VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES

Eyno Perevodchikov

NATURAL LANGUAGE PROCESSING AND CHAT BOT IMPLEMENTATION

Code name – NiBot

Technology 2019

VAASAN AMMATTIKORKEAKOULU
UNIVERSITY OF APPLIED SCIENCES
Tieto- ja viestintätekniikka

ABSTRACT

Author	Eyno Perevodchikov
Title	Natural Language Processing and chat-bot implementation
Year	2019
Language	English
Pages	37 + 1 Appendix
Name of Supervisor	Timo Kankaanpää

In recent years Artificial Intelligence (AI) research and development has become more and more popular, and solutions which use AI are becoming more ubiquitous. One of the problems AI deals with is Natural Language Processing (NLP), which is one of the most demanding problems as it deals with human speech. The aim of this thesis was to study NLP and Machine Learning (ML), and the implementation of a chat-bot, which can automate some tasks executed on a machine and provide an easy to use user interface.

In this implementation statistical algorithms were used for text analysis. The implementation is done using python, python's libraries scikit-learn and Flask, and Telegram Bot API for communication with a bot.

The results showed that even simple chat-bots can be extremely useful. The implementation of this bot can run any type of scripts on the machine it is running and find the required information from its knowledge database.

Keywords NLP, AI, chat bot

CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF APPENDICES

LIST OF ABBREVIATIONS AND DEFINITIONS

AI	Artificial Intelligence
API	Application Programming Interface
DDOS	Denial of service attack
EU	European Union
FAQ	Frequently Asked Questions
HTTP	Hypertext Transfer Protocol
ML	Machine Learning, subfield of artificial intelligence
NLP	Natural Language Processing
REST	Representational State Transfer
URL	Uniform Resource Locator

Corpora	plural of corpus
Corpus	large and structured set of texts
n-gram	sequence of items from a sample of text or speech
tweet	entry posted on a blog board in Twitter

1 INTRODUCTION

Computer programs and applications that leverage natural language processing (NLP) techniques to understand textual or audio data are becoming ubiquitous in our world. We became accustomed to such programs not realizing that they actually there, behind the scenes they filter out spam e-mails, saving us tons of traffic. We also use them in search engines to send us exactly to information we wanted, or virtual assistants, which are always there and ready to respond to our requests.

With time, NLP interfaces will become more common, replacing current point and click user interfaces. Natural language is of the most useful but unconquered data forms. It has the ability to make data products incredibly useful and essential in our lives.

In this thesis I developed a virtual assistant, which can be accessed through several interfaces such as a terminal emulator, a chat application and as a REST request to a server. This personal assistant is able to execute scripts which are mapped to words to words with a predefined prefix, search for an answer on user's question in its knowledge base, and search for the personal information of a requested person.

2 SUCCESS STORIES

2.1 Virtual Assistant

A virtual assistant is an agent that can perform tasks or services requested by the user. The requests can be verbal or typed. Some virtual assistants can generate human speech.

In recent years, capabilities and usage of virtual assistants has been increasing. Everyone has a virtual assistant in their phone, with which they can interact vocally. Examples are Siri on Apple products, Google Assistant on Android and Cortana on Windows. The tasks can be such as set up timer, send a message to someone or search for information on a search engine like Google.

Moreover, virtual assistants have moved from smartphone devices to special home devices. Amazon Alexa on Amazon Echo popularized such devices, then Apple made their home device called HomePod, and as Android is an open-source operation system, companies started making their home devices with Google Assistant.

This type of bot is helpful in achieving simple tasks like setting an alarm or a timer, opening apps, sending a message or searching for information.

2.2 Customer service chat bot

Chat bots are good for customer service of small businesses, and more and more companies choose to use them. Chat bots provide answers on customer queries in short period of time, they can handle high amount of queries simultaneously and good chat bot in closed environment like online shop produces less errors than people.

Customer service type of a chat bot can be used with chat applications, which are accessible from a smartphone, or can be integrated directly to a website.

Popular examples are ChatBot, Claire.AI, Twyla.

Such chat bots save time and money for users and businesses. You do not need to wait for a time when a customer service agent will be available to answer your request, and a company does not need to hire a high amount of people for customer service.

3 THEORY

In this chapter I will briefly describe problems which NLP techniques try to solve and present the usual methods used for it.

3.1 Language

There are three most important linguistic elements of language in NLP: semantics, syntax, and morphology. They allow us to extract data or meaning from even a simple text string.

3.1.1 Semantics

Semantics is the meaning of a text. Extraction of semantics is the most difficult task. Semantic analysis is a method of generation data structures to which logical reasoning can be applied.

For example a sentence “He picked up a car from a car shop.” has a structure as follows: subject – verb – subject – object. This structure can be used as a template of relationships between words or entities in a sentence. A text like “John gave a book to Jane.” have the same exact structure and follows the template.

> “Semantics is the linguistic and philosophical study of meaning, in language, programming languages, formal logics, and semiotics. It is concerned with the relationship between signifiers—like words, phrases, signs, and symbols—and what they stand for in reality, their denotation.” [1]

3.1.2 Syntax

Syntax refers to rules of sentence formation defined by grammar. Sentences have more meaning than just words and encode more information, therefore sentence can be considered as the smallest logical unit of language.

Syntactic analysis is designed to show the meaningful relationship of words, usually by separating a sentence into meaningful parts or separating into words and showing the relationship of them in a tree structure.

> “In linguistics, syntax is the set of rules, principles, and processes that govern the structure of sentences (sentence structure) in a given language, usu-

> ally including word order. The term syntax is also used to refer to the study of such principles and processes. The goal of many syntacticians is to discover the syntactic rules common to all languages.” [2]

3.1.3 Morphology

Morphology in linguistic is a form of words, their structure. Structure of words helps us to identify forms such as plurality, tense, etc.

- dog – dogs
- do – did

Most languages have many exceptions and special cases which makes morphological analysis challenging. Sometimes we can just add ending to a word to get a different form, for example dog and dogs. Sometimes we get complete transition of word, for example mouse and mice. Sometimes there are no change at all, fish is both singular and plural, and also a verb.

The goal of morphology is detection and understanding of classes of words: singular, plural or proper noun. The same can be applied to tense of verb and other types of words.

> “In linguistics, morphology is the study of words, how they are formed, and their relationship to other words in the same language. It analyzes the structure of words and parts of words, such as stems, root words, prefixes, and suffixes. Morphology also looks at parts of speech, intonation and stress, and the ways context can change a word's pronunciation and meaning. Morphology differs from morphological typology, which is the classification of languages based on their use of words, and lexicology, which is the study of words and how they make up a language's vocabulary.” [3]

3.2 Natural language analysis

3.2.1 Stemming

Stemming is the process of a word form reduction to its stem. This process allows us to remove information, which we do not need for our analysis. For example, for classifica-

tion of text we can reduce such words as hack, hacking and hacker to a single form hack and count its occurrences, which could correlate with the technical type of texts.

> “In linguistic morphology and information retrieval, stemming is the process of reducing inflected (or sometimes derived) words to their word stem, base or root form—generally a written word form. The stem need not be identical to the morphological root of the word; it is usually sufficient that related words map to the same stem, even if this stem is not in itself a valid root. Algorithms for stemming have been studied in computer science since the 1960s. Many search engines treat words with the same stem as synonyms as a kind of query expansion, a process called conflation.
>
> A computer program or subroutine that stems word may be called a stemming program, stemming algorithm, or stemmer.” [4]

3.2.1 Lemmatization

Lemmatization is process of matching words into groups by their meaning. We can group words as pond, lake, reservoir, lagoon as synonyms with meaning of a large area of water surrounded by land. This way we can identify phrases like “A boy swam in a pond.” and “A boy swam in a lake.” as equal and identical.

> “Lemmatisation (or lemmatization) in linguistics is the process of grouping together the inflected forms of a word so they can be analysed as a single item, identified by the word's lemma, or dictionary form.
>
> In computational linguistics, lemmatisation is the algorithmic process of determining the lemma of a word based on its intended meaning. Unlike stemming, lemmatisation depends on correctly identifying the intended part of speech and meaning of a word in a sentence, as well as within the larger context surrounding that sentence, such as neighboring sentences or even an entire document. As a result, developing efficient lemmatisation algorithms is an open area of research.” [5]

3.2.1 Bag of Words

Bag of words model is model which considers co-occurrence of words in text.

For example bag of words model of poem

To the swinging and the ringing

of the bells, bells, bells-

Of the bells, bells, bells, bells

Bells, bells, bells-

To the rhyming and the chiming of the bells! [6]

looks like this: to, the, swinging, and, ringing, of, bells, rhyming, chiming.

Unfortunately we cannot see a lot of meaning in this kind of model, for example in phrases “John gave a book to Jane” and “Jane, book me a seat in theatre.” the word book has a different meaning. To add meaning to words, instead of dividing text by co-occurrence of words, text can be divided by co-occurrence of n-grams. A model in which text divided by n-grams would be called “Bag of n-grams”. The letter n in n-gram, stands for a natural number: 1, 2, 3 and so on.

If we divide our example phase “Jane, book me a seat in theatre.” into 2-grams, we would get: (Jane book), (book me), (me a), (a seat), (seat in), (in theatre); even with 2-grams we can see more meaning in our “bag”. If we would divide the text into 3-grams, we would get: (Jane book me), (book me a), (me a seat), (a seat in), (seat in theatre); this n-gram also has a meaning for us.

Some of the n-grams are nonsensical and with an increase of n the likelihood of n-gram appearing in other texts is decreasing.

To vectorize the corpus bag of words model commonly used and represent every document as a vector whose length is equal to the vocabulary of the corpus.

3.2.2 Vectorization

Vectorization is process of converting data into numeric vector representation. Vectorizers use bag of words model, but it also can be used with n-gram model.

The next simple corpus containing single sentences used as an example:

1. This is the first sentence.
2. This is the second sentence.
3. This is nor first nor second sentence.

Vocabulary of this corpus is as follows: this, is, the, first, second, sentence, nor. Vocabulary contains all words from corpus and analysis of each corpus entry shows presence of every word from corpus in this entry.

3.2.3 Frequency vectorizer

A frequency vectorizer is a simple vectorizer which just counts occurrences of words in a text string. After running our example corpus (section **3.2.2**) through this vectorizer, we will get the result shown in Table 1.

Table 1. Frequency vectorizer analysis

Vocabulary	this	is	the	first	second	sentence	nor
Entry 1	1	1	1	1	0	1	0
Entry 2	1	1	1	0	1	1	0
Entry 3	1	1	0	1	1	1	2

With this type of vectorization it should be considered that the most frequent words are not the most descriptive, for example words like “the”, “and”, “or” appear frequently but do not have important information for analysis of the text type.

3.2.4 One-Hot vectorizer

As frequency vectorizer suffers from frequent, but not informative words, the one-hot vectorizer can be useful for reducing influence of those uninformative words.

One-hot vectorizer is a vectorizer which counts the appearance of a word only once. This way every word has the same influence.

Table 2. One-hot encoding analysis

Vocabulary	this	is	the	first	second	sentence	nor
Entry 1	1	1	1	1	0	1	0
Entry 2	1	1	1	0	1	1	0
Entry 3	1	1	0	1	1	1	1

This type of a vectorizer is useful for the analysis of small amounts of text like sentences, tweets or small messages, i.e. text which does not have a lot of repeatings. As every word is equal in importance, text normalization is recommended.

3.2.5 Term Frequency-Inverse Document Frequency or TFIDF

Another way of improving the accuracy of word significance is to consider the relative frequency of words in a document against the frequency of the same word in other documents. The idea is that the meaning is most likely encoded in more rare words from a document. For example, in technology articles terms such as “cryptography”, “spoofing”, “ddos” are more significant in articles related to security, while words like “type”, “enter”, “look” appear more frequently and are less important.

The relevance or value of a word is calculated with consideration to the whole corpus. As seen in Table 3, the word “nor” has really high value in the third entry as it appears only there and several times.

It should be taken into consideration that with this vectorizer moderately frequent words can be not descriptive of the text topic.

Table 3. TFIDF encodinog analysis

Vocabulary	this	is	the	first	second	sentence	nor
Entry 1	0.39	0.40	0.51	0.51	0	0.40	0
Entry 2	0.39	0.40	0.51	0	0.51	0.40	0
Entry 3	0.24	0.24	0	0.31	0.31	0.24	0.80

3.2.3 Classification

Classifier algorithms are used for classification of a text. The task of a classifier is the determination to which category the text belongs.

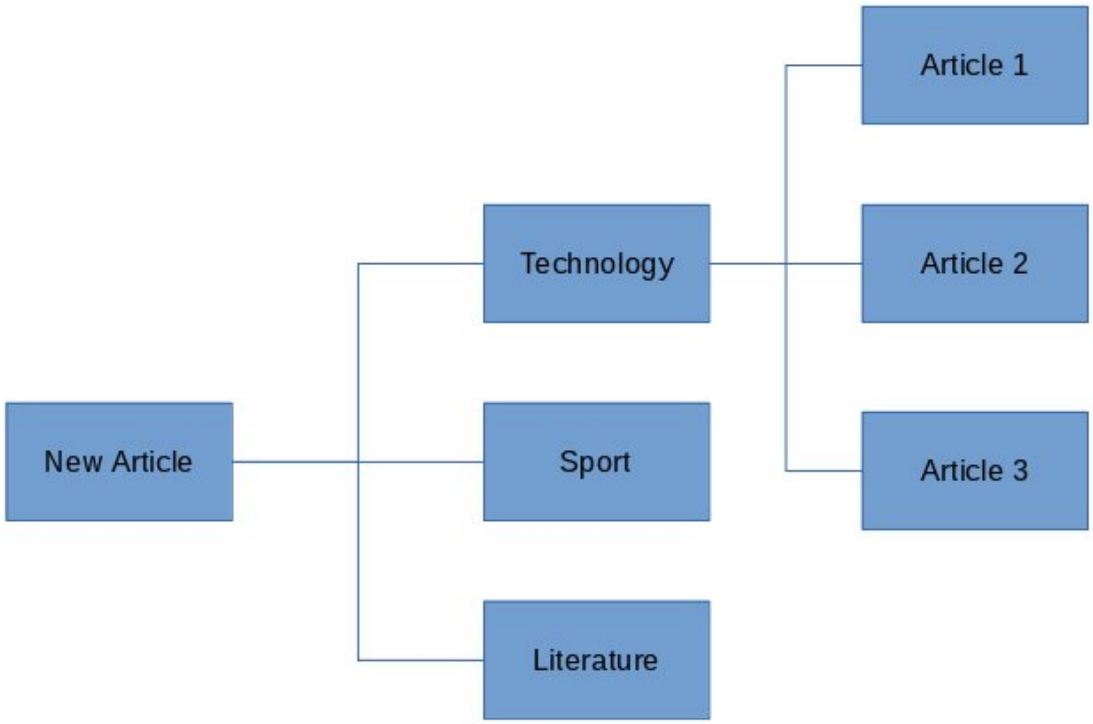

Figure 1. Classification scheme

In sections **3.2.4** and **3.2.5** are descriptions of two classifiers used in this thesis. Those classifier are used for **F6** requirement in section **4.1**.

3.2.4 Naive Bayes

> “In machine learning, naive Bayes classifiers are a family of simple "probabilistic classifiers" based on applying Bayes' theorem with strong (naive) independence assumptions between the features.
>
> Naive Bayes has been studied extensively since the 1960s. It was introduced under a different name into the text retrieval community in the early 1960s, and remains a popular (baseline) method for text categorization, the problem of judging documents as belonging to one category or the other (such as

spam or legitimate, sports or politics, etc.) with word frequencies as the features. With appropriate pre-processing, it is competitive in this domain with more advanced methods including support vector machines. It also finds application in automatic medical diagnosis.

Naive Bayes classifiers are highly scalable, requiring a number of parameters linear in the number of variables (features/predictors) in a learning problem. Maximum-likelihood training can be done by evaluating a closed-form expression, which takes linear time, rather than by expensive iterative approximation as used for many other types of classifiers.

Naive Bayes is a simple technique for constructing classifiers: models that assign class labels to problem instances, represented as vectors of feature values, where the class labels are drawn from some finite set. There is not a single algorithm for training such classifiers, but a family of algorithms based on a common principle: all naive Bayes classifiers assume that the value of a particular feature is independent of the value of any other feature, given the class variable. For example, a fruit may be considered to be an apple if it is red, round, and about 10 cm in diameter. A naive Bayes classifier considers each of these features to contribute independently to the probability that this fruit is an apple, regardless of any possible correlations between the color, roundness, and diameter features.

For some types of probability models, naive Bayes classifiers can be trained very efficiently in a supervised learning setting. In many practical applications, parameter estimation for naive Bayes models uses the method of maximum likelihood; in other words, one can work with the naive Bayes model without accepting Bayesian probability or using any Bayesian methods.

Despite their naive design and apparently oversimplified assumptions, naive Bayes classifiers have worked quite well in many complex real-world situations. In 2004, an analysis of the Bayesian classification problem showed that there are sound theoretical reasons for the apparently implausible efficacy of naive Bayes classifiers. Still, a comprehensive comparison with other classification algorithms in 2006 showed that Bayes classification is outperformed by other approaches, such as boosted trees or random forests.

An advantage of naive Bayes is that it only requires a small number of training data to estimate the parameters necessary for classification." [7]

As this classifier requires a small amount of training data, it can be perfect classifier for one of tasks of this thesis.

3.2.5 KNeighbors

"In pattern recognition, the *k*-nearest neighbors algorithm (*k*-NN) is a non-parametric method used for classification and regression. In both cases, the input consists of the *k* closest training examples in the feature space. The output depends on whether *k*-NN is used for classification or regression:

- In *k-NN classification*, the output is a class membership. An object is classified by a plurality vote of its neighbours, with the object being assigned to the class most common among its *k* nearest neighbours (*k* is a positive integer, typically small). If *k* =1, then the object is simply assigned to the class of that single nearest neighbour.

- In *k-NN regression*, the output is the property value for the object. This value is the average of the values of its *k* nearest neighbours.

k-NN is a type of instance-based learning, or lazy learning, where the function is only approximated locally and all computation is deferred until classification. The *k*-NN algorithm is among the simplest of all machine learning algorithms.

Both for classification and regression, a useful technique can be used to assign weight to the contributions of the neighbours, so that the nearer neighbours contribute more to the average than the more distant ones. For example, a common weighting scheme consists in giving each neighbour a weight of 1/*d*, where *d* is the distance to the neighbour.

The neighbours are taken from a set of objects for which the class (for *k*-NN classification) or the object property value (for *k*-NN regression) is known. This can be thought of as the training set for the algorithm, though no explicit training step is required.

> A peculiarity of the *k*-NN algorithm is that it is sensitive to the local structure of the data.” [8]

This type of classifier is also a potential winner as it is sensitive to structure of data.

4 IMPLEMENTATION

This bot will be able to execute scripts directly on a machine and transfer the results back to the user, for example user could pass a mathematical equation to a script which would then solve it. The bot will reduce time consumption spent on a search for information from a local database. An example is search for a phone number of a specific person or a search for an answer on a question from a database of known questions. With implementation of all functionalities, this bot can be used by specialist and customers who do not have technological knowledge. The bot also can be used as a helper for customers who can ask FAQ questions from the bot.

4.1 Requirements

Table 4. Functional requirements

Reference	**Description**	**Priority** **(1 is highest)**
F1	The user can make a request to the Bot.	1
F2	The bot can identify a command by its text representation and then execute it. Example: Command `!time` should return current time.	1
F3	The User/Administrator can add a new commands to the chat Bot library.	1
F4	The bot can identify a person or their name from the text. Example: John Doe	2
F5	The bot can find person’s information, which user requested. Example: phone number, address	2
F6	The bot can find answer on a user’s question from its knowledge database.	3

	Example: (User) How much does delivery costs? (Bot) We provide free delivery to any country inside of EU.	

Table 5. Interfaces

Reference	**Description**
I1	Terminal emulator
I2	Internet Browser
I3	Chat Application (Telegram)

Table 6. Non-Functional Characteristics

Characteristic	**Description**
Usability	It should be fairly easy to make a requests to the bot and make changes to it.
Response time	The bot should be able to respond to a user in no longer than 5 seconds.
Safety	Only authenticated users can make requests to the bot.

4.2 Tools Used

This project was implemented purely in Python 3 and I wanted to use as small amount of dependencies as possible for this project. For the bot itself I decided to use ***scikit-learn***, ***Python Flask*** as a Web-Framework and ***Telegram Bot API*** for chat application integration.

4.2.1 Scikit-learn

Scikit-learn is machine learning library for Python. Its main purpose is mining and analysis of data. It is built on **NumPy**, **SciPy** and **matplotlib** Python libraries. The library contains **classification**, **regression**, **clustering** and **preprocessing** algorithms among others.

Scikit-learn is a lightweight library which has all minimal needed functionalities required for accomplishing the targets of the thesis.

The main purposes of using this library are vectorization, classification and model management. An example code is displayed in **Figure 2**.

Additional libraries like **NLTK** and **Gensim** could have been used for such tasks as text normalization and part of speech detection, but for initial implementation it was decided to keep it as simple and as independent as possible.

4.2.2 Flask

Flask is Python's micro web framework. This web framework does not use any additional libraries, except for the standard libraries of Python. Flask allows the user to rapidly develop and deploy a web service which uses **REST** like **URL**s.

An alternative to ***Flask*** could be ***Django***, but as ***Flask*** is small and a server API can be developed more rapidly in it, ***Flask*** was chosen. A minimal ***Flask*** application is displayed in **Figure 3.**

4.2.3 Requests

Requests is a python library for HTTP request handling. It allows easy sending and retrieving of information through any type of a HTTP request.

“Requests is the only *Non-GMO* HTTP library for Python, safe for human consumption.

Requests allows you to send *organic, grass-fed* HTTP/1.1 requests, without the need for manual labor. There’s no need to manually add query strings to your URLs, or to form-encode your POST data. Keep-alive and HTTP connection pooling are 100% automatic, thanks to urllib3.” [9]

```
>>> from sklearn import svm
>>> from sklearn.datasets import samples_generator
>>> from sklearn.feature_selection import SelectKBest
>>> from sklearn.feature_selection import f_regression
>>> from sklearn.pipeline import Pipeline
>>> # generate some data to play with
>>> X, y = samples_generator.make_classification(
...     n_informative=5, n_redundant=0, random_state=42)
>>> # ANOVA SVM-C
>>> anova_filter = SelectKBest(f_regression, k=5)
>>> clf = svm.SVC(kernel='linear')
>>> anova_svm = Pipeline([('anova', anova_filter), ('svc', clf)])
>>> # You can set the parameters using the names issued
>>> # For instance, fit using a k of 10 in the SelectKBest
>>> # and a parameter 'C' of the svm
>>> anova_svm.set_params(anova__k=10, svc__C=.1).fit(X, y)
...
Pipeline(memory=None,
         steps=[('anova', SelectKBest(...)),
                ('svc', SVC(...))])
>>> prediction = anova_svm.predict(X)
>>> anova_svm.score(X, y)
0.83
>>> # getting the selected features chosen by anova_filter
>>> anova_svm.named_steps['anova'].get_support()
...
array([False, False,  True,  True, False, False, True,  True, False,
       True,  False,  True,  True, False, True,  False, True, True,
       False, False])
>>> # Another way to get selected features chosen by anova_filter
>>> anova_svm.named_steps.anova.get_support()
...
array([False, False,  True,  True, False, False, True,  True, False,
       True,  False,  True,  True, False, True,  False, True, True,
       False, False])
```

Figure 2. scikit-learn Pipeline code example [10]

A minimal Flask application looks something like this:

```
from flask import Flask
app = Flask(__name__)

@app.route('/')
def hello_world():
    return 'Hello, World!'
```

Figure 3. minimal Flask application [11]

Behold, the power of Requests:

```
>>> r = requests.get('https://api.github.com/user', auth=('user', 'pass'))
>>> r.status_code
200
>>> r.headers['content-type']
'application/json; charset=utf8'
>>> r.encoding
'utf-8'
>>> r.text
u'{"type":"User"...'
>>> r.json()
{u'private_gists': 419, u'total_private_repos': 77, ...}
```

Figure 4. requests example

4.2.4 Telegram Bot API

Telegram Bot API is an HTTP based interface for accessing telegram bots. Each bot has a personal authorization token which can be used inside of the URL or in a packet header. A bot can be created by talking with **@botfather** bot. This requires an existing Telegram account.

> "All queries to the Telegram Bot API must be served over HTTPS and need to be presented in this form:
>
> `https://api.telegram.org/bot<token>/METHOD_NAME.`

Like this for example:

```
https://api.telegram.org/bot123456:ABC-DE-
F1234ghIkl-zyx57W2v1u123ew11/getMe
```

We support **GET** and **POST** HTTP methods. We support four ways of passing parameters in Bot API requests:

- URL query string
- application/x-www-form-urlencoded
- application/json (except for uploading files)
- multipart/form-data (use to upload files)" [12]

4.2.5 Pytest

Pytest is python library for testing. ***Pytest*** makes it easy and fast to write small tests. It has such features as:

- Autodiscovery – discover automatically test suits and test functions
- Detailed information about failed tests
- Can run python's unittest tests

An example of a simple test:

```
# content of test_sample.py
def inc(x):
    return x + 1


def test_answer():
    assert inc(3) == 5
```

Figure 5. pytest simple example [13]

4.2.6 Pipenv

Pipenv is a package manager like **node**'s **npm**. It uses ***pip*** and ***virtualenv*** modules. ***Pip*** itself manages packages, but it's better to install libraries into a separate environment and for this purposes ***virtualenv*** is used. ***Pipenv*** uses ***pip*** and ***virtualenv*** separately for every project, this way it is possible to install and initiate a virtual environment easily and without any headache.

Pipenv create two files for dependency management inside a project folder: ***Pipfile*** and ***Pipfile.lock***. To install dependencies on other machine, you simply need to run command

```
>> pipenv install
```

which will read dependencies from ***Pipfile***. And to activate virtual environment you need to run next command

```
>> pipenv shell
```

which will spawn a shell inside the virtual environment.

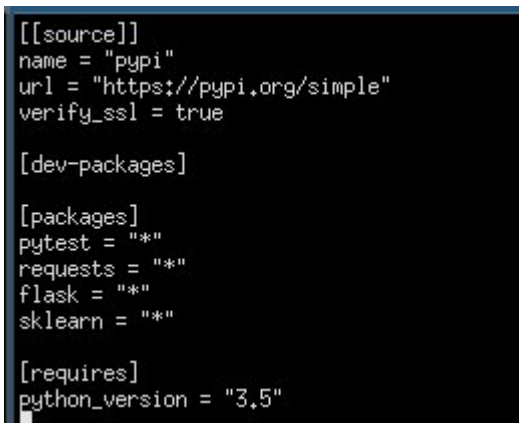

Figure 6. Pipfile example

4.2 Architecture

The root of the project contains (**Figure 7**) program runners, bot package, directory with configurations, directory with tests implemented with ***pytest***, and corpus directory.

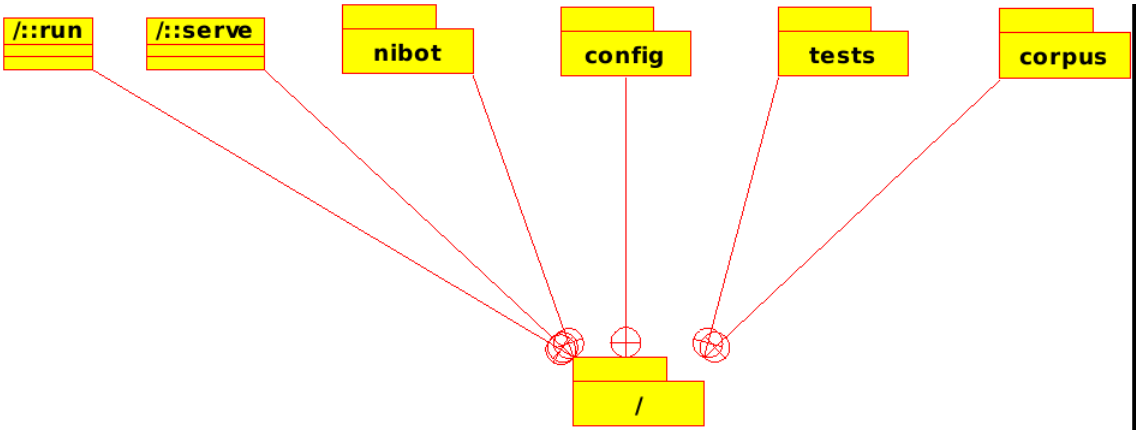

Figure 7. Root package diagram

The bot's package (**Figure 8**) contains modules and packages with the functionality of the bot. ***agent*** module has implementations of the bots with different communication interfaces: terminal emulator, http, telegram. ***model*** module responsible for creation and management of current train-model and dialogues. ***command*** package contains command modules (section **4.3**). ***chat*** module contains ***Chat*** class and dialogue classes, which responsible for parsing, interpreting and responding on user's request. ***text*** module contains classes responsible for analysis of a text. And finally ***telegram*** module contains classes responsible for communication with telegram's bot.

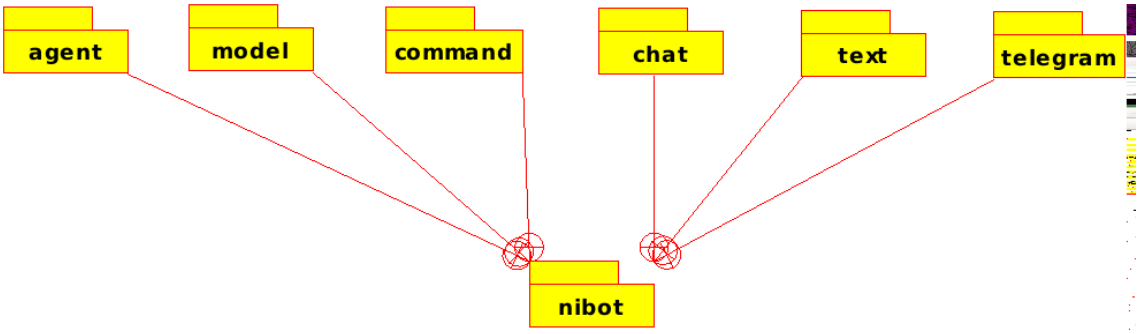

Figure 8. nibot package diagram

4.3 Commands

command package (**Figure 8**) contains modules with commands. ***api*** module contains help functions such as *is_command*, *get_command*, etc and responsible for mapping of commands to their text aliases.

Each command should extend ***Command*** class from ***base*** module (**Figure 9)** and implement ***execute*** function, which is the main function of any command and runs whenever

you want to execute a command. Commands can use any library and main function can run any other function. Agent have to send reference to itself as a first argument of ***args** and any parameters are sent in ***kwargs** under key **params**. Parameters are whatever user writes after command's alias and separated by white-space, for example "!roll 12" is command which rolls a dice with 12 sides, where 12 is parameter. A user can send any amount of parameters with a command, for example if user can not decide on a colour of something, they can use "!choose red green blue", where red green and blue are parameters of a command. And, of course, a user can execute a command without any parameters: "!name", which will return the name of a bot.

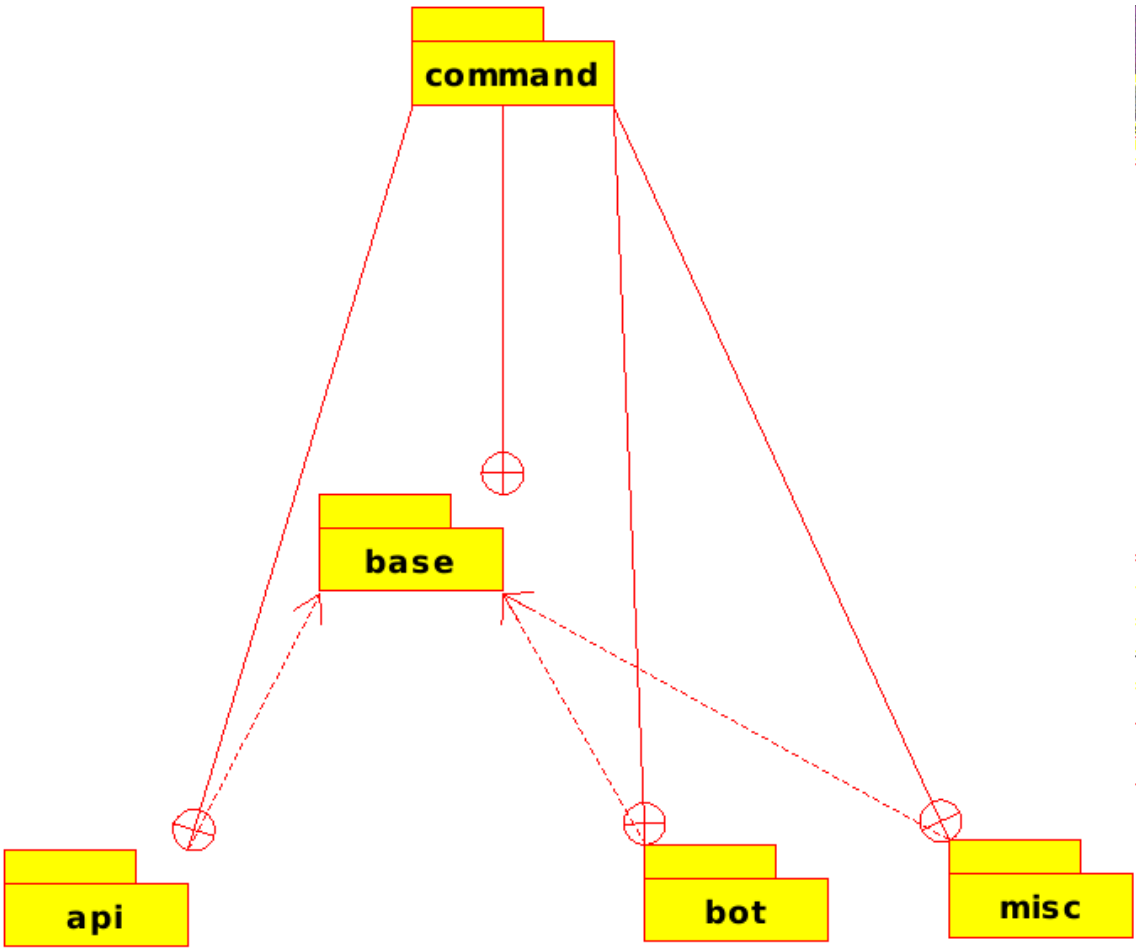

Figure 9. Command package diagram

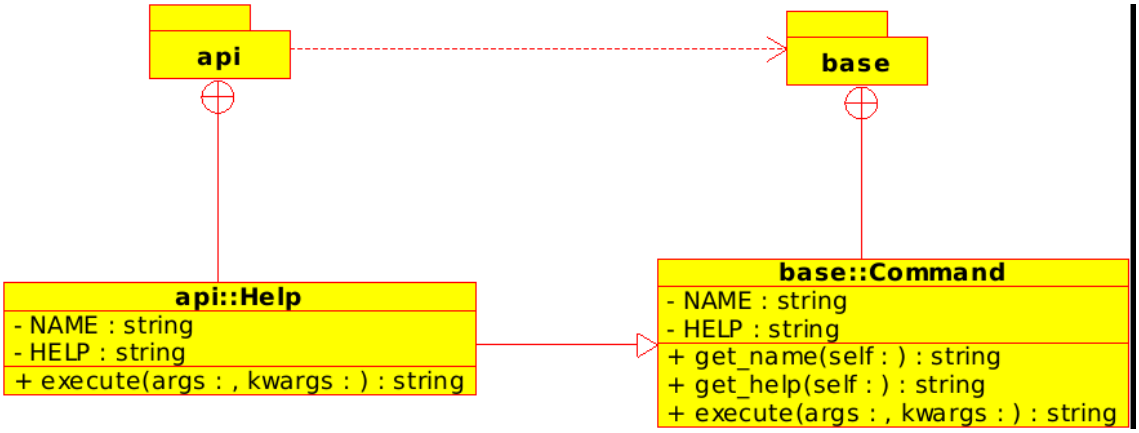

Figure 10. Command's API class diagram

4.4 Person information search

A search for a person's information is accomplished by pattern comparison of user's request. To be able to transform a request into a pattern, we need to identify entities from this request. For example question "What is John Doe's phone number?" can be transformed into a pattern "question – person – item". This kind of transformation accomplished by text comparison. We have a text document in corpus with all kind of person's names and by comparing words from sentence with names we can identify a person. Identification of an item requested is accomplished in the same way, but this time we are comparing with data fields of known people and if we have information only of person's phone number, only this item will be identified in question.

4.5 Answer search

The bot can answer on a questions by searching for text files for an answer by comparing request with questions from text files. Firstly, model is trained by some data and for this two different FAQ pages used: the first is from bookdepository.com [14] and the second is from wordpress.com [15]. Text files are in a special **json** (**Figure 11**) format with objects which have only two keys: "key" and "value". As the files are special, a unique extension were given to this format, which I named **jsonkv**, because it only has keys and values. The file contains the question under "key" and the answer under "value".

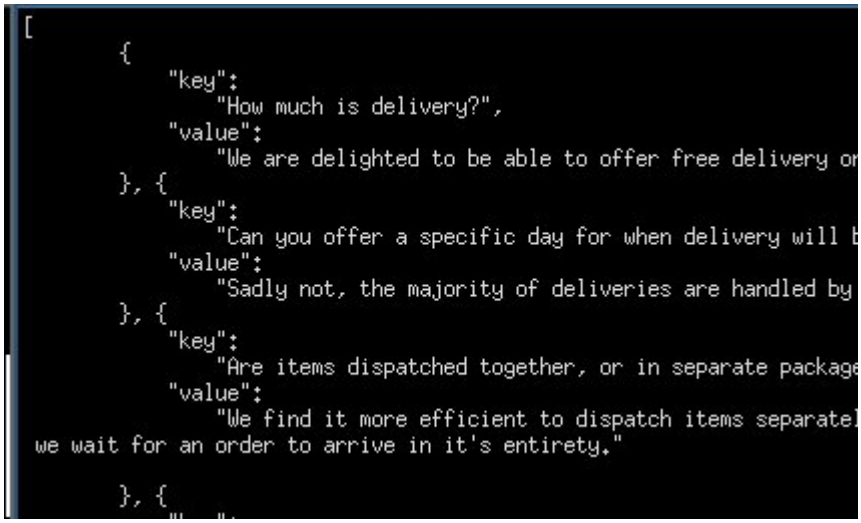

Figure 11. jsonkv format, bookdepository' FAQ

The bot compares the user's question with known questions and returns an answer to the question which is most similar to the user's one.

4.6 Telegram Integration and bot's workflow

The integration of Telegram is simple and straightforward. ***Telegram Bot API*** provides the necessary endpoints for receiving chat updates and sending messages. By means of ***requests*** library it is even easier.

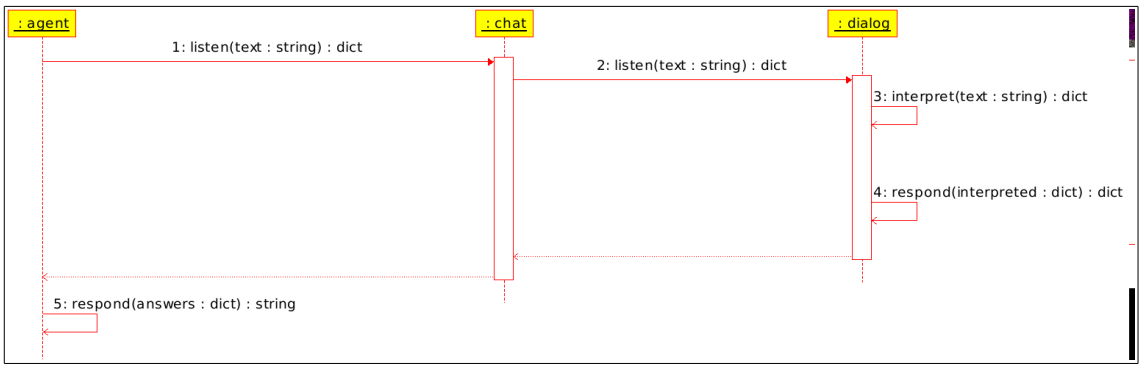

Figure 12. bot's workflow

A user sends a message to the bot in Telegram, an agent reads it and passes it to the **chat** object which contains **dialog** objects. Each **dialog** object is responsible for its own task: command execution, a person's information search, a search for an answer on the question. **Dialog** returns the possible response with a value of certainty, then the agent chooses the most certain response and sends it to the user who made the request.

4.7 Tests

Unit tests were written with use of **pytest** library (section **4.2.5**).

The tests are written into a separate module for testing each module of the program separately, for example **commands_test.py** and **telegram_test.py** which test prefix commands (section **4.3**) and telegram methods (section **4.6**) respectively.

You can either run separate modules of tests by specifying names of files or all modules by from directory by specifying it.

```
>> pytest tests/commands_test.py
```

or

```
>> pytest tests/
```

Every tests will be run even if some tests fail in the middle. Whenever a test fails, information about the failed test will be displayed (**Figure 13**). If all tests pass, no additional information will be displayed (**Figure 14**).

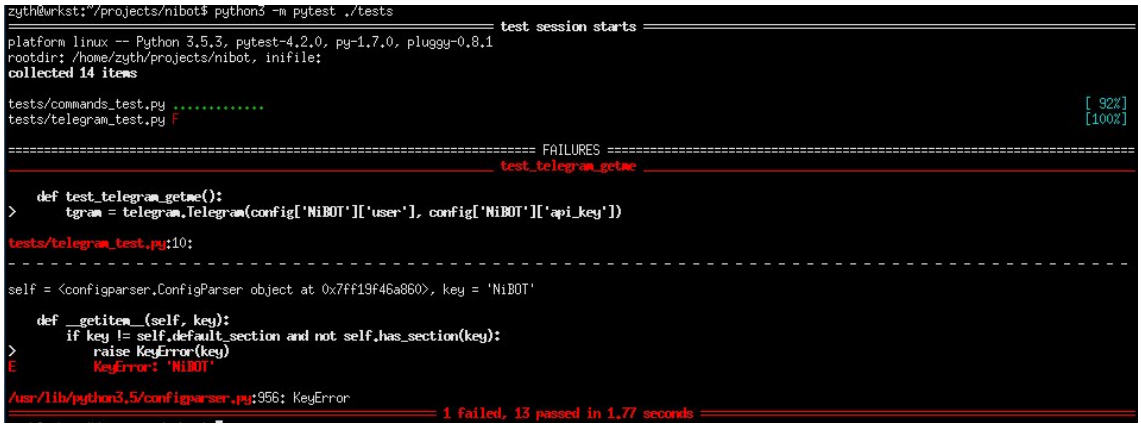

Figure 13. Tests failed

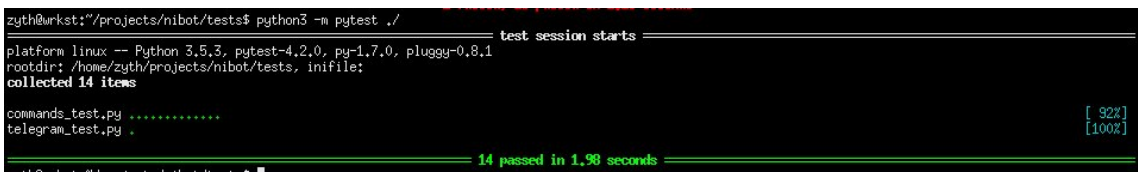

Figure 14. Tests passed

5 RESULTS

For the main testing of the solution I have used Telegram app. Also snippets from browser and terminal emulator will be included.

The program is passing requests through all dialogues, each of them has its own independent task: executing command, searching for person's information or searching for answer on a question from request. None of the tests returned a result from a wrong dialogue.

5.1 Commands

The first sent command is ***!name***, which does not require any arguments and uses only the agent's object itself. This command successfully returns the name of the bot specified in the agent.

The second command is ***!coin***. This command uses python's ***random*** library and returns either "HEAD" or "TAIL", and does not require any arguments.

For the third command I chose ***!choose***, which also uses ***random*** library, but requires arguments from which a choice will be made. One of the arguments is chosen randomly and then returned.

Results of this test can be seen on **Figure 15**.

5.2 Person information search

The database for this test contains several records of people with their name, phone number and address.

First, I have requested Jack Black's phone number by passing the exact semantic pattern for which the dialogue searches. Expectedly a correct answer is received.

The second task has a partial match to the pattern—the request does not contain a question. In this test the address of an other person was requested and the result was correct.

Results of this test can be seen on **Figure 16**.

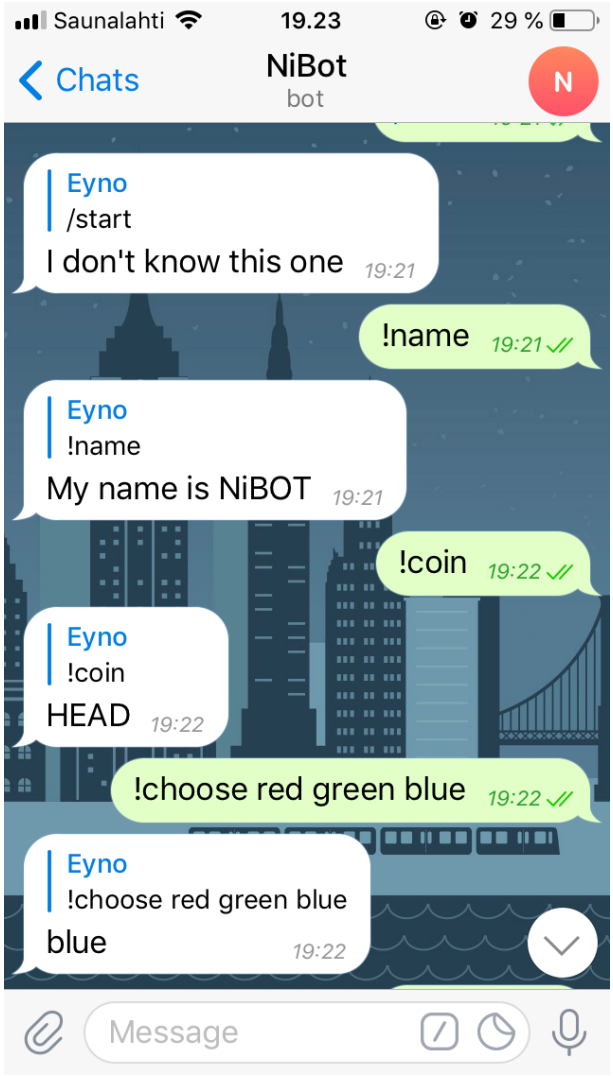

Figure 15. Command use in Telegram

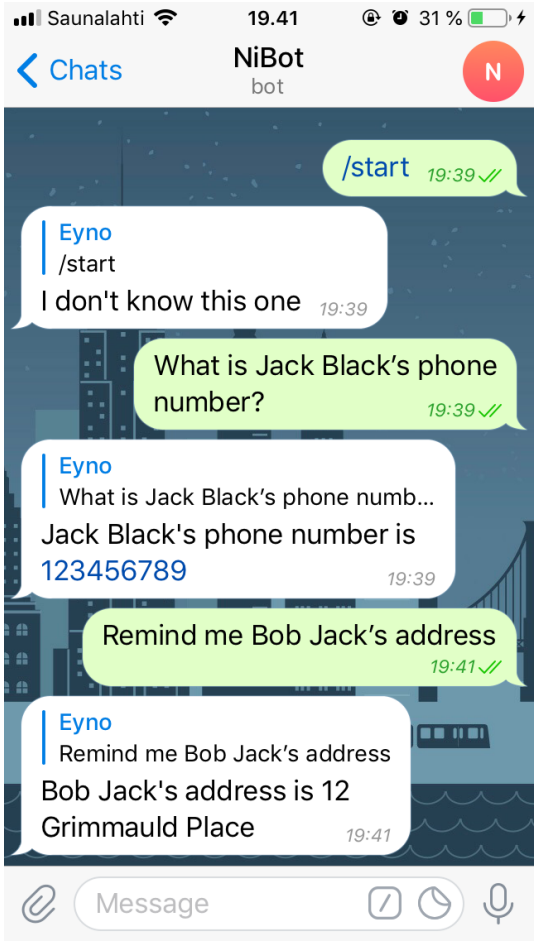

Figure 16. person's information request in Telegram

5.3 Answer search

For the purposes of this test, two different FAQ pages were used, as stated in section **4.5**.

For testing I have modified the original questions from FAQs when used in the requests. In both cases the bot returned a correct answer.

Table 7. Questions asked

Test	Site	Original	Modified
1	bookdepository.com	How much is delivery?	How much does delivery cost?
2	wordpress.com	Can I install WordPress on Windows Server?	Can I install Wordpress on to windows server?

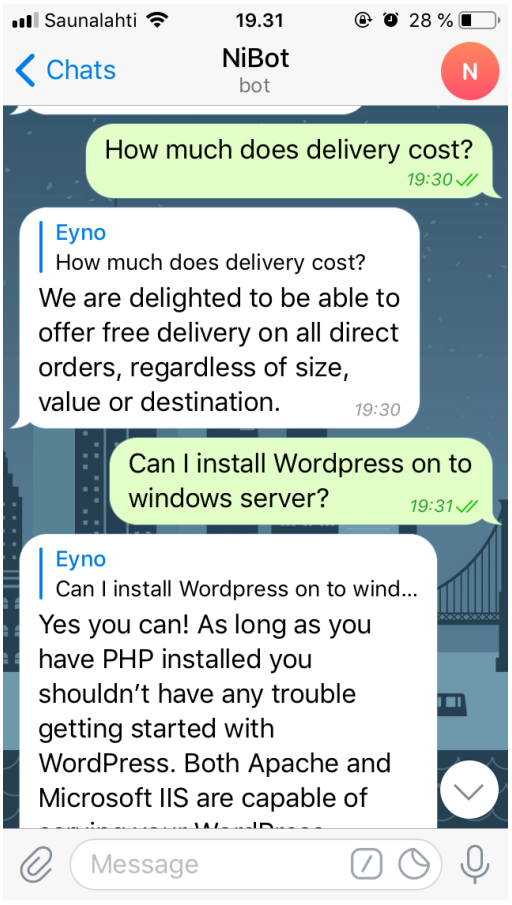

Figure 17. Search for an answer in Telegram

5.4 Terminal emulator and browser

Terminal emulator (**Figure 18)** ran by ***run.py*** file. The prompt for user's input is "What do you want?", which can be easily modified. After the prompt, the user types their request and receives an answer or "I don't know this one" if the question cannot be answered.

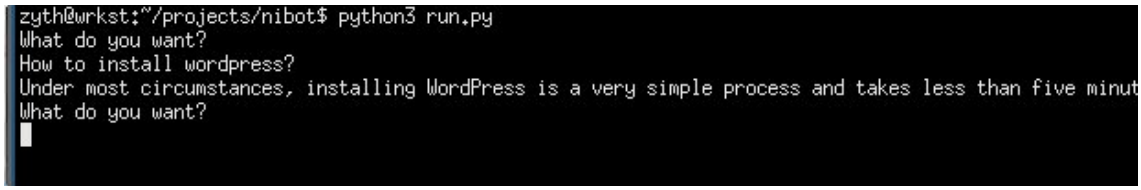

Figure 18. Terminal emulator

As I did not have any requirements for the UI, browser (**Figure 19**) interaction was made as simple as it can be. Was created simple form for user's request and a button for sending it. After the request is sent, it appears at the top of the page in the "REQUEST" field, and after the response is received, it appears in the "RESPONSE" field. The requests are made to python Flask server.

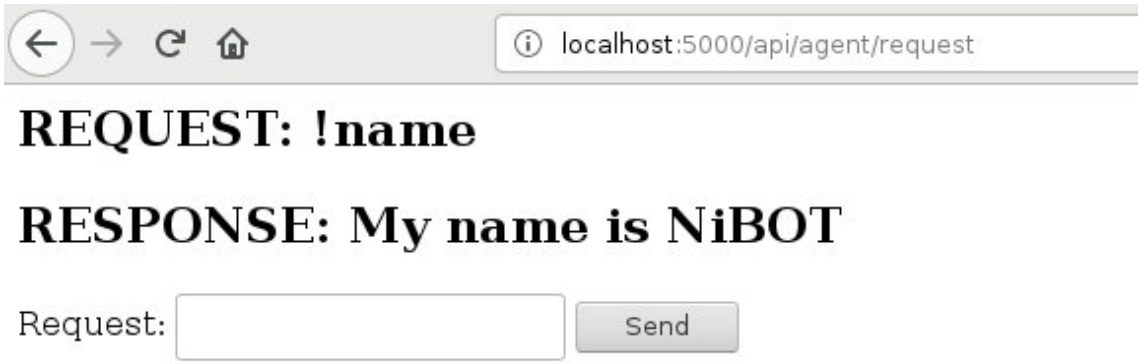

Figure 19. Browser

5.5 Restrictions

The current implementation of the answer search (section **5.3**) has a low level of answer certainty. And as certainty is low, the difference of certainty between the answers is low too, which makes it hard to determine the threshold for correct answers.

The search for a person’s information (section **5.2**) is pattern based. Word or phrases from the question of the answer search can match the pattern of a person’s information dialogue and trigger its certainty. As the answer search has low certainty, the dialogue of person’s information would dominate. The solution could be to use the predefined threshold of certainty for each dialogue and reduce the certainty to 0, if it is lower than threshold.

6 CONCLUSION

The bot designed in this thesis is a strong interface between a person and their computer/server. The bot is highly customizable and extendable. With time and concentration on one function of the bot at a time – be it command execution on machine or search of information – the bot can become an everyday program for tasks which involve execution of other programs on a server or search of information.

However, Further development is required.

REFERENCES

[1] Wikipedia, Semantics

https://en.wikipedia.org/wiki/Semantics

Last access 27.02.2019

[2] Wikipedia, Syntax

https://en.wikipedia.org/wiki/Syntax

Last access 27.02.2019

[3] Wikipedia, Morphology

https://en.wikipedia.org/wiki/Morphology_(linguistics)

Last access 27.02.2019

[4] Wikipedia, Stemming

https://en.wikipedia.org/wiki/Stemming

Last access 27.02.2019

[5] Wikipedia, Lemmatization

https://en.wikipedia.org/wiki/Lemmatisation

Last access 27.02.2019

[6] The Bells, Edgar Allan Poe

[7] Wikipedia, Naive Bayes

https://en.wikipedia.org/wiki/Naive_Bayes_classifier

Last access 27.02.2019

[8] Wikipedia, *k*-nearest neighbors

https://en.wikipedia.org/wiki/K-nearest_neighbors_algorithm

Last access 27.02.2019

[9] Python requests description from official site

http://docs.python-requests.org/en/master/

Last access 27.02.2019

[10] scikit-learn Pipeline code examples

https://scikit-learn.org/stable/modules/generated/sklearn.pipeline.Pipeline.html

Last access 27.02.2019

[11] minimal Flask application

http://flask.pocoo.org/docs/1.0/quickstart/

Last access 27.02.2019

[12] Telegram Bot API description

https://core.telegram.org/bots/api

Last access 27.02.2019

[13] Pytest example

https://docs.pytest.org/en/latest/

Last access 27.02.2019

[14] Bookdepository FAQ

https://www.bookdepository.com/help

Last access 27.02.2019

42
[15] WordPress FAQ

https://wordpress.org/support/article/faq-installation/

Last access 27.02.2019

Appendix A. Research materials

Artificial Intelligence: A Modern Approach

Peter Norvig, Stuart J. Russell

2009

NLTK, Natural Language Processing with Python

Steven Bird, Ewan Klein, and Edward Loper

2015

Applied Text Analysis with Python: Enabling Language Aware Data Products with Machine Learning

Benjamin Bengfort, Rebecca Bilbro, Tony Ojeda

2018