PERFORMANCE COMPARISON OF APACHE SPARK MLLIB

A Paper
Submitted to the Graduate Faculty
of the
North Dakota State University
of Agriculture and Applied Science

By

Pallavi Sharma

In Partial Fulfillment of the Requirements
for the Degree of
MASTER OF SCIENCE

Major Department:
Computer Science

August 2018

Fargo, North Dakota

North Dakota State University
Graduate School

Title

PERFORMANCE COMPARISON OF APACHE SPARK MLLIB

By

Pallavi Sharma

The Supervisory Committee certifies that this ***disquisition*** complies with North Dakota State University's regulations and meets the accepted standards for the degree of

MASTER OF SCIENCE

SUPERVISORY COMMITTEE:

Dr. Simone Ludwig
Chair

Dr. Kendall Nygard

Dr. Ravi Kiran Yellavajjala

Approved:

08/17/2018	Dr. Kendall Nygard
Date	Department Chair

ABSTRACT

This study makes an attempt to understand the performance of Apache Spark and the MLlib platform. To this end, the cluster computing system of Apache Spark is set up and five supervised machine learning algorithms (Naïve-Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression) were investigated. Among the available cluster modes, these algorithms were implemented on two cluster modes, Local and GPU Cluster mode. The performance metrics such as classification accuracy, area under ROC and area under PR for the algorithms were investigated by considering three datasets. It is concluded that the algorithms are computed in parallel in both the modes with GPU Cluster mode performing better than the Local mode for all algorithms in terms of time taken for completion. However, the mentioned performance metrics were not affected in the two modes hinting that the parallel computation does not play a major role in determining these metrics.

ACKNOWLEDGEMENTS

I would like to thank my advisor, Dr. Simone Ludwig, for her constant guidance throughout the project. Her timely feedback and encouraging words helped to complete the project within time. I’d also like to thank my graduate committee members, Dr. Kendall Nygard and Dr. Ravi Kiran Yellavajjala, for their support with the study. A special thanks to Computer Science department System Administrator, Guy Hokanson, for his time and patience in answering my queries. I would also like to extend my gratitude to Computer Science IT department for assistance in technical problems. I am greatly indebted to Dr. Dayakar Naik Lavadiya for his valuable comments as the second reader of this report. Lastly, I wish to acknowledge my family and everyone who has been a part of my graduate life in North Dakota State University, for their unfailing support.

DEDICATION

I would like to dedicate this project to my mother, father and brother who’ve encouraged and supported me through every decision in my life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF APPENDIX TABLES

LIST OF APPENDIX FIGURES

1. INTRODUCTION AND RELATED WORK

With the rapid growth of the digital age, there has been an explosion in the amount of data generated. This data is generated through almost every action, whether it is using GPS-equipped smartphones or using social media or online shopping or even browsing history. There is a trail of digital footprints with everything one does digitally. Some of the areas generating this massive amount of data are scientific experiments, industrial processes, healthcare records, weather sensors and business transactions. This collection of data, called Big Data, can be utilized to our advantage to know reliably about any process, gain new insights and even predict future trends. With the increasing amount of data there is an ever-growing demand for faster data ingestion and processing. Big Data infrastructures have come up with faster, more reliable and scalable computing architectures for efficient mining of such massive datasets [1]. Machine learning systems help to manage, analyze and use the data far more successfully than before. The insight gained is deeper and faster by automating analytical model building which is far better than human analysis. Several machine learning frameworks have contributed to scientific applications in healthcare informatics [2], [3], [4], genome data analysis [5], [6], text mining [7], [8] and stochastic modelling [9] to name a few.

The Apache Hadoop project was developed as an open-source, reliable, scalable solution for distributed computing [10]. It allows for distributed processing of huge data across clusters of computers. It is fault-tolerant and delivers leading services. Apache Spark was developed as a faster alternative to Hadoop. Hadoop MapReduce reads and writes from disk, which slows down the processing speed. Spark, on the other hand, stores the data in-memory and reduces the read/write cycle. This results in running the applications 100x faster in memory and 10x faster on disk than Hadoop MapReduce [11].

Apache Spark MLlib has emerged as one of the prominent platform-independent and open source libraries for distributed computing. It is provided with Apache Spark [12] and offers a variety of machine learning algorithms for classification, regression, clustering, feature extraction, etc. [13]. There have been limited studies on Apache Spark MLlib so far, even though the machine learning has found many applications in the research community. [14] offers a study on Apache Spark MLlib and its capabilities through a series of experiments. This study initiates a study of big data machine learning on massive datasets and performs a comparative study with the Weka library [15] to evaluate Apache Spark MLlib. It is established that Apache Spark MLlib works at par with the mentioned software.

Apache Spark is developed in the Scala programming language, and offers interfaces for Java, Python and most recently the R language. It makes use of the distributed computing architecture of Apache Spark to run the machine learning algorithms faster in an iterative manner without compromising their performance. The performance can be further leveraged by using different mode of operations Spark offers. The present studies presenting the performance of the Apache Spark Machine Learning library are focused on measuring the performance through Scala and Java implementations. Keeping [14] as the motivation, the current study plans to explore the Apache Spark architecture that powers the MLlib and evaluate its performance within the different modes of cluster operation it offers using Python as our main language of implementation.

In this paper, we implement five supervised machine learning classification algorithms namely Naive-Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression and measure their performance. These performance metrics consists of accuracy, execution time, area under ROC and area under PR. These algorithms are implemented in two modes of cluster operation, Local Cluster mode and GPU cluster mode to better compare the performance. Since Apache Spark can be evaluated best when it is working in parallel on large datasets, we use datasets of considerable size (>5 GB). The objectives of the study can be summarized as follows:

1) To understand the cluster computing system of Apache Spark and implement on large datasets.
2) To understand the MLlib platform and measure performance metrics such as accuracy, execution time, area under ROC and area under PR of five supervised machine learning classification algorithms.
3) To evaluate the performance of algorithms on large datasets between different mode of cluster operations.

The rest of the study is organized as follows: Chapter 2 presents the concepts of Machine Learning and Apache Spark used in this study. Chapter 3 presents the methodology followed throughout the study. The experiments and results are presented in Chapter 4 and 5, respectively. Chapter 6 presents the conclusion and discusses limitations and possible future work.

2. MACHINE LEARNING AND APACHE SPARK

In this chapter, the various architectural and mathematical concepts involved in the process for implementation of the current study are presented. The five machine learning algorithms (Naive-Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression) for supervised learning are briefly discussed first followed by the explanation of the Apache Spark architecture and the various components with the primary focus on the Apache Spark Machine Learning library. The concept of Resilient Distributed Dataset (RDD), the associated transformations and actions, and the Spark Context is explained thereafter and concluding with an explanation of various modes of cluster operation in Spark.

2.1. Machine Learning Algorithms

Machine learning is a way to make a computer learn on the basis of hundreds of examples and experiences without programming them explicitly [16]. It uses the data given to build the logic using different algorithms. Machine learning can be broadly classified into three categories: 1) Supervised Learning, 2) Unsupervised Learning, and 3) Reinforcement Learning.

In Supervised Learning, the algorithms can be used to predict the output values (Regression) or classify the category (Classification). There are two types of supervised algorithms, Parametric and Non-Parametric. Parametric algorithms utilizes fixed number of features to predict the unknown class label whereas Non-Parametric is flexible with number of features for prediction and parameters can grow as it learns from more data [17]. Examples include trying to classify mails as “Spam” or “Not Spam”, predicting the house prices for the following years based on current market values, etc.

In Unsupervised Learning, the system tries to find the hidden pattern or meaningful structures in the unlabeled data. There are no pre-declared output class labels to help the system learn. It can be used to discover either association between data points (Association) or to cluster them together depending on their inherent properties (Clustering). Examples include market basket analysis, identifying fraudulent credit card purchases, recommender systems, etc.

In Reinforcement Learning, the system must interact with the dynamic environment and make decisions to arrive at a goal. It works on collecting feedback to evaluate expected or unexpected behavior of the system. Examples include self-driving cars, self-cleaning vacuum cleaners, etc. [16] [18].

2.2. Supervised Learning Algorithms

In Supervised Learning, we focus on five algorithms out of a number of supervised machine learning algorithms. These are Naive-Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression. A detailed explanation of these algorithms is available in [19]. However, a brief description is presented here for completeness.

2.2.1. Naive-Bayes Classification

Naive-Bayes Classifier belongs to the family of probabilistic classifiers and is based on the Bayes' Theorem. It assumes strong independence between the features [20]. Bayes' Theorem states that the probability of occurrence of a hypothesis H given an event E has occurred P(H|E) is given by:

$$P(H|E) = \frac{P(E|H) * P(H)}{P(E)} \tag{1}$$

where P(E) is probability of the event, P(H) is the probability of the hypothesis H before the event E, and P (E|H) is the probability of the event E given the hypothesis H has occurred [21]. We extend Bayes' Theorem to a number of independent variables $X=\{x_1, x_2,\ldots, x_m\}$. For these given set of variables, we construct the posterior probability for event C_i from a set of possible outcomes $C=\{c_1, c_2, \ldots, c_d\}$. Using Bayes' rule:

$$p(C_i|x_1, x_2, .., x_m) \propto p(x_1, x_2, .., x_m|C_i) * p(C_i) \tag{2}$$

where $p(C_i|x_1, x_2, .., x_m)$ is the posterior probability that X belongs to C_i. Now, Bayes assumes the conditional probabilities of the independent variables are independent, we can break down the likelihood to a product given as:

$$p(X|C_i) \propto \prod_{k=1}^{m} p(x_k|C_i) \tag{3}$$

The posterior probability can be rewritten as:

$$p(C_i|X) \propto p(C_i) * \prod_{k=1}^{m} p(x_k|C_i) \tag{4}$$

Now, a data point from a testing set is assigned a class label C_i that achieves the highest posterior probability [22].

2.2.2. Decision Tree

Decision Tree Classifier is a tree-based classifier that consists of hierarchy of decisions to predict the unknown class label. A model is created where the class of the target dependent variable is predicted based on several input variables. The tree is learned by splitting the input features into subsets based on the information gain. This process is repeated on each derived subset until all data points have a class label assigned. Information gain is used to determine which feature to split in building the tree. The Information gain is given as:

$$IG(T, a) = H(T) - H(T|a) \tag{5}$$

where $IG(T, a)$ is the information gain for parent node T and children nodes a, $H(T)$ is the entropy of the parent node and $H(T|a)$ is the weighted sum of entropy of the children node. Entropy measures the amount of disorder or uncertainty in a system [23]. Equation 5 can be rewritten as:

$$IG(T, a) = -\sum_{i=1}^{J} p_i \log_2 p_i - \sum_{a} p(a) \sum_{i=1}^{J} -\Pr(i|a) \log_2 \Pr(i|a) \tag{6}$$

where p_1, p_2,… are the fractions that add to 1 and represent the percentage of each class presented in the child node that results from a split in the tree [24].

2.2.3. Random Forest

Random Forest is an ensemble learning method used for classification and regression. Ensemble learning methods combine more than one algorithm of the same or different type for classifying purposes. Random Forest builds an ensemble of decision trees. It creates a set of decision trees from a randomly selected subset of the training set. It adds randomness to the model while growing the trees. Instead of searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. It then aggregates the votes from different decision trees to decide the final class label of the testing data point. The data point is assigned a class which gets maximum votes. Random Forest offers to improve the problem of overfitting with decision trees [25].

2.2.4. Support Vector Machine

In an SVM model, the data points are represented as points in space and they are separated by a clear gap to assign them into class labels. While training the data, this gap is created by constructing hyperplanes in high-dimensional space. In 2-D space, this hyperplane is represented as a line that divides the plane in two parts. Each class label lies in each of these parts. When a new data point is required to be classified, it is mapped into the space and predicted to belong to a class based on which side of the hyperplane they lie [26].

2.2.5. Logistic Regression

Logistic Regression is the regression method to describe the relationship between a dependent variable and one or more nominal, ordinal, interval or ratio-level independent predictor variables. When this dependent variable is binary in nature, it is called binary logistic regression model, and multinomial logistic regression when there are more than two categories [27]. Here, we employ a linear combination of one or more predictor variables to calculate the log-odds of the probability of an event. Mathematically, it is represented as:

$$logit(p_i) = \frac{p(y=1)}{1-p(y=1)} = \beta_0 + \beta_1 x_{1,i} + \ldots + \beta_m x_{m,i} \tag{7}$$

where p_i is the probability outcome, $\beta_0, \beta_1, \ldots, \beta_m$ are the regression coefficients, and $x_1, x_2, \ldots, x_m$ are the predictor variables for each data point i. The data point i varies from 1 to the length of the training set [28].

2.3. Apache Spark

Apache Spark is an open-source cluster computing framework. It was originally developed by Matei Zaharia in 2014 and later donated to the Apache Spark Foundation. The framework is built on top of the Hadoop Distributed File System (HDFS). It works on distributed processing of data, handing out data to separate worker nodes for processing. The worker nodes are managed by a master node which dispatches and schedules the distributed tasks. Hence, Spark requires a cluster manager and distributed storage system [29]. Its faster in-memory data engine and developer-friendly API makes it the framework of choice [11].

2.3.1. Architecture

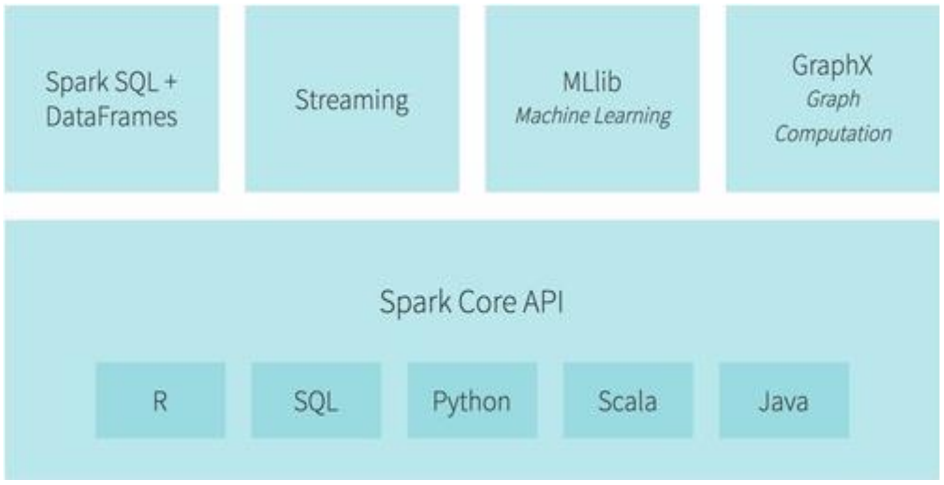


Figure 1. Apache Spark Ecosystem [30]

The architecture of Spark is presented in Figure 1. Spark Core serves as the underlying general execution engine and is the foundation of Apache Spark. It supports dispatching of distributed tasks, their scheduling and I/O functionalities. This is made possible through an application programming interface (API) for Java, R, Python and Scala. With interfaces present for many languages, it can support a wide variety of languages. It delivers speed utilizing the in-memory computing capabilities [29]. The API follows a higher-level programming approach with use of a driver program that calls parallel operations on a Resilient Distributed Dataset (RDD) by passing a function to Spark. The core schedules the execution in parallel on the available clusters. Until Spark 2.1.x, Resilient Distributed Dataset (RDD) was the primary API. It is a fault-tolerant, read-only collection of dataset which is distributed over the cluster machines for processing. From Spark 2.2.x, the Dataset API is promoted, though RDD is still in use. We discuss RDD in a later section. Other than using RDD, Spark also uses Broadcast variables, which help in sharing of common read-only data among clusters, and Accumulators, which help in program reductions [31].

On top of the Spark Core, there are four components namely Spark SQL, Spark Streaming, Machine Learning library and GraphX. Spark SQL is the module of Spark that deals with structured data using dataframes. It has a SQL-like interface to query and process data [31]. Spark Streaming allows to add real-time processing of data in addition to Spark's batch processing. It breaks down the incoming data stream into micro batches and processes them in the same way as batch processes in Spark. Both processes work on the same code in the same framework, thus reducing overhead [31]. GraphX is a distributed framework on top of Spark for processing graph structures. It allows users to build and process graph structured data interactively [30]. The Machine Learning library allows to implement the machine learning pipelines in a distributed manner, thereby decreasing the overall processing time significantly. There are a growing number of machine learning algorithms for Classification, Clustering, Collaborative Filtering, Regression and Dimensionality Reduction. All algorithms are implemented in a distributed fashion and allow for easy execution of feature extraction, selection, and transformation on structured dataset. It provides tools for constructing, evaluating and tuning ML pipelines, along with saving and loading algorithms, models and pipelines [32].

2.3.2. Resilient Distributed Dataset (RDD)

Resilient Distributed Dataset is the primary API in Spark since its deployment. It is an immutable collection of elements of data, which is distributed among the different nodes in the cluster. These nodes could then be operated in parallel using the operations defined for RDDs [30]. The operations are split across the cluster, executed in parallel leading to reduced time in processing [31]. RDDs are fault-tolerant since they can be reconstructed in case of loss by keeping track of the sequences that produced them [29]. They can be created in two ways:

parallelizing an existing dataset in your program or referencing a dataset from an external storage program like HDFS, Hbase, etc [33].

There are two types of operations available with RDDs, “transformations” and “actions”. Transformations create a new RDD from an existing one, and Actions return a value to the driver program after running computation on a RDD. All transformations are lazy in Spark, which is they do not compute the results immediately. Instead, they remember the computation that needs to be performed and apply that computation only when an action is applied by driver program. This design helps to run Spark efficiently [33]. The most popular transformation is “map” which passes each RDD element through a defined function, returning a new RDD, and an action is “reduce” which reduces the RDD into a collective value by passing through a defined function. As explained, transformations like map is a lazy operation and will only be evaluated when reduce or any other action will be called on the RDD. We have used map, filter transformation and reduce, count actions in the current study. “Filter” filters the RDD based on a filter function, and “Count” returns the number of elements in the RDD passed to it. One of the interesting properties of Spark is the ability to cache datasets in memory. When a dataset is persisted, each node that has used the dataset to do some computation will store it in partitions and reuse it later for other actions. This helps in faster usage of datasets. The datasets can be persisted in a number of different ways: only in memory (default level), memory and disk, disk only. These storage levels are available for simultaneous replication on two nodes as well [33].

2.3.3. Spark Context

The Spark Context is the main entry point for Spark functionality. It allows the driver program to access the clusters through the means of a cluster manager like YARN, Mesos, etc. There must only be one active Spark Context per JVM and it should be stopped using stop()

before creating a new one [34]. We require Spark Conf to create the Spark Context. It stores configuration parameters like cluster size, cores, application name to connect driver program to the cluster, etc.

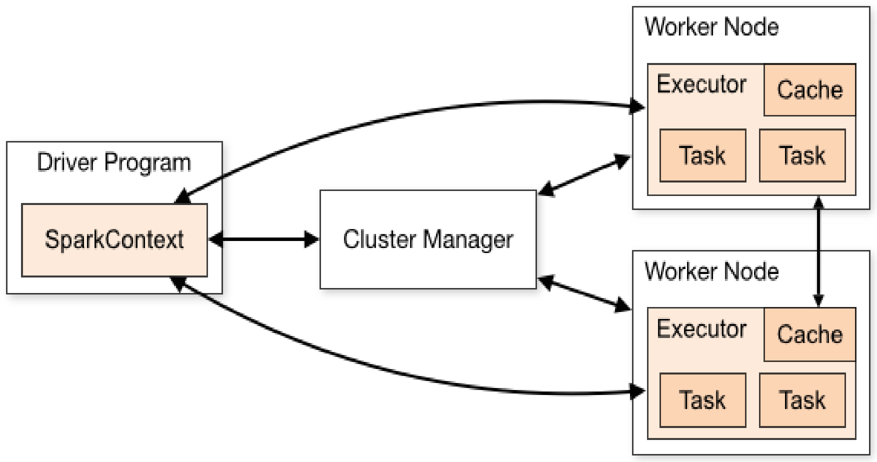


Figure 2. Relationship between driver application, Cluster Manager and executors [35]

The relationship between the driver application, the cluster resource manager and executors is shown in Figure 2. Each cluster has one driver node and one or more worker nodes. These worker nodes have executors, which can be accessed by the driver application through the Spark Context. As soon as the driver program starts executing, the Spark Context creates a job and breaks it into stages. These stages are further broken into tasks which are scheduled by the Spark Context on each executor. These executors run the user code, run computations and cache the data for the application. They return the result back to driver application [35].

2.3.4. Modes of Cluster Operation

The Spark Context coordinates the Spark applications that are run on a cluster. It can connect to different types of cluster managers for allocation of resources across applications, send out tasks to executors, and collect the results from them [36]. There are four modes of operations supported by Spark: 1) Local mode, 2) Standalone mode, 3) using Apache Mesos, and 4) using Hadoop YARN.

The Local mode or the Pseudo cluster mode is a single JVM deployment mode where the driver, executor and master are on the same machine. The default parallelism arises from the number of threads specified in the master URL [37]. Standalone mode is the basic cluster manager available with Spark. It consists of master and workers with specified memory and cores. By default, it takes up all the cores unless otherwise specified. It allows for automatic recovery, SSL encryption and web UI for cluster and job status. Apache Mesos is helpful for deploying and managing large-scale clusters. It makes use of dynamic resource allocation and isolation to handle workload. It authenticates workers' registration with the master, has frameworks that allow for request and allocation of resources to workers. Hadoop YARN separates the functionalities of resource manager and job scheduling such that there is a Global Resource Manager and per-application Application Manager. It contains security for authorization, web UI for managing resources, and supports manual recovery.

We observe the Local mode and the Standalone mode in this study as the Standalone mode provides the same features as high-end cluster managers and is comparable.

3. METHODOLOGY

The aim of the study is to investigate the performance of machine learning algorithms in two cluster modes of the Apache Spark MLlib platform and later compare them. For this purpose, five machine learning algorithms are implemented on large datasets with size in tens of GBs so that the cluster computing system of Apache Spark can be better studied and the performance can be evaluated. The objectives of the study are accomplished by following a methodology that is described in detail in this chapter. The methodology adopted in this study involves five steps: 1) Data Description, 2) Prerequisite installation, 3) Data splitting and Training, 4) Testing the model, and 5) Output metrics.

3.1. Data Description

Prior to employing any machine learning algorithm, there are three steps that needs to be completed: 1) Data Acquisition from a reliable source, 2) Data Cleaning, and 3) Evaluation of descriptive statistics for the cleaned data. Apart from these steps, we also convert the acquired data into LabeledPoint format as required by the algorithms. We describe these steps in detail in this section.

3.1.1. Data Acquisition

Data Acquisition is the process of collecting data from various sources and utilizing it to address the problem statement and analyze the results. Data can be collected primarily by two methods: 1) Primary Data Collection where the data is first-hand and collected through surveys, interviews, site works, etc., and 2) Secondary Data Collection where data is available to use through public libraries, books, web information, etc. [38]. In this study, the datasets are collected through the Primary Data Collection from the UC Irvine Machine Learning Repository [39]. This repository consists of databases, domain theories and domain generators that are useful for the machine learning community [40]. The datasets are selected based on the number of instances present, attribute type, and default task. The default task is chosen as “Classification”, attribute type as “Real”, and the number of instances to be equal to or more than five million. Based on these filters, the three datasets which are finalized are: 1) HIGGS, 2) SUSY, and 3) Hepmass. The explanation for them is given below.

The datasets HIGGS and SUSY are part of a same physics experiment for classification where exotic particles are generated by collisions at high-energy particle colliders. The collisions that produce these exotic particles are called Signal process and the collisions that produce other particles are called Background process [41]. The datasets are produced by Monte Carlo simulations and used to solve the classification problem to identify the signal process and background process. The HIGGS dataset contains information about collisions that produce the HIGGS boson and those which do not. The SUSY dataset contains information about collisions that produce supersymmetric particles and those which do not.

The dataset Hepmass is also part of a physics experiment for search of exotic particles. This experiment requires sorting through a large number of collisions to find the signatures of a process that produces exotic particles. These signatures are learned from Monte Carlo simulations of the collisions that produce the expected particles [42]. There are three datasets according to the mass of the particle. We have used the dataset where the mass of the particles produced is 1,000. The classification problem is to identify the signal process that produces the exotic particles and the background process that does not.

3.1.2. Data Cleaning

Data cleaning is the process of correcting the inconsistencies in the data with the aim of generating more organized and structured data [43]. Very often, the data acquired from the primary source (called as raw data) consists of missing information, typographic errors and inconsistent format, etc. It is almost impossible to perform any kind of statistical analysis on such unorganized inconsistent data. Therefore, the data needs to be corrected for any such existing errors and inconsistencies using tools such EXCEL®, MATLAB® and SQL® etc.

There are a total of twenty-eight features in the HIGGS dataset where the first label is the class label (1 for signal, 0 for background), followed by 28 features. Among these 28 features, the first 21 features are low-level features, which depict the kinematic properties measured by the particle detectors in the accelerator, and the next 7 features are high-level features, which are functions of the first 21 features, derived by physicists to help discriminate between the two classes. The 28 features are as follows: lepton pT, lepton eta, lepton phi, missing energy magnitude, missing energy phi, jet 1 pt, jet 1 eta, jet 1 phi, jet 1 b-tag, jet 2 pt, jet 2 eta, jet 2 phi, jet 2 b-tag, jet 3 pt, jet 3 eta, jet 3 phi, jet 3 b-tag, jet 4 pt, jet 4 eta, jet 4 phi, jet 4 b-tag, m_jj, m_jjj, m_lv, m_jlv, m_bb, m_wbb, m_wwbb. There are 11 million data points in this dataset sizing to 8.0 GB. All features are in real number format and there are no missing values [44]. Positive examples amount to 53% of the dataset [41].

For the SUSY dataset, there are a total of eighteen features where the first label is the class label (1 for signal, 0 for background), followed by 18 features. Among these 18 features, the first 8 features are low-level features which depict the kinematic properties measured by the particle detectors in the accelerator, and the next 10 features are high-level features, which are functions of first 8 features, derived by physicists to help discriminate between the two classes. The 18 features are as follows: lepton 1 pT, lepton 1 eta, lepton 1 phi, lepton 2 pT, lepton 2 eta, lepton 2 phi, missing energy magnitude, missing energy phi, MET_rel, axial MET, M_R, M_TR_2, R, MT2, S_R, M_Delta_R, dPhi_r_b, cos(theta_r1). There are 5 million data points sizing to 2.5 GB. All features are in real number format and there are no missing values [45]. Positive examples amount to 46% of the dataset [41].

There are total of twenty-seven features in Hepmass dataset where the first label is the class label (1 for signal, 0 for background), followed by 27 normalized features [46]. Among these 27 features, the first 22 features are low-level features which depict the result of standard reconstruction algorithms and are roughly the four-vectors of the reconstructed events: the leading lepton momenta lepton pT, the momenta of the four leading jets jet 1 pT, the b-tagging information for each jets and the missing transverse momentum magnitude and angle MTM. The next 5 features are high-level features strictly to combine the low-level information to form approximate values of the invariant masses of the intermediate objects: mlv, mjj, mjjj, mjlv, mWWbb [42]. There are 7 million data points for the training purpose sizing to GB and 3.5 million data points for testing sizing to GB. All features are in the real number format and there are no missing values [44]. The dataset has equal positive and negative examples that is 50% positive examples [42].

There is no inconsistent format as all features are in real number format with class label as integer and no missing data [44][45][46].

3.1.3. Descriptive Statistics

Descriptive statistics consists of a set of techniques that are used to summarize and characterize the measurements of the given data. Generally, descriptive statistics includes (a) the basic statistical measures such as mean, median, range, standard deviation, (b) identifying the type of distribution, and (c) recognizing the patterns among variables. The Apache Spark MLlib Statistics library is utilized to accomplish this, and the results and patterns are discussed later in Chapter 5.

3.1.4. Data Conversion

The supervised machine learning algorithms in Apache Spark MLlib require the input data to be in the form of Labeledpoint libsvm format. As defined in the Apache Spark documentation, a labeled point is a local vector, either dense or sparse, associated with a label. It also supports data stored in the LIBSVM format [47]. It is a text format where each line first begins with the class label followed by a labeled sparse feature vector. Figure 3 presents an example where there are three features preceded by the label in libsvm format.

Label	Index1:Value1	Index2:Value2	Index3:Value3
0	1:1.532	2:2.251	3:5.652
1	1:4.521	2:8.521	3:1.564
0	1:1.245	2:5.215	3:3.584

Figure 3. An example of libsvm format [47]

Here, the class label can be binary or multiclass and all features are given an index starting from index 1. We use sparse training and testing data for this study. The three datasets are converted into the required format using a python script to include the class label and the corresponding index for their features. From now on, all input datasets used in machine learning algorithms are in the libsvm format.

3.2. Prerequisite Installation

In order to begin our study, we need to install the required software and fulfill some prerequisites. We discuss these in this section. We are using the Ubuntu operating system [48] and begin by installing Java 8 [49] and Python 3.6.4 [50] in the system. Next, we install Anaconda 5.1.0 [51] to access PySpark. Anaconda is a free and open-source distribution of the Python and R programming language. The graphical interface Anaconda Navigator helps to launch applications and manage packages, environments without command-line commands [52]. Following this installation, we install PySpark [53]. Spark is based on the Scala programming language. Hence, in order to use Spark with Python we need PySpark that exposes the Spark programming model to the Python language. We also install Spark 2.3.0 [53] alongside PySpark. After successful installation of all required software, we need to set the environment variables for the system so that the software can access the path variables. Path variables are environment variables in operating systems that specify the directories where executable programs are present [54]. We need to set path variables for Spark, PySpark, Java, and Anaconda. This process needs to be completed for both the Local cluster and the GPU cluster mode. After completion of this step, we can move to begin our implementation.

3.3. Data Splitting and Training

To begin the implementation, we first need to define the Spark Context for the program. As explained in the earlier section, it is the main entry point for Spark functionality [34] and must be specified before beginning to create RDDs. We specify three parameters for the Spark Context. They are the application name, cluster URL, and the number of cores. Application name should be a meaningful name defining the purpose of the program. Cluster URL is the URL for the cluster we want to connect to. For a local cluster, it is specified by the keyword "local". The number of cores specifies the number of worker nodes that will be created. As mentioned in an earlier section, worker nodes does the processing work in Spark [35]. We range the core values from two to four in this study. This means that the number of worker nodes for each algorithm for each dataset will span from 2 to 4. We use the same settings for the Local cluster and the GPU cluster mode. Due to technical limitations, the core value could not be extended beyond 4. Next, we input the desired dataset in libsvm format into input RDD using the "loadlibSVMFile" method of the MLUtils class. The MLUtils class defines helper methods to load, save, and pre-process data used in MLlib [55]. We now have the dataset loaded into an initial input RDD.

Next, we split the input RDD into training and testing RDD with split ratio of 75/25 and a random seed value. We now train the model with the training set using the “train” method and specifying the various parameters available for the five supervised machine learning algorithms (Naive-Bayes, Decision Tree, Random Forest, Support Vector Machine and Logistic Regression). The explanation and respective values used in algorithms are presented in Table 1-5.

Table 1. Parameters used for Naïve-Bayes Classification algorithm [56]

Parameter	**Explanation**	**Value used**
lambda	the smoothening parameter (default 1.0)	1.0

Table 2. Parameters used for Decision Tree Classification algorithm [57]

Parameter	**Explanation**	**Value used**
numClasses	number of classes for classification	2
categoricalFeaturesInfo	map storing arity of categorical features ({} for none)	{}
impurity	criterion used for information gain selection (gini or entropy)	entropy
maxDepth	maximum depth of tree (default 5)	6
maxBins	number of bins used for finding splits at each node (default 32)	32
minInfoGain	minimum info gain required to create a split (default 0.0)	0.15

Table 3. Parameters used for Random Forest Classification algorithm [58]

Parameter	Explanation	Value used
numClasses	number of classes for classification	2
categoricalFeaturesInfo	map storing arity of categorical features ({} for none)	{}
impurity	criterion used for information gain selec-tion (gini or entropy)	entropy
featureSubsetStrategy	number of features to consider for spilts at each node	auto
maxDepth	maximum depth of tree (default 5)	6
numTrees	number of trees in the random forest	5
maxBins	number of bins used for finding splits at each node (default 32)	32

Table 4. Parameters used for SVM Classification algorithm [59]

Parameter	Explanation	Value used
iterations	number of iterations	1000
numClasses	number of classes (default 2)	2
validateData	algorithm should validate the data before training	TRUE

Table 5. Parameters used for Logistic Regression algorithm [60]

Parameter	Explanation	Value used
iterations	number of iterations	1000
numClasses	number of classes (default 2)	2
validateData	algorithm should validate the data for singularity before training	TRUE

The next step is to test the trained model on the testing set. We do so by using the “predict” method. The predict method is executed for each row of the test set using the “map” transformation of Spark.

3.4. Performance Metrics

We now move to evaluate the performance of the model and calculate the results for the experiments. For supervised classification problems, the basic comparison is to match the true class label with the predicted one to get the accuracy. The result of a data point can be a True Positive, TP (label is positive and prediction is positive too), False Positive, FP (label is negative but prediction is positive), True Negative, TN (label is negative and prediction is negative too) and False Negative, FN (label is positive but prediction is negative). In addition to evaluating pure accuracy, area under ROC (Receiver Operating Characteristic) and area under Precision-Recall (PR) are considered as well. ROC is a plot of the true positive rate (TPR) versus the false positive rate (FPR) for every possible classification threshold [61]. The true positive rate is given by Equation 8, and the False Positive Rate is given by Equation 9.

$$TPR = \frac{TP}{TP + FN} \tag{8}$$

$$FPR = \frac{FP}{FP + TN} \tag{9}$$

The accuracy of the curve depends on how well the classifier can distinguish between the binary classes and is measured by the area under ROC. Area under ROC is given by Equation 10.

$$AUROC = \int_0^1 \frac{TP}{P} d(\frac{FP}{N}) \tag{10}$$

The value for the area ranges from 0 to 1. The more the area under ROC, the better the prediction.

PR is a plot of Precision against Recall. Precision is given by Equation 11, and Recall is given by Equation 12.

$$P = \frac{TP}{TP + FP} \tag{11}$$

$$R = \frac{TP}{TP + FN} \tag{12}$$

The accuracy of the curve depends on how well the classifier can distinguish between the unbalanced binary classes and is measured by the area under PR. Area under PR is given by Equation 13.

$$AUPRC = \int_0^1 \frac{TP}{TP + FP} d(\frac{TP}{P}) \tag{13}$$

Since PR does not take into account TN, it is better for evaluating the performance of the model. The value for the area ranges from 0 to 1. The more the area under PR, the better the prediction [61][62].

We first calculate the accuracy for the predictions on the test set by comparing the true class label for the test set with the predicted one. This is accomplished by using the “filter” action of Spark. As mentioned in Chapter 3, the transformations in Spark are lazy, that is they are not evaluated until an action is performed on them. So, when we apply the “filter” action on the test set, it is at this step the actual execution of the “predict” transformation takes place. We calculate the accuracy as the number of true positive and negative predicted by the model. We then calculate the area under ROC and area under PR for the predicted set using “areaUnderROC” and “areaUnderPR” method from the Binary Classification Metrics class. The Binary Classification Metrics class is a binary evaluator class available in Spark that evaluates performance such as area under ROC, area under PR, f measure etc. For the Python implementation, only the area under ROC and PR are available in Spark. We also report the tree structure for Decision Tree and Random Forest algorithms.

4. EXPERIMENTS

The experiments follow the steps as presented under the Methodology and are implemented using two modes of cluster operation in Apache Spark. These two modes of operation are namely the Local Cluster mode and the GPU cluster mode. The Local cluster mode is prepared on a personal laptop and the GPU cluster mode is run on a Nvidia Tesla K40 available through remote connection. The Local cluster has an Intel Core i7 processor with 12 GB RAM and 4GB Nvidia GEForce 940MX graphics card. The GPU cluster has 12GB of global memory with 2,880 stream processors and a memory bandwidth of 288GB/sec. We have utilized an incremental number of cores in both modes, ranging from 2 to 4. Next, we prepared the environment by installing the latest versions of Java, Python, Anaconda, PySpark and Spark and setting the path variables for the systems. We may point out at this time that these steps were followed for the Local Cluster laptop whereas the remote GPU cluster already had the required software installed with only the path variables not set.

After successful installation and completing other prerequisites, we moved to run experiments on a IRIS dataset in the Local Cluster mode. The IRIS dataset is a well-known dataset “IRIS Flower” which is often used for applying statistical classification techniques in machine learning [63]. The dataset is publicly available and is acquired from the UCI Machine Learning Repository [64]. The dataset contains 50 instances for each of the 3 classes and 4 features. These classes are the 3 species of Iris (Iris setosa, Iris virginica and Iris versicolor) and the features are length and width of the sepals and petals in cm [58]. For simplicity, we convert the categorical names of classes in the IRIS dataset to numerical classes represented as 1 for Iris Setosa, 2 for Iris Virginica, and 3 for Iris Versicolor. We plot boxplots and perform initial descriptive statistics on the dataset to understand the type of data, its range and various features.

We also generate QQplots for the dataset to know the distribution of data. If the data follows the normal distribution, we move onto the next steps in the process. If it does not, we apply Box Cox transformation to generate a normally distributed dataset. Next, we convert the dataset into the LabeledPoint format as required for further processing.

We implement Naive-Bayes classification on the IRIS dataset for testing purposes. The steps are followed according to the process described in Methodology. We start first by setting up the Spark Context for the experiment which serves as a connection to the Spark cluster [34]. We set up the cluster URL to connect to as “localhost”, number of data nodes to utilize as “2”, and application name for our application as “IRIS example”. We load the dataset in libsvm format into an input RDD and split the dataset into 75% training, and 25% testing RDD. We set the various parameters available with the model. Here, for the Naive-Bayes training model we have 2 parameters, namely the training RDD and smoothening factor Lambda. We set the smoothening factor Lambda to the default value 1.0. After the model is trained, we use it to predict the class labels for the testing RDD. We use the “map” transformation for RDD to spread the prediction to all rows of the testing RDD. We then calculate the accuracy of the predictions by applying “filter” and “count” actions on the predicted RDD. We run other metrics on the predicted RDD to calculate Area under ROC and Area under PR using Binary Classification Metrics library. Note here, the input and output data is in form of a RDD in every step. Similar to Naive-Bayes, there are different parameters available in all algorithms and they vary with datasets used as listed in Chapter 3 Table 1-5.

Following the same approach as with the IRIS example, we move on to implement the five machine learning algorithms on all 3 datasets (HIGGS, SUSY and Hepmass) using the two modes of cluster operation. These algorithms follow the same high-level steps as with the above example with IRIS dataset. The results and conclusions for the same in next chapter.

5. RESULTS

In this chapter, we present the results of our experiments and discuss the performance comparison between the two modes of cluster operations in Spark.

5.1. Descriptive Statistics

In descriptive statistics, we first try to understand the data by calculating the various summary measures, such as mean, maximum value, minimum value, non-zero values and variance, for all features. We begin by reporting the summary measures for the IRIS dataset in Table 6.

Table 6. Summary measures for IRIS dataset

	Feature 1	**Feature 2**	**Feature 3**	**Feature 4**
Mean	5.843	3.054	3.759	1.199
Maximum Value	7.9	4.4	6.9	2.5
Minimum Value	4.3	2.0	1.0	0.1
Non-zero Values	150	150	150	150

From Table 6, we can observe that the range for feature 1 of the IRIS dataset is 4.3-7.9, feature 2 is 2.0-4.4, feature 3 is 1.0-6.9, and feature 4 is 0.1-2.5. There are no zero values in the dataset and the mean for feature 1 is 5.843, feature 2 is 3.054, feature 3 is 3.759, and feature 4 is 1.199. The mean lies well within the median range of the features indicating a normal distribution of the data.

Figure 4 presents the boxplots for the IRIS dataset to visually understand the summary statistics.

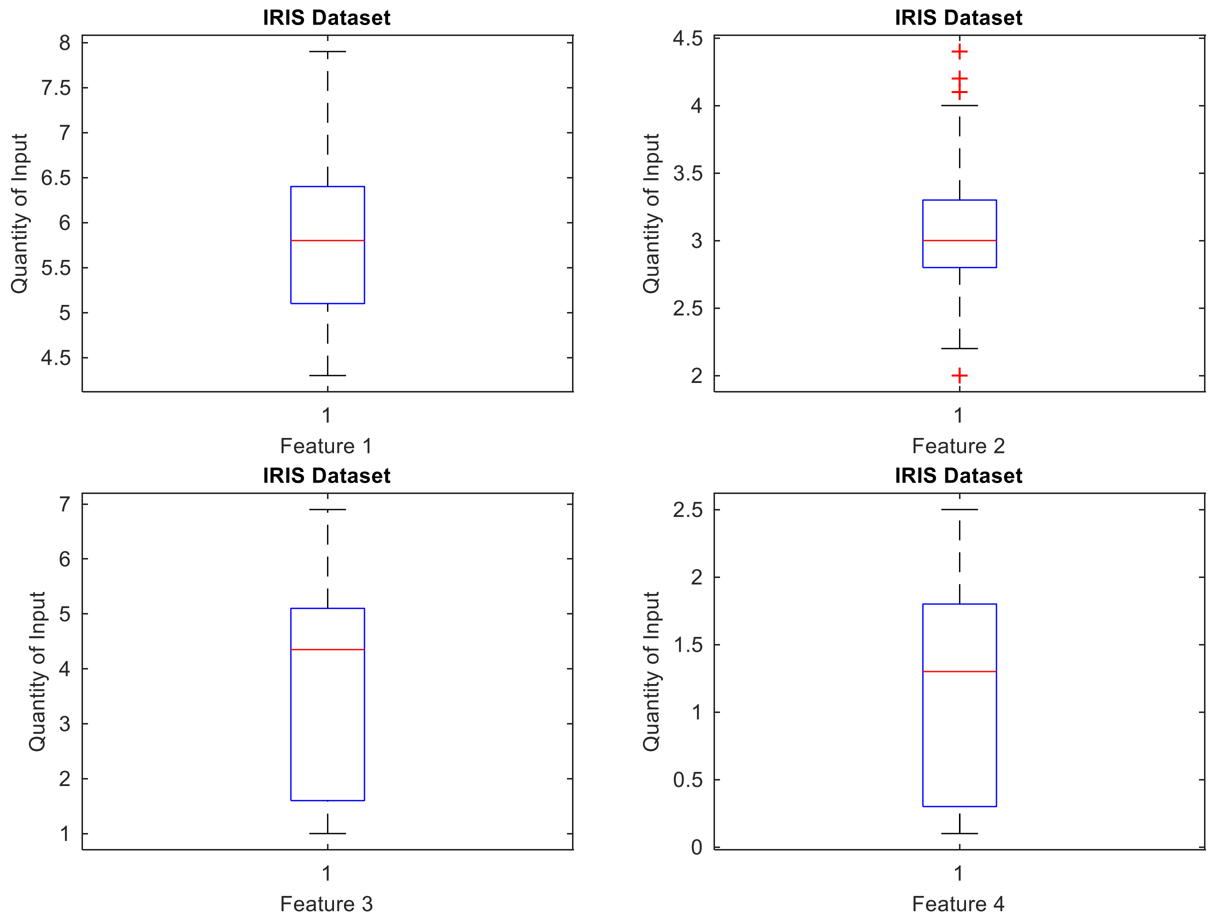


Figure 4. Boxplots for IRIS dataset

It is observed from Figure 4 that the mean for feature 1 lies in the median range indicating uniform values present in the data. The mean for feature 2 lies in the lower range indicating more number of lower values in the data. The mean for feature 3 and 4 lies in the upper range indicating there are higher values in the data. There are some outliers in feature 2 which are removed to gain an uniform distribution. There are no outliers in feature 1, 2 and 4.

Figure 5 presents the QQplots for the IRIS dataset to understand the distribution of data.

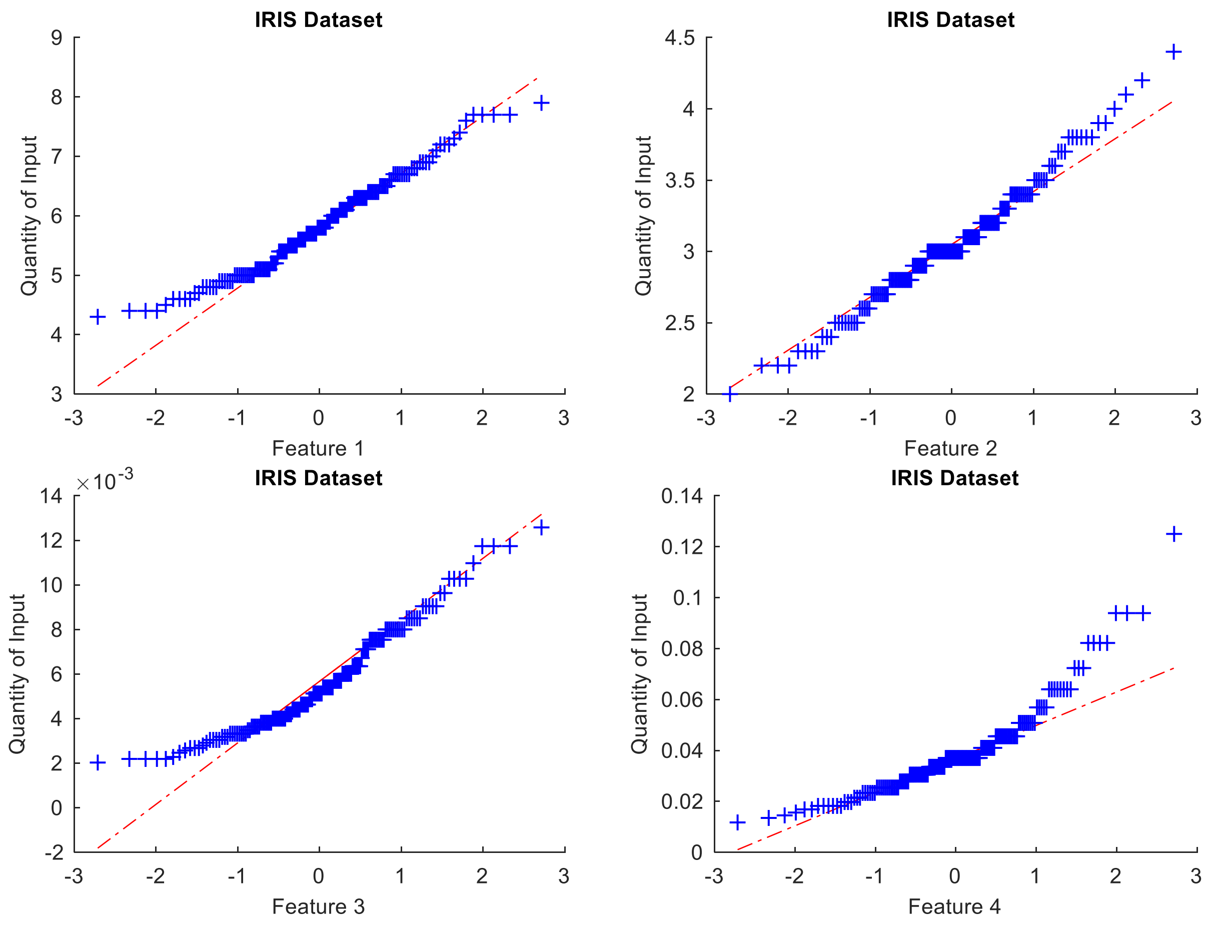


Figure 5. QQplots for IRIS dataset

It is observed from Figure 2 that all the features of IRIS dataset follow a normal distribution. Initially, feature 2 of the IRIS dataset did not have normal distribution. After applying Box-Cox transformation to it, we can observe that it now shows a normal pattern.

We follow the same procedure to report descriptive statistics for our three datasets, HIGGS, SUSY and Hepmass. We observed the results for all the datasets but explain the results for the SUSY dataset only here for the sake of brevity. The same implications can be extended to the other two datasets. The summary measures for the SUSY dataset is presented in Table 7.

Table 7. Summary measures for SUSY dataset

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6
Mean	1.003	2.192e-05	-4.994e-05	9.994e-01	-3.713e-05	-1.972e-05
Maximum Value	20.553	2.101	1.734	33.035	2.059	1.734
Minimum Value	2.548e-01	-2.102	-1.734	4.285e-01	-2.059	-1.734
Non-zero Values	5000000	5000000	5000000	5000000	5000000	5000000

	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11	Feature 12
Mean	9.997e-01	3.542e-05	1.001	-4.878e-05	1.003	9.995e-01
Maximum Value	21.068	1.746	23.386	20.487	21.075	16.168
Minimum Value	2.259e-04	-1.727	7.693e-08	-16.718	2.673e-01	1.041e-03
Non-zero Values	5000000	5000000	5000000	5000000	5000000	5000000

	Feature 13	Feature 14	Feature 15	Feature 16	Feature 17	Feature 18
Mean	9.991e-01	1.004	1.001	1.001	9.994e-01	2.249e-01
Maximum Value	6.731	20.686	21.152	15.613	1.596	1.0
Minimum Value	2.048e-03	0.00	2.734e-02	4.452e-03	3.211e-07	4.172e-08
Non-zero Values	5000000	3938127	5000000	5000000	5000000	5000000

From Table 7, we can observe that all features except feature 14 have no zero values. There are some features like feature 1, 4, 7, 9, 10, 11, 12, 13, 14, 15, 16 and 18 that show a wide range in their set of values indicating continuous set of values. Hence, we can expect these features to be normally distributed and if not, we can use transformations to convert them to normal distribution. The data in features 2, 3, 5, 6, 8 and 17 exhibit discrete behavior implying that most of the data points in these features will lie towards the maximum or minimum value. These features will not be normally distributed and we cannot use transformations to convert them either.

Figure 6 presents the boxplots for the SUSY dataset. From the figure we observe the mean for all features except 14 and 17 lie in the median range of values indicating uniform distribution of values. It lies in the lower range for both the features indicating more number of lower values which is evident from the large number of outliers in lower range for feature 14. There is a large number of outliers in feature 1, 4, 7, 9, 10, 11, 12, 13, 14, 15, 16 and 18. We remove these outliers to gain a uniform distribution. There are many methods available for outlier detection like Z-score, Principal Component Analysis, etc. but this process is out of scope of this study.

Figure 7 presents the QQplots for the SUSY dataset. In Figure 7, features 2, 3, 5, 6, 8 and 17 show slightly skewed-looking step pattern. This is in agreement with the earlier observation that the data points in these values lie mostly towards the extreme ends. These features cannot be normalized. Features 1, 4, 7, 9, 11, 12, 13, 15, 16 and 18 show a normal distribution. Initially, these features were not normalized due to a large number of outliers. We applied the Box-Cox transformation to normalize them. Feature 10 exhibits a skewed distribution in extreme ends and feature 14 is skewed only in the lower end. There was no effect of transformation on these features.

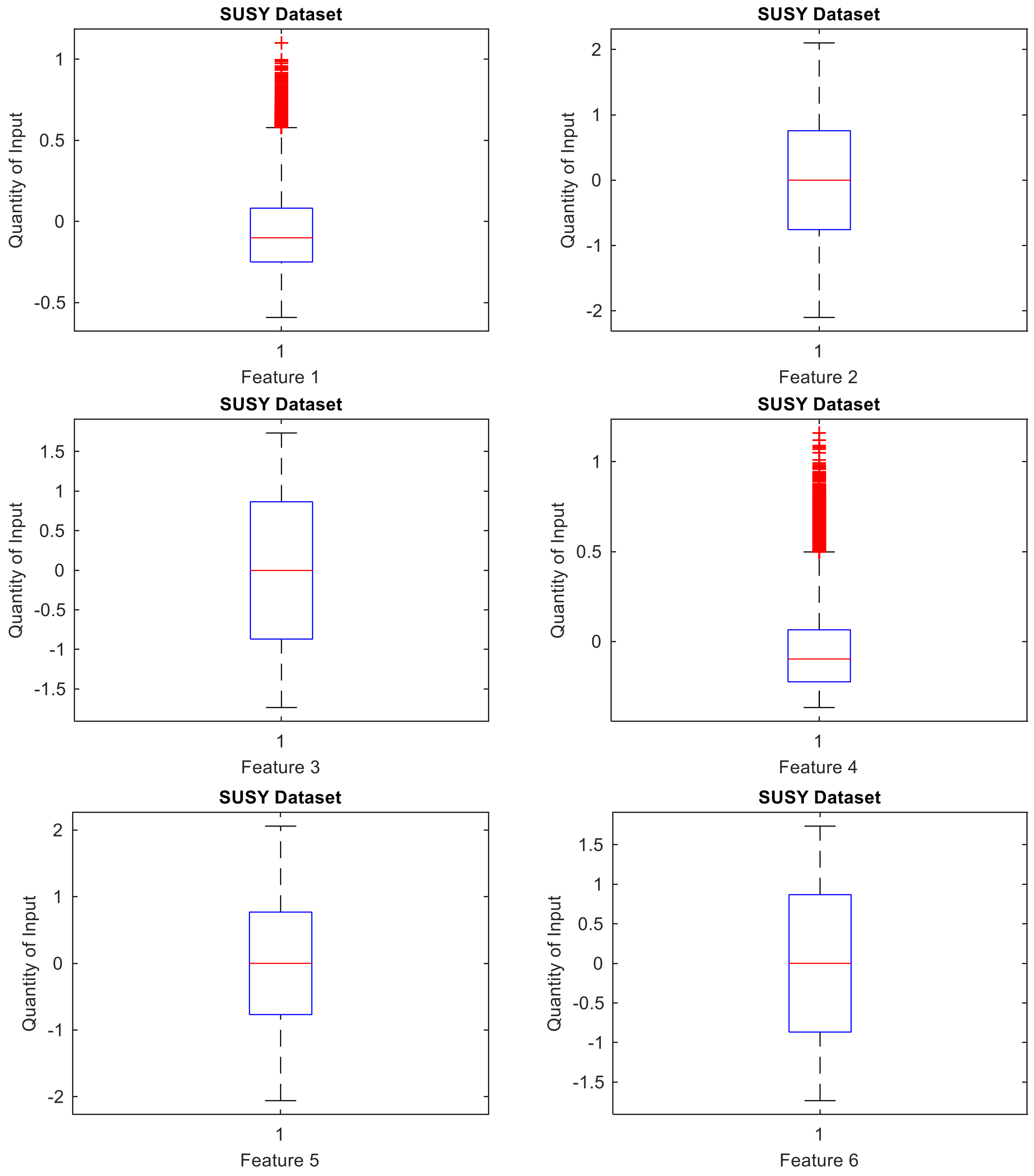


Figure 6. Boxplots for SUSY dataset

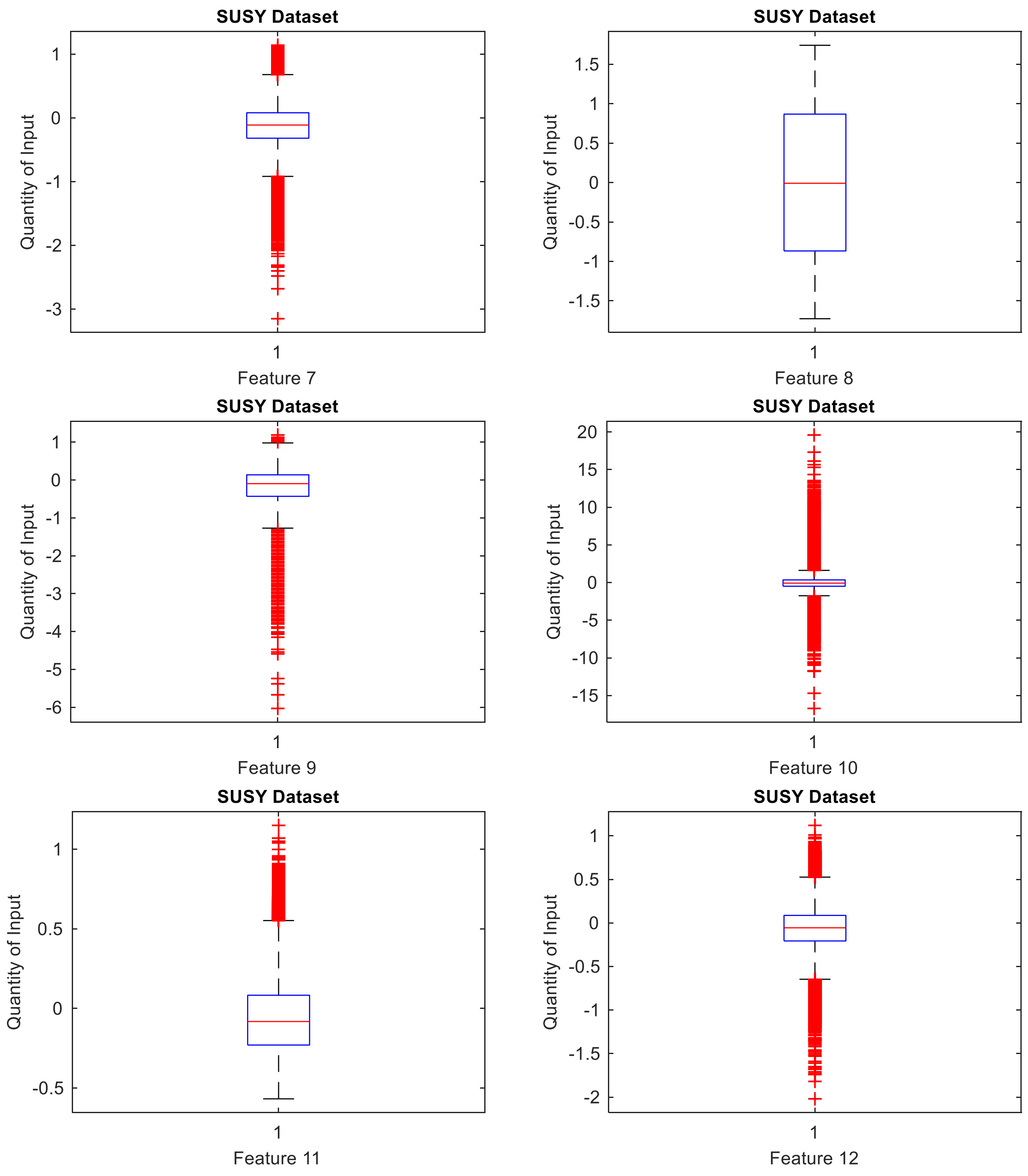


Figure 6. Boxplots for SUSY dataset (continued)

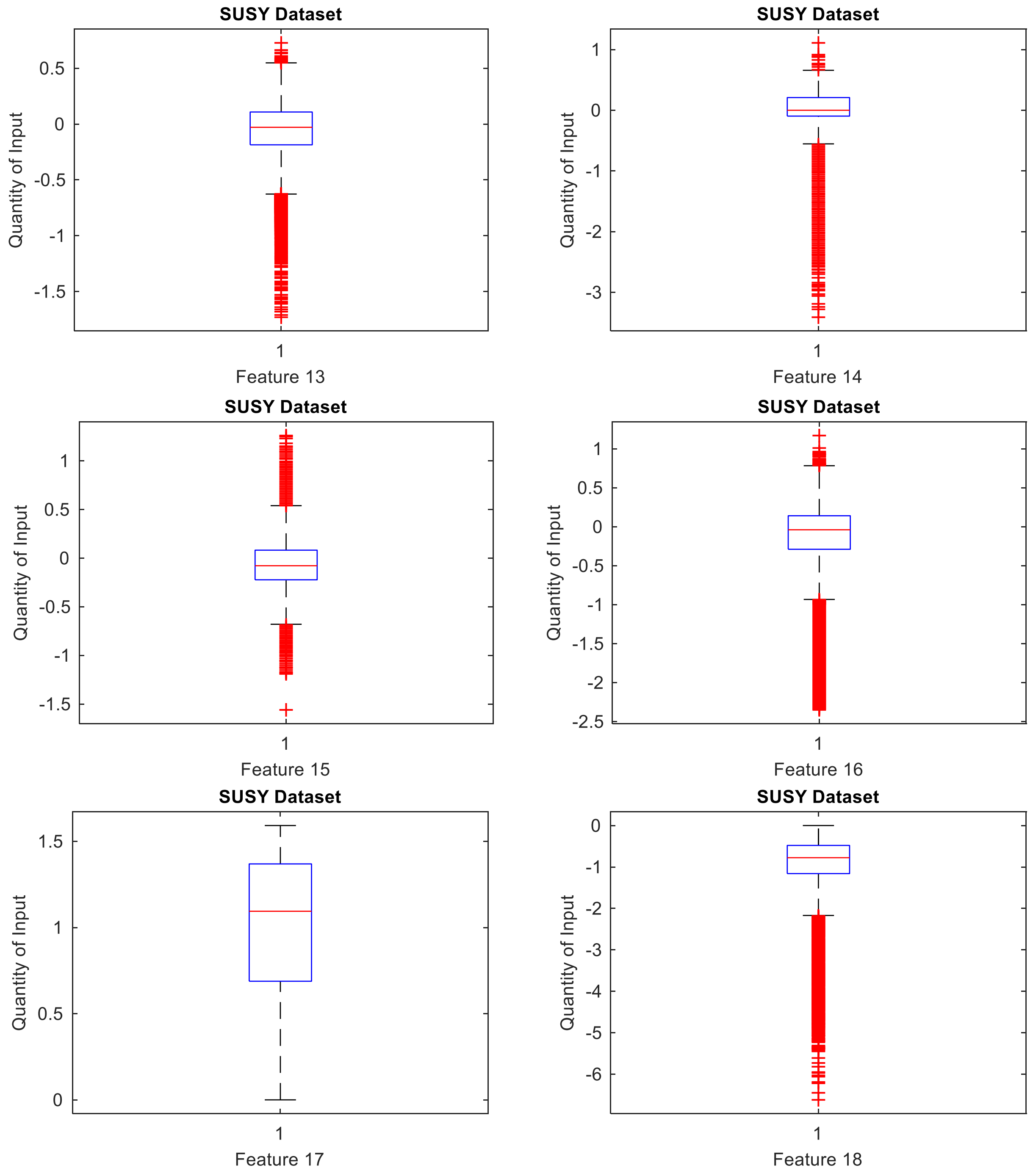


Figure 6. Boxplots for SUSY dataset (continued)

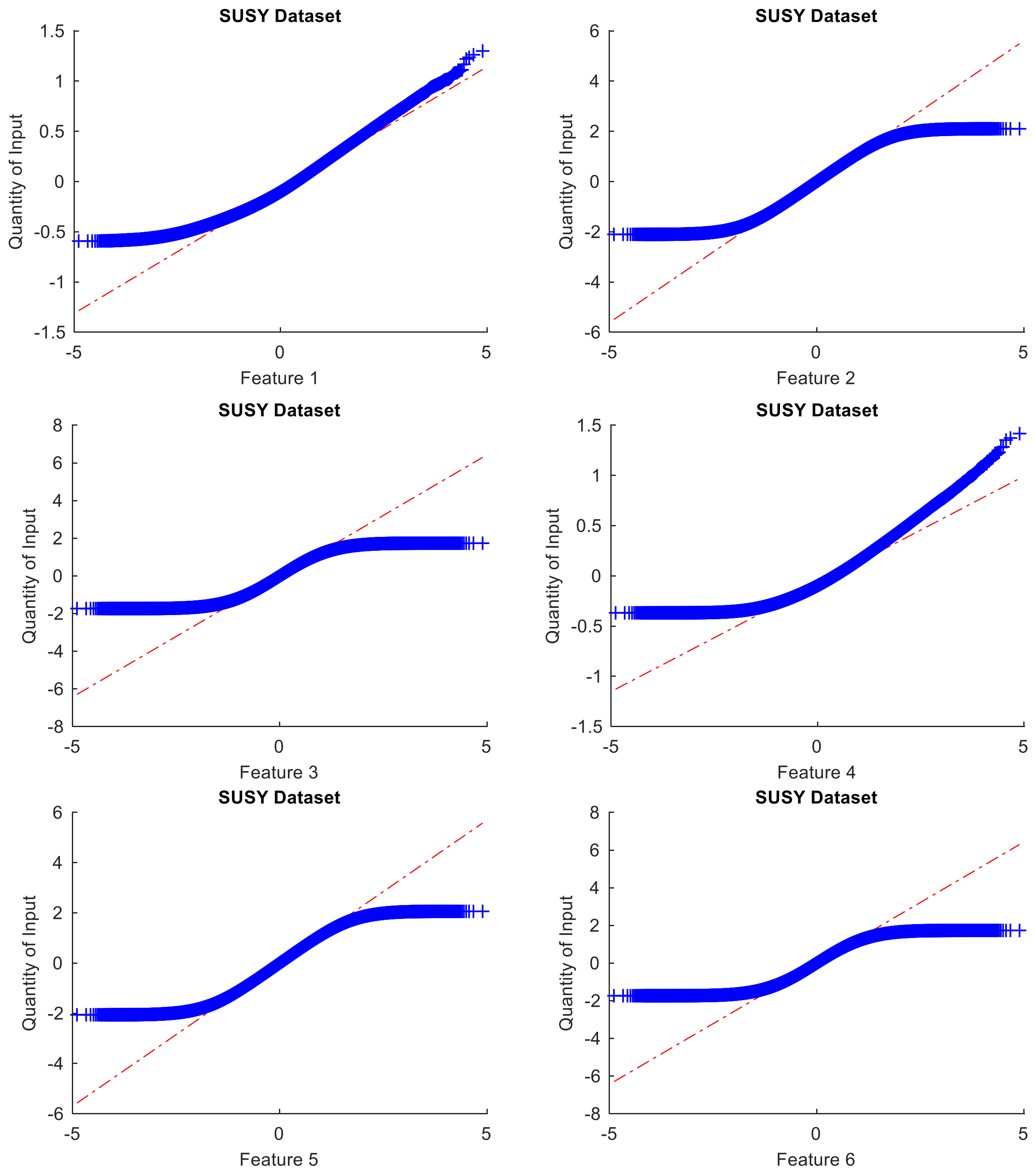


Figure 7. QQplot for SUSY dataset

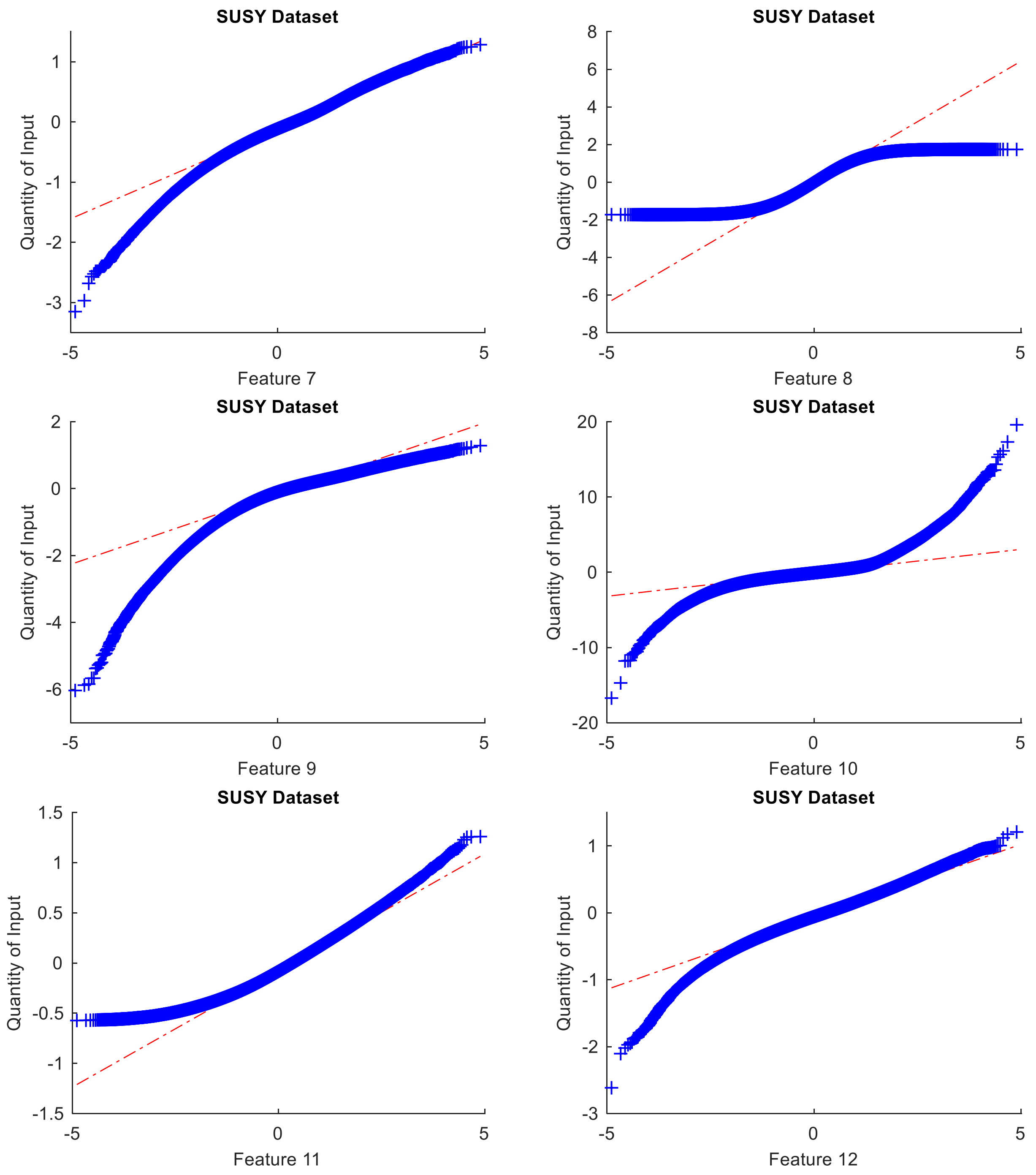


Figure 7. QQplot for SUSY dataset (continued)

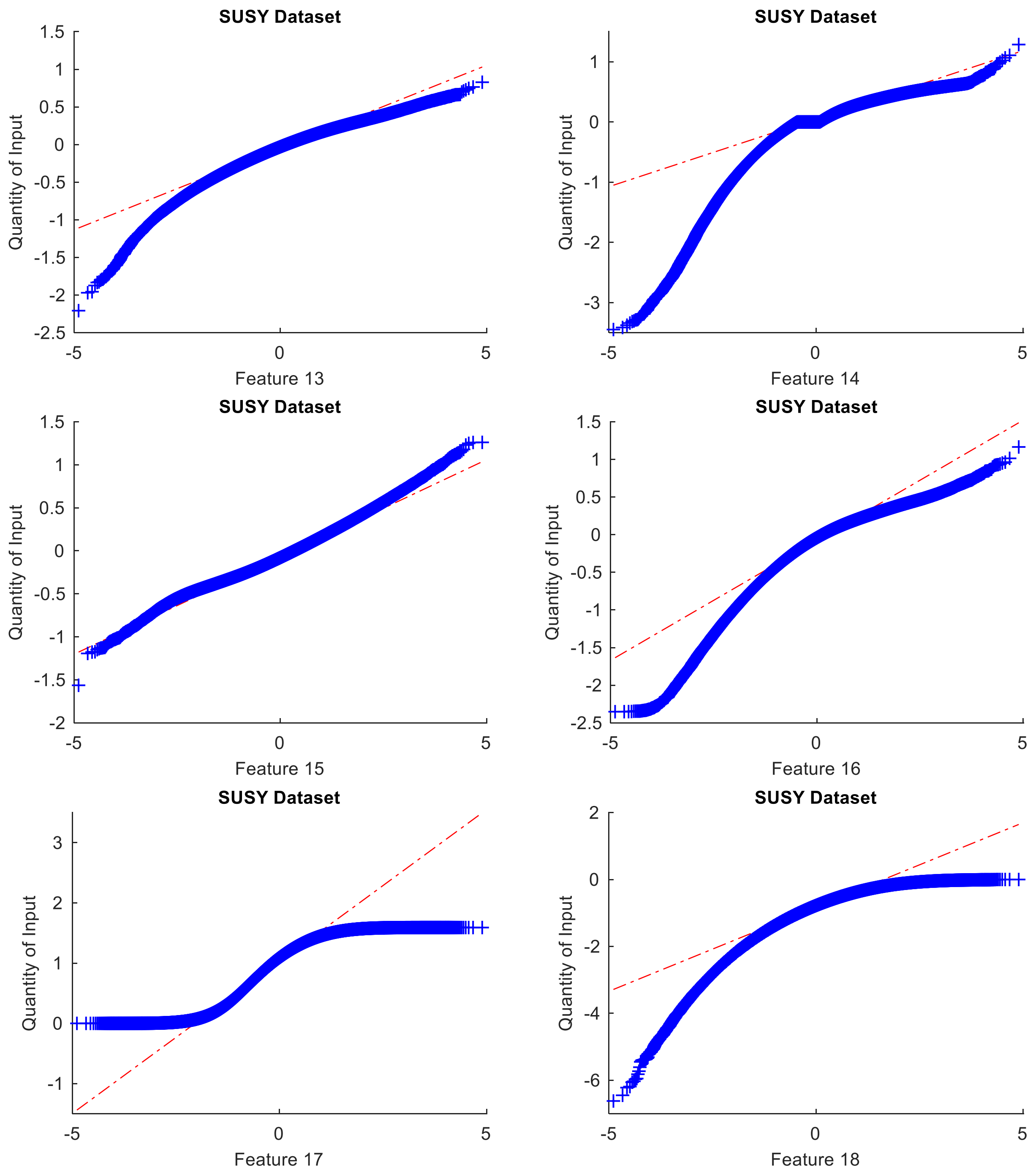


Figure 7. QQplot for SUSY dataset (continued)

5.2. Performance Metrics

Once we obtain the descriptive statistics, we convert the IRIS dataset into the libsvm format. Post the conversion, data is then fed to the Naïve-Bayes Classification algorithm and we obtain the performance measures in the Local Cluster mode. These performance measures are accuracy, time taken, area under ROC and area under PR. We repeat the process to obtain the results in the GPU Cluster mode. Table 8 presents the metrics of the Naïve Bayes algorithm for the IRIS dataset in the Local Cluster mode, and Table 9 presents the results for the GPU Cluster mode.

Table 8. Performance metric of Naïve Bayes algorithm in Local Cluster mode for different nodes for IRIS dataset

	Time taken (seconds)	**Accuracy (%)**	**Area under ROC**	**Area under PR**
Node 2	20	87.87	0.857	0.51
Node 3	17	87.65	0.861	0.62
Node 4	16	87.69	0.872	0.68

Table 9. Performance metric of Naïve Bayes algorithm in GPU Cluster mode for different nodes for IRIS dataset

	Time taken (seconds)	**Accuracy (%)**	**Area under ROC**	**Area under PR**
Node 2	12	87.89	0.861	0.54
Node 3	9	87.56	0.863	0.56
Node 4	7	87.65	0.871	0.61

We now compare the results from both modes to understand the behavior of the Naïve-Bayes algorithm in Apache Spark. From Table 8 we observe the time taken by the algorithm in the Local Cluster mode decreases with increasing number of cores though the reduction is not significant. This is because the Local cluster mode is essentially a single machine creating partitions to give the impression of parallel computing. We can observe from Table 9 that the time taken by the GPU mode has halved with increased number of cores. This emphasizes that the clusters are working in parallel to reduce the computation time. The accuracy, area under ROC and area under PR do not show much improvement in both modes for different cores. This can be attributed to the fact that the algorithm is implemented in same fashion in both modes. Increasing the number of nodes will affect the computation time but not the manner in which the algorithm is implemented.

We follow the same process as we did with the IRIS dataset to apply the five machine learning algorithms on our three datasets (HIGGS, SUSY and Hepmass) and obtain the performance measures for all of them in the Local and the GPU Cluster mode. We collect the performance of the two modes in terms of time taken, accuracy, area under ROC and area under PR for all the three datasets. However, we discuss our findings for the SUSY dataset for brevity and the same implications can be extended to other datasets as well. We present the time taken by the different machine learning algorithms in the Local Cluster mode for SUSY dataset using different number of nodes in Table 10, and in GPU Cluster mode in Table 11.

First, we begin by explaining the real, sys and user time. Real-time is the time taken by the script to run, User time is the time cores are involved in computing and sys time is the time required for input/output. We observe that the user time is more than the real time in both modes for SVM and Logistic Regression since these are computation intensive algorithms. This indicates the computation of algorithms in a parallel fashion. The higher the user time, the more cores are involved in the parallel execution. It increases with increasing number of cores for

SVM implying the cores are being utilized for execution. We observe the time taken by all algorithms decreases with the increase in number of cores in both modes. The time taken by the algorithms in Local mode is much more than in GPU mode. This reinforces the fact that the Local mode takes more time due to the usage of a single machine. Increasing the number of cores in the Local mode is almost equivalent to parallel processes executing on single machine. It gives an illusion of parallelism. Although the decrease is more significant in the GPU mode in comparison to the Local mode, we observe the average decrease in computation time per node is around 20% for all algorithms except for Logistic Regression, which shows a 50% decrease in computation time. All the algorithms can inherently be computed in parallel and show significant decrease in time with increasing nodes.

We illustrate the time taken by the algorithms for the SUSY dataset in both modes in Figure 8. As pointed out, we can observe the time taken by the algorithms decreases with an increase in the number of cores. This is because the algorithms use more parallel computing with an increase in the number of cores. It can be observed that the time taken by SVM and Decision Tree in both modes show considerable difference for each node added. For Logistic Regression, the difference between times taken in both modes to complete becomes more defined as the nodes are increased. This behavior is less defined for the Random Forest algorithm where the time taken in both modes becomes less pronounced with an increase in the number of nodes. The time taken by the Naïve Bayes algorithm is almost comparable in both modes. Overall, the GPU Cluster mode shows a significant decrease in time taken by algorithms as compared to the Local Cluster mode, further strengthening the claim that parallel computing using Apache Spark has involved more cores and reduced the computing time.

Table 10. Time taken (in minutes) by machine learning algorithms in Local Cluster mode using different nodes for SUSY dataset

		Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
	Real	15.36	19.29	18.47	28.52	29.56
Node 2	**User**	2.56	8.56	9.58	35.05	32.15
	Sys	0.56	0.25	0.56	0.56	0.39
	Real	13.59	17.29	15.24	26.52	25.45
Node 3	**User**	2.46	7.69	7.45	36.41	30.15
	Sys	0.55	0.26	0.48	0.39	0.29
	Real	12.10	14.26	12.12	25.14	20.15
Node 4	**User**	2.47	7.69	7.32	36.12	29.85
	Sys	0.55	0.25	0.48	0.35	0.33

Table 11. Time taken (in minutes) by machine learning algorithms in GPU Cluster mode using different nodes for SUSY dataset

		Naïve Bayes	Decision Tree	Random Forest	SVM	Logistic Regression
	Real	12.15	13.53	12.39	14.14	26.34
Node 2	**User**	1.35	3.31	4.29	28.56	29.53
	Sys	0.60	0.11	0.15	0.29	0.29
	Real	11.20	11.21	10.47	13.34	15.45
Node 3	**User**	1.47	3.15	4.25	30.02	17.01
	Sys	0.70	0.22	0.5	0.22	0.22
	Real	9.49	9.11	10.8	11.51	13.80
Node 4	**User**	1.51	3.34	4.35	31.20	26.37
	Sys	0.70	0.33	0.26	0.24	0.23

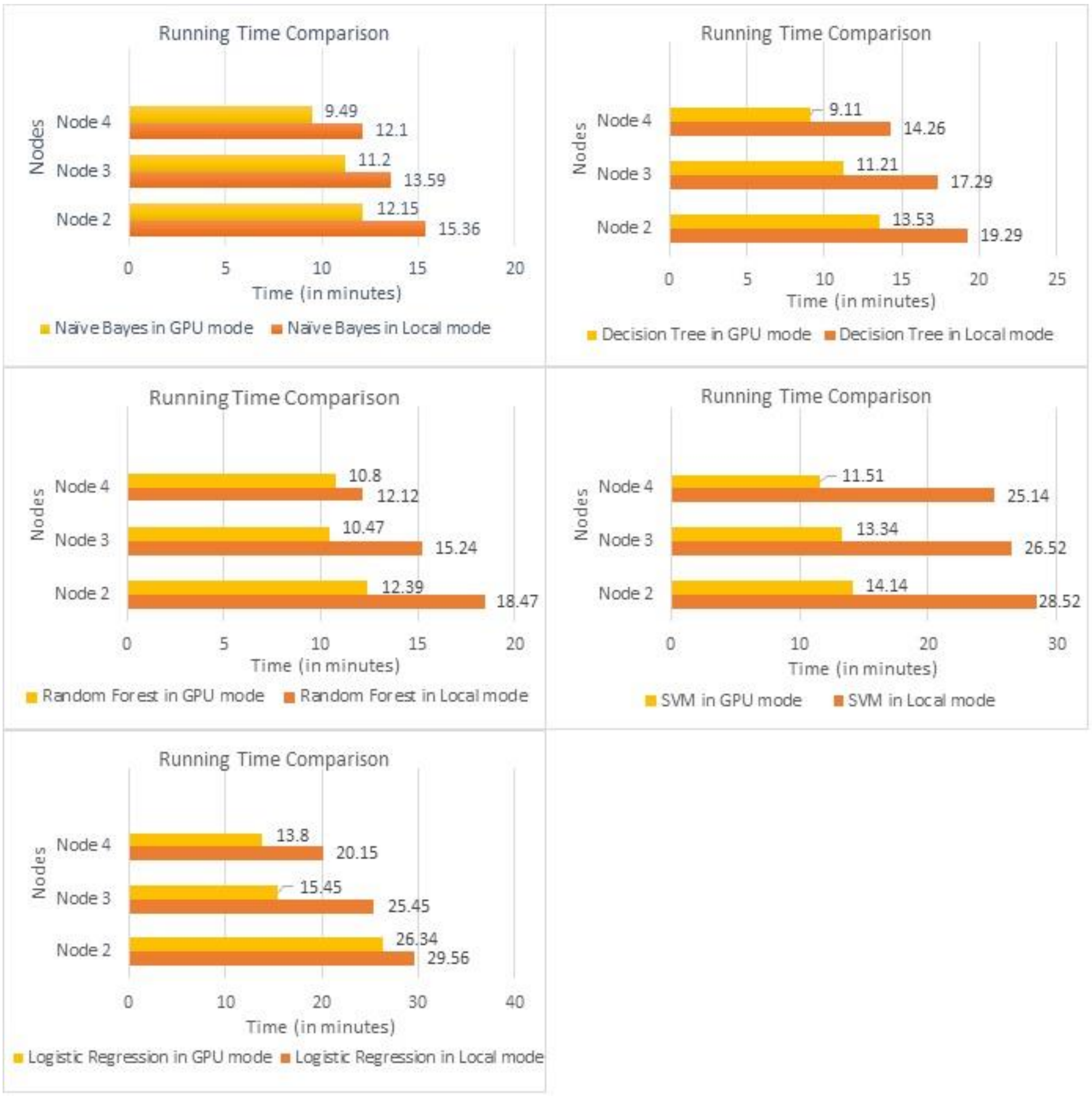


Figure 8. Running Time Comparison of algorithms for SUSY dataset

Table 12, 14 and 16 present the accuracy, area under ROC and area under PR acquired for the different machine learning algorithms for the SUSY dataset in the Local Cluster mode using different nodes respectively, and Table 13, 15 and 17 presents the results of the same metrics for the GPU Cluster mode.

Table 12. Accuracy (%) of machine learning algorithms in Local Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	74.89	74.05	77.69	76.54	78.48
Node 3	75.01	74.09	77.68	76.53	78.53
Node 4	74.98	74.04	77.71	76.53	78.54

Table 13. Accuracy (%) of machine learning algorithms in GPU Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	75.66	74.43	77.71	76.55	78.63
Node 3	75.65	74.43	77.95	76.54	78.62
Node 4	75.66	74.42	77.95	76.55	78.63

Table 14. Area under ROC of machine learning algorithms in Local Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.744	0.743	0.785	0.792	0.790
Node 3	0.741	0.742	0.785	0.792	0.782
Node 4	0.742	0.743	0.784	0.791	0.781

Table 15. Area under ROC of machine learning algorithms in GPU Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.752	0.750	0.785	0.800	0.788
Node 3	0.753	0.755	0.791	0.805	0.792
Node 4	0.741	0.749	0.792	0.802	0.791

Table 16. Area under PR of machine learning algorithms in Local Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.745	0.714	0.645	0.561	0.650
Node 3	0.752	0.719	0.646	0.562	0.649
Node 4	0.752	0.720	0.644	0.566	0.647

Table 17. Area under PR of machine learning algorithms in GPU Cluster mode using different nodes for SUSY dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.750	0.720	0.652	0.570	0.657
Node 3	0.749	0.724	0.652	0.579	0.657
Node 4	0.752	0.719	0.658	0.600	0.661

We observe that the results of accuracy, area under ROC and area under PR do not show much change in both modes. As explained for the IRIS dataset, this reiterates the fact that an increased number of nodes and parallel computation play no major role in improving the accuracy and other measures. Accuracy in itself comes out to an average of 80% which can be explained by the presence of some non-normal attributes in the dataset. Area under ROC and PR have higher values (more than 0.70), which shows confidence in the classification.

6. CONCLUSION AND FUTURE WORK

In this study, an attempt is made to compare the performance of two modes of clusters available in Apache Spark. We made this comparison by implementing five machine learning supervised classification algorithms using the Apache Spark Machine Learning library. We implement these algorithms on large datasets to ensure the parallel computing abilities of Apache Spark are utilized. We compare the performance of the algorithms in terms of time taken, accuracy, area under ROC and area under PR. The following are the conclusions from the study:

1. Both modes use the specified number of cores for distributed and parallel computing. This is evident from the increased user time when the actual execution time is much less. An increased user time indicates the use of cores for parallel computation.
2. The use of a larger number of cores in the Local Cluster mode decreases the running time of the algorithms but it is not significant. This is attributed to the Local mode using a single machine creating an impression of parallelism.
3. The reduction in running time of algorithms with an increased number of cores in the GPU Cluster mode is significant. It is faster than the Local Cluster mode which strengthens the fact that the use of the varied number of cores in clusters ensures less running time than a single machine with the varied number of cores. With more cores in a cluster, more parallelism is achieved in computing the predictions and running time of the algorithms decreases by 20% on average.
4. Other results of the performance metrics such as accuracy, area under ROC and area under PR show no significant change between the two modes. This indicates increasing number of cores do not affect the internal implementation of the machine learning algorithms. Although the predictions are computed faster but the inherent manner to

compute them remains the same and yields almost the same result every time. A higher accuracy can be achieved with more normally distributed datasets.

As with regards to future work, Apache Spark is originally developed in the Scala programming language and works most efficiently. It has extended APIs for Python, Java, and R from which we have used Python. Although the usage of these APIs does not convey degraded performance, it is better to point out that these APIs are interpreted into Scala by Spark for further computation. So, the overall running time of algorithms in Scala is slightly better than in Python. In addition, we used the remote connection to connect to the GPU. This limited the study to use a maximum of four cores. The connection resulted in a timeout when we tried to use more cores.

REFERENCES

[1] M. Parashar, X. Li, and S. Chandra, “Advanced computational infrastructures for parallel and distributed applications”. John Wiley & Sons, 2010, vol. 66.

[2] J. Archenaa and E. M. Anita, “Interactive big data management in healthcare using spark,” in Proceedings of the 3rd International Symposium on Big Data and Cloud Computing Challenges (ISBCC–16). Springer, 2016, pp. 265–272.

[3] R. Fang, S. Pouyanfar, Y. Yang, S.-C. Chen, and S. Iyengar, “Computational health informatics in the big data age: a survey,” ACM Computing Surveys (CSUR), vol. 49, no. 1, p. 12, 2016.

[4] Z. Lv, J. Chirivella, and P. Gagliardo, “Bigdata oriented multimedia mobile health applications,” Journal of medical systems, vol. 40, no. 5, p. 120, 2016.

[5] M. S. Wiewiorka, A. Messina, A. Pacholewska, S. Maffioletti, P. Gawrysiak, and M. J. Okoniewski, “Sparkseq: fast, scalable, cloudready tool for the interactive genomic data analysis with nucleotide precision,” Bioinformatics, p. btu343, 2014.

[6] D. Ding, D. Wu, and F. Yu, “An overview on cloud computing platform spark for human genome mining,” in Mechatronics and Automation (ICMA), 2016 IEEE International Conference on. IEEE, 2016, pp. 2605–2610.

[7] A. Garcıa-Pablos, M. Cuadros, and G. Rigau, “V3: Unsupervised aspect based sentiment analysis for semeval-2015 task 12,” SemEval-2015, p. 714, 2015.

[8] J. Ryan, “Rapidminer for text analytic fundamentals,” Text Mining and Visualization: Case Studies Using Open-Source Tools, vol. 40, p. 1, 2016.

[9] H. Ji, S. H. Weinberg, M. Li, J. Wang, and Y. Li, “An apache spark implementation of block power method for computing dominant eigenvalues and eigenvectors of large-scale matrices,” in Big Data and Cloud Computing (BDCloud), Social Computing and Networking (SocialCom), Sustainable Computing and Communications (SustainCom)(BDCloudSocialCom-SustainCom), 2016 IEEE International Conferences on. IEEE, 2016, pp. 554–559.

[10] Apache Hadoop “http://hadoop.apache.org/” Last retrieved on July 2018.

[11] Apache Spark “http://spark.apache.org/” Last retrieved on July 2018.

[12] M. Zaharia, M. Chowdhury, T. Das, A. Dave, J. Ma, M. McCauley, M. J. Franklin, S. Shenker, and I. Stoica, “Resilient distributed datasets: A fault-tolerant abstraction for in-memory cluster computing,” in Proceedings of the 9th USENIX conference on Networked Systems Design and Implementation. USENIX Association, 2012, pp. 2–2.

[13] Apache Spark MLlib “http://spark.apache.org/mllib/” Last retrieved on July 2018.

[14] M. Assefi, E. Behravesh, G. Liu and A. P. Tafti, "Big data machine learning using apache spark MLlib," *2017 IEEE International Conference on Big Data (Big Data)*, Boston, MA, 2017, pp. 3492-3498. doi: 10.1109/BigData.2017.8258338.

[15] Weka library "http://www.cs.waikato.ac.nz/ml/weka/" Last retrieved on July 2018.

[16] Introduction to Machine Learning "https://towardsdatascience.com/introduction-to-machine-learning-db7c668822c4" Last retrieved on July 2018.

[17] Parametric and Nonparametric Machine Learning Algorithms https://machinelearningmastery.com/parametric-and-nonparametric-machine-learning-algorithms/ Last retrieved on July 2018.

[18] An Introduction to Machine Learning "https://www.digitalocean.com/community/tutorials/an-introduction-to-machine-learning" Last retrieved on July 2018.

[19] Bonaccorso, G., "Machine Learning Algorithms". Packt Publishing 2017 pp.94-180.

[20] Naïve Bayes classifier "https://en.wikipedia.org/wiki/Naive_Bayes_classifier" Last retrieved on July 2018.

[21] Bayes' Theorem and Conditional Probability "https://brilliant.org/wiki/bayes-theorem/" Last retrieved on July 2018.

[22] Zaki, M., & Meira, Jr, W. (2014). Data Mining and Analysis: Fundamental Concepts and Algorithms. Cambridge: Cambridge University Press. doi:10.1017/CBO9780511810114 pp.535.

[23] Naïve Bayes Classifier "http://www.statsoft.com/textbook/naive-bayes-classifier" Last retrieved on July 2018.

[24] Decision tree learning "https://en.wikipedia.org/wiki/Decision_tree_learning" Last retrieved on July 2018.

[25] Random forest "https://en.wikipedia.org/wiki/Random_forest" Last retrieved on July 2018.

[26] Support vector machine "https://en.wikipedia.org/wiki/Support_vector_machine" Last retrieved on July 2018.

[27] What is Logistic Regression? "http://www.statisticssolutions.com/what-is-logistic-regression/" Last retrieved on July 2018.

[28] Logistic regression "https://en.wikipedia.org/wiki/Logistic_regression" Last retrieved on July 2018.

[29] Apache Spark “https://en.wikipedia.org/wiki/Apache_Spark” Last retrieved on July 2018.

[30] What is Apache Spark? “https://databricks.com/spark/about” Last retrieved on July 2018.

[31] What is Apache Spark? The big data analytics platform explained “https://www.infoworld.com/article/3236869/analytics/what-is-apache-spark-the-big-data-analytics-platform-explained.html” Last retrieved on July 2018.

[32] MLlib: Main guide – Spark 2.3.0. Documentation “https://spark.apache.org/docs/2.3.0/ml-guide.html” Last retrieved on July 2018.

[33] RDD Programming Guide – Spark 2.3.0. Documentation “https://spark.apache.org/docs/2.3.0/rdd-programming-guide.html#resilient-distributed-datasets-rdds” Last retrieved on July 2018.

[34] SparkContext (Spark 2.3.0 JavaDoc “https://spark.apache.org/docs/2.3.0/api/java/org/apache/spark/SparkContext.html” Last retrieved on July 2018.

[35] Understanding Spark’s SparkConf, SparkContext, SQLContext and HiveContext “https://blogs.msdn.microsoft.com/bigdatasupport/2015/09/14/understanding-sparks-sparkconf-sparkcontext-sqlcontext-and-hivecontext/” Last retrieved on July 2018.

[36] Cluster Mode overview – Spark 2.3.1. Documentation “https://spark.apache.org/docs/latest/cluster-overview.html” Last retrieved on July 2018.

[37] Spark local (pseudo-cluster) “https://jaceklaskowski.gitbooks.io/mastering-apache-spark/spark-local.html” Last retrieved on July 2018.

[38] What is Data Collection? “https://businessjargons.com/data-collection.html” Last retrieved on July 2018.

[39] Dua, D. and Karra Taniskidou, E. (2017). UCI Machine Learning Repository [http://archive.ics.uci.edu/ml]. Irvine, CA: University of California, School of Information and Computer Science. Last retrieved on July 2018.

[40] UCI Machine Learning Repository: About “http://archive.ics.uci.edu/ml/about.html” Last retrieved on July 2018.

[41] Baldi, P., P. Sadowski, and D. Whiteson. “Searching for Exotic Particles in High-energy Physics with Deep Learning.” Nature Communications 5 (July 2, 2014).

[42] P. Baldi, K. Cranmer, T. Faucett, P. Sadowski, and D. Whiteson, “Parameterized neural networks for high-energy physics,” Eur. Phys. J. C76 (2016) no. 5, 235, arXiv:1601.07913 [hep-ex].

[43] Ganti V, Sarma AD. “Data cleaning: A practical perspective. Synthesis Lectures on Data Management”. 2013;5:1-85.

[44] UCI Machine Learning Repository: HIGGS Data Set “http://archive.ics.uci.edu/ml/datasets/HIGGS” Last retrieved on July 2018.

[45] UCI Machine Learning Repository: SUSY Data Set “http://archive.ics.uci.edu/ml/datasets/SUSY” Last retrieved on July 2018.

[46] UCI Machine Learning Repository: Hepmass Data Set “http://archive.ics.uci.edu/ml/datasets/HEPMASS” Last retrieved on July 2018.

[47] Data types – RDD-based API – Spark 2.0.2. Documentation “https://spark.apache.org/docs/2.0.2/mllib-data-types.html#labeled-point” Last retrieved on July 2018.

[48] Ubuntu 17.10.1 (Atrful Aardvark) “http://releases.ubuntu.com/17.10/” Last retrieved on July 2018.

[49] Java SE Development Kit 8 “http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html” Last retrieved on July 2018.

[50] Python Release Python 3.6.4 “https://www.python.org/downloads/release/python-364/” Last retrieved on July 2018.

[51] Anaconda installer archive “https://repo.continuum.io/archive/” Last retrieved on July 2018.

[52] Anaconda (Python Distribution) “https://en.wikipedia.org/wiki/Anaconda_(Python_distribution)” Last retrieved on July 2018.

[53] Downloads – Apache Spark “https://spark.apache.org/downloads.html” Last retrieved on July 2018

[54] Path (variable) “https://en.wikipedia.org/wiki/PATH_(variable)” Last retrieved on July 2018.

[55] Overview (Spark 2.3.0. Documentation) “https://spark.apache.org/docs/2.3.0/api/java/index.html” Last retrieved on July 2018.

[56] PySpark 2.2.0. documentation “https://spark.apache.org/docs/2.2.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.NaiveBayes” Last retrieved on July 2018.

[57] PySpark 2.2.0. documentation "https://spark.apache.org/docs/2.2.0/api/python/pyspark.mllib.html#pyspark.mllib.tree.DecisionTree" Last retrieved on July 2018.

[58] PySpark 2.2.0. documentation "https://spark.apache.org/docs/latest/api/python/pyspark.mllib.html#pyspark.mllib.tree.RandomForest" Last retrieved on July 2018.

[59] PySpark 2.2.0. documentation "https://spark.apache.org/docs/2.2.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.SVMWithSGD" Last retrieved on July 2018.

[60] PySpark 2.2.0. documentation "https://spark.apache.org/docs/2.2.0/api/python/pyspark.mllib.html#pyspark.mllib.classification.LogisticRegressionWithLBFGS" Last retrieved on July 2018.

[61] Evaluation Metrics – RDD-based API – Spark 2.3.0 Documenatation "https://spark.apache.org/docs/2.3.0/mllib-evaluation-metrics.html" Last retrieved on July 2018.

[62] Differences between ROC AUC and PR AUC "http://www.chioka.in/differences-between-roc-auc-and-pr-auc/" Last retrieved on July 2018.

[63] Iris flower data set "https://en.wikipedia.org/wiki/Iris_flower_data_set" Last retrieved on July 2018.

[64] UCI Machine Learning Repository: Iris Data Set "https://archive.ics.uci.edu/ml/datasets/Iris" Last retrieved on July 2018.

APPENDIX A

Table A.1. Summary measures for HIGGS dataset

	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6
Mean	0.991	-8.29e-06	-1.32e-05	0.998	2.61-05	0.990
Maximum Value	12.098	2.434	1.743	15.396	1.743	9.940
Minimum Value	0.274	-2.434	-1.742	2.37e-04	-1.734	0.137
Non-zero Values	11000000	11000000	11000000	11000000	11000000	11000000

	Feature 7	Feature 8	Feature 9	Feature 10	Feature 11	Feature 12
Mean	-2.02e-05	7.71e-06	0.998	0.992	-1.02e-05	-2.07e-05
Maximum Value	2.969	1.741	2.173	11.647	2.913	1.743
Minimum Value	-2.969	-1.745	0.000	0.188	-2.729	-1.742
Non-zero Values	11000000	11000000	5605389	11000000	11000000	11000000

	Feature 13	Feature 14	Feature 15	Feature 16	Feature 17	Feature 18
Mean	1.001	0.992	1.45e-05	3.67e-06	1.000	0.986
Maximum Value	2.214	14.708	2.730	1.742	2.548	12.882
Minimum Value	0.000	0.263	-2.496	-1.742	0.000	0.365
Non-zero Values	5476088	11000000	11000000	11000000	4734760	11000000

Table A.1. Summary measures for HIGGS dataset (continued)

	Feature 19	Feature 20	Feature 21	Feature 22	Feature 23	Feature 24
Mean	-5.75e-06	1.74e-05	1.000	1.034	1.023	1.050
Maximum Value	2.498	1.743	3.101	40.192	20.372	7.992
Minimum Value	-2.496	-1.742	0.000	0.075	0.199	0.083
Non-zero Values	11000000	11000000	3869383	11000000	11000000	11000000

	Feature 25	Feature 26	Feature 27	Feature 28
Mean	1.009	0.927	1.033	0.959
Maximum Value	14.262	17.762	11.496	8.374
Minimum Value	0.132	0.047	0.295	0.330
Non-zero Values	11000000	11000000	11000000	11000000

Table A.2. Time taken by machine learning algorithms in Local Cluster mode using different nodes for HIGGS dataset

		Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	**Real**	21.56	35.89	36.30	39.45	38.45
	User	1.46	9.23	12.25	56.46	46.46
	Sys	0.49	2.31	2.45	4.01	2.56
Node 3	**Real**	19.45	33.56	34.25	37.81	35.22
	User	2.01	9.26	11.81	59.54	48.55
	Sys	0.59	1.49	2.25	4.23	2.45
Node 4	**Real**	18.56	31.12	31.12	35.25	33.56
	User	2.31	9.01	11.25	59.25	48.49
	Sys	0.56	1.40	2.45	4.25	2.44

Table A.3. Time taken by machine learning algorithms in GPU Cluster mode using different nodes for HIGGS dataset

		Naïve Bayes	Decision Tree	Random Forest	SVM	Logistic Regression
	Real	17.43	32.47	33.30	36.56	24.54
Node 2	**User**	1.57	8.53	11.80	132.05	53.46
	Sys	0.29	2.49	2.11	3.26	2.34
	Real	15.25	29.59	31.18	35.10	23.24
Node 3	**User**	2.13	9.26	10.58	135.45	52.24
	Sys	0.26	0.58	1.20	3.15	2.15
	Real	13.45	25.25	26.34	33.10	19.19
Node 4	**User**	2.36	9.11	11.59	125.02	50.06
	Sys	0.34	1.40	1.19	3.05	2.01

Table A.4. Accuracy of machine learning algorithms in Local Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	Decision Tree	Random Forest	SVM	Logistic Regression
Node 2	57.32	66.69	67.10	61.25	64.30
Node 3	57.33	66.71	67.10	61.25	64.31
Node 4	57.33	66.69	67.13	61.24	64.31

Table A.5. Accuracy of machine learning algorithms in GPU Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	57.53	67.30	67.16	61.23	64.25
Node 3	57.53	67.30	67.16	61.21	64.25
Node 4	57.52	67.30	67.15	61.23	64.25

Table A.6. Area under ROC of machine learning algorithms in Local Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.564	0.671	0.665	0.625	0.640
Node 3	0.564	0.672	0.665	0.624	0.641
Node 4	0.566	0.672	0.665	0.625	0.642

Table A.7. Area under ROC of machine learning algorithms in GPU Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.566	0.672	0.670	0.630	0.644
Node 3	0.566	0.672	0.671	0.632	0.644
Node 4	0.565	0.671	0.670	0.632	0.645

Table A.8. Area under PR of machine learning algorithms in Local Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.570	0.674	0.733	0.810	0.720
Node 3	0.569	0.674	0.732	0.810	0.721
Node 4	0.571	0.677	0.733	0.813	0.721

Table A.9. Area under PR of machine learning algorithms in GPU Cluster mode using different nodes for HIGGS dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.571	0.655	0.732	0.859	0.721
Node 3	0.571	0.655	0.733	0.861	0.721
Node 4	0.569	0.651	0.735	0.861	0.722

Table A.10. Summary measures for Hepmass dataset

	Feature 1	**Feature 2**	**Feature 3**	**Feature 4**	**Feature 5**	**Feature 6**
Mean	8.37e-02	2.63e-04	2.51e-04	5.37e-02	2.14e-04	1.14e-02
Maximum Value	4.093	2.365	1.732	4.265	1.731	4.482
Minimum Value	-1.960	-2.365	-1.732	-9.980	-1.732	-1.054
Non-zero Values	7000000	7000000	7000000	7000000	7000000	7000000

Table A.10. Summary measures for Hepmass dataset (continued)

	Feature 7	**Feature 8**	**Feature 9**	**Feature 10**	**Feature 11**	**Feature 12**
Mean	8.97e-02	-1.03e-04	-3.69e-05	-3.44e-02	8.50e-02	1.48e-04
Maximum Value	3.749	2.758	1.731	0.754	4.601	2.602
Minimum Value	-3.034	-2.757	-1.732	-1.325	-2.835	-2.602
Non-zero Values	7000000	7000000	7000000	7000000	7000000	7000000

	Feature 13	**Feature 14**	**Feature 15**	**Feature 16**	**Feature 17**	**Feature 18**
Mean	3.62e-04	9.97e-03	8.4e-02	2.94e-04	-4.43e-04	2.41e-02
Maximum Value	1.732	0.860	5.051	2.438	1.732	1.226
Minimum Value	-1.732	-1.161	-2.454	-2.437	-1.732	-0.815
Non-zero Values	7000000	7000000	7000000	7000000	7000000	7000000

	Feature 19	**Feature 20**	**Feature 21**	**Feature 22**	**Feature 23**	**Feature 24**
Mean	5.56e-02	1.63e-04	5.43e-04	1.68e-03	6.15e-02	4.77e-04
Maximum Value	5.788	2.282	1.732	1.743	7.419	9.374
Minimum Value	-1.728	-2.281	-1.731	-0.573	-3.590	-4.119
Non-zero Values	7000000	7000000	7000000	7000000	7000000	7000000

Table A.10. Summary measures for Hepmass dataset (continued)

	Feature 25	**Feature 26**	**Feature 27**
Mean	2.75e-02	-8.34e-03	7.45e-02
Maximum Value	14.927	4.613	4.729
Minimum Value	-20.622	-3.452	-2.622
Non-zero Values	6999995	7000000	7000000

Table A.11. Time taken by machine learning algorithms in Local Cluster mode using different nodes for Hepmass dataset

		Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
	Real	15.16	17.45	17.10	30.26	19.15
Node 2	**User**	2.05	4.45	8.45	6.56	10.25
	Sys	0.56	0.55	2.01	0.59	0.45
	Real	13.10	16.14	15.15	27.56	17.45
Node 3	**User**	2.15	4.44	8.46	5.34	10.24
	Sys	0.15	0.56	2.22	0.59	0.56
	Real	12.11	14.45	13.48	25.25	16.35
Node 4	**User**	2.45	5.05	7.56	6.01	14.45
	Sys	0.15	0.44	2.45	0.58	0.45

Table A.12. Time taken by machine learning algorithms in GPU Cluster mode using different nodes for Hepmass dataset

		Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
	Real	12.19	14.24	16.00	24.54	17.11
Node 2	**User**	1.51	4.43	7.16	5.39	9.00
	Sys	1.90	0.59	1.32	0.5	0.46
	Real	13.10	13.14	13.01	21.50	14.60
Node 3	**User**	2.30	5.31	7.26	5.34	11.10
	Sys	0.10	0.5	0.49	0.59	0.32
	Real	9.46	11.56	11.56	18.56	9.32
Node 4	**User**	2.7	5.42	7.56	6.8	11.60
	Sys	0.10	0.34	2.80	0.4	0.26

Table A.13. Accuracy of machine learning algorithms in Local Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	71.21	89.89	90.60	88.30	90.16
Node 3	71.22	89.92	90.59	88.26	90.16
Node 4	71.22	89.92	90.60	88.26	90.15

Table A.14. Accuracy of machine learning algorithms in GPU Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	70.08	90.29	90.30	90.16	90.21
Node 3	70.08	90.29	90.69	90.16	90.21
Node 4	70.08	90.29	90.69	90.16	90.21

Table A.15. Area under ROC of machine learning algorithms in Local Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.708	0.904	0.906	0.900	0.910
Node 3	0.710	0.905	0.906	0.901	0.910
Node 4	0.710	0.904	0.910	0.900	0.911

Table A.16. Area under ROC of machine learning algorithms in GPU Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.711	0.911	0.910	0.897	0.911
Node 3	0.712	0.911	0.909	0.900	0.912
Node 4	0.711	0.912	0.910	0.900	0.912

Table A.17. Area under PR of machine learning algorithms in Local Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.687	0.912	0.896	0.920	0.920
Node 3	0.688	0.912	0.895	0.921	0.920
Node 4	0.687	0.913	0.895	0.920	0.921

Table A.18. Area under PR of machine learning algorithms in GPU Cluster mode using different nodes for Hepmass dataset

	Naïve Bayes	**Decision Tree**	**Random Forest**	**SVM**	**Logistic Regression**
Node 2	0.690	0.910	0.896	0.920	0.919
Node 3	0.691	0.911	0.897	0.921	0.920
Node 4	0.691	0.911	0.896	0.920	0.920

APPENDIX B

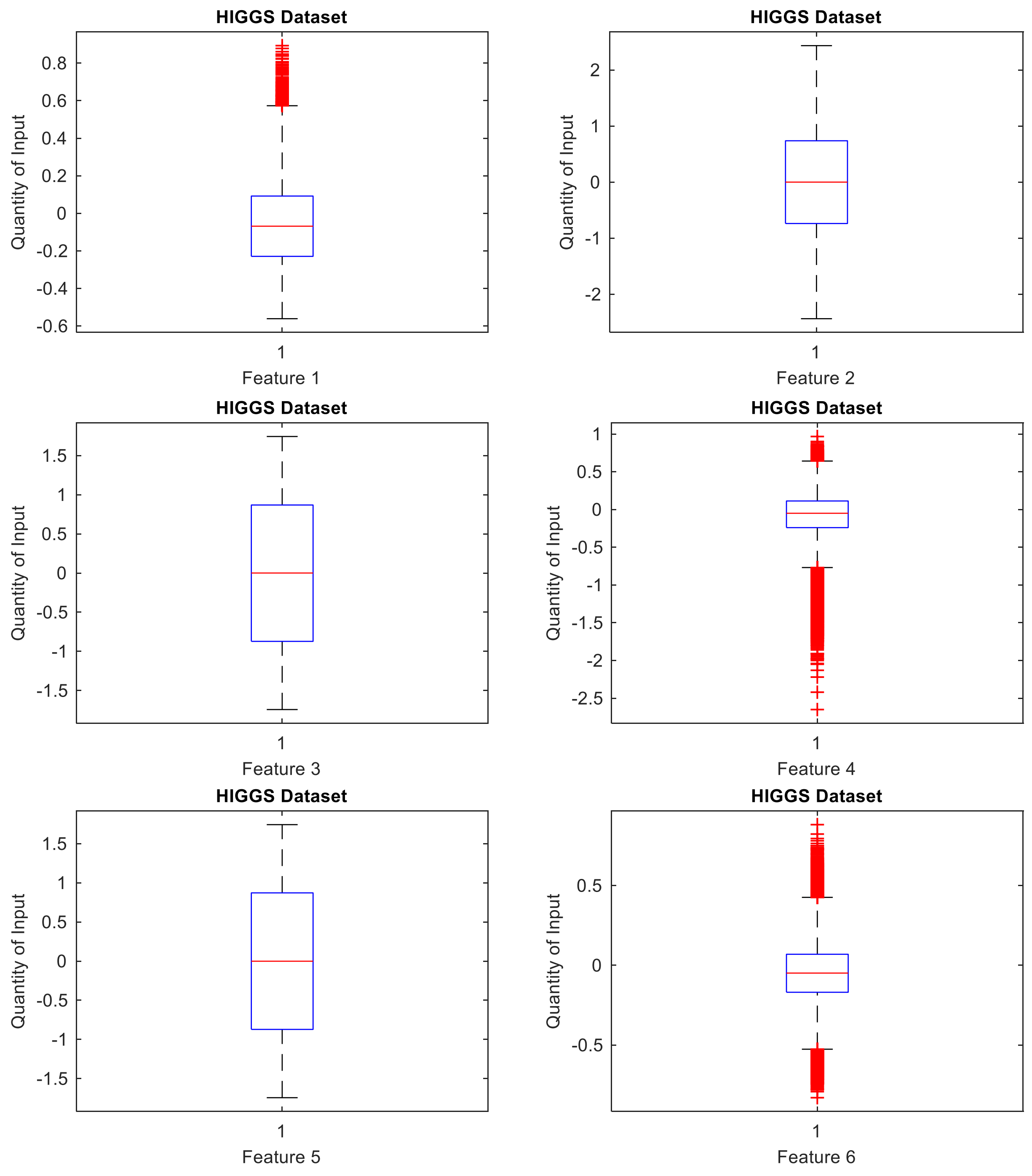


Figure B.1. Boxplots for HIGGS dataset

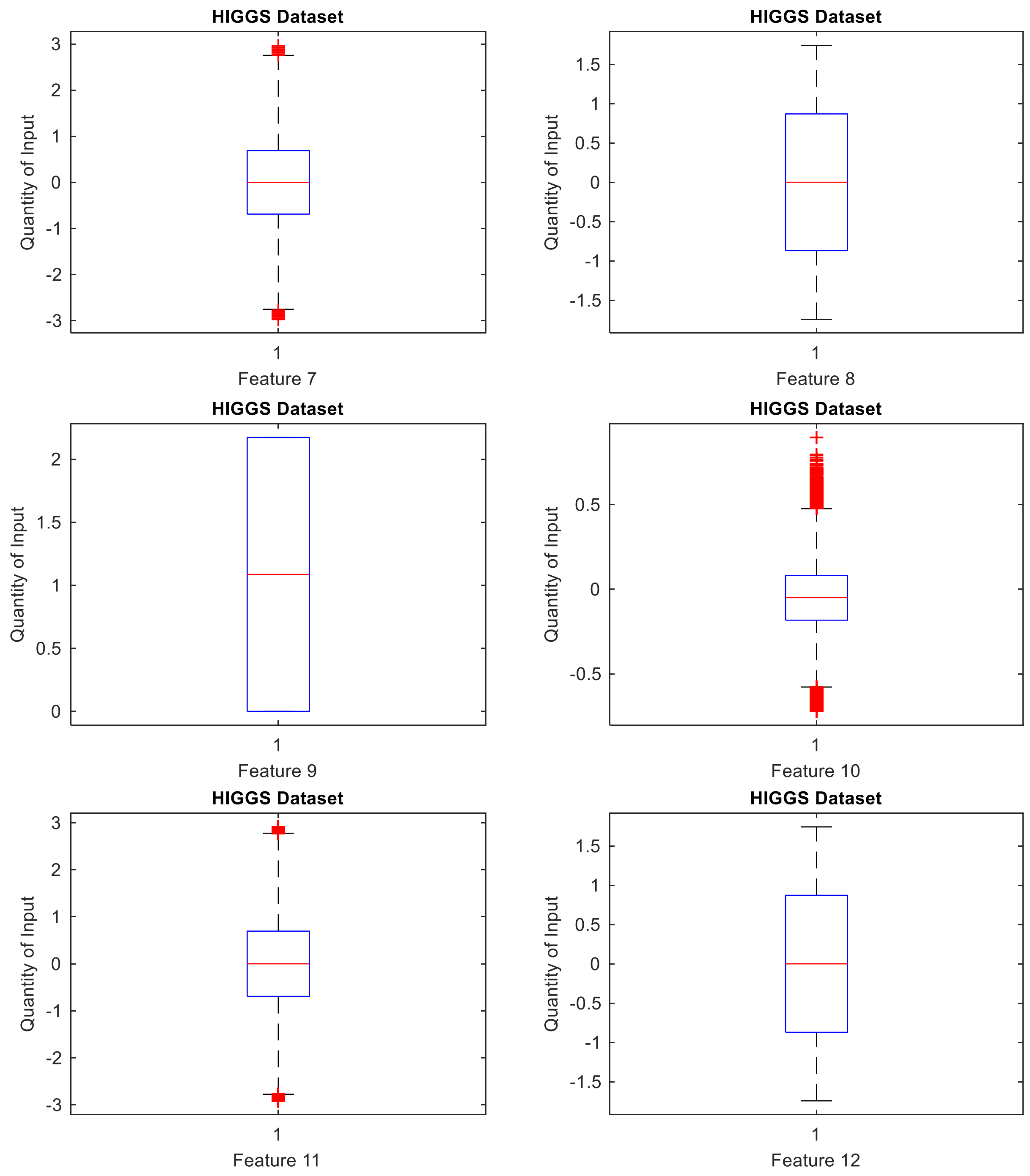


Figure B.1. Boxplots for HIGGS dataset (continued)

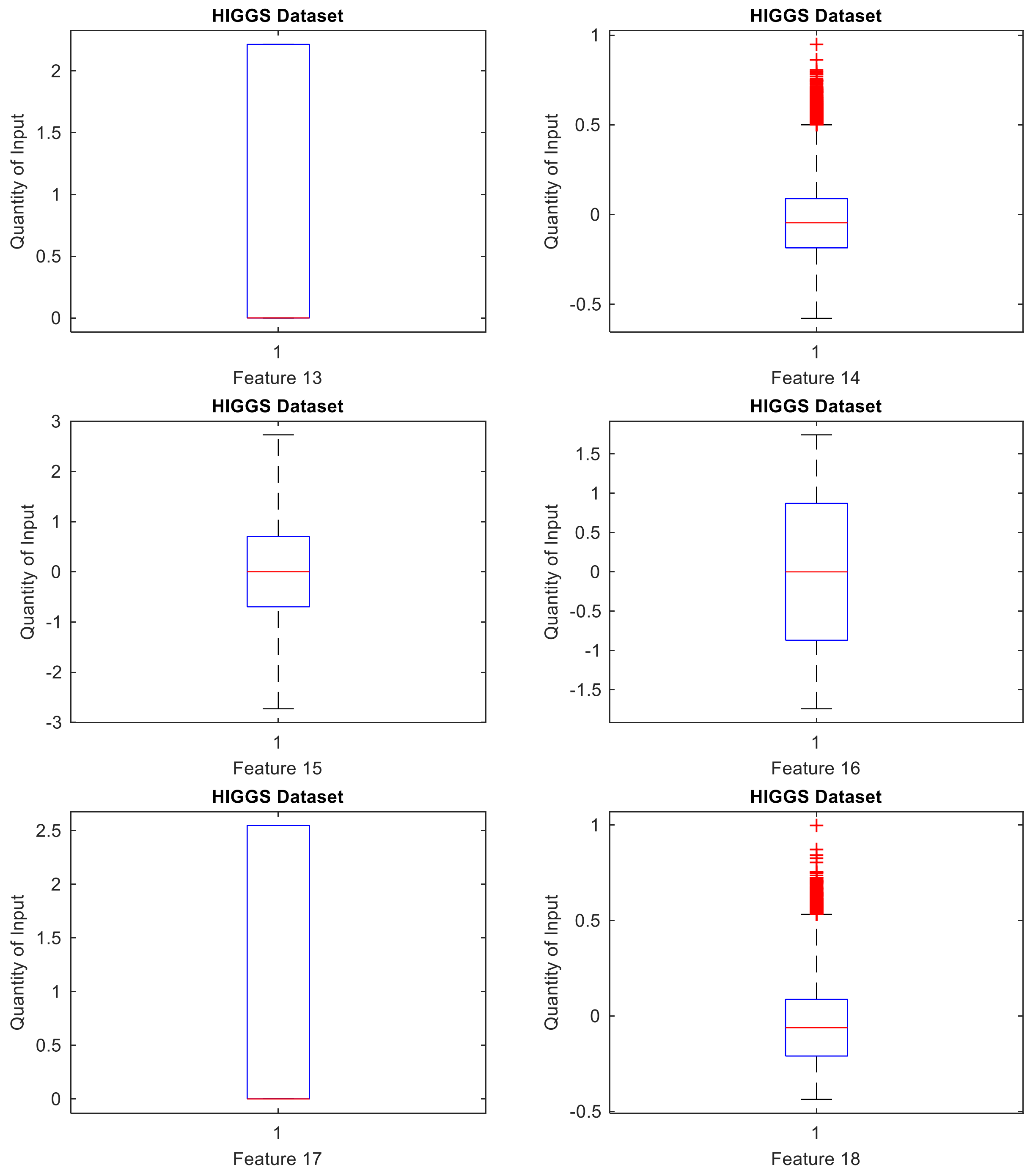


Figure B.1. Boxplots for HIGGS dataset (continued)

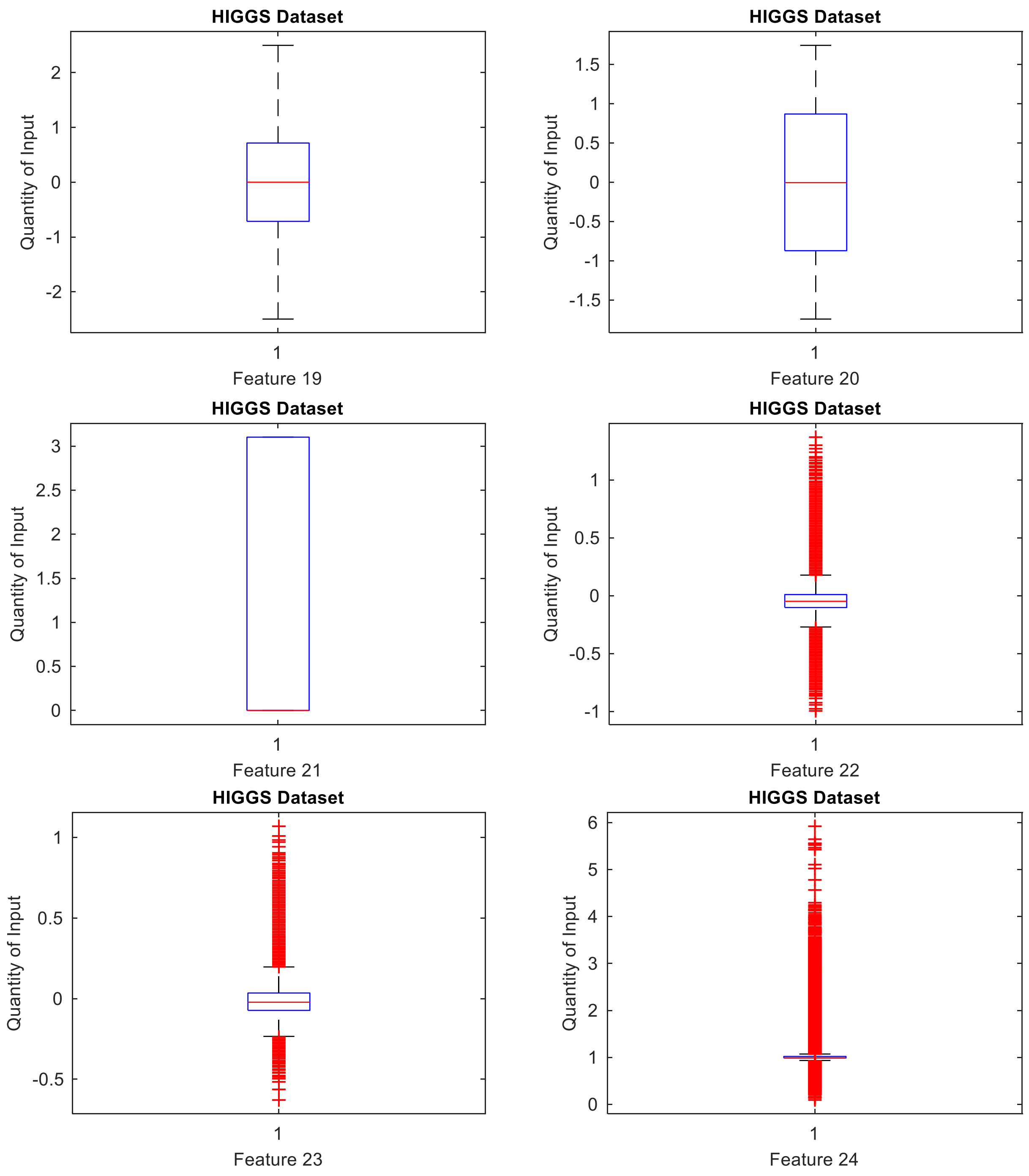


Figure B.1. Boxplots for HIGGS dataset (continued)

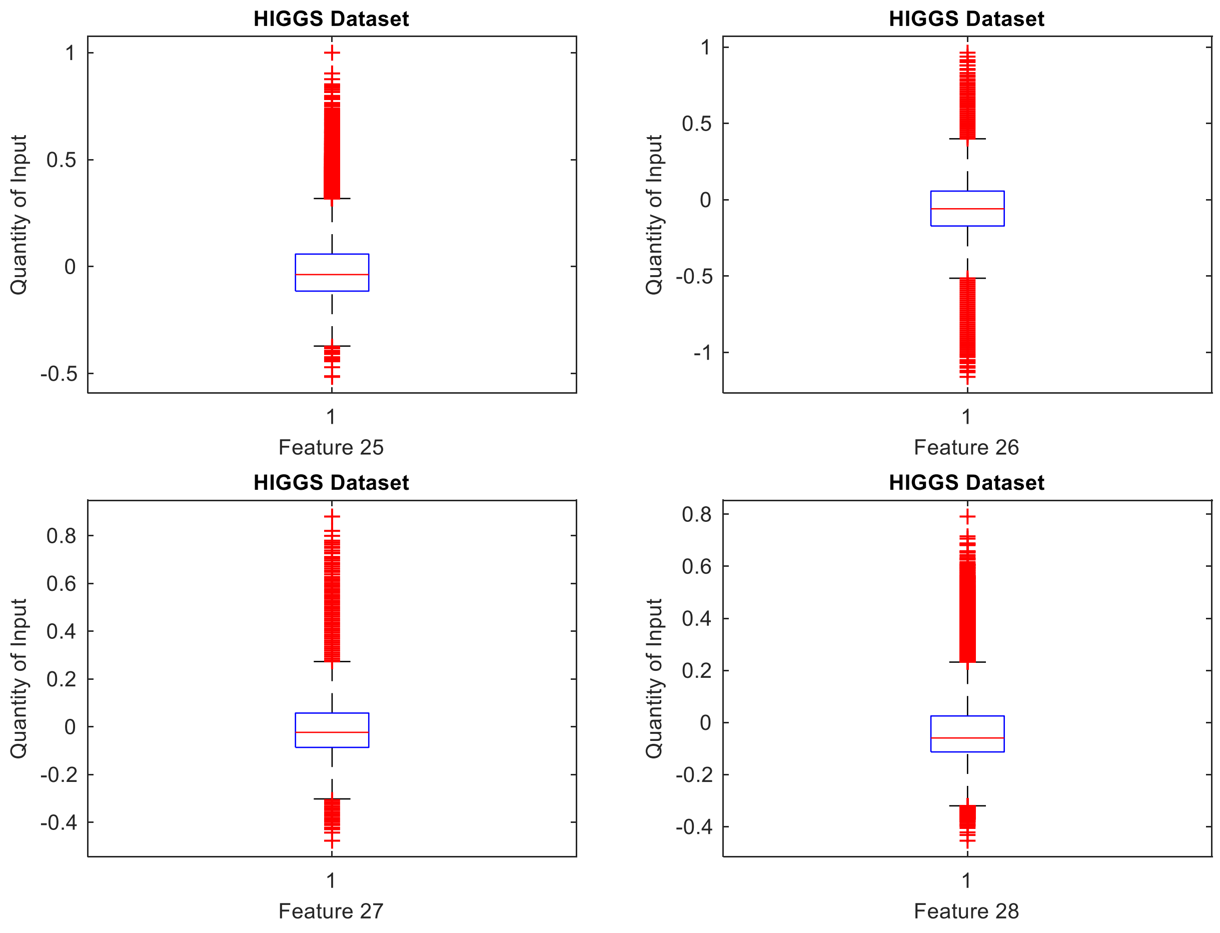


Figure B.1. Boxplots for HIGGS dataset (continued)

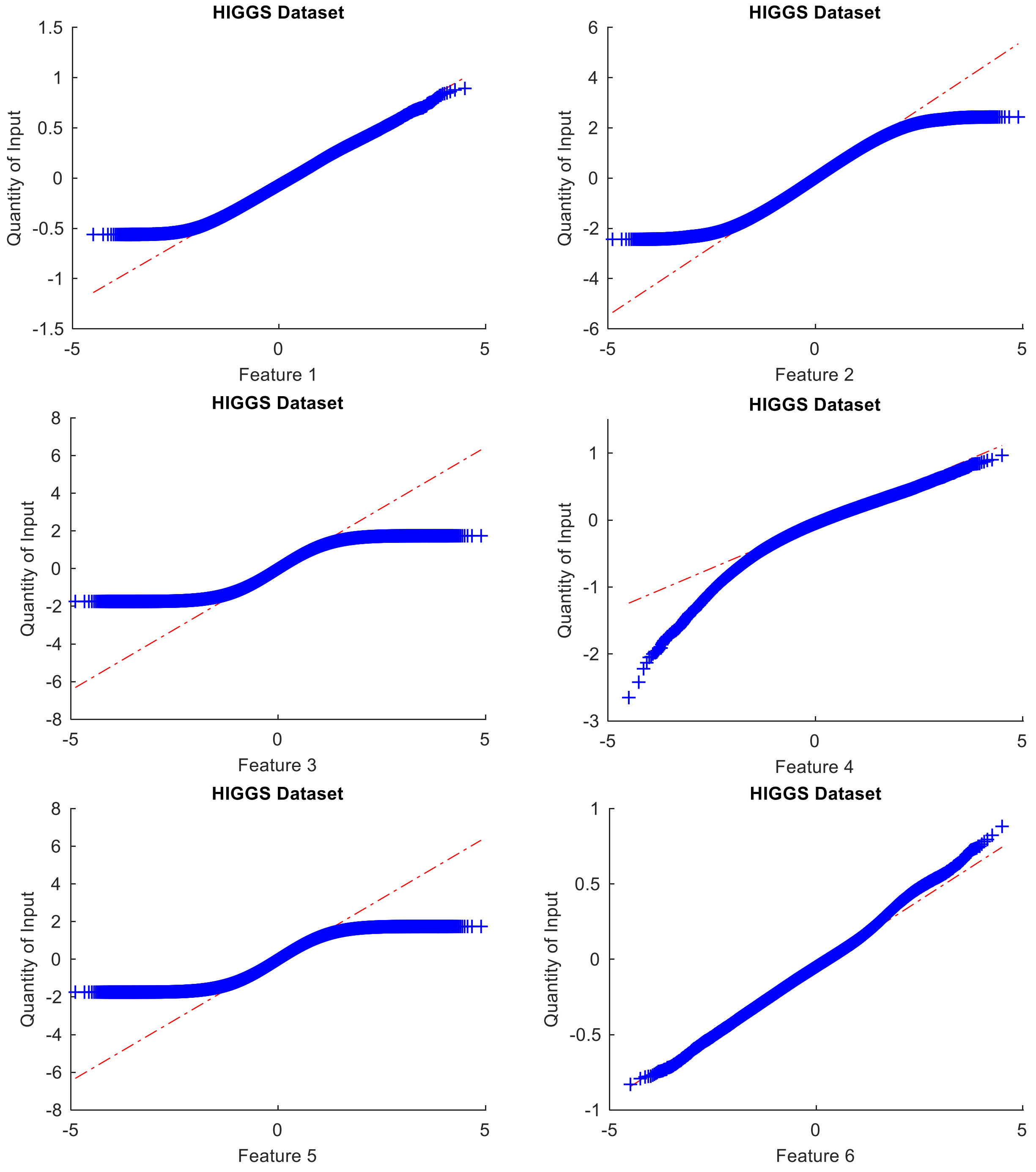


Figure B.2. QQplots for HIGGS dataset

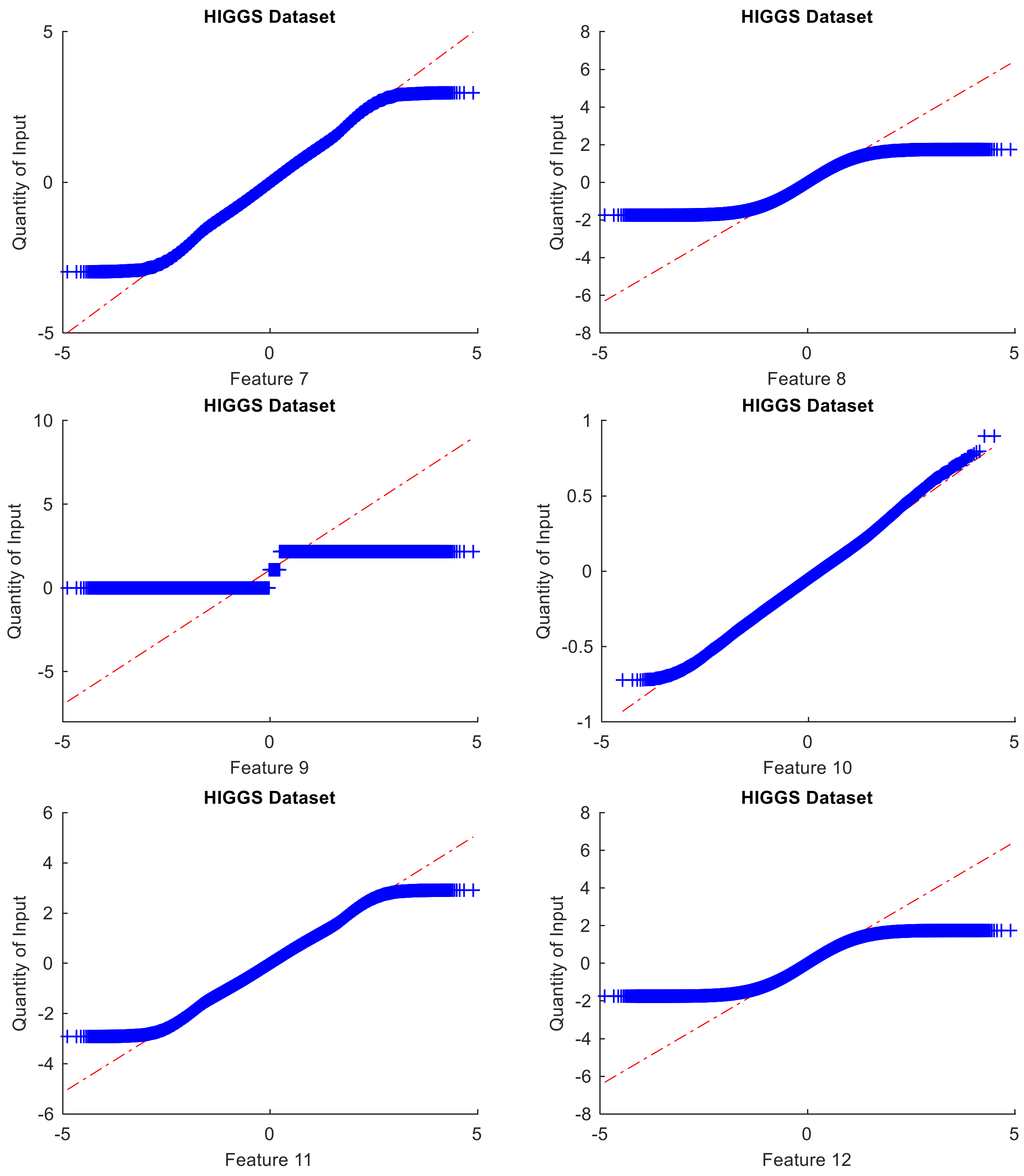


Figure B.2. QQplots for HIGGS dataset (continued)

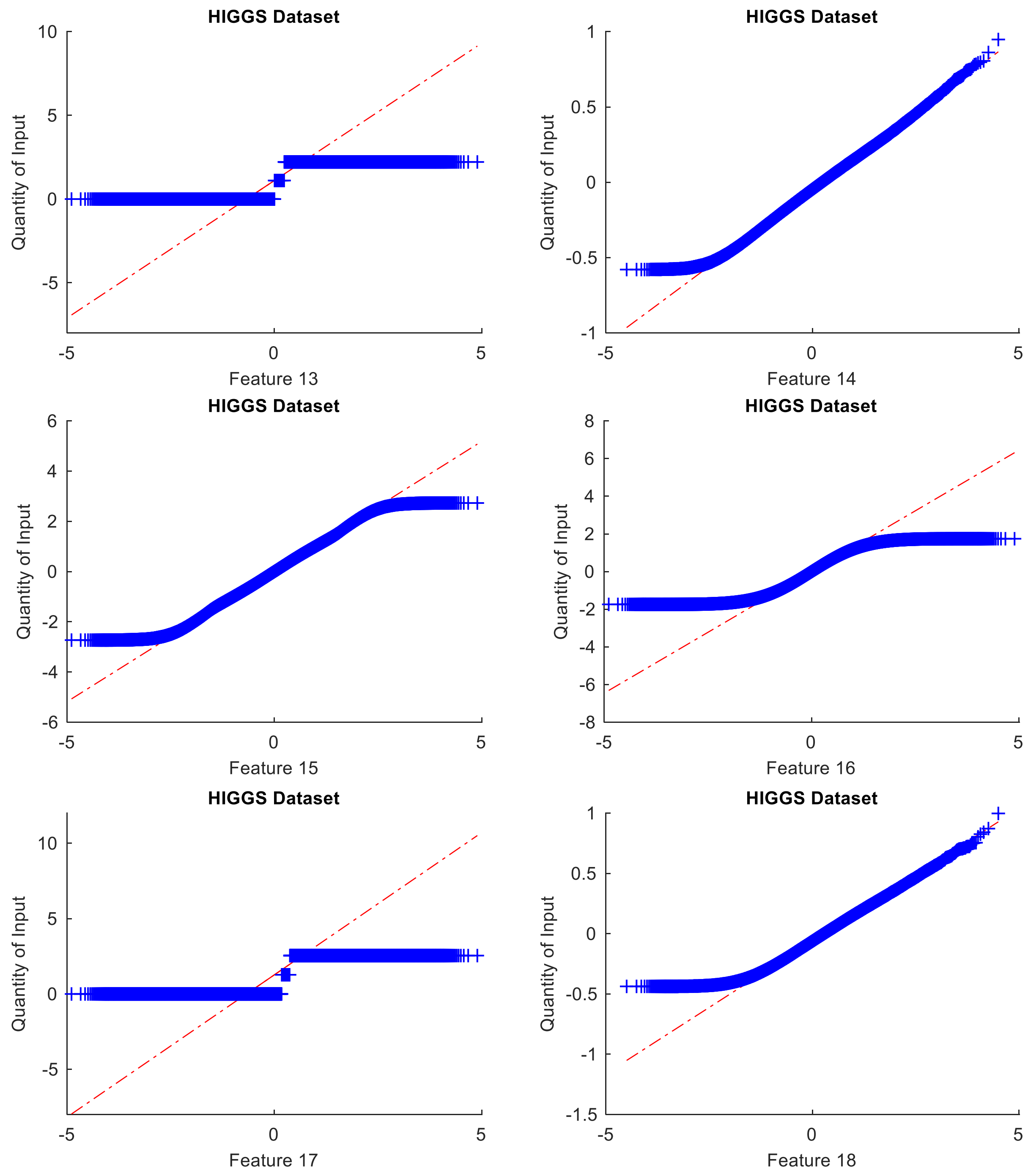


Figure B.2. QQplot for HIGGS dataset (continued)

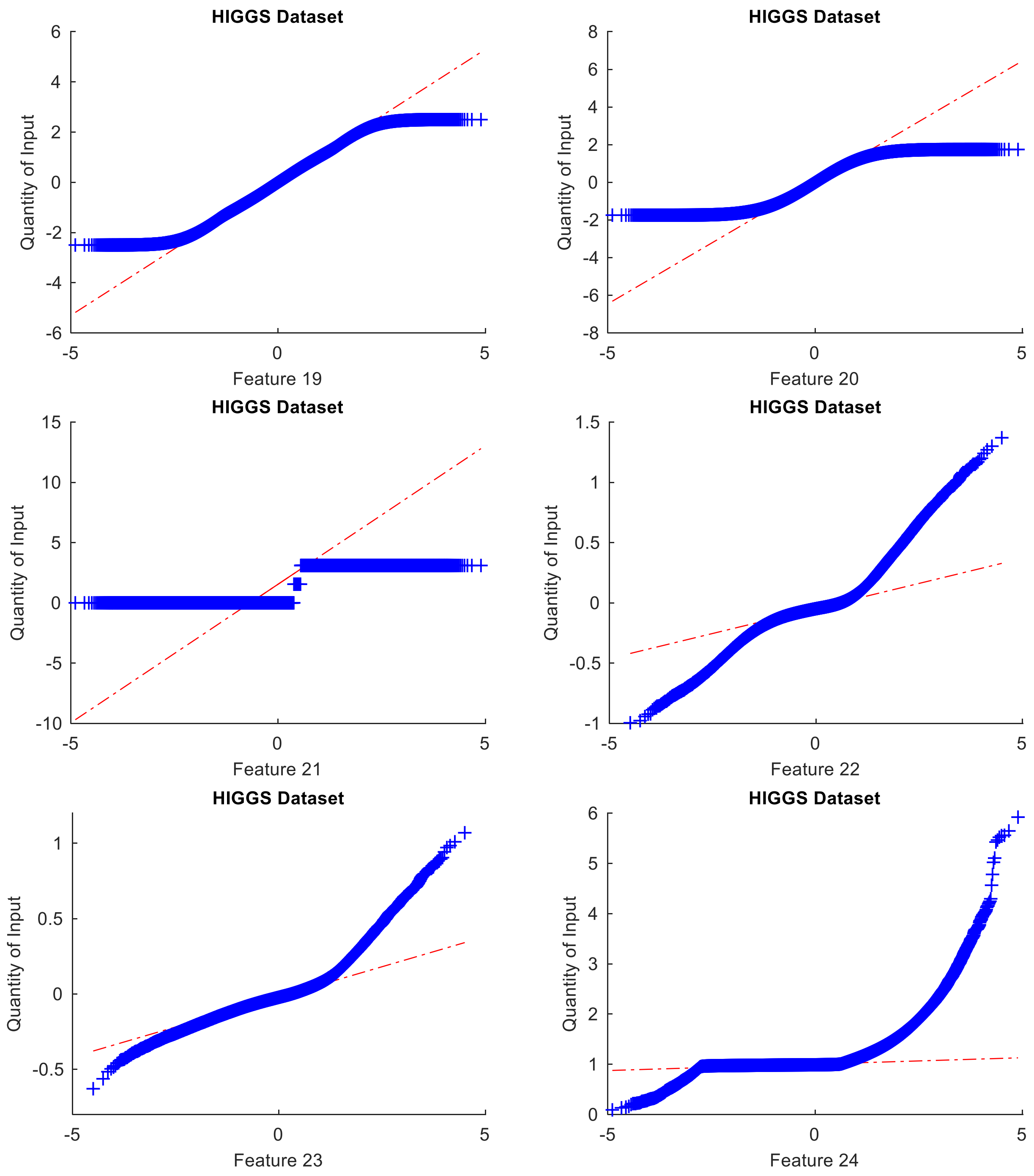


Figure B.2. QQplot for HIGGS dataset (continued)

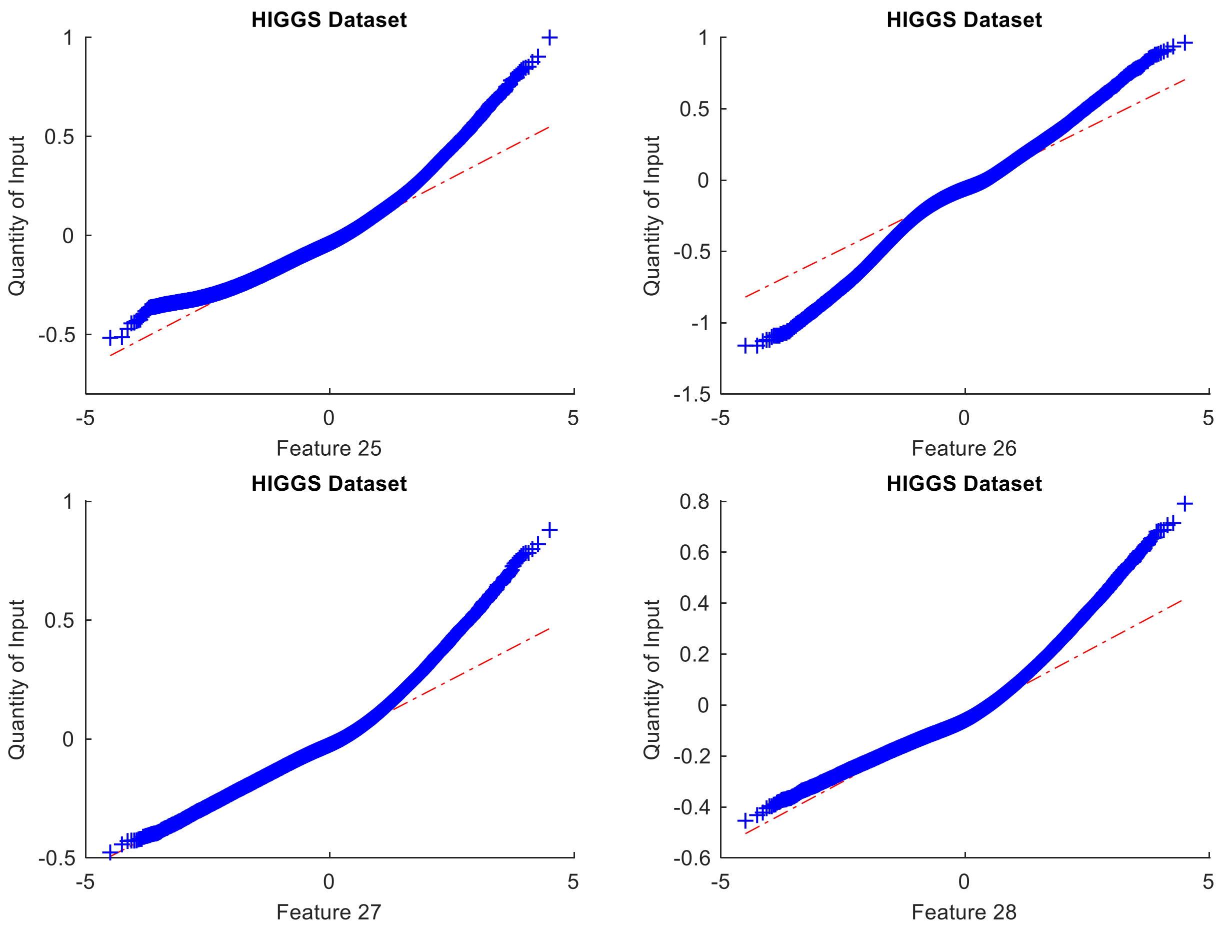


Figure B.2. QQplot for HIGGS dataset (continued)

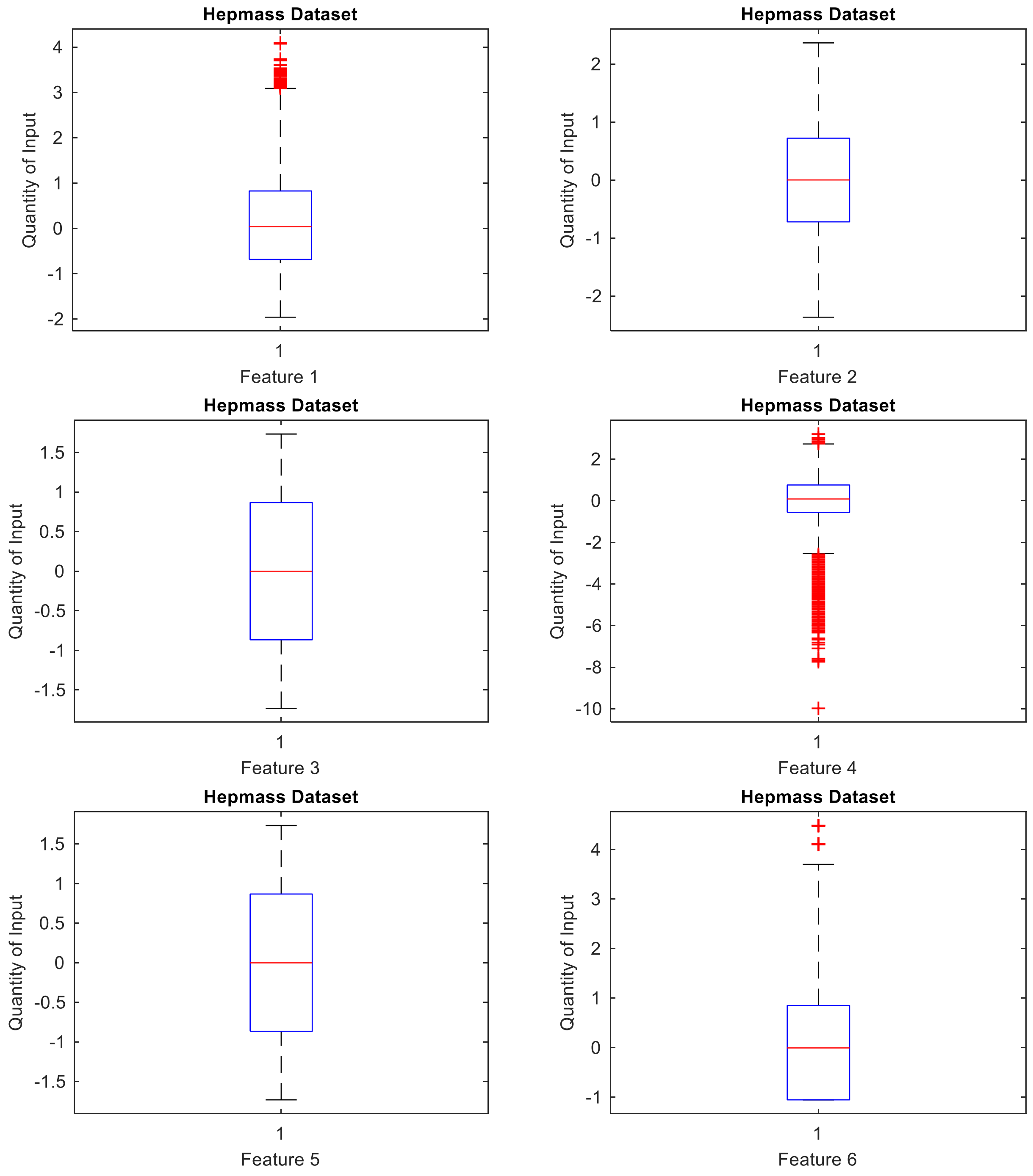


Figure B.3. Boxplots for Hepmass dataset

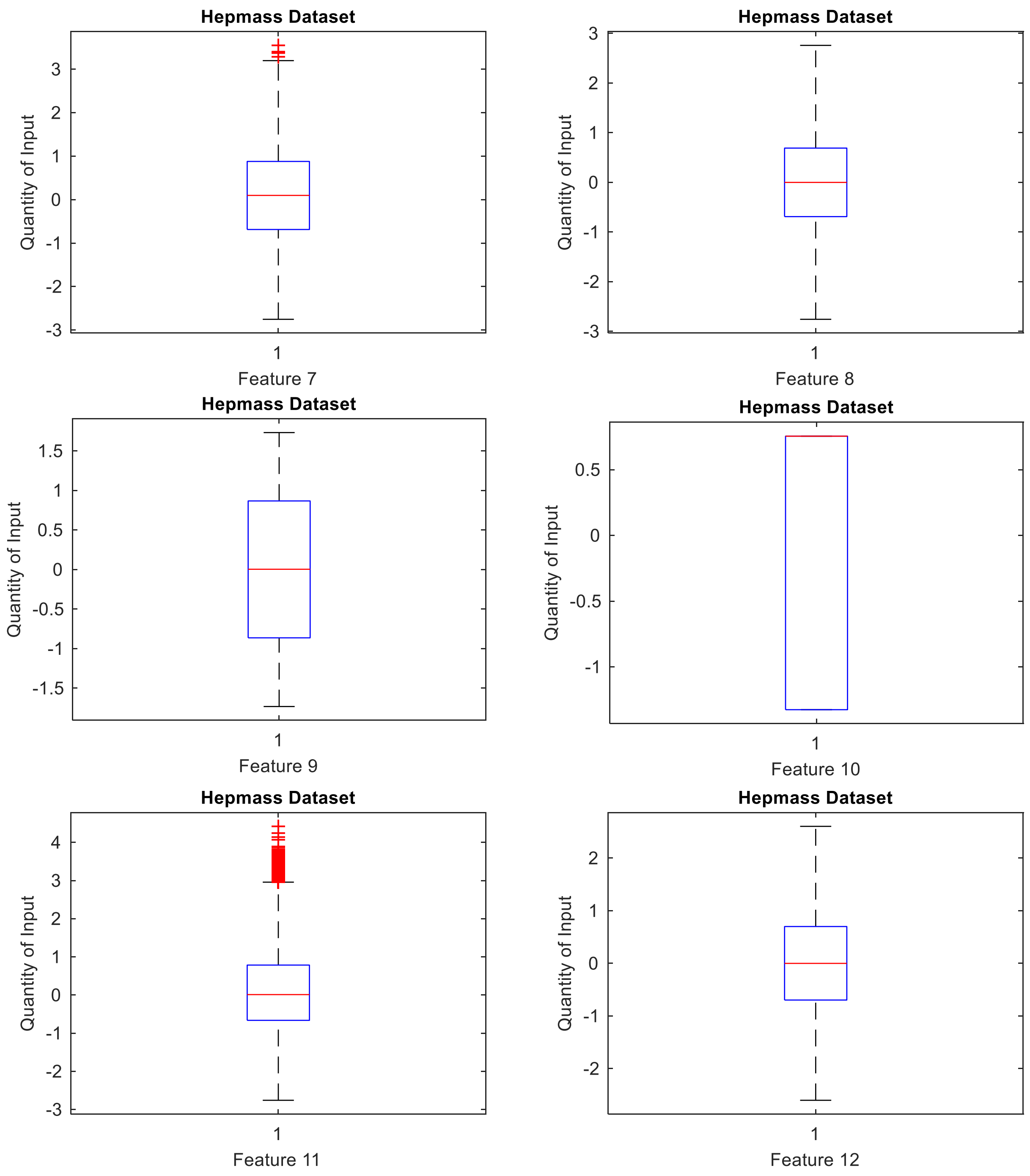


Figure B.3. Boxplots for Hepmass dataset (continued)

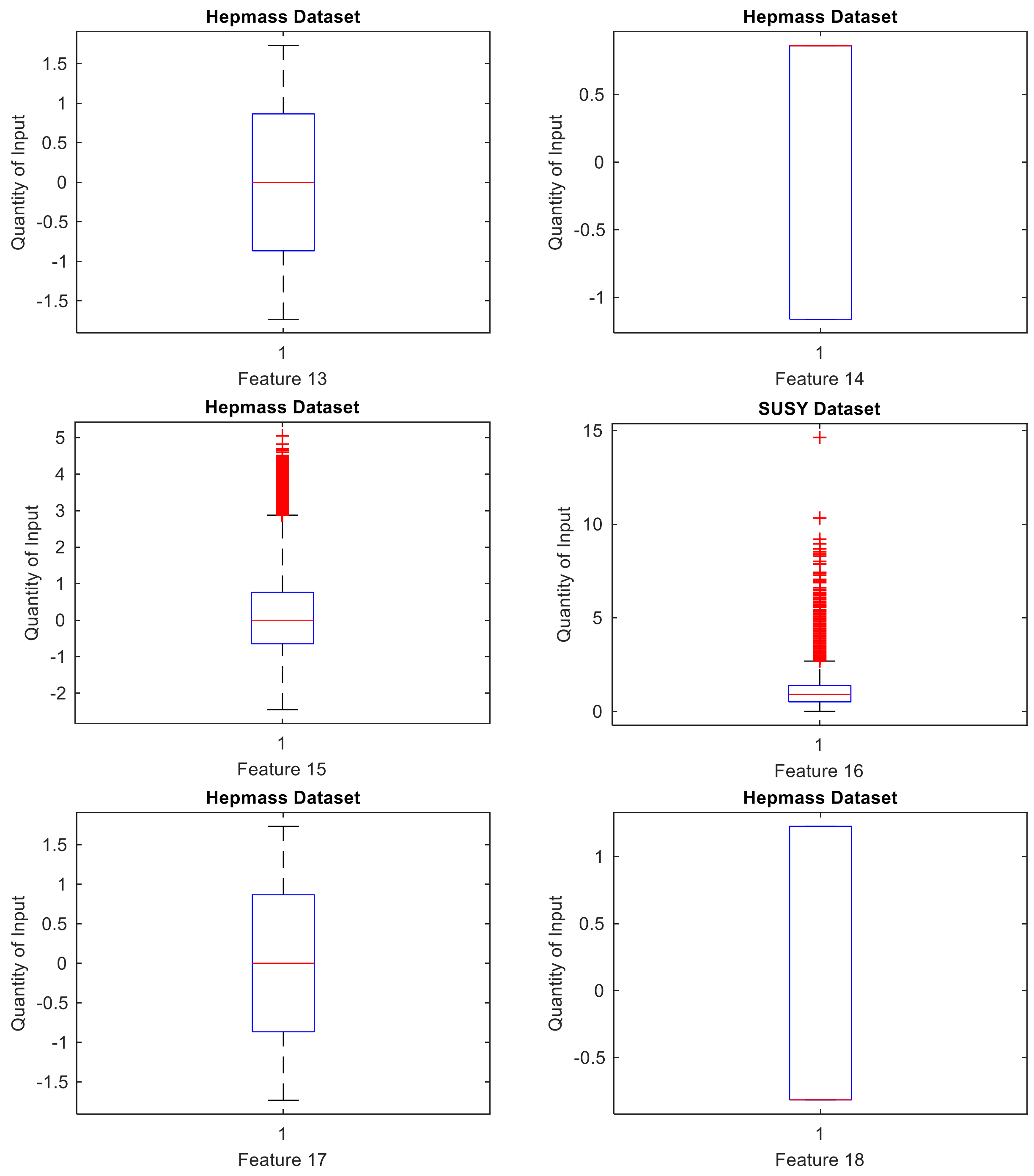


Figure B.3. Boxplots for Hepmass dataset (continued)

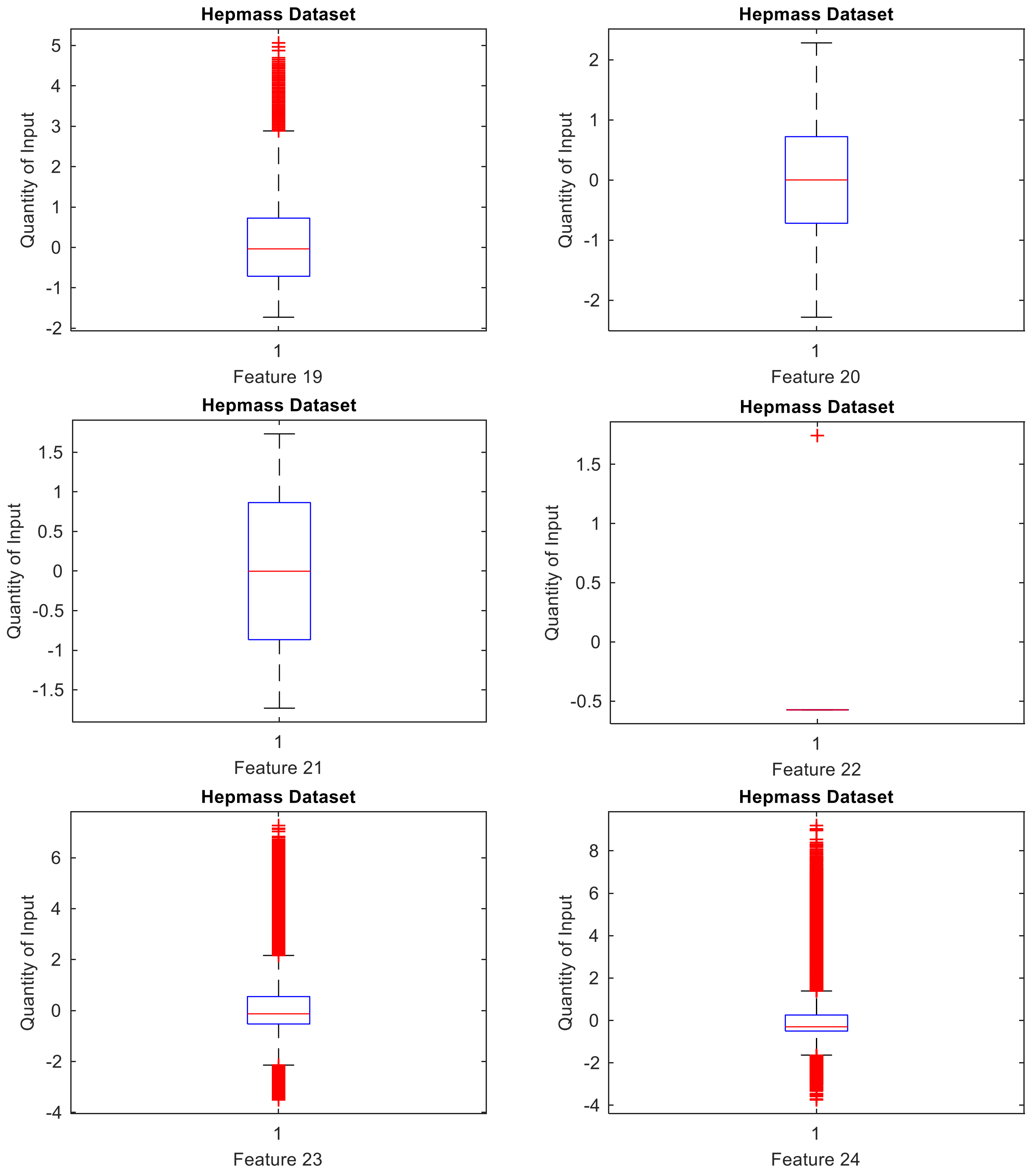
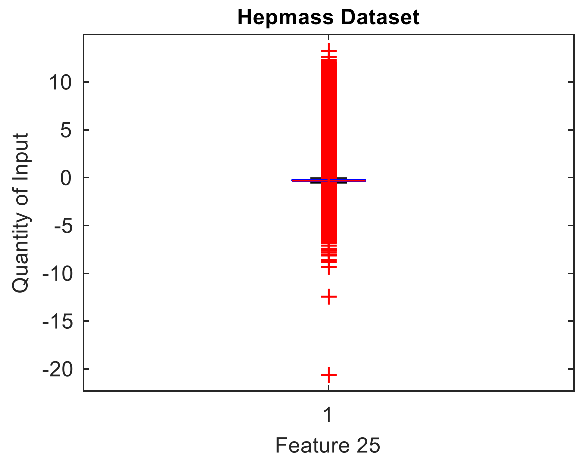
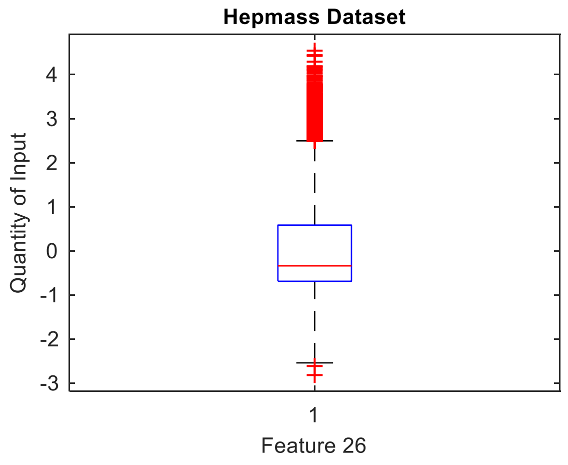


Figure B.3. Boxplots for Hepmass dataset (continued)





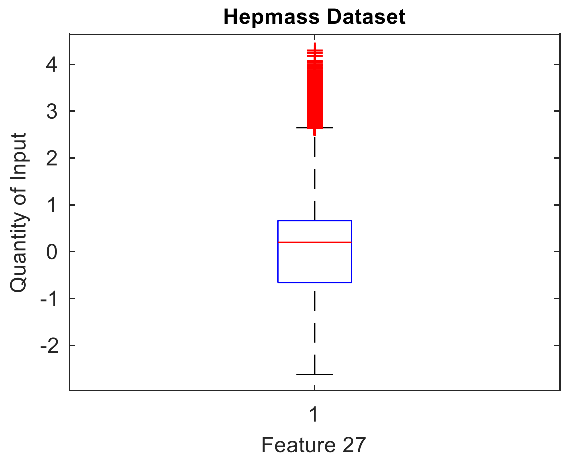


Figure B.3. Boxplots for Hepmass dataset (continued)

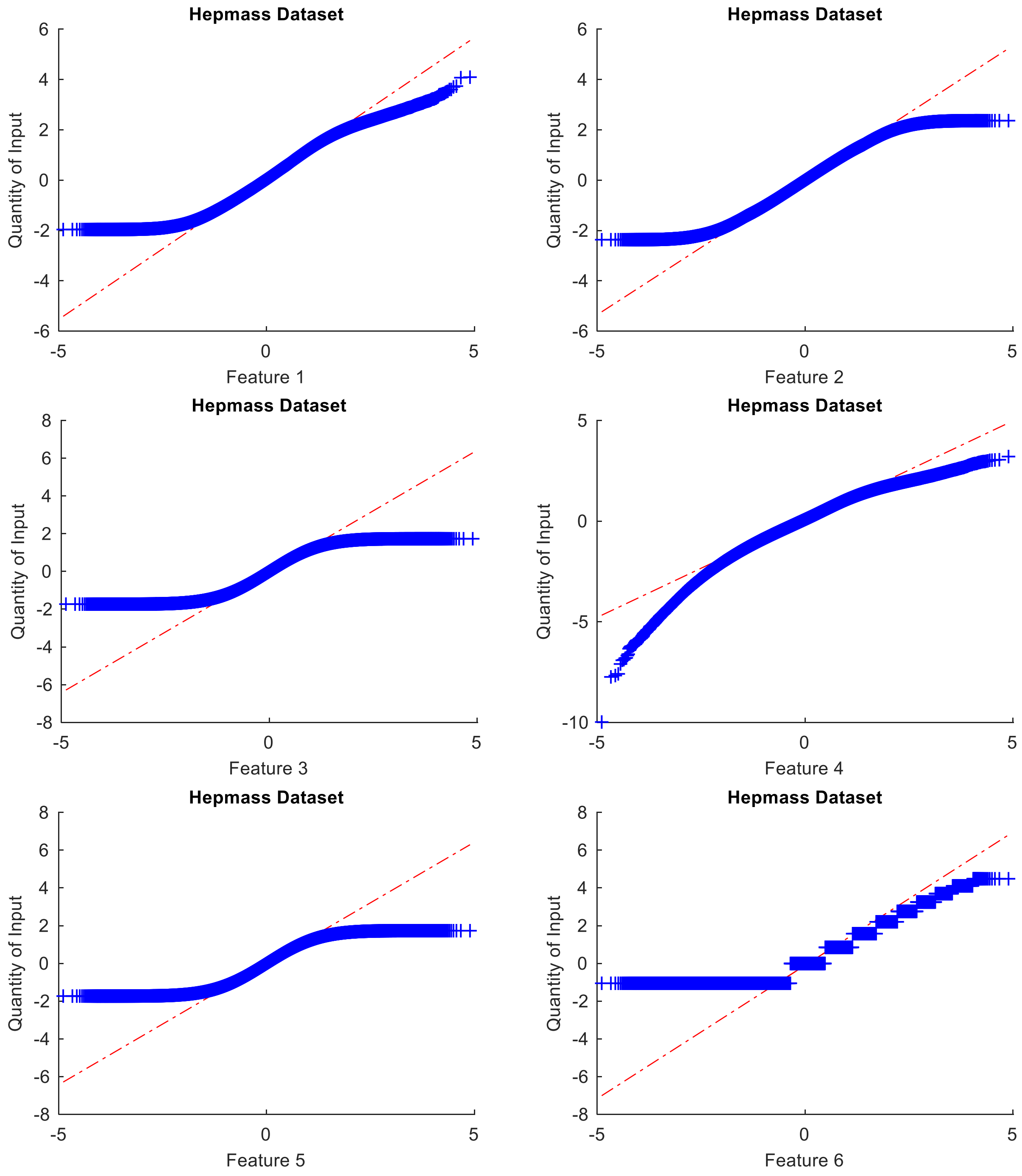


Figure B.4. QQplots for Hepmass dataset

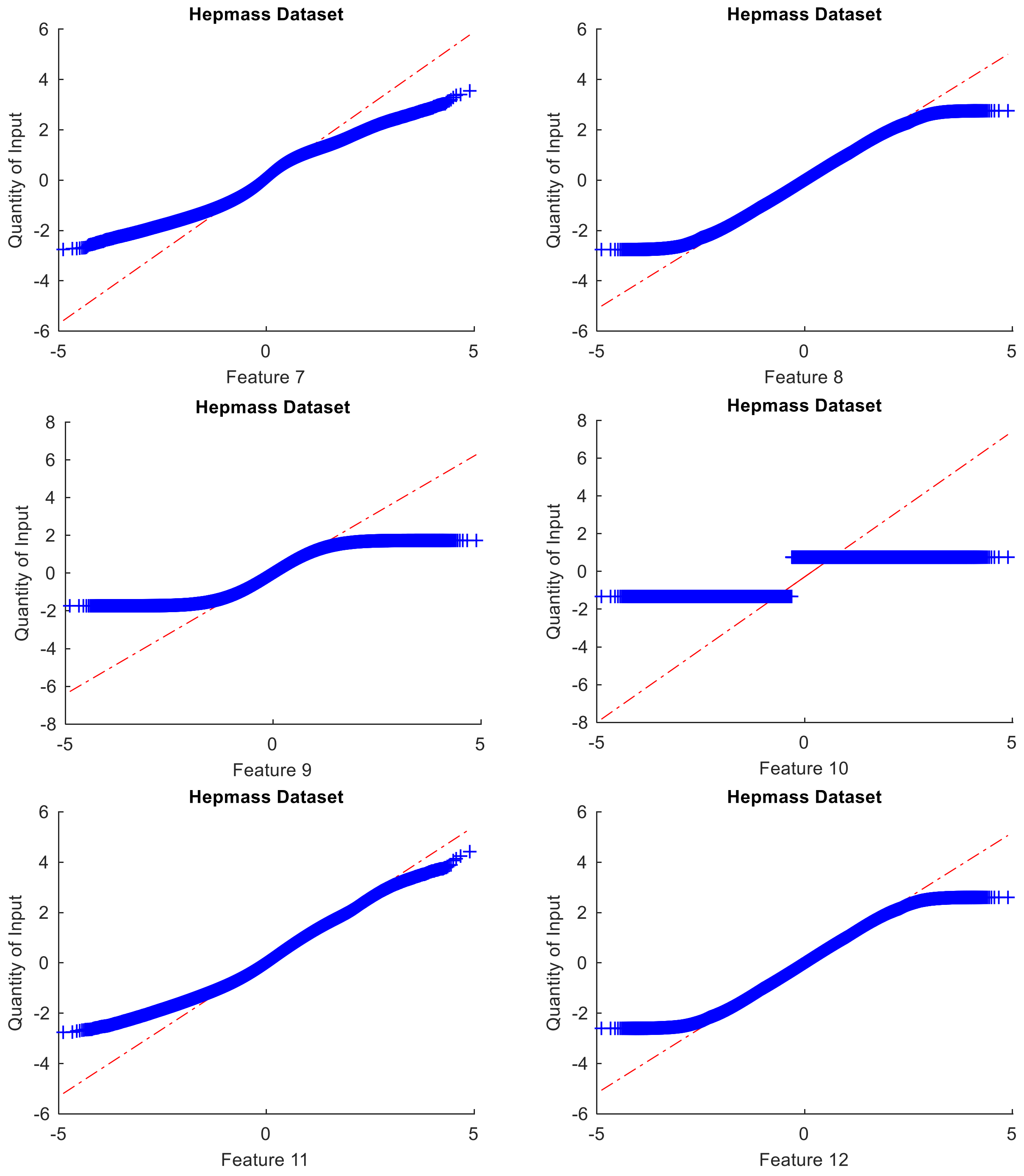


Figure B.4. QQplots for Hepmass dataset (continued)

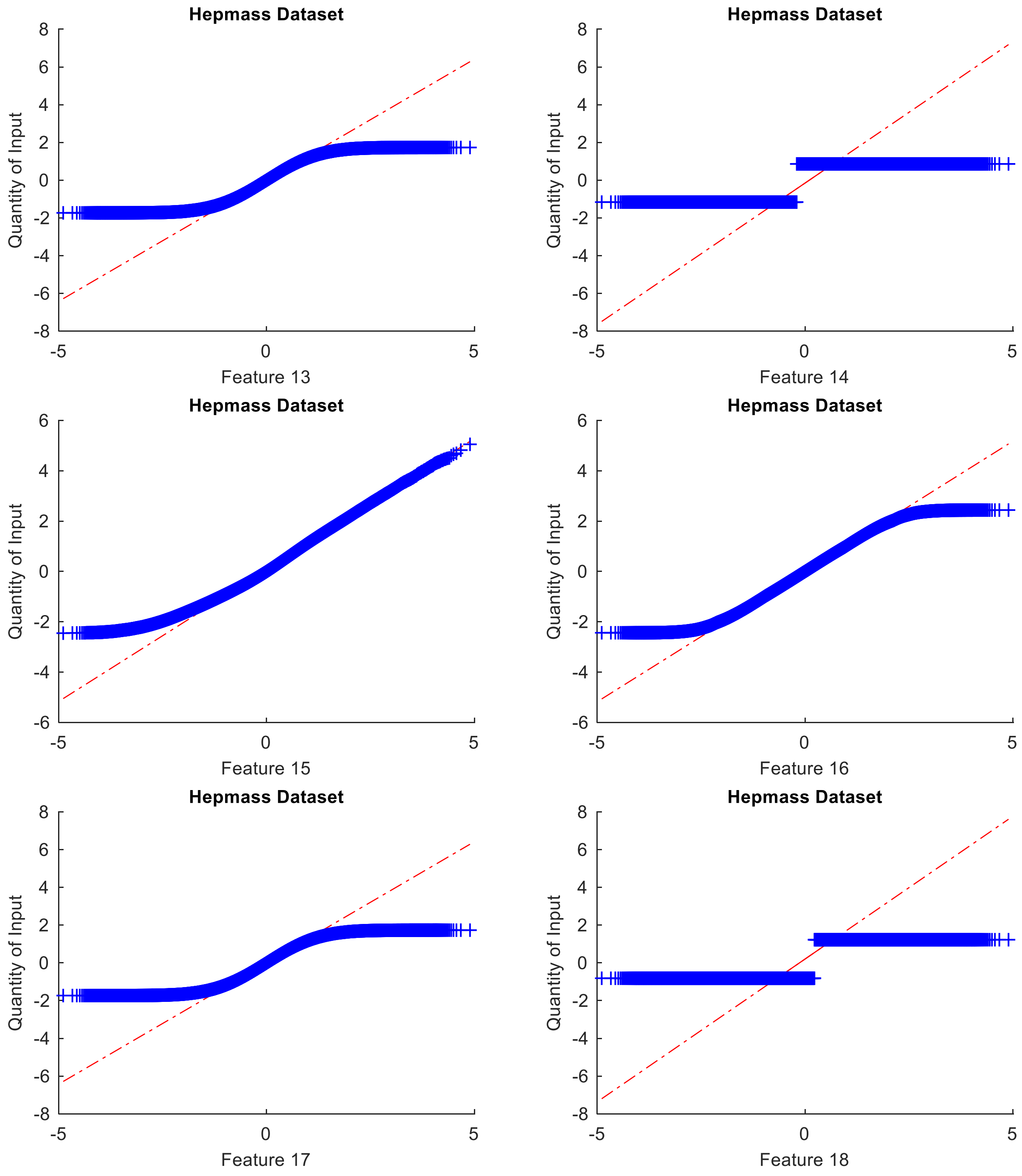


Figure B.4. QQplots for Hepmass dataset (continued)

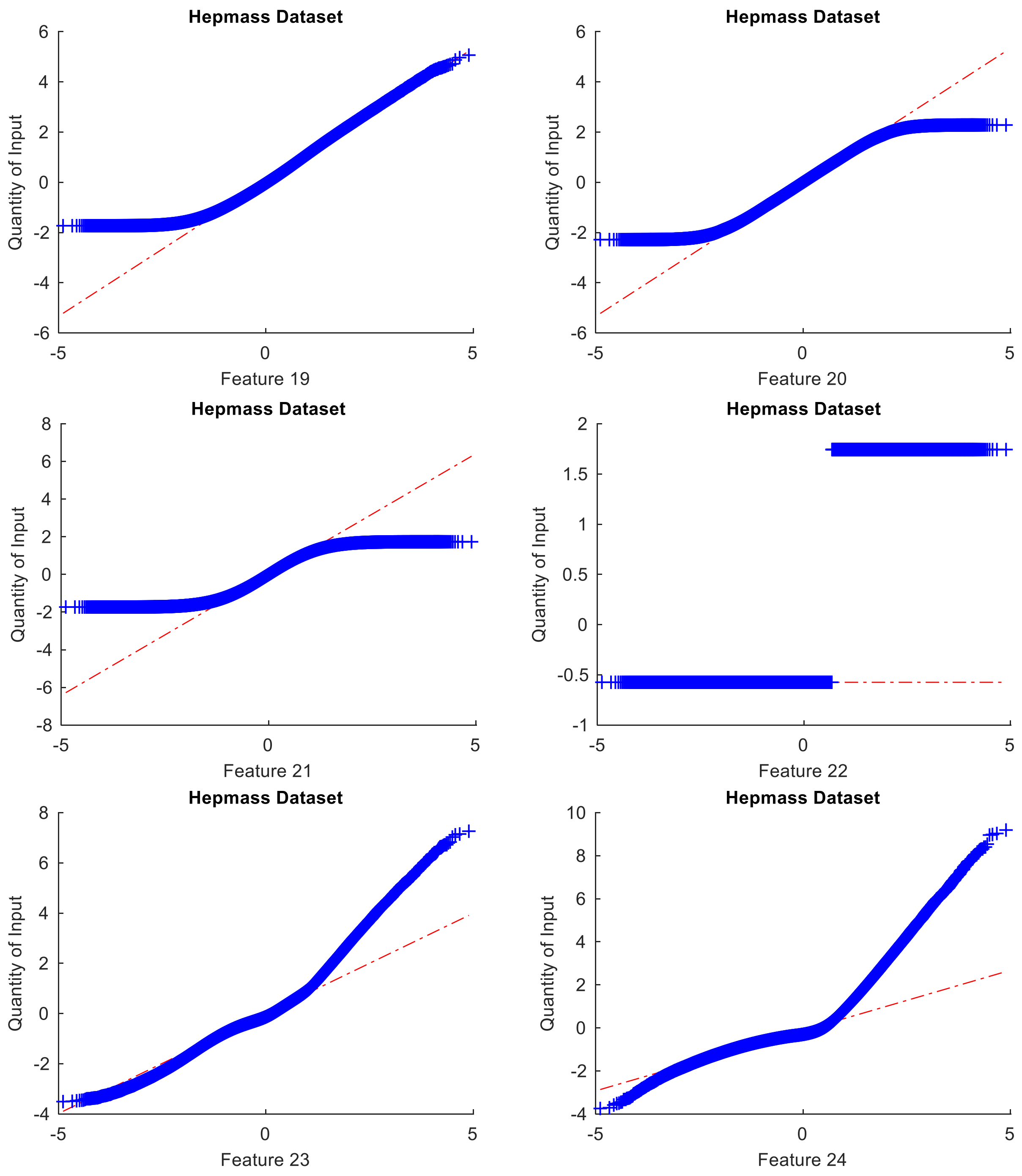
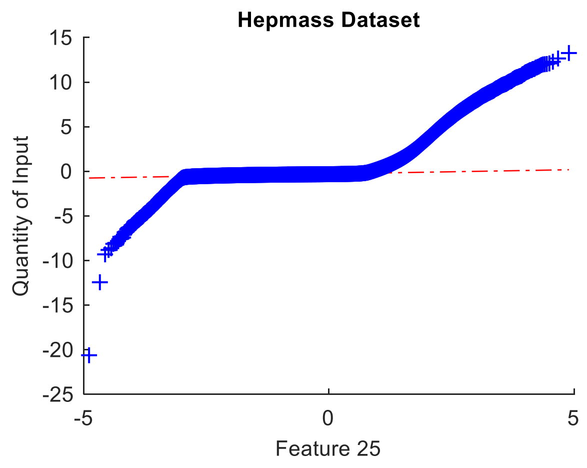
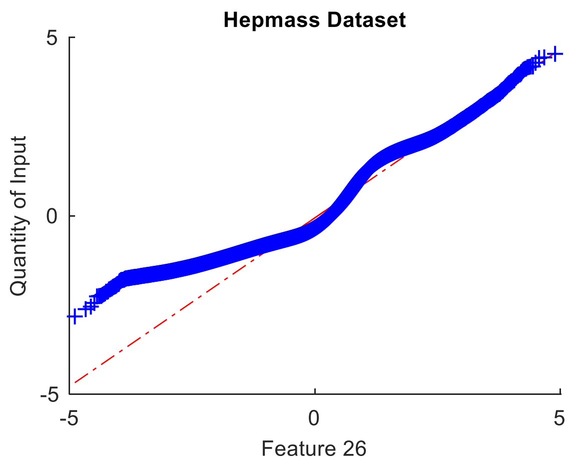


Figure B.4. QQplots for Hepmass dataset (continued)





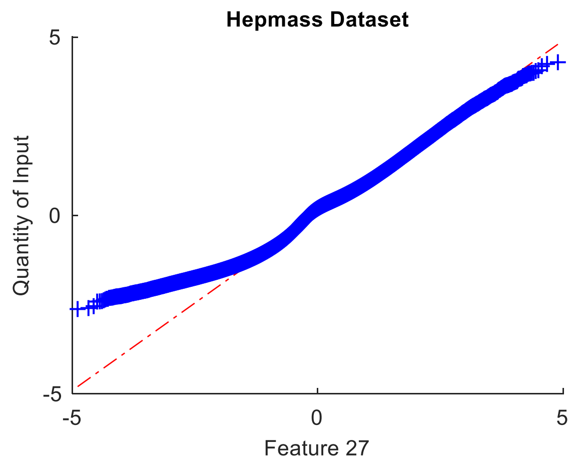


Figure B.4. QQplots for Hepmass dataset (continued)