DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Notification Timing for a Proactive Virtual Dietary Advisor

Oguz Gültepe

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Bachelor's Thesis in Informatics

Notification Timing for a Proactive Virtual Dietary Advisor

Timing von Benachrichtigungen für einen proaktiven virtuellen Ernährungsberater

Author:	Oguz Gültepe
Supervisor:	PD Dr. rer. nat. Georg Groh
Advisor:	Monika Wintergerst
Submission Date:	15.07.2019

I confirm that this bachelor's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15.07.2019 Oguz Gültepe

Acknowledgments

Abstract

Contents

Acknowledgments iii

Abstract iv

1 Introduction 1
- 1.1 Nutrition and Health 1
- 1.2 Health and Well Being Applications 1
- 1.3 Goal and Structure of the Thesis 1

2 Theoretical Background 2
- 2.1 Natural Language Processing 2
  - 2.1.1 Regular Expressions 2
  - 2.1.2 Information Extraction 6
  - 2.1.3 Dialog Systems and Chatbots 6
- 2.2 Machine Learning 6
  - 2.2.1 Basics of ML 6
  - 2.2.2 Relevant Models 6
- 2.3 Nutrition 6
  - 2.3.1 Dietary Assesment 6
  - 2.3.2 Dietary Goals 6

3 Methodology 7
- 3.1 Chatbot 7
  - 3.1.1 Design 7
  - 3.1.2 Telegram Integration 7
  - 3.1.3 Database 7
- 3.2 Machine Learning 7
  - 3.2.1 Frameworks 7
  - 3.2.2 Feature Templates 7
  - 3.2.3 Models 7
  - 3.2.4 Training and Testing 7
  - 3.2.5 Implementation 7

4 Results **8**
4.1 Section 8

5 Discussion **9**
5.1 Section 9

6 Conclusion **10**
6.1 Summary 10
6.2 Future Work 10

7 latex-guide **11**
7.1 Section 11
7.1.1 Subsection 11

List of Figures **13**

List of Tables **14**

Bibliography **15**

1 Introduction

1.1 Nutrition and Health

1.2 Health and Well Being Applications

1.3 Goal and Structure of the Thesis

2 Theoretical Background

2.1 Natural Language Processing

Natural language processing (NLP) is a subfield of computer science which focuses on, as implied by the name, processing of human (natural) languages. NLP enables us to make use of copious knowledge that is expressed in natural language. [RN10]

In this section we will be taking a look at some NLP concepts that are relevant to us. Most of the knowledge provided here is based on [JMb]. We will start with regular expressions, powerful tools able to catch text patterns. Then we will be looking at information extraction from natural language. Finally, we will finish this chapter on dialog systems and chatbot.

2.1.1 Regular Expressions

A regular expression is formally defined as an algebraic notation that represents a specific set of strings. However, this representation is not explicit. Regular expressions denote textual patterns which are able to produce implicit sets. These patterns are useful for searching in text. A regular expression search function finds every instance of text in the corpus that belongs to the pattern's implicit set. These instances are said to be 'matched' by the pattern.

Regular expressions are supported in every computer language, word processor and text procesing tool. However, there may be differeces in how they treat certain expressions. Here, we will be treating expressions as they are shown in [JMa]. Another thing to keep in mind is that this is not a comprehensive guide for regular expression. We will be looking at basics and some more complex operators we need for our specific problem.

Basic Regular Expression Patterns

Simplest regular expressions are sequences of characters. For example `snack` matches any string that contains the sequnce 'snack'. Here are some examples that would be matched by this regular expression:
"I just had a small <u>snack</u>."

"His username was 123snacko."
As we see, this regular expression only searches for the sequence 'snack'. Whether the found sequence actually is a word or not is irrelevant.

The period `.` is treated as a special character: the wildcard. It matches any character, so for example `r.n` matches:
"I ran 5k today."
"You better run."
Another special charcter is the question mark: `?` Adding a question mark after an element makes the element optional. For example, `hours?` matches:
"The hour arm of the clock was missing."
"I have been waiting for you for hours."
Special characters can also be used as regular characters. We just need to put a backslash \ before the special character. For example, `Inc\.` matches:
"Monsters Inc. was a great movie."

Table 2.1: Regular Expression Basics

Regular Expression	Match	Example
`duck`	'duck'	duck
`r.ck`	'r' and 'ck' with any character in between	rock, rack
`minutes?`	'minute' or 'minutes'	minute,minutes
`\?`	'?'	How?

Square Brackets, Range and Negation

It is important to note that regular expressions are case sensetive, so `meal` matches the first example, but not the second:
"We had a tasty meal."
"Meal is ready."
This problem is solved by the use of square brackets: `[]`. Square brackets signify a disjunction between the characters inside. So `[Mm]` means either 'M' or 'm' and `[Mm]eal` matches both examples:
"We had a tasty meal."
"Meal is ready."
Square brackets can be used for regular expression that match any digit `[1234567890]` or any letter `[abcdefghijklmnopqrstuvwxyz]`. However, for such common disjunctions we can also use the dash `-` operator. Dash operator denotes a range: `[0-9]` matches any digit between '0' and '9'. `[a-z]` and `[A-Z]` match any lowercase letter and any

uppercase letter, respectively.

Another use of square brackets is negation. When a caret `^` is the first character inside a bracket, any character other than the ones inside the brackets is matched. `[^ab]` matches any character that is not 'a' or 'b'. `[^0-9]` matches any character that is not a digit.

It is important to note that when used outside of these contexts, both dash `-` and caret `^` have different meanings. Outside of the brackets, dash operator is treated as a character. When the caret occurs in the brackets after the first character, it is also treated as a character. For example, `0-9` matches the first sentence but not the second:
"Please enter a value between 0-9."
"This course is worth 8 ECTS credits."

Table 2.2: Use of Square Brackets

Regular Expression	Match	Example
`[Ss]nack`	'Snack' or 'snack'	Snack,snack
`[0-9]`	Any digit	5 missed calls!
`[^abc]`	Any character that is not 'a','b' or 'c'	carb
`[3^2]`	'3','^', or '2'	n^1 equals n

Disjunction and Anchors

Now, with the help of square brackets, we are capable of basic disjunction between characters. But what if we need a pattern that matches either 'meal' or 'snack'? `[mealsnack]` wouldn't work as square brackets denote character level disjunction. What we need here is the pipe symbol `|`, the disjunction operator. `snack|meal` matches both of the following examples:
"I need a snack."
"When was your last meal"
Next, we have the anchors: special characters that can specify locations in text. For example, the caret operator `^` when used outside of brackets specify the start of a line. The dollar sign `$` specifies the end of a line. So `^I|\?$` matches any sentence that either starts with I or ends with a question mark:
"I thought we were gonna eat together."
"Are you available for lunch ?
We have two more anchors that we can make use of: `\b` to match word boundries and `\B` to match non-boundries. For example, `\bsnack\b` matches the first sentence, but not

the second:
"I just had a small snack."
"His username was 123snacko."
To use these anchors, it is important to understand what a word means. In terms of regular expressions, a word is defined as any sequence of digits, letters or underscores. So `\bap\b` would match 'A$ap'. As $ is neither a digit or a letter or underscore, it is treated as a word boundry.

Table 2.3: Disjunction and Anchors

Regular Expression	Match	Example
`eat|ate`	'eat' or 'ate'	I just ate.
`^\.`	Any character at the start of a line	I just ate.
`\.$`	Any character at the end of a line	I just ate.
`\b59\b`	'59' between word boundries	It costs $59.99.
`\B59\B`	'59' between non-boundries	It costs $2599.

Kleene Operators, Grouping and Precedence

There are cases where we need to match repetetive patterns. For example, we might want to match any 'hey' with an arbitary amount of 'y's. To do this, we can use the Kleene Star `*`. Kleene star means "0 or more occurrences of the preceeding element". This means `y*` matches '','y','yy' and any other number of 'y' characters. So in order to match any 'hey' with an arbitary amount of 'y's, we would use `heyy*`. This use, however, is so common that we have another operator for it: the Kleene Plus `+`. Kleene Plus means "1 or more occurrences of the preceeding element". So `heyy*` and `hey+` are functionally equal.

But what if we need to match a phrase such as as 'hahaha', which consists of an arbitary number of 'ha's? `ha+` wouldn't work, because the Kleene operator works on the preceeding element. So how can we set 'ha' as the preceeding element? This is where grouping comes into play. In regular expressions, grouping is done by wrapping a phrase in parantheses. From then on, the phrase inside the parantheses is treated as a singular entity by operators outside the parantheses. `(ha)+` matches any 'ha', 'haha' and any number of repetitions of the phrase 'ha'.

More Operators

Capture Groups

2.1.2 Information Extraction

2.1.3 Dialog Systems and Chatbots

2.2 Machine Learning

2.2.1 Basics of ML

2.2.2 Relevant Models

2.3 Nutrition

2.3.1 Dietary Assesment

2.3.2 Dietary Goals

3 Methodology

3.1 Chatbot

3.1.1 Design

Chatbot Architecture

Information Extraction

3.1.2 Telegram Integration

python-telegram-bot Library

Implementation

3.1.3 Database

SQLite3

Implementation

3.2 Machine Learning

3.2.1 Frameworks

3.2.2 Feature Templates

3.2.3 Models

3.2.4 Training and Testing

3.2.5 Implementation

4 Results

4.1 Section

5 Discussion

5.1 Section

6 Conclusion

6.1 Summary

6.2 Future Work

7 latex-guide

7.1 Section

Citation test [Lam94].

7.1.1 Subsection

See Table 7.1, Figure 7.1, Figure 7.2, Figure 7.3.

Table 7.1: An example for a simple table.

A	B	C	D
1	2	1	2
2	3	2	3

$$\begin{tikzcd}
R_1 \arrow[r, dash] \arrow[d, dash] & R_2 \arrow[r, dash] \arrow[d, dash] & R_5 \\
R_3 \arrow[r, dash] & R_4 &
\end{tikzcd}$$

Figure 7.1: An example for a simple drawing.

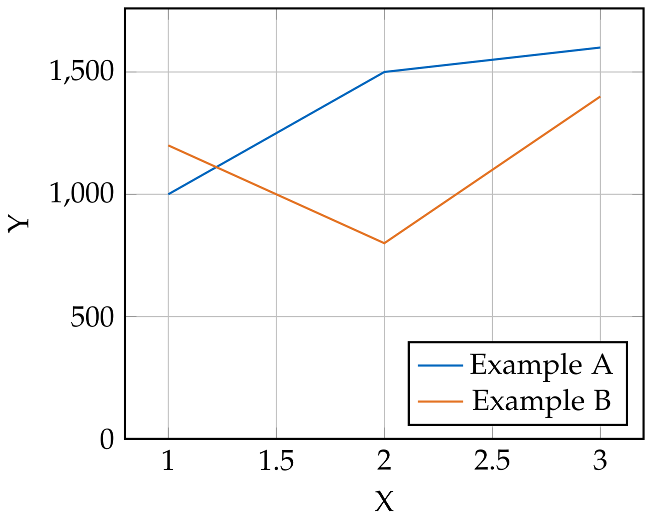

Figure 7.2: An example for a simple plot.

```
SELECT * FROM tbl WHERE tbl.str = "str"
```

Figure 7.3: An example for a source code listing.

List of Figures

7.1 Example drawing 11
7.2 Example plot 12
7.3 Example listing 12

List of Tables

2.1 Regular Expression Basics 3
2.2 Regular Expression Square Brackets 4
2.3 Regular Expression Disjunction and Anchors 5

7.1 Example table 11

Bibliography

[JMa] D. Jurafsky and J. H. Martin. "Regular Expressions, Text Normalization, and Edit Distance." In: *Speech and Language Processing*. 3rd ed. Chap. 2. In preparation.

[JMb] D. Jurafsky and J. H. Martin. *Speech and Language Processing*. 3rd ed. In preparation.

[Lam94] L. Lamport. *LaTeX : A Documentation Preparation System User's Guide and Reference Manual*. Addison-Wesley Professional, 1994.

[RN10] S. J. Russell and P. Norvig. "Natural Language Processing." In: *Artificial Intelligene: A Modern Approach*. 3rd ed. Pearson, 2010. Chap. 22.