Suleyman Demirel University
Faculty of Engineering and Natural Sciences
Department of Mathematics and Natural Sciences

Acting Dean of Faculty

Associate Professor, PhD Zhamanov A.

« _____ » 2023

Topic of the thesis:
Creation of effective chatbots based on neural networks

Thesis submitted as part of the requirements for the award of the MSc in "7M05401-Mathematics", SDU, 2021-2023

Head of Department _____ Madina Abdykarim

Academic Supervisor _____ Shirali Kadyrov

Master student _____ Yermek Daniyar

Kaskelen, 2023

The Ministry of Science and Higher Education of the Republic of Kazakhstan

Suleyman Demirel University

Yermek Daniyar

Creation of effective chatbots based on neural networks

THESIS

Presented in Partial Fulfilment for the

Degree of Master of Science in Mathematics

(degree code: 7M05401)

Department of Engineering

Faculty of Engineering and Natural Sciences

Supervisor: **Shirali Kadyrov**

Kaskelen, 2023

Declaration

I confirm that this is my own work and the use of all material from other sources has been properly and fully acknowledged.

Yermek Daniyar

June 2023

Abstract

This paper presents a comprehensive exploration of the development process for a neural network-based chatbot. Given the increasing popularity of neural networks and chatbots in recent years, investigating the creation process of such a chatbot is both pertinent and valuable.

The primary objective of this study is twofold: first, to construct an effective chatbot, and second, to identify the most precise model for text classification. Furthermore, the paper delves into the mathematical aspects of creating a chatbot and provides a detailed explanation of neural networks.

The initial step in our project involved the creation of a database containing a sufficient number of student questions, which were then categorized into ten distinct categories. To achieve this, we carefully selected the ten most commonly asked questions and generated multiple paraphrased versions of each question. These variations were employed for training and evaluating our models.

During the course of our research, we identified three text classification models that proved to be the most suitable for our purposes: Multinomial Logistic Regression, Naive Bayes, and Neural Network. We conducted extensive tests using these models and documented the results in a table, providing a comprehensive analysis of their performance.

Following the successful completion of the database collection and the identification of the optimal text classification model, we proceeded to create the chatbot using DialogFlow. Additionally, we integrated our chatbot into the Telegram messenger for wider accessibility and user convenience.

Аңдатпа

Бұл диссертациялық жұмыста біз нейронды желілер арқылы жасалған чат ботты құру процессін сипттап бердік. Қазіргі заманда нейронды желілер мен чат боттар қарқынды дамуда, сондықтан бұл жұмыс өзекті деп санаймыз.

Бұл жұмыстың мақсаты студенттерден жиі қойылатын сұрақтарға жауап беретін ботты құрастыру, және ең тиімді текст классификациялау моделін анықтау. Бұдан бөлек, біз бот құрастыру мен нейронды желілердің математикалық жағын ашып баяндадық.

Бірінші тапсырмамыз мәлеметтер қорын құрып, сол қордағы сұрақтарды 10 категорияға бөлу болды. Бұл мақсатымызға жету үшін біз студенттерден ең жиі келетін 10 сұрақ таңдадық. Мәлметтер қорын арттыру мақсатында берілген сұрақтарға ұқсас жаңа сұрақтар құрастырдық. Енді бұл қорды қолданып текст классификациялау моделдерін оқытып, тестілеуге болады. Жұмыс барысында біз үшін ең ыңғайлы келесі үш модель таңдадық: Multinomial Logistic regression, Naive Bayes, Neural Network. Және тестілеу нәтижелері көрсетілген кесте құрастырдық.

Аннотация

Целью работы является создание чат бота, а так же выявление наиболее точной модели классификации текста. Помимо этого, мы описали математическую часть создания чат бота и нейронных сетей.

Первая задача была создание базы данных с необходимым количеством вопросов студентов, которые были разделены на 10 категорий. Для этого мы выбрали 10 наиболее популярных вопросов и занялись разными вариантами перефразированная данных вопросов для обучения и теста наших модели. В ходе работы мы выяснили, что наиболее подходящими для нас моделями классификации текста являются: Multinomial Logistic regression, Naive Bayes, Neural Network. И предоставили таблицу с тестами и результатами каждой модели.

После сбора необходимой базы данных, выявления наиболее эффективной модели классификации текста мы приступили к созданию чат бота через DialogFlow. А так же интегрировали нашего чат бота в мессенджер Телеграмм.

Table of Contents

Chapter 1

Introduction

Modern schooling is incorporating cutting-edge technology like machine learning and artificial intelligence (AI), which include chatbots. The term "chatbot" originates from the fusion of "chat," denoting conversational interaction, and "bot," a concise form derived from "robot" [7]. Chatbots are software applications that use text-based or text-to-speech interfaces to interact with one or more people. Chatbots mimic human communication by conversing. It can get information, process data, customise replies, book restaurants, and even have informal conversations. Typically, chatbots leverage natural language processing techniques, enabling them to understand and generate human-like responses [11]. While they primarily rely on textual interactions, there are variations where structured exchanges resembling spoken language are employed, albeit with certain limitations. Alternatively, chatbots can operate using decision trees or be guided by artificial intelligence [5]. In advanced scenarios, chatbots can go beyond conventional database searches for predefined answers, instead employing vast repositories of previous conversations to autonomously generate contextually appropriate responses for users [5]. The primary objective of chatbots is to engage in conversational interactions, aiming to simulate ordinary human communication as closely as possible. They perform a wide array of tasks, such as information retrieval, data processing, adapting content to user preferences, responding to emails, making restaurant reservations, and even engaging in casual conversation.

An alternative approach to chatbot development involves the utilization of decision trees or the implementation of artificial intelligence (AI) algorithms for control [5]. In the most advanced scenarios, chatbots go beyond the traditional method of searching pre-existing databases for known answers. Instead, they engage in self-communication, drawing upon extensive collections of past conversations as their foundation. Through this process, chatbots leverage the acquired knowledge to generate contextually appropriate responses for users. This sophisticated approach allows chatbots to adapt and evolve their conversational abilities by learning from previous interactions, resulting in more dynamic and personalized interactions with users [7]. By relying on decision trees or AI-based frameworks, chatbots can enhance their performance and provide more advanced and intelligent

responses.

In the present day, individuals maintain constant interaction with technology to access a diverse array of services. Among the most convenient methods is the utilization of automated service delivery processes. This includes self-service ticket offices, online shopping platforms, and various other applications. Additionally, there exists a streamlined automated process through which users can promptly obtain the required information, typically with high accuracy and without the need for direct engagement with a human agent.

This efficient mechanism is made possible through the use of chatbots. These intelligent virtual assistants enable users to swiftly access relevant information, often receiving precise responses without the necessity of interacting with a real person [2]. Chatbots serve as efficient intermediaries, providing instant and accurate assistance to users in a wide range of tasks, from information retrieval to transactional processes. By reducing dependence on human involvement, these automated systems enhance convenience and accessibility, enabling users to obtain the services they require promptly and efficiently. And this process is provided to us by chatbots. Chatbots make it possible to minimize the human factor and thereby improve the quality of service or provision of services.

The integration of bots into systems has led to significant transformation in various processes enhancing the overall user experience, egov.Kz with its efficient system including Telegram Bot has made administrative procedures less burdensome by making it possible for users to obtain certificates or documents within just a few minutes instead of wasting several hours. In addition, online banking platforms such as the Kaspi guide have significantly transformed how people access information on financial services.They are now able to perform various transactions more efficiently with the help of bots providing real-time information and assistance in banking transactions, which has never been easier or quicker.

In addition, integrating chatbots into online platforms has become much more common with their ability to provide quick help and support for users [5]. The main objective of developing these chatbots is to offer quick assistance in addressing queries and ensuring a seamless user experience.

Language models like ChatGPT have broadened the range of possibilities when it comes to designing advanced bots, and interactive communication is made possible through the use of chat-based application known as ChatGPT that leverages on the potential of Conversational AI for generating text content with resemblance to human-like writing.Natural language conversations that are more personalized are possible thanks to the use of large amounts of text data by these bots, and incorporating such bots can potentially revolutionize the way in which people interact with modern technology. This has positive implications for improving overall efficiency and accessibility while also increasing user satisfaction.

Also, chatbots allow you to perform operational actions. For example, detecting errors in the code. In this article [23], the author used CHATGPT as a tool for finding errors in the written code. In addition, he tried to anticipate errors, replace

faulty code with serviceable code, find possible code vulnerabilities, and so on. And the author also compared the bot with other tools for detecting errors in the code.

Among the popular approaches, chatbots are commonly built using two main principles: rule-based and neural networks [7]. Rule-based bots operate based on predefined rules and patterns, while neural network-based bots leverage machine learning techniques for more advanced interactions. Rule-based bots do have certain disadvantages when compared to neural network-based bots [24].

Rule-based bots are limited in their potential, as their responses are constrained by the specific rules and patterns programmed into them. If a user asks a question in a way that deviates from the programmed rules, the bot may struggle to provide a relevant answer. This lack of flexibility can limit the bot's ability to handle a wide range of user queries [24].

Rule-based bots typically lack the capability to improve themselves, learn from interactions, or adapt to individual users. Their responses are predefined and fixed, lacking the ability to evolve and refine their understanding over time. This can hinder their effectiveness in providing personalized and contextually appropriate responses [24].

On the other hand, neural network-based bots, such as those powered by artificial intelligence and machine learning, can offer more advanced capabilities. These bots can learn from data and user interactions, enabling them to adapt and improve their performance over time. They can understand nuances in user queries and generate more accurate and contextually relevant responses. Bots based on neural networks are self-learning, adaptable to different requests, understand the context, have their own personal vocabulary and can develop it. Advanced versions of bots can also save an archive of correspondence with the user, making the work more convenient and efficient. For example, a ChatGPT bot, which, depending on the dialogues with it, can provide different answers [13].

Overall, while rule-based bots have their limitations, neural network-based bots provide greater potential for dynamic and intelligent conversational experiences. They can learn, adapt, and continuously enhance their abilities, leading to more effective and personalized interactions with users.

The purpose of this dissertation is to create an effective chatbot capable of answering students' questions about the university and the student life of the university. I think this is an actual problem, because sometimes you have to wait for weeks to get a response to your request. After all, the adviser is loaded for students. And being a freshman without answers is very hard. The chatbot that we will create will be based on neural networks and integrated in telegram, and we will be able to train it to answer the most popular questions of students.

Why we choose Telegram? In this article [25] author compared telegram, WhatsApp and Viber. Author compared this three messangers in four criteria: general information, users, contacts and security. In general information we got these results: WhatsApp 4/4, Telegramm 4/4 and Viber 2/4. In users we got: WhatsApp 1/4, Telegram 4/4 and Viber 0/4. From the contacts results was: WhatsApp 2/4,

Telegram 4/4 and Viber 1/4. And security results: WhatsApp 2/3, Telegram 3/3, Viber 0/3. Afterall here the results: Whatsapp got 9 points, Telegram 15 and Viber 3. So this shows us why Telegram is better.

In addition, we will describe the math part of chat bot creation. We will talk about what math formulas, theorems and etc. is behind text classifications models like Naive Bayes, Logsitic Regression, Neural network [8].

Chapter 2

Literature review

2.1 History of artificial intellegence

Can machines think? In 1950, Alan Turing asked this question in his article "Computing machinery and intelligence", and since then there have been many attempts to answer it in research in the field of artificial intelligence, including research related to chatbots [26].

The development of chatbot technology began in the 1960s. Joseph Weizenbaum, a professor at the Massachusetts Institute of Technology, developed the Eliza chatbot, which imitated a patient's dialogue with a psychotherapist [19]. Eliza's communication with the interlocutor was rather primitive: the program studied the user's message, identified keywords and, based on the found keyword, transformed the message, basically simply paraphrasing the user's message. If Eliza did not find a keyword in the database, she either gave a general answer or repeated one of the previous messages. Such poor "communication" was due to a lack of memory for storing previous conversations, the limitations of the database, as well as the limitations of methods that allow identifying keywords and combining them, structuring the chatbot's knowledge using decision trees and pre-existing scenarios created by developers.

Nevertheless, the principles used in Eliza provided the basis for determining the structure of the following chatbots: this is the presence of keywords, specific phrases and pre-programmed answers [19].

As for the term "chatbot" ("ChatterBot"), it was introduced by Michael Mauldin in 1994 to describe such dialog programs [15].

An important milestone in the history of chatbots was the appearance in 1995 of the chatbot Alice (A.L.I.C.E. - Artificial Linguistic Internet Computer Entity - artificial linguistic Internet computer being), developed by Richard Wallace. Unlike Eliza, Alice could enter into a dialogue with a human by applying some heuristic rules of natural language processing, comparing expressions with samples. The program was open source, and based on the chatbot A.L.I.C.E. developers could

create their own chatbots using AIML (Artificial Intelligence Markup Language) [1].

Another chatbot whose appearance played an important role in the development of technology was Jabberwacky, created in 1988 by British programmer Rollo Carpenter and launched on the Internet in 1997 [6]. The stated goal of this project was to create an artificial intelligence capable of passing the Turing test. Jabberwocky was designed to simulate a natural human conversation in an interesting, entertaining and humorous manner. Unlike more traditional artificial intelligence programs, the technology used in Jabberwacky was intended for entertainment, and not for use in computer support systems .

Another interesting chatbot— Mitsuku, was created using AIML technology by Steve Worswick [4]. The developer claims that this is an 18-year-old female chatbot from Leeds, England. The Mitsuku database contains all of Alice's AIML files, with a lot of additions from user conversations. The chatbot is always in the process of working. Mitsuku's intelligence includes the ability to reason in application to specific objects. For example, if someone asks: "Can you eat the whole house?", Mitsuku searches for properties for the "house" and answers, "No," since the house is not edible. She can play games and do various "tricks" at the request of the user. In 2015, she communicated on average more than a quarter of a million times a day [16].

The development of chatbot technology can be divided into three main stages [17], each representing significant advancements and capabilities:

- The first stage, which occurred in the 1970s and 1980s, marked the beginning of research into creating natural language interfaces. However, during this period, the interfaces were limited in their ability to simulate conversations between two people. The focus was on exploring the possibilities and limitations of natural language processing and understanding;
- The second stage coincided with the popularization of the Internet. With the advent of the Internet and its widespread adoption, conversations multiplied, and it became possible to communicate with a large number of users simultaneously. This level saw the emergence of chatbots that might interact with heaps of customers simultaneously, catering to their queries and providing records or assistance;
- In the third stage, chatbots have advanced to include a aggregate of advanced technologies. herbal language processing and expertise capabilities have advanced, permitting chatbots to have greater sophisticated and context-aware conversations. additionally, advancements in speech synthesis have enabled chatbots to communicate through spoken language. furthermore, the combination of real-time video capabilities has superior the interactive and immersive revel in of chatbot interactions. The 1/3-level chatbots leverage a combination of natural language technologies, speech synthesis, and actual-time video to offer customers with extra flexible and attractive stories. they are able to understand and reply to customers' queries in real-time, at the

same time as additionally incorporating multimedia factors like speech and video to complement the interaction.

The evolution of chatbot technology has progressed from simulating basic conversations to facilitating widespread online interactions, and finally to the integration of advanced features such as natural language processing, speech synthesis, and real-time video, enabling more immersive and dynamic chatbot experiences.

2.2 Main components and types of chatbots

A chatbot must have the following main components [10] in order for a dialogue to take place between it and a person:

- conversational artificial intelligence is the basis of chatbots, thanks to it all the control and processing of natural language expressions takes place. The first chatbots only recognized patterns and rules and interpreted them. Modern advanced chatbots use deep learning processes to analyze a person's speech, study previous conversations and compose the most appropriate response;
- database of previous conversations: a database of previous conversations allows chatbots to create optimal versions of answers, ensuring a natural, intelligent, and consistent conversation. By leveraging historical data, chatbots can learn from past interactions and improve their conversational abilities over time;
- interface: The interface serves as a way of interaction between a person and a chatbot. Therefore, the interface should be intuitive, meaningful and convenient. The interface allows you to interact with the chatbot verbally or in writing.
- understanding the dialogue: in order for the chat bot to correctly and correctly answer the question, the bot must be able to understand the context of the question. In other words, the bot needs to understand the user's intention at least a little. The chatbot needs to understand "Why I am being asked this or that question, and what they want to hear from me." To achieve this goal, it is necessary to use NLP technology, response generation and AI.

By incorporating these components, a chatbot can engage in effective and meaningful dialogues with users. Conversational artificial intelligence provides the foundation for processing natural language, while a database of previous conversations helps maintain consistency. The user interface enables seamless interaction, and the combination of natural language processing, speech recognition, response generation, and artificial intelligence enables the chatbot to interpret user intent and generate appropriate responses.

From this article [7] we know that based on user experience with chatbots, chatbots can be divided into the following categories:

- Menu/Button-based Chatbots: This type of chatbot is the easiest to create and use. It typically consists of a vertical menu with different buttons. The functionality of the chatbot is limited to what the creator has programmed. It may not be capable of handling complex or unusual tasks and is primarily suitable for simple interactions;
- Keyword Recognition-Based Chatbots: These chatbots utilize artificial intelligence techniques. Initially, the chatbot familiarizes itself with the data provided by the user. Then, using various algorithms, the chatbot identifies and highlights keywords from the user's query. Based on these keywords, the chatbot formulates a response. However, if there is an excessive number of keywords within a single sentence, the chatbot may not interpret the query correctly;
- Contextual Chatbots: This type represents the most advanced category of chatbots available today. Contextual chatbots leverage machine learning and artificial intelligence technologies. Their primary objective is to understand the user's intention when asking a question and provide an appropriate response based on that understanding. These chatbots continuously learn and improve their capabilities over time, enabling them to handle non-standard user requests and adapt to different contexts.

These categories reflect the varying levels of sophistication and capabilities exhibited by chatbots. While menu/button-based chatbots are simpler and have limited functionality, keyword recognition-based chatbots offer more intelligence by identifying keywords. Contextual chatbots, on the other hand, employ advanced technologies and self-learning mechanisms to understand user intent and provide tailored responses.

2.3 Modules used by chat bots to respond to user requests and examples

In article [5] it is clear that to work with the user's request chatbots are use these modules: NLP, NLU, Dialogue Manager, Data Sources, Response Generator.

Natural Language Processing (NLP) is a branch of artificial intelligence that focuses on the interaction between computers and human language [5]. It involves the processing and analysis of natural language text or speech to understand and extract meaning from it. NLP techniques enable machines to understand, interpret, and generate human language.

NLP encompasses various tasks and techniques, including:

Lemmatization: It is the process of reducing words to their base or dictionary form, known as the lemma [5].

Lemmatization: Linguistic rules and algorithms convert each word to its lemma based on the part-of-speech tag. Suffixes, prefixes, and word ends determine the base form.

Lemmatization output: The converted words are returned as their base or root form. Lemmatization uses language rules, stemming algorithms, and pattern matching, not arithmetic. These methods use morphology and context to determine word roots.

Because languages have various laws and structures, lemmatization is language-dependent. Lemmatization methods and implementations differ per language.

Lemmatization does not use formulas from mathematics. Some lemmatization algorithms employ statistical or probabilistic models. Two common methods:

Rule-based Lemmatization [10]:

Rule-based lemmatization relies on a set of linguistic rules or patterns to transform words into their base forms. These rules are often language-specific and are created based on linguistic knowledge. While there are no mathematical formulas involved, the rules may include regular expressions or pattern matching techniques to identify and modify word endings or suffixes.

Example rule:
If a word ends in "-s" and is tagged as a plural noun, remove the "-s" to obtain the base form.
Word: cats (plural noun)
Lemma: cat

Statistical Lemmatization [10]: Statistical lemmatization approaches leverage statistical models, such as hidden Markov models (HMMs) or conditional random fields (CRFs), to determine the most probable lemma for a given word. These models are trained on annotated corpora that provide word-to-lemma mappings. The probability of a lemma is calculated based on the observed frequencies of word-lemma pairs in the training data.

Example statistical model:

$$P(lemma|word) = Count(word, lemma)/Count(word) \tag{2.3.1}$$

This formula calculates the probability of a lemma given a word based on the frequency counts in the training data [10].

While these examples illustrate some aspects of the computational linguistics methods employed in lemmatization, it's important to note that the specific algorithms and formulas used can vary across implementations and languages. The focus in lemmatization is primarily on linguistic rules and statistical patterns rather than traditional mathematical formulas.

Tokenization [5]: It involves breaking down a text into smaller units, called tokens, such as words, phrases, or sentences. Tokenization allows for easier analysis and manipulation of text data.

Stemming [5]: It is a technique for reducing words to their root form by removing suffixes and prefixes. Stemming helps in reducing words to their basic form

to capture the core meaning and improve processing efficiency. From this article we know [3] that in chatbot architecture we have natural language understanding (NLU) methods. After the NLP stage has passed, we proceed to the NLU. This processing process is used to work with information that has already been entered by the user. And without NLP, this technology will not be able to be used. Because to use NLU, it is necessary that the model can understand a person. Currently, six methods are used: classification, dialogue planning, name entity recognition, vector recognition with cosine similarity, lexicon and long short-term memory.

Dialogue Manager: It is responsible for managing the flow of conversation between the chatbot and the user. The dialogue manager keeps track of the conversation context, maintains state, and decides appropriate actions or responses based on the current conversation context [5].

Data Sources. After processing the information in the dialog manager, the bot extracts it and then stores it. The database can be external and internal. Usually, possible answers are generated at this stage [5].

Response Generator: It is responsible for generating appropriate responses based on the input and dialogue context. The response generator can use predefined templates, machine learning models, or rule-based approaches to generate relevant and coherent responses [5].

The article [3] discusses the development of a chatbot specifically designed to answer questions related to the engineering field. The chatbot utilizes the Natural Language Processing (NLP) method and is implemented using the JSON format. One of the key advancements in this chatbot is the integration of a neural model directly connected to the university data, which helps to address the retraining issue. Additionally, the chatbot's live updating capability allows for real-time enhancements and improvements.

To ensure usability, the chatbot is deployed on the Facebook Messenger platform, where users can interact with it and receive responses within their Messenger interface.

The performance of the chatbot was evaluated through testing conducted in two stages [3]. In the initial stage, the accuracy of the answers was reported to be 0.46. However, in the subsequent stage, by increasing the number of keywords and training phrases, the accuracy improved significantly to 0.76. This highlights the importance of having a comprehensive and diverse database to enhance the accuracy of response predictions.

The article [3] emphasizes the significance of data collection and the direct correlation between the quantity of data and the accuracy of the chatbot's responses. By gathering and incorporating more data, the accuracy of the chatbot can be improved. The authors of the article [3] faced a similar challenge regarding accuracy and resolved it by increasing the volume of data. Further details regarding their approach to addressing this issue are likely discussed later in the article.

In this article [9], the author created a bot called Freudbot. The purpose of

creating this bot was to determine the impact of the chat bot on the perception of learning material during distance learning. AIML structure with source code was used in the creation of this bot. Fifty-three psychology students participated in this experiment. The bot was able to provide instruction on two specific questions. Students who participated in the experiment interacted with the bot for 10 minutes, and then filled out a questionnaire. The author, who observed the results of the questionnaires, evaluated the work of the bot neutrally. However, the students gave a positive assessment of the bot, and approved of the development of this bot in the long term, and even pointed out the direction for development. In general, the experiment showed that the chat bot can be useful in distance learning.

Also, bots mostly take information from public sources, and are designed to answer questions from the same sources. But the author of this article [20] tried to create a bot for generating questions and answering them. That is, the bot must get a certain set of data in question/answer format, and then go through a process of cleaning and preprocessing. Then generate a quiz based on this database. Prior to that there were no attempts to make such a bot, and did not exist. This bot is useful in education, and helps you quickly and efficiently create a quiz or test to check the progress of students.

CapacitaBOT [14], was created in 2022 to help people who have intellectual disabilities. If a person does not have the full ability to use his or her intellect, due to some disability, it is not a good thing. But if such a person cannot go out and develop his or her social skills because of a disability (e.g. COVID 2019), that's too bad. And this bot was designed to help people with disabilities develop their social skills through communication with the bot. The bot teaches the person how to behave in different situations, supports and gives hints if the person has a question regarding these situations.

2.4 Math part of chatbots

And now about the mathematical part of our work [8]. In the creation of chatbots the most commonly used are the following sections:

- Algorithms and data structures: algorithms are crucial in natural language processing and text analysis. Designing and optimizing algorithms for input processing, keyword searching, classification, and text mining are important aspects of chatbot development.
- Probability and statistics: probabilistic methods and statistical models are used to analyze and predict language structures, understand context, and classify text. Bayesian networks, hidden Markov models, and statistical-based machine learning methods are often used to train chatbots to recognize and generate text.
- Linguistics and grammar: understanding language is the basis for developing effective chatbots. Formal grammar, semantics, syntax, and morphology help in text analysis and processing. Concepts such as lemmatization, stemming,

and partitioning are often used to extract meaning and context from textual data.

- Linear algebra: linear algebra is applied when working with vector representations of words and text. Methods such as word2vec and GloVe use matrix operations and calculations to create semantic vector representations that allow models to understand semantic relationships between words.
- Machine learning and deep learning: machine learning techniques, including neural networks and deep learning, are widely used to classify text and create chatbots. Algorithms such as recurrent neural networks (RNNs), convolutional neural networks (CNNs), and transformers are used to analyze and generate text, determine tone, categorize, and respond to queries.

2.5 Chatbots in education

Let's consider some chatbots used for educational purposes. We have these chatbots: AniBot, BetterBot, CEU, CourseO, Differ, Duolingo [10].

Ani. A chatbot designed for teaching and capable of replacing human teachers when performing certain tasks. Its goal is to provide personalized learning and mentoring that helps students show commitment and involvement in the learning process [10]. The chatbot includes elements of motivation, evaluation and immediate feedback, it is able to adapt to the user's needs using automatic learning algorithms. The chatbot also contains a tutoring course for learning English.

Better. This is a physical robot, which is being tested at the University of Oberta de Catalunya [10]. Better is a chatbot that is designed to help and support students in monitoring their learning progress. He is able to interact with students using light signals, sound messages (motivating and frustrating sounds and phrases) and movements. Using such a chatbot implies a new way of communication between the campus and the student. The chatbot works as a cognitive learning technology, especially in that it contributes to changing the behavior of students.

CEU. Since 2017, a Microsoft Azure-based chatbot has been used to mentor students and respond to their requests without delay with constant availability [10]. Currently, he acts as a personal assistant to answer administrative requests, but the goal of the developers is to make him more active in the future, able to predict the behavior of students and advise them throughout the learning process.

CourseQ. Developed at Cornell University (USA) [10]. It can integrate with LMS and virtual learning environments (for example, with Moodle). The functions of the chatbot include receiving information for teachers and students, as well as providing reminders about application dates, schedules, materials and events. The chatbot database consists of text messages shared by teachers.

Differ. Used at the Norwegian BI Business School [10]. He is able to create communities that unite students in similar situations. The chatbot publishes rel-

evant messages and reminders in order to increase student engagement, as well as create a space where students do not feel judged because of the questions they ask. Duolingo. Designed for learning languages, conversation is used in communication along with gamification methods.

Chapter 3

Methodology

3.1 Used libraries

Libraries that we used in our work: "pandas", "numpy", "nltk", "sklearn", "gensim", "dialogflow". Let's discuss them and identify their features.

The pandas library is a powerful and popular open-source data manipulation and analysis tool for Python. It provides high-performance, easy-to-use data structures, such as dataframes, and a wide range of functions for working with structured data.

Key features:

- Data Structures: The core data structure in pandas is the dataframe, which is a two-dimensional tabular data structure with labeled axes (rows and columns). It allows for efficient handling and manipulation of structured data, similar to working with a spreadsheet or SQL table.
- Data Manipulation: Pandas offers a rich set of functions for data manipulation, including filtering, sorting, grouping, merging, reshaping, and aggregating data. These operations can be performed on dataframes, allowing for efficient data transformations and analysis.
- Data Cleaning and Preprocessing: Pandas provides tools for handling missing data, removing duplicates, handling outliers, and transforming data types. It simplifies the process of data cleaning and preprocessing, making it easier to prepare data for analysis.
- Data I/O: Pandas supports reading and writing data in various formats,including CSV, Excel, SQL databases, and more. It provides functions to import data from external sources and export processed data to different formats.
- Integration with NumPy: Pandas is built on top of the NumPy library, leveraging its fast and efficient numerical operations. It seamlessly integrates with NumPy arrays and provides additional functionality for data analysis.

- Visualization: Although pandas does not provide built-in visualization capabilities, it integrates well with popular visualization libraries such as Matplotlib and Seaborn. This allows users to create insightful plots, charts, and graphs based on the data stored in pandas dataframes.

Pandas is used in various domains, including data analysis, finance, scientific research, and machine learning. Its intuitive syntax and extensive functionality make it a valuable tool for working with structured data in Python.

NumPy (Numerical Python) is a fundamental open-source library for scientific computing in Python. It provides powerful tools for working with large, multi-dimensional arrays and matrices, along with a collection of mathematical functions to operate on these arrays efficiently. NumPy serves as the foundation for numerous other libraries and applications in the Python ecosystem.

Key Features:

- Multi-dimensional Arrays: NumPy's main feature is its ndarray (n-dimensional array) object, which allows for efficient storage and manipulation of homogeneous data. Arrays in NumPy can have any number of dimensions and are highly optimized for numerical operations.
- Mathematical Functions: NumPy provides a comprehensive set of mathematical functions for performing operations on arrays. These include basic arithmetic operations, mathematical functions (such as trigonometric, exponential, and logarithmic functions), linear algebra operations, statistical functions, and more.
- Broadcasting: NumPy enables broadcasting, which allows for performing arithmetic operations between arrays of different shapes and sizes. This feature simplifies code and eliminates the need for explicit loops, making it easier to work with arrays of different dimensions.
- Indexing and Slicing: NumPy offers powerful indexing and slicing capabilities to access and manipulate data within arrays. It allows for selecting specific elements, subsets, or entire sections of arrays based on various conditions or criteria.
- Array Manipulation: NumPy provides functions to reshape, resize, transpose, concatenate, split, and stack arrays. These operations enable flexible and efficient manipulation of array data.
- Integration with C/C++ and Fortran: NumPy is designed to seamlessly integrate with existing C, C++, and Fortran codebases. It provides tools for easily interfacing with low-level languages, allowing for efficient computation and integration with other scientific computing libraries.
- Performance Optimization: NumPy's underlying implementation is highly optimized, leveraging efficient algorithms and memory management techniques. It performs computations on arrays much faster than traditional Python lists, making it a preferred choice for computationally intensive tasks.

- Integration with Scientific Libraries: NumPy is a foundational library in the Python scientific computing ecosystem and integrates well with other libraries such as SciPy, Pandas, Matplotlib, and scikit-learn. This integration enables a seamless workflow for data analysis, numerical simulations, visualization, and machine learning.

NumPy is used in scientific research, data analysis, machine learning, and simulation. Its efficient array operations, broad mathematical functions, and integration with other libraries make it an essential tool for numerical computing in Python.

Scikit-learn, also known as sklearn, is a popular open-source machine learning library for Python. It provides a wide range of tools and algorithms for data preprocessing, feature extraction, model training, model evaluation, and prediction. Sklearn is built on top of other scientific computing libraries like NumPy, SciPy, and matplotlib, and it integrates well with the Python data science ecosystem.

Key Features:

- Supervised Learning: Sklearn supports various supervised learning algorithms, including regression, classification, and ensemble methods. These algorithms aim to learn patterns and relationships in labeled training data to make predictions or classify new unseen data points.
- Unsupervised Learning: Sklearn provides several unsupervised learning algorithms, such as clustering, dimensionality reduction, and anomaly detection. These algorithms aim to discover patterns, structures, or anomalies in unlabeled data without explicit guidance.
- Model Selection and Evaluation: Sklearn offers tools for model selection, hyperparameter tuning, and model evaluation. It provides techniques like cross-validation, grid search, and scoring metrics to assess the performance and generalization of machine learning models.
- Preprocessing and Feature Extraction: Sklearn provides a wide range of preprocessing techniques to prepare data for machine learning tasks. These techniques include scaling, normalization, encoding categorical variables, handling missing values, and feature extraction methods like Principal Component Analysis (PCA).
- Ensemble Methods: Sklearn includes ensemble learning methods such as Random Forests, Gradient Boosting, and AdaBoost. These methods combine multiple individual models to create a stronger, more accurate model.
- Model Persistence: Sklearn allows models to be saved to disk and loaded later for reuse or deployment. This feature enables developers to train models on large datasets and use them in production environments.
- Integration with Other Libraries: Sklearn integrates seamlessly with other Python libraries like NumPy, SciPy, and pandas. This integration facilitates data manipulation, feature engineering, and model evaluation within a

unified data science workflow.

Math behind the scikit-learn: Sklearn utilizes various mathematical formulas and algorithms for machine learning tasks. Here are a few examples:

1. Linear Regression: The formula for simple linear regression can be represented as

$$y = mx + b \tag{3.1.1}$$

where y is the dependent variable, x is the independent variable, m is the slope of the line, and b is the y-intercept.

2. Logistic Regression: The logistic regression formula involves the logistic function, which maps the linear combination of input features to a probability value between 0 and 1. It can be expressed as

$$p = 1/(1 + e^{-z}) \tag{3.1.2}$$

where p is the probability, and z is the linear combination of input features.

3. Support Vector Machines (SVM): SVM aims to find a hyperplane that separates data points of different classes with the maximum margin. The formula for the decision function in SVM can be written as

$$f(x) = sign(wx + b) \tag{3.1.3}$$

where f(x) is the predicted class, w is the weight vector, x is the input feature vector, and b is the bias term.

4. • Sklearn incorporates these mathematical formulas and algorithms to implement various machine learning techniques effectively. It provides a user-friendly interface to apply these algorithms and perform complex data analysis tasks with ease.

Gensim is a popular open-source library for natural language processing (NLP) and topic modeling in Python. It provides a range of algorithms and models for unsupervised learning, text representation, similarity calculation, and topic modeling. One of the key features of Gensim is its implementation of Word2Vec, a widely used algorithm for learning word embeddings.

Word2Vec is a neural network-based model that learns word embeddings, representing words as dense vectors in a continuous vector space. Gensim's implementation of Word2Vec has two variants: Continuous Bag of Words (CBOW) and Skip-gram.

CBOW: The CBOW model predicts a target word based on its surrounding context words. The objective is to maximize the conditional probability of the target word given the context words. The formula for CBOW can be represented as:

$$\sum \log P(w|c) \tag{3.1.4}$$

where w is the target word and c represents the context words.

Skip-gram: The Skip-gram model predicts the surrounding context words given a target word. The objective is to maximize the average log probability of context words given the target word. The formula for Skip-gram can be represented as:

Example: Let's consider a simple corpus with the following sentence: "I love to learn natural language processing." Using the Skip-gram model, Gensim can learn word embeddings where similar words are represented by vectors that are close in the vector space. For instance, after training, the vectors for "love" and "learn" might have high cosine similarity, indicating that these words have similar contexts and meanings.

Latent Semantic Analysis (LSA):

LSA is a technique used for dimensionality reduction and topic modeling. It represents documents and terms as vectors in a high-dimensional space and captures the latent semantic structure of the corpus.

Singular Value Decomposition (SVD): LSA uses SVD to reduce the dimensionality of the document-term matrix. SVD factorizes the matrix into three matrices: U, Σ, and VT. The formula for LSA is:

$$M \approx U\Sigma V^T \tag{3.1.5}$$

where M is the document-term matrix, U and VT are orthogonal matrices, and Σ is a diagonal matrix with singular values.

Example:
Suppose we have a corpus of documents related to sports. By applying LSA using Gensim, we can identify latent topics and extract key concepts from the corpus. The LSA model can reveal topics such as "football," "basketball," and "tennis" based on the co-occurrence patterns of words within the documents.

These are just two examples of models available in Gensim. The library also provides other models like TF-IDF, Latent Dirichlet Allocation (LDA), and Doc2Vec, each with its own mathematical foundations and applications. Gensim simplifies the implementation of these models and allows users to perform various NLP tasks, including text similarity calculation, document clustering, and topic modeling, with ease and efficiency.

The Dialogflow Python library lets Python developers use the Dialogflow API. It handles authentication, agent, intent, context, entity modification, and events. Using a service account JSON key file, pip installs the library. The library lets you query Dialogflow agents, get answers, manage conversational contexts, design and handle intents, deal with entities, and trigger events. It simplifies Dialogflow's natural language comprehension into Python apps for interactive chatbots.

Here is the code (Fig 3.1):

```
import pandas as pd
import numpy as np
import dialogflow
#for text pre-processing
import re, string
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer
from nltk.corpus import wordnet
from nltk.stem import WordNetLemmatizer
nltk.download('punkt')
nltk.download('averaged_perceptron_tagger')
nltk.download('wordnet')
#for model-building
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, f1_score
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.metrics import roc_curve, auc, roc_auc_score
# bag of words
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.feature_extraction.text import CountVectorizer
#for word embedding
import gensim
from gensim.models import Word2Vec
```

Figure 3.1: Imported libraries

3.2 Data collection and preprocessing

In the initial stage of data collection, we began by selecting the 10 most relevant and frequently asked questions from the university's FAQ. This resulted in 10 distinct classes of questions. To expand the dataset, each question was manually rephrased, leading to a total of 100 questions. However, this quantity was still considered insufficient for comprehensive testing of classification models.

To address this limitation, we employed a chatbot developed by OpenAI known as ChatGPT. ChatGPT is a powerful language model that falls under the umbrella of conversational AI. It has been trained on a vast corpus of text data, enabling it to generate human-like text responses. By leveraging ChatGPT, we were able to significantly increase the size of our question database. This collaboration allowed us to expand the dataset to encompass 598 questions, which were further categorized into the original 10 classes. You can refer to Fig 3.1 for a detailed breakdown of these question classes.

The inclusion of ChatGPT in our data collection process played a crucial role in enhancing the comprehensiveness and diversity of the dataset. This enlarged dataset forms a solid foundation for the training and evaluation of classification

models, empowering us to develop more robust and accurate systems for question classification and response generation.

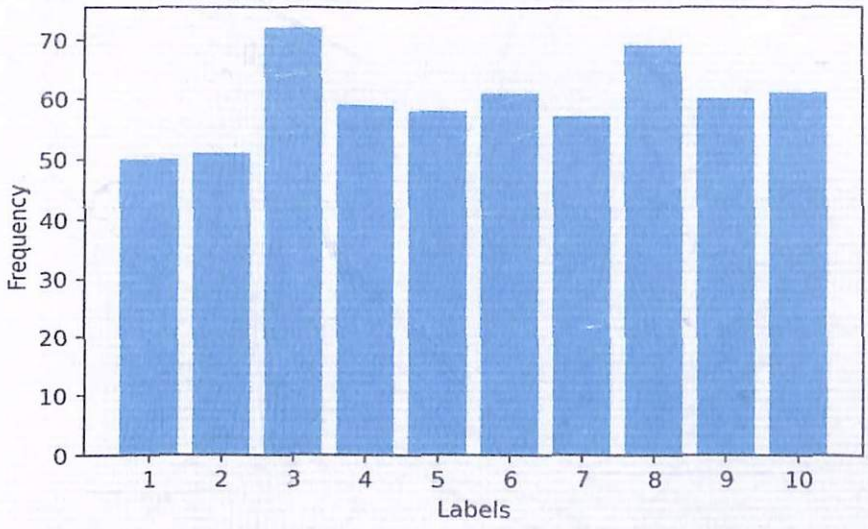

Figure 3.2: Frequencies of labels

Then, we started preprocessing the data with NLTK libraries.

This is our code for preprocessing the data (Fig 3.3):

As you can see here, we conberted text to lowercase, removed stopwords and after lemmatized and tokenized our sentences.

NLTK, short for the Natural Language Toolkit, is a comprehensive collection of libraries and programs designed for symbolic and statistical natural language processing (NLP) tasks. Specifically tailored for English, NLTK is implemented in the Python programming language and was originally developed by Steven Bird and Edward Loper at the University of Pennsylvania's Department of Computer and Information Science.

The NLTK package not only provides a wide range of functionalities for NLP but also includes graphical demonstrations and sample data, making it a versatile resource for researchers, educators, and practitioners in fields such as empirical linguistics, cognitive science, artificial intelligence, information retrieval, and machine learning. Its accompanying book delves into the foundational concepts underlying various language processing tasks supported by the toolkit, while the cookbook offers practical guidance and examples.

NLTK serves as a powerful tool for research, teaching, individual study, and the development of research systems. It has been widely adopted in academic settings, with 32 universities in the United States and 25 countries around the world incorporating NLTK into their courses. Among its notable features, NLTK supports essential NLP functionalities, including classification, tokenization, stemming, tagging, parsing, and semantic reasoning. Its versatility and extensive capabilities

```
def preprocess(text):
    text = text.lower()
    text=text.strip()
    text=re.compile('<.*?>').sub('', text)
    text = re.compile('[%s]' % re.escape(string.punctuation)).sub(' ', text)
    text = re.sub('\s+', ' ', text)
    text = re.sub(r'\[[0-9]*\]',' ',text)
    text=re.sub(r'[^\w\s]', '', str(text).lower().strip())
    text = re.sub(r'\d',' ',text)
    text = re.sub(r'\s+',' ',text)
    return text
def stopword(string):
    a= [i for i in string.split() if i not in stopwords.words('english')]
    return ' '.join(a)
wl = WordNetLemmatizer()
def get_wordnet_pos(tag):
    if tag.startswith('J'):
        return wordnet.ADJ
    elif tag.startswith('V'):
        return wordnet.VERB
    elif tag.startswith('N'):
        return wordnet.NOUN
    elif tag.startswith('R'):
        return wordnet.ADV
    else:
        return wordnet.NOUN
def lemmatizer(string):
    word_pos_tags = nltk.pos_tag(word_tokenize(string))
    a=[wl.lemmatize(tag[0], get_wordnet_pos(tag[1])) for idx,
    tag in enumerate(word_pos_tags)]
    return " ".join(a)
```

Figure 3.3: Preprocessing the data

make it a popular choice for NLP practitioners and researchers seeking to explore and implement various language processing tasks.

We used "stopword" function. The concept of "stop words" in natural language processing (NLP) involves removing commonly occurring words that are considered insignificant for analysis or do not contribute much to the overall meaning of a text. Examples of stop words include articles (e.g., "a", "an", "the"), prepositions (e.g., "in", "on", "at"), and conjunctions (e.g., "and", "or", "but").

Mathematically, the process of identifying and removing stop words is not based on mathematical calculations or formulas. Instead, it relies on a predetermined set of stop words that are typically compiled through linguistic analysis and domain expertise. This set of stop words can vary depending on the specific language and the context of the text being processed.

When implementing the stop word function in an NLP system, a comparison is made between each word in the input text and the set of predefined stop words. If a match is found, the word is considered a stop word and is subsequently removed from the text. This comparison is typically done using simple string matching techniques or hash table lookups for efficiency. The rationale behind removing stop words is to reduce the dimensionality of the text data and focus on the more meaningful and significant words for downstream NLP tasks such as text classification, sentiment analysis, or information retrieval. By eliminating these

common and less informative words, the computational resources can be better utilized and the accuracy of the analysis can be improved.

The function takes the input string and splits it into words using the split() method.

The "split" function is a commonly used operation in programming languages and is not directly associated with mathematics. It is primarily used to divide a string of characters into smaller parts or tokens based on a specified delimiter. Mathematically, the "split" function involves a straightforward process. Given an input string, the function scans the characters of the string and identifies occurrences of the specified delimiter. When a delimiter is found, it marks the end of a token and starts a new token. This process continues until the entire string has been processed. "Split" function does not involve complex mathematical calculations or algorithms. It is typically implemented using simple string manipulation techniques, such as iterating over the characters of the input string and checking for the delimiter. Once the delimiter is encountered, the function stores the extracted token and proceeds to the next part of the string.

The result of the "split" function is a collection of tokens or substrings that were separated by the specified delimiter. These tokens can then be used for further processing or analysis, such as counting word occurrences, creating lists, or extracting meaningful information from the input string. The resulting list of words is filtered using a list comprehension, removing any words found in the English stopword list provided by the Natural Language Toolkit (NLTK) library.

The final result is a string of words combined using the join() method. After, we lemmatized the text by using lemmatizer().

Now, we got the data that converted to lowercase, with removed punctuations and stopwords, and lemmatized. And now we need to classify it.

3.3 Testing of text classification models

Classifying text data by theme is determining whether text data belongs to a topic that the text deals with. The most common tasks of text data classification are determining the emotional coloring of text and classifying text data by topic. Classification by topic (multi-class classification) is often used to filter text data when it is necessary to cut out entries related to topics that are not of interest for analysis.

There are several approaches to classifying text. The first one - manual - is the manual determination of the document class. This method is accurate, but has a major drawback - the inability to process large amounts of data in a reasonable amount of time. The second approach is to write rules based on regular expressions. This approach consists in that a text classifier makes a set of rules on the basis of regular expressions, which allows you to process large amounts of data. However, the creation of such rules requires efforts to create and keep the rules up-to-date

on the part of the specialist. Moreover, before defining the rules, a specialist has to be deeply familiar.

The third approach is based on the machine learning approach [2], but creating such rules requires an effort to create and maintain the rules. The third approach is based on machine learning. In this approach, the class dependence on the sample text is determined automatically. This approach requires a preliminary manual marking of training data, but it is a simpler task than determining the rules of belonging of all samples to classes. This approach is currently the most used and promising, since it requires the least amount of human effort and has the ability to work automatically with large amounts of data [21].

Text classification based on machine learning is represented by several basic algorithms.

We used Naive Bayes, Logistic Regression, and Neural Network algorithms. Naive Bayes is a probabilistic model based on Bayes' theorem, assuming independence between features.

This method is a probabilistic classification method based on Bayes' theorem with some additions. Bayes' theorem gives the relation between the probabilities of two events and their conditional probabilities. The naive Bayesian classifier assumes that the presence or absence of a certain property of a class has no relation to the presence or absence of other properties. Suppose an object can be classified by attributes such as color, shape, and mass. A reasonable classification for a spherical yellow object weighing less than 60 grams might be a tennis ball. Even if these properties actually depend on each other or are dependent on some other property, a Naive Bayesian classifier would assume that all of these properties have independent contributions to the object being a tennis ball.

Let's go deeper into the algorithm for calculating this method. Let $P(c_i|d)$ be the probability that the document represented by vector $d = (t_1, \ldots, t_n)$ corresponds to the category c_i for $i = 1, \ldots, |C|$. The task of the classifier is to choose such values of c_i and d, at which the probability $P(c_i|d)$ will be maximal:

$$CSV(d) = argmax\ P(c_i|d) \tag{3.3.1}$$

To calculate the values of $P(c_i|d)$ use Bayes' theorem:

$$(c_i|d) = \frac{P(c_i)\,P(d|c_i)}{P(d)} \tag{3.3.2}$$

where $P(c_i)$ is the a priori probability that a document is assigned to category c_i; $P(d|c_i)$ is the probability to find a document represented by a vector $d = (t_1, \ldots, t_n)$ in category c_i; P(d) is the probability that an arbitrary document can be represented as a feature vector $d = (t_1, \ldots, t_n)$.

In essence, $P(c_i)$ is the ratio of the number of documents from the training sample L assigned to category c_i to the number of all documents from L.

P(d) does not depend on the category c_i, and the values $t_1, \ldots, t_n$ are pre-specified, so the denominator is a constant that does not affect the choice of the largest value of $P(c_i|d)$.

Calculating $P(d|c_i)$ is difficult because of the large number of features $t_1, \ldots, t_n$, so one makes the "naive" assumption that any two coordinates considered as random values do not statistically depend on each other.

Then we can use the formula

$$(d|c_i) = \prod_{k=1}^{n} P(t_k|c_i) \tag{3.3.3}$$

Then all probabilities are calculated using the maximum likelihood method. It is simple and easy to implement and is particularly well-suited for text classification tasks where the focus is on identifying the presence or absence of certain keywords. We used the Naive Bayes model with a term frequency-inverse document frequency (tf-idf) vectorizer. TF-IDF (term frequency-inverse document frequency) is a numerical statistic that reflects how important a word is to a document in a corpus. Our code instantiates a Multinomial Naive Bayes Model. Then fits the model on training data using arrays. Makes predictions on the test data and stores the predicted classes. Calculates the class probabilities for each test sample and stores them. After, print a classification report to evaluate the model's performance and a confusion matrix to visualize the accuracy of the predictions [2].

On the other hand, Logistic Regression is a linear model commonly used for binary classification problems. It provides a probability of an instance belonging to a certain class and is widely used for text classification tasks where the focus is on the relationship between features and the target variable. We will use a multinomial version of the model for our purposes. Our Logistic Regression does the same things as Naive Bayes: fits the model on training data, makes predictions, and shows results. But here, we used a multi-class strategy, called One-vs-Rest. The "One-vs-Rest" (also known as one-vs-all or OvR) multi-class strategy is a method for training a multi-class classifier when the target variable has more than two classes. It works by dividing the problem into multiple binary classification problems, where each binary classification problem predicts the presence or absence of a single class.

Let's go deeper into the math behind the logistic regression. Logistic regression is a linear model, described in formula (3.9), which allows predicting probabilities of object belonging to a given class. The advantages of logistic regression include: human interpretability and high speed of the algorithm, and the disadvantages are low accuracy, compared to more complex models.

$$z = \sum_{i=1}^{n} \theta_i * x_i + \theta_0 \tag{3.3.4}$$

where $\theta = (\theta_i)$, $i = 0 \ldots n$ - vector of model parameters; $x = (x_i), i = 1 \ldots n$ - vector describing the object. In a logistic regression, the assumption is made that the probability that object x belongs to class 1 will be expressed by formula (3.10)

$$P\{y = 1|x\} = f(z) = \frac{1}{1 + e^{-z}} \tag{3.3.5}$$

The training sample consists of of pairs of (x^i, y^i), $i = 1 \ldots m$.

The method of maximum likelihood is used in the selection of model parameters, according to which the parameters θ that maximize the value of the likelihood function are selected, as described in formula (3.3.6) on the training sample.

$$\theta_1 = \arg\max_{\theta} L(\theta) = \arg\max_{\theta} \prod_{i=1}^{m} P(y = y^{(i)}|x = x^{(i)}) \tag{3.3.6}$$

Maximizing the likelihood function is equivalent to maximizing its logarithm, according to formula (3.3.7).

$$\ln L(\theta) = \sum_{i=1}^{m} y^{(i)} \ln f(\theta^T x^{(i)}) + (1 - y^{(i)}) \ln(1 - f(\theta^T x^{(i)})) \tag{3.3.7}$$

The expression from formula (3.12) is usually maximized using a gradient descent [12].

A neural network is a system of neurons and the connections between them. In the process of training the weight of connections connecting different neurons gradually changes. The result of neural network training is a network whose connections have coefficients that satisfy the conditions of the problem. Neural networks are characterized by the number of trained layers with neural connections. When solving tasks with a large amount of data and a large number of parameters, neural networks with more layers are used. Learning of such networks is called deep learning.

Deep learning methods have two important properties: layer-by-layer creation of complex representations, as well as detailed exploration of intermediate representations, so that each layer is updated according to the needs of the representation of the layer above and the needs of the layer below. Together, these two properties make deep learning much more successful than previous approaches to machine learning.

The advantages of neural networks, including deep learning networks, are usually the high accuracy of text classification relative to other methods. In addition, the accuracy of this method usually greatly increases with the amount of data, i.e., in systems with the ability to accumulate data, this method will be more suitable. The disadvantage of neural networks is traditionally considered to be a higher demand for computational resources, which used to inhibit the application

of neural networks. However, with the spread of GPU computing, neural networks have been used more and more often, as their computations can be well paralleled [18].

Neural network models, including Convolutional Neural Networks (CNNs) and Recurrent Neural Networks (RNNs), are deep learning models widely used for text classification tasks. They have the ability to learn complex representations of text data and can outperform other models for certain tasks. One of the advantages of this model is in comfortable training model, where you can see your results after every "epoch" and make desicions and conclusions.

Convolutional neural networks (CNNs) are a very broad class of architectures, the basic idea of which is to reuse the same parts of a neural network to handle different small, local areas of inputs. Initially, convolutional networks were invented to solve problems related to image processing, for which they use two-dimensional convolutions. For problems related to sequence processing, one-dimensional convolutions are used instead of two-dimensional convolutions. The basic idea of convolutional networks is to train the network to identify features in the input data using convolutional layers, and then to feed the computed features to the input of the full-coherent network [11].

Recurrent neural networks are designed for solving problems related to the processing of sequences, and since the text is a sequence of symbols, these networks are often used for solving problems related to the processing of texts.

Unlike multilayer perseptrons, in recurrent neural networks the connections between neurons can go not only from the lower layer to the upper one, but also from a neuron to "itself", more precisely, to the previous values of this neuron or other neurons of the same layer. The architecture of a "simple" RNN is shown in Fig. 3.2.

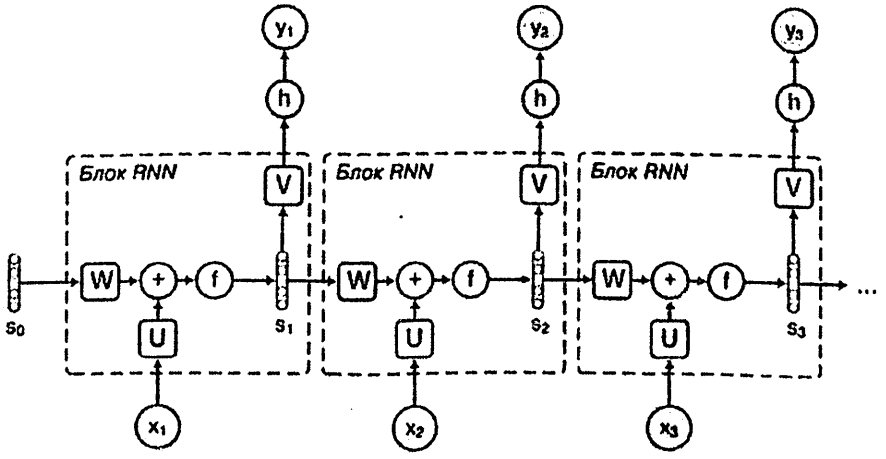

Figure 3.4: Architecture of the "simple" RNN

Description of the notations of the elements in Figure 3.2.: - W - matrix of weights for transition between latent states, - U - matrix of weights for inputs, - V

- matrix of weights for outputs, - S_i - hidden state at step i, - x_i - network input at step i, - y_i - is the output of the network at step i, - f and h are activation functions. From Figure 3.2. we can obtain a formal description of the of the network at time t (3.13).

$$a_t = b + Ws_{t-1} + Ux_t, s_t = f(a_t)\, o_t = c + Vs_t, y_t = h(o_t) \tag{3.3.8}$$

Often, instead of a "normal" RNN, two RNNs are used - two RNNs, one of which reads text words from left to right and the other from right to left. When training such a network, the latent states of both networks are used (usually the vectors describing the latent states are concatenated). The motivation for using bidirectional RNNs is to obtain a state that reflects the context from both left and right for each element of the sequence.

In our NN model we worked with training data, training labels, validation data, validation labels. Code compiles the model using the Adam optimizer and the Sparse Categorical Crossentropy loss function. The Sparse Categorical Crossentropy loss function is used for multi-class classification problems where the target labels are integers. Then converts the labels array and changes its data type to an unsigned 8-bit integer. After this, the Neural Network model fits the training data and labels for 300 epochs.

Chapter 4

Results

4.1 Evaluation

The entire database was split into two parts, with 80% designated for training the model and 20% reserved for testing and evaluating results. We utilize various evaluation metrics to evaluate the performance of classifiers, including accuracy, F1 macro average, and weighted F1 average. The accuracy metric gauges the ratio of correct predictions made by the model. The F1 macro average computes the F1 scores for each class, treating each class equally, regardless of the number of instances belonging to that class. The weighted F1 average computes the F1 scores for each class, but it gives more weight to the classes with more instances. The F1 score is a balanced metric between precision and recall in multiclass classification problems. It's defined as:

$$F1 = 2 * (precision * recall)/(precision + recall) \tag{4.1.1}$$

where precision = true positives / (true positives + false positives) and recall = true positives / (true positives + false negatives). F1 can be calculated for each class or as a macro/weighted average for all classes. This approach is useful when the class distribution is imbalanced, and the goal is to emphasize the performance on the under-represented classes. Additionally, we utilize the confusion matrix for a visual assessment of the models, enabling the calculation of further evaluation metrics such as precision, recall, and F1-score.

4.2 Results of experiment

Table 4.1. summarizes the performance of three machine learning models (Naive Bayes, Logistic Regression, and Neural Network) on the student FAQ classification task, as evaluated by three metrics: Accuracy, F1 Macro average, and Weighted F1 average.

All three models have a high accuracy of around 0.97. The Naive Bayes and

	Accuracy	F1 Macro average 3	F1 Weighted average
Naive Bayes	0.97	0.97	0.97
Logistic Regression	0.97	0.97	0.97
Neural Network	0.96	0.95	0.96

Table 4.1: Comparision of the results

Logistic Regression models have an F1 Macro average, and F1 Weighted average of 0.97, while the Neural Network model has an F1 Macro average of 0.95 and an F1 Weighted average of 0.96.

Below we provide confusion matrices for the three models. xamining these categories in the confusion matrix lets you evaluate a classification model. The matrix yields accuracy, precision, recall (sensitivity), specificity, and F1 score. The confusion matrix helps evaluate a classification model and discover misclassification trends and biases in predictions. It facilitates in model analysis and performance enhancement.

The confusion matrix (Fig. 4.1) shows the performance of a Naive Bayes model on a test data set with 10 different classes. The matrix shows the number of correct and incorrect predictions for each class. The values along the diagonal line indicate the number of correct predictions for each class. For example, the model correctly predicted 8 instances of class 1, 7 instances of class 2, and so on. The values outside the diagonal represent the number of incorrect predictions. For example, the model predicted 2 instances of class 9 as another class. The matrix indicates that the model performed very well, with few incorrect predictions.

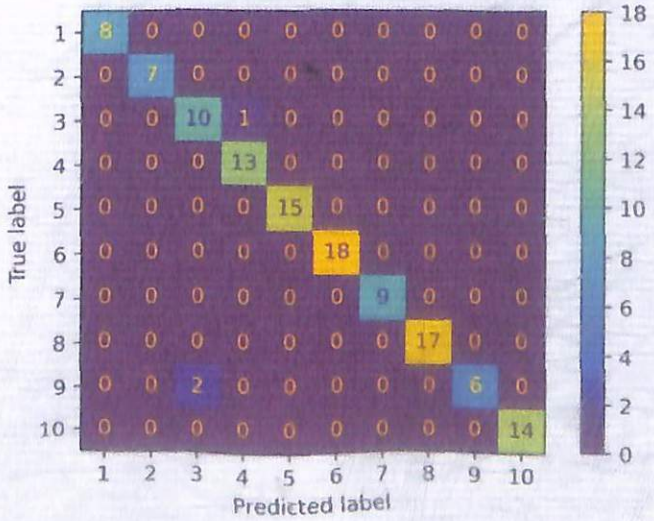

Figure 4.1: Confusion matrix of Naive Bayes

Here is the code for Naive Bayes model:

```
nb_tfidf = MultinomialNB()
nb_tfidf.fit(X_train_vectors_tfidf, y_train)
```

```
y_predict = nb_tfidf.predict(X_test_vectors_tfidf)
y_prob = nb_tfidf.predict_proba(X_test_vectors_tfidf)[:,1]
print(classification_report(y_test,y_predict))
actual = numpy.random.binomial(1,.9,size = 1000)
predicted = numpy.random.binomial(1,.9,size = 1000)
confusion_matrix = metrics.confusion_matrix(y_test,y_predict)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusi
cm_display.plot()
plt.savefig("conf_naive.png",bbox_inches='tight', dpi = 300)
plt.show()
```

The confusion matrix below (Fig. 4.2) shows the performance of a Multinomial Logistic model on a test data set with 10 different classes. We see that the model correctly predicted 8 instances of class 1, 7 instances of class 2, and so on. The values outside the diagonal represent the number of incorrect predictions. For example, the model predicted 2 instances of class 9 as another class. The matrix indicates that the model performed very well, with few incorrect predictions.

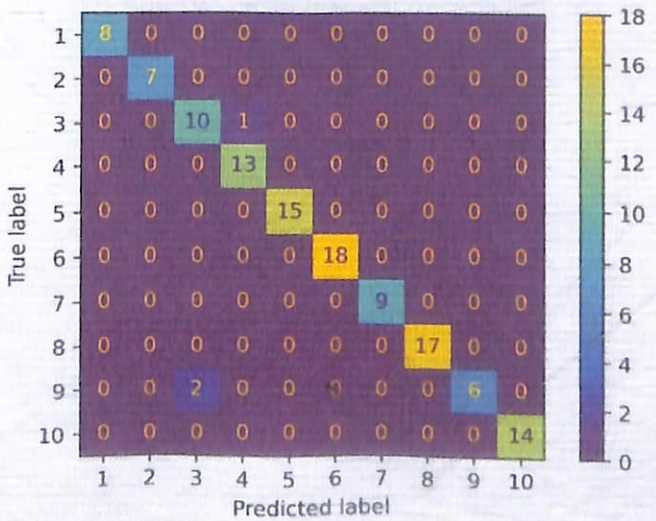

Figure 4.2: Confusion matrix of Multinomial Logistic

Here is the code for Logistic Regression:

```
lr_tfidf=LogisticRegression(multi_class='ovr')
lr_tfidf.fit(X_train_vectors_tfidf, y_train)
y_predict = lr_tfidf.predict(X_test_vectors_tfidf)
y_prob = lr_tfidf.predict_proba(X_test_vectors_tfidf)[:,1]
print(classification_report(y_test,y_predict))
print('Confusion Matrix:',confusion_matrix(y_test, y_predict))
confusion_matrix = metrics.confusion_matrix(y_test,y_predict)
cm_display = metrics.ConfusionMatrixDisplay(confusion_matrix = confusi
cm_display.plot()
plt.savefig("conf_logistic.png",bbox_inches='tight', dpi = 300)
```

```
plt.show()
```

Finally, in the confusion matrix report for the neural network model, we see that it correctly predicted 7 instances of class 1, 7 instances of class 2, and so on. The values outside the diagonal represent the number of incorrect predictions. Also, the model predicted 1 instance of class 3 as another class and 1 instance of class 8 as another class. The matrix (Fig. 4.3) indicates that the model performed well, with relatively few incorrect predictions.

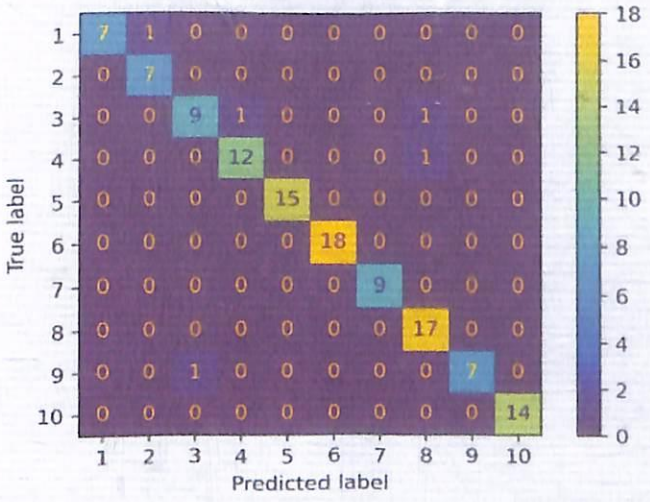

Figure 4.3: Confusion matrix of Neural Network

Here is the code for NN model:

```
train_labels = np.array(y_train)
train_labels = train_labels.astype(np.uint8)
model.fit(X_train_vectors_w2v, train_labels, epochs=300)
cm = confusion_matrix(predicted, test_labels)
print("Confusion Matrix:")
print(cm)
```

4.3 Creation of the chatbot.

To create the chatbot we used "telegram", "dialogflow" and integrated our database to it.

A natural language understanding (NLU) platform called Dialogflow was developed by Google, which allows developers to create conversational interfaces including virtual assistants or chatbots. In a natural way that is contextually relevant applications can comprehend user input thanks to the power of Dialogflow.

Advantages of Dialogflow:

- Recognizing user intents is made easier for Dialogflow through the use of machine learning algorithms analyzing given input. Booking a flight or getting product information are examples of intents that describe what action the user wants to take;
- Recognition of critical data in user input with the aid of entities is accomplished by Dialogflow. Different kinds of information like names or dates can be labelled as distinct entity types. The extraction of data enables us to deliver personalized responses and execute specific task;
- Developers employing Dialogflow could use context definition and management to improve the ability of the system to interpret user input within a conversational framework. This ability facilitates natural and meaningful conversations in the chatbot by maintaining context between user interactions;
- Dialogflow provides support for various languages helping you create chatbots capable of understanding and responding correspondingly;
- By using Dialogflow's integration capabilities you can choose from several ways to deploy a chatbot across sites or messaging applications like Facebook Messenger or Slack along with the option of integrating it with various voice assistants including the popular Google Assistant;
- With Dialogflow's support for generating rich response types such as cards, suggestions etc, creating attractive conversations has never been easier. With this functionality you can create conversational experiences that engage and interest your users;
- Measuring satisfaction of users with your chatbot is easier with Dialogflow's analytics that provides insight into interaction patterns. These metrics allow you to continuously improve and fine-tune the performance of your chatbot as time passes;
- Through the utilization of Google's machine learning technology Dialogflow can keep enhancing its natural language processing abilities, and by leveraging user feedback this AI system can continually improve its ability to recognize intentions and produce more accurate responses as time goes on.

We started creating the bot by getting a token for the bot via "BotFather" in Telegram. This token will allow us to configure the bot via DialogFlow. Immediately we gave the bot a name: *SDUFAQ_bot*.

Telegram's BotFather lets you develop and manage Telegram bots. It creates and configures bots.

BotFather creates Telegram bots. Chat with BotFather and follow the steps to set up your bot's name, username, and other parameters. After creating the bot, BotFather gives you an API token to control it via the Telegram Bot API.

BotFather allows bot configuration after creation. Set a bot's profile image, description, and other data. Inline mode, group privacy mode, and more may be

enabled or disabled in BotFather.

BotFather lets you create bespoke bot commands. These instructions are activated when visitors interact with your bot using certain keywords followed by "/". You may specify commands like "/start" or "/help" to deliver specialized functionality or information to users.

Bot Updates: BotFather displays and manages bot updates. You may query the Telegram server or utilize webhooks to get updates in real time. BotFather administers your bot. BotFather lets you manage bots, check statistics, and modify settings.

BotFather simplifies Telegram bot creation and administration. Create, configure, and manage bots with its simple UI. BotFather lets you construct sophisticated Telegram bots that engage people and offer numerous services.

The rest of the work was done through dialogflow. We added intents to the code of our bot, and for each intent we added a response. We also added possible events and context. And we repeated these actions 10 times, for each target.

Dialogflow's NLP chatbot creation process includes planning, setting up, defining intents, increasing Dialogflow's comprehension, testing and iterating, deploying, and monitoring and improving performance.

Start by identifying the chatbot's goal, audience, and features. Create and configure a Dialogflow agent via the console.

Next, create intentions for chatbot actions and requests. Make intents with sample user questions and replies. Train the agent by assigning training words to intents. For multi-turn discussions, employ entities to extract individual user inputs and set context and follow-up intents. Enable fulfillment choices to conduct tasks or obtain data from remote systems.

Test the chatbot with sample inputs, analyze the answers, and adjust the training words and intents. Based on user input, refine the design and agent comprehension and replies.

Integrate Dialogflow APIs or SDKs to deploy the chatbot on the desired platform (e.g., website, messaging platform). Webhook fulfillment for bespoke logic or other system connections.

Track chatbot performance and user comments. Examine user inquiries, intent matching, and satisfaction. Update the agent to improve performance and user experience.

Using real-world user interactions, the chatbot's training data, intent structure, and replies must be refined. Use user feedback to improve the chatbot's design.

Let's talk about integration with the Telegram messenger. To integrate with Telegram, you need to create a script that will redirect messages from the public account to Dialogflow. This service has a Web version of the chat, which allows you to integrate into the website of, for example, a university. All the messages

that will be sent through the web version are not encrypted, so you can redirect them to your Telegram account, knowing the access token of the public account and the backend token.

The principle of Telegram chatbots is as follows: the user sends a message to the public account, and it is sent to the server with a running script. This script sends the message to the web version of dialogflow. At this time, the service generates and sends a response back to the server, which in turn sends it to the user [22].

Chapter 5

Discussion

Our chatbot provides students with a convenient and efficient way to obtain answers to their questions, surpassing the need to approach an advisor, supervisor, or search through the university's website. Given the increasing number of Telegram users among students, integrating the chatbot into this messenger platform is a suitable solution in our opinion. Moreover, this chatbot has versatile applications beyond the university context and can be employed to enhance the quality of customer interactions across various services.

The chatbot demonstrates an understanding of context, allowing it to adapt to specific user requests and is not limited by rigid rules. It possesses the capability to identify the appropriate question category and provide an answer, even if the question is not present in the database. However, a drawback of the chatbot is its inability to identify the category and answer a question if the user makes a grammatical error.

Acquiring a database of questions specific to a particular university can be challenging, as most universities do not possess a structured repository of student inquiries. However, we managed to compile the necessary number of questions from the university's website and conducted thorough rephrasing of the text. As a result, we successfully gathered a database consisting of over 500 different sentences. It is possible to expand this database in the future to further enhance the bot's accuracy in providing answers.

Dialogflow offers several compelling advantages that make it a desirable choice for creating chatbots. One key advantage is its extensive range of integrations with various platforms, including Facebook Messenger, Slack, Telegram, WhatsApp, and others. This versatility enables the deployment of chatbots across multiple communication channels, expanding the reach and accessibility to a wider audience.

Dialogflow leverages Google's powerful natural language understanding (NLU) technology, enabling chatbots to comprehend user input in a more natural and nuanced manner. It excels in handling synonyms, word relationships, and contextual cues, resulting in more accurate intent recognition and the generation of better

responses.

Another advantage is that Dialogflow operates on Google Cloud infrastructure, which ensures scalability to meet increasing demand. It can efficiently handle a large volume of requests concurrently, ensuring optimal performance even during peak usage periods.

However, Dialogflow does have limitations. Its performance heavily depends on the quality and quantity of training data available. Insufficient or limited training data can hinder the chatbot's understanding of user inputs and lead to less accurate responses. It is crucial to collect diverse and representative training data and properly annotate it for optimal performance.

Additionally, Dialogflow may encounter challenges when handling complex dialogues involving multiple turns and intricate contextual dependencies. Maintaining a coherent conversation flow and accurately interpreting user intentions can be more challenging in such scenarios. It is important to design the chatbot's dialogue management system carefully to handle complex interactions effectively.

Overall, Dialogflow's extensive integrations, powerful NLU capabilities, and scalability make it a compelling choice for building chatbots, but it is essential to consider its limitations and carefully address them during development.

Chapter 6

Conclusion

In our research, we not only focused on the process of creating a chatbot but also developed a tangible product with the aim of assisting students in their interactions with the university. This aspect sets our work apart from other papers in the field, which primarily concentrate on creating bots to simplify the tasks of teachers or university libraries.

Furthermore, our paper encompasses the mathematical aspects of text classification, providing a comprehensive understanding of the underlying principles and techniques involved in building effective chatbots. By exploring and evaluating the top three models of text classification, we have ensured that our chatbot is based on robust and reliable methods.

By emphasizing student-university relationship, we've addressed students' particular needs. Our chatbot improves student experience by making information and support easier and faster.

Our work shows how to make a chatbot and delivers a product to improve student-university connection. We fill this vacuum in the literature to enhance chatbot technology for students.

This research will teach readers about chatbot database acquisition and text classification model comparison. These practical tips help readers implement comparable initiatives.

To showcase the outcome of our work, we have made our chatbot available on the Telegram platform. By searching for "$SDUFAQ_bot$" on Telegram, users can access our chatbot and interact with it directly. The chatbot covers 10 categories, including academic withdrawals, FX grade (25-49), drop, credit system and retake, discounts, scholarships, GPA calculating, and curriculum. Our chatbot can answer student life and university-related questions.

By providing access to our chatbot, readers can experience firsthand the capabilities and benefits of our developed system. It serves as a practical demonstration of the potential of chatbots in assisting and supporting students, enhancing their interaction with the university, and providing prompt and accurate information.

Overall, our work not only provides valuable insights and comparisons in the field of chatbot development but also presents a tangible implementation in the form of our accessible chatbot on the Telegram platform.

Future bot improvements are great and can improve functionality and user experience. Improvement suggestions:

Expanding the Database: More questions and responses will help the bot offer accurate and complete answers. Collecting additional student questions and updating the database to cover more areas will accomplish this. User input and student queries can help the bot learn. Upgrading the Bot's Interface: Personality and empathy can make user interactions more engaging and fulfilling. Natural language creation can make the bot's responses more conversational and human-like. Sentiment analysis can also help the bot comprehend and respond to users' emotions, improving the user experience.

Integrating with Other Messengers: Adding the bot to WhatsApp, Viber, or even web chat interfaces can increase its user base. Leveraging these platforms' APIs and tailoring the bot's features to each messenger can achieve this integration.

Grammatical Errors: Teaching the bot to handle grammatical errors might help it understand and interpret user requests. Spell-checking, grammar correction, and context-aware error handling can help the bot avoid grammatical problems and respond appropriately. This reduces annoyance and improves the bot's understanding of user inputs.

By implementing these improvements, the bot can become more sophisticated, user-friendly, and capable of delivering a higher quality of service to students, further enhancing their interaction with the university and improving overall user satisfaction.

Bibliography

[1] DeeAnn Allison. Chatbots in the library: is it time? *Library hi tech*, 2012.

[2] TV Batura. Metody avtomaticheskoy klassifikatsii tekstov. *Programmnye produkty i sistemy*, 30(1):85–89, 2017.

[3] Namrata Bhartiya, Namrata Jangid, Sheetal Jannu, Purvika Shukla, and Radhika Chapaneri. Artificial neural network based university chatbot system. pages 1–6, 2019.

[4] Stefania Boiano, Ann Borda, Guiliano Gaia, Stefania Rossi, and Pietro Cuomo. Chatbots and new audience opportunities for museums and heritage organisations. *Electronic visualisation and the arts*, pages 164–171, 2018.

[5] Erik Cambria and Bebo White. Jumping nlp curves: A review of natural language processing research. *IEEE Computational intelligence magazine*, 9 (2):48–57, 2014.

[6] S Chaya et al. Artificial intelligence chat bot. 2019.

[7] Aishwarya Gupta, Divya Hathwar, and A Vijayakumar. Introduction to ai chatbots. *International Journal of Engineering Research and Technology*, 9 (7):255–258, 2020.

[8] Jeff Heaton. Introduction to the math of neural networks (beta-1). *Heaton Research Inc*, 2011.

[9] Bob Heller, Mike Proctor, Dean Mah, Lisa Jewell, and Bill Cheung. Freudbot: An investigation of chatbot technology in distance education. In *EdMedia+ innovate learning*, pages 3913–3918. Association for the Advancement of Computing in Education (AACE), 2005.

[10] OE Kadeeva and VN Syritsina. Chat-bots i osobennosti ikh ispol'zovaniya v obrazovanii. *Informatika v shkole*, (10):45–53, 2020.

[11] A Kadurin, S Nikolenko, and E Arkhangelskaya. Glubokoe obuchenie. pogruzhenie v mir neyronnykh setey. *SPb.: Piter*, 480, 2018.

[12] Tatiana Viktorovna Logunova, Lidia Viktorovna Shcherbakova, Vasiliy Mikhaylovich Vasyukov, and Vyacheslav Vladislavovich Shimkun. Analiz algoritmov klassifikatsii tekstov. *Universum: tekhnicheskie nauki*, (2-2 (107)):4–20, 2023.

[13] Abbas Saliimi Lokman and Mohamed Ariff Ameedeen. Modern chatbot systems: A technical review. In *Proceedings of the Future Technologies Conference (FTC) 2018: Volume 2*, pages 1012–1023. Springer, 2019.

[14] Montserrat Mateos-Sanchez, Amparo Casado Melo, Laura Sánchez Blanco, and Ana M Fermoso García. Chatbot, as educational and inclusive tool for people with intellectual disabilities. *Sustainability*, 14(3):1520, 2022.

[15] Michael L Mauldin. Chatterbots, tinymuds, and the turing test: Entering the loebner prize competition. In *AAAI*, volume 94, pages 16–21, 1994.

[16] ELisa Miller-Out. How chatbots will help education. *Available at https://venturebeat.com/business/how-chatbots-will-help-education/ (2016/09/29).*

[17] Natalya Vladimirovna Petrova, Igor Pavlovich Petrov, and Yuriy Igorevich Petrov. Mobil'nye messendzhery kak odna iz sostavlyayushchikh sovremennogo obrazovaniya. *Informatizatsiya obrazovaniya i nauki*, (4):59–69, 2016.

[18] Benjamin Recht, Rebecca Roelofs, Ludwig Schmidt, and Vaishaal Shankar. Do cifar-10 classifiers generalize to cifar-10? *arXiv preprint arXiv:1806.00451*, 2018.

[19] Heung-Yeung Shum, Xiao-dong He, and Di Li. From eliza to xiaoice: challenges and opportunities with social chatbots. *Frontiers of Information Technology & Electronic Engineering*, 19:10–26, 2018.

[20] AS Sreelakshmi, SB Abhinaya, Aishwarya Nair, and S Jaya Nirmala. A question answering and quiz generation chatbot for education. In *2019 Grace Hopper Celebration India (GHCI)*, pages 1–6. IEEE, 2019.

[21] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[22] Nadezhda Timofeevna Sukhanova and Tatiana Mechislovasovna Vezhelis. Ispol'zovanie chat-botov dlya avtomatizatsii predostavleniya spravochnoy informatsii abiturientam i studentam vuzov. *Problemy sovremennogo pedagogicheskogo obrazovaniya*, (76-2):178–181, 2022.

[23] Nigar M Shafiq Surameery and Mohammed Y Shakor. Use chat gpt to solve programming bugs. *International Journal of Information Technology & Computer Engineering (IJITC) ISSN: 2455-5290*, 3(01):17–22, 2023.

[24] Sandeep A Thorat and Vishakha Jadhav. A review on implementation issues of rule-based chatbot systems. In *Proceedings of the international conference on innovative computing & communications (ICICC)*, 2020.

[25] Ivan Tsidylo, Sergiy Samborskiy, Stanislav-Ivan Mazur, and Maria Zamoroz. Designing a chat bot for learning a subject in a telegram messenger. In *ICTERI Workshops*, pages 1329–1340, 2020.

[26] Alan M Turing. *Computing machinery and intelligence*. Springer, 2009.