# UNIVERSITY OF PATRAS

**DEPARTMENT OF ELECTRICAL AND COMPUTER**

**ENGINEERING**

**DIVISION of electronics and computers (PC)**

**INTERACTIVE TECHNOLOGIES LABORATORY**



**Design and Development of a Location-Based Customer support application for Supermarkets**

**D I P L O M A T H E S I S**

## PANAGIOTIS ARVANITIS

**SUPERVISOR: NIKOLAOS AVOURIS**

**PATRAS –2024**

University of Patras, Department of Electrical and Computer Engineering.

Panagiotis Arvanitis

© 2024 – All rights reserved

The whole work is an original work, produced by Panagiotis Arvanitis, and does not violate the rights of third parties in any way. If the work contains material which has not been produced by him/her, this is clearly visible and is explicitly mentioned in the text of the work as a product of a third party, noting in a similarly clear way his/her identification data, while at the same time confirming that in case of using original graphics representations, images, graphs, etc., has obtained the unrestricted permission of the copyright holder for the inclusion and subsequent publication of this material.

# ABSTRACT

**Design and Development of a Location-Based Customer support application for Supermarkets**

|  |  |
| --- | --- |
| **STUDENT NAME, SURNAME:**  **SKARPETIS IOANNIS** | **SUPERVISOR NAME, SURNAME:**  **FEIDAS CHRISTOS** |

The subject of this thesis was the development and design of a location-based customer support application for supermarkets. The application aims to enhance customer experience by providing real-time assistance and personalized shopping recommendations. Customers can input their shopping list, and based on various factors such as weather, distance, means of transport, cost at each supermarket, and time of day, an AI model recommends the optimal supermarket for their shopping. The system integrates geolocation technologies to provide accurate guidance and suggestions, improving the overall shopping experience.

Key technologies used in the implementation include React and Bootstrap for the front end, offering a responsive and dynamic user interface. React's component-based architecture and Bootstrap's pre-designed components facilitate efficient UI development. Vite was used as the build tool for its fast development server and optimized build process. Node.js and Express power the backend, providing a robust and scalable server environment. Flask is employed alongside scikit-learn for the AI recommendation system, with the GradientBoostingRegressor algorithm implemented in Jupyter notebooks to handle the recommendation logic. Flask's lightweight framework and scikit-learn's extensive machine learning tools ensure efficient and effective AI model deployment.

Testing scenarios demonstrate the application's effectiveness in offering tailored shopping advice, leading to improved customer satisfaction and operational efficiency. A usability study evaluates the application's performance, providing insights into user interaction and potential areas for enhancement. The findings suggest that the application significantly enhances the customer support experience, making it a valuable tool for modern retail environments.

Keywords: Location-Based Services, Customer Support, Supermarket Application, Geolocation, AI Recommendations, React, Bootstrap, Vite, Node.js, Express, Flask, scikit-learn, GradientBoostingRegressor, Jupyter Notebook, User Experience, Retail Technology, Mobile Application Development.

# ΕΚΤΕΤΑΜΕΝΗ ΕΛΛΗΝΙΚΗ ΠΕΡΙΛΗΨΗ

**Μελέτη συστήματος ανίχνευσης τηλεφωνικής απάτης σε πραγματικό χρόνο με τεχνολογία μηχανικής μάθησης**

**ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΦΟΙΤΗΤΗ: ΟΝΟΜΑΤΕΠΩΝΥΜΟ ΕΠΙΒΛΕΠΟΝΤΟΣ:**

**ΑΡΒΑΝΙΤΗΣ ΠΑΝΑΓΙΩΤΗΣ ΑΒΟΥΡΗΣ ΝΙΚΟΛΑΟΣ**

Αντικείμενο της παρούσας διπλωματικής εργασίας ήταν η ανάπτυξη και ο σχεδιασμός μιας εφαρμογής υποστήριξης πελατών με βάση την τοποθεσία για σούπερ μάρκετ. Η εφαρμογή στοχεύει στη βελτίωση της εμπειρίας των πελατών παρέχοντας βοήθεια σε πραγματικό χρόνο και εξατομικευμένες προτάσεις αγορών. Οι πελάτες μπορούν να εισάγουν τη λίστα με τα ψώνια τους και με βάση διάφορους παράγοντες, όπως ο καιρός, η απόσταση, το μέσο μεταφοράς, το κόστος σε κάθε σούπερ μάρκετ και η ώρα της ημέρας, ένα μοντέλο τεχνητής νοημοσύνης συνιστά το βέλτιστο σούπερ μάρκετ για τα ψώνια τους. Το σύστημα ενσωματώνει τεχνολογίες γεωεντοπισμού για να παρέχει ακριβή καθοδήγηση και προτάσεις, βελτιώνοντας τη συνολική εμπειρία αγορών.

Οι βασικές τεχνολογίες που χρησιμοποιήθηκαν στην υλοποίηση περιλαμβάνουν το React και το Bootstrap για το front end, προσφέροντας μια ευέλικτη και δυναμική διεπαφή χρήστη. Η αρχιτεκτονική του React που βασίζεται σε συστατικά και τα προσχεδιασμένα συστατικά του Bootstrap διευκολύνουν την αποτελεσματική ανάπτυξη του UI. Το Vite χρησιμοποιήθηκε ως εργαλείο κατασκευής για τον γρήγορο διακομιστή ανάπτυξης και τη βελτιστοποιημένη διαδικασία κατασκευής. Το Node.js και η Express τροφοδοτούν το backend, παρέχοντας ένα στιβαρό και κλιμακούμενο περιβάλλον διακομιστή. Το Flask χρησιμοποιείται παράλληλα με το scikit-learn για το σύστημα συστάσεων AI, με τον αλγόριθμο GradientBoostingRegressor να υλοποιείται σε Jupyter Notebook για να χειριστεί τη λογική των συστάσεων. Το ελαφρύ framework του Flask και τα εκτεταμένα εργαλεία μηχανικής μάθησης του scikit-learn εξασφαλίζουν την αποδοτική και αποτελεσματική ανάπτυξη μοντέλων AI.

Τα σενάρια δοκιμών καταδεικνύουν την αποτελεσματικότητα της εφαρμογής στην παροχή εξατομικευμένων συμβουλών αγορών, οδηγώντας σε βελτιωμένη ικανοποίηση των πελατών και λειτουργική αποτελεσματικότητα. Μια μελέτη ευχρηστίας αξιολογεί την απόδοση της εφαρμογής, παρέχοντας πληροφορίες σχετικά με την αλληλεπίδραση των χρηστών και τους πιθανούς τομείς για βελτίωση. Τα ευρήματα υποδηλώνουν ότι η εφαρμογή βελτιώνει σημαντικά την εμπειρία υποστήριξης πελατών, καθιστώντας την πολύτιμο εργαλείο για σύγχρονα περιβάλλοντα λιανικής πώλησης.

Λέξεις-κλειδιά: Υπηρεσίες βάσει τοποθεσίας, Υποστήριξη πελατών, Εφαρμογή σούπερ μάρκετ, Γεωεντοπισμός, Συστάσεις AI, React, Bootstrap, Vite, Node.js, Express, Flask, scikit-learn, GradientBoostingRegressor, Jupyter Notebook, Εμπειρία χρήστη, Τεχνολογία λιανικής, Ανάπτυξη εφαρμογών για κινητά.

# ACKNOWLEDGEMENTS

With the completion of this dissertation, I conclude my studies at the University of Patras. I would like to express my deepest gratitude to my supervisor, Mr. Nikolaos Avouris, for his invaluable support and guidance throughout this journey. His insights and expertise were crucial in navigating the challenges I encountered, and I am sincerely thankful for his continuous encouragement.

I would also like to extend my heartfelt thanks to my parents, girlfriend, and friends. Their unwavering support, belief in me, and constant presence by my side were instrumental in my success.

# Table Of Contents

Table Of Figures ....................................................................................................................... 14

1. Introduction ......................................................................................................................... 17
2. Analysis of the problem under research ............................................................................. 18
   1. Vishing attacks and methods ........................................................................................ 18
      1. Typical Vishing Attack Examples ............................................................................. 20
   2. Current Approaches and Emerging Frontiers in Vishing Detection .............................. 20
   3. Why Real-Time In-Call Detection Is Essential ............................................................... 21
   4. Addressing the Identified Problem ............................................................................... 22
3. PySpark ................................................................................................................................ 23
   1. Apache Spark Fundamentals ........................................................................................ 23
      1. Spark Architecture .................................................................................................. 23
      2. Spark Components ................................................................................................. 24
      3. Spark Core .............................................................................................................. 24
         1. Resilient Distributed Dataset ........................................................................... 24
   2. Spark SQL ...................................................................................................................... 25
   3. Spark’s Machine Learning Package ............................................................................... 26
      1. Spark MLlib ......................................................................................................... 26
      2. Spark ML ............................................................................................................. 27
         1. Machine Learning Pipelines API .................................................................. 27
         2. Spark ML Model Tuning ............................................................................... 28
   4. Spark Structured Streaming .......................................................................................... 29
4. Speech-To-Text .................................................................................................................... 30
   1. Technology Analysis ...................................................................................................... 30
   2. Google Speech-To-Text API .......................................................................................... 31
5. Dataset Creation .................................................................................................................. 33
   1. Generative Pretrained Transformers ............................................................................ 33
      1. Transformers .......................................................................................................... 33
      2. Generative Pretrained Transformers ..................................................................... 34
         1. Utilizing GPT for Dataset Generation .............................................................. 34
   2. Data Generation with Chat GPT API ............................................................................. 35
      1. API Overview .......................................................................................................... 35
      2. Prompt Engineering ................................................................................................ 35
      3. Data Collection Strategy ......................................................................................... 36
      4. Data Quality and Future Improvements ................................................................. 37
   3. Dataset Preparation ...................................................................................................... 38
   4. Final Labelling Process .................................................................................................. 39
6. Dataset Preprocessing ......................................................................................................... 40
   1. Initial Preprocessing ...................................................................................................... 40
      1. Numeric Character Replacement ........................................................................... 40
      2. Special Character Removal ..................................................................................... 40
      3. Tokenization ........................................................................................................... 41
      4. Stemming ................................................................................................................ 42
   2. Preprocessing Pipeline for Model Training ................................................................... 43
      1. Flatten Transformer................................................................................................ 43
      2. TF-IDF ...................................................................................................................... 44
         1. HashingTF Transformer ................................................................................... 44
         2. IDF Transformer ............................................................................................... 45
      3. Vector Assembler Transformer .............................................................................. 45
   3. Preprocessing Pipeline for Deployment ....................................................................... 46
7. Model Architecture – Training – Tuning .............................................................................. 47
   1. Machine Learning Basics ............................................................................................... 47
   2. Pre-Modeling Considerations ....................................................................................... 48
      1. Overfitting .............................................................................................................. 49
      2. Cross-Validation ...................................................................................................... 51
      3. Model Evaluation Metrics ...................................................................................... 51
      4. Optimizers .............................................................................................................. 52
   3. Model Architecture ....................................................................................................... 53
      1. Logistic Regression ................................................................................................. 53
      2. Support Vector Machine ........................................................................................ 55
      3. Random Forest ....................................................................................................... 56
         1. Decision Trees ............................................................................................. 56
      4. Gradient Boosted Trees .......................................................................................... 58
      5. Neural Networks ..................................................................................................... 59
         1. LSTM Neural Networks .................................................................................... 64
   4. Model Training - Tuning – Results ................................................................................ 66
      1. Logistic Regression ................................................................................................. 66
      2. Random Forest ....................................................................................................... 67
      3. Support Vector Machines ....................................................................................... 68
      4. Gradient Boosted Trees .......................................................................................... 69
      5. Neural Network ...................................................................................................... 70
      6. LSTM Neural Network ............................................................................................ 72
      7. Overall Results ........................................................................................................ 75
8. Overall System Functionality ............................................................................................... 76
   1. Stage One ...................................................................................................................... 76
   2. Stage Two...................................................................................................................... 77
   3. Stage Three ................................................................................................................... 78
   4. Challenges and Future Enhancements ......................................................................... 78
9. Experimental Results ........................................................................................................... 80
   1. Model Performance on Familiar 'Vishing' Conversation .............................................. 81
   2. Model Performance on Unknown ‘Vishing’ Conversation ........................................... 82
   3. Model Performance on Familiar ‘Normal’ Conversation .............................................. 84
   4. Model Performance on Unknown ‘Normal’ Conversation ........................................... 85
   5. Response Time Performance Evaluation ...................................................................... 86
   6. Overall Results .............................................................................................................. 86
10. Deductions, Limitations and Future Research................................................................... 88

Bibliography ............................................................................................................................ 89

# Table Of Figures

Fig 1: Mobile Phone Phishing Attack ....................................................................................... 18

Fig 2: Spark Cluster Architecture [16] ...................................................................................... 23

Fig 3: Spark Main Components [17] ........................................................................................ 24

Fig 4: RDD lineage graph [13] .................................................................................................. 25

Fig 5: Spark SQL Interaction with Spark and RDDs [15] ........................................................... 25

Fig 6: Key Features of Spark’s Machine Learning package [18] .............................................. 26

Fig 7: Spark Machine Learning Pipeline Example [23] ............................................................. 28

Fig 8: Typical Streaming Process [24] ...................................................................................... 29

Fig 9: General Speech to Text Architecture [26] ..................................................................... 30

Fig 10: Speech to Text: Analog to Digital signal conversion .................................................... 31

Fig 11: The Transformer – Model Architecture [30]................................................................ 33

Fig 12: The Original Generative Pretrained Transformer Model [28] ..................................... 34

Fig 13: Dataset Creation Pipeline ............................................................................................ 36

Fig 14: Result of a normal (non-scam) conversation from Chat GPT API. ............................... 37 Fig 15: Result of a scam conversation from Chat GPT API. ..................................................... 37

Fig 16: Conversation Split to Support Better Training with the aim of supporting Real-Time

functionality. ........................................................................................................................... 38 Fig 17: Example of the above Dataset Preparation ................................................................. 38

Fig 18: Initial Preprocessing Pipeline ....................................................................................... 40

Fig 19: Word Tokenization Example [34]................................................................................. 41

Fig 20: Example of Stemming in the English language [38] ..................................................... 42

Fig 21: Indicative Data Post Initial Preprocessing .................................................................... 42

Fig 22: Example Output from the Initial Preprocessing Pipeline ............................................. 43

Fig 23: Preprocessing Pipeline for Model Training .................................................................. 43

Fig 24: Example of Data Post-Processing Through the Model Training Preprocessing Pipeline

................................................................................................................................................. 45

Fig 25: Example of Overfitting [53]. ......................................................................................... 49

Fig 26: Example of Early Stopping Point [50]........................................................................... 49

Fig 27: Diagram of K-fold cross-validation [54] ....................................................................... 51

Fig 28: Example of SVM hyperplane [60] ................................................................................ 55

Fig 29: Support Vectors [61] .................................................................................................... 55

Fig 30: A decision tree illustrating analysis of survival in Titanic Sinking [62] ......................... 56

Fig 31: Sequential Construction of Trees in Gradient Boosting [68] ....................................... 58

Fig 32: Depiction of the Structure of a Neural Network [69] .................................................. 59

Fig 33: Depiction of Neuron’s Output Calculation [70] ........................................................... 60

Fig 34: Simple Example of Neural network Structure ............................................................. 63

Fig 35: LSTM Cell Structure [78] .............................................................................................. 64

Fig 36: LSTM Neural Network Structure Example ................................................................... 65

Fig 37: Hyperparameter range of values for Logistic Regression ............................................ 67

Fig 38: Logistic Regression Training Results ............................................................................ 67

Fig 39: Hyperparameter range of values for Random Forest .................................................. 68 Fig 40: Random Forest Training Results .................................................................................. 68

Fig 41: Hyperparameter range of values for Support Vector Machines ................................. 68

Fig 42: Support Vector Machines Training Results .................................................................. 69

Fig 43: Hyperparameter Range of Values for Gradient Boosted Trees ................................... 69

Fig 44: Gradient Boosted Trees Training Results ..................................................................... 69

Fig 45: Optimal Hyperparameters as a result of the tuning process for the Neural Network 71

Fig 46: Neural Network Training Numerical Results ............................................................... 72

Fig 47: Hyperparameter Tuning Ranges for the LSTM Model Tuning ..................................... 73

Fig 48: Optimal Hyperparameters as a result of the tuning process for the LSTM Network .. 73

Fig 49: LSTM Network Training Results ................................................................................... 74

Fig 50: All Model Results ......................................................................................................... 75

Fig 51: Complete System Functionality Workflow .................................................................. 76

Fig 52: Stage One of the Complete System Workflow ............................................................ 76

Fig 53: Stage Two of the Complete System Workflow ............................................................ 77

Fig 54: Stage Three of the Complete System Workflow .......................................................... 78

Fig 55: Testing Workflow ......................................................................................................... 80

Fig 56: ‘Vishing' Conversation with Known Formatting .......................................................... 81

Fig 57: Model Results on Familiar ‘Vishing’ Conversation....................................................... 81

Fig 58: Vishing conversation with Unknown formatting ......................................................... 82

Fig 59: Model Results on Unknown ‘Vishing’ Conversation .................................................... 83

Fig 60: Normal conversation with Known Formatting. ........................................................... 84

Fig 61: Model Results on Familiar ‘normal’ Conversation ....................................................... 84

Fig 62: Normal conversation with Unknown Formatting. ....................................................... 85

Fig 63: Model Results on Unknown ‘normal’ Conversation .................................................... 85

# 1. Introduction

The motivation for this project stemmed from observing my parents' constant efforts to find the best deals on supermarket products through TV ads and flyers. Their dedication to saving money on groceries inspired me to think about how technology could simplify and enhance this process. With the rise of big tech platforms like Skroutz, which aggregate prices and deals from various online stores, I saw an opportunity to create a similar tool for supermarket shopping. My aim was to develop an application that harnesses the power of AI and machine learning to provide real-time, personalized recommendations on where to find the best deals, making it easier for families like mine to save both time and money.

Moreover, I was driven by a desire to merge the worlds of AI/ML and full-stack application development. This project represents a confluence of my interests in cutting-edge technology and practical, user-centric design. By incorporating AI algorithms, such as the GradientBoostingRegressor, and using tools like React, Bootstrap, Node.js, Express, Flask, and scikit-learn, I aimed to create a robust, scalable, and user-friendly application. The challenge was to build a system that not only performs complex data analysis and provides accurate recommendations but also delivers a seamless and engaging user experience. This endeavor allowed me to explore and integrate multiple technological domains, pushing the boundaries of what a supermarket shopping assistant could achieve.

Ultimately, the goal of this project is to develop an application that can be used by everyone, democratizing access to the best supermarket deals and enhancing the overall shopping experience. By leveraging location-based services and AI-driven recommendations, the app aims to provide users with real-time assistance and tailored shopping suggestions based on factors such as weather, distance, transportation means, and current supermarket prices. This application not only helps users save money but also promotes smarter and more efficient shopping habits. In doing so, it contributes to the broader objective of making everyday life easier and more cost-effective for people from all walks of life.

# 2. Analysis of the problem under research

## 2.1 Understanding the Problem Context

When creating a location-based customer support application for supermarket shopping, several critical factors must be considered. These factors influence the recommendation of the best supermarket for a user, based on real-time conditions and user-specific preferences. The primary objective is to aid users in making informed decisions that optimize for cost, convenience, and time efficiency.

## 2.2 Real-Life Scenarios and Challenges

Consider the scenario where a user has created their grocery list, but it is raining, and they are on foot. The app needs to determine whether it is more beneficial to visit a farther, cheaper supermarket or a closer, slightly more expensive one. Another scenario might involve the user shopping late in the day, with supermarkets closing soon. Here, the decision revolves around whether to risk going to a distant, cheaper supermarket or opting for a nearby one to ensure they make it in time. These examples highlight the practical dilemmas the app is designed to resolve by using AI-driven recommendations.

**User Stories:**

* **User Story 1:** As a pedestrian during rainy weather, I want the app to recommend whether I should go to a closer, slightly more expensive supermarket or a farther, cheaper one, so that I can make an informed decision without getting caught in the rain unnecessarily.
* **User Story 2:** As a shopper late in the day, I want the app to suggest whether I should risk going to a distant, cheaper supermarket or choose a nearby one to ensure I can shop before closing time, optimizing both cost and convenience.

## 2.3 Technological and AI Integration

To address these challenges, the application integrates advanced technologies such as geolocation services, AI models, PostgreSQL for database management, and pgAdmin for database administration. The AI component, leveraging machine learning algorithms like GradientBoostingRegressor from scikit-learn, processes various input factors to generate optimal shopping recommendations. This approach ensures that the recommendations are not just based on static data but adapt to real-time conditions and user-specific contexts, providing a personalized and efficient shopping experience.

**Technologies Used:**

* **Geolocation Services:** Utilized to pinpoint the user's location accurately and facilitate recommendations based on proximity to supermarkets.
* **AI and Machine Learning:** Employed GradientBoostingRegressor from scikit-learn to analyze dynamic factors such as weather conditions, time of day, and transportation means. This allows the app to generate tailored recommendations that optimize for cost, convenience, and real-time conditions.
* **Database Management:** PostgreSQL used to store and manage application data efficiently, ensuring robust data handling and retrieval capabilities.
* **Database Administration:** pgAdmin employed for database administration tasks such as monitoring, querying, and managing PostgreSQL databases, providing a user-friendly interface for database management.
* **Frontend Technologies:** Developed using React.js and Bootstrap for a responsive and user-friendly interface, ensuring seamless interaction and intuitive navigation.
* **Backend Technologies:** Implemented with Node.js and Express.js to handle server-side operations, manage data, and integrate with external APIs for real-time updates.
* **Data Science Tools:** Leveraged Flask in conjunction with scikit-learn for backend AI model deployment and Flask-RESTful for creating APIs that interact with the AI models.

**User Stories:**

* **User Story 3:** As a user of the app, I expect it to utilize AI-driven algorithms to consider factors like weather conditions, time of day, and transport means when recommending supermarkets, ensuring the suggestions are tailored to my current situation.

## 2.4 Enhancing User Decision-Making

The primary goal is to shift user decision-making from intuition-based choices to data-driven recommendations. By considering a comprehensive set of factors including weather, distance, transportation means, and store operation hours, the app aims to provide users with the most convenient and cost-effective shopping options. This shift not only enhances the shopping experience but also contributes to significant savings in time and money for the users.

**User Stories:**

* **User Story 4:** As a shopper, I want the app to help me make data-driven decisions by providing recommendations based on factors like cost, distance, and real-time conditions, ensuring I can optimize my grocery shopping experience.

## 2.5 Conclusion

The application addresses a common problem faced by many shoppers by integrating AI and full-stack development to provide practical solutions. By making use of real-time data and personalized recommendations, it effectively enhances user decision-making, ensuring an improved overall shopping experience. This innovation represents a significant advancement in retail technology, offering tangible benefits to consumers.

**User Stories:**

* **User Story 5:** As a frequent shopper using the app, I want to save and access my grocery lists for different days, and receive recommendations based on these lists in advance, enabling me to plan my shopping trips efficiently and effectively.

# 3. Application Ecosystem

Achieving our goals of creating a well-structured full stack web application while also seamlessly merging Machine learning and AI recommendations requires a complex and well-integrated technology ecosystem. This involves leveraging various technologies and frameworks at different layers to ensure scalability, efficiency, and maintainability. Below is a detailed overview of the technologies used across different layers of our web application architecture:

Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, διάγραμμα

Περιγραφή που δημιουργήθηκε αυτόματα

## 3.1. Frontend / Client Layer

### React

* **Description**: React [16] is a free and open-source JavaScript library used for developing user interfaces. It is widely recognized as the most popular solution for the rapid and efficient creation of complex user interfaces, and it is employed in the development of both web and mobile applications. Applications developed with React are divided into smaller parts called components, each with specific functionality. For example, a navigation bar at the top of a webpage could be a separate component. This modular approach results in a large and complex component, a webpage, that is composed of many smaller components [16]. One of the main advantages of using React is the ease it offers in application development. React achieves this speed through a Virtual DOM (Document Object Model). The Virtual DOM tracks the application's data, comparing previous states and updating only the elements in the actual DOM where a change is detected, instead of fully refreshing the page as in conventional web applications. React.js also benefits from a large and active community, providing a wealth of resources, libraries, and add-ons that can be easily integrated into applications. Overall, React.js is a suitable choice for creating dynamic and responsive user interfaces due to its component-based architecture, Virtual DOM, and efficient rendering.
* **Advantages**: Component-based architecture, fast rendering with Virtual DOM, and a strong ecosystem of libraries.
* **Used for**: Front end development

### ****Material-UI (MUI)****

* **Description**: A popular React UI framework that implements Google's Material Design.
* **Advantages**: Provides a comprehensive set of components, consistent design, and easy customization.
* **Used for**: Design purposes

### ****Leaflet****

* **Description**: An open-source JavaScript library for interactive maps.
* **Advantages**: Lightweight, easy to use, and highly customizable with a variety of plugins.
* **Used for**: Geolocation map rendering

## 3.2 API Layer

### **Web Services API**

* **Description**: This layer serves as the interface between the frontend and backend, handling HTTP requests and responses.
* **Advantages**: Decouples the frontend from the backend, facilitating independent development and scalability.
* **Used for**: Exchanging information between client and sever

## 3.3 Backend / Server Layer

### ****Node.js****

* **Description**: Node.js [15] is a free and open-source JavaScript runtime environment that has become one of the most popular choices for developers in web application development. Over the years, various programming languages have been used to process requests from web browsers on a server, such as PHP, Ruby, Python, ASP, and Java. In contrast, JavaScript was primarily used in browsers to provide interactivity on websites, such as dropdown menus [15]. In 2009, Ryan Dahl created Node.js, which is built on the V8 engine of the Google Chrome browser, enabling the execution of server-side JavaScript code. This allowed JavaScript to be used for developing applications on both the server side and the client side. Node.js implements an asynchronous, non-blocking input/output model through the use of events, using a single execution thread. Thus, when performing an input/output operation, such as reading data from a database, Node.js continues executing until the data is available instead of blocking the execution thread. Applications developed in Node.js are characterized by high speed and efficiency, making Node.js a suitable choice for applications that require high interactivity and real-time data updates. Another advantage of using Node.js is the ability to use a common language for developing both frontend and backend
* **Advantages**: Non-blocking, event-driven architecture, which allows handling multiple connections concurrently.
* **Used for**: Web server development

### ****Express.js****

* **Description**: Express.js [10] is a free and open-source library/framework for Node.js, used for the rapid and easier development of web servers that process requests from browsers. Express.js provides a higher level of abstraction compared to Node, making it easier for developers to utilize its functionality and additional capabilities [10]. One of the significant advantages of using Express.js is the use of JavaScript for both frontend and backend development in a web application. Therefore, developers do not need to learn or use any other language. Additionally, since it is based on the V8 engine of the Google Chrome browser, Express.js is an ideal choice for applications requiring fast response times and request processing, as well as for real-time applications.
* **Advantages**: Minimal and flexible, with a rich set of HTTP utility methods and middleware.
* **Used for**: Web server development

## 3.4. AI / Recommendation Layer

### Jupyter

* **Description**: Jupyter [23] is an open-source platform that provides interactive notebooks for data analysis, visualization, and machine learning. It supports various programming languages, including Python, R, and Julia, making it a versatile tool for data scientists and researchers. One of the significant advantages of using Jupyter is its ability to combine code, text, and visualizations in a single document, facilitating exploratory data analysis and collaboration. Jupyter notebooks enable users to document their analysis process, share insights, and reproduce results easily. Additionally, Jupyter's integration with various data science libraries and tools enhances its functionality, making it a powerful platform for data-driven projects.
* **Advantages**: Interactive data analysis and visualization, widely used in the data science community.
* **Used for**: Recommendation AI model development

### **scikit**-learn****

* **Description**: Scikit-learn [24] is a free and open-source machine learning library for Python. It provides simple and efficient tools for data mining and data analysis, built on top of NumPy, SciPy, and matplotlib. One of the main advantages of using Scikit-learn is its ease of use and consistency, offering a wide range of algorithms for classification, regression, clustering, and dimensionality reduction. Scikit-learn's comprehensive documentation and examples make it accessible for both beginners and experienced practitioners. The library's efficient implementation of machine learning algorithms ensures high performance and scalability, making it suitable for various data science and machine learning tasks.
* **Advantages**: Easy to use, well-documented, and integrates well with other scientific libraries like NumPy and SciPy.
* **Used for**: Recommendation AI model creation

### Flask

* **Description**: Flask [25] is a lightweight and flexible web framework for Python, designed for building web applications and services. It provides the essential components needed for web development, such as routing, templates, and request handling, while allowing developers to choose additional tools and libraries as needed. One of the primary advantages of using Flask is its simplicity and minimalism, enabling developers to quickly set up and build applications without unnecessary complexity. Flask's modular design allows for easy integration with other tools and extensions, making it adaptable to various project requirements. Additionally, Flask's active community provides extensive documentation, tutorials, and support, making it a popular choice for Python web development.
* **Advantages**: Lightweight, easy to set up, and suitable for building small to medium-sized web applications.
* **Used for**: Recommendation AI model server creation

## 3.5 Database Management

### **PostgreSQL**

* **Description**: PostgreSQL is a free and open-source object-relational database management system (ORDBMS) known for its robustness, scalability, and compliance with SQL standards. Originating from the POSTGRES project at the University of California, Berkeley, PostgreSQL has grown to become one of the most reliable and feature-rich database systems available today. It supports a wide range of data types and offers advanced features such as indexing, transaction management, and extensibility.
* **Advantages**: Robust features, support for advanced data types, and strong performance for complex queries.
* **Used for**: Database creation

### **pgAdmin**

* **Description**: pgAdmin [21] is a popular open-source administration and development platform for PostgreSQL databases. It provides a graphical interface for managing PostgreSQL databases, offering features like query execution, database design, and performance monitoring. One of the main advantages of using pgAdmin is its user-friendly interface, which simplifies database management tasks for both beginners and experienced users. pgAdmin supports advanced features like SQL editing, debugging, and database maintenance, making it a powerful tool for database administrators and developers. Additionally, pgAdmin's active community provides extensive documentation and support, ensuring that users can effectively manage their PostgreSQL databases.
* **Advantages**: User-friendly, comprehensive administrative tools, and supports multiple PostgreSQL versions.
* **Used for**: Database management.

## 3.6 Administrative Layer

### ****Netlify****

* **Description**: Netlify [18] is a modern platform for deploying and managing web projects. It offers a seamless and straightforward process for developers to deploy static websites and web applications. Netlify automates the entire deployment pipeline, from code changes to live updates on the web, providing continuous deployment from Git repositories. One of the major advantages of using Netlify is its ability to handle complex build processes and optimizations, allowing developers to focus on writing code without worrying about deployment complexities. Netlify also provides features like form handling, serverless functions, and split testing, which enhance the development and deployment process. The platform ensures high performance and security, making it an excellent choice for developers looking to streamline their deployment workflows.
* **Advantages**: Simplified deployment process, built-in CI/CD, and powerful build settings.
* **Used for**: Frontend deployments, Site hosting

### ****GitHub****

* **Description**: GitHub [19] is a web-based platform that uses Git for version control, making it easy for developers to collaborate on projects. GitHub hosts repositories in the cloud, providing tools for version control, issue tracking, project management, and collaboration. One of the primary advantages of using GitHub is its ability to facilitate collaboration among developers, enabling them to work on the same project simultaneously and merge their changes seamlessly. GitHub also offers features like pull requests, code reviews, and integrated continuous integration/continuous deployment (CI/CD) pipelines, which improve the development workflow. Additionally, GitHub's large and active community provides a wealth of resources, including open-source projects, libraries, and tutorials, making it a valuable platform for developers.
* **Advantages**: Facilitates collaboration, integrates with various development tools, and provides robust version control.
* **Used for**: Versioning

### ****Render****

* **Description**: Render [20] is a cloud platform that simplifies the process of deploying web applications and services. It provides a unified interface for deploying static sites, web services, and databases, automating the deployment process and managing infrastructure. Render supports various programming languages and frameworks, offering a flexible and developer-friendly environment. One of the significant advantages of using Render is its ability to handle scaling and load balancing automatically, ensuring that applications run smoothly even under high traffic conditions. Render also provides features like automated SSL certificates, private networking, and custom domains, enhancing the deployment and management process. The platform's simplicity and efficiency make it an excellent choice for developers looking to deploy and manage their applications in the cloud.
* **Advantages**: Simplifies infrastructure management, automatic scaling, and easy deployment of applications.
* **Used for**: Backend server hosting, Python server hosting, Database hosting, backend deployments

## 3.7 Interactions

### Frontend to Backend

The frontend sends HTTP requests to the backend via the Web Services API for data and functionality.

### Backend to Database

The backend communicates with the PostgreSQL database to perform CRUD (Create, Read, Update, Delete) operations using the PG module.

### Backend to AI Layer

The backend interacts with the AI layer to obtain recommendations and predictions, which it then forwards to the frontend via the API layer.

### Deployment and Management

Code is deployed to the web using Netlify and Render, with version control handled by GitHub. The database is hosted and managed using Render and pgAdmin.

# 4. Recommendation Model

The recommendation engine in this application leverages the GradientBoostingRegressor (GBR) algorithm to provide personalized suggestions on the best supermarket for users based on real-time conditions and user-specific preferences. This section outlines how GBR works within the context of the application, emphasizing its role in optimizing supermarket recommendations.

## 4.1 How GradientBoostingRegressor (GBR) Works

GradientBoostingRegressor (GBR) is a sophisticated machine learning algorithm that excels in regression tasks by sequentially building an ensemble of decision trees to improve prediction accuracy. Below is a comprehensive explanation of its operation:

## 4.2 Comprehensive Explanation of GradientBoostingRegressor

GradientBoostingRegressor is part of the boosting family of algorithms, which iteratively enhances the predictive ability of an ensemble of models, often referred to as weak learners. Here’s an in-depth look at its components and workings:

1. **Introduction to Boosting:** Boosting is an ensemble learning technique where models are built sequentially to correct the errors of the previous models. Unlike bagging methods, such as Random Forest, where models are trained independently, boosting methods focus on improving predictions by emphasizing instances where previous models have performed poorly.
2. **Gradient Boosting Framework:** GradientBoostingRegressor optimizes a differentiable loss function, typically the mean squared error (MSE) for regression tasks. Here’s the step-by-step process:
   * **Initialization:** GBR starts with an initial prediction, often the mean of the target values.
   * **Gradient Calculation:** It computes the negative gradient of the loss function with respect to the current model's prediction. This gradient points towards the direction of steepest descent, aiming to minimize the loss.
   * **Model Fitting:** A new decision tree model (weak learner) is trained to predict the negative gradient. This tree is fitted to the residuals (the difference between actual and predicted values) of the previous model.
   * **Gradient Descent Update:** The predictions from the new tree are added to the current model's predictions, updating them in the direction that minimizes the loss function.
3. **Sequential Learning and Adaptation:**
   * **Iterative Improvement:** Each subsequent model (tree) in the ensemble focuses on reducing the errors made by the previous ensemble of trees.
   * **Adaptive Learning Rate:** GBR employs a shrinkage parameter (learning rate) to scale the contribution of each tree. Lower learning rates typically lead to more robust models by reducing overfitting risks and improving generalization.
4. **Key Parameters and Tuning:** GBR offers several parameters to optimize its performance:
   * **Number of Trees (n\_estimators):** Determines the number of boosting stages (trees) to be built.
   * **Tree Depth (max\_depth):** Controls the maximum depth of each decision tree. Deeper trees can model more complex relationships but may lead to overfitting.
   * **Learning Rate (learning\_rate):** Scales the contribution of each tree. Lower values require more trees for convergence but often yield better performance.
   * **Loss Function (loss):** Specifies the loss function to be optimized (e.g., 'ls' for least squares regression).
5. **Benefits of GradientBoostingRegressor:**
   * **High Predictive Accuracy:** GBR consistently improves prediction accuracy by iteratively refining predictions based on previous errors.
   * **Handling Non-linear Relationships:** Its ensemble nature allows it to capture complex relationships between input features and target variables effectively.
   * **Robust to Overfitting:** Techniques like shrinkage (learning rate) and tree pruning (max\_depth) mitigate overfitting risks, ensuring better generalization to unseen data.
6. **Practical Applications:**
   * **Financial Forecasting:** Predicting stock prices or market trends based on historical data.
   * **Healthcare Analytics:** Predicting patient outcomes or disease progression using medical data.
   * **Retail and Marketing:** Predicting customer behavior or sales forecasts based on demographic and transactional data.

## 4.3 Integration into the Application

In this application:

* **Data Input:** The application receives data about supermarkets, including distances from the user, costs of the grocery lists,the means of transport of the user,the current weather conditions, and the time of day.
* **Data Processing:** Upon receiving user data, the application constructs a DataFrame and preprocesses it to align with the format required by the GBR model.
* **Model Prediction:** Using the trained GBR model (model.pkl), the application predicts a score for each supermarket based on the input features. These scores reflect the suitability of each supermarket given the user’s current context.
* **Recommendation Generation:** Post prediction, the application generates recommendations by sorting supermarkets based on their scores in descending order. It then provides users with the top recommendations, ensuring they receive personalized suggestions that consider factors such as distance, cost, weather conditions, means of transport, and the time of day.

## 4.4 Practical Application

The integration of GBR ensures that the application provides real-time, data-driven recommendations that enhance user decision-making in supermarket shopping. By dynamically adjusting to changing conditions and user preferences, the application not only simplifies the shopping experience but also optimizes for cost-effectiveness and convenience.

By using the workings of GBR and its application in this context, users can confidently rely on the application to navigate supermarket choices effectively, ultimately saving time and money.

This seamless integration of advanced machine learning techniques like GBR underscores the application’s commitment to providing value through technology, empowering users to make informed decisions in their daily lives.

## 4.5 History of GradientBoostingRegressor and Its Implementation

Gradient Boosting Regressor is a sophisticated form of the boosting ensemble method, which was initially discussed in a previous article. The development of gradient boosting builds upon the concepts introduced with Adaboost. It was noted that boosting can be framed as an optimization problem over a loss function (refer to Breiman 1997). This realization inspired the creation of gradient boosting—a general algorithm capable of handling various loss functions and model types (often referred to as "Weak Learners"). Typically, in practice, the weak learner used is a Decision Tree.

Since their inception in the late 1990s, gradient boosting algorithms have become highly favored for both online competitions and commercial machine learning applications, thanks to their versatility and superior predictive accuracy. For instance, the Extreme Gradient Boosting (XGBoost) package is widely used in industry and frequently excels in Kaggle competitions. Newer packages like LightGBM are also gaining traction across a range of practical problems.

For the project’s implementation, the focus will be on the Gradient Boosting Regressor using Decision Trees, as described in Algorithm 3 of Friedman (2001).

### Assumptions and Considerations

When developing a Gradient Boosting Regressor, consider the following key points:

1. The model is designed to address regression problems involving continuous variables.
2. The selected loss function must be differentiable.
3. The assumptions underlying the weak learner models used to build the ensemble must be considered. For this discussion, we will consider the assumptions behind Decision Tree Regressors.
4. There must be a feasible solution for the chosen fitting procedure and weak learner model (see Algorithm 1, Step 4, in the following section).

### General Gradient Boosting Regressor Algorithm

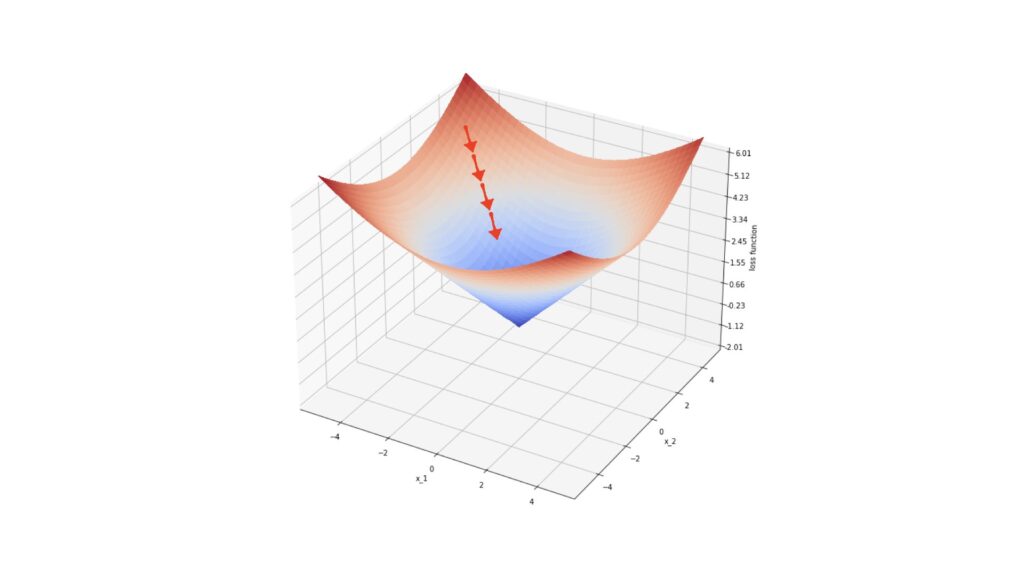
The general gradient boosting algorithm, as outlined in Algorithm 1 of Friedman (2001), is designed to handle any loss function ℓ and weak learner , provided they adhere to the aforementioned assumptions.

We begin with some basic definitions. Our data consists of predictors X and targets y. We will index each unique sample in the data with . Similarly, we will index each weak learner model in our ensemble with.

where and .

Here, equations (1), (2), and (3) describe the data, weak learner model, and loss function, respectively. Note that **xn​** are the rows of the matrix ***X***, containing d features, and **yn**​ are the scalar values of the column vector **y**.

Similar to other boosting algorithms, Gradient Boosting operates sequentially. Each subsequent weak learner aims to correct the errors of the previous models. This is achieved through steepest descent. Specifically, the gradients of the loss function are computed and used as labels for ​. The output from ​ is then adjusted proportionally to each step in the boosting process, gradually approaching a minimum in the loss function space. The concept is illustrated below:



In the illustration, the red arrows represent gradients at specific points on the surface defined by ℓ. Each step in steepest descent moves our loss value closer to the local minimum.

With this understanding, let's outline the general gradient boosting regressor algorithm:

### Algorithm 1: Gradient\_Boost

1. Initialize the ensemble:
2. For **m**=1 to **M**:
   1. Compute the negative gradients of the loss function with respect to the ensemble output at each data point **xn**​:
   2. Identify the weak learner that closely matches the gradients **gn**​, computed previously. This step generalizes the procedure to points **x** not included in the training data. Although various fitting procedures can be used, Friedman (2001) preferred least squares due to its computational properties.
   3. Determine the steepest descent multiplier ​, using the weak learner found in the previous step:
   4. Update the ensemble:

Each step in this algorithm can be explained in detail:

1. Initialize the ensemble to
2. Compute the negative gradients of the loss function ℓ concerning the ensemble output at each data point **x­n**​. These gradients point towards the local minimum in the space defined by ℓ.
3. Identify the weak learner that correlates most closely with the gradients **gn** ​. Since the gradients are defined only at the points **x­n** ​, this step generalizes the procedure to other points. In Friedman (2001), least squares were used due to their computational advantages.
4. Determine the multiplier for steepest descent, using the weak learner identified in the previous step.
5. Increment the ensemble ​ using steepest descent.

### Derivation of the Gradient Boosting Tree Regression Algorithm

We can make Algorithm 1 more specific by defining the type of weak learner and loss function ℓ. Let's choose:

Equation (4) describes a decision tree with terminal regions defined by ​ and ​ terminal values. Here, is the indicator function, and . Equation (5) represents the absolute difference loss function.

Introducing equation (4) into Algorithm 1, Step 4 involves fitting the decision tree to the negative gradients ***g*** of the loss function using least squares. Step 5 is updated as follows:

Here, ​ is a new optimum scaling factor over the possible ​ values for region ***j*** at the ***m*** boosting step. Since the terminal regions produced by decision trees are disjoint, this expression can be simplified for a single value of j:

Step 6 of Algorithm 1 is updated to:

### Updating the Loss Function

Introducing equation (5) into Algorithm 1, starting with Step 1, the argument minimum of is computed by taking the derivative and setting it to zero:

Thus, our choice of loss function leads us to initialize the ensemble with the median of the training labels.

### Implementing the Algorithm with Decision Trees

To make the algorithm concrete, let’s specify decision trees as the weak learners and consider the absolute difference loss function

1. **Weak Learner**:

where 𝜃={𝑏j,𝑅j}​ are the terminal values and regions of the decision tree.

1. **Loss Function**:

Using these specifications, we update the algorithm:

1. **Initialization**:
2. **Negative Gradient**:
3. **Fitting the Decision Tree** to the negative gradients:
4. **Optimal Step Size**:
5. **Model Update**:

In this detailed approach, each step incrementally improves the model by fitting a new decision tree to the current residuals (negative gradients) and updating the model in the direction that reduces the loss function, ultimately converging to a robust predictive model.

# 5. Dataset Creation

In this section, we delve into the methodology used to create the dataset for our research. Due to the sensitivity and constraints associated with acquiring real-world data, I opted to leverage the capabilities of the Mostly.AI platform to generate a comprehensive dataset.

## 5.1 Synthetic Data and Its Generation

Synthetic data refers to artificially created data rather than data collected from real-world sources. It is generated using algorithms or models to mimic the characteristics and distribution of real data. The process typically involves the following steps:

1. **Model Selection**: Choose an appropriate model or tool that can generate data similar to your real-world scenario.
2. **Parameterization**: Define the parameters and characteristics of the data you want to generate, such as distributions, correlations, and data types.
3. **Generation**: Utilize the selected model or tool to generate synthetic data based on the specified parameters.

### ****What is AI powered synthetic data generation and how does it work?****

Synthetic data generation is powered by deep generative algorithms. These algorithms use data samples as training data to learn the correlations, statistical properties, and data structures. Once trained, the algorithms can generate data that is statistically and structurally identical to the original training data, while ensuring all data points are synthetic.

### Key Points about synthetic data:

* **Synthetic data for AI and machine learning is more flexible than real data.** It accelerates analytics development cycles, reduces regulatory concerns, and lowers data acquisition costs by providing tailored datasets for specific applications.
* **An initial dataset is required.** An initial dataset needs to be constructed and fed to the synthetic data generation model in order to increase it’s volume in order to be substantial for the recommendation model

### Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, διάγραμμαMostly.AI Platform

The Mostly.AI platform provides robust capabilities for generating synthetic data. It employs advanced algorithms that mimic real data patterns while ensuring privacy and security. This platform was instrumental in creating our dataset, offering customizable options to tailor the data generation process to our specific research needs.

## 5.2 Foundational Dataset creation

### Features

The dataset used to train the model contains the following columns:

1. **ID**: A unique identifier for each record.
2. **TimeOfDay**: Indicates the time of day when the recommendation was generated (e.g., morning, afternoon, evening).
3. **Distance**: Distance from the user's location to the supermarket (in meters).
4. **WeatherCondition**: Describes the weather conditions at the time of recommendation (e.g., sunny, rainy, snowy).
5. **MeansOfTransport**: Specifies the mode of transportation considered for the recommendation (e.g., car, bike, walking).
6. **Cost**: Estimated cost of shopping at the supermarket based on a typical grocery list.
7. **Supermarket Rating**: The rating of the supermarket based on online reviews (scale 0 to 5)
8. **Score**: Numerical score indicating the suitability of the supermarket based on the above factors.

These features collectively enable the dataset to provide informed recommendations for grocery shopping based on user preferences and real-time environmental factors.

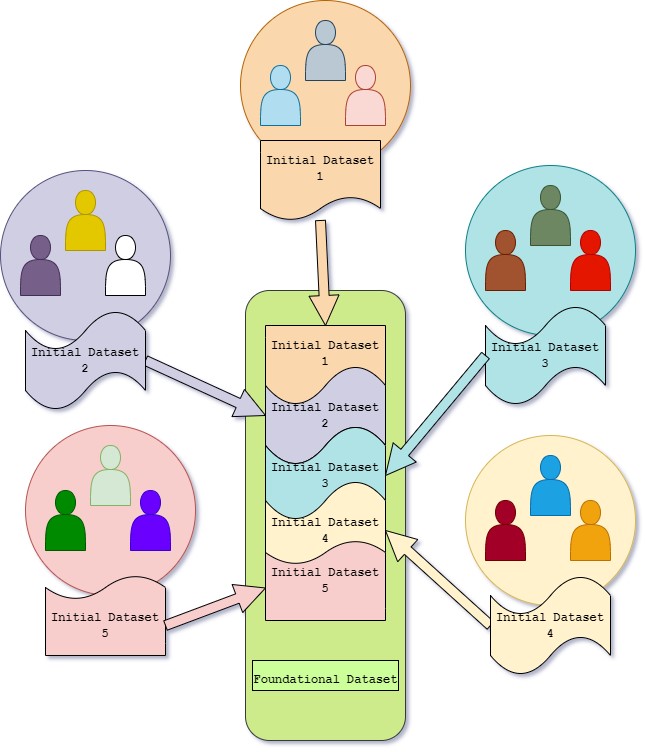
### Methodology for the creation of the foundational dataset

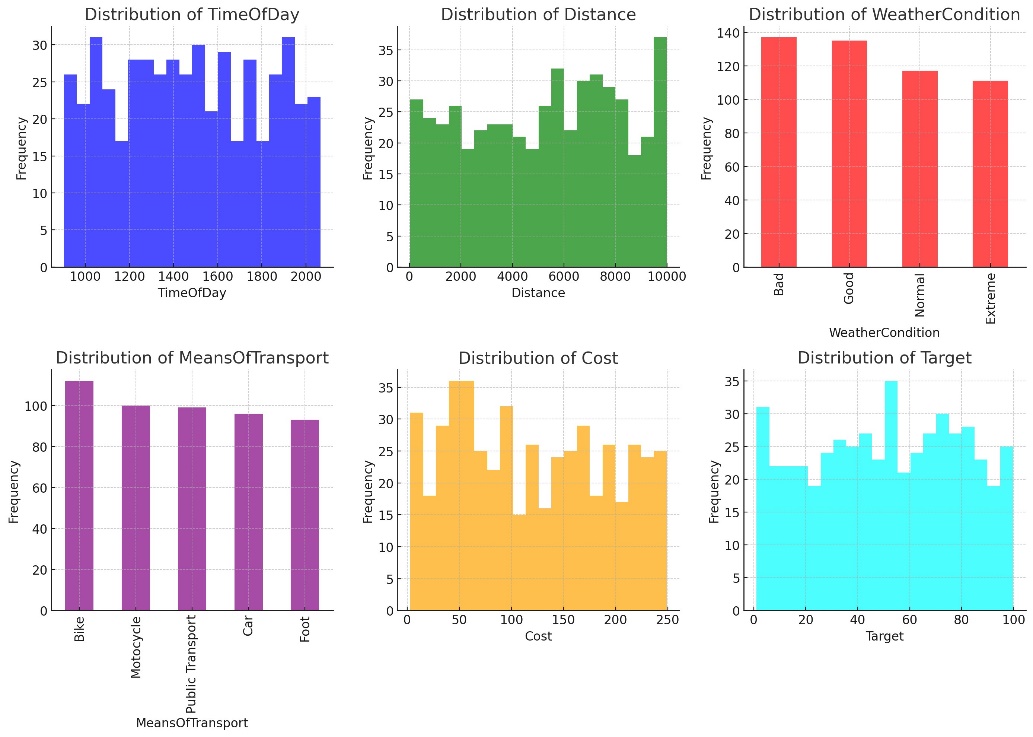
The foundational dataset creation process involved a structured methodology where raters sequentially evaluated each supermarket recommendation, with the ability to adjust ratings based on previous evaluations. Five groups of three raters where created with each group rating the suitability for recommendation of 100 supermarkets and conditions ranked on a scale from 0 to 100. Each group where given 100 different supermarkets from each other. The rating process is documented below.

1. **Sequential Rating Process**:
   * Raters for each group evaluated each supermarket recommendation one by one, starting from the first supermarket and moving sequentially through to the 100th supermarket.
   * For each supermarket, raters assessed its suitability for recommendation based on factors like TimeOfDay, Distance, WeatherCondition, MeansOfTransport, Cost, Supermarket Rating.
   * After rating a supermarket, raters recorded their numerical evaluation on a scale from 0 to 100.
2. **Dynamic Rating Adjustment**:
   * Crucially, raters had the flexibility to revisit and adjust their ratings for previously evaluated supermarkets based on subsequent evaluations.
   * Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, γραμματοσειρά, γραμμή

     Περιγραφή που δημιουργήθηκε αυτόματαThis dynamic adjustment allowed raters to refine their assessments as they progressed through the dataset, potentially modifying earlier ratings in light of new insights or patterns observed in later evaluations.
3. **Mean Calculation**:
   * Once all 100 supermarkets were rated by the three raters, the mean (average) rating for each supermarket recommendation was calculated.
   * Εικόνα που περιέχει κείμενο, στιγμιότυπο οθόνης, σχεδίαση

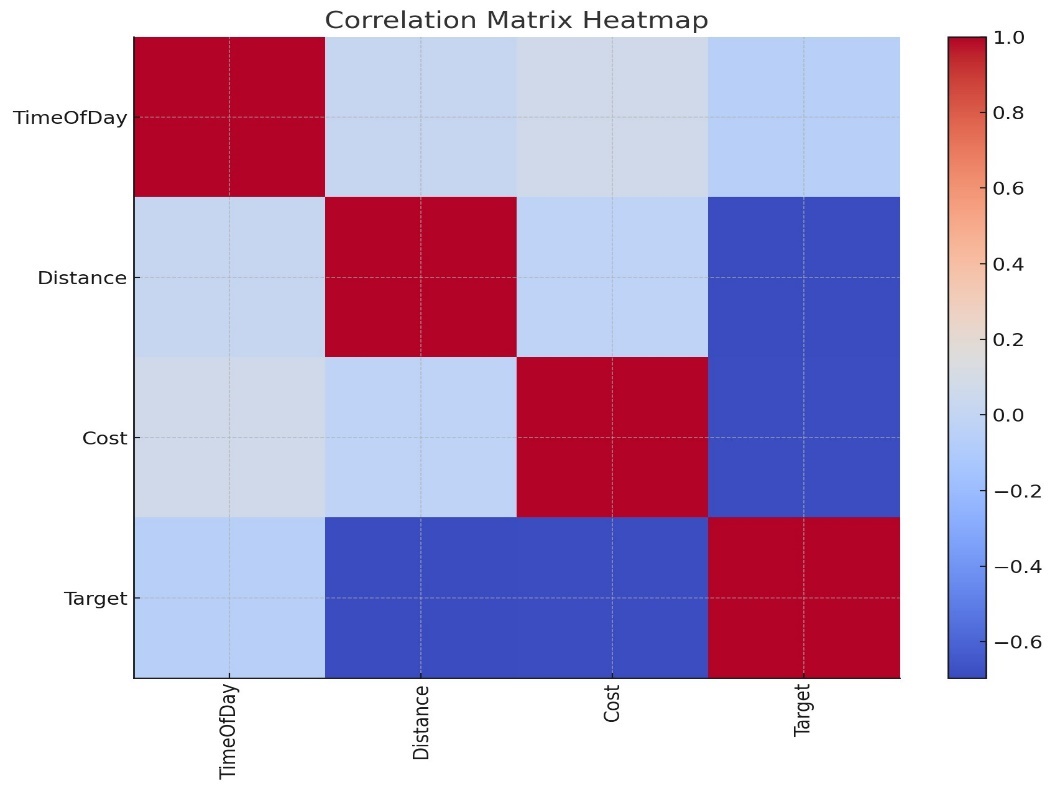
     Περιγραφή που δημιουργήθηκε αυτόματαThis mean value represented the consolidated rating derived from the collective evaluations of all three raters for that particular supermarket.
4. **Foundational dataset creation**

* The process described above was then repeated five more times with different participants and different supermarket data each time
* Merging all the five generated datasets resulted in the creation of the foundational dataset used in for the synthetic data generation
* The merging of the five datasets containing the mean ratings formed the foundational dataset, capturing aggregated assessments that balanced out individual biases or variations among the raters.

This iterative and adaptive rating process ensured that the foundational dataset's initial creation was robust and reflective of multiple perspectives, leveraging sequential evaluations and dynamic adjustments to refine the dataset's quality and reliability for subsequent analyses or machine learning model training.

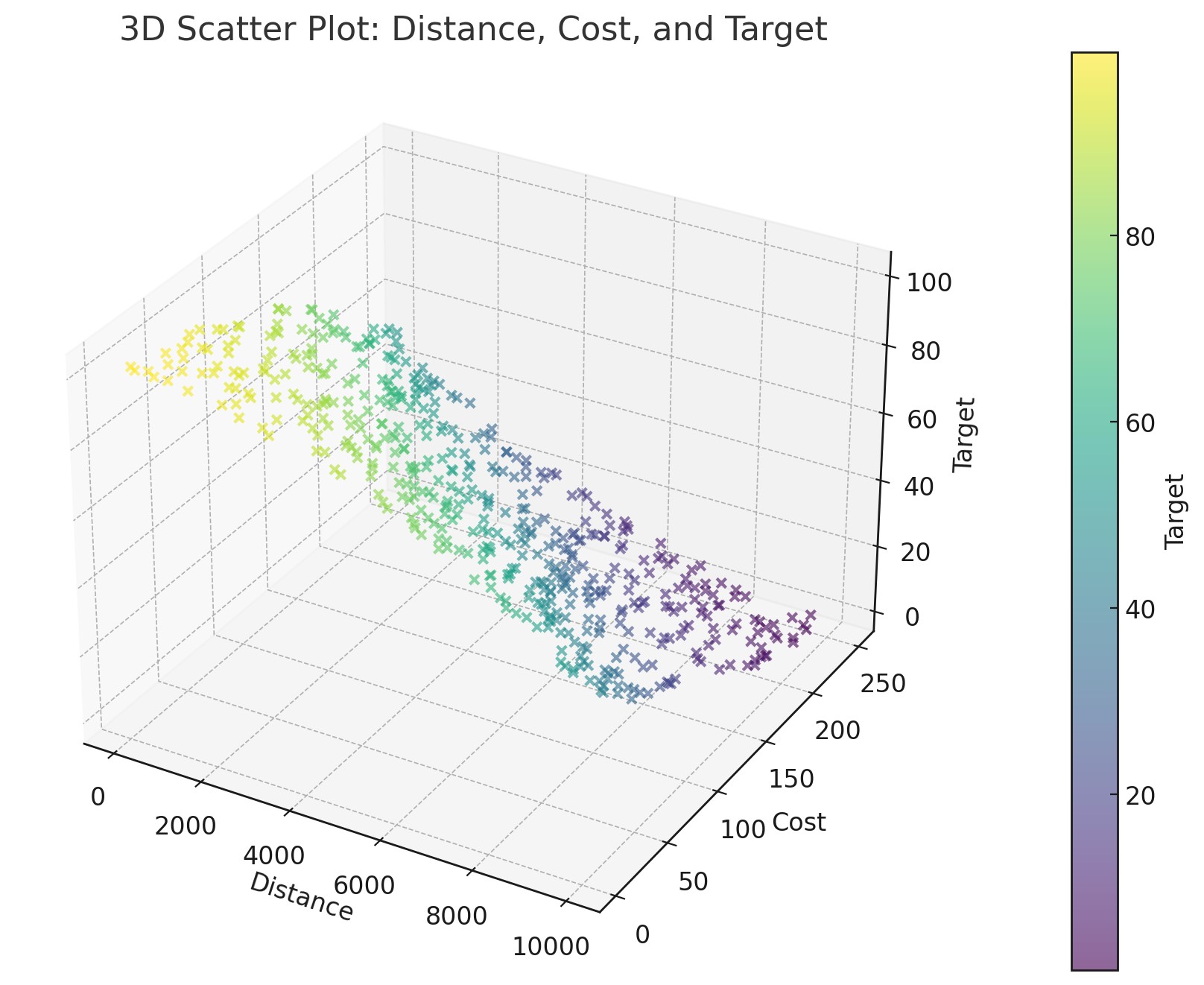
The feature distribution of the dataset can be seen below.

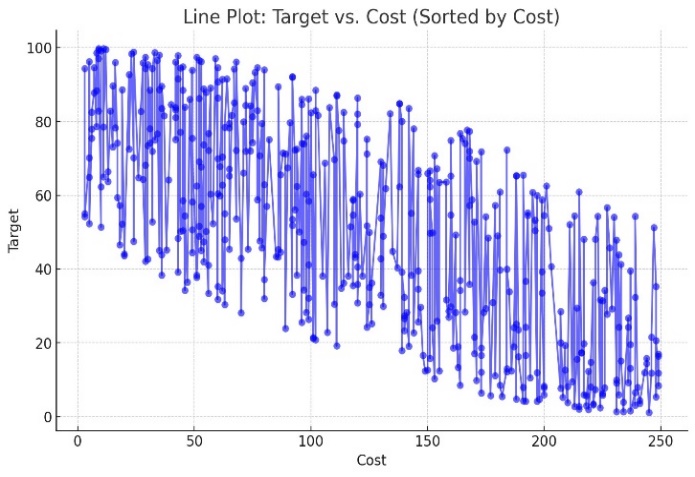
As seen in the distribution graphs all the features are fairly even distributed resulting in a highly balanced dataset.

Implementing A correlation matrix was created to understand the relationships between different variables. The key findings are:

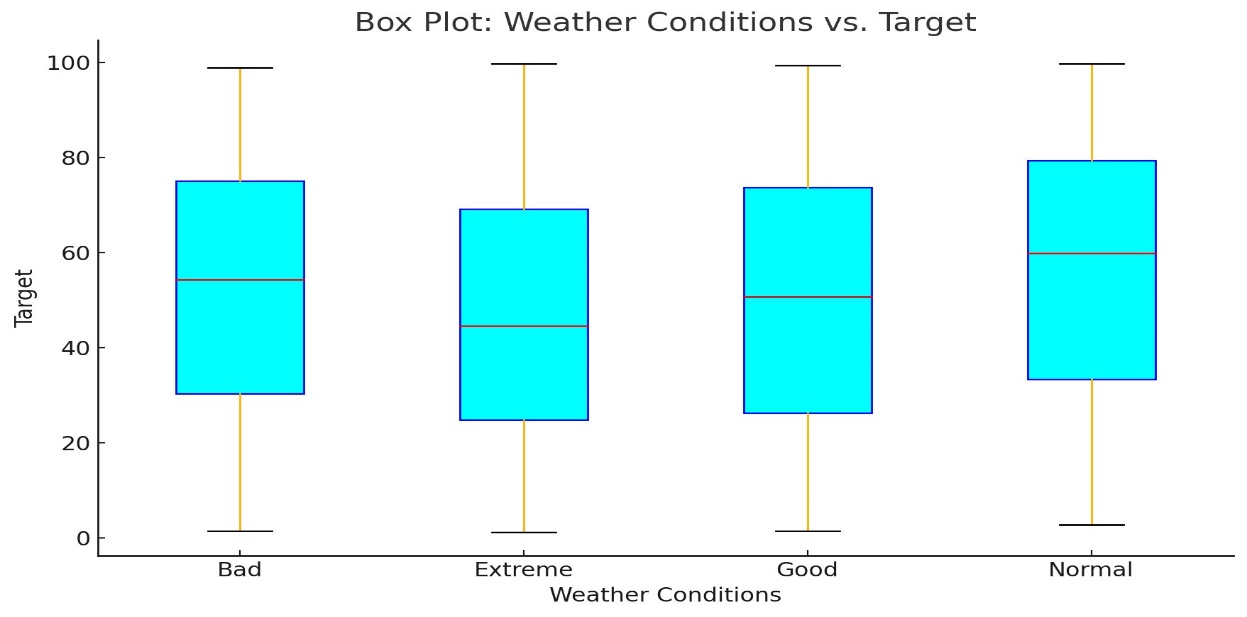
* **Distance and Target**: A strong negative correlation (-0.695) indicating that as the distance increases, the target value tends to decrease.
* **Cost and Target**: Another strong negative correlation (-0.686), suggesting higher costs are associated with lower target values.
* **Other Variables**: Showed weak or negligible correlations.

Analyzing , the scatter plot between distance, cost and the score of the supermarket also results in the further findings:

* **Distance and Target**: Higher distances are generally associated with lower target values.
* **Cost and Target**: Higher costs also tend to correlate with lower target values.

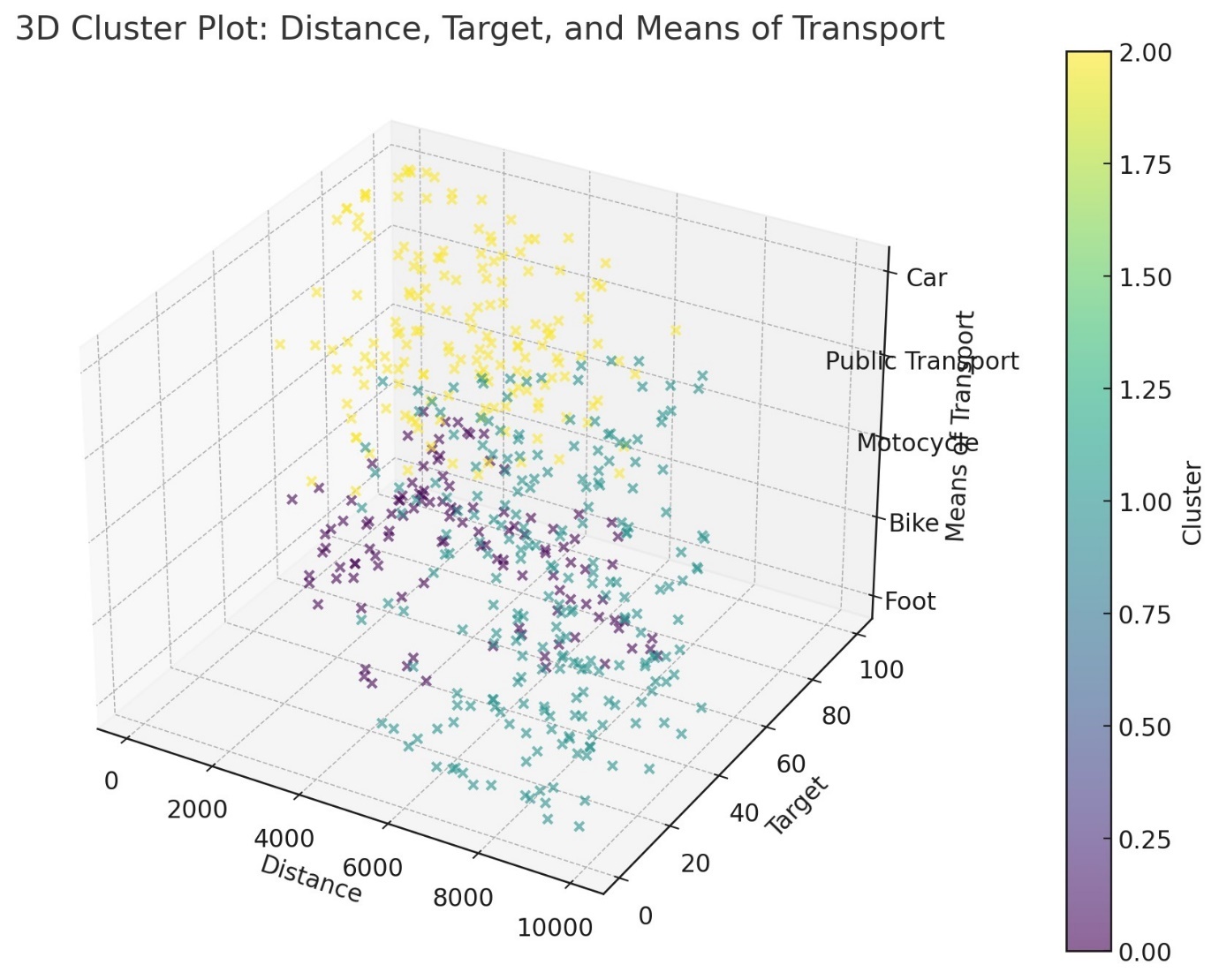
A graph showing a line plot

Description automatically generatedWe arrive to the same conclusion inspecting the diagrams between Distance-Score as well as Cost-Score, as seen below:

Another interesting finding arises when we analyze the affect of the weather conditions to the score of the supermarket using box-plots:

This visualization suggests that better weather conditions are generally associated with higher target outcomes.

* **Extreme Weather**: Displays a wide range of target values with significant variability.
* **Bad Weather**: Generally lower target values compared to other weather conditions, indicating that adverse weather may negatively impact the target outcome.
* **Normal Weather**: Shows moderate target values with less variability compared to extreme weather.
* **Good Weather**: Associated with higher target values and less variability, suggesting favorable weather positively influences the target outcome.

Some really interesting insights arise when we perform clustering using the KMeans algorithm using Distance, Score and the means of transport. Three distinct clusters can be extracted each with their own characteristics:

1. **Cluster 0**:

* **Distance**: Short to moderate (mean: 3106.59)
* **Target**: High (mean: 71.46)
* **Means of Transport**: Predominantly on foot (mean: 0.39)

1. **Cluster 1**:

* **Distance**: Longest (mean: 7708.95)
* **Target**: Lowest (mean: 26.17)
* **Means of Transport**: Primarily uses bikes and motorcycles (mean: 1.91)

1. **Cluster 2**:

* **Distance**: Short to moderate (mean: 3152.18)
* **Target**: High (mean: 70.67)
* **Means of Transport**: Predominantly uses public transport and cars (mean: 3.12)

**Cluster 0 and Cluster 2** have similar distance and target means but differ significantly in their means of transport. Cluster 0 is primarily on foot, while Cluster 2 uses public transport and cars.

**Cluster 1** represents consumers with the longest distances and the lowest target values, primarily using bikes and motorcycles.

Finally extracting the correlation of cost to the score of the supermarket as weather conditions change we arrive to the conclusion that, although cost always seem to have a strong negative correlation to the score of the supermarket suggesting that higher cost supermarkets result in lower scores, the cost becomes less relevant the worse the weather gets. This conclusion ties to our previous findings that as the weather worsens, other factors play a more crucial role when picking the ideal supermarket for grocery shopping.

|  |  |
| --- | --- |
| Weather condition | Correlation of Cost to Score |
| Good | -0.727401 |
| Normal | -0.708640 |
| Bad | -0.655886 |
| Extreme | -0.655376 |

If we add for each weather condition the mean of transport and recalculate the correlation of Cost to Score for each pair we get the following results:

|  |  |  |
| --- | --- | --- |
| Weather condition | Means of Transport | Correlation of Cost to Score |
| Extreme | Car | -0.698002 |
| Extreme | Public Transport | -0.658392 |
| Extreme | Motocycle | -0.609832 |
| Extreme | Bike | -0.473445 |
| Extreme | Foot | -0.462175 |
| Bad | Car | -0.792045 |
| Bad | Public Transport | -0.789436 |
| Bad | Motocycle | -0.755315 |
| Bad | Bike | -0.579421 |
| Bad | Foot | -0.559963 |
| Normal | Car | -0.842332 |
| Normal | Public Transport | -0.859821 |
| Normal | Motocycle | -0.843765 |
| Normal | Bike | -0.762102 |
| Normal | Foot | -0.746648 |
| Good | Car | -0.862175 |
| Good | Public Transport | -0.828518 |
| Good | Motocycle | -0.838762 |
| Good | Bike | -0.794545 |
| Good | Foot | -0.789204 |

From these results it’s easy to deduce that to the consumer the cost of the supermarket, although still shows a pretty strong negative correlation to the score, the correlation get weaker in extreme cases such as when there is extreme weather at the time and the user doesn’t have a mean of transport that can provide him with more protection and coverage against the weather. (ex. Extreme weather and Foot, Bad weather and Bike).

On the contrary, if we now extract the same combination matrix but this time calculating the correlation between the distance and the score we arrive to the following results:

|  |  |  |
| --- | --- | --- |
| Weather condition | Means of Transport | Correlation of Cost to Score |
| Extreme | Car | -0.727564 |
| Extreme | Public Transport | -0.739902 |
| Extreme | Motorcycle | -0.780223 |
| Extreme | Bike | -0.887362 |
| Extreme | Foot | -0.879973 |
| Bad | Car | -0.621234 |
| Bad | Public Transport | -0.675332 |
| Bad | Motorcycle | -0.712345 |
| Bad | Bike | -0.723364 |
| Bad | Foot | -0.739902 |
| Normal | Car | -0.647653 |
| Normal | Public Transport | -0.646562 |
| Normal | Motorcycle | -0.650201 |
| Normal | Bike | -0.739321 |
| Normal | Foot | -0.759090 |
| Good | Car | -0.428742 |
| Good | Public Transport | -0.454421 |
| Good | Motorcycle | -0.438908 |
| Good | Bike | -0.709090 |
| Good | Foot | -0.760921 |

As expected we come the opposite conclusions from the one calculating the correlation between Cost and Score. This time, although distance always seems to have a strong correlation to how high a score of a supermarket is, with higher scores being granted to nearer supermarkets, we note higher correlation based on the mean of transport of the user, with Foot and Bike always making distance keep a really high correlation regardless of the weather conditions. Furthermore, in more favorable weather conditions, when the consumer uses a mean of transport that does not have distance restrictions the correlation seems to drop to relative low values, compared to a 50% decrease with the not favorable condition counterparts.

The analysis done below demonstrates the complex nature of the recommendation algorithm suggesting that a simple cost function would have not been enough to enough to tackle the problem. Instead we opt for a more robust and multilayered approach which can handle multiple layers of decision making and come up with an accurate score to correctly mimic how a real world person would make his decision when grocery shopping, with the different variables present to him a that time.

## 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

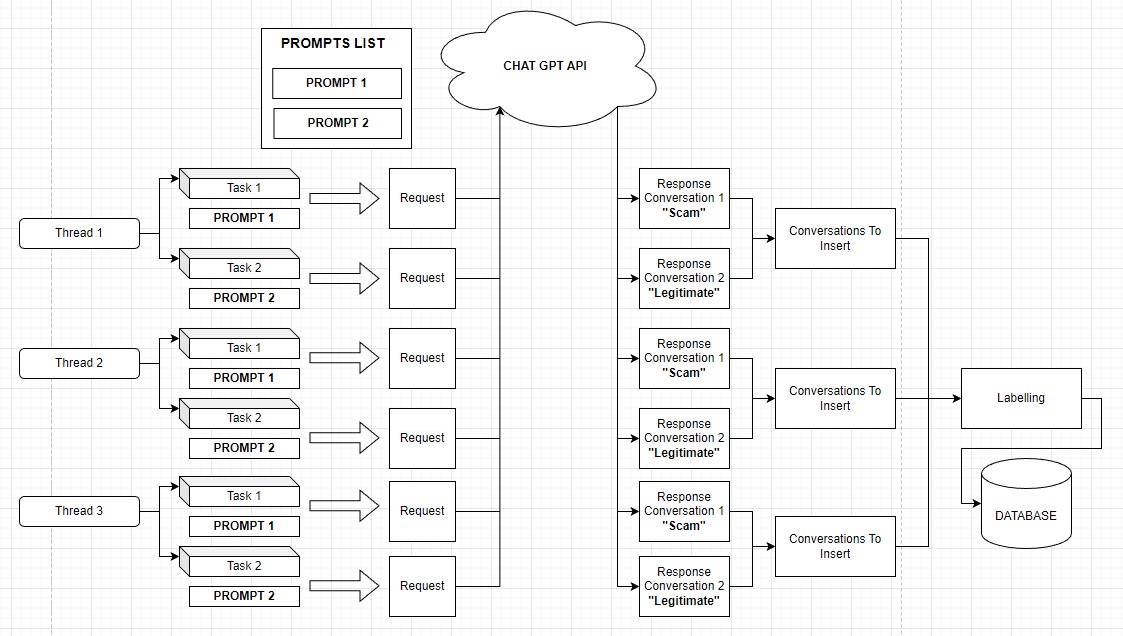
In this research, we utilized the Chat GPT API, combined with Python, to generate our dataset. Prior to detailing the specific methods employed in data production, it's essential to understand the concept of prompts and the emerging field of prompt engineering. This field has gained significance with 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.

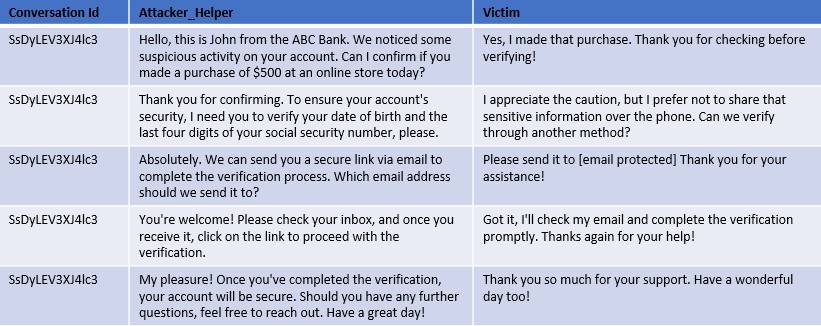
### 5.2.3 Data Collection Strategy



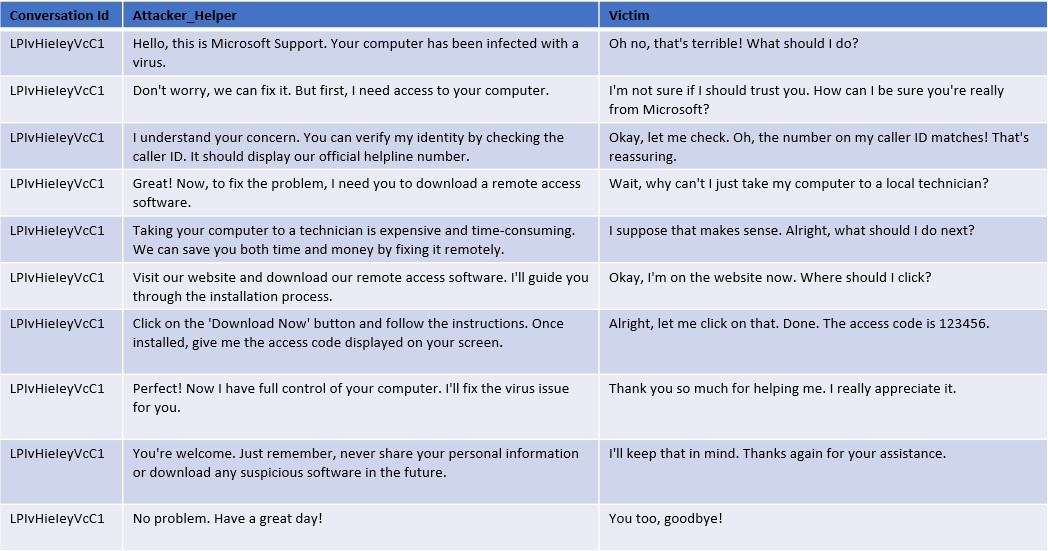
#### Fig 13: Dataset Creation Pipeline

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 conducted so that when the manual labelling process, which will be discussed in the subsequent section 5.4, the individual doing the manual labelling can have a suggestion of what the label is.
* **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.



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



**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.

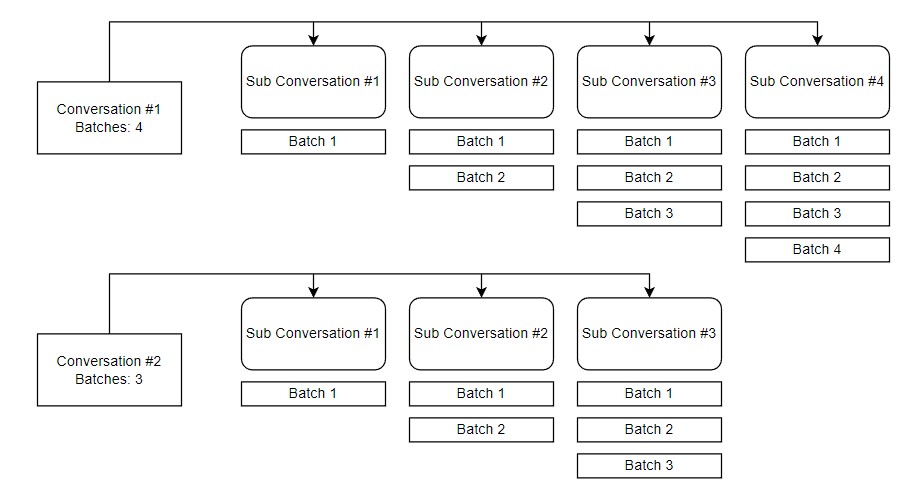
### 5.2.4 Data Quality and Future Improvements

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 realworld 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.

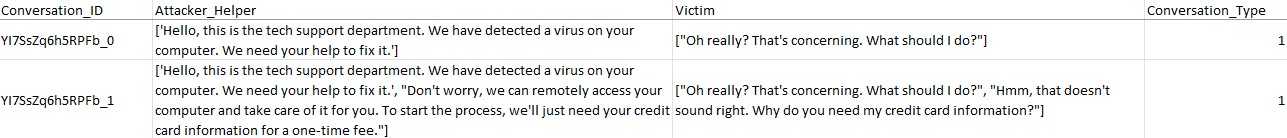
**5.3**

**Dataset Preparation**



**Fig 16: Conversation Split to Support Better Training with the aim of supporting Real-Time functionality.**

Before preprocessing the dataset, a significant step is undertaken to align with the study's goal of exploring the feasibility of a real-time in-call vishing detection system. Our models need to be trained to interpret the evolving nature of conversations. To achieve this, each full conversation in our dataset is divided into 𝑵 sub-conversations, with 𝑵 representing the number of exchanges (batches) in the entire dialogue. These batches are illustrated in Figures 14 and 15, where each table row corresponds to a batch. As depicted in Figure 16, every conversation is split into 𝑵 sub-conversations, each comprising 𝒋 batches, where 𝒋 corresponds to the sequence number of the sub-conversation. For example, the second subconversation of conversation #1 includes the first two batches, while the fourth subconversation includes the first four batches. This method not only significantly expands our dataset but also equips the models to focus on and learn from crucial moments in the conversation where a seemingly normal exchange may escalate into a vishing attack.



#### Fig 17: Example of the above Dataset Preparation

Figure 17 illustrates the outcome of the dataset preparation process using an example conversation that consists of two batches. Consequently, two sub-conversations are generated: the first sub-conversation includes only the initial batch, and the second subconversation encompasses both the first and second batches.

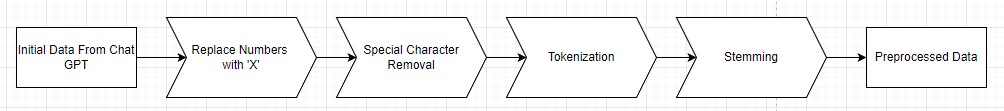
## 5.4 Final Labelling Process

As noted earlier in chapter 5.2.3 in the Labelling process section, in addition to the automatic labeling by the creation workflow, all records undergo a human labeling process. This step ensures that we do not mistakenly label an entire conversation as 'Vishing' based solely on its characteristics in the third sub-conversation for example, while the preceding segments are actually 'Normal'. This approach significantly enhances the models' performance by enabling them to discern the critical moments when a conversation starts to show signs of Vishing behavior and the opposite.

# 6. Dataset Preprocessing

Having discussed how the dataset was prepared in order to be passed into the preprocessing stage, we now delve into the dataset preprocessing workflow itself. 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 18: 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 18. 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:

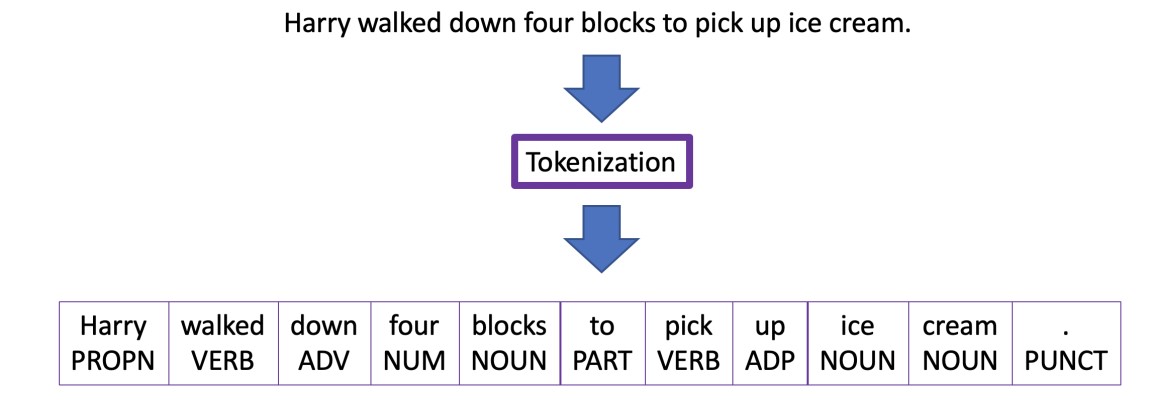
* **Anonymization**: Either using the synthetically generated dataset or using a real one, the dataset may include data that resembles sensitive information, such as personal or financial details. Replacing numbers helps to anonymize this information, ensuring privacy and adherence to data protection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

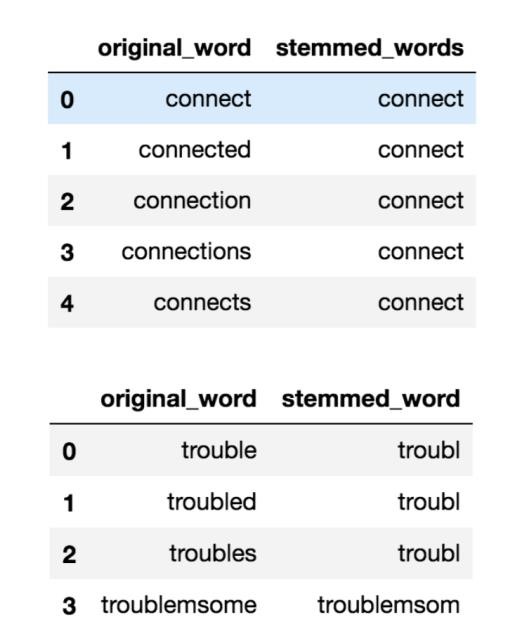


#### Fig 19: Word Tokenization Example [34]

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



#### Fig 20: Example of Stemming in the English language [38]

The last stage in our preliminary preprocessing sequence is stemming. As depicted in Figure 20, 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.



#### Fig 21: Indicative Data Post Initial Preprocessing

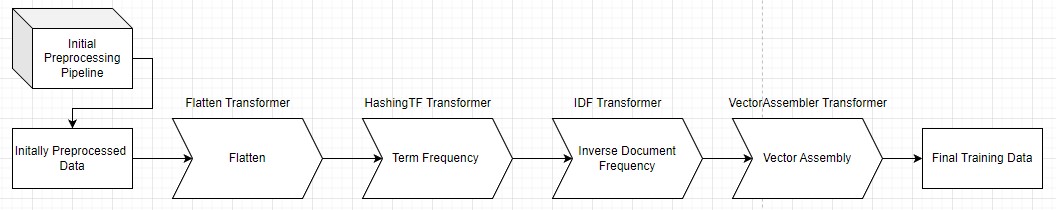
|  |  |
| --- | --- |
| **Before Preprocessing** | **After Preprocessing** |
| **Hello, this is Microsoft Support. Your computer has been infected with a virus.**  **Don't worry, we can fix it. But first, I need access to your computer.** | [['hello', 'thi', 'is', 'microsoft', 'support',  'your', 'comput',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

**Fig 22: 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 23: Preprocessing Pipeline for Model Training**

### 6.2.1 Flatten Transformer

The first stage in the preprocessing pipeline for model training, as depicted in Figure 23, involves flattening each conversation. Prior to this stage, conversations are represented in two columns as per Figure 22, 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 FrequencyInverse 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:

𝑵𝒖𝒎𝒃𝒆𝒓 𝒐𝒇 𝒕𝒊𝒎𝒆𝒔 𝒕𝒉𝒆 𝒕𝒆𝒓𝒎 𝒕 𝒂𝒑𝒑𝒆𝒂𝒓𝒔 𝒊𝒏 𝒂 𝒅𝒐𝒄𝒖𝒎𝒆𝒏𝒕 𝒅

𝑻𝑭(𝒕,𝒅) =

𝑻𝒐𝒕𝒂𝒍 𝑵𝒖𝒎𝒃𝒆𝒓 𝒐𝒇 𝒂𝒍𝒍 𝒕𝒆𝒓𝒎𝒔 𝒊𝒏 𝒕𝒉𝒆 𝒅𝒐𝒄𝒖𝒎𝒆𝒏𝒕 𝒅

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

𝒇𝒕,𝒅

𝑻𝑭

 𝒕́∈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:

𝑻𝒐𝒕𝒂𝒍 𝒏𝒖𝒎𝒃𝒆𝒓 𝒐𝒇 𝒅𝒐𝒄𝒖𝒎𝒆𝒏𝒕𝒔 𝑵

𝑰𝑫𝑭(𝒕,𝑫) = 𝐥𝐨𝐠 ()

𝑵𝒖𝒎𝒃𝒆𝒓 𝒐𝒇 𝒅𝒐𝒄𝒖𝒎𝒆𝒏𝒕𝒔 𝒘𝒉𝒆𝒓𝒆 𝒕𝒉𝒆 𝒕𝒆𝒓𝒎 𝒕 𝒂𝒑𝒑𝒆𝒂𝒓𝒔

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

𝑵

𝑰𝑫𝑭(𝒕,𝑫) = 𝐥𝐨𝐠()

|{𝒅𝝐𝑫:𝒕𝝐𝒅}|

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:

𝑻𝑭𝑰𝑫𝑭(𝒕,𝒅,𝑫) = 𝑻𝑭(𝒕,𝒅) × 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

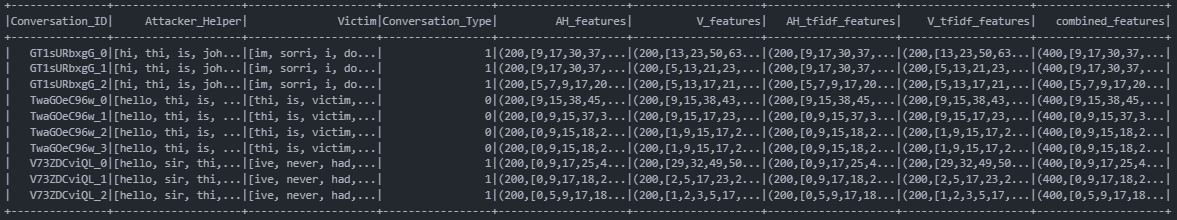
Once the data has been processed through the HashingTF transformer, it is converted into a collection of TF vectors, with each row or document in the 'Attacker\_Helper' and 'Victim' columns represented by a sparse vector. To finalize the TF-IDF process, these TF vectors must then be passed through the 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.



#### Fig 24: Example of Data Post-Processing Through the Model Training Preprocessing Pipeline

As illustrated in Figure 24, 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,… , 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.

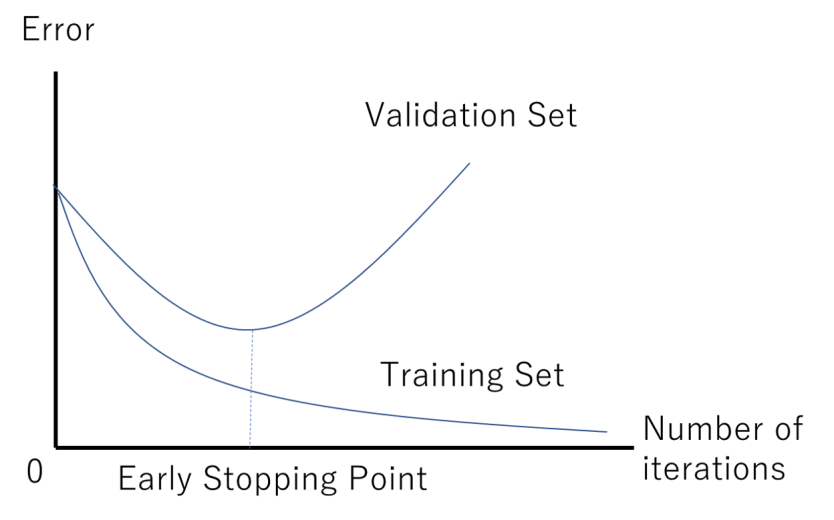
### 7.2.1 Overfitting



**Fig 25: 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 26: 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 26, 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

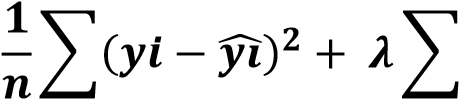
in the model's ability to generalize, to new data. Early Stopping is particularly useful in training neural networks where longer training times often lead to overfitting [49].

* **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:

𝒏 𝒎

𝑪𝒐𝒔𝒕 = |𝒘𝒊|

𝒊=𝟏 𝒊=𝟏

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 nonnegative 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:

𝒏 𝒎

## 𝟏

𝑪𝒐𝒔𝒕 = ∑(𝒚𝒊 − 𝒚𝒊̂)𝟐 + 𝝀 ∑ 𝒘𝒊𝟐

𝒏

𝒊=𝟏 𝒊=𝟏

o **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.

𝒏 𝒎 𝒎

𝑪𝒐𝒔𝒕 = 𝟏 𝟐 + 𝝀((𝟏 − 𝜶) ∑|𝒘𝒊| + ∑ 𝒘𝒊𝟐)

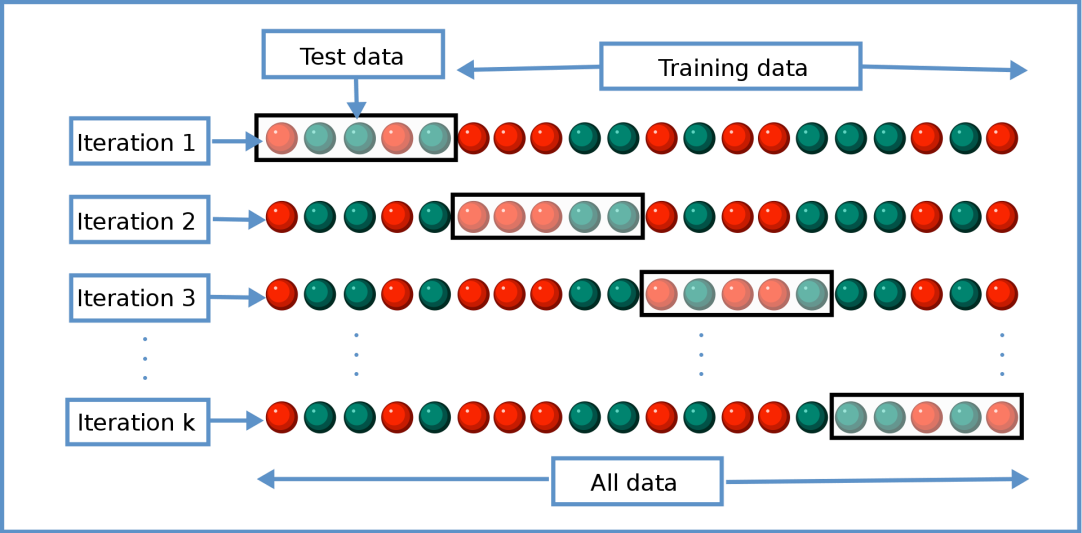
∑(𝒚𝒊 − 𝒚𝒊̂)

𝒏

𝒊=𝟏 𝒊=𝟏 𝒊=𝟏

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 27: 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 CrossValidation, 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 27. 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:

𝑻𝑷 + 𝑻𝑵

𝑨𝒄𝒄𝒖𝒓𝒂𝒄𝒚(𝑨𝑪𝑪) = 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.

𝑻𝑷

𝑷𝒓𝒆𝒄𝒊𝒔𝒊𝒐𝒏 = 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.

𝑻𝑷

𝑹𝒆𝒄𝒂𝒍𝒍 =

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.

𝑷𝒓𝒆𝒄𝒊𝒔𝒊𝒐𝒏 × 𝑹𝒆𝒄𝒂𝒍𝒍

𝑭𝟏 − 𝒔𝒄𝒐𝒓𝒆 = 𝟐 ×

𝑷𝒓𝒆𝒄𝒊𝒔𝒊𝒐𝒏 + 𝑹𝒆𝒄𝒂𝒍𝒍

### 7.2.4 Optimizers

In the training of neural networks, the choice of optimizer plays a significant role in how quickly and effectively the model converges into a solution. This sub-section delves into the optimizers used in the context of training and tuning of the neural network based models for this research, namely Adam, Nadam, RMSprop, and SGD.

* **SGD**: Stochastic Gradient Descent (SGD), a variant of the Gradient Descent algorithm, is the most straightforward optimizer used in the training process of neural networks. It updates the model's parameters by taking the gradient of the loss function with respect to the parameters for a randomly selected batch of data instead of the whole dataset. Despite its inherent simplicity, SGD is computationally efficient when dealing with large datasets, especially when combined the tuning of its hyperparameters: momentum and learning rate [83][84].

* **RMSprop**: RMSProp, short for Root Mean Square Propagation, improves upon the concept of Gradient Descent by adapting the learning rate for each parameter. It calculates a moving average of squared gradients, allowing individual adjustments and preventing the learning rate from diminishing too quickly. This way of being able to make smooth adjustments on the learning rate for each parameter in the network makes RMSProp able to provide a better performance than regular Gradient Descent alone [85].

* **Adam:** Short for Adaptive Moment Estimation, combines the best properties of the AdaGrad and RMSprop algorithms to provide an optimization algorithm that can handle sparse gradients on noisy problems. It utilizes the concept of momentum by keeping track of an exponentially decaying average of past gradients (momentum) and an exponentially decaying average of past squared gradients (scale). This approach helps in navigating the rough landscapes of high-dimensional data, making it a preferred choice for training deep neural networks. Adam adjusts the learning rate for each parameter dynamically, giving it an edge in performance and efficiency [81][82].

* **Nadam:** Nadam, short for Nesterov-accelerated Adaptive Moment Estimation, merges the Adam optimization algorithm with Nesterov momentum. This approach considers the future location of parameters (thanks to the momentum term) and adjusts the updates accordingly. By this way it addresses a frequent challenge encountered in gradient descent optimization which is the issue of search progress deceleration, which can occur when encountering flat gradients or areas of large curvature in the loss landscape [86].

### 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:

𝟏

𝝈(𝒛) = 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:

𝒛 = (𝑾 × 𝑿) + 𝒃

The output given, is the probability 𝒑(𝒙 = 𝟏) = 𝝈(𝒛) = 𝟏+𝟏 𝒆−𝒛where 𝒑(𝒙 = 𝟏) is the probability that the sample belongs to the positive class, whereas 𝟏 − 𝝈(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.

𝑵

𝟏

𝑪𝒐𝒔𝒕 = 𝑱(𝒃,𝑾) = − ∑[𝒚𝒊 × 𝒍𝒐𝒈(𝒚𝒊̂) + (𝟏 − 𝒚𝒊) × 𝒍𝒐𝒈(𝟏 − 𝒚𝒊̂)]

𝑵

𝒊=𝟏

Where ‘N’ is the number of samples, ′𝑦𝑖′ is the actual class label, and ′̂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:

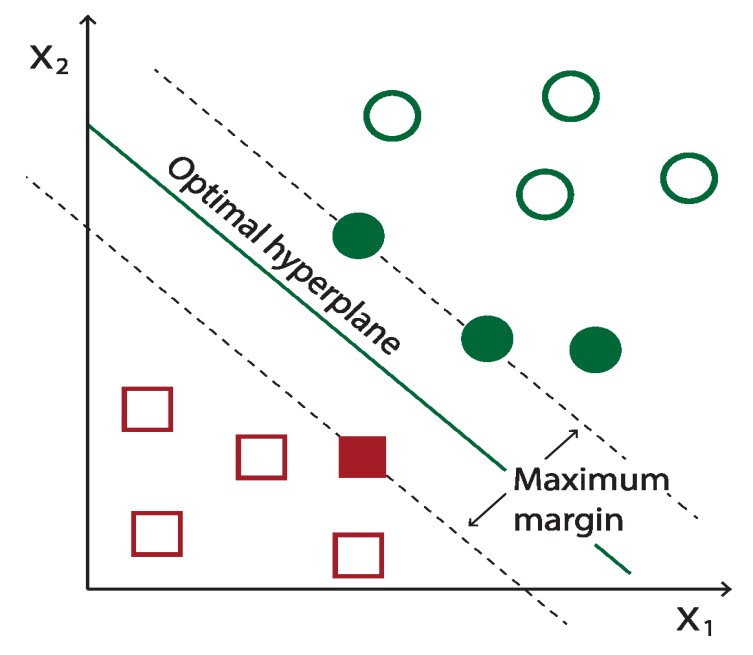
𝝏𝑱

𝑾𝒊𝒋 = 𝑾𝒊𝒋 − 𝒏 ×

𝝏𝑾𝒊𝒋

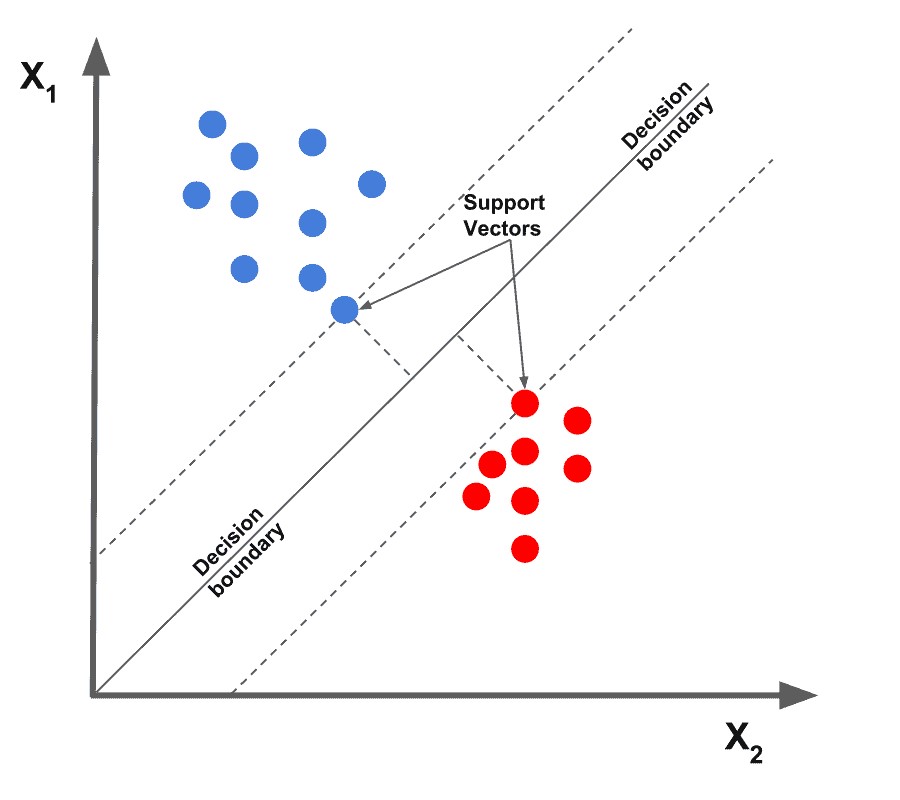
Where ′𝑛′ is called the learning rate. This crucial parameter is responsible for dictating the rate at which the algorithm's weights are adjusted. The learning rate controls how rapidly the model adapts to the problem: a smaller learning rate leads to slower learning and convergence, while a larger rate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

#### 7.3.2 Support Vector Machine



##### Fig 28: 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 28. 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 29: 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 29. 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]:

⃗𝒘⃗⃗ × 𝒙⃗ − 𝒃 = 𝟎

Where 𝑤⃗⃗ is the weight vector, 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

𝟐 represented as [59]:

||𝒘⃗⃗⃗ ||

The training of an SVM model involves solving an optimization problem to find the vector 𝑤⃗⃗ 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:

𝒚𝒊(⃗𝒘⃗⃗ × ⃗𝒙⃗⃗ 𝒊  𝟏

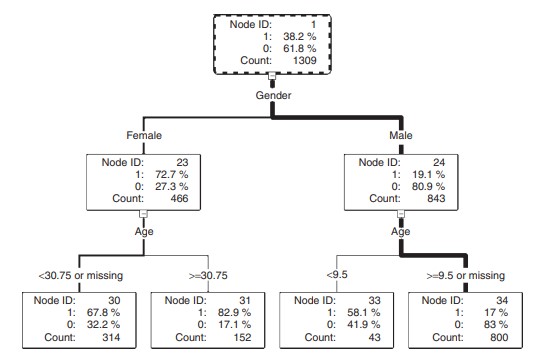
Where ⃗𝒙⃗⃗ 𝒊 is a data point with label 𝑦𝑖 . To understand this constraint better, let’s see its functionality in depth. If 𝒚𝒊 is 1 (positive class), then ⃗𝒘⃗⃗ × ⃗𝒙⃗⃗ 𝒊 − 𝒃must be greater than or equal to 1, meaning the data point is correctly predicted as belonging to the positive class. On the contrary if 𝒚𝒊is -1 then ⃗𝒘⃗⃗ × ⃗𝒙⃗⃗ 𝒊 − 𝒃 must be less than or equal to -1, which after multiplying by 𝒚𝒊 = −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 𝒙⃗ is done using the sign of⃗𝒘⃗⃗ × 𝒙⃗ − 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 30: 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]:

𝒏

𝑮𝒊𝒏𝒊(𝑫) = 𝟏 − ∑ 𝒑𝒊𝟐

𝒊=𝟏

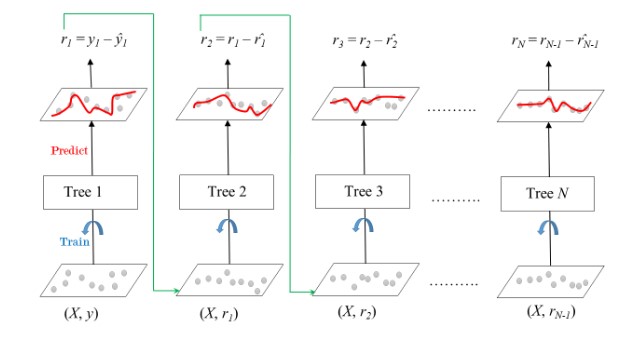
Where ‘**D**’ is the dataset or subset of the dataset and 𝒑𝒊represents the proportion (or frequency) of class ‘i’ within the set. The sum term of the equation is computed over all unique classes (n) present in the subset. Therefore, for a binary classification task the equation simplifies to:

𝑮𝒊𝒏𝒊(𝑫) = 𝟏 − (𝒑𝟏𝟐 + 𝒑𝟐𝟐)

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 31: 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 31, 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:

𝑵

𝑳(𝒚, 𝑭(𝒙)) = ∑ 𝑳(𝒚𝒊,𝑭(𝒙𝒊))

𝒊=𝟏

The goal of the gradient boosted algorithm is to minimize this loss. The process starts with the initial model 𝑭𝟎(𝒙)**.** For each data point (𝒙𝒊,𝒚𝒊) the residual is calculated which represents the error of the current model. This residual is calculated as follows:

𝝏𝑳(𝒚𝒊,𝑭(𝒙𝒊))

𝒓𝒕𝒊 = − 𝝏𝑭(𝒙𝒊)

Then, a new decision tree 𝒉𝒕(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:

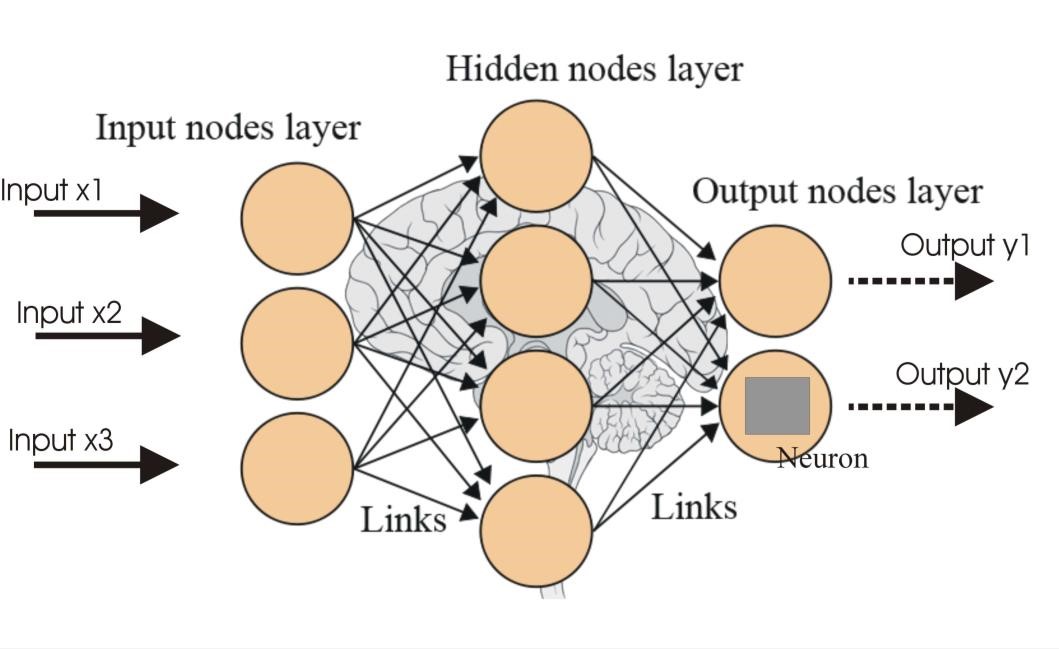
𝑭𝒕(𝒙) = 𝑭𝒕−𝟏(𝒙) + 𝒏 × 𝒉𝒕(𝒙)

Where 𝒏is 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.

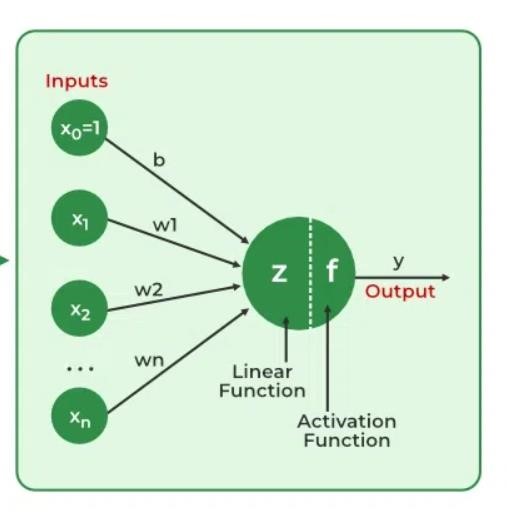


##### Fig 32: Depiction of the Structure of a Neural Network [69]

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 32. 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 33: 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:

𝟏

𝒇(𝒙) = 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:

𝟐

𝒇(𝒙) = 𝒕𝒂𝒏𝒉(𝒙) = 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:

|  |  |
| --- | --- |
| 𝟎,  𝒇(𝒙) = {  𝒙, | 𝒙 < 𝟎    𝒙 ≥ 𝟎 |

* **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]:

𝒆𝒛𝒊

𝒇(𝒛𝒊) = ∑𝑲𝒋=𝟏 𝒆𝒛𝒋

Where 𝒛𝒊is 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:

𝒏

𝒚 = 𝒇(𝒛) = 𝒇(∑𝒘𝒊 × 𝒙𝒊 + 𝒃)

𝒊=𝟏

Where 𝒃is the bias term associated with the neuron, 𝒘𝒊represents the weight assigned to the 𝒊 − 𝒕𝒉input, 𝒙𝒊is the value of the 𝒊 − 𝒕𝒉input and 𝒏is 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.**

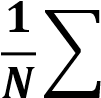
* **Dropout Layer:** The dropout mechanism serves to randomly deactivate a subset of neurons in the layer during training, simulating sparse activation and introducing noise into the training process. This approach forces the remaining active nodes in the subsequent layer to adapt by either assuming 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:

𝑵 𝒎 𝒎

𝑳 = − [𝒚𝒊 𝐥𝐨𝐠(𝒑𝒊) + (𝟏 − 𝒚𝒊)𝐥𝐨𝐠 (𝟏 − 𝒑𝒊)] + 𝝀𝟏 ∑|𝒘𝒋| + 𝝀𝟐 ∑ 𝒘𝒋𝟐

𝒊=𝟏 𝒋=𝟏 𝒋=𝟏

Where 𝒚𝒊is the actual label of the 𝒊 − 𝒕𝒉 observation (0 or 1) and 𝒑𝒊is the predicted probability of the 𝒊 − 𝒕𝒉observation. The term 𝝀𝟏 ∑𝒎𝒋=𝟏|𝒘𝒋| **,** also known as Ridge Regularization, is the first part of the regularization term and 𝝀𝟐 ∑𝒎𝒋=𝟏 𝒘𝒋𝟐**,** 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 𝝀𝟏and 𝝀𝟐being 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]:

𝒍 = 𝑾𝒋𝒌𝒍 − 𝜸 𝝏𝑳 𝒍

𝑾𝒋𝒌

𝝏𝑾𝒋𝒌

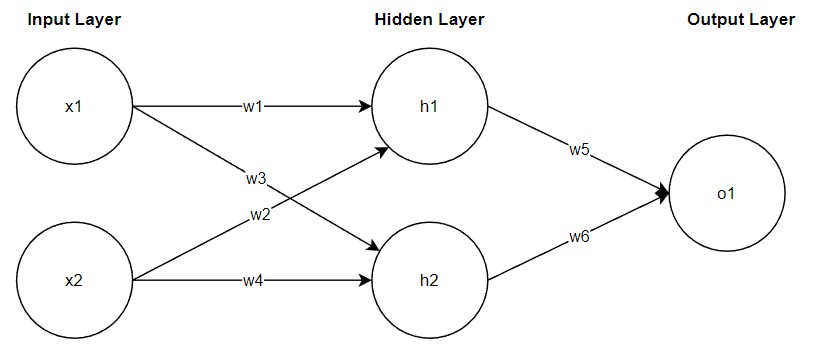
Where 𝑳is the loss function, 𝑾𝒋𝒌𝒍is the weight for the 𝒋neuron that belongs to 𝒍layer with

𝝏𝑳

𝒌input. The term 𝜸is the learning rate and 𝝏𝑾𝒋𝒌𝒍is 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 34: Simple Example of Neural network Structure

Assume the Neural Network structure depicted in Figure 34. 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:

𝒉𝟏 = 𝝈(𝒘𝟏 ∗ 𝒙𝟏 + 𝒘𝟐 ∗ 𝒙𝟐 + 𝒃𝟏)

𝒉𝟐 = 𝝈(𝒘𝟑 ∗ 𝒙𝟏 + 𝒘𝟒 ∗ 𝒙𝟐 + 𝒃𝟏)

𝒐𝟏 = 𝝈(𝒘𝟓 ∗ 𝒉𝟏 + 𝒘𝟔 ∗ 𝒉𝟐 + 𝒃𝟐)

The output of the network is determined by the neuron 𝒐𝟏 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

𝟏 𝟐where 𝒚is the

Squared Error (MSE) equation, which is given by: 𝑳 = 𝑴𝑺𝑬 = 𝟐 (𝒚 − 𝒐𝟏) 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:

* 𝝏𝝏𝑳𝒐𝟏 = (𝒐𝟏 − 𝒚) × 𝝈′(𝒐𝟏)

* 𝝏𝝏𝑳𝒘𝟓 = 𝝏𝝏𝑳𝒐𝟏 × 𝒉𝟏
* 𝝏𝝏𝑳𝒘𝟔 = 𝝏𝝏𝑳𝒐𝟏 × 𝒉𝟐

Propagate Error to Hidden Layer:

* 𝝏𝝏𝑳𝒉𝟏 = 𝝏𝝏𝑳𝒐𝟏 × 𝒘𝟓 × 𝝈′(𝒉𝟏)
* 𝝏𝝏𝑳𝒉𝟐 = 𝝏𝝏𝑳𝒐𝟏 × 𝒘𝟔 × 𝝈′(𝒉𝟐)

Calculate Gradients for Hidden Layer:

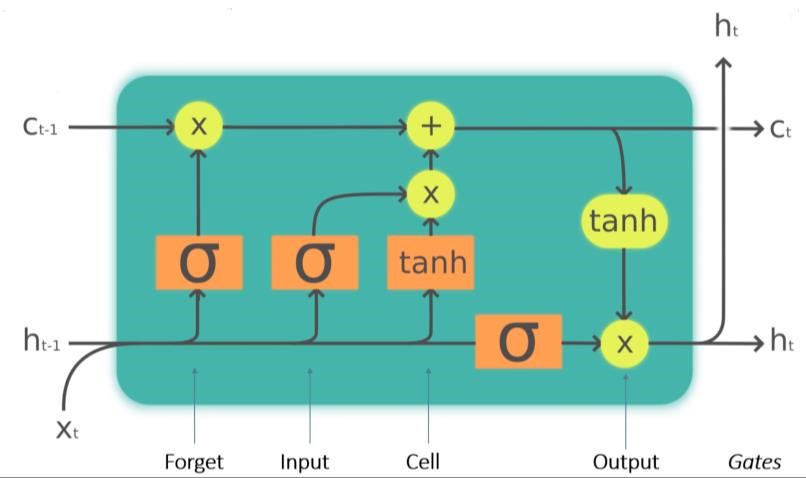
* 𝝏𝝏𝑳𝒘𝟏 = 𝝏𝝏𝑳𝒉𝟏 × 𝒙𝟏
* 𝝏𝝏𝑳𝒘𝟐 = 𝝏𝝏𝑳𝒉𝟏 × 𝒙𝟐
* 𝝏𝝏𝑳𝒘𝟑 = 𝝏𝝏𝑳𝒉𝟐 × 𝒙𝟏
* 𝝏𝝏𝑳𝒘𝟒 = 𝝏𝝏𝑳𝒉𝟐 × 𝒙𝟐

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

𝒍 = 𝑾𝒋𝒌𝒍 − 𝜸 𝝏𝑾𝝏𝑳𝒋𝒌𝒍𝑾𝒋𝒌

##### 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 ShortTerm 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 35: LSTM Cell Structure [78]

To achieve this functionality, the LSTM network makes use of LSTM cells that have the structure depicted in Figure 35. 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:

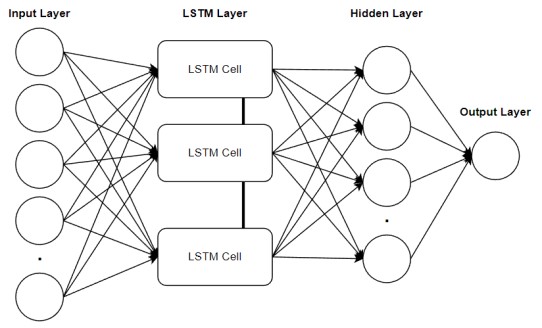
* + **Forget Gate:** This gate is responsible for discarding information deemed not useful. The gate takes two inputs, 𝑿𝒕(input at that particular time) and 𝒉𝒕−𝟏 (previous cell state). Those inputs are multiplied with weight matrices and are added with a bias. The result is passed through an activation 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 𝒉𝒕−𝟏and 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 𝒉𝒕−𝟏and 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 𝒉𝒕 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 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 36: 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

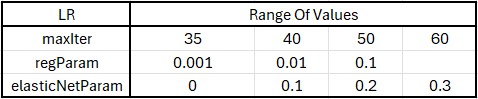
The training of the Logistic Regression model was conducted within the PySpark framework. For this process, we employed the Logistic Regression library, along with the ParamGridBuilder and CrossValidator components, which were used in the tuning process for the model.

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:



##### Fig 37: 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 40, a regParam of 0.01, and an elasticNetParam set to 0.1. 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.



##### Fig 38: Logistic Regression Training Results

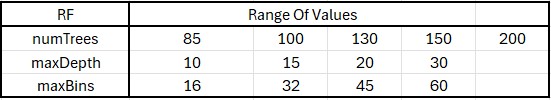
The metrics indicate strong performance by the model on the test set, which suggests effective learning of data patterns and successful generalization without overfitting to the training data. High accuracy points to good overall performance, while the high precision, recall, and F1 score metrics demonstrate 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:



##### Fig 39: 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 150, a maxDepth of 30, and a maxBins set to 16.

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



##### Fig 40: Random Forest Training Results

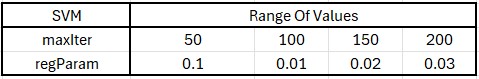
Much like the Logistic Regression model, the Random Forest model's test set results are very encouraging. The model shows high performance across all metrics, indicating a strong capability to identify 'Vishing' conversations. Moreover, the balanced F1 Score indicates the model's proficiency in maintaining 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:



##### Fig 41: 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 200, and regParam equal to 0.02.

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



##### Fig 42: Support Vector Machines Training Results

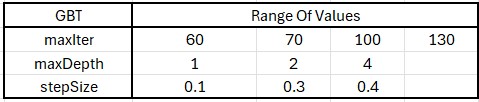
The results of the Support Vector Machines algorithm are slightly worse than those of the previous two algorithms. However, the metrics still indicate very good performance from the model all around.

#### 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:



##### Fig 43: 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 130, maxDepth equal to 4 and stepSize equal to 0.4.

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



##### Fig 44: Gradient Boosted Trees Training Results

The Gradient Boosted Trees model's performance in comparison to the previous models stands as the best one. It displays all four metrics above the 90% threshold signifying very good overall performance.

#### 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 training and optimization of Neural Network models can be a complex and timeconsuming process, requiring careful selection of hyperparameters to ensure the best possible performance. In the optimization of the Neural Network model for this study, we employed the HyperModel and HyperBand classes from the Keras Tuner library to automate this process. The HyperModel class allowed for flexible definition of the model's hyperparameter search space, while HyperBand was used because of its efficient tuning algorithm based on adaptive resource allocation and early stopping. This combination enabled a search across a wide range of hyperparameters, such as layer configurations, dropout rates, and optimizer settings, to find the optimal model configuration with minimal computational overhead. By evaluating and discarding suboptimal models early in the training process, HyperBand ensures that computational resources are concentrated on the most promising models, which finally leads to the identification of an optimized Neural Network model [79][80].

The hyperparameters that were tuned as a part of this process are the following:

* **Number of Neurons:** The number of neurons in each of the three dense layers of the model were tuned. For the first, second, and third layers, the number of neurons varied between 16 and 128, in steps of 16. This range was chosen to explore the impact of model complexity on performance, from relatively simple models to more complex ones that can capture intricate patterns in the data.

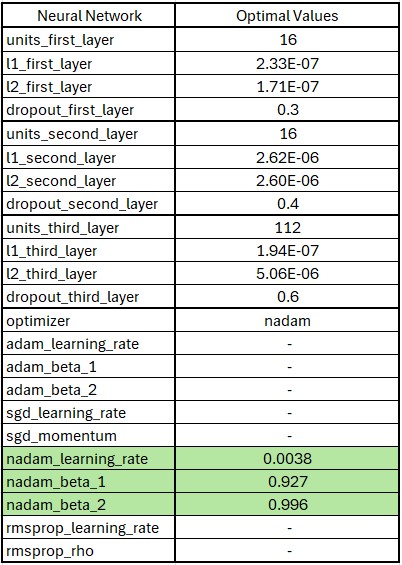
* **Regularization:** L1 and L2 regularization was employed for each layer to mitigate the risk of overfitting by penalizing large weights. The regularization strength for both L1 and L2 regularization was tuned within the logarithmic range of 1e-7 to 1e-1.

* **Dropout Rate:** To further deal with overfitting and promote model generalization, dropout layers were introduced after each dense layer. The dropout rate varied from 0.0 to 0.7 in steps of 0.05, aiming to find the balance between retaining and discarding information during training across the layers.

* **Optimizer:** Four optimizers were considered: Adam, SGD (Stochastic Gradient Descent), RMSprop, and Nadam. For each of these optimizers their respective tunable hyperparameters were tested. In detail, for the “adam” optimizer the learning rate, beta\_1 and beta\_2 variables were tested; for the “sgd” optimizer the learning rate and momentum variables were tested; for the “nadam” optimizer the learning rate, beta\_1 and beta\_2 variables were tuned and finally for the “rmsprop” optimizer the learning rate and rho variables were tuned.

* **Learning Rate:** Associated with each optimizer choice, the learning rate was tuned within a logarithmic scale of 1e-5 to 1e-2. The learning rate dictates the size of the steps taken during optimization, affecting the speed and stability of the training process.

As a result, the final architecture of the best model contained the following hyperparameters:



##### Fig 45: Optimal Hyperparameters as a result of the tuning process for the Neural Network

As can be seen in the above Figure, the optimal architecture output from the training and tuning process of the Neural Network is listed subsequently:

* **Input Layer:** 
  + Number of Neurons: The **first DENSE layer** consists of 16 neurons.
  + Regularization: L1 and L2 regularization strengths are set at approximately

2.33e-07 and 1.71e-07, respectively.

* + Dropout: A dropout rate of 30% is applied after the first dense layer.
  + Activation Function: The ‘ReLU’ activation function is employed in the first layer.
* **Second Layer (Hidden):** 
  + Number of Neurons: The **second DENSE layer** is composed of 16 neurons. o Regularization: For this layer, both L1 and L2 regularization strengths are finely tuned to approximately 2.62e-06 and 2.60e-06, respectively.
  + Dropout: A more moderate dropout rate of 40% is employed, balancing the need for regularization with the desire to preserve information flow through the network.
  + Activation Function: The ‘ReLU’ activation function is employed in the second layer.
* **Third Layer (Hidden):** 
  + Number of Neurons: The **third DENSE layer** contains 112 neurons.
  + Regularization: The L1 and L2 regularization strengths for this layer are approximately 1.94e-07 and 5.06e-06, respectively. o Dropout: The dropout rate climbs to 60%, in contrast to the previous layers, the dropout here is considerably higher. o Activation Function: Similarly to the previous layers, the ‘ReLU’ activation function is used once again in this Dense layer.
* **Output Layer:** 
  + The model concludes with a single-neuron output layer utilizing the sigmoid activation function.
* **Optimizer:** 
  + The optimizer that resulted in the best performance during the tuning process is the “nadam” optimizer with its beta\_1 and beta\_2 hyperparameters are 0.927 and 0.996 respectively and a learning rate of 0.0038 as shown in Figure 45.

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

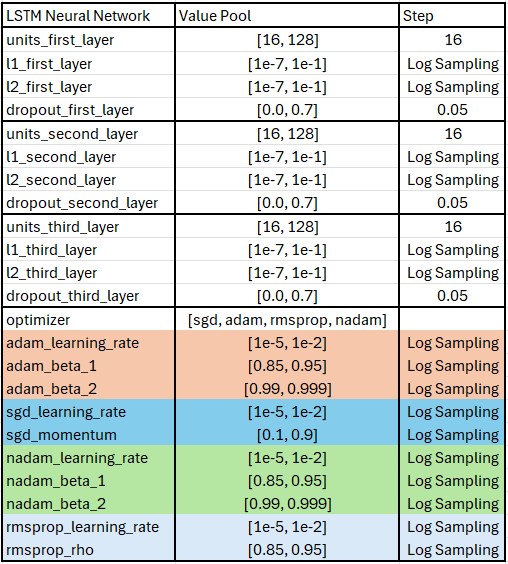


##### Fig 46: Neural Network Training Numerical Results

The Neural Network model shows strong performance across various metrics, particularly excelling in precision. Although its accuracy is slightly lower compared to other metrics, the model still performs very well overall.

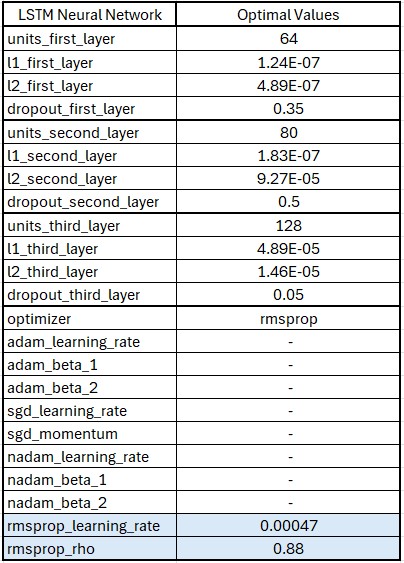
#### 7.4.6 LSTM Neural Network

Similar to the Neural Network model, the LSTM network was constructed using the Keras library and underwent training and tuning with Keras' HyperModel and HyperBand libraries. The hyperparameters adjusted during the tuning process were the same as those for the Neural Network and are outlined in the table below:



##### Fig 47: Hyperparameter Tuning Ranges for the LSTM Model Tuning

As a result, the final architecture of the best model contained the following hyperparameters:



##### Fig 48: Optimal Hyperparameters as a result of the tuning process for the LSTM Network

As can be seen in the above Figure, the optimal architecture output from the training and tuning process of the LSTM Network is listed subsequently:

* **Input Layer:** 
  + Number of Neurons: The **first LSTM layer** consists of 64 neurons.
  + Regularization: L1 and L2 regularization strengths are set at approximately

1.24e-07 and 4.89e-07, respectively.

* + Dropout: A dropout rate of 35% is applied after the first LSTM layer.
* **Second Layer (Hidden):** 
  + Number of Neurons: The **second LSTM layer** is composed of 80 neurons.
  + Regularization: For this layer, both L1 and L2 regularization strengths are finely tuned to approximately 1.83e-07 and 9.27e-05, respectively. o Dropout: A balanced dropout rate of 50% is utilized, balancing the dropout and the flow of information across the network.
* **Third Layer (Hidden):** 
  + Number of Neurons: The **third DENSE layer** contains 128 neurons. o Regularization: The L1 and L2 regularization strengths for this layer are approximately 4.89e-05 and 1.46e-05, respectively. o Dropout: The dropout rate climbs to 5%, in contrast to the previous layers, the dropout here is considerably lower. Suggesting that the information from that layer is mandatory for the subsequent output layer.
  + Activation Function: In this Dense layer the ‘ReLU’ activation function is used.
* **Output Layer:** 
  + The model concludes with a single-neuron output layer utilizing the sigmoid activation function.
* **Optimizer:** 
  + The "rmsprop" optimizer emerged as the best-performing optimizer during the tuning process, with a rho hyperparameter set at 0.88. This rho value, acting as a decay factor, influences the adaptation of the learning rate by controlling the moving average of the squared gradients. The learning rate was optimized to 0.00047.

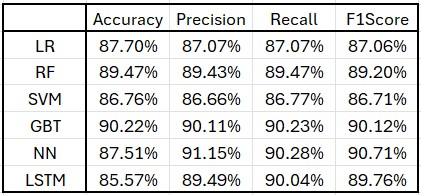
The results of the training process can be seen in the following table.



##### Fig 49: LSTM Network Training Results

Much like the previous models, the LSTM Network also shows strong performance on the test set, though it records the lowest accuracy among the metrics. However, it compensates with a high recall rate, which is crucial for this particular task.

#### 7.4.7 Overall Results



##### Fig 50: All Model Results

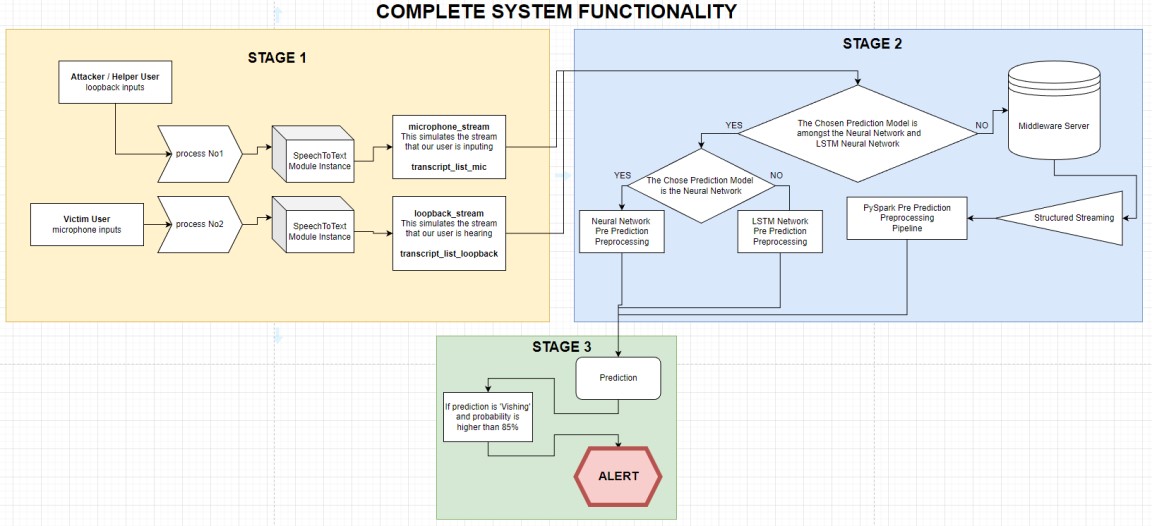
In summary, the Gradient Boosted Trees and Neural Network models demonstrate the best performance, with both achieving high scores across accuracy, precision, recall, and F1 metrics on the test set. These results suggest that they have learned to generalize well without overfitting to the training data.

However, the Logistic Regression model reveals some generalization challenges, as indicated by a noticeable performance drop compared to the other models.

Overall, while all models show potential, the Gradient Boosted Trees and Neural Network models stand out for their high performance metrics on the test set, while the rest of the models are not far behind in terms of performance.

# 8. Overall System Functionality

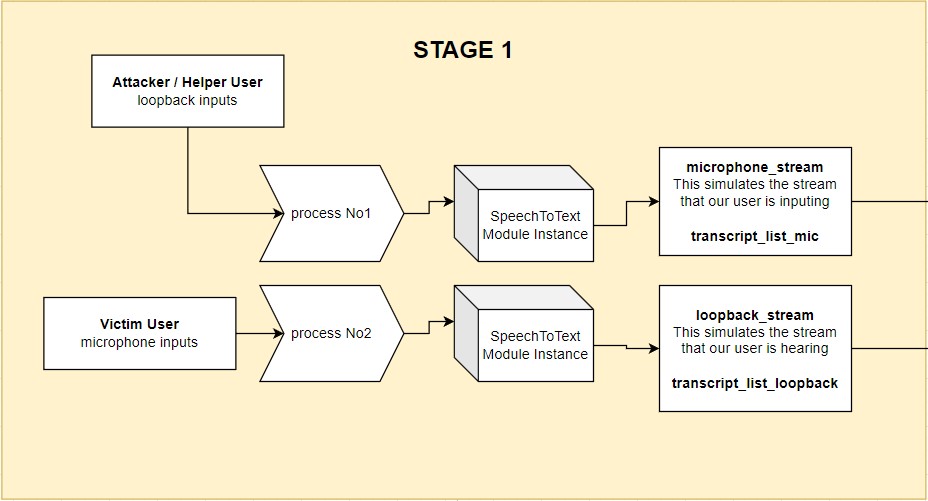
In this section, we delve into the operation of the complete system designed to detect 'Vishing' calls. We will dissect how each component within the system contributes to the identification of potential scam attempts during conversations. Initially, we will examine the system's overall functionality and then proceed to analyze the contribution of each individual component to the workflow of the complete system.



## Fig 51: Complete System Functionality Workflow

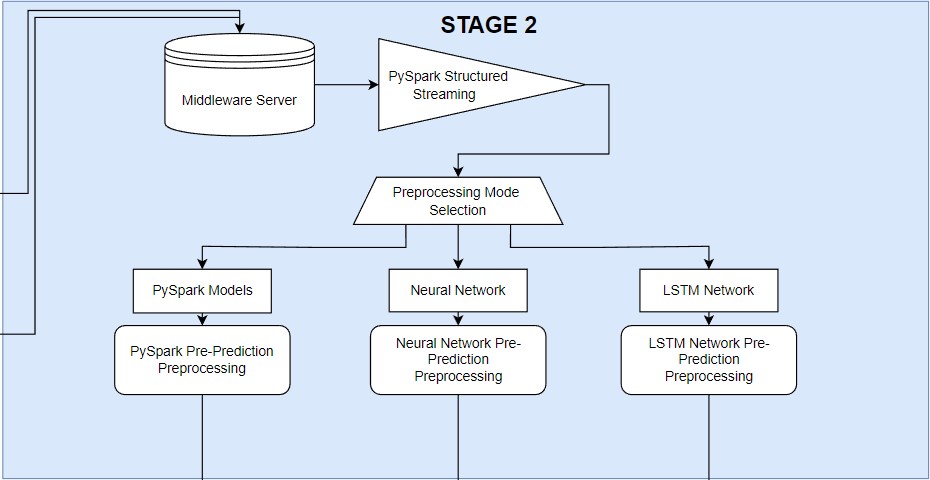
The proposed system is structured into three stages, each of which will be elaborated upon in the following sections. The first stage includes the mechanisms for capturing speech inputs and their subsequent conversion into text format, forming the basis of the conversation analysis. The second stage details the preprocessing of these textual inputs, preparing them for real-time prediction by the models. Lastly, the third stage focuses on the model predictions themselves and the specific criteria under which an alert for potential 'Vishing' activity is triggered.

## 8.1 Stage One



### Fig 52: Stage One of the Complete System Workflow

In the first stage of the system, two processes start at the same time, both using the SpeechTo-Text component described in Chapter 4, which utilizes the Google Api Speech Recognition engine. The first process turns on the Speech To Text feature and starts picking up sounds from the device's microphone. This microphone records what our user of interest is saying. At the same time, the second process starts another Speech To Text feature to record the device's loopback audio which is basically what the user hears on the phone’s speaker. Because of these two processes, we end up with two sets of data: one with words spoken by our user and the other with words spoken by the person on the other end of the call. These two sets of data are collected together as they happen at the same time and are stored in two data queues. These two queues are then passed on to stage two of the system. **8.2 Stage Two**



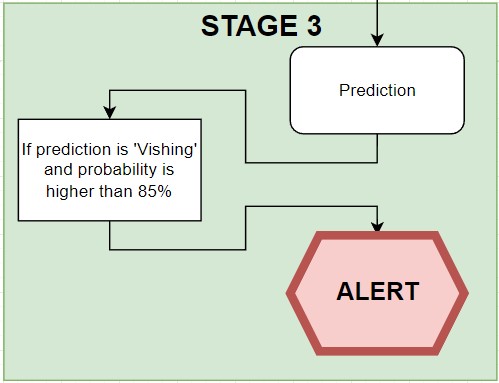
### Fig 53: Stage Two of the Complete System Workflow

Every time either of the two aforementioned data queues receive any piece of new data both data queues pass on to stage two of the complete system workflow. In this stage the data from the queues is sent to the middle-ware server whose job is to convert the data into a dataframe format and send it back to preprocessed depending on which preprocessing mode is selected. In total there are three preprocessing modes which can be outlined in the following points:

* **PySpark Preprocessing:** To support prediction from the PySpark models the data from the queues must pass through the Preprocessing pipeline for Deployment mentioned in chapter 6.3. Firstly, the text data from both queues are transformed into TF-IDF vectors, a numerical statistic that reflects how important a word is to a document in a collection. These two TF-IDF vectors (one for each data queue) are then combined to form one larger vector. This combined vector has twice the length of the individual vectors, totaling 400 elements. Initially, the process required converting these data queues into a DataFrame format for PySpark processing. This conversion was timeconsuming and increased response times. Therefore, to speed up the process, it was decided to send the data from the queues to a middleware server via a socket. This server handles the conversion more efficiently and streams the data back to the main application for faster preprocessing. For this process PySpark’s structured streaming feature was employed which allows for continuous processing of the streaming data. The outcome of this whole preprocessing is a single, 400-length vector which is in Dataframe format ready to be used by the PySpark models.

* **Neural Network Preprocessing:** As mentioned in Chapter 7.3.5, the Neural Network was developed independently of the PySpark framework. To ensure consistency across models, the same preprocessing pipeline designed for the PySpark models was applied to the Neural Network. Contrary to the PySpark models, this approach resulted in significantly longer response times for the Neural Network. While it is possible to reduce these delays by manually implementing the preprocessing steps included within the pipeline, this approach was not adopted in the current study.

* **LSTM Network Preprocessing:** The preprocessing needed for the LSTM Network is effectively the same as that of the Neural Network. However, the LSTM Network necessitates an additional step: the input data must be presented in a transposed format. This means that the data vector, typically a 400-length list as prepared for the Neural Network, is rearranged so that what were originally columns in the data become rows, and vice versa. This transposition aligns with the LSTM's requirement for sequential data input, where each element of the sequence (now a row in the transposed format) is fed into the network one at a time, facilitating the LSTM's ability to process and learn from sequential or time-series data effectively.  **8.3 Stage Three**



### Fig 54: Stage Three of the Complete System Workflow

In this final stage, the actual prediction by the selected algorithm occurs. The vector generated in the previous stage is inputted into the chosen model, which then provides a prediction along with its associated probability. Additionally, this stage might feature an option in the future to trigger an alert if the prediction is classified as 'Vishing' and the probability surpasses a specific threshold, such as 85%, as illustrated in Figure 54. This alert can vary in form according to the needs of the application, for example, it might be a simple pop up screen. For the PySpark models, the prediction output is presented in a DataFrame format. In contrast, for the Neural Network and LSTM Network models, the output is a simple list format that includes only the probability.

## 8.4 Challenges and Future Enhancements

During the development of our system, we encountered several challenges. One of the primary issues was the extended response time, which we managed to address using PySpark’s structured streaming. However, another significant challenge emerged with the PySpark models, particularly in implementing an alert system. The output from these models is in DataFrame format, which necessitates conversion into a more rapidly processable format for alerting purposes. This conversion adds an additional delay to the system's overall response time and is an area that requires further optimization in future iterations of the system.

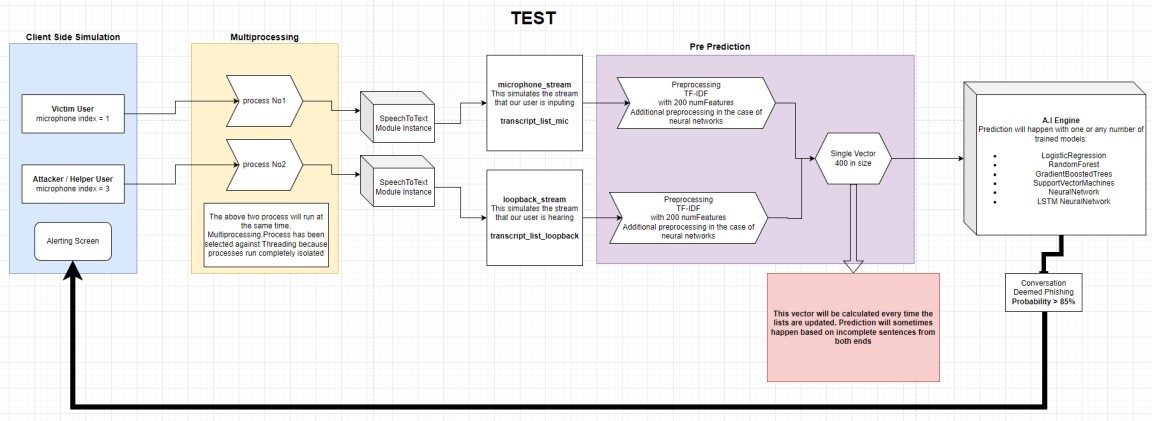
Furthermore, a significant challenge encountered during system testing was the accuracy of the speech-to-text transcription. The performance of the Speech To Text module was notably affected by the user's accent, leading to instances where spoken words were inaccurately transcribed into text, which in turn contributed to errors in the system's output.

Additionally, there's the prospect of employing a different framework, such as PyTorch, instead of PySpark. PyTorch provides the flexibility of working with simple lists and arrays, as opposed to DataFrames. While this could potentially streamline the data processing and reduce response times, it's important to note that such a shift might impact the inherent scalability and fault tolerance provided by the Spark framework. Therefore, while considering PyTorch as an alternative, these trade-offs must be carefully weighed to ensure the system maintains its efficiency.

# 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 test the real-time outcomes of each model when confronted with these varied conversational contexts. Our objective is to assess the system's operational competence and to draw insights into its practical applicability, and reliability in dynamic, real-life settings that mimic actual user interactions.

The testing setup involves the use of two distinct speech input devices operating consecutively. The one device is designated to record the dialogue from the potential victim's perspective, while the second captures the words of the attacker or helper, simulating a real life scenario. The test workflow, which is diagrammatically represented in the subsequent figure 48, replicates the system's functionality similarly to Chapter 8. The key distinction in this setup from the one described in Chapter 8 is the use of two audio inputs where both capture audio data in a single computer but in different channels instead of one capturing the microphone data and the other one capturing the loopback data:

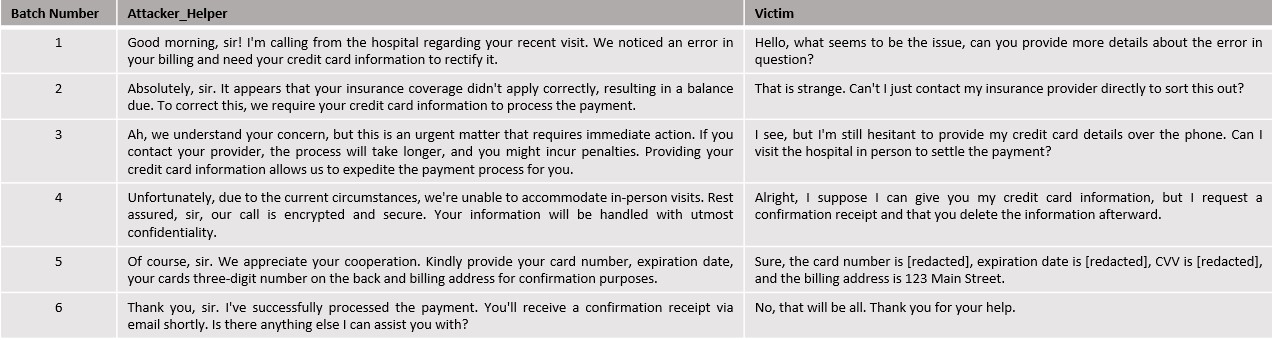


## Fig 55: Testing Workflow

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.

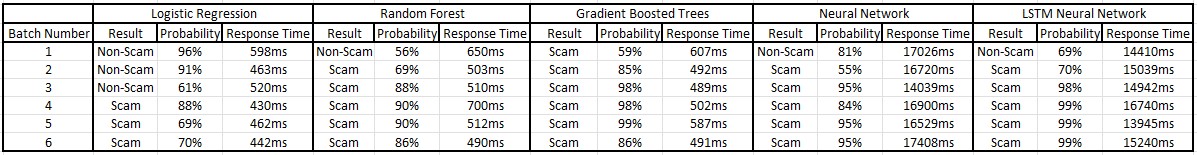
In this research, the Support Vector Machines (SVM) algorithm was determined to be unsuitable, because it does not provide probability estimates alongside its output predictions. As highlighted in Chapter 8, the ability to quantify prediction confidence in the form of probabilities is essential for the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



**Fig 56: ‘Vishing' Conversation with Known Formatting**

The context of the conversation depicted in figure 55 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.



**Fig 57: Model Results on Familiar ‘Vishing’ Conversation**

* For the **Logistic Regression** model, the initial response is a categorization of 'NonScam' 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 88%. Thus, the Logistic Regression model does succeed in identifying the conversation as a scam before the critical fifth batch.

* 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 90%. 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.

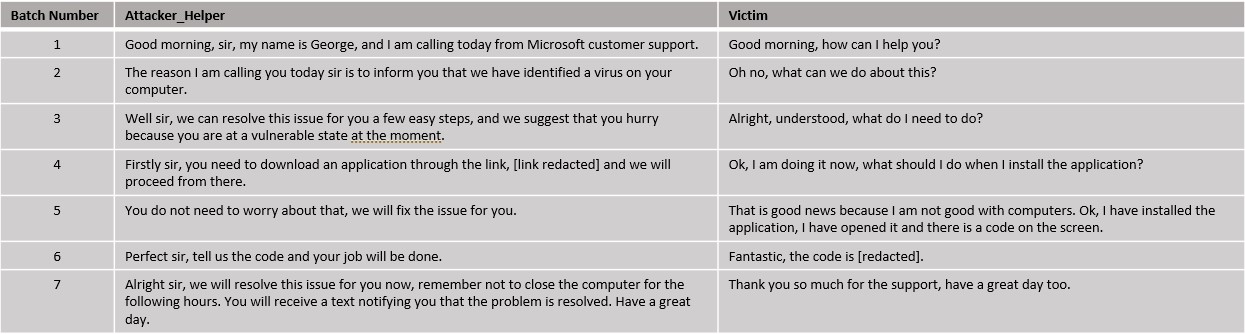
* The response of the **Gradient Boosted Trees** model is immediately decisive. From the very first batch of the conversation, it identifies the scenario as a scam with a moderate probability of 59%. By the fourth batch, the probability of the conversation being a scam increases to 98%. This immediate 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** initially classifies the conversation as ‘Normal’ with a probability of 81% but then the model consistently classifies the conversation as a 'Scam' throughout the remainder of the duration of the conversation, increasing its probability from 55% on the second batch to 84% on the fourth batch. 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 second batch, being more confident than the normal Neural Network model. Similarly, to the Gradient Boosted Trees algorithm, this model exhibits very high probabilities indicating that it too has identified a pattern similar to those witnessed in training.

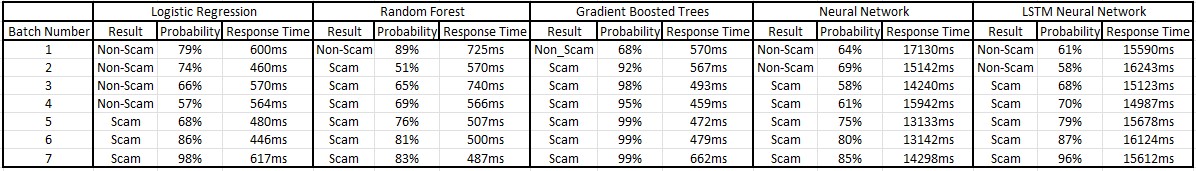
The results of this test highlight the **Random Forest** algorithm as the best performer among all evaluated models. Its effectiveness is marked by a balanced approach, where it maintains neutrality where 'Vishing' characteristics in the conversation are ambiguous. Furthermore, the algorithm demonstrates the ability to discern suspicious elements in the conversation before the critical fifth batch, where the user falls victim to the scam. This quick classification is crucial, as any model that fails to detect the scam before this stage is deemed unsuccessful in this test.

**9.2 Model Performance on Unknown ‘Vishing’ Conversation**



### Fig 58: Vishing conversation with Unknown formatting

The dialogue illustrated in figure 57 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.



**Fig 59: Model Results on Unknown ‘Vishing’ Conversation**

* Much like in the previous conversation, the **Logistic Regression** model exhibits very good performance. It begins to exhibit signs of suspicion by the fourth batch. The fact that the model starts to identify the conversation as potentially suspicious in the fourth batch suggests that it maintains a balanced approach and it is not overly sensitive to the scam characteristics of the conversation.

* 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 high confidence, with a probability of 89%. By the sixth batch, the model has formed its final stance, categorizing the conversation as a 'Scam' with a high probability of 81%.

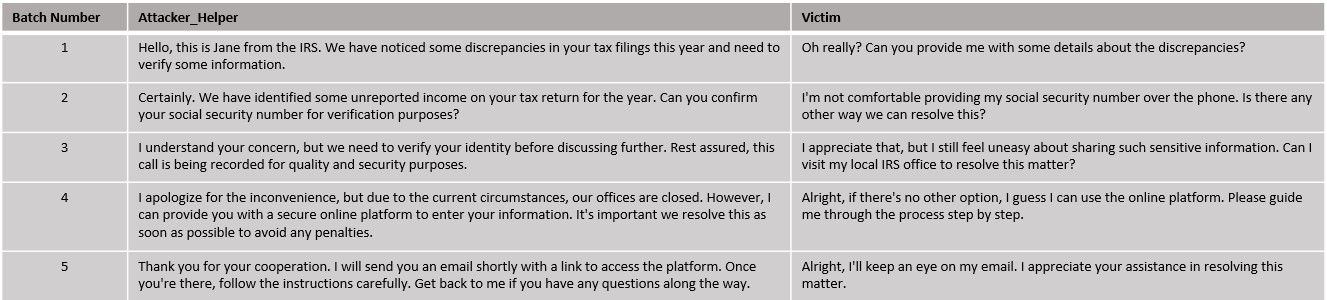
* The **Gradient Boosted Trees** model exhibits very high certainty in correctly identifying the nature of the conversation. It consistently categorizes the conversation as ‘Scam' with its probability fluctuating around 95%.

* The **Neural Network model** initially identifies the conversation as 'Normal. It maintains this assessment until the third batch where it changes its opinion and suggests that the conversation is a possible ‘Scam’ with a probability of 58%. This is a very good performance since the model successfully predicts the conversation as it unfolds.

* Similar to the Neural Network model and the previous conversation example, the **LSTM Network** also successfully classifies the conversation, but with higher probability. This suggests that the LSTM model is more conservative in comparison to the Neural Network model since it is more sensitive to the ‘Scam’ characteristics of the conversation.

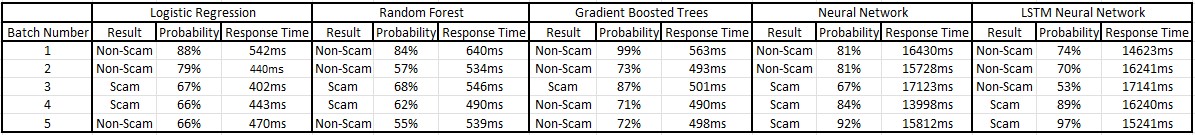
As a result, it appears that all models exhibit a desirable behavior with the **Neural Network** and **LSTM Network** showing the most promise. The **Logistic Regression** model also exhibits promising capabilities, as it successfully identifies the true nature of the conversation towards the middle. 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**



**Fig 60: Normal conversation with Known Formatting.**

The conversation illustrated in Figure 59 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.



**Fig 61: 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 around 84%. This high level of confidence is maintained up to the third batch of the conversation, after which the probability begins to fluctuate between 67% to 66%. This behavior suggests that while the algorithm initially recognizes a familiar pattern, it starts to exhibit skepticism by the fourth batch. On the fifth batch the model suggests that the conversation is indeed ‘Normal’ with a probability of 66%. 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 84%. Its confidence plateaus at 57% for the second batch. By the third batch the model begins showing signs of suspicion which are maintained until the fourth batch. Similarly to the Logistic Regression model on the fifth batch it concludes that the conversation is indeed normal, and hence is deemed successful in this test.

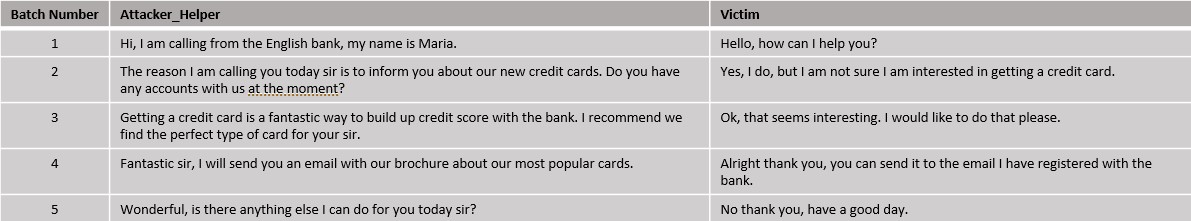
* The **Gradient Boosted Trees** model immediately classifies the conversation as ‘normal’ with high probability, again indicating that the algorithm has identified a familiar pattern. This algorithm also exhibits significant suspicion on the third batch much like the other models but its quicker to overturn this suspicion by the fourth batch and solidify that choice in the fifth batch. Since the model successfully identifies the conversation and correctly displays suspicion where it is necessary, its performance is very good.

* Both the **Neural Network** and **LSTM Network** models exhibit similar behavior, classifying the conversation as ‘Vishing’ and maintaining that assessment throughout

the first two batches of conversation in the case of the Neural Network and three in the case of the LSTM Network. The Neural Network model suggests that the conversation is ‘Vishing’ by the third conversation and keeps this assessment throughout the remainder of the conversation while the LSTM Network does so by the fourth batch.

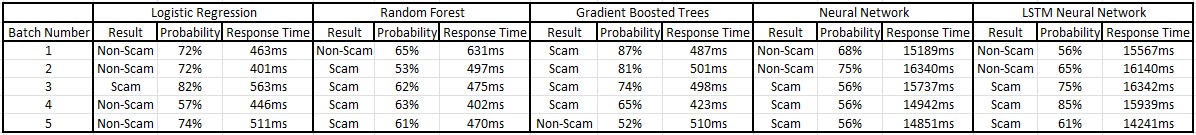
In conclusion, from this test, the **Logistic Regression**, **Random Forest**, and **Gradient Boosted Trees** models all exhibit good performance. Notably, the **Logistic Regression** and **Gradient Boosted Trees** models stand out, showing the most promise in their ability to accurately classify and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

**9.4 Model Performance on Unknown ‘Normal’ Conversation**



**Fig 62: 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.



**Fig 63: 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 72% by the second batch. In the third batch, it alters its prediction to 'vishing', with a probability of 82%, 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 74%. 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 a probability of 65%. In the third batch, it changes its prediction to 'Vishing', assigning a probability of 53%. 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.

* The **Gradient Boosted Trees** model fails to pick up the ‘Normal’ characteristics of the conversation and maintains a suggestion of ‘Vishing’ up until the fourth batch of the conversation where it switches its decision to ‘Normal’ but with a low probability of 52%. This behavior suggests that the model eventually somewhat understands the meaning of the conversation as it progresses but in the context of this test it is considered failed.

* Both the **Neural Network** and **LSTM Network** models exhibit a similar pattern in their response to the conversation. They initially classify the conversation as ‘Normal’ right from the start and maintain this classification up until the second batch. Both models then switch their prediction to ‘Vishing’ with the Neural Network maintaining a probability of around 56% and the LSTM Networks probability ranging from 75% to 61% on the final batch.

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 in ‘Normal’ scenarios, 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 and the LSTM Network model show a response time of approximately 15 seconds. 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 Logistic Regression 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.

The Neural Network and LSTM models also show great performance throughout but tend to label conversations as 'Vishing' more often than 'Normal', a behavior especially noticeable in the Neural Network model compared to the LSTM.

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

This dissertation aimed to explore the viability of a real-time voice phishing (vishing) detection system utilizing machine learning technologies. We delved into various components such as PySpark and the Speech-To-Text module, the creation and preprocessing of the dataset manufactured for this task, and the selection, functionality, training, and tuning of the models. As a result of the training process the Logistic Regression and Random Forest models stood out for their optimal performance in addressing our problem.

In the last chapters of this study, we demonstrated how these components and models integrate into a unified system, showcasing its real-time performance across four distinct conversational scenarios. Notably, the Logistic Regression and Random Forest models showed promising results, being able to adequately detect both ‘Vishing’ and ‘Normal’ conversations with the system's response times averaging around 500ms.

The research yielded encouraging outcomes, particularly with the strong real-time testing performance of the Logistic Regression and Random Forest models, showing that further exploration in this domain can result in the solution of the problem at hand which is identifying ‘Vishing’ conversations quickly.

However, the study faced challenges, notably the limited diversity in the training dataset created via the Chat GPT API, leading to a reduced generalization ability in the models. Additionally, the dataset's size was smaller than ideal for such a complex task, complicating the models' training and generalization. Despite implementing several techniques to mitigate overfitting and enhance generalization, these issues persisted in some models.

Future efforts should focus on developing a robust and diverse dataset. This dataset should include a wide range of everyday conversations and sophisticated scam dialogues, enabling the models to detect and counter advanced fraudulent tactics but understand at the same time when a conversation is completely normal.

This research paves the way for advancements in user protection against phone-based scams, a critical issue in today's technology-driven society. The development of an effective, sustainable system capable of real-time vishing detection could significantly enhance telephonic security.

Reflecting on this journey, the potential of machine learning in combating vishing is evident, yet the path consists of challenges as mentioned previously. The promising results from the Logistic Regression and Random Forest models underscore the feasibility of the task yet highlight the necessity for continued innovation and refinement. This dissertation adds valuable insights to the conversation about cybersecurity and highlights how important it is to keep people safe as we rely more on digital technology. Looking ahead, it's crucial to work on making our data more varied and our models more reliable. Doing so will help us fully unlock the possibilities of detecting vishing in real-time, leading to safer ways of communicating.

# Bibliography

1. “Inside the intricate world of voice phishing” [Online]. Available: [https://koreajoongangdaily.joins.com/2019/02/01/socialAffairs/Inside-the-intricate-worldof-voice-phishing/3059014.html](https://koreajoongangdaily.joins.com/2019/02/01/socialAffairs/Inside-the-intricate-world-of-voice-phishing/3059014.html)
2. “Voice Phishing” [Online]. Available: <https://en.wikipedia.org/wiki/Voice_phishing>
3. “What Is a Vishing Attack?” [Online]. Available: <https://www.fortinet.com/resources/cyberglossary/vishing-attack>
4. “What is Vishing?” [Online]. Available: <https://softwarelab.org/blog/what-is-vishing/>
5. “What is Dumpster Diving?” [Online]. Available: <https://www.techtarget.com/searchsecurity/definition/dumpster-diving>
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 & Preventions.” [Online]. Available: <https://www.imperva.com/learn/application-security/vishing-attack/>
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. 486492). 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.” [Online]. Available: [https://www.researchgate.net/figure/The-Spark-cluster-architecture-forresource-allocation-and-data-transfer\_fig3\_337305063](https://www.researchgate.net/figure/The-Spark-cluster-architecture-for-resource-allocation-and-data-transfer_fig3_337305063)
17. “Apache Spark Components.” [Online]. Available:

<https://intellipaat.com/blog/tutorial/spark-tutorial/apache-spark-components/>

1. “Key features of Spark’s MLlib.” [Online]. Available: [https://www.researchgate.net/figure/Key-features-of-Sparks-MLlib-asparkmllib-is-built-ontop-of-RDDs-bsparkml-is-built\_fig6\_309138349](https://www.researchgate.net/figure/Key-features-of-Sparks-MLlib-asparkmllib-is-built-on-top-of-RDDs-bsparkml-is-built_fig6_309138349)
2. 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.
3. “Optimization – RDD - based API.” [Online]. Available: <https://spark.apache.org/docs/latest/mllib-optimization.html>
4. “Broyden-Fletcher-Goldfarb-Shanno algorithm.” [Online]. Available:

[https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%9 3Shanno\_algorithm](https://en.wikipedia.org/wiki/Broyden%E2%80%93Fletcher%E2%80%93Goldfarb%E2%80%93Shanno_algorithm)

1. “Spark Machine Learning Pipelines.” [Online]. Available: <https://spark.apache.org/docs/latest/ml-pipeline.html>
2. “Spark Machine Learning Pipeline for regression.” [Online]. Available: <https://gbhat.com/spark/spark_ml_regression_pipeline.html>
3. “Spark Streaming Guide.” [Online]. Available: <https://spark.apache.org/docs/latest/streaming-programming-guide.html>
4. “What is Speech to Text?” [Online]. Available: [https://aws.amazon.com/whatis/speech-to-text/](https://aws.amazon.com/what-is/speech-to-text/)
5. 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.
6. “Speech-to-Text: Automatic Speech Recognition.” [Online]. Available: <https://cloud.google.com/speech-to-text/?hl=en#all-features>
7. “Generative pre-trained transformer.” [Online]. Available: <https://en.wikipedia.org/wiki/Generative_pre-trained_transformer>
8. “Transformer” [Online]. Available: <https://en.wikipedia.org/wiki/Transformer_(machine_learning_model)>
9. “What is a Transformer?” [Online]. Available: [https://medium.com/inside-machinelearning/what-is-a-transformer-d07dd1fbec04](https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04)
10. “Prompt Engineering” [Online]. Available: <https://en.wikipedia.org/wiki/Prompt_engineering>
11. “Master the Perfect ChatGPT Prompt Formula” [Online]. Available:

[Campaign\_GRE\_Tell2\_Instream (youtube.com)](https://www.youtube.com/watch?v=jC4v5AS4RIM)

1. “What is Tokenization?” [Online]. Available: [What is Tokenization? Types, Use Cases, Implementation | DataCamp](https://www.datacamp.com/blog/what-is-tokenization)
2. “Tokenization in Natural Language Processing” [Online]. Available: [Tokenization in Natural Language Processing | by Mohammad Derakhshan | Medium](https://medium.com/@m.derakhshan/tokenization-in-natural-language-processing-86e4e235a2aa)
3. Perkins, J. (2010). *Python text processing with NLTK 2.0 cookbook*. PACKT publishing.
4. “NLTK Documentation” [Online]. Available: <https://www.nltk.org/api/nltk.tokenize.word_tokenize.html>
5. “Introduction to Stemming” [Online]. Available: <https://www.geeksforgeeks.org/introduction-to-stemming/>
6. “All you need to know about text preprocessing for NLP and Machine Learning” [Online]. Available: [https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machinelearning.html](https://www.kdnuggets.com/2019/04/text-preprocessing-nlp-machine-learning.html)
7. “Stemming” [Online]. Available: <https://en.wikipedia.org/wiki/Stemming>
8. “Flatten PySpark function” [Online]. Available: [https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.fun ctions.flatten.html](https://spark.apache.org/docs/latest/api/python/reference/pyspark.sql/api/pyspark.sql.functions.flatten.html)
9. “Tf-Idf” [Online]. Available: <https://en.wikipedia.org/wiki/Tf%E2%80%93idf>[42] “Feature Hashing” [Online]. Available: <https://en.wikipedia.org/wiki/Feature_hashing>
10. “Machine Learning” [Online]. Available: <https://en.wikipedia.org/wiki/Machine_learning>
11. 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.
12. “Supervised Learning” [Online]. Available: <https://en.wikipedia.org/wiki/Supervised_learning>
13. “Unsupervised Learning” [Online]. Available: <https://en.wikipedia.org/wiki/Unsupervised_learning>
14. 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.
15. 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.
16. Ying, X. (2019, February). An overview of overfitting and its solutions. In *Journal of physics: Conference series* (Vol. 1168, p. 022022). IOP Publishing.
17. “Underfitting and Overfitting in machine learning and how to deal with it.” [Online]. Available: [https://towardsdatascience.com/underfitting-and-overfitting-in-machinelearning-and-how-to-deal-with-it-6fe4a8a49dbf](https://towardsdatascience.com/underfitting-and-overfitting-in-machine-learning-and-how-to-deal-with-it-6fe4a8a49dbf)
18. Shorten, C., Khoshgoftaar, T. M., & Furht, B. (2021). Text data augmentation for deep learning. *Journal of big Data*, *8*, 1-34.
19. “Data augmentation” [Online]. Available: <https://en.wikipedia.org/wiki/Data_augmentation>
20. “Regularization in Machine Learning” [Online]. Available: <https://www.geeksforgeeks.org/regularization-in-machine-learning/>
21. “Cross-validation (statistics)” [Online]. Available: [https://en.wikipedia.org/wiki/Crossvalidation\_(statistics)](https://en.wikipedia.org/wiki/Cross-validation_(statistics))
22. “Binary Classification” [Online]. Available: <https://en.wikipedia.org/wiki/Binary_classification>
23. “Logistic Regression in Machine Learning” [Online]. Available: <https://www.geeksforgeeks.org/understanding-logistic-regression/>
24. “Logistic Regression” [Online]. Available: <https://en.wikipedia.org/wiki/Logistic_regression>
25. “Support Vector Machine (SVM) Algorithm” [Online]. Available: <https://www.geeksforgeeks.org/support-vector-machine-algorithm/>
26. “Support Vector Machine” [Online]. Available: <https://en.wikipedia.org/wiki/Support_vector_machine>
27. “Support Vector Machine-Based EMG Signal Classification Techniques: A Review” [Online]. Available: <https://www.mdpi.com/2076-3417/9/20/4402>
28. “Support Vector Machines (SVM)” [Online]. Available: <https://learnopencv.com/support-vector-machines-svm/>
29. De Ville, B. (2013). Decision trees. *Wiley Interdisciplinary Reviews: Computational Statistics*, *5*(6), 448-455.
30. “Decision Tree” [Online]. Available: <https://en.wikipedia.org/wiki/Decision_tree>
31. “What is a Decision Tree?” [Online]. Available: <https://www.geeksforgeeks.org/decision-tree/>[65] “Random Forest” [Online]. Available: <https://en.wikipedia.org/wiki/Random_forest>
32. “Bootstrap aggregating” [Online]. Available:

<https://en.wikipedia.org/wiki/Bootstrap_aggregating>

1. “Gradient boosting” [Online]. Available: <https://en.wikipedia.org/wiki/Gradient_boosting#Gradient_tree_boosting>
2. “Gradient Boosting in Machine Learning” [Online]. Available: <https://www.geeksforgeeks.org/ml-gradient-boosting/>
3. “The Evolution and Core Concepts of Deep Learning and Neural Networks” [Online]. Available: [https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deeplearning-neural-networks/](https://www.analyticsvidhya.com/blog/2016/08/evolution-core-concepts-deep-learning-neural-networks/)
4. “What is a neural network?” [Online]. Available: <https://www.geeksforgeeks.org/neural-networks-a-beginners-guide/>
5. “Activation functions in Neural Networks” [Online]. Available: <https://www.geeksforgeeks.org/activation-functions-neural-networks/>
6. “Softmax function” [Online]. Available: [https://www.geeksforgeeks.org/activationfunctions-neural-networks/](https://www.geeksforgeeks.org/activation-functions-neural-networks/)
7. “A Gentle Introduction to Dropout for Regularizing Deep Neural Networks” [Online]. Available: [https://machinelearningmastery.com/dropout-for-regularizing-deep-neuralnetworks/](https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/)
8. “Gradient Descent Algorithm in Machine Learning” [Online]. Available: <https://www.geeksforgeeks.org/gradient-descent-algorithm-and-its-variants/>
9. “Backpropagation in Data Mining” [Online]. Available: <https://www.geeksforgeeks.org/backpropagation-in-data-mining/>
10. “Long short-term memory” [Online]. Available: <https://en.wikipedia.org/wiki/Long_short-term_memory>
11. “Understanding of LSTM Networks” [Online]. Available: <https://www.geeksforgeeks.org/understanding-of-lstm-networks/>
12. “Understanding LSTMs” [Online]. Available: [https://kushalj001.github.io/black-boxml/lstm/pytorch/gates/vanishing%20gradient/2019/12/28/Understanding-LSTMs.html](https://kushalj001.github.io/black-box-ml/lstm/pytorch/gates/vanishing%20gradient/2019/12/28/Understanding-LSTMs.html)
13. “Introduction to the Keras Tuner” [Online]. Available: <https://www.tensorflow.org/tutorials/keras/keras_tuner>
14. “Hyperband Tuner” [Online]. Available: <https://keras.io/api/keras_tuner/tuners/hyperband/>
15. Kingma, D. P., & Ba, J. (2014). Adam: A method for stochastic optimization. *arXiv preprint arXiv:1412.6980*.
16. “Intuition of Adam Optimizer” [Online]. Available: <https://www.geeksforgeeks.org/intuition-of-adam-optimizer/>
17. “Stochastic Gradient Descent (SGD)” [Online]. Available: <https://www.geeksforgeeks.org/ml-stochastic-gradient-descent-sgd/>
18. “Tuning Your Keras SGD Neural Network Optimizer” [Online]. Available: [https://kyle-rkieser.medium.com/tuning-your-keras-sgd-neural-network-optimizer-768536c7ef0](https://kyle-r-kieser.medium.com/tuning-your-keras-sgd-neural-network-optimizer-768536c7ef0)
19. “Gradient Descent With RMSProp from Scratch” [Online]. Available: <https://www.geeksforgeeks.org/gradient-descent-with-rmsprop-from-scratch/>
20. “Gradient Descent Optimization With Nadam From Scratch” [Online]. Available: [https://machinelearningmastery.com/gradient-descent-optimization-with-nadam-fromscratch/](https://machinelearningmastery.com/gradient-descent-optimization-with-nadam-from-scratch/)