

Czech University of Life Sciences Prague

Faculty of Economics and Management

Department of Information Engineering



Bachelor Thesis

**Development of chat bot with embedded machine
learning models**

Illia Prazdnyk

© 2020 CULS Prague

BACHELOR THESIS ASSIGNMENT

Illia Prazdnyk

Systems Engineering and Informatics
Informatics

Thesis title

Development of chat bot with embedded machine learning models

Objectives of thesis

The main goal of the thesis is to explain the process of development and deployment of a telegram chatbot using python, the bot would use various machine learning and deep learning methods. It is going to be able to execute commands and enter different modes, each designed to show a specific application of both natural language processing and computer vision. One of those is to be able to have a conversation. The final goal is to deploy it to the server to make it fully accessible to anyone using Telegram application.

Methodology

The methodology of the thesis is based on analysis and study of the relevant technical and scientific sources in the field of machine learning models based on statistical and deep learning approaches. Specific attention will be paid towards techniques of natural language processing, computer vision, transfer learning, and fine-tuning of machine learning models. Based on synthesis of gained knowledge, the chat bot will be implemented. The bot is going to work using Telegram as a platform. The code is going to be written in Python utilizing many of its libraries for data science, natural language processing, and chatbots, including NumPy, Pandas, SciKit learn, TensorFlow, Keras, TelegramBot API, OpenCV and many others. The bot is going to be functional and deployed to the server of future choice, which means that it's going to be fully accessible through Telegram application.

The proposed extent of the thesis

40-50 pages

Keywords

Chatbot, Deep learning, Supervised learning, Unsupervised learning, Machine learning

Recommended information sources

BENGFORT, Benjamin, Rebecca BILBRO a Tony OJEDA. Applied text analysis with Python: enabling language-aware data products with machine learning. 3rd ed. Sebastopol, CA: O'Reilly Media, 2018. ISBN 978-1491963043.

GÉRON, Aurélien, Peter NORVIG a Ernest DAVIS. Hands-on machine learning with Scikit-Learn and TensorFlow: concepts, tools, and techniques to build intelligent systems. 3rd ed. Boston: O'Reilly Media, 2017. ISBN 978-1491962299.

RUSSELL, Stuart J., Peter NORVIG a Ernest DAVIS. Artificial intelligence: a modern approach. 3rd ed. Upper Saddle River: Prentice Hall, c2010. ISBN 01-360-4259-7.

Expected date of thesis defence

2019/20 SS – FEM

The Bachelor Thesis Supervisor

Ing. Petr Hanzlík, Ph.D.

Supervising department

Department of Information Engineering

Electronic approval: 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Head of department

Electronic approval: 19. 2. 2020

Ing. Martin Pelikán, Ph.D.

Dean

Prague on 21. 02. 2020

Declaration

I declare that I have worked on my bachelor thesis titled " Development of chat bot with embedded machine learning models" by myself and I have used only the sources mentioned at the end of the thesis. As the author of the bachelor thesis, I declare that the thesis does not break copyrights of any their person.

In Prague on 23.03.2020

Acknowledgement

I would like to thank my thesis supervisor Ing. Petr Hanzlík, Ph.D. for his advice, support and time which helped me a lot with writing this thesis.

Development of chat bot with embedded machine learning models

Abstract

The following thesis is focused on development of informational services in a form of a chatbot. It is supposed to highlight the benefits and use cases of developing a chatbot as an assistance service or a standalone product, and then explain the modern techniques for implementation of its main components. The thesis covers the main stages in the process of designing, developing and deploying a chatbot system, including the selection of the platform, selection of development tools and configuration of the development environment, development of the basic chatbot functionality, configuration of conversational agents, development and deployment of deep learning models, data preprocessing, selection of the serving service, configuration of the serving machine and deployment of the chatbot. The result will be a working and served chatbot powered by machine learning models which is fully accessible through a messenger application.

Keywords: Chatbot, Deep learning, Supervised learning, Unsupervised learning, Machine learning

Vývoj chat botu s vestavěnými modely strojového učení

Abstrakt

Tato práce se zaměřuje na vývoj informačních služeb ve formě chatbota. Jejím cílem je zdůraznit výhody a případy použití chatbota jako asistenční služby nebo samostatného produktu, a následně vysvětlit moderní techniky pro implementaci jeho hlavních částí. Tato práce pokrývá hlavní fáze návrhu, vývoje a nasazení systému chatbota, včetně výběru platformy, výběru vývojových nástrojů a konfigurace vývojového prostředí, vývoje základní funkčnosti chatbota, konfigurace konverzačních agentů, vývoje a nasazení modelů hlubokého učení, předzpracování dat, výběru obsluhující služby, konfigurace stroje a nasazení chatbota. Výsledkem bude funkční a obsluhovaný chatbot využívající modely strojového učení, který je plně přístupný prostřednictvím chatové aplikace.

Klíčová slova: Chatbot, Deep learning, Supervised learning, Unsupervised learning, Machine learning

Table of content

1 Introduction.....	11
2 Objectives and Methodology.....	12
2.1 Objectives.....	12
2.2 Methodology.....	12
3 Literature Review.....	13
3.1 Recent development of artificial intelligence.....	13
3.2 Introducing Chatbots.....	14
3.2.1 What is a Chatbot?.....	14
3.2.2 Why do we need chatbots?.....	14
3.2.3 The short history of chatbots.....	15
3.3 Chatbot’s impact on the market.....	16
3.4 Capabilities of a chatbot.....	17
3.4.1 Is it possible to solve the problem via dialogue?.....	17
3.4.2 Is it possible to automate the solution?.....	18
3.4.3 How big is the informational scope for the bot?.....	19
3.5 Designing a chatbot.....	19
3.5.1 Definition of intents.....	20
3.5.2 Analysing the variance of the intent queries.....	20
3.5.3 Definition of entities.....	21
3.5.4 Design of the dialog’s algorithm.....	21
3.6 Natural Language Processing.....	22
3.6.1 Segmentation.....	23
3.6.2 Part-of-Speech Tagging.....	23
3.6.3 Lemmatization.....	24
3.6.4 Extraction of entities.....	24
3.6.5 Removing the stop words.....	24
3.6.6 Analysing dependencies.....	25
3.6.7 Combinations of nouns.....	26
3.6.8 Working with similar words.....	27
3.6.9 Tokenization.....	27
3.7 Toolkit of chatbot developer.....	28
3.7.1 Python programming language.....	29
3.7.2 What is a python environment?.....	30
3.7.3 The pip package manager.....	30
3.7.4 Python packages.....	31
3.7.5 SciPy.....	31

3.7.6	NumPy.....	32
3.7.7	Pandas.....	33
3.7.8	Matplotlib	34
3.7.9	IPython	35
3.7.10	SpaCy	37
3.7.11	Dialogflow	38
3.7.12	SciKit-Learn	39
3.7.13	TensorFlow	39
3.7.14	Keras.....	40
3.7.15	Google Collaboratory	41
3.7.16	Bot API's.....	41
3.8	Hosting the chatbot	42
3.9	Complete roadmap of making a working chatbot.....	43
4	Practical Part	44
4.1	Designing the chatbot.....	44
4.1.1	Definition of intents	44
4.1.2	Definition of entities	44
4.1.3	Draft of the bot's algorithm.....	45
4.2	Select the platform for a bot	46
4.3	Configuring the workspace.....	46
4.4	Implementing the basic bot functionality	47
4.5	Development of conversational agent	49
4.6	Digit recognition	53
4.6.1	Training and using a deep neural network	53
4.6.2	Working with real images	56
4.6.3	Connecting the network to the bot.....	56
4.7	Deploying the chatbot	58
5	Results	61
6	Conclusion.....	62
7	References	63
8	Appendix	63

List of pictures

Figure 1	Deep Learning architecture (Ajay, 2018).....	13
Figure 2	Top industries that will benefit most from chatbots (Sumit, 2019)	16
Figure 3	Graphical representation of dependency parsing using corenlp.run	26
Figure 4	Definition of complex function in Python (Aurélien, 2017)	29
Figure 5	Creating virtualenv (Aurélien, 2017)	30
Figure 6	The resulting plot using Matplotlib	35

Figure 7 Python interpreter access through command prompt.....	36
Figure 8 Access to IPython console through the command prompt	36
Figure 9 spaCy benchmarking results (Sumit, 2019).....	37
Figure 10 Working diagram of Dialogflow architecture (Sumit, 2019)	38
Figure 11 example of a code cell and its output	41
Figure 12 Checking the python version	46
Figure 13 Checking the pip version	47
Figure 14 Importing basic bot functions	48
Figure 15 Configuration of the updater and the logger	48
Figure 16 Definition of the "start" command	48
Figure 17 Creation of command handler.....	49
Figure 18 The first command sent to the bot.....	49
Figure 19 Agent creation window on Dialogflow	50
Figure 20 The Intents page of Small-Talk agent	51
Figure 21 Configuring dialogflow session	51
Figure 22 Text handling function	52
Figure 23 Making and adding the message handler.....	52
Figure 24 Working dialogue agent	52
Figure 25 Checking the shape of the sets	53
Figure 26 Reshaping the data	54
Figure 27 Converting class vectors.....	54
Figure 28 Importing the model's components	54
Figure 29 The code for building, compiling and training 2d conv network	55
Figure 30 Model's evaluation	55
Figure 31 Saving the model.....	55
Figure 32 Rebuilding the model from files	56
Figure 33 Image preprocessing function.....	56
Figure 34 Chatbot's part for predicting the digits	57
Figure 35 Bot detecting the digit	57
Figure 36 Instructions for connecting to the instance	59
Figure 37 FileZilla interface	59
Figure 38 Running the bot on server.....	60
Figure 39 Testing if the bot works.....	60

List of tables

Table 1 – a brief history of chatbots (Sumit, 2019).....	15
--	----

List of abbreviations

2D - Two Dimensional
API - Application
Processing Interface
AWS - Amazon Web
Services
DL - Deep Learning
IDE - Integrated
Development Environment
ML - machine learning
POS - Part of Speech

1 Introduction

Due to the exponential increase in general computational power in our machines over the last decade, humanity has significant improvements in the field of artificial intelligence which allowed us to create more advanced intelligent systems, for example – chatbots.

A chatbot is a program that simulates a conversation with a user and performs some operations with the inputs from those conversations. There are various ways of accessing a chatbot: through a widget on a website, through virtual assistants such as Amazon Alexa, through messaging applications like Telegram or Facebook Messenger, or as standalone applications.

Due to the various ways of implementation, chatbots get a wide range of use cases. They are used as: personal virtual assistants, small applications accessed through other apps, customer service assistants on websites, standalone mobile productivity applications and role-playing games.

There are many technologies that allow developers to create a conversational part of a chatbot. They can be built from scratch using natural language processing and deep learning techniques or built using different special frameworks which also utilize NLP and ML techniques, but also have their own front-end consoles and deployment tools.

Besides a conversational system, a chatbot requires a hosting platform, such as a website, messaging app or its own standalone program. The most popular choice among these is a bot's own account in a messaging application. This choice is made due to messaging applications already having a very big audience, so the potential user doesn't have to download any applications or open webpages, and of course the interface of a messaging application is exactly what a chatbot needs – a dialog.

The last component of many chatbot systems are the methods and functions used to perform the operations besides having a conversation. If chatbots need to fulfil complicated requests from a user that require interaction with systems outside the conversation, such as booking a table in a restaurant or buying something in e-shop, it requires to be able to execute some methods or interact with outside API's.

Implementation of all mentioned components is going to be covered in this thesis and hopefully will seem clear and simple.

2 Objectives and Methodology

2.1 Objectives

The main goal of the thesis is to explain the process of development and deployment of a telegram chatbot using python, the bot would use various machine learning and deep learning methods. It is going to be able to execute commands and enter different modes, each designed to show a specific application of both natural language processing and computer vision. One of those is to be able to have a conversation. The final goal is to deploy it to the server to make it fully accessible to anyone using Telegram application.

2.2 Methodology

The methodology of the thesis is based on analysis and study of the relevant technical and scientific sources in the field of machine learning models based on statistical and deep learning approaches. Specific attention will be paid towards techniques of natural language processing, computer vision, transfer learning, and fine-tuning of machine learning models. Based on synthesis of gained knowledge, the chat bot will be implemented. The bot is going to work using Telegram as a platform. The code is going to be written in Python utilizing many of its libraries for data science, natural language processing, and chatbots, including NumPy, Pandas, SciKit learn, TensorFlow, Keras, TelegramBot API, OpenCV and many others. The bot is going to be functional and deployed to the server of future choice, which means that it's going to be fully accessible through Telegram application.

3 Literature Review

3.1 Recent development of artificial intelligence

The concepts of artificial intelligence and machine learning were introduced in the 20th century but the large increase in popularity of artificial intelligence can be tracked back to 2006, when Geoffrey Hinton published a paper “A fast learning algorithm for deep belief nets.” The paper described a process of training a neural network capable of recognizing handwritten digits. This technique became called “Deep Learning”.

Deep Learning is a machine learning technique based on usage of complex artificial neural networks, for example: recurrent and convolutional neural networks (RNN & CNN). Nowadays developers use Deep Learning to train models that can make different kinds of predictions and perform complex analysis. They are used in pattern recognition, data classification, image and speech recognition, data generation and many other operations.

“Fast-forward 10 years and Machine Learning has conquered the industry: it is now at the heart of much of the magic in today’s high-tech products, ranking your web search results, powering your smartphone’s speech recognition, and recommending videos, beating the world champion at the game of Go. Before you know it, it will be driving your car” (Aurélien, 2017)

Deep neural networks allowed researchers to get better results in solving problems of natural language processing due to their ability to: train on bigger datasets, be used in encoding and decoding algorithms, perform feature extraction and generate new data, which allows developers to build more complex language models and natural language based systems. The systems that allow users to translate some text, classify intents, get an answer to specific question, get better search results, generate other kinds of data from text or even simulate a real dialogue, such systems are called chatbots.

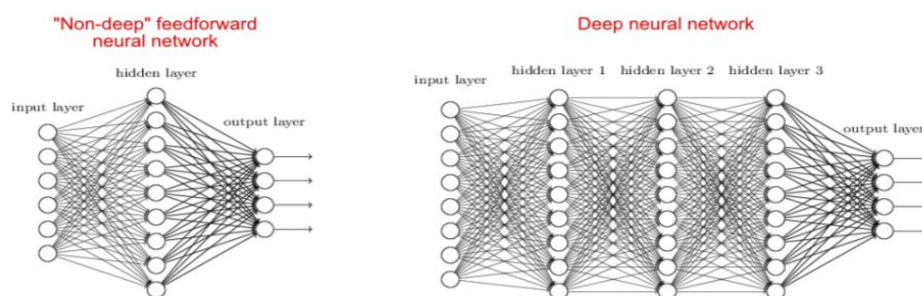


Figure 1 Deep Learning architecture (Ajay, 2018)

3.2 Introducing Chatbots

3.2.1 What is a Chatbot?

If you live in the 21st century, you should already have an idea of what a chatbot is – it is a system that is controlled by textual inputs which are sent by the user into the virtual dialogue with the chatbot agent. Then this system tries to interpret what is the user's request and produces the outputs according to this request, then the answers are sent as answers to the same dialogue from the chatbot to the user. In other words, the user asks the bot to do something and the bot responds to the user according to what he was asked. Such workflow allows chatbot applications to adapt to different use cases and get implemented in various forms, from customer assistance widget in the e-shop to a virtual assistant inside the operation system.

The bots are able to save some information from the users to use it in future, so they wouldn't have to ask for it every time, or even try to answer user's questions about some story.

The core technologies that can be found inside the most of the chatbots are intent prediction, entity recognition, answer query selection or generation. Those technologies allow bots to understand the needs of the user from what it asks for, save some specific variables from the users input queries, and choose or create the most suitable response to the user's message.

3.2.2 Why do we need chatbots?

The reason why chatbots are in such a high demand for businesses is due to many positive factors all of the chatbots share:

- They are always online
- One bot can be used by many customers at a time
- They need less money than employee's salary
- If they are trained correctly, they can work just as good or even better than humans
- People prefer texting a bot instead of calling for voice line or traveling

3.2.3 The short history of chatbots

Even though the main purpose of a chatbot is to simulate a conversation with a real person, they are still not sufficient enough to pass the Turing's test, which was invented by the famous Alan Turing in 1950. This test checks the machines ability to disguise as a human without being compromised. In the test one person gets the role of the judge, the judge is going to be typing questions into two dialogues, one dialogue with another person and second dialogue with the bot, but the judge doesn't know which dialogue is which. Both the user and the bot are going to try to answer those questions and the judge's job is to guess which answers were given by the virtual player.

The bots haven't yet passed the test, because the variety of questions a human judge may ask is multiplied by the many ways by which we humans can ask the same question. While another human can recognize all of these questions due to the incredible power and experience of our natural language system, the bots are able to work correctly only with the scenarios they were trained for.

We still can't train a language model as wide and as "smart" as the one we humans have, but over last years the chatbots have come a long way. Here is a list of the most significant dates in the chatbots history:

Table 1 – a brief history of chatbots (Sumit, 2019)

1966	Eliza – the first chatbot was created, it was supposed to work as a therapist
1981	The Jabberwocky chatbot was invented to simulate a funny human conversation
1995	A.L.I.C.E was invented by Richard Wallace. He was awarded the Nobel Prize
2001	Smarterchild – the first personal assistant was developed
2006	The IBM's Watson invented with the goal to answer questions in jeopardy
2010	Siri was launched as the first mobile speech recognizing personal assistant
2012	Google Now chatbot was launched
2014	Amazon Alexa virtual assistant was released
2016	Facebook created an API for simple development of bots for the messenger app
2018	300,00 bots were created only on Facebook Messenger platform

The 60 years of work of the scientists, programmers and users brought chatbots to where they are now, our job is to bring them further using the experience and technologies, given

to us by the brightest minds of the past and extend the borders of understanding the logic behind our own speech and mind.

3.3 Chatbot’s impact on the market

A chatbot can be implemented in many different systems such as websites, messenger applications and standalone software. Also, there are a lot of special frameworks and APIs which allow developers to create more “intelligent” chatbots and save time while doing it. This variety of tools and platforms allows chatbots to be implemented for various business purposes, for example digital marketing, educational software, virtual assistance and customer service.

“A research study by Mindbowser in association with Chatbots Journal collected data from 300+ individuals who participated from a wide array of industries including online retail, aviation, logistics, supply chain, e-commerce, hospitality, education, technology, manufacturing, and marketing & advertising.” (Sumit, 2019)

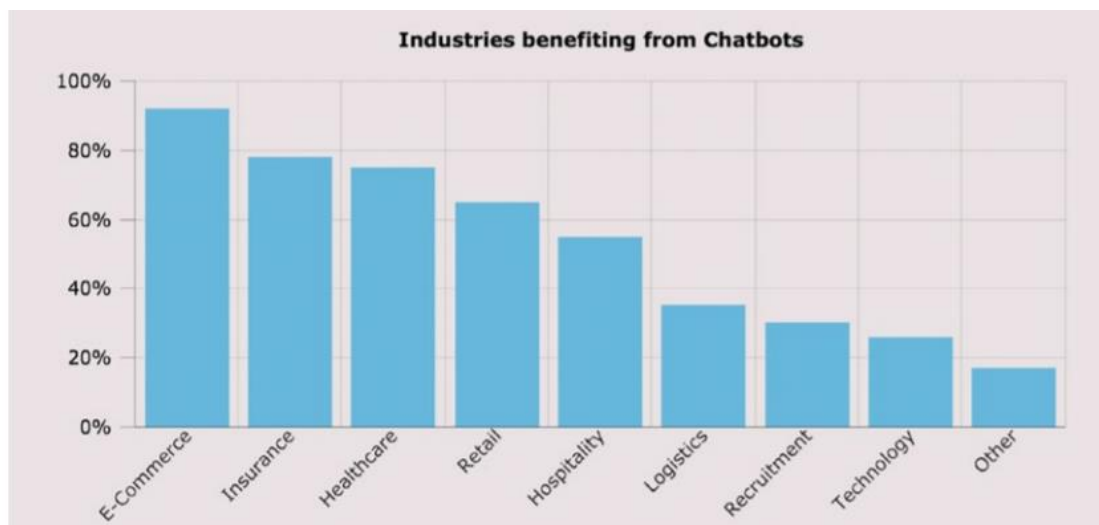


Figure 2 Top industries that will benefit most from chatbots (Sumit, 2019)

As it is shown on the Figure 2 the top three industries that will benefit from using a chatbot as a virtual agent are e-commerce, insurance and healthcare. This tendency is not strange, it is caused by the specifics of the client’s needs from those industries, such as immediate response and predictive scenarios. In e-commerce a customer would usually ask about item availability, sales and refunds. Insurance clients ask about the pricing of a plan, it’s coverage and available medical facilities. People who require medical assistance need to schedule appointments, call for medical help, ask to retrieve medical history. All of those

problems can be solved using chatbots so that explains why they are so popular in those industries.

“Fifty-four percent of the chatbot developers worldwide worked on chatbots for the first time in 2016.” (Sumit, 2019)

The demand for chatbot developers is growing exponentially, more and more companies decide to develop a bot for their brand, while a lot of new chatbot oriented start-ups are created. This is a good time to learn how to build a chatbot, but first it’s important to get better understanding of general capabilities for most chatbots.

3.4 Capabilities of a chatbot

For a problem to be solvable by any chatbot it has to match some specific criteria, which would allow the chatbot to keep the positive user experience and solve the problem. Since the structure of human language is complex and human needs are infinite it is hard to develop chatbots capable of doing everything what the users need. But if the problem matches those requirements, implementation of the bot should be completely possible.

3.4.1 Is it possible to solve the problem via dialogue?

The first important question you should ask yourself if you are planning to design a chatbot for a specific purpose is: “Is it possible to solve the problem via virtual dialogue?”. To answer this question, you have to imagine a use case scenario and it should become more obvious if the chatbot can fulfil the user’s needs.

Let’s imagine a situation: A user came up with an interesting project and needs to cut several sheets of metal into custom dimensions. He has to compare the cost of this operation from several manufacturers. To do that the user has to call each of the manufacturers and describe his needs, or he could use a chatbot which was developed by one of the manufacturers to find out the same kind of information. The chatbot would require user to send the dimensions, name of material and number of required pieces for the bot to calculate the price of manufacturing.

A benefit of such system would be its constant availability, low complexity due to the general prices being the same for every user and faster performance since a chatbot needs less time for calculations than a human.

Let's imagine another situation: A developer imagines a chatbot that would perform the image retouching instead of a photo editor. He creates a machine learning model trained on thousands of unedited and edited images so the model would mimic the retouching process. Then he gives the bot to his friend to try to edit his own photos. His friend uses the bot and receives a better-looking edited image, but it's nothing like what his friend was imagining this picture to be like, he is unsatisfied with the result.

The friend's disappointment is caused by the problem being unsuitable to being solved via dialogue, because photo editing requires very specific manual correction which make the whole editing algorithm different from photo to photo.

Those scenarios were supposed to demonstrate the difference between the problems that are suitable and unsuitable to be solved via dialogue, but there is a couple of other important questions we have to ask ourselves to continue.

3.4.2 Is it possible to automate the solution?

An automated solution means that it works accurately while taking the factors of the user's request and real world's situation into account. If it doesn't, it won't be able to fulfil the user's needs.

For example booking a table in a restaurant could be easily automated because the table reservation systems are connected to some kind of API, but if you would try to implement a bot that allows users to not only reserve a table but also order the food in advance a lot of new problems would rise. Sometimes the restaurants cannot provide specific dishes due to the lack of ingredients. Many users have specific requirements to their dishes due to allergies or personal preferences. This would significantly increase the variety of situations the bot would have to be working with and complexity of the whole system that has to be implemented, and even then, it still might not be able to satisfy the user since this task also requires more personal approach.

The above example demonstrates how trying to increase the number of chatbot's functions may reduce the overall quality of user's experience, so it is important to keep in mind the limits of your application and try to balance the complexity and functionality of a chatbot.

3.4.3 How big is the informational scope for the bot?

If the purpose of the bot is question answering, then the most significant for the future development of the bot would be: “How wide is the range of the question that could be asked?”. The smaller scope of questions – the easier it is to answer them.

For example a technical service chatbot of an e-shop would have to answer the questions related to the availability of the item, delivery time, status of the order, refunds policy, such questions can be easily answered by the chatbot by parsing the corresponding answers from the database since such scenarios would stay similar for most of the users.

Another example would be a medical chatbot. It is possible to make a chatbot that would recommend what medical facility a person should visit based on the character of the persons illness or request, and the person’s location. The scope of answers would be acceptable, it would consist of closest medical facilities and their field of treatment.

But if the developer would like to create a bot that would provide its own medical recommendation based on the users diagnose – it would be very complex to create a bot that has appropriate expertise to provide reliable treatment even when given the name of the illness. The scope of the possible treatment is very big for every specific disease, and the appropriate treatment is heavily dependent upon other factors of the user’s health.

Those examples show how the scope of the answers required from the chatbot determines how complex would it be to build the system, and if it would be possible at all. Chatbot creators have to keep that in mind and carefully analyse the task they want to solve in order to succeed with future projects.

Now when it’s known how to find out if it is possible to solve the desired problem by a chatbot, it’s important to know which steps to follow while designing a chatbot.

3.5 Designing a chatbot

The process of designing is one of the key stages of development of the chatbot. It requires specific attention in order to make the implementation process structured and consistent. If the design of a chatbot isn’t done well enough, developers will encounter unplanned problems which will drastically complicate the process of development.

The designing process can be divided into four stages: **definition of intents, analysing the variance of intents, entity definition** and **design of the dialog’s algorithm**.

3.5.1 Definition of intents

Intent is the operation which the user requests from the chatbot in his message. The intents are different for every operation that a chatbot is able to do and sometimes performing one complete operation requires resolving of several intents.

The process of designing a chatbot always starts with definition of intents since it's a crucial part of the whole chatbot system and it defines to what kinds of messages the chatbot will be able to react. This is a list of intents that can be a part of a chatbot developed by a delivery company:

- Show pricelist.
- Set users location.
- List the offices nearby.
- Show delivery information.
- Set declaration number.
- Leave a review

“Every task that you want your chatbot to do will define an intent.” (Sumit, 2019)

Even though this process doesn't seem much complicated, neglecting it would make the bot simple uncappable of understanding what the user is asking for. If the intent definition is done properly it will allow the bot to function accurately while paying attention to all the user's requirements.

3.5.2 Analysing the variance of the intent queries

After defining the intents which the bot should be able to classify, we must specify all the different ways the user may send the same intent. This is done in order to make the bot able to recognize the same question but in different ways. Same intent can occur in different form due to the existence of synonyms and different phrases that share the same meaning. For example, a user may ask a customer assistance bot about the availability of some product in many ways: “Is the product A available?”, “Do you have product A in store?”, “Is there a product A?”, etc.

Here is a list of possible queries that might occur for the possible intents of the delivery company's chatbot user:

- “Pricelist”, “What is the price for local delivery?”, “How much would it cost to send a letter from Prague to Brno?”, etc.

- “I am in Prague 6”, “I am in Prague”, “I am located in Brno”, etc.
- “Where is the closest office?”, “Show me the offices near me”, etc.
- “Where is my package?”, “What’s the status of my order?”, “When my package is going to come?”, etc.
- “#12345678”, “The number is 12345678”, “12345678”, etc.
- “I want to leave a review”, “I really enjoyed your services”, “How do I write a comment?”, etc.

If the variants of intents were specified correctly, chatbot will be able to classify the user’s intent correctly which is essential for a chatbot to function correctly.

3.5.3 Definition of entities

A request sent to a chatbot might contain not only an intent but also some parameters which are supposed to change the outcome of fulfilling the request. Those parameters are called an entity. For example, when the user asks, “Is product A in stock?” – the intent of the request would be “check availability” and the “Product A” would be the value of the entity “*product_name*”. This allows the machine to save the name of the product the user is looking for, look for its status in the database and send the status to the user.

Here is a list of the entities that should be defined for implementation of the example delivery company’s chatbot:

- “*delivery_type*”, “*delivery_from*”, “*delivery_to*”, etc.
- “*location*”
- “*search_location*”
- “*package_number*”

Without a proper definition of entities chatbot won’t be able to adapt to users need which will lower the scope of possible operations.

3.5.4 Design of the dialog’s algorithm

The final step in designing a chatbot system would be design of the dialog’s algorithm itself. It means that the designer has to direct the conversation between the user and the bot to create such a scenario that would make the user bounded to say what he has to say, while making the bot asking question that are needed to solve the problem. This step

defines how comfortable the user experience is going to be and what the dialogue between the user and the bot is going to look like.

Here is an example how a dialog between the bot and a user who wants to check the status of his delivery (User = U, Bot = B):

U: Hello

B: Greetings! How can I help you?

U: Where is my delivery?

B: Please, type in the declaration number to check the status of the order

U: 12345678

B: The status of your order is: on the way; The approximate delivery time is: 4 days. Do you need any other assistance?

U: No

B: It was a pleasure to help you!

This example demonstrates the structure of a dialogue needed to allow the user to use the bot to check the status of the delivery, it consists of several steps: activation of the bot, specifying the intent of users request, getting the order number, fulfilling the request, offering additional assistance.

This kind of planning must be performed for every scenario the bot should be able to perform, and for different variations of the same scenario. For example, the user could have specified the order number in the same message in which he asked to check its status. The bot should not ask for the order number again but start checking its status immediately.

The steps described above should be enough to design the functionality of a chatbot so now it is important to find out, which technologies can be used to implement it.

3.6 Natural Language Processing

Since a chatbot is a program that takes the input in form of a message sent from a user the core technology that must be implemented is the ability of a chatbot program to understand the human language. Techniques used to allow machines to interpret the language of humans is called “Natural Language Processing”.

The techniques that would be essential to use in the process of creation of a chatbot would be directed towards dealing with the useless information in the text, normalizing the text, transforming the words into numbers, entity recognition.

Nowadays there are many frameworks for creating the chatbot that take advantage of those technologies without a need for developer to implement them on his own, but it's still important to understand their purpose and underlying logic in order to create a good working dialogue agent.

3.6.1 Segmentation

Segmentation is a technique which is used to separate the text into sentences. For us humans it doesn't make much sense, but when we pass a paragraph of text into a machine it doesn't have any idea of what to do with it, so we try to present to it as many insights as possible.

So, if we would perform segmentation on this text:

“Is it raining outside? I want to go for a stroll.”

The output would look something like this:

([“Is it raining outside?”], [“I want to go for a stroll”])

Even though the input into chatbot application is already a single sentence, this technique might be used while building a corpus to train a language model.

3.6.2 Part-of-Speech Tagging

Part-of-Speech Tagging (POS Tagging) is used to specify which part of speech every word in the text is, such as noun, verb, adjective, etc. If the dialogue agent was trained on a dataset with specified POS tag for each word it will perform better than the agent that was trained on the dataset without those tags because those tags provide more specific information for each word in the text, which is not accessible to a machine that wasn't trained for it.

After performing POS Tagging on this text:

“*How can I reserve a table?*”

The result should look like this:

(‘How’, ‘Adverb’), (‘can’, ‘Verb’), (‘I’, ‘Pronoun’), (‘reserve’, ‘Verb’), (‘a’, ‘Determiner’), (‘table’, ‘noun’)

3.6.3 Lemmatization

The same verb in the human language may be written in many different forms while meaning the same thing for us – humans. For example, verbs “Ride, riding, rides” are all spelled differently while they mean almost the same process – riding. We should make all three of those verbs look the same by only leaving the **lemma** of this word. Lemma is the base of the verb in a human language. For the verbs “Ride, riding, rides” the lemma would be – ride.

“If we do the lemmatization of the original question before going to match the input with the documents, then we may get better results.” (Sumit, 2019)

3.6.4 Extraction of entities

When people hear a word they know a whole hurricane of associations starts to swirl in their head, when we hear “Jack Horwitz won 150K USD in a lottery.” we image what we could have bought with that money, when we hear “We expect a long draught.” we know that this is about climate. The machine doesn’t know what the meaning behind the string of words is, it only knows the meaning behind a little amount of codewords that were programmed inside of it, or it may use a language model pretrained for some purposes. The developers try to provide even more context for a machine by performing recognition of entities. They use a pretrained language model to try to find the entities in the sentences. For example, if we would perform named entity extraction of this text:

“My flat is on Prague 6, Suchdol, the rent is 20000 CZK”

The output would look something like this:

([“Prague”,”LOC”],[“Suchdol”,”LOC”],[“20000 CZK”,”MONEY”])

This technique is widely used in preprocessing the corpuses for training of language models and will be useful for entity recognition in chatbots.

3.6.5 Removing the stop words

The structure of a human language is deliberately complex and sometimes the words are in their place only in order to comply with the ancient rules and satisfy the inner logic, but not to make a difference to the sense or the general meaning of the sentence. Stop words are example of such redundant words, such as “a, to, for, on”.

Removing the stop words from the sentences doesn't reduce the amount of context in the sentences but makes the amount of data to analyse smaller and allows computers to easier identify which parts of the sentence provides the most important insides.

If we would remove stop words from this text:

“The house is located in a suburban area. There are many shops in a walking distance. If you would like to go to the city centre you have to use the bus and metro, it would take you about 25 minutes to get there, or 20 minutes by car. The area is calm and green. A big supermarket is on the way home. A lot of parks and hills are around the area. The closest hospital is nemocnice motol. Thy parking costs are 500 kc per month.”

The output would look something like this:

(“The house located suburban area. There many shops walking distance. If like go city centre bus metro, 25 minutes, 20 minutes car. The area calm green. A big supermarket home. A lot of parks and hills are around the area. The closest hospital is nemocnice motol. Thy parking costs 500 kc per month”)

Before trying to perform any kind of natural language analysis, removing stop words is a very important part of preprocessing for the textual data because it helps to remove the useless noise from it.

3.6.6 Analysing dependencies

Dependency parsing is a technique used to extract the syntactic structures from the sentences by tying together the phrases using special relations. To do that at first the leading word has to be found and then try to identify the relations with the words that are dependent from the leading word.

“If we try to think of a real-world scenario that we might actually face while trying to build a chatbot, we may come across some sentence like I want to book a cab to the hotel and a table at a restaurant. In this sentence, it's important to know what tasks are requested and where they are targeted (i.e., whether the user wants to book a cab to the hotel or the restaurant).” (Sumit, 2019)

If the dependency parsing would be performed on this sentence:

“A lot of parks and hills are in the area.”

The graphical result of dependency parsing would look like this:

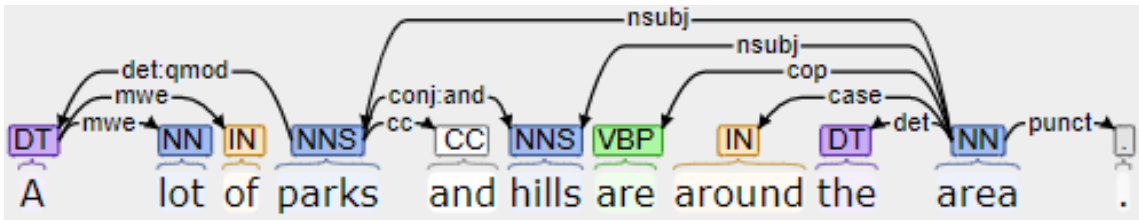


Figure 3 Graphical representation of dependency parsing using corenlp.run

This analysis shows us that the „parks“ and „trees“ are tied in a conjugation dependency which means that they are united in the context. Together they are related to the area in a subjective dependency.

This technique is widely used in all areas of applied natural language processing, but it is especially important for building chatbots from scratch.

There are quite a lot of possible use cases which can be achieved using this technology:

- Detect the dependencies between words.
- Separate sentences into logical parts.
- Detect several contexts in one sentence.

3.6.7 Combinations of nouns

In real texts it is very often that one entity is explained by many separate words, this phenomenon is called “noun chunks”. A noun chunk is a combination of a noun with the words that compliment it.

“You can think of noun chunks as a noun with the words describing the noun.”

(Sumit, 2019)

Identifying those combinations of words helps to increase the amount of information about the text which we provide to the computer thus making it easier for the computer to understand the given text.

If we would perform this kind analysis of on the given sentence:

“My favourite place, where I could walk my dog is the local park.”

This would be the resulting output:

([“My favourite place”], [“my dog”], [“local park”])

Such grouping of the words from the sentences makes it much easier for the machines to identify dependencies between them.

3.6.8 Working with similar words

The technique of detecting if the meaning of words is similar to one another and calculating by how much are they are semantically similar to each other, is one of the essential problems of natural language processing. It is very important to know if the two words would point to the same contexts or two separate ones, even though they might be expressed by almost the same or completely different characters.

The semantic similarity between two words is expressed by a numeric value, usually ranging from 0 to 1, with 0 stating that the meaning of the words is not similar at all and 1 stating that the meaning of the words is the same. This value is obtained from the tools that are used to perform NLP, such as NLTK, Scikit-learn or spaCy.

Let's have three words A, B and C:

["word A", "buy"], ["word B", "bye"], ["word C", "purchase"]

If we would analyse by how much the meaning of word A is similar to the meaning of word B and by how much the meaning word A is similar to the meaning of word C, the output will look similar to this:

("Similarity of word A to word B", 0,37212121)

("Similarity of word A to word C", 0,83455555)

As it is shown the meaning of the word "buy" is much more similar to the word "purchase" then to the word "bye", even though two out of three letters in the words "buy" and "bye" are the same. This is obvious for humans since the meaning of the words "buy" and "purchase" is the same – buying something.

In the process of building chatbots the technique of finding the similarities between words is mostly used for such purposes:

- Creating recommendation systems.
- Significantly improving classification of intents
- Excluding duplicates.
- Detecting grammatical errors.

3.6.9 Tokenization

After having the text split into messages, it must be split into single objects to allow the machine to analyse each one of them separately. A single entity of a message is a word.

This problem seems like a very easy to solve by just splitting the sentences on the spaces, but the problem isn't that simple. Sometimes a single entity is described by an abbreviation which might contain dots inside of it, or two words being responsible only for one single thing (not noun chunks). It is important to take the meaning of the word into account while performing the tokenization.

For the sentence:

“Prague is located in Czech Republic which is a part of European Union”

The tokenized output would be:

([“Prague”, “is”, “located”, “in”, “Czech Republic”, “which”, “is”, “a”, “part”, “of”, “European Union”.])

In the scope of chatbot development this technique helps with:

- Improving accuracy of intent prediction.
- Working with entities consisting of several words.

All the above-mentioned techniques are crucial for building systems that are supposed to perform Natural Language Processing. But what tools can a developer use to make NLP systems faster by taking the advantage of the technologies built by the developers for the developers.

3.7 Toolkit of chatbot developer

There are a lot of different tools which allow developers to create chatbots and any other programs faster and on a higher level. A chatbot developer can take advantage of different programming languages, frameworks and API's. Thanks to those tools, developers can concentrate on the bigger picture and save time developing the functions that can be used from the toolkit. It's important to select the tools which are most suitable for implementation of the system. A poor choice of those tools may lead to:

- Longer time of development.
- Bad performance of the resulting system.
- Problems with the maintenance of the system.

But, with the right choice of tools there shouldn't be any problem with making your own chatbot.

3.7.1 Python programming language

Python programming language is a scripting language which gained high popularity in fields of general software engineering, web development, data engineering, analytics and programming for scientific research.

Python was released in 1991 and was created by a Dutch programmer Guido van Rossum. The key features of python programming language are that it is an **interpreted, high-level, general-purpose** programming language, capable of having separate **environments**.

Interpreted programming language is the opposite of compiled programming language, it means that the execution of the program doesn't require the transformation of the original code into machine instructions. Instead, a program written in python would require a separate special program called interpreter to run it. This interpreter program runs the commands from the program which was opened by interpreter using the packages that are stored in the separate environment of this interpreter.

High-level programming languages are the languages that are easy to understand by humans, unlike low-level programming language. The assembly, C, C++ are low-level programming languages, the code written in low-level language is quite similar to machine code which makes it very hard to interpret by a person who doesn't have any experience with it. Another benefit of using high level programming language is that they have a bigger set of libraries and packages for all kinds of task that are going to be implemented using the mentioned programming language.

```
def fetch_housing_data(housing_url=HOUSING_URL, housing_path=HOUSING_PATH):
    if not os.path.isdir(housing_path):
        os.makedirs(housing_path)
    tgz_path = os.path.join(housing_path, "housing.tgz")
    urllib.request.urlretrieve(housing_url, tgz_path)
    housing_tgz = tarfile.open(tgz_path)
    housing_tgz.extractall(path=housing_path)
    housing_tgz.close()
```

Figure 4 Definition of complex function in Python (Aurélien, 2017)

Because a higher-level programming language requires more complex transformations before it can be executed by the machines which means that they will require longer time to run.

Python is a general-purpose programming language because it's use cases can be found in very different areas. A big part of YouTube's backend is written in python. It is very popular for automation of repetitive tasks. Also, it is widely used in data analytics due to availability of many packages designed to perform various operations such as natural

language processing, machine learning, deep learning, web scraping, creating desktop applications, frontend and backend web development, implementation of dialogue agents. The variety of tools made for data analysis using python makes it the most suitable language to make chatbots with, but a big variety of packages makes the process of selecting those tools more complicated.

3.7.2 What is a python environment?

The first version of python was released in 1991, the latest stable version python of 3.8 was released in December 2019. To allow developers to work with different versions of python and its packages the environment exist. A python environment is a separate folder that contains the interpreter of the installed python version and the packages that were installed in it.

To create an environment the “*virtualenv*” package is used.

```
$ cd $ML_PATH
$ virtualenv env
Using base prefix '['...']'
New python executable in [...]/ml/env/bin/python3.5
Also creating executable in [...]/ml/env/bin/python
Installing setuptools, pip, wheel...done.
```

Figure 5 Creating virtualenv (Aurélien, 2017)

To use the environment for running the scripts it must be activated through the command line interface by running the “*activate*” file.

```
$ cd $ML_PATH
$ source env/bin/activate
```

After activating the environment, the pip package manager will install the new packages into that environment and the scripts which are executed through the environment will use the packages that are stored in it.

3.7.3 The pip package manager

It’s quite rare for a python program to be implemented without using any packages. To use the package, first it must be installed on the machine. To do that the pip package manager was created. It can install, upgrade, downgrade or remove any python package including itself. It is a default component of a python distribution since python version 3.4.

The commands that are most commonly used to operate the pip package manager are:

- *pip --version* – outputs what version of pip is used in the activated environment.
- *pip install *package_name** - to install a package.

- *pip install *package_name==1.0.3** - install a specific version of a package
- *pip install --upgrade *package_name** - upgrades a package to the latest version.
- *pip uninstall *package_name** - remove a package from the environment.
- *pip list* – outputs the list of all packages and their versions installed in the environment

The pip package manager is very easy to use and allows developers to effortlessly control which packages are installed into the virtual environment.

3.7.4 Python packages

A big advantage of python over the other programming language is that it has the biggest number of packages created to enhance the development process, but first, what is a package?

A package is a collection of predefined functions that allow developers to use them without the need to make those functions by themselves. Each package is installed using the package manager into the active environment which allows python developers to store different versions or combinations of packages on one computer.

Python’s performance speed is relatively slow compared to low level programming languages, that is why a lot of packages that are used in the python scripts have their core functionality written in lower-level programming language such as C and C++. This technique helps to combine the performance speed of low-level programming languages with the rapid development of python scripts.

There are various packages each made for its own purposes. The “big” and popular packages get a lot of support from community which allows to improve them and create the new updated version. Sometimes the syntax and the core logic behind the package (and the python core itself) can change. This might lead to problems if your program was built using the previous version of the package or python itself because the new syntax of logic might not be compatible with you program. Also, it is possible to get lost in all of the different packages and their versions, so it’s important to know which ones are the most suitable for the system which is going to be created.

3.7.5 SciPy

Python is very popular language for scientific research. Scientific programmers who use python have to work with arrays, matrices, perform statistical data analysis and visualize

their data. Since all of these functions are not available in python itself, except some statistical and mathematical operations, the developers have to use some other tools and python packages.

SciPy is a whole ecosystem of scientific python programming tools, which is sponsored by NumFOCUS. These tools are used by developers to create complex arrays and data structures, visualize data and use interactive shells. SciPy has received a great community support which is working everyday towards both improving and keeping the system stable.

The most popular tools from SciPy are:

- NumPy
- Pandas
- Matplotlib
- IPython

It is quite hard to find python project nowadays that wouldn't take advantage of those libraries. All of the mentioned tools are capable of performing very complex operations, but first the developer has to learn how to use them. The documentation of SciPy is very well structured and it is available at the official website.

3.7.6 NumPy

It was already mentioned that python developers have to create and work with complex arrays and matrices. To do that a SciPy system includes a package called "NumPy". The main functions of NumPy is creation of multi-dimensional NumPy arrays (ndarrays) and performing operations on the data stored in those arrays. The core functionality of the package is written in C, C++ and Fortran programming languages, which allows this package to work on high performance speeds, while keeping the simple high-end syntax to be used in python.

Since the python itself has the list and the dictionary data structures which are capable of storing the data of almost any type – it is not obvious, why the developers might need a separate package for that, the answer lies in the way how a developer would perform operations on data in this array. To perform an operation upon each specific value within the list or a dictionary using only built-in python functions the iterative approach would have to be used.

“The Python list object can store nearly any type of Python object as an element. But operating on the elements in a list can only be done through iterative loops, which is computationally inefficient in Python.” (Bressert, 2013)

A developer would have to create a cycle which would go select the first element of the data structure, perform the operation, go to the next element and repeat. Let’s imagine a problem, a programmer has to increase each value of an array by one. The code to solve such problem using python would look like this:

```
py_array = [1, 2, 3]
for i in py_array:
    py_array[i] = py_array[i] + 1
```

if the same problem would be solved using NumPy the code would look like this:

```
import numpy as np
numpy_array = np.array([1, 2, 3])
numpy_array = numpy_array + 1
```

It is obvious that performing an operation upon the whole array at once is easier than having to create a cycle to go through all of the elements of an array.

Another data structure available for creation using the NumPy package is a matrix. The main difference of NumPy matrix from a NumPy array is that matrix can only be two dimensional, while an array can be multi-dimensional. Another feature of NumPy is availability of mathematical functions for matrices which allow developers to perform linear algebra operations upon given matrices.

“Should we need linear algebra operations, we can use the matrix object, which does not use the default broadcast operation from ndarray. For example, when you multiply two equally sized ndarrays, which we will denote as A and B, the n_i, j element of A is only multiplied by the n_i, j element of B. When multiplying two matrix objects, the usual matrix multiplication operation is executed.” (Bressert, 2013)

The numpy package can be installed using pip package manager using this command:

```
pip install numpy
```

3.7.7 Pandas

Since python is widely used for data analysis the developers have to be able to create more complex and bigger data structures, that are able to store data of different times, while have

high performance speeds due to the need of analysing big volumes of data. The SciPy collection contains a special package for those needs called Pandas.

It includes functions to create rich data structures and perform different operations upon them, which allow data scientists to wrangle the data faster and easier.

The fundamental object that is used in pandas is called “DataFrame”, a simple representation of a DataFrame in the real world is a spreadsheet. DataFrames are structured as a two-dimensional table of rows and columns. Each row and column have a specified name which can be used to access them. Each row is supposed to represent a single entity in a dataset and each column is supposed to represent a single feature of this entity. For example, a list of company’s personnel, which specifies their name, the city they are from, the number of hours each of them has worked and the amount of money they have earned would be represented in DataFrame consisting from as many rows as there are workers and four columns:

- Name
- City
- Hours
- Money

The Pandas library is widely used in financial analysis thanks to the functions used for working with financial and time series data. The creator of Pandas says:

“I initially designed pandas as an ideal tool for financial data analysis applications”.

(McKinney, 2012)

The pandas package can be installed by executing this command:

```
pip install pandas
```

3.7.8 Matplotlib

It is well known that images contain much more information than words and numbers.

While working with data it is important to get the best understanding of the dataset you are working with and create informative reports, this is done by performing analysis on the data and visualizing it. The developers have to create different plots and diagrams in order to make easier to interpret the data for themselves or people to who they present it.

The SciPy ecosystem contains a special package called “Matplotlib” which was specifically designed for such purposes. It allows developer to create plots and diagrams of

various types. The plots created in Matplotlib can be visualized as output in python shell or exported in PNG, JPG and BMP formats.

Let's imagine some data that has to be plotted:

```
X = [1, 2, 3, 4, 5]
```

```
Y = [2, 4, 6, 8, 10]
```

To plot this data two separate lists must be created:

```
x = [1, 2, 3, 4, 5]
```

```
y = [2, 4, 6, 8, 10]
```

Then, we just have to plot from those images:

```
import matplotlib.pyplot as plt
```

```
plt.plot(x, y)
```

That's it, to show the plot in console we have to write:

```
plt.show()
```

The resultant graph would look like this:

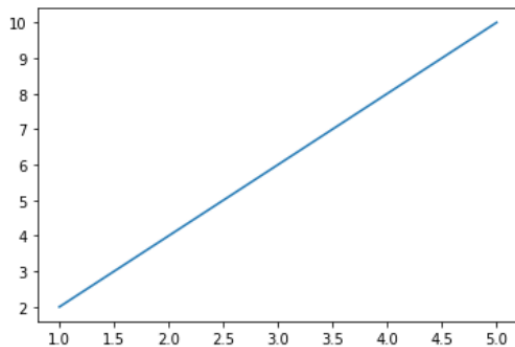


Figure 6 The resulting plot using Matplotlib

“Matplotlib is not the first attempt at making the plotting of graphs easy. What matplotlib brings is a modern solution to the balance between ease of use and power.”

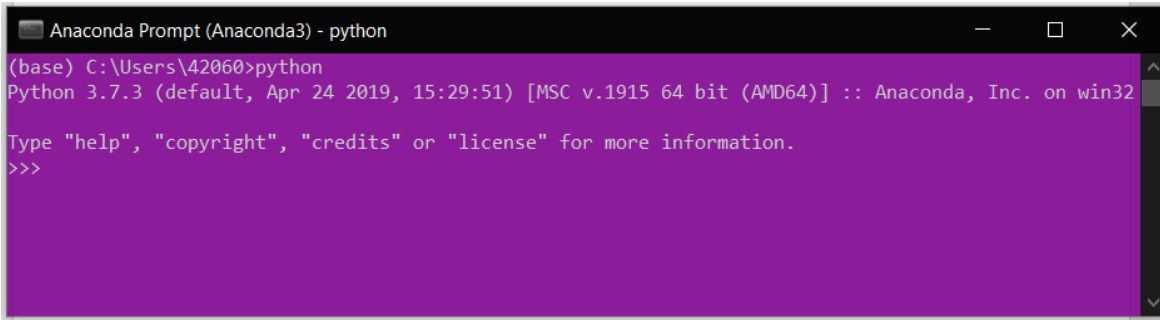
(Devert, 2014)

To install the matplotlib package this command should be executed:

```
pip install matplotlib
```

3.7.9 IPython

There are various ways of developing python applications. The developers may use IDE programs such as Atom, PyCharm and sublime text. Also, python commands can be executed through a python shell which is accessible by typing the *python* command into the command line.



```
Anaconda Prompt (Anaconda3) - python
(base) C:\Users\42060>python
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)] :: Anaconda, Inc. on win32
Type "help", "copyright", "credits" or "license" for more information.
>>>
```

Figure 7 Python interpreter access through command prompt

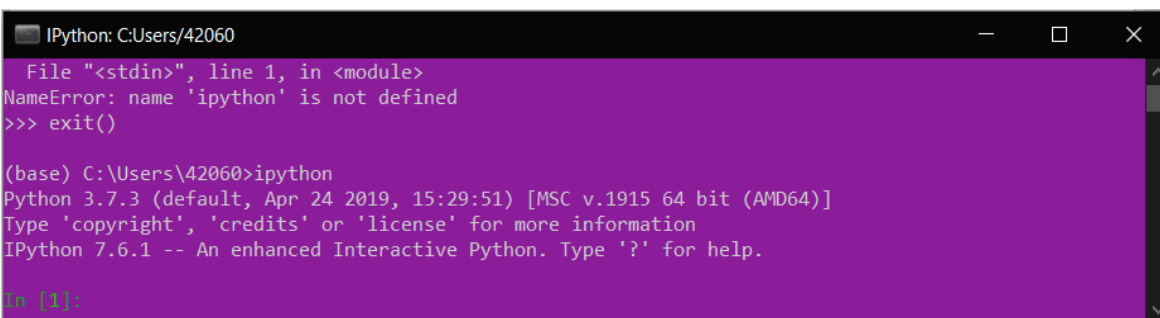
But, this type of development is not interactive enough since all of the commands in a python script would be executed one by one, this approach is suitable for development of python programs, such as scripts for web applications. But, this type of programming is not much interactive, the developers might want to execute block of code separately while keeping all of the outputs visible.

The SciPy system of tools has one made to bring interactive programming to the new level. This tool is called “IPython”. It allows developers to use python for scientific purposes in a much more interactive manner.

IPython is used to create an interactive python shell, where the commands have to be written in logical blocks, which usually perform one logical operation. Those blocks can be executed in the order desired by developers and the outputs are printed in line with those blocks.

“Part of why IPython is so widely used in scientific computer is that it is designed as a companion to libraries like matplotlib and other GUI toolkits.” (McKinney, 2012)

This workflow allows developers to proceed to the next step of development after they get the previous block working like they want it to, or even go back a couple of steps, change some logic in the previous block, execute it, and proceed to the last step with the new values. To launch the IPython console we can execute the *ipython* command in the command prompt.



```
IPython: C:\Users\42060
File "<stdin>", line 1, in <module>
NameError: name 'ipython' is not defined
>>> exit()

(base) C:\Users\42060>ipython
Python 3.7.3 (default, Apr 24 2019, 15:29:51) [MSC v.1915 64 bit (AMD64)]
Type 'copyright', 'credits' or 'license' for more information
IPython 7.6.1 -- An enhanced Interactive Python. Type '?' for help.

In [1]:
```

Figure 8 Access to IPython console through the command prompt

To install Ipython console using pip this command should be executed:

```
pip install ipython
```

3.7.10 SpaCy

To create a working chatbot system the developer has to be able to perform advanced natural language processing. To do that there are various python packages available. The one which has gained the biggest popularity among developers is spaCy. This library is capable of performing all of the natural language processing techniques required for development of chatbots while keeping the record speed of performance. The core logic of SpaCy library is written in python and cython. Cython is a package that allows to translate python code into C language code to boost the performance.

”Two peer-reviewed papers in 2015 confirmed that spaCy offers the fastest syntactic parser in the world and that its accuracy is within 1% of the best available. The few systems that are more accurate are 20 times slower or more.“ (Sumit, 2019)

SYSTEM	YEAR	LANGUAGE	ACCURACY	SPEED (WPS)
spaCy v2.x	2017	Python / Cython	92.6	n/a ?
spaCy v1.x	2015	Python / Cython	91.8	13,963
ClearNLP	2015	Java	91.7	10,271
CoreNLP	2015	Java	89.6	8,602
MATE	2015	Java	92.5	550
Turbo	2015	C++	92.4	349

Figure 9 spaCy benchmarking results (Sumit, 2019)

The most significant feature of spaCy is its built-in deep learning models which allow developers to use them for natural processing “out of the box”.

To install spaCy using PIP package manager this command should be executed:

```
pip install spacy
```

3.7.11 Dialogflow

The main component of a chatbot is the dialogue agent. Dialogue agent is the part of the chatbot system which allows it to predict intents of user's messages, detect entities and select appropriate response. A dialogue agent may be created from scratch using natural language processing and machine learning tools. But such approach is quite complicated and requires a lot of work from the developers which would have to make all of this logic by themselves. Luckily there is a special platform owned by Google corporation specifically designed to develop and host dialogue agents. This platform is called "Dialogflow".

The key functions of dialogue platform are:

- Creation of dialogue agent
- Creating mega agents which consist of several agents
- Definition of intents
- Definition of entities
- Extraction of parameters
- Fulfilment

The Dialogflow API takes advantage of many built-in machine learning and natural language processing functions developed by Google. The logic of working dialogflow agent can be expressed using a diagram:

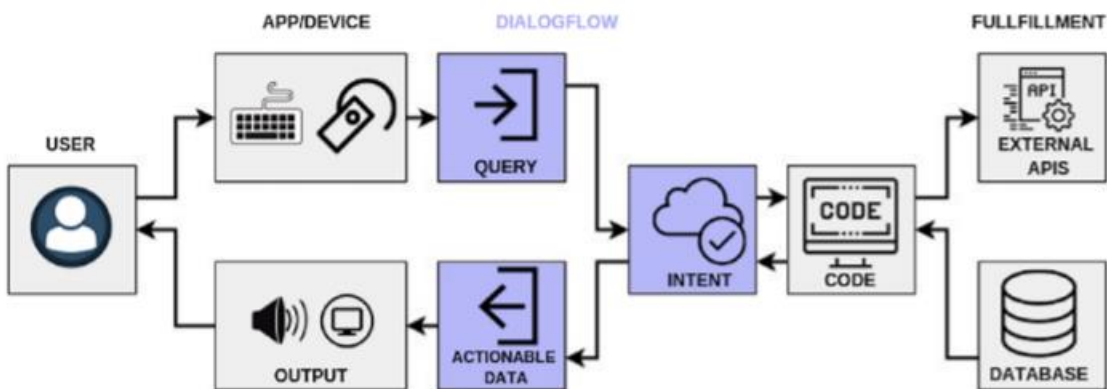


Figure 10 Working diagram of Dialogflow architecture (Sumit, 2019)

“Here is what happens: 1. User talks to the input device. 2. User query goes into the Dialogflow engine. 3. Dialogflow tries to recognize the intent. 4. Based on the intent, a fulfilment is done, and data is returned from database. 5. Response is returned to the intent. 6. Response is converted into actionable data. 7. User's request for information is given back to the output device” (Sumit, 2019)

To use the Dialogflow API in your python scripts the google cloud software developer kit and Dialogflow package for python must be installed. Those two packages can be installed using pip package manager by running these commands:

```
pip install google-cloud-storage  
pip install dialogflow
```

3.7.12 SciKit-Learn

Since the process of developing a chatbot system may require machine learning the developers need special tools for that. The most popular open source tool for creating machine learning models is SciKit-Learn. It contains various estimators, transformers and predictors designed to make the process of creating machine learning models easy and intuitive. This package can create various machine learning models ranging from regression models to classifiers. It allows developers to take advantage of supervised or unsupervised learning approaches.

To use SciKit-Learn to create a linear regression model and make a prediction using it such logic should be followed:

```
x = [*independent data*]  
y = [*dependent data*]  
model = sklearn.linear_model.LinearRegression()  
new_x = [*data to predict*]  
prediction = model.predict(new_x)
```

SciKit-Learn package could be installed using the pip package manager using this command: *pip install sklearn*

3.7.13 TensorFlow

During development of chatbot some functionality is too complex to be fulfilled by basic models. That is when deep learning models come in handy. This is the time to implement some deep learning models. Such models can perform more complex analysis such as natural language understanding, image classification and many others. In November 2015 Google has open sourced their tool for creation of deep learning models called TensorFlow.

“TensorFlow is a powerful open source software library for numerical computation, particularly well suited and fine-tuned for large-scale Machine Learning. Its basic principle

is simple: you first define in Python a graph of computations to perform and then TensorFlow takes that graph and runs it efficiently using optimized C++ code.”

(Aurélien, 2017)

One of the TensorFlow’s problems is that creation of complex deep learning models might be too hard for the begginer due to a complex syntax of this package, luckily there is a tool overcome this problem.

3.7.14 Keras

TensorFlow is a very powerful library for creating deep learning models but using it might be too complicated for a beginner. To make the process of developing easier and faster, there is special higher-level library for TensorFlow called “Keras”. It allows developers to build deep learning model using less complicated approach, while keeping all the TensorFlow’s features. It was developed by a google engineer Francois Chollet.

The main feature of Keras is how easy it is to create and use a deep learning model in just a few main steps:

1. Define the model architecture.
2. Compile the model.
3. Train the model
4. Use the model

This simple approach makes it possible even for complete beginners or people from other industries to understand the process of building deep learning models.

Let’s imagine a simple neural network: The first layer is the input layer, that takes 10 separate input values. The second layer is the interconnected dense layer of 64 neurons, powered by rectified linear unit function. The third layer is also interconnected and serves for output, it is able to classify 2 states, using softmax function to decide which of the two choices to select.

Here is how the code for creating such neural network would look like:

```
model = Sequential()  
layer1= Input(10)  
layer2 = Dense(64 activation = relu)  
layer3 = Dense(2, activation = softmax)  
model.add(layer1)  
model.add(layer2)
```



```
model.add(layer3)
```

```
model.compile(loss='mean_squared_error', optimizer = Adam, dropout=0.1)
```

Only 8 lines of code were needed to create the neural network, two more would be needed to train it and make a prediction:

```
model.fit(x, y)
```

```
print(model.predict(value_to_predict))
```

As the result the prediction would be printed out to console.

There are several ways to save the model for the future use, the most popular one is saving the models architecture in json file and saving the weights separately into hdf5 file format.

3.7.15 Google Collaboratory

The training of deep learning models requires a lot of computing power that's why training them on a machine that is not powerful enough might take a lot of time. Luckily for the developers who don't possess a powerful machine – there exist a special platform called Google Colab.

Google Colab allows developers to write their code through the special website and execute it on Google's powerful servers absolutely for free, the limit of length for a single session is 12 hours, after that it just has to be restarted. The data you want to save must be exported, since the data is deleted upon restart. The servers allow developers to use very powerful Tesla GPU's and TPU specifically designed for high computational load.

The platform is designed like an interactive jupyter notebook, which is based on IPython console. To set it running you simply have to login to your google account. All of the google collab notebooks are saved on the google drive and might be accessed later. It is possible to upload and download files to the notebook.

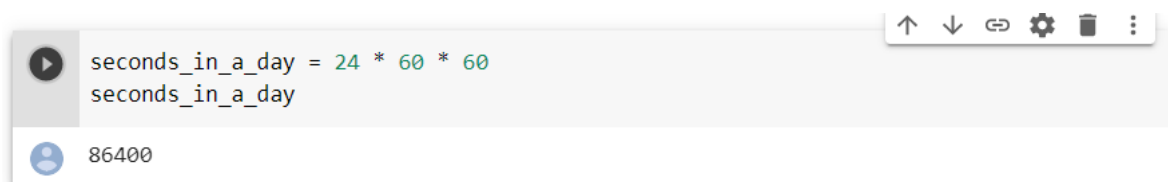


Figure 11 example of a code cell and its output

3.7.16 Bot API's

The easiest way to implement a chatbot once you have a working dialogue agent is to put it onto the messenger platform such as WhatsApp, Viber, Facebook Messenger and

Telegram. To do that the developer would require a special bot API which would allow them to receive and send the messages.

The APIs that get the most community support are Facebook Messenger's Bot API and Telegram Bot API. They allow developers to create bots that are capable of reacting to messages, processing audio and image data, download and send files. The choice between those two is left for the developers and depends on which of those two messengers is used more by the target audience of the chatbot.

For telegram bot API there are several wrappers that make developing a chatbot in python easier thanks to the predefined functions to fulfil the basic bot functions, the most popular one on GitHub is python-telegram-bot with 9,7k stars.

Python-telegram-bot allows developers to create the fundamental script of the bot, which is able to autonomously check for updates, create the dispatcher and different kinds of handler which are executed when the bot receives a command, text or a file, and send messages of different types.

3.8 Hosting the chatbot

Deployment of a chatbot written in python consists of several main steps:

- Selecting the hosting platform and machine
- Configuring the hosting machine
- Uploading the project
- Running the bot script

The selection of the hosting platform is dependent upon the specifics and requirements of the system, such as the expected number of users, processing power needed for the system to work, the pricing of the servers and their location.

After selecting the platform, the process of configuring the machine is straightforward. The dependencies of the chatbot system must be installed and the machine should be ready to go. Uploading the project could be done using special software such as FileZilla or using file sharing services. Another way is to upload the project to a GitHub repository and clone it to the machine.

Running the bot simply means opening the base script of the chatbot and disconnecting from the hosting machine.

3.9 Complete roadmap of making a working chatbot

Many things must be done in order to create a working chatbot system. The bot must be designed, developed and deployed. The main steps for these stages are listed below:

1. Design the chatbot
2. Select the platform
3. Configure the workspace
4. Create the base script for the bot
5. Develop the dialogue agent
6. Connect the dialogue agent to the bot
7. Develop the functionality of chatbot
8. Connect the functionality to a chatbot
9. Test the bot
10. Select the hosting service
11. Configure the server machine
12. Upload the bot to the server
13. Enable the bot

The process of developing a chatbot that has to utilize a lot of different technologies might seem complicated at first sight, but thanks to the work of previous developers who have been working hard to give us many libraries and packages, frameworks and a lot of other tools that make development of chatbots and general software development easier, there should not be many problems while working with chatbots.

4 Practical Part

4.1 Designing the chatbot

The first step in development of any system is designing it, so it is also the first step of designing a chatbot. I've decided to make a chatbot capable of three things:

- Chit-chat for the most common scenarios
- Tell jokes upon request
- Recognize handwritten digits.

4.1.1 Definition of intents

The first step in designing the chatbot is definition of intents. Since we want our chatbot to be able to have a small conversation, it has to be able to recognize the common intents for such scenario. Such intent would be greeting, saying goodbye, answering the simple questions about the bot like who it is, where is it from how it's feeling today and many others. In fact, the more intents are going to be included into the "chatting" part of the conversational agent – the better it is going to talk.

To make a chatbot be able to tell a joke it has to be able to recognize the request for a joke and understand if the user liked it or not. If the user liked it means that it might intent to hear another joke, so the bot should ask for it. The variance of the intents would be high since the bot would have to try to keep an actual small conversation. It means that a lot of different variants for intents must be specified.

For each time the bot doesn't understand what the intent of the user is, there should be specified the fallback intent, which makes the bot to ask the user to rephrase his query or apologize for not understanding him.

The digit recognition functions are not going to be connected to the dialogue agent since it is not needed, the package which is going to be used to create the bot is able to filter the images from other messages.

4.1.2 Definition of entities

The next step after defining the intents which could be detected by the conversational agent is the definition of entities. The entities are used to detect variables for the complex

scenario in the user's messages. The chatbot doesn't necessarily need entities involved in chitchatting scenario since it is performed in a single question-answer form.

The implementation of entities is needed for the joke-telling scenario. To allow the chatbot to understand if the user likes or dislikes his joke it is possible to implement a "feedback entity", but since the emotion of the feedback can be expressed in many different ways it is easier to implement this scenario using intents too, because to use the entities we have to predefine each specific word by which the user could express his feedback. If this problem would be approached using two intents: "good_feedback" and "bad_feedback" the bot would be able to detect such intent from a wider range of words sent from the user which would significantly increase the comprehensive abilities of the bot. It means that the entities will not be used for the implementation of this bot.

4.1.3 Draft of the bot's algorithm

The sequence of the messages in a chatting part of the bot functionality would be quite random, for example it might look like this: user (U), bot (B)

U: Hello!

B: Greetings!

U: How are you?

B: I am fine, what about you?

U: I am good

B: Glad to hear that.

To fulfil this functionality, we just have to specify as many intents as possible. The bot won't be able to reply to queries if it won't be able to detect their intent.

The algorithm of a joking part of the conversation is much more predictable, user ask for a joke, bot tells a joke, user reacts, bot either tells another joke or doesn't. It would look something like this: user (U), bot (B)

(U): Tell a joke, please

(B): I told my girlfriend she drew her eyebrow too high. She seemed surprised.

(U): Hahaha

(B): I like that one too. You want to hear another one?

(U): No, thank you

(B): No problem.

The logic for recognizing handwritten digits from a photo sent to a bot would lie in the program itself, not in the dialogue. After the bot would receive an update with the image in it the bot would have to download it to the server, pre-process the image to feed it into the neural network, use the neural network to give a prediction and send the prediction to the user. Also, the user should be notified when each of those steps is executed so he wouldn't get bored.

4.2 Select the platform for a bot

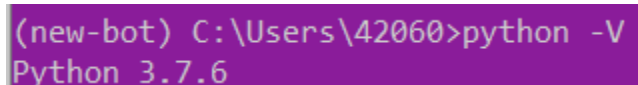
Since this bot isn't related to any company – it doesn't have to be connected to its own product. A great choice of a platform for a bot would be a messaging application such as Facebook Messenger or Telegram. Both of them have a great API for the chatbots, with many active bots using them right now, and a lot of resources online on how to use them. This bot is going to be put onto the Telegram platform since I, the developer, use this messenger the most.

Telegram's API for chatbots is called Telegram Bot API, it is HTTP-based. To access this API through the python scripts the developer could manually write a lot of functions, or use the python-telegram-bot package, which would make the development of basic bot functionality very straightforward.

4.3 Configuring the workspace

To start implementing the chatbot, the machine should be configured for the development process, which means all of the tools and dependencies must be installed. My machine uses the Windows 10 operating system.

The first step would be to install the python programming language. It can be downloaded from the Microsoft store or from the internet. I am using the anaconda distribution which has the python language included together with many other data analytical tools. The python version used for this project is Python 3.7.6.



```
(new-bot) C:\Users\42060>python -V
Python 3.7.6
```

Figure 12 Checking the python version

Now it's time to create a virtual python environment for the chatbot's project. This can be done in anaconda distribution or using the python itself. The conda environment is created

using the `conda create --name new-bot` command. After that the environment has to be activated everytime we want to access it through the command prompt. In anaconda environment is activated using `conda activate new-bot` command.

After installing python, creating and activating the environment it is time to check if the pip package manager is installed by running any pip command in the interpreter, for example `pip list`. It should be installed by default with the python distribution.

```
(new-bot) C:\Users\42060>pip -V  
pip 20.0.2 from C:\Users\42060\.conda\envs\new-bot\lib\site-packages\pip (python 3.7)
```

Figure 13 Checking the pip version

The next step would be using the pip package manager to install all of the dependencies for the development of chatbot. Using `pip install *package_name*==*version*` these packages must be installed: dialogflow 0.7.2; google-api-core 1.16.0; h5py 2.10.0; Keras 2.2.5 (and it's dependencies); matplotlib 3.1.3; numpy 1.18.1; opencv-python 4.2.0.32 (and it's dependencies); python-telegram-bot 12.4.2; tensorflow-gpu 1.15.0 (and it's dependencies).

After installing the dependencies, they should be listed in the output of `pip list` command. Now that the dependencies are installed it is time to download the integrated development environment, the software which is used to develop python scripts and projects. I am using the PyCharm IDE made by JetBrains. I am licensed to use the full versions of the IDEs developed by JetBrains as a student, but the functionality of the free version is enough for this project.

The neural networks used for the image recognition of the bot will be created on the development platform called Google Colab, a Google account is needed to set it up.

4.4 Implementing the basic bot functionality

To allow the chatbot's python script to work with the Telegram Bot API, the bot has to be able to check for the new messages, be able to send responses to the same dialogue, detect the datatype of the message, start the functions corresponding to what kind of message the bot has received. All of the fundamental bot functions for a python script are implemented in the wrapper package telegram-bot-api.

The first step would be to register a new bot using another bot which is created by Telegram and called BotFather. The developer has to start the dialogue with the BotFather by sending the `/start` command, after that the `/newbot` command must be used to create the

bot. The BotFather would ask for the bot's name and the username. The username of the chatbot must end with "bot" to make it obvious for the users. I've called my bot "Yuri" and set the username to "Yuri_bot". As the result the BotFather sends a token which is going to be used to access the bot.

Next step would be to create a new project using PyCharm, I've called the project "py-bot-yuri". Now the python script "yuri-bot.py" which is going to become the chatbot has to be created.

The components used for the fundamental functionality of a chatbot are the logger, updater, message filters, command and message handlers. First step is to import those into the script.

```
from telegram.ext import Updater
from telegram.ext import CommandHandler
from telegram.ext import MessageHandler
from telegram.ext import Filters
```

Figure 14 Importing basic bot functions

The next step would be to save the bot's token into the variable to make the access to it easier throughout the project. Then the token has to be added into the updater. Updater is the function that checks if the bot has received any new messages. Then the logger has to be configured using the instructions from the python-telegram-bot documentation, the logger is responsible for saving the messages that were sent to the bot.

```
token = '842440069:AAGNacGGPSgufJmJgeyCo29sB-zMp5vsp9Q'
updater = Updater(token=token, use_context=True)

logging.basicConfig(format='%(asctime)s - %(name)s - %(levelname)s - %(message)s', level=logging.INFO)
```

Figure 15 Configuration of the updater and the logger

Now it is time to create the first "start" command, this command is executed by the user to start the dialogue with the bot. It gets the chat id from the new message object called "update" and sends the message ". First, the function which is going to be executed after the bot receives the /start command must be defined:

```
def start(update, context):
    context.bot.send_message(chat_id=update.effective_chat.id, text="Hello! My name is Yuri, I am glad to help you")
```

Figure 16 Definition of the "start" command

Now the command handler must be created from the command and this handler must be added into the dispatcher:


```
start_handler = CommandHandler('start', start)
dispatcher.add_handler(start_handler)
```

Figure 17 Creation of command handler

The final step of implementing the basic bot script would be to enable the updater to check for the new messages. This is done by adding the `updater.start_polling` line to our script. Now when our python-telegram-bot tools are imported, the script is connected to the bot, the logger and updater are enabled, the first command is defined, handler is created and added to the dispatcher it is time to check if everything works fine. To do that the bot has to be found in telegram using its username and the `/start` command has to be sent as a message.

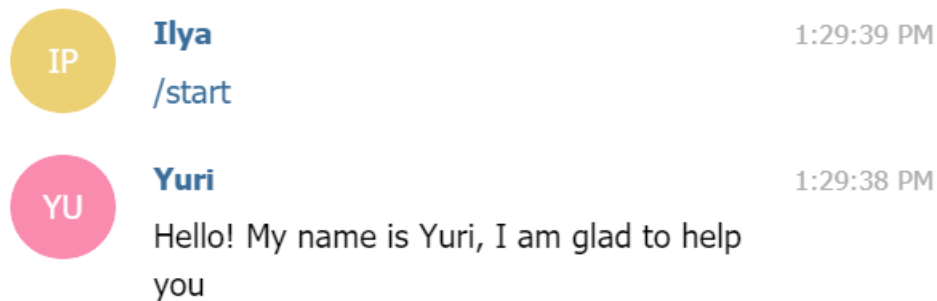


Figure 18 The first command sent to the bot

Everything works which means that now the bot's script is can receive new messages, execute commands and send a response.

4.5 Development of conversational agent

The part of a chatbot which works with text messages sent from a user is called a dialogue agent. It has to be implemented to allow the bot to chat and tell jokes. Conversational agent is responsible for detecting the intent of the query sent by the user and selecting the response.

For creating the dialogue agent for the chatbot Yuri, the Google's Dialogflow API is going to be used. It can add together several dialogue agents and unite them under single mega agent.

The first step would be to go to the Dialogflow website, then to sign up for the Dialogflow using the Google account. Then the developer would get redirected to the dialogflow console. The bot is going to take advantage of two separate agents for chit-chatting and telling jokes, which are going to be controlled by the mega agent. First step is to create the

new agent by pressing the "create new agent" button. The agent's name has to be specified and the agent has to be set as the mega agent.

The screenshot shows the Dialogflow agent creation interface. At the top, there is a text input field labeled "Agent name" and a blue "CREATE" button. Below this, there are two dropdown menus: "DEFAULT LANGUAGE" set to "English - en" and "DEFAULT TIME ZONE" set to "(GMT+1:00) Europe/Madrid". Underneath these are two lines of explanatory text. The next section is "GOOGLE PROJECT" with a note: "New GCP project will be automatically linked to the agent after saving". The final section is "AGENT TYPE", which has a toggle switch labeled "Set as Mega Agent" that is currently turned on. Below the toggle is a descriptive sentence: "Combine multiple Dialogflow agents (i.e. sub agents) into a single agent (i.e. mega agent)."

Figure 19 Agent creation window on Dialogflow

This agent isn't going to analyse the user queries by itself, instead it will allow both of the subagents to analyse it and select one that gets the higher confidence score.

Since our chatbot has to work in two different scenarios like chitchatting and telling jokes, the developer has to create two subagents for the corresponding tasks. Google's Dialogflow has a prebuilt agent for both of those tasks, the first one is called Small Talk and the second one is called Jokes.

To start, the prebuilt agents have to be imported from the "prebuilt agents" page. After importing, new separate agents are created, Small-Talk and Jokes. The Small-Talk agent is powered by a big set of almost a hundred of predefined intents designed to make the bot capable of keeping up in a small conversation. The intents include greeting, goodbye, emotions expressed to the bot, offering for help, accepting the gratitude and many others. They can be accessed through the "Intents" page. On the "Intents" page the developer can see all the intents defined for the bot and their priority. Each of the intents can have a place to define the training phrases which are used to demonstrate how the message with the corresponding intent should look like. New training phrases that are added to each intent will increase the agent's ability to detect the mentioned intent. Also, each intent has a set of responses to allow the agent to send different responses for each of the intents. The response is chosen randomly from the list of responses. It is important to add training phrases that represent the intent correctly or the agent won't be able to detect the intent.

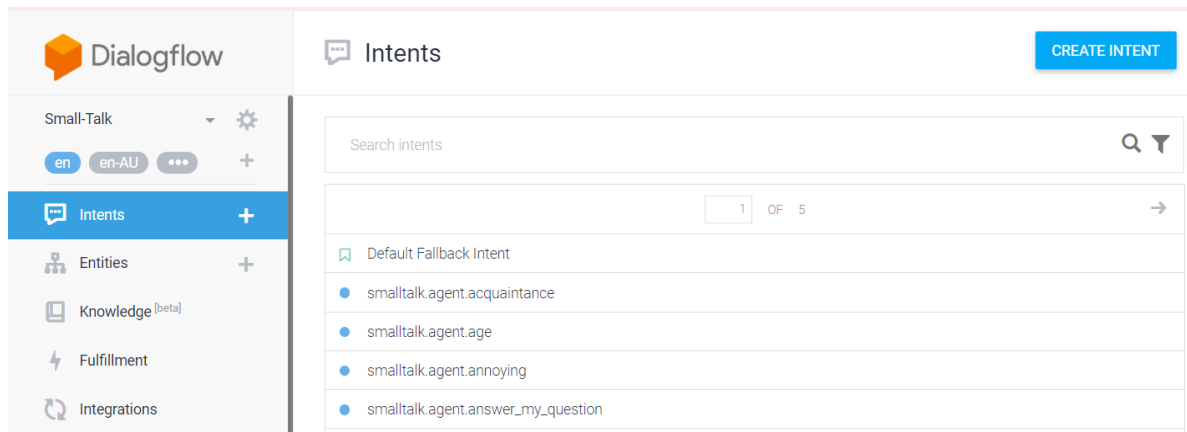


Figure 20 The Intents page of Small-Talk agent

Now that both agents are imported, the developer has to unite the two sub agents under one mega agent. It is done by assigning the “Dialogflow API client” role to the mega agent in the IAM page of each sub agent using the service account address of the mega agent.

Since the machine learning models used for Small-Talk and Jokes are imported predefined this is a good example of transfer learning. Configuring the intents and retraining the bot is a good example of fine-tuning neural network. Since the messages are not labelled when they are sent to the bot and it has to guess the intent by itself it is unsupervised learning.

After creating the complete dialogue agent for the bot, the developer has to connect it to a telegram script. First the developer has to obtain the JSON file used to work with the bot from the google service account page. After getting the JSON, developer has to specify the project id, language code and session id.

```
DIALOGFLOW_PROJECT_ID = 'yuri-mega-vaksua'
DIALOGFLOW_LANGUAGE_CODE = 'en-US'
os.environ["GOOGLE_APPLICATION_CREDENTIALS"] = 'yuri-mega-vaksua-8317d57a354d.json'
SESSION_ID = 'yuri-session'
session_client = dialogflow.SessionsClient()
session = session_client.session_path(DIALOGFLOW_PROJECT_ID, SESSION_ID)
```

Figure 21 Configuring dialogflow session

Now it is time to implement a function which is going to be executed when the bot receives a text message. It is going to parse the text from the update, input it into dialogflow and send the agents response to the user.

```
def yuri_response(update, context):
    print(update.message)
    received_text = update.message.text
    text_input = dialogflow.types.TextInput(text=received_text, language_code=DIALOGFLOW_LANGUAGE_CODE)
    query_input = dialogflow.types.QueryInput(text=text_input)
    response_intent = session_client.detect_intent(session=session, query_input=query_input)
    print(response_intent.query_result)
    response = response_intent.query_result.fulfillment_text
    context.bot.send_message(chat_id=update.effective_chat.id, text=response)
```

Figure 22 Text handling function

The command has to be made into message handler that filters the text messages, and the handler must be added to the dispatcher.

```
text_handler = MessageHandler(Filters.text, yuri_response)
dispatcher.add_handler(text_handler)
```

Figure 23 Making and adding the message handler

Now it is time to check if everything works, by saying a couple words to the bot and asking it to tell a joke.

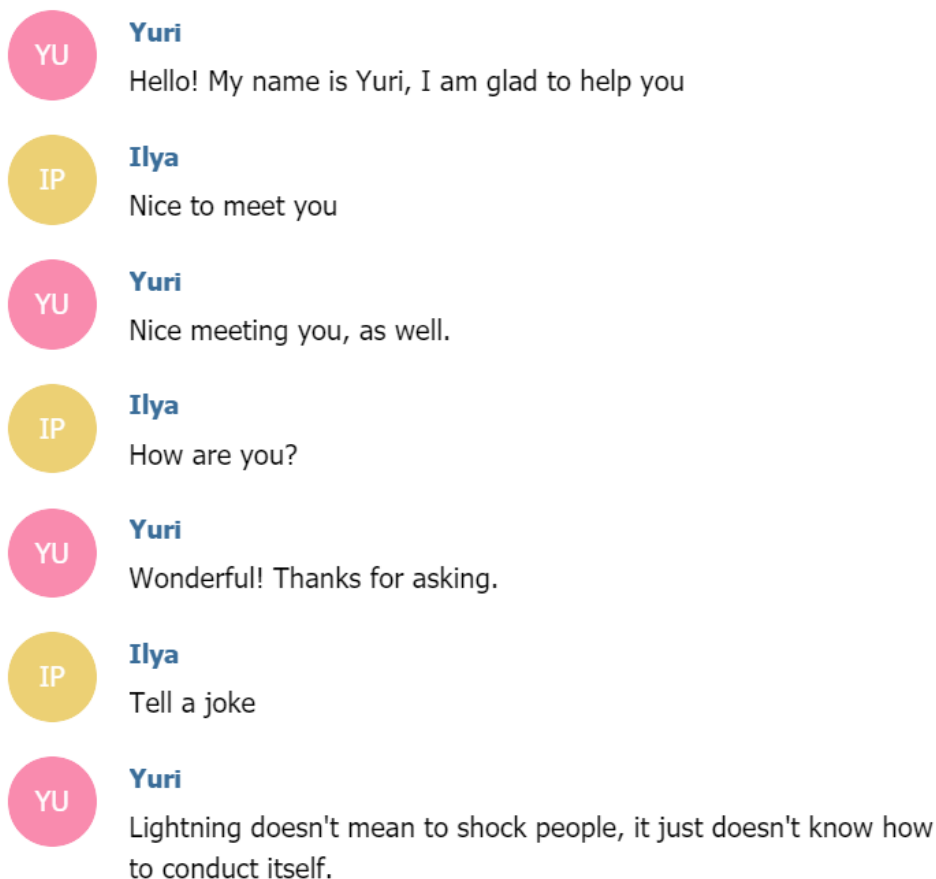


Figure 24 Working dialogue agent

If everything works, it is time to continue to the next step.

4.6 Digit recognition

Another function designed for our bot is to make it capable of recognizing the hand-written digits that were sent to it. To do that first a neural network has to be implemented.

4.6.1 Training and using a deep neural network

The state-of-the-art approach is using convolutional neural network (CNN). Convolutional networks separate the object that is fed into the network into separate parts called features. To train the network for recognizing handwritten it is possible to use well known MNIST dataset (LeCun, Y. & Cortes, C. 2010), which contains 60000 labelled, 28 by 28, grayscale images for training and 10000 labelled grayscale images for testing. The first step would be to download the dataset. There is a very easy way to do that it Keras due to a high popularity of the dataset, by using `keras.datasets.mnist`. After importing the dataset has to be split into training and testing data using the `mnist.load_data()`. Now it's time to check the shape of the sets corresponding to images:

```
[ ] print(x_train.shape)
    print(x_test.shape)

↳ (60000, 28, 28)
   (10000, 28, 28)
```

Figure 25 Checking the shape of the sets

Those numbers stand for three values in our sets:

1. Number of images (60000 for training and 10000 for testing.)
2. Number of horizontal pixels
3. Number of vertical pixels So we can see that our training test contains 60000 images of 28 by 28 pixels and the validation set contains 10000 images of 28 by 28 pixels.

For Keras to work with image data it need not only the values for each pixel, but also the information about the channel encoding of the image.

It is specified in the last value of the image set.

Number "1" stands for grayscale encoding, since we don't have that value - we have to add it by reshaping the sets.

```
x_train = x_train.reshape(60000, 28, 28, 1)
x_test = x_test.reshape(10000, 28, 28, 1)
```

Figure 26 Reshaping the data

Next step would be the normalization of our data, which means bringing our values for single pixel in range from 0-1. It is a simple operation since its already known that our pixel values range from 0 to 255, so we can just divide all of them by 255.

After normalizing the data in the set, the class vectors have to be converted to binary class matrices:

```
[ ] from keras.utils import to_categorical
```

```
[ ] y_classes = 10
```

```
y_train = to_categorical(y_train, y_classes)
y_test = to_categorical(y_test, y_classes)
```

```
[ ] print(y_test[0])
```

```
↳ [0. 0. 0. 0. 0. 0. 0. 1. 0. 0.]
```

Figure 27 Converting class vectors

That is all data preprocessing that had to be done before actually making and training the neural network.

To start building the network, it's type and layers have to be imported from Keras. I am using the TensorFlow version 1.15.0 as the backend engine for Keras.

```
[ ] from keras.models import Sequential
    from keras.layers import Conv2D, MaxPooling2D, Dropout, Flatten, Dense
    from keras.losses import categorical_crossentropy
    import tensorflow as tf
    print(tf.__version__)
    print(keras.__version__)
```

```
↳ 1.15.0
   2.2.5
```

Figure 28 Importing the model's components

Now the network has to be assembled, compiled and trained. I am using the state-of-the-art architecture for two-dimensional convolutional network from Keras documentation.

```

layer1 = Conv2D(32, kernel_size=(3, 3), activation='relu', input_shape=(28,28,1))
layer2 = Conv2D(64, (3, 3), activation='relu')
layer3 = MaxPooling2D(pool_size=(2, 2))
layer4 = Dropout(0.25)
layer5 = Flatten()
layer6 = Dense(128, activation='relu')
layer7 = Dropout(0.5)
layer8 = Dense(y_classes, activation='softmax')

model = Sequential()
model.add(layer1)
model.add(layer2)
model.add(layer3)
model.add(layer4)
model.add(layer5)
model.add(layer6)
model.add(layer7)
model.add(layer8)

model.compile(loss=categorical_crossentropy, optimizer=keras.optimizers.Adadelta(), metrics=['accuracy'])
model.fit(x_train, y_train, batch_size=128, epochs=10, verbose=1, validation_data=(x_test, y_test))

```

Figure 29 The code for building, compiling and training 2d conv network

After training, the network has to be evaluated. mine has an evaluated accuracy of 0.9928 which is a great result with 99% of correct answers on the testing dataset.

```

evaluation = model.evaluate(x_test, y_test, verbose=1)
print(f'Evaluated loss: {evaluation[0]}; Evaluated accuracy: {evaluation[1]}')

10000/10000 [=====] - 1s 53us/step
Evaluated loss: 0.02513501503208099; Evaluated accuracy: 0.9928

```

Figure 30 Model's evaluation

The network has been trained but it is still located on the Google's servers. To be able to use the model in the project locally it has to be saved and accessible on the machine. First the networks architecture has to be saved in JSON format, and its weights in hdf5 format:

```

[ ] model_json = model.to_json()
    with open("model2.json", "w") as json_file:
        json_file.write(model_json)

[ ] model.save_weights('mnist_weights.h5')

```

Figure 31 Saving the model

Now those files can be downloaded to the machine. After downloading them it's just two files, but not a model capable of making predictions. To make it back into a model we have to read the json file, compile the model from json format and load the weights into it the network:

```

mnist_file = open('models/mnist/model2.json', 'r')
loaded_mnist = mnist_file.read()
mnist_file.close()
model_mnist = model_from_json(loaded_mnist)
model_mnist.load_weights('models/mnist/mnist_weights.h5')

```

Figure 32 Rebuilding the model from files

Now the network can make predictions on our machine, but it still can't recognize the images from the real world because they have to be preprocessed.

4.6.2 Working with real images

The network for recognizing the handwritten digits was trained on the greyscale, white on black images with the resolution of 28 by 28 pixels. If the user would take a photo of a digit that he has drawn it will be colourful, dark on light image with high resolution. Our goal is to make the new images match the images that the network was trained on.

To do that the image has to be turned into grayscale and thresholding has to be performed to remove the unwanted details. Also, binary threshold inverts the image so it would be white on black. Then the image has to be resized to 28 by 28 shape to fit into the neural network, pixel values have to get normalized to allow the network to “understand” them. Then the data has to be reshaped to work with 2D network. All of those operations can be made using the opencv package, and must be saved into the function to allow the bot to preprocess new images automatically:

```

def preprocess(src):
    img = cv.imread(src)
    gray_img = cv.cvtColor(img, cv.COLOR_BGR2GRAY)
    ret, tresh = cv.threshold(gray_img, 100, 255, cv.THRESH_BINARY_INV)
    img_resized = cv.resize(tresh, (28, 28), interpolation=cv.INTER_AREA)
    img_norm = img_resized / 255
    img_norm = img_norm.reshape(28, 28, 1)
    img_processed = img_norm.reshape(1, 28, 28, 1)
    return img_processed

```

Figure 33 Image preprocessing function

4.6.3 Connecting the network to the bot

After making the digit recognition model and the function which allows it to work with the real images this functionality has to be connected to the bot. The compilation of the model

and definition of the preprocessing function is done in a separate script so it has to be imported to the bot script. The next step is to add the function which would handle the images that are sent to the bot, it has to execute the following steps: notify the user that it has received the photo, get the id of the photo from the update, download the photo, notify the user that the photo has been downloaded and the prediction process has started, predict the digit from the image, send the result back to the user. The code for the function :

```
def photo_get(update, context):  
  
    context.bot.send_message(chat_id=update.effective_chat.id, text='Image received, digit recognition mode activated')  
    photo_id = update.message.photo[-1]['file_id']  
    new_photo = context.bot.get_file(photo_id)  
    new_photo.download('pic.jpg')  
    time.sleep(3)  
    context.bot.send_message(chat_id=update.effective_chat.id, text='Image downloaded to the server')  
    time.sleep(1)  
    context.bot.send_message(chat_id=update.effective_chat.id, text='Calculating...')  
    time.sleep(1)  
    result = 'The prediction is: ' + mnist_model.do_prediction()  
    context.bot.send_message(chat_id=update.effective_chat.id, text=result)  
  
img_handler = MessageHandler(Filters.photo, photo_get)  
dispatcher.add_handler(img_handler)
```

Figure 34 Chatbot's part for predicting the digits

Now after the model is compiled, ready to work with real images and connected to the bot, we have to test it. To do that a digit must be drawn on a clear piece of paper, then the user has to take a picture of the drawing and send it to the bot:

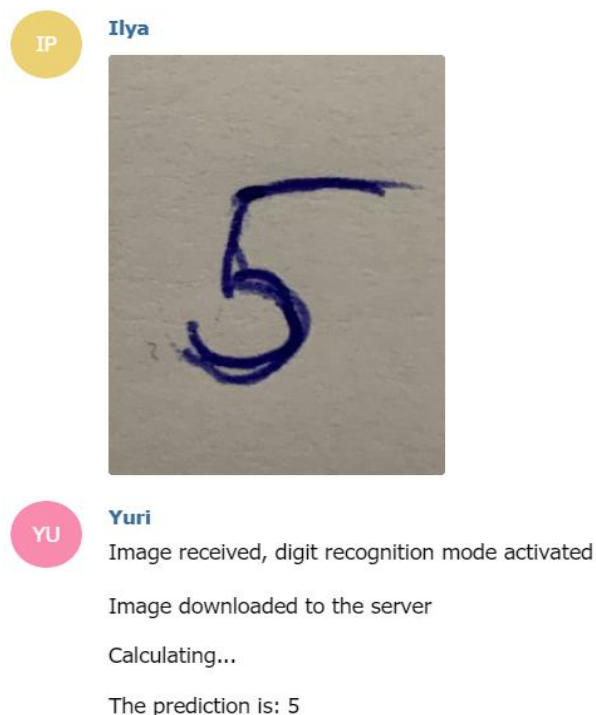


Figure 35 Bot detecting the digit

Since everything works on the local machine it is time to proceed to the next step which is hosting the chatbot.

4.7 Deploying the chatbot

For any software that is accessed by the users through the internet there has to be some machine which allows it to work. Such machine is called a server. It is needed because the software has to be accessible all the time and it might require a much bigger computing power if it has to work with many users.

The first step in deployment of any project is choosing the hosting platform. I've selected Amazon Web Services (AWS) since it's the biggest hosting platform in the world. AWS has great reputation of its technological advancements and stability, it's even used by US military. They are covering a wide territory with machines available all across the Americas, Europe and Asia.

Amazon Web Services consist of a wide range of services used for hosting and development of software. It is necessary to choose the one needed for the project. For hosting of my chatbot project the most suitable one is Amazon EC2, which stands for Elastic Compute Cloud. It's a service that provides configurable, secure and scalable machines. They are quite popular for hosting different projects. EC2 machine can run different operating systems, depending on needs of the developer. For my project I have decided to use Ubuntu 18.04 operating system since I already had some experience with it and it's one of the most popular Linux distributions.

A machine that is used by the developer is called an instance. To create an instance the developer has to set the technical configurations according to his needs. I have used a preconfigured setting for Deep Learning provided by Amazon. Also, the developer has to select what type of instance to use, which defines its processing capabilities. For my project I have selected the t2.small which has a single virtual CPU, 2GB of memory and elastic storage. While creating the instance a keypair is generated. It's a special file which has to be used every time for connecting to the server. After configuring the server, it is automatically launched and is ready to be used. To see the instructions for how to connect to the instance it is possible to right click on the instance in the AWS console and click the connect option.

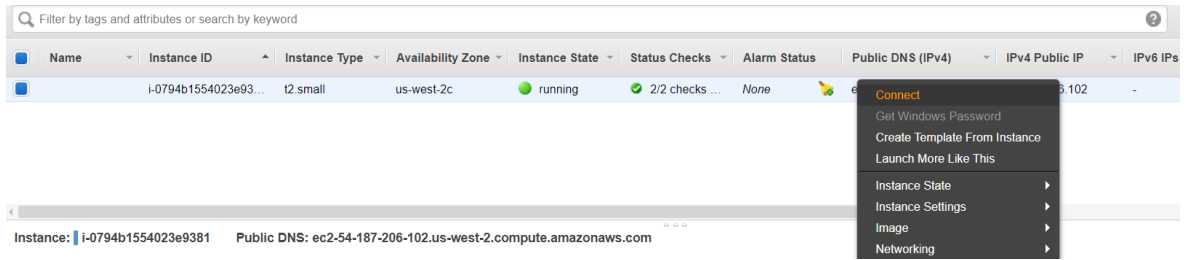


Figure 36 Instructions for connecting to the instance

I am connecting to the instance using the ssh command through bash console. The command used to connect to the server looks like this:

```
ssh -i "*keypair name*" *username*@*public DNS*
```

The machine has python 3.6.10 installed so I don't have to install python again. Next step is to install all of the dependencies used in my project in exactly the same way as during the development of chatbot. After installing all of the dependencies it's time to transfer the project to the server. It can be done through git or special software. I have used the FileZilla program for that purpose. To transfer the files to a server using FileZilla it has to be connected using the keypair and it's public DNS. After that the software is operated in explorer-like fashion by drag and dropping.

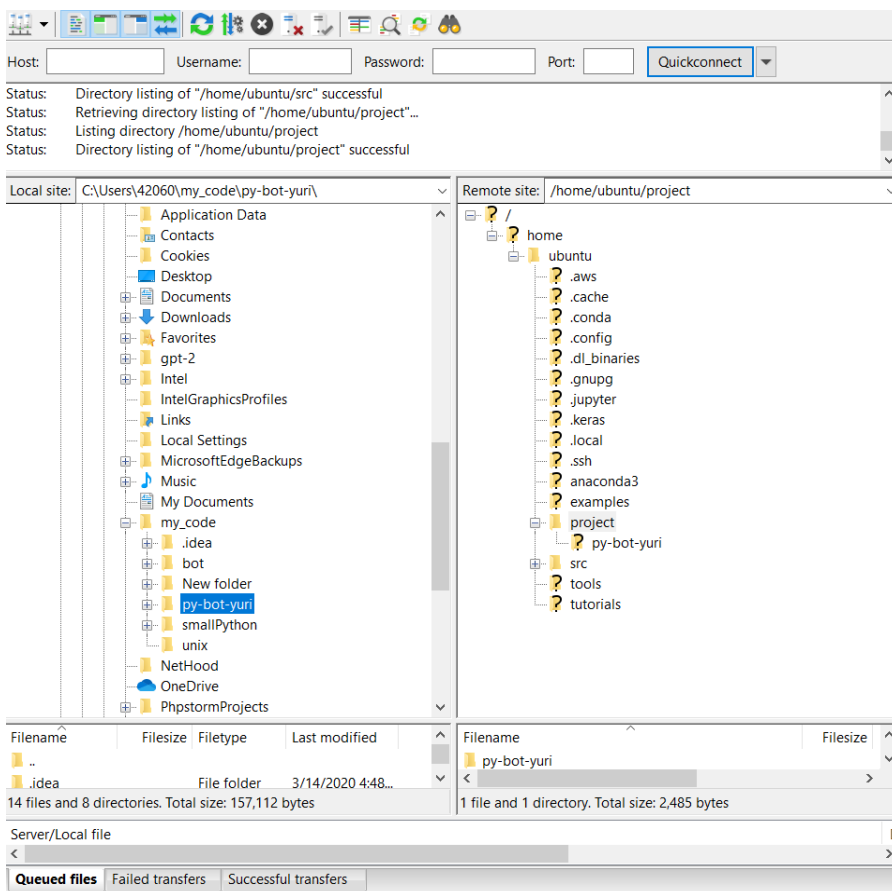


Figure 37 FileZilla interface

After transferring the files, it's time to run the script of the bot and it should work just the same as on the machine used for development.

```
ubuntu@ip-172-31-6-180:~$ cd project/py-bot-yuri
ubuntu@ip-172-31-6-180:~/project/py-bot-yuri$ python yuri-bot.py
Using TensorFlow backend.
```

Figure 38 Running the bot on server

To test if everything works a message has to be sent to the bot.

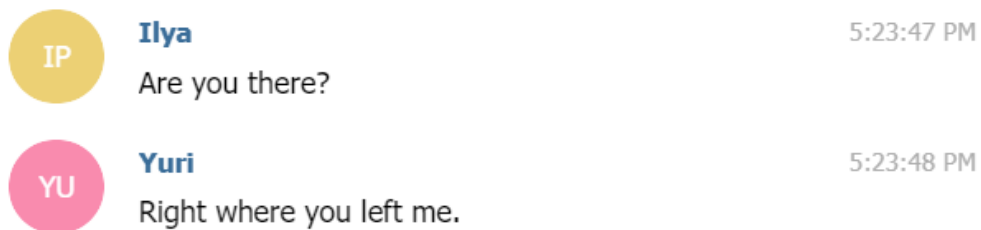


Figure 39 Testing if the bot works

The bot works which means that it was successfully developed and hosted.

5 Results

The result of this bachelor thesis is the working chatbot which can be accessed through the Telegram platform using its username: “@Yuri_theIntelligentBot”. The bot can have a basic conversation, answer questions, tell jokes and recognize hand-written digits from a photo.

The source code for the bot is attached to the thesis, it consists of three python scripts and two files used for compilation of deep learning model. The first script “yuri_bot.py” is responsible for the main functionality together with the message handler. The second script “weather.py” is used to create a function which would make the bot send the weather in Prague after it receives the “/temp” command. The last script mnist_model is supposed to compile the deep learning model and define the functions used for image preprocessing and making the prediction.

For development of this chatbot a lot of different tools were used. It is powered by Python programming language, the Dialogflow API was used to develop the conversational agent using transfer learning, fine-tuning and unsupervised learning. The python-telegram-bot wrapper allows it to work with Telegram’s bot API. The deep learning model used to recognize hand-written digits was trained on MNIST dataset in a supervised learning approach using TensorFlow and Keras on Google Colab platform. OpenCV package is used to preprocess the photos to feed them into the neural network. There weren’t many problems while using these tools for development of the bot thanks to a wide range of useful guides and books that are available to help the developers.

After developing, the chatbot system was hosted on a remote server machine using Amazon Web Services as a hosting platform, which allows it to be constantly available. The platform is famous for its quality so there weren’t any problems with configuring and using the serving machine.

6 Conclusion

This thesis was meant to describe the key stages of developing a chatbot. It was supposed to show that nowadays the developers are able to take advantage of various approaches and technologies designed to make the development of the chatbot easier. The literature review has explained the basics of what a chatbot is, why are they needed, how they impact the industries and what are they capable of. The process of designing a chatbot was presented with simple examples. The next part was dedicated to explaining the fundamental techniques of natural language project that are needed to make the chatbot work with human language. The last chapter of literature review is meant to show various tools that can be used for developing chatbots and other types of software.

To show how to use the mentioned technologies and tools, the practical part was focused on development and deployment of a standalone chatbot system which is capable of working with natural language and image data by taking advantage of deep machine learning techniques. The contents of the practical part were meant to show each stage that the development team has to complete in order to create the chatbot system. It consists of the parts dedicated to designing the chatbot, configuring the development workspace, creating dialogue agents, enabling the basic chatbot functionality, creating and using a deep learning model, and hosting the chatbot system.

Today is the day to build your first chatbot and it never was a better time, new technologies allow developers to create stable and scalable systems, the demand is higher than ever and is rising constantly. Also there are a lot of different informational resources such as blogs, guides, courses, articles and books, which are able to demystify any technical, economical, artistic or any other field there if you are really interested in it and enjoy working in the area of **your** choice.

7 References

- Aurélien, Géron. 2017.** *Hands-On Machine Learning with Scikit-Learn*. Sebastopol, CA : O'Reilly Media, Inc., 2017. ISBN: 978-1-491-96229-9.
- Bhagwat, Vyas Ajay. 2018.** *Deep Learning for Chatbots*. San Jose, CA : SJSU ScholarWorks, 2018. DOI: <https://doi.org/10.31979/etd.9hrt-u93z>.
- Bressert, Eli. 2013.** *SciPy and NumPy*. Sebastopol, CA : O'Reilly Media, Inc., 2013. ISBN: 978-1-449-30546-8.
- Devert, Alexandre. 2014.** *matplotlib Plotting Cookbook*. Birmingham, UK : Packt Publishing Ltd., 2014. ISBN: 978-1-84951-326-5.
- McKinney, Wes. 2012.** *Python for Data Analysis*. Sebastopol, CA : O'Reilly Media, Inc., 2012. ISBN: 978-1-449-31979-3.
- Sumit, Raj. 2019.** *Building Chatbots with Python*. Bangalore : Apress, 2019. DOI: <https://doi.org/10.1007/978-1-4842-4096-0>.
- LeCun, Y., Cortes, C. and Burges, C.J., 2010.** *MNIST handwritten digit database*.

8 Appendix

Files attached to the thesis:

- yuri_bot.py** – main script of the bot with the dispatcher, updater, and handler functions;
- mnist_model.py** – script responsible for compilation and use of digit recognition model;
- degrees.py** - a script with a function that gets the weather in Prague;
- models** – a folder with files for compilation of deep learning models.