

Using Machine Learning to Enable Material Substitutions in Maker Projects

Master's Thesis
submitted to the
Media Computing Group
Prof. Dr. Jan Borchers
Computer Science Department
RWTH Aachen University

by
Mark Christy George

Thesis advisor:
Prof. Dr. Jan Borchers

Second examiner:
Marcel Lahaye

Registration date: 24.02.2022
Submission date: 24.08.2022

Eidesstattliche Versicherung

Statutory Declaration in Lieu of an Oath

Name, Vorname/Last Name, First Name

Matrikelnummer (freiwillige Angabe)

Matriculation No. (optional)

Ich versichere hiermit an Eides Statt, dass ich die vorliegende Arbeit/Bachelorarbeit/
Masterarbeit* mit dem Titel

I hereby declare in lieu of an oath that I have completed the present paper/Bachelor thesis/Master thesis* entitled

selbstständig und ohne unzulässige fremde Hilfe (insbes. akademisches Ghostwriting)
erbracht habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel benutzt.
Für den Fall, dass die Arbeit zusätzlich auf einem Datenträger eingereicht wird, erkläre ich,
dass die schriftliche und die elektronische Form vollständig übereinstimmen. Die Arbeit hat in
gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

independently and without illegitimate assistance from third parties (such as academic ghostwriters). I have used no other than
the specified sources and aids. In case that the thesis is additionally submitted in an electronic format, I declare that the written
and electronic versions are fully identical. The thesis has not been submitted to any examination body in this, or similar, form.

Ort, Datum/City, Date

Unterschrift/Signature

*Nichtzutreffendes bitte streichen

*Please delete as appropriate

Belehrung:

Official Notification:

§ 156 StGB: Falsche Versicherung an Eides Statt

Wer vor einer zur Abnahme einer Versicherung an Eides Statt zuständigen Behörde eine solche Versicherung
falsch abgibt oder unter Berufung auf eine solche Versicherung falsch aussagt, wird mit Freiheitsstrafe bis zu drei
Jahren oder mit Geldstrafe bestraft.

Para. 156 StGB (German Criminal Code): False Statutory Declarations

Whoever before a public authority competent to administer statutory declarations falsely makes such a declaration or falsely
testifies while referring to such a declaration shall be liable to imprisonment not exceeding three years or a fine.

§ 161 StGB: Fahrlässiger Falscheid; fahrlässige falsche Versicherung an Eides Statt

(1) Wenn eine der in den §§ 154 bis 156 bezeichneten Handlungen aus Fahrlässigkeit begangen worden ist, so
tritt Freiheitsstrafe bis zu einem Jahr oder Geldstrafe ein.

(2) Straflosigkeit tritt ein, wenn der Täter die falsche Angabe rechtzeitig berichtet. Die Vorschriften des § 158
Abs. 2 und 3 gelten entsprechend.

Para. 161 StGB (German Criminal Code): False Statutory Declarations Due to Negligence

(1) If a person commits one of the offences listed in sections 154 through 156 negligently the penalty shall be imprisonment not
exceeding one year or a fine.

(2) The offender shall be exempt from liability if he or she corrects their false testimony in time. The provisions of section 158 (2)
and (3) shall apply accordingly.

Die vorstehende Belehrung habe ich zur Kenntnis genommen:

I have read and understood the above official notification:

Ort, Datum/City, Date

Unterschrift/Signature

Contents

Abstract	xiii
Überblick	xv
Acknowledgements	xvii
Conventions	xix
1 Introduction	1
1.1 Structure of thesis	3
2 Foundation	5
2.1 Data and Representation	5
2.1.1 Types of Data	5
2.2 Machine Learning	7
2.2.1 Supervised vs. Unsupervised Learning	8
2.2.2 Deep Learning	8
2.2.3 Neural Network	9

	Training of Neural Network	12
2.3	Natural Language Processing	13
2.3.1	Named Entity Recognition	13
2.3.2	spaCy	14
2.3.3	Word2Vec	18
	Learning Word Vectors	19
2.3.4	Transformers	19
	Pre-trained Models	21
2.3.5	Evaluation of Machine Learning Models	23
	Confusion Matrix	23
	F1-Score	25
3	Related work	27
4	Concept	31
4.1	Conceptual Design	32
4.2	Data Scraping	33
4.3	Data Preprocessing	37
	Tokenization	37
	Stop word removal	38
	Token Normalization	38
4.3.1	Applying Preprocessing	39

Customized Pre-Processing for Huggingface	42
Customized Pre-Processing for Huggingface	42
4.4 Training	46
5 Evaluation	51
5.1 Named Entity Recognition	52
5.1.1 Quality Metrics	53
5.1.2 Evaluating in depth: The problem areas	57
Evaluating the generative data set approach	62
5.1.3 Optimization possibilities	63
5.1.4 GUI for interactive optimization	66
The technical process of the GUI	69
6 Future work	71
7 Conclusion	75
7.1 Overview	75
A Computational Metrics before optimization	79
B Computational Metrics after optimization	81
Bibliography	83

List of Figures

1.1	Instructables.com and the search for bird-houses.	2
2.1	Types of Structured Data	6
2.2	digits ML	7
2.3	Neural Network Structure	9
2.4	Weighting and Bias	11
2.5	NER Example	14
2.6	Word2Vec Representation	18
2.7	Transformer Architecture	22
2.8	Confusion Matrix	23
4.1	Pipeline Overview	33
4.2	Search on Instructables for birdhouses and browsers inspection tool.	35
5.1	Confusion Matrices before optimization.	55
5.2	The error classes.	61

5.3	Visualization Tool	64
5.4	GUI for optimization	67
5.5	Adding an material with the GUI.	67
5.6	Deleting the marking with the GUI.	68
A.1	Confusion Matrices before optimization. . .	80
B.1	Confusion Matrices after optimization. . . .	82

List of Tables

2.1	Transition sequence for Mark Christy George visited HCI with the Stack-LSTM model. . .	17
5.1	Computational metrics of all three models . .	54
A.1	Computational metrics before optimization .	79
B.1	Computational metrics after optimization . .	81

Abstract

DIY tutorial platforms like Instructables and Thingiverse often contain multiple tutorials of similar projects, which only differ slightly in the utilized materials and tools. Users who look up tutorials have limited possibilities to filter the given tutorials aside from reading the titles and skimming the tutorial's content. However, Makers tend to explore tutorials looking for certain materials or tools, and those limited filter possibilities can be frustrating for the user. Thus, extended filtering on characteristics such as materials or tooling will benefit the user. On these platforms, there is a large number of existing tutorials that accommodate information such as what materials or tools are used. Manually extracting such information from the tutorials can become very cumbersome as the community and the number of newly created tutorials are growing.

In recent years, NLP has gained a lot of attention, and with it, the techniques for understanding textual data have evolved, which we can use for the problem mentioned above. In our work, we will explore the named entity recognition technology (NER). We will develop a pipeline that will allow us to create such NER models very efficiently. Moreover, we will also use this pipeline to develop models that can extract materials from these tutorials. This pipeline provides a foundation for building models based on different neural network architectures. We will form and compare three different models. We will build a convolutional neural network with and without word embedding and a transformer model, specifically the BERT model. The result shows that the transformer model performs the best. We will analyze and address the errors of the models. Furthermore, we will conceptualize a tool that platform users can use interactively to improve the models. In addition, we will talk about how our results benefit the user and what other research questions can be derived from the results for HCI.

Überblick

Auf Tutorial-Plattformen wie Instructables und Thingiverse finden sich oft mehrere Anleitungen ähnlicher Projekte, die sich nur in den verwendeten Materialien und Werkzeugen minimal unterscheiden. Nutzer, die nach Tutorials suchen, haben nur begrenzte Möglichkeiten, die gegebenen Tutorials zu filtern, abgesehen vom Lesen des Titels und dem Überfliegen des Inhalts des Tutorials. Jedoch neigen Maker dazu, Tutorials nach bestimmten Materialien oder Werkzeugen zu durchsuchen, sodass diese begrenzten Filtermöglichkeiten für den Nutzer frustrierend sein können. Daher ist eine erweiterte Filterung nach Merkmalen wie Materialien oder Werkzeugen für den Benutzer von Vorteil. Auf diesen Plattformen gibt es eine große Anzahl von Tutorials, die Informationen über die verwendeten Materialien oder Werkzeuge beinhalten. Die manuelle Extraktion solcher Informationen aus den Tutorien kann sehr mühsam werden, da die Community und die Anzahl der neu erstellten Tutorien immer größer werden.

In den letzten Jahren hat NLP viel Aufmerksamkeit erlangt, und damit haben sich auch die Techniken zum Verstehen von textuellen Daten entwickelt, die wir für das oben genannte Problem nutzen können. In unserer Arbeit werden wir die Technologie der Named Entity Recognition (NER) untersuchen. Wir werden eine Pipeline entwickeln, mit der wir solche NER-Modelle sehr effizient erstellen können. Darüber hinaus werden wir diese Pipeline auch zur Entwicklung von Modellen verwenden, die Materialien aus diesen Tutorials extrahieren können. Diese Pipeline bietet eine Grundlage für den Erstellen von Modellen, die auf verschiedenen neuronalen Netzwerkarchitekturen basieren. Wir werden drei verschiedene Modelle bilden und vergleichen. Hierzu werden wir ein Gefaltetes Neuronales Netzwerk mit und ohne Worteinbettung und ein Transformer-Modell, nämlich das BERT-Modell, erstellen. Das Ergebnis zeigt, dass das Transformer-Modell am besten abschneidet. Wir werden die Fehler der Modelle analysieren und beheben. Darüber hinaus werden wir ein Tool konzipieren, mit dem Plattformnutzer interaktiv die Modelle verbessern können. Außerdem werden wir darüber sprechen, wie unsere Ergebnisse dem Nutzer zugutekommen und welche weiteren Forschungsfragen sich aus den Ergebnissen für HCI ableiten lassen.

Acknowledgements

First of all, I would like to thank Prof. Dr. Jan Borchers and PD Dr. Ralf Klamma for examining this thesis. Furthermore, I would like to thank my advisor Marcel Lahaye, who supported me through the whole Thesis with his invaluable patience and feedback. Additionally, this endeavor would not have been possible without my friends and family, for their editing help, late-night feedback sessions, and moral support.

Conventions

Throughout this thesis we use the following conventions.

Text conventions

Definitions of technical terms or short excursus are set off in coloured boxes.

EXCURSUS:

Excursus are detailed discussions of a particular point in a book, usually in an appendix, or digressions in a written text.

Definition:
Excursus

The whole thesis is written in Canadian English.

For the use of the first person, the pronoun "we" is used.

Chapter 1

Introduction

The rise of Do-it-yourself (DIY) culture in the past decade has brought hobbyists and designers together who now share the ideals of personal fabrication and customization of handcrafted projects. The movement has been supported by online communities that help democratize the making process [Tseng and Resnick [2014]]. This also induced a large audience for creators, who can share their skills and knowledge in a step-by-step instructional tutorial. These tutorials are vital for sharing knowledge, learning from others, and exchanging with the community through commenting and asking questions [Glaser et al. [1968]]. Moreover, this emergence of DIY tutorials also disrupted the way how design projects are produced and especially who produces them now. A decentralization of design and production evolved within these practices, empowering amateurs and hobbyists to become designers and producers by active participation in these communities [Wakkary et al. [2015]]. A tutorial is filled with a lot of information and explains the components, tools, and processes required to do a DIY project. However, the overview of all these tutorials, which are very rich in information, is very limited available for a user. For this, we will devote ourselves to the platform Instructables in order to understand the problem in more detail.

The usage of DIY platforms and the need to extract information out of these tutorials.

Instructables is a popular DIY community for “passionate people [to] share what they do and how they do it, and

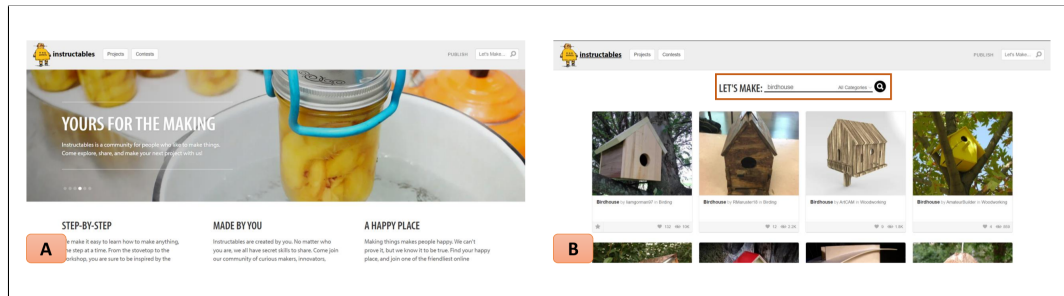


Figure 1.1: Two snapshots of the website *instructables.com*. (A) The starting page, with some general information of *instructables* and (B) a search on birdhouses on *instructables*, which will list all the tutorials related to birdhouses.

Example of limitation
based on
instructables.

learn from and collaborate with others [ins [2022]]. When we enter the website, we will get some information about what *Instructables* is and suggestions for recently created DIY tutorials. You can use the search bar as a filter option to find a specific domain or tutorial and limit the listed tutorials to the search itself (see figure 1.1 (A)). A search for birdhouses shows us several tutorials that are specifically dedicated to the creation of birdhouses. What you can observe in figure 1.1 (B) is that the elements only list a title image, title, and a few meta information, such as the number of likes and number of views. But we as readers cannot grasp information such as what materials are used or what kind of tools are necessary for building such a birdhouse. This limitation may make it difficult for the user to find individual tutorials that they find relevant. This relevance can be justified by the availability of materials to rebuild such a tutorial or to replace materials with others. In a survey on Representing and Appropriating DIY Projects Online by Teng et al. [Tseng and Resnick [2014]], one question was about the ways of using online platforms. The number one reason for using the forum is to get ideas for a project. Learning a particular technique and looking for projects to recreate were also crucial for many survey participants. This showcases that many users can benefit from tools such as filtering and searching and some more information that are only available within each tutorial.

The need for NLP to
extract materials
from DIY tutorials.

One way of achieving this information retrieval is with the help of natural language processing (NLP). Therefore, we will primarily use the technology of Named Entity Recog-

nition (NER) to extract keywords, which would be extracting materials out of the DIY tutorials. Named Entity Recognition (NER) aims to detect chunks in a sentence representing an entity and assign a class to that entity [Syed and Chung [2021]]. The task will be to create models that allow such extractions and make them perform well, especially for DIY projects documented in online platforms like Instructables. Thus, the present work will address the question of whether modern neural networks and appropriately trained models can provide added value for the development of this extraction task for specific domains. For this purpose, we will restrict ourselves to the domain of birdhouses on the platform Instructables and develop three different models to achieve the extraction of materials out of the tutorials. The approach is generalized and is not only restricted to materials. The approach can be used to create models for different kinds of categories in various types of domains.

1.1 Structure of thesis

The rest of the thesis is structured as follows: Chapter 2 will cover some related work, starting with the research on DIY platforms, which shows how users use this platform and what difficulties are hidden in the documentation. Here, we will focus on the platform's usage and reaffirm the benefits of information extraction. We will also discuss related work that deals with NLP and models that deal specifically with Named Entity Recognition. Moreover, we will also discuss what problems NLP is exposed to and how to collaborate HCI with NLP to improve NLP tasks and dig deeper into a paper that focuses on the collaboration of NLP and HCI. Chapter 3 will explain the fundamental knowledge of the used technologies. Chapter 4 describes the conceptual approach and the implementation of the system. A data pipeline will be built, which can be used to utilize the approach and build other models for different use cases. This work will explain a more detailed example of the pipeline and the creation of the models using the domain of birdhouses within the Instructables platform. The next chapter will cover the evaluation of these

created models, and we will address the quality measures described in the fundamentals chapter. In addition, the review will also go beyond purely computational measures and evaluate the results for impact on the user. In chapter 6, we will address the advantages of these built models for HCI and how we can use this extracted information to generate value for the user of these DIY platforms. Finally, Chapter 7 concludes the thesis with a summary and a discussion of the thesis.

Chapter 2

Foundation

This chapter explains the principles and technologies used in this thesis to extract the materials used from Maker projects.

2.1 Data and Representation

This section provides an overview of data and their attribute representation in the context of machine learning. We will focus on a generic representation approach for machine learning algorithms, specifically for the use of text representations.

2.1.1 Types of Data

In the approach section, we will elaborate on how the data for our task will be extracted. Therefore, we will now provide a general idea of what type of data exist and how we can utilize it. In general, data can be distinguished between structured data and unstructured data. When we refer to structured data, we are talking about numerical data, as well as categorical data. Numerical data can be, for example, the age, time or temperature and are mostly used as a form of measurement. Categorical data, on the other hand,

Explanation of what types of structured data exist.

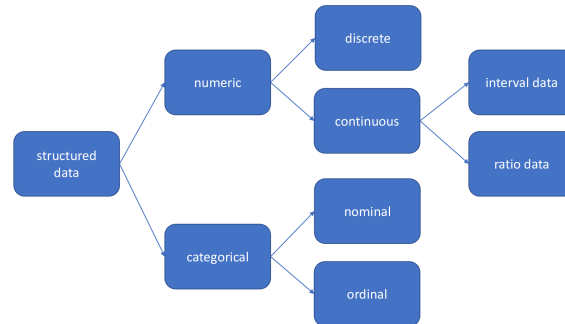


Figure 2.1: Visualization of types of structured data.

is data such as gender, color, or country, which is classified into groups or categories with the aid of names or labels. In figure 2.1 you can see that categorical data can be divided into nominal data and ordinal data. Nominal data is used to label the data without providing any quantitative value. For example, the data value wood can be categorized as nominal data. Ordinal data is very similar with the difference that it can be ranked, ordered, or assigned a rating scale, e.g. school grades. Numerical data can be divided into discrete, as well as continuous data. Discrete data are either be countably finite or countably infinite, meaning the elements have a one-to-one mapping with natural numbers. Simple examples would be age or the number of students in a lecture hall. Continuous data on the other hand, is a numerical data type with uncountable elements. They are represented by the real numbers. Continuous data can be further divided into interval data and ratio data. Interval numbers have no absolute zero and are usually not multiplied or divided. For example, the temperature in Celsius. Just like interval numbers, ratio data have units of the same size, but this time on a scalar with an absolute zero.

Explanation of what unstructured data is.

The other type of data are unstructured data. This data can be text, audio, video, or images. We talk about unstructured data when they are not defined or have a structure. Humans can work very efficiently with such data, but most machines and algorithms are designed on structured data.

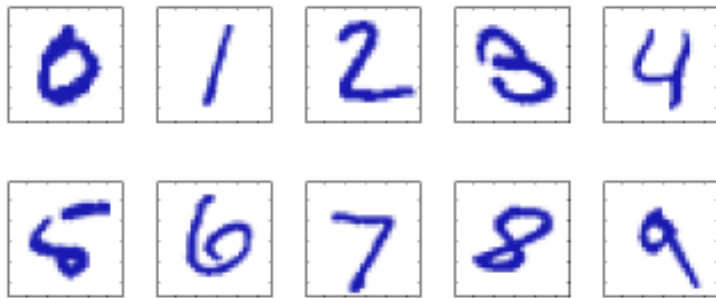


Figure 2.2: Handwritten digits presented by Bishop [[Bishop and Nasrabadi, 2006]], originally taken from US zip code.

It is possible to represent this data mathematically, but it does not change the fact that it is still unstructured. To cope with this, Deep Learning comes to our rescue.

2.2 Machine Learning

Machine learning is a subarea of artificial intelligence. With the aid of machine learning, it is possible to recognize patterns and regularities on the basis of existing databases and algorithms as well as to develop solutions. In order for the software to learn and find solutions independently, prior action by individuals is usually necessary. For example, the models must first be supplied with the data and algorithms relevant for learning. A preprocessing of the data is also usually necessary to achieve good results. [Raschka and Mirjalili [2017]]

General description
of Machine Learning.

To get a feeling for machine learning in general, we will take a look at an example of machine learning known as the handwritten numbers problem. The problem is often used in the context of machine learning, but we will stick to the version of Bishop [[Bishop and Nasrabadi, 2006]].

Figure 2.2 represents a subset of the input data, a number of hand-written digits. Each digit is represented as a 28×28

Machine Learning explained with an example.

pixel images, i.e. it can be represented as a flat vector X consisting of 784 numbers. In this case, a binary scale is used, whereas 0 represents white and 1 represents blue or real numbers which are capable of representing more color nuances. Now apart from the X vector, we also need another variable, which tells us which of the digits $0, \dots, 9$ our vector corresponds to. This variable is often called $f^*(x)$, $f(x)$ or simply y and is also called label or target variable. The task of machine learning now is, to use a learning algorithm to learn a so-called predictor or model $f()$. To train this model, we use a large number of hand-written digits X and their actual label y , i.e. we have a large set $(X_1, y_1), \dots, (X_n, y_n)$. The trained model $f()$ takes a hand-written digit representation X , which it has never seen, as input and will predict which digit is most likely meant to be.

2.2.1 Supervised vs. Unsupervised Learning

Explanation of supervised and unsupervised learning.

The handwritten numbers problem described above is form of supervised learning. This requires predefined data that is labeled. Also, in most cases, this labeled data must be created manually to generate qualitative training data. The other type of learning is unsupervised learning. In this case the training data has no label and we usually do not make predictions on unlabeled data points but rather want to classify a set of unpredicted data points [Bishop and Nasrabadi, 2006].

2.2.2 Deep Learning

Overview of Deep Learning

Deep Learning is a subfield of machine learning and thus also a subfield of artificial intelligence. In Deep Learning, neural networks are used to analyze and process large amounts of data [[Mehlig, 2021]]. The algorithms used in Deep Learning to process these large amounts of data are based on the function of the human brain and are modeled on it. Especially in recent years, there have been major breakthroughs in image recognition, text understanding, and decision problems. [[van der Aalst, 2021]]

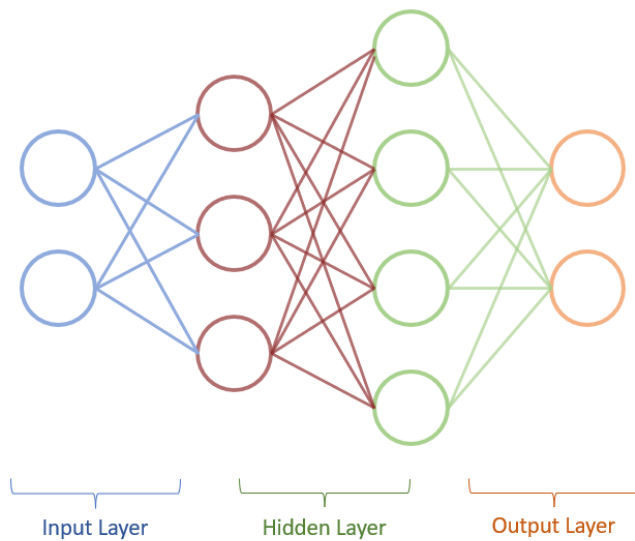


Figure 2.3: Neural network example structure: Consisting of an input layer with 2 nodes (neurons), two hidden layers with 3 and 4 nodes (neurons) respectively and an output layer consisting of 2 nodes (neurons).

2.2.3 Neural Network

Neural networks, often called artificial neural networks (ANN), are a set of algorithms that mimic the way the human brain works. Similar to its biological counterpart, an ANN consists of many neurons that are interconnected to process and transmit data among themselves. Artificial neural networks can learn by restructuring and adapting these neuron-to-neuron connections. [[Mehlig, 2021]]

Figure 2.3 shows a typical neural network. Such a neural network usually consists of an input layer, the hidden layer, and an output layer. Each of these layers can have any number of nodes (neurons). The hidden layer can also consist of several layers that follow each other. The neural network shown in Figure 2.3 has an Input layer consisting of 2 nodes, which are drawn in blue. The hidden layer in this example consists of 2 separate layers, one with 3 nodes drawn in brown and the other with 4 nodes drawn in green. Finally, the output layer follows, which in this example con-

Introduction into
neural network.

sists of the two orange nodes. It should be noted that the number of nodes in the input and output layers does not have to be identical. The number of nodes in a layer depends entirely on what is required of the neural network.

Explanation of the layers that a neural network consist of.

The hidden layers are optional and turn out to be more difficult to calculate in the sense of how many of the hidden layers are needed and its optimal number of layers and nodes can often only be determined by trial and error. Some approaches and recommendations do exist, e.g., that the number of nodes should be less than $2 \times \text{\#NodesInput-Layer}$. Another commonly used formula for determining a suitable number of nodes the following: $(2/3 \times \text{\#NodeInput}) + \text{\#NodeOutput}$.

However, as mentioned earlier, these are only recommendations. Often these formulas are only used as a starting point to test the performance of the models, and then these parameters are further optimized to improve the result.

Explanation on neural networks bias.

Apart from the different layers and nodes that make up a neural network, the second most important feature of a neural network is the paths that connect the nodes in an ordinary NN, a node from one layer is connected to all nodes of the following layer. These paths usually have a weighting. In addition, there is a so-called bias, for which different notations exist. Bias in Neural Networks can be thought of as analogous to the role of a constant in a linear function, whereby the line is effectively transposed by the constant value. [[Mehlig, 2021]] [[Bishop and Nasrabadi, 2006]] Often the bias is noted directly on a node, so that it is applied to all incoming paths of this node. Another notation is the one used in Figure 2.4, where the bias is noted on the path as well as the weighting. However, it is important to note that all paths to a single node have the same bias. So, it is not possible to have a different bias of two separate paths to the same node.

Let's take a closer look at Figure 2.4. We have mapped an input node connected to two nodes in a hidden layer. The upper path has a weight of 0.64 and a bias of 2.12. If there were another input node, it would also have a path to that very node in the hidden layer. The path would also have a weight and a bias. The bias must have a bias of 2.12 as well. However, the weighting can have any other value.

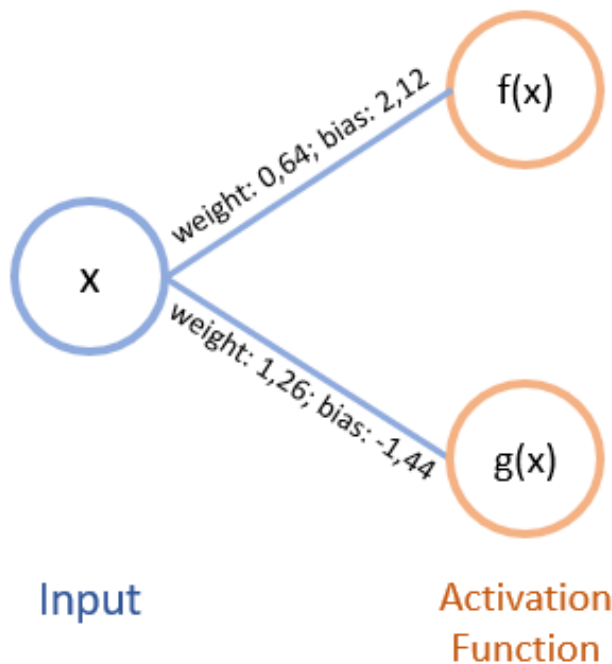


Figure 2.4: Example connection between Input Layer nodes and 2 nodes in a Hidden Layer. Weighting and bias is shown.

Each node in a hidden layer has a so-called activation function [[Bishop and Nasrabadi, 2006]]. This is always applied to any value that arrives at the node before the value is passed on to the next node. There are many different activation functions that are applied to neural networks, including the following: Sigmoid, Tanh, ReLU, softplus.

Description of neural networks activation function.

Initially, sigmoid was very often used as an activation function to map the incoming numbers between -1 and 1, but it has since been established that using ReLU as an activation function is more efficient and gives a better result. Which activation function is used in the end depends on the neural networks use case, and this is often determined by testing. [[van der Aalst, 2021]]

Training of Neural Network

Explanation of supervised learning in neural network.

We covered the topic of supervised and unsupervised learning, which differ in whether you have labeled data or not and you execute the training with that labeled data or without that data. The neural network can also be used to train a model using both methods. Since we will only deal with supervised learning later on, I will now discuss how supervised training of neural networks operates. In general, training a neural network can also be considered as solving an optimization problem. The goal is to optimize the weights of the individual paths in such a way that the results of the model improve with each training step and approach the desired behavior [[van der Aalst, 2021]] [[Mehlig, 2021]].

Introduction of error-based learning.

In supervised training, the model is provided with input training data as well as the corresponding output training data during the training process, so that the model can update the weights of the individual paths between the neurons accordingly. That is, the network can process the input passed in and calculate a prediction for that input. This prediction can then be compared to the actual output from the network. An error is calculated, which is then propagated from the output node of the network back through the entire network. That is why this type of technique belongs to the area of error-based learning. In this process, the weights used for the calculated prediction are updated according to the error, so that the error is minimized. This is also called minimizing the loss function. This function can be defined differently depending on the application, but often the root mean square error function is used as loss function, but this can be adapted by the developer. [[Bishop and Nasrabadi, 2006]]

Having validation set to improve generalizing the outcomes.

This process just described is repeated many times during training, and thus the weighting of each path between the different nodes is further adjusted with each repetition. A single repetition is also known as an epoch. However, a model should also not be explicitly trained for too long on a certain training set, in the sense of too many epochs, because otherwise there is a danger of overfitting. This means

that a network learns the existing training set extremely well, but then performs poorly on previously unseen input data because it is too fixated on the training data. This can be improved by using a validation set in order to estimate how well your model has been trained. The validation set is like a test set during training such that the model can validate its current training results on an unseen data set. Conclusively, we can say that training a network is also not a trivial process and there is no single best approach for all networks, but one often requires a lot of experimentation based on the training dataset itself, the network architecture and the use case, to find an optimal training for a specific network.

2.3 Natural Language Processing

Natural language processing (NLP) is the process by which a computer processes natural language text and extracts and understands the semantic information contained in the text [[Chowdhary, 2020]]. However, in order for a computer or any machine to process a natural language text, some preprocessing steps must first take place. As described in the subchapter 2.1.1, there is structured and unstructured data. The natural language is unstructured data and this data must first be converted into a certain structure for the training of the models. NLP is used, for example, in machine translation, virtual assistants, spam detection, or sentiment analysis [[Wolf, 2019]]. This work focuses on the method of extracting used materials from tutorials, such as Instructables which then can be used in other research areas of HCI. Extracting specific keywords from an unstructured corpus is also a use case in NLP and is called Named Entity Recognition (NER).

Introduction into natural language processing.

2.3.1 Named Entity Recognition

Named entity recognition (NER), sometimes referred to as entity chunking, extraction, or identification, is a natural language processing (NLP) technique that automatically

Named Entity Recognition is a NLP technique.

This ^{Time} **winter** I will write my thesis in ^{Location} **Aachen**.

Figure 2.5: The NER model recognized winter and Aachen as entities and categorized winter as Time and Aachen as Location.

identifies named entities in a text and classifies them into predefined categories. Any word or group of words that consistently refers to the same item is considered an entity. People, organizations, locations, times, amounts, monetary values, percentages, and other entities are examples of categories. [Eisenstein [2019]]

Explanation of the NER example.

In figure 2.5, we identified two types of entities. One of them is a "Time": winter and the other category that was found is "Location": Aachen. Training such entities requires using the method of supervised learning, whereas we train the model via labeled data. So, we need training data set, which consists of a lot of text and their corresponding labels. For example, we could train a model only for the categories "Time" and "Location". Our training data would be a lot of text were we specifically mark the instances that belong to the category of "Time" or "Location". We would use a NN or another architecture that allows to train a NER and train that network or model on that given data.

2.3.2 spaCy

Introduction into spaCy.

SpaCy is a library for advanced Natural Language Processing in Python and Cython. It comes with pretrained pipelines for specific NLP task and features state-of-the-art NN models. Moreover, it is commercial open-source software, released under the MIT license. [[Honnibal et al., 2020]]

SpaCy has its own deep learning library called thinc used under the hood for different NLP models. For most tasks,

spaCy uses a deep neural network based on CNN. Specifically for Named Entity Recognition, spaCy uses:

- **A transition based approach**

A transition based approach inspired by shift-reduce parsers, which is described in the paper Neural Architectures for Named Entity Recognition by Lample et al. [[Lample et al., 2016]]. The model relies on stack data structure in order to incrementally construct chunks and labels. It uses two stacks, the output and the stack representing, respectively, completed chunks and scratch space and a buffer that contains the words that have yet to be processed. Then there is a transition inventory, which consists of SHIFT, REDUCE and OUT, which are used as action to cooperate with the stack architecture. The SHIFT transition moves a word from the buffer to the stack, the OUT transition moves a word from the buffer directly into the output stack. The REDUCE transition creates a chunk and labels it accordingly and moves the entity with its label to the output stack. The algorithm is complete if the buffer and stack are completely empty [[Lample et al., 2016]]. Consider table 2.1, which shows the sequence of operations required to process the sentence Mark Christy George visited HCI. At the starting point both stack are empty and only the buffer contains all the words of that sentence. At the beginning, the first three words are popped and pushed into the stack. After that these three entities are considered as a chunk and are labeled with label PER and the chunk will be moved into the output stack. Thereafter, the word visited will be moved into the output stack immediately with the transition OUT. The last two steps will shift the word HCI into stack and then it will be labeled with ORGANIZATION and lastly, pushed into the output stack. Since the buffer and the stack are empty, the algorithm terminates. Now, the question is how the model decides on which transition is performed in each step. This is done with a statistical model which predicts each transition in each step. So, we have to ask on how our neural network is structured?

SpaCy uses a transition based approach borrowed from shift-reduce parsers and a framework that is called Embed, Encode, Attend, Predict

- **A framework that's called "Embed. Encode. Attend. Predict"**

The underlying framework that is used in spaCy is "Embed, Encode, Attend, and Predict". With Embed, words are embedded using a Bloom filter, which means that word hashes are kept as keys in the embedding dictionary, instead of the word itself. This maintains a more compact embeddings dictionary, with words potentially colliding and ending up with the same vector representations. Encode computes a representation that is called sentence matrix, where each row represents the meaning of each token in the context of the rest of the sentence. To allow it to be transferred to a typical feed-forward network for prediction, the attend step converts the matrix representation created by the encode step into a single vector. Once the text or pair of texts has been reduced into a single vector, we can learn the target representation and do structured prediction, by using the network as the controller of a state machine such as a transition-based parser. [[Honnibal et al., 2020]]

Transition	Output	Stack	Buffer	Segment
	[]	[]	[Mark, Christy, George, visited, HCI]	
SHIFT	[]	[Mark	[Christy, George, visited, HCI]	
SHIFT	[]	[Mark, Christy]	[George, visited, HCI]	
SHIFT	[]	[Mark, Christy, George]	[visited, HCI]	
REDUCE	[(Mark Christy George)-PER]	[]	[visited, HCI]	(Mark Christy George)-PER
OUT	[(Mark Christy George)-PER, visited]	[]	[HCI]	
SHIFT	[(Mark Christy George)-PER, visited]	[HCI]	[]	
REDUCE	[(Mark Christy George)-PER, visited, (HCI)-ORG]	[]	[]	(HCI)-ORG

Table 2.1: Transition sequence for Mark Christy George visited HCI with the Stack-LSTM model.

		King	Queen	Man	Woman
Royalty		0,99	0,99	0,02	0,03
Masculinity		0,99	0,02	0,97	0,07
Femineity		0,05	0,93	0,03	0,99
Wealth		0,93	0,89	0,5	0,51
⋮	⋮	⋮	⋮	⋮	⋮

Figure 2.6: Word2Vec Structure: Example shows how the vector would look for the basic king, queen, man and woman universum.

2.3.3 Word2Vec

Word vectors used to understand the context in a mathematical way.

Word vectors are mathematical representations of words in a multi-dimensional space. With this machine learning models can understand words and context. The word vector in its simplest form is a simple 1-to-N (one-hot) encoding, where the vector in which the corresponding element is set to one, and all other elements are zero. For example if we consider that our vocabulary has only four words with King, Queen, Man, and Woman, then Queen could be represented as (0,1,0,0). With the introduction of word2vec a distributed representation of a word is used. Here, most words are represented with hundreds or even thousands of dimensions. Each word is represented by a weighting distribution among those elements. The representation of a word is therefore distributed throughout all of the items in the vector rather than being mapped one to one to a word, and each element in the vector contributes to the definition of a number of words. In figure 2.6 we can see how it would look like for the example of King, Queen, Men, Woman. [[Mikolov et al., 2013a]][[Mikolov et al., 2013b]][[Mikolov et al., 2013c]]

Learning Word Vectors

Mikolov et al. introduced a practical way to learn high dimensional word vectors on a large amount of data [[Mikolov et al., 2013a]]. Two architectures are proposed, the Continuous Bag-of-Words model (CBOW), and the Continuous Skip-gram model. The contexts forms the input layer of the neural network. Each word is encoded in one-hot form, so as a simple 1-to-N mapping. So with a vocabulary size of N every input is N long, with the corresponding element set to one and the rest to zeros. The goal of training is to increase, given the input context words and weights, the conditional probability of observing the actual output word (the focus word). The skip-gram model is the opposite of the CBOW model. It is constructed with the focus word as the single input vector, and the target context words are now at the output layer. Here the goal is to minimize the summed prediction error across all context words in the output layer.

Word vectors learn the context with a given window size.

2.3.4 Transformers

Over the years, research continues on newer neural models. In the field of NLP, Transformer models were introduced in 2017 in the paper 'Attention Is All You Need' [Vaswani et al., 2017]. These neural networks are models that can be applied particularly effectively to common NLP tasks. [Vaswani et al., 2017] The special feature of the Transformer models is the novel architecture, which although also similar to the Sequence-to-Sequence (Seq2Seq) architecture [Sutskever et al., 2014], has the task of transforming input sequences while solving a previously common problem with Seq2Seq architectures.

Transformers model were introduced with the paper Attention is all you need.

The Seq2Seq model has already been used in various architectures, especially in the older and widely used LSTM (Long Short Term Memory) models. As the name reflects, these models built on such an architecture are designed to transform the sequence of elements (in NLP usually a sequence of words) into another sequence. Hence, such models are often used for tasks such as translating from

Seq2Seq model has a similar architecture as the Transformer model.

one language to another. LSTM models have been the preferred models for solving these tasks for a long time, because LSTM models can filter and store important information from a sequence while "forgetting" unimportant information in the case of sequence-dependent data. [Sutskever et al., 2014]

Architectural design which includes an encoder and a decoder.

Seq2Seq models consist of an encoder block and a decoder block, where the encoder receives an input sequence and transforms it into a multidimensional space by word embedding. The size of the dimensions varies from model architecture to model architecture. The resulting vector is decoded in the decoder block to an output sequence. This output is then usually changed from the input, for example into a different language or a particular sentence structure. This output is then usually changed from the input, for example into a different language or a particular sentence structure. [Sutskever et al., 2014]

Attention mechanism as main driver for Transformer models.

Before the introduction of Transformer models, two single LSTM models were often used for these tasks, acting as the encoder and decoder of a Seq2Seq architecture. The big advance now is the introduction of the Transformer models. The Transformer architecture is designed to convert an input sequence into another new sequence, just like the Seq2Seq architecture. In contrast to LSTM or GRU models, however, Recurrent Neural Networks (RNN) are not used, but an approach called 'Attention Mechanism (AM)', hence the name of the paper published on this topic 'Attention Is All You Need' [Vaswani et al., 2017].

Explanation of Attention Mechanism.

First, let's clarify what is meant by Attention. When an input sequence is passed, the AM decides in each step which sequence parts are perceived as important, this is a similar process that a human unconsciously performs when reading a text sequence. When reading a text, humans always focus on the word they are reading, but at the same time the brain remembers the important keywords from the text to generate context. The AM works similarly for a passed sequence and creates an attention matrix at word embedding (i.e., transforming the sequence into a vector). The Attention Matrix calculates a value for each token in the sequence, which indicates how important this token is for

the whole sequence. It also calculates the position of the token in the sequence. This way, AM can be used to create a context for an input sequence, which ensures that the sequences are transformed into a multidimensional space much more accurately and contextually.

This very principle is used by the Transformer models in their respective encoder and decoder block and therefore distinguishes them from the usual Seq2Seq models which are based on recurrent networks. In the presented paper, it was shown and proved that the Transformer architecture with the AM could achieve better results in many NLP tasks than the previously used Seq2Seq models based on RNN. [Vaswani et al., 2017]

In figure 2.7, the transformer architecture model is illustrated, which was initially presented in the paper of 'Attention Is All You Need'. It can be seen that the architecture consists of two blocks, as with the Seq2Seq architecture, the Transformer architecture consists of a decoder (left) and encoder (right) block. The gray area of the two areas indicates that both blocks are of modular design. These modules can be arranged in sequence as often as desired and consist mainly of multi-head attention and feed-forward layers. Moreover, during embedding, which is the transformation in an N-dimensional space, an additional positional encoding is generated for the input sequence as well as for the output sequence. This is added to the generated vector to encode the position of each element in the sequence. For a more detailed explanation of each of the multi-head attention and feed-forward levels, the reader is referred to the corresponding paper [Vaswani et al., 2017].

Explanation of the Transformer architecture.

Pre-trained Models

Since the release of the Transformer architecture, a number of new models based on the Transformer architecture have been developed and presented. For this purpose, these models have been trained on huge data sets. Especially with respect to NLP, this approach has been used to develop models that understand a language in a better

Training Transformer model from scratch is a highly expensive task. Fine-tuning pre-trained model was deduced by it.

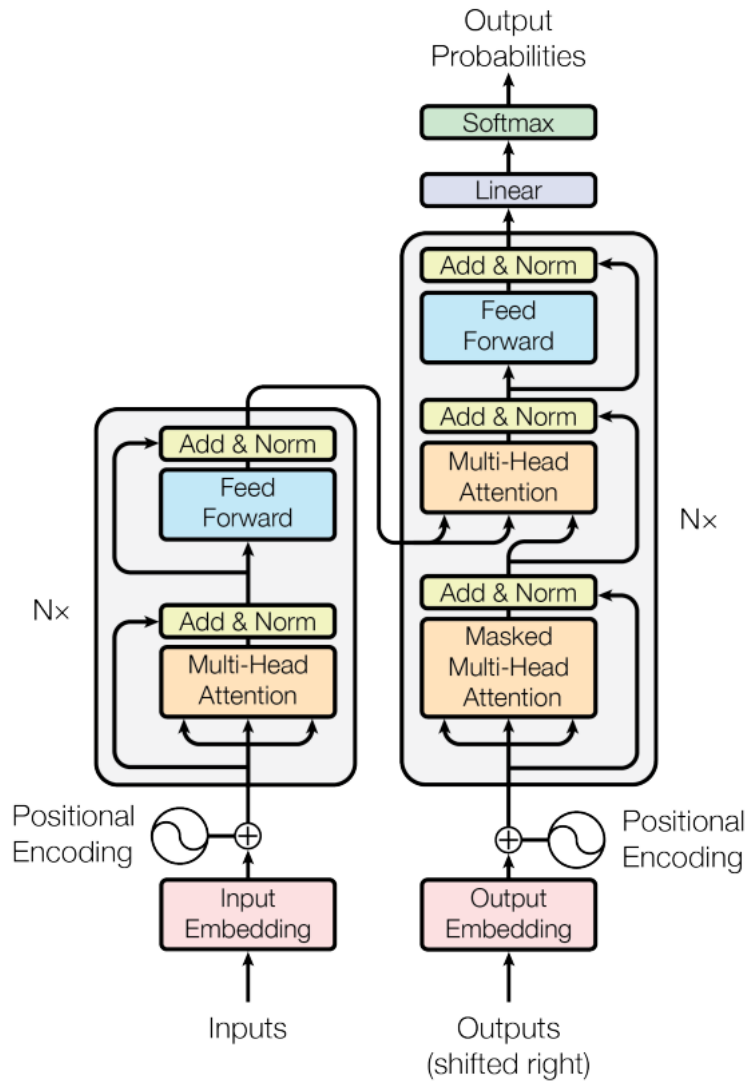


Figure 2.7: Transformer Model Architecture [Vaswani et al., 2017].

way. To do this, the model is trained beforehand on a huge amount of data, for example the entire Wikipedia dump. In this way, the model learns to understand the language. The model can then be further trained on specific tasks using a task-specific training set. This procedure is also called fine tuning.

A 2x2 confusion matrix with 'True Label' on the y-axis and 'Predicted Label' on the x-axis. The y-axis has labels 'MATERIAL' and 'O'. The x-axis has labels 'MATERIAL' and 'O'. The cells contain: (MATERIAL, MATERIAL) is 'TP' (light blue), (MATERIAL, O) is 'FN' (light blue), (O, MATERIAL) is 'FP' (light blue), and (O, O) is 'TN' (dark blue).

True Label	Predicted Label	
	MATERIAL	O
MATERIAL	TP	FN
O	FP	TN

Figure 2.8: Confusion Matrix depicting all four classes. True Positive, True Negative, False Positive, False Negative.

2.3.5 Evaluation of Machine Learning Models

In this section, we will list out the evaluation metrics that will be used to evaluate the performance of our created models. We will start by explaining the confusion matrix.

Confusion Matrix

The confusion matrix is a performance measurement for machine learning classification problems where the output classes can be two or more. It tries to visualize the outcomes in a matrix form where the rows represent the predicted value, and the columns represent the actual value. With this, additional metrics for performance measurement can be calculated such as precision, recall, or f1-score. Let's first focus on the confusion matrix with two classes as output.

In figure 2.8, we can see a confusion matrix of the output class. The matrix depicts four different combinations of predicted and actual values. So, we compare the predicted

Description of confusion matrix.

results of the model being evaluated with the actual values from the test data. The true positive value describes that the model predicted the value to be true or positive and the actual value is also true. The true negative describes that our model predicted the value to be false or negative and the actual value is also negative. False positive means that the model predicted a positive value but which is incorrect and the similar for false negative, where the model predicted the outcome to be negative or false but the actual value is positive. The latter two combination are also called the misclassification rate.

As stated before, with these values we can easily calculate the Accuracy, Precision, Recall, and F1-Score.

ACCURACY:

Definition:
Accuracy

$$Accuracy = \frac{TN + TP}{TN + FP + TP + FN}$$

Description of
accuracy.

Accuracy represents the number of correctly classified data instances over the total number of data instances. We try to reach a high number close to one, so that our misclassification rate, the incorrectly identified entities, are close to 0. Anyhow, the accuracy can be misleading. If the dataset is not balanced, so both negative and positive classes have different size, the model will predict always the class which size is bigger. This will lead to an high accuracy if the test dataset unbalanced in a similar way. But in reality our model will most likely predict only one of the classes and thus, it isn't a good model.

PRECISION:

Definition:
Precision

$$Precision = \frac{TP}{TP + FP}$$

Description of
precision.

Precision should ideally be 1 (high) for a good classifier. This means the False Positive rate should be zero. As FP increases the value of denominator becomes greater than the numerator and precision value decreases, which is un-

desired. Precision answers the following question: What proportion of positive identifications was actually correct? So, we only focus on the positive outcomes of our models and verify its correctness.

RECALL:

$$Recall = \frac{TP}{TP + FN}$$

Definition:
Recall

Recall should also be 1 (high) for a good classifier. Similar to Precision as False Negative increases the value of denominator becomes greater than the numerator and recall value decreases. The question recall is answering is what proportion of actual positives were identified correctly. Here, we focus on the actual positives instead of only what the model predicted and then compare it with the positive outcomes of the model.

Description of recall.

F1-Score

Ideally, we want both precision and recall to be one which also means that the false positive and false negative are zero. Therefore we need a metric that takes into account both precision and recall. This is where the F1-Score comes into play.

Description of
F1-Score.

F1-SCORE:

$$F1 - Score = 2 * \frac{Precision * Recall}{Precision + Recall}$$

Definition:
F1-Score

F1 score is the harmonic mean of precision and recall. F1 Score becomes 1 only when precision and recall are both 1. F1 score becomes high only when both precision and recall are high.

Chapter 3

Related work

This chapter discusses the related work, addressing the area of HCI and NLP. In doing so, we address the problems of DIY portals and give evidence that HCI can benefit from machine learning methods, such as NLP. Furthermore, we refer to papers that use different methods to enable tasks such as NER, especially in the domain-specific understanding of texts. We will also briefly address what problems NLP is exposed to and how to collaborate HCI with NLP to improve NLP tasks. For this, we will dedicate ourselves to a paper that composes methods for designing and evaluating HCI and NLP systems.

The paper of Tseng et al. [Tseng and Resnick [2014]] focuses on the methods by which project documentation is created and utilized. Therefore, they made a case study through interviews and surveys about how design documentation is created and used by authors and readers of Instructables. Until now, the documentation tools designed specifically for their unique workflow are disruptive and time-consuming. With the results' aid, they encourage HCI community members to develop tools to better support people sharing creative work online. It should be mentioned that user-generated content has become increasingly important as a form of knowledge sharing with the change of tools, materials, and techniques [Wakkary et al. [2015]]. This indicates that the amount of data in human written form is large, and drawing conclusions or gaining valuable infor-

Paper talks about the usage of platforms like Instructables.

mation from this amount of data can be a challenge for humans. One of their survey questions is how the users of Instructables use the platform. For this, the interviewees were asked to rank the following use cases: “To learn a particular technique, To look for projects I want to recreate, and Getting ideas for a project”. The results clearly show that “Getting ideas for a project” is the primary use case on Instructables, which reinforces that the user could benefit from further information about these tutorials. For example, Instructables does not offer any special filter or search options on materials or tools used in the instructions. Furthermore, the case study shows that modification and personalization are essential practices for people who recreate Instructables as they don’t necessarily have all the required materials or tools. Extracting information such as materials or tools can benefit the user. To deal with the tremendous amount of data and be able to extract these keywords for further tooling possibilities, we will be using NER, a sub-area from NLP.

A comparison of a suite of neural network models and how we will lay our focus.

We will be creating three different models using the same training data and performing evaluations on each of them. Korpusik et al. present a comparison of deep learning methods for language understanding in their paper. Here, they compare a suite of neural networks (NN) such as recurrent NN, convolutional NN, and Transformers, especially the BERT model with hand-crafted features on three semantic corpora. In general, they state that the performance improvement from eliminating examples where the model is less certain indicates that the model is more confident when its predicted tag is correct and less confident when it makes errors. Our approach will also focus on improving the training data set rather than on the data sets’ sparsity and inaccuracies. They have demonstrated that BERT outperforms prior state-of-the-art methods on three spoken language understanding tasks. Yet, this paper focuses on spoken languages and doesn’t incorporate a feedback mechanism that can improve the model. [Korpusik et al. [2019]]

The paper MenuNER by Syed et al. focuses on menu entity extraction from online user reviews for the restaurant and proposes a simple approach for NER tasks on a new

domain where a large dataset is rarely available or difficult to prepare. They are facing a similar challenge of user-generated content with its unique characteristics and use of informal language, which is typically short context, noisy, sparse, and ambiguous content [Lin et al. [2017]]. Instead of creating a neural network model such as BERT, developing such a pre-trained model for a new domain from scratch will be pretty expensive since the BERT-base model has 12 layers with roughly 110 million parameters. So, instead, they further pre-train the original off-the-shelf BERT with a limited target-domain dataset. When further pre-trained on domain-specific corpora, the BERT pre-trained model can improve performance in domain-specific tasks while maintaining good performance in general domain tasks [Rongali et al. [2020], Ma et al. [2019]]. Post-training approaches for domain adaptation are highly effective for tasks where the training dataset is limited [Rongali et al. [2020]]. Thus, we will use a similar approach of fine-tuning a given Transformer model instead of training it from scratch. [Syed and Chung [2021]]

Related work on domain-specific NER task, while using pre-trained models to further finetune it on its specific use case.

During our research, we found out that the field of NLP is still very low, regarding DIY content. The paper “Extracting terms and their relations from German texts: NLP tools for the preparation of raw material for specialized e-dictionaries” by Rösiger presents an approach of data extraction from German texts in the domain of do-it-yourself (DIY) instructions, where the objective is to extract nominal term candidates with high quality. This paper’s objective is to collect German terminological data from heterogeneous corpora. This paper, however, was published in 2015 and applies outdated methods of NLP to perform data extraction. Moreover, the focus here is to extract predicate-argument structures involving the term candidates, relate German word formation products with syntactic paraphrases, and use only a prototype to perform the extracting task. [Rösiger et al. [2015]]

Related work on NLP research on DIY content.

Many modern NLP tasks will face the challenge of data sparsity with unique fine-tuning methods by changing the parameters and using particular architectures [Alawad et al. [2017]]. Another way to improve the model performance is by improving the data sets used [Syed and Chung

Methods for designing and evaluating HCI combined with NLP systems.

[2021]]. Heuer et al. present methods for designing and evaluating HCI combined with NLP systems [Heuer and Buschek [2021]]. For this purpose, they developed practices that we consider useful in advancing research in both fields. The first method describes a user-centered NLP approach where user studies can ensure that users understand the output and the explanations of the NLP systems. The idea of user-centered NLP was inspired by an unpublished machine learning-based fake news detection. The outcomes have been excellent on paper, but the rationale was meaningless to the user [Heuer and Buschek [2021]]. Moreover, they discussed a method called Co-Creation, where involving users in designing, implementing, and evaluating interactive computing systems can yield insights into UI and interaction design and its output. Our work uses this as inspiration, and we will analyze if user involvement in the NLP task can be a possible improvement of the model's results. Moreover, we will also explore the meaningfulness behind the results. For this purpose, we will first evaluate the results using computational metrics and manually analyze the significance. We will take a part of the data and improve it with our understanding of materials and eliminate errors due to erroneous data sets or erroneous predictions. We will investigate the model properties of the finetuning and related to the models' optimizations by further training and show if such an improvement enhances the results. Once such optimization has been validated, we will create a concept that allows the user to perform such an evaluation and be involved in the training process. This concept will provide a GUI, and the user will be able to complete the improvement steps we have performed manually for our proof of concept. Validation of the user's participation and whether the user-manipulated data is as accurate as our manual corrections will need to be tested in future work.

Chapter 4

Concept

The rise of DIY culture over the past decade has allowed designers to share their ideas and skills on platforms such as Instructables or Thingiverse and share them with other designers or interested individuals. Thus, large amounts of data have accumulated in the form of a step-by-step tutorial, which becomes increasingly difficult for the reader to manage. Information such as materials or tools used in the designers' projects are not comprehensible at first glance, and filter options within a domain are also only possible very restrictive. Tooling like filtering and searching for domain-specific projects can help the reader or users who want to recreate such projects to find more relevant tutorials. This leads us to the path of Natural Language Processing, particularly Named Entity Recognition. The existing data must first be understood to enable such filter or search tooling. More precisely, specific information must be extracted from the individual tutorials to offer filtering or search options based on these information. In this work, we will deal with techniques that enable such extraction of specific data, focusing on materials that are needed or have been used for the construction such projects. Benefits that can be gained from the information of such materials have been described in the introduction chapter 1 and future work 6.

To extract material out of DIY tutorials, we will rely Named Entity Recognition.

4.1 Conceptual Design

Structure of conceptual design.

In the beginning, we will give you a general idea of the chosen approach of how we enable tackling the problem of extracting materials from the platform of Instructables. Here, we will briefly discuss the pipeline that allows us to set up and train NER models. After that, we will discuss the pipeline's elements and explain our strategies to develop each model.

Overview of the chapters procedure.

To ensure the extraction of materials from tutorials, we will turn to Named Entity Recognition. As described in the foundation chapter 2, this enables the targeted training of categories in the NER model, which can filter out words of these categories for new and unseen texts. In our case, the category material will be trained, and we will prepare the models to recognize materials from any texts. It should also be mentioned that for this work, we are limited to the domain of birdhouses, and our results are optimal for such tutorials. However, the models are trained to understand from the context if it is a material. Thus, theoretically, these models could be applied to any domain, but it remains to be shown how well they perform. Furthermore, we will build a pipeline that allows us to generate such models very quickly and efficiently. We recommend using this pipeline for different domains and, thus, training the models on specific domains. Our NER models perform particularly well when trained on a particular domain, and we will demonstrate this using our example domain of birdhouses. The pipeline will be depicted in figure 4.1:

Explanation of the three steps of the pipeline.

The first part is Data Scraping. In order to train such models, it requires data. We will use the data from Instructables by using an HTTP library and HTML parser library. After we have our data, we need to perform some pre-processing steps. Since machines can be influenced by noisy data, those that do not add value to the context but facilitate the reader's reading flow. We will also generate three models using the same data and evaluate them with the same test data. We use two libraries for the models, whereby we also adapted our pipeline with an adjusted pre-processing step. In the end, we can train our model using the built-in func-

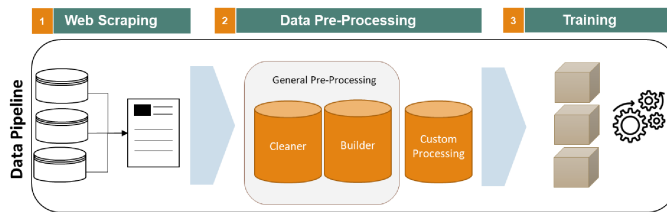


Figure 4.1: An overview of the pipeline with which we will create three models to extract information out of DIY tutorials.

tions of both libraries

4.2 Data Scraping

A lot of data is required to apply machine learning models and obtain information from the Maker Community forums. Before processing and analyzing this data, it is necessary to acquire it from certain sources and make it available for the processes just mentioned. Nowadays, the Internet offers vast amounts of data, most of which are in unstructured forms, such as text, images, or videos. To effectively harvest such data from the web, one can apply a technique called web scraping.

Motivation on why data scraping is necessary.

Web scraping is the process of gathering information from the internet [Glez-Peña et al. [2014]]. One easy way of web scraping would be copying the text and pasting it into a file. However, with the current amount of data, web scraping is used as an automatic process. With the help of tools and scripts, larger amounts of data can be automatically collected and made available. In general, there are many tools available that can be used with different programming languages. The two main tools used in the python context are BeautifulSoup and Selenium.

Description of web scraping.

Beautiful Soup is a python library built for scraping structured HTML or XML data. One could ingest a web page's

Beautiful Soup allows working with HTML code. You can parse the code and grab information that is of interest.

source code and filter through it with Beautiful Soup to find specific content used on the page. For that, we first need to understand how HTML is structured. HTML consists of a head element, which can be a title, section, footer or divider, and many more. These elements can then be populated with class attributes or certain ID flags, mostly used to keep the elements unique. This makes it possible to find certain elements more easily. E.g. if you want to include the main part of an article in HTML, you can include this with `<div_class = "main">Main Part</div>`. The class "main" is only used in this context. We can now use Beautiful Soup's helper functions to filter out the main part easily. But using Beautiful Soup requires another python dependency, which allows us to get the HTML document we want to use. One of the libraries is called Request. It allows you to get the HTML source page into your script that you can parse with Beautiful Soup and scrape the necessary content.

Selenium as alternative method to perform web scraping.

Selenium, on the other hand, is a general-purpose rendering tool [Sel [2022]]. It was initially designed to render web pages for test automation of web applications. You could consider Selenium as a barebone web browser that executes JavaScript and renders HTML to your script. Since many websites rely on JavaScript to create dynamic content on the page, Selenium can also be beneficial for web scraping. Selenium is an excellent alternative for scraping when a page needs to be loaded first before JavaScript can display the dynamic content. It is a versatile tool that can do activities like button clicks and choosing items from dropdown menus. So, Selenium allows scraping on more complex web applications with some dynamic content. On the other hand, it comes with the disadvantage of being slower in the process than Beautiful Soup with Request.

The use of Beautiful Soap over Selenium is justified by the given simplicity of Instructables.

The choice between these tools depends on the complexity of the web application you want to get the data. Our main goal is to understand the context of Maker tutorials and extract the materials used in these tutorials. This can help us advance in user-friendly features such as filter and search and, in general, give the user and Maker a better overview of the possibilities of making such hand-craft ideas. We will restrict ourselves to Instructables as our primary source for this work and build our approach

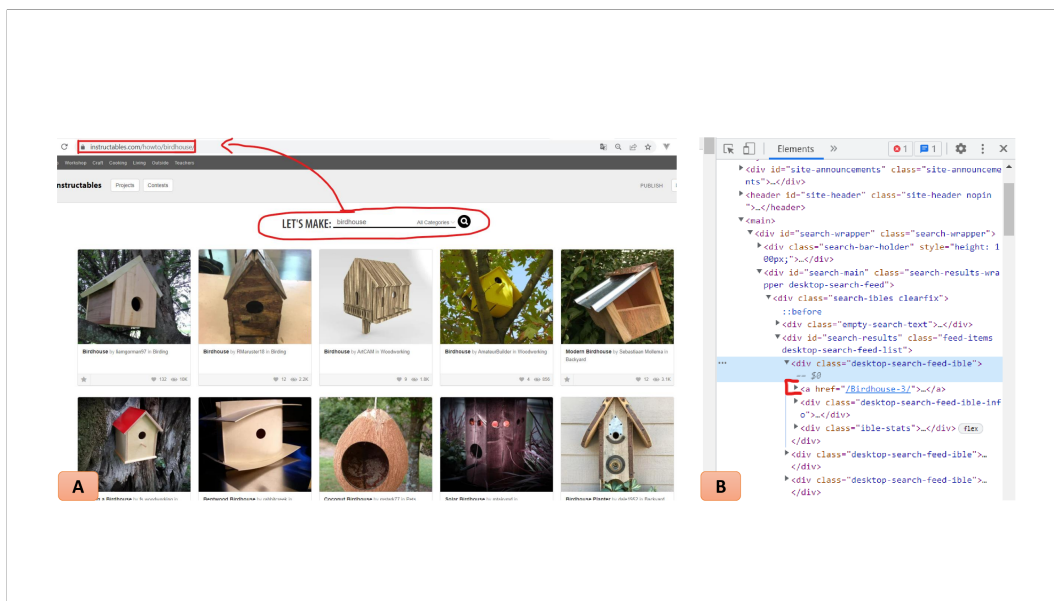


Figure 4.2: Two snapshots of the website [instructables.com](https://www.instructables.com). (A) The search of birdhouses listing all tutorials related to that and (B) inspection tool showing how to find the class attributes to filter on with BeautifulSoup.

using this data. Instructables is a website specializing in user-created handcraft tutorials. The user post instructions on a specific domain, usually accompanied by visual aids, and then interact through the comment section below. The website is built in a simple way so that you can look at the content without much interaction with it. For this reason, we will use BeautifulSoup and Request in this work to scrape the necessary data from Instructables. The approach works for static websites, such as Instructables. For more complex web applications, Selenium can be used as described above. We will now explain the web scraping procedure with BeautifulSoup and Request in the context of birdhouse tutorials on Instructables.

When we are on the website [instructables.com](https://www.instructables.com), we can quickly use the search bar to filter for birdhouses by typing in "birdhouse". This will lead to a new URL opening up a new site listing out the birdhouse tutorials on the platform (c.f. figure 4.2 (A)). But the content we want is inside each one of these tutorials. Clicking on one of them will lead us to the tutorial with the textual data we want

Getting the information from browsers inspection tool for BeautifulSoup

to retrieve. One possibility is to use Selenium to use the click functionality so that we can reach each tutorial. But instead, we know that each tutorial can be acquired by the URL `https://www.instructables.com/{id}/`. Every tutorial has a specific id, which can be fetched by the HTML Document that lists all the available birdhouse tutorials (c.f. figure 4.2 (A)). If we inspect by using the browser's Inspection tool, we can click on one of the elements and see its HTML element. Every tutorial item is built in a div and has the class attribute "desktop-search-feed-ible". One of their child elements is an anchor element `<a>`. This special element of HTML, which is used to create a hyperlink on the webpage, is provided with an absolute reference or a relative reference as its "href" value. In figure 4.2 (B), we can see that the relative link is provided as the id of the tutorial. So the ids of each tutorial will be extracted with the following code:

```
url = "https://www.instructables.com/howto/birdhouse/"
def scrape_data():
    result = ""
    html_text = requests
        .get(url)
        .text
        .encode('utf8')
        .decode('ascii', 'ignore')

    soup = BeautifulSoup(html_text, 'lxml')
    jobs = soup
        .findAll(
            'div',
            class_ = 'desktop-search-feed-ible'
        )

    links = []
    for job in jobs:
        component = job.find('a', href=True)
        links.append(component['href'])
```

Using the inspection tool to cope with infinity scroll and data loading, fetching over 160 documents.

With the request and the given URL, we can retrieve the whole HTML source code and then encode it with encoding into a uniform encoding, which we then convert with the decoder into a usable format. We use UTF-8 encoding,

the most widely used Unicode character [uni [2022]]. The URL we used to retrieve the data only included 20 tutorials. If we want a more representative dataset, we need more tutorials. So, by scrolling down, we will retrieve 20 more data sets. This, again, is a complex process that is not reproducible in Beautiful Soup and Request. But when we go back to our browser inspection tool under the tab network, we can see all the requests made and responses that came back. One request is responsible for retrieving the following 20 tutorials. It has the parameters limit and offset. Every time this call is called, we limit the call by 20 new data points, and the offset will be set on how many tutorials are already loaded, in this case, 20 as we already have loaded the first 20. We will use this request with these parameters to load the next 140 tutorials.

Like before, with BeautifulSoup, you have to parse all the fetched tutorials and then pull out the desired data with the given functions. To do this, you can use the browser's inspection tool and filter out the class attributes you are looking for. Here, we have to look out for the HTML elements "div" with the class "main-content" and in it get all HTML elements "section" with the class attribute of "step".

After fetching bigger data set, the data needs to be processed with BeautifulSoup with the same procedure as before.

4.3 Data Preprocessing

After obtaining the data, we will be using it to train our model and test it. The first thing we have to do is clean some noisy data and prepare the data in a way that our models can utilize it. Some common preprocessing steps are used in the context of machine learning. In the following, we will cover some of them and discuss which of them we will use.

Tokenization

Tokenization is a way of separating a piece of text into smaller units called tokens. Thereby, each segment or text can be divided into words, characters, or subwords. In

Description of
Tokenization

many NLP tasks, the models process the raw text at the token level [Bishop and Nasrabadi [2006]]. Splitting the sentences into words can still be challenging as you have to understand the rules of that language. Let's focus on the English language. A simple strategy for splitting is to use the white space as a splitting operator. But sometimes words can consist of many other words, i.e., New York City could be considered as one word and in the given split strategy as three words. Another problem could also be that multiple words are concatenated together. For example, the two words "He is" is mainly used as "He's". SpaCy and Huggingface are libraries that provide many pre-trained models in the area of NLP and offer several preprocessing functionalities. We will make use of their tokenization functions.

Stop word removal

Description of Stop
word removal

Stop word removal refers to removing words that are not informative. For example, articles, prepositions, and pronouns, such as "the", "as", "which", etc., add some meaning for the reader. But for a ML model, it can be noisy. A stop list, which are the stop words that will be removed, consists commonly of "be", "have" verbs, articles, auxiliary verbs and prepositions. There are some common stop lists that can be imported. [Bishop and Nasrabadi [2006]] [Mehlig [2021]]

Token Normalization

Description of Token
Normalization

Stemming and Lemmatization are a type of token normalization. In Stemming, the words will be reduced to their word stem, base, or root form. So, for example, the word "compute", "computer", "computers", "computing", and "computational" will all be reduced to "comput". Lemmatization is similar, but unlike stemming, it applies vocabulary and morphological analysis. This means it aims to remove inflectional endings only and to return the base or dictionary form of a word, the lemma. If we look at the previous example, "compute", "computer", "computers", "computing", and "computational" the words now will be

reduced to compute with an e. The main goal is to reduce the randomness, bringing the language to a predefined standard and reducing the vocabulary used for training. In our task for Named Entity Recognition, this preprocessing isn't sufficient as it will generalize, and we especially want to extract words such as "wooden dowels" and "wood". In contrast, token normalization would have transformed the "wooden" into "wood". [Bishop and Nasrabadi [2006]] [Mehlig [2021]]

4.3.1 Applying Preprocessing

Before we apply these NLP preprocessing steps, we will also have to clean the corpus and bring it into a more standardized way. When we look at our acquired data corpus, there are some white spaces we have to remove. Then we can start removing the labels from the enumerations, such as numeric or alphabetic numbers and Latin numerals. Herefore, we will run our corpus over a customized function that will catch all these unnecessary words or characters by using regular expressions. A regular expression is a sequence of characters specifying a text search pattern. Moreover, we will revise the whole corpus with lowercase letters to avoid redundancy. This will allow the word "Wood" and "wood" to be the same and make the training more efficient. Another preprocessing step is to consider missing values. Some documents will have no material listed or mentioned in their text. These could be regarded as outliers, meaning we won't consider them in our data set. But as we will see, we will have to label our data to train our model. For that, we will specify which parts of the corpus are material and which aren't. The outliers will be marked as not material and can still be used in our training data set.

Performing steps to clean the fetched corpus.

After cleaning the corpus and bringing it into a standardized form, we can start building our training data set out of this corpus. As mentioned before, we have to label our data set as we perform supervised learning techniques. In the foundation section 2, we mentioned our focus on the particular type of NLP Named Entity Recognition. This allows

Splitting the corpus after building the training and test data set in the JSON format.

us to automatically identify entities in a text and classify them into predefined categories, in our case, into materials. Thus, we have to go through the corpus and mark the labels so that the model can learn which part of a segment or text materials are used. Furthermore, after we have trained our model, we can use a test data set that is also labeled in the same way to test how well our model performs such entity extraction task. For this purpose, we will split our corpus into segments into two equal parts, and for labeling our data set, we will use a JSON format to mark the material position in that segment. The following example represents how such a training dataset would look in a JSON format.

```
[ "1/2 or 3/4 inch thick wood. (whatever kind you like)",
  "entities": [[22,26,"MATERIAL"]] ]
```

Marking the material in the training and test data set will be done in the JSON format.

The JSON object would have two attributes, one containing the segment and the second containing a list of entity objects. The entity object will have the label, start-, and end-position of each material occurring in the given segment. In that way, we know precisely which parts of the segments are material and which aren't. One way of generating such training data is to go manually over the corpus and extract the information by hand. Since our corpus is already very large and this approach generally also takes too much time, let us introduce an automated process for generating such datasets. For this, we will use the EntityRuler pipeline provided by spaCy. The matches found by the EntityRuler are in the doc attribute of the EntityRuler class. By creating these special pattern labels, we can run EntityRuler on the corpus and transform the entities we find in the form we expect them to be in our training data. Creating such a pattern requires a list of materials.

Definition:
EntityRuler

ENTITYRULER :

EntityRuler will help to find matches in a document or text and add them as entities to its properties, using specified pattern label as the entity label.

Creation of material list to mark the training and test data sets will be done manually.

A material list can be gained from other sources on the internet, i.e., Wikipedia. But to keep the training data's ac-

curacy high and thus generally increase the performance of the models yet to be created, we will manually create a list of materials from the corpus. High accuracy refers to mistakes that will occur through auto-generated data. In the evaluation chapter 5, we will elaborate more on how well our data sets are and what a higher accuracy means in terms of model performance. Having such a list, we will make sure that the list contains all materials from the corpus within singular form and plural form. You can do this manually because of the relatively small size of the list of materials. Otherwise, you could also use English language rules to generate each word's plural form. Besides, we will apply the lower-case letter function here as well.

```
def create_training_data(file, type):
    data = load_data(file)
    patterns = []
    for item in data:
        pattern = {
            "label": type,
            "pattern": item
        }
        patterns.append(pattern)
    return (patterns)

def generate_rules(patterns, model_name):
    nlp = English()
    ruler = nlp.add_pipe("entity_ruler")
    ruler.add_patterns(patterns)
    nlp.to_disk(f"./entity_rulers/{model_name}")
```

This code shows how we can now create an empty model with the EntityRuler as a pipeline. We will use the model `English()` from `spaCy` that is trained on English text. With the utility function `add_pipe`, we include the pipeline `EntityRuler` in the model. Now, this model can use rules-based matching. The `EntityRuler` has a function called `add_patterns`, which will integrate the pattern we created before. This allows the model to find these entities in the corpus. We will save this model and use it for generating the training dataset.

To generate the training dataset, we will run over the cor-

Using `EntityRuler` to generate training and test data sets automatically.

pus and split it into segments. Each of these segments is then run over with the EntityRuler model, and the matches the model finds are stored in the JSON object form described earlier. This gives us a training dataset with about 1400 entries, where each entry contains at least one material but may have more. How good the approach of automated generation of training data is will be discussed in the evaluation chapter. If we train our data on this specific data set and test it on the same data, the results will always be excellent, even if our model is very bad. Therefore, we also need a test data set that is unknown for the trained model. The structure of the test data set will be similar to that of the training data set. We have a large dataset partitioning into training, and test data is still sufficient to achieve good results with the trained model. Moreover, it is common to have a validating set in training a machine learning model. Validation datasets ensure that the model is not overfitted to the training dataset. It helps generalize the model, which can perform better on unseen datasets. The validation step can be performed after each epoch, after several epochs, or even for each data point. The user must try these strategies and apply the one that best suits his use case. We stuck with validating after each training epoch as it gained the best results.

Customized Pre-Processing for Huggingface

Convert data in a binary form for efficiency.

SpaCy models allow the training with the command line with one command. But for this, the data has to be in a particular form. Once we have the JSON-formatted training data, we can use spaCy's utility function `make_doc()`, which will convert the JSON into a DocBin. The DocBin class is a binary serializer, which let's you be more efficient in the training process as the data is less in size [spa [2022]].

Customized Pre-Processing for Huggingface

Change data format for the library Huggingface.

The Huggingface model requires specific input set to start the training process. For this purpose, the library offers a specialized data structure called Datasets. It is a

lightweight library that provides one-line data loaders for many public datasets and efficient data preprocessing. We want to go from the JSON-Object, where the first attribute is the segment and the second is a list of the absolute position of materials occurring in the segment, to a tokenized form. This means we will tokenize each segment and have a list of tokens. Besides the list of tokens, we create another list, which has the same length, and we will fill this with the label "MATERIAL" at the same indices where the material occurs in the token list otherwise, we fill it with the label "O". "O" stands for other and should indicate that the token at that position is not material. Since we have only one label, this word label list is a binary classification with the labels "MATERIAL" and "O". An example of this conversion can be found in example (c.f. List of labels and tokens).

BILOU TAGGING :

BILOU encodes the Beginning, the Inside, and the Last token of multi-token chunks. Moreover, it differentiates a single unigram token and marks it with U for Unit or Unigram. The O stands for others and marks all the non-materials.

Definition:
BILOU Tagging

Another technique that we will make use of is BILOU tagging. As the NER task is commonly viewed as a sequential prediction problem in which we aim to assign the correct label for each token, one token could also be just a part of the whole named entity. For example, "wooden dowel" is a material consisting of two words. Instead of considering it as two materials by just tagging it with the label MATERIAL, we will be using BILOU. The sentence "The douglas fir board was shipped from South Korea to Germany." would be labeled as follows: [O, B-MATERIAL, I-MATERIAL, L-MATERIAL, O, O, O, B-LOC, L-LOC, O, U-LOC, O]. For encoding the labels, we can use one of spaCy's utility functions `offsets_to_biluo_tags`, which will use the positions of the JSON dataset and convert it into the form we need.

Using BILOU tagging for multi-word recognition.

After transforming the training, validating, and test data to the object containing the sentence, which is the list of to-

Aligning the input data with the labels as Huggingface uses special token to understand the start and end of each segment.

kens, and the word labels, which is the list of labels, the three objects can be added into a new DatasetDict. The Huggingface model requires another preprocessing step for the inputs. They expect the input to be in a mathematical representation. To achieve this, we can use the AutoTokenizer of their library, which will tokenize the inputs and put them into a format the model expects. It will also convert the tokenized tokens into a corresponding ID that the models have used in their pre-trained vocabulary. With the `Autotokenizer.from_pretrained`, we can fixate the model we want to use from the large variety of models available at Huggingface. However, using the tokenizer will raise the problem that the record length will no longer match the size of the labels. This leads to the fact that these two lists do not correspond anymore, which is necessary to inform the model correctly whether the tokens are material or not. To elaborate on this problem and how we can fix it, we will use an example to explain the tokenizer in more detail. Consider the following code snippet:

```
def tokenize_and_align_labels(examples):
    label_all_tokens = True
    tokenized_inputs = tokenizer(
        examples["sentence"],
        truncation=True,
        is_split_into_words=True
    )

    labels = []
    for i, label in enumerate(
        examples["word_labels"]
    ):
        word_ids = tokenized_inputs
            .word_ids(batch_index=i)
        previous_word_idx = None
        label_ids = []
        for word_idx in word_ids:
            if word_idx is None:
                label_ids.append(-100)
            elif label[word_idx] == '0':
                label_ids.append(0)
            # We set the label for
            # the first token of each word.
            elif word_idx != previous_word_idx:
```

```
        label_ids.append(
            label_encoding_dict[
                label[word_idx]
            ]
        )
    # For the other tokens in a word,
    # we set the label to either the current label
    # or -100, depending on
    # the label_all_tokens flag.
    else:
        label_ids.append(
            label_encoding_dict[
                label[word_idx]
            ] if label_all_tokens else -100
        )
    previous_word_idx = word_idx

    labels.append(label_ids)

tokenized_inputs["labels"] = labels
return tokenized_inputs
```

By tokenizing the following sentence, the tokenizer will return a dictionary with three items: The `input_ids`, which are the indices corresponding to each token in the sentence, the `attention_mask`, which indicates whether a token should be attended to or not and the `token_type_ids` that identifies which sequence a token belongs to when there is more than one sequence. If we check now the length of the list of sentence tokens and the tokenized `input_ids`, they will be different. With the utility function `convert_ids_to_tokens`, we can convert the ids back into readable words. A closer look shows us that besides the sentence there are also some special tokens. The CLS token is the classifier token and always appears at the start of the text, whereas the SEP token helps the model to understand the end of one input and the start of another input. That is why it is called the separator token. It should also be noted that transformers are pre-trained with subword tokenizers and thus use such. This means that even if the sentences are already split into words, each of those words could be split again by the tokenizer into subwords. In the code we can see, how one of the token is split into multiple token. All these will lead to

a bigger length of inputs. By using the `word_ids()` method, we can inspect whether the token is a special token by the id of -100 or is split into subwords where the ids are the same. This allows us to write a general alignment function that will set the length of the `input_ids` equal to the length of the labels list. We can set the labels of all special token to be -100 which indicates PyTorch to ignore that token and the labels of all other tokens to the label of the word they come from. Then we can run over the sentence and align the labels, where we get the information out of the `word_ids()` method.

4.4 Training

We have built a labeled dataset, split into training, validating, and testing sets. The spaCy model and the Huggingface model will be used to train specific NER models, for which we have preprocessed the datasets. For spaCy, we had to convert the data into the particular DocBin format, and Huggingface expected the datasets to be tokenized so that we could train the models in both of these libraries. This chapter will briefly discuss how to conduct such training and the strategy we used to do so.

Training and
configuration of
spaCy CNN model.

Since spaCy3.0, the training can be performed in a few steps. The standard way to complete spaCy training is via the `spacy train` command on the command line. It only requires a single configuration file that includes all settings and hyperparameters. A first starter setting can be found on their main page, [spaCy.io](https://spacy.io). Under the hood, the training config uses the configuration system provided by the machine learning library Thinc. You can configure the path for training and validating data, the optimizer function, how many iterations it should train, and many more. We will use our preprocessed data and set the epoch to 10, as the results decreased by increasing the epoch more than 10. We used the default settings for the learning rate as it is modular and will be adjusted after each epoch. For the dropout rate, we set it to be 0.1, which means that about 10% of the neurons used in this model will be dropped randomly during the training. This is another regulator for reducing

overfitting in artificial neural networks by preventing complex adaptations on training data. The main idea behind the settings is to increase the performance by trial and error. Now by using the terminal, we can start training the model with the given training and validating data and the configuration. We trained our model on the CPU, as we are using only fine-tuning the bert-base model, and the training does not consume too much capacity as if you had to train a complete empty model from the beginning. This will generate a model that can extract materials from a human-written text. To test this, we can load our model and the test data and run them against each other. The result can be seen in the appendix [table A.1 (Spacy CNN Model) and figure A.1 (A)]. The evaluation chapter will discuss more detailed evaluation of these results and improvements.

In the foundation chapter 2, we also discussed the word vectors and the benefits of using these word vectors. In essence, it can help us to generalize the model and, thus, perform better on unseen data. SpaCy offers the possibility to integrate third-party word vector libraries into their model. We will be using the Gensim library to create such word vectors out of our corpus and then incorporate them into a spaCy model, where we perform similar training steps as before. We start making the word2vec model by first preprocessing the corpus. In this case, we are no longer interested in the sequence, rather focusing on the relevant words that are available in the domain of birdhouses. That's why we will first remove the stopwords and punctuation. We will use the NLTK library to get a stop list and common punctuations, which we will use for cleaning the corpus. Afterward, we can start building our Word2Vec model that is done with Word2Vec class from Gensim. Here, the parameters have to be adjusted. We will set the min_count to be 1. This will ensure that our model will use all the words, even if the frequency of the word in the whole corpus is only one. We use this strategy as we want to extract all kinds of materials that are used for birdhouses, especially those which are used very rarely. Moreover, since we have already cleaned the corpus, irrelevant words will not affect the model much either. For the window size parameter, we set it to 2 as recommended. Word2Vec works with a specific window

Creation of word vectors and embedding it into our existing model for further training.

size to detect similarities to other words. When adjusting this value, there were no substantial improvements that were relevant. Also, the size parameters were adjusted a few times and tested for performance. The size represents the dimensionality of each individual word. The dimensionality increases the ability of the model to harmonize a more extensive vocabulary and bring words closer together. Here, a size of 500 was enough for good results. As soon as we build and set up the Word2Vec object, we can start to build the vocabulary with it, which will integrate the corpus into the model. After that, we will train the model for 30 epochs and save the created word vectors as a text file, which we can use in spaCy. The results on the model with word embeddings is also depicted in the appendix [table A.1 (Spacy WV Model) and figure A.1 (B)].

Introduction into
training of
Huggingface models.

The last strategy is to use a transformer model to help us extract the material from the birdhouse tutorials. Therefore, we will be using one of the Huggingface pre-trained models. Some of the primarily used Huggingface models are BERT, RoBERTa, BART, DeBERTa, GPT-3 and many more [Huggingface.com]. Transformers are language models and have been trained on a large amount of text in a self-learning fashion. The architecture and how transformer work is explained in the foundation chapter 2. We will be using the BERT model that is already tested to be a performing model [Korpusik et al. [2019]]. To use the Huggingface libraries, we performed some pre-processing steps and built a Datasetdict, which is tokenized. Now for training, we first will download the BERT model from the repository of Huggingface. Since we want to perform NER, which is also regarded as a unique form of token classification, we will be using AutoModelForTokenClassification class. With this, we need to specify which model we are using, and in addition, we have to determine how many labels we are using. We now have more than two labels because we are using the BILOU label tagging approach, so we need to find out how many labels we are using. This can be done by running over each label list and checking for a new label or with pandas dataframe functions explode() and unique().

Training and
configuration of the
BERT-Transformers
model.

To perform the training with our pre-processed data, we

will use the Trainer class of Huggingface. The Trainer class provides an API for feature-complete training in PyTorch for most standard use cases. Moreover, the Trainer class is optimized for Huggingface Transformer models and can be customized by overriding the methods of this class. To instantiate a Trainer, we will need to define three more things. One is the TrainingArguments which is a class that contains all the attributes to customize the training. It requires one folder name, which will be used to save the model, and all other arguments are optional. Some of the configurations we made are setting the evaluation to be done at the end of each epoch, tweaking the learning rate, using the batch_size and defining it to be 16, and customizing the number of epochs for training. Then, we need a Data Collator that are objects that will form a batch by using a list of dataset elements as input. To build batches, data collators may apply some processing (like padding), and each pad will be padded to the length of its longest example. This is a necessary step as a neural network's input should always be the same length. The last thing to define for our Trainer is how to compute the metrics from the predictions. Here, we will load the seqeval metric, commonly used to evaluate results on the CONLL dataset, via the Datasets library, a standard dataset to test NER models [con [2003]]. This metric takes a list of labels for the predictions, and we had to do some post-processing on the predictions to convert them back to strings and ignore the special tokens. Now, we can use the Trainer to start training. The results will be depicted in the appendix [table A.1 (Transformer Model) and figure A.1 (C)].

Chapter 5

Evaluation

This section will evaluate the concepts that have been applied so far. We will only discuss the conceptual design in general depth and refer to chapter 4 for a deeper understanding, which covers the solution approach and implementation in detail. Our work deals with the extraction of materials from DIY tutorials, which are mostly written by humans. Platforms that provide such tutorials usually do not offer the possibility to get an overview of which materials are used in tutorials from specific domains. To gain information from these user-generated tutorials and to cope with the large number of available tutorials, we turn to Natural Language Processing (NLP). With this technology, we have created a pipeline that can be used to generate extraction models very efficiently and quickly. We developed three different models using two different neural network architectures. For the model with the same architecture, we introduced the concept of word embedding [Mikolov et al., 2013a][Mikolov et al., 2013b][Mikolov et al., 2013c] to increase generalization so that the model works better on unseen datasets. In this chapter, we will discuss the evaluation of these three models. In doing so, we will address the quality measures described in the fundamentals 2 and evaluate them according to these values. In addition, the evaluation will also go beyond purely computational measures and assess the results for impact on the user. For this purpose, isolated specific results will be picked out, and their meaningfulness will be analyzed. In the following, we will

Short summary of conceptual design.

Structure of evaluation.

talk about several concepts and use technical words that we have already described in more detail in the fundamentals chapter 2 and the concept chapter 4. Therefore, we point out to read these chapters in case some terminologies are not clear.

5.1 Named Entity Recognition

As described above, NLP was utilized to ensure the generic extraction of materials from Maker projects. In particular, we make use of Named Entity Recognition, which is the task of identifying and categorizing key information (entities) in text. An entity can be any word or series of words that consistently refers to the same thing. Every detected entity is classified into a predetermined category. As mentioned, our task is to extract materials from unstructured text, such as handwritten tutorials from *instructables.com*. [Eisenstein [2019]]

Test data was developed in a similar way as training data.

In the concept section 4, we developed three models to extract materials out of the tutorials for birdhouses. Therefore, we generated a training data that was preprocessed to fit the use of different models. Hence, all three models were trained by the same training data and are comparable to each other. The comparison requires unseen test data, where all three models perform their prediction, and their results will be evaluated with the corresponding test data. The test data is generated similarly to the training data. In fact, when we took the data from *Instructables*, we used half of the document to train the data and the other half to test it. To generate the labels, we performed the same steps by using the Entity Ruler and the manually created materials list described in the concept chapter 4 while extending the list of materials. Besides, the same preprocessing steps were executed on the test data as on the training data (c.f. chapter 4).

As it is described in the concept chapter 4, we are using the approach of the Entity Ruler in order to generate more extensive data sets in a faster way. But this comes with a tradeoff that the accuracy of the data degrades. In the fur-

ther stage of the evaluation, we will deal with the accuracy of the test dataset and train dataset and analyze the impact in more detail 5.1.2.

5.1.1 Quality Metrics

To calculate the confusion matrix for all three models, we will be using the confusion matrix tool from the scikit-learn library. The confusion matrix is a performance measurement for machine learning classification problems where the output classes can be two or more. It tries to visualize the outcomes in a matrix form where the rows represent the predicted value, and the columns represent the actual value [Bishop and Nasrabadi, 2006]. Read the Evaluation of Machine Learning chapter 2.3.5 for more detail. This will lead to a diagram where the True Positives (TP) and True Negatives (TN), as well as the False Positives (FP) and False Negatives (FN), are depicted. Multiple indicators can be computed based on the TP, TN, FN, and FP to measure a model's performance. We will calculate quality metrics such as F1 Score, Precision, Recall, Accuracy, and Error. These metrics are a common way of evaluating the outcomes of machine learning models and can help us compare the performance of different models [Bishop and Nasrabadi, 2006]. We need to change the test data set to calculate these metrics, where each tokenized word is represented by the corresponding label. Consider the following example:

Quality metrics to evaluate all three models.

Spacy format: ['I will be using wood to build the birdhouse.', entities: [16, 19, MATERIAL]]

Transform to:

List of labels and tokens

Labels: ['O', 'O', 'O', 'O', 'MATERIAL', 'O', 'O', 'O', 'O', 'O']

Tokenized Sentence: ['I', 'will', 'be', 'using', 'wood', 'to', 'build', 'the', 'birdhouse', '.']

	Spacy CNN Model	Spacy WV Model	Transformer Model
Precision	0,861	0,860	0,863
Recall	0,649	0,637	0,710
F1-Score	0,741	0,732	0,779
Accuracy	0,978	0,977	0,979
Error	0,022	0,023	0,021

Table 5.1: Computational metrics of all three models

Transformation of outcomes for evaluation.

We will make use of the utility function “offsets_to_biluo_tags” from the spaCy library to transpose the spacy data format into the list of entity labels. The spaCy models’ output will also be laid out in the spaCy format. Thus, we will have to transpose the outcome in the same fashion as we did with the test data. For the transformer model, the output is already in the desired form.

First description of the results depicted in the confusion matrix.

In figure 5.1, the confusion matrices of all three models are illustrated. The rows reflect the true labels, and the columns the predictions. The two possible classes are “O” (negative entity) if the entity is not a material; otherwise, it is tagged as “MATERIAL” (positive entity). Even though the two spaCy models and the Transformer model have been given the same test data set, the number of individual instances is different from each other. The total number of instances of the two spaCy models is 35589, whereas the number of instances of the Transformer model is 36648. This is mainly due to the fact that both use different tokenizers, splitting the sentences with various rules. However, the difference is not too big and can be neglected when comparing the two models.

Transformer model performs the best according to the confusion matrix.

Having a high score in the confusion matrix means having all entities represented in the True Positive or True Negative class, as the prediction predicted all instances correctly. At first glance at figure 5.1, we can observe that the transformer model has performed best of the three models. We can also support this statement with the quality metrics F1-score, recall, and precision (cf. Table 5.1).

In general, both spaCy models including the model with word embedding performed about similarly, with the Transformer model performing better in each metric.

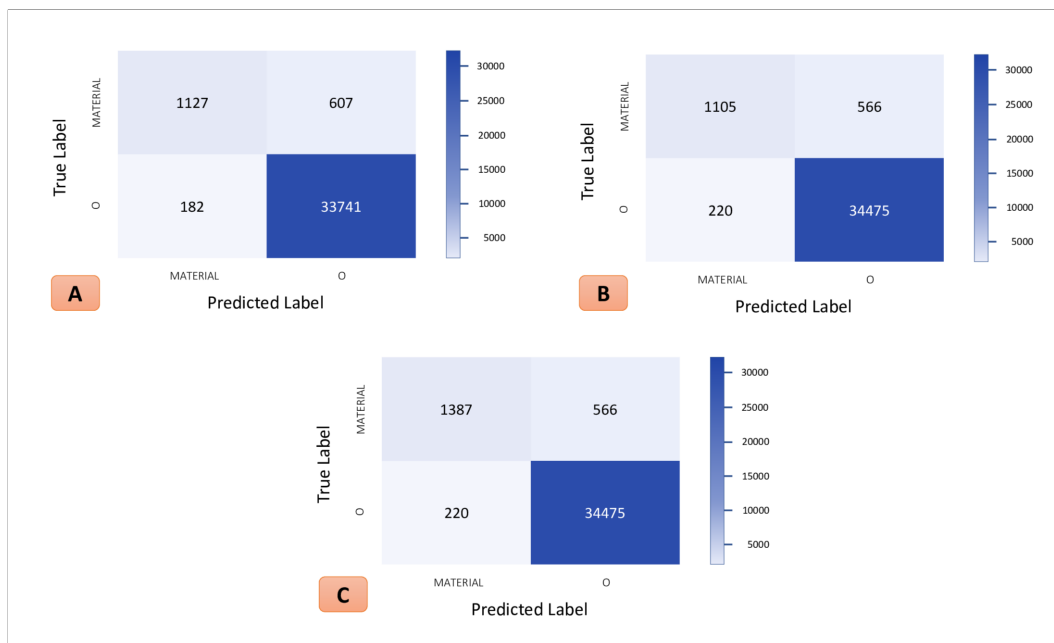


Figure 5.1: Confusion matrices of all three models before optimization. These results came out after the pipeline was built and the model were trained once with the given pipeline. (A) Spacy CNN Model (B) Spacy Word Vector Model (C) Transformer Model

Precision attempts to answer the following question: What proportion of positive identifications was actually correct? In all three models, the precision was about the same. When we look closer and take the transformer model (figure 5.1 (C)) as reference, we observe that we predicted 1607 identifications to be true, whereas 220 out of these were actually false, which means our model was a bit too generalized. But overall, all three models have a decent precision score (table 5.1) with precision 1 being the best case.

Explanation of precision and outcomes based on precision.

When it comes to recall, we try to answer the following question: What proportion of actual positives was identified correctly? Looking at the result of all three models (table 5.1), it is apparent that the Transformer model performed the best. Out of 1953 positive entities, the Transformer model predicted 1387 entities to be positive and 566 entities as negative (figure 5.1 (C)). Thus, the value of the recall is approximately 71% (table 5.1). Examining the Confusion Matrix of the Spacy model with word embeddings

Explanation of recall and outcomes based on recall.

(figure 5.1 (B)), the model missed 566 out of 1671 positive entities. These missed entities denote that the model could not identify unknown materials from the given sequence or even misidentified known materials from the context of the sequence as negative examples. Both spaCy models, perform worse than the Transformer model, with a recall slightly less than 65%. This indicates that the Transformer model is better at recognizing unknown positive entities than the other two models.

Explanation of
f1-score and
outcomes based on
f1-score.

The F1-score is a way to combine the precision and recall of the model, and it is defined as a harmonic mean of the precision and recall of the model. Since, it considers both precision and recall scores in its calculation, the Transformer model performs best. Also, this value is commonly used in evaluations of machine learning models proving to be a good reference to evaluate these models. With an approximate F1 score of 80% (table 5.1), the transformer model performs solidly. Optimizations will be described in the course of the evaluation chapter 5.

Unbalanced dataset
can influence
accuracy results.

In terms of accuracy, all three models are very high and close to 100% (table 5.1). Accuracy is the fraction of predictions the model got right. However, these values should only be taken with caution. Looking back at the confusion matrix's number of each class by rows, we can see that the negative entities are much more represented and thus the dataset is also unbalanced (e.g.: Transformer model: Number of 'O'-Entities: 34475; Number of 'MATERIAL'-Entities: 1953) (figure 5.1). In general, you want the data set to be balanced, in the sense that all classes are more or less equally represented. In our case, we are dealing with NER and thus we are trying to extract certain entities from a longer sequence. For this reason, the positive class is significantly less represented than among the negative. Hence, the prediction of negative entities is very good and this in turn increases the accuracy, since it is determined almost exclusively by this. But the F1-score reflects that the models also recognize positive entities relatively well and are thereby not influenced by the high number of negative instances. Since the error rate is inverse to the accuracy, the same reasoning applies here, except that it is close to zero.

So far, we have mentioned that the Transformer model has performed the best out of the three. The two spaCy models perform similarly, whereas the model with the word embeddings performs minimally worse. In the approach section, we mentioned that word embeddings are used to get a better generalization. However, if you look at the recall of both SpaCy models, you will see that it is even worse, and more positive entities are missed (figure 5.1(A), figure 5.1(B) and table 5.1). This has the simple reason that word vectors are limited to a window size, which then bundles "related" words in the vector space which also often occurred together in the entire corpus. A more detailed explanation of word vectors can be found in the foundations section 2. But with this set up a material like wool, which replaces wood will never be in a close distant to the word wood in the vector space as they barely occur together in a document. So, if the model never saw wool as a material, it will most likely not get it with its prediction. Thus, the generalization is worse and the purpose of word embeddings (word vectors) has no additional value.

SpaCy model with word embeddings performs worse than without.

5.1.2 Evaluating in depth: The problem areas

So far, we have limited our evaluation of the models purely to computational metrics and evaluated them using the F1-score, precision, recall, and accuracy. The scores are generally high with the F1 Score being around 0,75. In general, the values can go up to 1 and this would correspond to the best-case scenario. For values of 0, the metrics would correspond to the worst-case scenario. We can observe that the Transformer model is a very good model for extracting materials related to human-written tutorials in the field of Maker projects as they performed the best. Now, we are more engaged in the meaningfulness of the models' results and trying to develop a better comprehension of the problem areas. We try to categorize the problem areas to address them better. These can help us to clarify the optimization possibilities. Below we will list some example sentences with the markings (yellow) that we will talk about. Keep in mind, the marking does not necessarily mean, that the prediction marked it as a material. In some cases we also

highlighted the entity to emphasize on inaccuracies. With it, we will evaluate how well the model did on some specific cases and also elaborate on the error cases.

First of all, we have already listed two categories of errors in the Confusion Matrix (figure 5.1), which divides all incorrect entities into the classes False Positive as well as False Negative.

“hang with a **string** from some where that will get smelly and could get dirty!” (Example 1)

Explanation of False Negative example.

The example 1 belongs to the class False Negative, as the prediction missed the entity *string*. The material *string* was not present in the training dataset; therefore all three models did not detect it. Also, the models could not identify *string* as material from the context of the sequence. All three models have the ability to identify never before seen materials from the context of the given sequence. This is exceptionally well solved in the Transformer model, which can also be deduced from the recall value (table 5.1). An example of this kind can be reproduced with the material *oil* (Example 2).

“I also used a spray bottle to soak the roof and get the **oil** into all the nooks and crannies.” (Example 2)

Example, in which the Transformer model found a material that was unknown before.

Here, the Transformer model correctly identified *oil* as a material, even though it never occurred in the training data. Since, it was also marked in the test data to be an positive entity, the identification was categorized into the class of True Positives. Even with the ability to recognize unknown materials, False Negatives are noted as a problem with all three models (c.f. figure 5.1), which are all missed positive entities. To create a reference to the context, we call the class of False Negatives as Missed Entities. The Missed Entities are of different sizes in the different models (c.f. figure 5.1). However, if you take a closer look at the Missed Entity class, you will see that some of these entities are not necessarily materials. In example 3, the entity *fence post* was

Correctly identified non-material, which was falsely marked in the test data.

marked as a material (positive entity) in the test data set. Yet, the models correctly did not identify this as a material. The entity *fence post* is not used to create a birdhouse, which we have set as a definition of materials (positive entities) and therefore the entity should not be considered as a material (positive entity). It follows that the test data set accommodate some errors. We briefly discussed this problem at the beginning of the evaluation. Since, we have chosen an approach to quickly and easily generate more extensive training as well as test data sets, we must deal with a trade-off of getting a loss in accuracy in comparison to a manual creation. Later, when the optimization possibilities are discussed, we will see how to tackle this problem. However, since the computational results are generally high, we can assume that the tradeoff does not impact much.

"You can mount it on a tree or a **fence post**, close to a source of water." (Example 3)

Most errors are not caused by the generation of the training or test data sets themselves, but by the manual preparation and extraction of the materials. As in example 3, *fence post* seems to appear in the list of materials that was used to generate the test data set, even if no fence post is needed directly in the creation of a birdhouse. This is one of the main reasons for the inaccuracy of the data sets. The occurrence of fence post in the list of materials could be that in one tutorial some old fence post were used for the birdhouse case. So, this shows again one of the disadvantages of auto-generating training data sets. This means if in any tutorial a fence post is used, it will be listed in the materials list. With it and the auto-generation of training data, all fence post will be marked as materials. Later, we will talk about how we can involve user to improve the dataset 5.1.3, 5.1.4. One should also not neglect the ambiguity of language, which can also lead to undesirable inaccuracies.

The materials list can have mistakes as well and does not consider content of tutorials.

"**stain** or clear sealer may be applied on outside of box only." (Example 4)

“Stubborn **stains** and rubber residues which cannot be removed.” (Example 5)

Error example caused by ambiguity of the language.

Example 4 uses the word *stain* as a type of paint utilized to color wood, whereas Example 5 shows *stain* in a context where it is not a material. This type of error is shallow among the three existing models and in the training and test data sets.

Error class of False Positives.

The second category of errors found in the Confusion Matrix (c.f. 5.1) is the class of False Positives. The models here have attempted to identify certain entities as materials from the context, which are not materials according to the test data. In Example 6, *mosaic* is recognized as a material, but in that sentence, there is only one material, which is *tile*. Thus, the models are bit over generalized and misinterpreted the as a material out of the sequence. To create the reference to the context here, we also name the False Positives class as Over Generalized.

“Step 4: Preparing the tile for the **mosaic**.” (Example 6)

“Depending on the **hinge** you use, you’ll likely need to make some last - minute adjustments to get everything lined up.” (Example 7)

Inaccuracy of test data causing another error class.

We should also keep an eye on the inaccuracy of the test data in this class. This imprecision leads us to assign positive entities found by the models, to the class of Over Generalized, although they are actually materials. If we look at example 7, the entity *hinge* was considered an over generalized prediction. Nevertheless, this is material that is used in the constructed birdhouse. The reason for the error is the sole fact that our test dataset incorrectly did not mark it as a material. Compared to the incorrect records of the Missed Entities class, the frequency of the incorrect records in this class is much higher. Fixing these inaccurate datasets might lead to an improvement in the overall score of the models. In general, we can say that the Transformer model handled

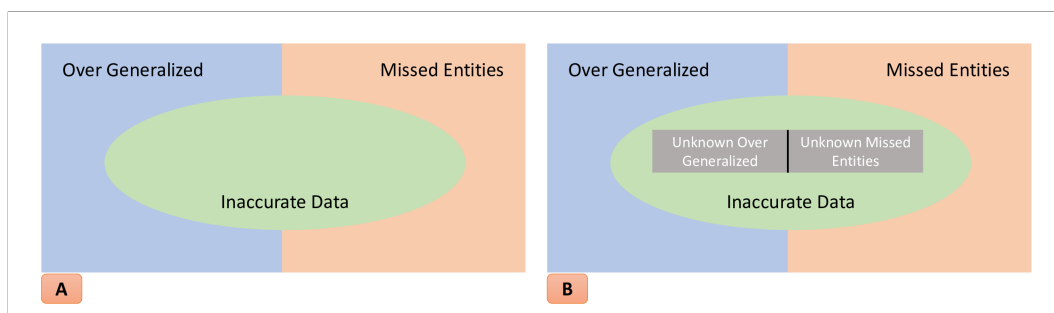


Figure 5.2: The graphic depicts the error classes. (A) Describes the error classes that will be caught by our evaluation methods. (B) Describes the error classes that are not picked up by our evaluation.

the over generalization as well as the missed entities the best.

The figure 5.2 (A) illustrates the class of errors we described above. As we can see, these errors intersect with the imprecision of the test data set. As described earlier, we can examine these areas collectively or individually and try to address these issues. We have seen that the Confusion Matrix depends on the test data's accuracy. Moreover, we will see in the optimization part 5.1.3, the accuracy of the test data elevates the score of the evaluation metrics. So far, we have been able to use the error classes to identify the differences that occur between predicted entities and those from the test dataset and thereby analyze the errors selectively. However, the Confusion Matrix is not only limited on the test data, but also limited on the quality of the training data. This leads to the fact that our models are trained with smaller errors due to the inaccuracy of the training data, although this is not visible in the evaluation. This is the case that the predictions of the models and the actual instances from the test data match and thus these instances are evaluated as correct.

The error classes that are known in the data set

"Typically the **filler** will have set and hardened sufficiently in 10 to 15 minutes" (Example 8)

"Hot **glue** gun with enough hot **glue**" (Example 9)

The error classes that are unknown in the data set

Example sentence 8 shows an exemplary case where an error is introduced that the word *filler* in the test data set as well as the predictions of the models have recognized it as a non-material, but in reality, it should be a material. The other case is illustrated in the example sentence 9, where the word *glue* appeared twice in the sentence, but the first referring to a tool and only the second entity is a material. Then again, these erroneous classes are not so frequently represented in the whole data set. The figure 5.2 additionally shows the error classes of the unknown area, which is very small in this case, but should not be neglected. We talk about unknown, as they won't be recognized by our tools. This error, as well, can easily be increased through the ambiguity of the language. While this is not the case with us, it can impact the error class depending on the domain.

Evaluating the generative data set approach

The generative approach to creating data sets introduces some errors, but the tradeoff is justified because larger data can be produced in less time.

To get an approximate sense of how much the individual training data or test data lost accuracy from the generation process, we took a small portion of the data and manually checked how much of the data suffered. For this, we limited ourselves to the test data and looked at the first 10000 entities, whereas we used the tokenization of spaCy which leads to an overall number of 35589 entities. So, we tested approximately 30% of the data. We also made sure that those chosen entities will cover about half of the segments with negative instances. In these first 10000 entities, we discovered about 237 erroneous instances. This makes up about 2% of the tested data. Moreover, we took more than half of the segments with negative instances. If we look more closely, we can also see that most of them come from the Missed Entities class and also many of those that belong to the Incorrect Data. Also, additional words were identified as positive entity that specify the material more precisely. For example, the word *lumber* was marked as a material in the test data set, but two out of three models predicted the word *scrap lumber* as a material. In addition, in the 30% of instances we have already covered a lot of the few wrong instances, which also account for most of the erroneous data. Conclusively, we can say that the generative

approach of test and training data is a valid approach as they only produce minimal errors in contrast to the possibilities of generating big data sets in less time.

5.1.3 Optimization possibilities

This subsection will focus on ways to optimize models that already perform very well, but where some problem areas can be eliminated through pre- and post-processing. We can focus on the graphic (5.2) and its depicted classes to improve our model. In the previous chapter, we described each class in more depth. We will use that knowledge to leverage the feasibility to optimize the model.

The most accurate way to improve on the inaccuracies of the test data set is to address the issues in that data manually. For this, we used 30% of the test data and built a visualization tool to compare test data with the predicted data in order to see how much improvement can be made. In figure 5.3, a snippet of the visualization tool can be seen. The model's prediction is shown at the top and the actual segment from the test data set is shown below. We could quickly see differences, which helped us tackle the issue. We can bootstrap the data manually by going over the visualization, focusing first on the marked entities and whether they are correct. As mentioned earlier, most errors related to erroneous datasets occur in the Missed Entities class. Eliminating these can vastly improve the data set. We will also be fixing problems where the model found a more descriptive material whilst the test data set only marked the general material. We can ensure in later steps, e.g. through post-processing that the meaning of both descriptive and general forms of material are the same. The other class Over Generalized data can be also bootstrapped manually in a similar manner.

The Over Generalized class describes entities found from the model but not marked from the actual data set. These can't be improved by manually bootstrapping the test data if the marking from the test data is also accurate. However, the Confusion Matrix (figure 5.1) shows that this class

Using the visualization tool to bootstrap 30% of the data.

Handling the class Over Generalization.

Prediction	Find a fence post to attach the soda can birdhouse with a nail gun and then use a string for additional support (the cord should be weatherproof).
True Label	Find a fence post to attach the soda can birdhouse with a nail gun and then use a string for additional support (the cord should be weatherproof).

Figure 5.3: Snippet of the visualization tool to compare test data with predicted outcome. The given example is fictitiously chosen to represent all classes in this segment.

(False Positives) is generally low, so there is not much room for improvement. But out of these, many of the predictions seem to be correct and ergo the test data set lacks accuracy. An adjustment in that particular area can improve the overall score.

Erroneous data in the unknown area can worsen the score.

We also talked about how the Confusion Matrix depends on the input's quality. This means that there are correctly classified instances, meaning that the prediction and the actual test data match, whereas they should be classified incorrectly. A correction for this does not improve the overall score but may as well worsen the score. Fortunately, the erroneous data in the unknown area are few, so they won't impact the end result much.

8% boost by just pre- and post-bootstrapping the data.

The pre-bootstrapping and post-bootstrapping data results can be seen in the appendix as Confusion Matrices [table B.1 (Transformer Model) and figure B.1]. As you can see, the bootstrapping part was most effective with the Transformer model. Here, we got a total boost of about 8% in the F1 score, and the recall values have improved enormously. On the other hand, the improvements in the 30% of the test data have only minimally improved the spaCy models. This is also due to the fact that the spaCy models contained more errors in the area of the Unknown Inaccuracies than those of the Transformer model. We were able to show that an improvement of the test data can improve the quality of the model and that such a step is reasonable.

Another possible optimization area is to tackle the train data. Due to the generative approach, we will be facing some issues in the training data set. Before we even train the model, this must be handled in a pre-processing step. But manual bootstrapping can improve that data as well, when retraining the model with it. [KAVZOGLU][[Brownlee, 2018]]

Fine-tuning as possible improvement step.

Up to now, we have limited our optimization possibilities to the improvement of the erroneous test as well as training data. These steps mainly help to eliminate or reduce the classes of inaccuracies. However, there are also correctly identified differences between test data and prediction. These are then also correctly recorded in the classes Missed Entities and Over Generalized. We can assume that more accurate training data would reduce these problems [KAVZOGLU]. However, there will always be a small amount of error. To tackle this issue and to understand why fixing this problem might be necessary, let's first try to understand the problem and motivation of this thesis in more detail. The thesis approach is heavily focused on extracting materials from Maker tutorials, limiting ourselves to the scope of birdhouses. These tutorials are all human-written and therefore cannot be equipped with tools such as filtering and explicit category searching or can only be equipped with great difficulty. We can use extraction to solve the fundamental problem of understanding the materials needed to build a birdhouse. In the evaluation, we also found that the accuracy of the models that address the described approach is very high. However, our intention with this task is to extract the materials that can replace main materials, such as wood, used in the construction of birdhouses, in order to ensure the functionalities mentioned in the introduction. These materials can vary from a soda can or wool, all the way up to a plastic detergent bottle. Now, if the models do not find new main materials, very valuable information can be lost to the user, and the high scores of the models are obsolete if something like this occurs more often. To target this specific problem and thereby provide the user valuable extraction solution, we will utilize Transfer Learning.

Motivation of why Transfer Learning is needed.

Transfer learning is a technique in which a deep learning model trained on a large dataset is used to perform simi-

Further fine-tuning helps recognizing materials that were unknown and also not recognized by the models before.

lar tasks on a different dataset. We call such a deep learning model a pre-trained model. This proves to be an extremely robust technique to train an existing model further [Lee et al. [2017]]. This allows you to up the computational metrics and solves the problem with unknown main materials. To realize this, one would need more training data, including new labeled materials. This data could then be used for further fine-tuning. The new model should now recognize the materials that were previously unknown to it.

Motivation of an interactive Gui that helps optimizing the models.

Over time, more and more birdhouse tutorials will emerge, and some will also use creative materials that the models have never seen before. Now, how can we verify our models against the new documents? For optimal handling of undetected new materials and erroneous predictions in general, we should create an interactive GUI that allows us to evaluate and improve the prediction evaluation. In the following, we will discuss the concept of the GUI and explain how to use it.

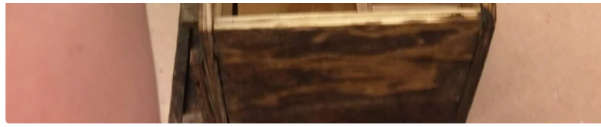
5.1.4 GUI for interactive optimization

Building an GUI that will use the design of the platform.

The graphical user interface should give us an option to evaluate the prediction and improve the data set. This should not only be used by our side, but external users from the Maker projects should be able to operate this tool. The tool's design will be very similar to the tutorials of Instructables itself. We believe that a familiar and unobtrusive design can help the user quickly and easily understand the user interface while allowing them to focus on what matters most: reviewing and improving the prediction. Similar to the visualization tool, we will mark the individual entities as material accordingly if the prediction dictates so. Otherwise, the words will remain unmarked. In the graphic 5.4, we see a small section of how the tool marks the individual positive instances accordingly.

Operations that are needed for the interactive GUI.

Now to make this tool also interactive, we will include some operations. On the one hand, we would like to be able to mark positive instances that were not found as material.



Now it is time to add on the **MATERIAL hinge**. This is arguably the most difficult part of the process, as you have to be very careful. You don't want to get too big of a **MATERIAL hinge**, as it won't be able to close all the way. So get a **MATERIAL hinge** on the smaller side, and **MATERIAL glue** one side of the **MATERIAL hinge** onto the half of the already glued on the house. Then take the other side of the **MATERIAL hinge** and **MATERIAL glue** it on the part of the roof not on the house. Now the roof is perfect, as one side can flip over onto the other, allowing you to easily pour birdseed inside the house.





 Add Tip
  Ask Question
  Comment
  Download

Figure 5.4: Screenshot of GUI used on one of the tutorials in Instructables

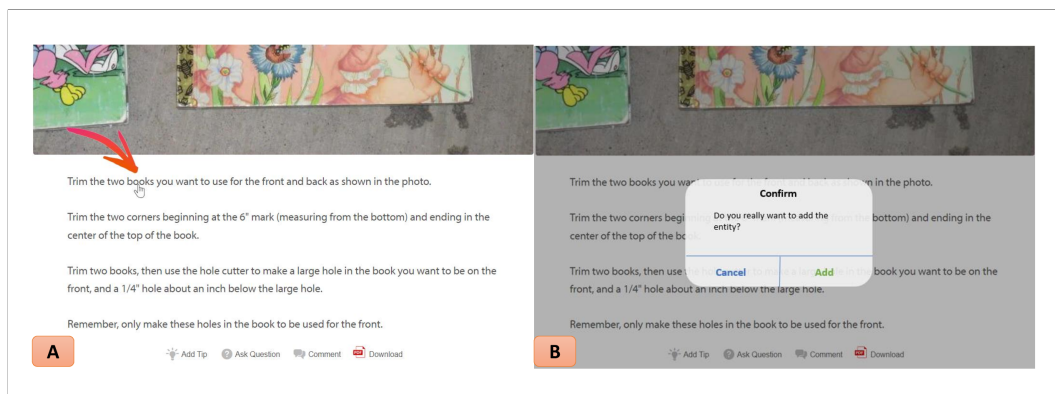


Figure 5.5: The graphic depicts an add item procedure. (A) By clicking onto any entity you can mark them as material. (B) When clicked you have to confirm first before it gets accepted as material.

On the other hand, we would also like to be able to unmark instances that were incorrectly marked as material. This is also made very simple and user-friendly so that one click with another confirmation on one of the unmarked words is enough to mark it as material. The deletion should also be easy to perform. All marked materials will have a small X symbol, with which you can remove the marking. This should also be combined with a confirmation, as it is easy to misclick (c.f. figure 5.5 and 5.6).

Before describing the technical process in more detail, we

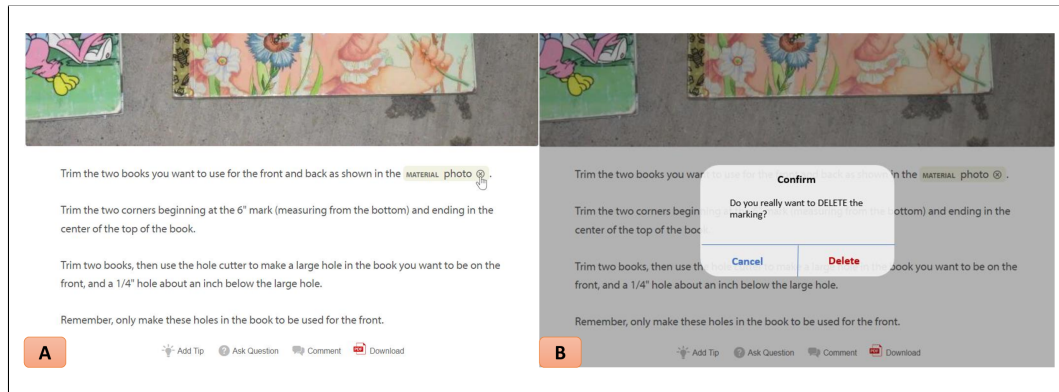


Figure 5.6: The graphic depicts an delete marking procedure. (A) By clicking onto the X symbol of an marked entity, you can delete the marking. (B) When clicked you have to confirm first before the marking will be deleted.

Procedure of deleting
and adding item with
the GUI.

would discuss how to use the tool. As described above, the tool is designed very intuitively. It corresponds to the design of the tutorials, and the individual predictions are all marked in the text. A user can go over the tutorial and verify the individual marked materials. If it is a material, it is kept. Otherwise, the user can click on the X-symbol for the corresponding material and deletes the marking after confirmation (c.f. figure 5.6). Furthermore, the user can also click on unrecognized materials to classify them as material (c.f. figure 5.5). In order to prevent the whole system from being only run by individual users or even being negatively influenced by them, we will introduce you to the concept of multi-user evaluation. We will continue to focus on the Maker projects for Instructables, but this principle can also be used on all Maker projects.

Concept on how user
can use this tool.

To take advantage of the tool, it must be approached actively and with precision. For this, you need active users who perform the evaluation voluntarily and constantly. Since most domains like birdhouses don't necessarily generate a large number of new tutorials every week, a number of 5-10 users is sufficient. You can prompt users either after a certain time interval or after a certain number of newly created tutorials to perform the improvement or verification. Either way, the users can be notified as soon as such a request is due. If an evaluation of a document is done, by marking the unidentified materials, deleting in-

correctly identified materials or taking no actions as in verifying its correctness, the document can be submitted and the changes will be saved in the background. The user would start with the next document, until he is finished with all given documents. The number of documents to be evaluated should be limited and can be configured by someone who provides this tool. The reason for the multi-user evaluation is to avoid manipulations or also error susceptibilities of individual users. After all users have performed the evaluation, a threshold can be used to evaluate which entities have been correctly evaluated and which not. For example, you could always initiate a match of 80% of users as a threshold. This means that at least 8 out of 10 user should have made the same correction so that the change is included in our new data set. The threshold will be also configurable and should be adjusted after some trial runs.

The technical process of the GUI

We have already seen at the beginning that the data for the evaluation occur in the form of a list with tokens. The tool will also make use of this form, and as described at the beginning of the evaluation, the individual tokens will be provided with the corresponding label (c.f. List of labels and tokens). This way all tokens marked as material in the tool have the label "MATERIAL" and the other tokens have the label "O". A deletion of the marking means then simply that the label is changed in the data record from "MATERIAL" to "O", and an addition of materials sets the labels the other way around. Now, in addition to an evaluation, a training dataset can also be generated to utilize Transfer Learning. This is where the pipeline, which was introduced in the concept chapter, comes in handy. The pipeline is set up in such a way that we can fine-tune all three models very simply with training data. These are even already in the desired form and do not have to go through all the pre-processing steps and can go directly to the fine tuning part of the concept. For the spaCy models, however, the training data should still be converted into the object formats required by spaCy. However, these can be easily performed

The training data and test data can now be manipulated with the tool.

by the utils functions of the library of spaCy. After training, the models should recognize the new learned materials without any problems and the other errors should be better understood in the models as well. Over time, the models, especially the Transformer model, will be very reliable in terms of extracting materials.

Chapter 6

Future work

The idea of our research question on how we can gain information from these human-written tutorials and how we can utilize modern neural networks and models to perform these tasks are based on the motivation of how we can improve users' usability on such DIY platforms. Our research focused on building models and performing good extraction models rather than on how information such as used materials from these tutorials can now be leveraged for the users' usability. Thus, this chapter will be dedicated to the value of the results and how they can now be used as a starting point for further research in the field of HCI area. In the beginning, we will elaborate again on the motivation.

DIY tutorial platforms like Instructables and Thingiverse often contain multiple tutorials of similar projects, which only differ slightly in the utilized materials and tools. Users who look up tutorials have limited possibilities to filter the given tutorials aside from reading the titles and skimming the tutorial's content. However, Makers often look for projects that use certain materials or tools they either have at hand or are interested in learning more about. Thus, the limited possibility to filter tutorials by different parameters extends the user's search time and leads to frustration. Now that we can extract specific categories, we can apply this extraction to all tutorials and collect the data. Directly you can link this extracted data to the tutorials, e.g., as tags, and thus use a filter option. A simple multi-select box can

Concept of using material for filtering and searching.

now list all available materials in the case of materials. By selecting specific materials, only those materials that have been tagged will be displayed. Similarly, using the search bar, you can now use these tags to narrow the search to specific materials. This can help the user find tutorials that are relevant to them much more efficiently and quickly.

Conceptual idea of material substitution.

Another advantage of the work results is that the comprehension of the materials can be used to replace certain materials. In a use case, this may be that the user does not have one or more materials available for a particular tutorial. Now instead of filtering tutorials based on available materials, it would be beneficial for the user to be able to swap out individual materials. This could be conceptualized in a simplified way as follows. First, we run our extraction model on each tutorial and generate tabular data this way, which stores all tutorial materials in a row. Now, we can use association rules, a concept for understanding the associations between entities. We will calculate the associations between materials using the rules in this case. After the calculation, we have the probability of which materials are likely to occur with which others. Thus, we can now allow for simplified substitution. One way to realize this is to provide a graphical tool that is given with each tutorial. It could be a collection of multi-select boxes, which first shows all materials of the tutorial itself. Now, you can click on a single select box with the material you want to substitute. The select box will show you the options that are most likely to occur in this combination with the other available materials. Here, it is referred back to the calculation of the association rules. However, this idea limits the possibilities of substituting materials since only associations with an identical combination of materials are considered. One could improve this by including at least subsets. Thus also, such associations are considered, which do not necessarily contain all materials of the tutorial, but only parts of it. This can also be used the other way around, that you include the superset for subsets.

Another extension would be to include the order of the materials and to allow the determination of the materials by this order. This could look like this. First, a specific set of materials is shown. E.g. Wood → Nail → Oil → Paint. Now,

if you want to change Nail to Glue, everything before that would stay that way, and the algorithm would calculate the most common materials with Wood and Glue. However, it can be challenging to determine the order of materials. The order is important to maintain the main materials.

The last benefit we will point out is giving recommendations based on materials. The user can save the materials that are available in his user profile and activate a kind of notification, with which he can be notified when certain materials are used in new tutorials. Additionally, with the user's consent, the platform can store data and send a recommendation to the user based on the data, e.g., frequently searched materials. These types of utilization of materials are just further examples but can be challenging to test in the context of these platforms without having extended access permissions to these platforms.

Recommendation
system based on
materials.

Chapter 7

Conclusion

In this chapter, we want to recapture the thesis's main results and critically discuss our results' meaning. We will point out that we abstract the results here, as we have extensively discussed them in previous parts of this thesis. Hence, we will only refer to the outcomes of these discussions and comparisons. Afterward, we will shortly also summarize other research areas that we already discussed in chapter 6.

7.1 Overview

The goal of the thesis was to develop a framework or pipeline that facilitates the process of creating NER models which enable the extraction of certain keywords out of DIY tutorials – mainly to reuse this information for further HCI research questions. The pursued approach strongly leaned on current NLP technologies, combined with an interactive improvement tool inspired by the paper “methods for design and evaluations of HCI combined with NLP systems” [Heuer and Buschek [2021]]. We have also looked at other related work and demonstrated the extent to which our work is differentiated and what aspects we find useful to incorporate into our work. Our approach chapter describes the general pipeline to create models efficiently and

Summary of the thesis.

elaborates on all components. Moreover, we describe how this implementation has to be realized. We will also explain individual strategy points in more detail and justify the decision points. In this work, we create three different models and also prepare our pipeline for all three models. To make a comparison, we will use our training and test data in the same way for all three models. The evaluation showed us the results of each model executed on the same test data. We saw that the transformer model performed best out of all three models. In addition, through the manual and targeted analysis, we found that some of the predictions are wrong, and the training data and test data harbor some errors due to their auto-generated creation. Improving this data and fine-tuning with the new, improved data could boost our results and reduce inconclusive errors. At the end of the evaluation, we conceptualized a tool with the idea of improving erroneous data, which users can use interactively. The goal is to enable similar improvements of the datasets as we did manually in the evaluation as a proof of concept. On the one hand, this should improve the training and test data and avoid direct errors. On the other hand, the user can utilize our pipeline with the outcome of the interactive tool. The pipeline enables further training as a basis for the improvement of the results. That is, finetuning with better datasets will improve the model's performance, as shown in the evaluation section. Finally, we also discussed the benefits of now having such extraction tools. For this, we used our results as a starting point for further research in the area of HCI. In the future work chapter, we discussed application examples based on the extracted information, such as materials. In addition, we have also developed concepts on how to design individual applications that can provide added value for the user of such DIY platforms.

Future work should include a user study testing these tools.

In the context of this work, we have evaluated the correctness and performance of our extraction models with regard to quality measure-based benchmarks and baselines and the requirements as we formulated them in Section 2. Moreover, we also evaluated a part of the outcome manually to understand the erroneous predictions. Our data generation still showed some errors, which can now be improved by interactive user participation. Yet, we have not

designed how the tool for enhancing these data sets can be used within a platform as Instructables. Thus, an evaluation of the usage of this tool has to be conducted, and the value has to be validated. What cannot be left out is that this work has been restricted to one particular domain for the development of the approach. It is possible to adapt this approach to other domains or to apply the models to other domains. But also here, the performance has to be validated first. The creation of such models, however, is greatly simplified by the pipeline.

Appendix A

Computational Metrics before optimization

	Spacy CNN Model	Spacy WV Model	Transformer Model
Precision	0,861	0,860	0,863
Recall	0,649	0,637	0,710
F1-Score	0,741	0,732	0,779
Accuracy	0,978	0,977	0,979
Error	0,022	0,023	0,021

Table A.1: Computational metrics before optimization

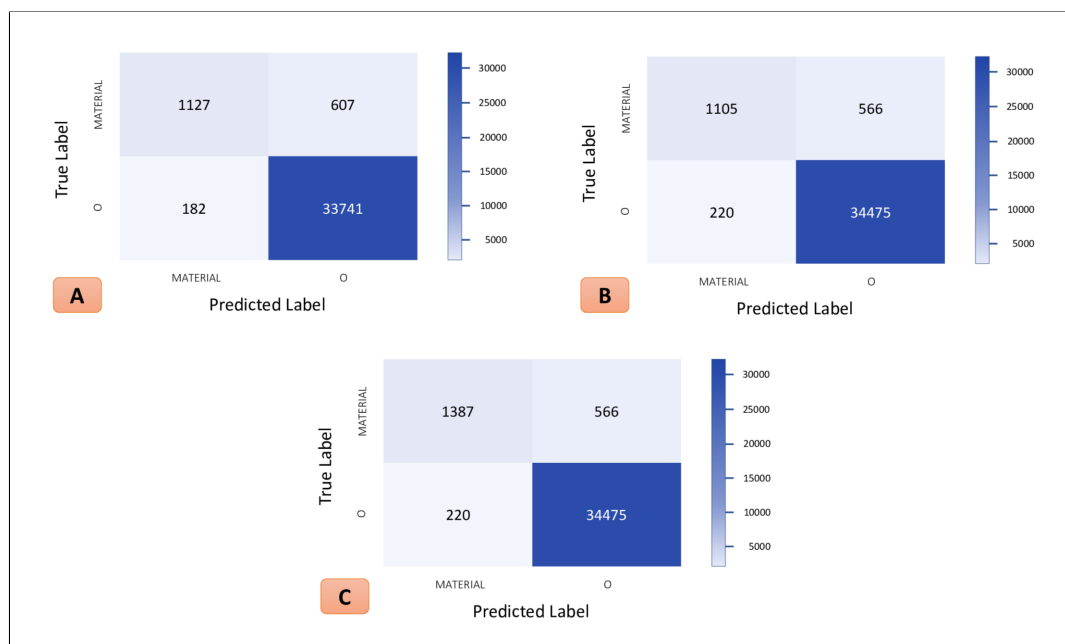


Figure A.1: Confusion matrices of all three models before optimization. These results came out after the pipeline was built and the model were trained once with the given pipeline. (A) Spacy CNN Model (B) Spacy Word Vector Model (C) Transformer Model

Appendix B

Computational Metrics after optimization

	Spacy CNN Model	Spacy WV Model	Transformer Model
Precision	0,891	0,893	0,919
Recall	0,699	0,687	0,796
F1-Score	0,783	0,777	0,853
Accuracy	0,981	0,981	0,985
Error	0,019	0,019	0,015

Table B.1: Computational metrics after optimization

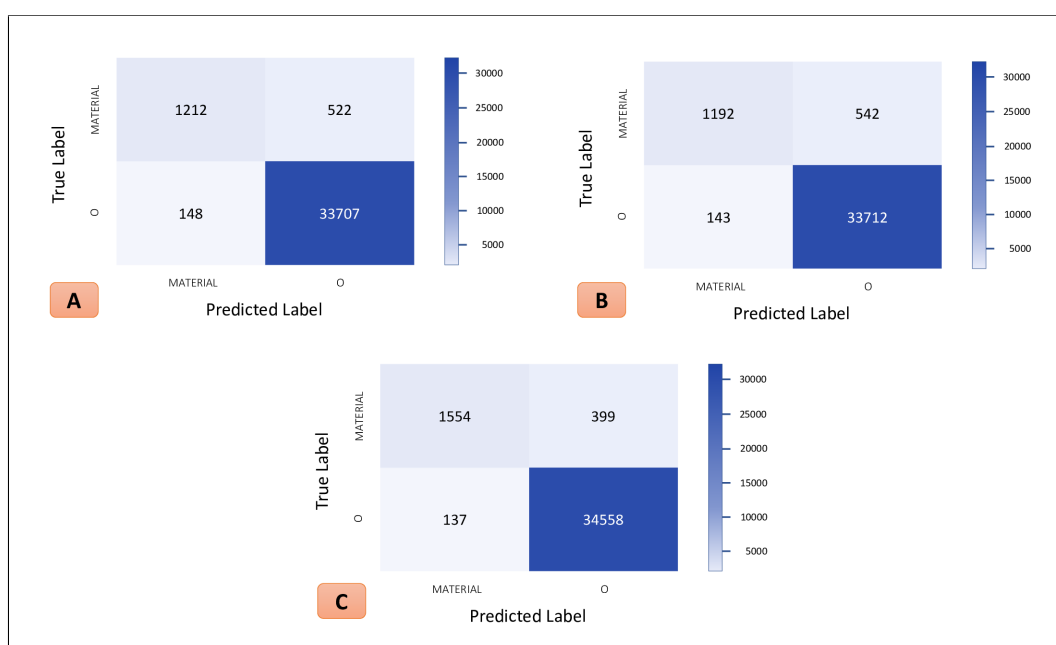


Figure B.1: Confusion matrices of all three models after optimization. Here, about 30% of the data was picked out and manually removed errors. (A) Spacy CNN Model (B) Spacy Word Vector Model (C) Transformer Model

Bibliography

Conll data set in huggingface. <https://huggingface.co/datasets/conll2003>, 2003. Accessed: 2022-08-15.

Selenium.dev. <https://www.selenium.dev/>, 2022. Accessed: 2022-08-15.

Instructables about page. <https://www.instructables.com/about/>, 2022. Accessed: 2022-08-15.

spacy.io. <https://spacy.io/>, 2022. Accessed: 2022-08-15.

unicodebook.readthedocs.io. <https://unicodebook.readthedocs.io/encodings.html>, 2022. Accessed: 2022-08-15.

Mohammed Alawad, Hong-Jun Yoon, and Georgia Tourassi. Energy efficient stochastic-based deep spiking neural networks for sparse datasets. In *2017 IEEE International Conference on Big Data (Big Data)*, pages 311–318. IEEE, 2017.

Christopher M Bishop and Nasser M Nasrabadi. *Pattern recognition and machine learning*, volume 4. Springer, 2006.

Jason Brownlee. *Better deep learning: train faster, reduce overfitting, and make better predictions*. Machine Learning Mastery, 2018.

KR1442 Chowdhary. Natural language processing. *Fundamentals of artificial intelligence*, pages 603–649, 2020.

Jacob Eisenstein. *Introduction to natural language processing*. MIT press, 2019.

- Barney G Glaser, Anselm L Strauss, and Elizabeth Strutzel. The discovery of grounded theory; strategies for qualitative research. *Nursing research*, 17(4):364, 1968.
- Daniel Glez-Peña, Anália Lourenço, Hugo López-Fernández, Miguel Reboiro-Jato, and Florentino Fdez-Riverola. Web scraping technologies in an api world. *Briefings in bioinformatics*, 15(5):788–797, 2014.
- Hendrik Heuer and Daniel Buschek. Methods for the design and evaluation of hci+ nlp systems. *arXiv preprint arXiv:2102.13461*, 2021.
- Matthew Honnibal, Ines Montani, Sofie Van Landeghem, and Adriane Boyd. spaCy: Industrial-strength Natural Language Processing in Python. 2020. doi: 10.5281/zenodo.1212303.
- KAVZOGLU. Increasing the accuracy of neural network classification using refined training data. *Environmental modelling*, 24; Jg. 2009(7):850–858. ISSN 1364-8152 (P), 1873-6726 (E). Quelldatenbank: info:sid/sfx:ULBD.
- Mandy Korpusik, Zoe Liu, and James R Glass. A comparison of deep learning methods for language understanding. In *Interspeech*, pages 849–853, 2019.
- Guillaume Lample, Miguel Ballesteros, Sandeep Subramanian, Kazuya Kawakami, and Chris Dyer. Neural architectures for named entity recognition. *arXiv preprint arXiv:1603.01360*, 2016.
- Ji Young Lee, Franck Dernoncourt, and Peter Szolovits. Transfer learning for named-entity recognition with neural networks. *arXiv preprint arXiv:1705.06273*, 2017.
- Bill Yuchen Lin, Frank F Xu, Zhiyi Luo, and Kenny Zhu. Multi-channel bilstm-crf model for emerging named entity recognition in social media. In *Proceedings of the 3rd Workshop on Noisy User-generated Text*, pages 160–165, 2017.
- Xiaofei Ma, Peng Xu, Zhiguo Wang, Ramesh Nallapati, and Bing Xiang. Domain adaptation with bert-based domain classification and data selection. In *Proceedings of the 2nd Workshop on Deep Learning Approaches for Low-Resource NLP (DeepLo 2019)*, pages 76–83, 2019.

- Bernhard Mehlig. *Machine Learning with Neural Networks: An Introduction for Scientists and Engineers*. Cambridge University Press, 2021.
- Tomas Mikolov, Kai Chen, Greg Corrado, and Jeffrey Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013a.
- Tomas Mikolov, Ilya Sutskever, Kai Chen, Greg S Corrado, and Jeff Dean. Distributed representations of words and phrases and their compositionality. *Advances in neural information processing systems*, 26, 2013b.
- Tomáš Mikolov, Wen-tau Yih, and Geoffrey Zweig. Linguistic regularities in continuous space word representations. In *Proceedings of the 2013 conference of the north american chapter of the association for computational linguistics: Human language technologies*, pages 746–751, 2013c.
- Sebastian Raschka and Vahid Mirjalili. Python machine learning: Machine learning and deep learning with python. *Scikit-Learn, and TensorFlow. Second edition ed*, 3, 2017.
- Subendhu Rongali, Abhyuday Jagannatha, Bhanu Pratap Singh Rawat, and Hong Yu. Continual domain-tuning for pretrained language models. *arXiv preprint arXiv:2004.02288*, 2020.
- Ina Rösiger, Johannes Schäfer, Tanja George, Simon Tannert, Ulrich Heid, and Michael Dorna. Extracting terms and their relations from german texts: Nlp tools for the preparation of raw material for specialized e-dictionaries. In *Proceedings of the eLex 2015 conference, Ljubljana*, 2015.
- Ilya Sutskever, Oriol Vinyals, and Quoc V Le. Sequence to sequence learning with neural networks. *Advances in neural information processing systems*, 27, 2014.
- Muzamil Hussain Syed and Sun-Tae Chung. Menuner: Domain-adapted bert based ner approach for a domain with limited dataset and its application to food menu domain. *Applied Sciences*, 11(13):6007, 2021.
- Tiffany Tseng and Mitchel Resnick. Product versus process: representing and appropriating diy projects online. In

Proceedings of the 2014 conference on Designing interactive systems, pages 425–428, 2014.

Wil van der Aalst. Lecture notes in introduction to data-science, March 2021.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

Ron Wakkary, Markus Lorenz Schilling, Matthew A Dalton, Sabrina Hauser, Audrey Desjardins, Xiao Zhang, and Henry WJ Lin. Tutorial authorship and hybrid designers: The joy (and frustration) of diy tutorials. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, pages 609–618, 2015.

Thomas Wolf. Lysandre debut, victor sanh, julien chaumond, clement delangue, anthony moi, pierric cistac, tim rault, r'emi louf, morgan funtowicz, and jamie brew. 2019. *Huggingface's transformers: State-of-the-art natural language processing*. *ArXiv, abs*, 2019.

