

Malicious URL(s) classification

MSc Research Project
Cloud Computing

Rohan Dsouza
Student ID: x18139540

School of Computing
National College of Ireland

Supervisor: Dr. Sachin Sharma

National College of Ireland
Project Submission Sheet
School of Computing



Student Name:	Rohan Dsouza
Student ID:	x18139540
Programme:	Cloud Computing
Year:	2018
Module:	MSc Research Project
Supervisor:	Dr. Sachin Sharma
Submission Due Date:	12/12/2019
Project Title:	Malicious URL(s) classification
Word Count:	6192
Page Count:	25

I hereby certify that the information contained in this (my submission) is information pertaining to research I conducted for this project. All information other than my own contribution will be fully referenced and listed in the relevant bibliography section at the rear of the project.

ALL internet material must be referenced in the bibliography section. Students are required to use the Referencing Standard specified in the report template. To use other author's written or electronic work is illegal (plagiarism) and may result in disciplinary action.

I agree to an electronic copy of my thesis being made publicly available on TRAP the National College of Ireland's Institutional Repository for consultation.

Signature:	
Date:	28th January 2020

PLEASE READ THE FOLLOWING INSTRUCTIONS AND CHECKLIST:

Attach a completed copy of this sheet to each project (including multiple copies).	<input type="checkbox"/>
Attach a Moodle submission receipt of the online project submission , to each project (including multiple copies).	<input type="checkbox"/>
You must ensure that you retain a HARD COPY of the project , both for your own reference and in case a project is lost or mislaid. It is not sufficient to keep a copy on computer.	<input type="checkbox"/>

Assignments that are submitted to the Programme Coordinator office must be placed into the assignment box located outside the office.

Office Use Only	
Signature:	
Date:	
Penalty Applied (if applicable):	

Malicious URL(s) classification

Rohan Dsouza
x18139540

Abstract

Malicious URLs are a serious threat to the realm of online security and are one of the most fundamental ways to attack any online user. URL(s) can work as a primary source for distribution of malware/viruses over the internet which has led to an increasing urge for classification of URLs. To prevent users from being attacked, various anti-virus companies use black listing methods and block such URLs at a client end. However, there are millions of malicious URLs that are generated everyday and adding all these URLs in a blacklisting database becomes a monotonous method. Furthermore, it tends to lack newly generated URLs. To solve such problems, machine learning has grabbed attention in recent years to find out the hidden patterns from a dataset of URLs. Although it shows promise, it seems to be inefficient when the size of data is extremely large. This leads to the introduction of big data technologies where we apply machine learning algorithms in a distributed environment. In this research, we have critically compared the performance of traditional machine learning technologies with distributed modern machine learning technologies using Spark MLlib. We have used Logistic regression and Support Vector Machine algorithms in our model to determine the credibility of a URL. Our results conclude that each technique's performance is relative to the size of the data it is working on.

1 Introduction

In the realm of advanced telecommunication technology, the internet has affected every aspect of human lives which range from banking, communication, social media and much more. This has led to the increase in usage of the World Wide Web, which has come with a downside of some malevolent aspects like fraudulent activities performed by online scamsters. For a novice user, internet is a collection of websites that users open via a Uniform Resource Locator (URL) using a client software viz. Browsers. Each website has some human understandable domain name, which when clicked is translated into a machine-readable IP address He et al. (2010). The two main parts of the URL are: protocol identifier (identifies which protocol it is using) and resource name (it is the IP address, indicating the location of the resource. Both identifiers and resource names are separated by colon and two forward slashes.

The internet is loaded with malice activities which include creating websites that are aiding criminal activities such as financial frauds, spam-advertisements to sell products, malware etc. It is estimated that about one-third of the URLs present till date are in one or other way malicious or compromised Sahoo et al. (2017). These activities can be of a wider range where sometimes the hacker tricks the users into revealing their

personal information, stealing valuable data of the individuals and companies, hacking credit cards, denial of services attacks etc. There is a constant need to curb such activities and propose new techniques that can handle such cybersecurity thefts. The inception of all the above mentioned attacks are creation and spreading of malicious URLs over the internet across novice internet users.

The proposed model can be applied at multiple levels where Machine learning can be used on huge amount of data available. An ideal level of this application is aimed at the Domain Registrar level or the Web hosting provider level. Any domain name that has been purchased can be checked against our model in real time for analysis of the Registrant's purpose of registration of the domain name. Also, web hosting providers can analyze logs for traffic coming in for Virtual hosts and map it with our model. Furthermore, a Real-time Blackhole List (RBL) blacklist can be created wherein a thick client like a browser when connected to a website, can detect the legitimacy of that webpage.

The aim of creating these websites with URLs is to attract traffic and bring visitors to visit their websites Ma et al. (2009a). These URLs which act as a doorway to provide unauthorized access to the confidential data of the individuals and companies are termed as malicious URLs Naveen et al. (2019). On the other hand, those websites which are not harmful for the users are termed as benign. In order to prevent users from visiting these malicious websites, the "Blacklisting service" approach was developed which informed the users beforehand about the danger. A variety of techniques have been used for blacklisting including heuristics, web crawlers and manually reporting, but the problem is that it cannot cater to the much larger number of malicious websites. The reason being, there are new websites every day which are never or properly evaluated for its maliciousness. There is a constant need to address the issue of detecting and deleting such malicious websites so that the safety of data can be ensured. In order to cater to this problem, researchers have made use of machine learning techniques, because there are billions of websites and extracting relevant URLs from such a large dataset is not an easy task.

Machine learning is a technique that takes a large sum of data and transforms it into something meaningful Kraska et al. (2013). It is a field of computational science which helps in making reasoning and decision-making process based on provided patterns and structures of data which is presented to it. It has the ability to handle data and provide the desired results which are otherwise impossible to find with normal human comprehension. It solves these problems by learning from its experience and identifies the hidden patterns and underlying structures of the data thus making predictions. Machine learning works in the following manner: It takes the number of URLs as an input of training set, and performs computations based on statistical properties. The machine-learning algorithm then runs on with its predictive function and classifies the URLs as either malicious or benign. The only set of URLs is not sufficient to be represented to the machine learning algorithms, some features need to be extracted as well. These features are sometimes lexical features (consisting of text and words) and sometimes host-based (consisting of some geographical location of the host). These features are then presented to the machine learning algorithm so that it can train its model. Post feature extraction, there are some classification algorithms (which in our case of research are logistic regression and support vector machine) that are applied to the model to get our results.

According to Sahoo et al. (2017), real-world URL data is a form of big data. Due to limited availability of the resource, traditional Machine Learning fails to handle large

dataset. It is almost impossible to train a malicious URL detection model on all the available URL data. No one till date has considered this to be a big data problem. Evaluation time for traditional models may be too high to be practical due to size of URL data. Traditional ML techniques means running ML on a single system with limited memory and compute capacity . Hence, researchers used smaller data sets and/or not the complete dataset. Sahoo et al. (2017) stated that design principles required for real-world malicious URL classification models primarily are : Accuracy, Speed of Detection, Scalability (by using Spark or Hadoop). Distributed machine learning techniques were introduced because local work stations and desktops don't have enough memory and powerful hardware which can accommodate big data in itself. The solution for this problem was proposed by making use of distributed machine learning where each node (i.e. workstation) can perform functions on any subset of the data present on a distributed set of machines Provost and Hennessy (1996). This research is aimed to work on big data specifically in the context of big data of website URLs. A large data set consisting of URLs will be exposed to the system (which is distributed among a number of systems) and results will be recorded. In our research, we are using Apache Spark which is an open-source distributed data processing engine specifically designed to handle large data sets and machine learning tasks *Spark 101: What Is It, What It Does, and Why It Matters* (2018). As the dataset grows in size, it provides more accurate results which are more feasible as well. In order to train data for machine learning algorithm, spark has proven itself as a best choice, because of its ability to store data in memory and iterative learning, thus providing the best possible solution in minimum amount of time which makes it most suitable for machine learning for larger datasets Zaharia et al. (2010). Spark works with different libraries, one of which includes MLlib, a prominent library which is specifically designed for distributed machine learning algorithms. MLlib provides standard support for solving classical machine learning problems including regression, classification, filtering, and clustering because of its iterative nature Meng et al. (2016). MLlib is a scalable open-source library and it works with APIs like java and python. Other APIs like MLI which is a component of MLBase are also present which works on spark and it provides great support for distributed machine learning algorithms including logistic regression and support vector machine.

This research will make use of the MLlib library in order to handle the issue of the identification of malicious and benign URLs. Spark is well known for using a Resilient Distributed Dataset (RDD) by representing a read-only pool of objects which are distributed across a number of machines. This RDD can be cached in memories of machines that are distributed on different locations and can be reused and rebuilt for the purpose of parallel operations and data loss respectively Zaharia et al. (2010). Another open-source software that is used in this research is Hadoop, which will cater to the needs of using distributed machine learning algorithms Patel et al. (2012).

This research is aimed at applying distributed machine learning techniques using Spark Mllib which can exactly classify the URL of the websites as malicious or benign. As the size of the dataset containing information about malicious and non-malicious URLs is large, big data tools and frameworks will be used so that processing of data can be done in an effective manner. We have used two algorithms of distributed machine learning which include Support vector machine (SVM) and Logistic Regression(LR) to determine the accuracy and execution time. The LR algorithm is used when the problem domain is a discrete set of data and when there is a need to address classification problem consisting of Binary sets. It is important to note that accuracy and execution time are the most

important factors which need to be calculated for these two algorithms. In this case, the binary set of data is used to detect whether the specified URL is malicious or benign. Our results concluded that Spark Mllib performed well and will continue to perform better than traditional ML techniques when the size of the data is huge. However, when the size of the data was not as big, traditional ML techniques performed better.

2 Related Work

As discussed in section 1, this research is aimed to apply distributed machine learning techniques on large datasets. The large dataset which will be ingested to the machine learning algorithms in this study will be URLs that are classified as malicious and benign. Researchers have applied different machine learning techniques over the past and contributed to making the identification of malicious and benign URLs easier of research purpose. These techniques include both traditional as well as online machine learning techniques. In this section, a thorough review of those studies will be conducted which have implemented machine learning algorithms in order to identify malicious and benign URLs.

Recent research has been done by Naveen et al. (2019) in which they have successfully implemented the machine learning techniques which provide the ability to judge malicious URLs based on the provided feature set after making use of classification methods. According to them, traditional methods of identifying malicious URLs are not evolving with the introduction of new URLs especially with the existence of the dark web. Their approach took into account the syntactical nature of URL as well as semantic and lexical interpretation of the nature of URLs which are changing dynamically. This classification method has outperformed the already existing techniques for detecting malicious website URLs. However, the dataset they were using had only 18 features which is not as much.

Ma et al. (2009a) has a lot of contribution in the field of detecting malicious and benign URLs using machine learning techniques (i.e. both traditional and distributed). One of the prominent works done by them used various classification models including support vector machine, logistic regression and naive bayes. The lexical features of the URL which were used include IP address properties, WHOIS properties, DNS and geographic properties). For binary classification support vector machine was used and DNS labeling, the logistic regression was used. The classifier which has been obtained as a result shows an accuracy of 95–99%, by detecting a large number of malicious websites. Authors have concluded that the result of this research shows that applying machine learning techniques outperforms the traditional methods of identification of malicious URLs such as blacklists and heuristics.

Ma et al. (2009b) explored online approaches for detecting malicious websites based on lexical and host-based features. To classify the malicious websites, the URL reputation dataset has been used. They have claimed that online algorithms can be accurate as batch processing techniques. They achieved the highest accuracy using the Confidence-Weighted (CW) algorithm against the Support vector machine using different set configurations.

Ma et al. (2011) has shown extensive work done to solve the problem of the identification of malicious URLs. The approach is based on programmed classification of URL by making use of statistical methods. This method has discovered the significant lexical and host-based properties of URLs with malicious websites. The results were obtained

by applying the online learning algorithms and it shows that online learning algorithms learn more efficiently than the dataset using batch algorithms. They have developed a machine-learning algorithm by taking data from real-time systems with labeled URLs. The result of the algorithm has shown the accuracy of 99% as compared to traditional techniques. Previously in another research done by Ma et al. (2009b) has studied the ability of online learning algorithms based on predictive models that have extracted hundreds of features of suspicious URLs and then also analyze them.

While using online learning, another research is done by Blum et al. (2010). A real-time URL extraction has been used in order to detect a phishing URL. Features have been derived automatically from the URLs. A confidence weighted algorithm has been used which is a linear classification method and results were recorded. The model has accurately predicted the phishing URLs as weights were adjusted in the algorithm making it different from other research. The model used by Blum et al. (2010) is based on the prediction model used by Ma et al. (2011).

Vanhoenshoven et al. (2016) considered the detection of malicious URL as the binary classification problem. Various well-known classifiers were studied in order to achieve better prediction rates. Without using the advance feature selection technique they have achieved higher accuracy using random forest followed algorithm by Multi layer perceptron methods. In the result they also specified that classification algorithms achieves higher accuracy when numerical features are used for training.

Kulkarni (2019) used the classification algorithms to classify the URLs as legitimate site, suspicious site or phishing site. They performed this experiment with 1,353 real world URLs. Over 90 of the fake websites were distinguishable from real ones by using their classifiers. MATLAB scripts was written to implement the four classifiers which are Naive Bayes, Support Vector Machine (SVM), decision tree and Neural Networks. The lowest accuracy was achieved using neural network classifier.

A survey has been conducted by Sahoo et al. (2017) in which authors have extensively investigated the whole process of malicious URL detection. This study provides the principles on which the detection of malicious URL works. There are two techniques which include heuristic approach of extraction and another one is applying machine learning algorithms. A prediction model is built which consists of training data having URLs which are malicious and benign. Two types of features are extracted including static and dynamic. In static approach the information of web page was extracted without executing the URL, so this was proved to be an effective approach.

Le et al. (2018) proposed an end to end deep learning framework using URLNet. To learn the URL string, they applied convolution neural networks for both character and words of the URL and jointly optimized the network.

A CatchPhish approach has been proposed by Rao et al. (2019) according to which they have proposed a light weight approach to detect the phishing sites. The novelty of this approach is that it checks the legitimacy of the URL even when the website has not yet visited. It makes use of following dimensions including: complete URL of the website, hostnames, those words which are phish-hunted, and Term Frequency-Inverse Document Frequency (TF-IDF). Results show that by only making use of Term Frequency-Inverse Document Frequency (TF-IDF), the accuracy of 93.25% has been achieved.

He et al. (2010) has used the following models in his research including logistic regression, random forest, decision tree, and Markov models. Two types of features have been extracted including textual and zone. The challenge reported by He et al. (2010) in the classification of malicious and benign URL is twofold: the malicious URL should be

identified without wasting resources and classification should be done before providing the web page content knowledge. The classifier developed by them has shown better results while capturing malicious URLs with a low false-positive rate. The accuracy was not given much importance in this work as it was not their target.

A series of experiments have been done by Dong et al. (2017) by putting the URL under the microscope. They have extracted the features of URL including host domain name, user agent, URL path and URL parameters. Hashing and sorting methods are used and URLs are classified into malware and clean traffic. Both supervised and unsupervised machine learning methods have been applied. While applying supervised learning the new data and labels were introduced and system relearned. In unsupervised learning, only clustering was used. The authors have taken the approach of a semi-supervised clustering approach with pruning. Results show that over the dataset of 950,000 URLs the detection rate of malicious with benign was 84%.

Another contribution is made by Feroz and Mengel (2015) that has implemented a hybrid approach that combines the machine learning approach of both classification and clustering. First clustering is performed cluster IDs are collected. These IDs are used to obtain high classifier efficiency for the purpose of generalizing. This paper has also made use of Microsoft Reputation Services (MRS) for the purpose of categorizing URLs that are malicious. The URLs have been ranked in this approach and results show that the accuracy of 93-98% has been achieved after implementing the clustering and classification techniques.

A new approach of machine learning has been used for detecting malicious URLs is by Choi et al. (2011). The malicious URLs have been identified by their attack types and the nature of the attack with which they attempt to launch. The data was collected from a real-time website consisting of 42000 and 32000 benign and malicious URLs respectively. The learning algorithms used by Choi et al. (2011) is a binary classification method and the other one is a support vector machine algorithm. Both network and DNS features were extracted. Support vector machine-assisted in the identification of malicious URLs whereas other algorithms were used for attack type identification.

There is a number of ways in which malicious URLs are propagated across the systems so that users can click on them. One way of propagating malicious URLs is via email. The legitimate content is presented to the users with malicious URLs sometimes. The target of sending such URLs via email is to steal the data of the users including names, passwords, and details about financial records such as credit card numbers. Ranganayakulu and Chellappan (2013) proposed a URL analyzer method that extracts the lexical as well as host-based features from the emails. Almost 92% accuracy was achieved after applying this method.

In order to cater to the needs of this research, which is handling big data, different approaches have been discussed. As the volume of enterprises is growing at an exponential rate, so is the need to store, process and analyzing this data is increasing for a number of reasons. To process big data, parallel processing is required, along with distributed storage of data as well Patel et al. (2012). A number of tools and techniques have been introduced by the researchers which fulfill the need to process big data on distributed systems. As our research uses Spark, another tool or framework that is required for distributed processing of data is Hadoop. Hadoop has the ability to work on thousands of computers working independently for the purpose of computation and data which is available in petabytes. The filing system which will be used by this research is the Hadoop Distributed File System (HDFS). Data on the HDFS is saved across hundreds

and thousands of servers, which also provide a mean of working. The master and slave architecture is supported by the Hadoop Distributed File System which makes it more suitable for the distributed storage of data.

The problem of dealing with big data is that it is messy and very much diverse. In order to run large scale SQL queries, distributed processing of data and most importantly learning part of running the machine learning algorithm is needed and a special engine is required to serve the purpose. This special engine used by this research is Spark. Spark has the same working principle of other engines like MapReduce Zaharia et al. (2010), but Spark runs with the libraries making it more efficient and easier. Assefi et al. (2017) compared performance of Apache Spark MLlib and Weka distributed ML with multiple Supervised and Unsupervised ML algorithms on multiple big-data datasets. However, he did not compare the performance of Spark MLlib with any traditional ML technique. Spark has gained popularity in the last decade because of a number of reasons including the ability to use unified APIs, it efficiently syndicates the scalable processing units, and most important streaming machine learning Zaharia et al. (2010). Because of the high quality of features of Spark like it is fault-tolerant, supports high-level libraries, and supports the number of applications, we decided to make use of it.

3 Methodology

To gain further insights into the methodology of our thesis, we have broken it down into various subsections. The primary goal of our research is to evaluate the performance between our ML model which is in a distributed environment and the traditional machine learning technologies. We shall do so by using several metrics mentioned below in order to comprehend our results on different sizes of data. Figure 1 gives a holistic view of our approach.

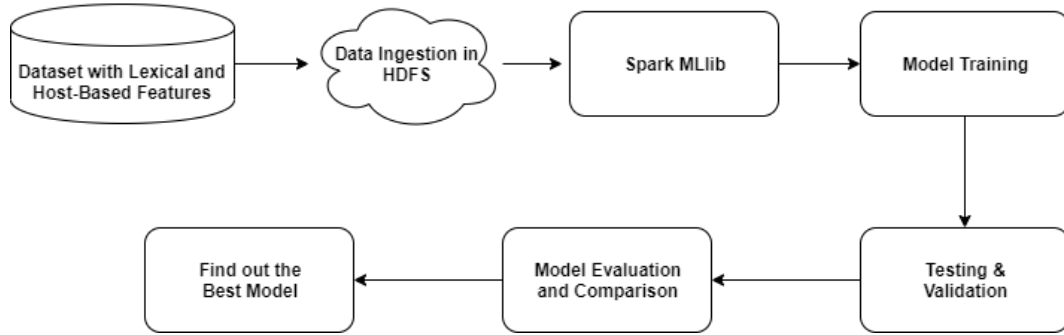


Figure 1: Methodology

3.1 Data Ingestion in HDFS

To primarily enable distributed machine learning, data should be placed across multiple servers. To achieve that we are using Hadoop Distributed File System (HDFS) which is an important component of the Hadoop ecosystem. HDFS is a file system where immense amount of data can be stored with high availability. HDFS automatically distributes the data across multiple servers in the form of blocks where the number of block depends on the block size. One more reason to use the HDFS is its scalability and fault tolerant mechanisms. The data from local system to HDFS can be ingested either with the help of sqoop or the hadoop binary.

3.2 Spark MLlib

In order to process this humongous amount of data available in HDFS, spark engine has been used. Spark performs in-memory computation for execution of queries. Furthermore, spark is well known for its resilient distributed dataset (RDD). Transformation and actions are the important steps in spark to work on RDD. Integration of various libraries with spark, enables it to perform multiple operations on datasets in an effective manner. Spark MLlib library enables us to apply the machine learning algorithms in a distributed environment. MLlib contains high-quality algorithms that leverage iteration and produce better results.

3.3 Model Training

Detecting malicious URL is a binary classification problem. Various supervised machine learning algorithms can be applied in order to predict the results. We are using logistic regression and support vector machine to classify the URL. All these algorithms will be

used to train the model on dataset for both traditional as well as distributed machine learning methods. In order to predict the results on unseen data, the test dataset will be used which is important to evaluate the model performance. The dataset is randomly split into training (60%) and test (40%) data each time we run our code. Our model first trains on the training data with the features and its corresponding labels and then predicts the label for the test data.

3.4 Model Evaluation

Various metrics can be used in order to evaluate the performance of our model based on the labels it predicted. We will use the confusion matrix which involves True Positives (TP), True Negatives (TN), False Positives (FP) and False Negatives (FN) for our evaluation. Some of the metrics which will be used for our evaluation are as follows:

3.4.1 Precision

Precision is the fraction of the urls that were predicted to be malicious that are actually malicious. Precision can be thought of as a measure of a classifiers exactness. A low precision can also indicate a large number of False Positives. Formula for precision is:

$$Precision = TP / (TP + FP)$$

3.4.2 Recall

Recall can be thought of as a measure of a classifiers completeness. A low recall indicates many False Negatives. Formula for recall is:

$$Recall = TP / (TP + FN)$$

3.4.3 F1-score

F1 score conveys the balance between the precision and the recall. The formula for calculating f1-score is:

$$F1score = 2 * ((precision * recall) / (precision + recall))$$

3.4.4 Accuracy

It is the number of correct predictions made divided by the total number of predictions made, multiplied by 100 to get a percentage. The formula for calculating the accuracy is:

$$Accuracy = (TP + TN) / (TP + FP + FN + TN)$$

3.5 Dataset Description

The publicly available URL dataset can be found at: <http://archive.ics.uci.edu/ml/datasets/URL+Reputation> to decide whether the URL is malicious or benign. Several features can be extracted from the dataset. In our dataset there are two kind of features viz. Lexical Features and Host Based Features which are described in Figure 2.

The dataset consist of about 2.4 million URLs (examples) and 3.2 million features. The lexical features describes the lexical properties of URL such as the length of the URL, number of dots. It also contains the bag of words features where the features will be hostname, primary domain name, path token and so on. The Host based features consist of the features as WHOIS information, IP prefix, connection speed and so on. The Figure 2 show the description of each feature in detail.

Lexical		Host-Based	
Feature type	Count	Feature type	Count
Hostname	835,764	WHOIS info	917,776
Primary domain	738,201	IP prefix	131,930
Path tokens	124,401	AS number	39,843
Last path token	92,367	Geographic	28,263
TLD	522	Conn. speed	52
Lexical misc.	6	Host misc.	37
Lexical	1,791,261	Host-Based	1,117,901

Figure 2: Feature description of the Dataset. Source: Ma et al. (2009b)

4 Implementation

For our Thesis implementation, I am using our college’s in-house Private cloud setup based on an opensource cloud platform named Openstack. We shall be using the Nova service of the Openstack Cloud Platform along with the Cinder service for persistent storage allotted to our Virtual Machine (VM). Our instance would be of the following specifications:

- Instance Name: x18139540-Thesis
- Operating System: Ubuntu 18.04 LTS Bionic Beaver 64 bit
- Instance Flavour: m1.xlarge
- CPU: 8 VCPU’s
- RAM: 16 GiB
- Storage: 160 GiB
- Network: MSCCLOUD-net1

To get access to the VM, a Public Floating IP address from the public1 subnet has been attached to the IP. Since the dataset is huge as mentioned above, it shall be ingested in an opensource distributed file system named Hadoop Distributed File System by Apache Hadoop. We shall be using a stable build of Apache Hadoop 2.9.2 for this implementation. Post ingesting the huge dataset in HDFS, we are using Apache Spark 2.4.4 to apply Machine Learning algorithms on it. We are using Jupyter notebooks as an IDE for writing the code in Python 2. Since Jupyter notebooks run on port 8888 which is blocked at the enterprise firewall, we have changed it to run on port 80.

5 Results and Evaluation

In order to predict the URL as malicious or benign, we have implemented Logistic regression and SVM using spark as well traditional machine learning methods. To evaluate the performance of our model, various metrics have been used such as time, precision, recall, f1-score and accuracy. Several experiments has been performed ranging from small to large number of records to find as to when and where do traditional and our spark model perform well. For a comparative analysis, our primary focus would be on time and accuracy.

5.1 Experiment 1 / Results of Logistic Regression Using Traditional ML

In our first experiment we used traditional machine learning methods wherein we apply logistic regression on a dataset ranging from small to large number of records. The result with different number of records using various metrics have been shown in Table 1

Table 1: Results of Logistic Regression Using Traditional ML

Traditional ML (Logistic Regression)					
No of Records	Precision	Recall	F1-Score	Accuracy	Time(in sec)
1000	96.45%	96.75%	96.27%	96.75%	0.103
10000	96.68%	96.65%	96.65%	96.65%	0.85
100000	95.86%	95.86%	95.86%	95.86%	2.81
1000000	96.21%	96.21%	96.21%	96.21%	87
1500000	95.97%	95.86%	95.87%	95.87%	134.5
2000000	95.96%	95.97%	95.96%	95.97%	212.77

The precision, recall, F1-score and accuracy in logistic regression using traditional Machine learning approach lies between 95-96%. For small data size, logistic regression with traditional machine learning approach calculates the results in milliseconds. In case of large dataset (for 2 million number of records) the highest time taken by algorithm is 212.77 seconds with the accuracy of 95.96%.

5.2 Experiment 2 / Results of Logistic Regression Using Spark MLlib

In this experiment, Logistic regression with Spark MLlib has been used. The time taken to train the model is 11.04 seconds for 1000 number of records with the accuracy of 95.17%. The highest time consumed to train the model is 173.67 sec for 2 million number of records.

Table 2: Results of Logistic Regression Using Spark MLlib

Spark MLlib (Logistic Regression)					
No of Records	Precision	Recall	F1-Score	Accuracy	Time(in sec)
1000	95.17%	93.87%	94.52%	96.15%	11.04
10000	94.91%	95.86%	95.38%	96.25%	15.64
100000	93.64%	94.16%	93.90%	95.70%	38.98
1000000	94.53%	94.40%	94.46%	96.10%	101
1500000	95.79%	94.07%	93.48%	93.77%	135.38
2000000	94.25%	93.40%	93.82%	95.93%	173.67

It has been observed that traditional machine learning works best, when the size of data is small. The case where the dataset size increases upto 2million, it takes high time to train and test the model when compared to spark. Traditional machine learning required 212.77 sec to train and test the LR model whereas, Spark required only 173.67 sec. This time difference between these methods shows that traditional machine learning fails when the size of the data is extremely large and spark outperformed in our case but by a low margin.

5.3 Experiment 3 / Results of SVM Using Traditional ML

In this experiment Support Vector Machine algorithm is implemented using traditional machine learning method for finding results, which are shown in Table 3

Table 3: Results of SVM Using Traditional ML

Traditional ML (SVM)					
No of Records	Precision	Recall	F1-Score	Accuracy	Time(in sec)
1000	95.75%	96.22%	95.75%	95.80%	0.059
10000	95.45%	95.67%	95.45%	95.46%	1.67
100000	96.40%	96.41%	96.40%	96.40%	159.1
1000000	97.23%	97.23%	97.23%	97.23%	14453.29
1500000	97.26%	97.26%	97.26%	97.26%	33322.88
2000000	97.30%	97.30%	97.30%	97.30%	54941.55

The precision, recall, f1-score and accuracy are almost same for different number of records. It has been observed that SVM takes least time when the size of data is small. As the data size grows, the time taken increases exponentially. Table 3 indicates that, SVM with traditional ML techniques achieves higher accuracy and predicts the output in milliseconds for small number of records. For larger number of records, SVM takes a significant amount of time.

5.4 Experiment 4 / Results of SVM Using Spark MLlib

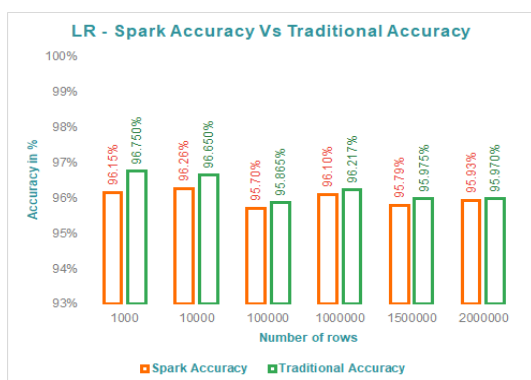
Support Vector Machine (SVM) has been applied with the Spark MLlib. In case of Spark, SVM performs the job in minimum time. However, in our experiments, SVM

with spark MLlib showed less accuracy, precision, recall and f1-score when compared to traditional approaches when we gradually increased the size of our dataset.

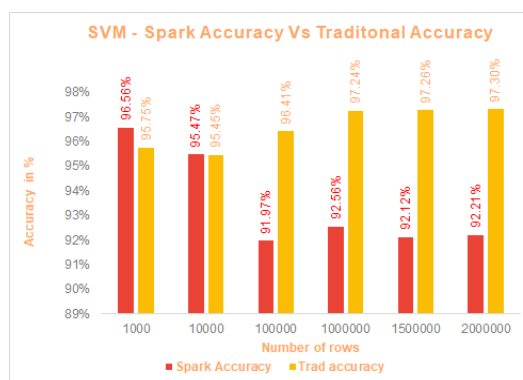
Table 4: Results of SVM Using Spark MLlib

Spark MLlib (SVM)					
No of Records	Precision	Recall	F1-Score	Accuracy	Time(in sec)
1000	96.56%	91.19%	95.18%	95.39%	9.16
10000	95.46%	91.36%	98.64%	94.86%	11.9
100000	91.97%	82.72%	97.55%	89.53%	33.07
1000000	92.55%	84.16%	97.26%	90.24%	105.11
1500000	92.12%	82.82%	96.92%	89.32%	115.2
2000000	92.2%	82.57%	96.95%	89.19%	152.04

After analysing all the results it has been observed that, traditional machine learning performs well when only a small amount of data is to be worked on. It fails when a large amount of data is taken into consideration. In case of Spark MLlib, it takes more time when trained for small amount of data. However, in today’s day and age when data is generated from multiple sources and is huge, applying Machine learning using distributed techniques with Spark would be a good idea as it would reduce training and testing time significantly without hampering the accuracy of the model.

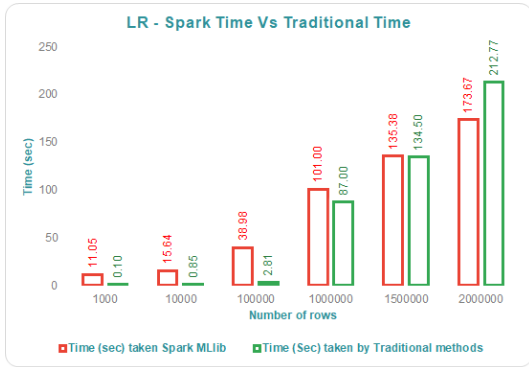


(a) Accuracy comparison for Logistic Regression

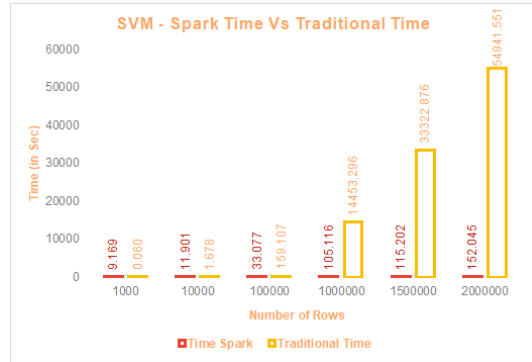


(b) Accuracy comparison for SVM

Figure 3: Graphical comparison of Accuracy v/s Number of rows for Spark MLlib and Traditional ML techniques



(a) Execution time comparison for Logistic Regression



(b) Execution time comparison for SVM

Figure 4: Graphical comparison of Execution time v/s Number of rows for Spark MLlib and Traditional ML techniques

Figure 3 and Figure 4 give a holistic view of the accuracy and the total execution time taken by Spark MLlib and traditional ML techniques for both Logistic regression and SVM. While in Logistic regression, we see that Spark performs better marginally as the time difference is not as much, for SVM, spark outperforms traditional ML techniques outrightly. In case of SVM, time taken by traditional ML techniques increases exponentially as the number of records are increased subsequently.

6 Conclusion and Future Work

After experimenting with the URL reputation dataset, we were able to compare our Distributed Machine learning model with the traditional Machine learning model using Logistic Regression and SVM algorithms. We came to a conclusion that when the data being worked on was relatively small, traditional machine learning algorithms were more efficient and performed computation in significantly lesser time. This is caused by the overhead caused by Spark and Java wherein a lot of complexity is wasted for small computations. However, when we increased the size of the data being worked on gradually, Spark was outperforming traditional machine learning techniques.

Due to lack of availability of nodes while performing the experiments, Spark was installed in a standalone mode i.e the Master and the worker were installed on a single node. However, in the future we plan to repeat this experiment with multiple nodes and with Spark installed in a distributed environment where we see Spark outperforming traditional ML techniques by a very high margin as the compute capacity would be much more. To get into a detailed analysis, we would also perform experiments using various ML algorithms viz. Naive Bayes, Random Forest etc. Furthermore, since our code was written in Python 2, Python 3 support will be added as Python 2 has reached EOL on 1st January, 2020.

References

- Assefi, M., Behravesh, E., Liu, G. and Tafti, A. P. (2017). Big data machine learning using apache spark mllib, *2017 IEEE International Conference on Big Data (Big Data)*, IEEE, pp. 3492–3498.
- Blum, A., Wardman, B., Solorio, T. and Warner, G. (2010). Lexical feature based phishing url detection using online learning, *Proceedings of the 3rd ACM Workshop on Artificial Intelligence and Security*, ACM, pp. 54–60.
- Choi, H., Zhu, B. B. and Lee, H. (2011). Detecting malicious web links and identifying their attack types., *WebApps* **11**(11): 218.
- Dong, H., Shang, J., Yu, D. and Lu, L. (2017). Beyond the blacklists: Detecting malicious url through machine learning, *Proceedings of BlackHat Asia* .
- Feroz, M. N. and Mengel, S. (2015). Phishing url detection using url ranking, *2015 IEEE international congress on big data*, IEEE, pp. 635–638.
- He, Y., Zhong, Z., Krasser, S. and Tang, Y. (2010). Mining dns for malicious domain registrations, *6th International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom 2010)*, IEEE, pp. 1–6.
- Kraska, T., Talwalkar, A., Duchi, J. C., Griffith, R., Franklin, M. J. and Jordan, M. I. (2013). Mlbase: A distributed machine-learning system., *Cidr*, Vol. 1, pp. 2–1.
- Kulkarni, A. (2019). Phishing websites detection using machine learning.
- Le, H., Pham, Q., Sahoo, D. and Hoi, S. C. (2018). Urlnet: learning a url representation with deep learning for malicious url detection, *arXiv preprint arXiv:1802.03162* .
- Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2009a). Beyond blacklists: learning to detect malicious web sites from suspicious urls, *Proceedings of the 15th ACM SIGKDD international conference on Knowledge discovery and data mining*, ACM, pp. 1245–1254.
- Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2009b). Identifying suspicious urls: an application of large-scale online learning, *Proceedings of the 26th annual international conference on machine learning*, ACM, pp. 681–688.
- Ma, J., Saul, L. K., Savage, S. and Voelker, G. M. (2011). Learning to detect malicious urls, *ACM Transactions on Intelligent Systems and Technology (TIST)* **2**(3): 30.
- 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.
- Naveen, I. N. V. D., Manamohana, K. and Verma, R. (2019). Detection of malicious urls using machine learning techniques, *International Journal of Innovative Technology and Exploring Engineering* **8**(4S2): 389–393.
- Patel, A. B., Birla, M. and Nair, U. (2012). Addressing big data problem using hadoop and map reduce, *2012 Nirma University International Conference on Engineering (NU-iCONE)*, IEEE, pp. 1–5.

- Provost, F. J. and Hennessey, D. N. (1996). Scaling up: Distributed machine learning with cooperation, *AAAI/IAAI, Vol. 1*, Citeseer, pp. 74–79.
- Ranganayakulu, D. and Chellappan, C. (2013). Detecting malicious urls in e-mail—an implementation, *AASRI Procedia* **4**: 125–131.
- Rao, R. S., Vaishnavi, T. and Pais, A. R. (2019). Catchphish: detection of phishing websites by inspecting urls, *Journal of Ambient Intelligence and Humanized Computing* pp. 1–13.
- Sahoo, D., Liu, C. and Hoi, S. C. (2017). Malicious url detection using machine learning: a survey, *arXiv preprint arXiv:1701.07179* .
- Spark 101: What Is It, What It Does, and Why It Matters* (2018). Available at: <https://mapr.com/blog/spark-101-what-it-what-it-does-and-why-it-matters/>. Accessed: 2019-11-11.
- Vanhoenshoven, F., Nápoles, G., Falcon, R., Vanhoof, K. and Köppen, M. (2016). Detecting malicious urls using machine learning techniques, *2016 IEEE Symposium Series on Computational Intelligence (SSCI)*, IEEE, pp. 1–8.
- Zaharia, M., Chowdhury, M., Franklin, M. J., Shenker, S. and Stoica, I. (2010). Spark: cluster computing with working sets in: Proceedings of the 2nd usenix conference on hot topics in cloud computing, 10–10, *USENIX Association, Berkeley, CA, USA* .

Appendices : Configuration Manual

A Installation of Hadoop Distributed File System (HDFS)

Post creation of Ubuntu 18.04 VM and getting access to it, kindly do the following steps.

Note: Please login as root user or use sudo to perform below mentioned actions. This will prevent any permission issues.

- Step 1: Install Java Run the following command: **apt install openjdk-8-jdk**

A terminal window showing the command 'apt install openjdk-8-jdk' and its output. The output lists a large number of packages to be installed, including gtk-update-icon-cache, libatk-bridge2.0, libatk-wrapper-java, libatk-wrapper-java-jni, libatk1.0, libatk1.0-data, libatspi2.0, libavahi-client3, libavahi-common-data, libavahi-common3, libcairo2, libcroco3, libcups2, libdatrie1, libdrm-amdgpu1, libdrm-intel1, libdrm-nouveau2, libdrm-radeon1, libflac8, libfontconfig1, libfontenc1, libgail-common, libgail18, libgdk-pixbuf2.0, libgdk-pixbuf2.0-bin, libgdk-pixbuf2.0-common, libgl1, libgl1-mesa-dri, libgl1-mesa-glx, libglapi-mesa, libglvnd0, libglx-mesa0, libglx0, libgraphite2-3, libgtk2.0, libgtk2.0-bin, libgtk2.0-common, libharfbuzz0b, libice-dev, libice6, libjbig0, libjpeg-turbo8, libjpeg8, liblcms2-2, libllvms8, libnspr4, libnss3, libogg0, libpango-1.0, libpangocairo-1.0, libpangoft2-1.0, libpcaaccess0, libpcsclite1, libpixman-1-0, libpthread-stubs0-dev, libpulse0, librsvg2-2, librsvg2-common, libsensors4, libsm-dev, libsm6, libsndfile1, libthai-data, libthai0, libtiff5, libvorbis0a, libvorbisenc2, libx11-dev, libx11-doc, libx11-xcb1, libxau-dev, libxaw7, libxcb-dri2-0, libxcb-dri3-0, libxcb-glx0, libxcb-present0, libxcb-render0, libxcb-shape0, libxcb-shm0, libxcb-sync1, libxcb1-dev, libxcomposite1, libxcursor1, libxdamage1, libxdmcp-dev, libxf86-dga1, libxf86-dga1-dev, libxf86-dga1-xorg-dev, libxinerama1, libxmu6, libxpm4, libxrandr2, libxrender1, libxshmfence1, libxt-dev, libxt6, libxtst6, libxv1, libxxf86dga1, libxxf86vm1, openjdk-8-jdk, openjdk-8-jdk-headless, openjdk-8-jre, openjdk-8-jre-headless, ubuntu-mono, x11-common, x11-utils, x11proto-core-dev, x11proto-dev, xorg-sgml-doctools, xtrans-dev. It also shows suggested packages and a confirmation prompt 'Do you want to continue? [Y/n]'. The user has entered 'Y'.

Figure 5: Installation of Java

Enter Y if asked for a confirmation prompt.

- Step 2: Make the following changes in /etc/sysctl.conf and then type the command: **sysctl -p** to re-read the sysctl.conf file.

A terminal window showing the configuration of sysctl.conf. The user enters 'tail -n 3 /etc/sysctl.conf' and the output is: net.ipv6.conf.all.disable_ipv6 = 1, net.ipv6.conf.default.disable_ipv6 = 1, net.ipv6.conf.lo.disable_ipv6 = 1. Then the user enters 'sysctl -p' and the output is the same three lines. The prompt 'root@thesis:~#' is visible throughout.

Figure 6: Changes in the Kernel Module

- Step 3: Make changes in /etc/profile to add a global path for JAVA_HOME variable

```
root@thesis:~# echo "JAVA_HOME=/usr" >> /etc/profile
root@thesis:~# source /etc/profile
root@thesis:~# echo $JAVA_HOME
/usr
root@thesis:~# █
```

Figure 7: Defining JAVA_HOME variable globally

- Step 4: Add user hduser and group hadoopgroup

```
hduser@thesis:~$ su - hduser
hduser@thesis:~$ groupadd hadoopgroup
hduser@thesis:~$ useradd -s /bin/bash -m -g hadoopgroup hduser
hduser@thesis:~$ ssh-keygen -t rsa
Generating public/private rsa key pair.
Enter file in which to save the key (/home/hduser/.ssh/id_rsa):
Created directory /home/hduser/.ssh.
Enter passphrase (empty for no passphrase):
Enter same passphrase again:
Your identification has been saved in /home/hduser/.ssh/id_rsa.
Your public key has been saved in /home/hduser/.ssh/id_rsa.pub.
The key fingerprint is:
SHA256:z0L6+l2B15ubBy041y9kzK13h6Eg1653urrXbZc8c hduser@thesis
The key's randomart image is:
+--[RSA 2048]-----
```

Figure 8: Adding hduser and hadoopgroup

We shall be running HDFS using the user hduser as running it as root is not a best practise. Furthermore, ssh keys have also been generated for this user.

- Step 5: Download Hadoop
Move the extracted hadoop directory to /usr/local so that everyone has access to it. Furthermore, change the owner and group of the directory so that the hduser has access to the files under that directory.

```
hduser@thesis:~$ wget http://ftp.heanet.ie/mirrors/www.apache.org/dist/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz
2019-11-09 13:35:15 -- http://ftp.heanet.ie/mirrors/www.apache.org/dist/hadoop/common/hadoop-2.9.2/hadoop-2.9.2.tar.gz
Resolving ftp.heanet.ie (ftp.heanet.ie)... 67.44.34.235
Connecting to ftp.heanet.ie (ftp.heanet.ie)|67.44.34.235|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 366447449 (349M) [application/x-gzip]
Saving to: 'hadoop-2.9.2.tar.gz'

hadoop-2.9.2.tar.gz 100%[=====] 349.47M 2.62MB/s in 2m 28s
2019-11-09 13:37:43 (2.36 MB/s) - 'hadoop-2.9.2.tar.gz' saved [366447449/366447449]
hduser@thesis:~$ tar -xvf hadoop-2.9.2.tar.gz > /dev/null
hduser@thesis:~$ ls
hadoop-2.9.2 hadoop-2.9.2.tar.gz
hduser@thesis:~$ cd hadoop-2.9.2
root@thesis:~/hadoop-2.9.2$ mv /home/hduser/hadoop-2.9.2 /usr/local/
root@thesis:~/hadoop-2.9.2$ ln -sf /usr/local/hadoop-2.9.2 /usr/local/hadoop
root@thesis:~/hadoop-2.9.2$ chown -R hduser:hadoopgroup /usr/local/hadoop-2.9.2/
root@thesis:~/hadoop-2.9.2$ █
```

Figure 9: Downloading and Extracting Hadoop

- Step 6: Adding export variables for HDFS to run properly.

```
root@thesis:~# su - hduser
hduser@thesis:~$ vim .bashrc
hduser@thesis:~$ tail -n 15 .bashrc
# Hadoop config export
HADOOP_PREFIX=/usr/local/hadoop
export HADOOP_HOME=/usr/local/hadoop
export HADOOP_MAPRED_HOME=${HADOOP_HOME}
export HADOOP_COMMON_HOME=${HADOOP_HOME}
export HADOOP_HDFS_HOME=${HADOOP_HOME}
export YARN_HOME=${HADOOP_HOME}
export HADOOP_CONF_DIR=${HADOOP_HOME}/etc/hadoop
# Native path
export HADOOP_COMMON_LIB_NATIVE_DIR=${HADOOP_PREFIX}/lib/native
export HADOOP_OPTS="-Djava.library.path=${HADOOP_PREFIX}/lib/native"
# Java path
export JAVA_HOME="/usr"
# OS path
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_HOME/bin:$HADOOP_HOME/sbin
hduser@thesis:~$ source .bashrc
hduser@thesis:~$ vim /usr/local/hadoop/etc/hadoop/hadoop-env.sh
hduser@thesis:~$ grep -i "JAVA_HOME" /usr/local/hadoop/etc/hadoop/hadoop-env.sh
# The only required environment variable is JAVA_HOME. All others are
# set JAVA_HOME in this file, so that it is correctly defined on
#export JAVA_HOME=${JAVA_HOME}
export JAVA_HOME="/usr"
hduser@thesis:~$ █
```

Figure 10: Exporting hadoop variables globally

Add the mentioned variables in Figure 10 to hduser's .bashrc file in his home directory for proper functioning of HDFS.

- Step 7: Making changes to Hadoop's configuration files.

All configuration files are stored under `/usr/local/hadoop/etc/hadoop`. We are going to make changes to the following files: **core-site.xml**, **hdfs-site.xml**, **mapred-site.xml** and **yarn-site.xml**

```

hduser@thesis:~$ cd /usr/local/hadoop/etc/hadoop
hduser@thesis:/usr/local/hadoop/etc/hadoop$ tail -n 6 core-site.xml
<configuration>
<property>
  <name>fs.default.name</name>
  <value>hdfs://localhost:9000</value>
</property>
</configuration>
hduser@thesis:/usr/local/hadoop/etc/hadoop$ tail -n 14 hdfs-site.xml
<configuration>
<property>
<name>dfs.replication</name>
<value>1</value>
</property>
<property>
<name>dfs.name.dir</name>
<value>file:/usr/local/hadoop/hadoopdata/hdfs/namenode</value>
</property>
<property>
<name>dfs.data.dir</name>
<value>file:/usr/local/hadoop/hadoopdata/hdfs/datanode</value>
</property>
</configuration>
hduser@thesis:/usr/local/hadoop/etc/hadoop$ tail -n 6 mapred-site.xml
<configuration>
<property>
  <name>mapreduce.framework.name</name>
  <value>yarn</value>
</property>
</configuration>
hduser@thesis:/usr/local/hadoop/etc/hadoop$ tail -n 6 yarn-site.xml
<configuration>
<property>
  <name>yarn.nodemanager.aux-services</name>
  <value>mapreduce_shuffle</value>
</property>
</configuration>
hduser@thesis:/usr/local/hadoop/etc/hadoop$ █

```

Figure 11: Changes in Hadoop's configuration files

- Step 8: Create the HDFS file system

Run the following command to create the HDFS File system.

```
cd /usr/local/hadoop/bin && ./hdfs namenode -format
```

```

hduser@thesis: /usr/local/hadoop/bin
19/11/09 14:06:34 INFO namenode.FSDirectory: GLOBAL serial map: bits=24 maxEntries=16777215
19/11/09 14:06:34 INFO util.GSet: Computing capacity for map INodeMap
19/11/09 14:06:34 INFO util.GSet: VM type = 64-bit
19/11/09 14:06:34 INFO util.GSet: 1.0% max memory 889 MB = 8.9 MB
19/11/09 14:06:34 INFO util.GSet: capacity = 2*28 = 1048576 entries
19/11/09 14:06:34 INFO namenode.FSDirectory: ACLs enabled? false
19/11/09 14:06:34 INFO namenode.FSDirectory: XAttrs enabled? true
19/11/09 14:06:34 INFO namenode.NameNode: Caching file names occurring more than 10 times
19/11/09 14:06:34 INFO snapshot.SnapshotManager: Loaded config captureOpenFiles: false skipCaptureAccessTimeOnlyChange: false
19/11/09 14:06:34 INFO util.GSet: Computing capacity for map cachedBlocks
19/11/09 14:06:34 INFO util.GSet: VM type = 64-bit
19/11/09 14:06:34 INFO util.GSet: 0.25% max memory 889 MB = 2.2 MB
19/11/09 14:06:34 INFO util.GSet: capacity = 2*18 = 262144 entries
19/11/09 14:06:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.window.num_buckets = 10
19/11/09 14:06:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.num.users = 10
19/11/09 14:06:34 INFO metrics.TopMetrics: NNTop conf: dfs.namenode.top.windows.minutes = 1,5,25
19/11/09 14:06:34 INFO namenode.FSNamesystem: Retry cache on namenode is enabled
19/11/09 14:06:34 INFO namenode.FSNamesystem: Retry cache will use 0.03 of total heap and retry cache entry expiry time is 600000 millis
19/11/09 14:06:34 INFO util.GSet: Computing capacity for map NameNodeRetryCache
19/11/09 14:06:34 INFO util.GSet: VM type = 64-bit
19/11/09 14:06:34 INFO util.GSet: 0.629999999329447746% max memory 889 MB = 273.1 KB
19/11/09 14:06:34 INFO util.GSet: capacity = 2*15 = 32768 entries
19/11/09 14:06:34 INFO namenode.FSImage: Allocated new BlockPoolID: BP-1424456513-172.31.30.221-1573368304260
19/11/09 14:06:34 INFO common.Storage: Storage directory /usr/local/hadoop/hadoopdata/hdfs/namenode has been successfully formatted
19/11/09 14:06:34 INFO namenode.FSImageFormatProtobuf: Saving image file /usr/local/hadoop/hadoopdata/hdfs/namenode/current/fsimage.ckpt.000000000000000000 using no compression
19/11/09 14:06:34 INFO namenode.FSImageFormatProtobuf: Image file /usr/local/hadoop/hadoopdata/hdfs/namenode/current/fsimage.ckpt.000000000000000000 of size 325 bytes saved in 0 seconds
19/11/09 14:06:34 INFO namenode.NNStorageRetentionManager: Going to retain 1 images with txid >= 0
19/11/09 14:06:34 INFO namenode.NameNode: SHUTDOWN_MSG:
/*****
SHUTDOWN_MSG: Shutting down NameNode at ip-172-31-30-221.ec2.internal/172.31.30.221
*****/
hduser@thesis: /usr/local/hadoop/bin$

```

Figure 12: HDFS File system formatting

Furthermore, it is important to look for the Success message in the stdout pertaining to the creation of the filesystem as see in Figure 12.

- Step 9: Start the HDFS and the Yarn service.

```

hduser@thesis:~$ start-dfs.sh
Starting namenodes on [localhost]
localhost: starting namenode, logging to /usr/local/hadoop-2.9.2/logs/hadoop-hduser-namenode-thesis.out
localhost: starting datanode, logging to /usr/local/hadoop-2.9.2/logs/hadoop-hduser-datanode-thesis.out
Starting secondary namenodes [0.0.0.0]
The authenticity of host '0.0.0.0 (0.0.0.0)' can't be established.
ECDSA key fingerprint is SHA256:Qs5AD+Bv/vLhDSsFuNPer5cPTZAQNO1YKVBDHLB0hu4.
Are you sure you want to continue connecting (yes/no)? yes
0.0.0.0: Warning: Permanently added '0.0.0.0' (ECDSA) to the list of known hosts.
0.0.0.0: starting secondarynamenode, logging to /usr/local/hadoop-2.9.2/logs/hadoop-hduser-secondarynamenode-thesis.out
hduser@thesis:~$ start-yarn.sh
Starting yarn daemons
starting resourcemanager, logging to /usr/local/hadoop-2.9.2/logs/yarn-hduser-resourcemanager-thesis.out
localhost: starting nodemanager, logging to /usr/local/hadoop-2.9.2/logs/yarn-hduser-nodemanager-thesis.out
hduser@thesis:~$ jps
8305 ResourceManager
8646 NodeManager
8775 Jps
7911 DataNode
7720 NameNode
8143 SecondaryNameNode

```

Figure 13: Starting services

Run the **start-dfs.sh** and the **start-yarn.sh** commands simultaneously as seen in Figure 13. Confirm that the services have started by running the **jps** command. The **jps** binary shows minimalistic information of all the Java processes running on the OS. There should be 6 processes in total as output of the **jps** command for proper functioning of HDFS.

- Step 10: Check proper function of HDFS by adding and removing files

```

hduser@thesis:~$ man ls > abc.txt
hduser@thesis:~$ hadoop fs -ls /
hduser@thesis:~$ hadoop fs -put abc.txt /
hduser@thesis:~$ hadoop fs -ls /
Found 1 items
-rw-r--r--  1 hduser supergroup    7810 2019-11-09 14:53 /abc.txt
hduser@thesis:~$ hadoop fs -rm /abc.txt
Deleted /abc.txt
hduser@thesis:~$ hadoop fs -ls /
hduser@thesis:~$

```

Figure 14: Confirm the working of HDFS

B Installation of Apache Spark

Post installation of HDFS, proceed with installation of HDFS. For this exercise as well, perform the steps with either switching to the root user or using sudo.

- Step 1: Download Spark

```
root@thesis:~# wget http://mirrors.estointernet.in/apache/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
--2019-11-09 15:01:09-- http://mirrors.estointernet.in/apache/spark/spark-2.4.4/spark-2.4.4-bin-hadoop2.7.tgz
Resolving mirrors.estointernet.in (mirrors.estointernet.in)... 103.123.234.254, 2403:8940:2::f
Connecting to mirrors.estointernet.in (mirrors.estointernet.in)|103.123.234.254|:80... connected.
HTTP request sent, awaiting response... 200 OK
Length: 230091034 (219M) [application/octet-stream]
Saving to: 'spark-2.4.4-bin-hadoop2.7.tgz'

spark-2.4.4-bin-hadoop2.7.tgz 100%[=====] 219.43M 1.44MB/s in 2m 50s
2019-11-09 15:04:00 (1.29 MB/s) - 'spark-2.4.4-bin-hadoop2.7.tgz' saved [230091034/230091034]

root@thesis:~# tar -xvf spark-2.4.4-bin-hadoop2.7.tgz > /dev/null
root@thesis:~# ls
snap spark-2.4.4-bin-hadoop2.7 spark-2.4.4-bin-hadoop2.7.tgz
root@thesis:~# mv spark-2.4.4-bin-hadoop2.7/ /usr/local/spark
root@thesis:~# chown -R hduser:hadoopgroup /usr/local/spark/
root@thesis:~#
```

Figure 15: Downloading Apache spark

Post downloading and extracting spark, change the owner and group recursively of the spark directory as shown in Figure 15

- Step 2: Add SPARK_HOME variable to hduser's .bashrc file

```
hduser@thesis:~$ tail -n 4 .bashrc
# Spark path
export SPARK_HOME=/usr/local/spark
# OS path
export PATH=$PATH:$HADOOP_HOME/bin:$JAVA_PATH/bin:$HADOOP_HOME/sbin:$SPARK_HOME/bin:$SPARK_HOME/sbin
hduser@thesis:~$ source .bashrc
hduser@thesis:~$
```

Figure 16: Adding export variables to hduser's .bashrc file

- Step 3: Start Master and slave service of Spark.

```
hduser@thesis:~$ start-master.sh
Starting org.apache.spark.deploy.master.Master, logging to /usr/local/spark/logs/spark-hduser-org.apache.spark.deploy.master.Mast
gr-1-thesis:out
hduser@thesis:~$ curl -s localhost:8080 | grep -i "Spark Master at"
<title>Spark Master at spark://thesis:7077</title>
Spark Master at spark://thesis:7077
hduser@thesis:~$ start-slave.sh spark://thesis:7077
Starting org.apache.spark.deploy.worker.Worker, logging to /usr/local/spark/logs/spark-hduser-org.apache.spark.deploy.worker.Work
sr-1-thesis:out
hduser@thesis:~$
hduser@thesis:~$
```

Figure 17: Start Apache Spark Master and Slave

THE HTTP services runs by default on port 8080. You can browser it either via a thin or a thich client. The URL will be in the form of http://IP_Address:8080 as seen in Figure 17

- Step 4: Confirm if slave is added to the master.

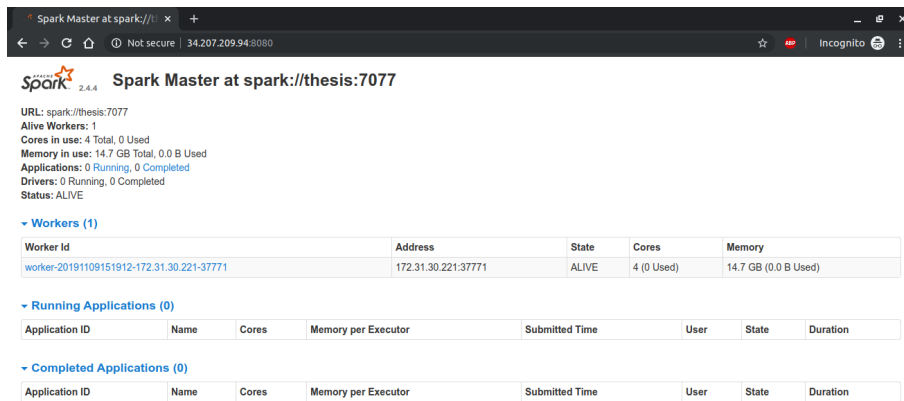


Figure 18: Confirmation of Slave connecting to Master

In Figure 18, we see that under the workers section, one of our workers have connected to the Master server and is online.

- Step 5: Confirm if Spark shell is working

We can simply confirm if spark shell is working by simply running the spark-shell binary which is under the /bin directory of \$SPARK_HOME directory. A simple print command is run which prints a line to console output in Figure 22

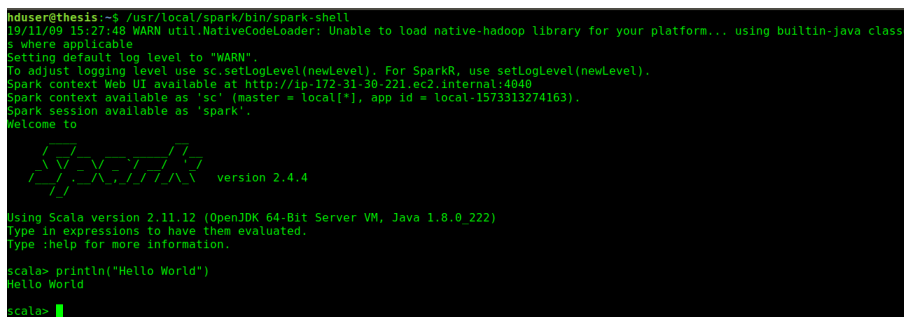


Figure 19: Spark Shell

- Step 6: Install Jupyter notebook (Optional)

To install the Jupyter notebook and to configure it to run with python 2, run the following command: **apt install python-pip; pip install jupyter; python2 -m pip install ipykernel; python2 -m ipykernel install --user**

You can either run the code in Jupyter notebook or command line. To start the jupyter notebook, simply type the following command in your terminal: **jupyter notebook**

```
root@x18139540-thesis: ~
File Edit View Search Terminal Help
root@x18139540-thesis:~# apt install python-pip > /dev/null
WARNING: apt does not have a stable CLI interface. Use with caution in scripts.
root@x18139540-thesis:~# pip install jupyter > /dev/null
root@x18139540-thesis:~# python2 -m pip install ipykernel > /dev/null
root@x18139540-thesis:~# python2 -m ipykernel install --user █
```

Figure 20: Installation of Jupyter notebook

C Running experiments

- Running experiment for Spark

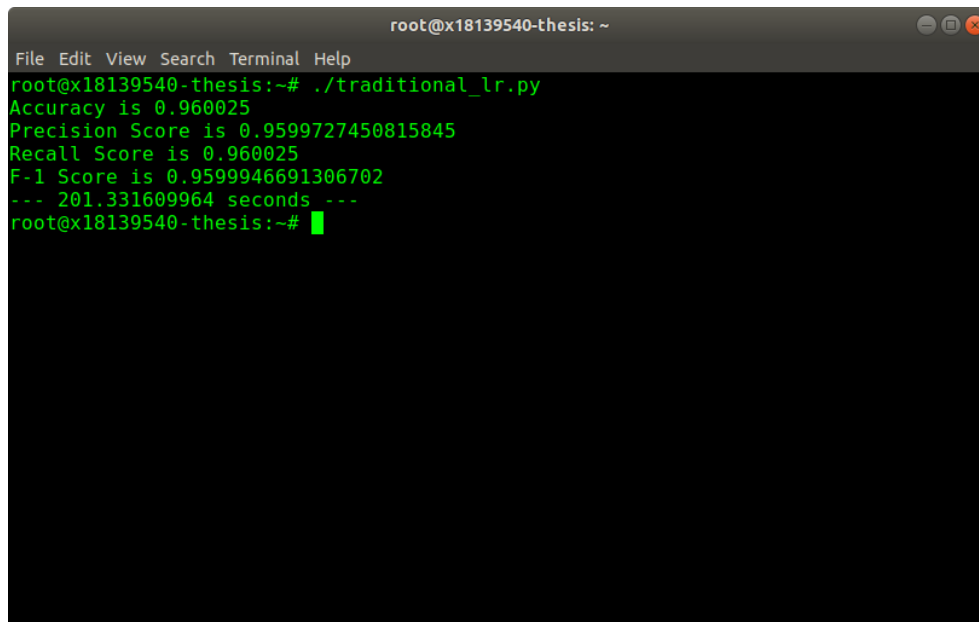
When a job is submitted to spark, it tends to log everything to the console output. Hence it is ideal to redirect the output to a file. This can be done using the following command: `spark-submit spark_lr.py > output`

```
root@x18139540-thesis: ~
File Edit View Search Terminal Help
19/12/02 15:18:14 INFO storage.ShuffleBlockFetcherIterator: Getting 2 non-empty blocks including 2 local blocks and 0 remote blocks
19/12/02 15:18:14 INFO storage.ShuffleBlockFetcherIterator: Getting 0 non-empty blocks including 0 local blocks and 0 remote blocks
19/12/02 15:18:14 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 0 ms
19/12/02 15:18:14 INFO storage.ShuffleBlockFetcherIterator: Started 0 remote fetches in 1 ms
19/12/02 15:18:14 INFO executor.Executor: Finished task 1.0 in stage 132.0 (TID 262). 1091 bytes result sent to driver
19/12/02 15:18:14 INFO scheduler.TaskSetManager: Finished task 1.0 in stage 132.0 (TID 262) in 9 ms on localhost (executor driver) (1/2)
19/12/02 15:18:14 INFO executor.Executor: Finished task 0.0 in stage 132.0 (TID 263). 1410 bytes result sent to driver
19/12/02 15:18:14 INFO scheduler.TaskSetManager: Finished task 0.0 in stage 132.0 (TID 263) in 9 ms on localhost (executor driver) (2/2)
19/12/02 15:18:14 INFO scheduler.TaskSchedulerImpl: Removed TaskSet 132.0, whose tasks have all completed, from pool
19/12/02 15:18:14 INFO scheduler.DAGScheduler: ResultStage 132 (countByValue at MulticlassMetrics.scala:42) finished in 0.015 s
19/12/02 15:18:14 INFO scheduler.DAGScheduler: Job 129 finished: countByValue at MulticlassMetrics.scala:42, took 0.394664 s
19/12/02 15:18:14 INFO server.AbstractConnector: Stopped Spark@45a26493(HTTP/1.1,[http://1.1]:(0.0.0.0:4040))
19/12/02 15:18:14 INFO ui.SparkUI: Stopped Spark web UI at http://x18139540-thesis:4040
19/12/02 15:18:14 INFO spark.MapOutputTrackerMasterEndpoint: MapOutputTrackerMasterEndpoint stopped!
19/12/02 15:18:14 INFO memory.MemoryStore: MemoryStore cleared
19/12/02 15:18:14 INFO storage.BlockManager: BlockManager stopped
19/12/02 15:18:14 INFO storage.BlockManagerMaster: BlockManagerMaster stopped
19/12/02 15:18:14 INFO scheduler.OutputCommitCoordinatorOutputCommitCoordinatorEndpoint: OutputCommitCoordinator stopped!
19/12/02 15:18:14 INFO spark.SparkContext: Successfully stopped SparkContext
19/12/02 15:18:15 INFO util.ShutdownHookManager: Shutdown hook called
19/12/02 15:18:15 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-94aa4532-c6b9-4e6a-8f22-a0df28766256
19/12/02 15:18:15 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-6fd31207-d329-4eb3-bac1-9aca219842b0
19/12/02 15:18:15 INFO util.ShutdownHookManager: Deleting directory /tmp/spark-94aa4532-c6b9-4e6a-8f22-a0df28766256/pyspark-dab8dcaf-2bdc-4a36-9f92-c988bfaeebcd
root@x18139540-thesis:~# cat output
Summary Stats
Precision = 0.949163050217
Recall = 0.958672510958
F1 Score = 0.953894080997
model accuracy 0.962597927723
--- 15.7784090042 seconds ---
root@x18139540-thesis:~# █
```

Figure 21: Submitting job to spark

- Running experiment for Traditional ML code

This code can either be run on the jupyter notebook or directly from the terminal using the command: `python traditional_lr.py` or `./traditional_lr.py`

A terminal window titled "root@x18139540-thesis: ~" with a menu bar containing "File", "Edit", "View", "Search", "Terminal", and "Help". The terminal shows the execution of a Python script:

```
root@x18139540-thesis:~# ./traditional_lr.py
Accuracy is 0.960025
Precision Score is 0.9599727450815845
Recall Score is 0.960025
F-1 Score is 0.9599946691306702
--- 201.331609964 seconds ---
root@x18139540-thesis:~#
```

Figure 22: Running conventional Python code