UNIVERSITY OF PATRAS
DEPARTMENT OF ELECTRICAL AND COMPUTER ENGINEERING

DEPARTMENT of electronics and computers (PC)
INTERACTIVE TECHNOLOGIES LABORATORY

Study of a real time voice phishing detection system with machine learning technology

DIPLOMA THESIS

IOANNIS SKARPETIS

SUPERVISOR: FEIDAS CHRISTOS

PATRAS – FEBROUARY 2024

University of Patras, Department of Electrical and Computer Engineering.

Ioannis Skarpetis

CERTIFICATION

It is certified that the Diploma Thesis titled

Study of a real time voice phishing detection system with machine learning technology

of the Department of Electrical and Computer Engineering student

IOANNIS SKARPETIS

Registration Number: 1066539

was presented publicly at the Department of Electrical and Computer Engineering at

……../……../……

and was examined by the following examining committee:

Name Surname, Title, Affiliation (supervisor)

Name Surname, Title, Affiliation (committee member)

Name Surname, Title, Affiliation (committee member)

The Supervisor	The Director of the Division
Feidas Christos Substitute Professor	Name Surname Title

ΠΑΝΕΠΙΣΤΗΜΙΟ ΠΑΤΡΩΝ

ΤΜΗΜΑ ΗΛΕΚΤΡΟΛΟΓΩΝ ΜΗΧΑΝΙΚΩΝ ΚΑΙ ΤΕΧΝΟΛΟΓΙΑΣ ΥΠΟΛΟΓΙΣΤΩΝ

ΤΟΜΕΑΣ ΕΠΙΒΛΕΠΟΝΤΟΣ
ΕΡΓΑΣΤΗΡΙΟ ΕΠΙΒΛΕΠΟΝΤΟΣ

ΤΙΤΛΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

Δ Ι Π Λ Ω Μ Α Τ Ι Κ Η Ε Ρ Γ Α Σ Ι Α

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ

ΕΠΙΒΛΕΠΩΝ: ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΕΠΙΒΛΕΠΟΝΤΟΣ

ΠΑΤΡΑ - ΜΗΝΑΣ ΕΤΟΣ

Πανεπιστήμιο Πατρών, Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών.

Ονομαπεπώνυμο φοιτητή/τριας

ΠΙΣΤΟΠΟΙΗΣΗ

Πιστοποιείται ότι η Διπλωματική Εργασία με τίτλο

ΤΙΤΛΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

του/της φοιτητή/τριας του Τμήματος Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών

ΟΝΟΜΑ ΕΠΩΝΥΜΟ ΤΟΥ **ΠΑΤΡΩΝΥΜΟ**

Αριθμός Μητρώου: XXXXXXX

Παρουσιάστηκε δημόσια στο Τμήμα Ηλεκτρολόγων Μηχανικών και Τεχνολογίας Υπολογιστών στις

……../……../……

και εξετάστηκε από την ακόλουθη εξεταστική επιτροπή:

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (επιβλέπων)

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (μέλος επιτροπής)

Όνομα Επώνυμο, Βαθμίδα, Τμήμα (μέλος επιτροπής)

Ο/Η Επιβλέπων/ουσα	Ο/Η Διευθυντής/τρια του Τομέα
Ονοματεπώνυμο Βαθμίδα	Ονοματεπώνυμο Βαθμίδα

PREFACE

The Preface is optional. Here the author puts any information that is not directly related to the scientific content of the Diploma Thesis, such as acknowledgements, etc.

The structure, form and extent of the preface are at the student's discretion, unless otherwise specified by the supervisor.

ABSTRACT

DIPLOMA THESIS TITLE

STUDENT NAME, SURNAME: **SUPERVISOR NAME, SURNAME:**

The objectives, methods, procedures, experiments and results of the Diploma Thesis are briefly described.

The structure, format and scope of the abstract are at the student's discretion, unless otherwise specified by the supervisor.

ΕΚΤΕΤΑΜΕΝΗ ΕΛΛΗΝΙΚΗ ΠΕΡΙΛΗΨΗ

ΤΙΤΛΟΣ ΔΙΠΛΩΜΑΤΙΚΗΣ ΕΡΓΑΣΙΑΣ

ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ: **ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΕΠΙΒΛΕΠΟΝΤΟΣ:**

Σύμφωνα με τον κανονισμό Διπλωματικών Εργασιών, η συγγραφή της Διπλωματικής Εργασίας στην Αγγλική γλώσσα θα συνοδεύεται απαραιτήτως από εκτενή περίληψη στα Ελληνικά, τύπου επιστημονικής εργασίας (paper).

Οι υπόλοιπες λεπτομέρειες σχετικά με τη δομή και τη μορφή της Εκτεταμένης Ελληνικής Περίληψης είναι στη διακριτική ευχέρεια του φοιτητή, εκτός αν προδιαγράψει διαφορετικά ο επιβλέπων.

Table Of Contents

Table Of Figures

1. Introduction

Since the dawn of the 21st century, humanity has embarked on a remarkable journey of technological progression. As our society becomes ever more reliant on technology for its daily functions, we've seen a corresponding escalation in the complexity of cybercriminal activities. The advancement of technology has not only streamlined our daily tasks but has simultaneously opened numerous vulnerabilities. These gaps are frequently exploited by cybercriminals, targeting those who struggle to stay abreast of the latest technological developments.

Voice phishing, or 'vishing', is a notable vulnerability within the cybersecurity landscape, manifesting as a sophisticated form of phishing conducted via phone calls. While phishing as a broader concept is not novel, having roots that extend well before the internet age, vishing represents its adaptation to the telecommunication realm. Phishing, at its core, is about manipulating individuals to reveal confidential information under misleading pretenses, a strategy that has seen various incarnations over many decades. The term "phishing" gained widespread recognition during the 1990s alongside the internet's growth. Originally, it denoted email-based frauds wherein culprits masqueraded as legitimate entities to coax out sensitive details like login credentials and financial data from unsuspecting victims.

Vishing exploits the same fundamental principle of deception but leverages the immediacy and personal touch of voice communication. Here, fraudsters make use of phone calls to impersonate legitimate organizations or authorities, aiming to instill a sense of urgency or fear in victims. This tactic often involves compelling narratives designed to prompt quick action, leading individuals to inadvertently disclose personal or financial information.

Efforts to bolster security against such attacks have been ongoing, yet the advent of caller ID spoofing technology has notably heightened the efficacy of vishing attacks. This technology enables attackers to disguise their actual identity, presenting a significant obstacle in identifying and preventing these fraudulent calls. Consequently, the dynamic nature and sophistication of vishing have established it as a significant and continually evolving threat within the cybersecurity domain. This scenario highlights the critical necessity for advanced and adaptable security measures capable of effectively neutralizing the ever-changing tactics of cybercriminals. Consequently, the focus has shifted towards innovative methods for detecting vishing attacks, particularly by analyzing the conversation content between the two parties in real-time. This approach represents a significant step forward in proactive cybersecurity, aiming to identify and mitigate vishing threats as they occur.

2. Analysis of the problem under research

Before delving into the analysis of the research problem addressed in this dissertation, it is essential to first define the central focus, which is Phone Call Scam Detection. This foundational concept forms the basis of our investigation and discussion, setting the stage for a thorough exploration of the strategies and technologies employed in identifying and mitigating fraudulent phone communications.

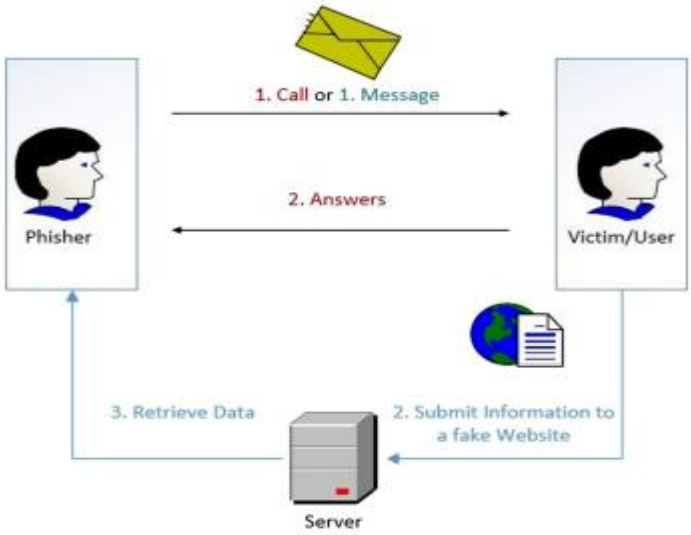


Fig 1: Mobile Phone Phishing Attack [..]

2.1 Vishing attacks and methods

To effectively develop security measures against vishing attacks, it is crucial to first comprehend the vast array of techniques employed by Vishers. Understanding their strategies and tactics provides the foundational knowledge necessary to design and implement robust countermeasures. The techniques implemented by Vishers can be broadly categorized into two main phases [4]:

Before the vishing attack:

- **Automated information gathering:** Vishers frequently use automated tools to 'scrape' large amounts of user data. This bulk collection can provide a broad base from which specific targets are identified and proceed to be evaluated as potential victims.

- **Victim Evaluation:** Vishing scams often begin with phishing attacks via automated emails, texts, or social media messages, posing as legitimate entities to gather contact information. Those who respond to these initial contacts are more likely to fall for subsequent vishing calls. Some Vishers skip initial contact and directly call numerous potential victims using automated services, aiming to reach susceptible individuals.

- **Data accumulation via research**: A key element of preparation for Vishers involves extensive research and collection of personal and private information about their potential victims. This phase often involves the use of social media platforms to gather

details such as phone numbers, employment data, and location information, which, while not overly sensitive, can lay the groundwork for a more targeted approach.

- **Direct Methods:** In some instances, Vishers may resort to direct, personal methods of information gathering, like sifting through an individual's trash also known as Dumpster Diving. This approach is more physical and hands-on where the perpetrators physically search through garbage to find discarded documents, letters, bills, and other materials that contain personal or sensitive information. Even seemingly harmless details such as phone lists, calendars, or organizational charts can be instrumental for attackers when planning strategy. [5]

- **Caller ID Spoofing:** An essential element in the success of vishing attacks is the ability to avoid detection by the victim. To achieve this, Vishers use a tactic known as "Caller ID Spoofing." This technique involves altering the phone number that appears on the recipient's caller ID. It proves highly effective because many individuals depend on the caller ID to verify the legitimacy of the caller. [6]

During the vishing attack:

- **Impersonation and Authority:** Vishers often pretend to be from reputable organizations, such as banks, government agencies, tech support, law enforcement agencies or members of a financial institution. They use authoritative tones and language to gain the victim's trust and compliance. [8]

- **Confirmation Bias:** They use known information about the victim (gathered during the preparation phase) to build trust and credibility. For example, referencing a recent transaction or a known issue to make the call seem legitimate.

- **Creating urgency:** Vishers frequently employ the tactic of instilling a sense of immediate urgency. They may assert that urgent action is required to address issues like a supposed security breach in the victim's bank account or a pressing financial matter. The aim of the attacker is to prompt the victim to act quickly and impulsively, bypassing the usual steps of call verification and thoughtful consideration.

- **Emotional Manipulation:** Vishers adeptly exploit the emotions of their victims, wielding tactics that evoke fear, sympathy, or excitement to steer their responses. They might threaten legal action or warn of imminent financial loss to instill fear and a sense of urgency. Conversely, they can appeal to the victim's emotions through sympathetic stories or scenarios, creating a false sense of connection or trust. By manipulating emotions in these ways, Vishers aim to cloud the victim's judgment, making it easier to extract sensitive information or coerce them into specific actions that further the scam. [7]

- **Information Overload:** When individuals are quickly presented with an excessive amount of information, it can lead to sensory overload, impairing their ability to logically process and evaluate the arguments being made. In such scenarios, people are more inclined to accept the statements presented to them, as their capacity for critical thinking is overwhelmed by the sheer volume of information. This tactic increases the likelihood of the person acquiescing to the Visher's demands or assertions. [8]

- **Diversion Tactics**: Victims are often redirected to alternate communication channels or actions. This could involve being instructed to call a different number, visit a fraudulent website, or mail sensitive documents to a specific address. These tactics serve to deepen the victim's involvement in the scam, making the deception appear more legitimate and complicating the victim's ability to discern the fraud.

- **Follow-Up and Persistence:** In vishing scams, Vishers often employ persistent follow-up calls, especially if the victim shows initial doubt or reluctance. This relentless approach aims to wear down the victim's defenses, increasing the likelihood of succumbing to the scam. Such persistence is a key tactic in gradually eroding skepticism and reinforcing the scam's credibility.

2.1.1 Typical Vishing Attack Examples

Vishers select from a diverse range of scenarios to deceive their victims. The subsequent examples will elaborate on the most frequently utilized themes in vishing scams. [2]

- **IRS and Immigration Scams:** Scammers impersonate IRS officials or immigration officers, threatening arrest or deportation if alleged debts aren't paid.

- **Romance Scams:** Fraudsters establish romantic connections, often through dating apps or phone calls, claiming to be past lovers in need of emergency funds.

- **Tech Support Scams**: Scammers pose as tech support, alleging urgent computer issues like viruses, gaining remote control of computers to access sensitive information or install malware.

- **Debt Relief and Credit Repair Scams:** Fraudsters offer debt relief or credit repair services for a fee, often worsening the victim's financial situation.

- **Business and Investment Scams:** Scammers pretend to be financial experts, persuading victims to invest money in fraudulent schemes.

- **Charity Scams:** Perpetrators pose as charity workers soliciting donations for fake causes, with funds going directly to the scammers.

- **Auto Warranty Scams:** Scammers make fake calls about car warranties, seeking personal information or payments for nonexistent warranty renewals.

- **Parcel Scams:** Aimed at immigrants, scammers claim a parcel is linked to a financial crime, posing as couriers and police to coerce money for a fake investigation.

- **Kidnapping Scams:** Scammers claim to have kidnapped a relative, using research or social engineering to extract ransom payments.

2.2 Current Approaches and Emerging Frontiers in Vishing Detection

It's evident that vishers possess an extensive arsenal of options, tactics, and scenarios, making the challenge of effectively detecting vishing attacks a complex one. Historically, methods like real-time detection of caller ID spoofing, with software like STIR/SHAKEN [2] and iVisher [6] and blacklisting phone numbers, represented a critical line of defense against such attacks.

However, the integration of machine learning into cybersecurity presents an evolving frontier. These new tools offer enhanced capabilities to security professionals, equipping them to more effectively counteract the sophisticated strategies employed by vishing scammers. This evolution in anti-vishing technology reflects the dynamic nature of the cybersecurity field, where constant innovation is essential to stay ahead of threats.

Efforts to integrate machine learning into vishing detection have primarily focused on analyzing metadata, both statistical and non-statistical, such as CALLING_NUMBER, CALLED_LOCATION, and CALL_DURATION [10]. However, the area of real-time detection that involves processing the actual speech content within a scam call, post-answer by the victim, remains a largely unexplored frontier in the field. This gap suggests potential for future advancements in vishing detection methodologies that could more directly address the nuances of in-call scam tactics. In this dissertation, we shift our focus to developing a system capable of real-time vishing detection. This system will leverage the actual conversational content of calls, utilizing machine learning algorithms. This approach represents a new direction in vishing detection, aiming to directly analyze the dynamics of scam calls as they occur.

2.3 Why Real-Time In-Call Detection Is Essential

Previous efforts to mitigate vishing attacks have predominantly focused on the pre-answer phase of the phone call. Pioneering research in this domain, such as the work of Xing [10], Song [6], and Manh-Hung Tran [12], has been directed towards proactive strategies against vishing. The concept of real-time detection during a call is relatively novel in this field. Lee and Park's [11] research was among the first to explore real-time vishing detection. Utilizing machine learning techniques, their study specifically addressed vishing attacks in the Korean language and yielded some promising results.

Why though, is real-time in-call detection essential? Real-time in-call vishing detection is mandatory for several reasons, which can be effectively communicated through bullet points:

- **Limited Individual Defense**: Once the phone call is answered, individuals primarily rely on their personal knowledge, emotional stability, and understanding to defend themselves against vishing attacks. This reliance is often insufficient as scammers are typically well-prepared, extensively researched, and skilled in manipulative tactics.

- **Emotional Manipulation**: As mentioned before, Vishers exploit emotional vulnerabilities, using tactics like fear, urgency, and sympathy, which can overwhelm the victim's rational decision-making. Real-time detection can provide a safeguard during these high-pressure situations, where individuals might not be able to think clearly.

- **Information Discrepancy**: Scammers often have more information about the victim than the victim has about the caller, creating an imbalance that can be exploited. Real-time detection systems can help level this playing field by providing additional context or warnings about the call.

- **Sophistication of Scams**: Vishing scams have evolved to be highly sophisticated, often bypassing traditional pre-answer detection methods. Real-time detection can adapt to these evolving tactics, offering a dynamic line of defense as the call progresses.

- **Support for vulnerable populations**: Certain groups, like the elderly or less tech-savvy individuals, may be more susceptible to vishing. Real-time detection can provide an additional layer of protection for these vulnerable populations.

Hence, it becomes clear that there is a pressing need for an enhanced defensive system. Such a system would bolster individual protection during the post-answer phase of phone calls, particularly in situations where proactive vishing detection systems fall short.

2.4 Addressing the Identified Problem

The objective of this dissertation is to explore the feasibility and effectiveness of a system that processes phone call conversations in real time for vishing detection. This proposed system will integrate technologies like speech-to-text conversion, machine learning algorithms, and streaming techniques to accomplish its goals. The research will focus on key questions:

- Is it possible to train a machine learning algorithm to detect vishing within a conversation as it happens?

- Can such an algorithm provide real-time predictions with a sufficiently low response time to effectively protect the victim?

This dissertation is structured to methodically address the research questions posed. Chapter 3 delves into the PySpark framework, which is pivotal for data preprocessing, model training, and enabling the streaming functionality of the system. Chapter 4 discusses the implementation of the Speech to Text module, with a particular focus on the integration of the Google Speech to Text API.

Chapters 5 and 6 are dedicated to dataset creation and preprocessing, detailing the methodologies and approaches used in preparing the data for analysis. Chapter 7 outlines the architecture of the selected machine learning algorithms, covering aspects of their training, hyperparameter tuning and useful knowledge on the machine learning sector.

Chapter 8 provides an overview of the system’s functionality as a cohesive unit. The testing procedures and their corresponding results are the focus of Chapter 9, offering insights into the system's performance and efficacy.

The dissertation concludes with Chapters 10, which reflect on the challenges encountered during system development and propose potential future enhancements. This final chapter also presents overall deductions and conclusions drawn from the research.

3. PySpark

PySpark, serving as the Python API for Apache Spark, allows us to utilize the capabilities of Spark in a Pythonic way. In this section, we will examine the components of Spark that contributed to the proposed solution. As the underlying framework upon which PySpark is built, the advanced data processing capabilities of Spark are pivotal to our research. Its potent architecture facilitates rapid and scalable handling of extensive datasets. This provides a

critical backdrop for appreciating how PySpark leverages these features, tailoring them to meet the specific requirements of our application.

3.1 Apache Spark Fundamentals

Apache Spark stands as a formidable open-source engine for analytics, renowned for its proficiency in handling extensive data processing tasks. This unified analytics engine is adept at managing both batch and real-time data processing, positioning it as an invaluable resource in data science and machine learning fields. Its versatility in processing massive datasets efficiently makes it a cornerstone technology for modern data-driven applications.

3.1.1 Spark Architecture

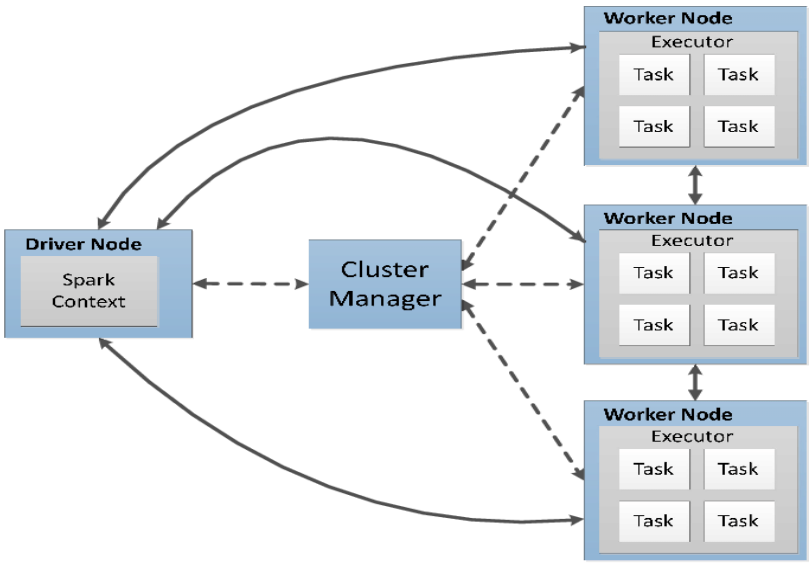


Fig 2: Spark Cluster Architecture [16]

Spark operates as a distributed computing system; in this section we explore its architecture. The architecture is depicted in Figure 3.1 as a **Spark Cluster**, showcasing its ability to coordinate complex processing tasks across multiple machines for efficient data management and computation. At the center of the architecture lies the **Driver Node**, in which the Spark Context is housed. This central coordinator initiates and manages the lifecycle of various Spark jobs. It converts the user application into tasks that are then distributed across the cluster for execution. The **Cluster Manager** is the resource negotiator responsible for allocating resources among the various applications running on the cluster. It ensures optimal utilization of resources and manages the distribution of tasks to the worker nodes. The **Worker Nodes** represent the distributed computational units of the cluster. Each worker node has Executors, which are processes responsible for executing tasks assigned to them by the driver node. Executors run the tasks in multiple threads, which allows Spark to execute tasks in parallel, significantly improving performance. **Tasks** are the smallest units of work in Spark, and they carry out computation and data processing. They are executed by the executors to process the data. The distribution of tasks across executors allows Spark to handle large datasets efficiently [13].

3.1.2 Spark Components

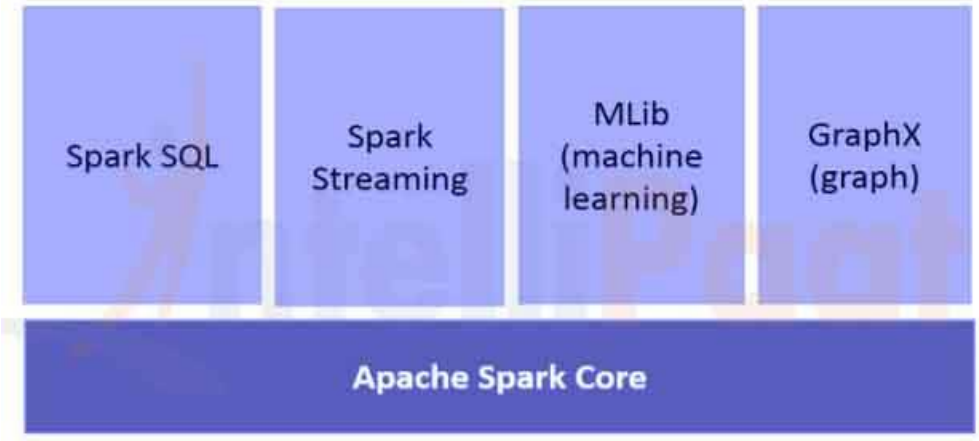


Fig 3: Spark Main Components [17]

The Apache Spark system is composed of several core components, which include the Spark Core and a range of high-level libraries. These libraries cater to specific needs, such as MLlib for machine learning applications, GraphX for graph analytics, Spark Streaming for real-time data processing, and Spark SQL for handling structured data.

3.1.3 Spark Core

Spark Core is the foundational part of Spark, upon which all other functionalities are built. Spark Core provides a variety of core features for distributed task dispatching, scheduling, and basic I/O functionalities, which form the heart of the Spark engine.

To further enhance our comprehension of Spark Core's functionality, and consequently the overall operation of Spark, our examination will pivot to the Resilient Distributed Dataset (RDD).

3.1.3.1 Resilient Distributed Dataset

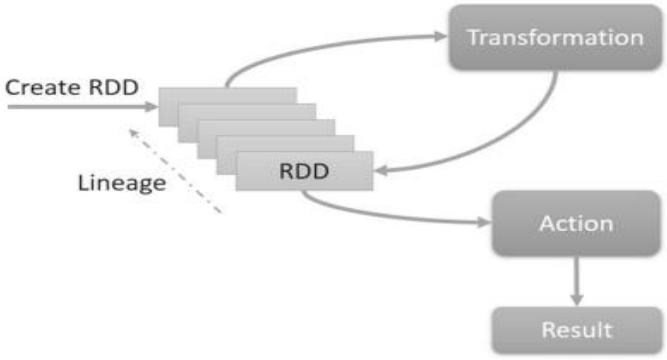


Fig 4: RDD lineage graph [13]

The Resilient Distributed Dataset (RDD) is the foundational element of Spark, providing the framework for fault-tolerant data distribution and parallel processing that underpins the entire system. RDDs can be generated from external data sources or transformed from

existing RDDs. As a key characteristic, RDDs maintain a lineage graph that allows for the efficient recompilation of data in case of node failures, as shown in Figure 4, rather than relying on data replication. This facilitates robust in-memory data processing and supports a variety of computational models. Moreover, RDDs enable complex operations through transformations, which are operations that create a new RDD from an existing one (e.g. map, filter), and actions, which are operations that trigger computation and produce non-RDD values (e.g. reduce, collect). Both transformations and actions are executed across the cluster only when necessary, optimizing resource utilization and performance [13].

3.2 Spark SQL

Having established the foundational role of Resilient Distributed Datasets (RDDs) in Spark's architecture, we now transition to another cornerstone of Spark's data handling capabilities—Spark SQL. This component is integral to our system, particularly in the context of data preprocessing, a topic we will delve into in detail in Chapter 6. Spark SQL extends Spark's functionality to allow for structured data processing and SQL queries, **building upon the groundwork laid by RDDs.** This integration not only simplifies working with structured data but also optimizes query execution through advanced optimization techniques, bridging the gap between traditional database management systems and modern big data processing.

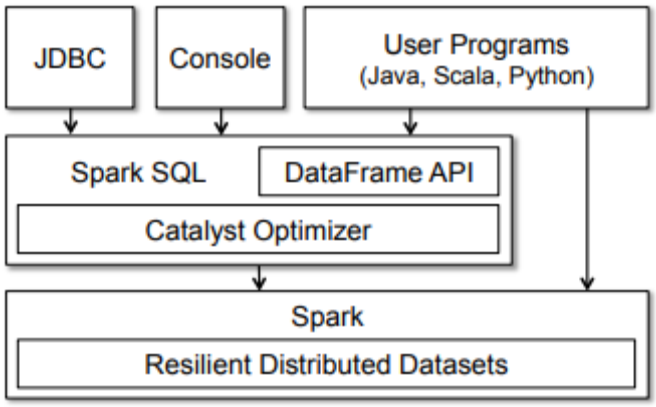


Fig 5: Spark SQL Interaction with Spark and RDDs [15]

The foundational concept in Spark SQL is the DataFrame, which is essentially a distributed set of rows that all conform to the same structure. Analogous to a table in a relational database, a DataFrame maintains awareness of its structure (schema), enabling it to support a variety of relational operations. These operations allow for more efficient query execution due to the inherent optimization in DataFrames, setting them apart from the more general RDDs. DataFrames in Spark SQL are adept at handling a variety of relational database operations, which include:

- **Projection**: Selecting specific columns from the DataFrame.
- **Filtering**: Applying conditions to the data, akin to the SQL 'WHERE' clause.
- **Join**: Combining two DataFrames based on a common key.
- **Grouping**: Aggregating data by specific categories, like the SQL 'GROUP BY' operation.

3.3 Spark's Machine Learning Package

In this section of the thesis, we focus on the Machine Learning package of Apache Spark, recognized for its scalable machine learning applications. Referencing the work of Assefi, Behravesh, and Liu (2017) [14], who tested Spark's machine learning capabilities on extensive datasets, it's evident that Apache Spark's machine learning package stands as a leading platform in machine learning for big data analysis. The Machine Learning package of Spark has been crucial in the development of our proposed system, augmenting Spark's foundational features with advanced machine learning functionalities. This package is an essential part of the Spark ecosystem, facilitating the effective implementation of machine learning algorithms on distributed data structures. The selection of Apache Spark's ML package for this project is primarily due to its proficiency in managing and processing large datasets, a requirement anticipated to be crucial for the solution of the problem addressed by our proposed system.

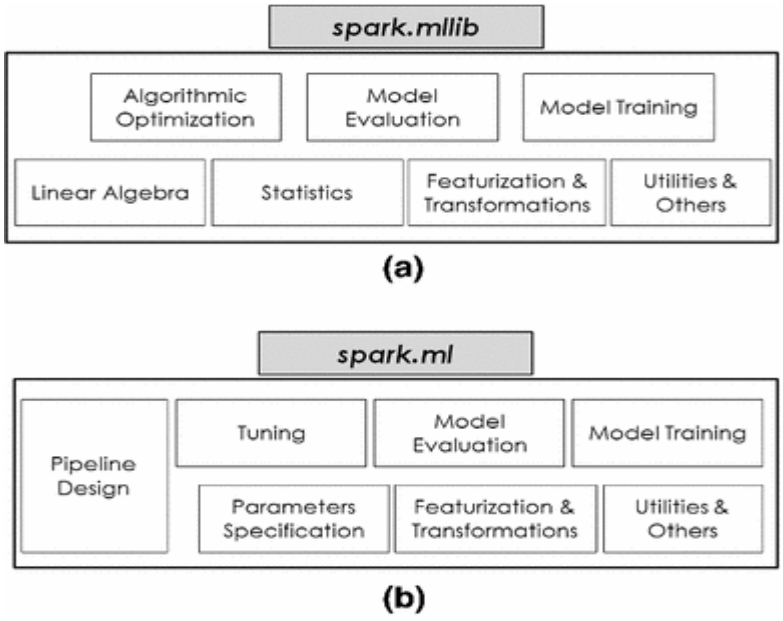


Fig 6: Key Features of Spark's Machine Learning package [18]

The Spark ML package consists of two libraries, MLlib and ML, which are going to be discussed in detail in the following sections.

3.3.1 Spark MLlib

Spark MLlib, the original machine learning library for Spark, is based on the Resilient Distributed Dataset (RDD) abstraction. Therefore, MLlib is suited for RDD-based applications and provides a wide range of machine learning algorithms that are scalable and optimized for big data processing. MLlib's capabilities, that differentiate it from Spark ML, include the following:

- **Linear Algebra**: Spark's MLlib offers both distributed and local abstractions for matrices and vectors, accommodating dense and sparse data representations. Sparse data structures are particularly valuable in big data analytics, as they are frequently encountered due to various factors such as high dimensionality, specific feature transformations, and the prevalence of missing values in large datasets [13].

- **Statistics**: This package features functions for summary statistics, correlation analysis, hypothesis testing, and sampling. Additionally, it provides kernel density estimation, streaming data testing, and random data generation from various distributions [13].

- **Algorithmic Optimization**: Spark MLlib supports two core optimization methods [20]:

    - **Gradient Descent Methods**: These methods involve iterative procedures to minimize the objective function, which in the context of machine learning is often the loss function representing the error between predictions and actual values. Key gradient descent methods included are:

        - **Gradient Descent**: This optimization method seeks a local minimum of the function. In each iteration, it updates the parameters in the direction of the negative gradient of the function at the current point, effectively moving towards the steepest descent to reduce the loss.

        - **Stochastic SubGradient descent (SGD)**: SGD is a variation of gradient descent where the gradient is estimated using a randomly selected subset of data rather than the full dataset. This estimation makes the method particularly suitable for large-scale data processing, as it significantly reduces computational cost while still converging towards the minimum.

    - **Limited-Memory BFGS (L-BFGS**): An optimization algorithm in the family of quasi-Newton methods. L-BFGS improves upon the Broyden–Fletcher–Goldfarb–Shanno (BFGS) algorithm [21] by using a limited amount of computer memory. It's designed to optimize the performance of machine learning algorithms, especially when dealing with large datasets or in cases where the number of parameters is too high for the full BFGS method to handle efficiently.

3.3.2 Spark ML

In the evolving landscape of Apache Spark's machine learning offerings, Spark ML represents the next generation library, built upon the robust DataFrame API. Diverging from MLlib's reliance on the RDD framework, Spark ML caters to DataFrame-centric applications, offering a sophisticated, high-level API tailored for the construction of machine learning pipelines.

The inception of Spark ML marks a significant advancement over its predecessor, MLlib, by incorporating the Machine Learning Pipelines API, which will be discussed in the following sub-section. Additionally, Spark ML enhances model development through built-in hyperparameter tuning capabilities, facilitating the optimization of model parameters for improved performance.

Finally, Spark ML is integrated with the broader Spark ecosystem, most notably Spark SQL. This integration enables fluid data handling, allowing for efficient transitions from data preprocessing, performed with SQL-like commands, to subsequent phases of machine learning model development.

3.3.2.1 Machine Learning Pipelines API

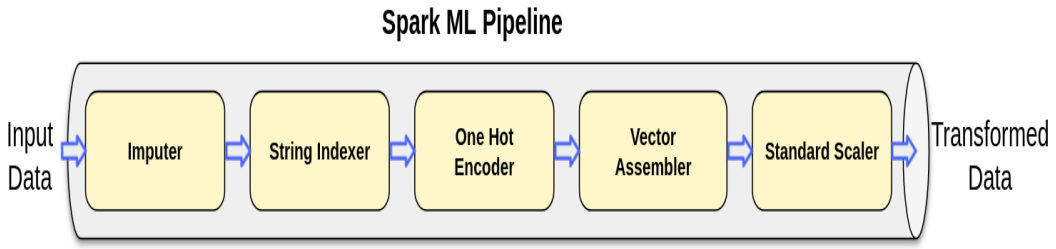


Fig 7: Spark Machine Learning Pipeline Example [23]

The Pipelines API introduced by Spark ML revolves around several core concepts:

- **DataFrame**: Within this ML API, the DataFrame, derived from Spark SQL, serves as the dataset for machine learning. It can accommodate diverse data types, for instance, columns that contain text, feature vectors, actual labels, and predicted values.

- **Transformer**: A Transformer is a type of algorithm that can convert one DataFrame into another. For example, a machine learning model can be viewed as a Transformer, taking a DataFrame with features and outputting a DataFrame that includes predictions.

- **Estimator**: Estimators are algorithms that, upon being "fitted" to a DataFrame, yield a Transformer. This fitting process typically involves training, as seen in learning algorithms that output a predictive model.

The Pipeline itself is a mechanism for stringing together multiple Transformers and Estimators to define a machine learning workflow. This sequence ensures that data transformation and algorithm application are carried out in an orderly and meaningful manner [22].

3.3.2.2 Spark ML Model Tuning

Spark ML provides built-in tools specifically designed for tuning hyperparameters, which are configurable parameters that are not directly learned from the data. Such tools are:

- **ParamGridBuilder**: This tool is used to construct a grid of parameters to search through during the tuning process. It systematically builds out a combination of parameters and their respective values that you want the tuning algorithm to test. This grid will be used in conjunction with a **model evaluation tool** to systematically work through multiple combinations, evaluating the performance of each to determine the best solution.

- **CrossValidator**: Cross-validation is a model validation technique for assessing how the results of a statistical analysis will generalize to an independent dataset. The CrossValidator tool in Spark ML divides the dataset into a set number of folds, which are used as separate training and test datasets. For each parameter combination specified by ParamGridBuilder, CrossValidator trains the model on the training folds and evaluates the results on the test fold. It repeats this process for each combination of parameters in the grid, allowing it to identify the most effective parameters.

- **TrainValidationSplit**: This is another model-tuning tool provided by Spark ML. Unlike CrossValidator, which uses multiple folds, TrainValidationSplit only splits the data into

two sets: one for training and the other for validation. It is a simpler and faster approach, especially useful when you want a quick estimation of the best model parameters. The TrainValidationSplit method can be particularly useful when dealing with large datasets, where the computational cost of cross-validation may be too high.

The decision to select a particular library for our project depended on multiple factors, each playing a crucial role in determining the most suitable tool for our needs. For our specific solution, we opted for Spark ML, influenced primarily by its array of advanced tuning features and functionalities, primarily the access to the **pipelines API**, its ease of use and its scalability.

3.4 Spark Structured Streaming

Fig 8: Typical Streaming Process [24]

Structured Streaming extends Spark SQL's powerful data processing capabilities to handle real-time data. In essence, Spark Streaming can source data from a variety of origins, including Apache Kafka, Kinesis, or simple TCP sockets. This data is converted into streams and segments them into discrete batches, and is subsequently processed by the Spark engine, culminating in a continuous output of processed data, also delivered in batch form [24]. In Chapter 8, we will delve into how Spark Streaming significantly enhances our system by substantially reducing response times. Its role in the proposed system is crucial, contributing critically to overall efficiency and performance.

4. Speech-To-Text

4.1 Technology Analysis

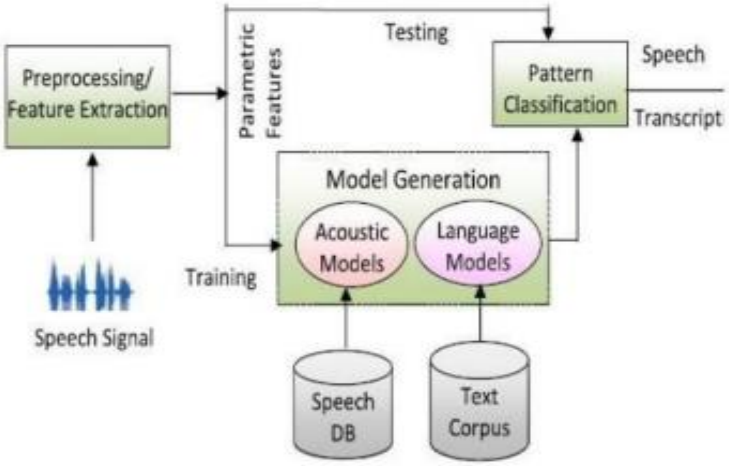


Fig 9: General Speech to Text Architecture [26]

This part focuses on the speech-to-text (STT) module integrated into the proposed system. STT technology, at its core, is a form of speech recognition software that leverages computational linguistics to convert spoken language into text. It is often referred to as speech recognition or computer speech recognition and it operates by capturing audio input and transforming it into an accurate, editable transcript. The functioning of a speech-to-text system hinges on advanced computational processes. It employs linguistic algorithms to distinguish and interpret auditory signals from spoken words, subsequently converting these signals into text represented by Unicode characters. This conversion process is underpinned by a machine learning model. The process of converting speech to text generally involves the following key steps [25]:

- Speech, which originates as a series of vibrations from vocal articulations, is captured by the speech-to-text technology. These sound vibrations are then translated into a digital format using an analog-to-digital converter, enabling the subsequent processing by the system's computational components.

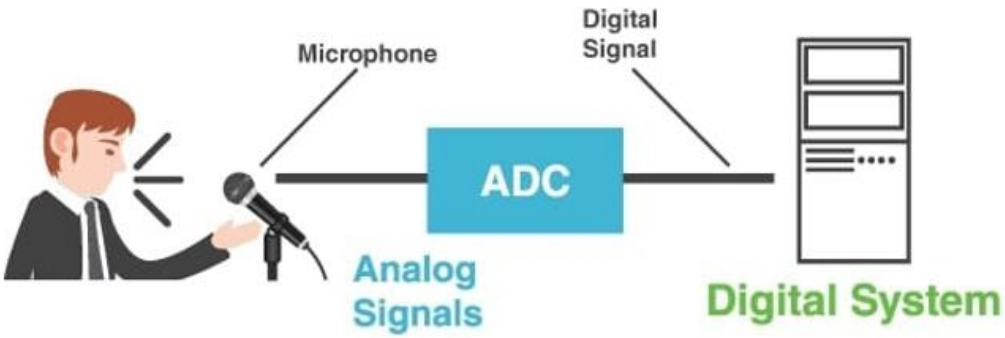


Fig 10: Speech to Text: Analog to Digital signal conversion

- The analog-to-digital converter takes sounds either from an audio file or from a real time sound stream such as a microphone, measures the waves in detail, and filters them to distinguish the relevant sounds.

- The sounds are then segmented into hundredths or thousandths of seconds and are then matched to phonemes which are the smallest distinctive sound units in a language that differentiate one word from another, such as the difference between the ‘c’ and ‘b’ sounds in ‘cat’ and ‘bat’.

- The identified phonemes are processed through a network using a mathematical model. This model compares the phonemes to a database of recognized sentences, words, and phrases for accurate interpretation.

- Finally, the speech-to-text system generates the output, presenting it either as written text or as a command for the computer, based on the most probable interpretation of the audio input.

4.2 Google Speech-To-Text API

In the proposed system, we have chosen the Google Speech-to-Text API for reasons that will be detailed subsequently. This API exemplifies major advancements in speech recognition technology. Developed by Google, it leverages sophisticated artificial intelligence and machine learning algorithms to accurately transcribe spoken language into text. As a part of Google's Cloud services, it provides developers with an efficient means to integrate speech recognition capabilities into diverse applications. The development of this API shows a significant advancement in digital communication technologies, effectively improving the interaction between human speech and computational interpretation.

The Google Speech-To-Text API offers an extensive set of features [27]:

- **Wide Language Support:** It supports over 125 languages and variants.

- **Real-Time recognition:** It provides immediate speech recognition results, processing audio input either streamed live from a microphone or from prerecorded files, whether inline or via Cloud Storage. This capability was a key factor in our decision to utilize Google Speech-to-Text in our system. We primarily leverage this feature for streaming both microphone and loopback audio data from the application to the API, subsequently receiving the transcribed text for analysis.

- **On-Premises Solution:** With Speech-to-Text On-Prem, users have the ability to leverage Google's technology within their own private data centers, offering full control over infrastructure and data privacy which is a key point when dealing with sensitive data.

- **Multichannel Recognition:** This feature makes the API able to identify and annotate distinct audio channels in complex audio environments, such as multichannel recordings or video conferences.

- **Noise Robustness:** Google Speech-to-Text efficiently processes audio with background noise, minimizing the need for additional noise reduction. This feature significantly influenced our choice to use this API in our system.

- **Domain-Specific Models:** Google offers specialized models for different applications like voice control, phone call transcription, and video transcription, each optimized for specific quality requirements.

- **Content Filtering:** An inbuilt profanity filter that helps in detecting and filtering inappropriate or unprofessional content in audio data.

- **Transcription Evaluation:** Users can evaluate the quality of transcriptions by uploading their voice data and iterating on the configuration, all without the need for coding.

- **Automatic Punctuation:** The API effectively punctuates transcriptions, accurately placing commas, question marks, periods, and other punctuation marks.

- **Speaker Diarization:** In the time of writing this dissertation this feature is still in its beta phase. It predicts the speaker for each part of the conversation, aiding in identifying who said what in multi-speaker scenarios.

The combined capabilities of the Google Speech-to-Text API align well with the requirements of our proposed system. Particularly noteworthy are its real-time prediction capabilities, enabling streaming speech-to-text conversion, and its inherent noise reduction proficiency as previously mentioned.

5. Dataset Creation

In the subsequent sections, we explore the dataset creation methodology of our research. Given the sensitivity of the data involved and the constraints in acquiring it, we opted to construct the dataset independently. This involved utilizing advanced AI tools, particularly the Chat GPT API, to generate a comprehensive dataset. To fully understand this process, we will first examine the workings of Generative Pretrained Transformers (GPT), setting the foundation for our discussion on the implementation of the Chat GPT API in developing our dataset.

5.1 Generative Pretrained Transformers

5.1.1 Transformers

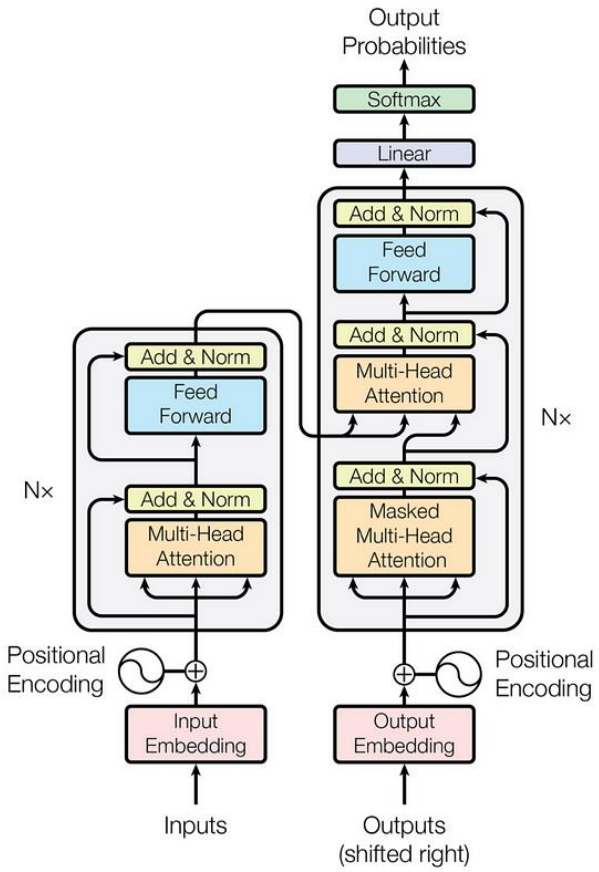


Fig 11: The Transformer – Model Architecture [30]

The transformer, a deep learning model employing the multi-head attention mechanism, distinguishes itself by the absence of recurrent units, unlike traditional sequence-to-sequence architectures like LSTM. This design feature allows for faster training compared to recurrent neural networks. Like LSTM in its objective to transform one sequence into another, the transformer is divided into two parts: the Encoder and the Decoder. This architecture has become prominent in training large language models on extensive datasets. The process involves splitting input text into n-grams and encoding these as tokens, with each token subsequently converted into a vector via word embeddings. Within the architecture, each

layer processes tokens contextually in relation to others within the context window, utilizing a parallel multi-head attention mechanism. This allows the model to amplify the importance of key tokens and diminish less significant ones, enhancing the model's ability to focus on relevant information [29][30].

5.1.2 Generative Pretrained Transformers

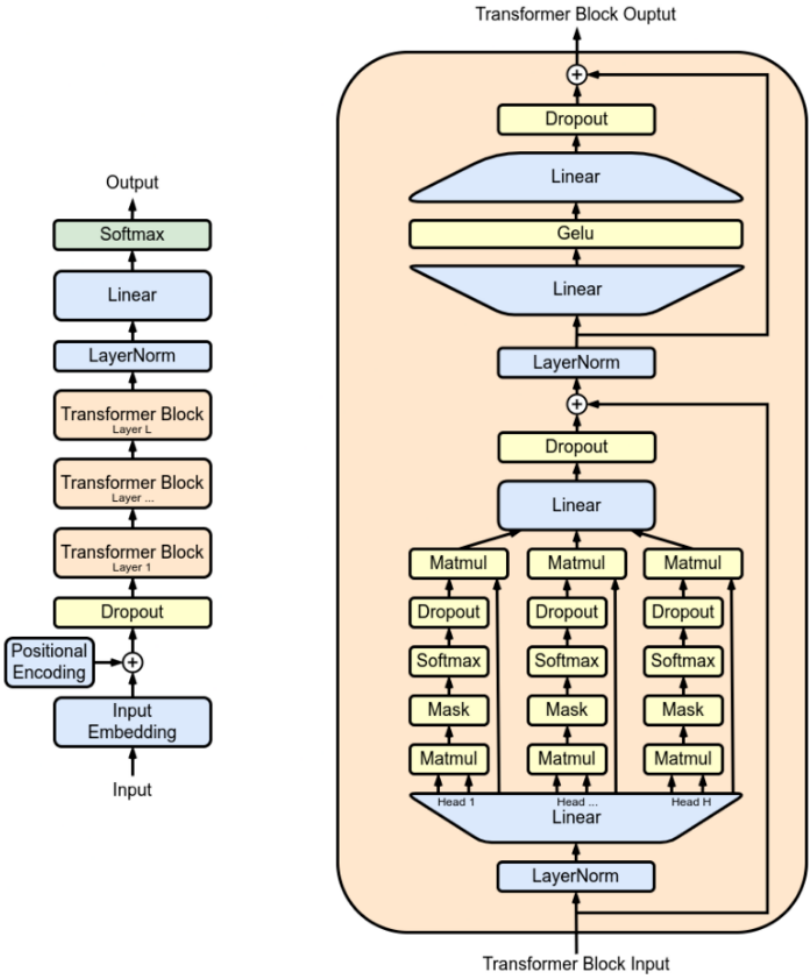


Fig 12: The Original Generative Pretrained Transformer Model [28]

Generative Pretrained Transformers (GPT) represent a class of Large Language Models (LLMs) and are a key component in the field of generative artificial intelligence. Utilized for a variety of natural language processing tasks, these neural networks are based on the transformer architecture. Pretrained on vast corpora of unlabeled text, GPTs exhibit the capability to generate text that mirrors human-like writing [28].

5.1.2.1 Utilizing GPT for Dataset Generation

Utilizing Generative Pretrained Transformers (GPT) for dataset generation capitalizes on their features, including rich language understanding and generation, the ability to produce customizable outputs, and generating data with diverse linguistic styles. Additionally, GPT models excel in simulating real-world applications and addressing data availability challenges, which was a primary factor in our decision to choose this method for dataset creation.

5.2 Data Generation with Chat GPT API

5.2.1 API Overview

Chat GPT is built upon the Generative Pretrained Transformers architecture and its fine tuned to handle conversational contexts. OpenAI, the developer of Chat GPT, provides the Chat GPT API which facilitates the creation of interactive and responsive chatbot systems capable of understanding and generating human-like text responses. This API can process and respond to a wide range of conversational inputs, maintain context over a series of interactions, and generate replies that are not only contextually relevant but also exhibit natural language coherence.

5.2.2 Prompt Engineering

In this research, we utilized the Chat GPT API, combined with Python, to generate our dataset. Prior to detailing the specific methods employed in data production, it's essential to understand the concept of prompts and the emerging field of prompt engineering. This field has gained significance with the advent of GPT systems in recent years and plays a crucial role in effectively interacting with the Chat GPT API [31].

Prompt engineering is the technique of crafting text inputs that are optimized for interpretation by generative AI models, such as Chat GPT. A prompt is essentially a piece of natural language text that outlines a task for the AI to execute. The art of prompt engineering might involve the formulation of a specific query, choosing a particular style, adding pertinent context, or assigning a role to the AI, e.g., "Act as a native French speaker" [31].

The ensuing pipeline provides a clear example of constructing a prompt designed to ensure that the AI algorithm fully understands the extent of the request.

- **Task:** A good prompt provides a clear task for the generative AI algorithm to pursuit.

- **Contextual Balance in Prompts:** In prompt creation, it's crucial to provide context, but with moderation. Striking a balance is key, as providing too much information can be as counterproductive as giving too little. The goal is to include just enough context to elicit an effective response.

- **Examples:** Providing an example of an ideal response to the algorithm can significantly enhance the quality of the resultant output, especially if the request is of complex nature.

- **Persona:** Incorporating a persona into the AI system can improve the probability of obtaining a response that closely mirrors real-life interactions. While not essential, this aspect of the prompt design is particularly beneficial for scenarios aiming to emulate specific individuals or target demographics.

- **Format:** Utilizing a specific format can significantly enhance the response quality, especially if the output is intended for specific uses, such as dataset creation. For instance, structuring the response in JSON format, which can later be converted into a dataframe, is an effective approach to streamline data processing.

5.2.3 Data Collection Strategy

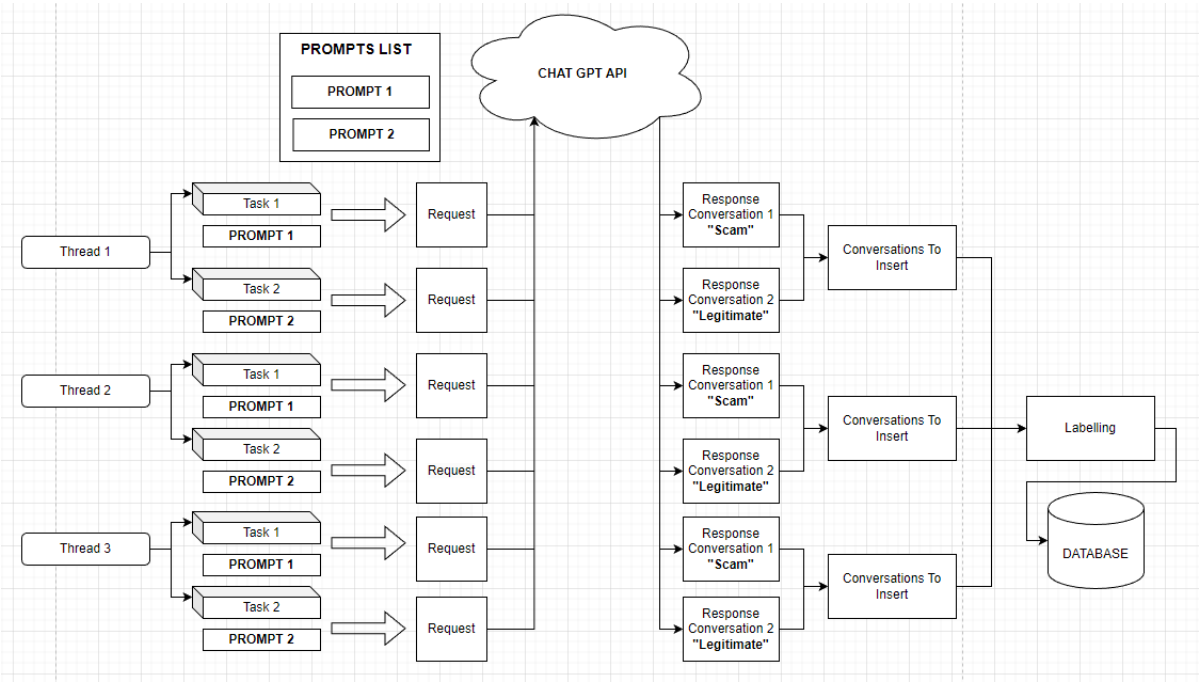


Fig 13: Dataset Creation Pipeline

To understand how the dataset was constructed the following points must be discussed:

- **Prompts**: Two prompts were created in order facilitate the construction of the dataset.

    - The first directs the AI algorithm to simulate a conversation between two individuals, with one person (the attacker) attempting to scam the other (the victim). The algorithm is guided to present the response in JSON format. Additionally, it is instructed to consider the characteristic attitudes and tones of scammers and their typical targets in phone scams, ensuring a realistic portrayal of such interactions.

    - The second prompt similarly guides the AI algorithm, but with a different context. Here, the algorithm is tasked with generating a routine conversation between a client and customer support. The objective is to create a dataset contrasting with the previous scam scenario, focusing instead on a typical, everyday interaction between a customer and support representative.

- **Multithreading:** To expedite dataset generation, a multithreading approach was employed. When launching the dataset creation program, multiple threads (denoted as 'N') are activated. Each thread is assigned two tasks: One to invoke the Chat GPT API using the first prompt (focused on simulating a scam interaction) and one to use the second prompt (geared towards creating a normal customer support dialogue).

- **Data Labeling Process:** Post-generation by the Chat GPT API, each conversation is appropriately labeled to facilitate subsequent analysis. Conversations derived from the scam-focused prompt are labeled as “1”, indicating a scam scenario, while those from the normal interaction prompt are marked as “0”, denoting a typical, non-scam

exchange. This labeling is crucial for distinguishing between the two types of interactions in the dataset.

- **Storing Conversations in CSV Format:** The concluding phase of the dataset creation involves saving the generated conversations into a CSV file, which serves as the dataset for our system. Each conversation is assigned a unique identifier upon entry into the CSV, ensuring easy differentiation and retrieval within the dataset. This step solidifies the structure and accessibility of the data for future use.

Conversation Id	Attacker_Helper	Victim
SsDyLEV3XJ4lc3	Hello, this is John from the ABC Bank. We noticed some suspicious activity on your account. Can I confirm if you made a purchase of $500 at an online store today?	Yes, I made that purchase. Thank you for checking before verifying!
SsDyLEV3XJ4lc3	Thank you for confirming. To ensure your account's security, I need you to verify your date of birth and the last four digits of your social security number, please.	I appreciate the caution, but I prefer not to share that sensitive information over the phone. Can we verify through another method?
SsDyLEV3XJ4lc3	Absolutely. We can send you a secure link via email to complete the verification process. Which email address should we send it to?	Please send it to Thank you for your assistance!
SsDyLEV3XJ4lc3	You're welcome! Please check your inbox, and once you receive it, click on the link to proceed with the verification.	Got it, I'll check my email and complete the verification promptly. Thanks again for your help!
SsDyLEV3XJ4lc3	My pleasure! Once you've completed the verification, your account will be secure. Should you have any further questions, feel free to reach out. Have a great day!	Thank you so much for your support. Have a wonderful day too!

Fig 14: Result of a normal (non-scam) conversation from Chat GPT API.

Conversation Id	Attacker_Helper	Victim
LPlvHieleyVcC1	Hello, this is Microsoft Support. Your computer has been infected with a virus.	Oh no, that's terrible! What should I do?
LPlvHieleyVcC1	Don't worry, we can fix it. But first, I need access to your computer.	I'm not sure if I should trust you. How can I be sure you're really from Microsoft?
LPlvHieleyVcC1	I understand your concern. You can verify my identity by checking the caller ID. It should display our official helpline number.	Okay, let me check. Oh, the number on my caller ID matches! That's reassuring.
LPlvHieleyVcC1	Great! Now, to fix the problem, I need you to download a remote access software.	Wait, why can't I just take my computer to a local technician?
LPlvHieleyVcC1	Taking your computer to a technician is expensive and time-consuming. We can save you both time and money by fixing it remotely.	I suppose that makes sense. Alright, what should I do next?
LPlvHieleyVcC1	Visit our website and download our remote access software. I'll guide you through the installation process.	Okay, I'm on the website now. Where should I click?
LPlvHieleyVcC1	Click on the 'Download Now' button and follow the instructions. Once installed, give me the access code displayed on your screen.	Alright, let me click on that. Done. The access code is 123456.
LPlvHieleyVcC1	Perfect! Now I have full control of your computer. I'll fix the virus issue for you.	Thank you so much for helping me. I really appreciate it.
LPlvHieleyVcC1	You're welcome. Just remember, never share your personal information or download any suspicious software in the future.	I'll keep that in mind. Thanks again for your assistance.
LPlvHieleyVcC1	No problem. Have a great day!	You too, goodbye!

Fig 15: Result of a scam conversation from Chat GPT API.

As illustrated in Figure 14 and Figure 15, the sample conversations generated by the Chat GPT API, while not flawless, effectively mirror a real-life scenario in its fundamental aspects.

5.2.4 Data Quality and Future Improvements

Although the dataset created via the outlined pipeline sufficiently serves this research's needs, enhancing its diversity is recommended for a more comprehensive representation. This enhancement could involve developing more sophisticated prompts or integrating real-

world data into the dataset. Additionally, since the conversations in the dataset adhere to a pattern as it would be expected from data generated by a machine, including a variety of more complex scam conversations in the dataset is advisable to accurately reflect the range of tactics employed by scammers and the intricacies of their schemes.

6. Dataset Preprocessing

In any machine learning pipeline, data preprocessing is a vital step. It involves various processes designed to enhance data quality. Within this research, data preprocessing was conducted in two phases: the initial preprocessing to clean and prepare the data, and subsequent processing tailored for the data's compatibility with machine learning models.

6.1 Initial Preprocessing

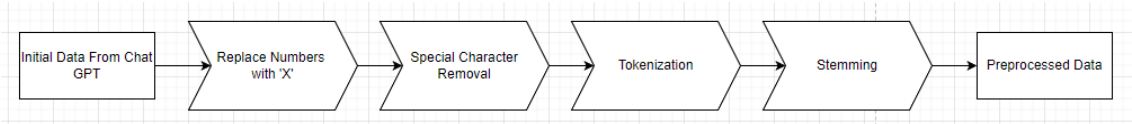


Fig 16: Initial Preprocessing Pipeline

In the initial preprocessing phase, the objective is to structure and standardize the data to ensure consistency for more intricate processing stages that follow. The initial preprocessing pipeline comprises three critical steps, as depicted in Figure 16. These steps are detailed in the ensuing sub-sections.

6.1.1 Numeric Character Replacement

During this step, numeric characters in each conversation are substituted with the letter 'X'. This approach serves several purposes:

- **Anonymization**: Either using the synthetically generated dataset or using a real one, the dataset may include data that resembles sensitive information, such as personal or financial details. Replacing numbers helps to anonymize this information, ensuring privacy and adherence to data protection standards like GDPR.

- **Standardization**: In the context of this study, the presence of a numeric value is more significant than the actual number itself. Substituting numbers with 'X' enhances data uniformity across the dataset.

- **Complexity Reduction**: Numeric values can add unnecessary complexity to machine learning models, particularly in natural language processing tasks.

- **Emphasis on Textual Data**: The objective is for the machine learning models to concentrate on textual and categorical data, rather than numeric values, to improve their performance on the relevant tasks.

6.1.2 Special Character Removal

In this step special characters are removed from each conversation; this aims to support the following key points:

- **Feature Simplification**: To improve the focus of our models, during the training process, on those features that provide important information about the conversation we have opted to remove special characters since those might introduce

unnecessary additional features that can complicate the model without adding informative value.

- **Tokenization Consistency**: During tokenization, special characters can lead to the creation of tokens with embedded special characters or tokens that consist only a single special character. Removing these characters helps achieve a more consistent and precise tokenization process.

- **Improving Model Focus**: As already mentioned before, the aim is to direct the machine learning model's attention toward text that expresses sentiment or factual content, rather than parsing potentially extraneous details signified by special characters.

6.1.3 Tokenization

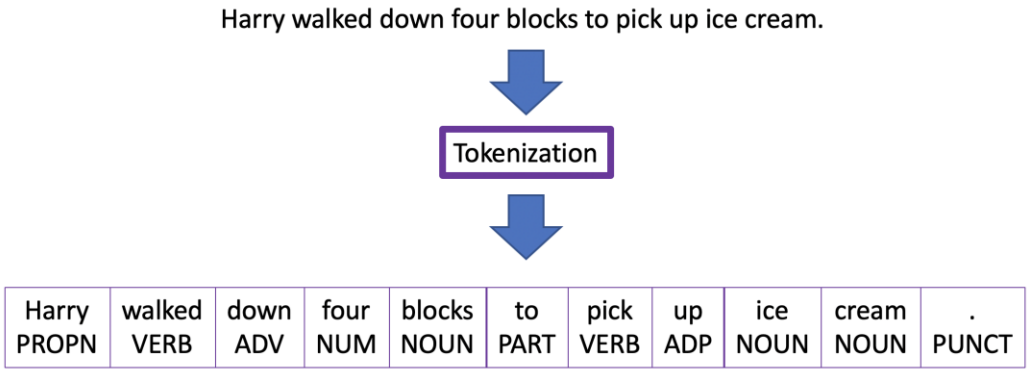


Fig 17: Word Tokenization Example [34]

In this stage, we explore tokenization, a fundamental step in natural language processing and machine learning. This process involves breaking down sentences, words, or larger bodies of text into smaller units called tokens. Tokenization serves the purpose of rendering human language into segments that are more manageable and interpretable for computational analysis. Tokenization can be categorized into three types: Word Tokenization, Character Tokenization, and Sub word Tokenization. For the purposes of this research, we focus on Word Tokenization, which segments text into distinct words [33].

In this dissertation, the focus is on utilizing the Natural Language Toolkit (NLTK) library in Python, with a specific emphasis on the word_tokenize function. This function operates using the PunktSentenceTokenizer, which is adept at identifying sentence boundaries through punctuation and character markers. Additionally, it incorporates the TreebankWordTokenizer, which effectively separates words within a sentence based on spaces and punctuation. This combination is highly effective for processing European languages, making it well-suited for this study where English, is the language of focus [35][36].

6.1.4 Stemming

	original_word	stemmed_words
0	connect	connect
1	connected	connect
2	connection	connect
3	connections	connect
4	connects	connect

	original_word	stemmed_word
0	trouble	troubl
1	troubled	troubl
2	troubles	troubl
3	troublemsome	troublemsom

Fig 18: Example of Stemming in the English language [38]

The last stage in our preliminary preprocessing sequence is stemming. As depicted in Figure 18, stemming involves condensing words with inflections down to their basic stem, which represents the core or fundamental form of the word. It's important to note that this stem may not always match the morphological root of the word [37][39].

A widely recognized and frequently utilized stemming algorithm is the Porter Stemmer. Originating in the 1980s, the Porter Stemmer stands as one of the foremost popular methods for stemming. Characterized by its simplicity and computational efficiency, it excels in handling English language's inflectional variations, like pluralization and tense changes. This ability significantly aids in generalizing patterns within text data. Tailored specifically for English, the algorithm demonstrates robust performance across a broad spectrum of English vocabulary, effectively accommodating even irregular word formations for example the word “running”. The irregularity in this instance is the double “n” before the addition of “-ing”. The algorithm would reduce “running” to its stem, “run” [37].

As a conclusion to the preprocessing pipeline, all conversations are saved in a new csv file with the format depicted in the subsequent image.

Conversation_ID	Attacker_Helper	Victim	Conversation_Type
GT1sURbxgG	[['hi', 'thi', 'is', 'john', 'from', 'the', 'ir', 'you', 'owe', 'us', 'xxxxx', 'in', 'back', 'tax'], ['ye', 'of',	[['im', 'sorri', 'i', 'dont', 'believ', 'you', 'can', 'you', 'provid', 'me', 'with', 'your', 'badg	1
TwaGOeC96w	[['hello', 'thi', 'is', 'xyz', 'bank', 'call', 'may', 'i', 'speak', 'with', 'victim', 'name', 'pleas'], ['goc	[['thi', 'is', 'victim', 'name', 'how', 'can', 'i', 'help', 'you'], ['im', 'sorri', 'i', 'dont', 'feel	0
V73ZDCviQL	[['hello', 'sir', 'thi', 'is', 'a', 'call', 'from', 'the', 'ir', 'we', 'have', 'found', 'discrep', 'in', 'your', '	[['ive', 'never', 'had', 'ani', 'issu', 'with', 'my', 'tax', 'befor', 'can', 'you', 'provid', 'me'	1
wNsWdbpeId	[['hello', 'thi', 'is', 'john', 'from', 'the', 'bank', 'i', 'am', 'call', 'to', 'confirm', 'a', 'recent', 'tran	[['sure', 'but', 'befor', 'that', 'can', 'you', 'pleas', 'confirm', 'which', 'bank', 'you', 'are	0
x0pSxAQx1K0abm	[['hello', 'thi', 'is', 'the', 'ir', 'call', 'our', 'record', 'show', 'that', 'you', 'owe', 'xxxx', 'in', 'tax',	[['oh', 'no', 'i', 'had', 'no', 'idea', 'can', 'you', 'give', 'me', 'some', 'more', 'inform', 'ab	1
I8QHkmsq5AdwAq	[['hello', 'thi', 'is', 'xyz', 'bank', 'can', 'i', 'speak', 'with', 'john', 'smith', 'pleas'], ['we', 'notic',	[['thi', 'is', 'john', 'smith', 'speak', 'may', 'i', 'know', 'whi', 'are', 'you', 'call'], ['oh', 'th	0
v9vN0BYTWa0e16	[['hello', 'thi', 'is', 'john', 'from', 'the', 'ir', 'your', 'tax', 'return', 'from', 'the', 'last', 'year', 'ha	[['im', 'sorri', 'i', 'dont', 'believ', 'you', 'can', 'you', 'give', 'me', 'your', 'employe', 'id'	1
pVeYjXj2DelffL	[['hello', 'thi', 'is', 'john', 'from', 'your', 'bank', 'may', 'i', 'speak', 'to', 'mr', 'smith'], ['hi', 'mr	[['speak', 'thi', 'is', 'mr', 'smith', 'what', 'can', 'i', 'help', 'you', 'with'], ['no', 'i', 'didnt	0
42e0AXIkESKc6y	[['hello', 'is', 'thi', 'xxxxxxxxxx'], ['i', 'am', 'call', 'from', 'the', 'ir', 'and', 'you', 'owe', 'us', 'mc	[['ye', 'who', 'is', 'thi'], ['i', 'dont', 'believ', 'you', 'can', 'you', 'prove', 'that', 'you', 'ar	1
vk2TK8MFd0hoZg	[['hello', 'thi', 'is', 'john', 'from', 'the', 'bank', 'im', 'call', 'in', 'regard', 'to', 'some', 'recent', 'i	[['oh', 'my', 'is', 'everyth', 'alright'], ['sure', 'my', 'name', 'is', 'sarah', 'johnson', 'and	0
YX24MIG7RJy5zD	[['hello', 'there', 'am', 'i', 'speak', 'with', 'mr', 'john', 'doe'], ['thi', 'is', 'toni', 'from', 'your', 't	[['ye', 'thi', 'is', 'john', 'who', 'is', 'thi'], ['no', 'i', 'did', 'not', 'make', 'ani', 'such', 'purc	1
VH7eX10IAQbZOV	[['hello', 'thi', 'is', 'the', 'ir', 'we', 'notic', 'that', 'you', 'owe', 'back', 'tax', 'and', 'are', 'at', 'risl	[['im', 'sorri', 'but', 'i', 'dont', 'think', 'that', 'true', 'i', 'alway', 'pay', 'my', 'tax', 'on', 't	1
VBwnvERVuU3jv9	[['hello', 'thi', 'is', 'john', 'from', 'the', 'bank', 'i', 'need', 'to', 'confirm', 'some', 'account', 'in	[['ok', 'sure', 'what', 'do', 'you', 'need'], ['i', 'dont', 'feel', 'comfort', 'give', 'that', 'ov	0
1gJHIiw7AGlHxH	[['hello', 'thi', 'is', 'the', 'ir', 'we', 'have', 'detect', 'fraudul', 'activ', 'on', 'your', 'account'], ['y	[['oh', 'realli', 'can', 'you', 'provid', 'me', 'with', 'some', 'more', 'inform'], ['im', 'sorri	1

Fig 19: Indicative Data Post Initial Preprocessing

Before Preprocessing	After Preprocessing
Hello, this is Microsoft Support. Your computer has been infected with a virus. **Don't worry, we can fix it. But first, I need access to your computer.**	[['hello', 'thi', 'is', 'microsoft', 'support', 'your', 'comput', 'ha', 'been', 'infect', 'with', 'a', 'viru'], ['dont', 'worri', 'we', 'can', 'fix', 'it', 'but', 'first', 'i', 'need', 'access', 'to', 'your', 'comput']]

Fig 20: Example Output from the Initial Preprocessing Pipeline

6.2 Preprocessing Pipeline for Model Training

For effective model training, it is important to convert the data into a format that can be comprehended and processed by machine learning algorithms. This involves transforming the raw data, from the initial preprocessing pipeline, into a numerical representation, as machine learning models inherently require data in a quantifiable format for analysis and pattern recognition.

To achieve the necessary numeric representation, the data will be subjected to a series of transformation steps, employing the PySpark pipelines API, as detailed in Section 3.3.2.1. This process involves the data passing through a sequence of transformers, collectively forming a pipeline, which is illustrated in the subsequent figure.

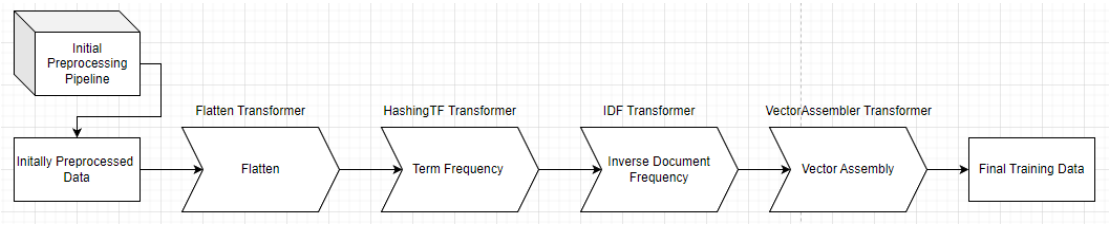


Fig 21: Preprocessing Pipeline for Model Training

6.2.1 Flatten Transformer

The first stage in the preprocessing pipeline for model training, as depicted in Figure 20, involves flattening each conversation. Prior to this stage, conversations are represented in two columns as per Figure 19, with each column containing the sentences spoken by an individual in the conversation. To facilitate more efficient preprocessing, it is essential to convert these lists of sentences into a single array for each individual. This transformation will result in an array that encompasses all the words uttered by each participant in the conversation.

To facilitate this transformation, the Flatten Transformer provided by the PySpark Machine Learning package is employed. This transformer is designed to merge an array of arrays into a single, unified array, deeming it suitable for our requirements [40].

6.2.2 TF-IDF

To transform the textual data into a numeric format, we utilize the TF-IDF (Term Frequency-Inverse Document Frequency) algorithm. This algorithm evaluates the significance of a word within a document and is extensively employed in text-based systems. The TF-IDF algorithm comprises two key components, Term Frequency (TF) and Inverse Document Frequency (IDF) [41].

- The term frequency is given conceptually as follows:

$$\boldsymbol{TF(t,d) = \frac{Number\ of\ times\ the\ term\ t\ appears\ in\ a\ document\ d}{Total\ Number\ of\ all\ terms\ in\ the\ document\ d}}$$

In a more analytical representation, the term frequency can be calculated as [41]:

$$\boldsymbol{TF(t,d) = \frac{ft,d}{\sum_{\acute{t}\in d} f\acute{t},d}}$$

where ft, d signifies the raw count of a term in a document 'd'. In this research, the term 'document' refers to an individual row within the Attacker_Helper or Victim columns, as depicted in Figure 19.

- The inverse document frequency is given conceptually as follows:

$$\boldsymbol{IDF(t,D) = \log\left(\frac{Total\ number\ of\ documents\ N}{Number\ of\ documents\ where\ the\ term\ t\ appears}\right)}$$

In a more analytical representation, the inverse document frequency can be calculated as [41]:

$$\boldsymbol{IDF(t,D) = \log\left(\frac{N}{|\{d \epsilon D : t \epsilon d\}|}\right)}$$

Where 'N' represents the total number of documents in the corpus: In the context of this research, the corpus is defined as each individual row in the column to which the IDF is applied. Consequently, 'N' equates to the total number of rows in the dataset [41].

- Finally, the TF-IDF metric is given by the following equation:

$$\boldsymbol{TF_{IDF(t,d,D)} = TF(t,d) \times IDF(t,D)}$$

The TF-IDF metric proficiently signifies the relevance of a word within a document in a corpus. A higher TF-IDF score for a particular term suggests its rarity in both the document and the corpus, thereby denoting greater significance.

6.2.2.1 HashingTF Transformer

To incorporate the term frequency step into the pipeline, the HashingTF transformer provided by PySpark is utilized. This transformer transforms a document into a fixed-size vector, a requirement for machine learning models that necessitate a constant input vector size to function optimally.

To achieve this fixed vector sized the HashingTF transformer uses the “Hashing Trick”. The "Hashing Trick" is an efficient technique for encoding features, particularly useful in handling high-dimensional data like text. Using a hash function, it maps features into indices within a provided fixed-size vector. This approach is efficient as it does not require a vocabulary of all possible features, saving memory and reducing complexity. A notable limitation is the occurrence of collisions, where different features map to the same index, potentially leading to information loss, hence is it important to find the correct size for the vector in order to avoid such collisions [42].

Considering that phone conversations typically involve a limited number of unique words; we have selected 200 as the number of features for the HashingTF transformer. This decision is based on the expectation that no more than 100 unique words will be utilized in any given conversation.

6.2.2.2 IDF Transformer

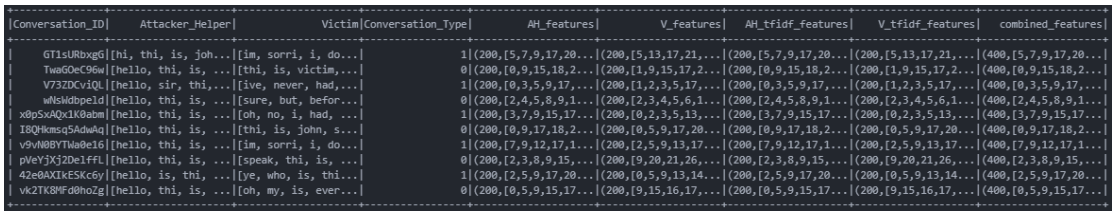
Once the data has been processed through the HashingTF transformer, it is converted into a collection of TF vectors, with each row or document in the 'Attacker_Helper' and 'Victim' columns represented by a sparse vector. To finalize the TF-IDF process, these TF vectors must then be passed through the IDF transformer. This transformer is specifically tailored to operate on the numerical vectors generated in the TF phase, adjusting the term frequencies within these vectors according to the terms' prevalence or rarity throughout the entire dataset.

The final output after applying IDF will be TF-IDF sparse vectors. These vectors represent both the term frequencies and the importance of the terms in the context of the entire dataset.

6.2.3 Vector Assembler Transformer

The concluding phase in the preprocessing pipeline for model training is Vector Assembly. During this stage, the feature vectors generated from the 'Attacker_Helper' and 'Victim' columns in previous steps are merged into a single, comprehensive vector. The resultant vector is effectively twice the size of either individual vector. This concatenation is achieved by inputting the data from both columns into the Vector Assembler transformer provided by PySpark.

An indicative result of the complete pipeline output can be seen in the subsequent figure.



Conversation_ID	Attacker_Helper	Victim	Conversation_Type	AH_features	V_features	AH_tfidf_features	V_tfidf_features	combined_features
GT1sURbxgG	[hi, thi, is, joh...	[im, sorri, i, do...	1	(200,[5,7,9,17,20...	(200,[5,13,17,21,...	(200,[5,7,9,17,20...	(200,[5,13,17,21,...	(400,[5,7,9,17,20...
TwaGOeC96w	[hello, thi, is, ...	[thi, is, victim,...	0	(200,[0,9,15,18,2...	(200,[1,9,15,17,2...	(200,[0,9,15,18,2...	(200,[1,9,15,17,2...	(400,[0,9,15,18,2...
V73ZDCviQL	[hello, sir, thi,...	[ive, never, had,...	1	(200,[0,3,5,9,17,...	(200,[1,2,3,5,17,...	(200,[0,3,5,9,17,...	(200,[1,2,3,5,17,...	(400,[0,3,5,9,17,...
wNsWdbpeld	[hello, thi, is, ...	[sure, but, befor...	0	(200,[2,4,5,8,9,1...	(200,[2,3,4,5,6,1...	(200,[2,4,5,8,9,1...	(200,[2,3,4,5,6,1...	(400,[2,4,5,8,9,1...
x0pSxAQx1K0abm	[hello, thi, is, ...	[oh, no, i, had, ...	1	(200,[3,7,9,15,17...	(200,[0,2,3,5,13,...	(200,[3,7,9,15,17...	(200,[0,2,3,5,13,...	(400,[3,7,9,15,17...
I8QHkmsq5AdwAq	[hello, thi, is, ...	[thi, is, john, s...	0	(200,[0,9,17,18,2...	(200,[0,5,9,17,20...	(200,[0,9,17,18,2...	(200,[0,5,9,17,20...	(400,[0,9,17,18,2...
v9vN0BYTWa0e16	[hello, thi, is, ...	[im, sorri, i, do...	1	(200,[7,9,12,17,1...	(200,[2,5,9,13,17...	(200,[7,9,12,17,1...	(200,[2,5,9,13,17...	(400,[7,9,12,17,1...
pVeYjXj2De1fFL	[hello, thi, is, ...	[speak, thi, is, ...	0	(200,[2,3,8,9,15,...	(200,[9,20,21,26,...	(200,[2,3,8,9,15,...	(200,[9,20,21,26,...	(400,[2,3,8,9,15,...
42e0AXIkESKc6y	[hello, is, thi, ...	[ye, who, is, thi...	1	(200,[2,5,9,17,20...	(200,[0,5,9,13,14...	(200,[2,5,9,17,20...	(200,[0,5,9,13,14...	(400,[2,5,9,17,20...
vk2TK8MFd0hoZg	[hello, thi, is, ...	[oh, my, is, ever...	0	(200,[0,5,9,15,17...	(200,[9,15,16,17,...	(200,[0,5,9,15,17...	(200,[9,15,16,17,...	(400,[0,5,9,15,17...

Fig 22: Example of Data Post-Processing Through the Model Training Preprocessing Pipeline

As illustrated in Figure 22, the "combined_features" sparse vector represents the final output that will be utilized for training the machine learning models.

6.3 Preprocessing Pipeline for Deployment

In addition to the pipeline developed for model training, a separate pipeline is established to manage data preprocessing in the system's actual deployment and usage. This deployment pipeline diverges from the training pipeline only in its initial step. In real-world scenarios, the data is already in a flattened format, rendering the flattening step redundant. Therefore, the data bypasses the flatten transformer. Subsequent steps of this pipeline remain consistent with those detailed in the preceding sections.

7. Model Architecture – Training – Tuning

Until now, this dissertation has detailed the generation and preprocessing of data, crucial steps to prepare them for utilization in machine learning models. We have meticulously navigated through the intricacies of data creation and the comprehensive preprocessing pipeline, ensuring that the data are optimally structured for both training and deployment phases. With these foundational aspects established, we now transition to a critical segment of this research: the exploration of the machine learning models themselves.

This section delves into the selection of models, dissecting their architectural frameworks and internal mechanisms. We will thoroughly examine the rationale behind choosing specific models, their design intricacies, and how they align with the unique challenges of our dataset. Additionally, this section will provide a detailed account of the training process, including strategies employed to optimize model performance.

Furthermore, we will explore the nuances of model tuning, discussing the techniques and methodologies implemented to fine-tune the models for enhanced performance and efficiency. The section will culminate in an evaluation of the models' performance during training, offering insights into their efficacy, strengths, and limitations. Through this comprehensive exploration, we aim to provide a clear understanding of the model lifecycle, from architectural design to performance evaluation, in the context of our specific machine learning endeavor.

7.1 Machine Learning Basics

Machine Learning is a transformative branch of artificial intelligence (AI) that focuses on the development and application of algorithms capable of learning from and making decisions or predictions based on data and patterns. Unlike traditional programming where rules and decision criteria are explicitly coded, machine learning algorithms build a model based on sample data, known as ‘training data’, to make predictions or decisions without being programmed to perform the relevant task. This ability to automatically learn and improve from experience makes machine learning a core technology in today’s data driven world [43].

There are 3 types of machine learning, each has its unique approach and application areas:

- **Supervised Learning:** In the realm of machine learning, Supervised Learning (SL) is a core approach. It involves training an algorithm using a dataset that comprises of both the input elements, often in the form of predictor variable vectors, and their corresponding desired outcomes, known as supervised labels. The training encompasses a set of observations $T = (xi, yi), i = 1, \dots, N$ where the inputs xi are processed by a learning

algorithm which subsequently produces outputs $f(xi)$. *T*he fundamental objective of supervised learning is to develop a function that can accurately predict expected output values for new, unseen data, a process requiring the model to extend its learned patterns from the training data to novel situations effectively. This extension, or generalization, of the model's learning from the training set to unfamiliar scenarios is a critical aspect. Throughout training, the supervised learning algorithm refines its input-output relationship, adapting in response to the error $yi - f(xi)$ between the actual and predicted outputs. This iterative adjustment, referred to as learning by example, aims to reduce the aforementioned error, thus honing the algorithm's predictive capabilities. Following the learning phase, the expectation is that the algorithm's output predictions $f(xi)$ will sufficiently match the true outputs for the entire range of potential input scenarios. This precision renders supervised learning particularly effective for tasks that require precise output prediction from given inputs, such as in classification and regression scenarios [44][45].

- **Unsupervised Learning:** Another core type is unsupervised learning; In this machine learning type, the learning phase involves the network attempting to replicate the input data it receives. This replication process is key to the network's self-improvement. The network uses the discrepancies between its generated outputs and the original input data to adjust itself, specifically by altering its weights and biases. This error, which drives the learning process, can manifest in various forms. In some instances, it's represented as a low probability, indicating the unlikelihood of the network's output being correct. Alternatively, this error might be depicted as a high-energy state within the network, signifying instability and deviation from the expected output [46]. In contrast to supervised learning there are no predefined labels or responses to guide the learning process, hence the network must rely on these internal error signals to refine its understanding and representation of the data. This approach underscores the exploratory and adaptive nature of unsupervised learning, where the network continuously evolves and adjusts its internal parameters in response to the intrinsic patterns and structures it discovers within the data.

- **Reinforcement Learning (RL):** Reinforcement learning, a distinct segment within machine learning, stands apart from the supervised and unsupervised learning approaches. This method involves an agent that learns by interacting with its environment: it makes decisions, performs actions, and receives feedback in terms of rewards or penalties. The central principle is the maximization of accumulated rewards over time. The agent must strike a balance between exploring new actions and exploiting known, rewarding ones. The learning cycle consists of the agent assessing the environment's state, acting accordingly, and then adapting based on the feedback and new state provided by the environment. This approach is aimed at continually enhancing the agent's strategy for optimal reward accumulation. Reinforcement learning, however, faces challenges like high computational demands and a significant need for data, particularly in scenarios where rewards are infrequent or delayed. This paradigm, centered around learning from trials, maximizing rewards, and managing the exploration-exploitation dichotomy, is fundamental to reinforcement learning [48].

In the context of our specific problem, supervised learning emerges as the optimal approach. Given that our task falls in the category of binary classification, a conversation is either a scam or it is not a scam, algorithms within the supervised learning paradigm are anticipated to be the most effective.

7.2 Pre-Modeling Considerations

Prior to delving into the specific models and algorithms chosen for this research, it is essential to understand several fundamental concepts.

7.2.1 Overfitting

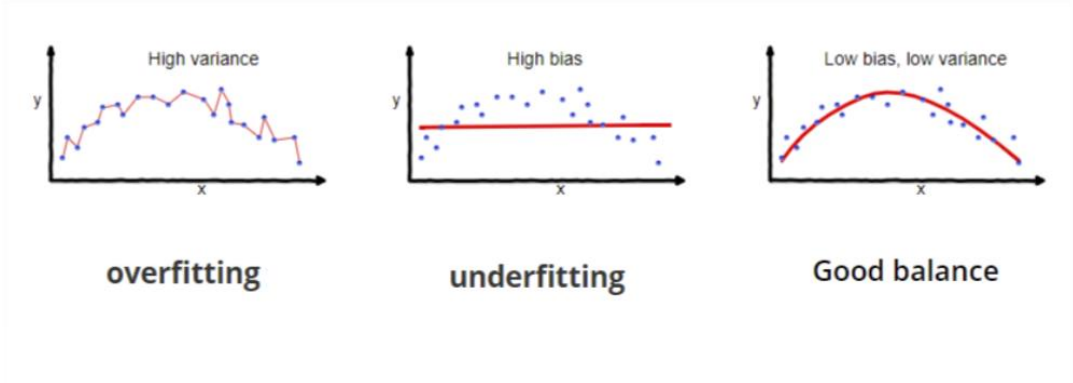


Fig 23: Example of Overfitting [53].

Overfitting primarily occurs in supervised learning and is characterized by a model's inability to generalize effectively from the training data to new, unseen data. This phenomenon is typically indicated by perfect or near-perfect performance during training and validation, but a marked decrease in performance when the model encounters the test set. However, if the test set closely resembles the training data, for instance, if both are derived from the same dataset through a train-validation-test split, the model may still exhibit high performance on the test set. Overfit models often memorize the data, including noise, instead of understanding and generalizing the underlying principles of the data [49].

To avoid overfitting in machine learning models, several strategies can be employed. Most notably:

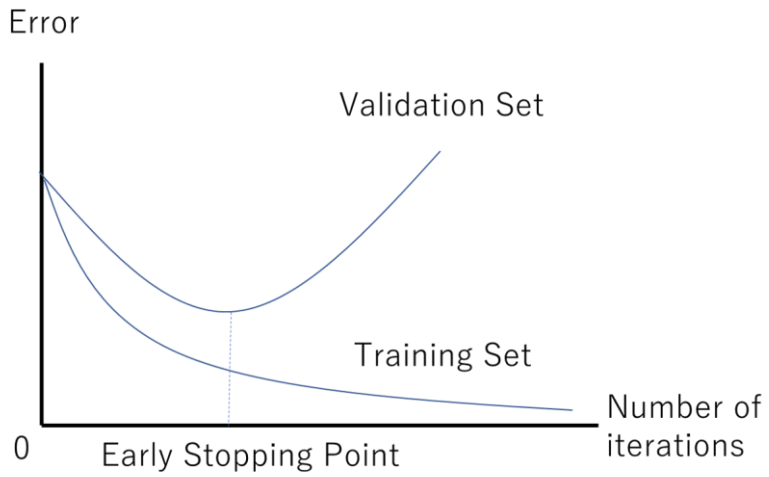


Fig 24: Example of Early Stopping Point [50]

- **Early Stopping:** This technique involves halting the training process when the model's performance on the validation set begins to decline. As illustrated in Figure 50, training ceases once the validation loss plateaus and then starts to increase. By implementing early stopping, we ensure that the model is retained in its optimal state of training. Continuing training beyond this point would likely lead to a deterioration in the model's ability to generalize, to new data. Early Stopping is particularly useful in training neural networks where longer training times often lead to overfitting [49].

- **Data Augmentation:** This technique involves generating additional data by introducing variations to the existing dataset. It's achieved through modifications such as rotations, scaling, and other transformations in the case of image data, or synonym replacement and sentence restructuring for text data. Data augmentation not only helps in mitigating overfitting but is especially beneficial when dealing with small datasets that lack diversity. By expanding the dataset and introducing a wider range of examples, models can learn more robust and generalized patterns, enhancing their performance and reliability on unseen data [51][52].

- **Regularization:** This technique involves adding a penalty to the loss function to encourage simpler models. The three types of regularization are [53]:

    - **Lasso Regularization (L1)** plays a key role in inducing sparsity within a model by shrinking certain coefficients to zero, effectively aiding in feature selection. The mechanism of Lasso Regularization is analytically represented in the following equation:

    $$\boldsymbol{Cost} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{yi} - \widehat{\boldsymbol{yi}})^{\mathbf{2}} + \boldsymbol{\lambda}\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}}|\boldsymbol{wi}|$$

    Where the first term of the equation is the mean squared error (MSE) over n samples, and the second term is the regularization term, where λ is a non-negative hyperparameter that controls the strength of the regularization. The regularization term is the sum of the absolute values of the model coefficients wi, summed over m features.

    - **Ridge Regularization** (L2) minimizes coefficients but doesn't reduce them to zero, helping to manage multicollinearity and model complexity. The mechanism of Ridge Regularization is analytically represented in the following equation:

    $$\boldsymbol{Cost} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{yi} - \widehat{\boldsymbol{yi}})^{\mathbf{2}} + \boldsymbol{\lambda}\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}}\boldsymbol{wi}^{\mathbf{2}}$$

    - **Elastic Net** combines both the L1 and L2 approaches, offering a balanced solution for enhanced model generalization. The algorithm for Elastic Net is analytically represented in the following equation.

    $$\boldsymbol{Cost} = \frac{\mathbf{1}}{\boldsymbol{n}}\sum_{\boldsymbol{i=1}}^{\boldsymbol{n}}(\boldsymbol{yi} - \widehat{\boldsymbol{yi}})^{\mathbf{2}} + \boldsymbol{\lambda}((\mathbf{1} - \boldsymbol{\alpha})\sum_{\boldsymbol{i=1}}^{\boldsymbol{m}}|\boldsymbol{wi}| + \sum_{\boldsymbol{i=1}}^{\boldsymbol{m}}\boldsymbol{wi}^{\mathbf{2}})$$

    In this regularization technique a new hyperparameter is utilized α which controls the ratio of the L1 and L2 regularization. When α approaches zero

then Elastic Net turns into L1 regularization and when α approaches one then it turns into L2 regularization.

- **Cross-Validation:** This method serves as an important instrument for evaluating the generalizability of statistical analysis results to independent datasets but also functions as a tool to avoid overfitting. The intricacies of this technique, given its critical relevance to our research, will be explored in the subsequent section.

7.2.2 Cross-Validation

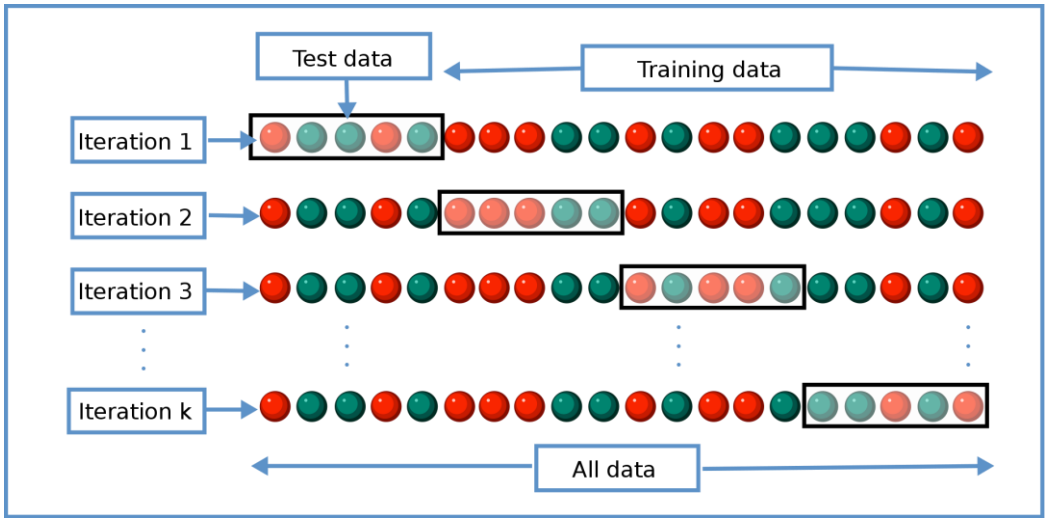


Fig 25: Diagram of K-fold cross-validation [54]

Cross-validation operates as a statistical method to estimate the proficiency of machine learning models, evaluating their capacity to generalize to independent datasets. Additionally, it plays a pivotal role in preventing overfitting, an aspect that becomes particularly essential when working with limited data. The prevalent technique of cross-validation is K-Fold Cross-Validation, which involves dividing the dataset into 'k' segments of equal size, known as 'folds.' The model undergoes training on 'k-1' of these folds and is evaluated on the remaining fold. This cycle is executed 'k' times, ensuring each fold is utilized once as the validation set, as illustrated in figure 25. The outcomes across all 'k' iterations are averaged to yield a comprehensive performance estimate. Typically, a 5-fold cross-validation is recommended for large datasets. However, for smaller datasets, such as in our study, increasing 'k' can yield a more reliable assessment. Consequently, we have selected 'k'=7 for our cross-validation process to align with the scale of our dataset [54].

7.2.3 Model Evaluation Metrics

As previously underscored, the evaluation of a model's performance is a critical element in the training process of machine learning models. Given that our task centers on binary classification, it is imperative to establish a foundational understanding related to the binary categorization of data, which will underpin all subsequent metrics.

- **True Positive Rate (TP):** The true positive rate is the count of data points correctly identified as belonging to the positive class.
- **True Negative Rate (TN):** The true negative rate is the count of data points correctly identified as belonging to the negative class.

- **False Positive Rate (FP):** The false positive rate is the count of data points incorrectly identified as belonging to the positive class when they actually belong to the negative class. In the context of this research this would mean a conversation is labelled as scam when in fact it is not.
- **False Negative Rate (FN):** The false negative rate is the count of data points incorrectly identified as belonging to the negative class when they actually belong to the positive class. This is of paramount importance to us since this would mean that a conversation is labelled as legitimate when in fact it is a scam.

The following section outlines the metrics that were employed to assess the performance of the models within the scope of this research.

- **Accuracy:** The percentage of correct predictions made by the model:

$$\boldsymbol{Accuracy(ACC)} = \frac{\boldsymbol{TP + TN}}{\boldsymbol{P + N}}$$

Where P is the total number of instances that belong to the positive class and N is the total number of instances that belong to the negative class.

- **Precision:** The ratio of correct positive predictions to the total predictions for the positive class. This metric is used in classification to measure the accuracy of the positive predictions made by a model.

$$\boldsymbol{Precision} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FP}}$$

- **Recall:** Recall, also referred to as sensitivity, is a vital metric in classification tasks, primarily focused on assessing a model's effectiveness in correctly identifying positive instances. This measure is especially critical in situations where the costs of missing a positive instance are significantly higher than those of wrongly categorizing a negative instance as positive.

$$\boldsymbol{Recall} = \frac{\boldsymbol{TP}}{\boldsymbol{TP + FN}}$$

- **F1-score:** This metric represents a balanced approach to evaluating both precision and recall, effectively accounting for both false positives and false negatives.

$$\boldsymbol{F1 - score} = \boldsymbol{2} \times \frac{\boldsymbol{Precision} \times \boldsymbol{Recall}}{\boldsymbol{Precision + Recall}}$$

7.3 Model Architecture

Having established a foundational understanding, we now shift our discussion to the machine learning algorithms chosen for this research and their respective architectural frameworks.

7.3.1 Logistic Regression

Logistic Regression, a supervised machine learning algorithm, is predominantly used for classification tasks and is particularly adept at binary classification. This predictive analysis technique models the probability of a binary outcome based on one or more predictor

variables. It employs a logistic, or sigmoid, function that processes the independent variables to yield a probability value ranging between 0 and 1. While Logistic Regression shares similarities with Linear Regression in terms of its foundational principles, their applications differ significantly. Linear Regression is typically employed for regression problems where the goal is to predict a continuous outcome. In contrast, Logistic Regression is utilized for classification problems, especially where the objective is to categorize data into distinct classes. This key distinction lies in the nature of their output: Logistic Regression provides a probability score indicating class membership, as opposed to the continuous numerical output given by Linear Regression [56][57].

As mentioned above, the core of the algorithm is the logistic function or sigmoid function. This function maps any real-valued number into a range between 0 and 1. The function can be written as follows:

$$\boldsymbol{\sigma(z) = \frac{1}{1 + e^{-z}}}$$

In this equation, $'z'$ represents the linear combination of the input features (X) and their corresponding weights (W), plus a bias term (b) and therefore we have the following:

$$\boldsymbol{z = (W \times X) + b}$$

The output given, is the probability $\boldsymbol{p(x = 1) = \sigma(z) = \frac{1}{1+ e^{-z}}}$ where $\boldsymbol{p(x = 1)}$ is the probability that the sample belongs to the positive class, whereas $\boldsymbol{1 - \sigma(z)}$ is the probability that the sample belongs to the negative class.

To effectively train the Logistic Regression algorithm, it's crucial to adjust the weights to optimal values. This is achieved by minimizing a cost function. In Logistic Regression, this cost function is known as "log loss," which is represented in the following equation.

$$\boldsymbol{Cost = J(b, W) = -\frac{1}{N}\sum_{i=1}^{N}[yi \times log(\widehat{yi}) + (1 - yi) \times log(1 - \widehat{yi})]}$$

Where 'N' is the number of samples, $'yi'$ is the actual class label, and $'\widehat{yi}'$ is the predicted probability that the sample belongs to the positive class.

Every time the cost function is calculated the weights are adjusted based on an optimization algorithm. Most of the time the optimization algorithm used is called "gradient descent" which is given in the following equation:

$$\boldsymbol{W_{ij} = W_{ij} - n \times \frac{\partial J}{\partial W_{ij}}}$$

Where $'n'$ is called the learning rate. This crucial parameter is responsible for dictating the rate at which the algorithm's weights are adjusted. The learning rate controls how rapidly the model adapts to the problem: a smaller learning rate leads to slower learning and convergence, while a larger rate accelerates the learning process but may overshoot the optimal solution.

7.3.2 Support Vector Machine

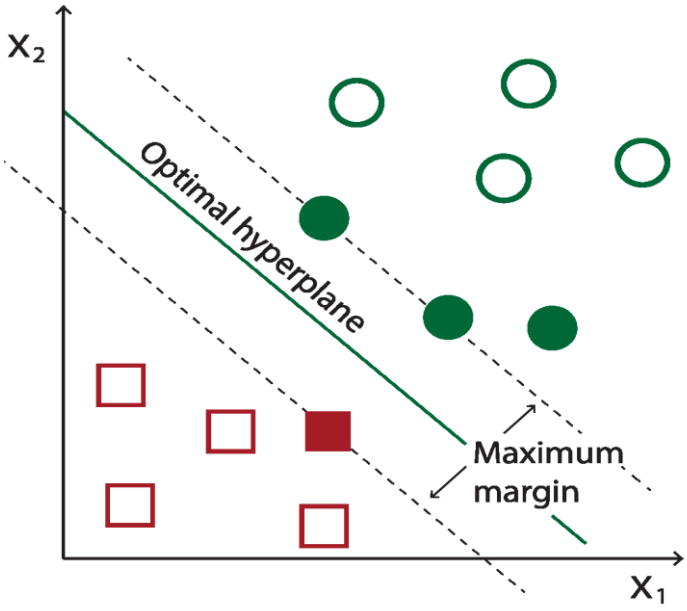


Fig 26: Example of SVM hyperplane [60]

The Support Vector Machine (SVM) stands as a powerful supervised learning algorithm, primarily applied in binary classification. Its primary function is to determine an ideal hyperplane that effectively divides two distinct classes within the feature space, ensuring the greatest possible margin, as illustrated in Figure 26. The fundamental objective is to accurately categorize new data points into one of the two classes, depending on their positioning in relation to this hyperplane [58][59].

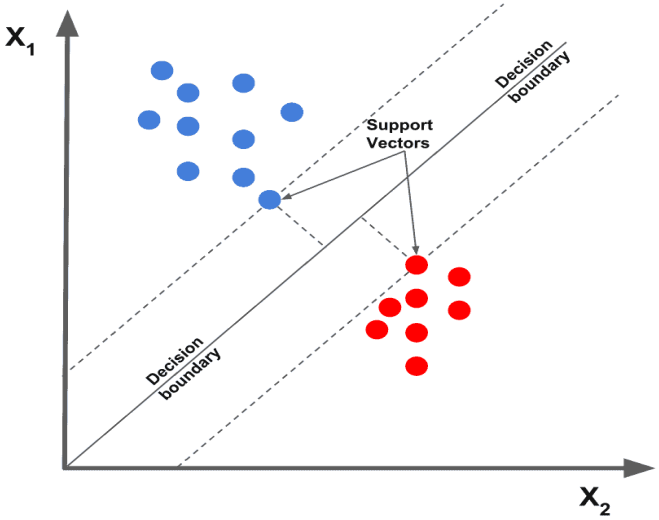


Fig 27: Support Vectors [61]

Central to the functionality of SVM are the support vectors, which are the data points nearest to the hyperplane. These points are instrumental in defining the hyperplane's position and orientation, as demonstrated in Figure 27. This hyperplane is often called the decision boundary. In two dimensions, it is a line; in three, a plane; and in higher dimensions, it is an

N-1 dimensional subspace. The equation that describes the hyperplane is generally given by [59]:

$$\vec{w} \times \vec{x} - b = 0$$

Where $\vec{w}$ is the weight vector, $\vec{x}$ is the feature vector and b is the bias. Another fundamental concept in SVM is the 'margin,' which refers to the distance between the support vectors and the hyperplane. As previously noted, SVM's objective is to maximize this margin, enhancing the classifier's ability to distinguish between different classes effectively. The margin can be represented as [59]: $\frac{2}{||\vec{w}||}$

The training of an SVM model involves solving an optimization problem to find the vector $\vec{w}$ and the bias b that maximize the margin while correctly classifying the training data. To achieve this optimization the algorithm places a constraint that all data points must be on the correct side according to their label which is represented by the following equation:

$$y_i(\vec{w} \times \vec{x_i} - b) \geq 1$$

Where $\vec{x_i}$ is a data point with label y_i. To understand this constraint better, let's see its functionality in depth. If y_i is 1 (positive class), then $\vec{w} \times \vec{x_i} - b$ must be greater than or equal to 1, meaning the data point is correctly predicted as belonging to the positive class. On the contrary if y_i is -1 then $\vec{w} \times \vec{x_i} - b$ must be less than or equal to -1, which after multiplying by $y_i = -1$, becomes greater than or equal to 1, ensuring correct classification.

Once the model is trained via the above procedure, classification of a new data point $\vec{x}$ is done using the sign of $\vec{w} \times \vec{x} - b$. If the sign is greater than zero then the data point is classified into one class, else into the other class [59].

7.3.3 Random Forest

Before we delve into the Random Forest algorithm, chosen for its robust performance in our task, it is essential to first comprehend the workings of a decision tree, which serves as the foundational building block of the Random Forest.

7.3.3.1 Decision Trees

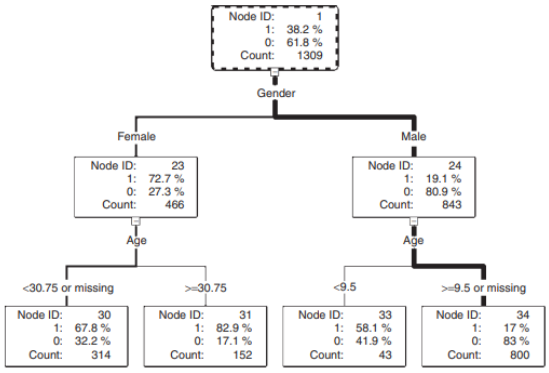


Fig 28: A decision tree illustrating analysis of survival in Titanic Sinking [62]

The decision tree stands as a formidable model in the realm of classification, functioning as a hierarchical decision support tool. This model is characterized by a tree-like structure of

decisions, resembling a flowchart. At each internal node of the tree, a decision based on a feature (or attribute) is made. Branches extending from these nodes denote the possible outcomes of these decisions, leading to subsequent nodes or terminating at leaf nodes, which represent the final outcomes or classification labels. The root node sits at the apex of this structure, initiating the partitioning of the dataset. This partitioning unfolds in a recursive manner, a process known as recursive partitioning, where the dataset is successively split according to the most discriminative features until the termination criteria are met, and the leaf nodes are reached [63].

The construction of a decision tree is a sequential process. It begins with the division of the root node, creating branches that lead to subsequent nodes or leaves. The nodes within a decision tree group together observations that are similar to each other. This similarity is determined based on the chosen features within a node. However, these groups are distinct and different from the groups in other nodes at the same level of the tree, ensuring that each branch represents a unique partition of the data based on specific criteria. The branching decisions are derived from an exhaustive examination of the dataset to identify the most informative data fields, referred to as 'inputs', which can effectively partition and capture the variability present in the target variable as represented in the root node. Upon selection of an appropriate input, the tree grows by forming new descendant nodes [62].

A decision tree uses a collection of algorithms to decide how to split the data at each node. In this study, our attention is primarily on the CART (Classification and Regression Trees) algorithm, which is the underlying mechanism for both the Random Forest and Gradient Boosted Trees algorithms. CART employs the Gini Impurity metric to decide on the optimal way to split the data at each node. Gini Impurity quantifies the likelihood of incorrect classification of an element in the dataset if it were randomly labeled, based on the current composition of the node. Essentially, it measures the 'purity' of a node in the tree, with lower values indicating more homogeneous nodes. During the decision-making process, the algorithm selects the feature for splitting those results in the largest **decrease** in Gini Impurity. This metric is calculated using the following formula [64]:

$$\boldsymbol{Gini(D) = 1 - \sum_{i=1}^{n} {p_i}^2}$$

Where '**D**' is the dataset or subset of the dataset and $\boldsymbol{p_i}$ represents the proportion (or frequency) of class 'i' within the set. The sum term of the equation is computed over all unique classes (n) present in the subset. Therefore, for a binary classification task the equation simplifies to:

$$\boldsymbol{Gini(D) = 1 - ({p_1}^2 + {p_2}^2)}$$

The Random Forest algorithm operates as an ensemble of Decision Trees, where each tree independently contributes a prediction. The algorithm aggregates these individual predictions, employing a majority voting system for classification tasks, to arrive at a final decision. This ensemble approach effectively mitigates risks such as overfitting, which might be prevalent in a solitary Decision Tree [65].

To create the individual decision trees, the Random Forest algorithm relies upon the method of **bagging**. The bagging process involves creating numerous bootstrap samples from the original training dataset. For each bootstrap sample—randomly drawn with replacement and

equal in size to the original dataset—a Decision Tree is trained, leading to slight variations in each tree due to the different subsets of data.

7.3.4 Gradient Boosted Trees

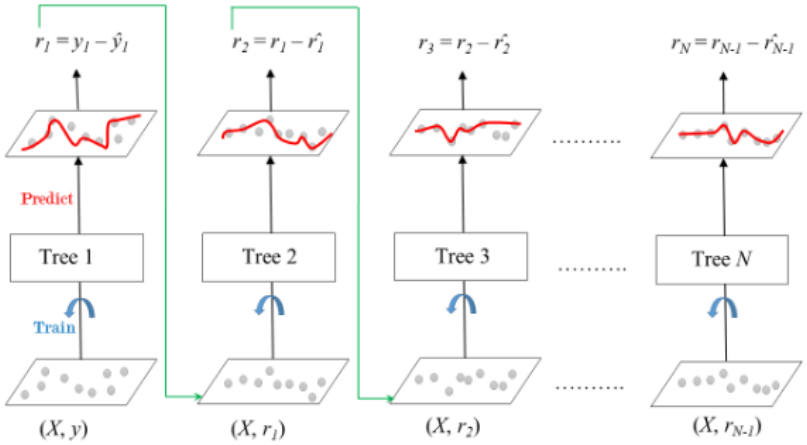


Fig 29: Sequential Construction of Trees in Gradient Boosting [68]

Gradient Boosted Trees (GBTs) are an ensemble learning technique that builds upon the concept of decision trees differing from those on how the trees are constructed. In Random Forests trees grow in parallel, whereas GBTs construct trees sequentially as depicted in Figure 29, where each new tree incrementally improves upon the previous trees by correcting the errors made by the predecessor. The "gradient boosting" part comes from the use of gradient descent to minimize the loss when adding new models. Each tree is fitted on the residual errors of the whole ensemble to date, effectively taking the gradient step in the space of possible predictions to reduce the loss. This compounding of small, weak decision trees continues until no significant improvements can be made or a specified number of trees is reached. The result is a potent predictive model that combines the strengths of numerous simple models into a single ensemble. The process of gradient boosting trees can be described by the following equations [67][68]:

The loss function, which is the difference between the actual and predicted values:

$$\boldsymbol{L(y, F(x)) = \sum_{i=1}^{N} L(y_i, F(x_i))}$$

The goal of the gradient boosted algorithm is to minimize this loss. The process starts with the initial model $\boldsymbol{F_0(x)}$. For each data point $(\boldsymbol{x_i, y_i})$ the residual is calculated which represents the error of the current model. This residual is calculated as follows:

$$\boldsymbol{r_{ti} = -\frac{\partial L(y_i, F(x_i))}{\partial F(x_i)}}$$

Then, a new decision tree $\boldsymbol{h_t(x)}$ is trained, not to predict the actual target values, but to predict these residuals. In essence the new tree is learning how to correct the mistakes of the previous tree and hence the updated model is given as follows:

$$F_t(x) = F_{t-1}(x) + n \times h_t(x)$$

Where n is the learning rate of the algorithm. This process is repeated either for a specific preset number of iterations or until the improvement becomes negligible.

7.3.5 Neural Networks

Neural networks represent a cornerstone of a specific area of modern machine learning called deep learning, drawing inspiration from the biological neural networks that constitute animal brains. At their core, neural networks are a series of algorithms designed to recognize underlying relationships in a set of data through a process that mimics the way the human brain operates.

A Neural Network operates through a network of interconnected neurons, collectively working to solve complex problems. Such a structure can be seen in Figure 30. In the following sections the workings and structure of Neural Networks will be thoroughly discussed.

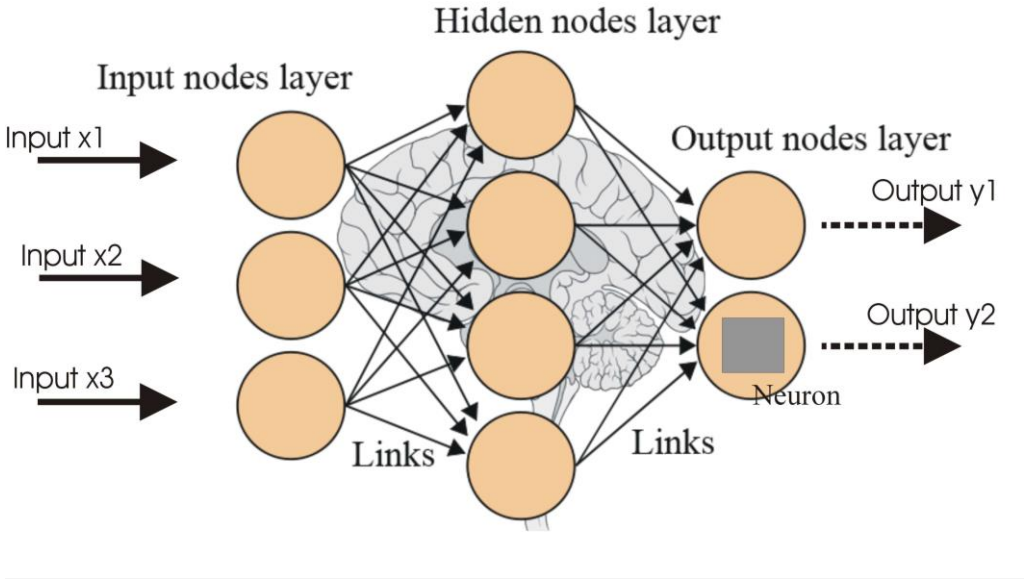


Fig 30: Depiction of the Structure of a Neural Network [69]

A Neural Network broadly consists of 3 consecutive layers of neurons. The first layer is called **input layer,** the layers following the input layer can be multiple in number and are called **hidden layers** and finally the last layer is called the **output layer**.

- **Input Layer**: The input layer serves as the initial point of data entry for the problem being addressed. It consists of input neurons, such as input x1, input x2, input x3 as illustrated in Figure 30. These neurons are responsible for receiving the input information but do not perform any computational processing. Instead, their role is to simply relay this information to the subsequent hidden layers.

- **Hidden Layers:** Hidden layers are composed of neurons that receive information from either the input layer or preceding hidden layers. Unlike neurons in the input layer, neurons in hidden layers actively perform computations. These computations depend on the information received from previous layers, combined with specific weights and

biases. Each neuron in these layers applies an activation function to the weighted sum of its inputs, introducing non-linearities that enable the network to learn complex patterns and relationships in the data. The output of these neurons is then passed on to the next layer in the network, whether it's another hidden layer or the output layer.

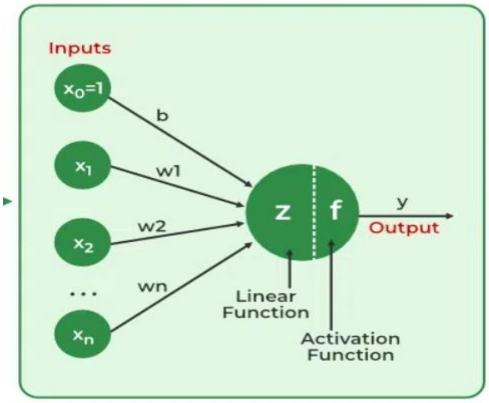


Fig 31: Depiction of Neuron's Output Calculation [70]

Before exploring the neuron's output calculation, it's essential to understand the role of the **'Activation Function'**. This function takes the neuron's output and transforms it into a format suitable for the problem at hand, often introducing non-linearity into the model. Some of the most prominent activation functions used in neural networks include the following [71]:

- **Sigmoid**: The **Sigmoid** activation function is designed to constrain the output of a neuron to a value between 0 and 1. It is particularly useful in the output layer of binary classification tasks, where the outcomes are binary: 0 typically represents the negative class, and 1 represents the positive class. When the Sigmoid function outputs a prediction greater than 0.5, the model interprets the input data as belonging to the positive class; conversely, a prediction less than 0.5 is interpreted as belonging to the negative class. The equation of the sigmoid function is as follows:

$$\boldsymbol{f(x) = \frac{1}{1 + e^{-x}}}$$

- **Tanh:** The **tanh**, or hyperbolic tangent, activation function serves as a scaled version of the sigmoid, and it is commonly employed in the hidden layers of a neural network. Its output ranges between -1 and 1, effectively centering the data and bringing the mean output of the neurons closer to 0, which can improve learning in deeper layers. The mathematical representation of the tanh function can be written as follows:

$$\boldsymbol{f(x) = tanh(x) = \frac{2}{1 + e^{-2x}} - 1}$$

- **ReLU:** The **ReLU**, or Rectified Linear Unit, activation function is a widely used choice for the hidden layers in neural networks. If a neuron's output is positive, the ReLU function allows that value to pass without change; if the

output is negative, ReLU sets it to 0. This function is computationally efficient compared to tanh and sigmoid functions, making it a preferred option in scenarios that demand rapid predictions. The mathematical representation of this function is given as follows:

$$f(x) = \begin{cases} 0, & x < 0 \\ x, & x \geq 0 \end{cases}$$

- **SoftMax:** The **SoftMax** function, an extension of the sigmoid, is mostly used for multiclass classification tasks within neural networks. Typically existing in the output layer when the network is designed to handle multiple classes, the SoftMax function normalizes the output into a probability distribution. It ensures that the output values fall between 0 and 1, and that the sum of all the probabilities equals 1. The function achieves this by exponentiating each output, then dividing each by the sum of all exponentiated outputs. This process allows the network to represent the probability that the input corresponds to each class. The formula for the SoftMax function can be written as follows [72]:

$$f(z_i) = \frac{e^{z_i}}{\sum_{j=1}^{K} e^{z_j}}$$

Where z_i is the input to the SoftMax function for the $i-th$ class and K is the total number of classes in the multiclass classification problem.

The comprehensive output of a neuron, as depicted in Figure 31, is calculated through a mathematical formula that uses the weighted sum of its inputs followed by the application of an activation function. This calculation can be mathematically expressed as follows:

$$y = f(z) = f(\sum_{i=1}^{n} w_i \times x_i + b)$$

Where b is the bias term associated with the neuron, w_i represents the weight assigned to the $i-th$ input, x_i is the value of the $i-th$ input and n is the total number of inputs to the neuron. Finally, $f(z)$ is denotes the activation function which was discussed above. This process yields the neuron's complete output which is then passed on to the following layers. This process of information passing through the network from one layer to another, with the mechanism outlined above, is called **Forward Propagation.**

- **Dropout Layer:** The dropout mechanism serves to randomly deactivate a subset of neurons in the layer during training, simulating sparse activation and introducing noise into the training process. This approach forces the remaining active nodes in the subsequent layer to adapt by either assuming more or less responsibility for the given inputs. Using dropout layers is an effective strategy for decreasing overfitting, as it prevents the model from becoming overly reliant on any specific set of neurons [73].

- **Output Layer:** This is the final layer of the neural network and is responsible for producing the end result that corresponds to the format required for the specific task the network

is designed for. For the problem at hand, the output will be a single value signifying the probability of the provided conversation being 'Vishing' or 'Normal'.

Having discussed Forward Propagation, we must now turn to how the Neural Networks learn. In the learning process of a Neural Network, the primary objective is to minimize the output error. To achieve this, a loss function is employed that quantifies the difference between the network's predictions and the actual labels during each training epoch. For binary classification tasks, such as the one undertaken in this study, the most commonly used loss function is 'Binary Cross-Entropy'. As outlined in Chapter 7.2.1, coupling this loss function with a regularization term is crucial to prevent overfitting. The combined loss function, incorporating both Binary Cross-Entropy and regularization, can be mathematically represented as follows:

$$\boldsymbol{L = -\frac{1}{N}\sum_{i=1}^{N}[y_i \log(p_i) + (1 - y_i)\log(1 - p_i)] + \lambda_1 \sum_{j=1}^{m}|w_j| + \lambda_2 \sum_{j=1}^{m} {w_j}^2}$$

Where $\boldsymbol{y_i}$ is the actual label of the $\boldsymbol{i-th}$ observation (0 or 1) and $\boldsymbol{p_i}$ is the predicted probability of the $\boldsymbol{i-th}$ observation. The term $\boldsymbol{\lambda_1 \sum_{j=1}^{m}|w_j|}$ **,** also known as Ridge Regularization, is the first part of the regularization term and $\boldsymbol{\lambda_2 \sum_{j=1}^{m} {w_j}^2}$**,** also known as Lasso Regularization, is the second part of the regularization term. As covered in chapter 7.2.1 those two terms together form the Elastic Net regularization with $\boldsymbol{\lambda_1}$ and $\boldsymbol{\lambda_2}$ being the regularization parameters that control the strength of each regularization term.

The loss function plays a primary role in helping the neural network update its weights and biases, thereby enabling the network to learn from its errors. This adjustment of weights and biases is accomplished using the gradient descent algorithm. Gradient descent aims to minimize the loss function by continuously modifying the neuron weights in the direction that most reduces the loss. This iterative process continues until the algorithm converges to a minimum, a point where further reductions in the loss function are not possible. The mathematical representation of the gradient descent algorithm can be written as follows [74]:

$$\boldsymbol{{W_{jk}}^{l} = {W_{jk}}^{l} - \gamma \frac{\partial L}{\partial {W_{jk}}^{l}}}$$

Where $\boldsymbol{L}$ is the loss function, $\boldsymbol{{W_{jk}}^{l}}$ is the weight for the $\boldsymbol{j}$ neuron that belongs to $\boldsymbol{l}$ layer with $\boldsymbol{k}$ input. The term $\boldsymbol{\gamma}$ is the learning rate and $\boldsymbol{\frac{\partial L}{\partial {W_{jk}}^{l}}}$ is the gradient for that neuron.

The gradient descent algorithm serves as an optimization technique for minimizing the loss function in a neural network. However, it alone does not facilitate the communication of how each weight should be adjusted throughout the network. A common way to achieve this communication is via the process of **Back propagation** [75]. Back propagation is employed to calculate the gradients of the loss function with respect to each weight in the network and communicate these gradients back through the network's layers. This process enables the gradient descent algorithm to accurately adjust the weights, and by doing so optimize the network's performance.

To better understand the complete process of Forward propagation and Back propagation let us look at a simple example.

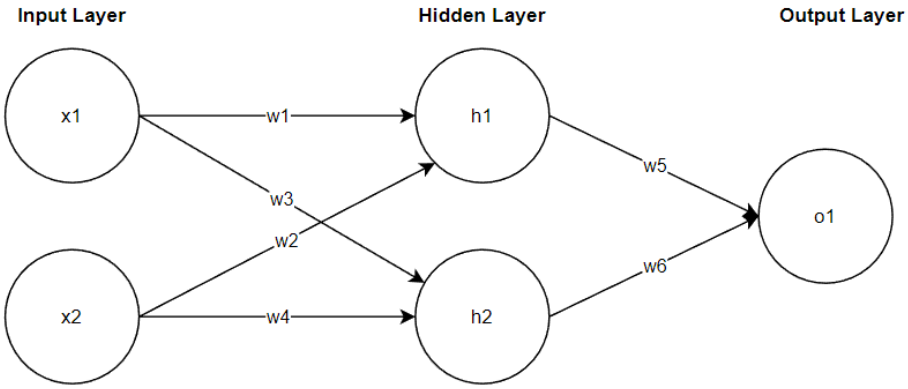


Fig 32: Simple Example of Neural network Structure

Assume the Neural Network structure depicted in Figure 32. This example Neural Network consists of two neurons in the input layer (x1, x2), two neurons in its single hidden layer (h1, h2) and a singular output neuron in the output layer (o1). Suppose that the activation function used throughout the network is the sigmoid activation function (σ).

Starting with the Forward Pass or Forward Propagation the output values of the neurons can be calculated as follows:

$$\boldsymbol{h_1 = \sigma(w_1 * x_1 + w_2 * x_2 + b_1)}$$

$$\boldsymbol{h_2 = \sigma(w_3 * x_1 + w_4 * x_2 + b_1)}$$

$$\boldsymbol{o_1 = \sigma(w_5 * h_1 + w_6 * h_2 + b_2)}$$

The output of the network is determined by the neuron $\boldsymbol{o_1}$ in the output layer. This marks the end of the forward pass. Subsequently, the weights of the network need to be updated based on the calculated loss. In this example, we assume that the loss is calculated using the Mean Squared Error (MSE) equation, which is given by: $\boldsymbol{L = MSE = \frac{1}{2}(y - o_1)^2}$ where $\boldsymbol{y}$ is the target output (label).

To achieve this the back propagation process implements the following steps, which can be represented in mathematical form as written below:

Calculate Gradients for Output Layer:

- $\boldsymbol{\frac{\partial L}{\partial o_1} = (o_1 - y) \times \sigma'(o_1)}$
- $\boldsymbol{\frac{\partial L}{\partial w_5} = \frac{\partial L}{\partial o_1} \times h_1}$
- $\boldsymbol{\frac{\partial L}{\partial w_6} = \frac{\partial L}{\partial o_1} \times h_2}$

Propagate Error to Hidden Layer:

- $\boldsymbol{\frac{\partial L}{\partial h_1} = \frac{\partial L}{\partial o_1} \times w_5 \times \sigma'(h_1)}$
- $\boldsymbol{\frac{\partial L}{\partial h_2} = \frac{\partial L}{\partial o_1} \times w_6 \times \sigma'(h_2)}$

Calculate Gradients for Hidden Layer:

- $\frac{\partial L}{\partial w_1} = \frac{\partial L}{\partial h_1} \times x_1$
- $\frac{\partial L}{\partial w_2} = \frac{\partial L}{\partial h_1} \times x_2$
- $\frac{\partial L}{\partial w_3} = \frac{\partial L}{\partial h_2} \times x_1$
- $\frac{\partial L}{\partial w_4} = \frac{\partial L}{\partial h_2} \times x_2$

Having calculated all the gradients, the gradient descent algorithm can now update the weights as was shown previously using this equation:

$$W_{jk}{}^{l} = W_{jk}{}^{l} - \gamma \frac{\partial L}{\partial W_{jk}{}^{l}}$$

7.3.5.1 LSTM Neural Networks

Having established the foundational principles of how Neural Networks function, we now transition to exploring a specialized Neural Network architecture known as the Long Short-Term Memory (LSTM) Network. This network is engineered to identify the relevance of information over extended sequences, essentially learning when to retain and when to discard certain pieces of information. Such capability enables the LSTM to identify and preserve information that may be useful later in a sequence while discarding what becomes irrelevant. A prime application of this is in natural language processing tasks, where the network adeptly grasps grammatical dependencies and other contextual nuances [76].

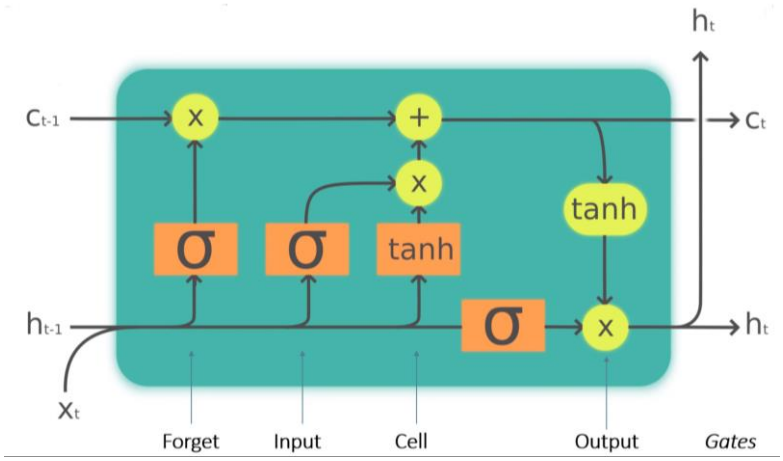


Fig 33: LSTM Cell Structure [78]

To achieve this functionality, the LSTM network makes use of LSTM cells that have the structure depicted in Figure 33. The key components and functions of the LSTM cell are the following [77]:

- **Cell State:** The cell state acts like the 'memory' of the LSTM cell. It carries information throughout the sequence of data. It has the ability to add or remove information using the gates.

- **Gates:** The gates are a way to optionally let information through. They are composed of a sigmoid net layer and a pointwise multiplication operation. There are three types of gates in an LSTM cell:

    - **Forget Gate:** This gate is responsible for discarding information deemed not useful. The gate takes two inputs, $\boldsymbol{Xt}$ (input at that particular time) and $\boldsymbol{h_{t-1}}$ (previous cell state). Those inputs are multiplied with weight matrices and are added with a bias. The result is passed through an activation function which outputs a binary result. If the output of that function is 0 then the piece of information is forgotten; in contrast if the output is 1 then the information is retained.

    - **Input Gate:** The input gate is responsible for integrating valuable information into the cell state. Initially, a sigmoid function regulates the incoming information, functioning analogously to the forget gate by filtering which values should be retained based on the inputs $\boldsymbol{h_{t-1}}$ and $\boldsymbol{Xt}$**.** Subsequently, a new vector is created with output values ranging from -1 to +1 using the tanh activation function. Finally, the product of the vector and the regulated values yields the information deemed useful for retention in the cell state.

    - **Output Gate:** The output gate's function is to select valuable information from the current cell state to form the output of the cell. The process begins with the application of the tanh activation function to the cell state. This vector is modulated by a sigmoid function which filters the information based on the current inputs $\boldsymbol{h_{t-1}}$ and $\boldsymbol{Xt}$**.** The final step involves the multiplication of this vector with the sigmoid output resulting in the final output of the cell which also serves as the input to the subsequent cell.

- **Hidden State:** The hidden state in an LSTM cell, often denoted as $\boldsymbol{h_t}$ is a dynamic component that carries information from one cell to the next time step within a sequence. As outlined above this hidden state along with the input $\boldsymbol{Xt}$ regulates the gates and is also used to generate the output of the cell.

A complete structure example of an LSTM Neural Network can be seen in Figure 34. The link between LSTM Cell is the hidden state being passed from one cell to the next as mentioned above.

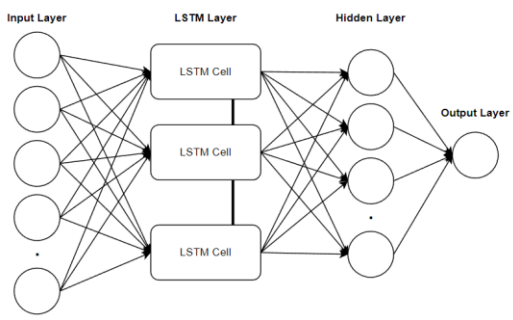


Fig 34: LSTM Neural Network Structure Example

7.4 Model Training - Tuning – Results

This section of the dissertation delves into the processes of model training, tuning, and the results obtained from validating these models. Each model underwent training using an identical dataset and was subsequently validated and tested using consistent sets to ensure comparability of results. The original dataset was divided as follows: 80% was allocated for training, which during the training phase was further split into a validation set constituting 20% of the training subset, with the remaining 80% used for the actual training. The final 20% of the original dataset was reserved for the test set. This structured approach to data allocation facilitates a thorough examination of each model's performance and generalization capabilities. In the following subsections, there will be a comprehensive analysis of how each model was trained and tuned along with its training results.

Given the nature of the task of this study, which emphasizes the importance of accurately identifying 'Vishing' conversations, all models were trained focusing on optimizing the 'Recall' metric. This approach helps the system minimize the risk of incorrectly classifying 'Vishing' conversations as 'Normal'.

7.4.1 Logistic Regression

The training of the Logistic Regression model was conducted within the PySpark framework. For this process, we employed the Logistic Regression library, along with the ParamGridBuilder and CrossValidator components, which were used in the tuning process for the model.

As outlined in Chapter 3.3.2.2, the ParamGridBuilder component, when used in conjunction with an evaluation tool, serves to hyperparameter tune the model through the exploration of a predefined grid of parameters. In essence, this process involves supplying a range of possible parameter values and testing each combination to identify the optimal configuration. The outcome of this procedure is the selection of the best-performing model variant based on the specified hyperparameter combinations.

For the training of the Logistic Regression model, three hyperparameters were used in the grid:

- **maxIter:** This parameter controls how many training epochs the model will be put through.
- **regParam:** This parameter controls the strength of the regularization applied to the model. A higher value means more regularization whereas a lower one means less regularization.
- **elasticNetParam:** This parameter handles the balance between L1 and L2 regularization, ranging from 0 to 1. A value of 0 indicates pure L2 regularization, whereas a value of 1 denotes exclusive L1 regularization, and values in between imply a mix of both.

The grid of values for these hyperparameters is the following:

LR	Range Of Values			
maxIter	10	50	100	
regParam	0.01	0.1	1	
elasticNetParam	0	0.4	0.6	1

Fig 35: Hyperparameter range of values for Logistic Regression

During the training phase, cross-validation was employed, utilizing a seven-fold approach. Following the training and cross-validation, the optimal model was determined based on its hyperparameters. The best-performing model had a maxIter of 50, a regParam of 0.01, and an elasticNetParam set to 0. This combination of hyperparameters represented the most effective configuration for the task.

The results of the training process for the Logistic Regression model can be viewed in the following table.

	Accuracy	Precision	Recall	F1Score
LR	94.57%	94.60%	94.56%	94.57%

Fig 36: Logistic Regression Training Results

The metrics indicate strong performance by the model on the test set, which suggests effective learning of data patterns and successful generalization without overfitting to the training data. High accuracy points to good overall performance, while the high precision, recall, and F1 score metrics demonstrate the model's capability to accurately identify 'Vishing' conversations while balancing between detecting as many true positives ('Vishing') instances as possible (high recall) and maintaining a low rate of false positives (high precision), as reflected in the balanced F1 score.

7.4.2 Random Forest

Similarly, to the Logistic Regression model, the Random Forest model was trained within the PySpark framework and with the same components. For the training of the model three hyperparameters were used in the grid.

- **numTrees:** This parameter determines the number of trees in the forest. Increasing the number of trees typically results in higher variance and adds complexity to the model.
- **maxDepth:** This parameter specifies the maximum depth of each tree in the forest. A deeper tree introduces more complexity but also increases the risk of overfitting to the training data. Conversely, a shallower tree might underfit, failing to capture patterns.
- **maxBins:** This parameter sets the maximum number of bins used for splitting continuous features and choosing split points for categorical features. A larger maxBins value enables the algorithm to consider a greater number of split points, potentially leading to more accurate models, particularly for continuous features.

The grid of values for these hyperparameters is the following:

RF	Range Of Values				
numTrees	10	20	30	50	100
maxDepth	2	5	10	15	
maxBins	32	64	128		

Fig 37: Hyperparameter range of values for Random Forest

Cross-validation with seven folds was similarly implemented in the training of this model. The process yielded an optimal set of hyperparameters for the Random Forest model. The best configuration identified from the training consisted of a numTrees set to 50, a maxDepth of 15, and a maxBins set to 32.

The results of the training process for the Random Forest model can be viewed in the following table.

	Accuracy	Precision	Recall	F1Score
RF	94.56%	95.00%	94.80%	94.54%

Fig 38: Random Forest Training Results

Much like the Logistic Regression model, the Random Forest model's test set results are very encouraging. The model shows high performance across all metrics, indicating a strong capability to identify 'Vishing' conversations. Moreover, the balanced F1 Score indicates the model's proficiency in maintaining a balance between precision and recall, ensuring that it neither misses too many actual 'Vishing' instances nor misclassifies too many 'Normal' conversations as 'Vishing'.

7.4.3 Support Vector Machines

For the training of the Support Vector Machine model, the following two hyperparameters were used in the grid.

- **regParam:** This is similar to the Logistic Regression's regParam, and it controls the strength of the regularization used in training.
- **maxIter:** This parameter controls how many training epochs the model will be put through.

The grid of values for these hyperparameters is the following:

SVM	Range Of Values			
maxIter	10	20	50	100
regParam	0.001	0.01	0.1	

Fig 39: Hyperparameter range of values for Support Vector Machines

In this model training, cross validation with seven folds was used as well. The training process resulted in the following best performing parameter configuration. The optimal solution consisted of a maxIter of 10, and regParam equal to 0.1.

The results of the training process for the Support Vector Machines model can be viewed in the following table.

	Accuracy	Precision	Recall	F1Score
SVM	93.40%	93.50%	93.88%	93.47%

Fig 40: Support Vector Machines Training Results

The results of the Support Vector Machines algorithm are slightly worse than those of the previous two algorithms. However, the metrics still indicate very good performance from the model all around.

7.4.4 Gradient Boosted Trees

For the training of the Gradient Boosted Trees model, the following three hyperparameters were used in the grid.

- **maxIter**: This parameter controls the maximum number of trees to be built.
- **maxDepth**: This parameter controls the maximum depth of each tree, and therefore how complex each tree can be.
- **stepSize**: This parameter regulates the contribution of each tree to the final model. A smaller step size means that each tree has a smaller impact, necessitating a larger number of trees for convergence but typically resulting in a better model.

The grid of values for these hyperparameters is the following:

GBT	Range Of Values		
maxIter	20	50	70
maxDepth	2	4	6
stepSize	0.01	0.1	0.3

Fig 41: Hyperparameter Range of Values for Gradient Boosted Trees

Seven fold cross validation was also implemented in the training of this model. The optimal parameter configuration that resulted from the training process had maxIter equal to 70, maxDepth equal to 2 and stepSize equal to 0.3.

The results of the training process for the Support Vector Machines model can be viewed in the following table.

	Accuracy	Precision	Recall	F1Score
GBT	85.86%	86.00%	86.14%	85.86%

Fig 42: Gradient Boosted Trees Training Results

The Gradient Boosted Trees model's performance on the test set stands in contrast to the previously discussed algorithms. With a metric performance drop of approximately 9% relative to the better-performing models, it appears that the Gradient Boosted Trees model shows challenges in generalizing effectively, as evidenced by its poor results on the test data.

7.4.5 Neural Network

The training of the Neural Network model in this study deviated from the approach used for the previous models, as it was not executed within the PySpark framework. Instead, the Keras library, a high-level neural networks API, was employed for model construction.

The architecture of the final model consists of four **Dense** layers: the first layer with 16 neurons, the second with 32 neurons, the third again with 16 neurons, and a single-neuron output layer. To mitigate overfitting, dropout layers were placed between these layers: a 50% dropout between the first and second layers, 30% between the second and third layers, and 10% between the third and fourth layers. These dropout rates were established through manual hyperparameter tuning, based on the model's performance on the validation and test sets. Finally, the ‘reLU’ activation function was used for the hidden layers and the ‘sigmoid’ activation function was utilized in the output layer.

Further manual tuning led to the inclusion of an elasticNetParam set at a value of 0.00001 and a learning rate of 0.0004. This specific combination of parameters and architecture yielded the best performance for the model. The 'Adam' optimizer was chosen as the optimization algorithm and for the loss function, 'Binary Cross Entropy' was selected, aligning with the binary nature of the classification task.

Due to the nature of the training dataset, in the training process of this model, cross validation was also utilized. It consisted of 3 folds in contrast to the 7 fold that were used in the training of the previous algorithms.

To further prevent overfitting, the EarlyStopping technique was utilized. This method stops the training when the model's performance on the validation set begins to worsen. At that point, the training is halted, and the weights from the most successful epoch are saved. In our case, the training was stopped at epoch 86.

The results of the training process can be seen in the following table.

	Accuracy	Precision	Recall	F1Score	Validation Loss
NN	93.47%	92.59%	96.15%	94.33%	18%

Fig 43: Neural Network Training Numerical Results

The improvement of each metric throughout the training epochs can be seen in the following figure.

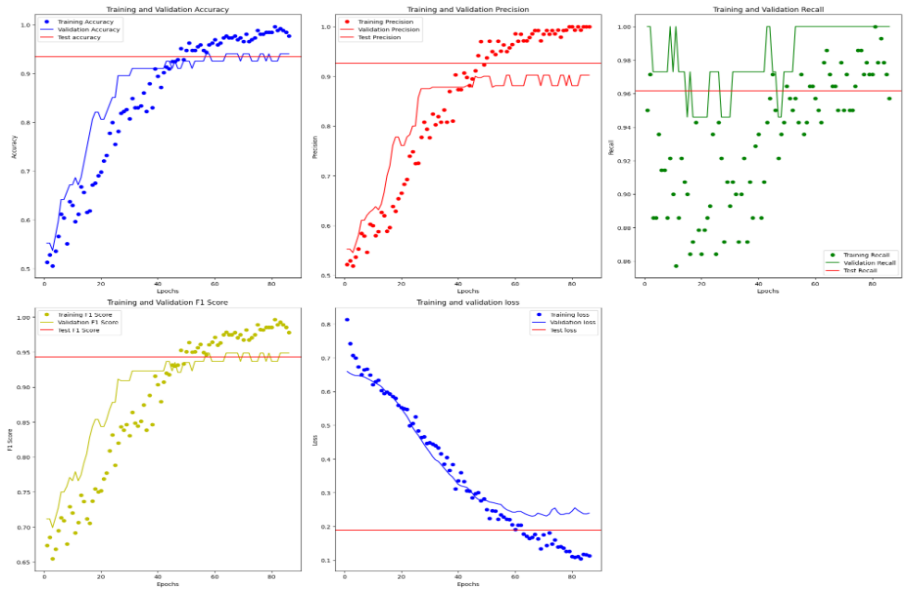


Fig 44: Neural Network Metric Improvement Graphs

The Neural Network model demonstrates good performance across all metrics, with a particularly high recall rate, which is significantly higher than the precision metric. This disparity suggests a tendency of the model to favor identifying 'Vishing' conversations. The validation loss at 18% points to a degree of overfitting, which is confirmed by the training charts where validation performance trails behind the training performance. In conclusion, despite the Neural Network model's strong test data results, the elevated validation loss and discrepancies between training and validation outcomes during the training phase suggest that there is a slight overfitting issue.

7.4.6 LSTM Neural Network

Like the Neural Network model, the LSTM network was also developed using the Keras library. The finalized structure of the LSTM model comprised four layers: the first and second layers were LSTM layers with 16 and 32 neurons, respectively, followed by a third Dense layer with 16 neurons, and concluding with a single-neuron output layer. To minimize overfitting, dropout layers were strategically incorporated: a 50% dropout was applied between the first and second layers, 20% between the second and third layers, and 10% between the third and fourth layers.

The training approach for the LSTM Network mirrored that of the Neural Network, including similar practices of cross-validation and the use of EarlyStopping. The 'Adam' optimizer was once again the preferred choice, set at a learning rate of 0.00015. An elasticNetParam was incorporated, set at a value of 0.00003. The activation function between the first three layers was 'ReLU', while the 'Sigmoid' function was used in the output layer. The training was concluded at epoch 108, based on the EarlyStopping criteria.

The results of the training process can be seen in the following table.

	Accuracy	Precision	Recall	F1Score	Validation Loss
LSTM	92.39%	90.56%	95.99%	93.20%	27%

Fig 45: LSTM Network Training Results

The improvement of each metric throughout the training epochs can be seen in the following figure.

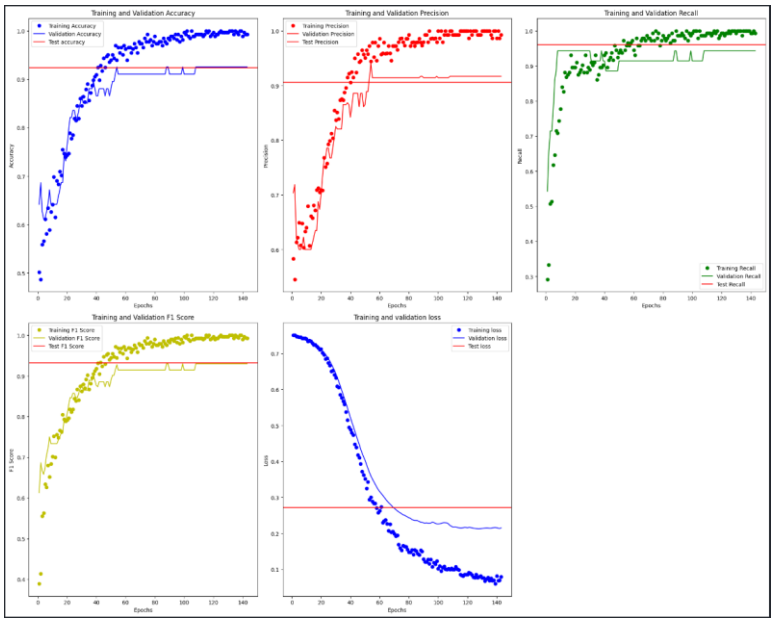


Fig 46: LSTM Network Metric Improvement Graphs

Similar to the Neural Network, the LSTM Network also presents high performance metrics on the test set, particularly with regard to the Recall metric. This elevated recall paired with a lower precision indicates a bias in the model towards classifying conversations as 'Vishing.' A substantial validation loss at 27% coupled with the noticeable gap between training and validation performance in the training graphs indicates that the model may have overfitted.

7.4.7 Overall Results

	Accuracy	Precision	Recall	F1Score	Validation Loss
LR	94.57%	94.60%	94.56%	94.57%	
RF	94.56%	95.00%	94.80%	94.54%	
SVM	93.40%	93.50%	93.88%	93.47%	
GBT	85.86%	86.00%	86.14%	85.86%	
NN	93.47%	92.59%	96.15%	94.33%	18%
LSTM	92.39%	90.56%	95.99%	93.20%	27%

Fig 47: All Model Results

In summary, the Logistic Regression and Random Forest models demonstrate the best performance, with both achieving high scores across accuracy, precision, recall, and F1 metrics on the test set. These results suggest that they have learned to generalize well without overfitting to the training data. The Support Vector Machines also show good performance, however slightly lagging behind the leading models.

However, the Gradient Boosted Trees model reveals some generalization challenges, as indicated by a noticeable performance drop compared to the other models, suggesting potential overfitting.

The Neural Network and LSTM models, while showing good performance in detecting 'Vishing' conversations as reflected by their high recall rates, exhibit signs of overfitting. This is evident from their substantial validation losses and the visible gaps between training and validation performance. The high recall yet lower precision suggests an inclination towards classifying conversations as 'Vishing', potentially at the expense of incorrectly classifying normal conversations.

Overall, while all models show potential, the Logistic Regression and Random Forest stand out for their high performance metrics on the test set, while the rest of the models lack behind each displaying their individual issues.

8. Overall System Functionality

9. Experimental Results

Having thoroughly explored the generation and preprocessing of data, the training methodologies of the models, and their performance during training and validation in Chapter 7.4, as well as the comprehensive functioning of the complete system, our focus now shifts to the real-world efficacy of the models. This chapter is dedicated to evaluating the system's performance in live scenarios, specifically analyzing its responsiveness and accuracy across four distinct types of indicative conversations. The ensuing subsections will dissect and scrutinize the real-time outcomes of each model when confronted with these varied conversational contexts. Our objective is to critically assess the system's operational competence and to draw insights into its practical applicability, reliability, and robustness in dynamic, real-life settings that mimic actual user interactions.

The evaluation of our system's performance is structured around four conversation types as previously mentioned. These conversations are selected to gauge the adaptability and generalization capabilities of the algorithms in use. First, we analyze the system's response to a 'Vishing' conversation, a scenario that mirrors the context and structure the models have been trained on. Second, we assess the system's reaction to an unfamiliar 'Vishing' conversation—one with content and format not previously encountered during training. Moving forward, the third conversation type pertains to 'Non-Vishing' interactions that align closely with the models' training, allowing us to evaluate the system's accuracy in recognizing legitimate communications. The fourth and final conversation type presents a 'Non-Vishing' scenario that is new to the models, testing their ability to comprehend and correctly classify unseen and new formats. Through the examination of these four distinct conversational contexts, we aim to derive an understanding of the models' performance.

In this research, the Support Vector Machines (SVM) algorithm was determined to be unsuitable, because it does not provide probability estimates alongside its output predictions. As highlighted in Chapter 8, the ability to quantify prediction confidence in the form of probabilities is essential for the system's operation. Consequently, the results for the SVM algorithm will not be included in the forthcoming sections.

9.1 Model Performance on Familiar 'Vishing' Conversation

Batch Number	Attacker_Helper	Victim
1	Good morning, sir! I'm calling from the hospital regarding your recent visit. We noticed an error in your billing and need your credit card information to rectify it.	Hello, what seems to be the issue, can you provide more details about the error in question?
2	Absolutely, sir. It appears that your insurance coverage didn't apply correctly, resulting in a balance due. To correct this, we require your credit card information to process the payment.	That is strange. Can't I just contact my insurance provider directly to sort this out?
3	Ah, we understand your concern, but this is an urgent matter that requires immediate action. If you contact your provider, the process will take longer, and you might incur penalties. Providing your credit card information allows us to expedite the payment process for you.	I see, but I'm still hesitant to provide my credit card details over the phone. Can I visit the hospital in person to settle the payment?
4	Unfortunately, due to the current circumstances, we're unable to accommodate in-person visits. Rest assured, sir, our call is encrypted and secure. Your information will be handled with utmost confidentiality.	Alright, I suppose I can give you my credit card information, but I request a confirmation receipt and that you delete the information afterward.
5	Of course, sir. We appreciate your cooperation. Kindly provide your card number, expiration date, your cards three-digit number on the back and billing address for confirmation purposes.	Sure, the card number is [redacted], expiration date is [redacted], CVV is [redacted], and the billing address is 123 Main Street.
6	Thank you, sir. I've successfully processed the payment. You'll receive a confirmation receipt via email shortly. Is there anything else I can assist you with?	No, that will be all. Thank you for your help.

Fig 48: 'Vishing' Conversation with Known Formatting

The context of the conversation depicted in figure 30 falls under the category of a "medical scam". In this scenario, the attacker attempts to deceive the victim by referencing a recent hospital visit, a tactic designed to instill a false sense of security in the victim. While all models in this study have been trained on various medical scam themes, this particular conversation has not been directly included in their training datasets. Our objective is to evaluate whether the models can identify that this conversation falls into the category of 'Vishing' sufficiently quick. The critical test in this case is to see if the algorithms can make the classification prior to the fifth batch of the conversation because in that batch the victim succumbs to the attack and provides the requested information to the attackers.

	Logistic Regression			Random Forest			Gradient Boosted Trees			Neural Network			LSTM Neural Network		
Batch Number	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time
1	Non-Scam	91%	598ms	Non-Scam	55%	650ms	Scam	88%	607ms	Scam	73%	126ms	Scam	66%	410ms
2	Non-Scam	89%	463ms	Scam	50%	503ms	Scam	65%	492ms	Scam	74%	40ms	Scam	65%	39ms
3	Non-Scam	69%	520ms	Scam	56%	510ms	Scam	95%	489ms	Scam	73%	39ms	Scam	59%	42ms
4	Scam	50%	430ms	Scam	68%	700ms	Scam	98%	502ms	Scam	73%	52ms	Scam	57%	40ms
5	Scam	84%	462ms	Scam	70%	512ms	Scam	99%	587ms	Scam	73%	40ms	Scam	58%	45ms
6	Scam	80%	442ms	Scam	72%	490ms	Scam	99%	491ms	Scam	72%	148ms	Scam	56%	40ms

Fig 49: Model Results on Familiar 'Vishing' Conversation

- For the **Logistic Regression** model, the initial response is a categorization of 'Non-Scam' with a high probability. It's not until the fourth batch of the conversation that

the model starts to indicate the possibility of a scam, and it is only in the fifth batch that it confidently categorizes the conversation as a scam with a high probability of 84%. Thus, the Logistic Regression model does not succeed in identifying the conversation as a scam before the critical fifth batch, indicating a delay in its recognition of the scam indicators.

- The performance of the **Random Forest** model contrasts with that of the Logistic Regression algorithm. Demonstrating a quicker response, the Random Forest model begins to suspect the 'Vishing' nature of the conversation earlier. By the fourth batch, it has already formed a relatively definitive conclusion, identifying the conversation as a scam with a probability of 68%. Given how quickly the model categorized the conversation correctly, the Random Forest algorithm can be regarded as successful in detecting the scam-like characteristics of the conversation.

- The response of the **Gradient Boosted Trees** model is immediately decisive. From the very first batch of the conversation, it identifies the scenario as a scam with a high probability of 88%. Although there is a slight decrease in the certainty of this prediction to 65% in the second batch, the model consistently maintains its initial assessment. By the fourth batch, the probability of the conversation being a scam increases to 98%. This immediate level of confidence suggests that the model has recognized a pattern similar to what it encountered during training. However, this could also indicate potential overfitting, a hypothesis that requires further analysis in the subsequent discussion of model results, where more concrete data can increase our understanding of this matter.

- The **Neural Network** model consistently classifies the conversation as a 'Scam' throughout the duration of the conversation, maintaining a probability around 73%. Therefore, the model successfully passes the challenge of this conversation which is to identify it correctly prior to the fifth batch of conversation.

- The **LSTM Neural Network** also successfully identifies the potential ‘scam’ nature of the conversation from the beginning, although it is not as confident as the normal Neural Network model. The decline in its probability though, raises questions about the model’s ability to accurately differentiate between genuine and fraudulent interactions. However, this hypothesis requires further investigation in the subsequent tests.

As a result of this test, the **Random Forest** algorithm seems to have the best performance out of the tested models. Its performance is distinguished by a neutral stance maintained until the conversation includes certain key phrases that are indicative of a potential scam.

9.2 Model Performance on Unknown ‘Vishing’ Conversation

Batch Number	Attacker_Helper	Victim
1	Good morning, sir, my name is George, and I am calling today from Microsoft customer support.	Good morning, how can I help you?
2	The reason I am calling you today sir is to inform you that we have identified a virus on your computer.	Oh no, what can we do about this?
3	Well sir, we can resolve this issue for you a few easy steps, and we suggest that you hurry because you are at a vulnerable state at the moment.	Alright, understood, what do I need to do?
4	Firstly sir, you need to download an application through the link, [link redacted] and we will proceed from there.	Ok, I am doing it now, what should I do when I install the application?
5	You do not need to worry about that, we will fix the issue for you.	That is good news because I am not good with computers. Ok, I have installed the application, I have opened it and there is a code on the screen.
6	Perfect sir, tell us the code and your job will be done.	Fantastic, the code is [redacted].
7	Alright sir, we will resolve this issue for you now, remember not to close the computer for the following hours. You will receive a text notifying you that the problem is resolved. Have a great day.	Thank you so much for the support, have a great day too.

Fig 50: Vishing conversation with Unknown formatting

The dialogue illustrated in figure 32 represents an example of a "technology scam." In this typical scenario, the scam revolves around deceiving the victim into believing there is a virus on their computer. The attackers then falsely claim they can remedy the issue, provided the victim follows a specific set of instructions they provide. The format and content of this conversation is unknown to the all the models. Our goal is to assess the models' ability to accurately categorize this conversation despite its novelty. Successfully doing so would be a good indicator of the models' capacity for generalization.

	Logistic Regression			Random Forest			Gradient Boosted Trees			Neural Network			LSTM Neural Network		
Batch Number	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time
1	Non-Scam	93%	600ms	Non-Scam	54%	725ms	Non-Scam	62%	570ms	Scam	69%	130ms	Scam	60%	590ms
2	Non-Scam	95%	460ms	Non-Scam	59%	570ms	Non-Scam	96%	567ms	Scam	72%	42ms	Scam	64%	40ms
3	Non-Scam	97%	570ms	Non-Scam	61%	740ms	Non-Scam	90%	493ms	Scam	71%	40ms	Scam	56%	40ms
4	Non-Scam	96%	564ms	Non-Scam	61%	566ms	Non-Scam	94%	459ms	Scam	70%	42ms	Scam	57%	41ms
5	Non-Scam	89%	480ms	Non-Scam	54%	507ms	Non-Scam	88%	472ms	Scam	72%	133ms	Scam	56%	40ms
6	Non-Scam	67%	446ms	Scam	52%	500ms	Non-Scam	60%	479ms	Scam	72%	142ms	Scam	57%	42ms
7	Scam	54%	617ms	Scam	62%	487ms	Non-Scam	87%	662ms	Scam	73%	98ms	Scam	58%	41ms

Fig 51: Model Results on Unknown ‘Vishing’ Conversation

- Much like in the previous conversation, the **Logistic Regression** model encounters difficulties in accurately categorizing this example. It only begins to exhibit signs of suspicion by the seventh batch. This lag in response is anticipated, considering the conversation's format and content vary compared to the model's training data. The fact that the model eventually starts to identify the conversation as potentially suspicious towards the end suggests that with a more diverse and extensive dataset, it could better capture and recognize the nuances of such unfamiliar scam scenarios.

- In contrast, the **Random Forest** model exhibits a more distinct level of suspicion. While it initially classifies the conversation as 'Non-Scam', it does so with relatively low confidence, with the probability fluctuating between 54% and 61%. This suggests a degree of uncertainty in its early assessments. By the sixth batch, the model shifts its stance, categorizing the conversation as a 'Scam'. In the seventh and final batch, it solidifies this classification, suggesting that the conversation is a ‘scam’ with a probability of 62%. This behavior indicates that the Random Forest model can understand the subtleties of the conversation.

- The **Gradient Boosted Trees** model exhibits difficulty in correctly identifying the nature of the conversation. It consistently categorizes the conversation as 'Non-Scam'. However, there's notable fluctuation in its level of confidence, with the predicted probability varying between 60% and 96%.

- In contrast to the other models, the **Neural Network model** immediately and successfully identifies the conversation as 'vishing'. While such quick classification is desirable, the model's stable and consistent probability in this classification raises concerns about its ability to accurately classify normal conversations. This potential issue of oversensitivity towards identifying conversations as 'vishing' is an aspect that will be further explored in the upcoming tests involving normal conversation scenarios.

- Similar to the Neural Network model and the previous conversation example, the **LSTM Network** also successfully classifies the conversation, but with lower probability. However, this model exhibits a declining probability as the conversation progresses. This suggests that the LSTM Network while initially identifying the conversation as 'vishing', gradually becomes less certain of this classification, indicating a potential difficulty in distinguishing 'vishing' from normal conversations.

As a result, it appears that both the Neural Network and LSTM Network models demonstrate the most effective responses to the conversation. The Random Forest model also exhibits promising capabilities, as it successfully identifies the true nature of the conversation towards its conclusion. This indicates a gradual but accurate understanding of the conversation's context, highlighting the model's potential in effectively identifying such scenarios.

9.3 Model Performance on Familiar 'Normal' Conversation

Batch Number	Attacker_Helper	Victim
1	Hello, this is Jane from the IRS. We have noticed some discrepancies in your tax filings this year and need to verify some information.	Oh really? Can you provide me with some details about the discrepancies?
2	Certainly. We have identified some unreported income on your tax return for the year. Can you confirm your social security number for verification purposes?	I'm not comfortable providing my social security number over the phone. Is there any other way we can resolve this?
3	I understand your concern, but we need to verify your identity before discussing further. Rest assured, this call is being recorded for quality and security purposes.	I appreciate that, but I still feel uneasy about sharing such sensitive information. Can I visit my local IRS office to resolve this matter?
4	I apologize for the inconvenience, but due to the current circumstances, our offices are closed. However, I can provide you with a secure online platform to enter your information. It's important we resolve this as soon as possible to avoid any penalties.	Alright, if there's no other option, I guess I can use the online platform. Please guide me through the process step by step.
5	Thank you for your cooperation. I will send you an email shortly with a link to access the platform. Once you're there, follow the instructions carefully. Get back to me if you have any questions along the way.	Alright, I'll keep an eye on my email. I appreciate your assistance in resolving this matter.

Fig 52: Normal conversation with Known Formatting.

The conversation illustrated in Figure 34 presents a scenario depicting an interaction between a person and an IRS officer. This conversation poses a unique challenge due to its ambiguous nature; it could easily be classified incorrectly as a scam. The format and content of this dialogue are familiar to the models since similar conversations have been included in the training data. The objective is to evaluate the models' ability to accurately identify and classify a normal, non-scam conversation.

	Logistic Regression			Random Forest			Gradient Boosted Trees			Neural Network			LSTM Neural Network		
Batch Number	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time
1	Non-Scam	98%	542ms	Non-Scam	64%	640ms	Non-Scam	92%	563ms	Scam	61%	30ms	Scam	56%	623ms
2	Non-Scam	97%	440ms	Non-Scam	73%	534ms	Non-Scam	95%	493ms	Scam	59%	44ms	Scam	55%	41ms
3	Non-Scam	96%	402ms	Non-Scam	73%	546ms	Non-Scam	96%	501ms	Scam	57%	42ms	Scam	56%	41ms
4	Non-Scam	54%	443ms	Non-Scam	62%	490ms	Non-Scam	90%	490ms	Scam	60%	40ms	Scam	57%	40ms
5	Non-Scam	69%	470ms	Non-Scam	59%	539ms	Non-Scam	79%	498ms	Scam	63%	39ms	Scam	57%	41ms

Fig 53: Model Results on Familiar 'normal' Conversation

- In analysis of the **Logistic Regression** model's performance, it's observed that the model initially classifies the conversation as 'normal' with a very high probability of 98%. This high level of confidence is maintained up to the third batch of the conversation, after which the probability begins to fluctuate between 54% to 69%.

This behavior suggests that while the algorithm initially recognizes a familiar pattern, it starts to exhibit skepticism by the fourth batch. Overall, despite these fluctuations, the model demonstrates a good performance in this test.

- The **Random Forest** model, like the Logistic Regression model, initially classifies the conversation as 'normal', starting with a lower probability of 64%. Its confidence peaks at 73% for the third batch. However, mirroring the trend seen in the Logistic Regression model, the Random Forest model begins to show skepticism by the fourth batch, with its probability dropping to 62% and eventually decreasing further to 59% in the final batch. Overall, the Random Forest model still manages a decent performance in this test, showcasing its ability to adapt and reassess its classification as the conversation evolves.

- The **Gradient Boosted Trees** model immediately classifies the conversation as 'normal' with high probability, again indicating that the algorithm has identified a familiar pattern. In this algorithm the decline in probability is evident in the fourth batch, as we have seen on the previous algorithms, but it is much smaller, dropping from 96% on the third batch to 90% on the fourth and finally 79% on the final batch. Consequently, the Gradient Boosted Trees model is considered successful in this test.

- Both the **Neural Network** and **LSTM Network** models exhibit similar behavior, classifying the conversation as 'Vishing' and maintaining that assessment throughout the conversation. The Neural Network model's probability fluctuates around 60% while the LSTM Network's probability varies around 56%. This result confirms our previous thoughts about those two models possibly having issues with correctly classifying 'normal' conversations exhibiting oversensitivity in the 'scam-like' features of the conversation.

In conclusion, from this test, the Logistic Regression, Random Forest, and Gradient Boosted Trees models all exhibit good performance. Notably, the Logistic Regression and Random Forest models stand out, showing the most promise in their ability to accurately classify and adapt to the conversation presented.

9.4 Model Performance on Unknown 'Normal' Conversation

Batch Number	Attacker_Helper	Victim
1	Hi, I am calling from the English bank, my name is Maria.	Hello, how can I help you?
2	The reason I am calling you today sir is to inform you about our new credit cards. Do you have any accounts with us at the moment?	Yes, I do, but I am not sure I am interested in getting a credit card.
3	Getting a credit card is a fantastic way to build up credit score with the bank. I recommend we find the perfect type of card for your sir.	Ok, that seems interesting. I would like to do that please.
4	Fantastic sir, I will send you an email with our brochure about our most popular cards.	Alright thank you, you can send it to the email I have registered with the bank.
5	Wonderful, is there anything else I can do for you today sir?	No thank you, have a good day.

Fig 54: Normal conversation with Unknown Formatting.

The conversation depicted in the above Figure illustrates a phone conversation that revolves around the marketing of a bank's new credit cards. Such a conversation has not been included in the training of the models and hence the challenge it poses is of critical importance. The aim of this test is to evaluate the model's generalization abilities in a 'normal' unknown conversation.

	Logistic Regression			Random Forest			Gradient Boosted Trees			Neural Network			LSTM Neural Network		
Batch Number	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time	Result	Probability	Response Time
1	Non-Scam	66%	463ms	Non-Scam	56%	631ms	Scam	58%	487ms	Scam	69%	89ms	Non-Scam	54%	567ms
2	Non-Scam	81%	401ms	Non-Scam	63%	497ms	Scam	80%	501ms	Scam	70%	40ms	Non-Scam	55%	40ms
3	Scam	57%	563ms	Scam	54%	475ms	Scam	94%	498ms	Scam	72%	37ms	Scam	57%	42ms
4	Non-Scam	57%	446ms	Scam	52%	402ms	Scam	89%	423ms	Scam	73%	42ms	Scam	57%	39ms
5	Non-Scam	86%	511ms	Scam	54%	470ms	Scam	84%	510ms	Scam	73%	51ms	Scam	56%	41ms

Fig 55: Model Results on Unknown 'normal' Conversation

- The performance of the **Logistic Regression** model in this scenario is impressive. Initially, the model classifies the conversation as 'normal' with a probability of 81% by the second batch. In the third batch, it alters its prediction to 'vishing', but with a lower probability of 57%, suggesting it detected a feature it deemed suspicious. However, by the fourth batch, it reverts to its original classification of 'normal', again at 57% probability, and further reinforces this classification in the final batch with a heightened probability of 86%. This fluctuation and eventual stabilization in the model's predictions showcase its ability to dynamically assess and correctly categorize the conversation as 'normal', marking a successful performance in this test.

- The **Random Forest** model initially categorizes the conversation as 'normal', yet it does so with a relatively low confidence level, showing probabilities ranging from 56% to 63% up to the second batch. Exhibiting similar behavior to the Logistic Regression model, in the third batch, it changes its prediction to 'Vishing', assigning a probability of 54%. This probability level is approximately maintained for the remainder of the conversation. This pattern of behavior suggests that while the model's initial assessment is correct, it detects something potentially suspicious in the third batch and consequently maintains a state of alertness and suspicion for the duration of the conversation.

- Both the **Neural Network** and **Gradient Boosted Trees** models exhibit a similar pattern in their response to the conversation. They consistently classify the conversation as a scam right from the start and maintain this classification throughout. The Gradient Boosted Trees model does so with a relatively high probability, averaging around 85%, while the Neural Network model maintains a probability of about 72%. This consistent misclassification of a normal conversation as a scam indicates a limitation in the models' ability to generalize to new, non-scam scenarios. Consequently, this behavior suggests that both the Neural Network and Gradient Boosted Trees models fail to pass this test in terms of accurate generalization.

- The **LSTM Network** displays a response pattern similar to that of the Random Forest algorithm. In the initial two batches, the LSTM model classifies the conversation as 'normal'. From the third batch, there is a noticeable shift in its assessment, with the model reclassifying the conversation as 'Vishing'. This reclassification, however, is characterized by a relatively low probability, indicating a degree of uncertainty. This behavior suggests a cautious approach by the LSTM model in altering its classification.

As a result of this test, the Logistic Regression model emerges as the best performer among the algorithms evaluated, demonstrating a more accurate and consistent classification ability. This suggests that, of the five models tested, the Logistic Regression model possesses the strongest capability for generalization, effectively adapting to and correctly identifying the conversational context.

9.5 Response Time Performance Evaluation

As emphasized in Chapter 8, the response time of the system is crucial for the task at hand. In terms of overall performance, the PySpark models (comprising Logistic Regression, Random Forest, and Gradient Boosted Trees) demonstrate a consistent response time within the full system functionality, averaging at about 500ms. On the other hand, the Neural Network model excels as the top performer, with its response times being around 40ms. The LSTM Network, while initially showing a higher response time of approximately 500ms for the first batch of each conversation, adjusts to a similar rapid response rate of around 40ms from the second batch onwards. This discrepancy in response times between the PySpark models and the Neural Network-based models is in line with the expectations and analysis presented in Chapter 8.

9.6 Overall Results

In conclusion, the Logistic Regression and Random Forest models demonstrate the most effective performance among the tested algorithms, with Random Forest model being the better of the two. These models have successfully learned underlying patterns during training, enabling them to differentiate between 'Vishing' and 'Normal' conversations. Throughout the various conversation tests, they have shown an ability to make reasoned decisions and adapt their responses when necessary. This is observed in instances where a suspicious phrase is mentioned; both models show a shift in their probability predictions, as illustrated in the test example 9.4 for reference.

On the other hand, the Gradient Boosted Trees algorithm tends to perform well on data similar to its training set, often showing high confidence in its decisions. However, this behavior suggests a potential overfitting issue, as evidenced by its less effective performance on unfamiliar and ambiguous conversations. This result is in alignment with the training results discussed in chapter 7.4.4 where the slightly poor generalization ability of the model was highlighted.

The Neural Network and LSTM models tend to label conversations as 'Vishing' more often than 'Normal', a behavior especially noticeable in the Neural Network model compared to the LSTM. This is evident in tests where these models frequently label conversations as 'Vishing', even in the initial stages of the conversation where the interactions are normal. Such behavior could imply that these models have learned to associate certain contexts, like a bank-related call, with scams. This indicates a need for training on a more diverse dataset, including a range of completely normal conversations.

Overall, the Logistic Regression and Random Forest models demonstrate promising results, whereas the other models face challenges in accurately classifying conversations. This outcome aligns with expectations, given the relatively small size of the training dataset compared to what is typically required for such tasks. Nevertheless, the results are encouraging: the aforementioned models successfully identified 'Scam-like' characteristics in conversations that were new to them, indicating a good ability to generalize effectively despite the limited data and diversity of those data encountered during training.

10. Deductions, Limitations and Future Research

Bibliography

[1] Inside the intricate world of voice phishing (joins.com)

[2] Voice phishing - Wikipedia

[3] What Is a Vishing Attack? | Fortinet

[4] What is Vishing (Voice Phishing)? Examples You Need to Know (softwarelab.org)

[5] What is dumpster diving? (techtarget.com)

[6] Song, J., Kim, H., & Gkelias, A. (2014). iVisher: Real-Time Detection of Caller ID Spoofing. *ETRI Journal*, *36*(5), 865-875.

[7] Norris, G., & Brookes, A. (2021). Personality, emotion and individual differences in response to online fraud. *Personality and Individual Differences*, *169*, 109847.

[8] Jones, K. S., Armstrong, M. E., Tornblad, M. K., & Siami Namin, A. (2021). How social engineers use persuasion principles during vishing attacks. *Information & Computer Security*, *29*(2), 314-331.

[9] What Is a Vishing Attack | Examples & Prevention | Imperva

[10] Xing, J., Yu, M., Wang, S., Zhang, Y., & Ding, Y. (2020). Automated fraudulent phone call recognition through deep learning. *Wireless Communications and Mobile Computing*, *2020*, 1-9.

[11] Lee, M., & Park, E. (2023). Real-time Korean voice phishing detection based on machine learning approaches. *Journal of Ambient Intelligence and Humanized Computing*, *14*(7), 8173-8184.

[12] Tran, M. H., Hoai, T. H. L., & Choo, H. (2020). A third-party intelligent system for preventing call phishing and message scams. In *Future Data and Security Engineering. Big Data, Security and Privacy, Smart City and Industry 4.0 Applications: 7th International Conference, FDSE 2020, Quy Nhon, Vietnam, November 25–27, 2020, Proceedings 7* (pp. 486-492). Springer Singapore.

[13] Salloum, S., Dautov, R., Chen, X., Peng, P. X., & Huang, J. Z. (2016). Big data analytics on Apache Spark. *International Journal of Data Science and Analytics*, *1*, 145-164.

[14] Assefi, M., Behravesh, E., Liu, G., & Tafti, A. P. (2017, December). Big data machine learning using apache spark MLlib. In *2017 ieee international conference on big data (big data)* (pp. 3492-3498). IEEE.

[15] Armbrust, M., Xin, R. S., Lian, C., Huai, Y., Liu, D., Bradley, J. K., ... & Zaharia, M. (2015, May). Spark sql: Relational data processing in spark. In *Proceedings of the 2015 ACM SIGMOD international conference on management of data* (pp. 1383-1394).

[16] The Spark cluster architecture for resource allocation and data transfer | Download Scientific Diagram (researchgate.net)

[17] Apache Spark Components (Updated) (intellipaat.com)

[18] Key features of Spark's MLlib: aspark.mllib is built on top of RDDs,... | Download Scientific Diagram (researchgate.net)

[19] Meng, X., Bradley, J., Yavuz, B., Sparks, E., Venkataraman, S., Liu, D., ... & Talwalkar, A. (2016). Mllib: Machine learning in apache spark. *The journal of machine learning research*, *17*(1), 1235-1241.

[20] Optimization - RDD-based API - Spark 3.5.0 Documentation (apache.org)

[21] Broyden–Fletcher–Goldfarb–Shanno algorithm - Wikipedia

[22] ML Pipelines - Spark 3.5.0 Documentation (apache.org)

[23] Spark ML pipeline for regression | gbhat.com

[24] Spark Streaming - Spark 3.5.0 Documentation (apache.org)

[25] What is Speech to Text? - Speech to Text Explained - AWS (amazon.com)

[26] Trivedi, A., Pant, N., Shah, P., Sonik, S., & Agrawal, S. (2018). Speech to text and text to speech recognition systems-Areview. *IOSR J. Comput. Eng*, *20*(2), 36-43.

[27] Speech-to-Text: Automatic Speech Recognition | Google Cloud

[28] Generative pre-trained transformer - Wikipedia

[29] Transformer (machine learning model) - Wikipedia

[30] What is a Transformer?. An Introduction to Transformers and… | by Maxime | Inside Machine learning | Medium

[31] Prompt engineering - Wikipedia

[32] Master the Perfect ChatGPT Prompt Formula (in just 8 minutes)! (youtube.com)

[33] What is Tokenization? Types, Use Cases, Implementation | DataCamp

[34] Tokenization in Natural Language Processing | by Mohammad Derakhshan | Medium

[35] Perkins, J. (2010). *Python text processing with NLTK 2.0 cookbook*. PACKT publishing.

[36] NLTK :: nltk.tokenize.word_tokenize

[37] Introduction to Stemming - GeeksforGeeks

[38] All you need to know about text preprocessing for NLP and Machine Learning - KDnuggets

[39] Stemming - Wikipedia

[40] pyspark.sql.functions.flatten — PySpark 3.5.0 documentation (apache.org)

[41] tf–idf - Wikipedia

[42] Feature hashing - Wikipedia

[43] Machine learning - Wikipedia

[44] Hastie, T., Tibshirani, R., Friedman, J., Hastie, T., Tibshirani, R., & Friedman, J. (2009). Overview of supervised learning. *The elements of statistical learning: Data mining, inference, and prediction*, 9-41.

[45] Supervised learning - Wikipedia

[46] Unsupervised learning - Wikipedia

[47] James, G., Witten, D., Hastie, T., Tibshirani, R., & Taylor, J. (2023). Unsupervised learning. In *An Introduction to Statistical Learning: with Applications in Python* (pp. 503-556). Cham: Springer International Publishing.

[48] Mnih, V., Kavukcuoglu, K., Silver, D., Rusu, A. A., Veness, J., Bellemare, M. G., ... & Hassabis, D. (2015). Human-level control through deep reinforcement learning. *nature*, *518*(7540), 529-533.

[49] Ying, X. (2019, February). An overview of overfitting and its solutions. In *Journal of physics: Conference series* (Vol. 1168, p. 022022). IOP Publishing.

[50] overfitting - Search Images (bing.com)

[51] Shorten, C., Khoshgoftaar, T. M., & Furht, B. (2021). Text data augmentation for deep learning. *Journal of big Data*, *8*, 1-34.

[52] Data augmentation - Wikipedia

[53] Regularization in Machine Learning - GeeksforGeeks

[54] Cross-validation (statistics) - Wikipedia

[55] Binary classification - Wikipedia

[56] Logistic Regression in Machine Learning - GeeksforGeeks

[57] Logistic regression - Wikipedia

[58] Support Vector Machine (SVM) Algorithm - GeeksforGeeks

[59] Support vector machine - Wikipedia

[60] Applied Sciences | Free Full-Text | Support Vector Machine-Based EMG Signal Classification Techniques: A Review (mdpi.com)

[61] Support Vector Machines (SVM) | LearnOpenCV #

[62] De Ville, B. (2013). Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, *5*(6), 448-455.

[63] Decision tree - Wikipedia

[64] Decision Tree - GeeksforGeeks

[65] Random forest - Wikipedia

[66] Bootstrap aggregating - Wikipedia

[67] Gradient boosting - Wikipedia

[68] Gradient Boosting in ML - GeeksforGeeks

[69] Evolution and Concepts Of Neural Networks | Deep Learning (analyticsvidhya.com)

[70] What is a neural network? - GeeksforGeeks

[71] Activation functions in Neural Networks - GeeksforGeeks

[72] Softmax function - Wikipedia

[73] A Gentle Introduction to Dropout for Regularizing Deep Neural Networks - MachineLearningMastery.com

[74] Gradient Descent Algorithm in Machine Learning - GeeksforGeeks

[75] Backpropagation in Data Mining - GeeksforGeeks

[76] Long short-term memory - Wikipedia

[77] Understanding of LSTM Networks - GeeksforGeeks

[78] Understanding LSTMs | Black Box ML (kushalj001.github.io)