# **Real-time Anomaly Detection using Neural Networks and Big Data**

Abstract

Anomaly detection has become increasingly popular in recent years, and there are many different types of algorithms that can be used for this purpose. In this study, we explored the applicability of Autoencoders to detect anomalies in telecommunication data. This approach, together with Big Data technologies, can be effective in identifying anomalies in large datasets like large scale cellular network activity, used for this work. Processing large amount of data can be challenging using traditional tools, and technologies like Hadoop and Spark have been of great help to load and pre-process the data.

1. Introduction

Data analytics has evolved dramatically in recent years, developing new techniques and technologies for analyzing large amounts of data. One such approach is the use of neural networks, which shows great promise in the ability to learn from and make predictions based on complex data sets. It has been used in a wide range of applications, from image recognition to natural language processing, with impressive results.

In this paper, we will explore the use of neural networks combined with big data storage and processing technologies to develop new anomaly detection techniques. Anomaly detection is a method of identifying unusual or unexpected patterns in data, which may indicate errors, fraud, or other issues [1]. Some areas in which anomaly detection is popular include:

* Fraud detection (insurance, banking)
* intrusion detection (computer networks, national surveillance)
* medical informatics (diagnosis, disorder detection)
* fault/damage detection (commerce, industry)

Our approach uses big data technology to collect and store large amounts of data related to telecommunication activity, which can then be analyzed by neural networks. These networks are trained to recognize patterns and relationships in the data, enabling them to detect anomalies that might be difficult to detect with traditional methods that. By using this technique, we hope to improve our ability to identify abnormalities as they occur, resulting in faster response times and more effective decision-making.

This paper will provide an overview of our approach, including a discussion of relevant literature and a description of our methodology. We will also present the results of our research, including our findings and implications for further analysis of the field. Finally, we will discuss the limitations of our approach and propose future research directions.

1. Literature Review

In recent years, there has been a growing interest in using neural networks and big data technologies for real-time anomaly detection, as well as their comparison to traditional techniques.

One of the key advantages of using neural networks for anomaly detection is their ability to learn from and make predictions based on complex data sets. For example, Fu Z. (2016) presented a method to monitor and prevent malicious attacks on computer networks using a recurrent neural network model to process sequential data and do risk analysis. The proposed system was tested through massive experiments, demonstrating its reliability and the effectiveness of using neural networks for real-time anomaly detection.

Sometimes anomaly detection models, especially in the case of intrusion detection systems (IDS), can produce false positives. An example on how deep learning can help in this case can be found in X (20XX). The paper proposes using deep learning models, especially long short-term memory (LSTM), instead of traditional machine learning models, such as support vector machines (SVM), to optimize the anomaly detection performance and reduce the false-positive rate. The paper reports that LSTM achieves 10% lower false-positive rate than SVM, and higher accuracy on unseen data. The paper also claims that LSTM can detect more types of attacks, such as contextual and collective attacks, than traditional models.

For X (2022) a small dataset has been used to train the model, but in the discussion the author mentioned that deep learning models perform better with big data, as seen in Fig. Because of this, in addition to the use of neural networks, there has also been significant interest in the use of big data technologies for real-time anomaly detection. These technologies, such as Hadoop and Spark, allow for the efficient storage and processing of large amounts of data, enabling real-time analysis. For example, X (2018) conducted a comprehensive review of existing research on real-time intrusion detection using big data and machine learning techniques, including neural networks. They found that big data technologies can significantly improve the performance of intrusion detection systems by enabling real-time analysis of large amounts of data.

Al Jallad, K., Aljnidi, M. & Desouki (2019) is another good example of how using big data analysis with deep learning in anomaly detection is an excellent combination that may be optimal solution. The system proposed in this paper uses a distributed framework (Apache Spark) to handle large-scale and heterogeneous data sources, such as flow traffic, traffic aggregation, and labels. A recurrent neural network with long short-term memory (LSTM) has then being used to learn the normal pattern of the network traffic and classify each time frame as normal or anomalous. Their model got 10% less false positive than a traditional learning model.

Overall, the existing literature suggests that the combination of neural networks and big data technologies can be highly effective for real-time anomaly detection in cybersecurity applications. The most used neural network is the recurrent type, while in few cases like Lu,Wei, Li & Wang (2018) Convolutional Neural Networks have being used to train a neural network to detect anomalies in system logs. The paper compares the CNN-based model with other approaches using Long Short term memory (LSTM) and Multilayer Perceptron (MLP) on big data logs. The paper claims that the CNN-based model has better accuracy (reaches to 99%) than the other approaches.

Most of the studies show and compare the effectiveness of RNN and CNN. Therefore, in this paper, we introduce Autoencoder-based network anomaly detection method on processed big data. This technique allows the use of both CNN and RNN, and several papers exist on its usage for anomaly detection like Chen, Yeo, Lee & Lau (2018).

1. Methodology

In this study, we used a combination of big data storage and processing technologies to collect and analyze large amounts of data. We then applied a deep learning approach using neural networks to identify patterns and anomalies in the data. This section will cover some theory in regard to the technologies.

* 1. Data

The dataset used was provided by Telecom Italia as part of the Big Data Challenge competition []. I’s available at the Harvard Dataverse [] and consists of 62 text files sized 19.4 GB in total with information about the telecommunication activity over the city of Milan, Italy. Each file contains users Call Detail Records (CDRs) for a single day over a 10-minute interval, with activities being spatially aggregated using a regular grid overlayed on the territory.

There are many types of CDRs and Telecom Italia has recorded the following activities in this dataset:

1. Square ID: The identification number of a given square of Milano GRID.
2. Time stamp: The date and time instant of each activity record.
3. SMS-in activity: The number of received SMSs inside a particular square.
4. SMS-out activity: The number of SMSs sent out inside a particular square.
5. Call-in activity: The number of inbound calls inside a particular square.
6. Call-out activity: The number of outbound calls inside a particular square.
7. Internet activity: The amount of Internet access generated inside a particular square.
8. Country code: The phone country code.
   1. Neural Networks

Neural networks, also known as artificial neural networks (ANNs), are a subclass of machine learning, at the core of deep learning algorithms. Their names and structures are inspired by the human brain, as they mimic the signals biological neurons send to each other.

At their core, neural networks are composed of interconnected processing units known as neurons or artificial neurons. These neurons are organized into layers, typically consisting of an input layer, one or more hidden layers, and an output layer. Each connection between neurons is associated with a weight, which adjusts during the training process to learn patterns in data.

As discussed in the previous chapter, most of the existing literature review for anomaly detection uses RNN and, in some cases, CNN.

* RNNs have connections that loop back on themselves, allowing them to process sequences of data. This makes them suitable for tasks such as time series prediction and natural language processing.
* CNNs are designed for processing grid-like data, such as images and videos. They use convolutional layers to automatically learn features from the data, making them highly effective in image recognition tasks.

Those types of networks are typically used for supervised learning tasks where labeled data is available. In our case, we have unlabeled data with no target variable to predict or classify.

* + 1. Autoencoder

Autoencoder was a natural choice for the type of data we had along with the objective of detecting anomalies within it.

They are neural networks that aim to copy their inputs to their outputs. They work by compressing the input into a latent-space representation, and then reconstructing the output from this representation. This suits our research as we deal with big data.

This kind of network is composed of two parts: an Encoder, which compresses the input, and a Decoder, which tries to reconstruct it. During training, the network learns the best weights and biases to minimize the reconstruction error (the difference between the original input and the output). When an anomaly occurs, this reconstruction error will be significantly higher, which can be used as an indication of an anomaly.

Additionally, Autoencoders do not require labels, making them ideal for unsupervised learning tasks. They learn to extract useful features from the input data in an unsupervised manner, which can be used for detecting anomalies.

* 1. Big Data

In this research we are dealing with ‘Big Data’: this term usually refers to large and complex sets of data that are beyond the capacity of traditional data processing and analysis tools to handle effectively.

Big data is characterized by three Vs:

* Volume: big data involves vast amounts of data that can span from terabytes to petabytes and so on.
* Velocity: data is generated and collected at unprecedent speeds.
* Variety: big data comes in many different formats and types, including structured data (like databases), semi-structured data (like XML files), and unstructured data (like text, images, and videos).

As we are dealing with large amount of text files in this research, open-source distributed computing frameworks, Hadoop and Apache Spark, have been used to store and process the data.

Hadoop is an open-source framework designed for distributed storage and batch processing of large datasets. It comprises the Hadoop Distributed File System (HDFS) for storage and the MapReduce programming model for data processing.

Apache Spark is a versatile, open-source distributed computing framework that supports both batch and real-time data processing. It excels in speed due to in-memory processing and offers high-level APIs in multiple programming languages.

To summarize, Hadoop and Spark have been selected for their performance in dealing with large size of data (in our case, ~19GB of text files). As the goal of this work is to detect anomalies on a full dataset of unlabeled text files, an autoencoder model has been trained. The next section will provide more in-depth details of the practical use of those technologies, with a high-level view of the machine learning pipeline and tools used in each stage shown in Figure 1.

TODO ADD FIGURE

1. Results and Discussion

In this section, the approach discussed in the methodology chapter is explained in detail and evaluated. Please note that a virtual machine with limited hardware power has been used to process the data and train the autoencoder; the implementation available in the accompanying Jupyter notebooks is purely for demonstration purpose as only a sample of the data has been used.

* 1. Data Processing

Hadoop has been locally installed on the virtual machine and the datafiles have been moved to the hdfs under the path /user1/data/.

Once data has been fully uploaded on Hadoop, PySpark has been used as it allows writing Spark applications using Python, making it easier to leverage the power of Spark for processing large-scale data. It provides high-level API for various data processing tasks that allows running tasks in a distributed fashion.

The first step has been removing lines with missing values and creating a schema to easily allow data manipulation. To reduce data size, following features engineering has been performed:

* Data has been aggregated from 10-min intervals to 1-hour intervals.
* Inbound and outbound sms and call activity has been summed into a single feature.
* Extracted additional features from the timestamp to identify if the observation is related to an activity performed during the week or on a weekend, as well as the exact moment of the day.
* Categorical values have been encoded with numerical values.

All this has been done with the help of PySpark SQL [[link](https://spark.apache.org/docs/2.4.0/api/python/pyspark.sql.html)], a component of Pyspark that provides APIs for working with structured and semi-structured data, making it easier to perform data manipulation, querying, and analysis on large datasets in a distributed computing environment.

Data before and after processing is shown in Table 1 and 2.

TODO ADD TABLE

Despite reducing the size of the analyzed data to ~2.6GB, those processing operations were still taking time, for this reason three data format supported by PySpark (text, csv, and Parquet) have been evaluated to export the data in a more performant format. Apache Parquet is a columnar storage format available to any project in the Hadoop ecosystem [[link](https://parquet.apache.org/docs/overview/)], regardless of the choice of data processing framework, data model or programming language. It is built to support very efficient compression and encoding schemes [[link](https://parquet.apache.org/docs/overview/motivation/)] and it has proven to be the optimal choice for the data analyzed in this study. Several tests have been run to compare performances between Parquet, text and csv, results are shown in Table 3.

TODO ADD TABLE

PySpark doesn’t provide APIs to integrate directly with visualization tools like matplotlib, but it’s still possible getting some initial insights from the data in a tabular output form. A visualization has been produced to identify usage patterns across the data converting the data to a pandas dataframe, it eventually works but it’s not the best approach with big datasets as it can be a long running operation.

From a preliminary analysis, some patterns have been identified such as:

* significant part of the telecom traffic is towards countries like Switzerland, France, UK and China (excluding Italy).
* Users tend to use Internet more than performing calls or SMS, especially during lunch and afternoon times.
* Average sms, call and internet activity is higher on weekdays when comparing with weekends.

TODO ADD FIGURE

It is important making sure that there are strong patterns in the data as, while autoencoders are capable of learnings patterns and trends, their performance in anomaly detection depends on the quality and relevance of the patterns present in the training data.

* 1. Data Modeling

TensorFlow has been used to create and train a model for this project. Before doing so, data has been further explored to make sure it can be used to fit the Autoencoder model.

After preprocessing, the dataset contains 7 features and 7516761 observations; despite the reduction of the files uploaded on Hadoop and preprocessing, the dataset is still quite big.

The focus has been on the core features containing the actual traffic: describe () function shows a big difference between min and max values, such as between standard deviation and mean indicating skewed data. This has been confirmed by the skewness utility from PySpark and one visualization using a histplot. Only one feature has been plotted due to the performance constraints of sns with such a big amount of data.

Based on the above findings, data has been normalized but was still too big for the machine used for this project as, while converting into a Numpy array, the Jupyter kernel died. To address this issue two measures have been applied:

1. Float values have been downcasted to double.
2. PySpark sample utility [] has been used to select a sample containing 20% of the processed data.

4.2.1 Model definition and results

To build the autoencoder model, the architecture chosen is a Sequential model with Dense layers as this type of model in TensorFlow provides a simple and intuitive way to define a feedforward neural network architecture [[link](https://www.tensorflow.org/guide/keras/sequential_model)]. Autoencoders can be thought of as feedforward neural networks with an encoder and a decoder, making the Sequential model a natural fit. This applies to both the encoder and decoder, that are then combined into a single Sequential model (the actual autoencoder).

After the model is created, its compile() method needs to be called to specify two crucial parameters in neural networks: loss function and optimizer.

A loss function, also known as a cost function, is a measure of how well the neural network is performing with respect to its given training sample and the expected output. It calculates the difference between the network’s prediction and the actual truth value. The goal of training a neural network is to minimize this loss value.

An optimizer is an algorithm or method used to adjust the attributes of the neural network such as weights and learning rate in order to reduce the losses. Its goal is to find the optimal set of parameters that make the model perform well on the task at hand.

For the model built in this study, the loss function applied is the Mean Squared Error (MSE): Autoencoders aim to minimize the difference between the input data and the reconstructed output. MSE is a common choice for the loss function because it measures the average squared difference between corresponding elements of the input and output. [[link](https://saturncloud.io/blog/optimizing-keras-autoencoder-with-the-right-loss-function-and-optimizer/)] Minimizing this loss encourages the autoencoder to learn to approximate the input data accurately.

Adam has been chosen as optimizer as it computes adaptive learning rates for different parameters [[link](https://saturncloud.io/blog/optimizing-keras-autoencoder-with-the-right-loss-function-and-optimizer/)]. This makes it suitable for problems with large data or many parameters, like in autoencoders. The adaptive learning rate can help the neural network learn faster and converge more quickly towards the optimal set of parameters that minimize the loss function.

The model has been trained with the previously extracted sample that has been used fully, both as input and the targets.

When evaluating the model on the dataset, it was noticed that it performed very well in terms of loss ( 0.0023) and accuracy (0.9981). Figure x below shows the training loss over the epochs. Please note that all models in this study have been intentionally trained over a low number of epochs due to the hardware constraints.

A graph with a line

Description automatically generated

In total, using a threshold of 2, 96 anomalies were found over a dataset of 1505750 records. A graphical representation of the results is shown in Figure x, where the red lines represent the anomalies within the analyzed data.

A graph of a graph

Description automatically generated

Despite this initial model being so performant, GridSearchCV and KerasRegressor estimator have been used to tune hyperparameters and find the optimal set. Even in this case, the test has been limited to few epochs and cv to be able to produce results in the current environment. The resulting best parameters and model architecture are quite similar to what has been previously built, confirming that the initial model was optimal for the processed data.

1. Conclusion and future work

In this paper, we performed anomaly detection using CDR data from a cellular network by leveraging Big Data technologies and neural networks.

Our proposed method first load data into a Hadoop hdfs, and then processes it leveraging Apache Spark built-in libraries. An Autoencoder neural network has then been built and trained with a subset of the initial dataset. It resulted in very good performances and identified anomalies in the analyzed subset of the data.

All this work has been done using limited hardware resources, this opens the door to further analysis using the full dataset and additional tuning. Further Big Data technologies can be explored, such as running the preprocessing directly in Hadoop using the map reduce programming paradigm.

Furter study can be done in the field of anomalous data by training regression models with both full datasets, and anomalous-free samples, compare the accuracy and evaluate the impact of anomalies in machine learning.

References:

[1] [A Guide to Anomaly Detection: AI's Role, Examples, Types & More (datrics.ai)](https://www.datrics.ai/anomaly-detection-definition-best-practices-and-use-cases)