Scalable Extreme Learning Machine for handling Big Data and its application to One Class Classification

A PROJECT REPORT

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

of
BACHELOR OF TECHNOLOGY
in

COMPUTER SCIENCE & ENGINEERING

Submitted by:
Varun Joglekar

Guided by:
Dr. Aruna Tiwari
Associate Professor

INDIAN INSTITUTE OF TECHNOLOGY INDORE
November 2017

Declaration

I hereby declare that except where specific reference is made to the work of others, the contents of this dissertation are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and Acknowledgements.

Submitted by: Varun Joglekar
November 2017

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
Associate Professor
Discipline of Computer Science & Engineering
Indian Institute of Technology Indore
(Project Guide)

Preface

This report on **"Scalable Extreme Learning Machine for handling Big Data and its application to One Class Classification"** is prepared under the supervision of Dr. Aruna Tiwari,Associate Professor,Computer Science & Engineering, IIT Indore.

Throughout this report, detailed description of the theoretical concepts used to develop and implement the Scalable Extreme Learning Machine Classifier is provided. The implemented scalable classifier is tested against four Big Datasets of varying sizes and the results are presented in a clear and concise manner. Further, detailed description of the cluster setup used to implement the classifier and instructions for implementing the same on Apache Spark framework are also provided in Appendix.

Acknowledgements

I would like to thank my B.Tech Project supervisor **Dr. Aruna Tiwari** for her constant support in structuring the project and her valuable feedback throughout the course of this project. She gave me an opportunity to discover and work in such an interesting domain. Her guidance proved really valuable in all the difficulties I faced in the course of this project.

I am really grateful to **Mr. Chandan Gautam** who also provided valuable guidance and helped with the problems while working with Big Data and Apache Spark framework. He provided the initial pathway for stating the project in right manner and provided useful directions to proceed along whenever necessary.

I am really thankful to my BTP partner **Shubham Goyal** for his contribution and support throughout the project.

I am also thankful to my family members,friends and colleagues who were a constant source of motivation. I am really grateful to Dept. of Computer Science & Engineering, IIT Indore for providing with the necessary hardware utilities to complete the project. I offer sincere thanks to everyone who else who knowingly or unknowingly helped me complete this project.

Varun Joglekar
140001037
Discipline of Computer Science & Engineering
Indian Institute of Technology Indore

Abstract

The One Class Classification (OCC) problem is different from the conventional binary/multi-class classification problem in the sense that in OCC, the negative class is either not present or not properly sampled. The problem of classifying positive (or target) cases in the absence of appropriately-characterized negative cases (or outliers) has gained increasing attention in recent years. 'Big Data' is gaining momentum in every field under the sun. Large volumes of data are created in short periods of time and the detection of anomalous behavior is a common problem when such huge data. OCC in Big Data has many applications like novelty/anomaly/outlier detection.

Through this project, we have tried to develop a Scalable ELM based One Class Classifier that can handle Big Data and identify outliers from inliers. Kernel based approaches have shown promising results when used for One Class Classification and Extreme Learning Machine being faster than traditional learning methods, Kernel based ELM becomes the choice for developing One Class Classifier for Big Data.

This report presents how the original training and testing algorithms of One Class ELM have been modified to make it scalable for handling Big Data. Apache Spark, which performs several times faster than conventional Big Data frameworks, has been used to implement the Scalable ELM classifier. Further included is the discussion on various Big Data sampling methods namely Simple Random Sampling, K-Means Cluster Sampling and Bisecting K-Means Cluster Sampling which have been used in order to remove redundant samples thereby reducing training time. In the end, the results obtained on testing the scalable classifier on various Big Datasets and the computational speedup obtained are presented.

Table of contents

List of figures

List of tables

Chapter 1

Introduction

This chapter highlights the background and motivation for the project. The problem statement of the project has been described and the importance of the results is also clearly portrayed. Towards the end of the chapter the objectives and expectation from this project are also outlined.

1.1 Background

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. 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.

The motivation behind this project is to develop a scalable classifier for One Class Classification in Big Data Environment. The One Class Classification (OCC) problem is different from the conventional binary/multi-class classification problem in the sense that in OCC, the negative class is either not present or not properly sampled. The problem of classifying positive (or target) cases in the absence of appropriately-characterized negative cases (or outliers) has gained increasing attention in recent years.

To highlight the importance of one-class classification, let us consider some scenarios. One-class classification can be relevant in detecting machine faults, for instance. A classifier

should detect when the machine is showing abnormal/faulty behaviour. Measurements on the normal operation of the machine (positive class training data) are easy to obtain. On the other hand, most faults will not have occurred so one will have little or no training data for the negative class.

As regards Big Data, one of its signature traits is that large volumes are created in short periods of time. This data often comes from connected devices, such as mobile phones, vehicle fleets or industrial machinery. The reasons for generating and observing this data are many, yet a common problem is the detection of anomalous behavior (application of One Class Classification). 'Big Data' is gaining momentum in every field under the sun. 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 Map-Reduce, Apache Spark etc. Among these, Apache Spark performs upto 100 times faster than conventional frameworks like Hadoop Map-Reduce. For the effective analysis and interpretation of this data, scalable machine learning methods are required to overcome the space and time bottlenecks. This forms the basis of our motivation for the project.

Earlier work done in OCC for Big Data include application of Support Vector Data Description(SVDD)[8], Kernel Principal Component Analysis(KPCA)[15] and One-Class SVM (OCSVM)[1]. Kernel based approaches have shown promising results when used for One Class Classification and Extreme Learning Machine being faster than traditional learning methods, Kernel based ELM becomes the choice for developing One Class Classifier for Big Data.

1.2 Objective

As per the above discussion, the objective thus is the Development of a Scalable ELM based One Class Classifier & its implementation on Apache Spark. The above objective has been divided into following goals:

- To design/enhance Extreme Learning Machine(ELM) to make it scalable for handling Big Data.
- To propose and develop Scalable algorithm for computing Radial Basis Function(RBF) Kernel and inverse of large kernel matrix in distributed manner.
- To design and implement algorithm for training the scalable ELM One Class Classifier on Apache Spark.
- To test the developed One Class Classifier on various Big Datasets.

Chapter 2

Literature Review

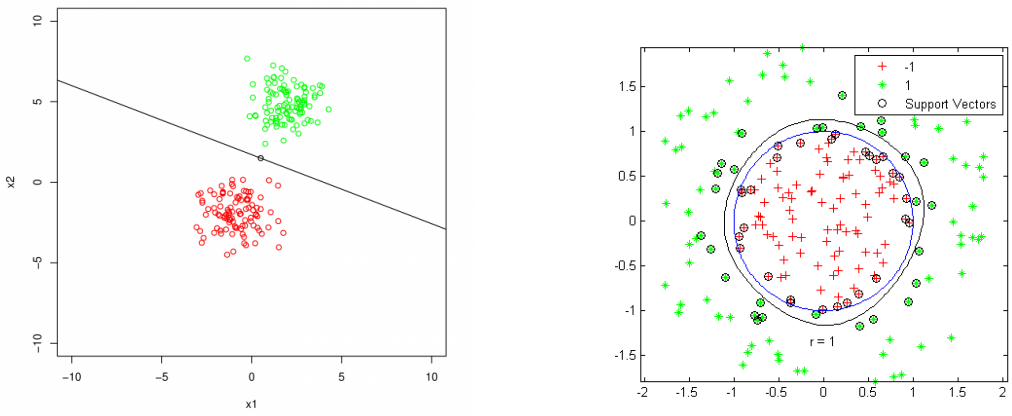
This chapter discusses the various concepts used during the course of this project. The chapter starts with discussion on One Class Classification, followed by explanation of Extreme Learning Machine, its structure & algorithm and description of various methods of sampling used during this project. The end of the chapter comprises of discussion on Apache Spark, its features and advantages.

2.1 One Class Classification

In machine learning, one-class classification, also known as unary classification, tries to identify objects of a specific class amongst all objects, by learning from a training set containing only the objects of that class. This is different from and more difficult than the traditional classification problem, which tries to distinguish between two or more classes with the training set containing objects from all the classes. An example is the classification of the operational status of a nuclear plant as 'normal'. In this scenario, there are few, if any, examples of catastrophic system states; only the statistics of normal operation are known. The term one-class classification was coined by Moya & Hush (1996) and many applications can be found in scientific literature, for example outlier detection, anomaly detection, novelty detection. A feature of one-class classification is that it uses only sample points from the assigned class, so that a representative sampling is not strictly required for non-target classes.

2.1.1 Fundamental difference between Binary and One Class Classification Algorithms

Binary classification algorithms are discriminatory in nature, since they learn to discriminate between classes using all data classes to create a hyperplane(as seen in the fig. below) and



(a) Binary Classification

(b) One Class Classification

use the hyperplane to label a new sample. In case of imbalance between the classes, the discriminatory methods can not be used to their full potential, since by their very nature, they rely on data from all classes to build the hyperplane that separate the various classes.

Where as the one-class algorithms are based on recognition since their aim is to recognize data from a particular class, and reject data from all other classes. This is accomplished by creating a boundary that encompasses all the data belonging to the target class within itself, so when a new sample arrives the algorithm only has to check whether it lies within the boundary or outside and accordingly classify the sample as belonging to the target class or the outlier.

2.1.2 Applications of One Class Classification

One Class Classification find wide use in the following areas:

- In mammograms for breast cancer detection
- The one-class recognition of cognitive brain functions
- In prediction of protein-protein interactions
- In the lung tissue categorization of patients affected with interstitial lung diseases

2.2 Extreme Learning Machine

Extreme learning machines are feedforward neural network for classification, regression, clustering, sparse approximation, compression and feature learning with a single layer or multi layers of hidden nodes, where the parameters of hidden nodes (not just the weights connecting inputs to hidden nodes) need not be tuned. These hidden nodes can be randomly assigned and never updated (i.e. they are random projection but with nonlinear transforms), or can be inherited from their ancestors without being changed. In most cases, the output weights of hidden nodes are usually learned in a single step, which essentially amounts to learning a linear model. The name "extreme learning machine" (ELM) was given to such models by its main inventor Guang-Bin Huang. these models are able to produce good generalization performance and learn thousands of times faster than networks trained using backpropagation [2]. In literature, it also shows that these models can outperform support vector machines (SVM) and SVM provides suboptimal solutions in both classification and regression applications[5].

2.2.1 Algorithm

Given a single hidden layer of ELM, suppose that the output function of the i-th node is $h_i(x) = G(a_i, B_i, x)$ where a_i and b_i are the parameters of the i-th hidden node [3]. The output function of the ELM for SLFNs with L hidden nodes is:

$f_L(x) = \sum_{i=1}^{L} \beta_i h_i(x)$, where β_i is the output weight of the i-th hidden node.
$h(x) = [G(h_i(x), ...h_L(x))]$ is the hidden layer output mapping of ELM. Given N training samples, the hidden layer output matrix H of ELM is given as:

$$H = \begin{bmatrix} h(x_1) \\ \vdots \\ h(x_N) \end{bmatrix} = \begin{bmatrix} G(a_1, b_1, x_1) \dots G(a_L, b_L, x_1) \\ \vdots \\ G(a_1, b_1, x_N) \dots G(a_L, b_L, x_N) \end{bmatrix}$$

and T is the training data matrix

$$T = \begin{bmatrix} t_1 \\ \vdots \\ t_N \end{bmatrix}$$

Generally speaking, ELM is a kind of regularization neural networks but with non-tuned hidden layer mappings (formed by either random hidden nodes, kernels or other implementations), its objective function is:

Minimize: $||\beta|| + C||H\beta - T||$

2.2.2 Architectures

In most cases, ELM is used as a single hidden layer feedforward network (SLFN) including but not limited to sigmoid networks, RBF networks, threshold networks, fuzzy inference networks, complex neural networks, wavelet networks, Fourier transform, Laplacian transform, etc. Due to its different learning algorithm implementations for regression, classification, spare coding, compression, feature learning and clustering, multi ELMs have been used to form multi hidden layer networks, deep learning or hierarchical networks.

A hidden node in ELM is a computational elements, which need not be considered as classical neuron. A hidden node in ELM can be classical artificial neurons, basis functions, or a subnetwork formed by some hidden nodes.

2.3 Kernel Methods

In machine learning, kernel methods are a class of algorithms for pattern analysis, whose best known member is the support vector machine (SVM) [4]. The general task of pattern analysis is to find and study general types of relations (for example clusters, rankings, principal components, correlations, classifications) in datasets. For many algorithms that solve these tasks, the data in raw representation have to be explicitly transformed into feature vector representations via a user-specified feature map: in contrast, kernel methods require only a user-specified kernel, i.e., a similarity function over pairs of data points in raw representation.

Kernel methods owe their name to the use of kernel functions, which enable them to operate in a high-dimensional, implicit feature space without ever computing the coordinates of the data in that space, but rather by simply computing the inner products between the images of all pairs of data in the feature space. This operation is often computationally cheaper than the explicit computation of the coordinates. This approach is called the "kernel trick". Kernel functions have been introduced for sequence data, graphs, text, images, as well as vectors.

Why Kernels?

Why kernels, as opposed to feature vectors? One big reason is that in many cases, computing the kernel is easy, but computing the feature vector corresponding to the kernel is really really hard. The feature vector for even simple kernels can blow up in size, and for kernels like the RBF kernel $k(x,y) = exp(-||x-y||^2)$ the corresponding feature vector is infinite dimensional. Yet, computing the kernel is almost trivial.

Many machine learning algorithms can be written to only use dot products, and then we can replace the dot products with kernels. By doing so, we don't have to use the feature vector at all. This means that we can work with highly complex, efficient-to-compute, and yet high performing kernels without ever having to write down the huge and potentially infinite dimensional feature vector. Thus if not for the ability to use kernel functions directly, we would be stuck with relatively low dimensional, low-performance feature vectors.

2.3.1 Radial Basis Function Kernel

For the purpose of this thesis, RBF Kernel is being used. In machine learning, the (Gaussian) radial basis function kernel, or RBF kernel, is a popular kernel function used in various kernelized learning algorithms. In particular, it is commonly used in support vector machine classification.

The RBF kernel on two samples x and x', represented as feature vectors in some input space, is defined as [14]:

$$K(x,x') = exp(-\frac{||x-x'||^2}{2\sigma^2})$$

$||x-x'||^2$ may be recognized as the squared Euclidean distance between the two feature vectors. σ is a free parameter. An equivalent, but simpler, definition involves a parameter $\gamma = \frac{1}{2\sigma^2}$

$$K(x,x') = exp(-\gamma||x-x'||^2)$$

Since the value of the RBF kernel decreases with distance and ranges between zero (in the limit) and one (when x = x'), it has a ready interpretation as a similarity measure.

2.4 ELM Based One Class Classifier

When data only from the target class are available, the one-class classifier is trained to accept target objects and reject objects that deviate significantly from the target class. In the training phase, the one-class classifier, which defines a distance function d between the objects and the target class, takes in the training set X to build the classification model. In general, the

classification model contains two important parameters to be determined:threshold θ and modal parameter λ . A generic test sample is accepted by the classifier if $d(z|X,\lambda) < \theta$

In the training phase, not all the training samples are to be accepted by the one-class classifier due to the presence of outliers or noisy data contained in the training set. Otherwise, the trained classification model may generalize poor to unknown test set when the training set includes abnormal data samples. Usually, threshold θ is determined such that a user-specified fraction of training samples most deviant from the target class are rejected. For instance, if one is told five percent of training samples are mislabeled, setting $\mu = 0.05$ makes the classifier more robust. Even when all the samples are correctly labeled, rejecting a small fraction of training samples helps the classifier to learn the most representative model from the training samples.

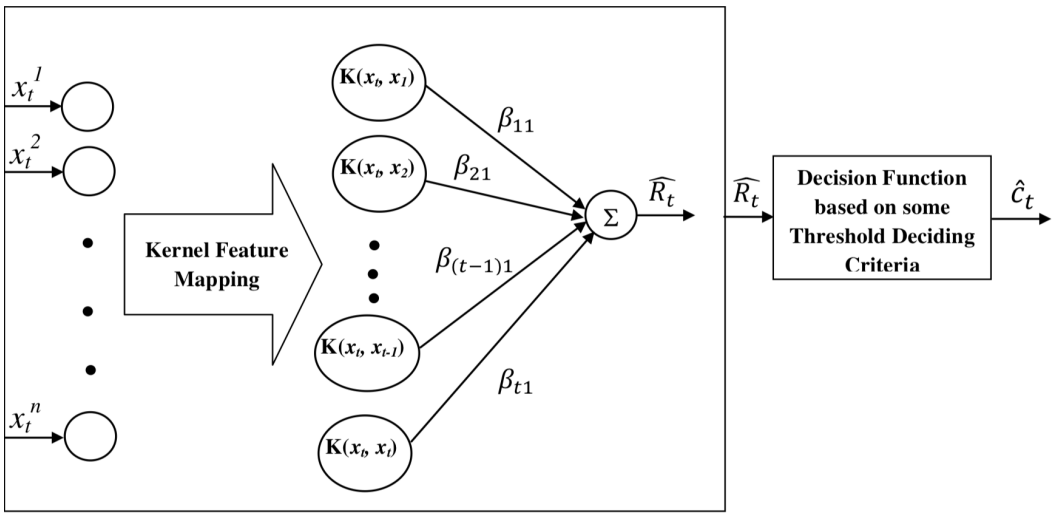


Fig. 2.2 ELM Based One Class Classifier

2.5 Sampling Big Data

Data sampling is a statistical analysis technique used to select, manipulate and analyze a representative subset of data points in order to identify patterns and trends in the larger data set being examined.

Sampling allows data scientists, predictive modelers and other data analysts to work with a small, manageable amount of data in order to build and run analytical models more quickly, while still producing accurate findings. Sampling can be particularly useful with data sets that are too large to efficiently analyze in full – for example, in big data analytics applications.

An important consideration, though, is the size of the required data sample. In some cases, a very small sample can tell all of the most important information about a data set. In others, using a larger sample can increase the likelihood of accurately representing the data as a whole, even though the increased size of the sample may impede ease of manipulation and interpretation. Either way, samples are best drawn from data sets that are as large and close to complete as possible.

Sampling Methods can be classified into one of two categories:

- **Probability Sampling**: Sample has a known probability of being selected
- **Non-probability Sampling**: Sample does not have known probability of being selected as in convenience or voluntary response surveys

Probability Sampling

In probability sampling it is possible to both determine which sampling units belong to which sample and the probability that each sample will be selected. The following sampling methods are examples of probability sampling:

- Simple Random Sampling (SRS)
- Stratified Sampling
- Cluster Sampling
- Systematic Sampling
- Multistage Sampling (in which some of the methods above are combined in stages)

For the purpose of this thesis, Simple Random Sampling and Cluster Sampling methods have been employed. Brief description of how both have been used is below.

Simple Random Sampling

A simple random sample is a subset of a statistical population in which each member of the subset has an equal probability of being chosen.Ease of use represents the biggest advantage of simple random sampling. Unlike more complicated sampling methods such as stratified random sampling and cluster sampling, no need exists to divide the population into subpopulations or take any other additional steps before selecting members of the population at random. A simple random sample is meant to be an unbiased representation of a group. It is considered a fair way to select a sample from a larger population, since every member of the population has an equal chance of getting selected.

Cluster Sampling

In this, the input samples are divided into groups called clusters and one representative from each group is taken as input sample.The clusters are mutually exclusive and collectively exhaustive.For the purpose of this thesis, K-Means & Bisecting K-Means clustering techniques are used to obtain cluster centers which are treated as input samples.

K Means Clustering

k-means clustering is a method of vector quantization [10], originally from signal processing, that is popular for cluster analysis in data mining. k-means clustering aims to partition n observations into k clusters in which each observation belongs to the cluster with the nearest mean, serving as a prototype of the cluster. This results in a partitioning of the data space into Voronoi cells.

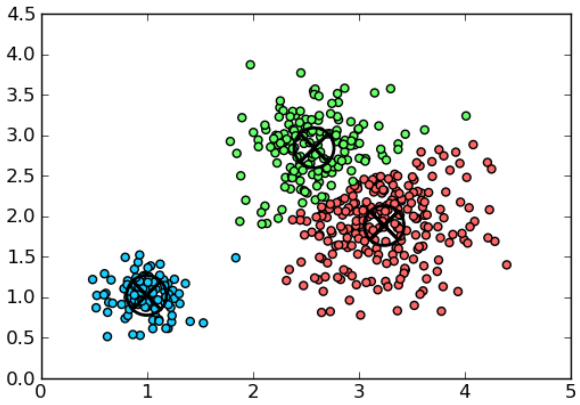


Fig. 2.3 K-Means Clustering

Bisecting K-Means Clustering

Bisecting K-means can often be much faster than regular K-means, but it will generally produce a different clustering[9].Bisecting k-means is a kind of hierarchical clustering. Hierarchical clustering is one of the most commonly used method of cluster analysis which seeks to build a hierarchy of clusters.

As Bisecting k-means is based on k-means, it keeps the merits of k-means and also has some advantages over k-means. First, bisecting k-means is more efficient when ‘k’ is large. For the k-means algorithm, the computation involves every data point of the data

set and k centroids. On the other hand, in each Bisecting step of Bisecting k-means, only the data points of one cluster and two centroids are involved in the computation. Thus, the computation time is reduced. Secondly, Bisecting k-means produce clusters of similar sizes, while k-means is known to produce clusters of widely different sizes

2.6 Big Data Handling Framework - 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 provides an interface for programming entire clusters with implicit data parallelism and fault tolerance. Apache Spark is a lightning-fast cluster computing technology, designed for fast computation. It is based on Hadoop MapReduce and it extends the MapReduce model to efficiently use it for more types of computations, which includes interactive queries and stream processing [7]. The main feature of Spark is its in-memory cluster computing that increases the processing speed of an application.

2.6.1 Features of Apache Spark

Apache Spark has following features.

- **Speed** - Spark helps to run an application in Hadoop cluster, up to 100 times faster in memory, and 10 times faster when running on disk. This is possible by reducing number of read/write operations to disk. It stores the intermediate processing data in memory.
- **Supports multiple languages** - Spark provides built-in APIs in Java, Scala, or Python. Therefore, you can write applications in different languages. Spark comes up with 80 high-level operators for interactive querying.
- **Advanced Analytics** - Spark not only supports 'Map' and 'reduce'.It also supports SQL queries, Streaming data, Machine learning (ML), and Graph algorithms.

2.6.2 Apache Spark Architecture

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 these libraries. Spark's flexibility and

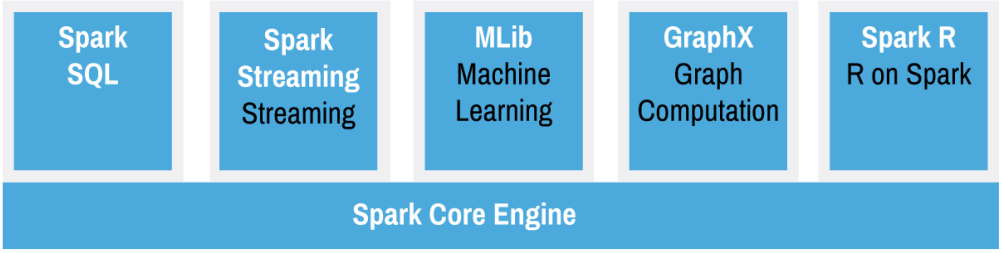


Fig. 2.4 Spark Architecture

power become most apparent in applications that require the combination of two or more of these libraries on top of Spark Core.These libraries are :

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

Spark Built on Hadoop

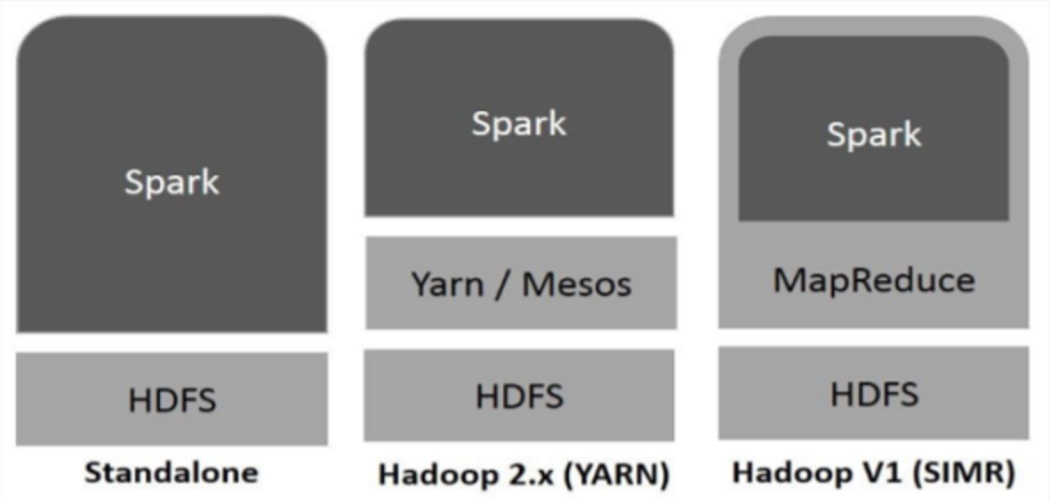


Fig. 2.5 Spark Deployment Methods

There are three ways of Spark deployment as explained below.

- **Standalone** - Spark Standalone deployment means Spark occupies the place on top of HDFS(Hadoop Distributed File System) and space is allocated for HDFS, explicitly. Here, Spark and MapReduce will run side by side to cover all spark jobs on cluster.
- **Hadoop Yarn** - Hadoop Yarn deployment means, simply, spark runs on Yarn without any pre-installation or root access required. It helps to integrate Spark into Hadoop ecosystem or Hadoop stack. It allows other components to run on top of stack.
- **Spark in MapReduce (SIMR)** - Spark in MapReduce is used to launch spark job in addition to standalone deployment. With SIMR, user can start Spark and uses its shell without any administrative access.

2.6.3 Resilient Distributed Dataset (RDD)

The Resilient Distributed Dataset is a concept at the heart of Spark. It is designed to support in-memory data storage, distributed across a cluster in a manner that is demonstrably both fault-tolerant and efficient. Fault-tolerance is achieved, in part, by tracking the lineage of transformations applied to coarse-grained sets of data. Efficiency is achieved through parallelization of processing across multiple nodes in the cluster, and minimization of data replication between those nodes. Once data is loaded into an RDD, two basic types of operation can be carried out:

- **Transformations** which create a new RDD by changing the original through processes such as mapping, filtering, and more;
- **Actions** such as counts, which measure but do not change the original data. The original RDD remains unchanged throughout. The chain of transformations from RDD1 to RDDs are logged, and can be repeated in the event of data loss or the failure of a cluster node.

Transformations are said to be lazily evaluated, meaning that they are not executed until a subsequent action has a need for the result. This will normally improve performance, as it can avoid the need to process data unnecessarily. It can also, in certain circumstances, introduce processing bottlenecks that cause applications to stall while waiting for a processing action to conclude.

Where possible, these RDDs remain in memory, greatly increasing the performance of the cluster, particularly in use cases with a requirement for iterative queries or processes.

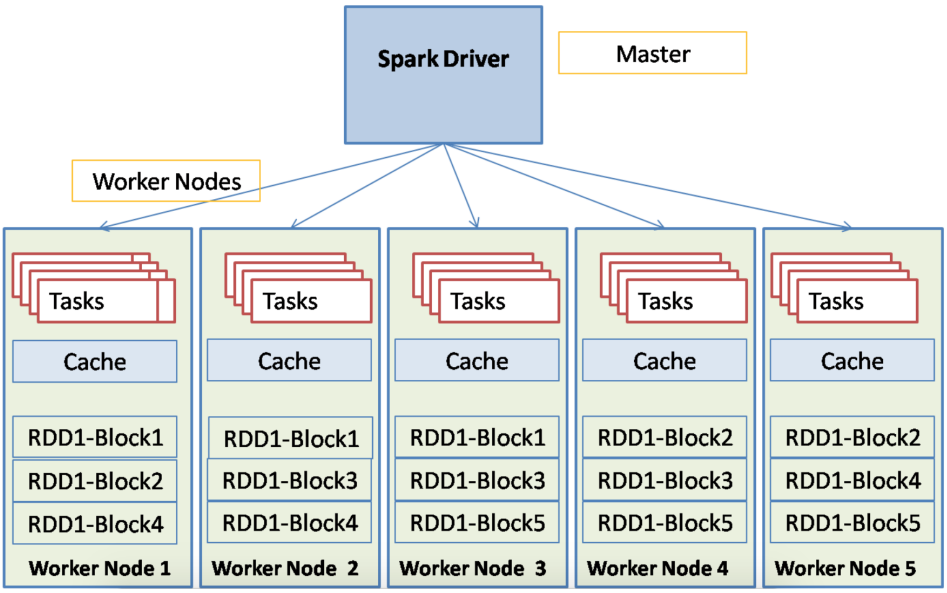


Fig. 2.6 Apache Spark RDDs

There are two ways to create RDDs - parallelizing an existing collection in your driver program, or referencing a dataset in an external storage system, such as a shared file system, HDFS, HBase, or any data source offering a Hadoop Input Format.

Spark makes use of the concept of RDD to achieve faster and efficient MapReduce operations

Data Sharing using Spark RDD

Data sharing is slow in MapReduce due to replication, serialization, and disk IO. Most of the Hadoop applications, they spend more than 90% of the time doing HDFS read-write operations.

Recognizing this problem, researchers developed a specialized framework called Apache Spark. The key idea of spark is Resilient Distributed Datasets (RDD); it supports in-memory processing computation. This means, it stores the state of memory as an object across the jobs and the object is sharable between those jobs. Data sharing in memory is 10 to 100 times faster than network and Disk.

The section below discusses how the iterative and interactive operations take place in Spark RDD.

Iterative Operations on Spark RDD

The illustration given below shows the iterative operations on Spark RDD. It will store intermediate results in a distributed memory instead of Stable storage (Disk) and make the system faster.

If the Distributed memory (RAM) is not sufficient to store intermediate results (State of the JOB), then it will store those results on the disk.

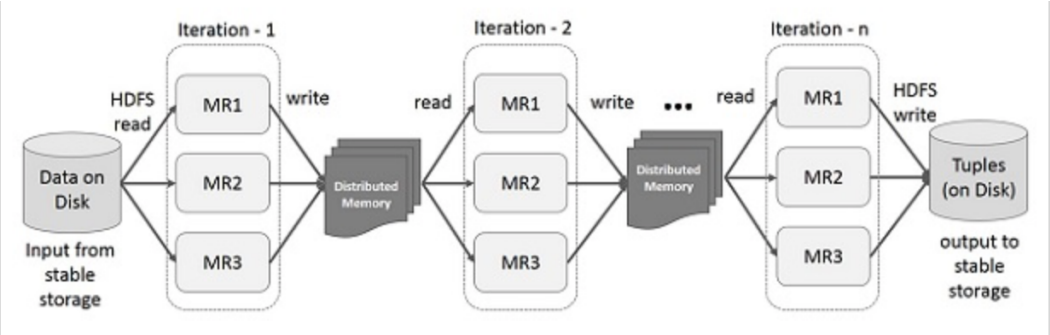


Fig. 2.7 Iterative Operations on Spark RDD

Interactive Operations on Spark RDD

This illustration shows interactive operations on Spark RDD. If different queries are run on the same set of data repeatedly, this particular data can be kept in memory for better execution times.

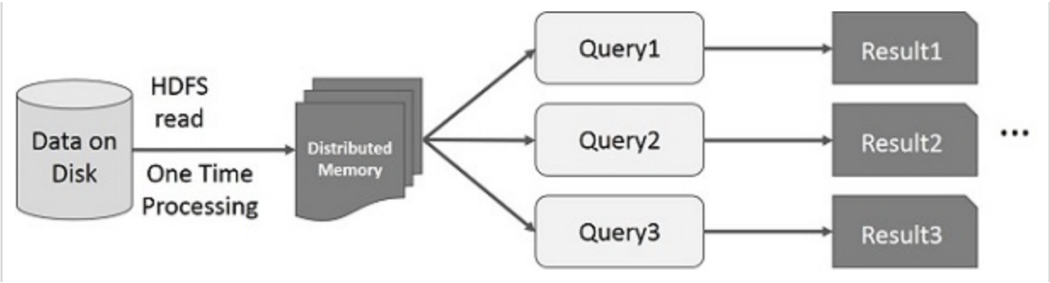


Fig. 2.8 Interactive Operations on Spark RDD

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.

Chapter 3

Design of Scalable ELM for One Class Classification

In this chapter, we will go through design and implementation of the Scalable ELM classifier. This includes discussion on modifications to kernel & inverse computation algorithms and hence the overall modified ELM training and testing algorithm. This chapter also discusses the implementation of the sampling methods on Apache Spark

3.1 Design of Scalable RBF Kernel Algorithm

The first step in training the hidden layer in ELM is kernel mapping from input layer to the hidden layer. As per discussion in previous chapter, RBF Kernel is used to map input layer to hidden layer. The RBF Kernel matrix K for a set of N samples is a $N \times N$ symmetric matrix where the matrix entry K_{ij} represents the similarity measure between the i^{th} and j^{th} sample. The algorithm to compute the kernel matrix is discussed below:-

Input :

Samples[] $\rightarrow$ List of Samples

RDD[Samples[]] $\rightarrow$ RDD of Samples

Output :

KernelMatrix $\rightarrow$ Distributed Kernel Matrix

function : ComputeKernelMatrix(Samples[],RDD[Samples])

```
rowsRDD = emptyRDD()
foreach X in Samples[] do:
    row = ComputeKernelMatrixRow(X,RDD[Samples[]])
    rowsRDD.union(row)
```

KernelMatrix = RddToDistributedMatrix(rowsRDD)
return KernelMatrix
end function

The function *ComputeKernelMatrixRow()* is as follows:
function : ComputeKernelMatrixRow(Sample X,RDD[Samples] rdd)
rowRDD = rdd.map(Y => exp(-sqdist(X,Y))
return rowRDD
end function

3.2 Design approaches used for Scalable Algorithm for kernel matrix inverse

In order to compute the weights from hidden layer to output layer to finish the training, inverse of the above computed kernel matrix is needed. This section presents the various approaches used to compute inverse of the kernel matrix.

3.2.1 Inverse of matrix using Singular Value Decomposition(SVD)

The inverse is computed using Singular Value Decomposition which is computed using the Spark MLlib API. The algorithm to compute inverse is as follows:
Input :
$M \rightarrow$ Matrix to be inverted
Output :
$M^{-1} \rightarrow$ Inverse of M
function : Inverse(M)
$(U,S,V^T) = computeSVD(M)$
$M^{-1} = VS^{-1}U^T$
return M^{-1}
end function

The *computeSVD()* function will compute matrices U, S, V such that A = U * S * V', where S contains the leading N (order of matrix) singular values, U and V contain the corresponding singular vectors.

3.2.2 SVD using BLAS & LAPACK libraries

The BLAS (Basic Linear Algebra Subprograms) are routines that provide standard building blocks for performing basic vector and matrix operations. Because the BLAS are efficient, portable, and widely available, they are commonly used in the development of high quality linear algebra software, LAPACK for example, which provide significant performance improvements.

LAPACK is written in Fortran 90 and provides routines for solving systems of simultaneous linear equations, least-squares solutions of linear systems of equations, eigenvalue problems, and singular value problems.

Pre-built version of Apache Spark doesn't include BLAS and LAPACK. Hence, Spark was recompiled on all system of the cluster to include these libraries. The SVD along with LAPACK provides significant performance improvements.

3.2.3 Inverse using Block LU Decomposition

A block-recursive algorithm based on LU decomposition to compute the inverse of a large-scale matrix [6]. Below is the algorithm for computing inverse:

Input :

$A \rightarrow$ Matrix to be inverted

Output :

$A^{-1} \rightarrow$ Inverse of A

function : BlockInverse(A)

$(L^{-1}, U^{-1}, P) = BlockLUDecompose(A)$

$A^{-1} = U^{-1}L^{-1}P$

return A^{-1}

end function

The algorithm for ***BlockLUDecompose()*** is as follows:

Input :

$M \rightarrow$ Input Matrix

Output :

$L^{-1} \rightarrow$ Inverse of L

$U^{-1} \rightarrow$ Inverse of U

$P \rightarrow$ Pivot Matrix

function : BlockLUDecompose(M)
if M is small enought **then**
$(L,U,P) = LUDecompose(M)$
L^{-1} = Inverse(L)
U^{-1} = Inverse(U)
else
$$\begin{bmatrix} M_1 & M_2 \\ M_3 & M_4 \end{bmatrix} = M$$
$(L_1^{-1}, U_1^{-1}, P_1) = BlockLUDecompose(M_1)$
$U_2 = L_1^{-1}(P_1 M_2)$
$L_2' = M_3 U_1^{-1}$
$M' = M_4 - L_2' U_2$
$(L_3^{-1}, U_3^{-1}, P_3) = BlockLUDecompose(M')$
$L_2 = P_2 L_2'$
$$L^{-1} = \begin{bmatrix} L_1^{-1} & O \\ -L_3^{-1} L_2 L_1^{-1} & L_3^{-1} \end{bmatrix}$$
$$U^{-1} = \begin{bmatrix} U_1^{-1} & -U_1^{-1} U_2 U_3^{-1} \\ O & U_3^{-1} \end{bmatrix}$$
$$P = \begin{bmatrix} P_1 & O \\ O & P_2 \end{bmatrix}$$
end if
return (L^{-1}, U^{-1}, P)
end function

3.3 Implementation of Sampling Methods

As per the discussion in the previous chapter, to reduce the size of Big Dataset, three sampling techniques, namely Simple Random Sampling, K-Means clustering and Bisecting K-Means clustering have been applied. The implementation procedures of each are discussed below.

3.3.1 Simple Random Sampling

Apache Spark API provides a sampling method - $RDD.sample()$ as a part of RDD class. The prototype of the method $sample()$ is as follows:
public RDD<T> sample(boolean withReplacement,double fraction,long seed)

The function parameter description is as follows:

- withReplacement - can elements be sampled multiple times (replaced when sampled out)
- fraction - expected size of the sample as a fraction of this RDD's size without replacement: probability that each element is chosen; fraction must be [0, 1] with replacement: expected number of times each element is chosen; fraction must be >= 0
- seed - seed for the random number generator

From the samples generated using the above function, $take(int)$ is used to draw the desired number of samples.

3.3.2 K Means Clustering

K-means is one of the most commonly used clustering algorithms that clusters the data points into a predefined number of clusters. The *spark.mllib* implementation includes a parallelized variant of the k-means++ method called kmeans||. The implementation in spark.mllib has the following parameters:

- k is the number of desired clusters. Note that it is possible for fewer than k clusters to be returned, for example, if there are fewer than k distinct points to cluster.
- maxIterations is the maximum number of iterations to run.
- initializationMode specifies either random initialization or initialization via k-means||.
- initializationSteps determines the number of steps in the k-means|| algorithm.
- epsilon determines the distance threshold within which we consider k-means to have converged.
- initialModel is an optional set of cluster centers used for initialization. If this parameter is supplied, only one run is performed.

The KMeans method upon computation returns an array of samples which are to be treated as input samples during training the classifier.

3.3.3 Bisecting K-Means

Bisecting k-means algorithm is a kind of divisive algorithms i.e- it is a “top down” approach: all observations start in one cluster, and splits are performed recursively as one moves down the hierarchy.

The implementation in Spark MLlib has the following parameters:

- k: the desired number of leaf clusters (default: 4). The actual number could be smaller if there are no divisible leaf clusters.
- maxIterations: the max number of k-means iterations to split clusters (default: 20)
- minDivisibleClusterSize: the minimum number of points (if >= 1.0) or the minimum proportion of points (if < 1.0) of a divisible cluster (default: 1)
- seed: a random seed (default: hash value of the class name)

This method too upon completion returns an array of samples which are to be treated as input samples during training the classifier.

3.4 Modified Training Algorithm for Scalable ELM Classifier

Putting together the above discussed algorithms, the algorithm to train the Scalable ELM classifier is as follows:

1. Read the input Dataset from HDFS
2. Reduce the dataset using above discussed sampling methods
3. Compute kernel matrix from the reduced sample set
4. Compute inverse of the kernel matrix
5. Multiply with column matrix of ones
6. Computer error by taking difference with desired output
7. Determine the threshold as per number of rejection samples

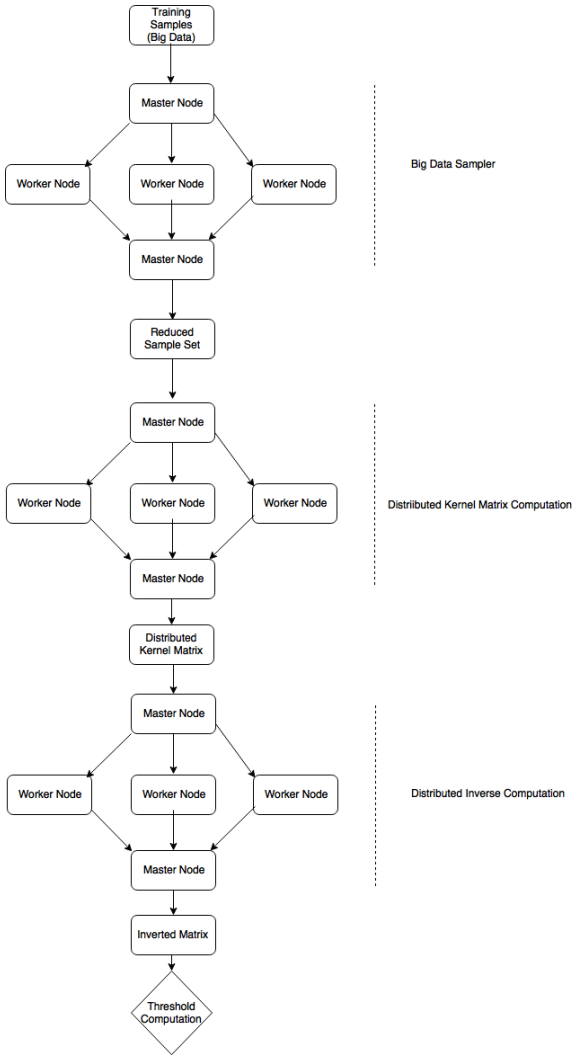


Fig. 3.1 Training the Scalable ELM Classifier

During the training, the Big Data Sampler reduces the number of input samples to a number N such that the kernel matrix of order N can be inverted on the cluster. The sampler implementation is such that it works in scalable distributed manner handling Big Data, thereby producing a smaller sample set yet representing the original dataset without losing much of accuracy. The overall training algorithm is designed in a way that it leverages the compute power of cluster through effective use of Spark MLlib APIs.

3.5 Testing the Scalable ELM Classifier

The testing of the classifier takes place in following manner:

1. Read the input Dataset from HDFS
2. Compute kernel matrix from the sample set
3. Multiply with weight matrix
4. Take the difference from column matrix of ones
5. Determine whether the sample is inlier or outlier
6. Compare with the actual required output to determine accuracy

Chapter 4

Setup & Implementation

This chapter discusses about the hardware & software setup utilized to implement training & testing algorithms on the Apache Spark framework.

4.1 Apache Spark Cluster setup

As discussed in chapter 2, Apache Spark cluster can be deployed in any of the three modes:

1. Standalone
2. Hadoop Yarn
3. Spark in Map Reduce

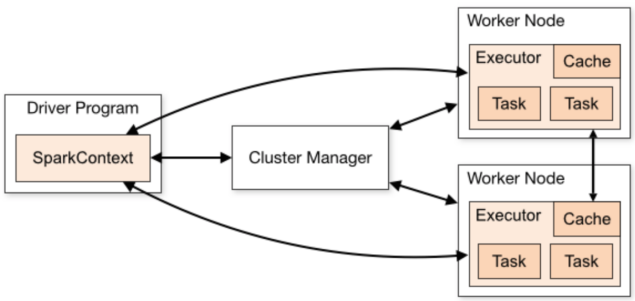


Fig. 4.1 Apache Spark Cluster Overview

For implementing the previously discussed algorithms, the Standalone mode of deployment is used as it is simple to implement and meets all the requirements of this project.

Hadoop Filesystem(HDFS) was installed on all the nodes in the cluster which serves as source to read input dataset and destination for writing the intermediate output.

Spark v2.1 built for Hadoop 2.7 and later was used in the course of this project. The prebuilt version is available to download from the official Apache Spark website. For detailed explanation to get HDFS & Apache Spark up and running on a local cluster, refer the Appendix section.

4.2 Hardware Setup for Cluster Deployment

The standalone cluster was setup using 3 High Performance Computing System. One of the systems doubled up as the Master Node as well as Worker Node while the rest were simply Worker Nodes. All systems run Ubuntu 16.04LTS operating system. The specifications of Master Node and Worker nodes are given below:

Master Node

- 4 cores
- 64GB RAM
- Intel Xeon Processor

Worker Node - 2 Nodes

- 4 cores
- 32GB RAM
- Intel Xeon Processor

Chapter 5

Experimental Analysis & Results

This chapter presents the results obtained by implementing the algorithms discussed in Chapter 3. All three sampling techniques were applied on all the datasets mentioned in Appendix A. This chapter also discusses the performance gains obtained on scaling up the cluster.

5.1 Classification Accuracy on various datasets

This section discusses the classification accuracy achieved by the ELM classifier on application of different sampling methods to reduce the Big dataset. The next page consists of tabular representation of the 'Threshold Value' and classification accuracy obtained by each sampling method.
In all the sampling methods, the Big Dataset was sampled down to 3K samples. In case of Random Sampling, 3K samples were picked with equal probability without replacement. In cluster sampling techniques, the number of clusters were set to 3K i.e.- 3K cluster centers were picked as representative samples.

Table 5.1 Dataset: http KDDCUP99 (567K)

Sampling Method	Threshold Value	Accuracy
Random Sampling	5.81	80.73%
K-Means	13.79	90%
Bisecting K-Means	109.94	92.46%

Table 5.2 Dataset: Forest Cover (286K)

Sampling Method	Threshold Value	Accuracy
Random Sampling	0.0	93%
K-Means	0.0	93%
Bisecting K-Means	0.0	93%

Table 5.3 Dataset: Shuttle (49K)

Sampling Method	Threshold Value	Accuracy
Random Sampling	0.268	89.33%
K-Means	0.00104	62.66%
Bisecting K-Means	0.0456	78.7%

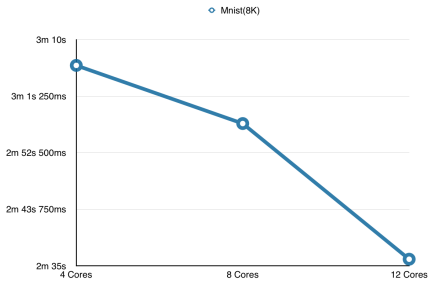
Table 5.4 Dataset: Mnist (8K)

Sampling Method	Threshold Value	Accuracy
Random Sampling	1.0	93.2%
K-Means	1.0	93.2%
Bisecting K-Means	1.0	93.2%

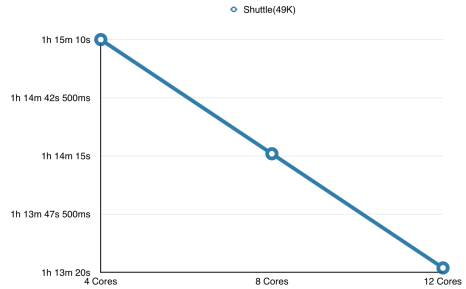
The results depict that even with large reduction ratios(especially in case of KDDCUP99 and Forest Cover datasets), the classification accuracy is pretty reasonable. Due to limit on matrix size order for inverse computation, large sample sizes couldn't be tested. Theoretically, increased number of samples should improve the training of the ELM classifier thereby increasing classification accuracy.

5.2 Computational Speedups

This section presents the performance gains obtained on scaling the cluster in terms of number of compute units. The results represented are for varying core counts for datasets of varying sizes.



(a) RBF Kernel Order 8K



(b) RBF Kernel Order 49K

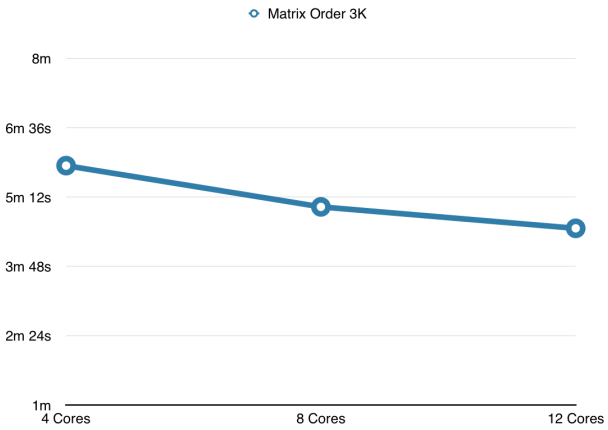


Fig. 5.2 Matrix Inverse Order 3K

The above graphs clearly depict the scalability of the algorithms. Increased core counts is reflected in reduced compute times. The algorithm leverages the ability of Apache Spark framework to increase performance with upscaling of cluster.

Chapter 6

Conclusion and Future Scope

Extreme Learning Machine is a promising straightforward algorithm which can achieve a relatively high Overall Accuracy and can significantly decrease the time of the training phase. However, massive data or Big Data is a big challenge for the practical use of Kernel ELM in terms of both space complexity and computational time complexity. Through this project we implemented the One Class Kernel ELM and obtained significant levels of computational speedup when scaling up the cluster retaining same levels of accuracy as of kernel ELM on a single machine. Through this project, we realized the advantages, capabilities and limitations of Apache Spark framework and gained significant insights in the fields of Machine Learning and Big Data Handling.

While implementing the algorithm,we hit a roadblock where we got restricted on the size of the invertible matrix. For future research, perhaps a better distributed way to compute kernel matrix inverse can be developed and implemented which would improve the training of the ELM thereby increasing the overall accuracy of the classifier.

References

[1] Castillo, E., Peteiro-Barral, D., Berdiñas, B. G., and Fontenla-Romero, O. (2015). Distributed one-class support vector machine. *International journal of neural systems*, 25(07):1550029.

[2] De Chazal, P., Tapson, J., and van Schaik, A. (2015). A comparison of extreme learning machines and back-propagation trained feed-forward networks processing the mnist database. In *Acoustics, Speech and Signal Processing (ICASSP), 2015 IEEE International Conference on*, pages 2165–2168. IEEE.

[3] Gautam, C., Tiwari, A., and Leng, Q. (2017). On the construction of extreme learning machine for online and offline one-class classification - an expanded toolbox. *CoRR*, abs/1701.04516.

[4] Hofmann, T., Schölkopf, B., and Smola, A. J. (2008). Kernel methods in machine learning. *The annals of statistics*, pages 1171–1220.

[5] Huang, G.-B., Zhou, H., Ding, X., and Zhang, R. (2012). Extreme learning machine for regression and multiclass classification. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):513–529.

[6] Liu, J., Liang, Y., and Ansari, N. (2016). Spark-based large-scale matrix inversion for big data processing. *IEEE Access*, 4:2166–2176.

[7] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., Freeman, J., Tsai, D., Amde, M., Owen, S., et al. (2016). Mllib: Machine learning in apache spark. *The Journal of Machine Learning Research*, 17(1):1235–1241.

[8] Rekha, A. (2015). A fast support vector data description system for anomaly detection using big data. In *Proceedings of the 30th Annual ACM Symposium on Applied Computing*, pages 931–932. ACM.

[9] Spark, A. Bisecting k-means clustering.

[10] Spark, A. K-means clustering.

[11] UCI. Forest cover/covertype dataset [https://archive.ics.uci.edu/ml/datasets/covertype].

[12] UCI. Kddcup 1999 [http://archive.ics.uci.edu/ml/datasets/kdd+cup+1999+data].

[13] UCI. Shuttle dataset[https://archive.ics.uci.edu/ml/datasets/statlog+(shuttle)].

[14] Vert, J.-P., Tsuda, K., and Schölkopf, B. (2004). A primer on kernel methods. *Kernel Methods in Computational Biology*, pages 35–70.

[15] Wang, H., Hu, Z., and Zhao, Y. (2006). Kernel principal component analysis for large scale data set. *Lecture Notes in Computer Science*, 4113:745.

Appendix A

Outlier Detection Datasets (ODDS)

A description of the datasets used for training and testing purposes.

The datasets used were obtained from ODDS repository of Stony Brook University, New York.Following is a small description of various datasets used:

Shuttle Dataset

The original Statlog (Shuttle) dataset from UCI machine learning repository [13] is a multi-class classification dataset with dimensionality 9. Here, the training and test data are combined. The smallest five classes, i.e. 2, 3, 5, 6, 7 are combined to form the outliers class, while class 1 forms the inlier class. Data for class 4 is discarded.

http KDDCUP99 Dataset

The original KDD Cup 1999 dataset from UCI machine learning repository [12] contains 41 attributes (34 continuous, and 7 categorical), however, they are reduced to 4 attributes $(service, duration, src_bytes, dst_bytes)$ where only 'service' is categorical. Using the 'service' attribute, the data is divided into $(http, smtp, ftp, ftp_data, others)$ subsets. Here, only 'http' service data is used. Since the continuous attribute values are concentrated around '0', we transformed each value into a value far from '0', by y = log(x + 0.1). The original data set has 3,925,651 attacks (80.1%) out of 4,898,431 records. A smaller set is forged by having only 3,377 attacks (0.35%) of 976,157 records, where attribute 'logged-in' is positive. From this forged dataset 567,497 $http$ service data is used to construct the http (KDDCUP99) dataset.

Mnist Dataset

The original MNIST dataset of handwritten digits has a training set of 60,000 examples, and a test set of 10,000 examples. It is a subset of a larger set available from NIST. The digits have been size-normalized and centered in a fixed-size image. This dataset is converted for outlier detection as digit-zero class is considered as inliers, while 700 images are sampled from digit-six class as the outliers. In addition, 100 features are randomly selected from 784 total features.

Forest Cover Dataset

The original ForestCover/Covertype dataset from UCI machine learning repository [11] is a multiclass classification dataset. It is used in predicting forest cover type from cartographic variables only (no remotely sensed data). This study area includes four wilderness areas located in the Roosevelt National Forest of northern Colorado. These areas represent forests with minimal human-caused disturbances, so that existing forest cover types are more a result of ecological processes rather than forest management practices. This dataset has 54 attributes (10 quantitative variables, 4 binary wilderness areas and 40 binary soil type variables). Here, outlier detection dataset is created using only 10 quantitative attributes. Instances from class 2 are considered as normal points and instances from class 4 are anomalies. The anomalies ratio is 0.9%. Instances from the other classes are omitted.

Appendix B

Apache Spark and Hadoop - Setup Guidelines

B.1 Setting up Environment

B.1.1 Installing Java & Scala

In order to get Hadoop and Apache Spark running on your systems, Java(Java 8) and Scala(2.12 and above) are required. This section provided instruction to install the same.

Installing Java

```
1. Open terminal
2. sudo apt-get update
3. sudo apt-get install openjdk-8-jdk
```

Installing Scala

```
1. Open terminal
2. sudo apt-get update
3. sudo apt-get install scala
```

B.1.2 Setting up password-less SSH

All the systems in the cluster need to communicate with each other seamlessly in order to start and stop processes and services.Hence setting up of password-less SSH is essential. The link provided below explains all the steps to do the same.

http://www.linuxproblem.org/art_9.html

B.2 Setting up Hadoop on Ubuntu

With Java installed on your system, the below linked URL provides detailed guidelines to install Hadoop(Hadoop 2.7.3) on each system of your cluster Since Java is already installed on the system, proceed directly to Step 2 in the tutorial.

Link:- https://www.digitalocean.com/community/tutorials/how-to-install-hadoop-in-stand-alone-mode-on-ubuntu-16-04

Once Hadoop is installed on each system of the cluster, follow the steps in the URL given below to setup of Hadoop cluster.

Link:- http://www.michael-noll.com/tutorials/running-hadoop-on-ubuntu-linux-multi-node-cluster/

B.3 Setting up Apache Spark

The prebuilt version of Apache Spark is available on the offical Apache Spark website.The link for the same is given below. This prebuilt version needs to be downloaded and extracted on each system which is part of the cluster.

Link:- https://archive.apache.org/dist/spark/spark-2.1.0/spark-2.1.0-bin-hadoop2.7.tgz

Once Apache Spark spark has been downloaded and installed on each system, follow the steps in tutorial on the link below to setup Spark Standalone Cluster:

Link:- http://paxcel.net/blog/how-to-setup-apache-spark-standalone-cluster-on-multiple-machine/

B.4 Running Spark Jobs on Cluster

In order to run spark jobs written in scala follow the steps given below:

```
1.On master system,open terminal
2.cd <path-to-prebuilt-spark>
3. ./bin/spark-shell --master spark://<master IP>:7077 --total-executor-cores <no. of
4. To compile the code -> :load <name of file>.scala
5. To run the code -> <name of object>.main(Array(<arguments>))
```