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

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. (Russell and Norvig 2010b)

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 (Jurafsky and Martin n.d.[d]). 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. As the knowledge provided here is based on (Jurafsky and Martin n.d.[c]), we will be treating expressions as they are shown there. 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 snack."
"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."

There are some shorthands for commonly used ranges or disjunctions. `\d` is equivalent to `[0-9]` and matches any digit. `\s` matches any whitespace.

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
`\d`	Any digit	5 missed calls!
`\s`	Any whitespace	5 missed calls!

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]` would not 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+` would not 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'.

Grouping also comes in handy when using the disjunction operator. For example, let's assume we need a regular expression that matches 'hour', 'minute', 'hours' and 'minutes'. We ca not use `hour|minutes?`, because the disjunction is between `hour` and `minutes?` This is because the `?` is evaluated first, then the sequences and only then the disjunction. Instead, we can use `(hours|minute)s?`. This expression works since the parantheses are evaluated before all the other operators.

The collection of rules determining which operator is evaluated first is called the operator precedence hierarchy. An operator x is said to have precedence over another operator y, if x is to be evaluated before y. Regular expression operator precedence hierarchy is as follows, from highest precedence to lowest precedence:

Table 2.4: Regular Expression Operator Precedence Hierarchy

Parantheses	`()`
Counters	`* + ?`
Sequences and anchors	`seq ^I \.$`
Disjunction	`|`

Table 2.5: Regular Expression Kleene Operators and Grouping

Regular Expression	Match	Example
`b*`	0 or more repetitions of 'b'	aaabbb
`a+`	1 or more repetitions of 'a'	aaabbb
`(ab)+`	1 or more repetitions of 'ab'	aaabbb

Capture Groups

Let's assume we have a list of names and we want to make sure all the names occur only once in the list. The following regular expression accomplishes that:
`\b([A-Z][a-z]*)\b.*\b\1\b`
When we use parantheses for grouping, the text that matches the expression inside is stored in the memory. This stored text is referred to as a capture group. Capture groups are numbered from left to right and can be referenced by the use of a backslash, followed by the group number.

When we need to group expressions but do not want to store the matches in the memory, we can use non-capturing groups. Adding `?:` after the opening paranthesis makes a group non capturing.

2.1.2 Information Extraction

We have mentioned that massive amount of information is expressed in natural languages. The process of converting this information into a structured data format is called information extraction. In this subsection, we will be taking a look at information extraction techniques that are relevant to us.

We will start with extracting temporal expressions from text. We will continue with normalizing the extracted temporal expressions. Finally, we will end this subsection on template filling.

Knowledge provided in this subsection is based on (Jurafsky and Martin n.d.[b]).

Extracting and Normalizing Temporal Expressions

Temporal expressions are parts of text that refer to durations or points in time. Here, we will be focusing less on durations and more on points in time. Temporal expressions such as "17th of July" and "19.30" that directly refer to point in time are called absolute temporal expression. Expressions such as "last week" and "2 hours ago" that refer to points in time in relation to some other point are called relative temporal expressions.

In order to extract temporal information, we first need to find spans of text that contain temporal expressions. This task can be accomplished with the help of lexical triggers. Lexical triggers are textual phrases that imply the existence of a temporal expression. For example; yesterday, past, next or hour. It is possible to build automatas that search and find lexical triggers in text. Additional layers of automatas can be built over these in order to find the spans of temporal expressions based on the lexical triggers.

After extraction, temporal expressions need to be normalized. This process includes mapping the expressions to points in time, be it a calendar date, a time of the day or both. This is fairly simple for absolute time expressions, as they refer to points in time directly. Relative temporal expressions, however, refer to points in time in relation to the document's temporal anchor. A document's temporal anchor is the point in time in which the documents is set. This can for example be the date an article was published or the time a text message was sent. Based on the temporal anchor, relative temporal expressions can be mapped to points in time using temporal arithmetic.

Finally, the points in time, which the temporal expressions were mapped to, must be saved in a standardized way. The standards for saving temporal expressions can be seen in (*Date and time – Representations for information interchange – Part 1: Basic rules* 2019)

Template Filling

Many times we have to deal with texts describing common and stereotypical events. These events usually follow certain patterns or have certain participants with certain roles. Our knowledge of these common events may come in handy in information extraction. For example, when we hear the following:
"Would you like to join me for dinner at L'Osteria tomorrow at 8pm?"
Even if we have never heard of L'Osteria, we can assume that it is a restaurant. This is because inviting somebody for a dinner is a common event which usually includes an inviter, invitee, a restaurant and a time. In a simple way, such common events can be represented as templates with fixed slots to fill. A template for a dinner invitation would look something like Table 2.6

Table 2.6: Example Template For a Dinner Invitation

Key	Value
Inviter	You
Invitee	Me
Restaurant	L'Osteria
Time	Tomorrow 8pm

Such templates make it easy to infer details that are not explicitly mentioned, such as the fact that L'Osteria is a restaurant. The task of template filling refers to finding the events that fit particular templates and filling the associated templates with information extracted from the text.

2.1.3 Dialog Systems

In this subsection, we will be talking about dialog systems. Dialog systems are programs use natural language to communicate with the user. In this subsection, we will first go through the difference between task-oriented dialog agents and chatbots. Next we will be taking a look at modern task-oriented dialog systems: frame based dialog agents. Finally, we will end this subsection on control structures for such dialog agents. The knowledge provided in this subsection is based on (Jurafsky and Martin n.d.[a]).

Task-Oriented Dialog Agents vs Chatbots

The words chatbot and dialog agents are often used interchangably by many people. However, in the context of NLP, this is a mistake. In NLP, chatbots refer to a distinct class of dialog agents, the other class being task-oriented dialog agents. So what is the difference between the two classes? Let's start with task-oriented dialog agents.

Task-oriented dialog agents are dialog systems build to -as the name suggest- serve specific tasks. These tasks can include for example finding restaurants, booking hotels or sending messages. Popular digital assistants such as Alexa and Siri fall under this category. Chatbots, on the other hand, are dialog agents without specific goals, built to mimic human conversation.

Here we will be focusing on task based dialog agents rather than chatbots.

Frame Based Dialog Agents

What is a frame based dialog agent? In order to answer this question, we first need to understand some concepts.

Domains are specific areas of knowledge or activity such as computer science or fishing. Domains are modelled by abstract constructs called domain ontologies. A domain ontology defines concepts and relations that belong to a domain.

We build frames based on domain ontologies. A frame is a collection of slots that accept predefined types of values. The slots of a frame defines the information that the systems needs for the task. Dialog agents that are based on such frames are called frame based dialog agents.

Table 2.7 shows an example frame for a task-oriented dialog agent designed to make reservations at a restaurant:

Table 2.7: Example Frame for a Task-Oriented Dialog Agent

Slot	Type
Restaurant	String
Number of People	Integer
Reservation Date	Date
Reservation Time	Time

Control Structure for Frame Based Dialog

A task-oriented dialog agent needs to gather appopriate data to accomplish the task at hand. In order to achieve this, dialog systems implement control structures that guide the conversation. For a frame base dialog agent, the control structure is built around the frame. These control structures are usually finite-state automatas that are hand-designed for the task.

Let's consider the example frame shown at Table 2.7. First, for each slot, we come up with an associated question that elicits an answer to fill the slot. Table 2.8 shows these questions for our example frame.

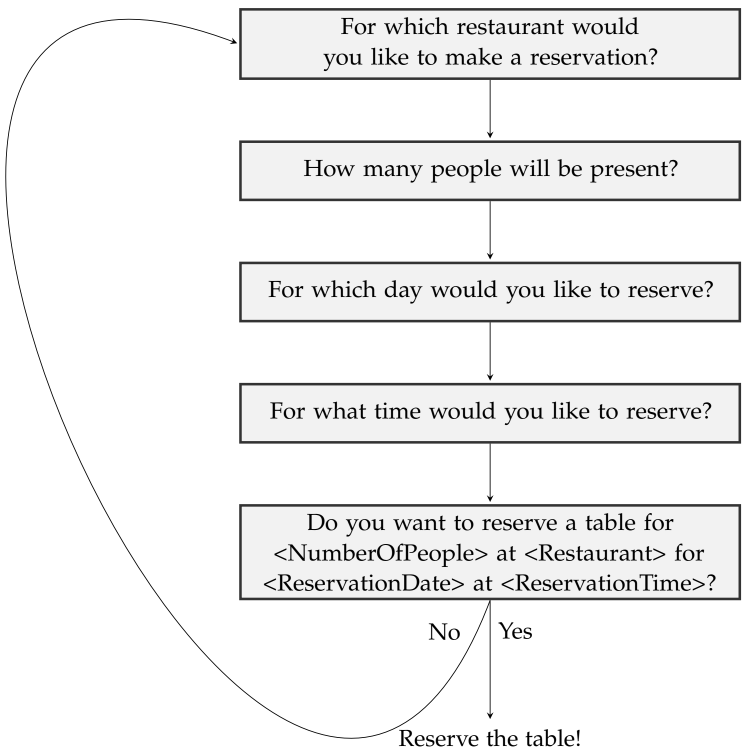
Table 2.8: Questions for the Example Frame

Slot	Question
Restaurant	For which restaurant would you like to make a reservation?
Number of People	How many people will be present?
Reservation Date	For which day would you like to reserve?
Reservation Time	For what time would you like to reserve?

Based on these questions, we build a finites-state automata to guide the conversation. In this automata, questions are the states and user's answers are the transitions. Take a look at Figure 2.1 for an example finite-state automata that is built based on the questions at Table 2.8.

In our example, the conversation is controlled entirely by the automata. The user does not have much option other than answering the questions. This means that the user has no initiative; no control over the conversation. Systems where the user has no initiative such as the one shown at Figure 2.1 are called system-initiatives. The major advantage of system-initiative architecture is the fact that the system always knows what question is being answered. This alleviates the need to match answers with slots. It also allows for fine tuning the answer processing for the expected type of answer. However, such systems offer the user no flexibility. Consider the following input from the user:
"I would like reserve a table at L'Osteria for tomorrow"
This sentence already includes the information needed to fill two slots: Restaurant and Reservation Date. It is possible to augment a dialgo system with the ability to recognize such inputs, allowing for multiple slots to be filled at once. Such a system would pose the user questions for the empty slots, and simply skip the questions for the filled ones. These systems are called mixed-initiatives, as the user also holds some inititive to control the conversation.

Figure 2.1: Example Finite-State Automata as Control-Structure



2.2 Machine Learning

Machine Learning is the science (and art) of programming computers so they can learn from data. (Géron 2017d) In this section, we will take a look at some machine learning concepts that are relevant to this thesis. We will start with basics of ML, explaining some basic terminology. Afterwards we will take a brief look at the models used in the thesis. Finally, we will end the section on some additional relevant concepts. Machine learning is a broad subject with many subcategories. Here we will be focusing on supervised learning; learning from labeled data.

2.2.1 Basics of Machine Learning

How does a program learn from labeled data? Assume we have a dataset consisting of N input-output pairs such as the following:
$(x_1, y_1), (x_2, y_2), \ldots, (x_n, y_n)$

For any of these pairs, we call the input value x_i the predictor and the y_i the labels.(Géron 2017d) Supervised learning is based on the assumption that there is function dependency between the predictors and the labels. This functional dependency can be modeled as $f(x_i) = y_i$, where f is called the true function. The goal of the supervised learning is to construct a function h -called hypothesis- that approximates the true function f. In general, we split supervised learning instances into two groups: classification and regression. Classification refers to the instances where the labels are selected from a finite set of values (such as cat, dog or cow). Regression on the other hand refers to the instances where the labels are numeric values (such as 0, 10 or 42). (Russell and Norvig 2010a)

We usually construct hypotheses by training models on the data. But what does it mean to train a model on the data? To explain this we need to understand what a loss function is: A loss (or cost) function is a function that maps models to numeric values. Specifically, a loss function tells us how much our model fails to explain some data. In this context, training means tuning the adaptive parameters of a model in order to minimize the loss function. (Bishop 2006)

We are of course interested in models that are capable of explaining more than just the training data. We want to see if the model generalizes well; if it can accurately predict labels for previously unseen inputs. Becuase of this we test the trained models on heldout data. Usually, we split our data into three distinct sets: training set, validation set, and test set. We train our models on the training set, compare them using the validation set and report the performance of the final model on the test set. It is important that the test set is held out during the training phase. Otherwise it is not possible to see if the model generalizes well or just memorizes the data.(Bishop 2006)

2.2.2 Relevant Models

Let's take a brief look at some popular machine learning models.

Linear Regression

A linear regression model is a linear function of the input variabes. It corresponds to a weighted sum of the input features plus a constant called the bias term.(Géron 2017f) For example, a linear regression model for a dataset with two input features would look like this:

$f(x_i) = \theta_0 + \theta_1 x_{i1} + \theta_2 x_{i2} = y_i$

For more information on linear regression models, please refer to (Géron 2017f).

Ridge Regression

Ridge regression is very similar to linear regression. The only difference is that ridge regression adds a regularization term $a \sum_{i=1}^{n} \theta^2$ to the cost function. This penalizes large weight in order to prevent overfitting, a phenomenon where a model memorizes the data instead of generalizing over it.(Géron 2017f)

a is called the regularization strength and determines how much the large weights get penalized. Regularization strength is a hyperparameter, a parameter that is set before the learning process and is not optimized over the data.(Géron 2017f)

More information about ridge regression can be found at (Géron 2017f)

Support Vector Machines

Support Vector Machines construct hyperplanes that act as decision boundries between classes. What makes these hyperplanes special is that they are positioned to be as far away as possible from the closest data points. The closest data points therefore determine where the hyperplane lays, and are called the support vectors. (Géron 2017e)

This is all better understood with some visualisation. Please take a look at Figure 2.2. It is easy to see in this picture that the hyperplane is positioned to maximize the margin between the support vectors. It is also important to realize that the data points that are not support vectors do not affect the hyperplane in any way. (Géron 2017e)

Support vector machines can also be used for regression. This is also done by constructing a hyperplane. However, the objective is reversed. This hyperplane is positioned to include as many data points in the margin as possible. The size of the margin is determined by hyperparameter ϵ. (Géron 2017e)

For the mathematical background and more information on support vector machines, please refer to (Géron 2017e).

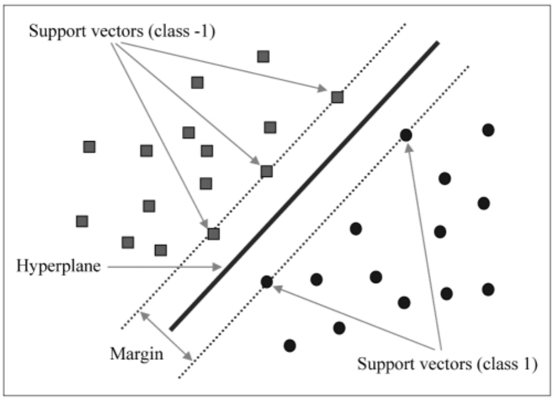


Figure 2.2: A Support Vector Machine (Chen, Hsiao, Huang, et al. 2009)

Decision Trees

Decision trees are a form of graph based models.(Bishop 2006) More specifically, decision trees are trees that contain basic conditional statements on their non-leaf nodes. For any given feature vector, a decision tree runs tests these conditions to reach a decision. These tests start at the root of the tree. Depending on the outcome of the test, a child node is selected. This process continues iteratively until a leaf node is reached, where a decision is stored. Decision in this context is the label for the feature vector.(Russell and Norvig 2010a)

For the mathematical background and more information on decision trees, please refer to (Géron 2017a).

Ensemble Learning

Instead of training a singular model, we can train many weak models and aggregate the results. Such aggregations of models are called ensembles, and the learning process is called ensemble learning. One simple yet particularly powerful example is the random forest: A random forest is an ensemble of decision trees, each trained on different random subsets of the training set. A random forest makes predictions by aggregating the predictions of each individual tree. For regression, the predictions are averaged. For classification, the most predicted class is the final prediction.(Géron 2017c)

More information on ensemble learning can be found at (Géron 2017c)

2.2.3 More Machine Learning

Finally, let's go through some more advanced machine learning concepts that will be relevant in the next chapter.

Data Preprocessing

It is often the case that we have to work on imperfect data. Some values might be missing for some instances, data might not be uniform or features might not be helpful as they are. Collection of transformations we apply to the data before feeding it into to our models is called preprocessing.(Géron 2017b) This preprocessing includes:

Cleaning the data to make sure it is uniform.

Extracting features from the existing input variables.

Scaling the features to make sure all the features have similar scales.

The reasoning behind these transformations and more information on preprocessing can be found in (Géron 2017b)

Cross Validation

Cross validation is a technique used in model selection. The data is first divided into N parts. Afterwards the model is trained on $N-1$ parts and validated on the remaining part. This process is repeated N times, each with a different part as the validation set. Finally, the best performing model is selected.(Bishop 2006)

Hyperparameter Optimization

Hyperparameter optimization refers to the process of fine tuning the model hyperparameters to improve performance. This is usually done automatically using gridsearch.(Géron 2017b)

3 Methodology

3.1 Chatbot

3.1.1 Design

Chatbot Architecture

Information Extraction

3.1.2 Telegram Integration

python-telegram-bot Library

Implementation

3.2 Machine Learning

3.2.1 Preprocesseing

3.2.2 Feature Extraction

3.2.3 Models

3.2.4 Training and Testing

3.2.5 Implementation

4 Results

4.1 Section

5 Discussion

5.1 Section

Which models behave the best How long it takes for accurate observations.

6 Conclusion

6.1 Summary

6.2 Future Work

Initial ballpark times When to reach to the user? How often to reach the user? Intervention any time food choices are made Collab Filtering Context sensetive meal logging Real intervention behaviour AB tests between clustering and initial defaults

7 latex-guide

7.1 Section

Citation test (Lamport 1994).

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.

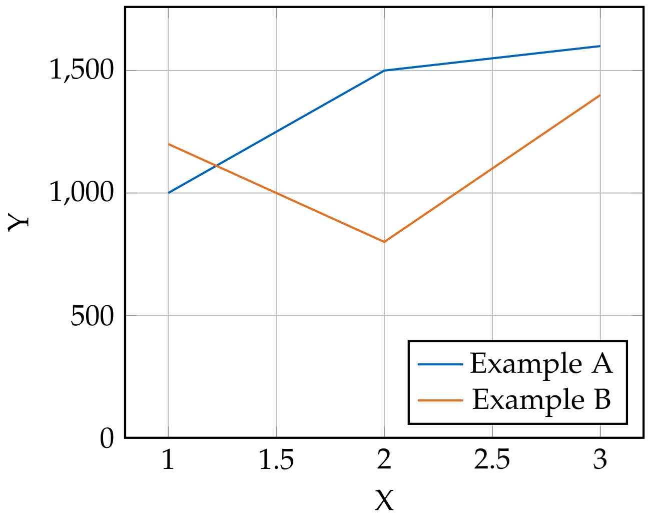


Figure 7.2: An example for a simple plot.

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

Figure 7.3: An example for a source code listing.

List of Figures

List of Tables

Bibliography

Bishop, C. (2006). *Pattern Recognition and Machine Learning*. Springer.

Chen, S.-T., Y.-H. Hsiao, Y.-L. Huang, S.-J. Kuo, H.-S. Tseng, H.-K. Wu, and D.-R. Chen (Aug. 2009). "Comparative Analysis of Logistic Regression, Support Vector Machine and Artificial Neural Network for the Differential Diagnosis of Benign and Malignant Solid Breast Tumors by the Use of Three-Dimensional Power Doppler Imaging." In: *Korean journal of radiology : official journal of the Korean Radiological Society* 10, pp. 464–71. DOI: 10.3348/kjr.2009.10.5.464.

Géron, A. (2017a). "Decision Trees." In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. Chap. 6.

– (2017b). "End-to-End Machine Learning Project." In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. Chap. 2.

– (2017c). "Ensemble Learning and Random Forests." In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. Chap. 7.

– (2017d). *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc.

– (2017e). "Support Vector Machines." In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. Chap. 5.

– (2017f). "Training Models." In: *Hands-On Machine Learning with Scikit-Learn and TensorFlow*. O'Reilly Media, Inc. Chap. 4.

Date and time – Representations for information interchange – Part 1: Basic rules (Feb. 2019). Standard. Geneva, CH: International Organization for Standardization.

Jurafsky, D. and J. H. Martin (n.d.[a]). "Dialog Systems and Chatbots." In: *Speech and Language Processing*. 3rd ed. Chap. 24. In preparation.

– (n.d.[b]). "Information Extraction." In: *Speech and Language Processing*. 3rd ed. Chap. 18. In preparation.

– (n.d.[c]). "Regular Expressions, Text Normalization, and Edit Distance." In: *Speech and Language Processing*. 3rd ed. Chap. 2. In preparation.

– (n.d.[d]). *Speech and Language Processing*. 3rd ed. In preparation.

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

Russell, S. J. and P. Norvig (2010a). "Learning From Examples." In: *Artificial Intelligene: A Modern Approach*. 3rd ed. Pearson. Chap. 18.

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