. We then use this decision tree for predicting labels on training data and calculate its accuracy/precision/recall depending on what fitness function is used and then return the fitness of the individual. We finally chose fitness function based on the accuracy. This is because we wanted to maximize the accuracy of prediction in case of each datasets.

3 Algorithms and Experimental Methods

Traditional Algorithm

There were various parameters to choose in case of traditional algorithm like whether to use gain ratio or information gain for splitting the features, whether we should make algorithm to do pruning or not etc. We chose to use gain ratio as it is a better heuristics than information gain. The results of comparison of algorithm using gain ratio and using information gain is given in the next section. The algorithm also have parameter for using pruning in order to combat over-fitting. The experimentation was done using algorithm with pruning and without pruning. The accuracy, precision and recall was calculated on training set as well as testing set and was compared with the other algorithm. A comparison of time taken in training the algorithms on various datasets is also done.

Genetic Algorithm

There were various tunable parameters such as number of generations, population size, whether to use rank based or fitness proportionate strategy and whether to use complete replacement or elitism as the replacement strategy. The experimentation was done by varying population size and found that a population size of 1000 is a good selection for this problem. Also the number of generations was varied from 100 to 10000 and found that almost all the data sets converge after 800 generations. Hence the number of generations was fixed to 1000 to be safe. We also experimented with the selection strategy and found that rank based selection strategy is good for our problem. We further experimented with replacement strategy and found that the results were same for elitism and complete replacement strategy. The results of these experimentations are given in the next section. The accuracy, precision and recall was calculated on training set as well as testing set and was compared with the other algorithm. A comparison of time taken in training the algorithms on various datasets is also done.

Data Sets

All the datasets used in performing the experiment was obtained from UCI machine learning database [2]. All the datasets were processed by parsers implemented by us specific to each datasets and was converted into integer format from string format. This was done so that we can have homogeneous type of data. This way we decoupled the algorithm from the type of data present in the file i.e. no matter what kind of data we have in the file, the algorithm always gets data in the integer format. Therefore if we want to use any other dataset with this algorithm, all we need to do is write a parser that converts the data into integer format and we could still use this algorithm in predicting labels for that dataset. The details pertaining to each datasets, their properties, and how they are processed is described in the sections given below.

1. **Congressional Voting Data:** This dataset was obtained from UCI machine learning database[1]. This dataset contains two class labels namely *republican* and *democrat*. There are 17 binary attributes in the datasets. The first attribute represents the class label and the rest of attributes define the feature of the instance that is associated with it. Each line in this data file contains an example comprising of 17 attributes. All the attributes were of yes/no type. The given data file had 435 instances in it. We randomly chose 70% of examples for training the classifier and remaining 30% for testing the classifier.

   The class label *democrat* was encoded to 0 and *republican* was encoded to 1. The attribute value y was encoded to 1 and value n was encoded to 0. We also had missing data in some attributes which was represented by '?' in the original data file. We treated this value as third value which meant *abstain* i.e. no opinion was recorded. This value was encoded to 2.

2. **Monk's Problems Data:** This dataset was obtained from UCI machine learning database[3]. The Monk's problem dataset had 6 files 3 each for testing and training the classifier. This dataset contains two class labels namely 0 and 1. There are 8 attributes in the datasets. The first attribute represents the class label and the rest of attributes define the feature of the instance that is associated with it. Each line in this data file contains an example comprising of 8 attributes. Some of the attributes given in this data file had 4 distinct values however none of the attributes took more than 4 distinct values. Each of the given data file had 432 instances in it.

The class label 0 was encoded to 0 and 1 was encoded to 1. The attribute value $1, 2, 3, 4$ were represented as $1, 2, 3, 4$ in integer format after parsing. These files did not have any missing data.

3. **Mushroom Data:** This dataset was obtained from UCI machine learning database[4]. This dataset contains the information about the physical properties of the mushroom specimen. This dataset contains two class labels namely e which stands for edible mushroom and p which stands for poisonous mushroom i.e. they are unfit for consumption. There are 23 attributes in the datasets. The first attribute represents the class label and the rest of attributes define the feature of the instance that is associated with it. Each line in this data file contains an example comprising of 23 attributes. Some of the attributes given in this data file had many distinct values in them. The given data file had 8124 instances in it. We randomly chose 70% of examples for training the classifier and remaining 30% for testing the classifier.

   The class label e was encoded to 1 and p was encoded to 0. The attribute value varied from $a \dots z$ so we encoded them as $a = 1, b = 2, c = 3 \dots$. We also had missing data in some attributes which was represented by '?' in the original data file. There was only one attribute that contained missing values in it so that attribute was dropped.

4. **Splice Junction Data:** This dataset was obtained from UCI machine learning database[5]. This dataset contains data from a molecular biology problem. The goal is to recognize boundaries between introns and exons. This dataset had 2 files 1 each for testing and training the classifier. This dataset contains multiple class labels namely ie, which represents intron-exon boundary, ei, which represents exon-intron boundary, and n which means neither boundary. There are 62 attributes in the datasets. The first attribute represents the class label, the second attribute gives the instance name and the remaining 60 fields are the sequence, starting at position -30 and ending at position +30. Each of these fields is almost always filled by one of a, g, t, c. Each line in this data file contains an example comprising of 62 attributes. All the attributes in this file were multi valued attributes.The given data file had 3190 instances in it. We randomly chose 70% of examples for training the classifier and remaining 30% for testing the classifier.

   The class label ie was encoded to 0, ei was encoded to 1 and n was

Dataset	Train/Test	Heuristic	use pruning?	Accuracy	Precision	Recall
Congressional	Train	IG	no	1.0	1.0	1.0
Congressional	Test	IG	no	0.9389	0.94	0.9038
Congressional	Train	GR	no	1.0	1.0	1.0
Congressional	Test	GR	no	0.9389	0.94	0.9038
Congressional	Train	IG	yes	0.9145	1.0	0.9719
Congressional	Test	IG	yes	0.9549	1.0	0.9342
Congressional	Train	GR	yes	0.9145	1.0	0.9719
Congressional	Test	GR	yes	0.9549	1.0	0.9342

Table 1: Performance results of traditional algorithm with different variations using congressional datasets.

encoded to 2. The attribute value a, c, g, t were represented as $0, 1, 2, 3$ respectively after parsing. The other characters in attributes were also encoded to 3. These files did not have any missing data.

4 Results

Table 1 through 4 shows the results of traditional algorithm on congressional dataset, monks datasets, mushroom dataset and splice junction dataset respectively. The value of parameters were varied as shown in the table and corresponding accuracy, precision and recall was recorded.

Table 5 through 8 shows the results of genetic algorithm on congressional, monks problem, mushroom and splice junction datasets respectively. The value of parameters were varied as shown in the table and corresponding accuracy, precision and recall was recorded. The value of population size was set to 1000 and number of generations was fixed to 1000. These parameters were chosen after experimenting with their different values and recording the accuracy. Here FP means fitness proportionate, RB means rank based, CR means complete replacement and El means elitism.

Table 9 shows the time taken by algorithms to train on each datasets. The time was recorded by choosing gain ratio as heuristics and using tree pruning in case of traditional algorithm and using fitness proportionate selection strategy and elitism as replacement strategy in case of genetic algorithm. The population size was fixed to 1000 and number of generations was also

Dataset	Train/Test	Heuristic	use pruning?	Accuracy	Precision	Recall
Monks 1	Train	IG	no	0.9355	0.9091	0.9677
Monks 1	Test	IG	no	0.8194	0.776	0.8982
Monks 1	Train	GR	no	0.9355	0.8971	0.9838
Monks 1	Test	GR	no	0.8194	0.7593	0.9351
Monks 1	Train	IG	yes	0.8226	0.8704	0.7581
Monks 1	Test	IG	yes	0.7431	0.7664	0.6991
Monks 1	Train	GR	yes	0.9355	0.9091	0.9677
Monks 1	Test	GR	yes	0.8333	0.7951	0.8981
Monks 2	Train	IG	no	0.9408	0.9355	0.9063
Monks 2	Test	IG	no	0.838	0.7195	0.838
Monks 2	Train	GR	no	1.0	1.0	1.0
Monks 2	Test	GR	no	0.8935	0.7857	0.9296
Monks 2	Train	IG	yes	0.9145	0.9291	0.8951
Monks 2	Test	IG	yes	0.8549	0.7374	0.8342
Monks 2	Train	GR	yes	1.0	1.0	1.0
Monks 2	Test	GR	yes	0.8935	0.7857	0.9296
Monks 3	Train	IG	no	0.8279	0.8421	0.8
Monks 3	Test	IG	no	0.6389	0.6915	0.5702
Monks 3	Train	GR	no	0.8525	0.85	0.85
Monks 3	Test	GR	no	0.6296	0.67	0.5878
Monks 3	Train	IG	yes	0.8279	0.8421	0.8
Monks 3	Test	IG	yes	0.6389	0.6915	0.5702
Monks 3	Train	GR	yes	0.8525	0.8621	0.8333
Monks 3	Test	GR	yes	0.6389	0.6915	0.5702

Table 2: Performance results of traditional algorithm with different variations using Monk's Problem datasets.

Dataset	Train/Test	Heuristic	use pruning?	Accuracy	Precision	Recall
Mushroom	Train	IG	no	0.9954	0.9913	1.0
Mushroom	Test	IG	no	0.9943	0.9889	1.0
Mushroom	Train	GR	no	0.9958	0.992	1.0
Mushroom	Test	GR	no	0.9934	0.9872	1.0
Mushroom	Train	IG	yes	0.9954	0.9913	1.0
Mushroom	Test	IG	yes	0.9943	0.9889	1.0
Mushroom	Train	GR	yes	0.9924	0.9858	1.0
Mushroom	Test	GR	yes	0.9881	0.9768	1.0

Table 3: Performance results of traditional algorithm with different variations using Mushroom datasets.

Dataset	Train/Test	Heuristic	use pruning?	Accuracy	Precision	Recall
Splice	Train	IG	no	0.9996	0.9994	0.9991
Splice	Test	IG	no	0.8025	0.7773	0.6339
Splice	Train	GR	no	0.9996	0.9994	0.9991
Splice	Test	GR	no	0.8025	0.7773	0.6339
Splice	Train	IG	yes	0.9167	0.8863	0.8392
Splice	Test	IG	yes	0.8036	0.7535	0.6320
Splice	Train	GR	yes	0.9167	0.8863	0.8392
Splice	Test	GR	yes	0.8036	0.7535	0.6320

Table 4: Performance results of traditional algorithm with different variations using Splice junction datasets.

Dataset	Train/Test	Selection	Replacement	Accuracy	Precision	Recall
Congressional	Train	FP	CR	0.9994	0.9991	0.9954
Congressional	Test	FP	CR	0.9032	0.8653	0.9375
Congressional	Train	RB	CR	0.9985	0.9975	0.9951
Congressional	Test	RB	CR	0.8935	0.8773	0.9271
Congressional	Train	FP	El	0.9145	0.9942	0.9719
Congressional	Test	FP	El	0.8955	1.0	0.9342
Congressional	Train	RB	El	0.934	0.9322	0.9619
Congressional	Test	RB	El	0.9054	1.0	0.9117

Table 5: Performance results of genetic algorithm with different variations using congressional datasets.

fixed to 1000. These parameters were chosen after experimenting with their different values and recording the accuracy.

Figure 2 shows the plot of number of generations vs fitness for the datasets used in this experiment.

5 Discussion

We compare the results of various implementations in this section.

Congressional voting dataset

Table 1 shows the results of experimentation with traditional algorithm by varying its various parameters. It can be seen from the results that there is no change in accuracy, precision and recall when we use the classifier for predicting data on training as well as testing dataset no matter what heuristics we use. This happens because each attribute in this dataset takes at most two values and hence gain ratio heuristics gives results similar to information gain heuristics. It can also be seen that we get perfect accuracy when predicting on training dataset. Therefore we experimented with a version that uses pruning and found that prediction accuracy on unseen data increases. This corroborates the fact that we were over-fitting the training data and pruning helped us in combating over-fitting.

Dataset	Train/Test	Selection	Replacement	Accuracy	Precision	Recall
Monks 1	Train	FP	CR	0.8321	0.8491	0.8654
Monks 1	Test	FP	CR	0.7532	0.696	0.8051
Monks 1	Train	RB	CR	0.8309	0.8472	0.8683
Monks 1	Test	RB	CR	0.7654	0.7152	0.7966
Monks 1	Train	FP	El	0.8226	0.8704	0.7581
Monks 1	Test	FP	El	0.7431	0.7664	0.6991
Monks 1	Train	RB	El	0.8135	0.8291	0.8635
Monks 1	Test	RB	El	0.7374	0.7092	0.8265
Monks 2	Train	FP	CR	0.7925	0.8086	0.7851
Monks 2	Test	FP	CR	0.7532	0.696	0.7784
Monks 2	Train	RB	CR	0.8321	0.8491	0.8654
Monks 2	Test	RB	CR	0.7532	0.696	0.8051
Monks 2	Train	FP	El	0.9408	0.9655	0.875
Monks 2	Test	FP	El	0.83	0.7237	0.7746
Monks 2	Train	RB	El	0.9355	0.9091	0.9677
Monks 2	Test	RB	El	0.8132	0.7856	0.8783
Monks 3	Train	FP	CR	0.7577	0.7532	0.7231
Monks 3	Test	FP	CR	0.6054	0.6374	0.5306
Monks 3	Train	RB	CR	0.7754	0.8032	0.7941
Monks 3	Test	RB	CR	0.6352	0.6723	0.5459
Monks 3	Train	FP	El	0.8177	0.8232	0.77
Monks 3	Test	FP	El	0.63	0.677	0.5502
Monks 3	Train	RB	El	0.8171	0.8229	0.7673
Monks 3	Test	RB	El	0.6365	0.6712	0.5512

Table 6: Performance results of genetic algorithm with different variations using Monk's Problem datasets.

Dataset	Train/Test	Selection	Replacement	Accuracy	Precision	Recall
Mushroom	Train	FP	CR	0.9991	1.0	1.0
Mushroom	Test	FP	CR	0.9933	1.0	1.0
Mushroom	Train	RB	CR	0.9925	0.9968	0.9954
Mushroom	Test	RB	CR	0.9914	0.9964	0.9995
Mushroom	Train	FP	El	0.9953	0.99	0.9975
Mushroom	Test	FP	El	0.9946	0.9895	0.9894
Mushroom	Train	RB	El	0.9955	0.9991	0.9977
Mushroom	Test	RB	El	0.9933	0.9951	0.9981

Table 7: Performance results of genetic algorithm with different variations using Mushroom datasets.

Dataset	Train/Test	Selection	Replacement	Accuracy	Precision	Recall
Splice	Train	FP	CR	0.9619	0.9457	0.9312
Splice	Test	FP	CR	0.7951	0.805	0.6343
Splice	Train	RB	CR	0.9551	0.949	0.9531
Splice	Test	RB	CR	0.8031	0.7743	0.6742
Splice	Train	FP	El	0.9755	0.9597	0.9587
Splice	Test	FP	El	0.813	0.793	0.6585
Splice	Train	RB	El	0.9639	0.9493	0.9482
Splice	Test	RB	El	0.8015	0.8132	0.6365

Table 8: Performance results of genetic algorithm with different variations using Splice junction datasets.

Dataset	Training Time (traditional)	Training Time (GA)
Congressional	41 ms	226505 ms
Monks 1	12 ms	266819 ms
Monks 2	20 ms	365224 ms
Monks 3	15 ms	268876 ms
Mushroom	246 ms	918150 ms
Splice	419 ms	1415455 ms

Table 9: Training time taken by various algorithms on all datasets.

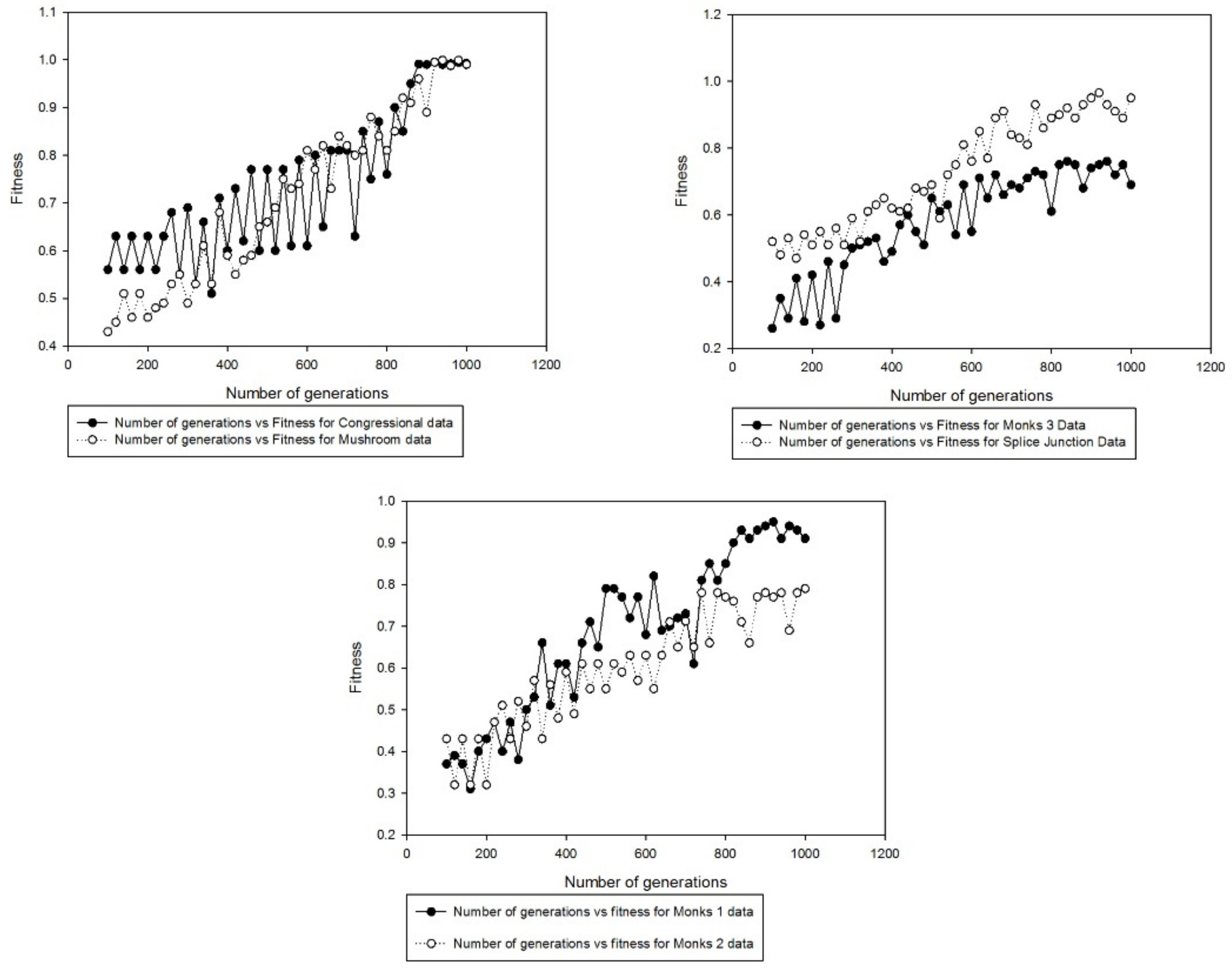

Figure 2: Number of generations vs fitness plot for various datasets.

Table 5 shows the results of experimentation with genetic algorithm by varying its various parameters. It can be seen from the results that we get highest accuracy when we use rank based selection strategy and elitism as replacement strategy. All the other variations give almost similar results.

It can be seen from the results of Table 1 and 5 that traditional algorithm performs better as compared to genetic algorithm in this case however the difference in prediction accuracy is very small.

Monk's Problem dataset

Table 2 shows the results of experimentation with traditional algorithm by varying its various parameters. It can be seen from the results that there is no change in accuracy, precision and recall when we use the classifier for predicting data on training as well as testing dataset no matter what heuristics we use. This might be happening because the attributes does not have many distinct values in case of this dataset.We also experimented with a version that uses pruning and found that prediction accuracy on unseen data is almost similar corroborating the fact that we were not over-fitting the training data. It can also be seen that the classifier doesn't do well on the Monks 3 dataset. This would be because the given dataset contains more noise as compared to the other datasets thereby decreasing the prediction accuracy.

Table 6 shows the results of experimentation with genetic algorithm by varying its various parameters. It can be seen from the results that we get highest accuracy when we use rank based selection strategy and complete replacement as replacement strategy in case of Monks 1 data. Monks 2 data gives highest accuracy when we use fitness proportionate selection scheme and elitism as replacement strategy. Monks 3 data gives highest accuracy when we use rank based selection scheme and elitism as replacement method. All the other variations give almost similar results.

It can be seen from the results of Table 2 and 6 that traditional algorithm performs significantly better than genetic algorithm on all the given datasets of Monk's problem. This could be because the state-space of Monk's problem is very large and genetic algorithm would not be able to explore it fully.

Mushroom dataset

It can be seen from the table 3 that varying the parameters of traditional algorithm does not affect the accuracy, precision and recall in this case. This might be happening because the data set might not be having any noise as well as all the attributes take almost similar number of distinct values thereby rendering gain ratio and pruning useless in this case.

Table 7 shows similar results as table 3 substantiating the fact that genetic algorithm performs similar to traditional algorithm.

Splice Junction dataset

It is evident from table 4 that there is no change in accuracy, precision and recall when we use the classifier for predicting data on training as well as testing dataset no matter what heuristics we use. This might be happening because the attributes does not have many distinct values in case of this dataset. We also experimented with a version that uses pruning and found that prediction accuracy on unseen data is almost similar to the version that does not use pruning. This might be happening because the datasets might not be having much noise in it.

Table 8 shows similar results as table 4 substantiating the fact that genetic algorithm performs similar to traditional algorithm in this case.

Training time on datasets

It can be seen from table 9 that training classifier using traditional algorithm requires substantially lesser time than genetic algorithm. This happens because genetic algorithm uses many agents to search the space which requires a large amount of computation time and hence training the algorithm using GA takes longer.

Fitness of individuals with respect to generation

Figure 2 shows the number of generations vs fitness plot for various datasets . The population size was fixed to 1000 for plotting these graphs. It can be seen from the graphs that all the classifiers had bad fitness when algorithm started its execution and eventually their fitness kept growing. The algorithm converged in almost all the cases near 800th generation. After that there was very less fluctuation of fitness in all the cases. Hence the parameter for number of generations was fixed to 1000. It might have been possible that the algorithm would have found even better solution. This would require more number of generations which would consume even more time.

6 Conclusions

It have been shown that traditional algorithm have performed well in almost all the cases as compared to the genetic algorithm. It is also seen that training classifier using genetic algorithm takes longer as compared to traditional algorithm. However GAs are capable of solving a large variety of problems where traditional decision tree algorithm might fail. For example, we can get really good accuracy in case of noisy data with genetic algorithm while the traditional algorithm would fail to do that as it might also learn noise while training on the given data.

The limitation of our approach was that we were doing completely random search and then trying to move towards a favorable state. Combining ID3 algorithm with GAs might help in overcoming this limitation. This approach would have some idea about the point from where search could be started and it might get to more optimal solution while exploring the state-space.

References

[1] D.J. Newman A. Asuncion. Uci machine learning repository congressional voting data[http://archive.ics.uci.edu/ml/datasets/congressional+voting+records], 2007.

[2] D.J. Newman A. Asuncion. Uci machine learning repository [http://www.ics.uci.edu/ mlearn/mlrepository.html], 2007.

[3] D.J. Newman A. Asuncion. Uci machine learning repository monk's problems data[http://archive.ics.uci.edu/ml/datasets/monk

[4] D.J. Newman A. Asuncion. Uci machine learning repository mushroom data[http://archive.ics.uci.edu/ml/datasets/mushroom], 2007.

[5] D.J. Newman A. Asuncion. Uci machine learning repository splice junction data[http://archive.ics.uci.edu/ml/datasets/molecular+biology+2007.

[6] L.H. Bodlaender and H. Zantema. Finding small equivalent decision trees is hard. *International Journal of Foundations of Computer Science*, 11(2):343–354, 2000.

[7] D. L. Goldberg. *Genetic Algorithms in Search, Optimization, and Machine Learning.* Addison-Wesley, 1989.

[8] H. Vafaie K. DeJong J. Bala, J. Huang and H. Wechsler. Hybrid learning using genetic algorithms and decision tress for pattern classification. *Proceedings of the 14th International Joint Conference on Artificial Intelligence, Montreal, Canada*, pages 719–724, 1995.

[9] T. M. Mitchell. *Machine Learning.* McGraw hill, 1997.

[10] A. Papagelis and D. Kalles. Ga tree: genetically evolved decision trees. *Proceedings of 12th IEEE International Conference on Tools with Artificial Intelligence*, pages 203–206, 2000.

[11] P. E. Hart R. O. Duda and D. G. Stork. *Pattern Classification, 2nd Ed.* Wiley interscience, 2001.

[12] S. Russell and P. Norvig. *Artificial Intelligence : A Modern Approach, 3rd Ed.* Prentice Hall, 2009.

[13] D. H. Wolpert and W. G. Macready. No free lunch theorems for optimization. *Evolutionary Computation, IEEE Transactions on*, 1(1):67–82, 1997.