*This study investigates the relationship between Big Data management tools such as Relational Database Management System (RDBMS), Hadoop, and APIs, and how they can be separately interlinked with advanced data analytics, specifically neural networks. The purpose of the study is to compare a RDBMS with Hadoop when processing a 1.31 GB dataset, and then apply a Neural Network. To expand the scope, this study will also include the usage of APIs (Keras library) for implementing Neural Networks. This study was conducted using my personal laptop to load a 1.31 GB dataset into a RDBMS and Spark. I utilized Jupyter Notebooks to interact with these two technologies, exploring computing times, roadblocks faced, and other insights. Following this, we applied the same Neural Network to predict if certain jobs are more popular based on gender. Another aspect of the study involves utilizing an API; for this, we are employing Keras and a Convolutional Neural Network (CNN). Our aim is to evaluate the performance of the CNN model in classifying movie reviews as positive or negative based on their sentiment. The research findings indicate that using RDBMS or Hadoop for data processing is not as quick and straightforward as using an API like Keras, where you simply import the data without the need to worry about how to push it into databases, this becomes clear when modelling data using NN via Jupyter Notebooks.*

Relational Database Management Systems have been well-established since the late 1970s; at that time, the concept of Big Data was not the same as it is today. As technology rapidly advanced, the industry needed to process large amounts of data. To address this need, an open-source framework for writing and running distributed applications, called Hadoop, entered the scene. These two technologies, RDMS and Hadoop, are great; however, the implementation of both requires a high level of technical software skill. This is where APIs offer a solution to this problem, which the industry refers to as Machine Learning as a Service (MLaaS), e.g., Azure ML or AWS ML, just to mention a few

The intention of this paper is to explore all three technologies—RDBMS, Hadoop, and APIs—to determine which one is the best fit for data extraction and processing in the context of Neural Networks implementation. This consideration is crucial, given that many individuals interested in Machine Learning are not software developers, and the need for a 'plug-in' to deploy their ML models is evident.

The chosen topic is Big Data and Neural Networks, with NN being considered a type of Machine Learning (ML) process known as Deep Learning. The field of Big Data is constantly growing and encompasses a need for efficient data management and processing tools. Two well-known tools for handling and analyzing large datasets are Relational Database Management Systems (RDBMS) and Hadoop. However, the rampant advancement of Machine Learning and Neural Networks, the integration of these data management tools with advanced analytics technologies is the focus of this paper.

How do Relational Database Management Systems (RDBMS) and Hadoop compare in terms of efficiency and effectiveness in processing large datasets for the application of neural networks, and how can APIs, particularly the Keras library, streamline the implementation of neural network models in data analytics.

• Examine the current state of RDBMS, Hadoop, and APIs when used in modeling NN.

• Store a 1.31 GB dataset in both an RDBMS (SQL) and Hadoop, and then retrieve the data into a Jupyter Notebook to model a neural network.

• Utilize an API (Keras) to model a neural network and compare its performance in conjunction with RDBMS and Hadoop.

• Discuss the rationale behind the selection of the NN model for both scenarios

The current state of RDBMS has evolved substantially with enhancements in storage, speed, and scalability by using cloud-based solutions. The future holds a shift for RDBMS transitioning to a NoSQL database or having a unified metamodel for both. To understand why NoSQL is taking over RDBMS, it is crucial to talk about: Schemas, where NoSQL uses dynamic instead of static schemas; the type of data to be stored, with NoSQL databases offering advantages for hierarchical data storage due to their flexible data models and scalability, while RDBMS are not that flexible; scalability, with NoSQL depending on horizontal scalability and RDBMS on vertical scalability; and other points where NoSQL surpasses RDBMS, including data warehouse, complexity, cloud, and big data handling, and output performance.

In the era of Big Data, where we sometimes run out of storage and face difficulties on a single host due to the volume of data, Hadoop came into the scene to tackle this by offering computational capabilities over huge amounts of data. The present and future look bright for Hadoop, as some of the major Big Data companies, such as Google, Facebook, eBay, Twitter and Spotify, rely on this technology.

Machine Learning APIs have helped developers integrating data flows into complex algorithms without requiring deep expertise. These APIs were once primarily used for basic tasks like picture and speech recognition, but they have since grown to include a variety of machine learning activities, such as predictive analytics and natural language processing. Today, they are essential to leading tech companies and cutting edge industries, being ML accessible and customizable than in the past. Looking at the future ML APIs will remain as they have simplified model development across diverse environments.

The development of Deep Learning Neural Networks (DLNNs) traces back to the 1950s. The method has improved since the introduction of Convolutional Neural Networks (CNNs) by LeCun et al. in the late 1980s, which showed how good deep learning architectures are for image processing. In 2012 this field achieved another milestone with AlexNet's success in the ImageNet challenge, showcasing DLNNs' potential in image recognition. The introduction of Transformer models by Vaswani et al. in 2017 marked another significant advancement, in natural language processing. Deep learning NN have the future guaranteed as they are extensively used to model data stored in RDBMS, HDSF and ML APIs.

This section aims to explain the demo work with all its components.

Hardware and software configuration of the host laptop and the VM.

• Laptop: HP 250 G8 PC.

o Operating System (OS): Microsoft Windows 10 Pro.

o Processor: 11th Gen Intel® Core™ i7-1165G7 @ 2.80GHz, 2803 Mhz, 4 cores, 8 logical processors.

o RAM: 16GB.

o Hard Disk Drive (HDD): 237GB.

• VirtualBox: Version 7.0.14.

o OS: Ubuntu 22.04 LTS (Jammy Jellyfish) (64-bit).

o Processor: Configured with 2 cores and 2 logical processors from the host's 11th Gen Intel® Core™ i7-1165G7 processor.

o RAM: 4GB.

o Memory (Disk Space): 100GB.

Essential software versions installed on VM:

• Hadoop: 3.3.6.

• Spark: 3.4.2.

• MySQL: 8.0.36.

• MySQL Workbench: 8.0.36.

• Jupyter Notebook: 6.4.8.

The dataset loaded into Hadoop and MySQL was sourced from the Datablist website (www.datablist.com, n.d.). It comprises nine columns and two hundred rows, containing personal data such as names, surnames, gender, job positions, etc. A significant consideration with this type of data is privacy; however, the data was randomly generated by the Python Faker package, as seen on the Datablist GitHub account (GitHub, 2023). This approach ensures GDPR compliance, carefully avoiding any conflict with it. Importantly, this dataset does not require a license for use, as this data is dummy generated for testing purposes.

At first people.csv weighted 0.23 GB however for testing purposes it was increased to 1.6 GB using 1. Increasing\_dataset\_size.ipynb script. The dataset was duplicated seven times, resulting in people\_increased.csv. This choice was made with the intention of approaching Big Data. Although 1.6 GB is not near what Big Data looks like nowadays, it is close to concept in terms of overwhelming most conventional applications. For instance, Excel CSV grid will crash automatically when attempting to open this file. Yes, we can use Notepad ++ to how the data looks but no more than that.

### *Hadoop people\_increased.csv load: First step creating a new directory and moving the dataset into it:*

After a successful load Hadoop UI shows:

Above figure shows a 1.52GB file that is replicated once with a block size of 128MB. To get a sense of how HDFS works we need to run the following command:

After running it, we see the file divided and stored across 13 blocks:

This means the HDFS has filled 12 blocks completely 128MB (134,217,728 bytes) in size, consistent with HDFS's default block size setting except the last one as is the remainder, with 20.24MB (21,234,682 bytes). This is common as the final block not using the default block size unless is a multiple of that block size.

1. MySQL people\_increased.csv load: Before loading the file an schema and a table inside must be created:

After successful schema and table creation, data will be loaded via 2.Importing\_1.6GB\_CSV\_to\_MySQL.ipynb script. This approach was chosen because MySQL Workbench server import failed, displaying the error : ”Error Code: 2013. Lost connection to MySQL server during query”. MySQL workbench appears to struggle with importing large CSV files into a schema. However, the script did with a time of 7 minutes and 28 seconds. Let us examine the size of the table:

3) HDFS vs MySQL loading process: After this implementation step, it is clear that HDFS is quicker in terms loading time. It required just two commands and took only 5 to 10 seconds for the file to be integrated in the system whereas, MySQL took 7 minutes and 28 seconds. Furthermore, MySQL required the creation of a schema and table. A second advantage of HDFS is memory comsumption; from an intial 1.63GB csv file when loaded, it was reduced to 1.52GB, whereas MySQL, upon loading, increased to 1.87GB.

Script 3.HDFS\_Data\_to\_model\_NN.ipynb will guide the entire process outlined in this section. Once the data is loaded into HDFS, it must be extracted to implement the neural network. To facilitate this, a Python package called PySpark is used to read, process, and analyze the data stored in HDFS. The Apache Spark Web UI can be accessed at localhost:4040, where all the application stages can be viewed.

The running time for extracting data from HDFS and printing it into a pandas Data Frame is 2 minutes and 41 seconds; this will be compared with the running time for data extraction from MySQL.

The next step is to implement the NN, which comprises three distinct phases:

1) Initial Model Setup: The model is a fully connected feedforward neural network, often known as Multilayer Perceptron (MLP). The architecture breaks down in:

• The first layer is a dense layer with 12 neurons, indicated by input\_dim=2, which means the model should expect two input features: Job\_Title and Age. The relu activation function is used as a linear function that outputs the input directly if it is positive, and zero if it is negative. This function was chosen due to the simplicity it brings to the model and its suitability for the binary nature of the problem, classifying gender as male or female based on Job\_Title and Age.

• The second layer is a dense layer with 8 neurons and relu activation.

• The output layer, the final layer, is a dense layer with a single neuron that uses the sigmoid activation function. For binary classification tasks the sigmoid function it is very convenient, as it maps any input value to an output between zero and one.

The compilation is carried out using *binary\_crossentropy* loss function which measures the difference between predicted binary outcomes and actual binary labels. The Adam optimizer adjusts the weights and the model tracks the accuracy as a metric for evaluation.

The results after training for 100, the model achieved 50% accuracy and showed 70% loss rate with no variation whatsoever. Clearly, it is not performing well and needs adjustment. The model running time is 7 minutes and 3 seconds.

2) Model Data Input Adjustment: Initially, we had 2 input features. Upon realizing that neural networks perform well with more features, we transposed the Job\_Title column, treating each job category as a separate feature with people Age as values. After this transformation, the number of input features increased to 639. With the new data distribution, over 100 epochs, accuracy increased at 99.86% and loss decreased at 1.34%. However, without further checks, there is a possibility that the model might be overfitted. Model running time takes 8 seconds, by the distributing the data accordingly the model not only performs better it takes less computing time.

3) Overfitting Assesment and Mitigation: To assess overfitting, data validation must be conducted. To determine if overfitting is present, a comparison between training and validation accuracy should reveal a significant difference; the same applies to training and validation loss. Over 100 epochs, the training accuracy is 99.74% and validation accuracy is 49.48%. Training loss is 1.93%, and validation loss is 188.22%. These results clearly indicate an overfitted model.

By adding more data to the model, overfitting was reversed, leading to a well-fitted model where training and validation scores converged. The number of rows increased from 2,907 to 11,628 by concatenating the data frame four times. Over 100 epochs, the training accuracy reached 99.97%, and the validation accuracy was 99.61%. Training loss was 0.083%, and validation loss was 3.9%. The model is learning and generalizing well on unseen data.

In this section, data is pulled from a MySQL database, however, NN configuration remains unchanged. The script used for this demonstration is *4.MySQL\_Data\_to\_model\_NN.ipynb*. When comparing running times, HDFS Spark took 2 minutes and 41 seconds, whereas MySQL took just 6 seconds. At first glance, it appears that MySQL is outperforming Spark, but this is not actually the case. For the Spark demonstration, 14,000,000 rows were loaded into a Spark data frame variable, while in the MySQL scenario, the selection was limited to 2,000,000 rows. MySQL performance significantly deteriorates when pulling larger datasets. When attempting to fetch 14,000,000 rows, the following error was encountered: *“OperationalError 2013 (HY000): Lost connection to MySQL server during query”.* This indicates that there is not enough memory (RAM) available to handle the process.

The main difference between MySQL and HDFS Spark can be established. HDFS Spark did handle the 14,000,000 rows because it is designed for distributed computing, that means it process data parallelly across multiple nodes, in this case only one node was used, however the way HDFS stores data as blocks and the sequential data access patterns Spark uses, makes the combination of Hadoop and Spark a better choice than MySQL for large datasets. However, it is possible to extract the 14,000,000 rows using MySQL by emulating the loading process for *people\_data* table. The *chunksize* function can assist in batching the records and transferring them into a pandas data frame. While MySQL can be an excellent option for smaller datasets, in this scenario, it has been shown that HDFS and Spark surpass the performance of a traditional RDBMS.

Unlike the two sections mentioned above, data has not been processed in this section. This is because Keras comes with built-in data preprocessing capabilities, packaging the data in a form that is almost ready for model deployment. All relevant code can be found in *5.API\_Data\_to\_model\_CNN.ipynb.* Machine learning libraries like *Keras* are quite convenient for deploying models, as they require no specialized skills compared to setting up a Hadoop or MySQL database. Moreover, *Keras* provides subsidiary libraries like *TensorBoard* and *keras.utils,* which offer visual insights to understand model performance and architecture. Another advantage of using *Keras* is its quick loading time, which enables faster experimentation.

The example explored in this section involves training a 1D CNN for the IMDB sentiment analysis task. This example was chosen because it effectively demonstrates how a CNN can be implemented using the IMDB dataset in *Keras.*

1. Model Phase 1: The model tested in this phase suffers no variation from its source.

Breaking down above figure:

• Embedding layer transforms the input data into dense vectors of fixed size, in this case 128.

• Conv1D layer, performs convolution over the sequence, using 32 filters and a kernel size of 7. The intention is to recognize patterns among the word sequences.

• MaxPooling1D layer, reduces the input by taking the maximum value over a window of size 5. It helps reducing dimensionality and abstracting the features.

• A Second Conv1D layer, same function as the first one it helps to capture more complex patterns in the data.

• GlobalMaxPooling1D layer, takes the maximum value over the time dimension for each feature and reduces the output of the convolutions to a fixed size vector.

After training the model the scores indicate that it is overfitted. The validation loss increases with more epochs even when the training loss decreases. This means the model is memorizing the training data and not being able to generalise unseen data.

2) Model Phase 2: As the intial model was poorly performing the following changes have been put in place:

1. Regularization in the Conv1D and dense layers, that will add a penalty for weight magnitude to the loss function. This forces the model to learn smaller weights, leading to a simpler model.

2. Including dropout after the embedding layer and before the final dense layer. Dropout randomly disables neurons during training, forcing the network to learn redundant patterns, this makes a robust model.

3. Smaller batch size of 64, as opposed to 128 in the first model. Smaller batch sizes can lead to a regularization effect and that might help the model to generalize better.

CNN architecture has changed, its components can be explained as it follows:

• Embedding layer same as the first model but with a smaller output dimension 64.

• Dropout layer randomly sets a fraction of input units to 0 at each update during training to prevent overfitting.

• Conv1D layer with L2 regularization applies convolution but penalizes large weights due to the regularization.

• MaxPooling1D layer, same as in the first model, reduces dimensionality and abstracts features.

• GlobalMaxPooling1D layer, also like the first model, condenses the feature information.

• Dense layer with L2 regularization and dropout, introduces another point where regularization and dropout are applied.

• Output dense layer with sigmoid activation, outputs a probability for the binary classification task.

Overall, this CNN model is better structured than the one provided by the book (Chollet, 2018, pp. 253-259). Overfitting is mitigated through the use of regularization and dropout. As a result, the model memorizes less of the training data and is more capable of generalizing from the patterns learned during training, leading to improved validation performance. In terms of computing times, the first model took 11 minutes and 22 seconds to train, while the second model required only 4 minutes and 18 seconds

Adjusting the model, the parameter that made a significant difference was the dropout rate. Rates close to one did not address overfitting, while rates closer to 0.5 began to properly fit the model. The ideal dropout rate was found to be 0.4, determined through a trial-and-error approach.