Design of Genetic Programming Algorithm with simultaneous Feature Selection and its implementation for handling Big Data

PROJECT REPORT

Submitted in partial fulfillment of the requirements for the award of the degree

of
BACHELOR OF TECHNOLOGY
in
COMPUTER SCIENCE AND ENGINEERING

Submitted by:
Parul Gupta

Guided by:
Dr. Aruna Tiwari,
Assistant Professor,
Computer Science and Engineering,
Indian Institute of Technology Indore

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November, 2016

CANDIDATE'S DECLARATION

I hereby declare that the project entitled **Design of Genetic Programming Algorithm with simultaneous Feature Selection and its implementation for handling Big Data** submitted in partial fulfillment for the award of the degree of Bachelor of Technology in **Computer Science and Engineering** completed under the supervision of **Dr. Aruna Tiwari, Assistant Professor, Computer Science and Engineering, IIT Indore** is an authentic work.

Further, I declare that I have not submitted this work for the award of any other degree elsewhere.

Parul Gupta
130001026
Discipline of Computer Science and Engineering
Indian Institute of Technology Indore

CERTIFICATE
by BTP Guide

It is certified that the declaration made by the student is correct to the best of my knowledge and belief.

Dr. Aruna Tiwari,
Assistant Professor,
Discipline of Computer Science and Engineering,
Indian Institute of Technology Indore
(Project Guide)

PREFACE

This report on **'Design of Genetic Programming Algorithm with simultaneous Feature Selection and its implementation for handling Big Data'** is prepared under the supervision of *Dr. Aruna Tiwari*, Assistant Professor, Computer Science and Engineering, IIT Indore.

Through this report, we have tried to provide a detailed description of the technologies that have been used to design and modify the Genetic Programming Algorithm to produce an efficient novel algorithm for classification. We have tried and tested the same on variable datasets of medical domain. Further, we have designed the scalable novel algorithm corresponding to the same for handling big data. We have also tried to implement this proposed algorithm to the best of our abilities.

We have tried our best to explain the proposed algorithms in detail. The comparison of proposed algorithm with already existing models is also discussed.

ACKNOWLEDGEMENTS

I would like to thank my btp supervisor, **Dr. Aruna Tiwari**, for her constant support in structuring the project and for her valuable feedback which helped me in the course of the project. She gave me the opportunity to discover and work in this domain. She guided me thoroughly and pulled me out of the craters of failures I faced all through the period.

I am especially grateful to **Dr. Arpit Bhardwaj** who guided me, explained me the problems while working with Genetic Programming and provided me the initial pathway for starting this project in right manner. He simultaneously guided me with useful direction to proceed along whenever necessary.

I also appreciate the efforts and time given by **Neha Bharill**, PhD scholar under Dr. Aruna Tiwari, who guided me by familiarising me with big data.

I am really thankful to my BTP partner, **Simmi Malhan**, for her contribution and support throughout the course of the project.

I am also thankful to all my family members, friends and colleagues who have been a constant source of motivation. Finally, I offer sincere thanks to everyone else who knowingly or unknowingly helped me complete this project.

Parul Gupta
130001026
Discipline of Computer Science and Engineering
Indian Institute of Technology Indore

ABSTRACT

Genetic programming (GP) is an evolutionary optimisation method that produces functional programs to solve a given task. It has seen various modifications over its existence for a decade to decrease time taken for its working and to increase accuracy. One such method is incorporation of feature selection for reducing size of data and improving performance of machine learning algorithm.

The term 'big data' has been the most popular topic in recent years in practice, for realising the value and volume of data. These data sets are so large or complex that traditional data processing applications are inadequate to deal with them. Challenges include analysis, capture, data curation, search, sharing, storage, transfer, visualisation, querying, updating and information privacy. Many information technologies and software are proposed to deal with big data, such as Hadoop, NoSQL databases, Apache Spark and cloud computing. Big data has been exploited to make it at use by various optimised algorithms and parallel computing concepts. In healthcare industry, the demand for storage and maintenance of large chunks of patients' data is ever increasing due to rising population and technology over the years which has resulted in the increase of details about clinical and laboratory tests, imaging, prescription and medication. These data can be called 'big data', because of their size, complexity and diversity. Big data analytics can be targeted to aim at improving patient care and identifying preventive measures proactively to save lives and recommend lifestyle changes for a peaceful and healthier life.

In our project, we have proposed a novel algorithm of genetic programming with simultaneous feature selection.The results on testing over two datasets, namely Parkinson Disease[2] and Diabetes Retinopathy[3], from UCI repository suggest that the feature selection approach incorporated in the process of hybrid GP[1] is more efficient and superior than regular hybrid GP. We also explore the problems faced by GP when applied on bigger sets of data and optimize the novel GP algorithm in such a way that it can be applied effectively to handle big data. We have implemented it in Apache Spark 1.6.2 on the replicated big data sets on Microsoft Azure clusters and recorded results. We are trying to optimize the implementation further for reducing time, thereby increasing its usability.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

This chapter highlights the background and motivation for the project. The problem statement has been described of the project and the importance of the results is also clearly portrayed. Towards the end, the objectives and expectations to solve the problem statement as far as possible.

1.1 Background

Classifier is an important concept of machine learning which involve training of the machine according to the data provided to it so that it can predict the nature of datapoint given to it. Medical classification is the process of transforming descriptions of medical diagnoses and procedures into machine classifier. The diagnoses and procedures are usually taken from a variety of sources within the healthcare record, such as the transcription of the physician's notes, laboratory results, radiologic results, and other sources and converted into useful dataset for training of the machine to predict useful results efficiently.

The motivation behind this project lies in the health domain and the data collected exponentially every coming day in this field. With the advancement of technology in medical domain, artificial intelligence can help in increasing the lifespan of this species by various ways and implementation. One such implementation is classification of disease and testing people with the machines to analyse if they are prone to disease or not. It is necessary as it can help to predict effectively the disease a person may have and can take appropriate precautions. In broader aspect, it can save the lives of many when implement for real time use. In this project, we try to propose, implement and compare modified variations of machine learning algorithm (evolutionary algorithm - Genetic Programming) to build a better classifier that enables us to train and test the dataset for benefit and progress in health domain. We aim to implement our algorithm for Parkinson Disease (PD)[2] and also for diabetes retinopathy[3] to classify whether a person is affected or not.

Genetic Programming is an evolutionary algorithm and if not optimised, takes a considerable amount of time for implementation. Hence, it needs to be modified and optimised to fully utilise its potential benefits of machine learning for big data sets. Multi-objective GP is a more sophisticated technique which uses more than one objectives for evaluating fitness function and henceforth, gives more accurate and desired outcomes.

Feature selection is important while classification. There are standardly two types of feature selection possible, namely wrapper method and filter method. To overcome the disadvantages of both of the methods, a hybrid method for feature selection is proposed to increase the accuracy of the classifier. Feature selection decrease the size of the data input in use and give a direction to the evolution of genetic programming classifier. Thus, a novel generic multi-objective based hybrid genetic programming algorithm is being proposed and implemented for small datasets from UCI repositories.

With the advancement in living, a huge amount of data containing useful information, called Big Data, is generated on a daily basis. Traditional machine learning has been largely concerned with developing techniques for small or modestly sized datasets. These techniques fail to scale up well for large data, a situation becoming increasingly common in today's world. 'Big data' is gaining momentum in every sense of the world. The term 'big data' often refers simply to the use of predictive analytics, user behaviour analytics, or certain other advanced data analytics methods that extract value from data, and seldom to a particular size of data set. In medical domain, the data collected from various clinics, hospitals and laboratories is increasing exponentially. To make technology beneficial for medicinal field, these data are utilised through and through everywhere. Hence, it is essential to introduce such algorithms that work aptly for big data. For processing such tremendous volume of data, there is a need of Big Data frameworks such as Hadoop MapReduce, Apache Spark etc. Among these, Apache Spark performs up to 100 times faster than conventional frameworks like Hadoop MapReduce. For the effective analysis and interpretation of this data, scalable Machine Learning methods are required to overcome the space and time bottlenecks. Most of the IT giants today are expanding their databases and using technologies to handle big data requirements. For example, Walmart's databases are estimated to contain more than 2.5 petabytes of data, and Facebook stores more than 30 Petabytes of user generated data[**20**]. This thesis introduces a technique to distribute GP over Apache Spark cluster. For the efficient novel algorithm for small or modestly sized datasets, we designed a distributed version of the same and tried implementing it on Apache Spark Azure cluster. The dataset of UCI repository is replicated manifolds on the cluster itself produce a dataset big enough to be considered big data.

1.2 Objectives

The project is divided into two parts - one to increase the accuracy of the classifier, and the other to handle the computational, storage, and communications bottlenecks in big data. We stress on fixing the domain as healthcare because it is the one where speed of the algorithm can be compromised but accuracy cannot be compromised at any cost. Thus, we had the following objectives:

1. Propose a novel algorithm for Genetic Programming with simultaneous Feature Selection.
2. Implement this algorithm and exhaustively test on benchmark datasets.
3. Enhance this algorithm for handling and extracting knowledge from big data.
4. Finally, implement this scalable algorithm on real Apache Spark clusters for handling Big Data.

Chapter 2

Literature Review

The problem statement and the objectives mentioned in chapter 1 are real life problems pertaining to classification. Accuracy and efficient classifier is very important in many domains such as in medical domain. For example, with the advent of machine learning, early detection of a disease helps to control and cater to the disease in a better and effective manner. Many classifiers have been designed and put to implementation before for medical diagnosis which use genetic programming as their base algorithm. Increasing the accuracy of the classifier has always been the struggle. One of the approaches of machine learning is evolutionary approach. In this chapter, basics of evolutionary algorithm are discussed, focusing on the approach used in this project - Genetic Programming. Further the chapter focuses on big data concepts and how its handling is done.

2.1 Evolutionary Algorithm

In artificial intelligence, an evolutionary algorithm (EA) is a subset of evolutionary computation, a generic population-based meta-heuristic optimization algorithm. An EA uses mechanisms inspired by biological evolution, such as reproduction, mutation, recombination, and selection. Candidate solutions to the optimization problem play the role of individuals in a population, and the fitness function determines the quality of the solutions. Evolution of the population then takes place after the repeated application of the above operators. Evolutionary algorithms often perform well approximating solutions to all types of problems because they ideally do not make any assumption about the underlying fitness landscape; this generality is shown by success in fields as diverse as engineering, art, biology, economics, marketing, genetics, operations research, robotics, social sciences, physics, politics and chemistry.

In most real applications of EAs, computational complexity is a prohibiting factor. In fact, this computational complexity is due to fitness function evaluation. Fitness approximation is one of the solutions to overcome this difficulty. However, seemingly simple EA can solve complex problems; therefore, there may be no direct link between algorithm complexity and problem complexity. In Figure 2.1, the cycle of evolutionary algorithm is depicted which is iterated through every generation. In this thesis, Genetic Programming has been used and thus, in following sections, the process undertaken in Genetic Programming is seen in detail along with other related components.

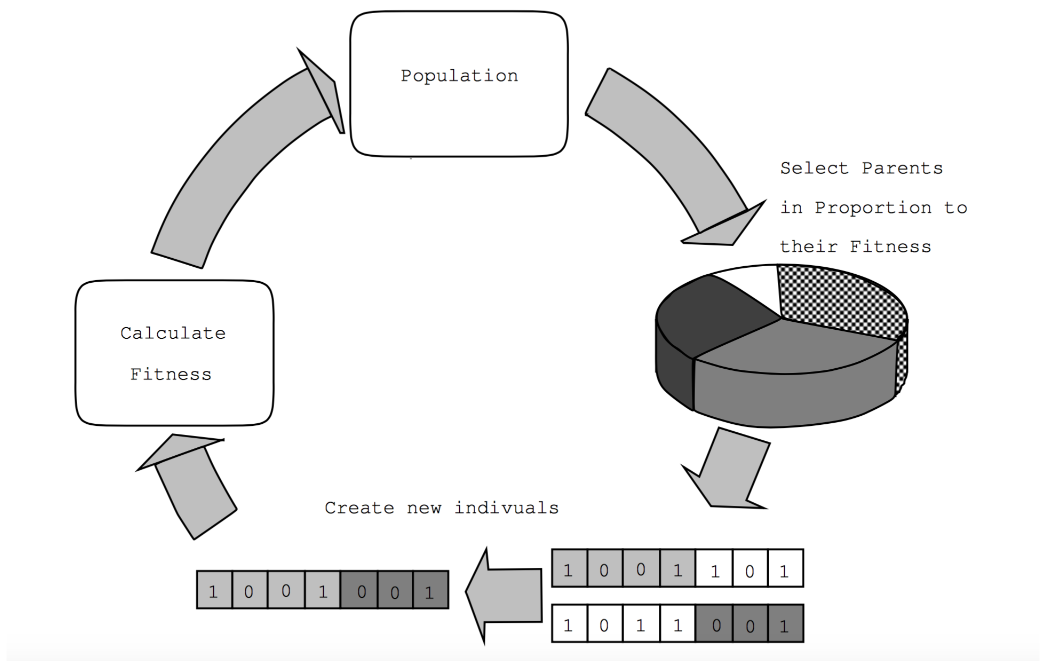


FIGURE 2.1: Evolutionary Algorithm Cycle

2.2 Genetic Programming

In artificial intelligence, Genetic Programming (GP) is a technique whereby computer programs are treated as a set of genes that are then modified (evolved) using an evolutionary algorithm. The result is a computer program able to perform well in a predefined task. GP can indeed be seen as an application of evolutionary algorithms to problems where each individual is a computer program. The methods used to encode a computer program in an artificial chromosome and to evaluate its fitness with respect to the predefined task are central in the GP technique and still a subject of active research.

The executional steps of genetic programming[8] are as follows:

- Randomly create an initial population (generation 0) of individual computer programs composed of the available functions and terminals.
- Iteratively perform the following sub-steps (called a generation) on the population until the termination criterion is satisfied:
  1. Execute each program in the population and evaluate it's fitness using the problem's fitness measure.
  2. Select one or two individual program(s) from the population with a probability based on fitness (with reselection allowed) to participate in the genetic operations in (3).
  3. Create new individual program(s) for the population by applying the following genetic operations with specified probabilities:
     (a) **Reproduction:** Copy the selected individual program to the new population.
     (b) **Crossover** Create new offspring program(s) for the new population by recombining randomly chosen parts from two selected programs.
     (c) **Mutation:** Create one new offspring program for the new population by randomly mutating a randomly chosen part of the selected program.
- After the termination criterion is satisfied, the single best program in the population produced during the run (the best-so-far individual) is harvested and designated as the result of the run. If the run is successful, the result may be a solution (or approximate solution) to the problem.

2.2.1 Reproduction

This is one of the basic operators of genetic programming. In this, selected individuals copy themselves into the new population. It is effectively the same as one individual surviving into the next generation. The best solutions are directly passed on to the next generation, following the concept of Darwin's survival of the fittest.

2.2.2 Mutation

The mutation operator encourages genetic diversity amongst solutions and attempts to prevent the genetic algorithm converging to a local minimum by stopping the solutions becoming too close to one another. In mutating the current pool of solutions, a given solution may change entirely from the previous solution. By mutating the solutions, a genetic program can reach an improved solution solely through the mutation operator. It, thus, adds diversity in the given population, giving chances to the newer features and functions to find their place in the tree.

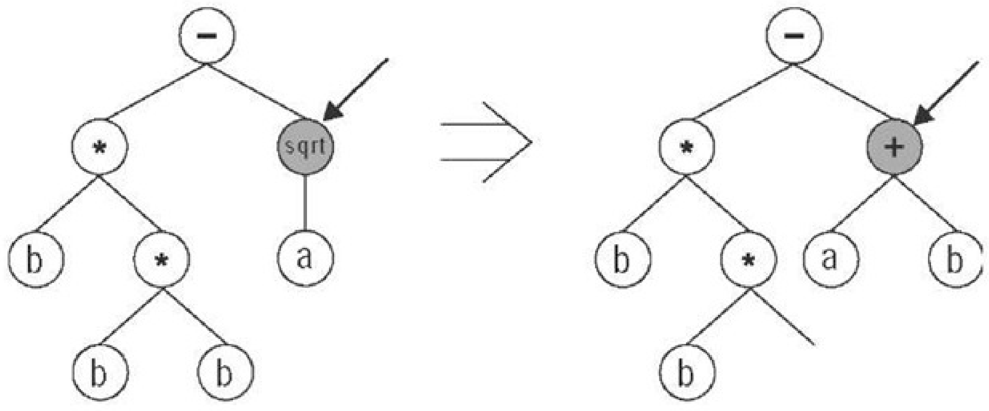


FIGURE 2.2: Tree Mutation

2.2.3 Crossover

Crossover is the process of taking more than one parent solutions (chromosomes) and producing a child solution from them. By recombining portions of good solutions, the genetic program is more likely to create a better solution. There are a number of different methods for combining the parent solutions, including the edge recombination operator (ERO) and the 'cut and splice crossover' and 'uniform crossover' methods. The crossover method is often chosen to closely match the chromosome's representation of the solution. Similarly, crossover methods may be particularly suited to certain problems.Thus, we generate new offsprings by exchanging features of two parents. This is applied on an individual by simply switching one of its nodes with another node from another individual in the population. This is done to improve the quality of the trees in the given population using the same function set and feature set as in the present trees. The different types of crossover used in the project are as follows:

- **Standard Crossover**
  Two new offsprings are generated from two parents by exchanging the subtrees at a random point and best two out of the four (parents+offsprings) are passed on to next generation.

- **Standard four-time Crossover**
  Four new offsprings are generated from two parents by exchanging the subtrees at a random point and best two out of the six (parents+offsprings) are passed on to next generation.

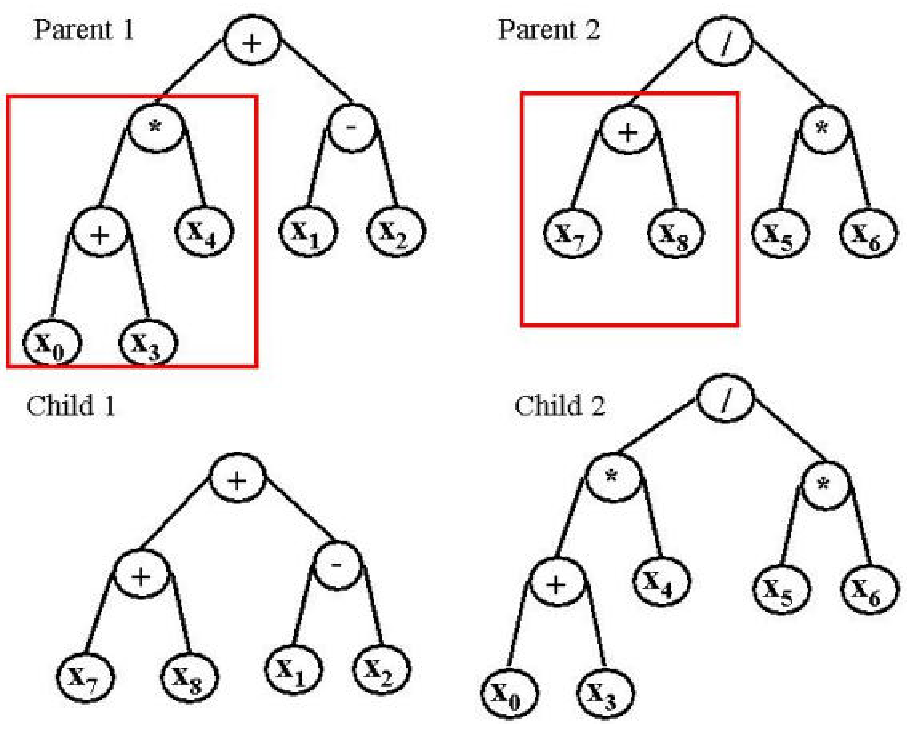


FIGURE 2.3: Tree Crossover

- **Two point Crossover**
  In this, two random points are chosen of the two parents and the subtrees from those two points are exchanged. Best two out of the four (parents+offsprings) are passed on to the next generation.
- **Hill Climbing Crossover**
  Hill climbing is an iterative algorithm that starts with an arbitrary solution to a problem, then attempts to find a better solution by incrementally changing a single element of the solution. If the change produces a better solution, an incremental change is made to the new solution, repeating until no further improvements can be found. In its application as crossover operator, offsprings are generated by exchanging subtrees of two parents at one random point iteratively till the child is no longer better than its immediate parent.

2.2.4 Expression Tree

An expression tree is a special kind of a binary tree used to represent expressions, either algebraic or boolean, and may contain both unary and binary operators. In this, each internal node corresponds to operator and each leaf node corresponds to operand. In genetic programming, expression tree is used as a classifier to evaluate the features of the dataset.

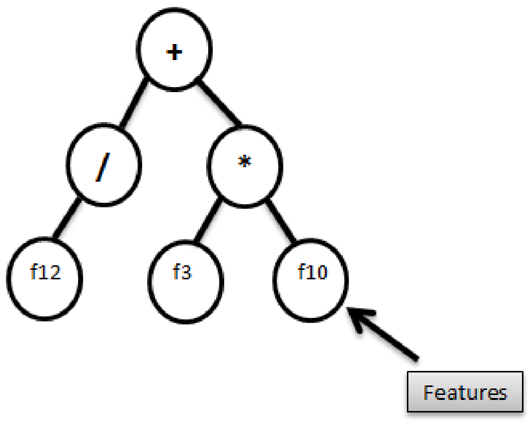


FIGURE 2.4: An Expression Tree: here, a tree (for a class) in a classifier

2.2.5 Fitness Function

A fitness function is a particular type of objective function that is used to summarise, as a single figure of merit, how close a given design solution is to achieving the set aims. The fitness function takes a candidate solution to the problem as input and produces as output of how 'fit' or how 'good' the solution is with respect to the problem in consideration. Calculation of fitness value is done repeatedly in GP and therefore it should be sufficiently fast. A slow computation of the fitness value can adversely affect a GA and make it exceptionally slow. In most cases the fitness function and the objective function are the same as the objective is to either maximise or minimise the given objective function.

2.2.6 Intron Detection

In nature, introns denote DNA segments in genes with information that is not expressed in proteins[15]. In GP, introns are referred to the part of the binary expression tree that need optimisation and are uselessly calculated. For faster and easier computations, these non optimised section must be reduced into simpler forms. This increases the scope for better computation and more reliable features in the classifier. This is one of the most important element in optimisation of trees in a classifier for Genetic Programming.

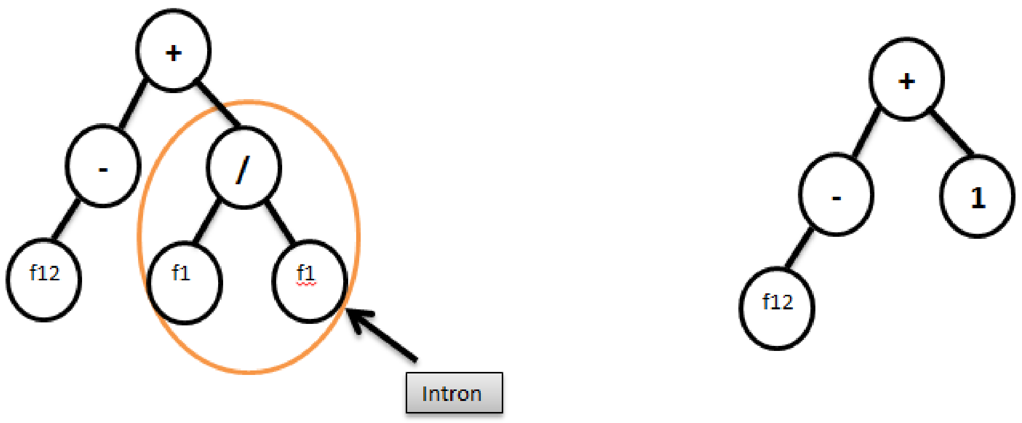


FIGURE 2.5: Intron Detection and Elimination

2.3 Feature Selection

The process of selecting best features out of the feature set in the given database is referred as feature selection. In machine learning and statistics, feature selection, also known as variable selection, attribute selection or variable subset selection, is the process of selecting a subset of relevant features (variables, predictors) for use in model construction. Feature selection techniques are used for three reasons:

- Simplification of models to make them easier to interpret by researchers/users.
- Shorter training time.
- Enhanced generalization by reducing overfitting.

2.3.1 Filter Method

In this, the subset selection procedure is independent of the learning algorithm and is generally a pre-processing step[16]. This leads to a faster learning pipeline but it is possible for the criterion used in the pre-processing step to result in a subset that may not work very well downstream in the learning algorithm. Thus, some useful features might be eliminated during feature selection while some useless ones might be incorporated, destroying the dataset and hence, decreasing the efficiency of classifier.

2.3.2 Wrapper Method

In this, the subset selection takes place based on the learning algorithm used to train the model itself. The expression wrapper approach covers the category of variable subset selection algorithms that apply a learning algorithm in order to conduct the search for the optimal or a near-optimal subset[16]. Every subset that is proposed by the subset selection measure is evaluated in the context of the learning algorithm. This means that computationally intensive learning algorithms cannot be used as it may lead to overfitting of the machine learning to the training database.

2.4 Preliminary of Multi-Objective

When an optimization problem involves more than one objective function, the task of finding one or more optimum solutions is known as multi-objective optimization[11]. It is a more natural way of representation of real world scenarios. The presence of multiple objectives (may or not be conflicting) is natural in many real world cases. Since no one solution can be termed as an optimum solution to multiple conflicting objectives, the resulting multi-objective resorts to a number of trade-off optimal solutions. Conventional optimization methods can at best find one solution in one simulation run, thereby making them inconvenient to solve multi objective problems. Whereas, Evolutionary Algorithms (like Genetic Programming) can find multiple optimal solutions in one single simulation run due to its population approach. This makes EA the ideal candidates for solving multi-objective problems.

For a nontrivial multi-objective optimization problem, there does not exist a single solution that simultaneously optimizes each objective. In that case, the objective functions are said to be conflicting, and there exists a (possibly infinite) number of Pareto optimal solutions. A solution is called nondominated, Pareto optimal, Pareto efficient or noninferior, if none of the objective functions can be improved in value without degrading some of the other objective values. The goal may be to find a representative set of Pareto optimal solutions, and/or quantify the trade-offs in satisfying the different objectives, and/or finding a single solution that satisfies the subjective preferences of a human decision maker (DM).

In the thesis, we use the concept of multi-objectivity in the sense that we find the features or the attributes most useful for the classification and use these good feature dataset to further make modifications in the classifiers. This is done so that we get better classifier as soon as possible. Also, it reduces the dataset size, thus, exploiting the dataset, instead of simply exploring it throughout. Exploitation is one of the most used ways of machine learning to get the best classifier in a given surrounding.

2.5 Basics of Classification

In machine learning and statistics, classification is the problem of identifying to which of a set of categories (sub-populations) a new observation belongs, on the basis of a training set of data containing observations (or instances) whose category membership is known. Classification is an example of pattern recognition. In the terminology of machine learning, classification is considered an instance of supervised learning, i.e. learning where a training set of correctly identified observations is available. The corresponding unsupervised procedure is known as clustering, and involves grouping data into categories based on some measure of inherent similarity or distance. An algorithm that implements classification, especially in a concrete implementation, is known as a classifier. The term 'classifier' sometimes also refers to the mathematical function, implemented by a classification algorithm, that maps input data to a category. Example of classifications are: text categorisation (e.g., spam filtering), fraud detection, optical character recognition, machine vision (e.g., face detection), natural-language processing (e.g., spoken language understanding), market segmentation (e.g.: predict if customer will respond to promotion), bioinformatics (e.g., classify proteins according to their function), etc.[17]

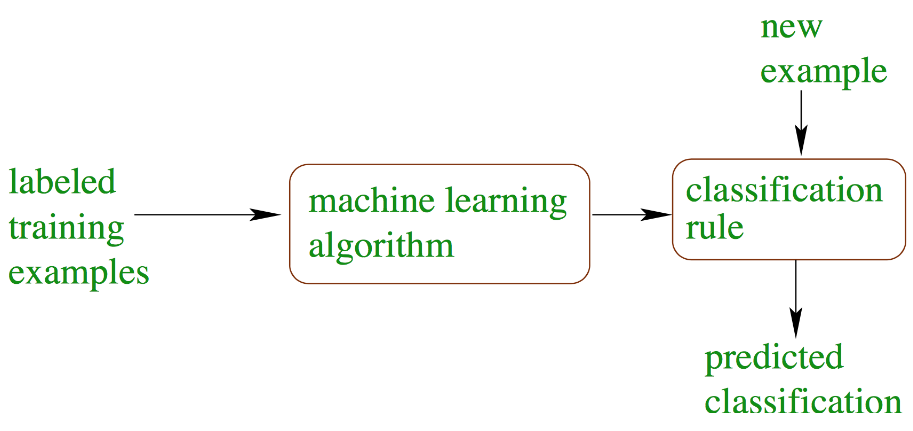


FIGURE 2.6: Classification Flowchart

2.6 Survey on Big Data

Big data is a term that describes the large volume of data - both structured and unstructured - that inundates a business on a day-to-day basis. But it's not the amount of data that's important. It's what organisations do with the data that matters. Big data can be analysed for insights that lead to better decisions and strategic business moves. Industry analyst Doug Laney articulated the now-mainstream definition of big data as the three Vs[18]:

1. **Volume**
   Organizations collect data from a variety of sources, including business transactions, social media and information from sensor or machine-to-machine data. In the past, storing it would've been a problem - but new technologies (such as Hadoop) have eased the burden.

2. **Velocity**
   Data streams in at an unprecedented speed and must be dealt with in a timely manner. RFID tags, sensors and smart metering are driving the need to deal with torrents of data in near-real time.

3. **Variety**
   Data comes in all types of formats - from structured, numeric data in traditional databases to unstructured text documents, email, video, audio, stock ticker data and financial transactions.

We focus on two main topics of interest as enlisted below:

- **Importance:** The importance of big data doesn't revolve around how much data we have, but what we do with it. We can take data from any source and analyse it to find answers that enable 1.) cost reduction; 2.) time reduction;

3.) new product development and optimised offerings; 4.) smart decision making. When we combine big data with high-powered analytics, we can accomplish business-related tasks such as:

  - Determining root causes of failures, issues and defects in near-real time.
  - Generating coupons at the point of sale based on the customer's buying habits.
  - Recalculating entire risk portfolios in minutes.
  - Detecting fraudulent behaviour before it affects your organisation.

- **Health Care:** Patient records, treatment plans, prescription information and other relevant information all collected together from various sources can amount to large amount of data. When it comes to health care, everything needs to be done quickly, accurately and, in some cases, with enough transparency to satisfy stringent industry regulations. When big data is managed effectively, health care providers can uncover hidden insights that improve patient care. A number of use cases in healthcare are well suited for a big data solution. Some academic or research focused healthcare institutions are either experimenting with big data or using it in advanced research projects. Those institutions draw upon data scientists, statisticians, graduate students, and the like to wrangle the complexities of big data. This is done with the belief that prevention is better than cure. Thus, analysis of big data is important for making technology advancement worth it.

2.7 Big Data Handling Framework

2.7.1 Apache Spark

Apache Spark is an open source cluster computing framework. Originally developed at the University of California, Berkeley's AMPLab, the Spark codebase was later donated to the Apache Software Foundation, which has maintained it since. Spark is optimised for iterative algorithms and interactive data analysis, which perform cyclic operations on the same set of data. Spark provides an interface for programming entire clusters with implicit data parallelism and fault-tolerance. Apache Spark is a lightning-fast cluster computing designed for fast computation. It was built on top of Hadoop MapReduce and it extends the MapReduce model to efficiently use more types of computations which includes Interactive Queries and Stream Processing. Languages supported by Apache Spark are Java, Scala, Python and R. Operating system are Linux, Windows and OSX.

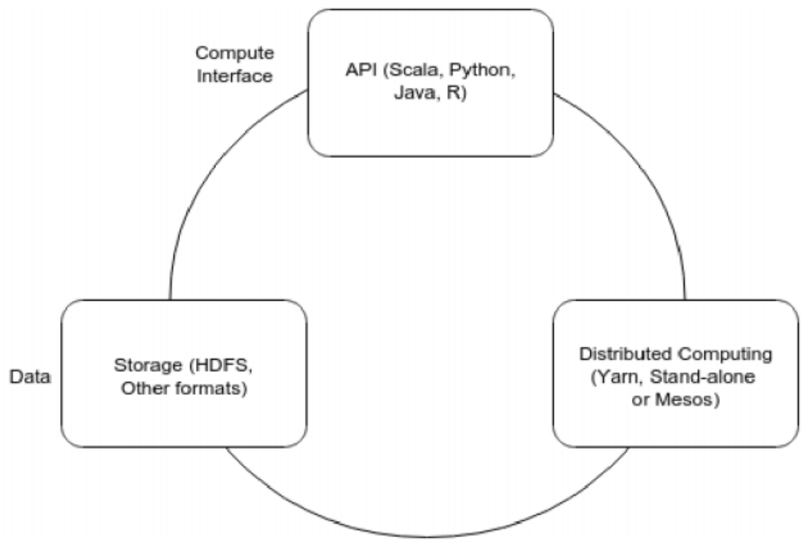

Compute Interface
API (Scala, Python, Java, R)
Data
Storage (HDFS, Other formats)
Distributed Computing (Yarn, Stand-alone or Mesos)

FIGURE 2.7: Components of Spark

The Spark project stack currently is comprised of Spark Core and four libraries that are optimized to address the requirements of four different use cases. Individual applications will typically require Spark Core and at least one of the libraries. These libraries are:

- Spark SQL
- Spark Streaming
- MLlib
- GraphX
- Spark R

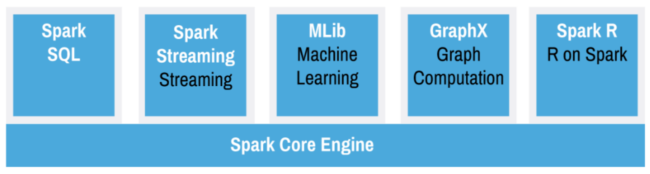


FIGURE 2.8: Apache Spark Architecture

Spark can run over a variety of cluster managers, a simple cluster manager included in Spark itself known as Standalone Scheduler, on Hadoop YARN [6], or on Apache Mesos [7]. Data can be stored in HDFS, HBase, and any Hadoop data source. For the purpose of this thesis, we are using Spark on YARN and HDFS for distributed data storage.

- **Resilient Distributed Dataset**
  Resilient Distributed Dataset (RDD) is the primary data abstraction in Apache Spark and the core of Spark (that many often refer to as Spark Core). A RDD is a resilient and distributed collection of records. RDD is the bread and butter of Spark, and mastering the concept is of utmost importance.With RDD, the creators of Spark managed to hide data partitioning and so distribution that in turn allowed them to design parallel computational framework with a higher-level programming interface (API) for four mainstream programming languages - Java, Scala, Python and R. RDD, by its name, implies the following:

  - **Resilient**
    Fault-tolerant with the help of RDD lineage graph and so able to recompute missing or damaged partitions due to node failures.
  - **Distributed**
    As data resides on multiple nodes in a cluster.
  - **Dataset**
    Collection of partitioned data with primitive values or values of values, e.g. tuples or other objects (that represent records of the data you work with).

  There are two ways to create RDDs - parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system. The Spark RDD API introduces few Transformations and few Actions to manipulate RDD.

  - **Transformation**
    This creates new RDD from existing RDD like map, reduceByKey and filter. Transformation is executed on demand. That means they are computed lazily.

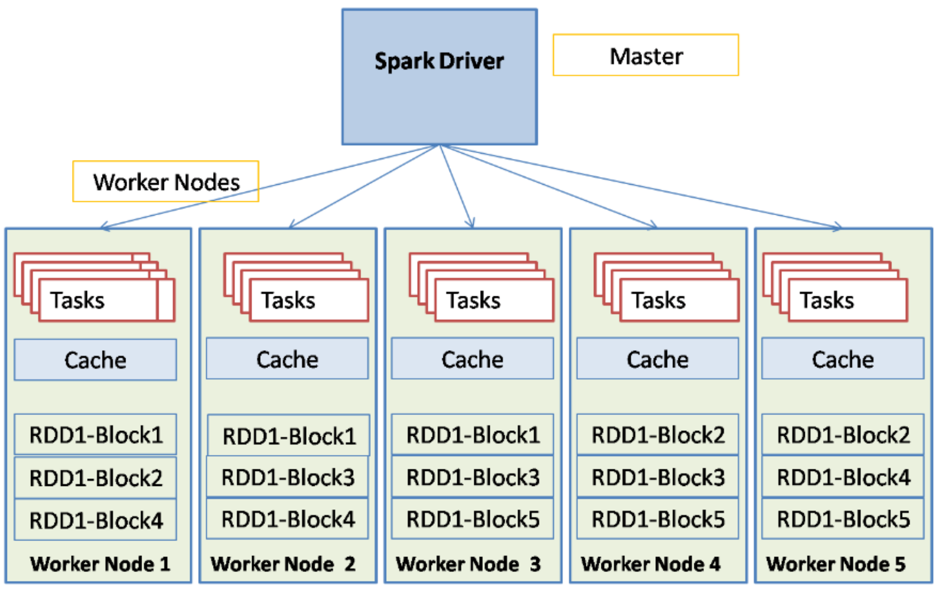
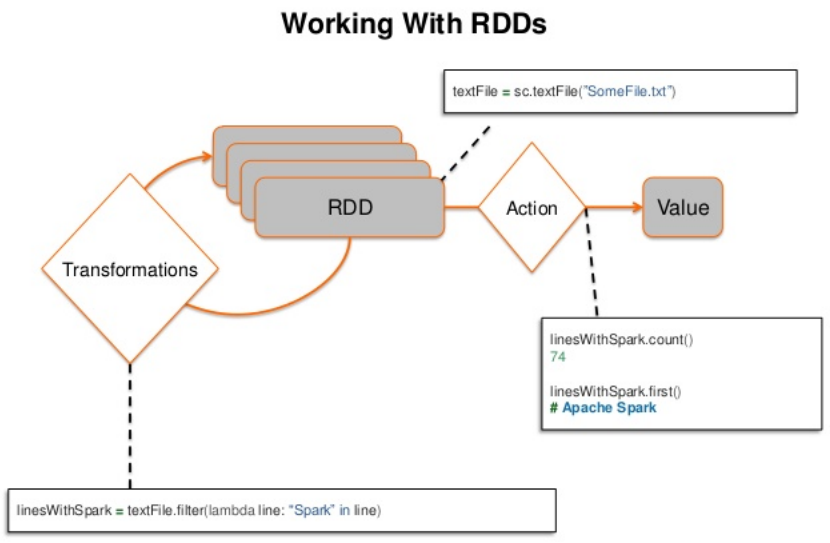


FIGURE 2.9: Apache Spark RDDs

- **Action**
  Actions return final results of RDD computations. Actions triggers execution using lineage graph to load the data into original RDD, carry out all intermediate transformations and return final results to Driver program or write it out to file system.



```
textFile = sc.textFile("SomeFile.txt")
```

```
linesWithSpark.count()
74

linesWithSpark.first()
# Apache Spark
```

```
linesWithSpark = textFile.filter(lambda line: "Spark" in line)
```

FIGURE 2.10: Apache Spark RDDs: Transformation and Action

In Spark RDD, data does not get loaded immediately, neither any of the transformation actually get instantly computed, till you call an action like collect or count or save output to file system. So, the data is not loaded until it is necessary saving memory blocking in advance. Spark uses lazy evaluation to reduce the number of passes it has to take over data by chaining operations together.

- **Interactive Operations on Spark RDD**
  Similarly, unlike as in Hadoop, if different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times. By default, each transformed RDD may be recomputed each time you run an action on it. However, you may also persist an RDD in memory, in which case Spark will keep the elements around on the cluster for much faster access, the next time you query it. There is also support for persisting RDDs on disk, or replicated across multiple nodes.

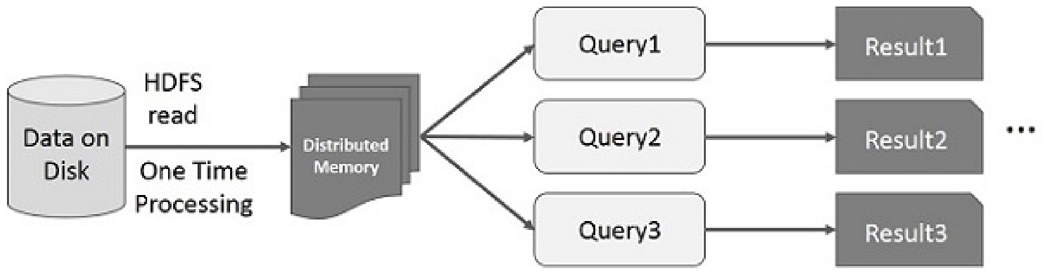


FIGURE 2.11: Interactive Operations on Spark

- **Iterative Operations on Spark RDD**
  Unlike MapReduce in Hadoop as discussed in later section, the iterative operations on Spark RDD will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

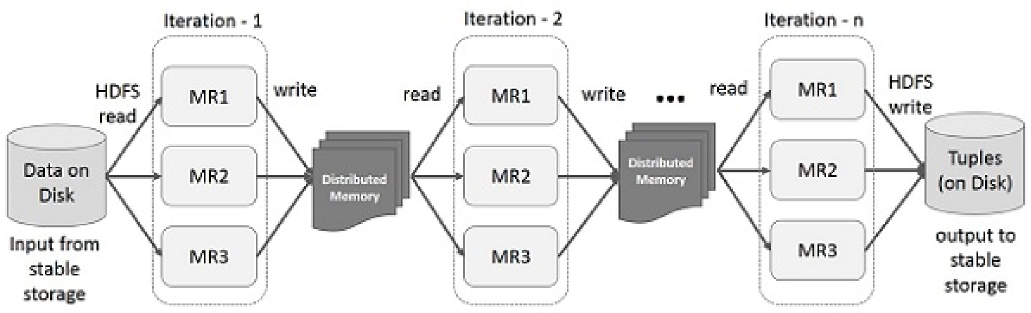


FIGURE 2.12: Iterative Operations on Spark

- **Spark vs Hadoop**
  Spark uses Hadoop in two ways- one is storage and second is processing. Since Spark has its own cluster management computation, it uses Hadoop for storage purpose only. Salient features are as follows:

  - **Faster**
    Execute batch processing jobs , about 10 to 100 times faster than the Hadoop MapReduce framework just by merely cutting down on the number of reads and writes to the disc. MapReduce does not leverage the memory of the Hadoop cluster to the maximum. In Spark the concept of RDDs (Resilient Distributed Datasets) lets you save data on memory and preserve it to the disc if and only if it is required and as well it does not have any kind of synchronization barriers that possibly could slow down the process. Thus the general execution engine of Spark is much faster than Hadoop MapReduce with the use of memory.
  - **Easy Management**
    With Hadoop, it is possible to perform Streaming, Batch Processing and Machine Learning all in the same cluster. With Spark, it is possible to control different kinds of workloads, so if there is an interaction between various workloads in the same process it is easier to manage and secure such workloads which come as a limitation with MapReduce.
  - **Spark Streaming- Real Time Method to Process Streams**
    In case of Hadoop MapReduce you just get to process a batch of stored data but with Hadoop Spark it is as well possible to modify the data in real time through Spark Streaming. Spark streaming is based on a paper Discretized Streams, which proposes a new model for doing windowed computations on streams using micro batches. Hadoop doesn't support this.
  - **Caching**
    Spark ensures lower latency computations by caching the partial results across its memory of distributed workers unlike MapReduce which is disk oriented completely.

- **Recovery**
  RDD is the main abstraction of spark. It allows recovery of failed nodes by re-computation of the DAG while also supporting a more similar recovery style to Hadoop by way of checkpointing, to reduce the dependencies of an RDD. Storing a spark job in a DAG allows for lazy computation of RDD's and can also allow spark's optimization engine to schedule the flow in ways that make a big difference in performance.
- **Spark API**
  Hadoop MapReduce has a very strict API that doesn't allow for as much versatility. Since spark abstracts away many of the low level details it allows for more productivity. Also things like broadcast variables and accumulators are much more versatile than DistributedCache and counters IMO.
- **Scheduler**
  As a product of in memory computation spark sort of acts as it's own flow scheduler. Whereas with standard MR you need an external job scheduler like Azkaban or Oozie to schedule complex flows.
- **Iterative computations**
  Spark has the upper hand as long as we're talking about iterative computations that need to pass over the same data many times. But when it comes to one-pass ETL-like jobs, for example, data transformation or data integration, then MapReduce is the deal - this is what it was designed for.

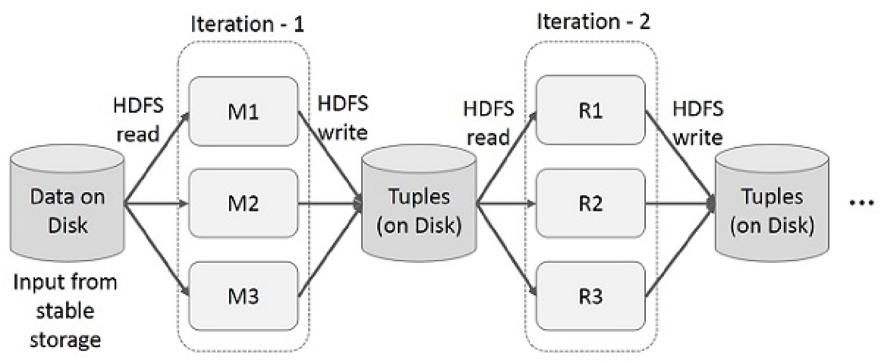


FIGURE 2.13: Working of MapReduce: Iterative

- **Cost**
  The memory in the Spark cluster should be at least as large as the amount of data you need to process, because the data has to fit into the memory for optimal performance. So, if you need to process really Big Data, Hadoop will definitely be the cheaper option since hard disk space comes at a much lower rate than memory space.
  On the other hand, considering Spark's benchmarks, it should be more cost-effective since less hardware can perform the same tasks much faster, especially on the cloud where compute power is paid per use.
- **Failure Tolerance**
  Spark has retries per task and speculative execution - just like MapReduce. If a process crashes in the middle of execution, it could continue where it left off, whereas Spark will have to start processing from the beginning.
- **Security**
  Spark security is still in its infancy; Hadoop MapReduce has more security features and projects like kerberos , sentry etc.

• **Broadcast and Accumulator**
Broadcast is part of Spark that is responsible for broadcasting information

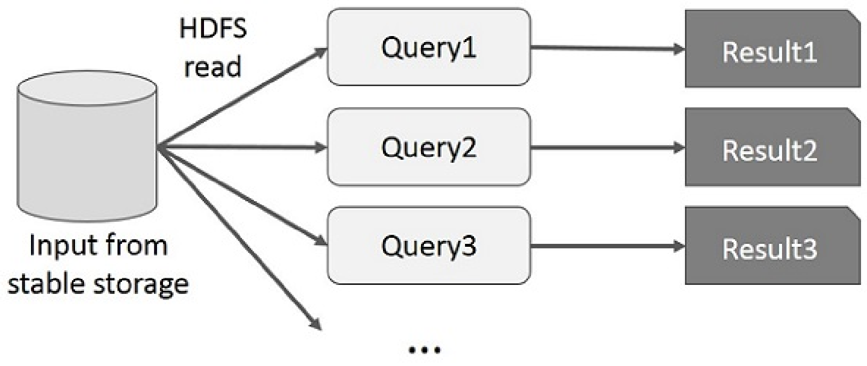


FIGURE 2.14: Working of MapReduce: Interactive

across nodes in a cluster. When you broadcast a value, it is copied to executors only once (while it is copied multiple times for tasks otherwise). It means that broadcast can help to get your Spark application faster if you have a large value to use in tasks or there are more tasks than executors. Broadcast variables allow the programmer to keep a read-only variable cached on each machine rather than shipping a copy of it with tasks. Explicitly creating broadcast variables is only useful when tasks across multiple stages need the same data or when caching the data in deserialized form is important.

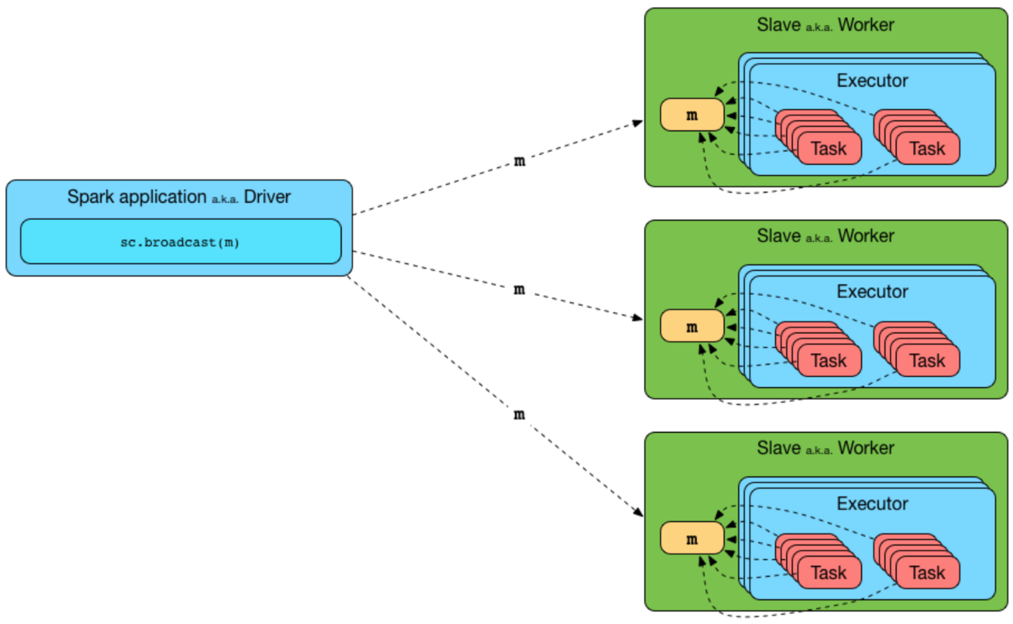


FIGURE 2.15: Broadcast working in master-worker Apache Spark

Accumulators are mutable variables that are only added through an associative and commutative operation and can therefore be efficiently supported in parallel. They can be used to implement counters (as in MapReduce) or sums. Spark natively supports accumulators of numeric types, and programmers can add support for new types. A numeric accumulator can be created by calling SparkContext methods to accumulate values of type Long or Double, as required. Tasks running on a cluster can then add to it using the add method. However, they cannot read its value. Only the driver (master) program can read the accumulator's value, using its value method.

2.7.2 Cloud Platform : Microsoft Azure

Microsoft Azure is an open, flexible and user friendly cloud computing platform and infrastructure created by Microsoft for building, deploying, and managing applications and services through a global network of Microsoft-managed data centers. It provides SaaS, PaaS and IaaS services and supports many different programming languages, tools and frameworks, including both Microsoft-specific and third-party software and systems. Azure HDInsight is Microsoft's managed Hadoop and other Big Data solutions. Spark for Azure HDInsight offers an enterprise-ready Spark solution that is fully managed, secured and highly available, made simpler for users.

Like Microsoft Azure, there are various other Cloud Platforms available by different IT giants such as IBM(Bluemix), Google(Cloud Dataproc), Amazon(Amazon Web Services), etc.

Chapter 3

Analysis and Design

In this chapter, we will go through all the algorithms designed and implemented. This includes the modified approach of feature selection for eliminating unnecessary attributes in datasets, followed by the three main algorithms that are compared. Towards the end of the chapter, we discuss about the scalable version of the best performing algorithm to handle big data efficiently.

3.1 Modified Feature Selection

Feature selection is a method of extracting important features that play significant role in classification of the data. In this method, the features that play important role to distinguish data into classes are being extracted so as to ease the further training of the model. Figure 3.1 shows where feature selection is incorporated in GP cycle.

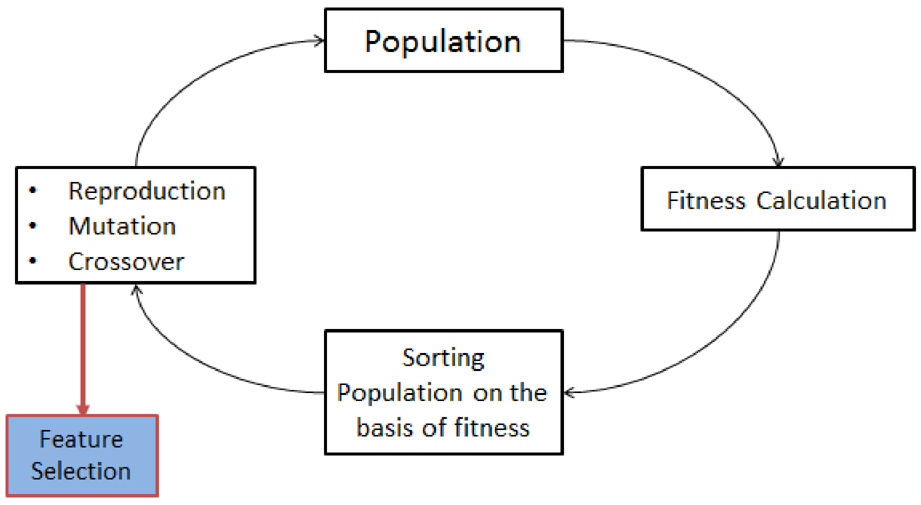


FIGURE 3.1: Incorporation of Feature Selection in GP Cycle

Following steps were followed for feature selection:

1. Get average fitness of all the fitness calculated, classwise.

$$average\_fitness[class\_j] = \sum_{i=1}^{n} \frac{fitness\_of\_tree[j][i])}{n} \tag{3.1}$$

where,
$'j'$ represents class $(1 \leq j \leq number\ of\ total\ classes\ = m)$
$'i'$ represents classifier number $(1 \leq i \leq number\ of\ classifiers\ in\ a\ generation\ = n)$.
Example : In a vehicle dataset, there are 4 classes, and 18 attribute features, and let there be 100 classifiers in each generation Each classifier will have 4 trees (for classifying the data among the four classes) therefore,

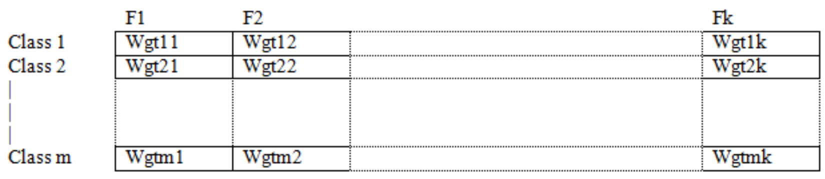
$$average\_fitness[1] = \sum_{i=1}^{100} \frac{fitness\_of\_tree[1][i])}{100}$$

$$average\_fitness[2] = \sum_{i=1}^{100} \frac{fitness\_of\_tree[2][i])}{100}$$

$$average\_fitness[3] = \sum_{i=1}^{100} \frac{fitness\_of\_tree[3][i])}{100}$$

$$average\_fitness[4] = \sum_{i=1}^{100} \frac{fitness\_of\_tree[4][i])}{100}$$

2. Create a table as shown and assign weights as *Wgt[j][k] = 0*, where *'k'* represents the feature/attribute number and ($1 \leq k \leq total\ number\ of\ features$).



	F1	F2		Fk
Class 1	Wgt11	Wgt12		Wgt1k
Class 2	Wgt21	Wgt22		Wgt2k
|				
|				
|				
Class m	Wgtm1	Wgtm2		Wgtmk

FIGURE 3.2: 2-D Matrix for weight assignment

For all those trees for which,

$$fitness\_of\_tree[j][i] \geq average\_fitness[j] \tag{3.2}$$

Assign weights as *Wgt[j][k] = Wgt[j][k] + 1*

3. Repeat till half of the generation.

4. Now, on reaching *(half + 1)th* generation, evaluate the average weight of the features for each class as,

$$average\_weight[j] = \sum_{k=1}^{100} \frac{wgt[j][k]}{m} \tag{3.3}$$

where,
'j' represents class ($1 \leq j \leq number\ of\ total\ classes$)
'k' represents the features number and ($1 \leq k \leq number\ of\ total\ features.$
For all those features, such that

$$wgt[j][k]\ \geq\ average\_weight[j] \tag{3.4}$$

Store them as good features for each class.

5. Now, from *(half + 1)th* generation onwards, for mutation operator, use features among these good features for the respective classes. This will help us use the wanted and important features for each respective class instead of total feature dataset.

3.2 Standard/Normal Feature Selection Algorithm

This section refers to one of the algorithm that was implemented and trained. It is incorporation of modified fitness selection as described in section 3.1 to general Genetic Programming algorithm. We will call it 'Normal Feature Selection Algorithm' in the rest of the thesis. Following are the steps:

1. Initial Population of randomly generated trees is created. This is considered as Generation 1. In this, total population of 100 classifiers are considered. Each classifier has a tree corresponding to every class respectively.

2. The classifiers are sorted according to cumulative fitness of each classifier in non increasing order.

3. First 0.25 classifiers of total population undergo reproduction operation.
4. Next 0.25 classifiers of total population experience mutation operation at random node of the trees in the respective classifier.
5. Last 0.5 classifiers of total population undergo crossover operation.
6. Apply modified feature selection as described in section 3.1 on this algorithm.
7. At the termination point, the first classifier of the sorted array is considered most efficient.

3.3 Hybrid Algorithm

This section refers to the second algorithm implemented for comparison used in [1]. It is the brief description of the hybrid algorithm used. Please refer [1] for details. The steps involved are as follows:

1. Initial Population of randomly generated trees is created. This is considered as Generation 1. In this, total population of 100 classifiers are considered. Each classifier has a tree corresponding to every class respectively.
2. The classifiers are sorted according to cumulative fitness of each classifier in non increasing order.
3. During application of genetic programming operators, first the given population is divided in the ratio of 2:8.
   Reproduction operator is applied to the first 2x population and hybrid crossover operator to the rest 8x.

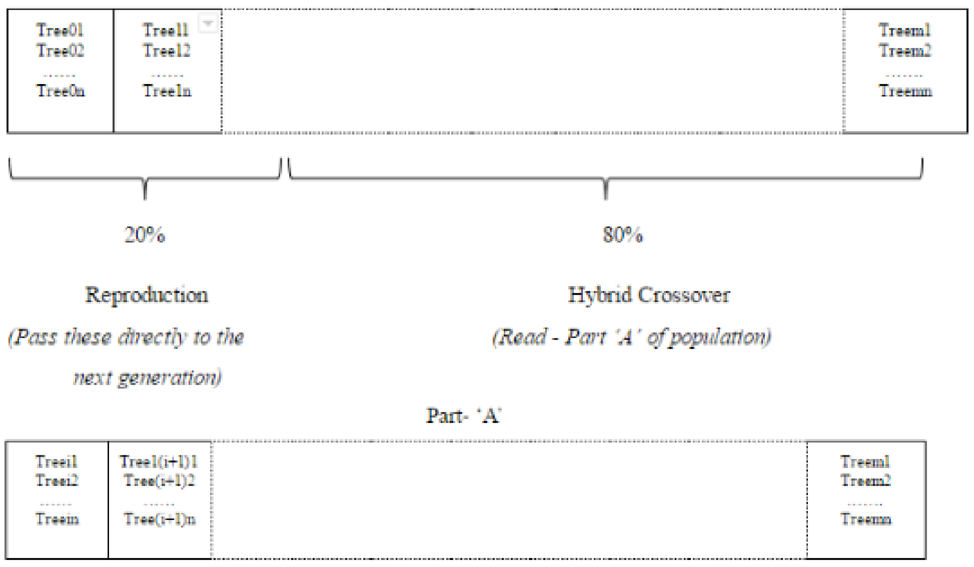


FIGURE 3.3: Structure of a particular generation

4. Out of these 80%, apply two point crossover, as described in section 2.2.3 to half of the population. To the other half, apply standard crossover, as described in section 2.2.3. Now, sort the parent classifier according to the cumulative fitness of the children.
5. For A1: Divide the children of this in two equal parts.
   Let us call them as A11 and A12.
6. For A11: Apply hill climbing, as in section 2.2.3 to the parents of children in this group.
   For A12: Apply standard crossover, as in section 2.2.3 to the parents of children in this group.

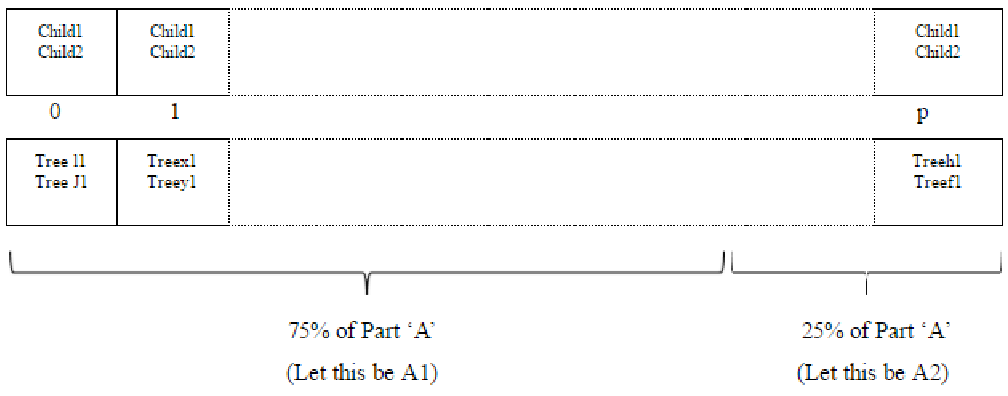


FIGURE 3.4: Distribution of Crossover and Mutation parts in a generation

7. For A2: Use the parents for mutation.
8. At the termination point, the first classifier of the sorted array is considered most efficient.

3.4 Hybrid Feature Selection Algorithm

This incorporates the benefits of both the hybrid version as well as the feature selection. Thus, producing a better classifier, theoretically. We implemented this theoretically better version as well to compare with the existing models of Genetic Programming for producing classifier. The steps of hybrid model of GP was followed as in section 3.3. The only change comes in the feature selection. The feature selection concept as discussed in 3.1 is followed as well till half of the generation. For the rest half generation, mutation is done based on the features present in the good feature set, making the unnecessary features out of question. Theoretical improvement in results are guaranteed due to feature selection extraction during the process of evolution. When implemented practically, we find the results that are outstandingly better, as shown in chapter 5.

3.5 Scalable Algorithm for big data

Due to expansion of data exponentially in today's time, it is next to impossible to run such serializable iterative algorithms as mentioned in section 3.2, 3.3 and 3.4. Genetic Programming, being iterative, is a slow process and system used might crash while being implemented for big data. Thus, making the algorithm scalable is of utmost importance. If we look at the algorithm carefully, we will be able to understand that the processes followed at each step and how distributed cluster setup can be put to use. Fitness calculation of each tree is most expensive operation in this algorithm as each data point is passed to each tree for evaluation of the fitness.

3.5.1 Steps Followed

It is not possible to load the complete data file in the memory of the system as the memory doesn't allow that much space, thus, making the algorithm useless. Hence, we use Apache Spark cluster to resolve such issues. This will also reduce the cost of machine learning and handling the big data appropriately. The steps are as follows:

1. The data file is taken as input in the form of RDDs by the master. As Spark is lazy, it is not directly loaded in the memory.

2. Initial population of 100 classifiers is generated by the master itself. This operation generates the trees randomly for each class in the classifiers and hence, it isn't a very costly operation. Thus, can be done in master node itself.

3. For calculation of fitness value for each tree in the classifiers, each data point must be evaluated by the expression tree. Thus, making it the most difficult task. So, we broadcast[12] (refer chapter 2 of this thesis) the trees of each classifier to worker nodes and pass RDD of data file to them. The implementation of passing RDDs has been done by 2 approaches described later.

4. The fitness of trees evaluated by workers are passed to the master and the master adds up these to evaluate the cumulative fitness of respective classifiers. The classifiers are then sorted in master node on the basis of the cumulative fitness.

5. The operations of reproduction, mutation and crossover, as described in hybrid feature selection algorithm in previous section, are followed thereafter in master node itself. For each of the operation, the calculation of fitness value for each tree is done by the worker nodes in the distributed manner. Hence, making these operations less expensive.

6. This is done till we reach termination stage. Thereafter, the first classifier from the sorted array of classifiers is taken to be the best.

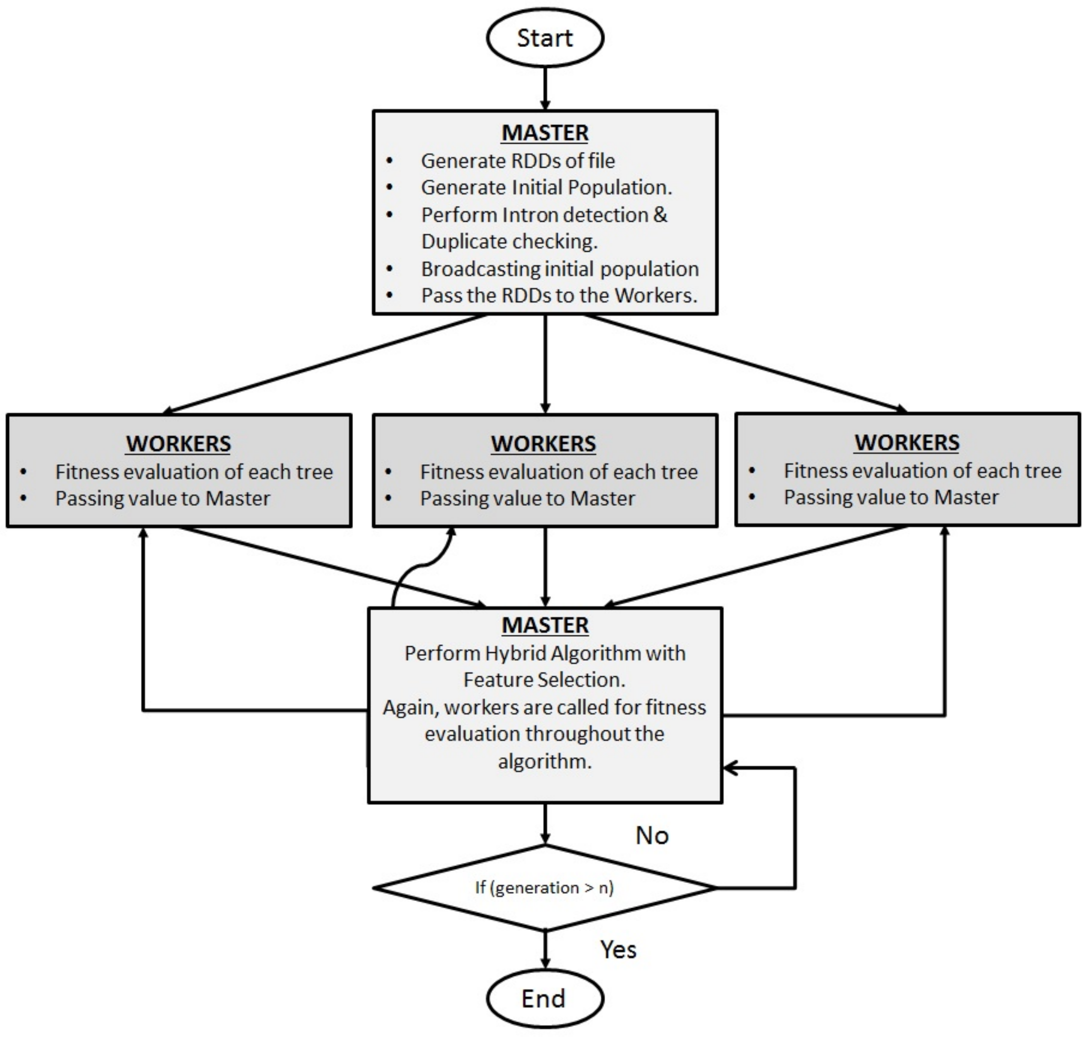

Start
MASTER
Generate RDDs of file
Generate Initial Population.
Perform Intron detection & Duplicate checking.
Broadcasting initial population
Pass the RDDs to the Workers.
WORKERS
Fitness evaluation of each tree
Passing value to Master
WORKERS
Fitness evaluation of each tree
Passing value to Master
WORKERS
Fitness evaluation of each tree
Passing value to Master
MASTER
Perform Hybrid Algorithm with Feature Selection.
Again, workers are called for fitness evaluation throughout the algorithm.
No
If (generation > n)
Yes
End

FIGURE 3.5: Flowchart of scalable algorithm of the hybrid feature selection algorithm

3.5.2 Approaches of passing RDDs to the Executors

Following are the two approaches implemented to pass the RDDs to the executors(worker nodes) for fitness value evaluation for each tree in a classifier:

- **1st Approach**
  The RDDs are passed as a single datapoint at a time, (line by line). This is achieved by using ′foreach()′ Spark function. The accumulator, as described in chapter 2, then increments itself as a part of fitness evaluation, and finally the value is passed back to the master for further calculations.

- **2nd Approach**
  The RDDs are passed as fragments to the worker nodes. This is achieved by using ′mapPartition()′ Spark function. The fitness values from all workers are added up by using ′reduce()′ Spark function in the master, and further calculations are performed.

3.5.3 Advantage

This algorithm helps in handling big data in many ways. Enlisted are a few:

- The distributive nature of this algorithm helps to calculate fitness values more efficiently and thus, reducing the cost.

- The RDDs passed help to read large datafiles in the form of partitions, rather than as a whole, which eases the handling of big data.

- The *cache()* function is used for RDDs. Spark supports pulling data sets into a cluster-wide in-memory cache. This is very useful when data is accessed repeatedly, such as when querying a small “hot” dataset or when running an iterative algorithm. Also, when RDD is used in multiple loops, it is best to keep it in cache. In the proposed algorithm, it is observed that the time decreases to more than half the time when RDDs are accessed without using cache(). Hence, keeping it in cache is a good option.

- The *Broadcast()* function is used to share immutable variables. Here, it is used to share the trees of all the classifier in a generation to the workers or the executors for evaluation of fitness. This helps the executors to access each tree independently for its evaluation through elements of RDD. The details about its working is described in chapter 2.

- The *Accumulator()* function is used to share mutable variables. Here, it is used to find the number of elements that are fit for the tree in the classifier. This keeps track of the results produced in all the workers or executors to calculate the collective result. The details about accumulators are described in chapter 2.

Chapter 4

Setup and Implementation

This chapter discusses about the two phases of the project namely- implementation on benchmark dataset and implementation on Big dataset. The first section talks about benchmark dataset whose results and comparative analysis are portrayed in next chapter. The second section is about the implementation configuration of the scalable algorithm on big data.

4.1 Benchmark dataset

The sequential implementation of the proposed algorithm as well as other algorithms- hybrid algorithm, standard/normal with feature selection and Hybrid with feature selection, as described in chapter 3, was done using following configurations:

- **Dataset** - UCI Repository : Parkinson Disease[2] as well as Diabetes Retinopathy[3]. Refer Appendix A for Dataset details.
- **Software specifications :**
  - **Language used** - Java
  - **IDE** - Eclipse
- **Code specifications :**
  - **Number of Generations** - 100
  - **Number of Total Population** - 100
  - **Maximum Depth of trees in a classifier** - 10
- **Hardware Specifications :**
  - **CPU specifications** - 8GB RAM

The results were compared and analysis based on training and testing accuracies and time taken for execution of the algorithms are discussed in chapter 5.

4.2 Big dataset

Cluster of master and worker nodes are used to put scalable algorithm as in chapter 3 to use. Broad implementation details are as follows:

- **Dataset** - UCI Repository for Parkinson Disease[2] and Diabetes Retinopathy[3] are replicated on the cluster itself for the execution of the algorithm to generate big data file. The replication is done 50 times, 100 times, 500 times and 1000 times. The generated data is again transferred to cluster storage. Refer Appendix A for Dataset details and Appendix D for replication details.
- **Language used** - Java
- **Cloud Platform** - Microsoft Azure
- **Cluster** - real cluster of master-worker concept was setup using HDInsight on Microsoft Azure. Refer Appendix B, C and D for further details.

- **Hardware details of machines in the cluster** -
  - Hardware virtual machine images running Ubuntu 14.04 LTS were used.
  - Spark cluster was established using HDP(2.4) distribution comprising of Spark 1.6.2 (HDI 3.4)[].
  - The cluster comprised of 2 worker nodes and 1 master node.
  - Specifications :
    - 4 Cores
    - 28GB RAM
    - 8 Disks
    - 200GB Local SSD
    - 0.35 times faster CPU
- **Code specifications :**
  - **Number of Generations** - 50
  - **Number of Total Population** - 100
  - **Maximum Depth of trees in a classifier** - 10
- **Storage** - Azure Blob Storage with HDFS interface.
- **Livy** - a REST interface for submitting jobs remotely to a Spark cluster

Chapter 5

Experimental Analysis and Results

In this chapter, we will analyse the results obtained from implementation of three serialized algorithms discussed in chapter 3, namely hybrid algorithm, standard / normal with feature selection algorithm and hybrid with feature selection algorithm. The testing was done for two types of dataset as written in chapter 4 and discussed in Appendix A. For both of the datasets, the result were calculated taking training to testing ratio as 7 : 3 and 8 : 2. Also, the both approaches of scalable algorithm were also implemented on the replicated dataset of Parkinson disease.

5.1 Hybrid Model for Genetic Programming Algorithm

- For Parkinson Disease dataset[2], the resultant tables, i.e., table 5.1 and table 5.2 are the following:

TABLE 5.1: Hybrid Model Result of Parkinson Disease Dataset[2]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	82.5	84	85	84.5
2	82	81	87	84
3	81	79	83	81
4	81	80	80	80
5	82.5	80	83	81.5

TABLE 5.2: Hybrid Model Result of Parkinson Disease Dataset[2]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	80	80	86	83
2	82	78	78	78
3	82	80	80	80
4	82.5	83	83	83
5	83.5	83	75	79

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 82.2 while for 8 : 2, we get 80.6. Also, the time taken is considerably high because of the iterative nature of GP.

- For Diabetes Retinopathy dataset[3], the resultant tables, i.e., table 5.3 and table 5.4 are the following:

TABLE 5.3: Hybrid Model Result of Diabetes Retinopathy Dataset[3]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	71.5	73	76	74.5
2	73.5	70	69	69.5
3	71.5	71	71	71
4	74	70	68	69
5	72.5	74	69	71.5

TABLE 5.4: Hybrid Model Result of Diabetes Retinopathy Dataset[3]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	73	72	69	70.5
2	72	70	70	70
3	71.5	70	70	70
4	72	75	72	73.5
5	71	70	71	70.5

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 71.1 while for 8 : 2, we get 70.9. Also, the time taken is considerably high because of the iterative nature of GP.

5.2 Standard/Normal Feature Selection Algorithm

- For Parkinson Disease dataset[2], the resultant tables are the following:

TABLE 5.5: Standard/Normal Feature Selection Model Result of Parkinson Disease Dataset[2]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	83	80	83	81.5
2	83	78	78	78
3	80	85	85	85
4	82.5	87	82	84.5
5	84	78	82	80

TABLE 5.6: Standard/Normal Feature Selection Model Result of Parkinson Disease Dataset[2]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	82	85	83	84
2	82.5	91	88	89
3	82	81	81	81
4	83	86	81	83.5
5	83.5	72	75	73.5

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 81.8 while for 8 : 2, we get 82.2. Also, the time taken is considerably high because of the iterative nature of GP.

- For Diabetes Retinopathy dataset[3], the resultant tables are the following:

TABLE 5.7: Standard/Normal Feature Selection Model Result of Diabetes Retinopathy Dataset[3]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	72.5	76	69	72.5
2	74	72	72	72
3	74	70	71	70.5
4	74	71	72	71.5
5	73	74	72	73

TABLE 5.8: Standard/Normal Feature Selection Model Result of Diabetes Retinopathy Dataset[3]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	72.5	72	78	75
2	73	70	70	70
3	72	70	72	71
4	73.5	74	71	72.5
5	71	71	72	71.5

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 71.9 while for 8 : 2, we get 72. Also, the time taken is considerably high because of the iterative nature of GP.

5.3 Hybrid with Feature Selection Algorithm

- For Parkinson Disease dataset[2], the resultant tables are the following:

TABLE 5.9: Hybrid with Feature Selection Model Result of Parkinson Disease Dataset[2]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	83	78	87	82.5
2	81	84	81	82.5
3	81.5	92	81	86.5
4	83.5	84	85	84.5
5	81	83	90	86.5

TABLE 5.10: Hybrid with Feature Selection Model Result of Parkinson Disease Dataset[2]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	82	81	83	82
2	82	85	83	84
3	82	81	89	85
4	80	88	81	84.5
5	79.5	89	88	89.5

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 84.5 while for 8 : 2, we get 85. Also, the time taken is considerably high because of the iterative nature of GP.

- For Diabetes Retinopathy dataset[3], the resultant tables are the following:

TABLE 5.11: Hybrid with Feature Selection Model Result of Diabetes Retinopathy Dataset[3]
7 : 3 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	71.5	71	74	72.5
2	72.5	71	70	70.5
3	71	75	72	73.5
4	73	74	74	74
5	73	75	73	74

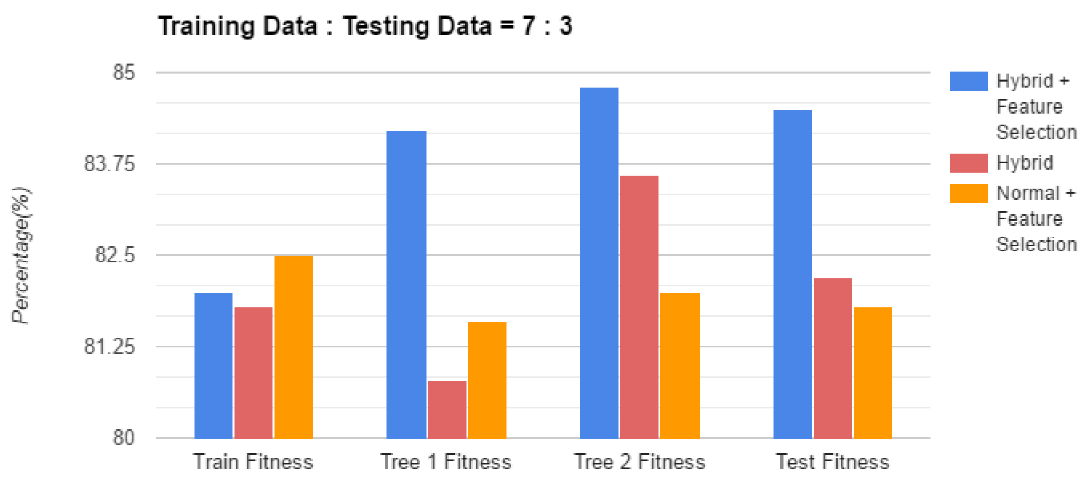
TABLE 5.12: Hybrid with Feature Selection Model Result of Diabetes Retinopathy Dataset[3]
8 : 2 - Train : Test

S.No.	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness
1	72	72	76	74
2	73	74	73	73.5
3	72	74	71	72.5
4	72	74	72	73
5	72.5	74	68	71.5

Taking the average, we get that for 7 : 3 type of data split-up give the test fitness = 72.9 while for 8 : 2, we get 72.8. Also, the time taken is considerably high because of the iterative nature of GP.

5.4 Comparative Study

For Parkinson Disease dataset[2], the resultant tables are the following:



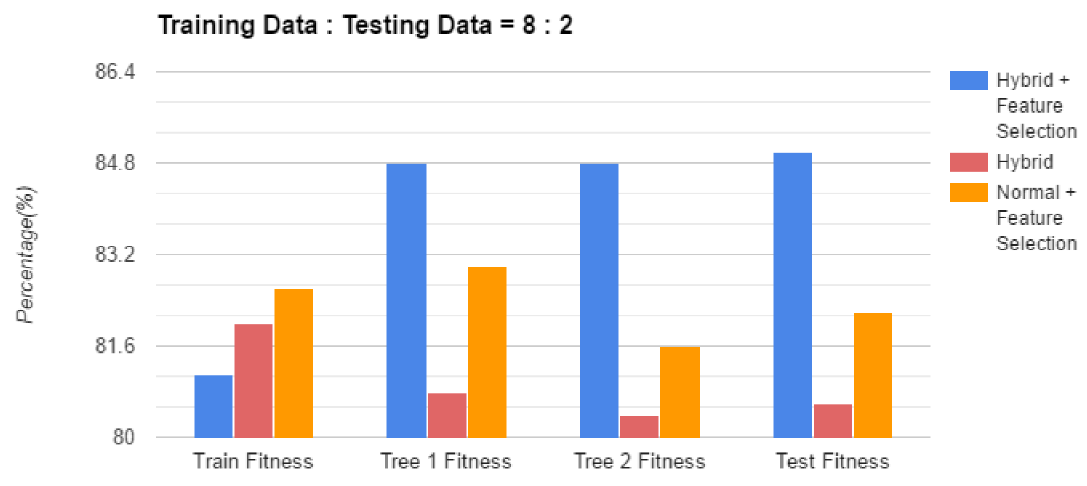


FIGURE 5.1: Statistics of Parkinson Disease

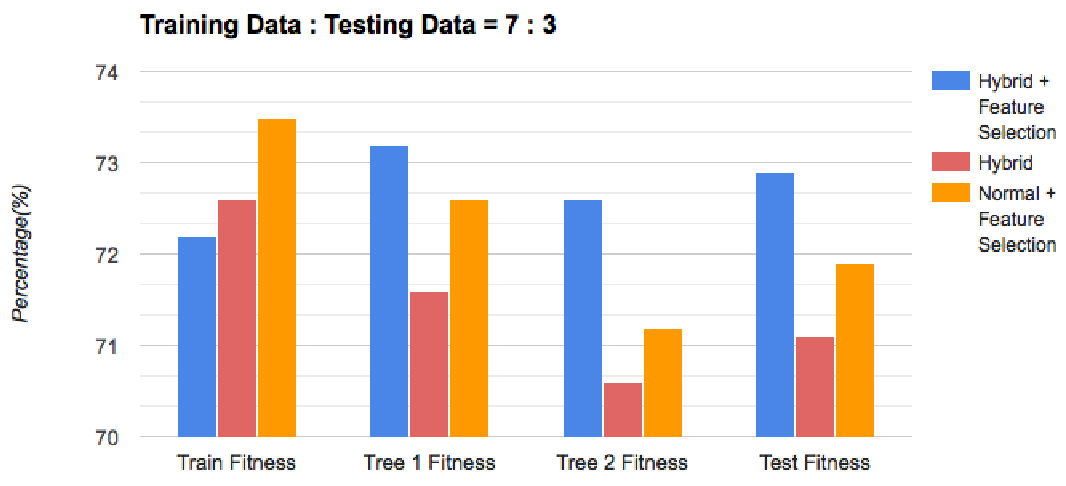
Clearly, the results of both types of train : test ratio show that accuracy is more in case of hybrid + feature selection algorithm as proposed in section 3.4 of this thesis.

While normal + feature selection algorithm also produced good results, best were exhibited by hybrid + feature selection algorithm.

- On comparison of 'Hybrid+Feature Selection' with 'Hybrid' model, ***the accuracy increased by 2.3% in case of 7 : 3 train : test ratio and 4.4% in case of 8 : 2***.
- On comparison of 'Hybrid+Feature Selection' with 'Normal+Feature Selection' model, ***the accuracy increased by 2.7% in case of 7 : 3 train : test ratio and 2.8% in case of 8 : 2***.

Thus, not only theoretically is the algorithm more efficient, even its implementation shows the same for Parkinson Dataset.

For Diabetes Retinopathy dataset[3], the resultant tables are the following:



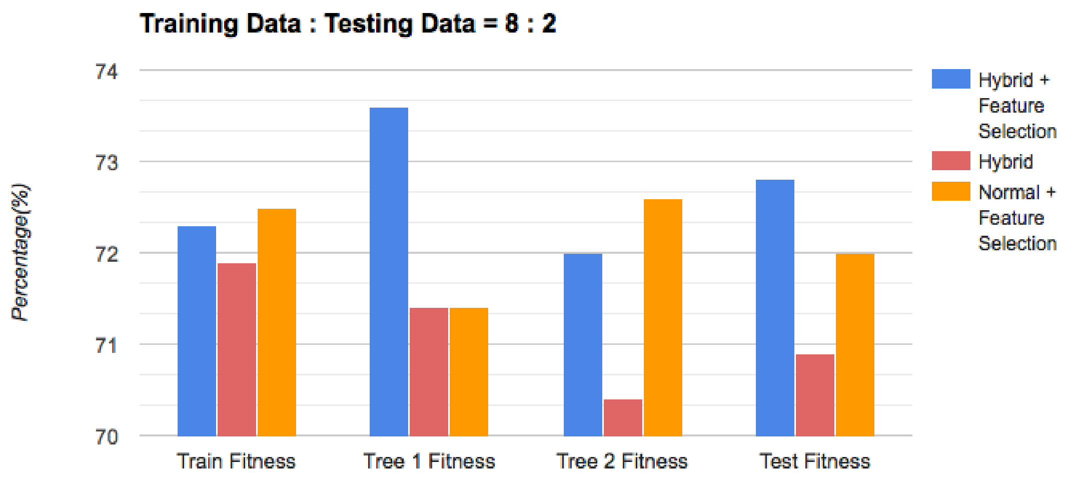


FIGURE 5.2: Statistics of Diabetes Retinopathy

Clearly, the results of both types of train:test ratio show that accuracy is more in case of hybrid + feature selection algorithm as proposed in section 3.4 of this thesis.

- On comparison of 'Hybrid+Feature Selection' with 'Hybrid' model, ***the accuracy increased by 1.8% in case of 7 : 3 train : test ratio and 1.9% in case of 8 : 2***.
- On comparison of 'Hybrid+Feature Selection' with 'Normal+Feature Selection' model, ***the accuracy increased by 1.0% in case of 7 : 3 train : test ratio and 0.8% in case of 8 : 2***.

Thus, not only theoretically is the algorithm more efficient, even its implementation shows the same for Diabetes Retinopathy Dataset.

5.5 Scalable Hybrid Feature Selection Algorithm

The two approaches implemented for scalable hybrid feature selection algorithm had comparable accuracy. But the time difference turned out to be significant. In 2nd approach, the time taken was reduced to half. This is due to the network latency as the 2nd approach passes fragments of data from the master to worker at a time whereas 1st approach passes each element(datapoint) of dataset at a time. This was done in distributed cluster using scalable algorithm because the file size became larger than the buffer size of the machine used for serialised algorithm.

Following is the table that shows the :

TABLE 5.13: 1st Approach vs 2nd Approach Time comparison

S.No.	Replication	Total instances	Time taken by 1st Approach	Time taken by 2nd Approach
1	50	9,750	04:27:03	02:44:46
2	100	19,500	07:38:38	04:22:51
3	500	97,500	13:46:11	09:03:09

Clearly, from the above table, we observed that the time taken was sufficiently large in case of 1st approach in comparison to 2nd approach though both are performing the same algorithm.

The results from 2nd approach are recorded in the following table:

TABLE 5.14: Scalable Hybrid Feature Selection Algorithm: 2nd Approach

S.No.	Replication	Toatal Instances	Train Fitness	Tree 1 Fitness	Tree 2 Fitness	Test Fitness	Time Taken
1	50	9,750	82.198	81.335	82.287	81.811	02:44:46
2	100	19,500	82.375	81.424	82.881	82.153	04:22:51
3	500	97,500	83.345	85.095	81.519	83.307	09:03:09

The above algorithm ran for 50 generations whereas the serialized versions were run for 100 generation. We can conclude from the above table that scalable algorithm can also produce similar results as serialized version and thus can be used to handle big data. Further optimization can be applied to the scalable version for reduction in time as described in the next chapter.

Chapter 6

Conclusion and Future Scope

In this chapter, we will discuss about what we conclude from our thesis, what more can be done and how one must take this research future to real time use for the betterment of the society. In simple words, the purpose of this project is to get the better classifier than the already existing. The serialized version using small data might generate a good classifier but at a slower rate. The time constraint is overcome here by using distributed concept of Apache Spark. Also, Big data analysis has been done so as to train the model well and hence get a good classifier that could classify the data with good accuracy.

As shown in results in chapter 5, it is clearly evident that feature selection in hybrid GP model has produced a little better results. Every bit percent of accuracy in medical domain is important and hence, this is a significant progress. For big data too, theoretically, we can expect better trees as the classifier will be trained with more amount of data.

As of now, the proposed scalable algorithm only incorporates the distributed nature of Apache Spark cluster. The points listed below can expand the scope of this research manifolds:

1. Parallelization over the cores in individual machines of the cluster can be done to further decrease the time taken by the algorithm to run.

2. In Apache Spark, we use the concept of RDDs. In recent times, Apache Spark has come up with Spark SQL which works on the concept of dataframes[**21**], which are built on top of RDD. This further speeds up the process of extracting information from the datafile.

3. This proposed algorithm can be implemented on Ignite framework, instead of Apache Spark framework. Ignite is an in-memory computing system, e.g. the one that treats RAM as the primary storage facility. Whereas others, including Spark, only use RAM for processing.

4. The proposed algorithm can be used for real time applications to generate a classifier at a faster rate by using the available big data, and thus, using that generated classifier to test the other real-time data. Thus, the classifier generated can be used in IoT devices and provide near real-time solutions.

Bibliography

[1] Arpit Bhardwaj, Aruna Tiwari, M. Vishaal Varma, M. Ramesh Krishna, *An Analysis of Integration of Hill Climbing in Crossover and Mutation operation for EEG Signal Classification*, GECCO'15, July 11-15, 2015

[2] For initial dataset, UCI Machine Learning Repository - Parkinson Disease *https://archive.ics.uci.edu/ml/datasets/Parkinsons*

[3] For initial dataset, UCI Machine Learning Repository - Diabetic Retinopathy *https://archive.ics.uci.edu/ml/datasets/Diabetic+Retinopathy+Debrecen+Data+Set*

[4] Little MA, McSharry PE, Roberts SJ, Costello DAE, Moroz IM, *Exploiting Nonlinear Recurrence and Fractal Scaling Properties for Voice Disorder Detection*, BioMedical Engineering OnLine 2007, June 26, 2007

[5] Balint Antal, Andras Hajdu, *An ensemble-based system for automatic screening of diabetic retinopathy*, Knowledge-Based Systems 60, April 2014

[6] V. K. Vavilapalli, A. C. Murthy, C. Douglas, S. Agarwal, M. Konar, R. Evans, T. Graves, J. Lowe, H. Shah, S. Seth et al., *Apache hadoop yarn: Yet another resource negotiator*, in Proceedings of the 4th annual Symposium on Cloud Computing. ACM, 2013

[7] B. Hindman, A. Konwinski, M. Zaharia, A. Ghodsi, A. D. Joseph, R. H. Katz, S. Shenker, and I. Stoica, *Mesos: A platform for fine-grained resource sharing in the data center*, in NSDI, vol. 11, 2011

[8] William B. Langdon, Adil Qureshi, *Computers using "Natural Selection" to generate programs*, Dept of Computer Science, University College London.

[9] N. Shamli, Dr. B. Sathiyabhama, *Parkinson's Brain Disease Prediction Using Big Data Analytics*, International Journal of Information Technology and Computer Science, 2016

[10] Arpit bhardwaj, Aruna Tiwari, Harshit Bhardwaj, Aditi Bhardwaj, *A genetically optimized neural network model for multi-class classification*, Elsevier 2016

[11] Kalyanmoy Deb, *Multi-Objective Optimization using Evolutionary Algorithms*, Wiley Publications 2013

[12] Broadcast variables, Mastering Apache Spark 2.0 - *https://jaceklaskowski.gitbooks.io/mastering-apache-spark/content/spark-broadcast.html*

[13] Microsoft Azure documentation for configuration - *https://azure.microsoft.com/en-us/documentation/articles/hdinsight-component-versioning/*

[14] UCI Repository of supersymmetric particles - *https://archive.ics.uci.edu/ml/datasets/SUSY*

[15] Markus Brameier, Wolfgang Banzhaf, *A Comparison of Linear Genetic Programming and Neural Networks in Medical Data Mining*, IEEE Transactions on Evolutionary Computation, Vol. 5, February 2001

[16] Ivan Kojadinovic, Thomas Wottka, *Comparison between a filter and a wrapper approach to variable subset selection in regression problems*, ESIT 2000, September 2000

[17] Rob Schapire, *Machine Learning Algorithms for Classification*, Princeton University

[18] *Big Data History and Current Considerations*, SAS Institute

[19] Munawar Hasan, *Genetic Algorithm and its application to Big Data Analysis*, January 2014

[20] J. Manyika, M. Chui, B. Brown, J. Bughin, R. Dobbs, C. Roxburgh, and A. H. Byers, *Big data: The next frontier for innovation, competition, and productivity*, pp. 1-137, 2011

[21] Jules Damji, *A Tale of Three Apache Spark APIs: RDDs, DataFrames, and Datasets - When to use them and why*, July 14, 2016

[22] Radovan Ondas, Martin Pelikan, Kumara Sastry, *Scalability of Genetic Programming and Probabilistic Incremental Program Evolution*, GECCO'05, June 25-29, 2005

[23] Nivranshu Hans, Sana Mahajan, SN Omkar, *Big Data Clustering Using Genetic Algorithm On Hadoop Mapreduce*, April 2015

Appendices

Appendix A

Datasets

A.1 Parkinsons Disease Data Set

TABLE A.1: Parkinson Dataset Description

Data Set Characteristics:	Multivariate
Number of Instances:	195
Area:	Life
Attribute Characteristics:	Real
Number of Attributes:	22
Number of Classes:	2

A.2 Diabetes Retinopathy Dataset

TABLE A.2: Diabetes Retinopathy Dataset

Data Set Characteristics:	Multivariate
Number of Instances:	1151
Area:	Life
Attribute Characteristics:	Integer, Real
Number of Attributes:	18
Number of Classes:	2

Appendix B

Installation of Azure CLI

Install the Azure Command-Line Interface (Azure CLI) to use a set of open-source shell-based commands for creating and managing resources in Microsoft Azure. There are several options to install these cross-platform tools on the computer:

- **npm package**
  Run npm (the package manager for JavaScript) to install the latest Azure CLI package on your Linux distribution or OS. Requires node.js and npm on your computer.
- **Installer**
  Download an installer for easy installation on Mac or Windows.
- **Docker container**
  Start using the latest CLI in a ready-to-run Docker container. Requires Docker host on your computer.

B.1 npm Package

To install the CLI from an npm package, make sure you have downloaded and installed the latest Node.js, curl and npm. Then, run npm install to install the azure-cli package:

```
sudo npm install -g azure-cli
```

B.2 Run Azure CLI

After the Azure CLI is installed, run the azure command from your command-line user interface (Bash, Terminal, Command prompt, and so on).

To see the version of the Azure CLI you installed, type the following:

```
azure --version
```

To update CLI, type the following:

```
npm update -g azure-cli
```

Appendix C

Cluster configuration and Initialisation for Microsoft Azure

C.1 Prerequisites

- An Azure subscription.
- A modern web browser. The Azure portal uses HTML5 and Javascript, and may not function correctly in older web browsers.

C.2 Create Cluster

The Azure portal exposes most of the cluster properties. Using Azure Resource Manager template, you can hide a lot of details.

- Sign in to the Azure portal.
- Click **NEW**, Click **Data Analytics**, and then click **HDInsight**.

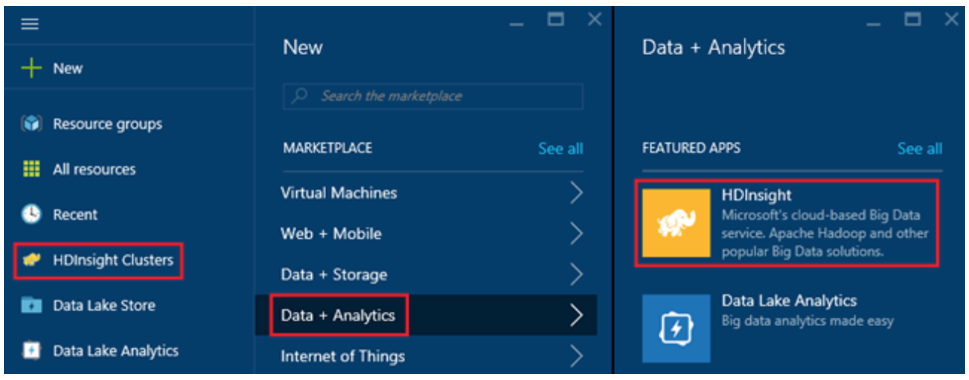


FIGURE C.1: Process of Cluster Creation

- Enter **Cluster Name**: This name must be globally unique.
- Click **Cluster configuration**, and then select:
  Cluster Type: Spark
  Operating System: Linux
  Version: Spark 1.6.2 (HDI 3.4)
  Cluster Tier: Standard
- Click **Credentials**, and enter **Cluster Login Username**, **Cluster Login Password, SSH Username** and **SSH Password**.
- Click **Data Source**. In the following menu, enter **storage account**, **container** and **Location**.
- Click **Pricing**, select the **Number of Worker nodes**, **Worker node size** and **Head node size**.
- Lastly, select **Resource Group** for the cluster.

The cluster usually takes about 20 minutes to be deployed along with all the changes applied appropriately.

Appendix D

Testing with Azure HDInsight Apache Spark Cluster

After installing the Azure Command-Line Interface (Azure CLI) you need to follow steps to to upload the test files, submit the jobs and get results.

D.1 Using Azure CLI with Azure Storage

To get started with the Azure, follow the below steps:

- Login to Azure

  ```
  $ azure login
  ```

- Connect to Azure storage account:

  ```
  $ export AZURE_STORAGE_ACCOUNT=<account-name>
  $ export AZURE_STORAGE_ACCESS_KEY=<key>
  ```

- Another way to set default storage account:

  ```
  $ azure storage account connectionstring show <account-name>
  ```

  copy the output connection string to the following:

  ```
  $ export AZURE_STORAGE_CONNECTION_STRING=<connectionstring>
  ```

D.2 Uploading files to storage blob

To upload blobs(jars, csv, txt, etc) in to a container, you can use the Azure Storage blob upload:

```
$ azure storage blob upload '~/job/Hello.jar' mycontainer myBlob
```

D.3 Replicating Data on master node

Replication of dataset to convert it into Big Data which can further be used to perform Big Data jobs.

- ssh to cluster

  ```
  $ ssh <username>@<clustername>-ssh.azurehdinsight.net
  ```

  This will give access to master node's terminal.

- Copy files from storage

  ```
  $ hadoop fs -copyToLocal ~/myfile.txt
  $ hadoop fs -copyToLocal ~/replicate.java
  ```

- run the replication code to create Big Data file on master node and then copy this file to the storage.

  ```
  $ hadoop fs -copyFromLocal bigdata.txt ~/bigdata.txt
  ```

D.4 Submitting Spark Jobs

Spark jobs are submitted through Livy Server that runs on the Master node. Curl commands are used for that.

- Submit Batch Job

  ```
  $ curl -k --user "<hdinsight user>:<user password>" -v -H
   <content-type> -X POST -d '{ "file":"<path to application jar>",
   "className":"<classname in jar>" }' 'https://<spark_cluster_name>.
   azurehdinsight.net/livy/batches'
  ```

- Getting information of running batches

  ```
  $ curl -k --user "<hdinsight user>:<user password>" -v -X GET
  "https://<spark_cluster_name>.azurehdinsight.net/livy/batches"
  ```

- Delete a batch job

  ```
  $ curl -k --user "<hdinsight user>:<user password>" -v -X DELETE
  "https://<spark_cluster_name>.azurehdinsight.net/livy/batches/{Id}"
  ```

D.5 Accessing logs

To access logs:
Goto->Azure Portal->'mycluster'->Cluster Dashboard->YARN
All application of the cluster will be enlisted.